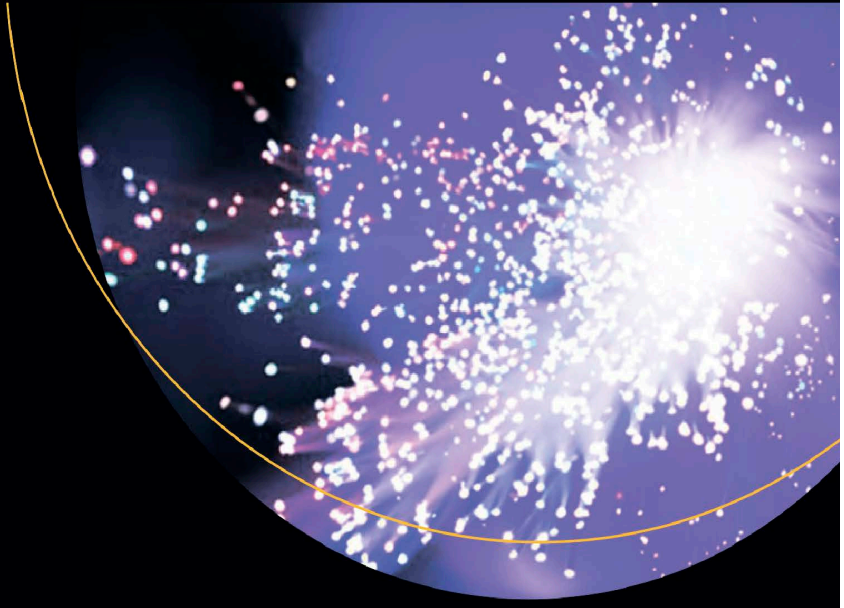


Инновации SQL Server 2019

Исользование технологий больших данных
и машинного обучения

Боб Уорд

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО

Инновации SQL Server 2019

Боб Уорд

Инновации SQL Server 2019

SQL Server 2019 Revealed

Including Big Data Clusters and Machine Learning

Bob Ward
Foreword by Rohan Kumar

Apress®

Инновации SQL Server 2019

Использование технологий больших данных и машинного обучения

Боб Уорд
Предисловие от Рохана Кумара (Rohan Kumar)

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО
Москва, 2020

УДК 004.655
ББК 32.973.26-018.2
У64

У64 Боб Уорд (Bob Ward)
Инновации SQL Server 2019. Использование технологий больших данных и машинного обучения / пер. с англ. Желновой Н. Б. – М.: ДМК Пресс, 2020. – 408 с.: ил.

ISBN 978-5-97060-595-0

В книге представлен исчерпывающий обзор SQL Server 2019 – инновационной версии популярной СУБД. Авторы рассказывают о производительности и безопасности, об использовании контейнеров и технологии Kubernetes, о работе с кластерами больших данных и средствах машинного обучения. Подробное описание новых функций SQL Server 2019 позволит читателю расширить свои навыки в области управления и извлечения информации из больших данных.

Книгу можно использовать в качестве справочника – при желании ее главы можно изучать по отдельности. Многочисленные примеры, рисунки и ссылки помогают разобраться в технических подробностях.

Издание адресовано разработчикам и профессионалам, работающим с данными и знакомым с базовыми функциями SQL Server.

УДК 004.655
ББК 32.973.26-018.2

ISBN 978-1-48425-418-9 (англ.)
ISBN 978-5-97060-595-0 (рус.)

Эта книга посвящена сообществу SQL Server, #sqlfamily.
Без этого сообщества удивительный продукт SQL Server
не был бы тем, чем он является сегодня.

Оглавление

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Apress Media очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторе

Боб Уорд (Bob Ward) – главный архитектор технической группы Microsoft Azure Data SQL Server, которая ведет разработку всех версий SQL Server. Боб проработал в Microsoft более 26 лет; он участвовал в выпуске каждой версии SQL Server, начиная с версии 1.1, поставляемой с OS/2, и заканчивая SQL Server 2019, включая Azure. Он популярный докладчик многих конференций; часто выступал с докладами о новых версиях SQL Server, его внутреннем устройстве и производительности на таких мероприятиях, как PASS Summit, SQLBits, SQLIntersection, Red Hat Summit, Microsoft Inspire и Microsoft Ignite. Вы можете подписаться на него в соцсетях, ссылки на его профиль: @bobwardms или www.linkedin.com/in/bobwardms.

Боб является автором книги *Pro SQL Server on Linux* («SQL Server на Linux для профессионалов»), выпущенной Apress Media.

О техническом рецензенте

Аарон Бертран (Aaron Bertrand) – технический специалист, обладающий более чем двадцатилетним опытом работы с SQL Server. Он работал непосредственно с несколькими командами разработчиков продуктов Microsoft и хорошо известен тем, что участвует в развитии технических навыков и передаче опыта для широкого сообщества разработчиков. Он автор многих статей, докладчик конференций и модератор форумов, посвященных технологиям баз данных.

Предисловие

Время, в которое мы живем, – это уникальный переломный момент в истории технологий. Наступает золотой век для данных, аналитики и искусственного интеллекта. Темпы роста данных превосходят все когда-либо достигнутые показатели, и прорыв в области цифровых технологий, случившийся благодаря технологиям искусственного интеллекта и машинного обучения, создал неограниченный потенциал для использования данных с целью обеспечения конкурентного преимущества для бизнеса. С резким ускорением процесса цифровизации основной вопрос, с которым мы сейчас сталкиваемся, заключается в том, как воспользоваться преимуществами этого огромного объема данных, чтобы помочь нашим компаниям и сообществам в их преобразованиях.

Мы видим огромный потенциал в области применения возможностей интеллектуального облака и интернета вещей с элементами искусственного интеллекта. SQL Server не имеет аналогов в отрасли по уровню стабильности и системности, который предоставляется разработчикам, инженерам данных и администраторам в разнообразных вариантах: на периферии, в локальной среде, частном облаке и общедоступном облаке. Наше сообщество SQL Server сыграло очень важную роль в этой эволюции, и я не могу в достаточной мере поблагодарить их за поддержку и отзывы за последние 25 с лишним лет.

Версия SQL Server 2019 – выдающаяся, особенная, и я горжусь тем, что сделала наша команда. SQL Server 2019 основан на инновациях, которые были заложены еще в SQL Server 2016 и SQL Server 2017. Несмотря на то что есть несколько новых возможностей, которые улучшат пользовательский опыт наших клиентов, как и ожидается от каждой новой версии SQL Server, я больше всего рад удивительным инновациям, расширяющим навыки, накопленные нашими клиентами на протяжении нескольких десятилетий, в области управления и извлечения информации из больших данных. Это нововведение сыграет решающую роль в поддержке цифровой трансформации у наших клиентов.

Боб Уорд работает в команде SQL Server с самого начала и оказал заметное влияние на продукт. Очень немногие обладают такими широкими и глубокими знаниями, и это объясняет, как ему удается сделать сложные концепции простыми и понятными. Я надеюсь, вы прочитаете эту книгу с удовольствием.

Рохан Кумар (Rohan Kumar),
корпоративный вице-президент Azure Data в Microsoft

Благодарности

В моей жизни есть множество вещей, за которые я испытываю благодарность, и возможность написать эту книгу – одна из них. Я верю, что все мои дарования приходят от Бога, и именно благодаря Его благодати я познал Иисуса Христа. Прежде всего я должен поблагодарить мою очаровательную и талантливую жену Джинджер. Она – мой партнер, моя родственная душа. Она слышала все мои жалобы, видела, как я засиживаюсь допоздна за работой над книгой, а иногда ей приходилось водить машину, чтобы я мог работать над книгой на пассажирском сиденье. Я не знаю никого, кто имеет более сильную веру, чем моя жена, и мне очень повезло, что рядом со мной находится такой человек. В этом году мы отпраздновали 30-летие нашего брака, и я до сих пор наслаждаюсь каждым моментом, проведенным с ней вместе. Я также хочу поблагодарить моих сыновей Троя и Райана. Сейчас Трой живет в Чарльстоне, штат Южная Каролина, где я написал последнюю часть книги, когда приближался ураган Дориан. Трой – человек, которым я восхищаюсь не только за его характер, но и за его неуклонное стремление сделать наш мир лучше. Райан учится на втором курсе юридического факультета Бейлор (Go Bears!). Он продолжает удивлять меня своим интеллектом, целеустремленностью и способностью сохранять уверенность, но все же находит время, чтобы совершенствоваться в игре в гольф. Я также хочу поблагодарить мою маму Аннетт Гибо (Annette Gibaud), которая была и остается для меня живым примером доброты к окружающим ее людям.

Множество людей помогали мне в работе и внесли свой вклад в эту книгу. Я хочу поблагодарить Apress Media за предоставленную мне возможность написать книгу. Джонатан Дженник (Jonathan Gennick) и Джилл Бальзано (Jill Balzano) поддерживали меня на каждом шагу, помогая мне довести дело до конца. И эта книга не была бы написана вовремя, да и вообще вряд ли появилась бы на свет без технического рецензента Аарона Бертрана (Aaron Bertrand). Когда я еще только обдумывал идею этой книги, Аарон был одним из первых, о ком я подумал как о возможном рецензенте, поскольку он обладает глубочайшими знаниями по SQL Server и заслуженной репутацией эксперта в сообществе. Аарон был сверхчеловеком, столь быстро он давал рецензии для каждой главы книги.

В Microsoft прежде всего спасибо Рохану Кумару (Rohan Kumar), Гейлу Шеппарду (Gayle Sheppard) и Асаду Хану (Asad Khan) за предоставленную мне возможность распространять знания о SQL Server 2019, что помогло мне в детальном изучении продукта для написания этой книги. Я также хочу лично поблагодарить двух моих ближайших коллег, Бака Вуди (Buck Woody) и Анну Хоффман (Томас) (Anna Hoffman (Thomas)). Я посетил много разных стран вместе с Баком и Анной в 2018 и 2019 годах, рассказывая об SQL Server, кластерах больших данных и Azure. Они оба поработали над

улучшением моих навыков выступлений и преподавательского мастерства и составили отличную компанию в дороге. Также я очень благодарен прекрасному коллективу Microsoft SQL Server Engineering. Я в восторге от работы с такими умными и профессиональными людьми, многие из которых помогли мне разобраться с техническими подробностями, которые вы найдете в этой книге. Сначала я хочу поблагодарить Славу Окса (Slava Oks) и Трэвиса Райта (Travis Wright), которые помогли мне рассказать историю о проектах «Сиэтл» и «Арис» и сыграли важную роль в продвижении этой версии SQL Server, в том числе новых возможностей, связанных с кластерами больших данных. Конор Каннингем (Conor Cunningham) не устает удивлять меня своими глубокими знаниями об этом продукте, внося весомый вклад в его улучшение.

Настоящие герои этой книги – члены команды инженеров, создавшие эту версию SQL Server и оказавшие мне помощь в работе над различными главами книги. Спасибо Педро Лопесу (Pedro Lopes), Пэм Лахуд (Pam Lahoud), Амиту Банерджи (Amit Banerjee), Брайану Карригу (Brian Carrig), Техасу Шаху (Tejas Shah), Вин Ю (Vin Yu), Сурабху Агарвалу (Sourabh Agarwal), Михаеле Блендеа (Mihaela Blendea), Нелли Густафссон (Nellie Gustafsson), Абиоле Оке (Abiola Oke), Джеймсу Роуланду-Джонсу (James Rowland-Jones), Скотту Конерсману (Scott Konersmann), Стюарту Пэдли (Stuart Padley), Дэвиду Кризу (David Kryze), Роберту Доррy (Robert Dorr), Митчелл Стернке (Mitchell Sternke), Россу Монстеру (Ross Monster), Мадлен Макдональд (Madeline MacDonald), Дилану Грею (Dylan Gray), Джо Сэку (Joe Sack), Шрее Верма (Shreya Verma), Якубу Шимашеку (Jakub Szymaszek), Йоахиму Хаммеру (Joachim Hammer), Рагхаву Каушику (Raghav Kaushik), Парагу Полу (Parag Paul), Панагиотису Антонопулосу (Panagiotis Antonopoulos), Майклу Нельсону (Michael Nelson), Пранджалу Гупте (Pranjal Gupta), Ярупату Джисароджито (Jarupat Jisarojito), Вэйюн Хуангу (Weiyun Huang), Джорджу Рейнье (George Reynya), Умачандару Джаячандрану (UC) (Umachandar Jayachandran (UC)), Сахаю Сайни (Sahaj Saini), Майку Хаббену (Mike Habben), Вакару Пирзаде (Vaqar Pirzada), Рони Чаттерджи (Rony Chatterjee), Вики Харп (Vicky Harp), Алану Ю (Alan Yu), Джеку Ли (Jack Li), Алексею Эксаревскому (Alexey Eksarevskiy), Джею Чо (Jay Choe), Арджениcу Фернандесу (Argenis Fernandez), Кевину Фарли (Kevin Farlee), Арье Библиовичу (Arieh Bibliowicz), Алексу Уманski (Alex Umansky), Маттео Таведжиа (Matteo Taveggia), Капилу Такер (Kapil Thacker), Ли Чжан (Li Zhang) и Дону Цао (Dong Cao).

Я также хочу поблагодарить членов групп Microsoft Marketing и CSS за их помощь: Аншула Рампала (Anshul Rampal), Мэтью Барроуза (Matthew Burrows), Марко Хотти (Marko Hotti), Дебби Лайонс (Debbi Lyons), Суреша Кандота (Suresh Kandoth) и Прадипа М. М. (Pradeep M. M.).

Эта книга и моя работа над ней были бы невозможны без таких партнеров, как HPE, DELL и Red Hat, которые позволили мне рассказать своим клиентам о новой версии SQL Server 2019. Спасибо Венди Хармс (Wendy Harms), Биллу Данмиру (Bill Dunmire), Урсу Ренггли (Urs Renggli), Робер-

ту Сондерсу (Robert Sonders), Луи Имершейну (Louis Imershein) и Стефану Бюро (Stephane Bureau) (тому, кто помогает мне с видео). Особая благодарность также Дэвиду ДеВитту за экскурс в историю Polybase, Брендану Бернсу, совершенствовавшему мои знания в области Kubernetes, и Энтони Ноцентино за глубокое знание Linux и контейнеров.

Наконец, спасибо всемирному сообществу SQL Server. Сейчас мы выпускаем новые версии SQL Server чаще и быстрее, чем когда-либо прежде, но вы все равно проявляете огромный энтузиазм каждый раз, когда я выступаю, рассказывая про SQL Server.

Вступление

Как над моей первой книгой *Pro SQL Server on Linux* («SQL Server на Linux для профессионалов»), так и над этой книгой я работал в разных местах. Я путешествовал в 2019 году больше, чем когда-либо в своей жизни. Это означало, что я должен был быть готов работать над своей книгой где угодно и когда угодно: в самолетах, отелях, поездах, в поездках по городам. Я писал главы этой книги в Сиэтле, Лондоне, Манчестере (Великобритания), Нэшвилле, Лас-Вегасе (я побывал там несколько раз), Сан-Антонио, Остине, Хьюстоне, Орландо, Сент-Люсии (дело было в отпуске), Дженеси (Колорадо), Чарльстоне, Бостоне, Дубае, Йоханнесбурге (Южная Африка), Гринвилле (Южная Каролина) и Индианаполисе. И конечно, я работал над книгой допоздна в своем домашнем офисе, в Норт-Ричленд-Хиллз (штат Техас).

Я подумал, что после завершения своей первой книги я вряд ли возьмусь еще за одну, однако я не мог удержаться от искушения поведать историю SQL Server 2019. Эта книга действительно написана «для души». Я вложил душу в обучение, преподавание, преодоление разнообразных препятствий, документирование, тестирование и работу с SQL Server 2019. Книга объединила все это, а также многое другое.

Я написал эту книгу для разработчиков и профессионалов, работающих с данными, которые обладают фундаментальными знаниями по SQL Server и хотят получить исчерпывающий обзор SQL Server 2019 в одной книге. Эта книга содержит множество примеров, рисунков и ссылок, которые помогут вам разобраться в технических подробностях. Я написал ее, чтобы не только дать своей аудитории полное понимание возможностей SQL Server 2019, но также предоставить возможность использовать ее в качестве справочного материала, к которому можно вернуться в любое время.

Хотя каждую из глав этой книги можно читать отдельно и независимо друг от друга, я настоятельно рекомендую вам начать с главы 1, так она знакомит читателей с историей и предысторией выпуска данной версии. Я также привел в ней обзор всех ключевых возможностей SQL Server 2019 и рассказал, почему я считаю его привлекательным продуктом. Начав с первой главы, вы можете просмотреть все главы по порядку или пропустить некоторые из них. Однако, чтобы извлечь максимальную пользу из главы 10, посвященной кластерам больших данных, сначала необходимо прочитать главы 6, 7, 8 и 9.

Книга разбита на следующие главы:

- глава 1, посвященная истории и обзору версии SQL Server 2019;
- главы 2, 3 и 4 – о производительности, безопасности и доступности. В одних только этих главах есть много интересного об SQL Server 2019;

- глава 5 написана для разработчиков;
- главы 6, 7 и 8 посвящены Linux, контейнерам и Kubernetes;
- глава 9 знакомит вас с виртуализацией данных с использованием Polybase;
- глава 10 – большая глава, посвященная большой теме: кластеры больших данных;
- глава 11 завершает книгу, в ней рассказывается о других новых функциях и миграции.

Мне нравится «учиться на примерах», поэтому в этой книге содержится множество примеров. Вы найдете их почти в каждой главе (и в некоторых случаях я объясняю, как использовать уже пройденный материал). Вы можете найти все примеры, приведенные в этой книге, на GitHub, по ссылке на дополнительные материалы к книге на сайте www.apress.com/9781484254189, или в моем репозитории GitHub по адресу https://aka.ms/bobsqldemos (https://github.com/microsoft/bobsql).

Я также рекомендую вам ознакомиться с бесплатными учебными ресурсами, созданными нашей командой по адресу https://aka.ms/sqlworkshops. Они предлагают бесплатное практическое обучение для SQL Server 2019.

В процессе работы над этой книгой я потратил много времени на размышления «чего хотел бы читатель». Я надеюсь, что вы увидите и почувствуете это, когда будете читать книгу. Если у вас возникнут какие-либо вопросы или замечания в процессе работы с книгой, мне хотелось бы услышать о них. Пожалуйста, напишите мне напрямую на bobward@microsoft.com.

Боб Уорд (Bob Ward)
Норт Ричленд Хиллз, Техас
Сентябрь 2019

Глава 1

Почему SQL Server 2019?

В июле 2017 года я регулярно посещал Редмонд (штат Вашингтон) как член команды разработчиков SQL Server. Я живу в Норт Ричленд Хиллз (в штате Техас), и современные технологии позволяют мне выполнять большую часть своей работы удаленно, вне стен офиса, где собрана основная часть команды SQL Engineering. Но я все еще немного приверженец «старой школы» и считаю, что в определенных случаях ничто не сравнится с личным присутствием, когда люди собираются вместе за работой. К июлю 2017 года я проработал в команде SQL Engineering более года, уделяя основное внимание SQL Server 2016 (пример моей работы над SQL Server 2016 можно найти в интернете по ссылке: https://channel9.msdn.com/Events/Ignite/2016/BRK3043-TS).

До этого времени я был членом знаменитой команды Tiger, но в рамках моего визита в 2017 году меня попросили взять на себя новые задачи, чтобы сосредоточиться на предстоящем выпуске SQL Server 2017. В мои задачи входил SQL Server на Linux, что в конечном итоге привело к написанию моей первой книги, *Pro SQL Server on Linux* («SQL Server на Linux для профессионалов», www.apress.com/us/book/9781484241271). Поэтому во время моего визита я начал встречаться и беседовать с различными членами команды об SQL Server 2017 – о повышении производительности, общем наборе новых функций и технических подробностях работы SQL Server в Linux и о контейнерах. Одним из людей, с которыми я говорил на той неделе, был Слава Окс (Slava Oks). Слава – ведущий менеджер по разработке SQL Server и один из изобретателей SQL Server для Linux. Он написал предисловие для книги *Pro SQL Server on Linux* («SQL Server на Linux для профессионалов», www.apress.com/us/book/9781484241271), и в первой главе этой книги рассказывается об истории его участия в проекте. В то время Слава любил приходить в офис рано; когда я нахожусь в Редмонде, я тоже пытаюсь работать в «техасское время» – а это значит, что я тоже прихожу очень рано.

Поэтому мы часто встречались за кофе тогда, когда большинство других разработчиков еще не добирались до офиса, в здании 16 (хотя сейчас наша команда работает в здании 43). Однажды утром, когда мы со Славой говорили об SQL Server 2017, он спросил меня: «Я рассказывал вам о наших планах относительно следующей версии SQL Server, которая выйдет после SQL Server 2017?» Я, конечно, сделал вид, что помню об этом: «Да, Слава,

я слышал об этом, но не знаю подробностей». Затем он пригласил меня прийти на встречу на следующий день, где он объяснил многим из нашей команды инженеров план проекта. Я провел целый год, сосредоточившись на SQL Server 2016, после чего мне поручили погрузиться в SQL Server 2017 и Linux, а теперь Слава хотел, чтобы я узнал о версии, следующей после той, которая пока еще не была выпущена? Конечно, я не собирался отказывать ему, потому что – ну, это Слава Окс. Это может прозвучать так, будто Слава – какой-то страшный человек, однако он один из самых приятных людей, которых я когда-либо знал в Microsoft.

Поэтому когда я начинал собирать информацию об SQL Server 2017, я пошел по такому пути: я собирался узнать, что мы делаем для будущей версии SQL Server с кодовым названием Проект SQL Server *Сиэтл*.

Проект «Сиэтл»

На встрече со Славой на следующий день я быстро узнал, что за несколько часов мы приступили к одному из самых амбициозных усовершенствований SQL Server, которые я когда-либо наблюдал за всю свою карьеру. Я говорю это, уже зная, что мы выводим на рынок SQL Server под Linux, что ранее считалось невозможным.

Слава и команда проекта выбрали для проекта кодовое название «Сиэтл», потому что в качестве кодового названия SQL Server 2017 использовалось «Хельсинки», и команда и искала новое наименование города для кодового названия проекта. По иронии судьбы, никто в Microsoft раньше не использовал название «Сиэтл», поэтому оно быстро прижилось. Я спросил Славу, когда он впервые начал планировать проект «Сиэтл». Я был поражен, услышав ответ: в январе 2017 года. Тот факт, что такие люди, как Слава, Конор Каннингем (Conor Cunningham) и Трэвис Райт (Travis Wright), планировали проект «Сиэтл», работая над завершением SQL Server 2017 на Linux, стал свидетельством их преданности команде, их стремления удерживать SQL Server на позиции лидера инноваций в отрасли баз данных.

Трудно было поверить, что мы могли так быстро запланировать нечто большее, после того как предоставили так много полезных и инновационных функций в SQL Server 2016 и SQL Server 2017.

В SQL Server 2016 мы добавили новые возможности диагностики производительности с помощью Query Store, а именно новые функции для разработчиков, такие как временные таблицы и интеграция с JSON. Мы повысили безопасность работы благодаря технологии Always Encrypted, динамическому маскированию данных и защите на уровне строк. И мы представили две новые инновационные возможности, выходящие за пределы «обычных» функций для реляционной системы управления базами данных. Одной из них была интеграция языка R для моделей машинного обучения. Второй была интеграция с системами Hadoop при помощи *Polybase* (что в итоге приведет к чему-то большему в 2019 году; однако я

забегаю вперед). Создание возможностей для включения новых сценариев, таких как машинное обучение и большие данные, привело меня и других сотрудников Microsoft к мысли о том, что SQL Server – уже не просто механизм управления реляционными базами данных, а *платформа данных*.

Однако, чтобы создать современную и полнофункциональную платформу данных, нам нужно было расширять возможности приложений в системах, отличных от Windows Server. Это привело к появлению в SQL Server 2017 поддержки Linux и Docker-контейнеров. Запуск на Linux и использование контейнеров стали очень большим шагом вперед для Microsoft, но SQL Server 2017 также включал другие возможности, такие как адаптивная обработка запросов (Adaptive Query Processing), автоматическая настройка, графовая база данных, группы доступности вне кластеров и интеграция с Python, в дополнение к поддержке языка R для служб машинного обучения.

Учитывая все эти новые возможности, как мы можем за короткий период времени спланировать и создать что-то более новое, замечательное и интересное, чем SQL Server 2016 и 2017? Я задал себе этот вопрос, внимательно слушая коллег во время моей первой встречи с командой проекта «Сиэтл». В первые несколько минут меня познакомят с идеей, которая, когда ее позже объявят общественности, будет считаться довольно радикальной. И это новшество было «краеугольным камнем» проекта «Сиэтл», который имеет собственный код проекта: *Aris*.

Проект Aris

В январе 2017 года Рохан Кумар, корпоративный вице-президент Azure Data, поставил перед Славой и руководством команды разработчиков SQL Server задачу разобраться, как интегрировать SQL Server с большими данными. Большие данные – это термин, широко используемый в отрасли и относящийся к системе, которая может обрабатывать большие объемы данных, обычно с использованием распределенной масштабируемой вычислительной платформы. Мне лично нравится определение термина «большие данные», сформулированное моим коллегой Баком Вуди (Buck Woody): «любые данные, которые вы не можете обработать в нужное время с помощью имеющихся у вас технологий». В течение многих лет в качестве системы обработки больших данных наиболее часто выбиралась платформа Hadoop. Итак, в течение нескольких месяцев весной и летом 2017 года команда обращалась к Трэвису Райту (Travis Wright) за консультациями о том, как воплотить в жизнь идею интеграции больших данных. Летом 2017 года у нашей команды Azure Data было несколько проектов с кодовыми названиями «Polaris», «Socrates» (Сократ) и «Plato» (Платон). Я спросил Славу: как ты выбрал кодовое название для проекта – Aris? И получил ответ: Сократ был наставником известного греческого философа Платона, а учеником Платона был Аристотель. Учитывая, что Aris – это часть име-

ни «Аристотель», а также оно входит в состав кодового названия проекта «Polaris», это новое название, Aris, нашло отклик у всех членов команды и у нашего руководства.

Поскольку интеграция с большими данными подразумевает *некоторое* отношение к Hadoop, Трэвис провел несколько встреч с командой, которая представила решение Polybase для SQL Server 2016 и хранилища данных Azure. Идея Polybase заключалась в том, чтобы позволить пользователям SQL Server запрашивать (и принимать) данные из системы Hadoop, используя привычный язык запросов T-SQL, хорошо знакомый нашим пользователям. Кроме того, вместо того чтобы строить простую систему извлечения данных, Polybase могла бы использовать возможности распределенных вычислений, которые существуют в Azure Data Warehouse и Analytics Platform System (ранее известной как Parallel Data Warehouse), для сокращения вычислений и распределенной обработки запросов, чтобы достигнуть увеличения производительности за счет масштабирования при работе с большими наборами данных в целевой системе Hadoop. Я никогда не видел, чтобы Polybase «взлетела» в SQL Server 2016 и 2017, поскольку интеграция систем Big Data Hadoop с реляционными системами, такими как SQL Server, была непростой задачей. Polybase требует значительных усилий по установке и настройке, а ее модель политики безопасности отличается от подходов к защите данных, используемых в системах Hadoop и SQL Server. Кроме того, реализация сокращенных («выталкивающих») вычислений опиралась на концепцию MapReduce, для чего требовалось установить Java на том же компьютере, на котором были размещены службы SQL Server и Polybase. Тем не менее архитектура и концепции интегрированных систем SQL Server и Big Data были пригодны для создания чего-то большего (включая расширение T-SQL под названием EXTERNAL TABLE). Если бы нам удалось упростить историю развертывания и настройки Polybase и добавить больше поддерживаемых источников данных, то такое решение могло бы стать более популярным в отрасли. Кроме того, Трэвис очень быстро понял, что если вы хотите, чтобы вас серьезно воспринимали в мире обработки больших данных, вам нужно рассмотреть другую технологию под названием *Spark*.

Вооружившись этими знаниями, Слава, Трэвис и основная группа членов команды, создавшей SQL Server для Linux, поставили перед собой задачу создать прототип интеграции SQL Server с большими данными, включая Spark. Они устроили многодневную встречу в большом конференц-зале и назвали ее «Хакатон Aris». Членами команды хакатона были Слава Окс (Slava Oks), Трэвис Райт (Travis Wright), Скотт Конерсманн (Scott Konersmann), Стюарт Падли (Stuart Padley), Майкл Нельсон (Michael Nelson), Пранджал Гупта (Pranjal Gupta), Джарупат Джисарохито (Jarupat Jisarojito), Вейюн Хуан (Weiyun Huang), Джордж Рейня (George Reynya), Дэвид Крайз (David Kryze), Умачандар Джаячандран (Umachandar Jayachandran) из Калифорнийского университета и Сахадж Сайни (Sahaj Saini). К моменту завер-

шения хакатона у них был рабочий кластер, который объединил существующую в SQL Server функциональность Polybase с технологией Spark. На рис. 1.1 показана примерная схема кластера, созданного этой командой.

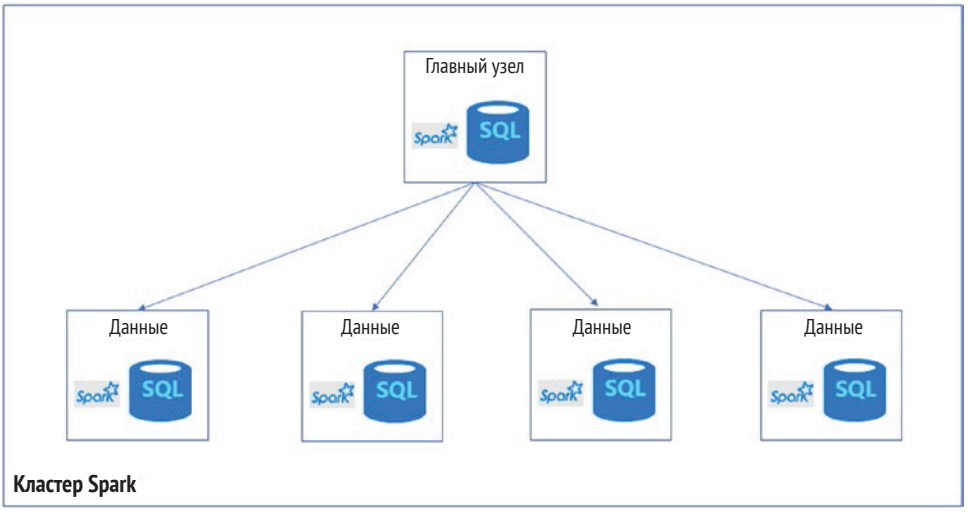


Рис. 1.1. Первый кластер Aris

В своем прототипе они создали кластер Hadoop, включающий компоненты для Apache Spark и HDFS, а также SQL Server Polybase. Они использовали Spark для потоковой передачи данных в узлы данных, а затем применили Polybase для объединения данных на главном узле SQL Server с данными, поступающими со Spark в HDFS. Идея прототипа состояла в том, чтобы доказать, что они могут интегрировать Spark, Hadoop и SQL Server.

Примерно в это же время Трэвис разговаривал с инженерами, которые недавно присоединились к команде. Это были сотрудники из компании Metanautix, которую приобрела Microsoft. Благодаря этому приобретению у нашей команды была технология для подключения к ряду источников данных, в том числе ORACLE, SQL Server, Teradata и MongoDB, через ODBC. Команда подумала, что если мы сможем использовать эту технологию в проекте Aris, то в результате получим довольно интересный пример *виртуализации данных*. SQL Server теперь может быть центром доступа к данным на разных платформах данных и в разных системах без необходимости перемещения данных на SQL Server (с помощью таких методов, как извлечение, фильтрация и загрузка данных (Extract, Transform, and Load, ETL)).

Прежде чем мы смогли предоставить программное обеспечение, которое пользователи могли бы попробовать и использовать, нам нужно было выбрать платформу для запуска всех этих компонентов. Нам была нужна платформа, которая позволила бы легко развернуть все программное обеспечение, включая Polybase, Hadoop и Spark; обеспечить управляемость и безопасность, гибкое масштабирование и высокую доступность. Казалось

логичным использовать контейнеры, учитывая простоту их развертывания, и в SQL Server 2017 мы реализовали поддержку SQL Server в контейнерах. Следующим шагом был выбор Kubernetes в качестве платформы для создания кластера, в котором работают эти контейнеры. Kubernetes быстро приобретал популярность как платформа, поддерживающая распределенные вычисления и масштабируемую производительность. Мы знали, что Linux является предпочтительной ОС для работы с системами Kubernetes и Hadoop, и, поскольку SQL Server уже работает на Linux, он хорошо подходит для создания такой платформы.

Итак, в конце 2017 года наша команда приступила к созданию кластера Aris, который позволил бы реализовать концепцию виртуализации данных и при этом интегрироваться с технологиями больших данных, такими как Spark и HDFS. С самого начала наша команда решила, что все это нужно поставлять как единое решение. Это означает, что если вы приобретаете SQL Server, мы устанавливаем все эти компоненты как часть лицензии (не знаю, будет ли это отдельным видом лицензии, но все это будет включено в SQL Server). Конечный продукт, который вы видите сейчас, –SQL Server 2019 и то, что мы называем *кластерами больших данных* (Big Data Clusters, BDC), – включает в себя гораздо больше, чем ранние прототипы Aris, но идея и концепция, положенные в их основу, одинаковы: предоставить простую в развертывании платформу виртуализации данных, обеспечивающую масштабируемую производительность, безопасность и управляемость.

«Сиэтл» становится SQL Server 2019

В то время как концепция кластеров Aris и больших данных была масштабной, инновационной и, честно говоря, немного пугающей, каждая новая версия SQL Server включала усовершенствования в нескольких областях. Это производительность, безопасность и доступность, три области, которые Конор Каннингем (Conor Cunningham) часто называет «жизненно важными аспектами SQL Server». Наша команда также включила возможность работы с SQL Server на Linux начиная с версии SQL Server 2017. Несмотря на то что первая версия SQL Server для Linux была впечатляющей, оставалось еще несколько возможностей, имеющихся в версии SQL Server для Windows, которые еще не были реализованы для Linux и которые также было необходимо добавить в версию для Linux. Мы также знали, что у контейнеров большое будущее – в том смысле, что они являются перспективным направлением для развертывания и запуска приложений, и SQL Server – не исключение из этого правила. Таким образом, в этом направлении необходимо было проделать определенную работу, включая исследование новых сценариев с кластерами Kubernetes (а не только решение, позволяющее создавать кластер больших данных).

В такой продукт, как SQL Server, вносят свой вклад множество команд разработчиков. Наша команда Enterprise (также известная как Tiger Team)

получила информацию о множестве новых возможностей, которые они хотели видеть в новой версии и представляющих реальную ценность для клиента (потому что именно это и является основой разработки новых решений!). Наши друзья, которые изобретают новые возможности для повышения производительности, доступности и безопасности Azure SQL Database, хотели бы увидеть плоды своего труда в проекте «Сиэтл», поскольку служба Azure и SQL Server используют одно и то же ядро баз данных. Увидев все это в 2017 году, я смог угадать момент исторического релиза.

Когда закончился 2017 год, мы все были настроены на выпуск следующей версии SQL Server, SQL Server 2018. Я видел в этом смысл. Мы выпустили две версии SQL Server за последние годы, SQL Server 2016 и SQL Server 2017, так почему бы не выпустить SQL Server 2018?

Конор Каннингем (Conor Cunningham), наш архитектор продукта, сказал мне, что благодаря нашему гибкому подходу к разработке мы можем выпускать новую версию SQL Server хоть каждый месяц, если захотим. И мы можем делать это, не снижая качества выпускаемого продукта. Конечно, мы этого не делаем, потому что хотим выпускать новые версии SQL Server, обеспечивающие должный уровень качества и имеющие гораздо большую ценность для наших клиентов, чем предыдущие. Когда мы начали продвигаться вперед в работе над нашим проектом в наступившем 2018 году, нам пришлось решить, хотим ли мы выпустить новую версию в этом году. Посмотрев на те новые возможности, которые мы могли бы включить в эту новую версию, включая кластеры больших данных, весной 2018 года мы приняли решение о выпуске нашей первой предварительной версии SQL Server vNext в 2018 году. (Когда мы не знаем точного названия, которое будет иметь следующая официальная версия, даже если у нас уже есть название проекта, например «Сиэтл», мы называем эту будущую версию «vNext».) И вы, возможно, заметили, что мы часто пытаемся делать объявления о планируемых важных релизах на больших мероприятиях. Если посмотреть на календарь, одним из крупнейших глобальных событий для клиентов Microsoft была и остается конференция Microsoft Ignite (проводимая в 2018 году в Орландо и собравшая около 30 000 человек). Поэтому летом 2018 года наше руководство решило включить предварительную демонстрацию возможностей SQL Server vNext в программу Microsoft Ignite и назвать эту версию SQL Server 2019. Это означает, что мы выпустим версию GA (данная аббревиатура означает «в общем доступе», General Availability) где-то в 2019 календарном году.

Подобный подход разделяли все члены нашей команды. Это дало нам больше возможностей для внедрения кластеров больших данных, а также больше возможностей для «традиционного» SQL Server, в основу которых легли отзывы и опыт клиентов. Моя задача в этом проекте – евангелизация и демонстрации SQL Server 2016 и 2017, чтобы показать нашим клиентам, отрасли и сообществу, что мы действительно создали современную платформу данных в SQL Server 2019.

Модернизация базы данных с помощью SQL Server 2019

На рис. 1.2 показана основная иллюстрация, с которой я обычно начинаю рассказ про SQL Server 2019. Созданная моей коллегой в Microsoft, маркетологом Дебби Лайонс (Debbi Lyons) (вы, возможно, видели, как мы с Дебби иногда появлялись вместе, рассказывая про SQL Server), она представляет собой полную картину новой современной платформы данных SQL Server 2019.

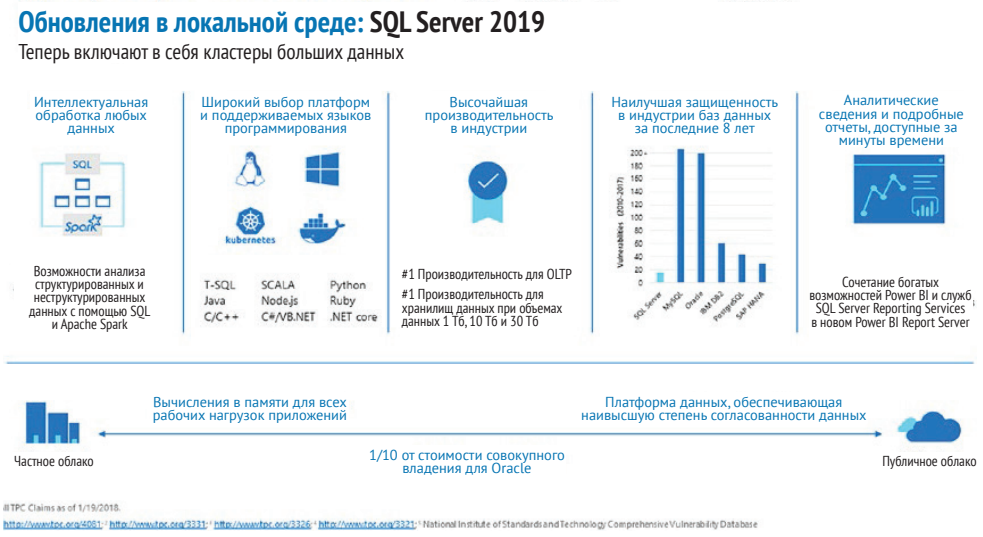


Рис. 1.2. Модернизация средствами SQL Server 2019

Если вы когда-нибудь слышали мои выступления, посвященные SQL Server 2016 или 2017, то, вероятно, заметили, что этот слайд немного похож на слайды, демонстрирующие возможности прошлых версий. Однако в нем имеется несколько ключевых отличий:

- интегрированное решение для виртуализации данных, объединяющее Spark, HDFS и SQL Server новым и инновационным способом (образно говоря, SQL Server «подружили с большими данными»);
- новые возможности использования платформы для наших клиентов, включающие разнообразие вариантов для выбора: Windows, Linux, контейнеры и Kubernetes.

SQL Server продолжает оставаться лидером в отрасли баз данных по производительности и является наименее уязвимой платформой данных за последнее десятилетие. Лицензия SQL Server включает для наших клиентов доступ к службам бизнес-аналитики, таким как Power BI Report Server.

Кроме того, благодаря новой службе Azure SQL Database Managed Instance функциональность SQL Server практически не отличается при развертывании как в частном облаке, так и в публичном облаке Azure. Однако, достигнув такого уровня согласованности, мы не останавливаемся на этом. Ваши навыки в области T-SQL применимы как к SQL Server, так и к Azure, а наши инструменты продолжают бесперебойно работать со службами SQL Server и Azure Data.

Еще несколько возможностей, немного обойденных вниманием при обсуждении новых функций, заключаются в том, что SQL Server (и Azure) предоставляет возможность оптимизировать определенные функции для обработки данных в оперативной памяти, что позволяет максимально эффективно использовать вычислительные ресурсы. Эти новые возможности включают оперативную транзакционную обработку данных в оперативной памяти (In-Memory OLTP) и столбцовые индексы (Columnstore Indexes). Все эти функции входят в состав версии SQL Server 2019. На рис. 1.3 представлена более подробная схема основных новых функциональных возможностей, впервые появившихся в SQL Server 2019.

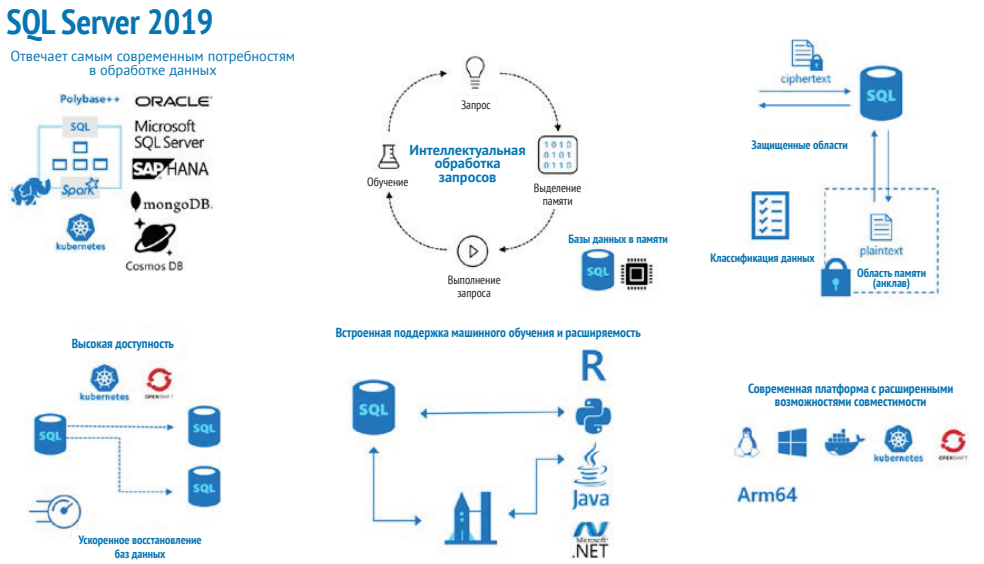


Рис. 1.3. Основные возможности SQL Server 2019

Я буду использовать эту диаграмму (слева направо, начиная с верхнего левого угла), чтобы выделить основные новые функции SQL Server 2019. Опираясь на них, можно составить план для чтения глав этой книги. Читая про эти новые возможности, имейте в виду, что SQL Server поддерживает Azure SQL Database, а это означает, что многие функции, о которых вы читаете в этой книге, работают точно так же в Azure SQL Database. Кроме того, все примеры, которые вы видите в этой книге, можно выполнить в

Azure, независимо от того, работаете ли вы с SQL Server на виртуальной машине Azure или используете контейнеры и Kubernetes в облаке.

Виртуализация данных

Ранее в этой главе мы обсуждали идею виртуализации данных в проекте Aris. SQL Server 2019 является реализацией этого подхода благодаря двум возможностям:

- **Polybase в SQL Server 2019**.

  Я называю эту возможность Polybase ++, потому что мы расширили функциональность Polybase, поставляемую с SQL Server 2016 (дополнительную информацию о Polybase см. по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-guide?view=sql-server-2017), чтобы предоставить различные коннекторы к источникам данных, включая Oracle, SQL Server, MongoDB (CosmosDB) и Teradata. Вы можете подключиться к этим источникам данных без установки какого-либо дополнительного клиентского программного обеспечения; встроенные возможности подключения SQL Server к этим источникам данных содержат все, что вам необходимо. Кроме того, вы можете подключиться к другим источникам, таким как SAP HANA, установив собственный драйвер ODBC. О новых возможностях Polybase в SQL Server 2019 будет рассказываться в главе 9;

- **кластеры больших данных**.

  Как я уже упоминал, описывая наше видение проекта Aris ранее в этой главе, мы решили создать законченное решение, которое развертывает SQL Server, включающий новые функциональные возможности Polybase, HDFS, Spark, а также другие компоненты для управления, безопасности и доступности. Их гораздо больше, чем я могу описать в этой главе; больше о кластерах больших данных можно узнать из главы 10.

Примечание. Изначально я планировал начать обсуждать эти темы во второй и третьей главах книги. Однако позже я решил, что если вам понадобится дополнительная информация о контейнерах и Kubernetes, вы можете прочитать специальную главу, посвященную этой теме. Поэтому я посвящу отдельную часть книги именно этим нововведениям. Если вам нужна информация по данной теме, перейдите сразу к главе 9.

Производительность

Какую бы версию SQL Server мы ни выпускали, мы всегда работаем над производительностью. Всегда. Однако недостаточно просто сделать так, чтобы ваши запросы выполнялись быстро. Нам нужно продолжать совершенствовать ядро SQL Server, делая его умнее и интеллектуальнее, предо-

ставляя возможности адаптации к вашей рабочей нагрузке, оборудованию и сложным шаблонам запросов. В главе 2 во всех подробностях рассматриваются вопросы производительности SQL Server 2019, в том числе следующие темы:

- *интеллектуальная обработка запросов*, которая является усовершенствованной возможностью адаптивной обработки запросов, представленной в SQL Server 2017;
- *анализ плана запросов в любом месте и в любое время*, когда вам это нужно, с использованием облегченного профилирования запросов, плана последнего исполнения и усовершенствованного хранилища запросов;
- ряд усовершенствований для поддержки *баз данных в памяти*, включая усовершенствованный ввод-вывод и гибридный буферный пул для постоянной памяти, а также оптимизированную с точки зрения памяти схему базы данных tempdb. Сочетание этих технологий с такими поддерживаемыми функциями, как столбцовые индексы и OLTP в памяти, делает SQL Server 2019 привлекательным решением с точки зрения технологий баз данных в памяти.

Безопасность

SQL Server является не только наименее уязвимой системой управления базами данных в отрасли за последнее десятилетие, но и включает в себя широкий спектр инструментов, отвечающих современным требованиям безопасности любого бизнеса. В SQL Server 2019 к ним относятся следующие улучшения:

- стратегия шифрования, названная **Always Encrypted with Secure Enclaves** (постоянным шифрованием с защищенными областями).

  В SQL Server 2016 впервые была представлена новая комплексная система безопасности для приложений данных под названием *постоянное шифрование* (Always Encrypted). Хотя эта система обеспечивает шифрование в обычном режиме, в оперативной памяти и во всем периметре сети, у нее существует несколько ограничений, наиболее важное из которых – *сложные вычисления*. В главе 3 я расскажу о том, как постоянное шифрование, использующее концепцию *защищенных областей* (Secure Enclaves), поддерживает сложные вычисления, а также приведу другие интересные сценарии из области безопасности;

- **классификация данных и встроенный аудит**.

  Общий регламент по защите данных (General Data Protection Regulation, GDPR) вступил в силу в Европейском союзе (ЕС) в мае 2018 года. С тех пор я общался со многими клиентами из ЕС, а также

с компаниями, которые работают с клиентами из ЕС. Наши новые встроенные функции классификации и аудита данных в сочетании с нашими инструментами могут быть очень полезны для сценариев, обеспечивающих соответствие требованиям законодательных актов, как GDPR, так и других, которые могут иметь отношение к вашему бизнесу.

Об этих новых функциях и о безопасности рассказывается в главе 3.

Непрерывная доступность, соответствующая требованиям для систем, критичных к сбоям

Быстрота и безопасность – это еще не все, что требуется от современной платформы данных. Клиентам, выбирающим SQL Server для ведения своего бизнеса, нужна непрерывная доступность как данных, так и работающих с ними приложений. SQL Server 2019 включает в себя новые возможности для удовлетворения требований к доступности, в том числе:

- возобновляемое создание индексов онлайн (Resumable Online Create Index) и кластеризованного онлайн-хранилища столбцовых индексов (Clustered Columnstore Online Create Index). Создание индексов поддерживается благодаря высокой доступности онлайн-индексов;
- улучшение нашей инновационной функции HADR в режиме непрерывной доступности. Поддержка групп, в том числе увеличение количества реплик и перенаправление основного соединения;
- представьте себе мир, в котором откат транзакции происходит немедленно, а время, необходимое для восстановления и сокращения журнала, не зависит от длительных или медленных транзакций. Такие невиданные доселе возможности обеспечивает ускоренное восстановление баз данных (Accelerated Database Recovery).

Подробнее об этих и других критически важных функциях, повышающих доступность приложений, рассказывается в главе 4.

Современная платформа разработки

Может показаться, что все те новые возможности SQL Server 2019, о которых я рассказывал до сих пор, предназначены только для администраторов баз данных или ИТ-специалистов. Однако это не так. Мы считаем, что разработчики – это ключевая аудитория, которой SQL Server обязан своим успехом, поэтому мы также инвестировали в следующие новые функции:

- в SQL Server 2016 мы представили новую платформу для машинного обучения в базе данных с использованием языка R. В SQL Server 2017 мы усовершенствовали эту модель, добавив поддержку Python. Используя эту же инфраструктуру, мы теперь позволяем разработчикам расши-

рять язык T-SQL с помощью классов Java. Фактически мы создали SDK расширяемости, чтобы другие языки могли быть частью SQL Server;

- мы расширили возможности графовой базы данных, которая впервые была представлена в SQL Server 2017, и реализовали новые функции для обеспечения этой новой возможности, такие как ограничения ребер и поддержка MERGE;
- мы хотим, чтобы разработчики использовали типы данных Unicode, поэтому добавили новые представления (т. е. правила для сравнения символов в наборе) UTF-8, которые могут помочь разработчикам управлять данными UTF-8, не увеличивая накладные расходы при работе с типами данных Unicode.

Я подробнее расскажу о функциях, ориентированных на разработчиков, в SQL Server 2019 в главе 5.

Инвестирование в выбранную вами платформу

Мы запустили SQL Server на Linux в версии SQL Server 2017, но у нас было несколько функций ядра СУБД, которые не вошли в эту версию. Однако мы стремимся к тому, чтобы наши пользователи выбирали операционную систему для запуска SQL Server, не беспокоясь об отсутствии некоторых возможностей в версии для Linux, и в совместимости версий SQL Server для различных операционных систем. Мы расширили возможности версии для Linux в SQL Server 2019, добавив репликацию, отслеживание изменений в базе данных (Change Data Capture, CDC), распределенные транзакции (Distributed Transactions, DTC), машинное обучение и группы Polybase.

Мы также потратили немало сил и средств, чтобы добавить в новую версию контейнеры, включая новый реестр контейнеров, поддержку Red Hat Enterprise Linux (RHEL) и постоянную поддержку Kubernetes, в том числе OpenShift. И хотя это выходит за рамки данной книги, стоит особо упомянуть, что мы расширили список поддерживаемых аппаратных платформ для SQL Server, когда в мае 2019 года объявили о выпуске предварительной версии с поддержкой процессоров Arm для Azure SQL Database Edge. Дополнительную информацию об Azure SQL Database Edge вы найдете на странице https://azure.microsoft.com/en-us/services/sql-database-edge/.

Остановитесь и рассмотрите все пиктограммы, иллюстрирующие поддерживаемые платформой компоненты и технологии, потому что SQL Server – это не просто выбранная вами платформа. Это платформа, обладающая широкими возможностями в плане *совместимости*. Вы можете создать резервную копию базы данных на любой из этих платформ и восстановить ее без изменений на любой из этих платформ.

В главах 6, 7 и 8 книги мы рассмотрим следующие возможности SQL Server: улучшения для Linux, контейнеры в SQL Server и SQL Server в Kubernetes.

Однако, кроме этих основных точек приложения усилий и вложения средств, в SQL Server 2019 существуют и другие новые возможности, которые стоит упомянуть.

Azure Data Studio

Основным инструментом – графическим интерфейсом пользователя для работы с SQL Server – на протяжении многих лет остается SQL Server Management Studio (SSMS). В прошлом году мы приступили к созданию нового инструмента для исследования и расширенной обработки данных под названием SQL Operations Studio. В сентябре 2018 года мы выпустили его официальную версию и назвали ее Azure Data Studio (ADS).

Azure Data Studio имеет несколько инновационных технологий, включая записные книжки, развертывание кластера больших данных, мастера внешних данных, а также инструменты для поддержки SQL Server, HDFS и других служб данных Azure.

Специальной главы, посвященной Azure Data Studio, в этой книге нет. Вместо этого вы увидите, как я использую этот инструмент (наряду с SSMS и другими) на протяжении всей этой книги.

Голос клиента

Имея опыт работы с клиентами, я всегда заинтересован в том, чтобы наша команда инженеров включала в новые версии те функции, которые можно привязать к прямой обратной связи клиентов или к проблемам, с которыми сталкивается служба поддержки нашей команды CSS.

Эта версия, следуя традициям общего подхода к выпуску версий, включает в себя ряд усовершенствований ядра базы данных. Ниже перечислены некоторые из них (однако далеко не все):

- переработанное сообщение об ошибке, связанной с «обрезанием» строк, включающее передачу контекста, который помогает устранить ошибку. Это был № 1 в голосовании клиентов; данное улучшение собрало 1000 голосов;
- новые динамические объекты управления, позволяющие получить представление о внутренней структуре заголовков страниц базы данных (да, вы тоже можете побыть Полом Рэндалом). Они могут помочь в устранении проблем, связанных с блокировкой страниц;
- улучшения масштабируемости в ядре СУБД, включая параллельные обновления PFS, параллельное массовое добавление данных и дополнительные контрольные точки.

Более подробно об этой серии улучшений будет рассказываться в главе 11.

Когда вы ознакомитесь с остальными главами этой книги, то заметите, что эти главы относительно независимы друг от друга. Тем не менее я настоятельно рекомендую сначала прочитать главы 7 и 8 как закладывающие основу, а затем перейти к главам 9 и 10, посвященным виртуализации данных и кластерам больших данных.

Начало работы с SQL Server 2019

Вот некоторые ресурсы, которые помогут вам развернуть и настроить SQL Server 2019 в процессе подготовки к изучению новых функций и разбора примеров, приведенных в последующих главах этой книги.

Загрузка SQL Server 2019

Чтобы загрузить и попробовать поработать с SQL Server 2019, перейдите по ссылке www.microsoft.com/en-us/sql-server/sql-server-2019#Install.

Развертывание SQL Server 2019

Инструкции по развертыванию SQL Server 2019 для Windows см. по ссылке https://docs.microsoft.com/en-us/sql/database-engine/install-windows/installation-for-sql-server?view=sql-server-ver15.

Инструкции по развертыванию SQL Server 2019 на Linux см. по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-overview?view=sql-server-ver15.

Чтобы узнать, как развернуть SQL Server в контейнере, перейдите по ссылке https://docs.microsoft.com/en-us/sql/linux/quickstart-install-connect-docker?view=sql-server-linux-ver15&pivots=cs1-bash.

Миграция на SQL Server 2019

В главе 11 будет обсуждаться миграция и инструменты для поддержки миграции на SQL Server 2019 с предыдущих версий SQL Server и других баз данных, разработанных сторонними компаниями-разработчиками ПО.

Что нового в SQL Server 2019

Обо всех новых возможностях SQL Server 2019 вы можете узнать, перейдя по ссылке https://docs.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-ver15?view=sqlallproducts-allversions.

Исходный код и базы данных, используемые в книге

Чтобы иметь возможность работать со всеми примерами, приведенными в этой книге, вам нужно будет клонировать репозиторий GitHub, созданный специально для этой цели, как было упомянуто во введении.

> **Совет.** Для пользователей Windows: обязательно используйте следующий синтаксис git для клонирования репозитория, во избежание проблем со сценариями CRLF для Linux:
>
> ```
> git clone --config core.autocrlf=false https://github.com/microsoft/
> sqlworkshops.git
> ```

Кроме того, вам нужно загрузить копии баз данных WideWorldImporters со страницы https://github.com/Microsoft/sql-server-samples/releases/tag/wide-world-

importers-v1.0 и WideWorldImportersDW со страницы https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImportersDW-Full.bak. Код, сопровождающий эту книгу, содержит примеры того, как восстановить резервную копию в Windows, Linux, Containers и Kubernetes.

SQL Server: материалы для изучения

В этой книге содержится много практических упражнений; однако на сайте http://aka.ms/sqlworkshops вы найдете еще больше бесплатных обучающих курсов, связанных с SQL Server (этот сайт развивает и поддерживает мой друг и коллега Бак Вуди (Buck Woody), один из лучших инструкторов, которых я знаю).

Это SQL Server вашего дедушки?

Мне очень понравилось работать над этой книгой не только потому, что мне нравится технология, о которой в ней рассказывается (я признаюсь в слабости к SQL Server), но и потому, что наша команда инженеров внедряет инновации со скоростью, невиданной ни для одного другого конкурирующего продукта или платформы данных в отрасли. И нужно признаться: очень интересно изучать новые вещи.

Возможно, следующая цитата из журнала *ITProToday* лучше всего расскажет об этом: «Я никогда не ожидал, что буду обсуждать возможности новой версии Microsoft SQL Server в том же контексте, что и Linux, Oracle и Apache Spark, но это дивный новый мир. Разработка Microsoft SQL Server продвигается такими темпами, которых нет у конкурентов» (www.itprotoday.com/sql-server/polybase-expansion-big-clusters-are-key-features-new-sql-server-2019).

Я помню, как мой коллега Трэвис Райт (Travis Wright) говорил о SQL Server 2019: «Это не SQL Server вашего дедушки». Это было сказано к тому, что продукт превратился из мощной системы управления реляционными базами данных в современную платформу данных и в настоящее время включает в себя такие технологии, как Spark, HDFS, Notebooks, Polybase, R, Python, Java, Linux, контейнеры и Kubernetes.

Я помню, как разместил эту цитату в Твиттере. Мой коллега Педро Лопес (Pedro Lopes) прочитал ее и сказал, что на самом деле SQL Server 2019 – это «SQL Server вашего дедушки». Так кто же прав? Они оба правы. SQL Server 2019 по-прежнему является системой управления базами данных, которую вы хорошо знаете и любите, – СУБД, обладающей масштабируемой производительностью, встроенными механизмами безопасности, удовлетворяющими требованиям критически важных приложений и систем, и высокой доступностью. В этой книге вы познакомитесь с новыми возможностями SQL Server 2019 во всех этих областях. Но SQL Server 2019 – это нечто намного большее. Это одна из самых популярных платформ баз данных на планете и новейший продукт в своем семействе. Он может быть и тем, и другим. Давайте начнем знакомиться с SQL Server 2019!

Глава 2

Интеллектуальная настройка производительности

Производительность SQL Server имеет решающее значение для работы платформы данных. В этой главе рассказывается о том, как SQL Server 2019 может помочь вам повысить производительность запросов, не меняя ничего в приложении. Это одна из самых длинных глав в книге со множеством примеров, так что наберитесь терпения и налейте свой любимый кофе.

Почему используется термин «интеллектуальная настройка производительности»?

Самое главное в этой книге – это ответ на вопрос, каким образом новые возможности в SQL Server 2019 могут принести вам пользу или решить определенную задачу либо проблему. Тема, касающаяся повышения производительности, включает способы, которые могут помочь вам повысить производительность ваших рабочих приложений, часто без необходимости каких-либо изменений в этих приложениях или связанных с ними запросах.

В сентябре 2018 года я готовился к презентации на конференции Microsoft Ignite в Орландо (штат Флорида). До этого момента все знали о наших планах относительно SQL vNext только в общих чертах. У меня и моего коллеги Амита Банерджи (Amit Banerjee) была задача продемонстрировать запуск SQL Server 2019 Preview на Ignite. Когда мы работали над стендом для демонстрации, мы знали, что нам предстоит продемонстрировать наше новое достижение – улучшенную производительность. У Амита появилась идея нового термина «Интеллектуальная база данных». Идея заключалась в том, что SQL Server предоставляет новые, ранее недоступные возможности, которые включают в себя интеллектуальные средства для поиска, адаптации и представления обобщенных аналитических данных.

Я использовал тот же термин, перенеся акцент на производительность, назвав это интеллектуальной настройкой производительности. Под этим термином мы объединяем следующие улучшения в SQL Server 2019:

- интеллектуальная обработка запросов;
- упрощенное профилирование запросов;
- база данных в памяти;
- конфликт вставки на последней странице.

Каждая из этих новых возможностей использует встроенный интеллектуальный механизм SQL Server, помогающий повысить производительность ваших систем, во многих случаях позволяя обойтись без каких-либо изменений. В ряде случаев SQL Server дает вам представление о производительности запросов на таком уровне, который ранее был недоступен. В других ситуациях SQL Server включает встроенные возможности для автоматического использования преимуществ новых аппаратных средств.

Когда вы пишете книгу, то принимаете множество разных решений. Одно из них – как разбить книгу на главы. Эта глава очень длинная, в основном из-за примеров, которые содержат много визуальных элементов. Я человек, восприимчивый к визуальной информации, поэтому я подумал, что визуализация упростит задачу показать вам эти новые функции. Каждый раздел этой главы сам по себе является отдельной главой, и вы можете рассматривать эту главу как собрание мелких глав. Я решил объединить всю информацию в одной главе, потому что хотел продемонстрировать все возможности в полном объеме и показать интеллектуальную настройку производительности в SQL Server 2019.

В каждом разделе главы перечислены предварительные условия для запуска каждого из примеров. В самом общем случае вам понадобится:

- установленный экземпляр SQL Server 2019 в Windows или Linux;
- SQL Server Management Studio (SSMS) 18.0 или более поздняя версия;
- Azure Data Studio (на любой ОС; минимальная версия, с которой вы сможете выполнить все примеры: 1.7.0).

Во многих приведенных примерах для просмотра планов выполнения запросов используется SSMS; но, просматривая и выполняя эти примеры, вы также можете использовать Azure Data Studio (вам просто нужно просмотреть XML-текст плана), используя новое расширение SentryOne Plan Explorer. Подробную информацию об этом расширении вы найдете по ссылке https://cloudblogs.microsoft.com/sqlserver/2019/07/11/the-july-release-of-azure-data-studio-is-now-available.

Интеллектуальная обработка запросов

Готовясь к выпуску SQL Server 2014, наша команда инженеров приняла смелое решение написать новый программный код для обработчика запросов в ядре, который будет принимать решения при оценке кардинальности (cardinality estimation, CE). Новая «модель CE» вступит в силу, если для базы данных будет использоваться значение уровня совместимо-

сти (compatibility level) 120 или более высокое (120 является значением по умолчанию для SQL Server 2014). Вы можете прочитать о том, как это работает и почему мы внесли это изменение, в нашей документации, доступной по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server.

Это решение породило множество споров о том, было ли оно правильным. Одна из проблем данного подхода заключается в том, что это было масштабное, негибкое изменение. Когда команда заканчивала работу над SQL Server 2016 и планировала SQL Server 2017, все согласились с тем, что нужна новая функция обработки запросов. Как говорит Джо Сэк (Joe Sack), один из ведущих менеджеров программ, работающий над Query Processor (QP): «Команда поняла, что попытка отделаться локальными изменениями, которые работали бы во всем многообразии встречающихся ситуаций, – это не наш путь, если мы хотим продвигаться вперед. Скорее, нам нужно тратить силы и время на функции, позволяющие адаптироваться к широкому спектру рабочих нагрузок клиентов в экосистеме SQL Server (высокие нагрузки, низкие нагрузки, OLTP, гибридные варианты, хранилища данных…)».

Так родилось новое семейство функций в SQL Server 2017, которое называется адаптивной обработкой запросов (Adaptive Query Processing, AQP). Идея заключалась в том, чтобы встроить в обработчик запросов способность самостоятельно *адаптироваться* после выполнения запроса (или до его *повторного* выполнения), чтобы ускорить выполнение без какого-либо вмешательства пользователя SQL Server или изменений приложения.

> **Примечание.** Примеры AQP для SQL Server 2017 можно найти по адресу: https://github.com/Microsoft/bobsql/tree/master/demos/sqlserver/aqp.

Когда мы готовились к выпуску SQL Server 2017 с поддержкой AQP, планировалось множество новых функций, которые команда хотела добавить в AQP, однако на реализацию всех планов в полном объеме не хватило времени. Команда начала внедрять новые функции для улучшения AQP в базе данных SQL Server Azure, планируя реализовать их в SQL Server 2019. Кроме того, слово «адаптивный» на самом деле не отражало суть работы, которую проделала команда. В течение многих лет обработчик запросов SQL Server был достаточно умным – он использовал для принятия решений сложный набор алгоритмов, в основе которых лежали вычисления затрат на выполнение запроса. Но команда хотела большего; они хотели, чтобы обработчик запросов стал еще более интеллектуальным. Таким образом, название «интеллектуальная обработка запросов» (Intelligent Query Processing, IQP) прижилось.

На рис. 2.1 показано «родословное древо» возможностей обработчика запросов, где представлены и SQL Server 2017, и SQL Server 2019.

Рис. 2.1. «Родословное древо» интеллектуальной обработки запросов

Давайте рассмотрим каждую из этих новых возможностей (новые возможности выделены на рисунке серым цветом), приведя наглядные примеры того, как работает каждая из них. При чтении этого раздела очень важно помнить, что мы создали эти возможности специально, **чтобы вы не думали о них**. Если мы хорошо справились со своей работой, то со временем интеллектуальная обработка запросов станет «всего лишь» обработчиком запросов, а вы как разработчик приложений, администратор баз данных или специалист по данным просто привыкнете к гибкому, интеллектуальному и адаптирующемуся к нагрузкам вашей системы механизму. Вы можете ознакомиться со всеми возможностями адаптивной обработки запросов в новой документации по IQP по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing.

> **Примечание.** Во всех сценариях, за исключением функции приблизительного подсчета количества уникальных значений Approximate Count Distinct, можно включить возможности интеллектуальной обработки запросов, установив значение уровня совместимости базы данных, равное 150. Approximate Count Distinct – это новая функция T-SQL для SQL Server 2019, для которой не требуется устанавливать уровень совместимости базы данных, равный 150.

Подготовительные шаги для использования примеров, иллюстрирующих интеллектуальную обработку запросов

Хотя преимущества интеллектуальной обработки запросов (IQP) очевидны для многих стандартных случаев, выигрыш в производительности при использовании IQP становится заметным при работе с большими наборами данных и базами данных, предназначенными для аналитических запросов. Поэтому для примеров, приведенных в этой главе, вам нужно использовать базу данных WideWorldImportersDW (более подробную

информацию об этой базе данных и ее схеме можно получить по ссылке https://docs.microsoft.com/en-us/sql/samples/wide-world-importers-dw-database-catalog).

Приведенные в данной главе примеры будут работать на SQL Server 2019, установленном на Windows, Linux и в контейнерах. Учитывая большой объем данных, SQL Server потребуется как минимум 12 ГБ ОЗУ, чтобы различия в производительности для приведенных примеров стали заметны. Кроме того, в некоторых приведенных примерах запросов используется параллельная обработка, поэтому для демонстрации всех возможностей в полном объеме лучше установить SQL Server в многопроцессорной системе.

Все сценарии, использованные в этой главе, можно найти в репозитории GitHub в каталоге **ch2_intelligent_performance\iqp**.

Я благодарен моему коллеге из Microsoft Джо Саку (Joe Sack) за подготовленные примеры, в том числе за дополнения, внесенные в базу данных WideWorldImportersDW. Эти примеры были созданы на основе материалов, хранящихся в репозитории Джо на GitHub по ссылке https://github.com/joesackmsft/Conferences/tree/master/IQPDemos.

Чтобы использовать на практике примеры из этой главы, выполните следующие действия:

1. Загрузите резервную копию базы данных WideWorldImportersDW с сайта https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImportersDW-Full.bak.
2. Восстановите эту базу данных на вашем экземпляре SQL Server 2019. Вы можете использовать для этого готовый сценарий **restorewwidw.sql**. Возможно, вам придется изменить путь к каталогу, где находится ваша резервная копия, и путь к месту для восстановления файлов базы данных.
3. Для запуска некоторых сценариев, приводимых в качестве примера, вам понадобятся таблицы большего размера, нежели те, которые добавлены по умолчанию в WideWorldImportersDW и которые не используют столбцовые индексы. Поэтому запустите сценарий **extendwwidw.sql**, чтобы создать две большие таблицы. Расширение этой базы данных увеличит ее размер, включая журнал транзакций, примерно до 8 Гб. Одна из этих таблиц называется **Fact.OrderHistory**. Взяв за основу таблицу **Orders**, мы намного увеличим ее размер и не будем использовать столбцовый индекс. Также мы создадим еще одну таблицу с именем **Fact.OrderHistoryExtended**. В ее основе будет таблица **Fact.OrderHistory**, однако новая таблица будет содержать большее количество строк.

Почти во всех примерах используются два метода:

- набор сценариев T-SQL, которые можно применять с любым инструментом для запуска сценариев, таким как SQL Server Management Studio, Azure Data Studio или sqlcmd;

- записная книжка T-SQL (T-SQL Notebook) в Azure Data Studio. Информацию о том, как пользоваться записной книжкой T-SQL в Azure Data Studio, вы найдете по ссылке https://docs.microsoft.com/en-us/sql/azure-data-studio/sql-notebooks.

Для выполнения одного из примеров потребуется клиент Windows, поскольку он использует известную утилиту нагрузочного тестирования ostress.exe. Подробная информация о том, как установить и использовать утилиту ostress.exe, приведена в разделе «Обратная связь по временно предоставляемому буферу памяти» этой главы. Я собрал все сценарии, предполагая, что вы будете запускать их под учетной записью системного администратора (я использовал учетную запись sa). В обычной практике вам потребуется создать другие учетные записи для использования SQL Server, но я хотел упростить примеры – поэтому просто используйте учетную запись с разрешениями sysadmin.

Обратная связь по временно предоставляемому буферу памяти (Memory Grant Feedback Row Mode)

До прихода в команду разработчиков SQL Server я долгое время работал в технической поддержке Microsoft. Одна из самых сложных проблем, с которыми я сталкиваюсь, когда имею дело с производительностью, – это проблемы с *предоставлением памяти* (memory grant). Что же такое предоставление памяти?

SQL Server выделяет память по разным причинам. Когда SQL Server выполняет запрос, память может использоваться для кеширования буферов, связанных со страницами, принадлежащими индексам или таблицам в запросе. В большинстве запущенных экземпляров SQL Server буферный пул может уже находиться в выделенной памяти, поэтому для помещения в нее страниц не потребуется дополнительная память.

Некоторые запросы интенсивно используют память, и для них требуется определенный тип временной области для хранения данных. Двумя разновидностями таких запросов являются *хеш-соединения* (hash joins) (или даже просто хеш-операторы) и *сортировки* (sorts) . Чтобы выполнить хеш-соединение, SQL Server фактически должен построить мини-таблицу в памяти. Для любого типа сортировки данных может потребоваться создание массива или структуры определенного типа. В SQL Server должно иметься место для выполнения этих операций, поэтому он выделяет память вне буферного пула. Процесс выделения этой памяти механизмом выполнения запросов называется предоставлением памяти.

Пока все выглядит достаточно просто. Однако существует проблема: предоставление памяти основано на том, что оптимизатору запроса известен план выполнения запроса даже тогда, когда этот запрос выполняется в первый раз. Решение такой задачи обычно сводится к оценке кар-

динальности, или уникального количества строк, затрагиваемых выполняемой операцией, для ее выполнения. Если SQL Server оценивает, что операция сортировки данных как часть плана выполнения запроса будет выполняться для столбцов данных, суммарный размер которых составляет 100 байт, и при этом число строк, на которых должен выполняться запрос, составит 1 миллиард, то он должен предоставить достаточно памяти, чтобы выделить память для сортировки такого количества строк данных указанного размера. Аналогичная концепция применяется для хеш-оператора.

> **Совет.** Существует блог команды разработчиков SQL Server, в котором подробно разъясняется функция предоставления памяти. Я рекомендую вам остановиться и обратиться к данному блогу, чтобы погрузиться в эту тему. Блог размещен по адресу https://blogs.msdn.microsoft.com/sqlqueryprocessing/2010/02/16/understanding-sql-server-memory-grant/.

Во многих случаях описанный подход работает просто отлично, и никаких заметных проблем не возникает. Однако что происходит, если предоставление памяти основано на неточных оценках кардинальности?

Может возникнуть два вида проблем:

- объем предоставляемой памяти может оказаться слишком мал по сравнению с действительно необходимым для выполнения операции объемом памяти, что может привести к печально известной и болезненной проблеме «утечки данных в tempdb» (tempdb spill). SQL Server не позволяет оператору хеш-соединения или сортировки получить необходимый объем памяти. Если запрошено очень много памяти (мы не указываем, какие конкретные цифры подразумеваются под «очень много», потому что можем изменить объем запрашиваемой памяти и не хотим, чтобы вы полагались на него как на абсолютную цифру), текущая выделенная память должна быть где-то сохранена. Но где? Вы наверняка догадались – в tempdb. Думайте об этом как о системе подкачки страниц, как о том, каким образом операционная система распределяет память, когда физическая память исчерпана;
- объем предоставляемой памяти может оказаться слишком велик по сравнению с действительно необходимым для выполнения операции объемом памяти. Это может снизить нагрузку на память для других элементов ядра SQL Server, однако более вероятна ситуация, когда несколько пользователей запускают запросы с чрезмерным объемом предоставляемой памяти, а SQL Server будет управлять выполнением запросов. В результате для некоторых пользователей «узким местом» окажется ожидание (wait_type) с именем RESOURCE_SEMAPHORE.

Обе эти ситуации могут привести к снижению производительности. В SQL Server 2017 мы представили концепцию, называемую обратной связью по временно предоставляемому буферу памяти для пакетного режима. Эта функция является прекрасным примером *адаптации*. После завершения выполнения запроса SQL Server обладает информацией о том, сколько памяти было в действительности использовано по сравнению с первоначально запрошенным объемом памяти. Если использованная память была намного меньше предоставленной, зачем продолжать запрашивать слишком много памяти при следующем выполнении того же кешированного плана выполнения запроса? То же самое происходит, если объем использованной памяти был намного больше, чем первоначально запрошенный объем памяти. Зачем продолжать сохранять текущую выделенную память в базе данных tempdb для кешированных запросов раз за разом?

Обратная связь по временно предоставляемому буферу памяти решает эту проблему путем сохранения в кешированном плане выполнения запросов информации о том, каким должен быть корректный объем предоставляемой памяти для будущих выполнений запроса. Пользователю при этом кажется, что SQL Server «исправился сам». Эта функция была очень полезной для SQL Server 2017, однако она поддерживалась только для операций в пакетном режиме, что означало, что она работала лишь для операций со столбцовыми индексами. Как вы узнаете из одного из следующих разделов этой главы под названием «Пакетный режим для хранилища строк», SQL Server поддерживает операции пакетного режима не только для столбцовых индексов. Однако почему бы не поддерживать обратную связь по временно предоставляемому буферу памяти, даже если пакетный режим не используется?

В результате появился адаптивный механизм SQL Server для сценариев предоставления памяти независимо от используемых типов таблиц или индексов.

Чтобы включить режим обратной связи по временно предоставляемому буферу памяти, достаточно просто изменить параметр уровня совместимости базы данных (dbcompat), установив для него значение, равное 150.

Если для параметра dbcompat установлено значение 150, вы можете также отключить или включить режим обратной связи по временно предоставляемому буферу памяти, не изменяя значение уровня совместимости базы данных. Для этого вам нужно использовать команду ROW_MODE_MEMORY_GRANT_FEEDBACK оператора ALTER DATABASE SCOPED CONFIGURATION. Вы также можете отключить эту функцию на уровне запроса, используя параметр запроса DISABLE_ROW_MODE_MEMORY_GRANT_FEEDBACK. Вы можете посмотреть примеры того, как нужно использовать эти возможности включения и отключения режима обратной связи по временно предоставляемому буферу памяти, по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?#row-mode-memory-grant-feedback.

Выделение слишком малого объема памяти

Рассмотрим несколько примеров. Давайте сначала рассмотрим сценарий, в котором SQL Server предоставляет слишком мало памяти по сравнению с фактически используемой памятью, что приводит к «утечке данных в tempdb». Все сценарии, используемые в этих примерах, можно найти в каталоге **ch2_intelligent_performance\iqp\rowmodemgf**. Есть два способа запустить примеры кода для этого сценария:

- использовать сценарий T-SQL **iqp_rowmodemfg.sql**;
- использовать записную книжку T-SQL в Azure Data Studio с именем **iqp_rowmodemfg.ipynb**.

Давайте выполним сценарий T-SQL **iqp_rowmodemfg.sql** шаг за шагом. Я буду применять для этого SQL Server Management Studio, попутно объясняя различия в плане выполнения запросов, но вы можете воспользоваться любым инструментом, отображающим план выполнения запроса. В приведенном в качестве примера сценарии T-SQL имеются комментарии для каждого шага рассматриваемого примера.

1. На **шаге 1** мы изменяем уровень совместимости базы данных, устанавливая для него значение 150, очищаем кеш процедур и заполняем буферный пул страницами из таблицы с именем Fact.OrderHistory, размещенной в базе данных WideWorldImportersDW, «разогревая» буферный пул. Для того чтобы включить обратную связь по временно предоставляемому буферу памяти для хранилища строк, необходимо установить значение параметра dbcompat, равное 150. Очистка кеша процедур – это всего лишь шаг, необходимый, чтобы убедиться, что мы «начинаем чистить». (Обратите внимание на использование опции ALTER DATABASE для очистки кеша процедур только для этой базы данных. Это очень хороший вариант!) Выгрузка страниц с диска для таблицы Fact.OrderHistory выполняется только для обеспечения сравнения производительности запросов с и без предоставления обратной связи по временно предоставляемому буферу памяти – это «честный бой».

```
-- Шаг 1. Убедитесь, что для текущей базы данных установлено значение
-- уровня совместимости 150, и очистите кеш процедур для этой базы данных.
-- Также поместите таблицу в кеш, чтобы сравнить «теплые» запросы кеша
USE [WideWorldImportersDW]
GO
ALTER DATABASE [WideWorldImportersDW] SET COMPATIBILITY_LEVEL = 150
GO
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE
GO
SELECT COUNT(*) FROM [Fact].[OrderHistory]
GO
```

2. На **шаге 2** создаются условия для предоставления недостаточного объема памяти. Я покажу вам один прием, позволяющий смоделировать такую ситуацию. Команда T-SQL UPDATE STATISTICS имеет специальный необязательный параметр для принудительного задания указанного количества строк или страниц, которые хранятся в статистике запроса. Вы вряд ли когда-либо будете использовать этот параметр при обычной работе с SQL Server. В документации, описывающей команду UPDATE STATISTICS (см. https://docs.microsoft.com/en-us/sql/t-sql/statements/update-statistics-transact-sql), об этом параметре говорится следующее: «Используется только в ознакомительных целях. Не поддерживается. Будущая совместимость не гарантируется». Так что рассматриваемый здесь вариант предназначен только для этой демонстрации. Для наших целей давайте принудительно увеличим кардинальность, хранящуюся в статистике выполнения запроса, для этой таблицы до 1000 строк:

```
-- Шаг 2. Имитация устаревшей статистики
UPDATE STATISTICS Fact.OrderHistory
WITH ROWCOUNT = 1000
GO
```

   На самом деле в этой таблице 3 702 592 строки; принудительно заданный параметр статистики, предполагающий, что в таблице всего 1000 строк, воспроизводит сценарий, когда данные статистики не синхронизированы с фактическими данными о количестве строк в таблице.

3. Перейдем к **шагу 3**. Теперь выполним наш запрос, используя таблицу Fact.OrderHistory.

```
-- Шаг 3. Запустите запрос, чтобы получить данные о заказе и наличии товара
-- При выполнении запроса НЕ выделяйте эти комментарии!
SELECT fo.[Order Key], fo.Description, si.[Lead Time Days]
FROM Fact.OrderHistory AS fo
INNER HASH JOIN Dimension.[Stock Item] AS si
ON fo.[Stock Item Key] = si.[Stock Item Key]
WHERE fo.[Lineage Key] = 9
AND si.[Lead Time Days] > 19
GO
```

   При выполнении этого запроса SQL Server пытается получить данные заказа и позиции товара. Обратите внимание на использование HASH JOIN в синтаксисе T-SQL – это необходимо, чтобы оптимизатор применял хеш-соединение. Это простой способ принудительно использовать хеш-соединение в запросе с недооцененным количеством строк с целью продемонстрировать, что при этом произойдет. Я сопроводил код запроса комментариями, однако предупреждаю

вас: очень важно, чтобы **при выполнении этого фрагмента кода T-SQL вы не включали в код комментарии**. Я обжегся на этом, когда впервые начал создавать свои демоверсии. Комментарии учитываются, когда речь идет об уникальной идентификации запроса, для которого существует кешированный план. Если при следующем выполнении запроса в нем не будет точно таких же комментариев, запросы не будут использованы повторно. В SSMS включите флаг **Include Actual Execution Plan** (Включить фактический план выполнения) (вы можете использовать сочетание клавиш **Ctrl**+**M** для его включения) перед выполнением запроса. О том, как включить этот флаг, рассказывается на следующей странице документации: https://docs.microsoft.com/en-us/sql/relational-databases/performance/display-an-actual-execution-plan.

```
-- Шаг 3. Выполните запрос, чтобы получить данные о заказе и имеющемся
-- товаре на складе
-- При выполнении запроса НЕ выделяйте эти комментарии!
SELECT fo.[Order Key], fo.Description, si.[Lead Time Days]
FROM Fact.OrderHistory AS fo
INNER HASH JOIN Dimension.[Stock Item] AS si
ON fo.[Stock Item Key] = si.[Stock Item Key]
WHERE fo.[Lineage Key] = 9
AND si.[Lead Time Days] > 19
GO
```

Этот запрос должен выполняться не менее чем за 30 секунд; он должен вернуть около 66 тыс. строк (ваши результаты могут отличаться от тех, которые приводятся здесь). Используйте режим SSMS для просмотра плана выполнения запроса. План выполнения запроса должен выглядеть примерно так, как показано на рис. 2.2.

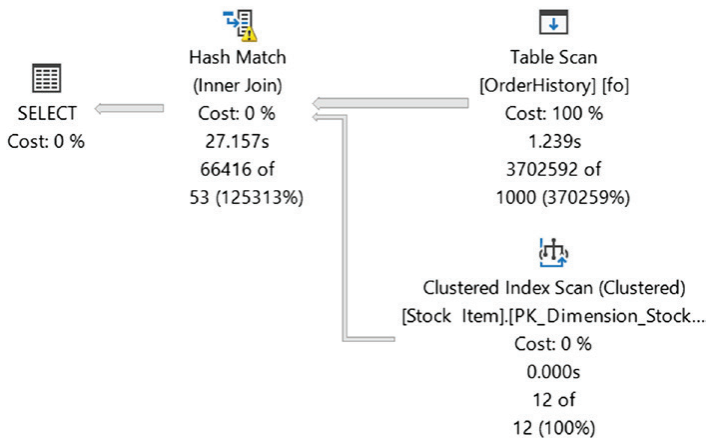


Рис. 2.2. План выполнения запроса при предоставлении недостаточного объема памяти

В этом плане есть несколько мест, на которые нужно обратить внимание. Если в SSMS вы наведете курсор мыши на оператор **Table Scan** (Сканирование таблицы), он должен выглядеть примерно так, как показано на рис. 2.3. Обратите внимание, что значение параметра **Estimated Number of Rows** (Количество строк при оценке) сильно отличается от фактического числа строк, считанных при сканировании (т. е. при последовательном просмотре строк) таблицы.

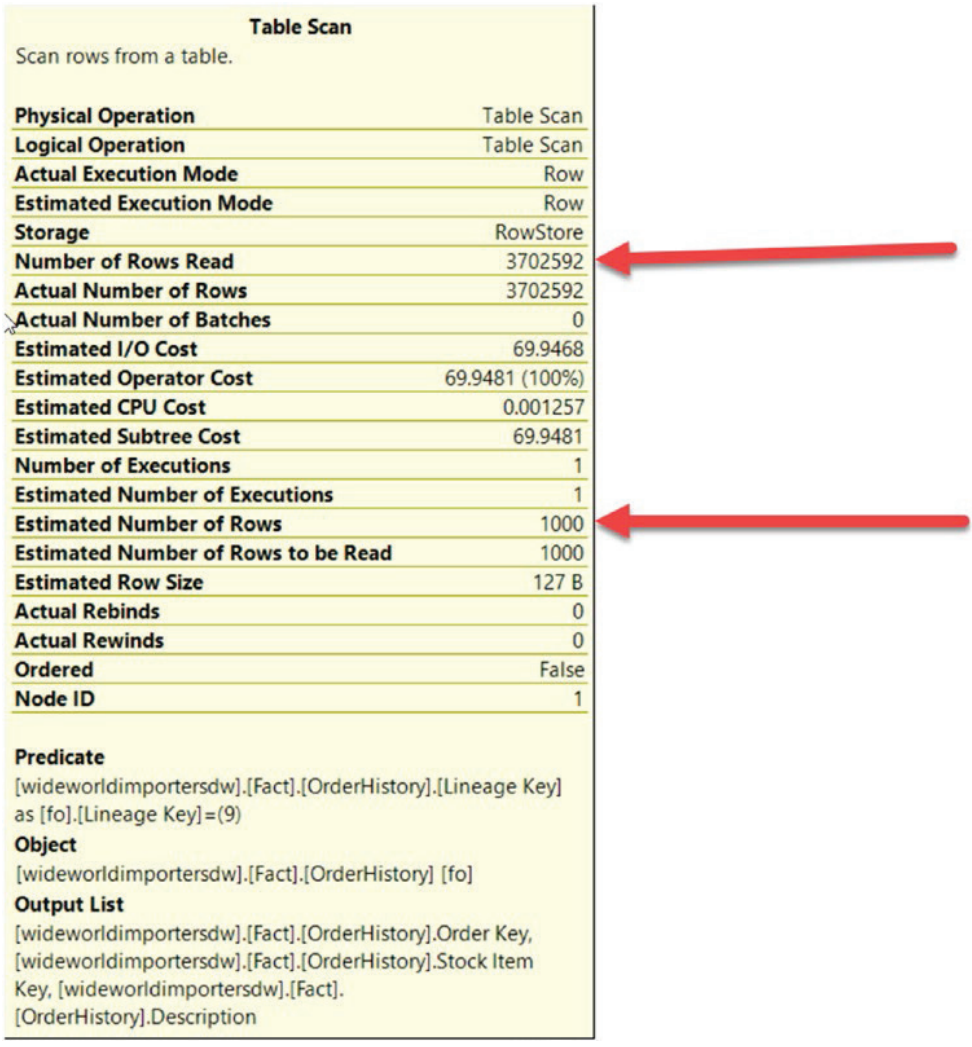


Рис. 2.3. Оценка и фактическое значение числа строк при сканировании таблицы Fact.OrderHistory

В рассмотренном нами случае таблица Fact.OrderHistory представляет собой *входные данные сборки* хеш-соединения. SQL Server будет запрашивать предоставление памяти для хеш-соединения на осно-

вании этих данных. В данном случае мы столкнемся с проблемой, поскольку предоставление памяти основано на оценке, составляющей всего 1000 строк. Перемещайте курсор мыши по строчкам данных о выполнении хеш-соединения, и вы обнаружите маленький значок предупреждения; обратите внимание на предупреждение об «утечке данных в tempdb», как показано на рис. 2.4.

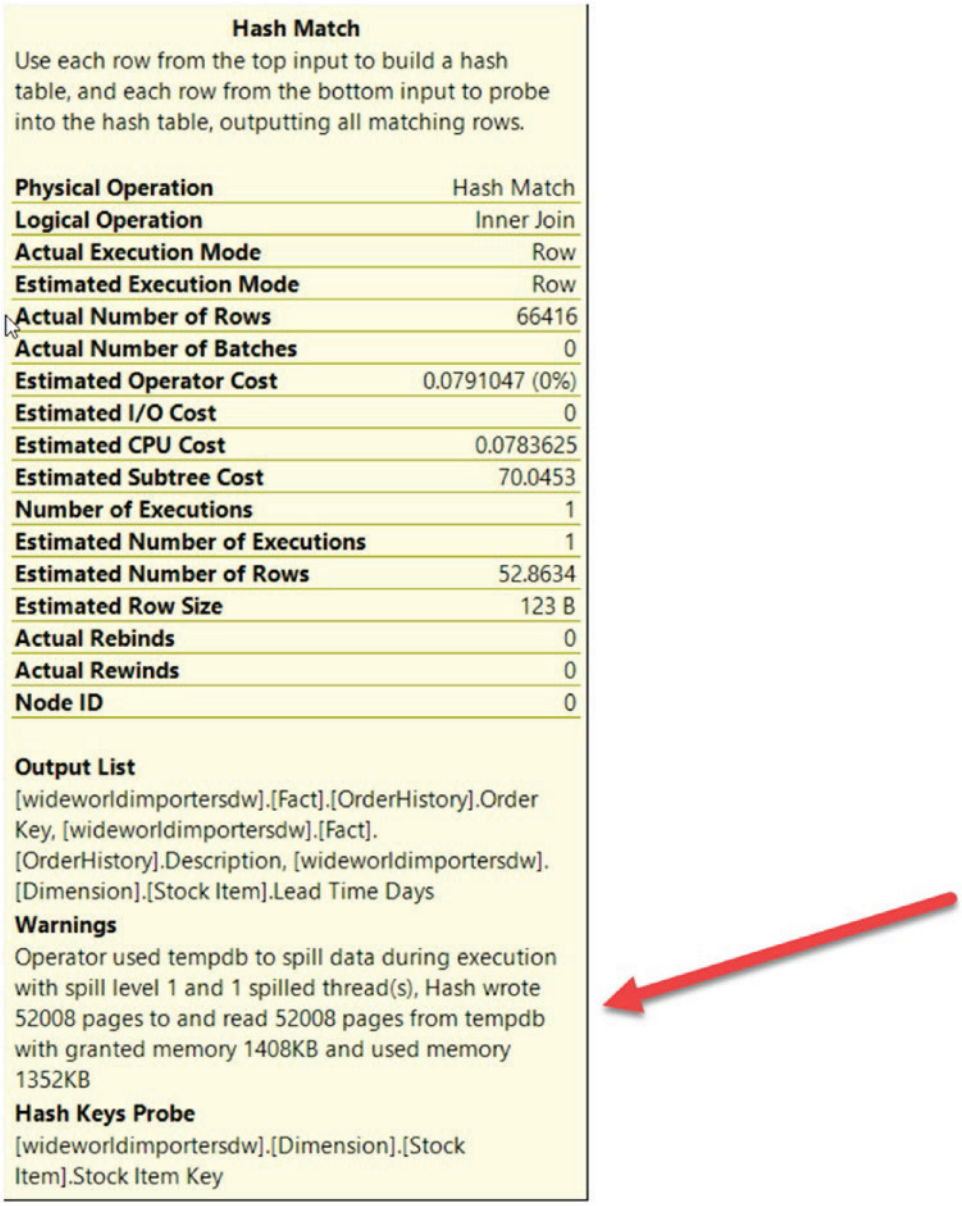


Рис. 2.4. «Утечка данных в tempdb» в хеш-соединении

Обратите внимание на цифры, приведенные в предупреждении. 52 008 страниц (8 Кб на страницу) – это ~ 426 Мб данных, используемых при выполнении операций ввода/вывода для файлов tempdb. Размеры «утечки» действительно ужасны, поскольку это не те данные, которые размещаются на странице буферного пула, связанного с tempdb. Файлы данных tempdb становятся «файлом подкачки» при предоставлении памяти для хеш-соединений (это не страницы tempdb для хранения временных таблиц, и потому я часто называю tempdb «мусорной свалкой» SQL Server).

Совет. Хотите узнать, как работает хеш-соединение? Прочитайте эту старую, но классическую статью в блоге от одного из наших ведущих инженеров группы Query Processor, Крейга Фридмана (Craig Freedman): https://blogs.msdn.microsoft.com/craigfr/2006/08/10/hash-join/.

Перемещаясь в левую часть плана выполнения запроса, наведите курсор на оператор SELECT. В этом операторе указаны сведения о количестве выделенной памяти согласно плану. На рис. 2.5 показано, что для выполнения этого запроса было запрошено ~ 1,4 Мб памяти.

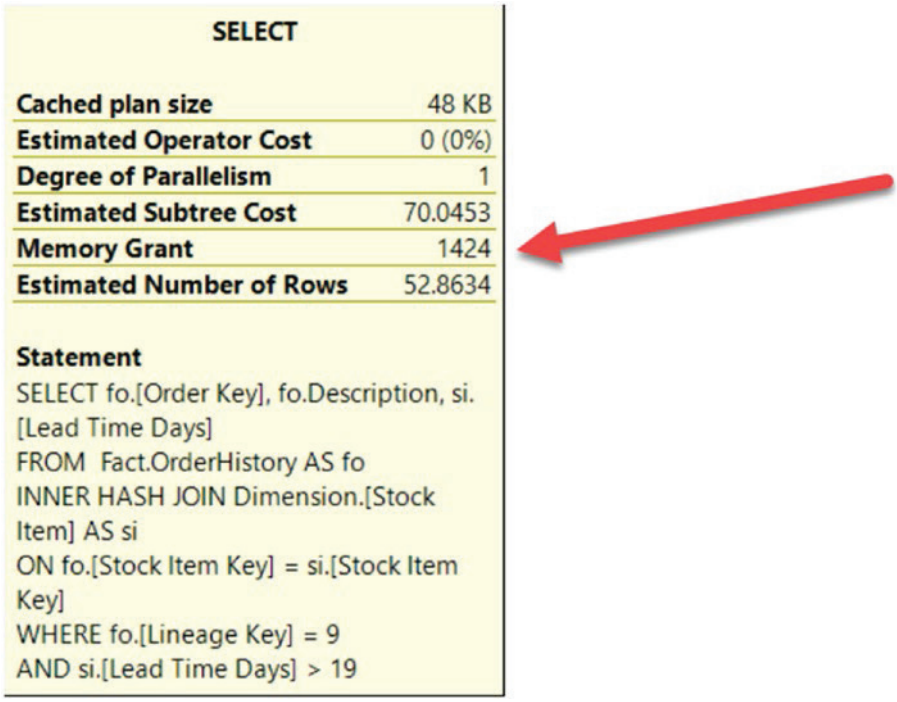


Рис. 2.5. В разделе с информацией об операторе SELECT показано запрошенное количество памяти

Запрашиваемого объема памяти 1,4 Мб недостаточно, чтобы разместить там все необходимые данные (в соответствии с информацией об «утечке данных в tempdb» необходимый объем памяти составляет ~ 400 Мб).

Еще одни интересные данные, представленные в плане выполнения запроса в виде XML-файла, находятся в разделе свойств плана. Чтобы просмотреть их, щелкните правой кнопкой мыши оператор **SELECT** и выберите **Properties** (Свойства). Раскройте раздел **MemoryGrant**, который будет выглядеть, как показано на рис. 2.6.

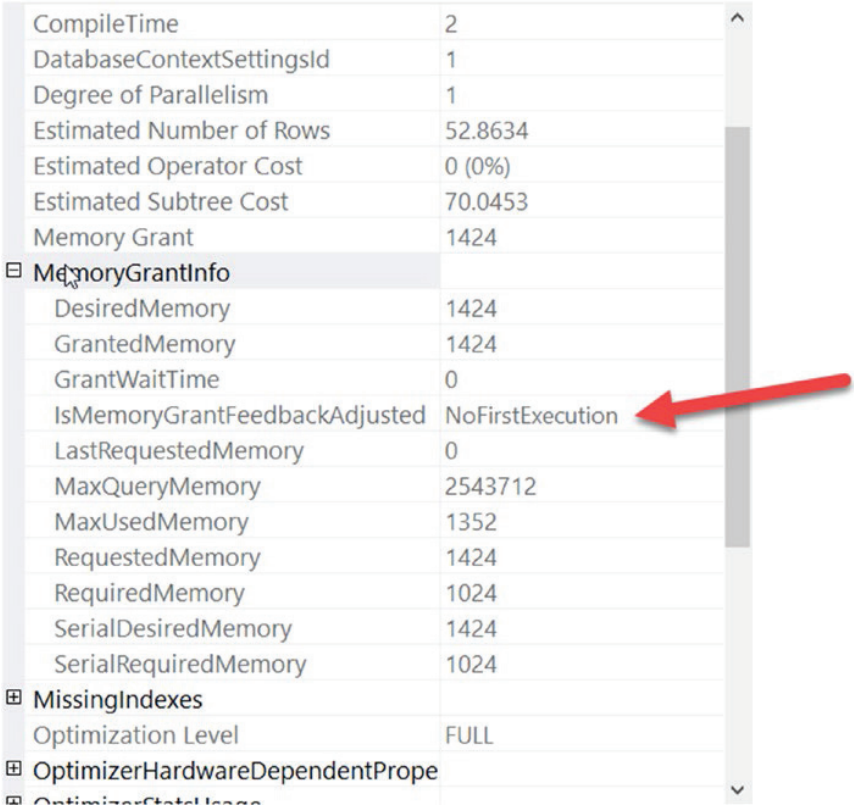


Рис. 2.6. Данные о выделении памяти приводятся в свойствах плана выполнения запроса

Наиболее важная информация в обратной связи по временно предоставляемому буферу памяти содержится в поле **IsMemoryGrantFeedbackAdjusted**. Значение *NoFirstExecution* означает, что это только первое выполнение запроса, поэтому никакой обратной связи не было получено.

Список возможных значений в этом поле приведен в документации по SQL Server (см. https://docs.microsoft.com/en-us/sql/relational-databases/

performance/intelligent-query-processing?view=sql-server-ver15#row-mode-memory-grant-feedback).

Поскольку обратная связь по временно предоставляемому буферу памяти включена, если тот же самый кешируемый запрос будет выполняться повторно, SQL Server адаптирует план выполнения запроса и изменит объем предоставляемой памяти с учетом недооценки, имевшей место в прошлый раз.

4. Перейдем к **шагу 4** нашего сценария и снова выполним тот же самый запрос.

   ВАЖНО: не включайте комментарии в текст сценария при выполнении запроса. Комментарии учитываются при сравнении повторно выполняемого запроса с первоначальным запросом в кеше плана. Обязательно сохраните включенным флаг **Include Actual Execution Plan** (Включить фактический план выполнения) в SSMS.

```
-- Шаг 4. Давайте попробуем снова
-- При выполнении запроса НЕ выделяейте эти комментарии!
SELECT fo.[Order Key], fo.Description, si.[Lead Time Days]
FROM Fact.OrderHistory AS fo
INNER HASH JOIN Dimension.[Stock Item] AS si
ON fo.[Stock Item Key] = si.[Stock Item Key]
WHERE fo.[Lineage Key] = 9
AND si.[Lead Time Days] > 19
GO
```

   Если в прошлый раз выполнение запроса занимало 30 или более секунд, то на этот раз запрос должен выполняться за 3 секунды или менее. Помните, концепция заключается в том, что план не меняется; поэтому, когда вы смотрите на фактический план выполнения запроса, он должен выглядеть так же, как и в прошлый раз, за исключением того, что отсутствует значок предупреждения в разделе **Hash Match Join**, а также нет предупреждения об «утечке данных в tempdb» . Перемещая курсор в раздел оператора **SELECT**, вы увидите, насколько в данном случае объем предоставленной памяти (**Memory Grant**) отличается от предыдущего случая, как показано на рис. 2.7.

   Как мы видим, для корректного выделения памяти для размещения хеш-соединения на самом деле требуется ~ 625 Мб.

   Щелкните правой кнопкой мыши оператор **SELECT** и выберите **Properties** (Свойства). Раскройте раздел **MemoryGrantInfo**, который теперь будет выглядеть, как показано на рис. 2.8. В этом разделе содержится обратная связь по временно предоставляемому буферу памяти.

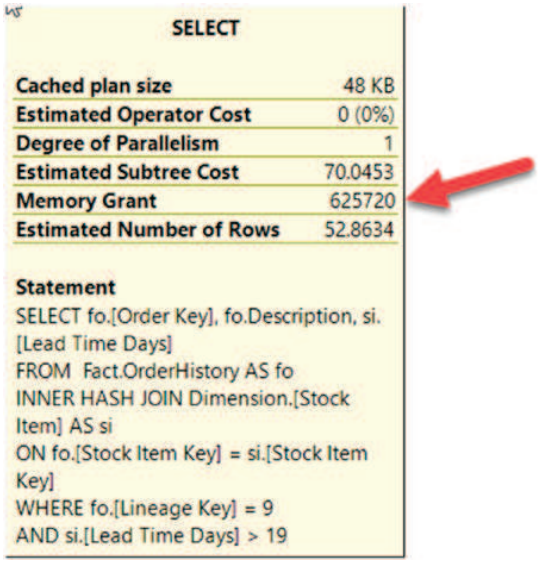


Рис. 2.7. В разделе с информацией об операторе SELECT приведен скорректированный показатель объема предоставляемой памяти

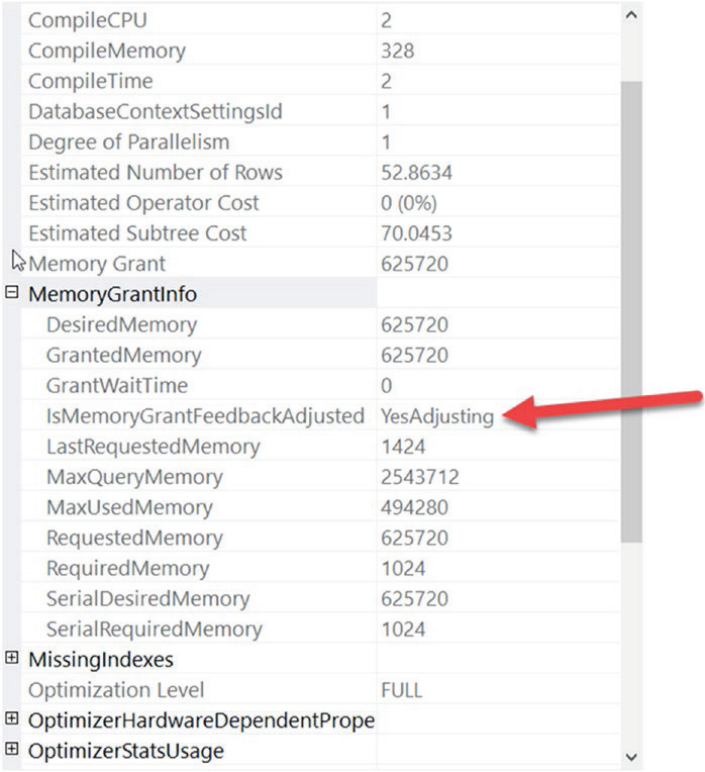


Рис. 2.8. Обратная связь по временно предоставляемому буферу памяти после коррекции предоставляемого объема памяти

5. Нам нужно убедиться, что данные статистики запроса вернулись в исходное состояние. Для этого нужно запустить код T-SQL **шага 5** в сценарии T-SQL:

```
-- Шаг 5. Восстановление таблицы и кластеризованного индекса в
-- исходное состояние.
UPDATE STATISTICS Fact.OrderHistory
WITH ROWCOUNT = 3702592;
GO
ALTER TABLE [Fact].[OrderHistory] DROP CONSTRAINT [PK_Fact_OrderHistory]
GO
ALTER TABLE [Fact].[OrderHistory] ADD CONSTRAINT [PK_Fact_OrderHistory]
PRIMARY KEY NONCLUSTERED
(
  [Order Key] ASC,
  [Order Date Key] ASC
)
GO
```

Выделение слишком большого объема памяти

Давайте рассмотрим пример, где объем предоставленной памяти слишком велик, гораздо больше того, который требуется в действительности. Как я уже упоминал ранее, если объем выделенной памяти чрезмерно велик, то, возможно, это не приведет ни к каким отрицательным последствиям, но в ряде случаев может привести к неожиданному дефициту памяти или ухудшению производительности.

Пример, иллюстрирующий выделение слишком большого объема памяти, немного сложнее; для него потребуется эмуляция одновременной работы нескольких пользователей. Поэтому для выполнения практических заданий этого примера вам понадобится бесплатная утилита для нагрузочного тестирования под названием ostress, которую можно загрузить, перейдя по ссылке www.microsoft.com/en-us/download/details.aspx?id=4511. В настоящее время для работы с этой утилитой требуется клиентский компьютер с установленной ОС Windows.

Чтобы увидеть, как выделение слишком большого объема памяти может привести к неожиданному ухудшению производительности и появлению режима ожидания (wait_type) RESOURCE_SEMAPHORE, выполните описанные ниже действия. Все используемые сценарии находятся в каталоге **ch2_intelligent_performance\iqp\rowmodemgf**. Я создал сценарии, запускаемые из командной оболочки с использованием учетной записи администратора sa.

1. Во-первых, нам необходимо настроить регулятор ресурсов (resource governor), чтобы использовать максимальный объем выделенной памяти для сервера, запустив исполняемый файл **adjustrg.cmd** (ко-

торый запускает сценарий T-SQL **adjustrg.sql**). В этом сценарии используются следующие допущения: имя сервера, для которого выполняются настройки bwsql2019, поэтому вам будет нужно отредактировать сценарий, указав вместо bwsql2019 имя своего сервера. Я выполняю эту настройку, чтобы позволить SQL Server получить очень большой объем избыточной памяти, необходимый для данного примера.

```
ALTER WORKLOAD GROUP [default]
WITH (REQUEST_MAX_MEMORY_GRANT_PERCENT = 50)
GO
ALTER RESOURCE GOVERNOR RECONFIGURE
GO
```

2. Теперь запустите исполняемый файл **turn_off_mgf.cmd** (который выполняет сценарий T-SQL **turn_off_mgf.sql**).

```
-- Отключаем обратную связь по временно предоставляемому буферу памяти
USE [WideWorldImportersDW]
GO
-- Шаг 2: Эмуляция устаревших данных статистики
UPDATE STATISTICS Fact.OrderHistory
WITH ROWCOUNT = 5000000000
GO
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;
GO
ALTER DATABASE SCOPED CONFIGURATION SET ROW_MODE_MEMORY_GRANT_
FEEDBACK = OFF
GO
ALTER DATABASE SCOPED CONFIGURATION SET BATCH_MODE_MEMORY_GRANT_
FEEDBACK = OFF
GO
```

В этом примере я буду использовать прием, аналогичный предыдущему примеру с выделением слишком малого объема памяти, на этот раз изменяя данные статистики таким образом, чтобы число строк в статистике запроса намного *превышало* число строк в таблице.

> **Примечание.** За прошедшие годы я видел несколько примеров, когда оценка кардинальности оказалась выше, чем реальное значение. Один из примеров такой ситуации – выполнение запросов к связанному серверу, когда отсутствует доступ к статистике для удаленного источника данных. В этих случаях оценка кардинальности может быть неточной и неоправданно завышенной.

3. Теперь запустите исполняемый файл **rowmode_mgf.cmd**, который выполняет сценарий T-SQL **rowmode_mgf.sql**.

```
SELECT fo.[Order Key], fo.Description, si.[Lead Time Days]
FROM Fact.OrderHistory AS fo
INNER JOIN Dimension.[Stock Item] AS si
ON fo.[Stock Item Key] = si.[Stock Item Key]
WHERE fo.[Lineage Key] = 9
AND si.[Lead Time Days] > 19
ORDER BY fo.[Order Key], fo.Description, si.[Lead Time Days]
OPTION (MAXDOP 1)
GO
```

Этот запрос похож на пример для выделения слишком малого объема памяти, но он содержит дополнительный оператор ORDER BY для добавления сортировки.

Командный файл, запускаемый из командной оболочки, будет использовать утилиту ostress для эмуляции выполнения этого запроса T-SQL десятью пользователями одновременно; запрос будет повторяться десять раз для каждого пользователя. Во время работы этого сценария используйте другой сеанс SQL для запуска сценария **dm_exec_requests.sql**, чтобы узнать, с какими типами ожидания (wait_type) могут столкнуться выполняемые пользователями запросы. Вы заметите значительное число элементов очереди ожидания освобождения ресурсов на RESOURCE_SEMAPHORE. Вы можете запускать этот файл несколько раз, пока работает сценарий ostress. То, что сценарий ostress выполняется долго, объясняется режимом ожидания освобождения ресурсов.

Общее время выполнения сценария ostress должно составлять более 40 секунд. Результат выполнения сценария (после его успешного завершения) должен быть таким:

```
<datetime> [0x000046CC] OSTRESS exiting normally, elapsed
time: 00:00:43.833
<datetime> [0x000046CC] RsFx I/O completion thread ended.
```

4. Выполните один запрос, используя сценарий *rowmode_mgf.sql*, и просмотрите данные обратной связи по временно предоставляемому буферу памяти в SQL Server Management Studio. Используйте те же приемы, что и в предыдущем разделе этой главы, чтобы просмотреть свойства оператора SELECT в плане. Раскройте раздел **MemoryGrantInfo**. Результаты должны выглядеть так, как показано на рис. 2.9.

   Ниже приведено описание ключевых параметров обратной связи.

   DesiredMemory – это *идеальный* объем предоставляемой памяти, основанный на оценке кардинальности. В данном случае это число составляет около 56 Гб. Это сумасшедшее количество памяти!

GrantedMemory – мы не можем предоставить 56 Гб памяти для этого запроса, поэтому предоставляем только около 5 Гб. Это все еще значительный объем предоставляемой памяти.

MaxUsedMemory – этот объем памяти фактически используется для предоставления памяти во время выполнения запроса. Как вы видите, он составляет всего 3 Мб. Это, безусловно, пример предоставления чрезмерно большого объема памяти по сравнению с необходимым объемом.

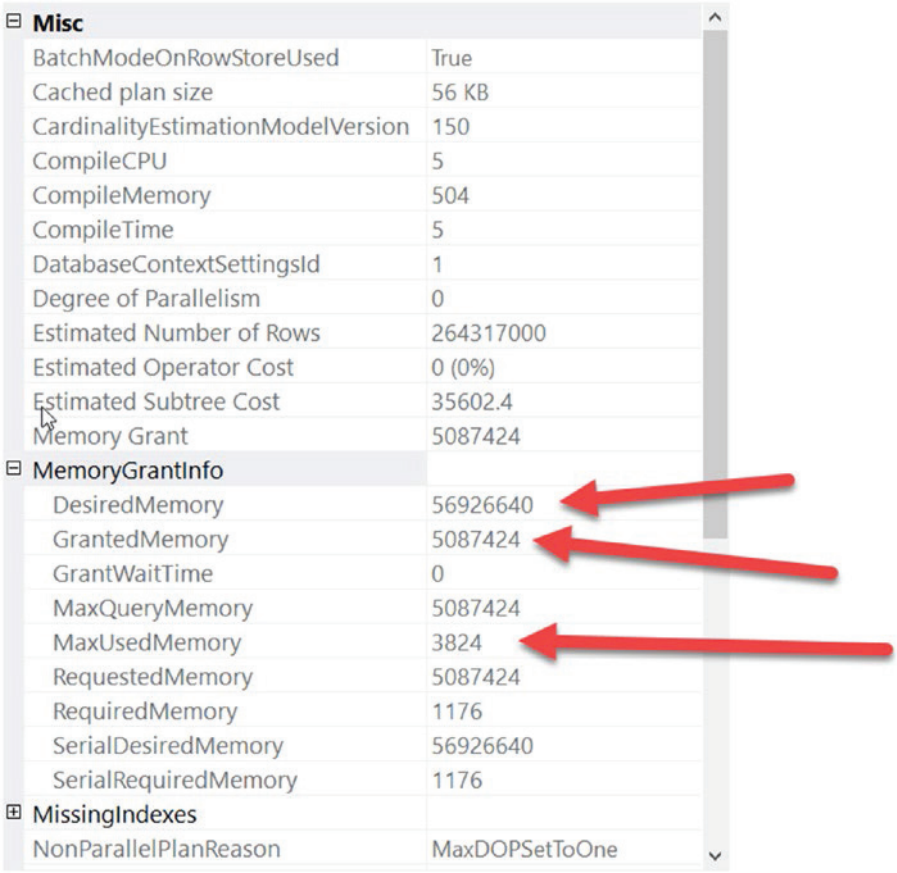


Рис. 2.9. Обратная связь по временно предоставляемому буферу памяти при чрезмерного большом объеме предоставляемой памяти

5. Теперь давайте включим обратную связь по временно предоставляемому буферу памяти, запустив исполняемый файл **turn_on_mgf.cmd** (который выполняет сценарий T-SQL **turn_on_mgf.sql**).
6. Давайте снова сэмулируем рабочую нагрузку, запустив исполняемый файл **rowmode_mgf.cmd**. Работа этого файла должна завершиться за время, вдвое меньшее предыдущего (обычно около 20 секунд).

Если вы запустите сценарий **dm_exec_requests.sql** во время работы сценария ostress, то увидите кратковременное увеличение числа элементов очереди ожиданий RESOURCE_SEMAPHORE, а затем эта очередь исчезнет, потому что сработала обратная связь по временно предоставляемому буферу памяти, уменьшив размер предоставляемой памяти в соответствии с объемом памяти, необходимым для выполнения запроса.

Совет. Попробуйте запустить исполняемый файл rowmode_mgf.cmd во второй раз. Будет ли он в этот раз работать быстрее? Да, сейчас он может работать немного быстрее. Это связано с тем, что при первом запуске rowmode_mgf.cmd первые выполнения запроса происходили очень быстро, и кешированный план не обновлялся при новом предоставлении памяти. Но по мере выполнения дальнейших запусков командного файла память каждый раз выделялась заново. Когда вы запустили rowmode_mgf.cmd во второй раз, во всех запросах использовалось новое выделение памяти.

7. Если вы посмотрите на свойства оператора SELECT, относящиеся к временно предоставляемому буферу памяти для **rowmode_mgf.sql**, то увидите, что объемы предоставляемой памяти изменились, и их обновленные значения гораздо ближе к значению объема памяти, необходимого для выполнения запроса.
8. Восстановите состояние базы данных, данные статистики для таблиц и регулятора ресурсов, запустив исполняемые файлы **adjustrgback.cmd** и **restore_orderhistory_state.cmd**.

Примечание. Даже при использовании функции обратной связи в некоторых случаях объем фактически необходимой памяти может быть очень большим. Достаточно большим, чтобы пользователи, работающие одновременно, сталкивались с режимом ожидания RESOURCE_SEMAPHORE, что, в свою очередь, приведет к ситуации дефицита памяти в SQL Server. В этих случаях вы можете использовать регулятор ресурсов (resource governor), чтобы ограничить объем выделяемой памяти. Документация по данной теме находится по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/create-workload-group-transact-sql. В ней разъясняется, как изменить соответствующие параметры. В SQL Server 2019 это значение настраиваемого параметра теперь может быть числом с плавающей запятой, поэтому допустимы значения < 1 %. Это может быть важно для систем с большим объемом памяти. Кроме того, вы можете установить эти значения на уровне отдельного запроса. См. документацию https://docs.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query#arguments.

Эта функция хорошо продумана и может реально помочь вам сэкономить время при настройке параметров SQL Server для рабочих нагрузок, требующих выделения памяти.

Существует несколько сценариев, когда обратная связь по временно предоставляемому буферу памяти не должна использоваться или же будет бесполезна:

- не обнаружено «утечки данных в tempdb» или используется 50 % выделенной памяти;
- имеют место флуктуации объема памяти, когда объем предоставляемой памяти постоянно то уменьшается, то увеличивается.

Отложенная компиляция табличных переменных

Если вы работаете в Microsoft уже 26 лет, то за это время вы, несомненно, встречались со многими людьми. Я видел очень много людей, которые умнее меня и, честно говоря, обладают лучшим характером, нежели я. Один из таких людей – Джек Ли (Jack Li). Джек много лет проработал в службе технической поддержки CSS в нашем офисе в Ирвинге (штат Техас). Несколько лет назад у Джека появилась возможность поработать в команде SQL Engineering – это случилось после того, как я присоединился к данной команде. Однажды он с присущей ему скромностью спросил меня, считаю ли я, что он пригоден для работы в этой команде. Я не колебался. Я сказал ему, что у него есть все необходимые навыки, чтобы стать первоклассным разработчиком, и, кроме того, он обладает уникальным и обширным опытом по улучшению производительности SQL Server. Несмотря на то что команда CSS потеряла одного из лучших ее членов, наша команда, безусловно, выиграла от того, что к ней присоединился Джек Ли.

Первым проектом, над которым Джек работал в своем новом статусе, было решение известной проблемы оценки кардинальности для табличных переменных. Когда используются табличные переменные, у нас наблюдается проблема, состоящая в том, что при оценке кардинальности оптимизатор SQL Server всегда добавляет в оценку одну строку, независимо от того, сколько строк заполняется в табличной переменной. Это происходит потому, что оптимизатор не обладает информацией о том, сколько строк на самом деле содержится в табличной переменной, поскольку они определяются и обычно заполняются как часть пакета или хранимой процедуры. Когда Джек работал в техподдержке, он писал об этой проблеме и предлагал решение с использованием флага трассировки. Описание предложенного Джеком решения можно найти по ссылке https://blogs.msdn.microsoft.com/psssql/2014/08/11/having-performance-issues-with-table-variables-sql-server-2012-sp2-can-help/.

Таким образом, когда Джек присоединился к команде, он уже был глубоко погружен в эту проблему. У руководства группы Query Processor была идея, которую они хотели реализовать в SQL Server 2019 в рамках интеллектуальной обработки запросов, называемой отложенной компиляцией табличных переменных. Они прислушались к Джеку при реализации этой возможности.

Приведем цитату из документации по SQL Server по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?view=sql-server-ver15#table-variable-deferred-compilation: «Отложенная компиляция табличной переменной откладывает компиляцию оператора, который ссылается на табличную переменную, до первого фактического выполнения оператора. Отложенная компиляция в данном случае выполняется так же, как для временных таблиц. Это изменение приводит к тому, что используется фактическое количество строк, а не одна строка, как предполагалось первоначально».

Вы можете прочитать и просмотреть примеры того, как включить и отключить отложенную компиляцию табличных переменных, а также параметры базы данных и указания запросов, по ссылке https://docs.microsoft.com/en-us/sql/t-sql/data-types/table-transact-sql?view=sql-server-ver15#table-variable-deferred-compilation.

Все примеры сценариев, приведенные в этом разделе, размещены в каталоге **ch2_intelligent_performance\iqp\tablevariable**.

Давайте рассмотрим пример этой концепции, используя записную книжку T-SQL (T-SQL Notebook) (вы также можете просмотреть этот сценарий T-SQL, открыв файл **iqp_tablevariable.sql**).

1. Откройте в Azure Data Studio записную книжку T-SQL с названием **iqp_tablevariable.ipynb**.
2. Выполните каждый шаг в записной книжке T-SQL, чтобы сравнить производительность при использовании табличных переменных с отложенной компиляцией и без нее.
3. Чтобы сравнить планы выполнения запросов для этих двух сценариев, мы можем использовать функцию под названием Query Store, которая впервые была представлена в SQL Server 2016. Возможно, вы этого не знали, но при восстановлении резервной копии WideWorldImportersDW в базе данных уже использовалось хранилище запросов Query Store.
4. Вот как использовать Query Store для сравнения двух запросов: один с использованием отложенной компиляции табличной переменной, а другой – без нее.
5. Откройте SSMS, подключитесь к SQL Server, на котором вы запустили примеры из записных книжек T-SQL, и найдите отчет **Top Resource Consuming Queries** (Самые ресурсоемкие запросы), как показано на рис. 2.10.
6. В отчете, приведенном на рис. 2.10, показаны данные Query Store после выполнения предыдущего примера использования обратной связи по временно предоставляемому буферу памяти в режиме строк и примеры табличных переменных из этого раздела. Ваше окно программного интерфейса может немного отличаться от приведенного на рисунке в зависимости от того, каким инструментом вы пользуе-

тесь и на каких данных выполняется запущенный вами запрос. Каждый столбец диаграммы представляет уникальный запрос, поэтому вам нужно найти запрос, в котором используется табличная переменная. Щелкните мышью каждую строку – ниже будет отображен текст запроса. Если вы просмотрите запрос в хранимой процедуре из этого примера в записной книжке T-SQL, то обнаружите, что этот запрос содержит следующий код:

```
SELECT top 10 oh.[Order Key], oh.[Order Date Key],oh.[Unit Price],
o.Quantity
FROM Fact.OrderHistoryExtended AS oh
INNER JOIN @Order AS o
ON o.[Order Key] = oh.[Order Key]
WHERE oh.[Unit Price] > 0.10
ORDER BY oh.[Unit Price] DESC
```

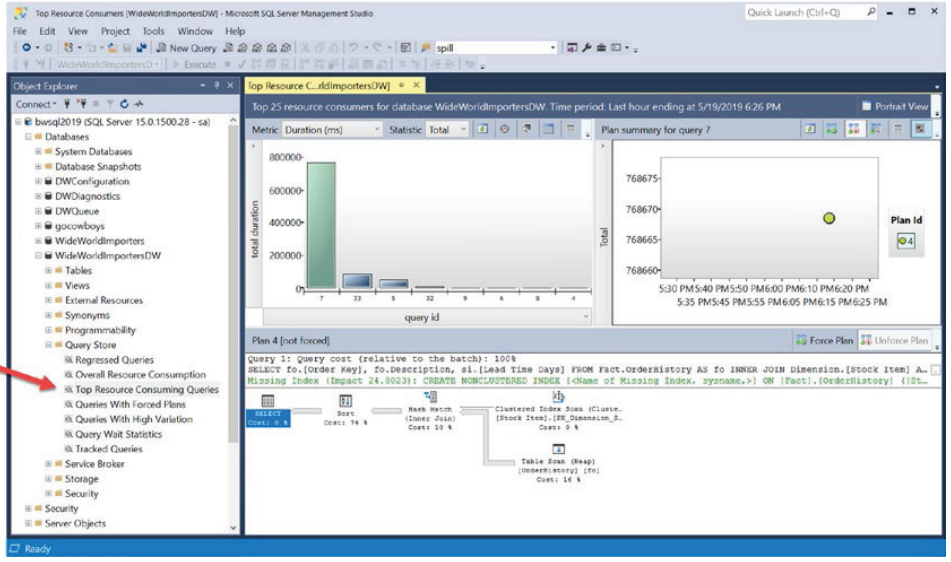


Рис. 2.10. Отчет Top Resource Consuming Queries (Самые ресурсоемкие запросы) в Query Store

7. Поочередно щелкайте мышью каждый столбец на диаграмме, пока не увидите этот запрос. Обратите внимание на две точки справа, которые представляют два плана выполнения для данного запроса. После завершения выполнения отчет должен выглядеть примерно так, как показано на рис. 2.11.

   Чем «выше» расположена точка на диаграмме, тем больше средний период времени для данного плана выполнения запроса. Если щелкнуть каждую точку, вы увидите, как изменяется план выполнения запроса в нижнем окне.

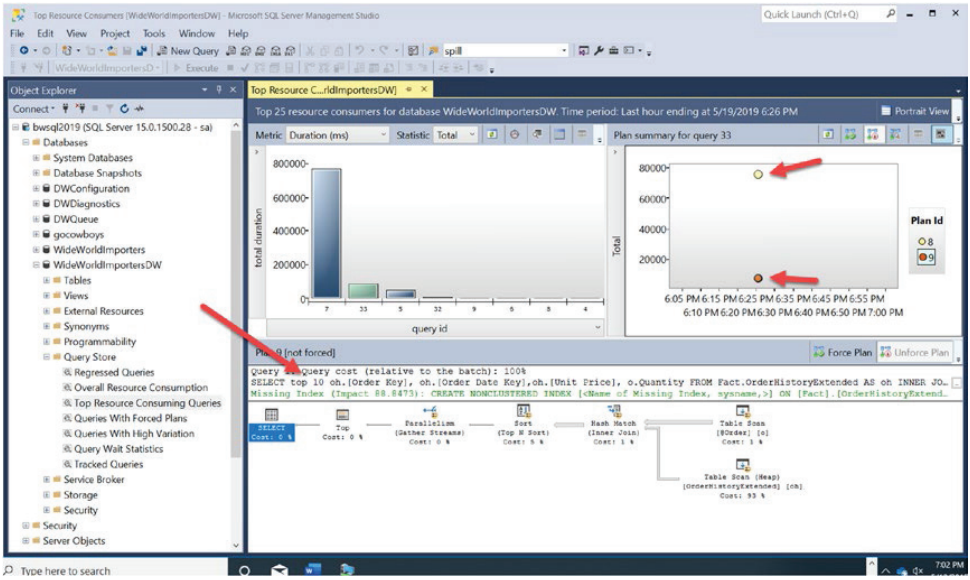


Рис. 2.11. Планы выполнения запросов для использования табличных переменных

8. Наведите курсор на верхнюю точку, чтобы увидеть данные статистики для плана выполнения запроса. Просмотрите данные статистики, например среднюю продолжительность, как показано на рис. 2.12.

Рис. 2.12. Средний период времени для более медленного плана выполнения запроса

После того как вы щелкнули мышью точку и просмотрели план выполнения запроса в нижней области окна, наведите указатель мыши на оператор сканирования таблицы (Table Scan). Обратите внимание на оценку числа обрабатываемых строк, равную 1, как показано на рис. 2.13.

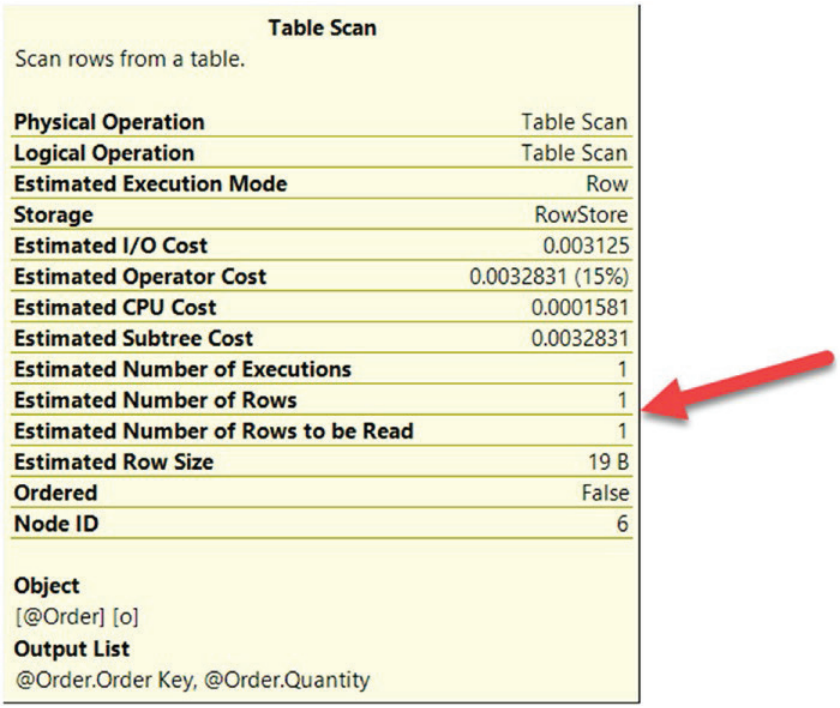
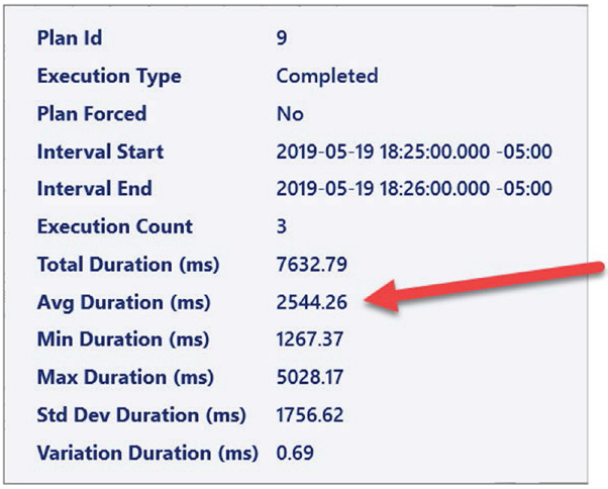


Рис. 2.13. Оценка в одну строку для табличной переменной

Обратите внимание на операцию соединения табличной переменной и таблицы OrderHistoryExtended. Здесь используется алгоритм соединения с применением вложенных циклов (Nested Loops Join). Это вполне разумный выбор для оптимизатора, поскольку он предполагает, что табличная переменная содержит только одну строку. Проблема в том, что в данном случае табличная переменная содержит около 3 млн строк! Использование подобного типа соединения (с вложенными циклами) для такого числа строк будет очень дорогостоящей операцией и в данном случае не имеет смысла.

9. Теперь щелкните «нижнюю» точку в окне с планами выполнения запросов. Наведите указатель мыши на точку, чтобы увидеть среднюю продолжительность запроса. Это должно выглядеть примерно так, как показано на рис. 2.14.

   В этом случае средняя продолжительность плана выполнения запроса составляет 2,5 секунды, что намного лучше, чем 25 секунд.



Plan Id	9
Execution Type	Completed
Plan Forced	No
Interval Start	2019-05-19 18:25:00.000 -05:00
Interval End	2019-05-19 18:26:00.000 -05:00
Execution Count	3
Total Duration (ms)	7632.79
Avg Duration (ms)	2544.26
Min Duration (ms)	1267.37
Max Duration (ms)	5028.17
Std Dev Duration (ms)	1756.62
Variation Duration (ms)	0.69

Рис. 2.14. Средняя продолжительность более быстрого плана выполнения запроса

Теперь просмотрите план выполнения запроса. Наведите указатель мыши на оператор сканирования таблицы (Table Scan). Обратите внимание, что оценки теперь точны, и, поскольку требуется выполнить сканирование таблицы, использование пакетного режима вполне оправдано. Это пример одновременного использования нескольких функций интеллектуальной обработки запросов. Информация, выводимая для этого оператора, должна выглядеть, как показано на рис. 2.15.

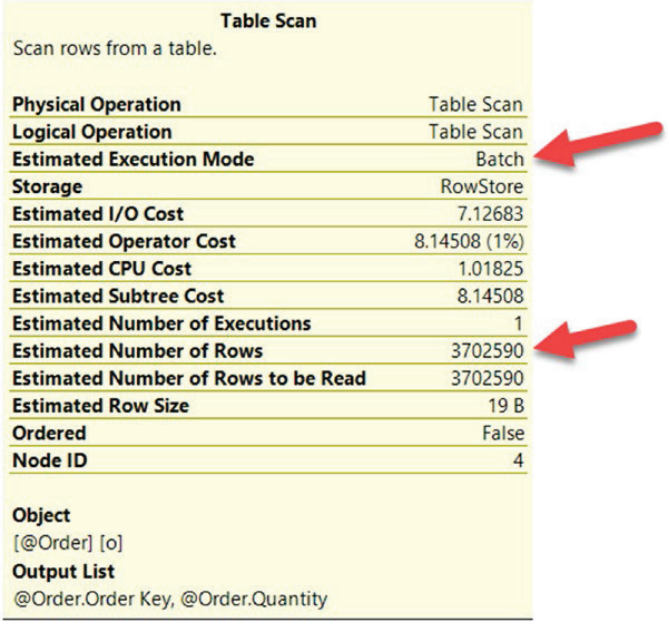


Table Scan

Scan rows from a table.

Physical Operation	Table Scan
Logical Operation	Table Scan
Estimated Execution Mode	Batch
Storage	RowStore
Estimated I/O Cost	7.12683
Estimated Operator Cost	8.14508 (1%)
Estimated CPU Cost	1.01825
Estimated Subtree Cost	8.14508
Estimated Number of Executions	1
Estimated Number of Rows	3702590
Estimated Number of Rows to be Read	3702590
Estimated Row Size	19 B
Ordered	False
Node ID	4

Object
[@Order] [o]
Output List
@Order.Order Key, @Order.Quantity

Рис. 2.15. Лучшая оценка при использовании табличной переменной

Теперь посмотрите на соединение табличной переменной и таблицы OrderHistoryExtended. Для этой цели используется хеш-соединение, а также выполняется сканирование таблицы OrderHistoryExtended.

Пакетный режим для хранилища строк

В SQL Server 2012 была добавлена отличная (какое преуменьшение!) возможность, называемая столбцовыми индексами. Это было сделано в рамках проекта Apollo (см. оригинальный блог по ссылке https://cloudblogs.microsoft.com/sqlserver/2011/08/04/columnstore-indexes-a-new-feature-in-sql-server-known-as-project-apollo/). При разработке этой новой функции был усовершенствован обработчик запросов – в нем появилась возможность выполнять обработку строк в пакетном режиме с использованием столбцовых индексов. До этих пор операторы, включаемые в план выполнения запросов, такие как сканирование таблицы (scan), выполняли операции и обрабатывали данные построчно, выбирая по одной строке таблицы целиком. Пакетный режим представляет новую парадигму обработки данных, позволяющую операторам обрабатывать пакеты строк, упорядоченных по столбцам и содержащих векторы для идентификации подходящих строк. Эта концепция очень хорошо согласуется со столбцовыми индексами, которые упорядочены по столбцам, а не по строкам.

Хотя столбцовые индексы очень полезны для выполнения аналитических запросов, для которых характерны сканирование и обработка большого количества строк, все же столбцовые индексы могут не соответствовать вашим потребностям в обработке данных; также могут иметься ограничения, препятствующие их использованию. Кроме того, у вас могут иметься запросы, соответствующие сценарию «аналитическая нагрузка». Другими словами, ваши потребности могут заключаться не в выполнении запросов к одной или нескольким строкам (что многие считают стандартным «сценарием OLTP»). Любая таблица или индекс, которые не организованы с помощью столбцового индекса, называется *хранилищем строк*.

В SQL Server 2019 обработчик запросов может автоматически определять, соответствует ли ваш запрос пакетному режиму для хранилища строк. Использование пакетного режима, опять же, может не иметь смысла для всех запросов, поэтому для его применения важно понимать и использовать некоторые основные принципы. Например, ваш запрос должен обрабатывать большое количество строк и включать операции, которые требуют агрегации данных (такие как count(*) или sum(), а также соединение или сортировку). Другими словами, пакетную обработку имеет смысл использовать, когда существует поток данных между несколькими операторами над большим числом строк для выполнения запроса. Насколько большим? В документации не приводится конкретная цифра (потому что в будущем она может измениться), но пороговое значение обычно составляет 128 тыс. строк.

Подробная информация о пакетном режиме для хранилища строк, в том числе о включении и отключении этой возможности, находится по адре-

су https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?#batch-mode-on-rowstore. Эта документация содержит сведения об истории возникновения данной возможности, а также о том, при каких рабочих нагрузках она принесет наибольшее преимущество, и информацию о существующих ограничениях.

> **Совет.** Вы действительно хотите углубиться в эту тему? Тогда я рекомендую вам статью в блоге эксперта SQL-сообщества Димы Пилюгина, который определил «роковой» предел 128 тыс. Этот блог расположен по адресу www.queryprocessor.com/batch-mode-on-rowstore/.

Мы будем использовать базу данных WideWorldImportersDW, которую восстановили и расширили в начале этой главы. Давайте рассмотрим пример, в котором пакетный режим для строкового хранилища может ускорить выполнение запросов. В каталоге **ch2_intelligent_performance\iqp\batchmoderow** размещены все примеры сценариев.

Вы можете выполнить следующие запросы, используя сценарий **iqp_batchmoderow.sql** или записную книжку T-SQL **iqp_batchmoderow.ipynb**.

Какой бы способ вы ни выбрали, давайте продвигаться вперед шаг за шагом. В этом разделе я рекомендую вам использовать записную книжку T-SQL, выполняя пример сценария. Вы можете также выполнить сценарий **iqp_batchmoderow.sql**, используя любой инструмент SQL, однако вам нужно будет проанализировать планы выполнения запросов в графическом инструменте, таком как SSMS или Azure Data Studio (или же иметь возможность прочитать и подробно разобрать XML-план).

Откройте Azure Data Studio, подключенную к вашему экземпляру SQL Server 2019, и затем откройте записную книжку **iqp_batchmoderow.ipynb**.

Одним из удобств записных книжек является то, что документация для каждого шага и ячеек находится в самой записной книжке. Записные книжки, сохраненные в репозитории GitHub для этой книги, уже содержат все ответы, так что вы знаете, чего ожидать!

Я даже визуально проиллюстрировал различия в планах выполнения запросов, используя Azure Data Studio и приводя ожидаемые результаты.

Прочитайте текст и выполните каждый шаг из записной книжки. Вы увидите, что использование пакетного режима для хранилища строк может дать значительный прирост производительности, особенно при работе с таблицами, содержащими большие наборы данных. Кроме того, пакетный режим теперь работает как для столбцового хранилища данных (реализовано в SQL Server 2017), так и для хранилища строк, поэтому вам не нужно об этом беспокоиться. Обработчик запросов знает, когда его использовать и как он может повысить производительность вашего запроса.

Чтобы вы могли выполнить самопроверку, на рис. 2.16 показано, как выглядит верхняя область записной книжки.

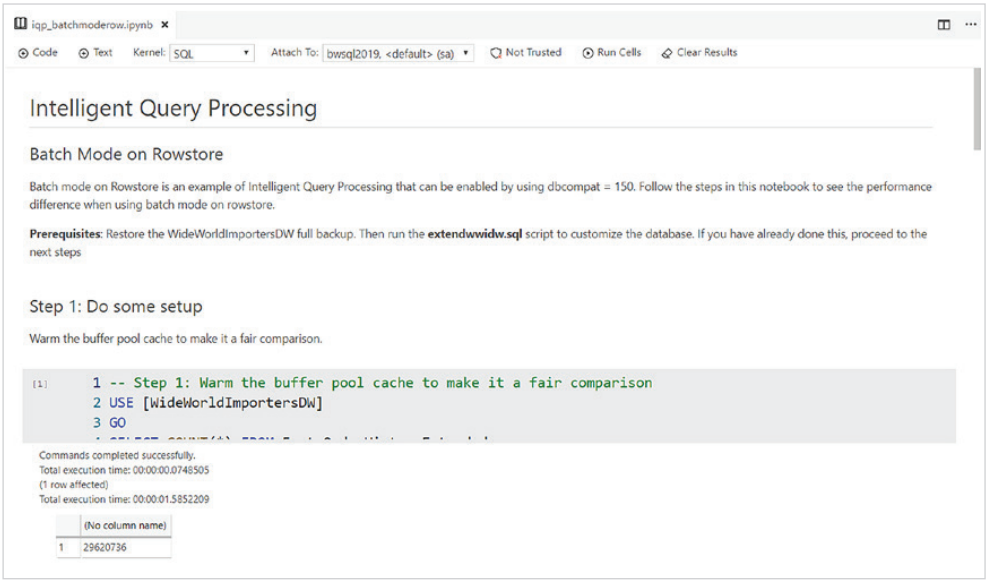


Рис. 2.16. Записная книжка T-SQL для демонстрации пакетного режима обработки для хранилища строк

Встраивание скалярных UDF

В SQL Server уже давно существует такая концепция, как *пользовательская функция* (user-defined function, UDF). Концепция заключается в том, что вы размещаете код T-SQL внутри оператора FUNCTION, который принимает на вход один или несколько параметров, и функция возвращает результирующее значение. Затем вы можете использовать функцию в любом операторе SELECT T-SQL. Существуют и иные популярные способы повторного использования кода, например хранимые процедуры, но функция обладает одним преимуществом: она может быть частью оператора SELECT.

> **Примечание.** Есть и другие способы использования пользовательских функций, о которых вы можете узнать больше, перейдя по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-ver15.

Существует два типа пользовательских функций:

- скаляр, который возвращает одно значение;
- табличное значение, которое возвращает результирующий набор данных в виде таблицы (тип TABLE).

Несмотря на популярность и преимущества UDF, их использование может привести к проблемам с производительностью из-за существующих ограничений. Эти ограничения связаны с тем, как UDF компилируют-

ся и интегрируются в общий план выполнения запросов. Например, всякий раз, когда скалярная UDF используется для того, чтобы вернуть значение как часть списка столбцов, каждая строка таблицы, к которой выполняется запрос, используется в UDF по одной строке за один раз. Существуют и другие ограничения, связанные с тем, как обработчик запросов обрабатывает пользовательские функции, что в некоторых ситуациях просто делает их весьма неэффективными с точки зрения производительности.

Теперь перейдем к *встраиванию* скалярных UDF. Обработчик запросов может взять код UDF (UDF может состоять из нескольких операторов T-SQL) и интегрировать эти операторы в общий запрос, отсюда и возник термин «встраивание».

Вы можете прочитать, как включить встраивание скалярных UDF, используя параметр dbcompat, в документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/scalar-udf-inlining?#enabling-scalar-udf-inlining.

О том, как отключить и включить встраивание скалярных UDF без изменения параметра dbcompat, рассказывается в документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/scalar-udf-inlining?#disabling-scalar-udf-inlining-without-changing-the-compatibility-level.

Как и в любом примере интеллектуальной обработки запросов, для понимания встраивания скалярных UDF лучше всего рассмотреть пример. В каталоге **ch2_intelligent_performance\iqp\scalarinlineudf** размещены все сценарии, которые приводятся здесь в качестве примеров.

Аналогично другим примерам из этой главы, у вас есть два способа прохождения сценариев, иллюстрирующих встраивание скалярных UDF. Вы можете использовать записную книжку T-SQL **iqp_scalarudfinlining.pynb** или набор сценариев T-SQL.

> **Примечание.** В основу приведенного мной примера были положены примеры кода, приведенные в следующей публикации в блоге Microsoft: https://docs.microsoft.com/ru-ru/archive/blogs/sqlserverstorageengine/introducing-scalar-udf-inlining; в этом блоге также есть некоторые важные и интересные подробности о предыдущих ограничениях скалярных UDF и о том, как интеллектуальная обработка запросов позволила значительно повысить производительность.

Для выполнения примеров из этого раздела мы будем использовать сценарии T-SQL и проверку фактического плана выполнения запроса в SSMS.

1. Откройте сценарий T-SQL **get_customer_spend.sql**.

   Код этого сценария выглядит так:

```
USE WideWorldImportersDW
GO
SELECT c.[Customer Key], SUM(oh.[Total Including Tax]) as total_spend
```

```
FROM [Fact].[OrderHistory] oh
JOIN [Dimension].[Customer] c
ON oh.[Customer Key] = c.[Customer Key]
GROUP BY c.[Customer Key]
ORDER BY total_spend DESC
GO
```

Этот сценарий определяет суммартные затраты на приобретение продукции на одного клиента, основываясь на данных из таблицы OrderHistory. Посмотрев на полученные результаты, вы увидите, что диапазон затрат клиентов составляет от 2 до 7 млн. Согласно условиям поставленной задачи, мы должны создать пользовательскую функцию, которая будет принимать в качестве входных данных идентификатор клиента и классифицировать клиента на основании его суммарных расходов на приобретение продукции.

Клиенты, суммарные расходы которых составляют ⩽ 3М, будут относиться к категории «ПРОСТЫЕ». Клиенты, потратившие от 3 до 4,5 млн, будут относиться к категории «ЗОЛОТЫЕ». Любой, кто потратит сумму более 4,5 млн, будет считаться «ПЛАТИНОВЫМ» клиентом. Преимущество использования функции в данном случае заключается в том, что мы можем изменять правила для определения категории клиента («ПРОСТОЙ», «ЗОЛОТОЙ» и «ПЛАТИНОВЫЙ»), не оказывая влияния на весь остальной код, использующий эту функцию.

2. Откройте сценарий T-SQL **iqp_scalarudfinlining.sql** и выполните все шаги в соответствии с комментариями, приведенными в сценарии.
3. Выполните фрагмент сценария **шаг 1**, который создаст скалярную UDF:

```
-- Шаг 1. Создайте новую функцию, чтобы получить категорию
-- клиентов на основе их суммарных расходов на заказы
USE WideWorldImportersDW
GO
CREATE OR ALTER FUNCTION [Dimension].[customer_category] (@CustomerKey INT)
RETURNS CHAR(10) AS
BEGIN
DECLARE @total_amount DECIMAL(18,2)
DECLARE @category CHAR(10)
SELECT @total_amount = SUM([Total Including Tax])
FROM [Fact].[OrderHistory]
WHERE [Customer Key] = @CustomerKey
IF @total_amount <= 3000000
  SET @category = 'REGULAR'
ELSE IF @total_amount < 4500000
  SET @category = 'GOLD'
```

```
ELSE
  SET @category = 'PLATINUM'
RETURN @category
END
GO
```

4. Задайте нужное значение переменной dbcompat, очистите кеш процедур и «разогрейте» кеш буферного пула, выполнив **шаг 2**.

```
-- Шаг 2. Установите для базы данных значение db compat 150,
-- очистите кеш процедур от предыдущих выполнений и добейтесь того,
-- чтобы сравнение было корректным, «разогрев» кеш
ALTER DATABASE WideWorldImportersDW
SET COMPATIBILITY_LEVEL = 150
GO
ALTER DATABASE SCOPED CONFIGURATION
CLEAR PROCEDURE_CACHE;
GO
SELECT COUNT(*) FROM [Fact].[OrderHistory]
GO
```

5. Давайте запустим запрос с использованием UDF, но применим указание запроса, чтобы временно отключить встраивание скалярной UDF. Включите фактический план выполнения (Actual Execution Plan) в SSMS и выполните **шаг 3** сценария.

```
-- Шаг 3. Запустите запрос, но отключите использование встраивания
-- скалярной функции, используя указание запроса.
SELECT [Customer Key], [Customer], [Dimension].[customer_category]
([Customer Key]) AS [Discount Price]
FROM [Dimension].[Customer]
ORDER BY [Customer Key]
OPTION (USE HINT('DISABLE_TSQL_SCALAR_UDF_INLINING'))
GO
```

Время выполнения запроса составляет не менее 30 секунд. Фактический план выполнения запроса должен выглядеть примерно так, как показано на рис. 2.17.

Если вы наведете указатель мыши на каждый оператор, то увидите, что он затрагивает 403 строки. Число строк не так уж велико; так почему же выполнение запроса занимает так много времени? Это происходит потому, что вы не видите, что скалярная функция обращается к таблице OrderHistory, которая содержит более 3 млн строк; для каждой строки в таблице Dimension.Customer она обращается ко всему множеству строк в таблице OrderHistory (а их более 3 млн). Это крайне неэффективно.

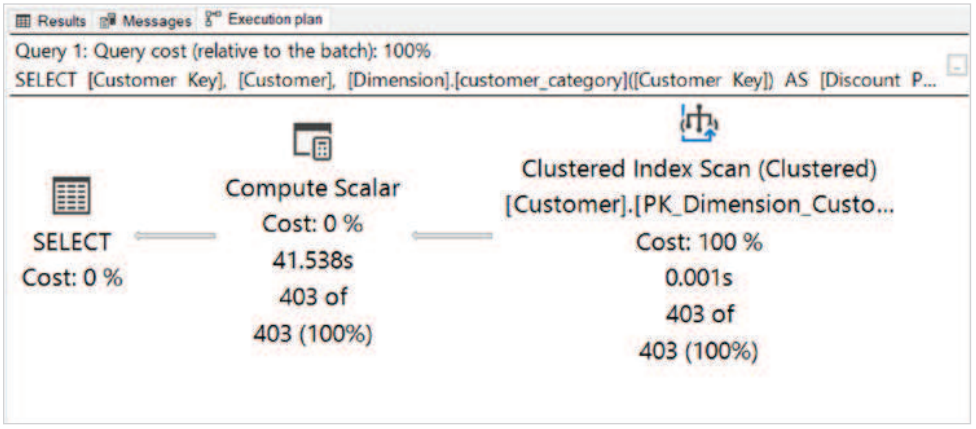


Рис. 2.17. План выполнения запроса для отключенной возможности встраивания скалярной UDF

6. Запустите шаг 4 сценария, который будет выполнять тот же самый запрос, не используя указание, тем самым включив встраивание скалярной UDF.

```
-- Шаг 4: Запустите запрос снова, но не используйте указание
SELECT [Customer Key], [Customer], [Dimension].[customer_category]
([Customer Key]) AS [Discount Price]
FROM [Dimension].[Customer]
ORDER BY [Customer Key]
GO
```

Запрос должен выполняться значительно быстрее. Если вы посмотрите на фактический план выполнения, то увидите, что в плане представлены операторы, необходимые для выполнения функции, а также новые операторы, оптимизирующие запрос к таблице OrderHistory, выполняющийся в функции. План будет выглядеть примерно так, как показано на рис. 2.18.

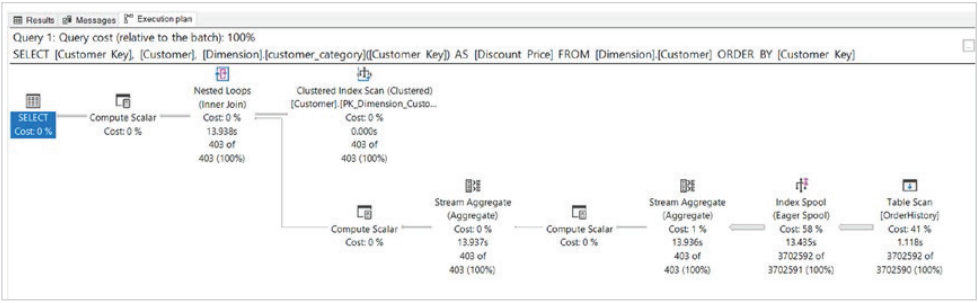


Рис. 2.18. План выполнения встраиваемой скалярной UDF

Теперь, когда вы смогли убедиться в мощи встраивания скалярных UDF, вы должны быть более мотивированы использовать скалярные UDF.

Узнать больше о встраивании скалярных UDF, в том числе обо всех связанных с ними требованиях и ограничениях, можно по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/scalar-udf-inlining.

Функция приблизительного подсчета числа уникальных значений Approximate Count Distinct

Есть сценарии, в которых вам нужно подсчитать количество строк в любой таблице. Это сделать легко. Просто используйте оператор SELECT COUNT(*) FROM <table> – и вы получите готовый ответ. Но также иногда бывает необходимо знать количество различных значений в определенном столбце по всем строкам таблицы. Эта задача не намного сложнее. Просто используйте SELECT COUNT(DISTINCT <col>) FROM <table>. Это кажется достаточно простым. Единственная проблема заключается в том, как должен работать обработчик запросов, чтобы найти все различные значения.

В SQL Server для этого часто используется оператор *Hash Match*. Этот оператор похож на Hash Join в том, что «хеш-таблица» используется для построения списка всех различных значений, чтобы дальше использовать этот список для подсчета. Как вы помните, ранее в этой главе рассказывалось, что для Hash Join необходимо предоставить память; таким образом, при использовании этого оператора могут возникнуть все проблемы, характерные для предоставления памяти. Кроме того, использование хеш-таблицы для подсчета всех различных значений – очень ресурсоемкая операция, и для нее может потребоваться много вычислительных ресурсов.

Есть ли лучший способ решения этой задачи? Да, есть *другой* способ, который может работать быстрее, при этом давая чуть менее точный ответ. Это **новая функция T-SQL APPROX_COUNT_DISTINCT()**. Это встроенная функция, подсчитывающая различные значения в столбце на основе выборочного приближения. Это не расширение функции COUNT(). Это совершенно новая функция, поэтому для ее использования не требуется устанавливать значение параметра dbcompat = 150. Данная функция использует концепцию *HyperLogLog* (вы можете прочитать больше об этой концепции, перейдя по ссылке https://en.wikipedia.org/wiki/HyperLogLog). Использование приблизительного подсчета числа уникальных значений дает вероятность ошибки 2 % с вероятностью 97 %. Это означает, что если вас устроит ответ, который будет весьма близок к реальности, вы можете использовать эту функцию.

Давайте рассмотрим пример использования данной функции и сравним ее с применением COUNT и DISTINCT.

Все примеры сценариев размещены в каталоге **ch2_intelligent_performance\iqp\approxcount**. Вы можете просмотреть примеры с помощью записной книжки T-SQL **iqp_approxcountdistinct.ipynb**. Я также создал сценарий T-SQL с названием **iqp_approxcountdistinct.sql**. Давайте воспользуемся сценарием T-SQL и пройдемся по его шагам, изучая различия между запросами и их планами выполнения.

1. Откройте сценарий **iqp_approxcountdistinct.sql** в SSMS.
2. Выполните операторы **шага 1**. При этом выполняется очистка кеша процедур, изменение значения уровня dbcompat (dbcompat = 130) и «разогревается» буферный пул (это «честный бой»).

```
-- Шаг 1. Очистите кеш, установите значение dbcompat, равное 130,
-- чтобы доказать, что сценарий работает, и «разогрейте» кеш
USE WideWorldImportersDW
GO
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE
GO
ALTER DATABASE WideWorldImportersDW SET COMPATIBILITY_LEVEL = 130
GO
SELECT COUNT(*) FROM Fact.OrderHistoryExtended
GO
```

   Вы можете удивиться, почему я установил значение переменной dbcompat, равное 130. Это сделано, чтобы показать вам, что вам не нужно использовать значение dbcompat = 150 для применения этой новой возможности. Это связано с тем, что новая функция T-SQL APPROX_COUNT_DISTINCT() *просто поставляется* с ядром SQL Server 2019, но для нее не требуется устанавливать значение dbcompat = 150, как для других функций интеллектуальной обработки запросов.

3. Включите фактический план выполнения запроса (Actual Execution Plan) в SSMS и выполните **шаг 2**:

```
-- Шаг 2. Сначала используйте COUNT и DISTINCT
SELECT COUNT(DISTINCT [WWI Order ID])
FROM [Fact].[OrderHistoryExtended]
GO
```

   Это займет не так много времени – в зависимости от мощности вашего компьютера, возможно, от 4 до 5 секунд. В результате у вас должно получиться число 29 620 736. Пять секунд для подсчета различных значений – не так уж и плохо. Однако что произойдет, если в этой таблице будет 100 млн строк или больше? Это не так уж необычно для больших баз данных.

Если вы посмотрите на план выполнения запроса, то увидите нечто похожее на то, что показано на рис. 2.19.

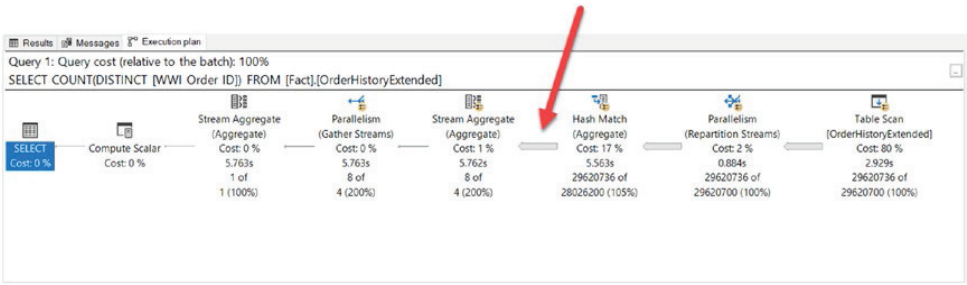


Рис. 2.19. План выполнения запроса для COUNT и DISTINCT

Обратите внимание на оператор Hash Match. Если вы наведете указатель мыши на этот оператор, то увидите, что он использует построчный режим (Row Mode) и должен обработать все 29 млн строк в хеш-операторе.

4. Теперь выполните **шаг 3** сценария следующим образом:

```
-- Шаг 3. Используйте новую функцию APPROX_COUNT_DISTINCT для
-- сравнения вычисленных значений и производительность запроса
-- Разность между результатами приблизительного и точного подсчета
-- не должна превышать 2% от точного значения (с вероятностью 97%)
SELECT APPROX_COUNT_DISTINCT([WWI Order ID])
FROM [Fact].[OrderHistoryExtended]
GO
```

На этот раз выполнение запроса должно занять всего секунду или две – это примерно на 50 % быстрее, чем раньше. Опять же, это может быть важно для очень больших наборов данных.

Если вы посмотрите на план выполнения запроса, он будет выглядеть аналогично, но с меньшим числом операторов, как показано на рис. 2.20.

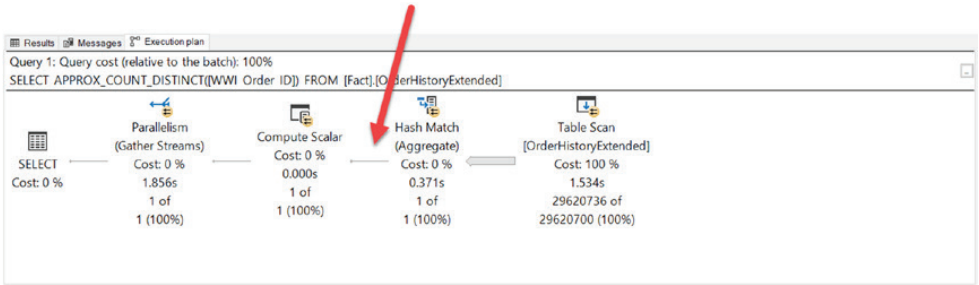


Рис. 2.20. План выполнения запроса для APPROX_COUNT_DISTINCT

Обратите внимание, что у оператора Hash Match отсутствует «толстая линия» в качестве иллюстрации выходных данных, потому что операция приближения применяется именно в этом операторе, что дает только одну строку для остальной части плана.

Как вы можете заметить, использование приблизительного подсчета числа уникальных значений может улучшить производительность, если вам нужно лишь «достаточно точное» значение.

5. Восстановите предыдущее значение уровня совместимости базы данных (dbcompat), равное 150, выполнив **шаг 4** сценария.

```
-- Шаг 4. Восстановление предыдущего значения уровня совместимости
-- базы данных.
ALTER DATABASE WideWorldImportersDW SET COMPATIBILITY_LEVEL = 150
GO
```

Подробнее о функции APPROX_COUNT_DISTINCT можно прочитать по ссылке https://docs.microsoft.com/en-us/sql/t-sql/functions/approx-count-distinct-transact-sql.

Интеллектуальная обработка запросов – это более интеллектуальный обработчик запросов, отвечающий вашим потребностям и избавляющий от необходимости внесения серьезных изменений в имеющиеся приложения. Большая часть его функциональности доступна, если вы просто измените знчение уровня совместимости вашей базы данных, установив для него значение 150.

Я с нетерпением жду появления новых возможностей в будущем, поскольку обработчик запросов расширяет поддержку сценариев, включая новые сценарии, в основу которых положены ваши отзывы.

Упрощенное профилирование запросов

Прежде чем я присоединился к команде разработчиков SQL Server в 2016 году, я работал в технической поддержке, решая самые сложные проблемы, с которыми когда-либо сталкивались наши клиенты. И проблемы с производительностью оказывались самыми трудными не только для меня, но и для всей команды CSS. Все проблемы, касающиеся производительности SQL Server, весьма сложны – в них изначально много неясного, они критичны по времени, и редко когда у вас под рукой есть вся нужная информация, чтобы их решить.

В SQL Server существуют великолепные средства диагностики проблем с производительностью, в число которых входят динамические административные представления (Dynamic Management Views, DMVs) и расширенные события (Extended Events). Мы создали DMV как механизм, позволяющий видеть, *что именно выполняет* SQL Server в любой момент времени. Это

отличный способ узнать об активных подключениях и о том, какие запросы они выполняют. Однако часто, чтобы решить сложную проблему с производительностью, информации об активных подключениях и выполняемых ими запросах недостаточно – вам нужна подробная информация из плана выполнения запроса.

Итак, разрыв углублялся. Вы можете видеть, какой запрос выполняется, но вы не можете углубиться в план выполняемого запроса. Кроме того, если вам необходимы данные из плана выполнения для уже завершенного запроса, вам необходимо использовать то, что может оказаться *сложной* диагностикой, – расширенные события (Extended Events). Или же вам нужно найти конкретный запрос и запустить его в автономном режиме (то есть вне приложения), используя отдельный инструмент, при этом включив диагностику плана выполнения запроса, чтобы получить всю необходимую информацию.

Когда я присоединился к команде инженеров, работающих над выпуском новых версий SQL Server, то обнаружил, что известная команда проекта Tiger ведет работу над решением подобных проблем. Педро Лопес (Pedro Lopes), Алексей Эксаревский и Джей Чоу (Jay Choe) уже работали над *инфраструктурой профилирования запросов*. Если вы спросите любого разработчика о том, как выполняется трассировка кода, он будет использовать термин *профилирование*. Итак, как мы профилируем запрос в SQL Server? Обычно это сводится к получению информации о плане выполнения запроса. Необходимо получить информацию о плане во время выполнения запроса и получить фактический план выполнения запроса после его завершения.

Эта команда уже создала концепцию *статистики запросов в реальном времени* (вы можете получить дополнительную информацию на данную тему по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/performance/live-query-statistics). Было очевидно, что команда могла сделать больше. Как говорит Алексей: «Я хотел реализовать эту функцию в продукте еще в 2009 году… я потратил столько времени на разглядывание планов выполнения запросов и хотел, чтобы они ожили, чтобы легче было видеть, что происходит. Итак, идея живой статистики запросов. Эти два слова прекрасно дополняют друг друга, хотя, конечно, упрощенное профилирование позволяет сделать гораздо больше».

На самом деле команда создала инфраструктуру профиля статистики выполнения запросов, или *стандартное профилирование*. Эта возможность выдает вам фактическую статистику плана выполнения для каждого оператора по строкам, ЦП и вводу-выводу. Это ключевая информация для профилирования запроса; однако есть один нюанс. Вы должны включить профилирование перед выполнением запроса либо включить расширенные события для всех запросов, что, в свою очередь, может повлиять на производительность. Дополнительная информация о стандартном профилировании доступна по ссылке https://docs.microsoft.com/en-us/sql/relational-

databases/performance/query-profiling-infrastructure?#the-standard-query-execution-statistics-profiling-infrastructure.

Я люблю работать с такими коллегами, как Педро, Алексей и Джей. Они всегда спрашивают: «Можем ли мы улучшить эту функцию?» И конечно же, все они очень умны. По своему опыту они знают, как сложно и неудобно использовать стандартное профилирование. Они создали инфраструктуру упрощенного профилирования статистики выполнения запросов, или *упрощенное профилирование*. Концепция упрощенного профилирования состоит в том, чтобы получить данные профилирования для запросов без дополнительных затрат, которые непременно возникают при стандартном профилировании. Чтобы «облегчить» и «упростить» профилирование, нам пришлось исключить из него статистику ЦП; однако при этом вы все равно будете получать данные статистики по количеству строк «на оператор» и статистику ввода-вывода. Дополнительная информация об упрощенном профилировании доступна по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/performance/query-profiling-infrastructure?#lwp.

Это замечательно, но… нам все равно придется включить упрощенное профилирование, чтобы оно заработало. Как узнать, когда нам следует включать упрощенное профилирование? Ну, большинство пользователей SQL Server этого не знает. Никто этого не делает. Правильный ответ – просто запустить упрощенное профилирование по умолчанию. И это то, что можно сделать в SQL Server 2019. Педро называет это «инсайты относительно производительности в любое время и в любом месте». Есть ли в этом какие-то нюансы? Да. Вы получаете информацию о количестве строк только для активно выполняющихся запросов, но часто этого достаточно, чтобы помочь разобраться с проблемами производительности. Однако здесь есть и небольшой бонус. Мы добавили в упрощенное профилирование возможность получить последний фактический план выполнения для большинства кешированных запросов.

Давайте рассмотрим два сценария, чтобы вы могли понять преимущества использования упрощенного профилирования запросов по умолчанию в SQL Server 2019.

Подготовительные шаги для использования примеров, иллюстрирующих упрощенное профилирование запросов

Итак, во-первых, вам нужно выполнить некоторые настройки, чтобы использовать примеры для двух сценариев. Для примеров в этой главе вы будете использовать базу данных WideWorldImporters (подробную информацию об этой базе данных и ее схеме можно получить по ссылке https://docs.microsoft.com/en-us/sql/samples/wide-world-importers-oltp-database-catalog).

Приведенные здесь примеры будут работать на SQL Server 2019, установленном на Windows, Linux, а также в контейнерах. Учитывая большой раз-

мер набора данных, SQL Server потребуется как минимум 12 Гб ОЗУ, чтобы вы смогли заметить различия в производительности. Кроме того, в некоторых примерах запросов используется параллельная обработка, поэтому для полной демонстрации возможностей лучше установить SQL Server в многопроцессорной системе.

Все сценарии, применяемые в этой главе, можно найти в репозитории GitHub в каталоге **ch2_intelligent_performance\lwp**.

Чтобы использовать на практике примеры из этой главы, выполните следующие действия:

1. Загрузите резервную копию базы данных WideWorldImporters, доступную по ссылке https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImporters-Full.bak.
2. Восстановите эту базу данных на вашем экземпляре SQL Server 2019. Вы можете использовать для этого готовый сценарий **restorewwidw.sql**. Возможно, вам придется изменить путь к каталогу, где находятся ваша резервная копия, и путь к месту для восстановления файлов базы данных.
3. Для запуска некоторых примеров вам понадобятся таблицы большего размера, чем те, которые добавлены по умолчанию в WideWorldImporters. Поэтому запустите сценарий **extendwwi.sql**, чтобы создать таблицы большего размера. Расширение этой базы данных увеличит ее размер, включая журнал транзакций, примерно до 5 Гб. Одна из таблиц называется **Sales.InvoiceLinesExtended**. Взяв за основу таблицу **InvoiceLines**, мы намного увеличим ее размер и не будем использовать столбцовый индекс.

Нужно ли мне прерывать активный запрос?

Рассмотрим следующий сценарий. Вам говорят, что на SQL Server запущен запрос, который занимает много ресурсов ЦП на сервере. Вы используете DMV, например **sys.dm_exec_requests**, чтобы идентифицировать запрос и пользователя. Пользователь – ваш вице-президент, которому необходимо получить отчет, и при выполнении этого запроса используется кешированный план. Вы используете стандартные DMV, **sys.dm_exec_requests** и **sys.dm_exec_query_stats**, чтобы узнать, какой запрос выполняется. Как вы узнаете, завершится ли этот запрос в ближайшее время или же его следует прервать и исправить?

Давайте рассмотрим следующий пример и узнаем, как включенное упрощенное профилирование запросов может помочь вам найти ответ.

Вы можете запускать приведенные ниже сценарии T-SQL, применяя любой инструмент, который сможет подключиться к SQL Server, но лучше всего использовать SQL Server Management Studio (SSMS), чтобы увидеть все подробности.

1. Откройте сценарий T-SQL **mysmartquery.sql** (возможно, этот сценарий не столь «умен», как его название) и запустите выполнение пакета.
2. В новом соединении откройте сценарий T-SQL **show_active_queries.sql**.
3. Запустите пакет из **шага 1** сценария:

```
-- Шаг 1. Показывать только источники запросов с активными запросами,
-- кроме данного запроса
SELECT er.session_id, er.command, er.status, er.wait_type,
er.cpu_time, er.logical_reads, eqsx.query_plan, t.text
FROM sys.dm_exec_requests er
CROSS APPLY sys.dm_exec_query_statistics_xml(er.session_id) eqsx
CROSS APPLY sys.dm_exec_sql_text(er.sql_handle) t
WHERE er.session_id <> @@SPID
GO
```

   Этот код находит любые активные запросы (кроме текущего соединения). Если вы несколько раз повторно выполните данный запрос, то увидите, что значения cpu и logic_reads будут увеличиваться, и значение переменной wait_type будет wait_type = ASYNC_NETWORK_IO. Такая картина указывает на две вещи:

   - запрос отнимает большое количество ресурсов процессора и, вероятно, выполняет сканирование большой таблицы (значение logical_reads велико и продолжает увеличиваться);
   - большой объем данных (результатов) отправляется обратно клиенту (т. е. имеет место ожидание ASYNC_NETWORK_IO).

   По моему опыту, это не «хороший» запрос, а тот, который имеет «шансы на улучшение». Но вопрос в том, стоит ли прерывать его сейчас или он уже «почти выполнен».

4. Что было бы полезно узнать, когда запрос активен, – так это увидеть ход выполнения операторов из плана выполнения запроса, например статистики активных запросов (Live Query Statistics). Запустите **шаг 2** сценария:

```
-- Шаг 2. Как выглядит профиль плана выполнения для активного запроса
SELECT session_id, physical_operator_name, node_id, thread_id,
row_count, estimate_row_count
FROM sys.dm_exec_query_profiles
WHERE session_id <> @@SPID
ORDER BY session_id, node_id DESC
GO
```

Результаты должны выглядеть примерно так, как показано на рис. 2.21.

Results Messages

	session_id	physical_operator_name	node_id	thread_id	row_count	estimate_row_count
1	146	Index Scan	5	0	228265	228265
2	146	Index Scan	4	0	0	1
3	146	Hash Match	3	0	228265	228265
4	146	Table Spool	2	0	14639940	16094965150
5	146	Index Scan	1	0	65	70510
6	146	Nested Loops	0	0	14639940	16094965150

Рис. 2.21. Профиль плана выполнения для активного запроса

Заметьте, что значение estimate_row_count для операторов Nested Loops и Table Spool очень велико. И обратите внимание, что row_count (это количество строк, обрабатываемых в данный момент) намного меньше этого оценочного значения. Может быть, оценка является неточной; однако если она верна, этот запрос еще далек от завершения. Запустите этот запрос еще раз, чтобы увидеть, как изменяется значение row_count для этих операторов.

Примечание. Когда в SQL Server 2019 по умолчанию включено упрощенное профилирование запросов, то единственный параметр статистики, который собирается и сохраняется, – это row_count. Сбор данных, таких как загрузка процессора и объем ввода-вывода, включенный по умолчанию, может оказаться весьма дорогостоящим. Вы можете собрать эти данные, используя стандартное профилирование.

5. Давайте посмотрим на сам план выполнения запроса. Это предполагаемый план, но он может дать представление об этих больших оценочных числах row_count. Выполните **шаг 3** сценария:

```
-- Шаг 3. Вернитесь и просмотрите план выполнения запроса и текст
-- запроса для лучшего понимания
SELECT er.session_id, er.command, er.status, er.wait_type,
er.cpu_time, er.logical_reads, eqsx.query_plan, t.text
FROM sys.dm_exec_requests er
CROSS APPLY sys.dm_exec_query_statistics_xml(er.session_id) eqsx
CROSS APPLY sys.dm_exec_sql_text(er.sql_handle) t
WHERE er.session_id <> @@SPID
GO
```

В SSMS щелкните значение query_plan, после чего должно открыться новое окно с визуальным планом выполнения запроса.

План выполнения должен выглядеть так, как показано на рис. 2.22.

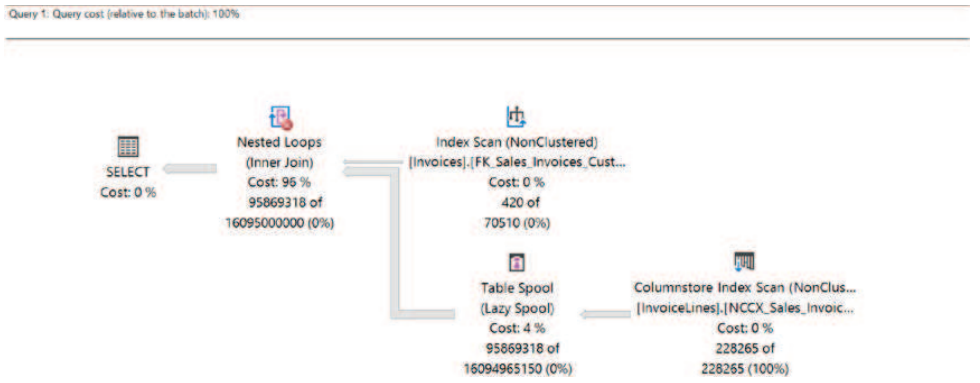


Рис. 2.22. План выполнения для активного запроса

Обратите внимание на значок с символом X на операторе соединения с использованием вложенных циклов (Nested Loops Join). Если вы наведете указатель мыши на оператор Nested Loops Join, то отображаемая информация будет выглядеть, как показано на рис. 2.23.

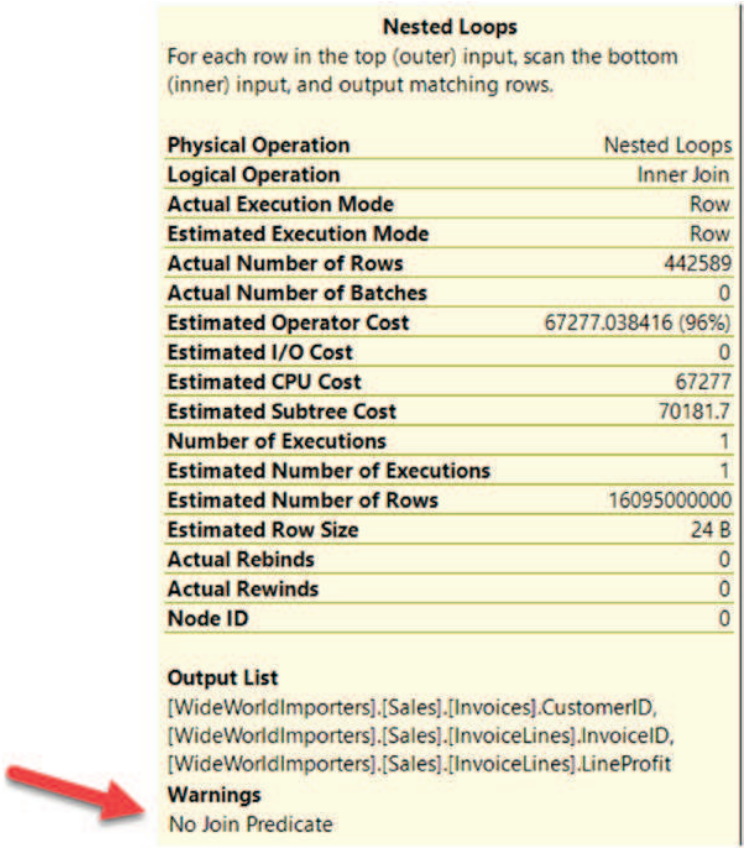


Рис. 2.23. Предупреждение, выведенное для оператора Nested Loops Join

Что означает текст «No Join Predicate» (отсутствует условие отбора для соединения)? Это означает, что существует серьезная проблема с оператором JOIN в запросе. Это означает, что соединение на основе равенства («equi» join) действительно не работает.

В результатах, возвращенных после выполнения шага 3, посмотрите на значение в текстовом столбце диагностики. Это выглядит так:

```
SELECT si.CustomerID, sil.InvoiceID, sil.LineProfit
FROM Sales.Invoices si
INNER JOIN Sales.InvoiceLines sil
ON si.InvoiceID = si.InvoiceID
OPTION (MAXDOP 1)
```

Поскольку проблема заключается в операторе JOIN, давайте сосредоточимся на операторе INNER JOIN:

```
INNER JOIN Sales.InvoiceLines sil
ON si.InvoiceID = si.InvoiceID
```

Вы видите, что этот запрос просто соединяет таблицу с самой собой. Простая опечатка: написано si, а не sil – вот в чем проблема. Этот запрос «зависнет», т. е. почти никогда не завершится. Его можно прервать или исправить, и тогда ваш вице-президент будет намного счастливее.

6. Отмените запрос из **mysmartquery.sql**, если он все еще выполняется.

Упрощенное профилирование запросов также включает в себя расширенные события и поддержку указаний запросов, и эти функции можно включить. Подробнее о том, как их включить, а также о том, как отключить эту функцию для каждой базы данных, можно узнать по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/performance/query-profiling-infrastructure?#lwp.

Я не могу его «поймать»!

Рассмотрим другой сценарий. Вы наблюдали увеличение загрузки ЦП SQL Server и не уверены, что нечто подобное должно происходить в нормальном режиме работы (потому что это не похоже на обычное поведение). Используя DMV, вы можете видеть значение различных параметров, например использовать **sys.dm_exec_query_stats**, чтобы узнать, где именно запросы больше всего загружают ЦП; однако при помощи этого динамического административного представления (DMV) вы получаете только предполагаемый план. Вы можете попытаться выполнить запрос самостоятельно в автономном режиме и получить фактический план, но вы хотите увидеть фактический план выполнения реального запроса из приложения, чтобы убедиться, что вы получаете корректные данные. Этот запрос постоянно выполняется многими пользователями, но время его выполне-

ния составляет всего несколько секунд (отсюда и постоянное увеличение загрузки ЦП), поэтому трудно использовать новые инструменты для того, чтобы «поймать» выполняемый запрос. Вы можете включить стандартное профилирование запросов, однако для этого требуется немало ресурсов, и использование стандартного профилирования может вызвать проблемы с производительностью приложений в момент выполнения запроса.

Вместе с упрощенным профилированием запросов в SQL Server 2019 появилась новая возможность. В SQL Server 2019 теперь есть новая функция динамического управления (Dynamic Management Function, DMF) с названием **sys.dm_exec_query_plan_stats**. Идея состоит в том, чтобы зафиксировать последний фактический план выполнения кешированного запроса. Вы можете прочитать об использовании этой DMF по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-query-plan-stats-transact-sql.

Давайте посмотрим, как применять эту DMF для поиска фактического плана выполнения запроса, который осуществляется постоянно, не применяя какие-либо специальные средства диагностики, не включая переключатели и флаги и не запуская запрос вручную. Единственный тонкий момент здесь заключается в том, что эту функцию облегченного профилирования запросов необходимо включить для каждой базы данных, где вы хотите ее использовать. Вы можете сделать это с помощью следующего оператора T-SQL, который будет использован в приведенном ниже примере:

```
ALTER DATABASE SCOPED CONFIGURATION SET LAST_QUERY_PLAN_STATS = ON
```

Все сценарии для этого примера размещены в каталоге **ch2_intelligent_performance\lwp**. Чтобы было проще увидеть планы выполнения в визуальном режиме, я рекомендую вам запустить этот пример с использованием SSMS.

1. Откройте сценарий T-SQL **mysmartquery_top.sql**.
2. Выполните предварительные настройки параметров для данного примера. Для этого выполните шаг 1 сценария:

```
-- Шаг 1. Очистите кеш процедур и установите значение параметра
-- dbcompat = 130, чтобы показать, что вам не нужно значение
-- dbcompat = 150 для вывода данных статистики последнего плана
выполнения.
USE WideWorldImporters
GO
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE
GO
ALTER DATABASE [WideWorldImporters] SET COMPATIBILITY_LEVEL = 130
GO
ALTER DATABASE SCOPED CONFIGURATION SET LAST_QUERY_PLAN_STATS = ON
```

```
GO
SELECT COUNT(*) FROM Sales.InvoiceLinesExtended
GO
```

Для параметра dbcompat установлено значение 130, чтобы показать, что вам не нужно устанавливать dbcompat=150 для использования этой функции.

3. Теперь давайте сэмулируем неверные данные статистики, выполнив **шаг 2** сценария:

```
-- Шаг 2. Имитация устаревших данных статистики: очень низкие значения
UPDATE STATISTICS Sales.InvoiceLinesExtended
WITH ROWCOUNT = 1
GO
```

4. Запустите запрос. Запрос будет выполняться всего несколько секунд, но он отнимет ресурсы ЦП. Выполните **шаг 3**.

Примечание. Вам не нужно выбирать фактический план выполнения, поскольку мы эмулируем ситуацию, когда вы будете просматривать планы независимо от приложения.

```
-- Шаг 3. Запустите запрос. Запрос будет выполняться всего несколько
-- секунд, но он отнимет ресурсы ЦП
SELECT si.InvoiceID, sil.StockItemID
FROM Sales.InvoiceLinesExtended sil
JOIN Sales.Invoices si
ON si.InvoiceID = sil.InvoiceID
AND sil.StockItemID >= 225
GO
```

5. Теперь давайте посмотрим на предполагаемый план выполнения для этого запроса, используя стандартные DMV. Выполните **шаг 4**, чтобы просмотреть предполагаемый план. Обратите внимание на то, что это позволяет вам увидеть план выполнения запроса из другого пользовательского подключения.

```
-- Шаг 4. Что говорит предполагаемый план выполнения? Выглядит
-- как корректный план, основанный на оценках
SELECT st.text, cp.plan_handle, qp.query_plan
FROM sys.dm_exec_cached_plans AS cp
CROSS APPLY sys.dm_exec_sql_text(cp.plan_handle) AS st
CROSS APPLY sys.dm_exec_query_plan(cp.plan_handle) AS qp
WHERE qp.dbid = db_id('WideWorldImporters')
GO
```

В выходных данных вам нужно найти строку, в которой значения текстового столбца начинаются с «Шаг 3». Щелкните значение query_plan для этой строки. Ваш план должен выглядеть подобно тому, что показано на рис. 2.24.

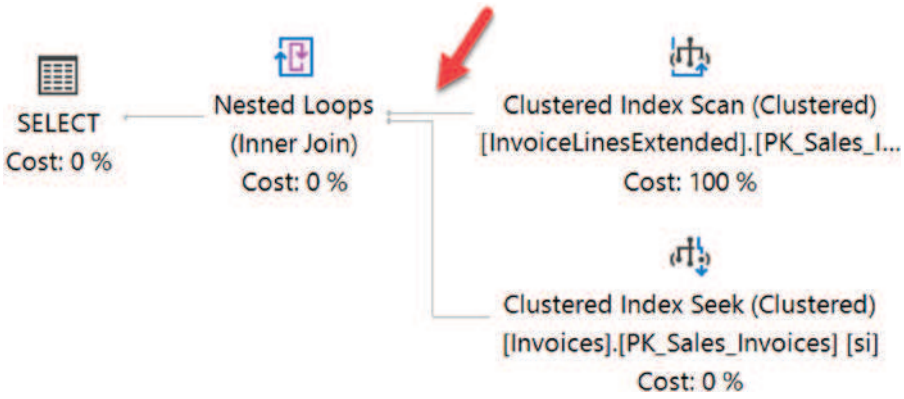


Рис. 2.24. Предполагаемый план выполнения для проблемного запроса

Обратите внимание, какая тонкая линия выходит из оператора сканирования кластеризованного индекса (Clustered Index Scan). Это связано с тем, что оптимизатор оценивает таблицу InvoiceLinesExtended как одну строку. Но это только предполагаемый план, и потому вы не знаете, что это не так (вы просто сэмулировали ошибочные строки предполагаемого плана, но притворились, что не знали, что эта оценка ошибочна).

6. Теперь давайте используем новую DMV, чтобы получить последний фактический план выполнения для этого запроса и посмотреть, являются ли строки предполагаемого плана неправильными. Выполните **шаг 5**.

```
-- Шаг 5. Что говорит последний фактический план выполнения? Ой-ой.
-- Расхождения между предполагаемым и фактическим планом очень велики.
SELECT st.text, cp.plan_handle, qps.query_plan, qps.*
FROM sys.dm_exec_cached_plans AS cp
CROSS APPLY sys.dm_exec_sql_text(cp.plan_handle) AS st
CROSS APPLY sys.dm_exec_query_plan_stats(cp.plan_handle) AS qps
WHERE qps.dbid = db_id('WideWorldImporters')
GO
```

В этом примере мы используем функцию **dm_exec_query_plan_stats** вместо dm_exec_query_plan. Снова найдите запрос в списке и щелкните значение query_plan. План выполнения запроса должен выглядеть так, как показано на рис. 2.25.

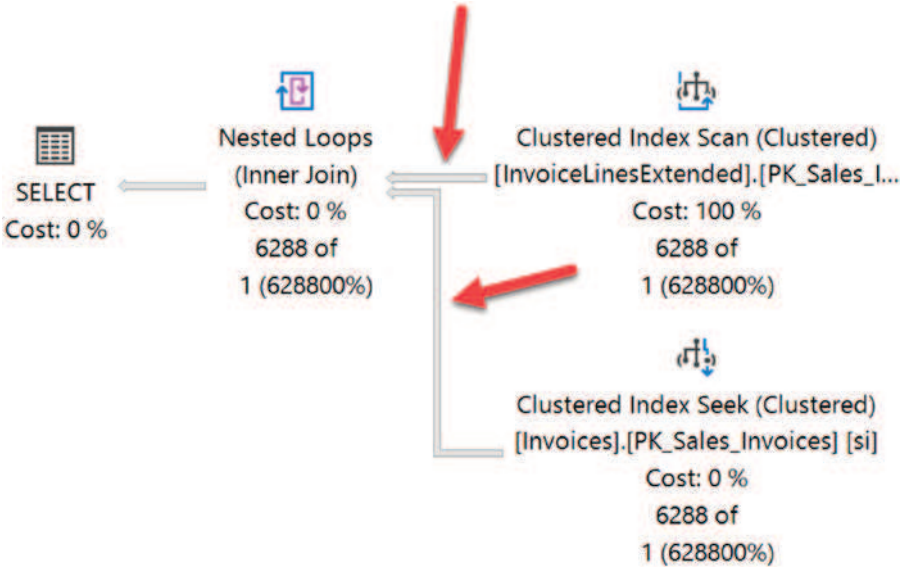


Рис. 2.25. Фактический план выполнения для проблемного запроса

Обратите внимание на «более толстые» линии. Они выглядят так потому, что фактическое количество обрабатываемых строк намного больше, чем оценочное значение. В этом заключается проблема, и это объясняет, почему оптимизатор принял решение использовать соединение таблиц с применением вложенных циклов (Nested Loops Join) и сделать таблицу InvoiceLinesExtended «внешней» таблицей: потому что он предполагал, что это только одна строка.

7. Обновите данные статистики, чтобы исправить их. После этого вы сможете увидеть, что на самом деле должен делать запрос. Выполните **шаг 6** сценария.

```
-- Шаг 6. Обновите статистику до правильного значения и очистите
-- кеш процедур.
UPDATE STATISTICS Sales.InvoiceLinesExtended
WITH ROWCOUNT = 3652240
GO
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE
GO
```

8. Запустите запрос еще раз, используя **шаг 7** сценария, и просмотрите новый фактический план выполнения. Вы заметите, что запрос работает немного быстрее:

```
-- Шаг 7. Запустите запрос еще раз. Он работает быстрее.
SELECT si.InvoiceID, sil.StockItemID
FROM Sales.InvoiceLinesExtended sil
```

```
JOIN Sales.Invoices si
ON si.InvoiceID = sil.InvoiceID
AND sil.StockItemID >= 225
GO
```

9. Выполните **шаг 8**, чтобы посмотреть, улучшился ли новый фактический план.

```
-- Шаг 8. Как выглядит фактический план на этот раз?
-- Он отличается от предыдущего, потому что статистика актуальна
SELECT st.text, cp.plan_handle, qps.query_plan
FROM sys.dm_exec_cached_plans AS cp
CROSS APPLY sys.dm_exec_sql_text(cp.plan_handle) AS st
CROSS APPLY sys.dm_exec_query_plan_stats(cp.plan_handle) AS qps
WHERE qps.dbid = db_id('WideWorldImporters')
GO
```

Щелкнув мышью query_plan для строки, которая соответствует тексту «Шаг 7. Запустите запрос еще раз...», вы увидите план, аналогичный представленному на рис. 2.26.

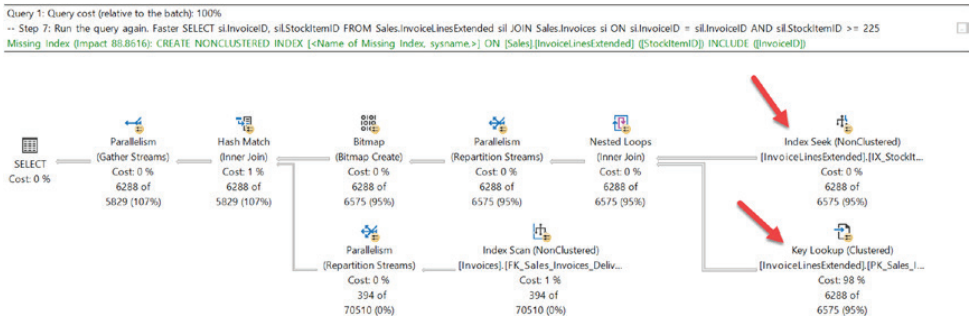


Рис. 2.26. Фактический план выполнения для улучшенного запроса

План выполнения запроса кардинально отличается от предыдущего. Вы можете видеть, что в этом случае оптимизатор создает план, в котором сначала выполняется поиск по индексу для таблицы InvoiceLinesExtended, а затем операция соединения таблицы с самой собой с помощью поиска по ключу кластеризованного индекса (Key Lookup). Для запроса, использующего доступные индексы, это гораздо более эффективный способ выполнить соединение промежуточного результата запроса с другими таблицами и отфильтровать полученные результаты.

Теперь, когда у вас есть возможность в любой момент увидеть фактический план выполнения запроса и вам не нужно включать специальные флаги, которые могут выполнять дорогостоящие операции, или запускать запрос в автономном режиме, вы видите, что эта возможность дополняет мощный арсенал средств настройки произво-

дительности запросов и устранения неполадок, имеющихся в вашем распоряжении.

Упрощенное профилирование запросов – это просто здорово! Если вы потратили какое-то время на поддержку SQL-серверов в промышленной среде, решая проблемы с производительностью, то согласитесь, что встроенная диагностика, доступная в любое время и в любом месте, – это поистине глоток свежего воздуха.

База данных в памяти

В SQL Server 2014 мы представили функцию OLTP в памяти (In-Memory OLTP), ключевой концепцией которой была концепция таблиц, оптимизированных для выделения памяти. При использовании этой функции вся таблица хранится в памяти, но именно *оптимизированный* (читайте: без кратковременных блокировок, latch-free) доступ делает ее особенной. В SQL Server 2016 мы значительно усовершенствовали функцию In-Memory OLTP. Более подробно об этой функции рассказывается по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp.

Когда мы работали над новыми функциями для SQL Server 2019, Слава Окс (Slava Oks), Пэм Лахуд (Pam Lahoud), Брайан Карриг (Brian Carrig), Аргенис Фернандес (Argenis Fernandez) и другие члены команды инженеров собрались вместе и решили совместно назвать новый набор функций *базой данных в памяти* (In-Memory Database). Это новое семейство функций, в которые входит и In-Memory OLTP.

То, что в SQL Server 2019 названо базой данных в памяти, включает в себя целый набор различных функций:

- In-Memory OLTP;
- оптимизированные для выделения памяти метаданные TempDB;
- гибридный буферный пул;
- поддержка постоянной памяти.

Полный список этого набора функций доступен по адресу https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-database?view=sqlallproducts-allversions.

В этом разделе мы рассмотрим все эти новые возможности, кроме In-Memory OLTP (которая не является «новшеством» SQL Server 2019).

Метаданные TempDB, оптимизированные для выделения памяти

На протяжении всего времени, в течение которого я работал с SQL Server, производительность параллельных подключений при использовании временных таблиц была очень низкой. Это приводило к тому, что почти каждый администратор SQL Server настраивал tempdb для использования

нескольких файлов. Вы можете прочитать больше об истории приключения с этими ресурсами:

- Inside TempDB (Подробно о TempDB) – выступление Боба Уорда (Bob Ward) на саммите PASS 2017 года (https://www.youtube.com/watch?v=SvseG-Mobe2w);
- блог Пола Рэндала (Paul Randal) о добавлении файлов tempdb (https://www.sqlskills.com/blogs/paul/correctly-adding-data-files-tempdb/).

Один из аспектов использования файлов tempdb, которому большинство профессионалов SQL не уделяют особого внимания (поскольку сейчас это стало обычной практикой), заключается в том, что вы создаете для механизма SQL Server схему разделения данных для доступа к страницам размещения, таким как страницы PFS, GAM и SGAM. Подобный тип схемы полезен, потому что работа с использованием временных таблиц приводит к «утяжелению» таких операций, как создание таблиц, выделение страниц, удаление таблиц. В этом случае возникает ситуация конкуренции на этих системных страницах размещения – так называемые кратковременные блокировки, или «защелки» (latches). Создавая несколько файлов, вы распространяете на них и конкуренцию за временные блокировки на страницах этих файлов, что повышает производительность в ситуации, когда с tempdb работает множество параллельных подключений.

После создания нескольких файлов (начиная с версии SQL Server 2016 программа установки может сделать это автоматически, или же вы можете настроить этот параметр установки вручную) при более высокой одновременной рабочей нагрузке на базу данных tempdb число ожиданий временно заблокированных страниц может увеличиться. Эти ожидания затрагивают страницы, принадлежащие объектам, которые вы не можете распознать, например такие, как sysschobjs. Это ожидания блокировки страниц для страниц системных таблиц в базе данных tempdb. При быстром создании и удалении таблиц SQL Server должен выполнять внутренние операции чтения/записи на страницах системных таблиц, чтобы поддерживать согласованность метаданных таблиц. Эти операции приводят к увеличению конкуренции пользователей за страницы, где установлены кратковременные блокировки, и, следовательно, к увеличению времени ожидания пользователей в очереди доступа к заблокированному ресурсу. В прошлом, когда клиенты сталкивались с высокой конкуренцией за заблокированные страницы в системных таблицах и обращались ко мне в службу поддержки, я отвечал: «К сожалению, вы должны уменьшить нагрузку на базу данных tempdb, чтобы избежать этой проблемы».

Пэм Лахуд (Pam Lahoud) очень хорошо описывает эту проблему в своем блоге, размещенном по адресу https://blogs.msdn.microsoft.com/sql_server_team/tempdb-files-and-trace-flags-and-updates-oh-my/.

В SQL Server 2019 появляется решение – метаданные tempdb, оптимизированные для выделения памяти. В таблицах, оптимизированных для

выделения памяти (помните известный проект Hekaton («Гекатон»)?) не может быть кратковременных блокировок страниц, поскольку все данные этих таблиц размещены и хранятся в памяти. Если таблицы, оптимизированные для выделения памяти, являются «просто схемой», то у них нет ограничений, связанных со временем их жизни. Это идеальная платформа для системных таблиц tempdb. Поскольку база данных tempdb создается заново при каждом перезапуске сервера, системные таблицы не должны быть долговечными. А так как в таблицах, оптимизированных для выделения памяти, хранятся только метаданные (а не данные во временных таблицах), потребление памяти для них должно быть небольшим. Равиндер Вуппула (Ravinder Vuppula), ведущий разработчик этого проекта, назвал создание системных таблиц tempdb «гекатонизированным».

Метаданные tempdb по умолчанию не используют таблицы, оптимизированные для выделения памяти, при установке SQL Server. Вы должны выполнить следующую команду T-SQL, чтобы включить эту возможность:

```
ALTER SERVER CONFIGURATION SET MEMORY_OPTIMIZED TEMPDB_METADATA = ON
```

Вы можете просмотреть мою демонстрацию данной функции на выступлении SQLBits 2019 по ссылке https://sqlbits.com/Sessions/Event18/Keynote. Брент Озар (Brent Ozar) в своем блоге сказал об этой возможности, увидев ее демонстрацию на саммите PASS в 2018 году: «…это улучшение работы с TempDB просто великолепно. Это реальное улучшение, которое будет иметь огромную популярность. Люди долго боролись с неразрешимыми проблемами конкуренции TempDB и с блокировками и никак не могли справиться с этим».

Но вы должны попробовать эту возможность сами, чтобы увидеть ее в действии. Давайте рассмотрим пример того, как метаданные tempdb, оптимизированные для выделения памяти, могут значительно улучшить параллельную работу приложений, использующих временные таблицы. Все сценарии для этого примера размещены в каталоге **ch2_intelligent_performance\inmem\tempdb**. Этот пример немного сложнее, и для него потребуется координировать или эмулировать одновременную работу нескольких пользователей. Поэтому вам понадобится бесплатный инструмент для нагрузочного тестирования под названием ostress, который можно загрузить по ссылке www.microsoft.com/en-us/download/details.aspx?id=4511. Этот инструмент в настоящее время доступен только для Windows. Данный пример будет по-прежнему работать с SQL Server, установленным на Linux; просто вам понадобится клиент Windows для управления параллельными пользовательскими подключениями с помощью ostress.

Кроме того, я настроил свой SQL Server на виртуальной машине с восемью логическими процессорами. Когда я запустил программу установки SQL Server, она автоматически создала восемь файлов данных tempdb. Я рекомендую вам убедиться, что у вас есть как минимум восемь файлов данных tempdb, если в вашей системе имеется восемь или более

логических процессоров. Подробнее о том, как это сделать, читайте в следующей статье технической поддержки: https://support.microsoft.com/en-us/help/2154845/recommendations-to-reduce-allocation-contention-in-sql-server-tempdb-d.

1. Запустите сценарий **disableopttempdb.cmd**, чтобы отключить функцию оптимизации метаданных tempdb для выделения памяти. По умолчанию эта возможность отключена; однако рекомендуется запустить этот сценарий на тот случай, если она была включена в какой-то момент. Вам нужно запустить данный сценарий *на том сервере, где установлен SQL Server* (или использовать другой метод для удаленного перезапуска SQL Server). Для этого сценария понадобится ввести имя учетной записи системного администратора и имя сервера. Вы можете изменить этот способ аутентификации и использовать встроенную аутентификацию, изменив -Usa на -E, и не забудьте заменить текущее значение параметра -S на имя вашего сервера:

```
sqlcmd -Usa -idisableopttempdb.sql -Sbwsql2019
net stop mssqlserver
net start mssqlserver
```

   Как видите, этот сценарий дает инструкцию Windows Server перезапустить службу SQL Server (mssqlserver). Вы можете написать свой собственный сценарий для Linux, используя команду типа `sudo systemctl restart mssql-server` для перезапуска SQL Server.

   Сценарий **disableopttempdb.sql** содержит следующую инструкцию T-SQL:

```
ALTER SERVER CONFIGURATION SET MEMORY_OPTIMIZED TEMPDB_
METADATA = OFF
GO
```

2. Запустите сценарий T-SQL **tempstress_ddl.sql**, чтобы создать базу данных и хранимую процедуру, которая просто создает временную таблицу:

```
DROP DATABASE IF EXISTS DallasMavericks
GO
CREATE DATABASE DallasMavericks
GO
USE DallasMavericks
GO
CREATE OR ALTER PROCEDURE letsgomavs
AS
CREATE TABLE #gomavs (col1 INT)
GO
```

Как видите, хранимая процедура на самом деле ничего не делает с временной таблицей. Это показывает минимальный объем рабочей нагрузки, который может повлиять на параллельную работу с метаданными временной таблицы (поскольку любой выход из хранимой процедуры автоматически удаляет временную таблицу).

3. Теперь вы готовы к тому, чтобы создать паралельную рабочую нагрузку на tempdb с помощью ostress. Используйте сценарий **tempstress.cmd** для создания этой рабочей нагрузки с применением ostress:

```
ostress -Usa -Q"exec letsgomavs" -n50 -r10000
-dDallasMavericks -Sbwsql2019
```

Вам может потребоваться изменить некоторые из параметров сценария, например изменить -Usa на -E для использования встроенной аутентификации Windows. Вы также можете изменить имя сервера (изменив параметр -S). Параметр -n50 – это количество пользователей, создающих рабочую нагрузку, а -r10000 – количество итераций для каждого пользователя. Обратите внимание на использование параметра -Q для непосредственного запуска хранимой процедуры – прием, который я узнал, работая над ранними версиями этого сценария для демонстрации. Использование параметра -Q для ostress для непосредственного запуска запроса работает быстрее, чем инструкция сценария с -i.

Если вы используете учетную запись администратора (-Usa), вам будет предложено ввести пароль, после чего гонки начнутся. В зависимости от мощности вашего компьютера этот нагрузочный сценарий будет выполняться несколько минут.

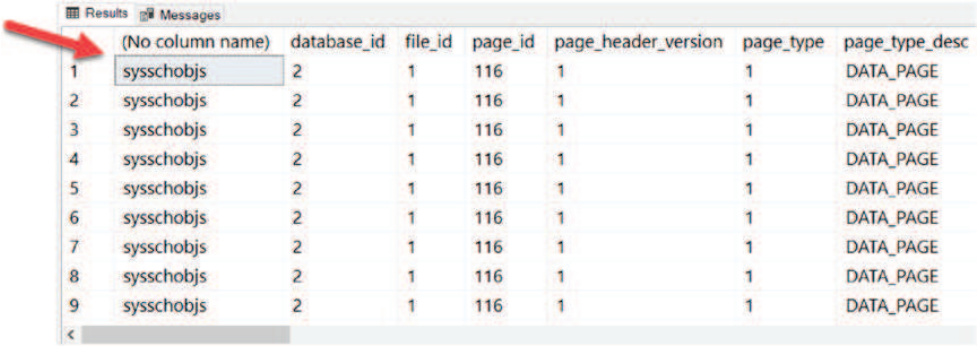
4. Пока он выполняется, создайте новое подключение к SQL Server и откройте сценарий T-SQL **pageinfo.sql**:

```
USE tempdb
GO
SELECT object_name(page_info.object_id), page_info.*
FROM sys.dm_exec_requests AS d
  CROSS APPLY sys.fn_PageResCracker(d.page_resource) AS r
  CROSS APPLY sys.dm_db_page_info(r.db_id, r.file_id,
r.page_id,'DETAILED')
    AS page_info
GO
```

Этот сценарий использует новые функциональные возможности в SQL Server 2019 для извлечения информации о странице из ресурса, указанного в **sys.dm_exec_requests**, и для выгрузки в многоколоночном формате полей заголовка страницы.

Почему и в каком случае вам нужно использовать этот сценарий? Потому, что когда у вас появляются ожидания кратковременных блокировок страниц для базы данных tempdb, вы одновременно с этим видите отчет о ресурсах в формате <dbid>: <fileid>: <pageid>. До того, как появилась эта функция, вам нужно было вручную выполнять команду DBCC PAGE (команду, которая официально не поддерживается), чтобы выяснить, к какому объекту относится страница, для которой наблюдается ожидание снятия кратковременной блокировки. Метод, продемонстрированный здесь, теперь официально поддерживается при определении страницы в сценарии ожидания снятия кратковременной блокировки.

Ваши результаты при выполнении запроса на предыдущем шаге должны выглядеть примерно так, как показано на рис. 2.27.



Results | Messages

	(No column name)	database_id	file_id	page_id	page_header_version	page_type	page_type_desc
1	sysschobjs	2	1	116	1	1	DATA_PAGE
2	sysschobjs	2	1	116	1	1	DATA_PAGE
3	sysschobjs	2	1	116	1	1	DATA_PAGE
4	sysschobjs	2	1	116	1	1	DATA_PAGE
5	sysschobjs	2	1	116	1	1	DATA_PAGE
6	sysschobjs	2	1	116	1	1	DATA_PAGE
7	sysschobjs	2	1	116	1	1	DATA_PAGE
8	sysschobjs	2	1	116	1	1	DATA_PAGE
9	sysschobjs	2	1	116	1	1	DATA_PAGE

Рис. 2.27. Кратковременная блокировка страницы: ожидание системных таблиц в базе данных tempdb

Как описано выше, sysschobjs – это системная таблица, являющаяся «узким местом», поскольку временные таблицы создаются и удаляются.

5. Теперь давайте включим оптимизацию метаданных tempdb для выделения памяти. Запустите сценарий **optimizetempdb.cmd** на сервере, где установлен SQL Server. Этот сценарий выполняет следующее, однако вы можете использовать другие методы для включения функции и перезапуска SQL Server:

```
sqlcmd -Usa -ioptimizetempdb.sql -Sbwsql2019
net stop mssqlserver
net start mssqlserver
```

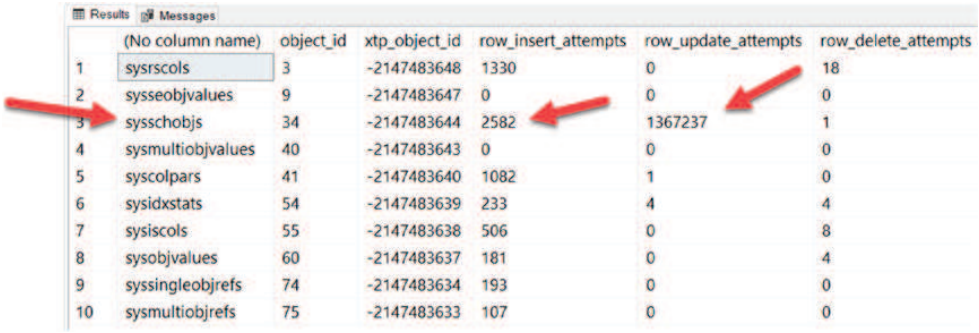
Сценарий **optimizetempdb.sql** содержит следующую инструкцию T-SQL:

```
ALTER SERVER CONFIGURATION SET MEMORY_OPTIMIZED TEMPDB_METADATA = ON
GO
```

6. Убедитесь, что оптимизация метаданных tempdb для выделения памяти включена, изучив файл журнала ошибок SQL Server ERRORLOG. Вы должны увидеть такую запись в ERRORLOG:

```
Tempdb started with memory-optimized metadata.
```

7. Теперь снова запустите сценарий **tempstress.cmd**, создающий рабочую нагрузку. На этот раз при той же рабочей нагрузке, без изменений в приложении, для выполнения сценария потребуется чуть более 30 секунд.
8. Запустите сценарий **pageinfo.sql** еще раз, чтобы увидеть, имеет ли место ожидание снятия кратковременной блокировки страницы. Ваши результаты должны содержать 0 строк!
9. Пока выполняется сценарий, запущенный на предыдущем шаге, создайте новое подключение к SQL Server и в нем запустите сценарий T-SQL **find_memoptimized_tables.sql**. Ваши результаты должны выглядеть примерно так, как показано на рис. 2.28.



Results | Messages

	(No column name)	object_id	xtp_object_id	row_insert_attempts	row_update_attempts	row_delete_attempts
1	sysrscols	3	-2147483648	1330	0	18
2	sysseobjvalues	9	-2147483647	0	0	0
3	sysschobjs	34	-2147483644	2582	1367237	1
4	sysmultiobjvalues	40	-2147483643	0	0	0
5	syscolpars	41	-2147483640	1082	1	0
6	sysidxstats	54	-2147483639	233	4	4
7	sysiscols	55	-2147483638	506	0	8
8	sysobjvalues	60	-2147483637	181	0	4
9	syssingleobjrefs	74	-2147483634	193	0	0
10	sysmultiobjrefs	75	-2147483633	107	0	0

Рис. 2.28. Системные таблицы tempdb после оптимизации для выделения памяти

В окне результатов вы можете видеть все системные таблицы, оптимизированные для выделения памяти (ваш результат может содержать даже больше таблиц, поскольку эта функция улучшается). Обратите внимание на серьезные изменения значений в sysschobjs (но это не единственная системная таблица, которую затронула оптимизация).

Вы можете задаться вопросом: сколько дополнительной памяти потребляется при использовании этой функции? Пока вы все еще работаете в данной среде, выполните запрос к DMV **sys.dm_os_memory_clerks**. Вы увидите строку, в которой будут содержаться записи: type = MEMORYCLERK_XTP и name = DB_ID_2. Столбец pages_kb – это примерное количество памяти, потребляемой метаданными tempdb, оптимизированными для выделения памяти. Как мы видим, для этого примера оно составляет около 200 Мб.

На данном этапе вы можете оставить флаг, переключающий оптимизацию для выделения памяти метаданных tempdb, включенным для

своего сервера, но если вы хотите выключить его, используйте сценарий **disableopttempdb.cmd**.

Вы можете видеть преимущества этой функции, встроенной в ядро СУБД. Вам нужно лишь установить один параметр конфигурации сервера, перезапустить SQL Server – и уже можно начинать работать.

Если вы имеете доступ к представлениям каталога (catalog views) в базе данных tempdb, то можете заметить, что при использовании метаданных tempdb, оптимизированных для выделения памяти, существует несколько ограничений. Вы можете прочитать об этих ограничениях, а также найти более подробную информацию об этой возможности, в документации, размещенной по адресу: https://docs.microsoft.com/en-us/sql/relational-databases/databases/tempdb-database?view=sqlallproducts-allversions#memory-optimized-tempdb-metadata.

Гибридный буферный пул

Устройства с постоянной памятью существуют уже несколько лет, но сейчас они начинают набирать популярность. Концепция основана на постоянно работающей аппаратной памяти с бесперебойным источником питания. Подумайте о скорости оперативной памяти, обладающей еще одним новым преимуществом: любые сохраненные данные гарантированно выдержат перезапуск питания. Одним из наиболее популярных решений в области постоянной памяти является постоянная память от Intel под названием Optane (www.intel.com/content/www/us/en/architecture-and-technology/optane-technology/optane-for-data-centers.html).

Наша команда разработчиков SQL Server все время ищет способы оптимизировать доступ к данным, и при наличии постоянной памяти существует несколько возможностей для такой оптимизации. Фактически еще в SQL Server 2016 существовала функция, называемая «остатком кеширования журнала», основанная на использовании постоянной памяти (см. сообщение в блоге Кевина Фарли (Kevin Farlee) на эту тему https://docs.microsoft.com/ru-ru/archive/blogs/sqlserverstorageengine/transaction-commit-latency-acceleration-using-storage-class-memory-in-windows-server-2016sql-server-2016-sp1).

Поскольку постоянная память – это одна из разновидностей памяти, SQL Server может обращаться к любым данным, хранящимся на постоянном запоминающем устройстве, как будто это обычная память. Это означает, что SQL Server может найти способ обойти код ядра при обработке ввода-вывода, выполняемого при доступе к данным на устройствах постоянной памяти.

Одной из таких новых возможностей является *гибридный буферный пул*. Идея заключается в том, что если вы разместите файлы данных вашей базы данных в постоянной памяти, SQL Server может просто получить доступ к страницам файла данных с этого устройства без необходимости копировать данные из файла данных на страницу буферного пула. Гибридный буферный пул использует вызовы ядра, привязанные к памяти, чтобы реализовать концепцию базы данных в памяти. Если страница базы данных была

изменена, ее необходимо скопировать в буферный пул и затем в конечном итоге записать обратно на постоянное запоминающее устройство.

Производительность такой системы может варьироваться в зависимости от возможности использования тех преимуществ, которые предоставляет использование гибридного буферного пула, но обычно от этой технологии можно ожидать некоторого повышения производительности, особенно если для вашей базы данных характерны высокие нагрузки на чтение.

В SQL Server, если вы разместили один или несколько файлов базы данных на постоянном запоминающем устройстве, вы можете включить гибридный буферный пул для всех баз данных для вашего сервера с помощью оператора T-SQL:

```
ALTER SERVER CONFIGURATION SET MEMORY_OPTIMIZED HYBRID_BUFFER_POOL = ON
```

> **Примечание.** При включении гибридного буферного пула для всех баз данных необходимо перезапустить SQL Server.

Вы можете включить гибридный буферный пул для конкретной базы данных с помощью следующего оператора T-SQL (при этом вам не потребуется перезапускать сервер):

```
ALTER DATABASE <имя_базы_данных> SET MEMORY_OPTIMIZED = ON
```

Чтобы узнать больше о том, как включить на ваших устройствах постоянную память для баз данных, как отключить гибридный буферный пул и как использовать гибридный буферный пул, обратитесь к документации по адресу https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/hybrid-buffer-pool.

Поддержка постоянной памяти

Если вы не хотите включать гибридный буферный пул, но хотели бы, чтобы SQL Server использовал возможности чтения и записи данных и журнала транзакций на устройствах с постоянной памятью, вы можете настроить используемое устройство как устройство постоянной памяти в Linux. SQL Server автоматически обнаружит его и будет использовать операции на основе памяти для перемещения данных в кеш SQL Server и на устройство, минуя стек ввода-вывода ядра Linux. Этот процесс называется «просветлением» (элементом паравиртуализации).

Dell EMC показала значительные улучшения производительности при использовании «просветления», как описано в публикации www.emc.com/about/news/press/2019/20190402-01.htm. По словам Dell, «благодаря новой постоянной памяти Intel® Optane™ DC клиенты могут ускорить работу баз данных в оперативной памяти, виртуализации и анализа данных, увеличив объем памяти в 2,5 раза для отдельных серверов PowerEdge. Используя постоянную память Intel® Optane™ DC, PowerEdge R740xd обеспечива-

ет увеличение объема обрабатываемых транзакций до 2,7 раза, по сравнению с числом транзакций в секунду, обеспечиваемым дисками NVMe в виртуализированной среде предварительного просмотра Microsoft SQL Server 2019, работающим на VMware ESXi.1».

Вы можете прочитать все подробности о том, как включить устройство постоянной памяти в Linux для SQL Server, по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-configure-pmem?view=sqlallproducts-allversions.

Конфликт вставки на последней странице

Давайте рассмотрим распространенную проблему пользователей SQL Server, которая оставалась нерешенной в течение очень долгого времени. Вы хотите построить таблицу с первичным ключом, который будет использоваться в кластеризованном индексе. И для этого первичного ключа используются *последовательные* значения. Другими словами, каждая вставка строки приводит к добавлению нового значения ключа в порядке увеличения. Наиболее распространенным примером такого ключа является столбец, использующий объект SEQUENCE или свойство IDENTITY.

Хотя этот подход в большинстве случаев работает нормально, все же здесь скрывается сложная проблема, связанная с производительностью приложений. Каждый раз, когда запрос должен изменить страницу, SQL Server должен физически защищать другие запросы от одновременного изменения или чтения структуры страницы (даже при наличии такой возможности, как блокировка на уровне строк), используя кратковременную блокировку страницы.

Если бы все пользователи пытались изменить одну и ту же страницу, производительность вашего приложения ухудшилась бы из-за конкуренции за блокировку страницы. Если вы строите кластеризованный индекс по последовательному ключу, данные сортируются по этому ключу. Каждая вставка будет пытаться вставить новую строку на последней странице листового уровня кластеризованного индекса. И если многие пользователи одновременно выполняют вставки, все они могут в конечном итоге попытаться изменить последнюю страницу индекса, отсюда и термин «конфликт вставки на последней странице».

Хотя описанная ситуация и не идеальна, она, как правило, не вызывает больших проблем, пока не произойдет явление, называемое «*колонной*» кратковременных блокировок. Пэм Лахуд (Pam Lahoud), старший руководитель программы в команде, работающей над SQL Server (известная также как @SQLGoddess), показала мне следующий ресурс, посвященный проблеме с «колонной»: https://blog.acolyer.org/2019/07/01/the-convoy-phenomenon/. В контексте проблемы конкуренции за вставку на последней странице для SQL Server разделение страницы является примером сценария, в котором может сформироваться «колонна». Разделение страницы может произойти, если на странице недостаточно строк для нового INSERT и в кластеризо-

ванном индексе должна быть создана новая страница. Пэм также провела отличную аналогию с проблемой «колонны». По словам Пэм, «пробки – обычная аналогия, используемая при описании этой проблемы. Если у вас есть дорога, по которой движется транспортный поток, равный максимальной пропускной способности этой дороги, то пока весь поток продолжает двигаться с одинаковой скоростью, движение будет постоянным, хотя и немного замедленным. Как только на дороге происходит что-то, приводящее к тому, что водители нажимают на тормоза, например впереди оказывается водитель, едущий слишком медленно, возникает опасность на дороге или автомобили приближаются к сложной развязке, движение затрудняется. Если автомобили продолжают подъезжать к месту затора движения с той же скоростью, что и раньше, ситуация на дороге становится все хуже и хуже. Водители все еще продвигаются вперед, но очень медленно. На этом этапе пропускная способность не восстановится до тех пор, пока число автомобилей, въезжающих на дорогу, резко не уменьшится и не станет намного ниже, чем то значение транспортного потока, которое соответствует пропускной способности дороги».

Многие пользователи в сообществе SQL, технической поддержке и разработке решали эту проблему различными способами на протяжении многих лет. О многих из них упоминается в статье технической поддержки, размещенной на странице https://support.microsoft.com/kb/4460004. А как насчет решения, реализованного внутри самого ядра СУБД и которое не потребует внесения изменений в приложение? Когда я увидел, что наше решение появилось в версии SQL Server 2019 CTP 3.1, я знал, что эта проблема уже обсуждалась нашей командой инженеров, рассмотревших множество возможных решений. Я расспросил Вонсок Ким (Wonseok Kim), ведущего разработчика этой функции, об истории ее реализации. Он показал мне ветку сообщений электронной почты, которая на самом деле имелась в моей почтовой папке, но я забыл о ней. Оказывается, в переписке, относящейся к работе над этим подходом, часто встречалось знакомое имя – Слава Окс (Slava Oks). Слава работал над этой идеей вместе со многими другими выдающимися программистами из команды разработчиков SQL Server.

Решение теперь существует в виде нового параметра для создания индексов – **OPTIMIZE_FOR_SEQUENTIAL_KEY**. Добавив этот параметр в ограничение индекса или первичного ключа, вы сообщаете SQL Server, что нужно использовать новый код, чтобы попытаться избежать проблем с «колонной» . Этот параметр не устраняет кратковременные блокировки страниц и не предотвращает проблему конфликта кратковременных блокировок. Что он делает, так это старается избежать серьезных проблем в ситуации, когда образуется «колонна», чтобы выровнять вашу рабочую нагрузку на сервер, сделать ее постоянной.

Подробнее об этой возможности и о том, как ее использовать, вы можете прочитать в документации по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/create-index-transact-sql?view=sqlallproducts-allversions#sequential-keys.

> **Примечание.** Если вы используете этот параметр, то можете заметить появившийся новый тип ожиданий (wait_type) с названием BTREE_INSERT_FLOW_CONTROL. Он является частью механизма, позволяющего избежать или уменьшить проблему с «колонной».

Однако этот вариант не для всех. Если вы не используете последовательные значения ключа для кластеризованного индекса или не видите серьезных конфликтов, я бы не рекомендовал включать этот параметр. Фактически вы можете сильно ухудшить производительность вашей системы, слепо применяя его в любом кластеризованном индексе.

Если вы хотите попробовать это сами, убедитесь, что у вас есть достаточно «широкая» таблица. Когда я тестировал эту возможность, в моем случае простое создание таблицы с одним столбцом IDENTITY не привело к существенному увеличению производительности. Что вам нужно сделать – так это вызвать условия, при которых происходит достаточное количество разбиений страниц, чтобы заметить проблему с «колонной».

> **Примечание.** Возможно, что методы, описанные в статье https://support.microsoft.com/kb/4460004, помогут вам обеспечить более высокую производительность, но этот новый параметр для индекса может дать вам необходимую стабильную производительность, и при этом вам придется гораздо меньше вмешиваться в работу вашего приложения.

Резюме

Эта глава была очень длинной и показала невероятные возможности интеллектуальной настройки производительности, встроенные в SQL Server 2019, призванные помочь вам повысить производительность без изменения уже работающих приложений. Я привел в ней много подробных примеров, чтобы вы могли сами убедиться в богатых возможностях новых функций, а также посмотреть, как они могут помочь повысить производительность и сэкономить ваше время, затрачиваемое на настройку производительности как при развертывании SQL Server 2019 в вашей организации, так и в разработке пользовательских приложений.

Глава 3

Новые возможности безопасности

Безопасность имеет важное значение для управления данными. У SQL Server надежная репутация не только в области создания защищенных продуктов, но и в области предоставления всех необходимых возможностей, которые помогут вам защитить ваши данные и получить доступ к экземпляру SQL Server. Эта глава о том, как мы расширили функции безопасности в SQL Server 2019.

Улучшение достигнутых показателей

Прочитав и попрактиковавшись в выполнении примеров из предыдущей, очень длинной главы о производительности, вы можете бегло просмотреть эту главу, сосчитать количество страниц в ней и спросить себя: «Разве безопасность – это не важно?» Ответ: разумеется, важно! Для SQL Server безопасность является одним из важнейших компонентов общей платформы данных.

Новые возможности безопасности в SQL Server 2019 и задачи, для решения которых они предназначены, включают в себя такие, как:

- **стратегия постоянного шифрования с защищенными областями** (Always Encrypted with Secure Enclaves).

  Цель – предоставить комплексное решение для шифрования, но не ограничивать возможности приложений в части запросов к данным;
- **классификация данных и аудит** (Data Classification and Auditing).

  Цель – предоставить встроенную систему классификации объектов SQL Server, а также обеспечить возможность аудита для просмотра данных, помеченных для классификации;
- **прозрачное шифрование данных** (Transparent Data Encryption, TDE) **с возможностью приостановить и возобновить его**.

  Цель – обеспечить механизм для планирования дорогостоящих операций шифрования данных, хранимых в базе данных, чтобы эти операции можно было выполнять в период минимальной нагрузки;

- **управление сертификатами** (Certificate management).

  Упрощенное управление сертификатами с помощью SQL Server, включая сценарии экземпляра отказоустойчивого кластера (Failover Cluster Instance, FCI) и группы постоянной доступности Always On (Always On Availability Group).

Эти улучшения могут показаться вам чем-то весьма небольшим, однако каждая новая функция нацелена на решение важных проблем безопасности, с которыми сталкиваются наши клиенты, и разработана на основыве их отзывов. Например, классификация данных была встроена в SQL Server для соответствия требованиям Генерального регламента о защите персональных данных (General Data Protection Regulation, GDPR), но может использоваться для многих других задач классификации и аудита.

Также очень важно помнить, что SQL Server 2019 поставляется с богатым набором функций безопасности, представленных в SQL Server 2016. В состав этих функций входят:

- функция постоянного шифрования (Always Encrypted);
- ограничение видимости важных данных для непривилегированных пользователей, или динамическая маскировка данных (Dynamic data masking);
- ограничение видимости данных на уровне строк, или защита на уровне строк;
- аппаратное ускорение прозрачного шифрования данных.

Вы можете прочитать обо всех этих функциях безопасности по ссылке https://docs.microsoft.com/en-us/sql/database-engine/whats-new-in-sql-server-2016?view=sql-server-2017#security-enhancements.

Важно иметь в виду, что в течение почти десятилетия SQL Server был наименее уязвимой базой данных и платформой данных, опережая остальные продукты с большим отрывом, согласно результатам мониторинга базы данных уязвимостей (National Vulnerability Database, NVD), поддерживаемой Национальным институтом стандартов и технологий (National Institute of Standards and Technology, NIST). Подробную информацию об этом можно найти на сайте https://nvd.nist.gov/.

Давайте рассмотрим каждую из новых функций безопасности SQL Server 2019 более подробно, начиная со стратегии постоянного шифрования с защищенными областями (Always Encrypted with Secure Enclaves).

Стратегия постоянного шифрования с защищенными областями (Always Encrypted with Secure Enclaves)

До появления версии SQL Server 2016 у SQL Server было несколько способов шифрования данных, в том числе:

- **шифрование соединений** (Encrypting connections) – все данные (данные протокола TDS), которыми обмениваются клиентское приложение и SQL Server, зашифрованы;
- **шифрование данных** (Encrypting data) в таблицах SQL Server с использованием **T-SQL** (иногда называемое шифрованием столбцов данных или ячеек таблиц);
- **прозрачное шифрование данных** (Transparent Data Encryption, TDE) – шифрование хранящихся данных или данные, зашифрованные на уровне файлов базы данных SQL Server.

Ни одно из этих решений не обеспечивает сквозного механизма шифрования. И что еще более важно, администраторы SQL Server контролируют ключи, используемые для шифрования данных. Поэтому не реализуется концепция *разделения обязанностей*. В мире современных требований безопасности владельцы приложений (т. е. владельцы бизнеса) хотят полностью контролировать безопасность своих данных. Они хотят, чтобы такие роли, как администраторы баз данных, управляли инфраструктурой платформы данных, но не имели доступа к бизнес-данным или ключам, используемым для управления доступом к этим данным.

В SQL Server 2016 мы представили функцию постоянного шифрования (Always Encrypted) для решения этих проблем. Постоянное шифрование берет свое начало от проектов в Microsoft Research. Как говорит главный инженер-программист Microsoft, Рагхав Каушик (Raghav Kaushik): «...есть два проекта, которые стоит упомянуть. Одним из них является проект Cipherbase в MSR-Redmond, который пытался построить обработку запросов на зашифрованных данных, вторым – проект Trusted Cloud в MSR-Cambridge, который больше фокусировался на построении архитектурных блоков вокруг безопасного оборудования».

На рис. 3.1 показан пример архитектуры и потока постоянного шифрования данных.

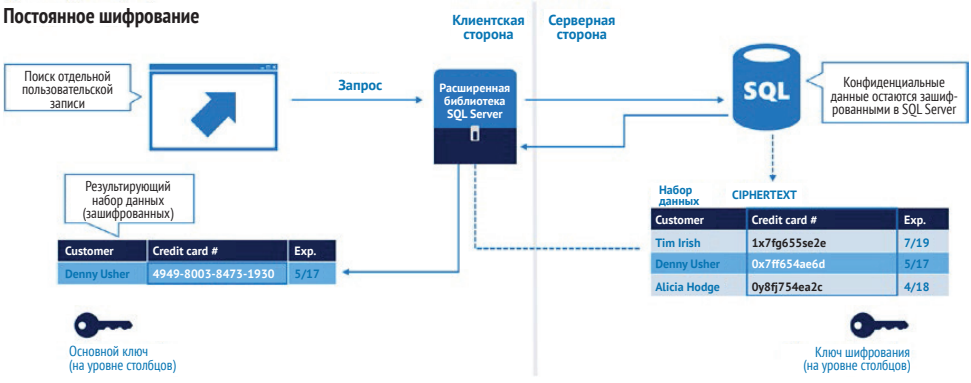


Рис. 3.1. Постоянное шифрование данных в SQL Server 2016

Основная идея этого подхода – клиентское приложение, владельцы которого контролируют жизненный цикл шифрования. Все данные передаются из клиентского приложения в SQL Server в зашифрованном виде, хранятся в SQL Server в зашифрованном виде (на уровне столбцов) и отправляются обратно в клиентское приложение в зашифрованном виде. Только клиентское приложение может расшифровать данные на уровне приложения. Кроме того, ключи, используемые для шифрования и дешифрования данных, на самом деле не хранятся в SQL Server. Место размещения ключей, принадлежащих владельцам приложений, хранится в SQL Server. Но доступ к этим ключам контролируется приложением.

Похоже, это отличное решение, однако в нем есть один недостаток. Поскольку все дешифрование происходит в клиентском приложении, некоторые виды запросов к данным не поддерживаются (например, поддерживаются только операторы равенства WHERE). Кроме того, индексы для данных, зашифрованных с использованием постоянного шифрования, не поддерживаются. Учитывая, что клиентское приложение – единственное место, где происходит дешифрование, не существует способа создать индекс, если постоянное шифрование включено. SQL Server должен будет отправить все данные для столбцов, зашифрованных как часть индекса, в клиентское приложение, чтобы построить индекс, а затем отправить его обратно на сервер. Таким образом, какие бы возможности ни обещала концепция постоянного шифрования, эти рамки делают ее… ну, сильно ограниченной, в которой реализуется всего несколько сценариев.

Существует ли выход из этого положения? Да, и он предлагается в концепции под названием «*Защищенные области*» (*Secure Enclaves*).

Почему используется термин «области»?

Словарь Вебстера определяет область, или анклав (enclave) (www.merriam-webster.com/dictionary/enclave), как «отдельную территориальную, культурную или социальную единицу, заключенную внутри чужой территории». В компьютерном мире это защищенная зона, которая является безопасной и независимой от враждебных захватчиков. Эти захватчики могут быть хакерами, но, к сожалению, они также могут быть системными администраторами или администраторами баз данных.

Intel выпустила концепцию анклава в своем чипсете, известную как Software Guard Extensions (SGX). Вы можете прочитать о ней по ссылке https://software.intel.com/en-us/blogs/2016/06/06/overview-of-intel-software-guard-extension-enclave. SGX поддерживает инструкции в ЦПУ, обеспечивающие возможность создания защищенных областей памяти, которые безопасны для шифрования и обеспечивают защиту от вторжения. Это интересно, но что, если у вас нет чипа SGX? Microsoft предложила решение в виде виртуализированного анклава под названием «области памяти с защитой целостности кода на основе виртуализации» (virtualization-based security, VBS). Вы можете прочитать все подробности о VBS по ссылке www.microsoft.com/security/

blog/2018/06/05/virtualization-based-security-vbs-memory-enclaves-dataprotection-through-isolation/.

Что это означает для концепции постоянного шифрования, и почему это важно?

Совместное использование постоянного шифрования и защищенных областей

Защищенные области (анклавы) предоставляют уникальное решение для «проблемы с индексами» при использовании постоянного шифрования. Данные, передаваемые из клиентского приложения на SQL Server и обратно, по-прежнему полностью зашифрованы. Также зашифрованы данные в памяти SQL Server и на диске. Однако когда необходимо расшифровать данные, например, для построения индекса или поддержки сложных вычислений, дешифрование может происходить в защищенной области на сервере. Эта защищенная область является защищенной областью памяти в пространстве процессов SQL Server. Эта область памяти мала и тесно интегрирована в экспресс-сервисы механизма баз данных с помощью API-интерфейсов защищенных областей (анклавов). Сложные вычисления – это запросы, которые используют такие возможности T-SQL, как запросы диапазонов или поиск по шаблону (т. е. LIKE). Защищенные области предоставляют подобную возможность для решений, использующих постоянное шифрование.

Просмотрите рис. 3.2 на странице документации по SQL Server (https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/alwaysencrypted-enclaves), где показано, как анклавы поддерживают расшифровку данных, при этом не ослабляя их защиту, но также обеспечивают большую гибкость для приложений.

Рис. 3.2. Постоянное шифрование с защищенными областями

Настройка постоянного шифрования никогда не выполнялась начинающими пользователями SQL Server. Это комплексное решение для сложной проблемы. Но это мощная функция, особенно сейчас, когда вы можете использовать ее совместно с защищенными областями.

Для использования постоянного шифрования совместно с защищенными областями потребуется еще один важный компонент, называемый службой *аттестации*. Служба аттестации используется клиентским приложением для проверки того, что защищенная область – анклав, используемый для шифрования, – является доверенным. Для анклавов VBS Windows предоставляет аттестацию вычислительного окружения (runtime attestation), реализуемую **Windows Defender System Guard** (которая использует службу с названием Host Guardian Service (HGS)). Вы можете узнать больше о Windows Defender System Guard по ссылке www.microsoft.com/security/blog/2018/04/19/introducing-windowsdefender-system-guard-runtime-attestation/. Вы также можете прочитать более подробно о том, как приложения взаимодействуют с защищенными областями (анклавами), по ссылке https://docs.microsoft.com/en-us/windows/desktop/api/enclaveapi/nf-enclaveapi-callenclave.

Кроме настройки VBS и службы Host Guardian Service, вам необходимо еще раз погрузиться в технические подробности: ваше приложение должно использовать поставщика, который поддерживает связь с анклавом. Информацию о поддержке анклавов различными провайдерами можно найти по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encryptedenclaves?view=sqlallproducts-allversions#secure-enclave-providers.

На момент написания этой книги SQL Server еще официально не поддерживал аппаратные анклавы, предоставляемые производителями микросхем, такими как Intel SGX. Я ожидаю, что эта поддержка вскоре появится, и вы можете регулярно сверяться с обновлениями документации по постоянному шифрованию для анклавов по ссылке https://docs.microsoft.com/enus/sql/relational-databases/security/encryption/always-encryptedenclaves?view=sqlallproducts-allversions#why-use-always-encrypted-withsecure-enclaves. Сегодня Linux не поддерживает виртуальные анклавы, такие как VBS. Тем не менее я ожидаю, что как только SQL Server обеспечит поддержку аппаратных анклавов, вскоре появится их поддержка и в Linux.

Я не создал практический пример настройки и использования постоянного шифрования и защищенных областей (анклавов). Как я уже говорил ранее в этой главе, это не по силам начинающему пользователю SQL Server. Это функция, реализуемая в масштабах предприятия, и для ее настройки потребуется много времени. Но потраченное на ее настройку время окупится, поскольку эта возможность приносит очень много практической пользы. Якуб Шимашек (Jakub Szymaszek), старший менеджер программы и руководитель, в чьей зоне ответственности находится постоянное шифрование (Always Encrypted), предоставил несколько полезных ссылок, относящихся к этой теме.

Для самостоятельного изучения примера постоянного шифрования с использованием анклава VBS вы можете подключиться к следующему репозиторию GitHub: https://github.com/microsoft/sql-server-samples/tree/master/samples/features/security/always-encrypted-with-secureenclaves. Якуб также провел отличную

презентацию на конференции Microsoft Ignite, из которой можно почерпнуть более подробную информацию о функции постоянного шифрования с использованием защищенных областей (анклавов). Презентация доступна по ссылке https://myignite.techcommunity.microsoft.com/sessions/65357#ignite-html-anchor.

Классификация данных

Готовясь к выпуску версии SQL Server 2017, наша группа безопасности в составе команды разработчиков SQL Server создала специальный набор инструментов в SQL Server Management Studio (SSMS), чтобы помочь клиентам *классифицировать* данные в базе данных SQL Server. Этот набор инструментов включает в себя мастер классификации, набор сценариев T-SQL и отчет. На рис. 3.3 показан мастер классификации данных в SSMS.

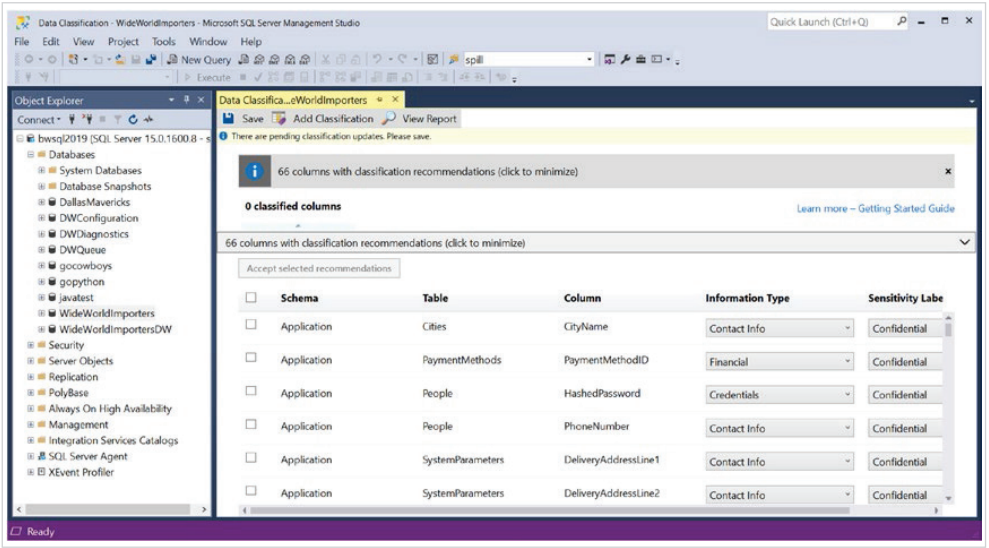


Рис. 3.3. Мастер классификации данных в SSMS

Одним из побуждающих факторов для создания такого инструмента была растущая тенденция в компаниях и регулирующих органах в отношении конфиденциальности данных. И решающим фактором стало то, что Европейский союз готовил Генеральный регламент о защите персональных данных (General Data Protection Regulation, GDPR) (https://eugdpr.org/).

> **Совет.** Генеральный регламент о защите персональных данных (GDPR) вступил в силу в мае 2018 года. Если вы хотите получить полное руководство по использованию SQL Server для удовлетворения требований GDPR в вашей организации, перейдите по ссылке www.microsoft.com/en-us/trustcenter/cloudservices/sql/gdpr.

Идея состояла в том, чтобы проанализировать имена столбцов в вашей базе данных и дать рекомендации о том, как классифицировать столб-

цы с помощью *метки* (label) и *типа информации* (information_type). Тип информации information_type можно использовать для указания типа данных в столбце (например: контактная информация, имя, финансы), а метку можно использовать для классификации *степени конфиденциальности* (sensitivity) данных, хранящихся в этом столбце (конфиденциально, конфиденциально-GDPR, конфиденциально-HIPAA и др.).

Созданный нами инструмент анализировал имена столбцов и искал известные примеры, соответствующие определенным типам информации и определенной степени конфиденциальности. Примером простого соответствия может быть любой столбец с именем, содержащим в нем слово Email. Инструмент предоставлял рекомендации по меткам и типам информации и позволял сохранить их в базе данных. После этого можно было построить отчет для просмотра информации о выполненной классификации.

Предложенный инструмент был неплох, но у него было два ограничения:

- этот инструмент использовал концепцию SQL Server, называемую расширенными свойствами (extended properties).

  Хотя этот подход поддерживается и работает, он не является наиболее эффективным способом хранения метаданных о классификации столбцов, поскольку представляет собой механизм поддержки общих свойств (подробнее о расширенных свойствах можно прочитать по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-addextendedproperty-transact-sql);
- отсутствовал встроенный аудит доступа к столбцам, помеченным для классификации. Аудит является важной частью любой системы классификации и необходим, чтобы удовлетворить требования GDPR.

Поэтому наша команда работала над новым решением для SQL Server 2019 (которое также работает в Azure SQL Database) для встроенной *классификации степени конфиденциальности* (sensitivity classifications). Слово «встроенная» здесь означает новый набор операторов T-SQL, представлений каталога и функций аудита.

Ниже приведен список операторов T-SQL, поддерживаемых в SQL Server 2019 для классификации:

- **ADD SENSITIVITY CLASSIFICATION** (https://docs.microsoft.com/en-us/sql/t-sql/statements/add-sensitivity-classification-transact-sql);
- **DROP SENSITIVITY CLASSIFICATION** (https://docs.microsoft.com/en-us/sql/t-sql/statements/drop-sensitivity-classification-transact-sql).

Эти операторы T-SQL позволяют сохранять метаданные непосредственно в системных таблицах (предоставляемых представлениями каталога), которые относятся к меткам и типам информации (information_types), связанным со столбцами в таблице.

Для просмотра этих метаданных используется новое представление каталога, названное **sys.sensitivity_classification** (https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-sensitivity-classifications-transact-sql).

Кроме того, службы аудита SQL Server Auditing теперь поддерживают новое свойство, называемое **data_sensitivity_information**, которое можно использовать для аудита доступа к секретным данным. Эта функция позволяет узнать, кто и когда пытался просматривать секретные данные, а также к каким именно данным осуществлялся доступ.

Эти новые возможности были интегрированы в мастере SSMS, чтобы соответствующие операторы T-SQL можно было использовать, работая с базой данных в SQL Server 2019. Теперь SQL Server предоставляет возможность встроенной классификации и аудита с помощью специального инструмента в SSMS и поддерживаемых операторов T-SQL.

> **Примечание.** Если вы использовали мастер с SSMS 17.0 или даже 18.0 в предыдущих версиях SQL Server (до SQL Server 2019) и восстановили эту базу данных в SQL Server 2019, расширенные свойства классификации данных будут перенесены в новые метаданные классификации степени конфиденциальности.

Давайте рассмотрим пример использования инструмента классификации данных SSMS, нового синтаксиса T-SQL, представлений каталога и функций аудита.

Подготовительные шаги для использования примеров, иллюстрирующих классификацию данных

Вначале вам нужно выполнить ряд настроек, чтобы использовать примеры, приведенные в этом разделе. В данной главе вы будете использовать тестовую базу данных **WideWorldImporters** (вы можете прочитать больше об этой базе данных и ее схеме по ссылке https://docs.microsoft.com/en-us/sql/samples/wide-world-importers-oltp-database-catalog). Если вы уже восстановили эту базу данных, выполняя примеры из главы 2, то можете просто продолжать использовать эту базу данных.

Эти примеры будут работать для версии SQL Server 2019, установленной на Windows, в Linux и в контейнерах.

Вам также понадобится SQL Server Management Studio (SSMS) версии 18.2 или выше, чтобы выполнить все шаги из этого примера. Некоторые шаги можно выполнить, используя предоставленные сценарии T-SQL, но несколько примеров основаны на использовании инструментов, встроенных в SSMS. Вы можете загрузить последнюю версию SSMS, перейдя по ссылке https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms.

Все сценарии, приведенные в этой главе, можно найти в репозитории GitHub, созданном для этой книги, в каталоге **ch3_new_security_capabilities\dataclassification**.

Чтобы использовать примеры из этой главы, вам необходимо выполнить следующие шаги (пропустите эти шаги, если вы уже восстановили базу данных из главы 2):

1. Загрузите резервную копию базы данных WideWorldImporters, доступную по ссылке https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImporters-Full.bak.
2. Восстановите эту базу данных на своем экземпляре SQL Server 2019. Вы можете использовать для этого готовый сценарий **restorewwi.sql**. Вероятно, вам придется изменить пути к каталогу, где размещена ваша резервная копия, и к месту восстановления файлов базы данных.

Использование классификации данных

Чтобы посмотреть на классификацию данных в действии, выполните следующие шаги. В следующем разделе вы узнаете, как настроить аудит для отслеживания действий пользователей, которые пытаются просматривать столбцы таблицы, классифицированные по степени конфиденциальности.

1. Вы могли выполнять эти примеры более одного раза, поэтому запустите сценарий **setup_classification.sql**:

```
-- Шаг 1. Если вы запускали эти демонстрации прежде, чем удалили
-- существующую классификацию данных
USE WideWorldImporters
GO
IF EXISTS (SELECT * FROM sys.sensitivity_classifications sc
WHERE object_id('[Application].[PaymentMethods]') = sc.major_id)
BEGIN
  DROP SENSITIVITY CLASSIFICATION FROM [Application].
  [PaymentMethods].[PaymentMethodName]
END
GO
IF EXISTS (SELECT * FROM sys.sensitivity_classifications sc
WHERE object_id('[Application].[People]') = sc.major_id)
BEGIN
  DROP SENSITIVITY CLASSIFICATION FROM [Application].
  [People].[FullName]
  DROP SENSITIVITY CLASSIFICATION FROM [Application].
  [People].[EmailAddress]
END
GO
```

2. Теперь используйте инструмент классификации данных SSMS, чтобы классифицировать данные двух столбцов в базе данных WideWorldImporters. Запустите SSMS и найдите базу данных WideWorldImporters в обозревателе объектов. Щелкните правой кнопкой мыши и выберите **Tasks → Data Discovery and Classification → Classify Data** (Задачи → Обнаружение и классификация данных → Классификация данных), как показано на рис. 3.4.

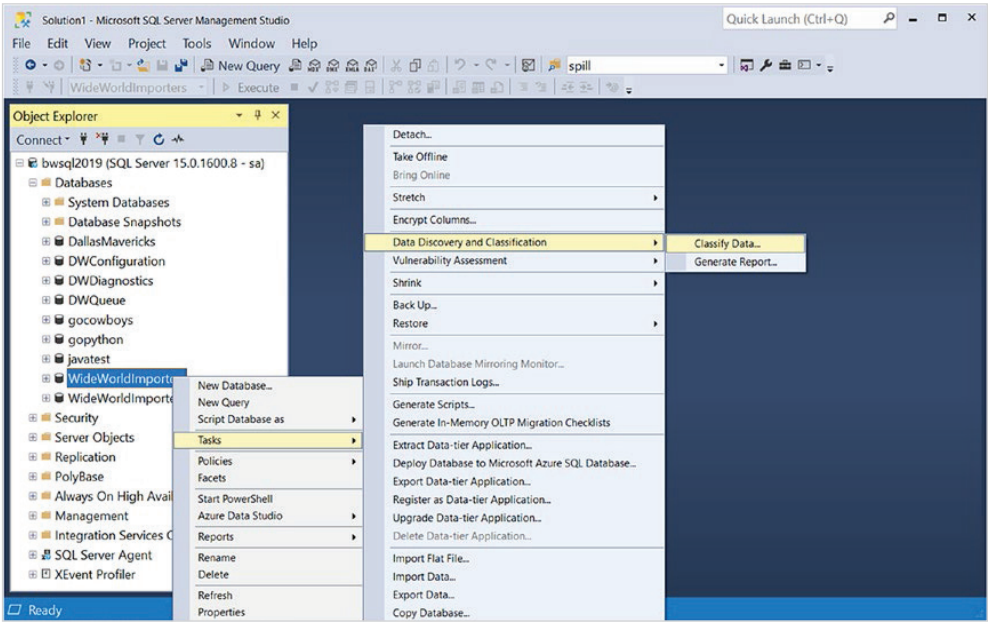


Рис. 3.4. Запуск инструмента классификации данных

3. Инструмент классификации анализирует имена столбцов для объектов в базе данных WideWorldImporters и создает рекомендации относительно того, какие столбцы классифицировать, а также какие метки и тип информации (information_type) использовать. Если вы запускаете инструмент классификации на базе данных WideWorldImporters, в результате должно получиться 66 столбцов с рекомендациями. Щелкните мышью эти рекомендации, чтобы увидеть результат, как показано на рис. 3.5.
4. Теперь вы увидите список столбцов с предлагаемыми вариантами меток, относящихся к типу информации (information_type) и степени конфиденциальности (sensitivity). Значения, используемые для этих рекомендаций, являются внутренней «механикой» этого инструмента, и поэтому их нельзя настроить. Однако, вооружившись T-SQL, я покажу вам, как «использовать свою собственную систему классификации». Сохраните некоторые из этих рекомендаций, проверив столбцы PaymentMethodName и FullName, затем нажмите

Accept selected recommendations (Принять выбранные рекомендации). Перед тем как вы щелкнете «Принять», ваш экран должен выглядеть примерно так, как показано на рис. 3.6.

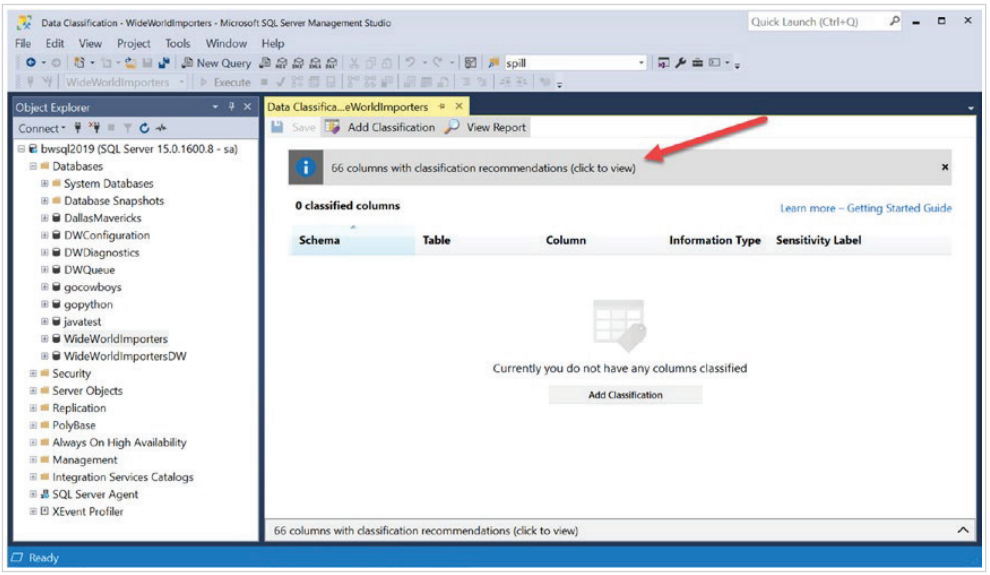


Рис. 3.5. Рекомендации по классификации данных в SSMS

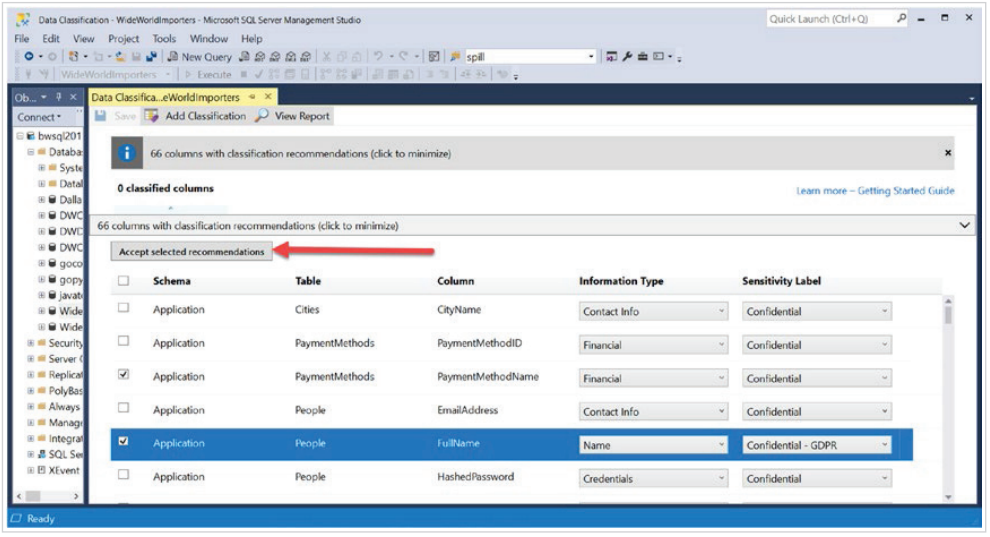


Рис. 3.6. Примите рекомендации по классификации

Обратите внимание, что для PaymentMethodName существует рекомендация Financial and Confidential (Финансовая и конфиденциальная информация): если вы выполните запрос к этой табли-

це, то увидите значения Cash, Check, Credit-Card и EFT (Наличные, Чек, Кредитная карта и Электронный платеж). Для FullName выдана рекомендация Name и Confidential-GDPR (Имя и Конфиденциально-GDPR).

Примечание. Данный инструмент не гарантирует соответствия GDPR и даже не рассматривает элементы GDPR. Это просто рекомендации, основанные на наших знаниях о GDPR. Если вам необходимо использовать вашу систему таким образом, чтобы она для удовлетворяла требованиям GDPR, обязательно следуйте политикам и процедурам вашей компании.

5. После того как вы нажмете «Принять», инструмент покажет вам, какие столбцы были выбраны, и позволит сохранить выбор. Значки «мусорной корзины» позволяют вам удалить выбор и затем выбрать новые столбцы. Обратите внимание, что количество рекомендуемых столбцов было уменьшено на 2. На данный момент выберите **Save** (Сохранить), как показано на рис. 3.7.

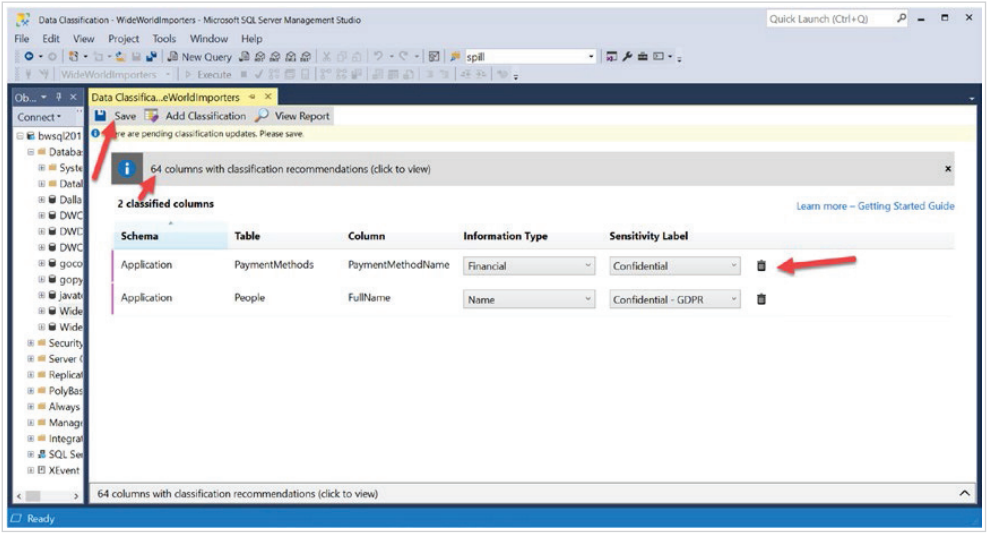


Рис. 3.7. Сохранение принятых рекомендаций

6. После сохранения принятых рекомендаций вы можете выбрать вариант **View Report** (Просмотреть отчет), чтобы просмотреть визуальные данные о классификации данных, сохраненные в вашей базе данных. Для отчета создается новая вкладка в SSMS. Обязательно нажмите значок «+» рядом со схемой **Application** (Приложение), чтобы увидеть все столбцы, для которых выполнена классификация. Отчет должен выглядеть так, как показано на рис. 3.8.

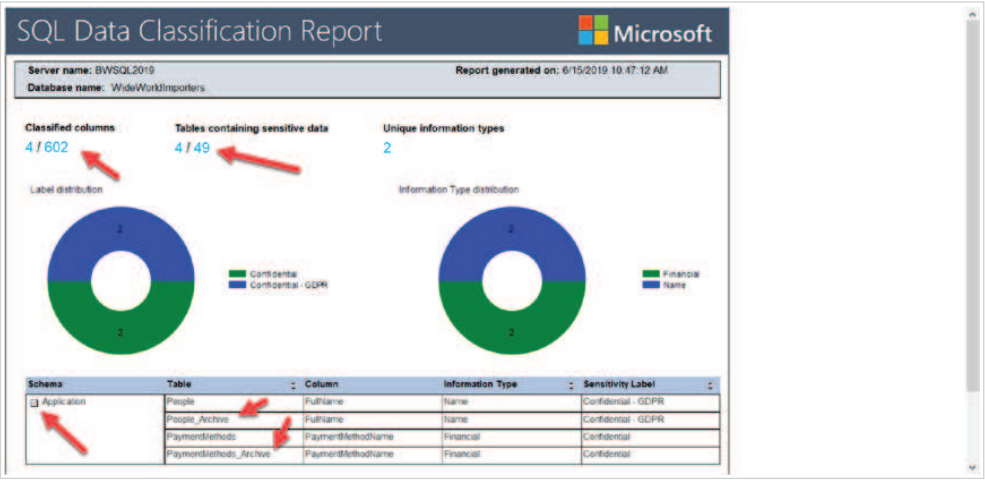


Рис. 3.8. Отчет о классификации данных

Отчет выполняет запрос к представлению каталога sys.sensitivity_classification, а также к другим метаданным в базе данных. Отчет показывает, сколько столбцов и таблиц из всех существующих столбцов и таблиц было помечено метками. В отчете также показано распределение значений label и information_type в базе данных. Обратите внимание, что в нижней части отчета в списке столбцов отображаются таблицы People_Archive и PaymentMethods_Archive. Почему? Это связано с тем, что для этих таблиц создаются временные таблицы с системными версиями. Временные таблицы, представленные в SQL Server 2016, предоставляют информацию об изменениях таблицы в базе данных на определенный момент времени (подробнее о временных таблицах можно прочитать по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables).

Поскольку мы приняли рекомендации для столбцов в таблице, которая имеет временную таблицу, мы хотим быть уверены, что также классифицируем столбцы в «скрытой» архивной таблице. У вас нет прямого доступа к этим столбцам, но SQL Server сохраняет архивную таблицу. Таким образом, любой доступ к временным данным также можно проверить.

Примечание. Нельзя добавлять классификацию по степени конфиденциальности непосредственно в архивные таблицы из временных данных. При удалении классификации по степени конфиденциальности для столбца вместе с ней удаляется классификация для архивной таблицы.

7. Если вы снова вернетесь на вкладку с сохраненными рекомендациями, в панели увидите группу элементов **Add Classification** (Добавить классификацию). Это способ вручную добавить классифика-

цию по степени конфиденциальности, чтобы либо переопределить рекомендации, либо выбрать столбец, для которого не были выданы рекомендации. Вы по-прежнему получаете варианты, предоставляемые инструментом: метки и тип информации (information_type). Если вы щелкнете **Add Classification**, то увидите на своем экране картину, похожую на рис. 3.9.

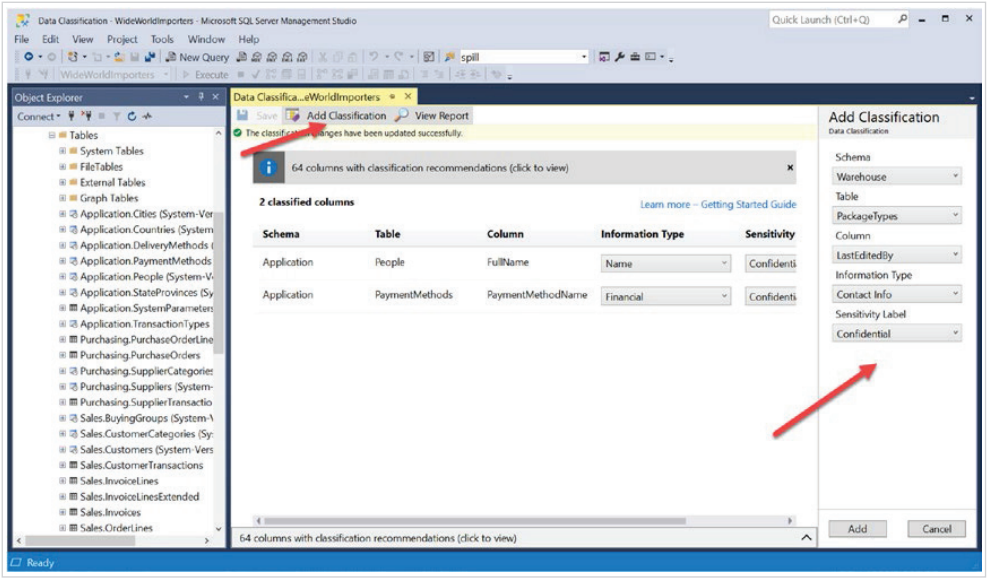


Рис. 3.9. Добавление классификации вручную

8. Инструмент, которым мы воспользовались при демонстрации, – отличный инструмент; но вы также можете использовать T-SQL, чтобы добавить классификацию и использовать свою собственную систему классификации данных. Сначала выполните сценарий **findclassification.sql**; он иллюстрирует использование T-SQL для просмотра всех существующих классификаций.

```
USE WideWorldImporters
GO
SELECT o.name as table_name, c.name as column_name, sc.information_
type, sc.label
FROM sys.sensitivity_classifications sc
JOIN sys.objects o
ON o.object_id = sc.major_id
JOIN sys.columns c
ON c.column_id = sc.minor_id
AND c.object_id = sc.major_id
ORDER BY sc.information_type, sc.label
GO
```

Ваши результаты должны быть аналогичны данным из полученного ранее отчета.

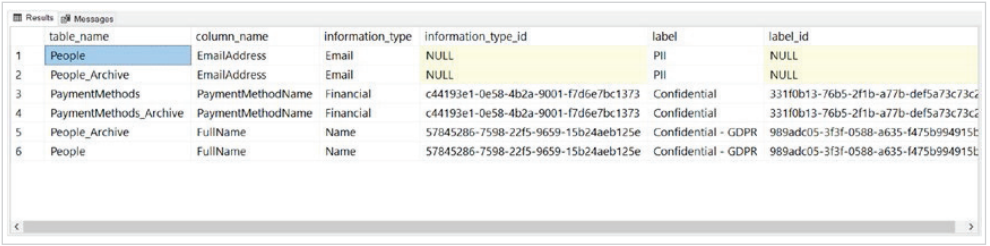
9. Чтобы добавить собственную классификацию при помощи T-SQL, используйте сценарий **addclassification.sql**. Выполните каждый шаг сценария, чтобы добавить классификацию и увидеть новые результаты. Вы можете ввести любые значения label и information_type, какие только захотите, для своих целей. Для этого примера я выбрал обозначения, отличающиеся от тех, которые используются в рассматриваемом инструменте. Так как в данном примере разбирается адрес электронной почты, я назвал свой тип данных **Email** и ввел метку **PII**, что означает личную информацию (Personally Identifiable Information). По сути, это просто строковые значения, связанные со столбцами, которые мы храним. Но, как и любая система, которая строится и проектируется, хорошая система классификации будет иметь некоторую структуру, определяющую, какие метки label и information_type следует использовать для всей компании в целом и для базы данных, и эти метаданные будут отображаться в отчетах аудита, как мы вскоре увидим.

```
-- Шаг 1. Добавим классификацию
ADD SENSITIVITY CLASSIFICATION TO
[Application].[People].[EmailAddress]
WITH (LABEL='PII', INFORMATION_TYPE='Email')
GO
-- Шаг 2. Просмотрим все имеющиеся классификации
USE WideWorldImporters
GO
SELECT o.name as table_name, c.name as column_name,
sc.information_type, sc.information_type_id, sc.label, sc.label_id
FROM sys.sensitivity_classifications sc
JOIN sys.objects o
ON o.object_id = sc.major_id
JOIN sys.columns c
ON c.column_id = sc.minor_id
AND c.object_id = sc.major_id
ORDER BY sc.information_type, sc.label
GO
```

Ваши результаты должны выглядеть приблизительно так же, как на рис. 3.10.

Просматривая эти результаты, обратите внимание, что столбцы, добавленные инструментом, имеют значения в столбцах **information_type_id** и **label_id**. Оператор T-SQL ADD SENSITIVITY CLASSIFICATION поддерживает значение GUID, размечающий строки, указывая мет-

ки и типы информации. Это может быть особенно ценно, если ваша компания создает систему классификации для хранения всех принятых результатов классификации, т. е. присвоенных меток и типов. Теперь вы можете ссылаться на любую метку или тип через значение GUID (при этом можете сгенерировать значения GUID самостоятельно).



Results Messages

	table_name	column_name	information_type	information_type_id	label	label_id
1	People	EmailAddress	Email	NULL	PII	NULL
2	People_Archive	EmailAddress	Email	NULL	PII	NULL
3	PaymentMethods	PaymentMethodName	Financial	c44193e1-0e58-4b2a-9001-f7d6e7bc1373	Confidential	331f0b13-76b5-2f1b-a77b-def5a73c73c2
4	PaymentMethods_Archive	PaymentMethodName	Financial	c44193e1-0e58-4b2a-9001-f7d6e7bc1373	Confidential	331f0b13-76b5-2f1b-a77b-def5a73c73c2
5	People_Archive	FullName	Name	57845286-7598-22f5-9659-15b24aeb125e	Confidential - GDPR	989adc05-3f3f-0588-a635-f475b994915b
6	People	FullName	Name	57845286-7598-22f5-9659-15b24aeb125e	Confidential - GDPR	989adc05-3f3f-0588-a635-f475b994915b

Рис. 3.10. Классификация, выполненная с помощью используемого инструмента и с помощью T-SQL

Совет. Функцию T-SQL NEWID() можно использовать для генерации уникальных значений GUID в SQL Server. Вы можете найти более подробную информацию по ссылке https://docs.microsoft.com/en-us/sql/t-sql/functions/newid-transact-sql.

Итак, кажется, это довольно простая и понятная система. Но вначале она ограничена метками и значениями data_type, которые вы решите использовать. Преимущество поддержки T-SQL заключается в том, что любое приложение, поддерживающее T-SQL, теперь может построить систему классификации и выполнять запросы к ней, поскольку в T-SQL также поддерживаются представления каталога.

А как насчет аудита? Давайте перейдем к следующему разделу, чтобы посмотреть, как он работает. Оставьте вашу базу как есть, чтобы выяснить, как работает аудит, используя результаты выполненных вами шагов из предыдущих примеров.

Аудит и классификация данных

Наличие метаданных классификации степени конфиденциальности для столбцов таблиц базы данных весьма ценно само по себе, но еще более нужной возможностью для аудита будет автоматическое отслеживание событий просмотра пользователями столбцов, помеченных как конфиденциальная информация.

Современные версии SQL Server включают в себя встроенную функцию **SQL Server Audit**. Основанная на использовании расширенных событий (Extended Events), функция **SQL Server Audit** имеет много различных возможностей и предоставляет расширенную систему аудита. Дополнительную информацию обо всех функциональных возможностях аудита SQL

Server вы можете найти по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/security/auditing/sql-server-audit-database-engine.

Аудит проводится в форме записи для каждого события аудита, куда включаются все типы свойств. SQL Server 2019 добавляет новое свойство события аудита под названием *data_sensitivity_information*. Так, например, если вы включаете аудит выполнения операторов SELECT для определенных таблиц, в которые вы добавили классификацию степени конфиденциальности для столбцов, то если эти столбцы являются частью списка оператора выборки данных SELECT, в столбце *data_sensitivity_information* будут отображаться эти события доступа.

Давайте посмотрим на предыдущем примере, как аудит работает с классификацией степени конфиденциальности.

1. Поскольку вы могли выполнять сценарии, приведенные в этих примерах, более одного раза и вам не хотелось бы восстанавливать базу данных, для того чтобы выполнить аудит «с чистого листа», сначала запустите сценарий **dropqlaudit.sql**.

```
-- Шаг 1. Отключите аудит и удалите все предыдущие данные.
USE WideWorldImporters
GO
IF EXISTS (SELECT * FROM sys.database_audit_specifications
WHERE name = 'People_Audit')
BEGIN
  ALTER DATABASE AUDIT SPECIFICATION People_Audit
  WITH (STATE = OFF)
  DROP DATABASE AUDIT SPECIFICATION People_Audit
END
GO
USE master
GO
IF EXISTS (SELECT * FROM sys.server_audits WHERE name = 'GDPR_Audit')
BEGIN
  ALTER SERVER AUDIT GDPR_Audit
  WITH (STATE = OFF);
  DROP SERVER AUDIT GDPR_Audit
END
GO
-- Шаг 2. Удалите файлы .audit из каталога, выбранного по умолчанию,
-- или из выбранного вами при установке каталога.
-- del C:\program files\microsoft sql server\mssql15.mssqlserver\
mssql\data\GDPR*.audit
```

Обратите внимание на то, что шаг 2 сценария содержит комментарий, описывающий действия, которые необходимо выполнить для удаления файлов. Когда вы запускаете аудит, SQL Server создает фай-

лы аудита в каталоге, размещенном по указанному вами пути. Когда вы отключаете аудит и удаляете его результаты, созданные файлы остаются в этом каталоге. Во избежание ошибок при выполнении действий в приведенном примере удалите вручную все файлы, оставшиеся после выполнения предыдущих шагов.

2. Откройте сценарий **setupsqlaudit.sql**, чтобы создать новую спецификацию и запустить работу аудита. Я не буду вдаваться в подробности работы аудита. Из текста представленных здесь запросов вы можете видеть, что аудит настроен для отслеживания выполнения операторов SELECT в таблице [Application].[People] в базе данных WideWorldImporters. Чтобы узнать об аудите больше, ознакомьтесь с документацией по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/security/auditing/sql-server-audit-database-engine.

```
USE master
GO
-- Создание журнала аудита сервера
CREATE SERVER AUDIT GDPR_Audit
  TO FILE (FILEPATH = 'C:\program files\microsoft sql server\mssql15.
mssqlserver\mssql\data')
GO
-- Включение аудита сервера
ALTER SERVER AUDIT GDPR_Audit
WITH (STATE = ON)
GO
USE WideWorldImporters
GO
-- Создание спецификации аудита базы данных.
CREATE DATABASE AUDIT SPECIFICATION People_Audit
FOR SERVER AUDIT GDPR_Audit
ADD (SELECT ON Application.People BY public )
WITH (STATE = ON)
GO
```

3. Теперь давайте запустим несколько запросов и посмотрим, что именно проверяется. Откройте сценарий **findpeople.sql** и выполните шаги 1 и 2, руководствуясь комментариями в сценарии:

```
-- Шаг 1. Выполните сканирование таблицы и посмотрите, были ли проверены
-- столбцы, классифицированные как конфиденциальная информация
USE WideWorldImporters
GO
SELECT * FROM [Application].[People]
GO
```

```
-- Шаг 2. Проверьте результаты аудита
-- Результаты аудита могут быть доступны НЕ СРАЗУ после запуска запроса,
-- а через несколько секунд.
SELECT event_time, session_id, server_principal_name,
database_name, object_name,
cast(data_sensitivity_information as XML) as data_sensitivity_information,
client_ip, application_name
FROM sys.fn_get_audit_file ('C:\program files\microsoft sql server\
mssql15.mssqlserver\mssql\data\*.sqlaudit',default,default)
GO
```

Ваши результаты должны выглядеть приблизительно так же, как на рис. 3.11.

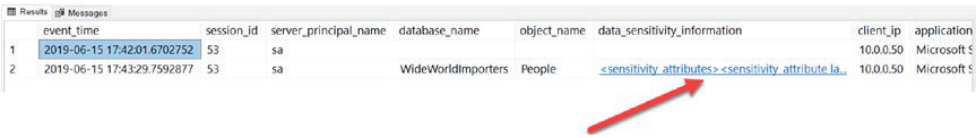


Рис. 3.11. Аудит одного просмотра таблицы с классификацированными данными

В этом примере вы запустили запрос, выбирающий все столбцы из таблицы People. Функция T-SQL **fn_get_audit_file** используется для получения результатов аудита в формате строки/столбца. Я взял только определенные столбцы из набора результатов этой функции. Вы можете просмотреть полный список аргументов и выходных столбцов для этой функции по адресу https://docs.microsoft.com/en-us/sql/relational-databases/system-functions/sys-fn-get-audit-file-transact-sql.

Первая строка – это запись о начале аудита. Вторая строка – это проверка оператора SELECT. Обратите внимание, что значение в столбце data_sensitivity_information имеет тип данных XML. Щелкните значение в этом столбце, и в SSMS откроется новое окно с полными данными XML. Ваши результаты должны выглядеть так, как показано на рис. 3.12.

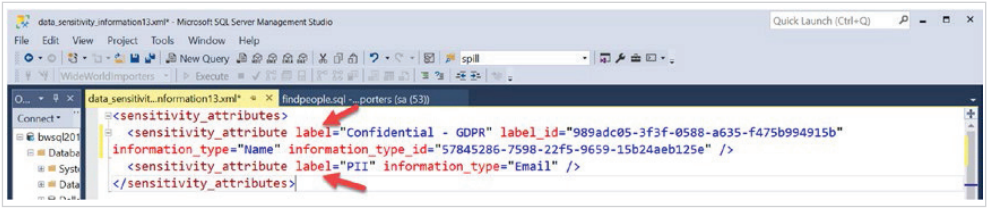


Рис. 3.12. Степень конфиденциальности для различных данных

Данные, представленные в виде XML, включают в себя атрибут для любой уникальной метки и тип информации information_type, к ко-

торому выполняется запрос с использованием оператора SELECT. Теперь вы можете посмотреть, какие столбцы указаны в этих данных, используя представление каталога **sys.sensitivity_classification**.

4. Выполните шаги 3 и 4 сценария **findpeople.sql**:

```
-- Шаг 3. Что, если я попробую обратиться только к одному
-- из столбцов данных напрямую?
SELECT FullName FROM [Application].[People]
GO
-- Шаг 4. Проверьте результаты аудита
-- Результаты аудита могут быть доступны НЕ СРАЗУ после запуска
-- запроса, а через несколько секунд.
SELECT event_time, session_id, server_principal_name,
database_name, object_name,
cast(data_sensitivity_information as XML) as data_sensitivity_information,
client_ip, application_name
FROM sys.fn_get_audit_file ('C:\program files\microsoft sql
server\mssql15.mssqlserver\mssql\data\*.sqlaudit',default,default)
GO
```

Результаты, полученные после выполнения шага 4, должны выглядеть так, как показано на рис. 3.13.

Results Messages

	event_time	session_id	server_principal_name	database_name	object_name	data_sensitivity_information	client_ip	application
1	2019-06-15 17:42:01.6702752	53	sa				10.0.0.50	Microsoft S
2	2019-06-15 17:43:29.7592877	53	sa	WideWorldImporters	People	<sensitivity_attributes><sensitivity_attribute la...	10.0.0.50	Microsoft S
3	2019-06-15 17:53:45.6213307	53	sa	WideWorldImporters	People	<sensitivity_attributes><sensitivity_attribute la...	10.0.0.50	Microsoft S

Рис. 3.13. Результаты аудита, включающие SELECT для одного столбца, данные в котором классифицированы как конфиденциальные

В результатах аудита присутствует третья строка (вы получите по одной строке для каждого оператора SELECT). Если щелкнуть столбец data_sensitivity_information, вы увидите только одну метку, поскольку запрос выполнялся лишь к столбцу FullName.

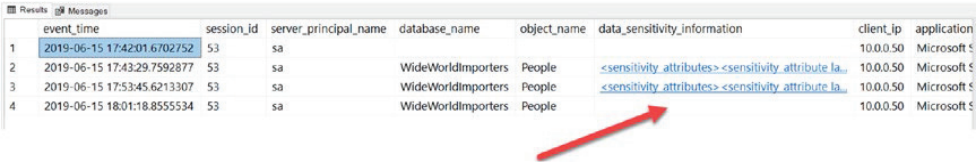
5. Аудит будет отслеживать доступ к данным, классифицированным как конфиденциальные, только в том случае, если классифицированный столбец является частью списка столбцов в операторе SELECT или содержится в выводимых результатах запроса.

   Покажем это. Запустите шаги 5 и 6 сценария **findpeople.sql**:

```
-- Шаг 5. Что, если я ссылаюсь на классифицированный столбец только
-- в выражении WHERE?
SELECT PreferredName FROM [Application].[People]
WHERE EmailAddress LIKE '%microsoft%'
```

```
GO
-- Шаг 6. Проверьте результаты аудита
-- Результаты аудита могут быть доступны НЕ СРАЗУ после запуска
-- запроса, а через несколько секунд.
SELECT event_time, session_id, server_principal_name,
database_name, object_name,
cast(data_sensitivity_information as XML) as data_sensitivity_information,
client_ip, application_name
FROM sys.fn_get_audit_file ('C:\program files\microsoft sql server\
mssql15.mssqlserver\mssql\data\*.sqlaudit',default,default)
GO
```

Результаты, полученные после выполнения шага 6, должны выглядеть так, как показано на рис. 3.14.



	event_time	session_id	server_principal_name	database_name	object_name	data_sensitivity_information	client_ip	application
1	2019-06-15 17:42:01.6702752	53	sa				10.0.0.50	Microsoft S
2	2019-06-15 17:43:29.7592877	53	sa	WideWorldImporters	People	<sensitivity_attributes><sensitivity_attribute la...	10.0.0.50	Microsoft S
3	2019-06-15 17:53:45.6213307	53	sa	WideWorldImporters	People	<sensitivity_attributes><sensitivity_attribute la...	10.0.0.50	Microsoft S
4	2019-06-15 18:01:18.8555534	53	sa	WideWorldImporters	People		10.0.0.50	Microsoft S

Рис. 3.14. Результаты аудита для классифицированного столбца в условии WHERE

В данном примере запрос выдает результаты для столбца PreferredName, используя в качестве критерия столбец EmailAddress. PreferredName не является классифицированным столбцом, а EmailAddress – является. Но поскольку EmailAddress не является частью списка SELECT, столбец data_sensitivity_information в результатах аудита не заполняется.

Классификация данных – это простая, но очень мощная новая возможность в SQL Server 2019, которую вы можете использовать для обеспечения безопасности данных и выполнения требований любых регуляторных политик в вашей организации. Эта функция работает как в SQL Server 2019, так и в Azure SQL Database. Ознакомьтесь с полным руководством по защите информации, написанным нашей командой, перейдя по ссылке https://docs.microsoft.com/en-us/azure/sql-database/sql-database-data-discovery-and-classification.

Другие новые функции безопасности

В SQL Server 2019 есть еще несколько не самых значительных, но важных новых функций безопасности, включая приостановку и возобновление прозрачного шифрования данных, а также улучшенное управление сертификатами шифрования для SQL Server.

Приостановка и возобновление TDE

Прозрачное шифрование данных (Transparent Data Encryption, TDE) – это шифрование хранящихся данных. Оно позволяет шифровать базу данных SQL Server и файлы журналов независимо от ядра базы данных SQL Server. Таким образом, если кто-то попытается получить доступ к вашей базе данных и/или файлам журнала транзакций, данные в этих файлах будут зашифрованы. Функция TDE была включена в несколько версий SQL Server; о том, как ее использовать, можно прочитать по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption.

Когда вы включаете TDE для существующей базы данных, SQL Server должен прочитать каждую страницу базы данных с диска в буферный пул и записать ее обратно в зашифрованный файл базы данных. Шифрование происходит в фоновом режиме, поэтому оно не влияет непосредственно на производительность системы для пользователей; однако при этом может выполняться интенсивное чтение и запись всех страниц базы данных и потребляться ресурсы процессора и подсистемы ввода-вывода. Для базы данных очень большого объема это может повлиять на работу критически важных приложений.

SQL Server 2019 представляет концепцию *приостановки* и *возобновления* шифрования TDE. Теперь вы можете включить TDE для базы данных, но затем приостановить шифрование в любой момент и возобновить шифрование с последнего момента, когда оно было приостановлено. Это позволяет эффективно планировать полное шифрование базы данных с использованием TDE в соответствии с потребностями бизнес-приложений.

Приостановить TDE просто. Для этого можно выполнить следующую команду T-SQL:

```
ALTER DATABASE <db_name> SET ENCRYPTION SUSPEND
```

Возобновить процесс шифрования с того места, где он был приостановлен, можно с помощью следующей команды T-SQL:

```
ALTER DATABASE <db_name> SET ENCRYPTION RESUME
```

Для облегчения диагностики этой новой функции в DMV sys.dm_database_encryption_keys добавлено три новых столбца, где содержатся сведения о состоянии сканирования данных TDE:

- **encryption_scan_state** – численное значение, показывающее, выполняется ли сканирование данных TDE, приостановлено или завершено;
- **encryption_scan_state_desc** – строковое описание состояния сканирования: RUNNING (выполняется), SUSPENDED (приостановлено), COMPLETE (завершено);

- **encryption_scan_modify_date** – дата/время последнего изменения состояния сканирования данных TDE.

Это небольшое по объему, но важное усовершенствование использования TDE выполнено для работы с очень большими базами данных SQL Server. Вы можете прочитать больше о приостановке и возобновлении TDE по адресу https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption.

Управление сертификатами безопасности

Допустим, вы хотите зашифровать соединения с SQL Server – это обычная практика для обеспечения шифрования протокола табличного потока данных (Tabular Data Stream, TDS) между клиентскими приложениями и SQL Server. Когда вы используете шифрование с помощью протокола, например TLS, вам необходимы сертификаты шифрования. Версия SQL Server для Windows предоставляет возможность использования сертификатов при помощи знакомого всем приложения SQL Server Configuration Manager. Однако сначала необходимо выполнить всю работу по установке сертификата на сервере или даже на нескольких серверах для экземпляра отказоустойчивого кластера (Failover Cluster Instance, FCI) или группы доступности Always On (Always On Availability Group).

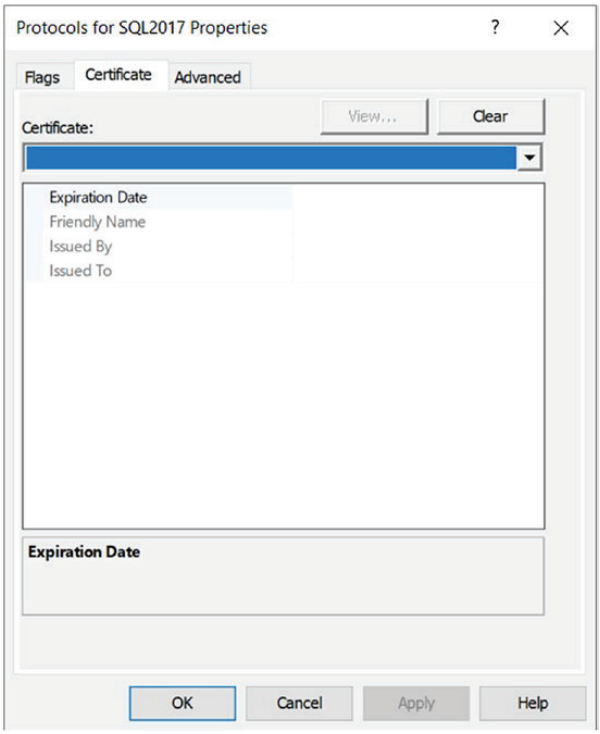


Рис. 3.15. Управление сертификатами безопасности в SQL Server 2017

На рис. 3.15 показано диалоговое окно SQL Server Configuration Manager, где предлагается выбрать установленный сертификат безопасности для использования с SQL Server 2017.

SQL Server 2019 теперь включает возможность через диспетчер конфигурации SQL Server импортировать сертификат безопасности, и даже более того – передать сертификаты между узлами экземпляра отказоустойчивого кластера (Failover Cluster Instance) или группы доступности (Availability Group); причем вы можете выполнять эти действия, используя основной экземпляр SQL Server.

На рис. 3.16 показано диалоговое окно SQL Server Configuration Manager для SQL Server 2019.

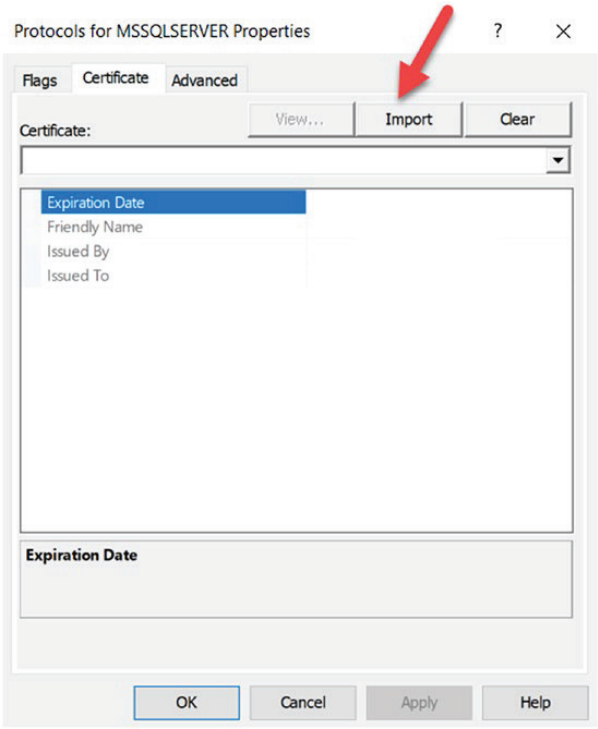


Рис. 3.16. SQL Server Configuration Manager для SQL Server 2019

Обратите внимание на новую кнопку **Import** (Импорт) в этом диалоговом окне. Документацию, в которой рассказывается, как использовать эту возможность на одном сервере или на кластере, вы можете найти по ссылке https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/manage-certificates.

Резюме

Используя в качестве основы богатые возможности SQL Server 2016, такие как постоянное шифрование, ограничение видимости на уровне строк и

динамическая маскировка данных, SQL Server 2019 представляет новые функции безопасности: защищенные области (Secure Enclaves), классификацию степени конфиденциальности данных, приостановку и возобновление TDE и упрощенное управление сертификатами безопасности. Все это в сочетании с функциями безопасности, унаследованными от предыдущих версий SQL Server, предоставляет платформу для обеспечения безопасности, надежности, совместимости и управления вашими данными.

Глава 4

Непрерывная доступность, соответствующая требованиям для систем, критичных к сбоям

В предыдущих двух главах вы узнали о новых возможностях SQL Server 2019, предназначенных для решения современных задач, связанных с производительностью и безопасностью. Для многих корпоративных клиентов существует третья задача – обеспечить, чтобы платформа базы данных отвечала высоким требованиям современных приложений и бизнеса к ее доступности.

SQL Server обеспечивает высокую доступность по умолчанию, поскольку почти все, что вы делаете с помощью SQL, может быть сделано онлайн. SQL Server 2019 повышает доступность данных в интернете с помощью следующих новых функций, предназначенных для решения новых задач:

- **усовершенствованный онлайн-индекс**.

  Пользователи всегда хотят всего и сразу. Они хотят, чтобы администраторы поддерживали индексы и обновляли их, но в то же время им нужен постоянный доступ к их данным в круглосуточном режиме. SQL Server 2019 расширяет возможности онлайн-индексов предыдущих версий благодаря **возобновляемому построению онлайн-индексов** (resumable online index creation) и **онлайн-поддержке кластеризованных столбцовых индексов** (online clustered columnstore index maintenance);
- **улучшенная группа доступности (Availability Group)**.

  Флагманская функция HADR в SQL Server, группа доступности Always On (Always On Availability Group), продолжает совершенствоваться с каждой новой версией SQL Server. В каждой новой версии она содержит больше реплик, а также улучшается возможность перенаправления соединений с приложениями;

- **ускоренное восстановление базы данных**.

  Читатели этой книги могут столкнуться с ситуацией, когда кто-то пытается прервать длительную транзакцию и, разочарованный несколькими неудачными попытками, перезапускает SQL Server. А потом они приходят к вам и еще больше разочаровываются, когда оказывается, что восстановление базы данных занимает очень много времени.

Почему SQL Server не восстановил базу данных сразу? Вы объясняете: потому что восстановление должно откатить транзакцию, которую вы прервали; в противном случае база данных не будет согласованной. Представьте, что ничего подобного больше не должно происходить. Мы представляем вам возможность ускоренного восстановления базы данных (Accelerated Database Recovery, ADR) – одну из самых инновационных технологий, которые я когда-либо видел в ядре SQL Server. ADR делает возможными мгновенный откат, агрессивное сокращение журнала транзакций и обеспечивает «постоянное восстановление» для пользовательской базы данных. Я хотел начать с этой темы; однако буду дразнить ваше любопытство до конца главы.

Поддержка онлайн-индекса

Индексы могут сильно влиять на производительность базы данных. Поэтому поддержка индексов является обычной задачей для сохранения работоспособности вашей базы данных. Одной из проблем при создании индексов является доступность ваших данных с учетом всех блокировок данных в таблице. Создание или перестройка кластеризованного индекса по сути блокирует таблицу на все время операции, выполняемой над индексом. Создание или перестройка некластеризованного индекса все еще может создать проблемы, поскольку для него требуется совместная (SH) блокировка таблицы.

В SQL Server 2005 (да, это было так давно) была введена концепция создания или перестройки индекса *онлайн*. Построение индекса в режиме онлайн обеспечивает лучшую доступность приложения, поскольку блокировка таблицы во время процесса построения индекса не требуется.

> **Примечание.** На данные в таблице устанавливаются определенные блокировки, но они затрагивают меньший объем данных, чем ранее, и выполняются поэтапно при построении индекса в режиме онлайн. Чтобы узнать больше о том, как создается онлайн-индекс, обратитесь к документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/indexes/how-online-index-operations-work. Еще один полезный ресурс для понимания оригинальной реализации онлайн-индекса находится по адресу https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2005/administrator/cc966402(v=technet.10.

Хотя создание онлайн-индекса может улучшить показатели доступности, операция создания индекса может оказаться ресурсоемкой, что, в свою очередь, повлияет на общую доступность приложения в зависимости от характера запросов этого приложения. Кроме того, могут возникнуть ситуации, когда создание индекса для большой таблицы за одну операцию может оказаться проблематичным, например если создание индекса завершилось неудачно. В случае ошибки во время построения индекса (например, у вас недостаточно места в базе данных) потребуется исправить проблему и перезапустить создание индекса с самого начала. Разве не было бы неплохо иметь возможность выполнить какую-либо операцию построения индекса, «начав с того места, где она остановилась», в случае ее неудачного завершения? Также было бы неплохо запланировать построение индекса в несколько приемов, скажем для распределения этой операции по доступным окнам времени обслуживания системы.

Возобновляемые операции с индексами

В SQL Server 2017 мы представили концепцию *возобновляемой* операции *перестройки* индекса. Идея состоит в том, что вы начинаете перестройку индекса с помощью оператора T-SQL ALTER INDEX REBUILD и затем можете использовать оператор ALTER INDEX с командой PAUSE, чтобы приостановить перестройку индекса. Все результаты процесса перестройки онлайн-индекса сохраняются, поэтому вы можете продолжить его с помощью ALTER INDEX, используя команду RESUME. У вас также имеется возможность отменить оперативную перестройку индекса с помощью оператора ALTER INDEX и команды ABORT. Полное описание всех возможностей использования ALTER INDEX для возобновляемых операций с индексом можно прочитать в документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-index-transact-sql. Более подробно о том, как работает возобновляемая перестройка индекса и какие ограничения при этом существуют, вы можете прочитать по адресу https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-index-transact-sql#online-index-operations.

SQL Server 2019 вводит понятие возобновляемых операций с индексами при *создании* индекса с помощью CREATE INDEX. Синтаксис возобновляемого создания индекса описан в документации, размещенной по адресу https://docs.microsoft.com/en-us/sql/t-sql/statements/create-index-transact-sql. Вы также можете получить более подробную информацию о возобновляемом создании индекса по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/create-index-transact-sql#online-option.

Кроме того, SQL Server 2019 вводит концепцию **настройки области базы данных по умолчанию** для онлайн- и возобновляемых операций с индексами. Эти новые параметры называются ELEVATE_ONLINE и ELEVATE_RESUMABLE. Подробная информация об использовании этих параметров находится по адресу https://docs.microsoft.com/en-us/sql/relational-databases/indexes/guidelines-for-online-index-operations?view=sql-server-ver15#online-default-options.

Однако, чтобы не просто читать о возобновляемом создании индекса, давайте попробуем разобрать пример, используя настройки области базы данных.

Подготовительные шаги для использования примеров, иллюстрирующих возобновляемые операции с индексами

Для выполнения примера, представленного в этой главе, необходимо установить SQL Server 2019. В SQL Server 2017 для возобновляемой перестройки индекса требуется версия Enterprise Edition. Поэтому для выполнения примеров, приведенных в этом разделе, вам потребуется установить одну из версий SQL Server 2019: Enterprise, Evaluation либо Developer Edition.

Все сценарии и файлы для приведенного в данном разделе примера можно найти в созданном для этой книги репозитории GitHub, в каталоге **ch4_mission_critical_availability\resumableindex**.

Есть три способа выполнения сценариев, приведенных в данном примере:

- используйте записную книжку T-SQL с именем resumableindex.ipynb в Azure Data Studio (требуется версия от июня 2019 г. или более поздняя). В записной книжке T-SQL есть все инструкции для выполнения примера;
- загрузите сценарий T-SQL resumableindex.sql в SQL Server Management Studio (SSMS) или Azure Data Studio и выполните каждый шаг, как указано в сценарии;
- запускайте каждый набор операторов T-SQL по отдельности, используя сценарий T-SQL resumableindex.sql, как будет показано в следующем разделе.

Возобновляемое создание индекса

Рассмотрим, как использовать возобновляемый онлайн-индекс, шаг за шагом.

1. Выполните **шаг 1** в сценарии **resumableindex.sql**, чтобы создать базу данных для этого примера:

```
-- Шаг 1. Создайте базу данных
USE master
GO
DROP DATABASE IF EXISTS gotexasrangers
GO
CREATE DATABASE gotexasrangers
GO
```

2. Выполните **шаг 2**, чтобы создать таблицу и заполнить ее данными. Я выбрал количество строк так, чтобы построение индекса заняло более минуты. Это необходимо для демонстрации использования параметра MAX_DURATION, минимальное значение для этого параметра составляет 1 минуту. Для выполнения этого шага может потребоваться до 10–15 минут, так что возьмите чашку кофе и вернитесь, когда выполнение данной операции закончится, после чего перейдите к шагу 3.

```
-- Шаг 2. Создайте таблицу как кучу без кластеризованного индекса.
-- Сделайте таблицу достаточно большой, чтобы построение индекса
-- занимало как минимум несколько минут. Параметр возобновляемого
-- индекса MAX_DURATION имеет минимальное значение, равное 1 минуте.
USE gotexasrangers
GO
DROP TABLE IF EXISTS letsgorangers
GO
CREATE TABLE letsgorangers (col1 int, col2 char(7000) not null)
GO
SET NOCOUNT ON
GO
BEGIN TRANSACTION
GO
INSERT INTO letsgorangers values (1, 'I would love to win the
World Series')
GO 750000
COMMIT TRANSACTION
GO
SET NOCOUNT OFF
GO
```

3. Выполните **шаг 3**, чтобы создать возобновляемый кластеризованный онлайн-индекс. Обратите внимание на использование параметра MAX_DURATION. Это означает, что построение индекса будет приостановлено, если оно не завершится через 1 минуту.

```
-- Шаг 3. Попробуйте создать индекс со следующими свойствами: онлайн,
-- возобновляемый, с параметром max_duration, равным одной минуте.
CREATE CLUSTERED INDEX rangeridx ON letsgorangers (col1) WITH
(ONLINE = ON, RESUMABLE = ON, MAX_DURATION = 1)
GO
```

По истечении времени, указанного в max_duration, при выполнении операции CREATE INDEX произойдет сбой. Обычно при этом потребуется «начать все заново», так как построение индекса будет

откатываться. Но поскольку вы создали индекс как возобновляемый, сборка индекса просто приостанавливается.

Когда произойдет сбой, вам будет выдано примерно такое сообщение:

```
Msg 3643, Level 16, State 1, Line 31
The operation elapsed time exceeded the maximum time
specified for this operation. The execution has been stopped.
The statement has been terminated.
Msg 596, Level 21, State 1, Line 29
Cannot continue the execution because the session is in the kill state.
Msg 0, Level 20, State 0, Line 29
A severe error occurred on the current command.
The results, if any, should be discarded.
```

Это сообщение означает, что при выполнении операции CREATE INDEX произошла ошибка и соединение было прервано. Ситуация похожа на ошибку, однако же это не так – процесс создания индекса просто приостанавливается.

Примечание. Еще один способ приостановить построение возобновляемого индекса – использовать команду ALTER INDEX с параметром PAUSE для другого соединения, когда выполняется создание индекса.

4. Выполните **шаг 4**, чтобы проверить ход построения индекса с помощью динамического административного представления (Dynamic Management View, DMV) **sys.index_resumable_operations**.

```
-- Шаг 4. Проверьте ход сборки индекса.
USE gotexasrangers
GO
SELECT * FROM sys.index_resumable_operations
GO
```

В вашем результирующем наборе данных значение параметра **state_desc** должно быть PAUSED, а значение параметра **percent_complete** должно составлять около 30 %. Это означает, что когда вы возобновите построение индекса, то до завершения этой операции должно оставаться около 70 %.

5. Чтобы возобновить операцию построения индекса и завершить ее, вы можете использовать оператор ALTER INDEX, как в **шаге 5**.

```
-- Шаг 5. Возобновите построение индекса
ALTER INDEX rangeridx on letsgorangers RESUME
GO
```

Пока выполняется эта операция, вы можете использовать ALTER INDEX с параметром PAUSE, чтобы снова приостановить построение индекса (а затем возобновить его снова).

6. Давайте попробуем создать возобновляемый индекс другим способом. Во-первых, выполните **шаг 6**, чтобы удалить существующий индекс, и задайте два параметра настройки области базы данных, чтобы использовать по умолчанию возобновляемое создание онлайн-индексов, если оно поддерживается.

Примечание. Не все индексы могут создаваться онлайн, и не все операции создания индекса могут быть возобновлены. Например, для XML-индексов такая возможность не поддерживается. Список индексов, которые не поддерживаются для онлайн-операций, можно найти по адресу https://docs.microsoft.com/en-us/sql/relational-databases/indexes/guidelines-for-online-index-operations?view=sql-server-ver15.

```
-- Шаг 6. Удалите первый индекс. Используйте параметр настройки
-- области базы данных, чтобы по умолчанию установить
-- возобновляемое создание индексов онлайн
USE gotexasrangers
GO
ALTER DATABASE SCOPED CONFIGURATION SET ELEVATE_RESUMABLE = WHEN_SUPPORTED
GO
ALTER DATABASE SCOPED CONFIGURATION SET ELEVATE_ONLINE = WHEN_SUPPORTED
GO
DROP INDEX IF EXISTS letsgorangers.rangeridx
GO
```

7. Теперь создайте индекс, выполнив **шаг 7**, не указывая специальных параметров. Запустите оператор CREATE INDEX, подождите около 30 секунд, а затем отмените эту операцию (в то время как она все еще будет выполняться). Используйте любой способ, чтобы отменить запрос. Используемый способ зависит от вашего инструмента (в SSMS нажмите красную кнопку остановки):

```
-- Шаг 7. Снова создайте индекс. Обратите внимание, что у оператора
-- создания индекса в данном случае отсутствуют параметры.
-- Отмените эту операцию примерно через 30 секунд (используйте
-- оператор CANCEL)
CREATE CLUSTERED INDEX rangeridx ON letsgorangers (col1)
GO
```

При этом вам будет выдано приблизительно такое сообщение:

```
The statement has been terminated.
Query was canceled by user.
```

Обычно отмена CREATE INDEX приводит к откату операции. Но поскольку по умолчанию для создания индексов были выбраны параметры ONLINE и RESUMABLE, построение индекса только приостановится, даже если вы явно не указали эти параметры.

8. Проверьте состояние приостановленной сборки индекса, выполнив **шаг 8**.

```
-- Шаг 8. Проверьте, как идет построение индекса
USE gotexasrangers
GO
SELECT * FROM sys.index_resumable_operations
GO
```

Как было продемонстрировано ранее в этом примере, у параметра state_desc должно быть значение PAUSED, а percent_complete должно составлять около 20–30%.

9. Выполните **шаг 9** для возобновления и завершения построения индекса.

```
-- Шаг 9. Возобновите построение индекса
ALTER INDEX rangeridx on letsgorangers RESUME
GO
```

Помните об этом возможном сценарии, когда вы думаете о возобновляемых индексах. Допустим, создание или перестройка индекса занимает 4 часа, и в течение всего этого времени создание индекса занимает определенный объем памяти, ресурсов ЦП и ресурсов ввода-вывода, которые могут оказать некоторое влияние на ваше приложение. Теперь вы можете использовать команду CREATE, а затем PAUSE/RESUME в нескольких интервалах. Выберите интервалы для создания или возобновления построения индекса, когда нагрузка на приложение минимальна (сейчас вы планируете построение индекса в несколько этапов). Вы можете создать задачу в SQL Server Agent, чтобы запланировать эти этапы на любое время, наиболее отвечающее требованиям вашего приложения.

Поддержка онлайн-индекса для столбцовых индексов

Кластеризация для столбцового индекса имеет решающее значение в области высокопроизводительных аналитических запросов, особенно в сценариях, где используются хранилища данных. Построение (или перестройка) кластеризованного столбцового индекса может занимать достаточно долгое время, учитывая размер таблиц, для которых он обычно строится. Поскольку сборка или перестройка кластеризованного столбцо-

вого индекса выполняется в *автономном* режиме, вся таблица должна быть заблокирована от других транзакций, что отнюдь не позволяет достичь высоких показателей доступности системы.

В SQL Server 2017 появилась возможность создавать и перестраивать некластеризованные столбцовые индексы онлайн. В SQL Server 2019 теперь можно создавать и перестраивать в режиме онлайн кластеризованные столбцовые индексы. Более подробно о командах SQL, позволяющих выполнять построение кластеризованных столбцовых индексов в режиме онлайн, вы можете прочитать по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/create-columnstore-index-transact-sql. Синтаксис команд для перестройки кластеризованного столбцового онлайн-индекса будет таким же, как и для стандартного индекса с использованием ALTER INDEX. Дополнительная информация доступна по адресу https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-index-transact-sql. Возобновляемые индексы для кластеризованных или некластеризованных онлайн-индексов еще не поддерживаются.

Улучшения в группе доступности Always On (Always On Availability Group)

Группы доступности Always On (я буду называть их группами доступности до конца главы) – это наиважнейшая функция в составе новой технологии восстановления высокой доступности после аварий (High Availability Disaster Recovery, HADR) в SQL Server. Впервые появившаяся в SQL Server 2012, она усовершенствуется с каждой новой версией, расширяя возможности групп доступности.

Например, в SQL Server 2016 мы представили концепцию поддержания работоспособности базы данных в случае отказа с помощью групп доступности (вы можете узнать больше об этой концепции на https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/sql-server-always-on-database-health-detection-failover-option). Мы также повысили производительность групп доступности, о которой вы можете прочитать по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-index-transact-sql?view=sql-server-ver15.

В SQL Server 2017 одной из ключевых новых возможностей для групп доступности является концепция групп доступности *без кластеров*. Это позволяет настроить группу доступности без программного обеспечения для кластеризации. Любое аварийное переключение выполняется вручную, но эта возможность может позволить вам установить платформу репликации с горизонтальным масштабированием для чтения или даже настроить группы доступности в Windows и Linux. Более подробная информация об этой возможности доступна по ссылке https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/read-scale-availability-groups.

Для SQL Server 2019 мы представили две новые возможности для групп доступности, добавленные на основе отзывов клиентов и технологических тенденций:

- поддержка большего количества реплик;
- новый метод, обеспечивающий подключение вашего приложения к первичной реплике.

Поддержка большего количества синхронных реплик

Теперь мы поддерживаем до пяти синхронных реплик в группе доступности; всего поддерживается до девяти реплик. Для получения дополнительной информации см. документацию по адресу https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/always-on-availability-groups-sql-server.

Перенаправление подключений чтения/записи от вторичной к первичной реплике

Перенаправление подключений чтения/записи от вторичной реплики к первичной – это новая возможность, позволяющая приложению всегда обращаться к первичной реплике, независимо от того, на каком экземпляре SQL находится первичная реплика для группы доступности.

В предыдущих версиях SQL Server единственным доступным разработчику способом убедиться, что вы подключены к основной реплике группы доступности, было использование концепции *прослушивателя*. Однако прослушиватель может не всегда быть доступен при настройке групп доступности SQL Server, таких как упомянутая ранее в этом разделе группа доступности без кластеров.

В синтаксис T-SQL в SQL Server 2019 теперь включены параметры для групп доступности и параметры строки подключения для клиентских приложений, позволяющие выполнить настройки так, чтобы приложение всегда было подключено к первичной реплике, независимо от того, к какому серверу в группе доступности подключено приложение. SQL Server предоставляет встроенную логику для этой концепции и использует концепцию перенаправления, если приложение подключено к серверу, который был вторичной репликой.

Вы можете прочитать все подробности о том, как настроить перенаправление подключений к первичной реплике для SQL Server и вашего приложения, в документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/secondary-replica-connection-redirection-always-on-availability-groups.

Ускоренное восстановление базы данных

Один из увлекательных аспектов моей работы в Microsoft – это узнавать об инновациях, которые начинаются как проекты, и видеть, как они превращаются в новую функцию выпущенного продукта, иногда спустя годы.

Я могу привести пример того, как в 2016 году я присоединился к проектированию в проекте с названием «постоянное время восстановления»

(constant time recovery CTR). Я помню, как потратил некоторое время, чтобы понять, какова цель этого проекта, увидев в числе участников проекта имя моего давнего коллеги Питера Бирна (Peter Byrne). CTR стал ускоренным восстановлением базы данных (Accelerated Database Recovery, ADR) в SQL Server 2019 и Azure SQL Database.

CTR фактически начался как проект в 2015 году. Его инициировал Ханума Кодавалла (Hanuma Kodavalla), выдающийся инженер (Distinguished Engineer) в Microsoft. Ханума привлек других для участия в проекте, в том числе Питера Бирна, Панагиотиса Антонопулоса (Panagiotis Antonopoulos) и Срикумара Рангараджана (Srikumar Rangarajan). Участники проекта пытались решить очень большую проблему, существовавшую в SQL Server: *длительные транзакции*. Когда инженеры – участники проекта закончили работу, они решили написать статью о разработанной концепции. Перед тем как вы продолжите чтение этой главы и будете изучать приведенные примеры, я рекомендую вам ознакомиться со статьей о проекте CTR по ссылке www.microsoft.com/en-us/research/publication/constant-time-recovery-in-azure-sql-database/. Я называю этот документ «статья про CTR» (CTR Paper) и далее буду называть его именно так. Я рекомендую вам проделать то, что проделал я, когда писал эту главу. Найдите этот документ и регулярно обращайтесь к нему, читая главу и используя примеры.

Проблема длительных активных транзакций

Длительные транзакции могут привести к тому, что **восстановление «выйдет из-под контроля»** (фактически оно займет очень много времени, которое вы не можете предсказать) и повлияет на доступность приложения, работающего с данными. Это не ошибка или проблема с ядром SQL Server; все работает именно так, как нужно. SQL Server не может предотвратить запуск приложением большого количества операций изменения данных внутри транзакции или же транзакции, которая вносит мало изменений, но не завершается или не выполняет откат (отмену транзакции) в течение длительного периода времени. Это классическое определение *длительной транзакции*.

Кроме того, для отката длительной транзакции требуется выполнение операции, называемой операцией компенсации. Откат требует логической отмены. Для выполнения операции DELETE на объеме данных в 1 миллион строк потребуется 1 миллион записей DELETE в журнале. При откате такой транзакции удаления данных потребуется отменить все выполненные удаления, а также внести в журнал 1 миллион записей INSERT. Это очень сильно удлиняет время, необходимое для отката транзакции. Я часто видел, как клиенты пытаются завершить сеанс с длительной активной транзакцией с помощью операции KILL и удивляются, почему KILL не работает мгновенно. Обычно это происходит из-за того, что для безопасного завершения сеанса требуется выполнить откат транзакции (иначе ваши данные могут стать несогласованными).

Другим следствием длительной транзакции является ее влияние на **сокращение журнала транзакций**. Журнал транзакций может быть сокращен только до самой старой активной транзакции. Вы не можете удалить записи журнала транзакций для транзакции, которая еще не завершена или не отменена. Но поскольку журнал транзакций является последовательным, одна «старая» активная транзакция (причем она может не иметь никакой связанной с ней активности) может задержать сокращение журнала транзакций для каждой другой транзакции, выполненной после нее. Для вас это выглядит так, что журнал транзакций, кажется, «выходит из-под контроля» (и часто вы не можете понять, почему).

Ускоренное восстановление баз данных было разработано для решения всех этих проблем.

Как работает ускоренное восстановление баз данных

Ускоренное восстановление баз данных (Accelerated Database Recovery, ADR) предназначено для решения проблем с длительными транзакциями. Для этого используются возможности, перечисленные в документации, доступной по ссылке https://docs.microsoft.com/en-us/azure/sql-database/sql-database-accelerated-database-recovery:

- **быстрое восстановление базы данных без потери согласованности данных**.

  При использовании ADR длительные транзакции не влияют на общее время восстановления, обеспечивая быстрое восстановление базы данных без потери согласованности данных, независимо от количества активных транзакций в системе или их размеров;

- **мгновенный откат транзакции**.

  С помощью ADR откат транзакции происходит мгновенно, независимо от времени, когда транзакция была активной, или количества выполненных обновлений;

- **агрессивное сокращение журнала**.

  С помощью ADR журнал транзакций быстро сокращается даже при наличии активных длительных транзакций, вместе с тем не позволяя ему выйти из-под контроля.

Обычное восстановление SQL Server

Чтобы понять, как ускоренное восстановление базы данных может решить описанные выше проблемы, сначала необходимо понять, как работает обычное восстановление в SQL Server.

Рассмотрите схему на рис. 4.1, приведенную в статье про CTR (www.microsoft.com/en-us/research/publication/constant-time-recovery-in-azure-sql-database/).

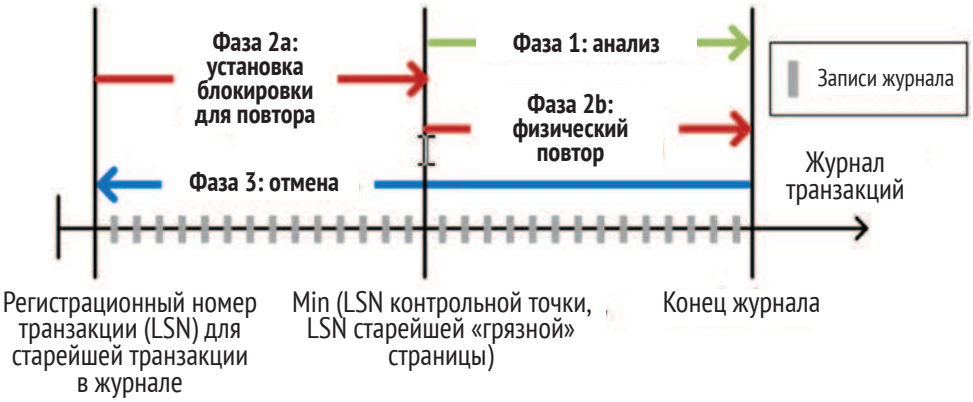


Рис. 4.1. Нормальный процесс восстановления SQL Server

Процесс восстановления для SQL Server состоит из трех этапов:

1. **Анализ.**

   Начните с записи журнала, с момента фиксации контрольной точки (CHECKPOINT), и последовательно просмотрите все записи журнала до конца журнала.

   Этот анализ позволяет SQL Server получить следующую информацию:

   - какие транзакции не были завершены (т. е. еще были активны) во время последнего перевода базы данных в автономный режим (это может быть просто завершение работы SQL Server). Вероятно, это транзакции, которые необходимо откатить для обеспечения согласованности;
   - запись журнала, содержащая самую старую измененную, или «грязную», страницу.

   Это нужно, чтобы при необходимости повторить любые подтвержденные транзакции, если страницы, связанные с подтвержденными транзакциями, не отражают состояние транзакции.

   Другими словами, анализ заключается в настройке восстановления для выполнения следующих двух этапов: **повтора** и **отмены**.

2. **Повтор.**

   Чтобы ваши данные были согласованными, SQL Server должен обеспечить точное отражение любых завершенных транзакций во время восстановления. Для того чтобы обеспечить согласованность, необходимо выполнить поиск записи журнала для самой старой измененной (или «грязной») страницы в журнале транзакций и сравнение регистрационного номера транзакции в журнале (Log Sequence Number, LSN) для каждой записи журнала в подтвержденной транз-

акции с LSN на соответствующей странице. Если LSN страницы меньше, чем LSN записи журнала, операция записи журнала (INSERT, UPDATE, DELETE и т. д.) должна быть изменена. Для подтвержденных транзакций каждая запись в журнале проверяется таким образом до конца журнала.

Однако фаза восстановления, фаза 2, фактически начинается с записи журнала, в которой отражена самая старая активная транзакция. Фаза повтора начинается в этом месте, потому что она должна установить блокировки для активных транзакций, чтобы база данных могла быть доступна для пользователей (которые будут при этом заблокированы, чтобы они не могли запрашивать и изменять данные активных транзакций) после завершения фазы повтора. Однако это означает, что на фазу восстановления влияет размер записей журнала, относящихся к самой старой активной транзакции. Теперь вы понимаете, почему длительность активной транзакции может повлиять на продолжительность восстановления? Как только повтор завершен, происходит третий (и последний) этап отмены.

3. **Отмена.**

   SQL Server должен обеспечивать следующие правила согласованности данных: на страницах данных должны отражаться лишь завершенные транзакции. Вы можете задаться вопросом: каким же образом SQL Server может защитить страницы данных на диске для транзакций, которые не были завершены? Это возможно благодаря тому, что SQL Server может в любое время записать измененную («грязную») страницу на диск, если страница нужна другому пользователю и при этом нет свободных страниц.

   Или же может быть выполнена операция установки контрольной точки (включая дополнительную контрольную точку), которая будет записывать измененные страницы на диск независимо от того, завершена транзакция или нет. Поэтому если работа SQL Server была завершена при существующих активных транзакциях, которые не были завершены, ядро базы данных должно гарантировать во время восстановления, что никакие изменения данных, связанные с незавершенной транзакцией, не были отражены на страницах базы данных. Любые активные транзакции в этом состоянии необходимо отменить (аналогично тому, как если бы ROLLBACK TRANSACTION была осуществлена во время выполнения).

   Для выполнения необходимых операций отката транзакций SQL Server производит все это путем последовательного просмотра журнала в обратном направлении, начиная с конца журнала до записи журнала, относящейся к самой старой активной транзакции. Теперь время отмены пропорционально длине самой старой активной транзакции. Именно поэтому клиенты удивляются, когда они

прерывают работу SQL Server в надежде на быстрое восстановление базы данных, однако длительная транзакция не может быть прервана (потому что она находится в состоянии отката), а восстановление занимает много времени. Это связано с тем, что SQL Server должен поддерживать согласованность ваших данных и завершать откат транзакции, который он выполнял ранее.

Эта система восстановления, основанная на концепции под названием ARIES (более подробную информацию см. в статье, опубликованной по адресу https://dl.acm.org/citation.cfm?id=128770), хорошо работала для SQL Server в течение 25 лет и прекрасно работает (и все еще требуется и используется) практически везде, за исключением одного сценария, который я назвал, – длительной активной транзакцией, которая удлиняет время как на обработку повтора (повторного выполнения транзакции), так и на отмену транзакции.

Теперь давайте посмотрим, как ускоренное восстановление баз данных (ADR) меняет картину.

Ускоренное восстановление базы данных (Accelerated Database Recovery, ADR) в SQL Server

Я не буду пытаться подробно рассказать о том, что описано в статье про CTR; однако рассмотрю основные компоненты, обеспечивающие работу ADR, и отличия ADR от стандартного подхода к восстановлению (ARIES), а затем приведу пример, дающий более глубокое понимание.

ADR представляет концепцию постоянного хранилища версий (Persistent Version Store, PVS). В SQL Server есть концепция, называемая хранилищем версий, которая используется для изоляции моментальных снимков, но это хранилище версий размещается в базе данных tempdb. Концепция PVS аналогична данному подходу в том, что версии изменений для строк сохраняются; но хранилище версий PVS является постоянным, поскольку оно находится в пользовательской базе данных (хранилище версий для изоляции моментальных снимков не является постоянным, так как оно расположено в базе данных tempdb, а tempdb воссоздается после каждого перезапуска сервера). После включения ADR SQL Server начнет отслеживать изменения, используя версии. Версии могут быть сохранены либо в строке страницы данных (*внутристрочно, in-row*), либо в *автономном* (*off-row*) хранилище в базе данных. Все версии сохраняют предыдущее состояние данных перед изменением и идентификатор транзакции, изменяющей данные и создавшей эту версию, чтобы легко определить, должна ли эта версия данных быть видимой для других транзакций.

Теперь, с использованием PVS, такие операции, как откат транзакции, становятся *простыми*. Если транзакция откатывается, SQL Server просто помечает ее как ABORTED. Теперь любой запрос, просматривающий строку данных, может определить, является ли версия строки видимой и следует

ли ее использовать. Если последняя версия строки связана с транзакцией, помеченной как ABORTED, запрос может игнорировать эту версию и искать предыдущую. Если версия строки связана с завершенной или активной транзакцией, применяются правила уровня изоляции транзакций, чтобы определить, является ли строка видимой.

SQL Server поддерживает состояние транзакций, чтобы все это работало с помощью концепции *карты прерванной транзакции* (*Aborted Transaction Map*). Это обсуждается более подробно в документе CTR.

> **Примечание.** Один из аспектов ADR, который может показаться сложным, – это уровни изоляции транзакций. Текущая функция версий (в базе данных tempdb) специально создана для поддержки уровней изоляции моментальных снимков. Версии для ADR не созданы для поддержки изоляции моментальных снимков, но могут использоваться для их поддержки вместе с другими возможностями ADR.

Постоянное хранилище версий (PVS) также обеспечивает преимущество для быстрого восстановления (отсюда появился термин «постоянное время восстановления»). Фаза повтора просто необходима, чтобы убедиться, что хранилище версий согласовано на уровне строк на страницах таблиц. Для отмены выполненных изменений просто нужно пометить любую активную транзакцию как прерванную, а процесс управления версиями, как описано ранее, сделает все остальное. Это делает восстановление очень быстрым.

Некоторые транзакции, в основном системные транзакции (например, распределение страниц, обновление данных статистики), не могут использовать новую схему PVS. Поэтому, когда ADR включен, SQL Server поддерживает вторичный журнал (secondary log – *Slog*, который хранится в журнале транзакций) для любых транзакций, которые не могут использовать управление версиями. Транзакции, связанные со вторичным журналом Slog, должны использовать обычный режим восстановления ARIES. К счастью, системные транзакции почти всегда короткие и потому не вызывают тех проблем, которые возникают в случае длительных пользовательских транзакций.

На рис. 4.2, приведенном в статье про CTR, показан новый процесс восстановления, когда ADR включен.

Как показано на этом рисунке, при ускоренном восстановлении базы данных SQL Server все так же выполняются три фазы восстановления: анализ, восстановление и отмена. Но сейчас процесс идет намного быстрее, поэтому срок восстановления сократился.

Анализ по-прежнему должен выполнять то же самое, что раньше, начиная с последней записи журнала для контрольной точки (CHECKPOINT), но повтор и отмена существенно различаются.

Повтор гарантирует, что записи журнала для операций Slog будут перемещены из самой старой активной транзакции в запись журнала из самой старой операции с «грязной» страницей. Начиная с этого момента во время

фазы повтора будут выполняться те же операции по обеспечению правильности фиксации изменений данных, что и при восстановлении ARIES. Но эта последовательность обычно должна быть короткой, при условии что для базы данных используются стандартные конфигурации контрольных точек.

Отмена просто помечает любую незафиксированную транзакцию как прерванную, но при этом потребуется отменить операции транзакций Slog так же, как выполняется отмена пользовательских транзакций ARIES. При этом, однако, системные транзакции, как правило, коротки, и их число по сравнению с общим объемом всех транзакций очень мало, поэтому данный процесс всегда должен быть быстрым.

В результате мы получаем новую, невероятно быструю систему восстановления, в основу которой положено хранилище версий в пользовательской базе данных.

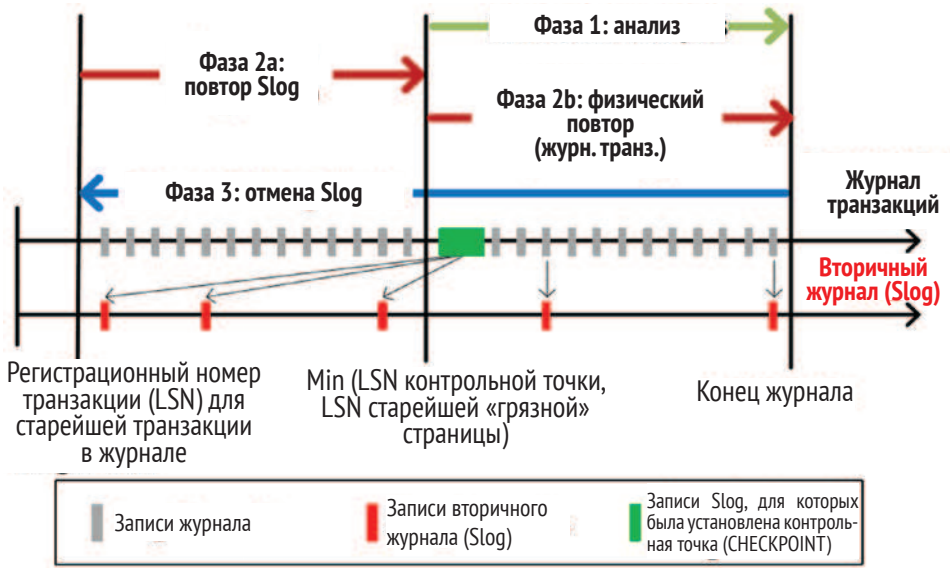


Рис. 4.2. Ускоренное восстановление базы данных в SQL Server

Давайте рассмотрим пример, чтобы подробнее узнать, чем отличается ведение журнала транзакций при использовании ADR. Этот пример полностью автономен и не зависит от сценариев других примеров, приведенных в книге. Для его запуска вам просто понадобится SQL Server 2019 и SQL Server Management Studio (SSMS) или Azure Data Studio (ADS). Вам нужно будет использовать сценарий **alookatadr.sql**, который находится в каталоге **ch4_mission_critical_availability\adr**.

1. Откройте сценарий **alookatadr.sql** и выполните **шаг 1** этого сценария, чтобы создать базу данных.

```
-- Шаг 1. Создайте базу данных с режимом упрощенного восстановления.
Default for ADR is OFF
```

```
USE master
GO
DROP DATABASE IF EXISTS gocowboys
GO
CREATE DATABASE gocowboys
GO
ALTER DATABASE gocowboys SET RECOVERY SIMPLE
GO
```

Совет. В этом примере используется простая модель восстановления базы данных, чтобы упростить изучение записей журнала.

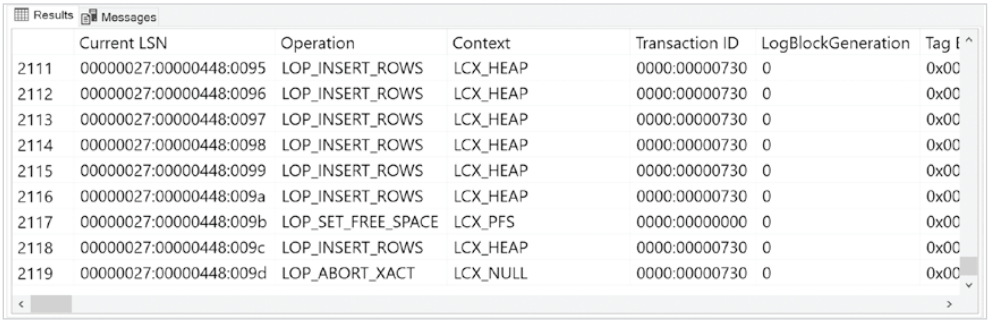
2. Запустите **шаг 2** сценария, чтобы создать таблицу и вставить строки. Обратите внимание, что сценарий добавляет в таблицу 1000 строк. Из-за оптимизации с использованием ADR вы не можете просто использовать одну строку (об этой оптимизации будет более подробно рассказываться далее в этой главе):

```
-- Шаг 2. Создайте очень простую таблицу и вставьте 1000 строк.
USE gocowboys
GO
DROP TABLE IF EXISTS howboutthemcowboys
GO
CREATE TABLE howboutthemcowboys (col1 int, col2 char(100) not null)
GO
INSERT INTO howboutthemcowboys VALUES (1, 'Whitten has returned')
GO 1000
```

3. Выполните **шаг 3**, чтобы сократить журнал с помощью CHECKPOINT (дабы было легче увидеть существующие записи журнала) и удалить 1000 строк в транзакции. Откатите транзакцию и просмотрите журнал, используя системную функцию **sys.fn_dblog**:

```
-- Шаг 3. Сократите журнал, удалите все строки, откатите назад
-- транзакцию и посмотрите записи журнала
CHECKPOINT
GO
BEGIN TRANSACTION;
DELETE FROM howboutthemcowboys
ROLLBACK TRANSACTION
GO
SELECT * FROM sys.fn_dblog(NULL, NULL)
GO
```

В результатах запроса к журналу должно содержаться более 2000 строк. Если вы просмотрите результаты до конца, то должны увидеть записи журнала, такие как показано на рис. 4.3.



Results Messages

	Current LSN	Operation	Context	Transaction ID	LogBlockGeneration	Tag I
2111	00000027:00000448:0095	LOP_INSERT_ROWS	LCX_HEAP	0000:00000730	0	0x00
2112	00000027:00000448:0096	LOP_INSERT_ROWS	LCX_HEAP	0000:00000730	0	0x00
2113	00000027:00000448:0097	LOP_INSERT_ROWS	LCX_HEAP	0000:00000730	0	0x00
2114	00000027:00000448:0098	LOP_INSERT_ROWS	LCX_HEAP	0000:00000730	0	0x00
2115	00000027:00000448:0099	LOP_INSERT_ROWS	LCX_HEAP	0000:00000730	0	0x00
2116	00000027:00000448:009a	LOP_INSERT_ROWS	LCX_HEAP	0000:00000730	0	0x00
2117	00000027:00000448:009b	LOP_SET_FREE_SPACE	LCX_PFS	0000:00000000	0	0x00
2118	00000027:00000448:009c	LOP_INSERT_ROWS	LCX_HEAP	0000:00000730	0	0x00
2119	00000027:00000448:009d	LOP_ABORT_XACT	LCX_NULL	0000:00000730	0	0x00

Рис. 4.3. Записи журнала для прерванного удаления (DELETE)

Обратите внимание на все записи журнала с названием LOP_INSERT_ROWS, которые следуют до LOP_ABORT_XACT, прерывающим транзакцию. Эти записи LOP_INSERT_ROWS являются записями журнала компенсации для операции DELETE. Логический откат DELETE – это INSERT. Все записи журнала LOP_INSERT_ROW и завершающая запись LOP_ABORT_XACT были сгенерированы как часть оператора ROLLBACK TRANSACTION.

4. На **шаге 4** включите ADR с помощью ALTER DATABASE, заново создайте таблицу и снова вставьте строки:

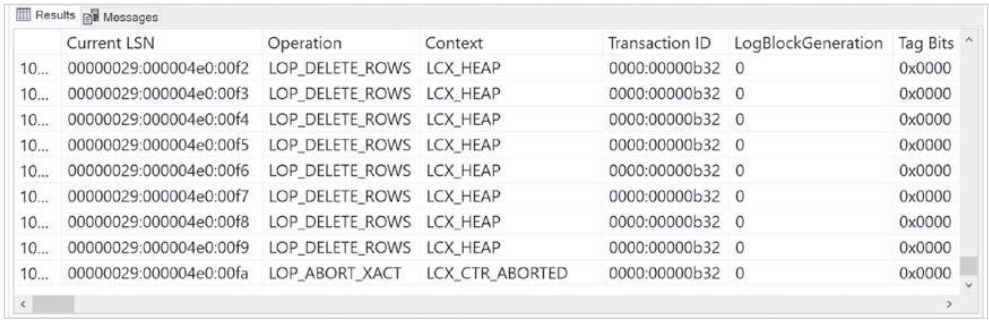
```
-- Шаг 4. Перейдите на использование ADR для БД. Создайте таблицу
-- заново.
ALTER DATABASE gocowboys SET ACCELERATED_DATABASE_RECOVERY = ON
GO
USE gocowboys
GO
DROP TABLE IF EXISTS howboutthemcowboys
GO
CREATE TABLE howboutthemcowboys (col1 int, col2 char(100) not null)
GO
INSERT INTO howboutthemcowboys VALUES (1, 'Whitten has returned')
GO 1000
```

5. На **шаге 5** повторите то же упражнение, чтобы сократить журнал, откатить DELETE и просмотреть записи журнала в журнале транзакций:

```
-- Шаг 5. Удаление, выполнение отката и повторный просмотр журнала
CHECKPOINT
GO
BEGIN TRANSACTION
DELETE FROM howboutthemcowboys
ROLLBACK TRANSACTION
GO
```

```
SELECT * FROM sys.fn_dblog(NULL, NULL)
GO
```

Теперь вы должны увидеть только 1000+ записей журнала. Просмотрите все записи до конца выведенных результатов; посмотрите на последние записи журнала. Они должны выглядеть как на рис. 4.4.



Results Messages

	Current LSN	Operation	Context	Transaction ID	LogBlockGeneration	Tag Bits
10...	00000029:000004e0:00f2	LOP_DELETE_ROWS	LCX_HEAP	0000:00000b32	0	0x0000
10...	00000029:000004e0:00f3	LOP_DELETE_ROWS	LCX_HEAP	0000:00000b32	0	0x0000
10...	00000029:000004e0:00f4	LOP_DELETE_ROWS	LCX_HEAP	0000:00000b32	0	0x0000
10...	00000029:000004e0:00f5	LOP_DELETE_ROWS	LCX_HEAP	0000:00000b32	0	0x0000
10...	00000029:000004e0:00f6	LOP_DELETE_ROWS	LCX_HEAP	0000:00000b32	0	0x0000
10...	00000029:000004e0:00f7	LOP_DELETE_ROWS	LCX_HEAP	0000:00000b32	0	0x0000
10...	00000029:000004e0:00f8	LOP_DELETE_ROWS	LCX_HEAP	0000:00000b32	0	0x0000
10...	00000029:000004e0:00f9	LOP_DELETE_ROWS	LCX_HEAP	0000:00000b32	0	0x0000
10...	00000029:000004e0:00fa	LOP_ABORT_XACT	LCX_CTR_ABORTED	0000:00000b32	0	0x0000

Рис. 4.4. Записи журнала для прерванной операции удаления (DELETE) с использованием ADR

Обратите внимание, что в выведенных результатах отсутствуют записи компенсации INSERT, а LOP_ABORT_XACT имеет контекст LCX_CTR_ABORTED. Все записи журнала LOP_DELETE_ROWS были созданы с помощью инструкции DELETE. Оператор T-SQL ROLLBACK TRAN сгенерировал только запись LOP_ABORT_XACT.

6. SQL Server 2019 предоставляет диагностику для проверки постоянного хранилища версий (PVS). Выполните **шаг 6**, чтобы увидеть операторы для PVS в этой базе данных.

```
-- Шаг 6. Просмотрите статистику PVS
SELECT * FROM sys.dm_tran_persistent_version_store_stats
WHERE database_id = db_id('gocowboys')
GO
```

Теперь, когда вы познакомились с тем, как работает ADR, давайте рассмотрим два примера, чтобы увидеть, как ADR улучшает производительность отката и сокращения журнала, а также обеспечивает более быстрое восстановление.

Использование ускоренного восстановления базы данных

Как вы видели в только что завершенном примере, для применения ускоренного восстановления базы данных (Accelerated Database Recovery, ADR)

не требуется никаких изменений вашего приложения. Вы просто используете следующий оператор T-SQL, чтобы включить ADR, – и работаете:

```
ALTER DATABASE <dbname> SET ACCELERATED_DATABASE_RECOVERY = ON
```

Давайте рассмотрим два примера, чтобы посмотреть:

- как быстро выполняется откат и как агрессивно сокращается журнал транзакций;
- как скоро восстанавливается, что позволяет вам быстрый доступ к базе данных.

Все, что вам нужно для выполнения этих примеров, можно найти в записных книжках и сценариях для каждого примера.

Быстрый откат и агрессивное сокращение журнала

Используйте следующий пример, чтобы увидеть, как можно выполнить быстрый откат с помощью ADR и насколько агрессивно теперь сокращается журнал транзакций, во избежание чрезмерного роста журнала. В этом примере мы сравним скорость отката и рост журнала транзакций с ADR и без него.

Вы можете выполнить этот пример, используя сценарий T-SQL **adr.sql** в каталоге **ch4_mission_critical_availability\adr**.

В этом примере я рекомендую применять записную книжку T-SQL **adr.ipynb**, размещенную в каталоге **ch4_mission_critical_availability\adr**. В записной книжке есть полные инструкции по созданию базы данных, созданию таблицы и вставке данных. Затем, при отключенном ADR, вы удалите все строки таблицы в транзакции. Далее вы обратите внимание на полный объем пространства журнала, который был использован, но не может быть сокращен даже после контрольной точки. Затем вы определите скорость отката всего удаления (или, если быть более точным, отсутствие скорости).

Потом, используя записную книжку, вы повторите эти шаги, но на этот раз при включенном ADR. Сценарий T-SQL содержит те же самые шаги, что и в первый раз. Изучив этот пример, давайте сделаем что-то более продвинутое. Посмотрим на скорость восстановления, используя тот же пример T-SQL, но с большим количеством строк, чтобы увидеть влияние восстановления.

Ускорение восстановления

Чтобы показать, как работает быстрое восстановление, необходимо создать пример, в котором фаза отмены восстановления должна попытаться откатить большое количество изменений или транзакций.

Итак, как создать сценарий, в котором SQL Server должен запускать этап отмены определенной транзакции? Для этого вам необходимо создать сце-

нарий для активной транзакции, для которого НЕ будет выполнен откат или фиксация до завершения работы SQL Server. Есть три способа сделать это:

- выполните инструкцию T-SQL **SHUTDOWN WITH NOWAIT**;
- завершите работу службы SQL Server (например, net stop mssqlserver);
- завершите процесс SQLSERVR.EXE (для Windows используйте команду **End Task** (Завершить задачу) в диспетчере задач).

Любой из этих перечисленных методов остановит SQL Server, не влияя на активную транзакцию. Есть еще одно соображение. Для того чтобы SQL Server откатывал активные транзакции, «должно быть что-то, что необходимо откатить». Если страницы данных, на которые влияет активная транзакция, никогда не записывались на диск, то когда SQL Server запускает восстановление, он не может «отменить» то, чего никогда не было. Поэтому при использовании любого из перечисленных методов вы должны выполнить команду CHECKPOINT для базы данных (завершение работы службы SQL Server корректно запускает контрольную точку для всех баз данных). Обратите внимание, что, возможно, Recovery Writer или Lazy Writer уже перенесли эти страницы на диск, однако вы не можете полностью положиться на это при демонстрации.

> **Совет**. Что делать, если вы хотите принудительно повторить транзакцию? Нужно использовать противоположный подход. У вас должна быть завершенная транзакция, однако страницы, затронутые транзакцией, не должны быть записаны на диск. Запустите транзакцию, аналогичную приведенным в этой главе, но при этом завершите транзакцию (COMMIT). Затем имитируйте «сбой» сервера, но сделайте это без контрольной точки, поэтому используйте команду **End Task** (Завершить задачу).

Вооружившись этими знаниями, вы можете просмотреть сценарий T-SQL **adr_recovery.sql** или записную книжку T-SQL **adr_recovery.ipynb**, размещенные в каталоге **ch4_mission_critical_availability\adr**. Я рекомендую использовать записную книжку, так как в ней имеются комментарии к каждому шагу, с указанием того, когда вам необходимо имитировать сбой сервера.

Далее вы будете просматривать ERRORLOG при прохождении шагов сценария (из записной книжки или размещенного в указанном каталоге сценария). Я приведу примеры того, что вы должны увидеть, когда восстановление выполняется с включенным ADR и без него.

Ниже приведен пример ERRORLOG, когда ADR не включен:

```
spid25s   Recovery of database 'gocowboys' (6) is 2% complete
          (approximately 697 seconds remain). Phase 2 of 3. This is an
          informational message only. No user action is required.
spid25s   Recovery of database 'gocowboys' (6) is 5% complete
          (approximately 682 seconds remain). Phase 2 of 3. This is an
```

```
          informational message only. No user action is required.
spid25s   Recovery of database 'gocowboys' (6) is 7% complete
          (approximately 667 seconds remain). Phase 2 of 3. This is an
          informational message only. No user action is required.
spid25s   Recovery of database 'gocowboys' (6) is 7% complete
          (approximately 667 seconds remain). Phase 3 of 3. This is an
          informational message only. No user action is required.
spid8s    Recovery of database 'gocowboys' (6) is 40% complete
          (approximately 113 seconds remain). Phase 3 of 3. This is an
          informational message only. No user action is required.
spid8s    Recovery of database 'gocowboys' (6) is 50% complete
          (approximately 94 seconds remain). Phase 3 of 3. This is an
          informational message only. No user action is required.
spid8s    Recovery of database 'gocowboys' (6) is 59% complete
          (approximately 79 seconds remain). Phase 3 of 3. This is an
          informational message only. No user action is required.
spid8s    Recovery of database 'gocowboys' (6) is 68% complete
          (approximately 65 seconds remain). Phase 3 of 3. This is an
          informational message only. No user action is required.
spid8s    Recovery of database 'gocowboys' (6) is 76% complete
          (approximately 48 seconds remain). Phase 3 of 3. This is an
          informational message only. No user action is required.
spid8s    Recovery of database 'gocowboys' (6) is 84% complete
          (approximately 32 seconds remain). Phase 3 of 3. This is an
          informational message only. No user action is required.
spid8s    Recovery of database 'gocowboys' (6) is 93% complete
          (approximately 15 seconds remain). Phase 3 of 3. This is an
          informational message only. No user action is required.
spid8s    1 transactions rolled back in database 'gocowboys' (6:0). This
          is an informational message only. No user action is required.
spid8s    Recovery is writing a checkpoint in database 'gocowboys' (6).
          This is an informational message only. No user action is
          required.
spid8s    Recovery completed for database gocowboys (database ID 6) in
          211 second(s) (analysis 15 ms, redo 56340 ms, undo 154549 ms.)
          This is an informational message only. No user action is
          required.
```

Примечание. Повтор здесь необходим для некоторых системных транзакций, связанных со статистикой индекса для этого примера.

Ниже приведен пример ERRORLOG, когда ADR включен. Восстановление при включенном ADR происходит очень быстро; SQL Server даже не выдает сообщение о том, сколько времени заняло восстановление!

```
spid25s   1 transactions rolled back in database 'gocowboys' (6:0). This
          is an informational message only. No user action is required.
spid25s   Recovery is writing a checkpoint in database 'gocowboys' (6).
          This is an informational message only. No user required.
```

Ускорение восстановления базы данных. Основные моменты

Все это звучит слишком хорошо, чтобы быть правдой; и я уверен, что вам интересно, нет ли здесь каких-либо подводных камней. Если ускоренное восстановление баз данных (ADR) настолько хорошо, почему бы нам не использовать его всегда?

Производительность и размер базы данных

Когда я разговаривал с клиентами об их использовании ADR, то задавал два вопроса:

Объем базы данных увеличивается?

Краткий ответ на этот вопрос – конечно, да. Более важный момент – насколько он увеличивается. Поскольку мы храним версии строк в базе данных в течение определенного периода времени, естественно, объем, необходимый для постоянного хранилища версий (PVS), будет больше, чем без него.

Проблема, как и в случае с любой подобной функцией, заключается в том, что «здесь возможны варианты».

Какие варианты возможны? Чтобы сделать осознанный выбор, вам надлежит принять во внимание ряд факторов, которые мы перечислим ниже:

- насколько интенсивно ваше приложение выполняет операции записи с большим количеством изменений данных? Чем больше число изменений, тем больший объем потребуется для хранения версий;
- сколько времени занимают транзакции, которые должны прочитать данные из сохраненных версий? Если версии данных не используются ни в одном запросе, их можно не использовать.

Ускоренное восстановление баз данных имеет встроенную оптимизацию, позволяющую минимизировать объем хранилища версий, включая следующее:

- **«по требованию».**

  При обновлении строки SQL Server может «повторно использовать» версию строки, которая была аннулирована, и вместо нее записать новую версию. Это происходит в процессе изменения данных;
- **фоновая очистка.**

  Что делать, если все еще существуют версии, которые больше не нужны (например, прерванные транзакции), а обновление еще не прои-

зошло? SQL Server использует существующую фоновую архитектуру рабочих потоков для планирования очистки (каждые несколько минут) любых версий, которые могут быть отклонены, как для внутристрочного (in-row), так и для автономного (off-row) хранения версий данных. SQL Server использует концепцию, называемую *логическим возвратом* (logical revert), для очистки ненужных версий. Логический возврат – это процесс, гарантирующий, что зафиксированная версия строки является «основной» строкой страницы, тем самым сокращая «список» версий для просмотра. Раздел 3.3 статьи про CTR (www.microsoft.com/en-us/research/publication/constant-time-recovery-in-azure-sqldatabase/) содержит отличное подробное описание того, как работает логический возврат. Кроме того, в разделе 3.7 этой же статьи описывается весь процесс очистки.

Раздел 4 статьи содержит результаты экспериментального тестирования роста объема баз данных на примере 50 млн операций вставки, обновления и удаления. Команда инженеров, проводившая тестирование, обнаружила, что после 50 млн обновлений PVS увеличило базу данных примерно на 1 Гб. Просмотрите эти результаты, поскольку они также показывают значительное уменьшение размера журнала транзакций из-за использования ADR (вы наблюдали это при выполнении примеров кода, приведенных в данной главе, когда использовали сценарий **alookatadr.sql**).

Влияет ли использование ADR на производительность?

Это, пожалуй, самый распространенный вопрос, касающийся использования ADR. Как и для вопроса о факторе роста, здесь нет однозначного ответа. Поскольку ADR отслеживает каждое изменение данных, создавая новые версии, наибольшее влияние на производительность будет наблюдаться в ситуациях с интенсивной записью данных. Также могут внести свой вклад и операции чтения для приложений с интенсивной записью: потребуется выполнение дополнительных операций, чтобы найти нужную версию строки.

В статье про CTR описано, что группа инженеров провела тестирование с использованием тестов на основе тестов производительности TPC-C и TPC-E, являющихся стандартом для индустрии (для получения дополнительной информации о тестах производительности TPC-C и TPC-E см. www.tpc.org). TPC-C – это более старый стандартный тест производительности, но он выполняет много операций записи. TPC-E является более сбалансированным, но «все еще OLTP» эталоном рабочей нагрузки при записи данных. Процесс выполнения тестов и их результаты описываются в разделе 4.2 статьи; здесь я приведу лишь краткие результаты: при запуске TPC-C влияние на производительность оказалось около 14 % для внутристрочного (in-row) хранения версий данных, а при запуске TPC-E влияние на производительность оказалось около 2,5 % для внутристрочного (in-row) хранения версий данных.

Я провел собственное «быстрое и грязное» тестирование с использованием инструмента с открытым исходным кодом HammerDB (подробнее см. по ссылке www.hammerdb.com). Этот инструмент поставляется с вариантом эталонного теста TPC-C. При использовании 10 хранилищ / 10 виртуальных пользователей в течение 5-минутного периода я наблюдал влияние на производительность около 15 % при использовании ADR.

> **Примечание.** Ни один из этих результатов тестов не гарантирует, что вы получите точно такие же цифры либо вообще заметите какое-либо влияние на производительность, если включите ADR для своей базы данных. Это связано с тем, что при проведении подобного тестирования используются тесты определенного типа рабочей нагрузки, которые могут соответствовать или не соответствовать профилю рабочей нагрузки вашего приложения. Разработайте стандартный способ тестирования производительности вашего приложения с ADR и без него, чтобы узнать истинное влияние ADR на производительность.

Просмотрите результаты в разделе 4 статьи – там приведены впечатляющие цифры по времени восстановления, которые команда наблюдала в Azure (вы уже видели эффект ускоренного восстановления на простой базе данных).

Еще одним важным преимуществом ускоренного восстановления баз данных является то, что время отработки отказа для групп доступности Always On (Always On Availability Groups) может быть быстрее, и быстрее выполняются запросы к репликам только на чтение.

Неожиданные сценарии

В некоторых случаях для хранилища постоянных версий (PVS) невозможно использовать режим хранения версий страницы данных «в строке» (in-row), поскольку данные просто не помещаются на одной странице. В этом случае версии хранятся во внутренней системной таблице.

Это так называемые сценарии «автономного» (off-row) хранения данных. Как вы наверняка догадываетесь, случаи, когда PVS использует «автономный» режим хранения, – это не идеальная ситуация. Поэтому таких сценариев следует по возможности избегать.

Версии, хранящиеся вне строки, в «автономном» хранилище, возникают главным образом тогда, когда выполняемая операция обновления данных существенно изменяет текущую версию строки. Если обновление является настолько значительным, невозможно или не имеет смысла хранить версию в строке; версия будет сохранена во внутренней системной таблице как автономная версия. Я провел несколько экспериментов на базах данных с включенным ADR, при этом для хранения PVS использовалась таблица с именем **persistent_version_store** в каждой базе данных. Эта таблица имеет тип INTERNAL_TABLE, напоминающий другие внутренние таблицы

для Query Store. Эта системная таблица содержит данные о версии и метаданные, чтобы связать ее с актуальной строкой на странице таблицы.

Если вас беспокоит, генерирует ли ваше приложение множество автономных версий, вы можете использовать счетчики производительности и расширенные события, о которых будет рассказываться в следующем разделе под названием «Контроль ADR».

> **Примечание.** На тот момент, когда я писал эту главу, версии PVS, хранящиеся вне строки, размещались в файловой группе PRIMARY вашей базы данных, и выбрать другое место для их хранения было невозможно. Команда инженеров обсуждала, могут ли они добавить возможность перемещения «автономного» PVS в другую файловую группу, выбранную пользователем при помощи ALTER DATABASE. Обратитесь к документации по ALTER DATABASE, чтобы узнать, появилось ли это усовершенствование в финальной версии SQL Server 2019.

Другой неожиданной ситуацей может стать *оптимизация коротких транзакций*. Это хорошая ситуация. Использование PVS не имеет смысла, когда транзакции очень короткие. Поэтому при тестировании ADR не следует рассматривать транзакции, в которых удаляется несколько строк для применения ADR. Если вы просматриваете журнал с использованием fn_dblog(), то можете увидеть транзакции, в которых ADR не будет применяться со следующими операциями: LOP_FORGET_XACT и LCX_XACT_DOES_NOT_SUPPORT_CTR.

Контроль ADR

Как и многие новые функции SQL Server, команда ADR имеет типы ожидания, расширенные события и счетчики монитора производительности для контроля выполнения ADR, использования хранилища постоянных версий (PVS) и обработки операций очистки.

На рис. 4.5 показаны некоторые счетчики производительности, доступные для отслеживания использования хранилища постоянных версий (PVS), включая отслеживание количества создаваемых версий вне строки (в «автономном» режиме).

Существует несколько расширенных событий, которые также можно использовать для отслеживания определенного поколения версий и для задач очистки. Вы можете получить информацию обо всех этих событиях, выполнив следующие запросы к представлениям динамического управления XE (XE Dynamic Management Views):

```
select * from sys.dm_xe_objects where name like '%pvs%'
select * from sys.dm_xe_objects where name like '%ctr%'
```

Одним из таких событий, интересных с точки зрения контроля ADR, является событие pvs_add_record. Вы можете использовать это событие

вместе с sql_text, чтобы выяснить, какие запросы генерируют версии вне строки.

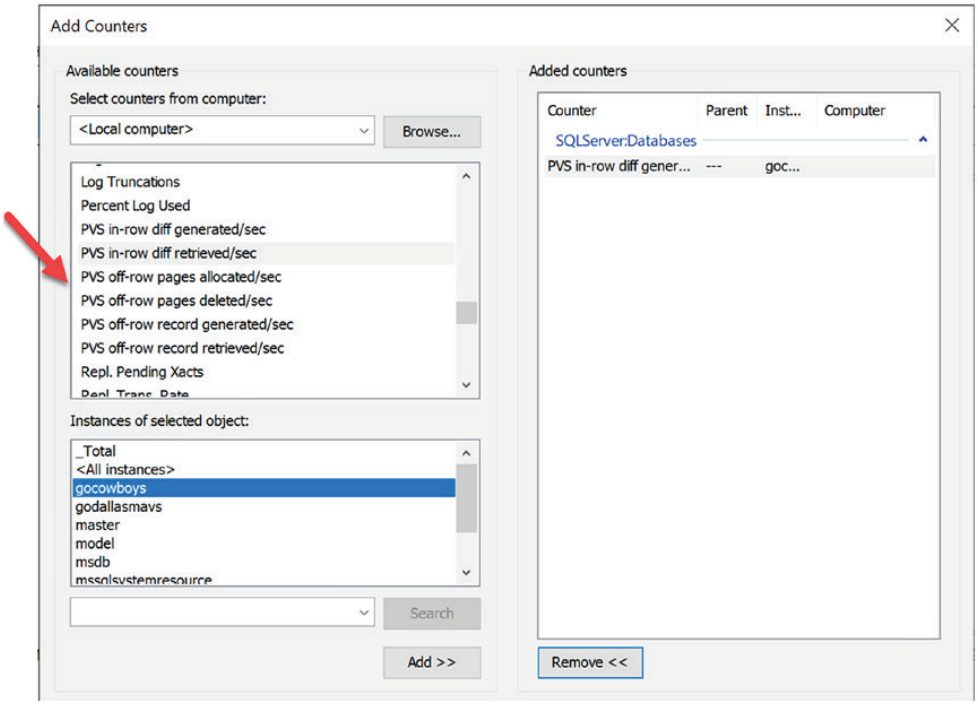


Рис. 4.5. Счетчики производительности PVS

> **Примечание.** Я не проводил никакого тестирования с использованием этих расширенных событий, поэтому не могу сказать, какое влияние они могут оказать на ваше приложение.

И наконец, для тех, кто действительно хочет углубиться в данную тему, есть несколько типов ожидания, перечисленных в статистике DMV sys.dm_os_wait_stats, которые применяются к ADR. Вы можете отслеживать их, чтобы контролировать выполнение таких операций, как очистка. Используйте следующую инструкцию T-SQL для поиска этих типов ожидания:

```
select * from sys.dm_os_wait_stats
where wait_type like '%pvs%' or wait_type like '%ctr%'
```

Нужно ли использовать ADR?

Если у вас нет длительных транзакций, ADR, скорее всего, не поможет и может отрицательно повлиять на ваше приложение. Если ваше приложение генерирует много автономных версий, воздействие может быть слишком

сильным, чтобы увидеть преимущества ADR. Имейте в виду, однако, что ADR все еще может быть очень полезным для сценариев восстановления после отказа, например с использованием групп доступности Always On.

Я рекомендую разработать методику тестирования ADR для вашего приложения. Мы не реализовали использование ADR по умолчанию (пока) для SQL Server 2019, потому что профили рабочих нагрузок весьма различны и не для всех – как уже обсуждалось в этой главе – использование ADR даст очевидное преимущество. Однако возьмите на заметку следующие последние замечания.

- ADR – одна из тех функций, про которые вы не можете определенно сказать, что они нужны вам сейчас, но когда они вам понадобятся… они вам понадобятся! Возможно, вы не сможете предсказать, в какой именно момент длительное восстановление приведет к существенным потерям для вашего бизнеса. Разве не хорошее решение – включить эту функцию, чтобы исключить подобную ситуацию?
- В большинстве случаев профили рабочей нагрузки приложений не похожи на тесты производительности TPC-C, выполняющие очень много операций записи данных. Наши результаты тестирования с более сбалансированной рабочей нагрузкой по чтению/записи, такой как TPC-E, не оказали большого влияния.
- Примите во внимание следующую цитату из статьи про CTR, так как команда инженеров, писавших эту статью, использовала «облачный» подход для развертывания ADR в Azure: «…CTR уже используется в пяти регионах, примерно в одном миллионе баз данных, и полученные результаты весьма обнадеживают».

Резюме

Обеспечение доступности ваших данных и приложений является важным аспектом любого продукта платформы данных. SQL Server 2019 продолжает улучшать основные функции доступности, такие как возобновляемая онлайн-индексация и группы доступности.

Кроме того, SQL Server 2019 предлагает инновационный подход к решению проблем простоя из-за длительных активных транзакций с помощью ускоренного восстановления баз данных.

Глава 5

Современная платформа разработки

Данные нужны практически любому разработчику, а такой продукт, как SQL Server, обладает всеми необходимыми возможностями, включая языки программирования, драйверы и платформы. Современному разработчику данных нужна платформа баз данных для решения современных задач. SQL Server 2019 отвечает этим потребностям, предоставляя следующие возможности:

- поддержка широкого спектра языков и драйверов для различных платформ операционных систем, таких как Windows, macOS и Linux, а также обеспечение совместимости между ними. Различные версии SQL Server предоставляют общий внешний слой для минимизации логики приложения;
- графовые базы данных, интегрированные с SQL Server, позволяют разработчикам реализовывать различные модели данных, такие как социальные сети, без необходимости включать в архитектуру своих приложений дополнительные продукты, и выполнять запросы к ним, используя знакомый язык T-SQL;
- разработчикам нужна способность создавать приложения для обработки данных Unicode, используя системы кодирования, широко распространенные в отрасли. SQL Server 2019 поддерживает кодировку UTF-8 благодаря новым правилам сравнения и хранения данных;
- разработчикам нужна платформа баз данных для поддержки новых типов приложений, интегрированных с машинным обучением, – приложений, которые масштабируемы, безопасны и интегрированы с базой данных. **Службы машинного обучения SQL Server (SQL Server Machine Learning Services)** включают эти новые возможности SQL Server 2019;
- язык T-SQL предоставляет множество возможностей, но разработчикам может потребоваться больше. Им нужна возможность расширения языка T-SQL, интегрированного с базой данных. **Расширения**

языка в SQL Server (SQL Server Language Extensions) в версии SQL Server 2019 позволяют разработчикам устанавливать и использовать новые языки, такие как Java, интегрированные с данными SQL Server.

Языки, драйверы и платформы

Когда я начал работать в Microsoft в 1993 году, разработчики, создававшие приложения для SQL Server, в основном использовали такие языки, как Visual C или Visual Basic, используя драйвер DB-Library. Все клиенты работали с DOS (да, DOS) или Windows, тогда как SQL Server был основным сервером баз данных Windows NT. Вскоре появились C ++ и ODBC, но выбор языков, драйверов и платформ был довольно ограничен. Сегодня разнообразие выбора языков, драйверов и платформ для клиентских приложений и SQL Server превосходит все те варианты, о которых я когда-либо мог подумать.

Языки и драйверы

SQL Server 2019 является современной платформой баз данных, и наряду с этим предлагается широкий выбор языков программирования, включая современные языки, популярные у разработчиков, которые традиционно не использовались с SQL Server.

Наряду с выбором языков предлагается широкий выбор драйверов для доступа к SQL Server, полностью соответствующих потребностям каждого языка. Кроме того, эти драйверы работают на различных клиентских платформах, включая macOS, Linux и Windows.

Также выбор драйверов, таких как ODBC, OLE-DB и .Net, стал более ясным, чем тот широкий, но иногда весьма запутанный набор средств, который предлагался в прошлом.

Итак, как вы выбираете язык и/или драйвер? Во-первых, выбор языка в некоторых случаях определяется драйвером, который вы намерены использовать. Например, если вы хотите написать код на PHP и получить доступ к SQL Server, то должны использовать драйвер PHP для SQL Server.

К счастью, Microsoft создала веб-сайт, который поможет вам принять решение о языке, выбрать правильный драйвер, одну или несколько клиентских платформ и увидеть примеры кода на этом языке для обеспечения доступа к SQL Server.

Чтобы ознакомиться с этими работающими примерами, перейдите на сайт http://aka.ms/sqldev. Основная страница этого сайта выглядит так, как показано на рис. 5.1.

Наведя курсор на один из вариантов выбора языка, вы можете выбрать язык клиентской платформы, чтобы получить подробные сведения об использовании этого языка с соответствующим драйвером и учебным примером кода (на рис. 5.2 показан пример использования Go).

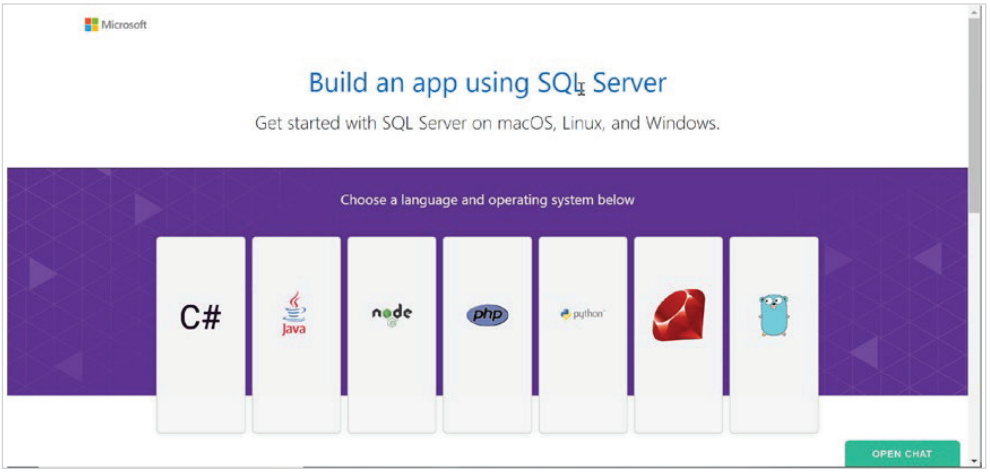


Рис. 5.1. Центр разработки SQL Server

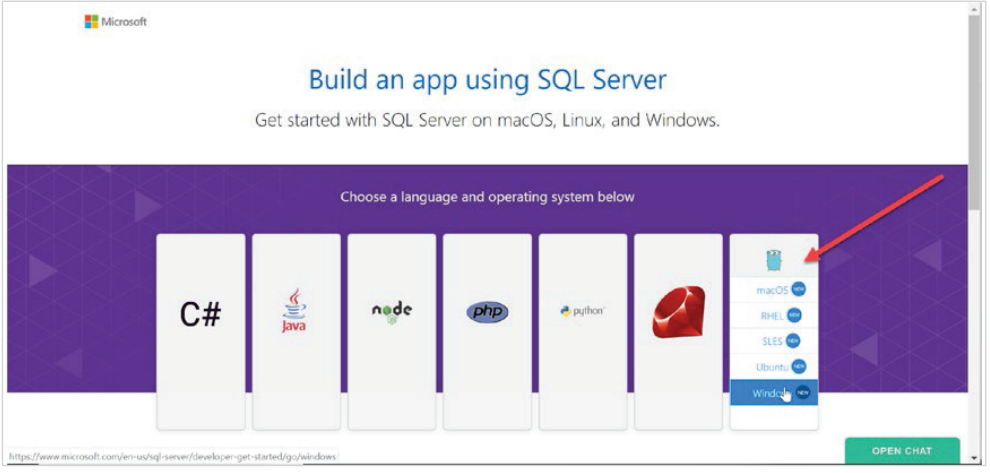


Рис. 5.2. Использование языка Go с SQL Server

Выбрав вариант Windows, вы получите полное руководство по созданию вашего первого приложения Go для Windows (ваш SQL Server может работать на Windows, Linux или даже в контейнерах), используя версию SQL Server Developer Edition. У каждого языка и платформы есть шаблон, аналогичный тому, который показан на рис. 5.3 для Go.

Многие разработчики, использующие такие языки, как C++ или C#, видели довольно запутанный список вариантов выбора драйверов для SQL Server.

После выхода версии SQL Server 2012 мы консолидировали список драйверов и версий, которые необходимо использовать, для ODBC, OLE-DB или ADO.Net. Я нашел соответствующую страницу документации https://docs.microsoft.com/en-us/sql/connect/connect-history, чтобы продемонстрировать богатую

историю прошлых драйверов и предоставить информацию о том, какие из них следует использовать сейчас, если вы работаете с SQL Server 2014 или более поздней версией.

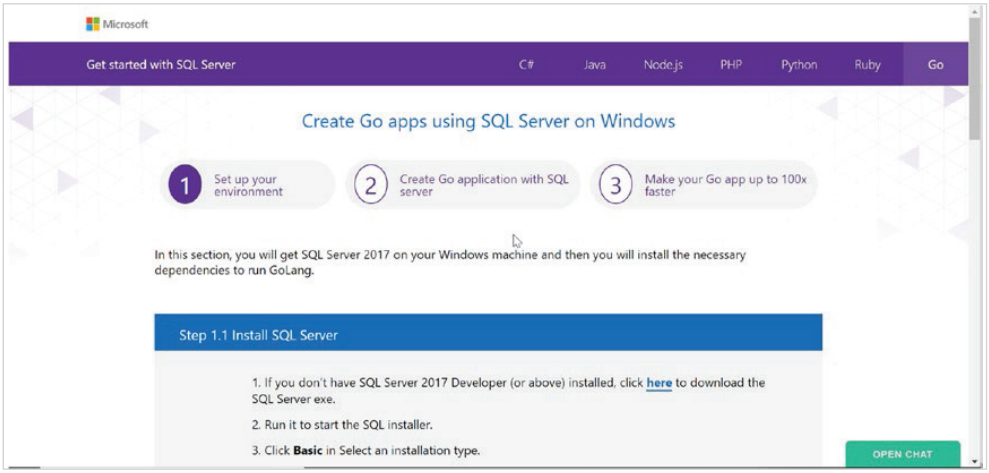


Рис. 5.3. Создание приложения Go для SQL Server

Кроме того, чтобы получить полный список языков и подходящих драйверов для создания приложения с помощью SQL Server, я рекомендую вам ознакомиться со следующей страницей документации: https://docs.microsoft.com/en-us/sql/connect/sql-connection-libraries (этот список включает драйвер для приложений, использующих интегрированную среду объектно-реляционного отображения (object-relational mapping, ORM)).

Я немного «старомодный» разработчик, поэтому чаще всего выбираю драйвер ODBC. И мы создали драйвер ODBC для SQL Server, который соответствует новым функциям SQL Server и доступен для Linux, macOS и Windows. Вы можете ознакомиться с полной документацией для последней версии драйвера ODBC для SQL Server по https://docs.microsoft.com/en-us/sql/connect/odbc/microsoft-odbc-driver-for-sql-server.

Платформы и версии

Начиная с SQL Server 2017 теперь SQL Server поддерживается в Windows, Linux и контейнерах Docker (Docker Containers). Это создает платформу для разработчиков, которые никогда раньше не рассматривали SQL Server как средство разработки. Независимо от того, на какой операционной системе работает промышленный SQL Server, разработчики могут тестировать свои приложения, применяя SQL Server Developer Edition для Windows, Linux или в контейнерах. А поскольку на всех этих платформах используется одинаковая кодовая база ядра SQL Server, разработчики могут создавать приложения, локальные для их среды, в которой установлен SQL Server, и при этом быть полностью уверенными, что они совместимы с SQL Server, работающим в промышленной среде. Этот новый подход включает те возмож-

ности, которые прежде были недоступны для SQL Server на Linux, такие как репликация и распределенные транзакции (Distributed Transactions, DTC), которые теперь также поддерживаются в SQL Server 2019.

Хотя это и не является новой возможностью SQL Server 2019 (такая возможность появилась в SQL Server 2016 SP1), версии SQL Server, отличные от Enterprise Edition, содержат аналогичные «верхнеуровневые» функции, которые упрощают создание приложений, работающих с разными версиями SQL Server. Например, OLTP в памяти (In-Memory OLTP) теперь доступна для SQL Server Enterprise Edition и SQL Server Standard Edition и даже для SQL Server Express Edition (хотя эта функция по-разному поддерживается для разных версий). Теперь вы можете создать свое приложение, использующее OLTP в памяти, в Developer Edition, и быть уверенными, что оно будет работать с разными версиями, не предусматривая в своем приложении дополнительную логику, позволяющую определить, с какой именно версией вы работаете.

Графовая база данных

Концепция реляционной базы данных охватывает все типы моделей проектирования, схемы данных и приложения. Тем не менее существуют определенные разновидности моделей данных, разработанные для решения задач в определенной области, которые не всегда полностью соответствуют стандартной реляционной модели и языку SQL. Такие модели обычно включают иерархические, «сетевые» или сложные отношения данных «многие ко многим». В Википедии есть хорошее описание этой проблемы и ее решения, доступное по ссылке https://en.m.wikipedia.org/wiki/Graph_database.

Некоторые разработчики все еще пытались использовать реляционную базу данных, чтобы «встроить» ее в модель графа, и писать сложные запросы T-SQL для навигации по графу. Были даже созданы специальные проекты для графовых данных, такие как популярная графовая база данных с открытым исходным кодом Neo4j (https://github.com/neo4j/neo4j). Другие платформы баз данных разрабатывали «надстройки» над своими реляционными базами данных, чтобы обеспечить возможность работы с данными графов.

В 2016 году члены команды SQL Engineering, в том числе Ханума Кодавалла (Hanuma Kodavalla), Крейг Фридман (Craig Freedman), Девин Райдер (Devin Rider) и Шрея Верма (Shreya Verma), создали проект для изучения возможностей построения графовой базы данных в SQL Server и Azure SQL Database. Их целью было включить встроенные возможности работы с графами в ядро SQL Server и найти способ использовать язык T-SQL для создания графовых таблиц, а также обрабатывать графовые данные и выполнять поисковые запросы к таким данным с помощью T-SQL. Это еще один замечательный пример расширения возможностей T-SQL.

Результатом этих усилий стала поддержка графовой базы данных в SQL Server 2017 и Azure SQL Database. Одним из огромных преимуществ графовой базы данных в SQL Server является то, что она обладает всеми мощ-

ными возможностями SQL Server. К этим возможностям относятся HADR, безопасность, производительность и все функции ядра. Другие платформы используют иной подход к решению данной проблемы. Вместо того чтобы включать эти возможности в ядро базы данных, они рассматривают такие функции как надстройки или полностью отдельные продукты.

Что представляет собой графовая база данных в SQL Server?

Графовая база данных в SQL Server использует таблицы для представления *узлов* и *ребер* в графовой модели с использованием расширений T-SQL. Использование термина «база данных» здесь оправдано, поскольку графовая база данных не является отдельной базой данных в SQL Server. В графовой базе данных узел – это сущность или объект, а ребро – это отношение между узлами.

Таким образом, графовая база данных представляет собой набор таблиц узлов и ребер, а также данных и метаданных, которые связывают их вместе. SQL Server поддерживает расширения языка T-SQL для определения таблицы узлов или ребер с помощью синтаксиса AS NODE или AS EDGE для оператора CREATE TABLE. Полное описание синтаксиса команды, используемой для создания таблицы узлов или ребер, можно найти по адресу https://docs.microsoft.com/en-us/sql/t-sql/statements/create-table-sql-graph.

Кроме того, SQL Server поддерживает новое ключевое слово T-SQL – MATCH, позволяющее перемещаться по таблицам узлов и ребер, как часть оператора SELECT. Описание правил использования ключевого слова MATCH можно найти в документации по адресу https://docs.microsoft.com/en-us/sql/t-sql/queries/match-sql-graph.

Вы можете ознакомиться с полной документацией для графовой базы данных в SQL Server и Azure SQL Database, перейдя по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-overview. Вы также можете просмотреть отличную презентацию, подготовленную сотрудником Microsoft Кевином Фарли (Kevin Farlee), на YouTube по адресу www.youtube.com/watch?v=xirfl_t4Gqs.

Однако лучший способ увидеть, как работает таблица узлов или ребер в SQL Server, – это рассмотреть пример.

Использование графовой базы данных в SQL Server

Многие из вас, вероятно, являются новичками в использовании графовой базы данных в SQL Server, поэтому я приведу простой пример, чтобы продемонстрировать всю мощь этой возможности. Фактически я буду использовать пример, приведенный в документации, который можно найти по адресу https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-sample, однако добавлю в этот пример несколько дополнений, а также сопровожу его разъясняющими комментариями.

Рассмотрим такое понятие, как социальная сеть. Многие из вас используют их каждый день, пользуясь такими платформами, как Facebook или LinkedIn. Сеть по своей природе представляет собой соединение объектов, обычно моделируемое в виде графа. Рассмотрим сеть друзей, показанную на рис. 5.4.

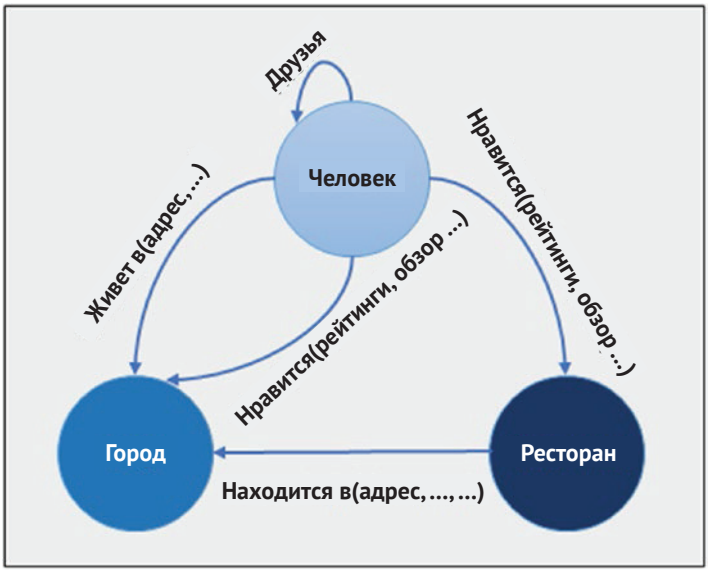


Рис. 5.4. Сеть друзей

В этой модели графа узлами являются Человек (Person), Город (City) и Ресторан (Restaurant). Стрелки представляют отношения между узлами; это – ребра графа. Обратите внимание на особые отношения под названием Friends, которые связывают людей друг с другом. Построить эту модель, используя набор реляционных таблиц, не так уж и сложно, но обход графа с использованием традиционных запросов T-SQL становится достаточно трудоемкой задачей.

Давайте создадим пример графовой базы данных с использованием предыдущей модели, чтобы вы могли познакомиться с основами, а затем посмотреть, что нового в SQL Server 2019.

> **Примечание.** Для практического выполнения примеров использования графовой базы данных не требуется особых предварительных настроек, кроме установки SQL Server 2019 (на Windows, Linux или в контейнере) и использования таких инструментов, как SQL Server Management Studio (SSMS) или Azure Data Studio (ADS) (июньской версии 2019 г. или более поздней). Пример графа в этом разделе разработан для SQL Server 2019.

Для нашего примера допустим, что некто Джон (John) является пользователем социальной сети. Он дружит с Мэри (Mary) и Джули (Julie), но еще

не знает, с кем они дружат (и с кем дружат их друзья и т. д.). Он хочет расширить свою социальную сеть, а также узнать, какие рестораны нравятся его друзьям.

Рассмотрим «социальную сеть друзей» на рис. 5.5.

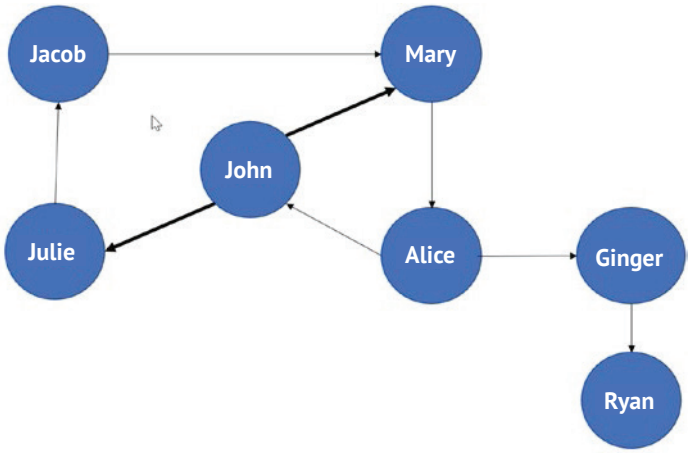


Рис. 5.5. Социальная сеть друзей

Джон не знает всей социальной сети, поэтому ему нужно использовать графовую базу данных для навигации.

Все сценарии для примеров использования графовой базы данных можно найти в каталоге **ch5_modern_development_platform\sqlgraph**. Используя модель на рис. 5.4 в качестве руководства, выполните шаги 1–7 в записной книжке T-SQL **socialnetwork.ipynb**, воспользовавшись Azure Data Studio. На примере этих шагов вы узнаете, как строить таблицы графов в виде узлов и ребер, добавлять данные, а затем выполнять обход графа с использованием синтаксиса T-SQL MATCH.

Также вы можете использовать сценарий T-SQL **socialnetwork.sql** для выполнения шагов 1–7 с применением SSMS или ADS.

Расширение возможностей работы с графовыми данными в SQL Server 2019

SQL Server 2019 включает несколько улучшений, которые помогли сделать графовую базу данных более мощной и привлекательной платформой для работы с данными графов по сравнению с другими продуктами. Эти возможности включают в себя:

- обход пути графа с использованием нового синтаксиса SHORTEST_PATH();
- поддержку производных таблиц (derived tables) и представлений (views) в графовой базе данных;

- ограничения ребер, необходимые для построения правильных отношений внутри графа;
- использование оператора T-SQL MERGE для поддержки таких сценариев, как *upsert*.

Давайте рассмотрим некоторые из этих новых возможностей.

SHORTEST_PATH

Одной из наиболее распространенных задач, которые необходимо решить с помощью графовой базы данных, является рекурсивный просмотр данных графа без необходимости вручную переходить на каждый уровень. SQL Server 2017 не поддерживает эту возможность, но SQL Server 2019 обеспечивает ее поддержку посредством нового синтаксиса T-SQL **SHORTEST_PATH()**.

Используя примеры **шагов 8** и **9** в **socialnetwork.ipynb** и **socialnetwork.sql**, посмотрите, как SHORTEST_PATH позволяет Джону и Джейкобу выполнять обход социальной сети друзей.

Вы можете ознакомиться с подробностями использования SHORTEST_PATH(), обратившись к документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-shortest-path, или к заметке в блоге Шреи Верма (Shreya Verma) по ссылке https://techcommunity.microsoft.com/t5/SQL-Server/Public-Preview-of-Shortest-Path-on-SQL-Server-2019/ba-p/721240.

Ограничения ребер

Хотя синтаксис NODE и EDGE T-SQL обеспечивает отличный новый способ построения данных графа с использованием таблиц SQL Server, в SQL Server 2017 не существовало способа обеспечить целостность данных узлов и ребер. Аналогично тому, как концепция ограничений внешнего ключа в таблицах SQL Server поддерживает целостность данных в таблицах, SQL Server 2019 поддерживает возможность обеспечения целостности данных для узлов и ребер.

Исследуйте только что созданную социальную сеть, используя примеры из этой главы. Было бы неплохо убедиться, что любые данные для таблицы ребер friendOf должны быть взяты из соответствующей строки в таблице Person. Ограничения ребер предоставляют такую возможность.

Кроме того, ограничения ребер поддерживают правильные связи внутри сети. В нашей модели социальной сети человеку может нравиться ресторан, но ресторану не может нравиться человек. Ограничение ребер также может обеспечить выполнение этого правила. Кроме того, ограничения ребер гарантируют, что ребра не останутся «оборванными», т. е. ссылающимися на несуществующий объект. Ограничения ребер не позволяют удалить узел, который является частью отношения ребер, если такие данные присутствуют в таблице ребер. Опять же, подобное поведение аналогично ограничениям внешних ключей в традиционных реляционных таблицах.

Вы можете получить более подробную информацию об ограничениях ребер, обратившись к документации по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/tables/graph-edge-constraints или к заметке в блоге Шреи по ссылке https://docs.microsoft.com/ru-ru/archive/blogs/sqlserverstorageengine/public-preview-of-graph-edge-constraints-on-sql-server-2019.

Использование MERGE с таблицами графов

SQL Server использует оператор T-SQL под названием MERGE, который выполняет операции вставки, обновления или удаления целевой таблицы на основе результатов объединения с исходной таблицей. В SQL Server 2017 вы можете использовать инструкцию MERGE для консолидации операций DML в таблицах узлов, но не в таблицах ребер. SQL Server 2019 теперь предоставляет возможность применять MERGE и для таблиц ребер.

Взгляните на несколько замечательных примеров на эту тему в блоге Шреи Верма (Shreya Verma), перейдя по ссылке https://blogs.msdn.microsoft.com/sqlserverstorageengine/2018/07/16/match-support-in-merge-dml-for-graph-tables/.

Поддержка UTF-8

Приложения и базы данных использовали Unicode в качестве стандарта для кодирования символьных данных еще в начале 1990-х годов. SQL Server в Windows включает типы данных и порядки сопоставления для поддержки кодировки символов Unicode почти с самого начала своего возникновения и развития как продукта. SQL Server 2019 представляет новый метод использования Unicode под названием UTF-8, который широко используется приложениями и базами данных, обычно в системах Linux.

Если вы хотите получить больше информации об основах Unicode перед прочтением этого раздела, то можете обратиться к следующим источникам:

- https://docs.microsoft.com/en-us/windows/win32/intl/unicode;
- https://en.wikipedia.org/wiki/Unicode;
- https://unicode.org/standard/WhatIsUnicode.html.

Unicode и SQL Server

Обычный способ, которым SQL Server поддерживает кодировку Unicode, – это использование типов данных nchar и nvarchar. SQL Server поддерживает кодировку Unicode этих типов данных с использованием схемы кодирования UTF-16. В UTF-16 потребуется как минимум 2 байта на «символ» для хранения данных. Символ ASCII, применяющий UTF-16 «a», использует 2 байта памяти. Когда вы определяете столбец, как показано ниже:

```
col1 nchar(10) not null
```

SQL Server использует 20 байт памяти для этого столбца, даже если вы храните в нем только стандартные символы ASCII.

Этот способ несовместим со следующим определением столбца:

```
col1 char(10) not null
```

для которого потребуется 10 байт, но в котором можно хранить только символы из набора символов ASCII.

> **Примечание.** Среди разработчиков и специалистов по обработке данных очень распространена следующая ошибка: они используют типы данных nchar, char и т. д. и при этом считают, что длина – это просто количество символов, а не объем хранилища памяти в байтах.

UTF-16 поддерживает весь набор символов Unicode, потому что многие символы для поддержки различных языков, кроме символов ASCII, используют 2 байта для представления символа.

Зачем нужно использовать UTF-8?

Хотя большинство приложений и баз данных, таких как SQL Server, используют UTF-16 для кодировки символов Unicode, многие в сообществе Linux применяют кодировку под названием UTF-8. UTF-8 похож на UTF-16 в том, что он поддерживает весь набор символов Unicode, но использует другую схему кодирования и хранения байтов в зависимости от того, какой символ хранится. Например, для символов ASCII потребуется только 1 байт памяти в UTF-8, но в UTF-16 для тех же символов потребуется 2 байта.

Рассмотрим в качестве примера приложение, где используется только набор символов ASCII, которое нуждается в обновлении, чтобы обеспечить поддержку всех символов Unicode. Возможно, это приложение использует типы данных char и varchar в SQL Server. Прежде чем SQL Server начал поддерживать UTF-8, специалист по данным был бы должен изменить типы данных всех столбцов SQL Server, заменив **char** на **nchar** и **varchar** на **nvarchar**. Использование ALTER TABLE для изменения типов данных таким образом для символов ASCII практически удваивает требования к объему памяти, необходимому для хранения данных этих столбцов, даже если вы изначально используете только символы ASCII.

Теперь, с появлением версии SQL Server 2019, существует альтернативный вариант решения этой проблемы – оставить ваши типы данных char и varchar, но изменить порядок сопоставления для столбцов, чтобы использовать новые варианты порядка сопоставления UTF-8, такие как LATIN1_GENERAL_100_CI_AS_SC_UTF8. ALTER TABLE позволяет изменить порядок сопоставления на уровне столбца. В кодировке UTF-8 для символов потребуется только 1 байт памяти для символов ASCII.

Принимайте эти решения с осторожностью, поскольку существуют некоторые ограничения для UTF-8, а для хранения некоторых символов в UTF-8 (не ASCII) потребуется больше памяти, чем для UTF-16.

UTF-8 поддерживается для SQL Server для Windows, Linux и контейнеров (помните, что порядок сопоставления «Windows» поддерживается для SQL Server в Linux благодаря архитектуре SQLPAL. В документации говорится, что UTF-8 поддерживается только для порядка сопоставления символов в Windows, но не порядка сопоставления SQL).

Чтобы узнать, подходит ли вам UTF-8, обратитесь к следующим ресурсам:

- https://cloudblogs.microsoft.com/sqlserver/2018/12/18/introducing-utf-8-support-in-sql-server-2019-preview/;
- https://docs.microsoft.com/en-us/sql/relational-databases/collations/collation-and-unicode-support;
- https://docs.microsoft.com/en-us/sql/relational-databases/collations/collation-and-unicode-support#utf8.

Службы машинного обучения SQL Server

Когда я присоединился к команде SQL Engineering в 2016 году, проработав в Microsoft более 20 лет, Джозеф Сирош (Joseph Sirosh) занимал должность вице-президента по SQL Server. SQL Server 2016 только что вышел, и я знал, что Джозеф особенно гордится работой по интеграции машинного обучения с SQL Server благодаря поддержке языка программирования R, проделанной командой, разрабатывающей SQL Server.

Джозеф – ученый, специалист в области науки о данных, увлекается машинным обучением и работой с данными. Как язык с открытым исходным кодом, R является одним из самых популярных языков программирования в области науки о данных и машинного обучения, поэтому его интеграция с SQL Server казалась естественной. Кроме того, в 2015 году Microsoft приобрела компанию Revolution Analytics, которая создала коммерческую версию R под названием **RevoScaleR**, включив в нее изменения, позволяющие сделать этот язык более масштабируемым. (Если вы хотите узнать больше об истории R, обратитесь к следующему ресурсу: https://en.wikipedia.org/wiki/R_(programming_language).) Теперь стало возможным объединить эти силы для создания платформы машинного обучения, используя данные, размещенные на SQL Server. Эта функция получила название **SQL Server 2016 R Services**. В SQL Server 2017 была представлена интеграция с языком программирования Python, в основу которой были положены те же концепции и архитектура. Благодаря этим новым возможностям R и Python совместно стали **службами машинного обучения SQL Server** (SQL Server Machine Learning Services).

Хотя изменения в службах машинного обучения SQL Server (ML Services) для SQL Server 2019 не являются кардинальными, эти возможности могут

привлечь ваше внимание в силу своей новизны, поэтому я потрачу немного времени на обсуждение того, как они работают. Это важно по нескольким причинам:

- понимание того, как работают ML Services, и преимуществ их использования позволит вам принять решение о том, подходят ли эти возможности для вас и вашего приложения;
- вы можете больше доверять ML Services для SQL Server, если знаете больше об интеграции, безопасности и управлении этими службами;
- архитектура, называемая **Extensibility Framework**, используемая для ML Services, та же, что и для так называемых языковых расширений SQL Server, новых для SQL Server 2019.

Перед прочтением оставшейся части этой главы ознакомьтесь с документацией по службам SQL Server ML Services, размещенной по ссылке https://docs.microsoft.com/en-us/sql/machine-learning/?view=sql-server-ver15.

Как это работает

До появления SQL Server ML Services специалисты по обработке и анализу данных разрабатывали и использовали свои *модели* (термин, часто используемый для программы машинного обучения) на отдельных компьютерах (рабочих станциях или серверах), где доступ ко всем данным, таким как SQL Server, осуществлялся удаленно. Во многих случаях программы на R или Python, используемые для этих моделей, просто «вытягивали» всю таблицу и фильтровали результаты в самой программе, не используя преимущества таких языков, как SQL.

Службы SQL Server ML Services предлагают новую возможность для создания и использования масштабируемых приложений машинного обучения в соответствии со следующими концепциями:

- приложения машинного обучения выполняются на том же компьютере, что и SQL Server, но в независимых от SQLSERVR.EXE процессах;
- SQL Server предоставляет интерфейс T-SQL посредством системной хранимой процедуры **sp_execute_external_script** для выполнения кода машинного обучения;
- SQL Server предоставляет архитектуру, делающую возможными интеллектуальный обмен данными и масштабируемость, используя специальный программный код для машинного обучения.

Рассмотрите рис. 5.6, иллюстрирующий архитектуру служб SQL Server ML Services, называемую расширяемой архитектурой служб машинного обучения SQL Server (Extensibility architecture in SQL Server Machine Learning Services). Эта иллюстрация приведена в документации по SQL Server,

доступной по ссылке https://docs.microsoft.com/en-us/sql/advanced-analytics/concepts/extensibility-framework#architecture-diagram.

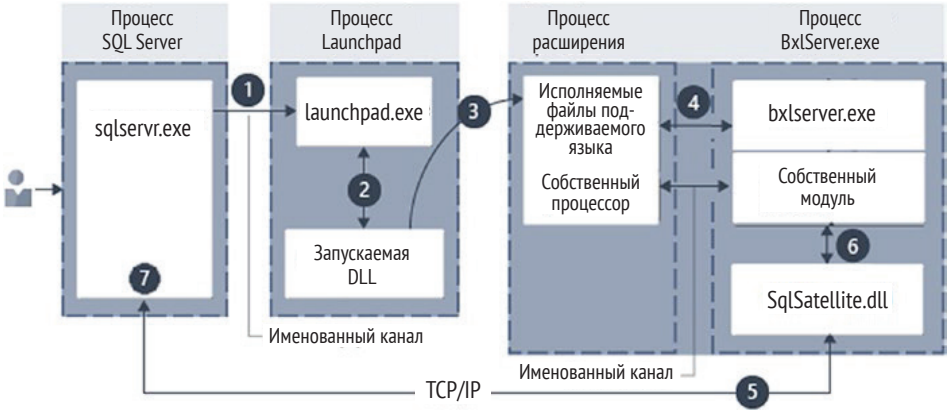


Рис. 5.6. Архитектура служб SQL Server ML Services

К сожалению, в документации не приводится подробное описание цифровых сносок, изображенных на диаграмме, поэтому я дам это описание здесь. Это поможет вам лучше понять, как работают службы SQL Server ML Services.

> **Примечание.** Я уже говорил на эту тему ранее в данной книге, и у меня есть очень подробная схема архитектуры с разъяснениями. Вы можете ознакомиться с ней на рис. 5.7, а также просмотрев материалы, опубликованные по ссылкам на www.youtube.com/watch?v=y52oBaI32Jo и www.slideshare.net/BobWard28/sql-server-r-services-what-every-sql-professional-should-know. Эти ресурсы раскрывают архитектуру SQL 2016 R Services, при этом архитектура, использующая Python, аналогична той, что описана в материалах, опубликованных по приведенным ссылкам. Обновленный слайд, иллюстрирующий поддержку Python, можно найти по адресу https://aka.ms/bobwardms. Найдите в папке SQL2017 строку с дополнительной информацией о статье, которая называется «Службы машинного обучения SQL Server» (Inside SQL Server Machine Learning Services).

Вот моя версия архитектуры при более глубоком погружении (рис. 5.7).

1. Пользователь выполняет сценарий **sp_execute_external_script**, выбрав язык программирования (R или Python), сценарий и другие параметры, такие как запрос T-SQL, для выполнения сценария. SQL Server вызывает отдельную программу с названием Launchpad (служба в Windows или демон в Linux) через именованные каналы, передавая все соответствующие данные (например, сценарий R или Python).

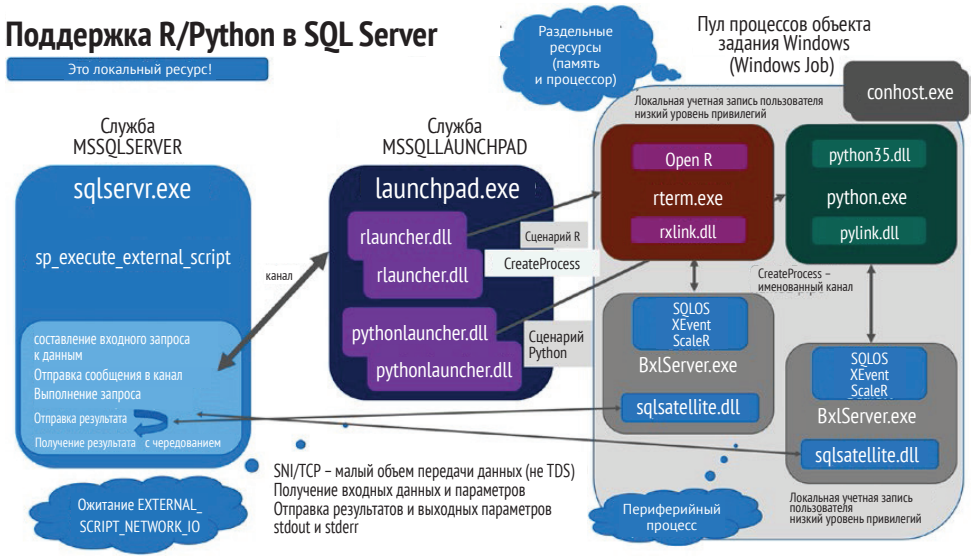


Рис. 5.7. Более глубокое погружение в службы SQL Server ML Services

2. Launchpad содержит код для выполнения DLL, соответствующей языку R или Python. Launchpad использует модель рабочего потока, аналогичную ядру SQL Server. Фактически он загружает систему SQLOS, которую SQL Server использует для служб ОС.
3. DLL-библиотека Launchpad запустит или создаст новый процесс для соответствующего языка (**rterm.exe** для R или **python.exe** для Python).
4. Другой процесс выполняет создание дочернего потока, который называется **bxlserver.exe** (часто его называют *периферийным* процессом). Этот программный модуль будет взаимодействовать с rterm.exe или python.exe для обмена данными.
5. bxlserver.exe обменивается данными с ядром SQL Server, используя частный канал TCP (это не аналогично подключению клиента к SQL Server) для получения данных из запроса T-SQL, выполняемого для передачи данных в программу на R или Python. Этот обмен данными выполняется с *чередованием*. Это означает, что механизм SQL Server может получить строки для запроса T-SQL, чтобы передать их в программный модуль машинного обучения, и в то же время получить результаты обратно. DLL, которая поддерживает такой обмен, называется **sqlsatellite.dll**.
6. sqlsatellite.dll работает с модулем в bxlserver.exe для обмена данными с rterm.exe или python.exe.
7. Все результаты (включая сообщения stdout) из программного модуля rterm.exe или python.exe передаются обратно на SQL Server через канал TCP.

В результате этого пользователь выполняет sp_execute_external_script и получает обратно результаты в виде таблицы (например, набора результатов SELECT) вместе с сообщениями stdout. У этой процедуры есть также специальные параметры оператора для выходных параметров и многое другое.

Ключевая концепция этого улучшения заключается в том, что код R или Python выполняется на том же компьютере, что и SQL Server (близко к данным), и SQL Server может эффективно обмениваться данными с программным кодом (сетевой трафик для обмена данными не используется).

Лучший способ понять, как взаимодействуют запрос T-SQL (или «входной запрос») и программа на R или Python, – это рассмотреть практический пример. Вместо того чтобы разбирать пример здесь, я настоятельно рекомендую вам использовать пример по ссылке https://aka.ms/sqldev или непосредственно для Python по ссылке https://microsoft.github.io/sql-ml-tutorials/python/rentalprediction/. Одна из причин, по которой я рекомендую вам применять этот пример, состоит в том, что он также включает в себя иллюстрацию того, как использовать **встроенную оценку** (native scoring) в T-SQL (https://docs.microsoft.com/en-us/sql/advanced-analytics/sql-native-scoring).

Мой коллега Бак Вуди (Buck Woody) также провел отличный семинар на данную тему. Я рекомендую просмотреть его, чтобы попробовать выполнить практический пример и увидеть его в действии, по ссылке https://github.com/Microsoft/sqlworkshops/tree/master/SQLServerMLServices. Что особенно замечательно в этом семинаре, так это то, что вы погрузитесь в тему интеллектуальной обработки данных (что будет полезно для тех из нас, кто, в отличие от Бака, не слишком глубоко погружен в науку о данных).

Безопасность, изоляция и функции регулирования

Одним из первых заданий, которые Джозеф Сирош поручил мне, было укрепление доверия со стороны сообщества SQL в целом к службам SQL Server R Services. Он обсуждал эти новые возможности с представителями нескольких крупных компаний, использующих SQL Server, и специалисты по данным в этих компаниях опасались запускать сценарии на R в SQL Server.

Одним из первых шагов, которые я предпринял, было разъяснение архитектуры, приведенное в предыдущем разделе и в презентации на www.slideshare.net/BobWard28/sql-server-r-services-what-every-sql-professional-should-know. Эта архитектура помогла объяснить модель изоляции служб SQL Server ML Services. Все сценарии на R и Python выполняются в отдельных процессах, изолированных от SQLSERVR.EXE, поэтому любые проблемы при выполнении этих сценариев не вызовут проблем с ядром базы данных. Эта концепция коренным образом отличается от других «расширенных» моделей SQL Server, таких как расширенные процедуры и SQLCLR, которые все запускаются внутри процесса SQLSERVR.EXE. Кроме того, периферийные процессы работают изолированно друг от друга, поэтому операции обработки

данных в R или Python различных пользователей не могут помешать друг другу. В дополнение к тому, что они выполняются как отдельные процессы, любой процесс, созданный на Launchpad, выполняется в изолированной модели с использованием концепции **AppContainer** в Windows (https://docs.microsoft.com/en-us/windows/win32/secauthz/appcontainer-isolation) и пространстве имен в Linux (https://en.wikipedia.org/wiki/Linux_namespaces).

Другая концепция, которую мне нужно было объяснить, – это безопасность. Рассмотрим модель безопасности при использовании R или Python для служб SQL Server ML Services:

- эта функция включена, только если она сперва установлена, а затем настроена с помощью sp_configure. О том, как установить службы SQL Server ML Services в Windows, можно прочитать по ссылке https://docs.microsoft.com/en-us/sql/machine-learning/install/sql-machine-learning-services-windows-install?view=sql-server-ver15, а для Linux – по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-setup-machine-learning. О параметре sp_configure вы можете прочитать по адресу https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/external-scripts-enabled-server-configuration-option;
- для системной процедуры T-SQL, sp_execute_external_script, требуется разрешение EXECUTE ANY EXTERNAL SCRIPT. Это разрешение по умолчанию предоставляется только тем пользователям или ролям, у которых имеется разрешение CONTROL, или учетным записям и ролям, обладающим разрешениями CONTROL SERVER. Любому другому пользователю или учетной записи, пытающимся выполнить сценарий R или Python, должны быть предоставлены явные разрешения;
- пользователям также потребуется разрешение для доступа к объектам, на которые есть ссылка во «входном запросе» sp_execute_external_script;
- процессы, разветвленные для выполнения R и Python (rterm.exe и python.exe), выполняются с использованием определенной учетной записи с низким уровнем привилегий. Получить дополнительную информацию об этом для Windows можно по ссылке https://docs.microsoft.com/en-us/sql/machine-learning/security/create-a-login-for-sqlrusergroup?view=sql-server-ver15. Для Linux эти программы запускаются с использованием учетной записи **mssql_satellite**;
- по умолчанию периферийные процессы не имеют доступа для подключения к сети за пределами компьютера, на котором работает SQL Server.

Третья концепция, которая дает больше уверенности в управлении выполнением модулей, написанных на R и Python, – это *регулирование*. Начиная с SQL Server 2008 SQL Server включал в себя концепцию регули-

рования, обеспечиваемую функцией под названием регулятор ресурсов (Resource Governor). Регулятор ресурсов предоставляет пользователям механизм управления ресурсами, обеспечивающими работу SQL Server, такими как ресурсы ЦП, памяти и ввода-вывода. Следовательно, регулятор ресурсов является естественным интерфейсом для управления использованием ресурсов для программных модулей ML Services.

Концепция внешнего пула ресурсов была добавлена в SQL Server, чтобы явно контролировать использование ресурсов для процессов, которые создаются с помощью sp_execute_external_script, включая rterm.exe, python.exe, bxlserver.exe и другие. В Windows внешние группы ресурсов реализуются с помощью концепции, называемой *заданиями* (*Jobs*) или объектами-заданиями (Job Objects). Вы можете получить дополнительную информацию об объектах-заданиях Windows, перейдя по ссылке https://docs.microsoft.com/en-us/windows/win32/procthread/job-objects. В Linux для управления использованием ресурсов используется концепция контрольных групп (cgroups). Вы можете прочитать больше о cgroups в статье из Википедии по ссылке https://en.wikipedia.org/wiki/Cgroups.

Внешние группы ресурсов могут не только помочь вам управлять процессором и памятью для внешних процессов; с их помощью вы также можете указать привязку к процессору. Таким образом, вы можете привязать периферийные процессы к определенному узлу или набору процессоров и сохранить привязку обработки SQL Server к другим процессорам или узлам. Это та самая архитектура, которая использовалась для достижения производительности в 1 млн прогнозов в секунду; более подробно вы можете прочитать об этом по ссылке https://cloudblogs.microsoft.com/sqlserver/2016/10/11/1000000-predictions-per-second/.

Что нового в SQL Server 2019?

Службы машинного обучения SQL Server (SQL Server Machine Learning Services) помогают усовершенствовать и модернизировать SQL Server, делая его не просто системой управления базами данных, а настоящей платформой данных. SQL Server 2019 расширяет службы SQL Server ML Services, вводя следующие новые функции:

- внешние библиотеки теперь можно устанавливать для новых пакетов R или Python с помощью оператора T-SQL CREATE EXTERNAL LIBRARY. Подробнее об этом можно прочитать по адресу https://docs.microsoft.com/en-us/sql/t-sql/statements/create-external-library-transact-sql (SQL Server 2017 поддерживал такую возможность для R);
- служба (или демон в Linux) Launchpad имеет решающее значение для архитектуры служб SQL Server ML Services. Теперь в SQL Server 2019 службы SQL Server ML Services могут быть частью экземпляра всегда включенного отказоустойчивого кластера, включая саму службу Launchpad;

- службы машинного обучения SQL Server теперь поддерживаются в Linux. Эта тема будет обсуждаться далее в главе 6;
- службы SQL Server ML Services теперь поддерживают создание и обучение моделей для сегментированных данных; для этого используются новые параметры сценария sp_execute_external_script. Дополнительная информация, включающая пример использования этой новой функции, находится по ссылке https://docs.microsoft.com/en-us/sql/advanced-analytics/tutorials/r-tutorial-create-models-per-partition.

Я думаю, что службы машинного обучения SQL Server – это инновационное решение, изменяющее привычные подходы и расширяющее границы возможного. Я попросил своего коллегу Бака Вуди (Buck Woody), специалиста по теории и методам анализа данных в Microsoft, поделиться своим мнением о значении интеграции SQL Server с машинным обучением. По словам Бака, «выполнение задач прогнозирования и категоризации в машинном обучении на SQL Server позволяет не только повысить производительность за счет размещения вычислений непосредственно над данными, но и имеет несомненные преимущества с точки зрения безопасности. SQL Server поддерживает один из самых высоких уровней безопасности в отрасли и с добавлением традиционных языков машинного обучения, таких как R и Python, а также различных библиотек прозрачно использует эту безопасность. Еще одно преимущество использования SQL Server в качестве платформы для машинного обучения заключается в том, что он предоставляет специалисту по анализу данных место для экспериментов и практической работы с рабочими задачами, а разработчик баз данных получает контроль за внедрением кода машинного обучения R и Python в Transact-SQL, что способствует эффективному разделению обязанностей».

Расширение языка T-SQL

В конце лета 2018 года я был в Редмонде (штат Вашингтон), в штаб-квартире Microsoft, и занимался презентацией, целью которой было подготовить официальное предварительное представление возможностей SQL Server 2019 на конференции Microsoft Ignite. Работая над этой презентацией, я консультировался с руководителями различных программ, чтобы убедиться, что моя презентация содержит точные данные, и просил руководителей программ подготовить демонстрации.

В числе руководителей программ, с которыми я встречался во время подготовки к выступлению, была Нелли Густафссон (Nellie Gustafsson). Нелли, среди прочего, является одним из ведущих менеджеров по службам машинного обучения SQL Server. Я говорил с Нелли о том, поддержку каких языков команда предполагала включить в SQL Server, в том числе для SQL Server ML Services в SQL Server 2019. На нашей встрече она застала меня врасплох, сказав, что следующим языком будет Java. Она пошла дальше: она сказала, что в идеале команда хотела бы открыть архитекту-

ру для ML Services с поддержкой собственного набора инструментальных средств разработки программ (Software Development Kit, SDK). Таким образом, любой, кто обладает достаточными техническими знаниями, сможет использовать свой собственный язык для расширения T-SQL, используя ту же архитектуру, которая применялась для запуска R и Python для служб SQL Server ML.

Однако в то время, когда мы выпускали CTP 2.0 для SQL Server 2019, мы решили отложить планы по включению SDK в эту версию и просто обеспечить поддержку Java в качестве третьего языка для служб SQL Server ML Services. Java не является общераспространенным и обязательным для использования языком для решения задач машинного обучения, поэтому мы запустили эту функцию с примерами, которые просто демонстрировали, как расширить T-SQL для поддержки новой функциональности (честно говоря, мы также использовали Java для демонстрации машинного обучения с кластерами больших данных).

В основу этой новой возможности положены те же концепции, что и в SQL Server ML Services. Вы должны использовать ту же системную хранимую процедуру sp_execute_external_script, но указать «Java» в качестве языка и предоставить скомпилированный класс Java. Несмотря на то что это была не полностью открытая архитектура с возможностью расширения, интеграция Java с SQL Server 2019 привлекла внимание многих разработчиков, предоставив новые возможности для работы с продуктами Microsoft.

Инфраструктура, обеспечивающая расширяемость

К тому времени, как версия SQL Server 2019 была готова к выпуску, мы решили открыть архитектуру ML Services. Мы назвали это *инфраструктурой, обеспечивающей расширяемость* (extensibility framework). Способ получить доступ к инфраструктуре, обеспечивающей расширяемость, – это языковое расширение. Мы проиллюстрируем использование этого нового фреймворка на примере Java и включим это языковое расширение, чтобы использовать его в новой версии продукта.

Чтобы сделать жизнеспособной инфраструктуру, обеспечивающую расширяемость, нам пришлось внести дополнения в существующую архитектуру SQL Server ML Services, в том числе:

- нам нужно оставить R и Python «как есть», чтобы эти языки считались «встроенными» в SQL Server. R и Python не считаются языковыми расширениями, а являются лишь частью SQL Server в качестве служб SQL Server ML;
- «средство запуска» для R и Python в службе Launchpad запускает rterm.exe и python.exe. Файл bxlserver.exe также был разработан специально для работы с R и Python. Мы создали «общий» модуль запуска на сервере Launchpad для запуска любого языка (вы увидите, что это связано с концепцией CREATE EXTERNAL LANGUAGE);

- нам нужна новая «хост»-программа для запуска других языков. Поэтому мы поставляем хост-программу под названием **Extension Host** (хост расширения). В Windows эта программа называется **exthost.exe**;
- узел расширения должен включать файл sqlsatellite.dll (или sqlsatellite.so в Linux) и обеспечивать возможность взаимодействия языкового расширения с ним для обмена данными с SQL Server.

На рис. 5.8 показана грубая картина этой архитектуры для Windows (там же есть диаграмма архитектуры Linux) из документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/language-extensions/concepts/extensibility-framework.

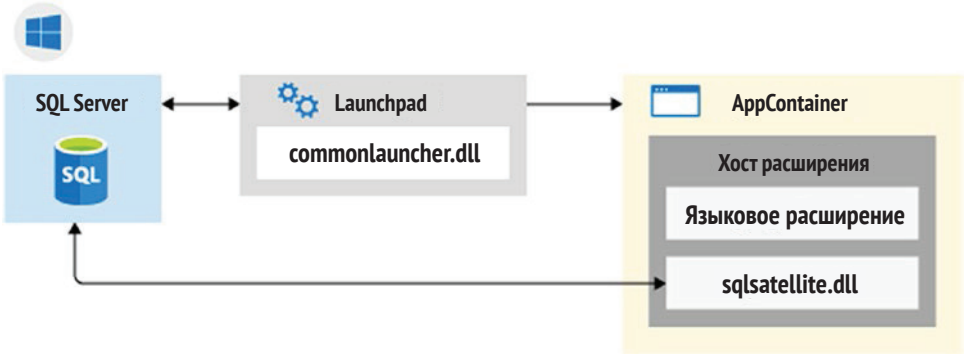


Рис. 5.8. Архитектура, обеспечивающая расширяемость для внешних языков в Windows

Теперь, используя эти новые возможности в SQL Server 2019, вы можете:

- применять sp_execute_external_script для запуска сценариев R или Python для модулей машинного обучения;
- расширять T-SQL с помощью sp_execute_external_script, обеспечивая поддержку других языков, таких как Java, при условии что вы установили языковое расширение. В SQL Server 2019 мы предоставляем все программное обеспечение для расширения T-SQL с помощью Java.

Особо заметим, что языковое расширение (которое представляет собой DLL в Windows или библиотеку совместно используемых объектов, .so, в Linux, обычно написанную на C ++) является ключевым элементом программного обеспечения для поддержки языкового расширения. Microsoft предоставляет языковое расширение для Java при установке SQL Server. Поскольку языковое расширение скомпилировано для Java, оно будет загружать виртуальную машину Java (JVM) для запуска ваших классов Java. Каким образом происходит установка JVM для запуска этого расширения? Об этом будет рассказываться в следующем разделе.

Кроме того, вам понадобится библиотека SDK для вашего языка. Как вы узнаете из следующего раздела, Microsoft предоставляет библиотеку SDK для Java. SDK будет реализовывать набор известных классов и методов, чтобы обеспечить поддержку исполнения вашего класса и вы могли обмениваться данными с SQL Server.

Расширение T-SQL с помощью Java

Вопрос, который вы, возможно, задавали себе: что представляет собой языковое расширение T-SQL? Использование R и Python для машинного обучения имеет смысл. T-SQL не имеет встроенных функций или возможностей машинного обучения. Так зачем вам нужен Java? Вы, вероятно, знакомы с термином «регулярное выражение», или regex (https://en.wikipedia.org/wiki/Regular_expression). Regex – это поиск выражений, имеющих определенную структуру, в строковых или символьных данных на основе заданного выражения. Выражение, на основе которого выполняется поиск в regex, – это очень мощный инструмент, намного более мощный, чем оператор LIKE и другие строковые функции T-SQL.

Поскольку T-SQL не обеспечивает встроенную поддержку полных возможностей регулярных выражений, вы можете создать класс Java, который поддерживает регулярные выражения, и интегрировать его в T-SQL, используя инфраструктуру расширяемости и расширение Java, поставляемое с SQL Server 2019. Так как инфраструктура предоставляет языковое расширение для возможности обмена данными с SQL Server, вы можете использовать класс Java с sp_execute_external_script для применения регулярных выражений при работе с данными, полученными на основе запроса T-SQL. Это показано в обучающих материалах, включенных в документацию, размещенную по ссылке https://docs.microsoft.com/en-us/sql/language-extensions/tutorials/search-for-string-using-regular-expressions-in-java.

Я не включил в эту книгу пошаговый пример, иллюстрирующий такую возможность; вместо этого я предлагаю вам воспользоваться указанным обучающим материалом. Я выполнил все описанные в обучающем материале действия на Windows; эти материалы также доступны и для Linux. У меня есть несколько подсказок и сценариев, которые я использовал для этого.

Эти обучающие материалы покажут вам следующее:

- как создать базу данных и образцы данных;
- как создать класс Java для реализации механизма выражений regex;
- как создать свой код, чтобы его можно было установить с SQL Server, используя SQL Server Java SDK;
- как подключить внешний язык и библиотеки для включения Java и установить ваш код. Внешний язык будет связан с DLL, поддерживающей расширение языка, или файлом .so. Внешними библиотеками будут SQL Server Java SDK и ваш код;

- как вызвать созданный вами класс Java с помощью sp_execute_external_script.

С технической точки зрения, вы можете создать свой код на отдельном компьютере, а не на том, на котором установлен ваш SQL Server. Но если вы сделаете это, вам понадобится SQL Server Java SDK с названием mssql-java-lang-extension.jar (для Windows и Linux). Один из способов установить этот SDK – поставить SQL Server с возможностью расширения Java. Поэтому я рекомендую вам работать с обучающим материалом и создавать практический пример на том же компьютере, где вы установили SQL Server. Вы также можете выполнить практический пример на своем ноутбуке, используя версию SQL Server Developer Edition, а затем установить скомпилированный вами код (который будет являться .jar-файлом) на рабочий SQL Server.

> **Примечание.** На момент написания этой книги мы опубликовали репозиторий GitHub по ссылке https://github.com/microsoft/sql-server-language-extensions для языковых расширений, включая SQL Server Java SDK. Но файл mssql-java-langextension.jar не был включен в него. Мы планируем сделать SDK доступным на GitHub, чтобы вы могли создавать свои собственные классы Java независимо от установки SQL Server.

Подготовительные шаги для использования обучающих материалов

Предварительные шаги, которые необходимо выполнить для использования обучающих материалов, описаны в документации, доступной по ссылке https://docs.microsoft.com/en-us/sql/language-extensions/tutorials/search-for-string-using-regular-expressions-in-java#prerequisites.

Одним из подготовительных шагов является установка Java Runtime Engine (JRE). Вот более радикальные вещи для вас. В SQL Server 2019 мы поставляем JRE из открытого источника Zulu Open JRE. Вот так. **Java бесплатно поставляется с SQL Server 2019!**

Экран установки в Windows для выбора вашей JRE выглядит так, как показано на рис. 5.9.

> **Примечание.** Несмотря на то что Zulu Open JRE бесплатна и поставляется с SQL Server, она полностью поддерживается Microsoft. У вас также есть возможность установить свою собственную среду исполнения (JRE). Если вы устанавливаете свою собственную JRE, вам придется выполнить несколько дополнительных шагов для настройки, которые описаны в документации по установке.

Не забудьте правильно установить переменную среды JRE_HOME в Windows и перезапустить службу Launchpad после установки SQL Server. Дополнительную информацию об этом вы можете получить по ссылке

https://docs.microsoft.com/en-us/sql/language-extensions/install/install-sql-server-language-extensions-on-windows#add-the-jre_home-variable. (Для Linux это JAVA_HOME, но процесс установки должен был автоматически сделать это.) Обратите внимание, что в руководстве приведен пример, где в качестве каталога установки указан C:\Program Files\Zulu\zulu-8\jre\, а на самом деле SQL Server размещает Azul Open JRE в каталоге C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\AZUL-OpenJDK-JRE.

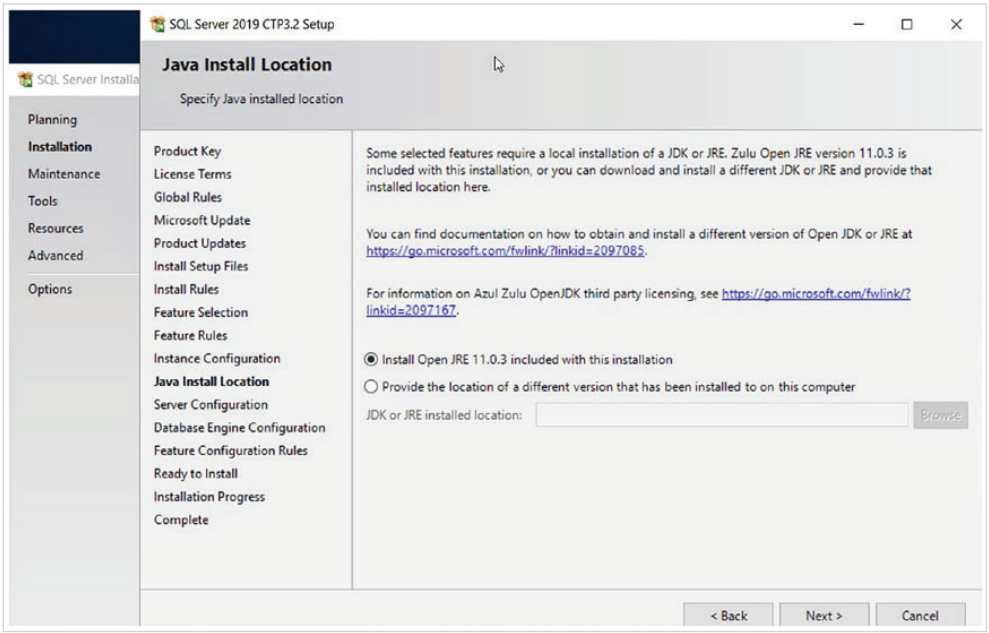


Рис. 5.9. Выбор JRE для SQL Server

В рамках процесса установки Microsoft также установит файл языкового расширения для Java. Для Windows он называется **javaextension.dll** и упакован в архивный файл **java-lang-extension.zip**. В Linux он называется **javaextension.so** и упакован в архивный файл с именем **java-lang-extension.tar.gz**. В обучающем материале показано расположение этих файлов, так как этот путь понадобится вам для подключения внешнего языка.

Теперь вы можете пройти курс из обучающего материала, чтобы создать базу данных и образцы данных, создать свой класс Java, подключить внешний язык, установить свой собственный код и затем вызвать свой класс Java.

Советы по использованию обучающих материалов

Ниже приведено несколько советов по использованию обучающих материалов. Я разместил набор сценариев, которые использовал, в каталоге **ch5_modern_development_platform\java**.

- **Выбор JDK для компиляции кода**

  В обучающем материале показан пример кода для создания класса regex и приведены инструкции по установке SQL Server Java SDK. К сожалению, обучающий материал не дает подробных инструкций относительно компиляции классов в Java. Я установил Zulu Open JDK (это можно сделать, перейдя по ссылке www.azul.com/downloads/zulu-community), чтобы при выполнении практического задания можно было использовать мой компилятор Java, javac. Поскольку я выполнял практическое задание, работая в Windows, то выбирал эти параметры, как показано на рис. 5.10.

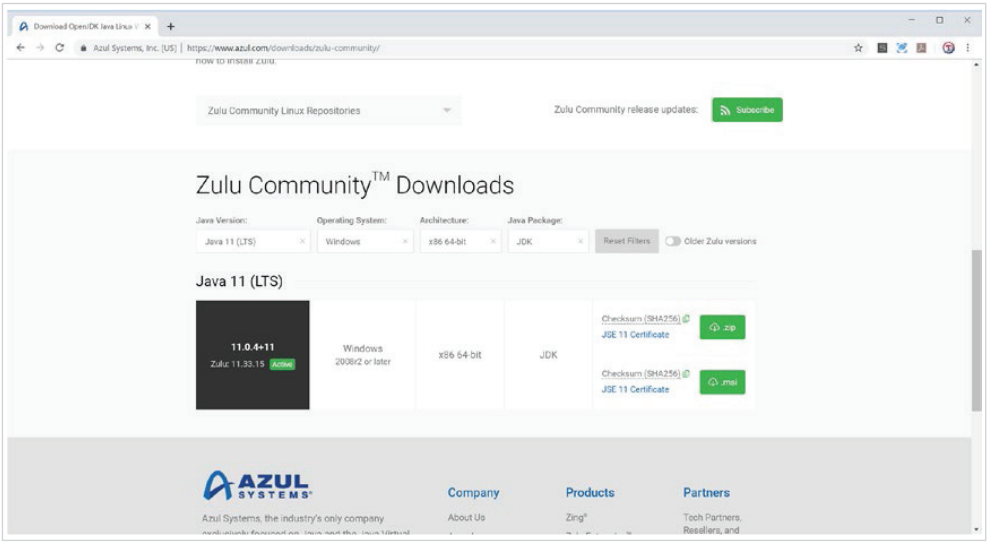


Рис. 5.10. Установка Zulu JDK для Windows

Возможно, на вашем компьютере уже установлен Java SDK. Вы можете использовать один из наиболее популярных инструментов – Visual Studio Code, IntelliJ или Eclipse. Мне просто был нужен простой способ компиляции своего Java-кода из командной строки, поэтому я выбрал Zulu JDK.

Zulu JDK поставляется в виде zip-файла, и я перетащил его в папку «Загрузки». Затем я распаковал zip-файл в текущий каталог. Я хотел использовать javac и jar, поэтому распаковал zip-файл и добавил этот каталог в системный путь: C:\Users\Administrator\Downloads\zulu11.33.15-ca-jdk11.0.4-win_x64\zulu11.33.15-ca-jdk11.0.4-win_x64\bin.

- **Компиляция кода**

  Я рекомендую следовать инструкциям, приведенным в руководстве, чтобы создать файл .jar для вашего кода, но при этом использовать концепцию пакета. Применение этого метода предполагает

наличие подкаталога pkg для вашего класса. Я подготовил сценарий buildclass.cmd, который выполнит эту работу. Он скомпилирует код RegexSample.java вместе с файлом SQL Server Java SDK mssql-java-langextension.jar. Затем он использует программу jar для сборки пакета из кода в подкаталоге pkg. (В учебном примере используется пакет, который при создании файла .jar будет применять подкаталог pkg.) Результатом всего процесса сборки является файл **sqlregex.jar**. Этот скомпилированный вами класс и будет установлен как **внешняя библиотека**.

- **Установка скомпилированного кода**

  Сценарий T-SQL **setuplanguage.sql** используется в Windows для подключения внешнего языка Java и создания двух внешних библиотек: (1) для SQL Server Java SDK и (2) для созданного вами кода.

  Важно отметить, что внешний язык и библиотеки установлены в пользовательской базе данных.

- **Запуск скомпилированного кода**

  Сценарий T-SQL **sqlregex.sql** показывает пример запуска вашего Java-класса, аналогичный показанному в учебном пособии.

  Я хочу предостеречь вас, что если вы пропустите какой-либо шаг, в том числе какой-либо из подготовительных шагов, то получите ошибку при выполнении sp_execute_external_script. И отладка ошибок в этой функции может стать весьма неприятным занятием.

Вот несколько вещей, о которых нужно помнить:

- необходимо включить *выполнение внешних сценариев*, включив соответствующий параметр конфигурации (*external scripts enabled*);
- если вы выбрали Zulu Open JRE во время установки, обязательно установите JRE_HOME при работе на Windows и перезапустите службу Launchpad;
- если вы используете свою собственную JRE, вам необходимо выполнить дополнительные шаги для получения разрешений для двоичных файлов JRE. В документации рассказывается, как это сделать в Windows и Linux;
- когда вы создаете .jar-файл для своего кода, вы должны поместить скомпилированный код (файл .class) в подкаталог с именем pkg, в котором вы создаете код. Это соглашение об использовании имени пакета (имя пакета может быть любым в вашем коде, и тогда ваша subdir должна соответствовать этому имени). В документации рассказывается о том, насколько проще выполнять все эти действия, используя среду разработки с Java (Java IDE). Я попробовал использовать IntelliJ и Visual Studio Code и в конце концов просто предпочел применять сценарии, запускаемые из командной строки, с javac и jar.

Использование других языков

Благодаря тому что мы создали инфраструктуру, обеспечивающую расширяемость, и обеспечили поддержку внешних языков, теперь в T-SQL можно использовать и другие языки программирования, отличные от Java. Java приводится лишь в качестве примера, который мы поставляем вместе с версией SQL Server 2019, чтобы позволить вам использовать расширение «из коробки», а также показать, как интегрировать другие языки программирования. Представьте, что вам доступны внешние языки платформы .Net или Go.

Ключ к этому – расширение языка. Расширение языка – это DLL в Windows или объект общей библиотеки в Linux, который понимает, как взаимодействовать с узлом расширения (Extension Host). Как только становится доступным расширение языка, набор классов SDK может быть скомпилирован на этом поддерживаемом языке. Ваш код будет использовать классы, реализованные в SDK, вместе с классом, который будет выполнен вызовом sp_execute_external_script.

Необходимый набор классов для SDK для включения расширения языка показан на примере Java по ссылке https://docs.microsoft.com/en-us/sql/language-extensions/how-to/extensibility-sdk-java-sql-server.

Кроме того, как я описал в предыдущем разделе, исходный код и документация по созданию языковых расширений будут доступны на GitHub по ссылке https://github.com/microsoft/sql-server-language-extensions.

Будет интересно посмотреть, как развиваются языковые расширения в сообществе SQL Server. Представьте себе сценарии, в которых вы не можете реализовать что-то в T-SQL сегодня, но хотели бы расширить язык и воспользоваться всеми возможностями SQL Server, включая безопасность, доступность и управление ресурсами, без того, чтобы писать дополнительный код вне T-SQL.

Все функции изоляции процессов, безопасности и управления, которые существуют для служб SQL Server ML Services, используют языковые расширения и инфраструктуру, обеспечивающую расширяемость.

Резюме

Из этой главы вы узнали, какие функции и инструменты предоставляет SQL Server 2019 для современных разработчиков. В число этих новых возможностей входят поддержка практически любого языка программирования, обновленные источники данных, графовая база данных, поддержка UTF-8, машинное обучение и расширения T-SQL. Все эти новые функции, в сочетании с и без того мощными возможностями интеллектуальной обработки запросов (Intelligent Query Processing) и оптимизацией метаданных tempdb, предоставляют необходимый инструментарий для современных разработчиков, специализирующихся на обработке данных.

Глава 6

SQL Server 2019 для Linux

В этой главе я расскажу, что нового в SQL Server 2019 для Linux. Однако если вы малознакомы с SQL Server для Linux, я начну главу с удивительной истории о том, как и почему мы включили поддержку работы в Linux для SQL Server.

История SQL Server для Linux

В октябре 2017 года наша команда разработчиков SQL Server всколыхнула индустрию, выпустив SQL Server для Linux на таких платформах, как Red Hat, Ubuntu и SUSE. Наша команда инженеров смогла вывести SQL Server на рынок, используя инновационную стратегию и архитектуру с программным обеспечением, известным как уровень абстракции платформы SQL (SQLPAL).

На рис. 6.1 показана фундаментальная архитектура SQL Server в Linux с SQLPAL.

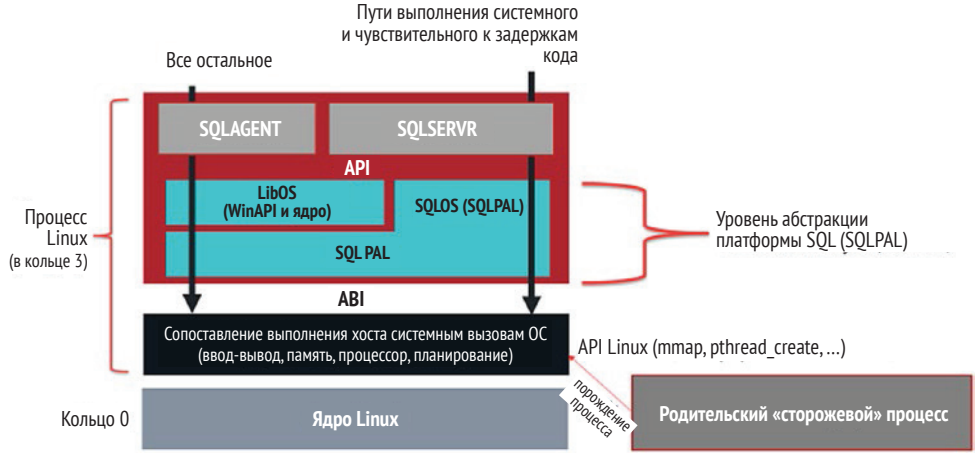


Рис. 6.1. Архитектура SQL Server в Linux

Я подробно рассказал об этой архитектуре в своей книге «SQL Server для Linux: руководство для профессионалов» (Pro SQL Server on Linux), выпущенной издательством Apress Media, поэтому не буду повторять свой рас-

сказ в этой книге (я знаю, бесстыдная реклама другой книги). Тем не менее я привожу здесь эту иллюстрацию и краткое обсуждение, позволяющие раскрыть историю выбора возможностей, обеспечивающих совместимость. Ядро СУБД SQL Server и его код созданы с использованием одной и той же базы исходного кода для обеих версий – SQL Server для Windows и для Linux. SQLPAL предоставляет программное обеспечение, обеспечивающее независимость SQL Server от платформы ОС. Это означает, что вы можете сделать резервную копию базы данных на SQL Server для Windows и восстановить ее на SQL Server для Linux, обеспечив полную совместимость.

По большей части набор функций SQL Server 2017 для Linux был таким же, как в Windows. Слава Окс (Slava Oks) однажды сказал мне: «Боб, процессор запросов – это… это процессор запросов, будь то в Windows или Linux». Он имел в виду, что один и тот же двоичный код для процессора запросов работает как в Windows, так и в Linux. Даже такие функции, как SQL Server Agent и SSIS, доступны для SQL Server в Linux.

Мы «синхронизируем» практически все версии SQL Server, стремясь максимизировать ценность того, что мы вкладываем в основную версию, учитывая сроки, необходимые для выпуска новых версий на рынок.

Хотя нам бы очень хотелось, чтобы все возможности SQL Server для Windows поддерживались в SQL Server 2017 для Linux, на стороне ядра были некоторые функции, которые мы не успели внедрить, такие как репликация (Replication) и координатор распределенных транзакций (Distributed Transaction Coordinator, DTC). Кроме того, было предпринято несколько улучшений платформы, чтобы SQL Server для Linux был таким же надежным и готовым к работе для корпоративных клиентов, как и наш SQL Server для Windows.

Прежде чем углубиться в эту тему, я должен сказать, что данная глава не содержит примеров. Возможно, вы будете удивлены, учитывая, что я написал целую книгу, посвященную SQL Server для Linux. Однако для демонстрации того, о чем рассказывается в этой главе, потребуется выполнение сложной работы по конфигурации системы; к тому же некоторые из затрагиваемых тем относятся к сценариям использования SQL Server, независимым от Linux. Теперь, когда я предупредил вас, просмотрите следующие примеры и демонстрации, посвященные SQL Server для Linux:

- **репозиторий GitHub *Pro SQL Server on Linux*** – https://github.com/Apress/pro-sql-server-on-linux;
- **демонстрации и примеры в репозитории GitHub bobsql** – https://github.com/microsoft/bobsql;
- **практические занятия Microsoft по SQL Server для Linux** – https://docs.microsoft.com/en-us/learn/paths/sql-server-2017-on-linux/;
- **сайт с материалами семинаров по SQL Server** – https://aka.ms/sqlworkshops. Хотя у меня пока нет отдельного семинара по SQL Server для Linux, не удивляйтесь, если он появится в какой-то момент.

В этих мастерских есть практические работы по репликации в SQL Server для Linux Replication с использованием контейнеров;

- наконец, в главе 7 я приведу пример, позволяющий попробовать выполнить репликацию в SQL Server для Linux , используя контейнеры.

Что нового в SQL Server 2019 для Linux

Взяв импульс при создании новой версии SQL Server для Linux, в SQL Server 2019 мы добавили несколько улучшений для решения задач профессионалов в области данных и обеспечения паритета функциональных возможностей с SQL Server для Windows, включая:

- **усовершенствования платформы и развертывания**, чтобы обеспечить корректное реагирование SQL Server для Linux на текущую доступность ресурсов и поддержку параметров развертывания, для паритета функциональных возможностей с SQL Server в Windows. Кроме того, мы работали над тем, чтобы обеспечить поддержку самой современной версии дистрибутивов Linux, которая включает в себя усовершенствования ввода-вывода в Linux, на создание которых повлияла команда разработчиков SQL Server;
- **улучшенная производительность ввода-вывода** с поддержкой постоянной памяти, позволяющая идти в ногу с достижениями в области аппаратных технологий;
- **репликация SQL Server** теперь поддерживается в Linux, облегчая процедуру и расширяя возможности синхронизации данных, – эта функция SQL Server пользуется популярностью в течение многих лет;
- **сбор данных об изменениях (Change Data Capture, CDC)** предоставляет разработчикам и специалистам по данным инструментарий для отслеживания изменений в структуре и данных таблиц в SQL Server. Эта функция остается популярной в SQL Server для Windows на протяжении многих версий; теперь она доступна и в Linux;
- **поддержка распределенных транзакций (Distributed Transactions)** в SQL Server на Linux, позволяющая разработчикам создавать приложения, использующие распределенные данные, – возможность, имевшаяся в SQL Server для Windows на протяжении многих лет;
- **упрощенное развертывание Active Directory** с использованием поставщиков OpenLDAP;
- **поддержка служб машинного обучения SQL Server и расширений языка T-SQL** в Linux для включения новых сценариев приложений, сближения моделей машинного обучения с данными с использованием безопасного и масштабируемого метода и обеспечения расширенных возможностей языка T-SQL;

- **виртуализация данных** в SQL Server для Linux, поддержка запросов Polybase к внешним источникам данных, таким как Hadoop, SQL Server, Oracle, Teradata и MongoDB, без необходимости перемещения данных.

Каждому из этих улучшений в этой главе посвящен отдельный раздел.

Улучшения платформы и процедуры развертывания

Для SQL Server 2019 мы внесли улучшения в процессор ядра СУБД и SQLPAL, чтобы SQL Server был полностью готов к работе, аналогично версии для Windows. Кроме того, мы поработали над усовершенствованием процедуры развертывания, чтобы поддерживать новые пакеты, включающие новые возможности, и предоставлять пользователям те же функциональные возможности, что и в Windows. Мы также хотим быть уверены, что SQL Server в Linux поддерживается в последних версиях Linux, таких как Red Hat Enterprise Linux 8.0, Ubuntu 18.04 и SUSE Linux Enterprise Server 15. В этих новых версиях Linux были предприняты усовершенствования ядра Linux для повышения производительности ввода-вывода и общей отказоустойчивости системы; эти усовершенствования были инициированы, чтобы удовлетворить требования к производительности ввода-вывода в SQL Server.

Улучшения платформы

Хотя ядро СУБД SQL Server создано с использованием одной и той же базы исходного кода для обеих версий – SQL Server для Windows и для Linux, в версии SQL Server для Linux включен дополнительный компонент SQL Server Platform Abstraction Layer (SQLPAL) (а также компонент, называемый расширением хоста, Host Extension), обеспечивающий взаимодействие SQL Server с ядром Linux при необходимости. После того как мы выпустили версию SQL Server для Linux, мы обнаружили, что необходимо внести в нее несколько улучшений, чтобы наша платформа баз данных работала так же, как SQL Server в Windows. Поскольку эти улучшения касались основных функций ядра СУБД, интегрированных с операционной системой, мы распространили эти изменения на версию SQL Server 2017. Если вы установите последний накопительный пакет обновления для SQL Server 2017, то заметите эти изменения.

Уведомления памяти

Система управления памятью SQL Server всегда создавалась так, чтобы она была гибкой и отвечала как потребностям в памяти SQL Server, так и требованиям вне ядра в общей среде операционной системы.

Хотя в SQL Server для Linux используется такое же ядро СУБД, как и в версии для Windows, такие понятия, как реагирование на нехватку памяти, специфичны для операционной системы. Мы обнаружили некоторые проблемы с уведомлениями памяти в SQL Server 2017 для Linux и усовершенствовали нашу интеграцию с Linux, чтобы гарантировать, что SQL Server для Linux работает так же, как SQL Server 2019 для Windows.

> **Примечание.** Мы также внесли изменения в последнее накопительное обновление, чтобы выполненные изменения были включены в SQL Server 2017 для Linux.

Выполненное усовершенствование позволит нам достичь нашей цели – установить верхний предел выделения памяти для SQL Server и обеспечить возможность его корректировки в случае нехватки памяти со стороны операционной системы. Другими словами, если в общей операционной системе недостаточно физической памяти, SQL Server уменьшит верхний предел выделения памяти, чтобы попытаться избежать подкачки памяти в ОС или вмешательства «ужасного OOMKiller» (подробнее о том, как работает OOMKiller, можно прочитать по ссылке https://unix.stackexchange.com/questions/153585/how-does-the-oom-killer-decide-whichprocess-to-kill-first).

Чтобы увидеть, как выделяемая память SQL Server корректируется с учетом нехватки памяти (т. е. нехватки физической памяти), нужно следить за значениями в столбце **ommitted_target_kb** в представлении динамического управления (Dynamic Management View, DMV) с названием **dm_os_sys_info**.

Динамические представления управления кольцевым буфером

В SQL Server есть DMV с названием **dm_os_ring_buffers**, который можно использовать для отслеживания загрузки ЦП для сервера и процесса SQLSERVR.EXE. Этот DMV официально не поддерживается, но используется для одного из ключевых инструментов мониторинга – панели мониторинга производительности (Performance Dashboard) в SQL Server Management Studio (SSMS). Более подробно о том, как использовать Performance Dashboard, рассказывается в документации по ссылке https://docs.microsoft.com/en-us/sql/relationaldatabases/performance/performance-dashboard.

Одно из удобств панели мониторинга заключается в том, что эта панель показывает данные об общем использовании ЦП компьютера и SQLSERVR.EXE (даже за последний час), чтобы помочь диагностировать проблемы с ЦП, характерные для SQL Server. Этот отчет основан на данных из dm_os_ring_buffers. Проблема в том, что в Linux мы всегда сообщали об использовании процессора как о 100%-ной фиксированной величине, поэтому в отчете не отображались правильные данные. Теперь в SQL Server 2019 (и новейшем накопительном обновлении для SQL Server 2017) этот DMV сообщает корректные данные об использовании ресурсов ЦП, и Performance Dashboard можно использовать в SQL Server для Linux.

Развертывание SQL Server 2019 в Linux

Если вы знакомы с процедурой установки SQL Server на Windows, то будете поражены простотой процедуры развертывания SQL Server на Linux. В документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-overview?view=sql-server-ver15, содержится краткое руководство для развертывания SQL Server на Linux.

В процедуру развертывания SQL Server на Linux было внесено несколько изменений, которые стоит упомянуть:

- созданы **новые пакеты** для поддержки новых функций. Одна из причин, по которой развертывание SQL Server на Linux выполняется легче и быстрее, заключается в том, что программный продукт развертывается в виде серии пакетов. В то время как пакет mssql-server включает ядро базы данных, SQL Agent, средства репликации, CDC и распределенные транзакции, для включения новых функций используются следующие пакеты:

  mssql-mlservices-mlm-py* и **mssql-mlservices-mlm-r*** – программное обеспечение для служб машинного обучения. Существуют и другие пакеты для ML Services, о которых я расскажу позже в этой главе;

  mssql-server-extensibility – программное обеспечение для включения внешних языков (поддержка инфраструктуры, обеспечивающей расширяемость);

  mssql-server-extensibility-java – программное обеспечение, обеспечивающее поддержку Java в качестве внешнего языка. Этот пакет также устанавливает пакет mssql-server-extensibility;

  mssql-server-polybase – программное обеспечение для включения функции виртуализации данных Polybase в SQL Server для Linux;

- как и в версии SQL Server для Windows, SQL Server для Linux теперь автоматически создает более одного файла данных для базы данных tempdb (их количество может варьироваться до 8) в зависимости от количества ядер процессора, обнаруженных во время установки. Эта возможность помогает избежать конфликта кратковременных блокировок страниц;
- параметры **mssql-conf**, добавленные для поддержки новых функций для SQL Server 2019. Например, были добавлены параметры mssql-conf для поддержки DTC, о которых вы можете прочитать подробнее по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-configure-msdtc.

Поддержка новых версий Linux

Мы в Microsoft считаем очень важным, чтобы SQL Server поддерживался в последних версиях основных дистрибутивов Linux. Поэтому в SQL

Server 2019 мы хотим обеспечить поддержку следующих основных версий Linux:

- Red Hat Linux Enterprise 8.0;
- Ubuntu 18.04;
- SUSE Linux Enterprise Server 15.

> **Примечание.** На момент написания данной книги мы планировали официально поддерживать эти версии Linux в SQL Server 2019. SQL Server действительно работает на всех этих версиях, но нам пришлось внести некоторые изменения в установочный пакет и убедиться, что они были хорошо протестированы. Возможно, возникнут некоторые проблемы, которые не позволят нам быть на 100 % готовыми к моменту выхода SQL Server 2019, но даже если поддержка всех этих версий не будет объявлена к этому моменту, я ожидаю, что она будет официально объявлена вскоре после этого.

Наряду с поддержкой последних версий Linux была улучшена производительность ввода-вывода. Мой давний коллега Боб Дорр (Bob Dorr) после выхода версии SQL Server 2017 заметил, что производительность ввода-вывода в SQL Server для Linux с обеспечением *общей отказоустойчивости* может оказаться одним из проблемных «узких мест». В результате мы добавили несколько параметров конфигурации SQL Server для реализации концепции, называемой «принудительная запись на диск» (forced flush), о которой рассказывается в статье, опубликованной Microsoft по ссылке https://support.microsoft.com/en-us/help/4131496/enable-forced-flush-mechanism-insql-server-2017-on-linux. По умолчанию мы решили обеспечить в SQL Server приоритет надежности над производительностью. Но, разумеется, наши клиенты хотят и того, и другого. Любой клиент может изменить значения параметров, используемых по умолчанию, и добиться и надежности, и производительности, если он знает, что используемая им дисковая система может поддерживать правильную запись на диск.

В течение 2018 года Боб Дорр (Bob Dorr) и другие сотрудники команды разработчиков SQL Server работали с командой, ведущей разработку Linux с открытым исходным кодом, особенно с командой, занимающейся разработкой версии Red Hat. Результатом этой работы стали изменения в ядре Linux для файловой системы XFS. Red Hat Enterprise Linux (RHEL) 8.0 включает эти изменения ядра. Другие версии Linux должны следовать за этими изменениями. Теперь пользователь может «отключить» наши принудительные изменения в SQL Server для Linux, но при этом достичь максимальной производительности и отказоустойчивости.

Как это часто случается со знаковыми событиями, Боб Дорр (Bob Dorr) хотел рассказать «историю про историю». И он сделал это в своей заметке в блоге, которую можно прочитать по ссылке https://bobsql.com/sql-server-on-linux-forced-unit-access-fua-internals/. Я продемонстрировал эти изменения на саммите Red Hat в мае 2019 года и показал невероятные улучшения производи-

тельности (на 100 % и более) при использовании RHEL 7.6 и RHEL 8.0, с SQL Server, сконфигурированным для применения улучшений FUA. Эта история кажется не такой уж заметной, но я призываю вас остановиться и задуматься: Microsoft помогла внести определенные улучшения в ядро Linux с открытым исходным кодом, чтобы улучшить показатели ввода-вывода для всех приложений!

Поддержка постоянной памяти

Мне очень нравится работать в команде разработчиков SQL Server – члены этой команды всегда смотрят в будущее, постоянно следят за последними достижениями в области технологий, чтобы SQL Server оставался на шаг впереди.

Меня не удивляет, что SQL Server для Linux может использовать преимущества постоянных запоминающих устройств. Постоянная память (pmem) – это запоминающее устройство с байтовой адресацией. Это означает, что к постоянной памяти можно получить доступ, как к обычной оперативной памяти, однако у этой памяти имеются все преимущества запоминающего устройства, поэтому данные, хранящиеся в ней, могут выдержать перебои с питанием и перезапуск компьютера.

Постоянные запоминающие устройства всегда можно рассматривать как устройства *блочного ввода-вывода* и в Windows, и в Linux. Другими словами, они могут быть представлены операционной системой как стандартный диск, и SQL Server может обращаться к ним, как к любому диску. В блочном режиме доступ к постоянным запоминающим устройствам может осуществляться быстрее, чем даже к некоторым самым быстрым современным SSD. Однако, поскольку устройства pmem являются устройствами с байтовой адресацией, такое приложение, как SQL Server, может передавать данные между устройством и стандартной оперативной памятью, как если бы вместо устройства использовалась память; подобная передача данных осуществляется с помощью вызовов API, таких как memcpy(), что позволяет еще больше увеличить производительность ввода-вывода.

Кроме тех улучшений производительности ввода-вывода, о которых мы уже говорили, в число улучшений SQL Server 2019 также были включены распознавание файлов базы данных и журналов транзакций, хранящихся на устройствах pmem, и обход стека ввода-вывода ядра Linux для передачи данных с этих устройств. Вы можете получить дополнительную информацию о том, как приложения могут использовать устройства pmem, используя концепцию DAX (www.kernel.org/doc/Documentation/filesystems/dax.txt), по ссылке https://docs.pmem.io/getting-started-guide/installing-ndctl.

О том, как настроить устройство pmem в SQL Server, вы можете узнать из документации, опубликованной по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-configure-pmem. Как ранее говорилось в главе 2, DELL EMC удалось добиться более высокой производительности SQL Server с помощью поддержки pmem. О результатах, полученных при тестирова-

нии этой конфигурации, вы можете узнать из материалов, размещенных по ссылке www.emc.com/about/news/press/2019/20190402-01.htm. Кроме того, инженеры HPE продемонстрировали улучшение производительности ввода-вывода в SQL Server 2019 в следующем видеоролике на YouTube: www.youtube.com/watch?v=8WUix125tQQ.

Репликация SQL Server в Linux

Репликация SQL Server (SQL Server Replication) – одна из самых популярных технологий для копирования и переноса данных между различными экземплярами SQL Server. Поскольку в SQL Server 2017 был включен SQL Server Agent, а ядро базы данных обеспечивает большую часть функций репликации SQL Server, мы хотели, чтобы эта возможность существовала в версии SQL Server 2017 для Linux. Однако время, отведенное для подготовки к выпуску новой версии, истекло, прежде чем мы смогли собрать вместе все необходимые для репликации компоненты и убедиться, что эта функция хорошо протестирована, и поэтому возможность репликации в SQL Server для Linux появилась в версии SQL Server 2019.

Продолжая традиции обеспечения совместимости версий, почти все возможности репликации SQL Server для Windows присутствуют в Linux. Список этих функций включает моментальные снимки (snapshot), транзакции (transaction), слияние (merge) и одноранговую репликацию (peer-to-peer replication). Кроме того, вы можете настроить репликацию так, чтобы использовать концепцию издателей (publishers) и подписчиков (subscribers) в Windows и Linux.

Ознакомиться с полным набором функций репликации SQL Server для Linux вы можете в документации по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-replication.

В главе 7, где речь пойдет о контейнерах, будет продемонстрирован пример использования репликации SQL Server в Linux с использованием контейнеров. В сопутствующую документацию также включены обучающие материалы, доступные по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-replication-tutorial-tsql.

Сбор данных об изменениях (Change Data Capture, CDC) в Linux

Аналогично случаю с репликацией, все компоненты для сбора данных об изменениях (Change Data Capture, CDC) входят в состав версии SQL Server для Linux. Однако мы не смогли включить эту функцию в SQL Server 2017 для Linux из-за нехватки времени. В SQL Server 2019 возможность CDC полностью поддерживается.

Если вы незнакомы с CDC, это отличная технология для сбора изменений данных в таблицах, особенно полезная для приложений, выполняю-

щих извлечение, преобразование и загрузку данных (Extract, Transform, and Load, ETL).

В SQL Server имеются все функции для отслеживания и запроса изменений. CDC использует некоторые из тех же внутренних технологий, что и модуль репликации SQL Server (SQL Server Replication), для сбора информации об изменениях в данных. Документация, в которой подробно рассказывается о CDC, доступна по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server.

DTC для Linux

После того как мы выпустили SQL Server 2017 для Linux, я спросил своего друга Боба Дорра (Bob Dorr), какими новыми направлениями он будет заниматься. Конечно, он ответил «всеми», но одна задача, которую ему поручил Слава Окс (Slava Oks), заключалась в том, чтобы поработать над обеспечением паритета функциональных возможностей для SQL Server в Linux. Одной из таких функций была поддержка распределенных транзакций, включая поддержку координатора распределенных транзакций Microsoft (Microsoft Distributed Transaction Coordinator, MSDTC).

Вместе с Капилом Такером (Kapil Thacker) и другими инженерами из нашей команды мы смогли использовать архитектуру SQLPAL, чтобы заставить основной сервис MSDTC и программное обеспечение работать на Linux, а не разрабатывать «новый» DTC для Linux с нуля. (Этот SQLPAL – прекрасная вещь. Я уверен, что когда-нибудь мы должны найти способ открыть архитектуру SQLPAL, чтобы другие разработчики могли так же просто заставить свои приложения Windows работать на Linux.)

Одним из наиболее распространенных способов использования DTC для SQL Server является распределенная транзакция между связанными серверами, от одного SQL Server к другому, которая запускается с помощью оператора T-SQL **BEGIN DISTRIBUTED TRANSACTION**. Запросы к связанным серверам, позволяющие передавать данные между различными экземплярами SQL Server, работали в версии SQL Server 2017 для Linux, однако распределенные транзакции в этой версии не поддерживались. Как заявил Боб Дорр (Bob Dorr) в сообщении в блоге, доступном по ссылке https://bobsql.com/sql-server-linuxdistributed-transactions-requiring-the-microsoft-distributed-transactioncoordinator-service-are-not-supported-on-sql-server-running-on-linux-sqlserver-to-sql-server-distributed-tr/, вы получите ошибку, если попытаетесь это сделать (подождите немного… это будет работать после установки последнего накопительного обновления на SQL Server 2017).

Если хотите узнать, как DTC работает и взаимодействует с SQL Server (или любой транзакцией XA), прочитайте подробную заметку в блоге Боба Дорра (Bob Dorr) по ссылке https://bobsql.com/how-it-works-sql-server-dtc-msdtc-and-xa-transactions/.

В то время как распределенные транзакции между связанными серверами были главной целью для команды, осуществляющей перенос DTC-

транзакций на SQL Server для Linux, существовали и другие сценарии, которые хотелось бы поддерживать. В их число входили:

- распределенные транзакции OLE-TX в SQL Server для Linux для поставщиков ODBC. Вы можете получить дополнительную информацию о создании приложений с помощью OLE-TX по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/native-client-odbc-how-to/use-microsoft-distributed-transaction-coordinator-odbc;
- распределенные XA-транзакции в SQL Server для Linux с участием провайдеров JDBC и ODBC. Более подробная информация о XA-транзакциях доступна по ссылке https://docs.microsoft.com/en-us/sql/connect/jdbc/understanding-xa-transactions.

Чтобы обеспечить поддержку всех этих возможностей, Капил, Боб и команда инженеров должны были спроектировать службу MSDTC таким образом, чтобы SQLPAL поддерживал существующую архитектуру обмена данными между портами, реализованную на настоящий момент в Windows. Получившаяся архитектура показана на рис. 6.2 (призвав на помощь Капила и Боба, мы вместе с Теджасом Шахом (Tejas Shah) создали эту схему).

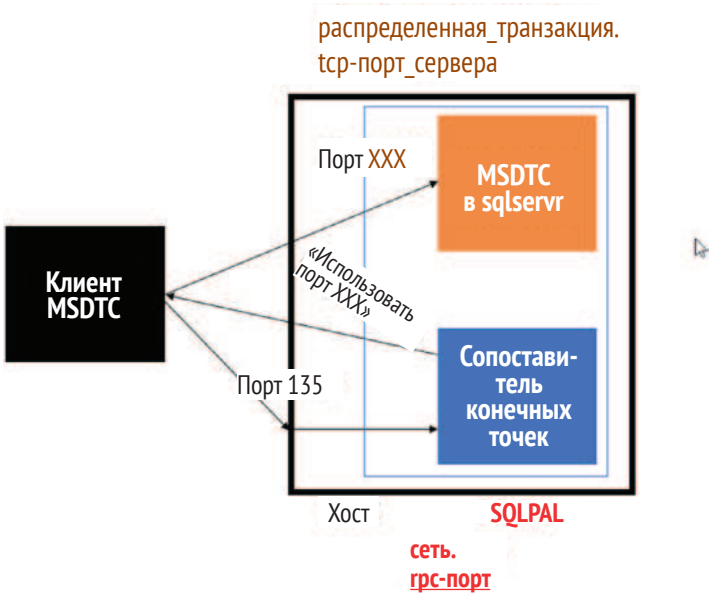


Рис. 6.2. MSDTC в Linux

Эта диаграмма определенно нуждается в пояснениях. Представьте себе, что клиент MSDTC – это распределенная транзакция SQL Server, идущая через связанный сервер (при этом другой SQL Server работает под управлением Linux или Windows). В группе, показанной на рисунке справа, в процессе Linux SQLSERVR с использованием SQLPAL работают два компо-

нента: сопоставитель конечных точек и MSDTC. Общий хост – это операционная система Linux, в которой работает процесс SQLSERVR.

MSDTC использует порт 135, и мы не сможем выбрать другой порт, если не изменим код MSDTC для Linux. Клиент MSDTC сначала пытается установить связь через порт 135. Мы создали «устройство сопоставления конечных точек», которое сопоставит порт 135 с портом, который мы можем прослушивать. Это настраивается при помощи параметра конфигурации (mssql-conf) **network.rpcport**. Затем сопоставитель конечных точек сообщит клиенту MSDTC, какой порт использовать для связи со службой MSDTC Linux, которая затем интегрируется с SQL Server. Порт для службы MSDTC может быть сгенерирован случайным образом, но вам необходимо обеспечить доступ брандмауэра к этому порту, поэтому вы должны настроить его, используя параметр конфигурации (mssql-conf) **distributedtransaction.servertcpport**.

Полная информация об этих параметрах конфигурации доступна по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-configure-msdtc. После того как вы выполните все настройки, вы можете просто запустить BEGIN DISTRIBUTED TRANSACTION на связанном сервере SQL Server. Я включил в эту книгу пример, позволяющий попробовать сделать это с использованием контейнеров. Чтобы ознакомиться с этим примером, перейдите по ссылке https://github.com/microsoft/sql-server-samples/tree/master/samples/containers/dtc.

Active Directory и OpenLDAP

Чтобы завоевать доверие представителей корпоративного бизнеса, в SQL Server для Linux нужно было поддерживать аутентификацию Active Directory (AD). Как и в случае других аспектов работы с SQL Server в Linux, хотя *процесс конфигурации отличается* от Windows, но *опыт и совместимость одинаковы*.

Мы приводим обзор процесса установки для Linux в документации по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-active-directory-authoverview, и я рассказываю об этом в главе 7 моей книги «SQL Server для Linux: руководство для профессионалов» (Pro SQL Server on Linux), выпущенной издательством Apress Media.

Одним из шагов по настройке поддержки AD для SQL Server в Linux является подключение сервера Linux, на котором размещен SQL Server, к домену Active Directory. Когда мы выпустили SQL Server 2017 для Linux, мы рассказали, как это сделать, используя пакет Linux под названием SSSD и программу под названием realmd. Мы слышали отзывы клиентов о том, что им нужны альтернативные методы для подключения к домену, в частности более простой интерфейс, позволяющий использовать сторонние пакеты, такие как PBIS, VAS или Centrify. Оказывается, SQL Server никак не запрещает использовать эти пакеты; нам просто нужно было выполнить несколько небольших изменений в конфигурации, чтобы все заработало. Мы обрисовываем весь процесс в общих чертах в документации, раз-

мещенной по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-active-directory-joindomain. Теджас Шах (Tejas Shah) совместно с командой инженеров потратили некоторое время на обновления всей документации для включения этих возможностей. Важно знать, что это не новая возможность SQL Server 2019, поскольку все будет также работать и в SQL Server 2017. Однако концепция достаточно нова, и я хотел бы подробнее рассказать о ней в этой книге.

Службы машинного обучения SQL Server и расширяемость в Linux

Как подробно рассказывалось в главе 5, службы машинного обучения SQL Server позволяют объединить возможности R и Python, интегрированных с SQL Server, для создания масштабируемых и мощных моделей и приложений машинного обучения.

Это было революционное изменение для SQL Server для Windows, и нам нужно было также перенести эту технологию в Linux. Кроме того, поскольку мы представили новую инфраструктуру, обеспечивающую расширяемость, и языковые расширения, в число которых входило расширение для Java, нам нужно было убедиться, что все эти новые возможности также доступны в Linux.

Развертывание служб SQL Server ML в Linux

Как и в SQL Server для Windows, для машинного обучения SQL Server в Linux мы облегчаем установку необходимых пакетов для развертывания сценариев R и Python с SQL Server.

Мы предлагаем различные варианты развертывания служб SQL Server ML – минимальный, полный или комбинированный. Ниже приводится описание каждого из этих вариантов:

- **полный (full)** – содержит все пакеты для R или Python и включает в себя предварительно подготовленные модели для использования в задачах машинного обучения. Этот пакет для языка R называется **mssqlmlservices-mlm-r**, а для Python – **mssql-mlservices-mlm-py**. Когда вы выбираете полный вариант, при установке этих пакетов также устанавливаются все зависимые пакеты (например, R Open);
- **минимальный (minimal)** – содержит все пакеты для R или Python, но не включает предварительно обученные модели. Этот пакет для языка R называется **mssql-mlservices-packages-r**, а для Python – **mssql-mlservices-packages-py**. Когда вы выбираете минимальный вариант, при установке этих пакетов также устанавливаются все зависимые пакеты (например, R Open);
- **комбинированный (combo)** – позволяет установить SQL Server 2019 (ядро базы данных) со службами SQL Server ML Services за один

шаг. Вы можете прочитать о том, как это сделать, по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-serverlinux-setup-machine-learning#install-all.

После выбора варианта установки полной версии SQL Server ML Services для R результат установки этих пакетов выглядит так, как показано на рис. 6.3.

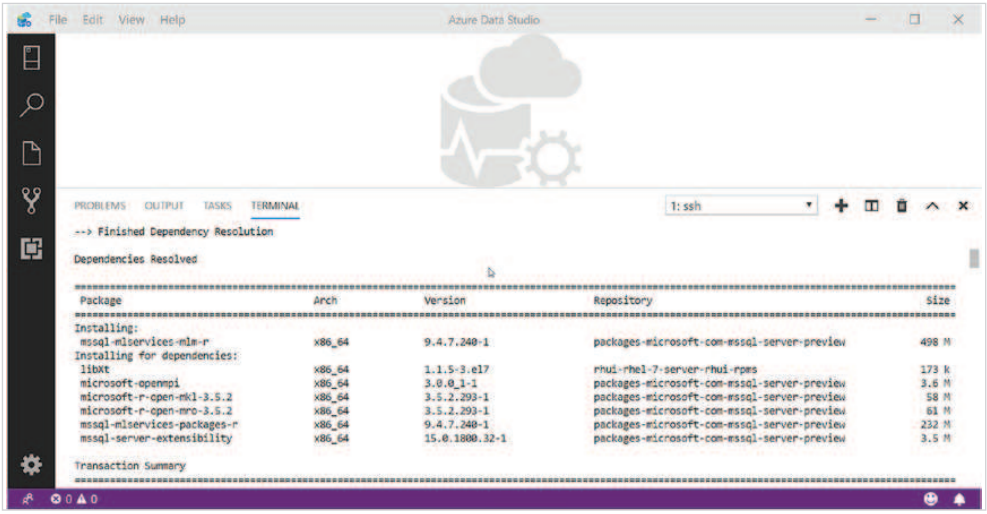


Рис. 6.3. Установка полной версии SQL Server ML Services для R

Совет. Если вы хотите использовать ssh для вашего SQL Server в Linux, то можете воспользоваться службой Terminal в Azure Data Studio. Как это сделать, будет показано в практических примерах из этой главы.

Обратите внимание, что в числе установленных пакетов имеется пакет **mssql-server-extensibility**, о котором я расскажу позже в этой главе; он – часть инфраструктуры, поддерживающей расширяемость, необходимой для языковых расширений (одна и та же среда используется как для служб SQL Server ML Services, так и для языковых расширений).

После установки всех пакетов вам нужно будет выполнить еще несколько дополнительных шагов, завершающих процесс установки, в том числе принять условия лицензионного соглашения для R или Python. Выполните действия, описанные в документации по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-setup-machinelearning#post-install-config-required.

Я также рекомендую, как и для Windows, использовать пример «Привет, мир», чтобы убедиться, что установка прошла успешно. Пример для R выглядит следующим образом:

```
EXEC sp_execute_external_script
@language =N'R',
@script=N'
```

```
OutputDataSet <- InputDataSet',
@input_data_1 =N'SELECT 1 AS hello'
WITH RESULT SETS (([hello] int not null));
GO
```

Также возможно, что вам как специалистам по обработке данных понадобятся дополнительные библиотеки R или Python для ваших приложений. Чтобы узнать, как их установить, обратитесь к документации по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-setup-machine-learning#add-more-rpython-packages.

> **Примечание.** Если вы использовали эту функцию со сборками одной из предварительных версий выпуска SQL Server 2019 SQL Server 2019 (CTP), обязательно удалите все эти пакеты, прежде чем пытаться использовать функцию окончательной версии ML SQL Server 2019 (RTM).

Как это работает

В главе 5 мы рассматривали архитектуру служб SQL Server ML Services, включая службу Launchpad и вспомогательные процессы.

В основу служб SQL Server ML Services в Linux положена та же концепция. В Linux процесс Launchpad представляет собой системный сервисный модуль с названием **mssql-launchpadd**. Вы можете просматривать или контролировать работу этого сервиса с помощью systemctl. На рис. 6.4 показан пример состояния этого сервиса в Linux.

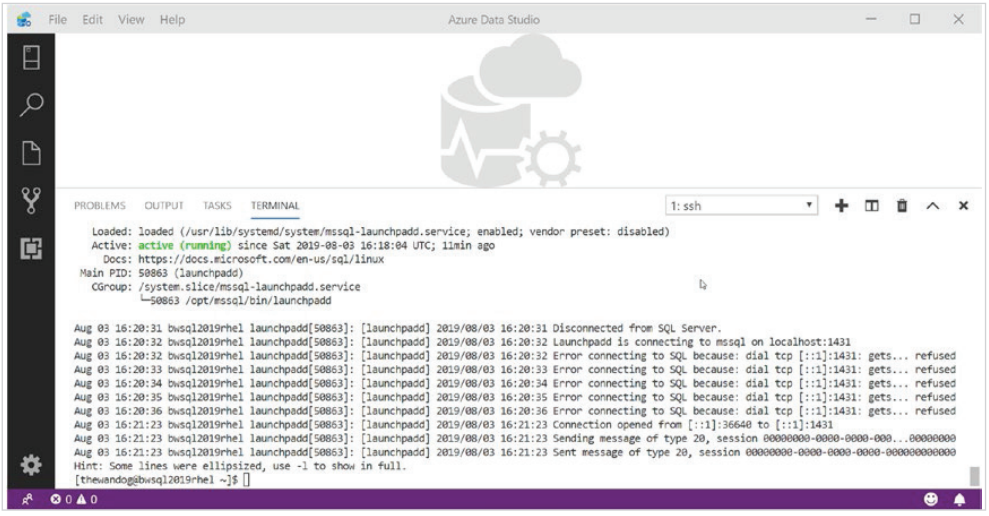


Рис. 6.4. Панель запуска в Linux

Служба Launchpad представлена в Linux процессом-демоном, называемым **launchpadd**. Та же концепция разветвления процессов в Windows для

спутниковых процессов работает в Linux, включая программу R и bxlserver. На рис. 6.5 показаны процессы, порожденные launchpadd для запуска сценария R.

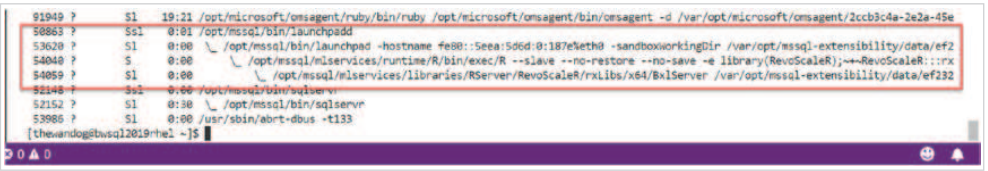


Рис. 6.5. Вспомогательные процессы для сценария R на SQL Server в Linux

Вы можете спросить, как мне удалось «поймать» эти процессы «на лету»? Чтобы сделать это, выполните следующие действия: из другого сеанса ssh запустите следующий сценарий T-SQL:

```
EXEC sp_execute_external_script
@language =N'R',
@script=N'
OutputDataSet <- InputDataSet
Sys.sleep(10)',
@input_data_1 =N'SELECT 1 AS hello'
WITH RESULT SETS (([hello] int not null));
GO
```

Обратите внимание на вызов Sys.sleep(), приостанавливающий выполнение сценария R.

В другом сеансе ssh выполните следующие команды из оболочки Linux:

```
ps -axf
```

Службы SQL Server ML (и инфраструктура, поддерживающая расширяемость) используют пространства имен, позволяющие изолировать процессы, для изоляции вспомогательных процессов. Вы можете использовать приведенный пример ранее, чтобы заставить сценарий R приостановить работу, и запустить команду в Linux:

```
sudo lsns
```

Вы увидите отдельное пространство имен, созданное для процесса launchpad (от которого порождаются вспомогательные процессы). На рис. 6.6 показан пример этого отдельного пространства имен.

На рис. 6.6 показано еще несколько важных деталей, относящихся к работе служб SQL Server ML Services (и инфраструктуры, поддерживающей расширяемость). Учетная запись пользователя, под которой работает вспомогательный процесс, называется mssql_s satellite. Это важно иметь в виду для того, чтобы предоставить необходимые разрешения для сценариев R или Python (и языков расширения).

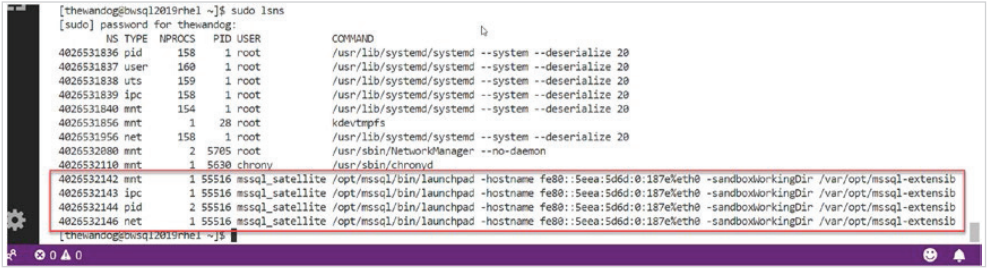


Рис. 6.6. Пространство имен для вспомогательных процессов в Linux

Примечание. Не забывайте, что функция оценки (скоринга) SQL Server встроена в ядро SQL Server, поэтому ее можно использовать в SQL Server в Linux (и даже в SQL Server 2017). Более подробная информация об этом приведена по ссылке https://docs.microsoft.com/en-us/sql/advanced-analytics/sql-native-scoring.

Границы расширяемости и языковые расширения

Основываясь на той же платформе, что и SQL Server ML Services, мы представили концепцию языковых расширений, которую я подробно описал в главе 5. Мы включили в поставку версии SQL Server поддержку языка Java в качестве примера языкового расширения.

Чтобы развернуть языковые расширения SQL Server в Linux, вы можете установить один из следующих пакетов:

- **mssql-server-extensibility** – это основное программное обеспечение, которое использует платформу расширяемости для любого языка. Это зависимый пакет, как вы видели ранее в данной главе, установленный совместно с SQL Server ML Services;
- **mssql-server-extensibility-java** – этот пакет устанавливает инфраструктуру, поддерживающую расширяемость, расширение языка Java и SDK, чтобы вы могли запустить свой код Java.

Как и в случае SQL Server ML Services, у вас также есть возможность выполнить комбинированную установку SQL Server с языковыми расширениями. Об этом способе установки вы можете прочитать по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-setup-language-extensions.

Кроме того, SQL Server установит Java Runtime Engine (JRE) версии 8, если он еще не установлен в Linux (как и при установке SQL Server ML Services на Windows, мы установим JRE Zulu).

Процесс развертывания вашего Java-класса практически идентичен аналогичной процедуре для SQL Server в Windows; вы можете просмотреть описание использования обучающих материалов по регулярным выражениям, приведенное в главе 5. На самом деле язык Java обладает широкими возможностями совместимости, поэтому вы можете использовать тот же

класс Java, который вы создали в главе 5, и скомпилировать/собрать JAR-файл в Linux. Вы используете те же шаги для подключения внешнего языка, внешней библиотеки для SDK и внешней библиотеки для вашего Java-класса (в виде файла JAR).

Когда вы запускаете свой код, та же архитектура Launchpad используется для разветвления процесса для обработки вспомогательных процессов. Как и SQL Server в Windows, этот процесс называется Extension Host и выглядит так, как показано на рис. 6.7, где приведены данные о процессах в Linux.

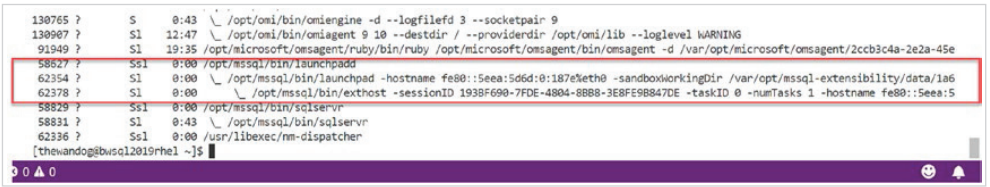


Рис. 6.7. Вспомогательный процесс exthost, используемый при запуске Java для SQL Server в Linux

Polybase в Linux

Материалы в книге трудно расположить в правильном порядке. Глава 9 этой книги посвящена концепции виртуализации данных и Polybase в SQL Server 2019. Я хотел бы вкратце упомянуть об этом здесь, поскольку поддержка Polybase – новая возможность в SQL Server для Linux.

Поддержка Polybase появилась в SQL Server 2016. Она используется для коннекторов Hadoop, а в SQL Server 2019 была дополнена коннекторами для SQL Server, Oracle, Teradata и MongoDB.

Как узнаете из главы 9, Polybase использует некоторые компоненты нашей системы аналитической платформы (Analytics Platform System, APS, ранее известной как Parallel Data Warehouse) для выполнения обработки запросов с горизонтальным масштабированием. Они существуют как службы в Windows и используют SQLPAL для SQL Server 2019 в Linux.

Мы представили пакет **mssql-server-polybase**, обеспечивающий поддержку Polybase для SQL Server в Linux. Вы можете получить подробные инструкции по его развертыванию, перейдя по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-linux-setup.

В Polybase в SQL Server для Linux используются те же самые концепции и операторы T-SQL, что и в SQL Server для Windows; однако в SQL Server 2019 существует несколько различий между версиями SQL Server для Linux и Windows:

- Polybase для SQL Server в Linux не поддерживает масштабируемые группы;
- базовый коннектор ODBC не поддерживается.

Наши планы на будущее состоят в том, чтобы обеспечить паритет функциональных возможностей для Polybase в SQL Server для Linux и SQL Server для Windows после выхода версии SQL Server 2019. Подробнее о том, как использовать Polybase, вы узнаете в главе 9.

Резюме

SQL Server в Linux – это история о выборе и совместимости. В SQL Server 2017 ядро базы данных для Linux было аналогично ядру SQL Server для Windows. В SQL Server 2019 мы существенно расширили список функциональных возможностей, общих для этих версий, включив в него репликацию, сбор данных об изменениях, распределенные транзакции, службы машинного обучения и языковые расширения, а также Polybase.

Кроме того, мы внесли улучшения в интеграцию платформы с Linux, обеспечили поддержку последних версий Linux, улучшили производительность ввода-вывода и отказоустойчивость, добавили поддержку постоянной памяти и пояснили, как используются поставщики OpenLDAP для настройки аутентификации Active Directory.

Глава 7

SQL Server и контейнеры

Зачем нужны контейнеры в SQL Server?

Когда я планировал содержание этой главы, то думал только о новых функциях контейнеров в SQL Server 2019. Хотя про эти новые возможности стоило бы написать целую главу, я все же решил расширить ее содержание и поговорить о концепции контейнеров, об их важности как нового способа развертывания SQL Server и, конечно же, о новых возможностях контейнеров в SQL Server 2019. Я мог бы написать целую книгу о контейнерах, дающую исчерпывающие знания по этому предмету. Есть много замечательных ресурсов, посвященных контейнерам (один из которых, www.docker.com, я считаю очень полезным!), которые могут дополнить эту главу. Цель данной главы – предоставить вам информацию в таком объеме, чтобы вы могли увидеть, как работают контейнеры с SQL Server и почему и где вы должны их использовать. Я уверен, что эта глава даст вам понимание того, как развертывать, управлять и использовать контейнеры с SQL Server.

Если вам кажется, что вы достаточно хорошо знакомы с концепцией контейнеров, то можете перейти к разделу «Что нового в SQL Server 2019». После этого раздела в главу включена серия примеров, позволяющих более глубоко погрузиться в тему контейнеров. Я должен сказать, что если вы знакомы с основами контейнеров, то можете начать с раздела «Как работают контейнеры с SQL Server», в частности с подраздела «Контейнер SQL Server», поскольку я расскажу о некоторых особенностях использования контейнеров с SQL Server.

Контейнеры решают проблему, которую виртуальные машины не позволяют решить сегодня. Виртуальные машины – это удивительная технология, позволяющая *абстрагировать приложения от базовой аппаратной платформы*, но они требуют загрузки и запуска всей операционной системы, для того чтобы ваше приложение могло работать. Виртуальные машины позволяют приложениям запускаться изолированно друг от друга на хост-системе, и для SQL Server это было отличным решением для сценариев консолидации (даже несмотря на то, что SQL Server допускает размещение нескольких экземпляров на одном компьютере). Контейнеры обеспечивают ту же концепцию изоляции, но они намного легче, чем виртуальные машины. Контейнеры часто считаются отдельным уровнем абстракции от операционной системы.

Вот важная концепция, которую нужно помнить, когда речь идет о контейнерах. Контейнеры не заменяют виртуальные машины. Контейнеры *дополняют* их. Фактически одна из самых распространенных сред для запуска контейнеров – это виртуальная машина. Позвольте мне дать определение контейнера, сначала определив *образ контейнера*. Образ контейнера представляет собой двоичный файл, который описывает набор файлов, организованных в файловой системе, и **программу**, запускаемую из этих файлов. *Контейнер* является экземпляром программы, запускаемой *изолированным способом* в образе контейнера вместе с файловой системой.

Рассмотрим следующую схему на рис. 7.1, которую я часто использую, чтобы пояснить, что такое контейнеры и почему они могут решать определенные задачи для современных приложений.

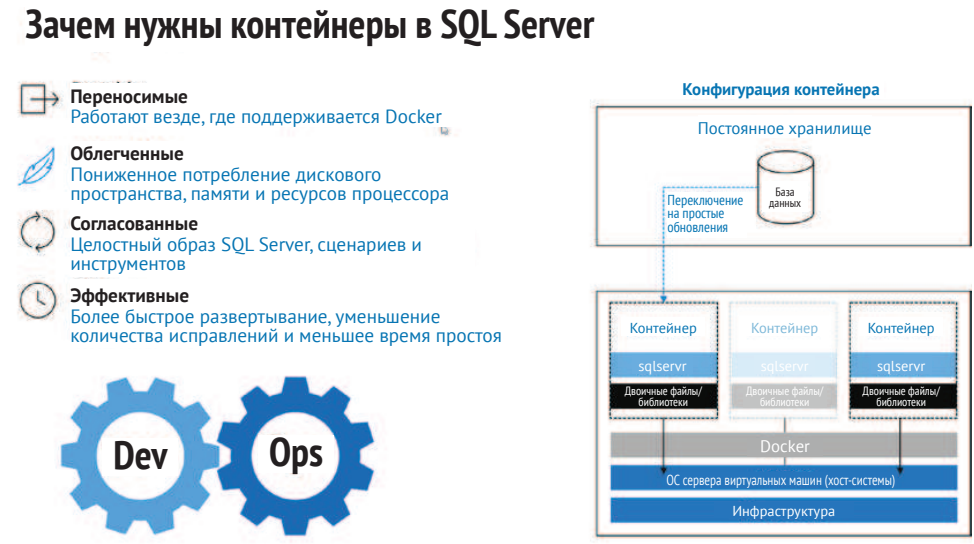


Рис. 7.1. Концепция контейнеров SQL Server

Ниже приведены комментарии к этому рисунку, начиная с его левой стороны:

Переносимые

Контейнеры являются переносимыми, потому что образ контейнера можно запускать везде, где можно запустить Docker, то есть практически везде, включая компьютеры, работающие под управлением ОС Windows, Linux и macOS, а также в облачных системах, поддерживающих эти операционные системы или Kubernetes (о Kubernetes рассказывается в главе 8). Вы можете взять образ контейнера SQL Server в виде двоичного файла и поместить его в любую из систем, где он будет работать точно так же.

Примечание. Для пользователей macOS: просмотрите, пожалуйста, мою заметку в блоге, где показано, как теперь можно запускать SQL Server и инструменты для работы с ним без установки Windows: https://bobsql.com/take-the-sql-server-mac-challenge/.

Облегченные

Контейнер – это запущенный экземпляр образа контейнера, и в конце концов, это всего лишь процесс, представляющий собой приложение, работающее изолированно. Это делает его намного более легким, чем виртуальная машина, предназначенная для размещения приложения. Требования контейнеров к ресурсам также оптимизированы, потому что если вы запускаете более одного контейнера из образа, часть файлов образов (называемых *читаемым* слоем, или слоем, доступным для чтения, о котором я расскажу далее в этой главе) распределяется между контейнерами. Это уменьшит объем ресурсов, необходимый для запуска нескольких экземпляров SQL Server на хост-системе или виртуальной машине.

Примечание. В Linux не поддерживается запуск нескольких экземпляров SQL Server, однако вы все же можете запустить несколько экземпляров, используя контейнеры.

Согласованные

Это один из аспектов контейнеров, который мне очень нравится, и он помогает решить огромную проблему для SQL Server. В течение многих лет во многих компаниях сложилась практика, когда администраторы устанавливали отдельные серверы с SQL Server, формируя платформы для тестирования и разработки. Это может вызвать сложности, так как несколько разработчиков, использующих один и тот же SQL Server (при этом им обычно требуется более высокий уровень доступа к SQL Server), могут создать множество проблем для администраторов баз данных.

Контейнеры предоставляют разработчикам согласованный способ использования SQL Server, при этом отказавшись от обязательного совместного использования экземпляра SQL Server. Например, если вы хотите, чтобы все разработчики использовали определенный образ версии SQL Server вместе с определенной базой данных, то можете просто создать образ контейнера. А поскольку контейнеры переносимы, и SQL Server теперь работает на Linux, вы можете предоставить один и тот же образ контейнера SQL Server для разработчиков, использующих разные платформы. Разработчики macOS могут использовать тот же образ SQL Server, что и разработчики Windows. Это великолепная история о согласованности! Символ DevOps в нижнем левом углу рисунка просто показывает, насколько важна концепция контейнеров для поддержки модели DevOps.

Эффективные

Контейнеры предоставляют новые эффективные возможности для обновления программного обеспечения, в том числе SQL Server. Если вам когда-либо приходилось устанавливать обновления для SQL Server, вы оцените то, как просто это делается при помощи контейнеров. Вы сможете существенно сократить время простоя и при необходимости быстрее откатывать обновления.

На рис. 7.1 с правой стороны представлена схема работы контейнеров. Контейнер в середине, выделенный серым цветом, представляет собой остановленный контейнер SQL Server. Контейнер слева – это новая версия SQL Server, которая запускается, но указывает на те же системные и пользовательские базы данных, размещенные в *постоянном хранилище* (при этом используется концепция *томов*, о которой я расскажу позже в данной главе). Эту технику, которую я покажу вам позже здесь, я называю «переключением» контейнеров, позволяющим применить или откатить накопительное обновление SQL Server.

Пока вы рассматриваете рисунок, я хочу сделать несколько замечаний:

- черные прямоугольники с надписью **Двоичные файлы/библиотеки** представляют собой двоичные файлы, необходимые для запуска SQL Server. Они иллюстрируют аспект «облегченности» контейнера, отличающий его от виртуальной машины. И что не показано на рисунке, так это то, что если все контейнеры созданы на основе одного и того же образа, эти файлы совместно используются всеми контейнерами;
- элемент с надписью **Docker** обозначает программное обеспечение Docker, которое используется для запуска и управления контейнерами. На самом деле это должно означать, что Container Runtime, или Docker, является просто примером среды выполнения контейнера;
- обратите внимание на стрелки, *проходящие* через Docker к **операционной системе хост-узла**. Docker не является слоем между контейнером и операционной системой хост-узла. Другими словами, SQL Server не нужно взаимодействовать через некоторый отдельный уровень для выполнения операций ОС ядра. Вот почему еще контейнеры считаются более легкими – поскольку они содержат программное обеспечение, напрямую взаимодействующее с операционной системой хост-узла. Уникальной особенностью контейнеров является то, что они работают изолированно друг от друга, отсюда и появился термин «контейнер».

Теперь, когда вы познакомились с основными концепциями работы контейнеров, давайте перейдем к изучению работы контейнеров. Понимание того, как работает та или иная технология, позволяет вам использовать ее более эффективно.

Как работают контейнеры с SQL Server

Прежде чем вы начнете использовать контейнеры с SQL Server, давайте разберемся, как они работают. Чтобы объяснить это, мне нужно потратить некоторое время на объяснение концепций *размещения контейнеров*, магии, стоящей за Docker, и *жизненного цикла контейнера*.

Размещение контейнеров

В предыдущем разделе я упоминал о том, что контейнеры – это программные модули, которые взаимодействуют непосредственно с операционной системой хост-узла. Операционная система хост-узла может находиться на виртуальной машине или непосредственно на аппаратной платформе.

Поскольку наша программа в контейнере взаимодействует напрямую с операционной системой хост-узла, так же как и любая другая программа в системе (за исключением того, что, разумеется, контейнеры запускаются специальным изолированным способом), она должна быть скомпилирована и запущена на этой операционной системе.

Когда мы выпустили версию SQL Server 2017 для Linux, мы также представили образы контейнеров SQL Server на основе Linux, а именно Ubuntu Linux. Почти каждый образ контейнера, существующий в мире, создан на базе операционной системы хост-узла, и абсолютное большинство из них использует Linux. Для хост-узлов, работающих на Linux, запуск контейнера SQL Server на Linux не является проблемой. Контейнеры с SQL Server в Linux могут использоваться независимо от того, установлена система Linux непосредственно на аппаратной платформе или на виртуальной машине.

А как насчет Windows и macOS? Ключевой концепцией здесь является Docker как среда выполнения для контейнера. Docker поддерживает контейнеры, работающие в Windows, посредством Docker Desktop для Windows. Любой контейнер, основанный на образе Linux, будет запускаться в контексте виртуальной машины с Linux, называемой **DockerDesktopVM**. В Docker Desktop для macOS используется концепция HyperKit (https://github.com/moby/hyperkit). Вы можете прочитать больше о Docker Desktop на сайте www.docker.com/products/docker-desktop.

В последнее время были достигнуты определенные успехи в реализации контейнеров с версией Docker для Windows благодаря поддержке концепции, называемой Linux Containers for Windows (LCOW). Команда Windows описывает эту концепцию как более легкий метод запуска контейнеров Linux в Windows, чем полноценная виртуальная машина. Вы можете прочитать больше о LCOW по ссылке https://docs.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/linux-containers.

Кроме того, Docker Desktop для Windows может использовать оптимизированный метод с использованием новой подсистемы Windows для Linux

(WSL). Дополнительную информацию об использовании WSL2 с Docker Desktop можно получить по ссылке https://engineering.docker.com/2019/06/docker-hearts-wsl-2/.

А как насчет контейнеров на основе образов для Windows или macOS? Если можно создать образ контейнера на основе операционной системы Linux, существуют ли образы контейнеров на основе Windows или macOS? Для Windows ответ на этот вопрос – да. В Windows существует концепция образа контейнера, основанного на Windows. Вы можете узнать больше о контейнерах Windows, перейдя по ссылке https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/. SQL Server пока не поддерживает версию контейнера Windows. Но летом 2019 года мы анонсировали закрытую программу предварительного тестирования SQL Server Windows Container. Я расскажу больше о контейнерах SQL Server Windows Containers в конце этой главы. На момент написания данной книги я еще не видел образов контейнеров на основе macOS. Вообще говоря, версий SQL Server специально для macOS не существует, но, как я уже говорил, SQL Server поддерживает Linux, который будет работать с использованием Docker Desktop для macOS (с применением HyperKit).

Docker – это магия?

Все, о чем до сих пор говорилось в этой главе, кажется неким волшебством для любого, кто знаком с компьютерными системами. Каждый раз, когда используется слово «контейнер», сразу же упоминается и Docker. Концепция виртуализации операционной системы, которая лежит в основе контейнеров, не нова (для получения дополнительной информации обратитесь к статье https://en.wikipedia.org/wiki/OS-level_virtualisation). Все, кто работал вместе со мной, знают, что когда я исследую, как что-то работает, я всегда задаю вопрос «с помощью какого API?». Другими словами, какой интерфейс программирования используется для достижения цели? Для контейнеров ответ на этот вопрос – используются API, предоставляемые операционной системой ядра.

Docker (как и другие среды выполнения контейнеров на рынке) принял концепцию контейнеров и создал платформу и экосистему, которые используются повсеместно. Но сам Docker использует возможности операционной системы для поддержки контейнеров (в Linux аналогичные концепции применяются к контейнерам Windows). Поддержка контейнеров включает в себя следующие основные понятия:

пространства имен – обеспечивают механизм, позволяющий запускать процессы изолированно друг от друга. Дополнительную информацию о пространстве имен вы можете получить по ссылке https://en.wikipedia.org/wiki/Cgroups#NAMESPACE-ISOLATION;

контрольные группы (Control groups, cgroups) – предоставляют механизм для управления использованием ресурсов для процессов

или множества процессов. Контейнеры по умолчанию имеют доступ ко всем вычислительным ресурсам, таким как память и процессор, но cgroups предоставляют метод для ограничения использования ресурсов для контейнера:

объединенная файловая система – позволяет представлять несколько каталогов как один. Эта концепция является ключевой для минимизации размера контейнеров и поддержки **слоев для чтения** и для **записи**. В системе Linux файловая система OverlayFS поддерживает объединенную файловую систему. Подробнее о том, как это работает для контейнеров, можно прочитать на странице https://docs.docker.com/storage/storagedriver/overlayfs-driver/#how-the-overlay-driver-works.

Позвольте мне остановиться и объяснить ключевую концепцию для контейнеров, о которой шла речь в предыдущем абзаце.

Слой для чтения – образ контейнера доступен только для чтения и состоит из набора файлов, размещенных в файловой системе. Для SQL Server это включает минимум поддерживаемых файлов из образа базовой операционной системы и файлов для SQL Server, включая двоичные файлы и системные базы данных.

Слой для записи – это любые изменения, внесенные в файловую систему контейнера после его запуска. Они могут включать любые изменения в файлах из слоя для чтения или добавление новых файлов. Слой для записи сохраняется в течение всего срока службы контейнера. После удаления контейнера слой для записи также удаляется. Как вы догадываетесь, для пользовательских баз данных SQL Server это представляет проблему.

Том – это место постоянного хранилища на хост-узле, которое связано с каталогом в доступном для записи слое контейнера. Для SQL Server вы увидите, что обычной практикой является использование тома, связываемого с каталогом в контейнере для хранения баз данных. Тома сохраняются независимо от времени существования контейнера, поэтому если контейнер удален, том все еще существует.

Одна вещь, за которую я люблю Docker, – это за то, что он представил свой собственный API, абстрагирующий концепции ОС, который поддерживает контейнеры, называемый **libcontainer**. Дополнительную информацию о libcontainer можно найти по ссылке https://github.com/opencontainers/runc/tree/master/libcontainer. Еще один интересный ресурс, где рассказывается о природе контейнеров с открытым исходным кодом, – Open Container Initiative (OCI), членом которой является Microsoft (www.opencontainers.org/).

Важно отметить, что Docker является примером среды выполнения контейнера – одной из самых популярных в отрасли. Существует среда выполнения контейнера с открытым исходным кодом, которая называется containerd, о которой вы можете прочитать по ссылке https://containerd.io/.

Жизненный цикл контейнера

При установке Docker в Linux, Windows или macOS устанавливаются следующие компоненты, которые обеспечивают работу контейнеров:

механизм Docker (docker engine) – состоит из демона Docker (который является «сервисом»), контролирующего все операции по созданию и запуску контейнеров. Механизм Docker поддерживает API для программ, взаимодействующих с механизмом построения и запуска контейнеров. Вы можете прочитать больше о механизме Docker, перейдя по ссылке https://docs.docker.com/engine/, а также об API механизма Docker, перейдя по ссылке https://docs.docker.com/develop/sdk/;

клиент Docker (docker client) – это приложение, которое использует API механизма Docker для создания и запуска контейнеров. Клиент Docker – это согласованная программа, которая поддерживает все возможности и ведет себя одинаково в Windows, macOS и Linux. Вы будете использовать клиента Docker для выполнения практических упражнений в этой главе;

инструмент для управления набором контейнеров Docker (docker compose) – это приложение под названием **docker-compose**, которое позволяет создавать и запускать многоконтейнерные приложения. В одном из практических примеров с использованием репликации SQL Server, приведенных далее в этой главе, будет использоваться данный инструмент.

Используя эти компоненты, рассмотрим следующую схему, приведенную на рис. 7.2, которую я называю жизненным циклом контейнера.

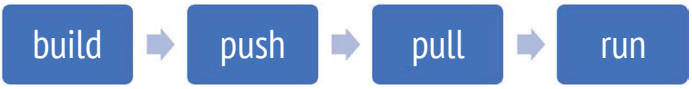


Рис. 7.2. Жизненный цикл контейнера

Давайте рассмотрим каждый из ее элементов более подробно.

build – команда сборки Docker, которая используется для создания нового образа контейнера. Хотя командой поддерживается SDK, стандартный подход заключается в определении образа для построения с использованием файла с именем **Dockerfile**. Дополнительную информацию о сборке Docker можно получить по ссылке https://docs.docker.com/engine/reference/commandline/build/. Ссылку на синтаксис Dockerfile можно найти по адресу https://docs.docker.com/engine/reference/builder/. Microsoft создает образы, которые содержат SQL Server, поэтому во многих случаях вам не придется самостоятельно проделывать эту процедуру. Однако существуют обстоятельства, при которых вы будете создавать настраиваемый образ на основе SQL Server. Я приведу примеры таких сценариев далее в этой главе.

push – после создания образа вам нужно будет предоставить другим людям возможность использовать его; для отправки или публикации образа контейнера в реестре используется команда **docker push**. Этот реестр может находиться на локальном сервере или в открытом доступе. Одним из наиболее распространенных реестров в публичном доступе является Docker Hub или hub.docker.com. Microsoft публикует образы своих контейнеров, в том числе и контейнеров с SQL Server, на сайте mcr.microsoft.com (так называемый реестр контейнеров Microsoft). Далее в этой главе я расскажу, как найти различные образы SQL Server в реестре контейнеров Microsoft. Дополнительная информация о docker push доступна по ссылке https://docs.docker.com/engine/reference/commandline/push/.

pull – любой, кто хочет использовать образ контейнера, должен получить его по запросу, даже если этот образ хранится на локальном сервере. Получить образ контейнера можно с помощью команды **docker pull**. Механизм Docker (docker engine) будет хранить копию образа локально на хост-узле. Дополнительную информацию о команде docker pull можно найти по ссылке https://docs.docker.com/engine/reference/commandline/pull/.

run – чтобы запустить контейнер на основе образа, используется команда **docker run**. Если вы запускаете контейнер на основе образа, который еще не был получен по запросу, Docker сначала получает образ, а затем запускает контейнер. В этой главе вы познакомитесь со всеми операциями, необходимыми для запуска контейнера SQL Server.

После того как вы запустите контейнер, вы захотите **управлять** им. С помощью клиента Docker вы сможете остановить, запустить, перезапустить и удалить контейнер. Кроме того, клиент Docker позволит вам управлять образами, в том числе удалять их.

Клиента Docker также можно использовать для **мониторинга** и управления экосистемой контейнеров: он позволяет получить списки запущенных и остановленных контейнеров, а также выводить данные статистики и журналов для запущенных и остановленных контейнеров.

Наконец, клиент Docker позволяет вам взаимодействовать с запущенными контейнерами, копируя файлы в слой для записи на хост-узле и запуская программу, которая существует в файловой системе контейнеров (которая будет выполняться в том же пространстве имен, что и основная программа-контейнер). Эти команды будут очень полезны для контейнеров SQL Server, как вы увидите в примерах, приведенных в данной главе.

Контейнер SQL Server

Образы контейнеров SQL Server содержат необходимые файлы для ядра SQL Server, SQL Server Agent, всех функций, включенных в механизм SQL

Server, таких как репликация и средства командной строки SQL Server (sqlcmd и bcp). Когда вы запускаете контейнер SQL Server, SQL Server уже предварительно установлен! Другими словами, когда вы получаете и запускаете контейнер SQL Server, вы готовы его использовать. Это одно из основных преимуществ использования контейнера SQL Server. После запуска контейнера установка SQL Server не требуется.

В предыдущем разделе я упоминал, что образ создается с помощью команды сборки Docker `docker build` с использованием файла с названием Dockerfile. Чтобы понять, как предварительно установлен контейнер SQL Server, вот примерное описание команд в Dockerfile для SQL Server:

```
FROM <базовый образ на основе ubuntu или rhel>
LABEL <информация о метке Microsoft>
EXPOSE 1433
COPY <библиотеки и двоичные файлы SQL Server>
RUN ./install.sh
CMD ["/opt/mssql/bin/sqlservr"]
```

Команда **FROM** задает базовый образ ОС, на котором построен образ контейнера SQL Server. Одной из замечательных особенностей контейнеров является возможность создавать новые образы на основе других, создавая эффект наложения образов. Далее в этой главе я покажу вам, как создать свой собственный образ на основе образа SQL Server (который основан на образе базовой ОС).

Команда **EXPOSE** позволяет контейнеру SQL Server устанавливать соединения, необходимые приложениям для обмена данными, с использованием порта 1433 внутри контейнера. Это важно, поскольку по умолчанию контейнеры изолированы. Вы увидите, что часто этот порт отображается на другой порт хост-узла, что позволяет нескольким контейнерам SQL Server работать на одном хост-узле (что обычно приводит к сбою, поскольку два приложения не могут быть привязаны к одному и тому же порту).

Команды **COPY** и **RUN** являются лишь частью процесса сборки для копирования всех двоичных файлов SQL Server в файловую систему образа контейнера и установки любых программных зависимостей.

Все эти команды в Dockerfile SQL Server до сих пор являются шагами создания образа контейнера. Когда выполняется команда сборки Docker `docker build`, каждый из этих операторов используется для построения образа. Оператор CMD сообщает Docker имя программы, запускаемой при запуске контейнера, в нашем случае это sqlservr. Это означает, что контейнер SQL Server не работает как «служба» (например, служба systemd в Linux). Когда я рассказал об этом нескольким профессионалам, они спросили: «Как же SQL Server продолжает работать?» Оказывается, приложение SQL Server (это работает таким же образом в Windows) создано как *программа-демон*, а это означает, что оно работает в фоновом режиме, пока не получит сигнал о том, что должно остановиться.

Итак, теперь давайте посмотрим, как запускается контейнер SQL Server, а затем поговорим о том, как мы «предварительно устанавливаем» SQL Server.

Основной синтаксис для запуска контейнера SQL Server выглядит, как показано ниже, и выполняется с помощью команды `docker run` (примечание: в Linux обычно нужно перед командами запуска контейнера вводить команду `sudo`):

```
docker run
-e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=Sql2017isfast'
-p 1401:1433
-v sqlvolume:/var/opt/mssql
--hostname sql2019latest
--name sql2019latest
-d
mcr.microsoft.com/mssql/rhel/server:2019-latest
```

Далее я расскажу о каждом из аргументов этой команды.

```
-e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=Sql2017isfast'
```

Параметры `-e` обозначают переменные среды, которые используются для запуска и работы контейнера. В случае с SQL Server вам необходимо как минимум принять пользовательское соглашение и ввести пароль sa. Также в команде могут использоваться другие переменные среды, позволяющие указать версию SQL Server или включить SQL Server Agent. Любая переменная среды, поддерживаемая SQL Server, может использоваться для предварительной настройки установки SQL Server при запуске контейнера. Полный список переменных среды приведен по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-configure-environment-variables.

```
-p 1401:1433
```

Этот параметр не потребуется, если вы собираетесь запускать только один контейнер SQL Server на одном хост-узле (причем на этом хост-узле не установлен SQL Server). Если у вас есть несколько экземпляров SQL Server, вам нужно связать порт 1433 с другим портом. Любое приложение, желающее подключиться к этому контейнеру SQL Server, теперь будет использовать порт 1401 вместо порта по умолчанию.

```
-v sqlvolume:/var/opt/mssql
```

Данный параметр указывает, какой том следует применять для связывания с каталогом SQL Server, где хранятся базы данных. Это необязательный параметр; однако если вы хотите, чтобы ваши базы данных сохранялись независимо от времени жизни контейнера, вам придется использовать тома для их хранения. Так, тома определенно будут исполь-

зоваться для любого контейнера SQL Server, используемого в промышленной среде.

```
--hostname sql2019latest
```

Данный параметр тоже необязательный, но он очень удобен. Это удобство заключается в том, что указанное вами имя вычислительного узла, на котором размещен контейнер, станет значением @@SERVERNAME в SQL Server.

```
--name sql2019latest
```

Данный параметр также не является обязательным, но он удобен для управления контейнером. Задав для контейнера имя, вы теперь можете легко идентифицировать контейнер по имени и управлять им. Например, после запуска этого контейнера вы можете остановить его, выполнив команду `docker stop sql2019latest`.

```
-d
```

Этот параметр определяет, что контейнер будет запускаться в фоновом режиме. Обычно данный параметр используется для контейнера SQL Server. Тем не менее хорошим способом отладки является удаление этого параметра, если вы не можете запустить контейнер SQL Server. Это связано с тем, что когда программа sqlservr запускается из командной строки, поведение по умолчанию – передать содержимое ERRORLOG в стандартный вывод (stdout); выведенные таким образом данные затем отобразятся при запуске контейнера. Вы также можете использовать команду `docker logs`, чтобы вывести ERRORLOG контейнера SQL Server.

```
mcr.microsoft.com/mssql/rhel/server:2019-latest
```

Это *тег* образа контейнера, который вы хотите запустить. В следующем разделе я покажу, как выяснить, какие теги использовать для определенного контейнера SQL Server. Если помеченный с помощью тега образ контейнера отсутствует на локальном узле, Docker сначала получит этот образ по запросу, а затем запустит контейнер.

Одним из интересных аспектов работы контейнера SQL Server является последовательность запуска. Когда программа sqlservr запускается в контейнере, каталог /var/opt/mssql не существует. Однако программа sqlservr достаточно интеллектуальна для того, чтобы создать этот каталог и извлечь системные базы данных из установленных файлов образа контейнера при запуске. Кроме того, sqlservr понимает, как принимать переменные среды и использовать их в качестве параметров запуска, чтобы было возможно принять пользовательское соглашение, указать пароль sa и задать значения других переменных среды. Проще говоря, программа sqlservr знает, как себя установить! Давайте посмотрим, что нового в SQL Server 2019 при работе с контейнерами, а затем перейдем к практическим примерам.

Что нового для SQL Server 2019

Теперь, когда вы получили представление о том, как работают контейнеры и как они работают с SQL Server, давайте посмотрим, какие новые возможности для использования контейнеров появились в SQL Server 2019.

- Сейчас мы предоставляем образы контейнеров SQL с базовым образом ОС Red Hat Enterprise Linux (RHEL) для SQL Server 2019. Далее вы узнаете, как выглядят эти образы. Для примеров в этой главе я буду использовать в основном образы RHEL.
- Контейнеры SQL Server 2019 по умолчанию запускаются без полномочий root, что позволяет официально поддерживать SQL Server в Red Hat OpenShif.
- Все образы контейнеров SQL Server теперь хранятся в реестре Microsoft Container Registry, размещенном по ссылке **mcr.microsoft.com**.

  Когда мы выпустили SQL Server 2017 и образы контейнеров, то опубликовали созданные образы в хранилище Docker Hub, доступном по ссылке https://hub.docker.com/_/microsoft-mssql-server. С тех пор в Microsoft мы установили стандарт, согласно которому официальные образы контейнеров Microsoft теперь будут публиковаться в реестре контейнеров Microsoft, размещенном по ссылке mcr.microsoft.com. Мы продолжаем публиковать ссылки на наши образы контейнеров в Docker Hub, но сами образы хранятся только на mcr.microsoft.com.

  Здесь я должен остановиться и объяснить правила именования для образов контейнеров.

  Образы контейнеров SQL используют следующее соглашение об именах:

  mcr.microsoft.com/mssql/server:<tag> – образы Ubuntu;

  mcr.microsoft.com/mssql/rhel/server:<tag> – образы Red Hat Enterprise Linux.

Примечание. Хотя в настоящий момент мы не поставляем образы контейнеров для SUSE, вы можете создать их самостоятельно, используя следующий пример, предоставленный сотрудником Microsoft Вин Ю (Vin Yu) по ссылке https://github.com/microsoft/mssql-docker/tree/master/linux/.

  В части имени <tag> указывается конкретная сборка, которую вы ищете, или «последняя» сборка.

  Например, чтобы получить последнюю сборку образа контейнера SQL Server 2017 для Ubuntu, вы должны использовать следующее имя образа контейнера:

```
mcr.microsoft.com/mssql/server:2017-latest-ubuntu
```

или для SQL 2017 CU10 для Ubuntu, вам нужно было бы использовать

```
mcr.microsoft.com/mssql/server:2017-CU10-ubuntu.
```

Примечание. Вы также можете использовать тег 2017-latest для последней версии образа Ubuntu, однако это не рекомендуется. Это были оригинальные теги, которые мы использовали при первой поставке SQL 2017. Лучше всего явно указывать имя базового образа.

В SQL Server 2017 мы не поставляли образы контейнеров RHEL. Все они относятся к версии SQL Server 2019. Например, чтобы получить последний образ контейнера SQL Server 2019 RHEL, вам нужно использовать следующую ссылку:

```
mcr.microsoft.com/mssql/rhel/server:2019-latest
```

Если вы загружаете образ контейнера SQL и не уверены, для какой версии SQL Server он был создан, используйте команду ~~`docker inspect`~~. Сначала выполните следующую команду:

```
docker images
```

В результате вы получите список образов, которые хранятся локально на вашем сервере. Столбец TAG может дать вам подсказку о версии SQL Server. Но если значение в столбце TAG будет примерно таким, как 2017-latest-ubuntu, вы не узнаете, что такое сборка CU для SQL Server 2017, пока не запустите контейнер из этого образа. Но если вы выполните команду

```
docker inspect <IMAGE ID>
```

где IMAGE ID – это идентификатор GUID из списка, полученного в результате выполнения команды `docker images`, то в результате получится файл JSON с описанием образа. Это может быть очень полезно для любого образа контейнера. Если вы найдете в тексте JSON раздел Labels, то увидите примерно следующий фрагмент:

```
"Labels": {
              "com.microsoft.product": "Microsoft SQL Server",
              "com.microsoft.version": "14.0.3223.3",
              "vendor": "Microsoft"
```

Здесь номер версии (com.microsoft.version) – это номер сборки SQL Server. Вы можете выполнить простой поиск в интернете и найти номер версии, соответствующий этому номеру сборки SQL Server. В данном примере 14.0.3223.3 соответствует SQL Server 2017 CU16.

Это, конечно, приятно, но как получить весь возможный список образов контейнеров с mcr.microsoft.com? Лучшую подсказку дал мне мой коллега Умачандар Джаячандран (Umachandar Jayachandran), который работает в Калифорнийском университете; она позволит вам сэкономить много времени.

Список всех образов Ubuntu находится по ссылке

https://mcr.microsoft.com/v2/mssql/server/tags/list.

Для образов RHEL вы можете воспользоваться ссылкой

https://mcr.microsoft.com/v2/mssql/rhel/server/tags/list.

Совет. Если вы устанавливаете расширение Docker с помощью Azure Data Studio или кода Visual Studio, то можете использовать это расширение для просмотра mcr.microsoft.com, включая образы mssql/server. В следующей заметке в блоге рассказывается о расширениях: https://jeeweetje.net/2019/07/10/exploring-containers-in-the-microsoft-container-registry-with-visual-studio-code/.

- Теперь мы поддерживаем **некорневые контейнеры** с SQL Server 2019. До этого времени все контейнеры для SQL Server запускались в контексте пользователя root в Linux. Хотя контейнеры работают изолированно, некоторые специалисты считают, что запуск от имени root не является безопасной моделью, что не позволяет официально поддерживать SQL Server в таких средах, как RedHat OpenShift.
- До настоящего момента контейнеры SQL Server поддерживают только аутентификацию SQL Server. Мы намерены поддерживать проверку подлинности Active Directory для контейнеров в SQL Server 2019. В то время когда я писал эту главу, было несколько сомнений относительно того, будет ли это официально поддерживаться для SQL Server 2019. Я подробнее расскажу об этой концепции позже в разделе «Развертывание контейнеров SQL в промышленной среде» данной главы.
- Когда мы готовились к выпуску SQL Server 2019, то объявили о проведении предварительного тестирования контейнеров SQL Server на основе **базового образа Windows**. Я называю это **образом Windows контейнера SQL Server**. В конце главы есть отдельный раздел, где обсуждается эта тема.

Как и во многих темах, связанных с компьютерными технологиями, вы можете прочитать о том, как что-то работает. Но, лишь используя что-то, вы можете собрать все фрагменты головоломки в единую картину. Давайте рассмотрим ряд тем о контейнерах SQL Server на конкретных примерах.

Подготовительные шаги для использования примеров, иллюстрирующих использование контейнеров с SQL Server

Возможно, вы были несколько разочарованы, прочитав предыдущую главу, не содержавшую примеров, однако в этой главе их более чем достаточно. Это одна из глав, над которой мне больше всего нравилось работать, потому что я люблю тему контейнеров.

В этой главе я собираюсь показать вам несколько разных способов запуска контейнеров в Windows и Linux. Я привел сценарии, в которых вы можете запускать все примеры на любой платформе (или в macOS), но в каждом из этих примеров я могу более подробно рассказать о том, как использовать приведенный код на конкретной платформе.

Работая над этой главой, я поставил себе цель, чтобы все примеры были основаны на сценариях оболочки Bash и использовали новую подсистему Windows для Linux (WSL2). Однако на момент написания данного материала для этого потребовалась бы *инсайдерская* сборка Windows 10, и я не хотел, чтобы читатели столкнулись с этим риском. Для пользователей Windows у меня есть примеры, использующие PowerShell (но вы помните, что в предыдущем разделе обсуждалось, что при этом все еще будет использоваться виртуальная машина Docker Desktop). С WSL2 ситуация в корне изменится, но для применения WSL2 вам потребуется использовать следующую официальную версию Windows 10 (если только вы не имеете доступа к инсайдерским сборкам).

Для выполнения всех примеров на всех платформах вам понадобится следующее:

- интернет-соединение, используя которое, будут извлекаться образы Docker из реестра контейнеров Microsoft;
- файл резервной копии базы данных WideWorldImporters, доступный по ссылке https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImporters-Full.bak;
- утилиты командной строки SQL Server, установленные на вашем компьютере, где будут выполняться практические примеры (если это еще не сделано). Пользователи Windows могут загрузить установочный пакет по ссылке https://docs.microsoft.com/en-us/sql/tools/sqlcmd-utility. Пользователи Linux могут воспользоваться документацией, размещенной по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-setup-tools. Пользователи macOS могут обратиться к следующей документации: https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-setup-tools#macos;
- Azure Data Studio или ADS (версия июня 2019 г. или более поздняя), доступная по ссылке https://docs.microsoft.com/en-us/sql/azure-data-studio/

download. ADS идеально подходит для этих примеров, так как это кросс-платформенный инструмент. Пользователям ADS я рекомендую установить следующее расширение: https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-docker. Для ADS вам нужно выбрать вариант **Download Extension** на странице установки. Загрузите файл VSIX на свой локальный компьютер и следуйте инструкции по установке ADS. О том, как добавить расширение для ADS, рассказывается в документации, доступной по ссылке https://docs.microsoft.com/en-us/sql/azure-data-studio/extensions.

Далее приводится список компонент для каждой платформы, которые вам нужно установить.

Пользователи Windows:

Установите Docker Desktop для Windows, доступный по ссылке https://hub.docker.com/editions/community/docker-ce-desktop-windows. Пользователи Windows Server также могут установить Docker, прочитав документацию, размещенную по ссылке https://docs.docker.com/install/windows/docker-ee/.

Еще один важный момент для пользователей Windows. Вероятно, вы будете использовать git для клонирования репозитория этой книги для всех примеров кода, и при этом вы воспользуетесь Git для Windows. При установке Git для Windows обязательно **отключите параметр autocrlf**. В противном случае сценарии оболочки Linux, необходимые для этой главы, не будут работать. Если вы не знаете, как отключить данный параметр, используйте следующую команду при клонировании репозитория:

```
git clone --config core.autocrlf = false <URL-адрес github>
```

Пользователи Linux:

Docker поставляется бесплатно, в версии Community Edition (CE), или платно, для версии Enterprise Edition (EE). Для CE существуют различные варианты установки в зависимости от вашего дистрибутива Linux. Например, пользователи Ubuntu могут установить Docker, перейдя по ссылке https://docs.docker.com/install/linux/docker-ce/ubuntu/ или https://hub.docker.com/search/?type=edition&offering=community.

Если у вас версия Docker EE, вы можете воспользоваться инструкциями по установке Ubuntu, RHEL и SUSE, перейдя по ссылке www.docker.com/products/docker-enterprise.

Пользователи macOS:

Установите Docker Desktop для Mac, перейдя по ссылке https://hub.docker.com/editions/community/docker-ce-desktop-mac. В сценариях, которые я создал для пользователей Linux и macOS, перед всеми командами docker используется команда системного администрирования `sudo`.

Хотя это не требуется для macOS, включение команды `sudo` в текст сценариев позволяет применять один набор сценариев для любой платформы.

Развертывание контейнера SQL Server

Чтобы оценить возможности контейнеров и посмотреть, как они работают, вам нужно увидеть их в действии. Как вы помните из обсуждения жизненного цикла контейнера, приведенного ранее в этой главе, Microsoft уже осуществила сборку и отправку контейнеров SQL Server. Позже в этой главе я расскажу, как вы сможете создавать свои собственные образы на основе SQL Server, но сейчас я покажу вам последовательность действий **pull** ⇒ **run** и **stop** ⇒ **start** ⇒ **remove**. Я также покажу другие команды Docker, которые вы можете использовать для исследования контейнеров.

> **ВАЖНО:** вы должны скопировать файл резервной копии базы данных WideWorldImporters в локальный каталог, где запускаете эти сценарии, чтобы выполнить данное действие. Вы можете загрузить эту резервную копию с сайта https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImporters-Full.bak.

Все примеры, приведенные в этом разделе, можно найти в каталоге **h7_inside_sql_containers\deploy**. Используйте каталог **dockerpowershell** для Windows. Для Linux и macOS используйте каталог **dockerbash** (убедитесь, что ваши сценарии выполняются с помощью `chmod u+x <имя_сценария>`). В этой главе я приведу примеры с применением PowerShell.

1. Запустите следующую команду из PowerShell, чтобы запустить контейнер SQL Server. (Я решил использовать параметр Terminal в Azure Data Studio (ADS) для запуска сценариев или сценария **step1_dockerrunsql.ps1**.) Поскольку образ для SQL Server 2019 не установлен локально на моем компьютере, Docker сначала извлекает, а затем запускает контейнер:

   ```
   docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=Sql2019isfast"
   -p 1433:1433 --name sql2019latest --hostname sql2019latest
   -d mcr.microsoft.com/mssql/rhel/server:2019-latest
   ```

   На рис. 7.3 показан пример выполнения в ADS этого сценария для извлечения образа RHEL для SQL Server 2019.

   Возможно, вы захотите получить информацию об используемом образе SQL Server 2019. Вы можете сделать это с помощью команды, аналогичной приведенной ниже:

   ```
   docker inspect mcr.microsoft.com/mssql/rhel/server:2019-latest
   ```

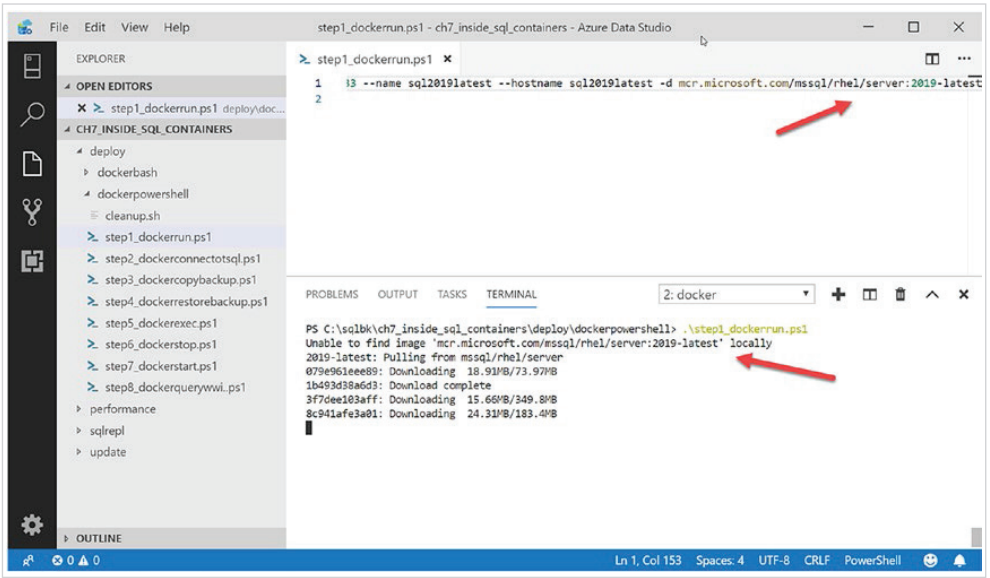


Рис. 7.3. Развертывание SQL Server в контейнере RHEL

В результате ее выполнения вы получите очень длинный JSON-файл. Обратите внимание на следующий интересный раздел этого JSON-файла:

```
"Cmd": [
            "/bin/sh",
            "-c",
            "#(nop) ",
            "CMD [\"/opt/mssql/bin/sqlservr\"]"
        ],
```

Данный фрагмент текста показывает вам команду CMD из Dockerfile, которая должна просто запустить sqlservr. К сожалению, не существует проверенного способа удостовериться в том, какой именно образ является базовым образом контейнера. Я использовал много различных программ, и во всех команда **docker history** для наших контейнеров не выдает имени базового образа.

К сожалению, при запуске контейнера вы не узнаете, правильно ли запущен SQL Server. Команда выведет длинное значение guid и вернется в режим командной строки. Мы можем использовать Docker, чтобы увидеть, запущен ли контейнер, и попытаться подключиться к SQL Server.

2. Выполните следующую команду, чтобы увидеть состояние контейнера SQL Server, который вы запустили:

```
docker ps
```

Если SQL Server был успешно запущен, то результат выполнения этой команды будет выглядеть примерно так:

```
CONTAINER ID      IMAGE
    COMMAND                  CREATED              STATUS
       PORTS                     NAMES
95345f25b901      mcr.microsoft.com/mssql/rhel/server:
2019-latest  "/opt/mssql/bin/sqls..."  About a minute ago
Up About a minute    0.0.0.0:1401->1433/tcp    sql2019latest
```

Единственный верный способ узнать, запущен ли SQL Server, – это попытаться подключиться к нему. Вы можете подключаться к нему вне контейнера или внутри контейнера. Давайте сначала воспользуемся способом вне контейнера, выполнив следующую команду или воспользовавшись сценарием **step2_dockerconnecttosql.ps1**:

```
sqlcmd -Usa -PSql2019isfast '-Slocalhost,1401' '-Q"SELECT @@VERSION"'
```

Результат выполнения данной команды должен выглядеть примерно так (версия SQL Server может отличаться; я использовал SQL Server 2019 CTP 3.2):

```
------------------------------------------------------------------
Microsoft SQL Server 2019 (CTP3.2) - 15.0.1800.32 (X64)
        Jul 17 2019 21:29:33
        Copyright (C) 2019 Microsoft Corporation
        Developer Edition (64-bit) on Linux (Red Hat Enterprise
        Linux Server 7.6 (Maipo)) <X64>
```

Обратите внимание, что SQL Server считает, что он работает на RHEL 7.6. Я также упомянул, что контейнер – это программа, работающая в изолированном режиме. Это можно проверить, если вы работаете в системах Linux, выполнив следующую команду на хост-компьютере:

```
ps -axf
```

Результат выполнения этой команды должен выглядеть примерно так (значения PID могут отличаться):

```
/usr/bin/containerd
 22846 ?        Sl         0:02    \_ containerd-shim -namespace
                                   moby -workdir /var/lib/containerd/
                                   io.containerd.runtime.v1.linux/
                                   moby/4f5c
 22864 ?        Ssl        0:00        \_ /opt/mssql/bin/sqlservr
 22909 ?        Sl        31:37            \_ /opt/mssql/bin/sqlservr
```

Вниманию пользователей Windows и macOS: у меня есть одна хитрость, чтобы полученные вами результаты выглядели примерно так же, как показано выше. Поскольку контейнер SQL Server Linux работает на виртуальной машине Linux, как вы можете получить доступ к самой виртуальной машине?

Попробуйте такой способ. Выполните следующие команды либо из PowerShell, либо из своего терминала macOS (см. следующую заметку в блоге https://nickjanetakis.com/blog/docker-tip-70-gain-access-to-the-mobylinux-vm-on-windows-or-macos):

```
docker container run --rm -it -v /:/host alpine
```

Вы должны получить запрос на доступ к root. Теперь запустите эту команду:

```
chroot /host
```

Вы находитесь в оболочке Bash в контексте виртуальной машины Linux на Windows. Действия, которые вы можете выполнить, ограничены, и команда ps, которую вы использовали ранее, не работает. Но вы можете запустить следующую команду:

```
ps -o ppid,pid,comm
```

Это исключит множество процессов, и поскольку sqlservr был только что запущен, он должен находиться в конце списка. Вы увидите два таких процесса sqlservr (ваши результаты, вероятно, будут отличаться от приведенных ниже):

```
2922 2946 sqlservr
2946 2991 sqlservr
```

Для первого процесса sqlservr значение слева – это ppid, или родительский pid. Теперь запустите такую команду (подставьте в нее значение ppid):

```
ps | grep 2922
```

Вы должны получить следующий результат:

```
2922 root        0:00 containerd-shim -namespace moby -workdir
/var/lib/docker/containerd/daemon/io.containerd.runtime.v1.linux/
moby/a0c005ccefe8c8a716e066b0a857e919bded6f50ac791cb82f6de2b0dbe
f220e -address /var/run/docker/containerd/containerd.sock
-containerd-binary /usr/local/bin/containerd -runtime-root /var/
run/docker/runtime-runc -debug
```

Это процесс Docker, который используется для порождения копии процесса программы, запущенной в контейнере (в данном случае это sqlservr).

3. Чтобы восстановить резервную копию WideWorldImporters, вы должны скопировать ее в слой контейнера для записи. Выполните следующую команду или запустите сценарий **step3_dockercopybackup.ps1**:

```
docker cp c:\sql_sample_databases\WideWorldImporters-Full.bak
sql2019latest:/var/opt/mssql
```

   Когда вы копируете файл резервной копии в /var/opt/mssql, файл резервной копии сразу же становится доступным для SQL Server в контейнере.

4. Когда резервная копия размещена в слое для записи, SQL Server в контексте контейнера может получить доступ к этой резервной копии, таким образом вы можете восстановить ее. Чтобы восстановить базу данных, вы можете использовать инструмент sqlcmd, который *находится в контейнере*. Для этого можно использовать команду `docker exec`, приведенную ниже, или выполнить аналогичную команду из сценария **step4_dockerrestorebackup.ps1**:

```
docker exec -it sql2019latest /opt/mssql-tools/bin/sqlcmd
-S localhost -U SA -P "Sql2019isfast" -Q "RESTORE DATABASE
WideWorldImporters FROM DISK = '/var/opt/mssql/WideWorldImporters-
Full.bak' WITH MOVE 'WWI_Primary' TO '/var/opt/mssql/data/
WideWorldImporters.mdf', MOVE 'WWI_UserData' TO '/var/opt/mssql/
data/WideWorldImporters_userdata.ndf', MOVE 'WWI_Log' TO '/var/
opt/mssql/data/WideWorldImporters.ldf', MOVE 'WWI_InMemory_Data_1'
TO '/var/opt/mssql/data/WideWorldImporters_InMemory_Data_1'"
```

   Резервная копия WideWorldImporters была создана с использованием версии SQL Server 2016, поэтому вы получите сообщение о том, что база данных восстанавливается и обновляется до версии 2019 года.

5. Возможно, вы хотите просмотреть файл ERRORLOG SQL Server, работающего в контейнере. Одним из способов сделать это является использование `docker exec` и перемещение по структуре каталогов контейнера с помощью следующей команды (или сценария **step5_dockerexec.ps1**):

```
docker exec -it sql2019latest bash
```

   При успешном выполнении этой команды ваш курсор переместится в приглашение оболочки Bash в контексте контейнера, как показано ниже:

```
root@sql2019latest:/#
```

   Теперь вы можете перейти в каталог /var/opt/mssql/log и вывести ERRORLOG с помощью команды `cat ERRORLOG`.

Помните, что одним из преимуществ контейнера является минимизация необходимого размера для выполнения программы, которая выполняется на виртуальной машине с загруженной ОС. Кроме того, я говорил вам, что контейнер – это на самом деле просто программа, работающая изолированно и совместно использующая ресурсы операционной системы хоста.

Вы можете увидеть это, выполнив следующую команду:

```
ps -axf
```

Результат выполнения этой команды должен выглядеть примерно так:

```
[root@sql2019latest /]# ps -axf
PID TTY      STAT   TIME   COMMAND
268 pts/0    Ss     0:00   bash
561 pts/0    R+     0:00   \_ ps -axf
1 ?          Ssl    0:00   /opt/mssql/bin/sqlservr
7 ?          Sl     1:25   /opt/mssql/bin/sqlservr
```

Вы можете видеть, что работают лишь bash и sqlservr. Это можно сравнить с выполнением следующей команды на сервере RHEL 7.6 или виртуальной машине на хост-системе (не в контейнере):

```
ps -axf | wc -l
```

Результатом выполнения данной команды будет являться число процессов. На «свежем» сервере RHEL 7.6, который я установил в Azure, я получил число 122! Программа bash запускается в том же пространстве имен, что и контейнер SQL Server, поэтому она изолирована для доступа только к файлам в доступном для чтения и доступном для записи слое этого контейнера.

Выйдите из оболочки, введя команду exit.

Примечание. Вы можете спросить, как можно выполнить эти команды Linux, если контейнер – это просто программа sqlservr. Это связано с тем, что `docker exec` может запускать программу в пространстве имен программы-контейнера (аналогично тому, как sqlcmd работает с контейнером SQL Server). Команда `docker exec` не будет работать, если сама программа не существует в структуре каталогов файлов для контейнера. Командная оболочка bash работает, потому что она размещена в базовом образе. Оболочка sqlcmd работает, поскольку мы устанавливаем sqlcmd в образе SQL Server.

Позвольте мне остановиться и наглядно показать преимущества использования расширения Docker с Azure Data Studio (или Visual Studio Code). В разделе «Подготовительные шаги для использования

примеров, иллюстрирующих применение контейнеров с SQL Server» я упомянул, что вы можете установить это расширение. На рис. 7.4 показан пример управления работающим контейнером SQL Server для «добавления» оболочки Bash.

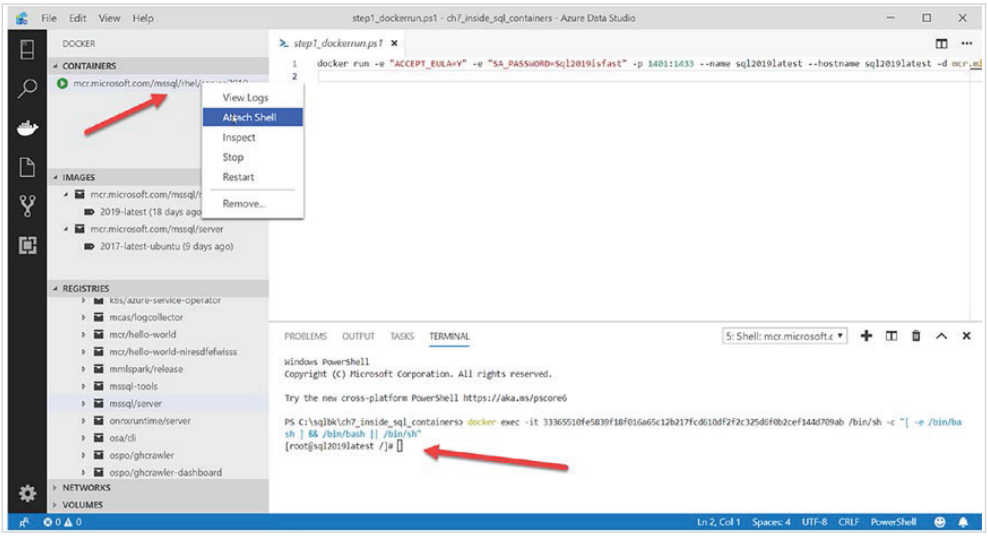


Рис. 7.4. Использование расширения Docker в Azure Data Studio

Как показано на рисунке, вы можете использовать возможности «проводника» данного расширения, чтобы просмотреть запущенные (или остановленные) контейнеры и образы, которые были извлечены, и даже просмотреть реестры контейнеров, такие как mcr.microsoft.com, или ваш собственный реестр контейнеров Azure или Docker Hub.

6. Если вам нужно завершить работу SQL Server в контейнере, технически вы можете выполнить команду T-SQL SHUTDOWN. Это остановит процесс SQLSERVR и закроет контейнер, поскольку он является основной программой, запущенной из контейнера. Также вы можете выполнить следующую команду или сценарий **step6_dockerstop.ps1**, чтобы остановить контейнер:

```
docker stop sql2019latest
```

Если команда будет выполнена успешно, она выведет имя контейнера, используя стандартные средства вывода.

Когда вы останавливаете контейнер, программа, запустившая контейнер, завершает работу. Для SQL Server это приведет к выполнению команды `kill` и, как следствие, «чистому» завершению работы.

Вы можете проверить, что SQL Server был выключен, выполнив следующую команду после остановки контейнера:

```
docker logs sql2019latest
```

Результирующие данные ERRORLOG для контейнера SQL Server будут отображаться в консоли, при этом будет выведена следующая инструкция:

```
<datetime> spid<n>s    SQL Server is terminating in response
to a 'stop' request from Service Control
```

На этом этапе сохраняется доступный для записи слой контейнера (в данном случае база данных WideWorldImporters является частью этого слоя, поскольку она была восстановлена), и контейнер считается *свободным*, но доступным для повторного запуска. Вы можете увидеть любые остановленные контейнеры, которые не были удалены, выполнив следующую команду:

```
docker ps -a
```

Ваш результат должен выглядеть следующим образом:

```
CONTAINER ID        IMAGE
   COMMAND                  CREATED
      STATUS                      PORTS               NAMES
95345f25b901        mcr.microsoft.com/mssql/rhel/server:
2019-latest      "/opt/mssql/bin/sqls..."   10 hours ago
      Exited (0) 11 seconds ago                 sql2019latest
```

7. Вы можете снова запустить контейнер с помощью следующей команды или сценария **step7_dockerstart.ps1**:

```
docker start sql2019latest
```

Так же, как и в прошлый раз, имя контейнера будет отображаться в консоли, и вы снова вернетесь в режим командной строки, к приглашению оболочки.

8. Вы можете выполнить запрос к базе данных WideWorldImporters с помощью следующей команды или сценария **step8_dockerquerywwi.ps1**:

```
sqlcmd -Usa -PSql2019isfast '-Slocalhost,1401' '-Q"SELECT
COUNT(*) FROM [WideWorldImporters].[Application].[People]"'
```

В качестве результата вы должны получить число 1111 – количество строк в таблице People.

9. Если вы остановите контейнер и удалите его, доступный для записи слой контейнера тоже будет уничтожен, как и ваша база данных (это не очень хорошо). Вы можете остановить и удалить контейнер с помощью следующих команд или сценария **step9_dockerstopandremove.ps1**:

```
docker stop sql2019latest
docker rm sql2019latest
```

В контексте работы с контейнерами рассматривайте остановку и удаление SQL Server как удаление ПО SQL Server с компьютера. Однако хорошая новость заключается в том, что других контейнеров SQL Server (даже основанных на том же образе) эта операция не затрагивает.

10. Как было упомянуто выше во время рассказа о том, как работает контейнер SQL Server, используйте тома для хранения своих баз данных в постоянном хранилище, размещенном на вычислительном узле. Том, в котором вы разместите базу данных, сохранится при удалении контейнера.

    Запустите контейнер в указанном томе, используя следующую команду или сценарий **step10_dockerrunvolume.ps1**:

```
docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=Sql2019isfast"
-p 1401:1433 --name sql2019latest --hostname sql2019latest -v
sqlvolume:/opt/mssql/data -d mcr.microsoft.com/mssql/rhel/
server:2019-latest
```

Примечание. Поскольку образ для SQL 2019 по-прежнему находится на вашем локальном компьютере, Docker не будет пытаться снова загрузить его с внешнего узла.

    В этом примере имя **sqlvolume** будет автоматически сопоставлено с каталогом, находящимся на хост-сервере или на виртуальной машине, который не является частью доступного для записи слоя контейнера. Все данные, которые будут записываться в каталог /var/opt/mssql в слое, доступном для записи, перенаправляются в каталог вычислительного узла, выделенный для тома SQL Server (sqlvolume).

    Вы можете узнать имя каталога, выделенного для sqlvolume, выполнив следующую команду:

```
docker inspect sqlvolume
```

    В результате вы должны получить приблизительно следующее:

```
[
    {
        "CreatedAt": "2019-08-07T02:24:50Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/sqlvolume/_data",
        "Name": "sqlvolume",
        "Options": null,
```

```
        "Scope": "local"
    }
]
```

В Windows /var/lib/docker/volumes/sqlvolume/_data является каталогом внутри виртуальной машины Linux, но при этом он все еще сохраняется.

Примечание. Во время написания данной книги мы обнаружили проблему с контейнерами SQL Server и томами Windows. Я хотел, чтобы примеры для пользователей Windows включали инструменты сопоставления томов, которые выглядят примерно так:

```
-v c:\data:/var/opt/mssql
```

Однако мы обнаружили проблему, имевшую место в SQL Server 2017 CU14, которая нарушает эту модель. Другие пользователи сообщали об этой же проблеме на GitHub по ссылке https://github.com/microsoft/mssql-docker/issues/441. Я верю, что эта проблема будет решена ко времени выхода данной книги. Отслеживайте прогресс исправления этой проблемы на указанной странице GitHub.

11. Скопируйте резервную копию WideWorldImporters и снова выполните восстановление базы данных из этой копии, как вы делали это на предыдущих шагах, запустив сценарий **step11_dockercopyandrestore.ps1** или выполнив следующие команды:

```
docker cp c:\sql_sample_databases\WideWorldImporters-Full.bak
sql2019latest:/var/opt/mssql
docker exec -it sql2019latest /opt/mssql-tools/bin/sqlcmd
-S localhost -U SA -P "Sql2019isfast" -Q "RESTORE DATABASE
WideWorldImporters FROM DISK = '/var/opt/mssql/WideWorld
Importers-Full.bak' WITH MOVE 'WWI_Primary' TO '/var/opt/mssql/
data/WideWorldImporters.mdf', MOVE 'WWI_UserData' TO '/var/opt/
mssql/data/WideWorldImporters_userdata.ndf', MOVE 'WWI_Log' TO
'/var/opt/mssql/data/WideWorldImporters.ldf', MOVE 'WWI_InMemory_
Data_1' TO '/var/opt/mssql/data/WideWorldImporters_InMemory_
Data_1'"
```

12. Теперь остановите и удалите контейнер. Затем запустите его снова с тем же именем тома, выполняя следующие команды или сценарий **step12_dockerrestart.ps1**:

```
docker stop sql2019latest
docker rm sql2019latest
docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=Sql2019isfast"
-p 1401:1433 --name sql2019latest --hostname sql2019latest
```

```
-v sqlvolume:/var/opt/mssql -d mcr.microsoft.com/mssql/rhel/
server:2019-latest
```

В этой ситуации, когда запускается новый контейнер SQL 2019, все системные и пользовательские базы данных уже существуют. SQL Server распознает эту ситуацию и просто «использует» эти базы данных и запускается.

13. Убедитесь, что ваши данные все еще размещены в томе, в котором вы их разместили, выполнив запрос к WideWorldImporters аналогично тому, как это было сделано на предыдущем шаге, выполнив следующую команду или сценарий **step13_dockerquerywwi.ps1**:

```
sqlcmd -Usa -PSql2019isfast '-Slocalhost,1401' '-Q"SELECT
COUNT(*) FROM [WideWorldImporters].[Application].[People]"'
```

В результате вы должны получить число 1111 – это число строк в таблице People.

Воспользуемся инструментом Azure Data Studio (ADS) для подключения к контейнеру. Запустите Azure Data Studio (если вы этого еще не сделали). Установите новое соединение и в поле **Server** (Сервер) введите **localhost, 1401** (или же <имя_сервера>, 1401). Введите пароль учетной записи sa, который вы использовали для запуска контейнера. ADS должна подключиться и взаимодействовать с контейнером, как с любым другим SQL Server.

На рис. 7.5 показаны соединение и запрос к базе данных WideWorldImporters с использованием ADS.

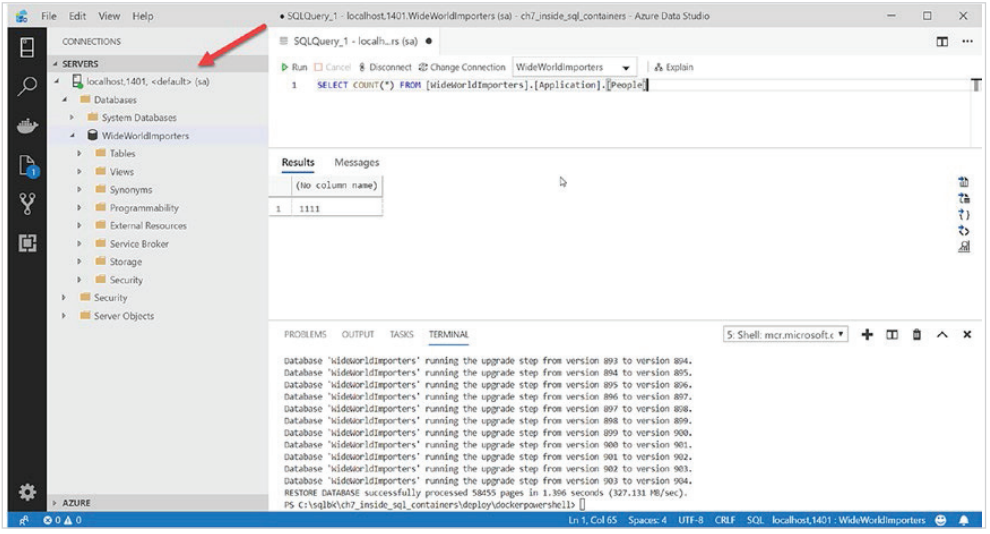


Рис. 7.5. Подключение к контейнеру с помощью Azure Data Studio

14. Теперь освободим все ресурсы, остановив контейнер, удалив его, удалив том и удалив образ, выполнив следующие команды или сценарий **cleanup.sh**:

```
sudo docker stop sql2019latest
sudo docker rm sql2019latest
sudo docker volume rm sqlvolume
sudo docker rmi mcr.microsoft.com/mssql/rhel/server:2019-latest
```

Теперь, когда вы познакомились с основными операциями по развертыванию и управлению контейнером SQL Server, включая сохранение пользовательской базы данных в томе, давайте применим эти навыки, чтобы изучить новый способ обновления SQL Server, используя контейнеры.

Новый способ обновления SQL Server

В самом начале этой главы я упоминал, что контейнеры предоставляют новый способ обновления SQL Server. Давайте посмотрим, как это работает. Поскольку официальная версия SQL Server 2019 еще не вышла, когда я писал эту книгу, не было выпущено и накопительных обновлений, позволяющих показать вам обновление с использованием контейнеров для SQL Server 2019. Поэтому в этом примере я покажу вам, как обновлять контейнеры с SQL Server 2017. После выхода SQL Server 2019 мы начинаем выпускать накопительные обновления, и вы сможете использовать тот же подход для работы с ними.

Чтобы понять данный пример, представьте себе следующий сценарий. В настоящее время вы используете накопительное обновление SQL Server 2017 (CU) 10 в контейнере в промышленной среде. Необходимо применить последнюю версию CU для SQL Server 2017. В Windows или Linux процесс заключается в исправлении или обновлении текущего экземпляра SQL Server, который требует перезапуска SQL Server.

Контейнеры предлагают новый подход, который также требует перезапуска, но быстрее обновляется и обеспечивает огромное преимущество при откате. Вы помните, что контейнеры SQL Server содержат предварительно установленные версии, включая пакеты обновлений. Поэтому, когда вы запускаете контейнер SQL Server на основе любого накопительного обновления, вы не вносите изменений в существующее программное обеспечение.

Все примеры, позволяющие увидеть, как работают обновления для контейнеров, можно найти в каталоге **ch7_inside_sql_containers\update**. Пользователи Windows могут применять каталог **dockerpowershell**, а пользователи Linux и macOS – каталог **dockerbash** (убедитесь, что ваши сценарии запускаются с помощью `chmod u+x <имя_сценария>`).

В этом разделе я покажу вам примеры, использующие PowerShell.

1. Выполните следующую команду или сценарий **step1_dockerrun.ps1**, чтобы развернуть контейнер SQL Server 2017 CU10:

```
docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=Sql2017isfast"
-p 1401:1433 --name sql2017CU10 --hostname sql2017CU10 -v
sqlvolume:/var/opt/mssql -d mcr.microsoft.com/mssql/server:
2017-CU10-ubuntu
```

Маловероятно, что вы уже извлекли образ SQL 2017 CU10, поэтому сначала он будет извлечен. Обратите внимание на использование тома для размещения данных – это ключевой момент для данного метода обновления SQL Server.

2. Подключитесь к SQL Server, чтобы найти версию, которую вы установили, выполнив следующую команду или сценарий **step2_dockerconnecttosql.ps1**:

```
sqlcmd -Usa -PSql2017isfast '-Slocalhost,1401' '-Q"SELECT@@VERSION"'
```

Ваш результат должен выглядеть так:

```
Microsoft SQL Server 2017 (RTM-CU10) (KB4342123) - 14.0.3037.1 (X64)
        Jul 27 2018 09:40:27
        Copyright (C) 2017 Microsoft Corporation
        Developer Edition (64-bit) on Linux (Ubuntu 16.04.5 LTS)
```

3. Выполните следующие команды для обновления контейнера или используйте сценарий **step3_dockerupdate.ps1**:

```
docker stop sql2017CU10
docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=Sql2017isfast"
-p 1401:1433 --name sql2017latest --hostname sql2017latest -v
sqlvolume:/var/opt/mssql -d mcr.microsoft.com/mssql/server:
2017-latest-ubuntu
```

Я немного расскажу о том, что здесь происходит. Первый контейнер остановлен и больше не имеет доступа к системным базам данных, размещенным в томе. Второй контейнер запускается с *использованием того же тома и порта*, но другого образа для последней сборки CU. Новый контейнер запускает SQL Server, который распознает, что системные базы данных уже существуют. Ядро базы данных достаточно интеллектуально, чтобы распознавать ситуацию, когда системные базы данных уже существуют, и просто использовать их. Кроме того, SQL Server выполняет все необходимые шаги по обновлению системных и пользовательских баз данных, чтобы гарантировать их совместимость с конкретной сборкой CU.

4. Выполните следующие команды для подключения к контейнеру (используется тот же порт, что и в предыдущем случае), чтобы убедиться, что версия SQL Server была обновлена, или используйте сценарий **step4_dockerconnecttosql.ps1**:

```
sqlcmd -Usa -PSql2017isfast '-Slocalhost,1401' '-Q"SELECT@@VERSION"'
```

В зависимости от того, как скоро вы выполните этот шаг после обновления, вы можете получить следующую ошибку:

```
Sqlcmd: Error: Microsoft ODBC Driver 17 for SQL Server : Login
failed for user 'sa'. Reason: Server is in script upgrade mode.
Only administrator can connect at this time..
```

Эта ошибка связана с тем, что SQL Server выполняет все необходимые действия для обновления системных и пользовательских баз данных, чтобы обеспечить их совместимость с новым CU. Пользовательские данные при этом не затрагиваются.

Технически не каждое обновление требует каких-либо изменений в метаданных для системных и пользовательских баз данных. Однако мы заметили, что «пытаемся» выполнить эти шаги обновления для любого CU. Это может замедлить процесс обновления – но даже в подобном случае обновление контейнера будет завершено намного быстрее, чем обычный способ обновления ПО SQL Server. Существует способ избежать этого замедления; однако данный способ используется только для целей отладки. Флаг трассировки 902 позволяет обойти любой из шагов обновления CU.

Это означает, что вы можете запустить свой контейнер, добавив следующий оператор в конец команд запуска Docker, которые вы использовали до этого момента:

```
/opt/mssql/bin/sqlservr -T902
```

Это интересная возможность, которую предоставляет Docker. Я показал вам, что Docker использует оператор CMD для запуска программы в контейнере. Оказывается, вы можете переопределить это для образа контейнера, указав вместо этого программу для запуска. Запустив SQL Server с этим флагом трассировки, вы можете запустить контейнер с sqlservr, используя флаг трассировки для переопределения значения по умолчанию. Я использую этот прием, когда демонстрирую новый способ обновления SQL Server с применением контейнеров, но только для презентационных целей. В нашей инженерной группе я обсуждал с одним из руководителей команд, Ли Чжаном (Li Zhang), возможность в будущем сделать процесс обновления SQL Server в контейнере более интеллектуальным, чтобы каждый шаг обновления выполнялся только тогда, когда это действительно необходимо.

В конце концов, запрос на подключение будет работать, и ваш результат должен выглядеть так:

```
Microsoft SQL Server 2017 (RTM-CU16) (KB4508218) - 14.0.3223.3 (X64)
        Jul 12 2019 17:43:08
```

```
Copyright (C) 2017 Microsoft Corporation
Developer Edition (64-bit) on Linux (Ubuntu 16.04.6 LTS)
```

Ваша версия Microsoft SQL Server может отличаться от указанной здесь, потому что, когда вы будете пытаться выполнить эти шаги, вероятно, уже будут выпущены другие, более поздние сборки CU для SQL Server 2017. Идея подобной демонстрации заключается в том, что ваша версия должна быть не ниже CU10, и вам не пришлось обновлять SQL Server вручную.

5. Хотя само по себе обновление наглядно демонстрирует преимущества нового метода, настоящая убедительная история – это откат обновления. Для нового метода процесс отката заменяется простым переключением. Это возможно потому, что сборки SQL Server CU совместимы друг с другом. Используя тот же том и тот же порт, вы теперь можете переключиться обратно на версию обновления CU10, просто выполнив следующие операторы или сценарий **step5_dockerrollback.ps1**:

   ```
   docker stop sql2017latest
   docker start sql2017CU10
   ```

   Поскольку параметры обновлений для контейнеров сохраняются, вы на самом деле просто переключаетесь на установленные версии SQL Server с тем же набором системных и/или пользовательских баз данных.

6. Запустите следующую команду или сценарий **step6_dockerconnecttosql.ps1**, чтобы убедиться, что вы вернулись и работаете с версией SQL 2017 CU10:

   ```
   sqlcmd -Usa -PSql2017isfast '-Slocalhost,1401' '-Q"SELECT@@VERSION"'
   ```

   Вы снова можете получить ошибку обновления, но довольно быстро в результате выполнения команды вы получите версию обновления SQL 2017 CU10.

   Теперь вы можете просто переключаться между различными версиями в соответствии с вашими потребностями. Представьте себе такой мир, где вы предварительно извлекаете образы для серии сборок CU, которые хотите использовать в рабочей среде на локальном сервере. Затем вы можете запускать контейнеры со сборкой, соответствующей любой версии CU, которая вам потребуется для вашего приложения.

   Что действительно важно, так это возможность применять контейнеры для тестирования определенного CU на тестовом сервере, а затем перенести его на рабочий сервер для обновления в заданное вами время.

7. Очистите все использованные ресурсы, выполнив сценарий **cleanup.ps1**. Если вы хотите удалить все ресурсы, но оставить образы для повторного, более быстрого выполнения этой серии шагов, используйте сценарий **reset.ps1**.

Развертывание контейнера как приложения

В некоторых ситуациях вы можете *настроить* образ контейнера с SQL Server. Концепция настраиваемых контейнеров заключается в использовании образа контейнера SQL Server в качестве основы и добавлении файлов в образ контейнера. Так вы можете, например, добавлять файлы резервных копий баз данных и/или файлы сценариев.

Одним из сценариев настройки образа контейнера SQL является развертывание нескольких контейнеров в качестве приложения. Примером этого, который часто демонстрирует мой коллега Вин Ю (Vin Yu), является развертывание контейнера SQL Server с базой данных и приложением ASP.Net. Вы можете посмотреть на пример построения контейнерного приложения с использованием SQL Server по ссылке https://docs.docker.com/compose/aspnet-mssql-compose/.

Docker-compose – один из инструментов, который очень полезен для создания нескольких образов контейнеров и запуска контейнеров на основе этих образов. Docker-compose позволяет объявлять определение образов контейнеров вместе с параметрами для запуска контейнеров на основе этих образов.

Отличным примером приложения, использующего SQL Server, для которого требуется несколько контейнеров, является SQL Server Replication. Поскольку репликация SQL Server теперь полностью поддерживается в SQL Server 2019, контейнеры предоставляют интересный метод развертывания. В 2018 году мы вместе с Вин Ю должны были представить новые возможности SQL Server в Linux на различных конференциях. Я попросил Вина подготовить сценарий репликации. Когда мы готовили наши демонстрации, он попросил меня посмотреть на созданный им результат. Он сказал, что использовал контейнеры для развертывания репликации моментальных снимков с участием издателя, распространителя и подписчика с помощью одной команды. Моя первая реакция на эти слова была: «Это невозможно сделать». И он доказал, что я неправ. Давайте воспользуемся примером, который он создал для этой демонстрации (который вы также можете найти в наших примерах на GitHub по ссылке https://github.com/microsoft/sql-server-samples/tree/master/samples/containers/replication).

Все файлы для этого примера можно найти в каталоге **ch7_inside_sql_containers\sqlrepl**.

Поскольку мы будем использовать docker-compose для этого примера, нам не понадобятся сценарии PowerShell или bash. Мы также предоставим набор сценариев для оболочки Bash и T-SQL, но они выполняются *в контексте каждого контейнера*.

Поскольку в этом примере потребуется только одна команда для развертывания и запуска контейнеров для развертывания репликации SQL Server, давайте рассмотрим структуру файлов, представленных в этом примере, прежде чем приступать к практике.

Файл docker-compose.yml

Docker-compose использует декларативный текстовый файл с именем **docker-compose.yml** (который является файлом YAML. YAML расшифровывается как Yet Another Markup Language – еще один язык разметки). Вы можете использовать другое имя файла, но по умолчанию docker-compose ищет файл с именем docker-compose.yml.

Давайте рассмотрим файл docker-compose.yml для этого примера. Тег версии в верхней части файла указывает, какую версию docker-compose следует использовать (последняя версия 3, но вы можете прочитать о версиях compose в документации, размещенной по ссылке https://docs.docker.com/compose/compose-file/compose-versioning/).

```
services:
  db1:
    build: ./db1
    environment:
      SA_PASSWORD: "MssqlPass123"
      ACCEPT_EULA: "Y"
      MSSQL_AGENT_ENABLED: "true"
    ports:
      - "2500:1433"
    container_name: db1
    hostname: db1
  db2:
    build: ./db2
    environment:
      SA_PASSWORD: "MssqlPass123"
      ACCEPT_EULA: "Y"
      MSSQL_AGENT_ENABLED: "true"
    ports:
      - "2600:1433"
    container_name: db2
    hostname: db2
```

В этом примере будут использоваться две «службы» (или два контейнера), которые будут созданы и выполнены при помощи docker-compose. Одна из них называется db1, а другая – db2.

Метод, при помощи которого работает docker-compose, состоит в том, чтобы сначала создать образ контейнера (если указана сборка), а затем

запустить контейнер на основе этого образа, используя другие параметры, определенные в файле docker-compose.yml.

Инструкция

```
build: ./db1
```

указывает, что docker-compose должен перейти в каталог db1 из текущего каталога и выполнить сборку Docker в этом каталоге. То же самое относится и к db2.

Оставшаяся часть определения в файле .yml указывает, как запускать встроенный контейнер в каждом каталоге.

```
environment:
  SA_PASSWORD: "MssqlPass123”
  ACCEPT_EULA: "Y"
  MSSQL_AGENT_ENABLED: "true"
ports:
  - "2500:1433"
  container_name: db1
  hostname: db1
```

Каждое из этих значений передается команде запуска Docker, используемой для запуска контейнера после его сборки. Обратите внимание на применение MSQL_AGENT_ENABLED в этом примере, поскольку для репликации используется агент SQL Server Agent.

Сборка каждого из контейнеров

Давайте посмотрим, что находится в каждом каталоге, представленном в примере создания и запуска контейнеров для репликации. Я называю этот сценарий «методом Вин Ю» как дань уважения моему коллеге Вин Ю, который научил меня, как это сделать.

Каждый каталог содержит следующие файлы:

Dockerfile – содержит определение того, как создать пользовательский образ на основе образа контейнера SQL Server;

entrypoint.sh – становится основной программой, запускаемой для контейнера. Он запускает сценарий с именем db-init.sh и программу sqlservr;

db-init.sh – этот сценарий вызывается entrypoint.sh, он приостанавливается на некоторое время, и затем выполняется сценарий db-init.sql;

db-init.sql – содержит операторы T-SQL для создания издателя, распространителя и подписчика и публикации моментальных снимков на db1. Это создаст базу данных подписчиков для db2. По сути, Вин создал сценарий, который воспроизводит действия SQL Server

Management Studio, чтобы настроить репликацию и сохранить ее для «умного» выполнения кода на T-SQL.

Файл Dockerfile для db1 и db2 будет выглядеть так:

```
FROM mcr.microsoft.com/mssql/rhel/server:2019-latest
COPY . /
RUN chmod +x /db-init.sh
CMD /bin/bash ./entrypoint.sh
```

Когда docker-compose выполняет операцию сборки для каждого контейнера, определение Dockerfile дает команду выполнить следующие действия:

- использовать последний образ контейнера SQL Server RHEL в качестве основы (тот образ, который применяет образ ОС RHEL);
- скопировать сценарии entrypoint.sh, db-init.sh и db-init.sql в файловую систему образа контейнера;
- изменить сценарий db-init.sh, чтобы он был исполняемым сценарием (вам не нужно это делать для сценария entrypoint.sh);
- создать программу по умолчанию для запуска оболочки Bash, выполнив сценарий entrypoint.sh.

Теперь давайте рассмотрим сценарий entrypoint.sh:

```
# Запуск SQL Server, запуск сценария для создания/настройки БД
# Вам нужен незавершающийся процесс, чтобы сохранить контейнер в рабочем
состоянии.
# В серии команд, разделенных символом "&", команды слева от крайнего
правого знака "&" выполняются в фоновом режиме.
# Так что если вы выполняете серию команд одновременно, используя один
символ "&", команда в крайней правой позиции не должна приводить к
завершению работы
 /db-init.sh & /opt/mssql/bin/sqlservr
```

Этот сценарий сначала выполнит сценарий db-init.sh и, пока он выполняется, запустит программу sqlservr (именно это и означает символ «&»: запустите одну программу, а затем запустите следующую).

db-init.sh для db1 выглядит так, как показано ниже:

```
# дождитесь запуска SQL Server
sleep 45s

mkdir /var/opt/mssql/ReplData/
chown mssql /var/opt/mssql/ReplData/
chgrp mssql /var/opt/mssql/ReplData/

echo "запуск сценария настройки"
```

```
# запустить сценарий настройки для создания БД и схемы в БД
/opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P MssqlPass123 -d master -i
db-init.sql
```

Поскольку сначала запускается db-init.sh, он должен дождаться запуска SQL Server, прежде чем выполнять какие-либо сценарии T-SQL.

Затем он создает несколько каталогов, необходимых для хранения моментальных снимков для репликации в слое контейнеров, доступном для записи.

Наконец, выполняет сценарий T-SQL db-init.sql с использованием утилиты sqlcmd, включенной в каждый контейнер SQL Server.

Сценарий **db-init.sql** для db1 довольно длинный; он содержит код T-SQL для настройки издателя, распространителя и подписчика и публикации снимков.

Сценарий db-init.sh для db2 приостановит запуск своего SQL Server, а затем выполнит сценарий db-init.sql в своем каталоге. db-init.sql для db2 требуется только создать базу данных для хранения данных для подписчика.

Это довольно интересный способ настройки контейнера SQL Server. Этот же метод можно использовать для настройки контейнера SQL Server для создания базы данных и запуска собственных сценариев T-SQL. В долгосрочной перспективе нам нужен лучший метод для достижения подобной цели, чтобы вам не пришлось вручную запускать сценарии в оболочке для выполнения пользовательского кода. На данный момент этот метод зарекомендовал себя как удобное средство запуска и выполнения пользовательского кода.

Запуск контейнеров для репликации

Попробуйте воспользоваться одним из следующих двух способов:

- запустите следующую команду (в Linux перед этой командой добавьте `sudo`). Переместитесь в каталог **ch7_inside_sql_containers\sqlrepl**, чтобы запустить эту команду.

  ```
  docker-compose up
  ```

  Если вы используете данный способ, контейнеры запускаются не в фоновом режиме, поэтому вы увидите выведенную в консоль информацию, в том числе содержимое файлов SQL Server ERRORLOG. НЕ нажимайте **Ctrl+C** на этом этапе, иначе вы закроете контейнеры;
- если у вас установлено расширение Docker, вы можете щелкнуть правой кнопкой мыши файл docker-compose.yml и выбрать **Compose Up**. Такой способ запускает контейнеры в фоновом режиме.

Примечание. В некоторых системах Windows 10 я видел всплывающее окно брандмауэра Windows, показанное на рис. 7.6, для службы, необходимой Docker для работы в сети. Если вы увидите это окно, нажмите **Allow Access**.

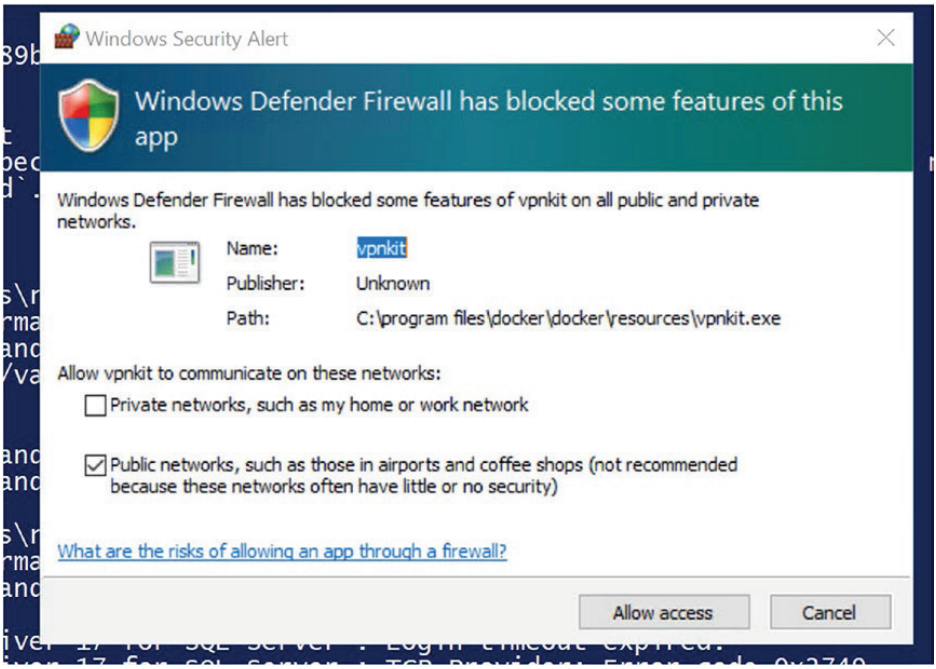


Рис. 7.6. Предупреждение брандмауэра Windows для Docker

Чтобы убедиться, что ваши контейнеры развернуты правильно и репликация SQL Server запущена, сначала проверьте, реплицированы ли данные подписчика. Используйте Azure Data Studio или SQL Server Management Studio (SSMS) для подключения к localhost,2600 и посмотрите, есть ли данные в таблице [Sales].[dbo].[customer]. Таблица должна содержать три строчки.

Кроме того, вы можете убедиться, что задание агента моментального снимка было успешно выполнено, с помощью SSMS, подключающейся к подписчику. Используйте Object Explorer, как показано на рис. 7.7, чтобы проверить статус этого задания.

Когда Вин впервые показал мне такой метод, мне пришлось остановиться и подумать. Я долго работал над сложной проблемой – репликацией, впервые представленной в SQL Server 6.0. Я никогда не думал, что увижу способ развернуть ее так легко. Разумеется, SQL Server Replication может быть сложнее, чем в приведенном примере, поэтому для использования контейнеров вам может потребоваться приложить больше усилий. Но обратите внимание на то, какие возможности предоставляют контейнеры для распределенной системы, такой как SQL Server Replication.

- Если вы запустили docker-compose из командной строки, нажмите **Ctrl+C**, чтобы остановить контейнеры.
- Вам понадобится освободить занятые ресурсы, поэтому, когда вы завершите выполнение практических заданий, выполните следующую команду (предисловие с sudo в Linux). Вы должны находиться в каталоге **ch7_inside_sql_containers\sqlrepl**, чтобы запустить ее.

  ```
  docker-compose down
  ```

  Это остановит и удалит работающие контейнеры.

  Это не приведет к удалению пользовательских образов Docker, поэтому для очистки всех ресурсов используйте сценарий **cleanup.ps1** или **cleanup.sh**, чтобы удалить настроенные образы.

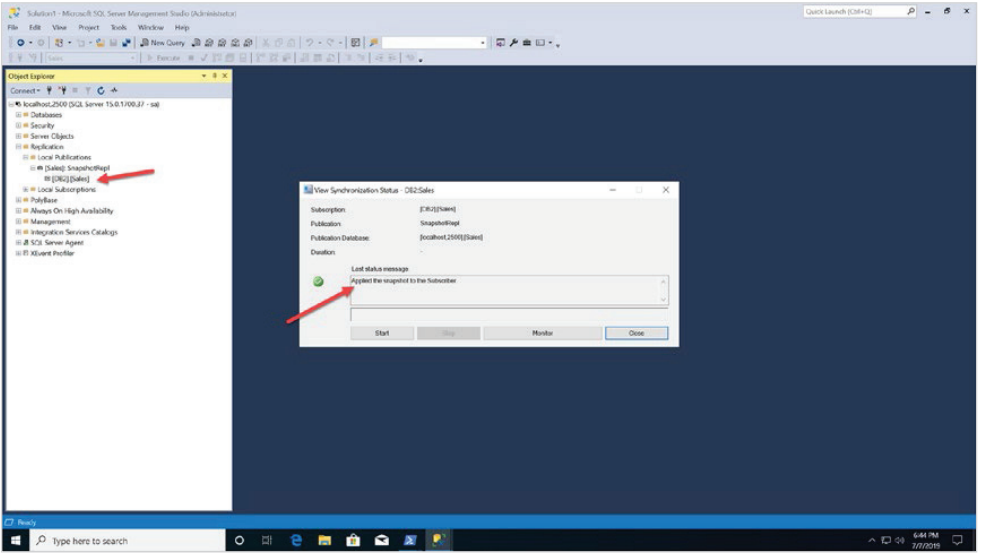


Рис. 7.7. Задание моментального снимка для контейнера репликации SQL Server

Развертывание контейнеров SQL в промышленной среде

Я провожу презентации и рассказываю о контейнерах с тех пор, как мы начали поддерживать их в SQL Server 2017. Поскольку я представил эту тему, говоря о том, что контейнеры на самом деле являются просто программой sqlservr, работающей специфическим образом, я столкнулся с некоторым скептицизмом по поводу того, что контейнеры SQL Server могут использоваться в рабочей среде. Я надеюсь убедить некоторых скептиков, предоставив им в этом разделе главы информацию о производительности, безопасности, доступности, управлении ресурсами и конфигурации сервера в контексте использования контейнеров.

Производительность

Как я уже говорил, один из мифов о контейнерах, которые я слышал, состоит в том, что контейнеры SQL Server могут работать не так, как экземпляр SQL Server, запущенный не в контейнере. Я рассказал о том, что контейнер SQL Server – это просто программа, работающая изолированно и имеющая прямой доступ к ресурсам операционной системы. Следовательно, с точки зрения производительности контейнеры SQL Server для Linux ни в чем не будут уступать SQL Server для Linux, который работает не в контейнере.

Однако вам, разумеется, нужны доказательства. Поэтому я взял инструмент для тестирования с открытым исходным кодом под названием HammerDB (www.hammerdb.com) и провел тест с использованием аналитического теста производительности, полученного из сценария TPC-H. С помощью HammerDB я сравнил производительность для SQL Server для Linux, работающего в контейнере, и SQL Server вне контейнера на той же виртуальной машине Linux. Я обнаружил, что производительность этих двух вариантов полностью идентична.

Я призываю вас попробовать выполнить подобную проверку самостоятельно. Вот описание шагов, которые я выполнил, чтобы использовать для этой цели HammerDB:

- я установил HammerDB на виртуальную машину Windows 10 в Azure;
- создал виртуальную машину Azure с RHEL 7.4, используя конфигурацию DS13 v2 (8 процессоров VCPU, 56 Гб ОЗУ);
- создал управляемый диск Premium SSD объемом 2 ТБ для размещения баз данных, которые будет использовать HammerDB. Я смонтировал этот диск в каталоге в Linux и назвал его /data. Вот ссылка на руководство, которое я использую для добавления диска для хранения данных в Linux на виртуальной машине Azure: https://docs.microsoft.com/en-us/azure/virtual-machines/linux/attach-disk-portal;
- установил SQL Server 2019 CTP 3.2 на виртуальной машине Linux;
- создал базу данных tpch, размер файла данных которой 35 Гб, и журнала – 5 Гб. Я использовал коэффициент масштабирования 10 для 16 виртуальных пользователей (клиент виртуальной машины Windows – это 8-ядерная виртуальная машина);

Совет. При использовании последней версии HammerDB измените драйвер БД на ODBC Driver 17 для SQL Server и используйте кластерный столбцовый индекс.

- затем я запустил тест TPC-H, подключив 16 виртуальных пользователей. Мне удалось достичь производительности ~ 160 000 запросов в час при запуске HammerDB как для контейнера SQL Server, так и для SQL Server вне контейнера, установленных на одной и той же виртуальной машине (я запускаю только один из них одновременно).

Я рекомендую вам самостоятельно запустить такой тест или любой другой тест производительности для контейнеров SQL Server. Просто убедитесь, что вы выполняете справедливое сравнение между SQL Server в контейнере и SQL Server вне контейнера.

Безопасность

Поскольку контейнеры SQL Server представляют собой просто SQL Server в Linux как изолированные программы, все возможности безопасности и все возможности версии SQL Server для Linux работают и для контейнеров. Эти возможности включают в себя функции безопасности ядра SQL Server для входа, защищаемых объектов и правил. Такие функции, как динамическое маскирование данных, безопасность на уровне строк, прозрачное шифрование данных (TDE) и другие, также работают для контейнеров.

Аутентификация Active Directory

Единственным исключением из полного набора функций безопасности является аутентификация Active Directory (AD) (Active Directory (AD) Authentication). На протяжении всего времени разработки версии SQL Server 2019 мы собирались документировать и поддерживать аутентификацию Active Directory для контейнеров. И все это еще может быть реализовано в нашем окончательном релизе. Однако, в то время когда я писал эту главу, я обсуждал данную тему с Диланом Греем (Dylan Gray), одним из наших ведущих разработчиков для Linux и контейнеров. Если на момент выхода официальной версии SQL Server 2019 AD еще не поддерживается и не снабжена подробной документацией, я ожидаю, что это будет сделано вскоре после выпуска SQL Server 2019.

Мы с Диланом обсудили, как будет выглядеть процесс с точки зрения документации; оказалось, что он очень похож на настройку аутентификации AD для SQL Server в Linux без контейнеров, которую вы можете найти по адресу https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-active-directory-authentication. Основные отличия состоят в том, чтобы передать в контейнер информацию об имени субъекта-службы Kerberos (Kerberos SPN) через файл ключа (keytab file) в томе, который может использовать контейнер SQL Server. Несколько клиентов говорили мне, что аутентификация AD является ключевым элементом для использования контейнеров SQL Server в промышленной среде, и мы намерены выполнить этот запрос.

Контейнеры без доступа root

С тех пор, как мы начали поддерживать контейнеры SQL Server, программа-контейнер запускается в контексте пользователя root в пространстве имен контейнера. Даже несмотря на то, что программа контейнера, запускаемого от имени root, не имеет полных привилегий пользователя root для компьютера, на котором она запущена, не рекомендуется запускать программу контейнера с правами root.

SQL Server 2019 будет поддерживать контейнеры, не наделенные правами root; и я говорил с Мэдлин Макдональд (Madeline McDonald), одним из наших ведущих разработчиков контейнеров, о поддержке контейнеров без предоставления им уровня доступа root. Она сказала мне, что когда мы работали над выпуском SQL Server 2019, то собирались поставлять все контейнеры SQL Server 2019 без прав доступа root, однако при этом контейнеры SQL Server 2017 останутся с этим доступом.

Я спросил Мэдлин, как запустить CTP-контейнер SQL Server 2017 или SQL Server 2019 без прав доступа root, и она привела мне пример.

Попробуйте использовать сценарии и файлы, которые я разместил в папке **ch7_inside_sql_containers\nonroot**. Вы будете применять Dockerfile для создания собственного контейнера. Dockerfile выглядит следующим образом:

```
FROM mcr.microsoft.com/mssql/rhel/server:2019-latest
RUN useradd -u 10001 -g root mssql
RUN mkdir -p -m 770 /var/opt/mssql && chgrp -R 0 /var/opt/mssql
USER mssql
CMD ["/opt/mssql/bin/sqlservr"]
```

Обратите внимание на команды RUN, которые добавят пользователя и группу для mssql и создадут необходимые каталоги SQL Server, к которым впоследствии будут применены разрешения для группы mssql. Команда USER Dockerfile указывает, в каком пользовательском контексте запускать контейнерную программу. Все остальное происходит точно так же, как обычный запуск контейнера SQL Server. Создайте образ Docker с помощью команды сборки Docker `docker build`, а затем используйте предоставленные мной сценарии для запуска контейнера.

Высокая доступность

Контейнеры SQL Server поддерживают основные функции резервного копирования и восстановления, и даже доставки журналов. Я показал вам пример использования SQL Server Replication, который может оказаться вариантом решения с высокой доступностью для некоторых клиентов.

Несмотря на то что можно установить группу доступности Always On (Always On Availability Group) между контейнерами, предпочтительным методом обеспечения высокой доступности для контейнеров является использование Kubernetes. Я расскажу о том, как сделать SQL Server высокодоступным с помощью Kubernetes, в главе 8.

Управление ресурсами

По умолчанию контейнеры SQL Server (как и все контейнеры) имеют доступ ко всем ресурсам ЦП и памяти на хост-сервере. Docker предоставляет способ контроля и управления доступом к этим ресурсам для любого контейнера. Например, команда `docker run` реализует следующие параметры:

-m – управляет объемом памяти, к которому может обращаться контейнер;

-cpuset-cpus – определяет, на каком процессоре могут выполняться потоки в контейнере. Будьте осторожны с этим параметром управления ресурсами. SQL Server не будет ограничивать количество планировщиков на основе этого параметра. Однако Docker (используя cgroups) будет принудительно определять, на каких процессорах работают все потоки SQL Server. Если вы включите этот параметр, я рекомендую вам использовать его совместно с привязкой к вычислительным ядрам SQL Server, используя команду ALTER SERVER CONFIGURATION.

Вы можете получить дополнительную информацию о применении ресурсов для контейнеров Docker по ссылке https://docs.docker.com/config/containers/resource_constraints/.

Хотя эти параметры можно использовать, для SQL Server я рекомендую использовать встроенные возможности конфигурации SQL Server для управления памятью и ресурсами ЦП.

Например, рассмотрите следующие варианты:

memorylimitmb – управляет объемом физической памяти, доступной для использования SQL Server в Linux. Вы можете получить дополнительную информацию об этой возможности, обратившись к документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-configure-mssql-conf?view=sql-server-ver15#memorylimit;

«Max server memory» – этот параметр sp_configure хорошо знаком пользователям SQL Server и контролирует объем памяти, используемый ядром SQL Server в пределах доступной памяти memorylimitmb;

ALTER SERVER CONFIGURATION – эта команда T-SQL позволяет вам определить, на каких узлах NUMA и/или процессорах будет работать SQL Server. Дополнительную информацию о команде можно найти по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-server-configuration-transact-sql;

регулятор ресурсов (Resource Governor) – позволяет контролировать ресурсы ЦП, памяти и ввода-вывода, особенно на уровне приложения или рабочей нагрузки. Дополнительную информацию об этой возможности можно найти по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/resource-governor/resource-governor?view=sql-server-ver15.

Справедливости ради, не все, что работает в sqlservr в Linux (с использованием SQLPAL), управляется этими параметрами T-SQL. Для обработки инструкций SQL Agent, DTC, Polybase или другого кода вне ядра SQL Engine, выполняемого в SQLPAL, при работе в контейнере может потребоваться контроль ресурсов, и параметры Docker, перечисленные в данном разделе,

могут вам в этом помочь. Однако большая часть потребления ЦП и памяти приходится на ядро базы данных, и SQL предоставляет вам возможность получить желаемый контроль.

Конфигурация сервера или базы данных

В этом разделе главы я привел несколько вариантов конфигурации SQL Server, которые вы можете использовать. Есть много других, предоставляемых сценарием mssql-conf, а также такими средствами T-SQL, как sp_configure и ALTER SERVER CONFIGURATION. Кроме того, базы данных имеют много иных параметров конфигурации, которые вы можете указать при создании базы данных или при выполнении команды ALTER DATABASE.

Хотя работающий контейнер SQL Server можно настроить с помощью операторов T-SQL, следует тщательно продумать, является ли это правильной стратегией. Например, если вы применили изменение sp_configure к контейнеру, который требует перезапуска, вам придется перезапустить контейнер, чтобы изменение вступило в силу. Кроме того, вам необходимо убедиться, что вы используете постоянный том для системных баз данных, чтобы ваши изменения не были потеряны.

Другой вариант – создать настроенный образ (как я показал вам в этой главе на примере использования контейнеров для репликации), применяя сценарии конфигурации, которые понадобятся вам после запуска SQL Server.

Например, допустим, вы хотите убедиться, что ваш SQL Server обеспечивает конкретное значение *максимальной степени параллелизма* для любого контейнера SQL Server в вашей среде. Один из способов сделать это – создать собственный образ контейнера с помощью сценария, который устанавливает желаемое значение maxdop. Вы даже можете пометить его тегом и назвать, чтобы всегда знать, какие параметры вы использовали для определенного контейнера SQL. Эти сценарии для создания контейнера совместно со сценариями T-SQL могут стать частью вашего процесса контроля изменений и жизненного цикла непрерывной интеграции и развертывания программного обеспечения (CI/CD).

Пожалуй, единственной ситуацией, где данный подход не даст положительного эффекта, является любое изменение конфигурации, при котором не требуется перезапуск SQL Server для экземпляра вашей базы данных. Для таких сценариев вы все равно можете создать конкретный образ контейнера с новыми параметрами конфигурации, но также можете напрямую применить изменения конфигурации к работающему SQL Server.

Другая интересная проблема заключается в применении изменений конфигурации, которые требуют перезагрузки (их меньше, чем вы думаете). Однако если у вас есть такой сценарий, вам придется перезапустить контейнер сразу после запуска с изменениями применяемой конфигурации.

Для любых изменений mssql-conf вы должны использовать переменные окружения, которые соответствуют целевым настройкам, как показано в примерах запуска Docker в этой главе. Отличный пример использования этой возможности для DTC можно найти по ссылке https://github.com/microsoft/sql-server-samples/tree/master/samples/containers/dtc. Если по какой-либо причине у вас имеется параметр mssql-conf, в котором отсутствует эквивалентная настройка переменной среды, вы можете создать настроенный образ с предварительно созданным файлом mssql.conf. Вот приблизительный пример того, как можно использовать Dockerfile для этой цели:

```
FROM microsoft/mssql-server-linux:latest
COPY ./mssql.conf /
RUN mkdir /var/opt/mssql
RUN mv ./mssql.conf /var/opt/mssql
CMD ["/opt/mssql/bin/sqlservr"]
```

где ваш mssql.conf содержит все необходимые значения параметров конфигурации. Дополнительную информацию о структуре и формате данного файла можно получить по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-configure-mssql-conf#mssql-conf-format.

Использование других пакетов

Я уже говорил вам, что образ контейнера SQL Server поставляется с ядром базы данных, агентом SQL Server Agent, и включает такие возможности, как репликация и DTC. SQL Server в Linux был построен на концепции пакетов. Поддержка таких возможностей, как Polybase, не входит в стандартный пакет SQL Server и отсутствует в образе контейнера SQL Server.

Мы собрали ряд примеров, чтобы вы узнали, как создать собственный настроенный образ, добавив образ SQL Server в те пакеты, которые вы хотели бы добавить в свой контейнер. Вы можете найти эти примеры по ссылке https://github.com/microsoft/mssql-docker/tree/master/linux/preview/examples.

Версии и лицензирование

По умолчанию, когда вы извлекаете и запускаете образ контейнера SQL Server, как я показал вам в этой главе, мы автоматически применяем версию SQL Server для разработчиков (SQL Server Developer Edition). Как вы, возможно, знаете, Developer Edition не подходит для промышленной эксплуатации.

Следовательно, при запуске контейнера SQL Server вы можете использовать переменную среды MSSQL_PID, чтобы выбрать нужную версию SQL Server. Вы можете получить дополнительную информацию о применении этой возможности по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-configure-docker?view=sql-server-ver15#production.

Один из самых распространенных вопросов о контейнерах, которые мне задавали, – это вопрос о лицензировании в применении к контейнерам.

С выходом версии SQL Server 2017 мы изменили наше руководство по лицензированию, включив в него раздел, относящийся к контейнерам. Вы можете скачать руководство, перейдя по ссылке www.microsoft.com/en-us/sql-server/sql-server-2017-pricing. Я рекомендую просмотреть, в частности, раздел Licensing SQL Server 2017 in Containers («Лицензирование SQL Server 2017 в контейнерах»). После выхода версии SQL Server 2019 будет выпущено новое руководство. Лицензирование для контейнеров аналогично лицензированию для виртуальных машин. Для многих пользователей применяется полная модель лицензирования по количеству ядер. Однако имеется одно интересное исключение, о котором вы должны прочитать: клиенты с Software Assurance (SA) и Enterprise Edition получают выгоду. Согласно руководству, «...с добавлением покрытия Software Assurance (SA) для всех основных лицензий Enterprise Edition (для полностью лицензированного сервера) права пользователей расширены, и разрешается запуск любого количества контейнеров на лицензированном сервере. Это ценное преимущество SA позволяет заказчикам развертывать неограниченное количество контейнеров для удовлетворения потребностей динамических рабочих нагрузок и в полной мере использовать аппаратные вычислительные мощности».

Контейнеры SQL Server в Windows

До сих пор в этой главе мы обсуждали контейнеры SQL Server на основе образов Linux. Вы видели, что эти контейнеры могут работать на любой платформе, включая Linux, Windows и macOS.

Тем не менее команда разработчиков Windows создала возможность запуска контейнеров в Windows на основе образов Windows. Летом 2019 года мы объявили закрытую программу предварительного тестирования возможностей поддержки контейнеров SQL Server на основе образов Windows.

Многие из уже обсуждавшихся ранее понятий будут применяться практически ко всему, что содержится в этой главе. Во многом это связано с замечательной историей совместимости SQL Server в Windows и Linux. Ключевые различия будут в некоторых аспектах, различных для Windows и Linux, таких как конфигурация для Active Directory. Кроме того, когда вы хотите напрямую работать с контейнером, то обычно используете PowerShell или командный процессор.

Windows поддерживает те же концепции, что и Linux, позволяющие раскрыть преимущество использования контейнеров, включая изоляцию через пространства имен. Вы можете получить дополнительную информацию о работе контейнеров Windows по ссылке https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/.

Концепция работы с контейнерами в Windows немного отличается от Linux. Контейнеры могут работать в двух режимах изоляции:

- **изоляция процесса** – контейнеры работают как изолированные процессы с использованием пространств имен;
- **режим изоляции Hyper-V** – контейнеры работают в «специальной» виртуальной машине (этот термин придуман не мной, он используется в документации).

Вы можете дополнительно прочитать об этих моделях изоляции, перейдя по ссылке https://docs.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/hyperv-container.

В Docker for Desktop для Windows вы можете запускать только контейнеры Windows или Linux. (*Примечание*: эта ситуация должна измениться с выходом версии Docker для WSL2. Подробнее об этом можно прочитать по ссылке https://engineering.docker.com/2019/06/docker-hearts-wsl-2/.)

По умолчанию Docker for Desktop поддерживает контейнеры Linux. Чтобы переключиться на использование контейнеров Windows, выберите этот параметр на значке Docker на панели задач Windows, как показано на рис. 7.8.

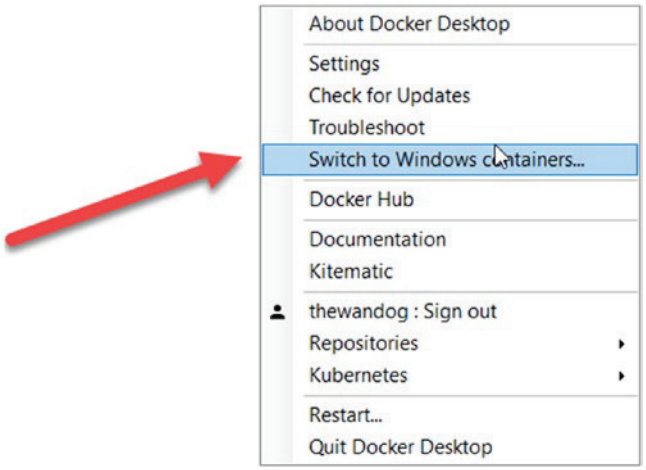


Рис. 7.8. Переход на контейнеры Windows с помощью Docker Desktop

Hyper-V и изоляция процессов поддерживаются для контейнеров Windows в последних сборках Windows 10 и Windows Server 2019. Подробнее о контейнерах Windows в Windows Server 2019 можно прочитать по ссылке https://docs.microsoft.com/en-us/virtualization/windowscontainers/quick-start/quick-start-windows-server. Я также рекомендую вам прочитать ответы на часто задаваемые вопросы о контейнерах Windows в документации, размещенной по ссылке https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/faq.

Кроме того, Windows Server 2019 поддерживает контейнеры Linux в Windows (Linux Containers on Windows, LCOW), что обеспечивает возможность применять контейнеры Linux с использованием режима изоляции Hyper-V. Вы можете прочитать больше о LCOW по ссылке https://docs.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/linux-containers. Это делает

Windows Server практически единственной платформой для одновременного запуска контейнеров Windows и Linux (контейнеры Windows можно запускать, установив виртуальную машину Windows на macOS, но Windows Server является единственной операционной системой, которая изначально поддерживает эти сценарии).

Я попробовал контейнеры Windows в их раннем варианте для закрытого тестирования в Windows Server 2019.

Вот пример синтаксиса команды запуска Docker для Hyper-V и изоляции процесса для контейнера SQL Server Windows:

```
docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=SafePassw0rd' -p
1401:1433 --isolation=process -d -e 'MSSQL_PID=Developer' --name sql1
private-repo.microsoft.com/mssql-private-preview/mssql-server:windows-ctp3.1

docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=SafePassw0rd' -p
1402:1433 --isolation=hyperv -d -e 'MSSQL_PID=Developer' --name sql2
private-repo.microsoft.com/mssql-private-preview/mssql-server:windows-ctp3.1
```

Можно видеть, что синтаксис этой команды почти такой же, как для контейнеров Linux. Обратите внимание на синтаксис для параметра `--isolation`.

Я использовал известный инструмент для просмотра того, что происходит внутри системы, – Process Explorer (https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer), чтобы посмотреть, как выглядит sqlservr с точки зрения изоляции процесса. На рис. 7.9 вы можете увидеть, как, аналогично демону docker, программа CExecSvc отвечает за ветвление контейнерной программы sqlservr.

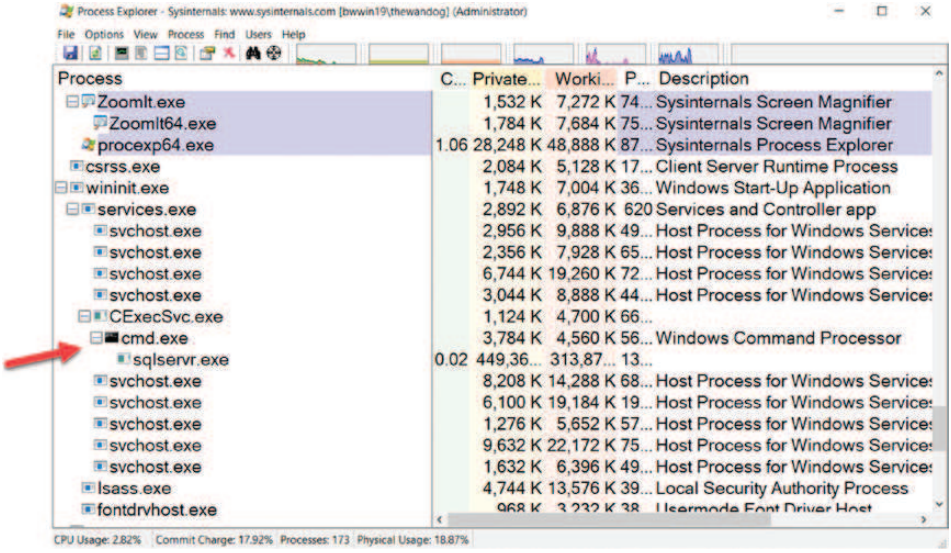


Рис. 7.9. Контейнер SQL Server в Windows: изоляция процесса

Документация по режиму изоляции Hyper-V довольно скудна; в ней практически упоминается лишь то, что программы-контейнеры размещаются в программе Windows под названием vmwp.exe.

Я надеюсь, что к тому времени, когда вы прочтете эту книгу, мы продолжим нашу работу над контейнерами SQL Server в Windows, так как полагаю, что многие клиенты хотят использовать контейнеры, но по ряду различных причин не используют или не могут использовать контейнеры на основе Linux. Я лично считаю, что если контейнеры LCOW хорошо зарекомендуют себя, то многие клиенты могут столкнуться со смешанной средой контейнеров Windows и Linux, размещенных на Windows Server.

Резюме

Это была очень длинная глава. Я рассказал о том, что такое контейнеры и почему с их помощью можно решать современные задачи по развертыванию таких программных продуктов, как SQL Server, и о разработке приложений. К очевидным преимуществам контейнеров можно отнести их портативность, легкость, надежность и эффективность.

Я показал, что контейнеры – это на самом деле просто программы, работающие изолированным и уникальным способом. Вы узнали о новых улучшениях для контейнеров SQL Server в версии SQL Server 2019, в том числе об образах RHEL и новом реестре контейнеров Microsoft.

Если вы внимательно прочитали последнюю часть главы, то увидели и, возможно, попробовали выполнить несколько практических примеров, включая развертывание контейнеров, новый способ обновления (и отката) SQL Server и развертывание многоконтейнерного приложения, такого как репликация SQL Server.

Вы узнали, что контейнеры готовы к эксплуатации в промышленной среде (несмотря на то что вам, возможно, доводилось слышать обратное) по всем параметрам, включая производительность, безопасность и доступность контейнеров SQL Server.

И наконец, добравшись до конца главы, вы получили представление о том, как будут работать контейнеры SQL Server для Windows.

Надеюсь, что теперь вы хорошо разбираетесь в контейнерах SQL Server и готовы узнать о платформе, созданной для развертывания и масштабирования контейнеров, под названием Kubernetes.

Глава 8

SQL Server и Kubernetes

Если контейнеры – это «новые виртуальные машины», то Kubernetes – это «новые серверы». В этой главе вы увидите, что Kubernetes – это важная технология для будущего контейнерных приложений, особенно работающих в промышленных средах с их высокими требованиями к рабочим нагрузкам. К числу таких приложений относится SQL Server.

Что такое k8s?

Если вы еще не читали главу 7, то можете вернуться и хотя бы просмотреть ее, прежде чем приступать к этой главе. Почему? Потому что Kubernetes, или k8s, – это о размещении контейнеров. Разумеется, k8s на самом деле представляет собой нечто гораздо больше, чем просто хостинг. В оставшейся части главы я буду использовать сокращение k8s для обозначения Kubernetes – это популярная аббревиатура (и, конечно, ее быстрее набрать на клавиатуре). k8s означает k<8 букв>s в слове Kubernetes (я должен был сам об этом догадаться, когда впервые увидел этот термин).

В отличие от главы 7, я не буду вдаваться в подробности, рассказывая о внутреннем устройстве k8, потому что для этого нужно было бы написать отдельную книгу. Однако я познакомлю вас с некоторыми терминами, добавлю несколько комментариев о том, как это работает изнутри, и дам несколько отличных ссылок.

Познакомив вас с основами k8s в данной главе, я расскажу о том, как k8s решает важные задачи по развертыванию контейнеров на масштабируемой платформе, которая обеспечивает встроенную высокую доступность (High Availability, HA). Я также покажу, как обновить все контейнеры SQL Server в среде k8s, аналогично процессу обновления одного контейнера, который я описал в главе 7. Я познакомлю вас с концепцией развертывания SQL Server в k8s, которая называется Helm Charts. Наконец, я расскажу о планах на будущее по повышению доступности SQL Server 2019 с применением k8s и о том, как мы собираемся интегрировать группы доступности Always On (Always On Availability Groups) с k8s.

Рекомендуемые источники информации по k8s

Начнем со ссылок – вы можете для начала просмотреть их.

Видеоматериалы Брендана Бернса о k8s (это мой термин, а не официальное название). Брендан Бернс (Brendan Burns), один из основателей k8s

и ныне выдающийся инженер (Distinguished Engineer) в Microsoft, создал серию обучающих видео по k8s. Я лично думаю, что вы можете посмотреть их и узнать, что вам потребуется, если собираетесь использовать k8s. Вы можете посмотреть их на канале www.youtube.com/playlist?list=PLLasX02E8BPCrIhFrc_ZiINhbRkYMKdPT. Я имел удовольствие выступить с Бренданом на саммите Microsoft MVP весной 2019 года и был по-настоящему удивлен тем, как он может упростить рассказ о k8s при помощи графических схем (он создает их на своем устройстве Surface Go с помощью графического пера).

Другой ресурс, который использует эти видео, находится на нашем сайте Azure по ссылке https://azure.microsoft.com/en-us/topic/what-is-kubernetes.

Managing Kubernetes («Управление Kubernetes») – это книга Крейга Трейси (Craig Tracey) и Брендана Бернса (Brendan Burns). Я считаю, что это отличный ресурс, позволяющий познакомиться со внутренним устройством k8s. Вы можете найти эту книгу по адресу https://learning.oreilly.com/library/view/managing-kubernetes/9781492033905/.

Документация AKS – это документация Azure Kubernetes Service (AKS), и есть отличные видеоролики и документация, в которой рассказывается не только об AKS, но и о ядре k8s. Вы можете найти эти ресурсы по ссылке https://azure.microsoft.com/en-us/services/kubernetes-service/.

https://kubernetes.io/ – это основной веб-сайт, посвященный k8s с открытым исходным кодом, где имеется множество примеров, подробностей и графических материалов по k8s. На https://kubernetes.io/docs/tutorials/online-training/overview/ имеется ссылка на отличные онлайн-курсы, посвященные k8s.

Вот еще два онлайн-курса обучения, которые я бы порекомендовал:

www.pluralsight.com/courses/getting-started-kubernetes и
www.pluralsight.com/courses/kubernetes-installation-configuration-fundamentals.

Эти курсы написаны моим другом и техническим редактором Pro SQL Server для Linux Энтони Ночентино (Anthony Nocentino).

Объекты k8s

Возможно, вы уже познакомились со всеми материалами, которые я рекомендовал вам. Однако в этой главе я дополнительно приведу некоторые термины, относящиеся к k8s, и немного расскажу о них с моей собственной точки зрения. Также я чуть-чуть расскажу о внутреннем устройстве, о том, что показалось мне интересным, когда я изучал эту тему.

Вот основные термины и объекты k8s, с которыми вам нужно познакомиться в начале этой главы.

Кластер – думайте о кластере k8s как о сервере или компьютере. Это основной вычислительный узел (*основной хост*) для всех программ, работающих на k8s. Обычно хосты, на которых размещены все объекты, называются кластером k8s.

Узел – рассматривайте узел как виртуальную машину, работающую в кластере. Узел будет хостом для запуска модулей (которые имеют контейнеры) в кластере. Как правило, в кластере k8s имеется несколько узлов. Вы можете прочитать о концепции узлов по ссылке https://kubernetes.io/docs/concepts/architecture/nodes/.

Под (pod) – это логическая коллекция контейнеров, работающих на узле в кластере. Под – это модуль развертывания, управления и отработки отказа контейнеров, работающих в кластере. Вы можете прочитать об этой концепции по ссылке https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/.

Служба – документация Kubernetes описывает службу как «абстракцию, которая определяет логический набор модулей и политику доступа к ним». Для целей SQL Server служба будет выполнять функции *балансировщика нагрузки* и абстракции внутреннего IP-адреса модуля, на котором размещен SQL Server. Это очень похоже на приемник (прослушивающий узел) в классической модели кластеризации SQL Server. k8s предоставляет концепцию сервиса, встроенного в программное обеспечение k8s, и такие приложения, как SQL Server, могут связываться с ним, так что независимо от того, где именно модуль SQL Server размещен в кластере, приложения всегда могут подключаться к сервису с использованием одного и того же IP-адреса и порта. Вы можете прочитать больше о службе k8s, перейдя по ссылке https://kubernetes.io/docs/concepts/services-networking/service/.

Секрет (secret) – это объект k8s, который позволяет хранить конфиденциальную информацию, например пароль. Для SQL Server это очень удобный способ для хранения пароля sa. Вы можете прочитать больше об этой концепции k8s, перейдя по ссылке https://kubernetes.io/docs/concepts/configuration/secret/.

Класс хранилищ (storage class) – это объект k8s, который представляет хранилище, аналогичное дисковой системе. Вы можете найти дополнительную информацию о классах хранилищ k8s по ссылке https://kubernetes.io/docs/concepts/storage/storage-classes/.

Persistent Volume Claim, PVC – это хранилище, резервное копирование которого осуществляется с использованием постоянного тома, который связан с классом хранилищ. Для меня это все равно, что запросить том на обычном диске для хранения данных. Работая с SQL Server, вы убедитесь, насколько удобно и полезно будет использовать эту концепцию для размещения файлов базы данных.

Разумеется, это неполный список терминов, относящихся к k8s. Другие термины я буду вводить и обсуждать по мере того, как мы будем рассматривать примеры в оставшейся части главы.

Несколько слов о внутреннем устройстве k8s

Ссылки, которые я привел ранее в этой главе, помогут вам по-настоящему погрузиться в тему k8s, однако я особо замечу, что один из аспектов k8s, который вы должны понимать, – это API. Мне нравится, что я могу посмотреть, как что-то работает, просмотрев интерфейс прикладного программирования (Application Programing Interface, API). Вся внутренняя часть k8s базируется на **сервере API**. Вы можете прочитать обо всех компонентах, которые приводят в действие кластер k8s, но сервер API – это часть программного обеспечения, которая обрабатывает запросы и «выполняет работу». Возможно, концепцию сервера API вам поможет понять пример с SQL Server. Рассматривайте сервер API как SQL Server. API SQL Server – это T-SQL: приложения могут отправлять команды T-SQL на SQL Server, и он «что-то делает». k8s работает аналогично. Вы можете дополнительно прочитать об API k8s, перейдя по ссылке https://kubernetes.io/docs/concepts/overview/kubernetes-api/. Сервер API является частью *уровня управления* k8s, дополнительную информацию о котором вы можете найти по ссылке https://en.wikipedia.org/wiki/Kubernetes#Kubernetes_control_plane_(primary). Ознакомьтесь с концепцией уровня управления и сервера API при использовании k8s, а затем переходите к теме кластеров больших данных SQL Server в главе 10.

Это означает, что если вам нравится писать код, вы можете развертывать и управлять контейнерами в k8s с помощью API; или вы можете использовать очень удобный интерфейс командной строки (command-line interface, CLI) с названием **kubectl** (Бак Вуди (Buck Woody) всегда произносит это как «кубекатл»), который взаимодействует с API-интерфейсом k8s. Вы будете программировать API k8s с помощью kubectl и декларативного протокола, использующего файлы YAML. Вы можете прочитать больше о kubectl по ссылке https://kubernetes.io/docs/reference/kubectl/overview/. Для того чтобы иметь под рукой краткий справочник, воспользуйтесь следующей «шпаргалкой»: https://kubernetes.io/docs/reference/kubectl/cheatsheet/.

Ознакомьтесь с компонентами, входящими в состав кластера k8s, по ссылке https://kubernetes.io/docs/concepts/overview/components/. И если вам интересно, как осуществляется развертывание и управление контейнерами в кластере k8s, то я подскажу, что Docker – это тот компонент, который устанавливается k8s и работает в каждом узле кластера. Я считаю k8s удобной, но непростой системой для развертывания, планирования, управления, масштабирования и управления приложениями-контейнерами, и в их числе SQL Server.

Поскольку k8s является системой с открытым исходным кодом, можно развернуть кластер различными способами, на разных платформах и системах. Давайте рассмотрим различные варианты развертывания k8s, которые соответствуют вашим потребностям.

Варианты развертывания k8s

Оригинальная версия k8s была разработана Google примерно в 2014 году, когда там работал Брендан, для создания системы для масштабирования

контейнерных приложений, и первоначально использовалась для внутренних нужд. В 2015 году была выпущена версия k8s 1.0 с открытым исходным кодом, и до сих пор k8s остается проектом с открытым исходным кодом (в Википедии приводится интересная история происхождения k8s по ссылке https://en.wikipedia.org/wiki/Kubernetes#History). Название Kubernetes происходит от греческого слова «лоцман» или «рулевой», и оно вполне подходит для того, чтобы задавать курс кораблю мира контейнеров.

С тех пор как k8s стал открытым, несколько компаний взялись за проект k8s и создали коммерческую систему k8s для пользователей. Вы можете просмотреть на сайте k8s список партнеров по ссылке https://kubernetes.io/partners/#kcsp.

В то время когда я писал эту книгу, мой опыт работы с ландшафтом k8s сводился к следующим вариантам развертывания:

- **k8s с открытым исходным кодом**. Я разговаривал с некоторыми клиентами, которые думают о возможностях применения k8s в работе с SQL Server, и они сказали, что собираются развернуть собственные платформы k8s в своем центре обработки данных или на виртуальных машинах в облаке с использованием последней сборки с открытым исходным кодом из k8s. Если вы идете по этому пути, то обычно используете инструмент развертывания с названием **kubeadm** (https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/).

  Другим популярным вариантом является инструмент под названием **kubespray** (https://kubernetes.io/docs/setup/production-environment/tools/kubespray/). Если вы думаете о развертывании собственной среды k8s в вашем центре обработки данных, то такой вариант дает вам максимальный контроль; однако при этом вы являетесь *владельцем* инфраструктуры. Другими словами, вам придется заниматься как обслуживанием, так и управлением кластером k8s вместе с запущенным в нем SQL Server;
- **Minikube**. Хотите, чтобы на вашем ноутбуке или на виртуальной машине работал быстрый и простой k8s с одним узлом? Minikube придет к вам на помощь; он предназначен для небольших тестов и разработки, и вы можете быстро приступить к работе. Вы можете прочитать о том, как настроить Minikube, по ссылке https://kubernetes.io/docs/setup/learningenvironment/minikube/;

Совет. Docker Desktop может автоматически развернуть Minikube. Посмотрите на пример по ссылке https://docs.docker.com/docker-for-windows/#kubernetes.

- **Azure Kubernetes Service (AKS)**. Если вы хотите почувствовать, что такое управляемый кластер k8s, выберите в качестве примера облачную службу, например Azure Kubernetes Services (AKS). Брендан

управляет командой, которая развивает и поддерживает этот сервис, поэтому я чувствую себя довольно уверенно, когда использую AKS, и понимаю, что получаю доступ к самым последним инновационным решениям k8, а также использую преимущества облачных сервисов. В примерах, приведенных в этой главе, я буду использовать AKS, однако они совместимы с любым дистрибутивом k8s. Поскольку AKS является облачным сервисом, они могут внедрять инновации со скоростью облаков (извините за каламбур). Например, AKS может поддерживать как контейнеры Linux, так и Windows (см. https://azure.microsoft.com/en-us/blog/announcing-the-preview-of-windows-servercontainers-support-in-azure-kubernetes-service/). И еще AKS поддерживает концепцию виртуальных узлов (https://docs.microsoft.com/en-us/azure/aks/virtual-nodes-cli). Погрузиться в AKS можно по адресу https://azure.microsoft.com/en-us/services/kubernetes-service;

- **Azure Stack** – это система устройств, предоставляющая клиентам сервисы Azure в их собственных центрах обработки данных. Azure Stack поддерживает вариант развертывания с использованием k8s, о котором вы можете прочитать по ссылке https://docs.microsoft.com/en-us/azure-stack/user/azure-stack-solution-template-kubernetes-deploy. Вы должны рассматривать k8s в Azure Stack как эквивалент AKS, работающий в вашем центре обработки данных. По мере развития AKS будет развиваться и k8s в Azure Stack;
- **Red Hat OpenShift**. Платформа OpenShift очень популярна в отрасли. OpenShift – это платформа k8s, которую можно запустить в вашем центре обработки данных или в общедоступных облаках, о чем вы можете прочитать на сайте www.openshift.com/. Хотя OpenShift совместим с k8s с открытым исходным кодом, все же существуют некоторые различия в использовании системы и платформы. Вы, вероятно, этому не поверите, но я возглавил команду инженеров Microsoft на саммите Red Hat в мае 2019 года, чтобы проконсультировать исследовательскую команду SQL Server 2019 в OpenShift. Проверьте это сами на https://github.com/Microsoft/sqlworkshops/tree/master/SQLonOpenShift. Несколько примеров, приведенных в данной главе, содержат версию OpenShift на GitHub, которую вы можете использовать в своей среде OpenShift. Microsoft предлагает управляемую платформу OpenShift (фактически AKS) под названием Azure Red Hat OpenShift. Вы можете найти дополнительную информацию об этой службе по ссылке https://docs.microsoft.com/en-us/azure/virtual-machines/linux/openshift-get-started#azure-red-hat-openshift;
- **Windows Server**. Как эта платформа попала в данную главу? Все, что я писал до сих пор о k8s, основано на Linux. И это неудивительно, ведь k8s берет свое начало в Linux. Для большей гибкости мы хотели, чтобы Windows Server стал частью мира k8s, поэтому теперь можно использовать Windows Server для размещения кластера k8s. Не все

в этом кластере будет «чистой» Windows, но при этом поддерживаются узлы, основанные на контейнерах Windows. Вы можете узнать больше о Kubernetes и Windows по ссылке https://docs.microsoft.com/en-us/virtualization/windowscontainers/kubernetes/getting-started-kubernetes-windows;

- **другие облачные провайдеры k8s.** Azure – не единственный провайдер облачных решений, поддерживающих k8s. У Amazon есть Elastic Kubernetes Service (EKS), у Google есть Google Kubernetes Engine (GKE), а также существуют и другие провайдеры;
- **другие провайдеры k8s.** На рынке есть другие провайдеры k8s. У SUSE есть несколько решений k8s, о которых вы можете прочитать на www.suse.com/solutions/kubernetes/. Одним из самых популярных из тех, о которых я слышал от своих клиентов, является Rancher (https://rancher.com/). Я уверен, что вы, возможно, слышали об иных решениях, которые хотят использовать клиенты. Еще один провайдер k8s, которому я буду уделять особое внимание, – это VMWare PKS (https://cloud.vmware.com/vmware-enterprise-pks).

В конечном счете ваш выбор зависит от того, хотите ли вы развернуть k8s в собственном центре обработки данных или же в общедоступном облаке. Остальные ваши решения должны основываться на том, какую поддержку вы получите, будете ли вы сами управлять кластером k8s, а также будет ли продолжаться дистрибуция k8s и будет ли она актуальна в будущем. SQL Server будет работать практически на всех платформах и решениях провайдеров k8s. Работая в Microsoft, мне будет интересно увидеть, как будет расти популярность k8s на AKS и Windows Server. По моему опыту работы с Linux, OpenShift является основной платформой k8s, которую используют и оценивают многие клиенты.

Пришло время перейти к практическим примерам, поэтому начнем следующий раздел, где описаны основные требования, которым должна удовлетворять ваша среда для использования примеров, демонстрирующих развертывание, высокую доступность и обновления SQL Server 2019 с использованием технологий k8s.

Подготовительные шаги для использования примеров, иллюстрирующих применение SQL Server и Kubernetes

Все примеры, разработанные для этой главы, используют кросс-платформенную утилиту командной строки kubectl. Вы должны быть готовы выполнять примеры, приведенные в этой главе, с любым дистрибутивом k8s, применяя kubectl.

Я использовал Azure Kubernetes Service (AKS) для развертывания моего кластера k8s для всех примеров. Поэтому при использовании моих при-

меров для вашего дистрибутива k8s, возможно, встретится два следующих различия:

- **класс хранилищ**. В моих примерах используется класс хранилищ для дисков Azure. Вам нужно будет указать класс хранения, соответствующий вашей платформе;
- **балансировщик нагрузки**. В моих примерах используется тип балансировки нагрузки для сервиса, но он реализован только в облачных провайдерах k8s. Если вы не применяете решение облачного провайдера для k8s, вам нужно использовать тип службы, называемый NodePort. Вы можете узнать больше о NodePort, перейдя по ссылке https://kubernetes.io/docs/concepts/services-networking/#nodeport.

Кроме этого, примеры, приведенные в этой главе, должны работать практически на любой настроенной вами платформе k8s.

Если вы применяете AKS, воспользуйтесь моим примером – я использовал для создания своего кластера AKS шаги, описанные в документации, опубликованной по ссылке https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough.

createaksrg.sh

```
az group create --name bwaks --location eastus2
```

createaks.sh

```
az aks create \
  --resource-group bwaks \
  --name bwsqlaks \
  --node-count 2 \
  --enable-addons monitoring \
  --generate-ssh-keys
```

connectoaks.sh

```
az aks get-credentials --resource-group bwaks --name bwsqlaks
```

Вы можете найти сценарии и команды, которые я использовал для создания группы ресурсов, создания кластера и подключения к кластеру в каталоге **ch8_sql_on_k8s**. Несмотря на то что это сценарии оболочки Bash, вы можете запустить эти команды в среде, где поддерживается интерфейс командной строки Azure. Мне нравится использовать облачную оболочку Azure, которую я покажу вам в этих примерах, поскольку интерфейс командной строки Azure встроен в этот инструмент. Если вы хотите применять платформу, вам необходимо установить интерфейс командной строки Azure, который можно найти по ссылке https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest.

Я также рекомендую установить расширение Kubernetes Visual Studio Code, чтобы облегчить использование кластеров и объектов k8s. Я хотел установить это расширение в Azure Data Studio. Для этого вам необходимо скачать следующие расширения:

- https://marketplace.visualstudio.com/items?itemName=redhat.vscode-yaml;
- https://marketplace.visualstudio.com/items?itemName=ms-kubernetes-tools.vscode-kubernetes-tools.

Чтобы использовать это расширение, вам также необходимо установить kubectl, перейдя по ссылке https://kubernetes.io/docs/tasks/tools/install-kubectl/.

Подробнее о том, как установить эти расширения в Azure Data Studio, см. https://docs.microsoft.com/en-us/sql/azure-data-studio/extensions.

Как я уже сказал, мне нравится использовать облачную оболочку Azure для демонстрации этих примеров. Облачная оболочка Azure поддерживает как PowerShell, так и bash и включает в себя множество встроенных инструментов, таких как Azure CLI, kubectl и sqlcmd. Вы можете узнать больше об облачной оболочке Azure, перейдя по ссылке https://azure.microsoft.com/en-us/features/cloud-shell.

Независимо от того, какой дистрибутив клиента или k8s вы используете, если вы можете запустить kubectl, то можете использовать все приведенные ниже примеры.

Развертывание SQL Server на k8s

В процессе практической работы в этом разделе я покажу вам, как развернуть модуль с одним контейнером SQL Server.

Вам понадобится использовать объект secret для хранения пароля sa, хранилище для баз данных и балансировщик нагрузки для подключения к SQL Server. Все это будет сделано с помощью ряда команд kubectl и декларативных файлов YAML.

Кроме того, я рекомендую вам создавать свои собственные модули, используя концепцию пространств имен в k8s. Пространство имен дает вам область для объектов (например, pods), которые вы создаете в кластере k8s, отделяющих их от других объектов. Пространства имен предоставляют удобный механизм для организации и управления вашими объектами k8s.

Давайте шаг за шагом рассмотрим процесс развертывания и подключения к контейнеру SQL Server в модуле. Эти шаги предполагают, что у вас есть кластер k8s. Я создал кластер, в который входят два узла, используя AKS, как описано в разделе «Подготовительные шаги для использования примеров, иллюстрирующих применение SQL Server и Kubernetes», но даже кластер с одним узлом будет работать.

Когда я просматривал эти примеры, то использовал облачную оболочку Azure, которую можно запустить из любого браузера, как показано на рис. 8.1.

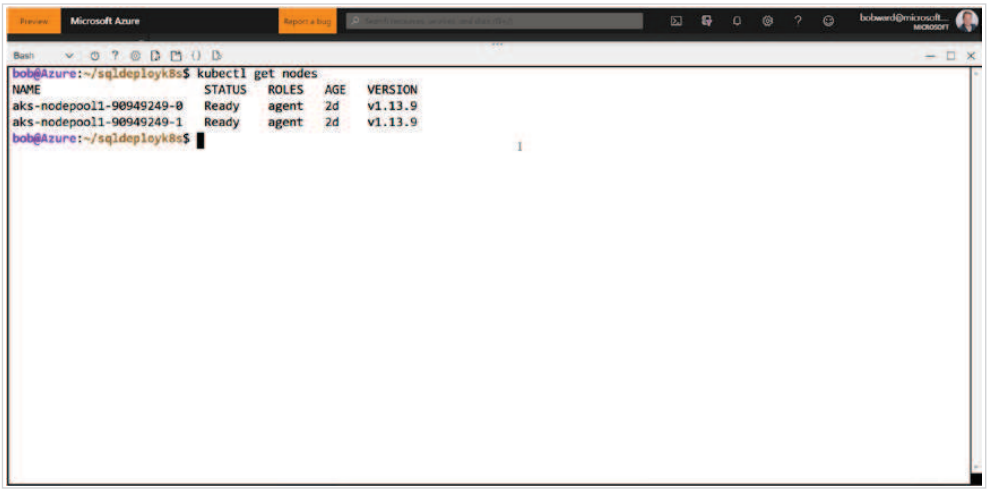


Рис 8.1. Использование Kubernetes в облачной оболочке Azure

Я люблю Azure Cloud Shell. Однажды я летел самолетом обратно в Техас, и на моем ноутбуке разрядился аккумулятор. Мне нужно было поработать с AKS для подготовки очередной демонстрации. Мой iPhone все еще работал, и я слышал о мобильном приложении Azure (https://apps.apple.com/us/app/microsoft-azure/id1219013620). Я установил приложение, установил соединение с моим узлом и просмотрел доступные ресурсы Azure. Я заметил, что в приложении есть инструмент для работы с облачной средой Cloud Shell. Я выбрал его и вернулся к работе. На рис. 8.2 показан пример использования Azure Cloud Shell с моего телефона.

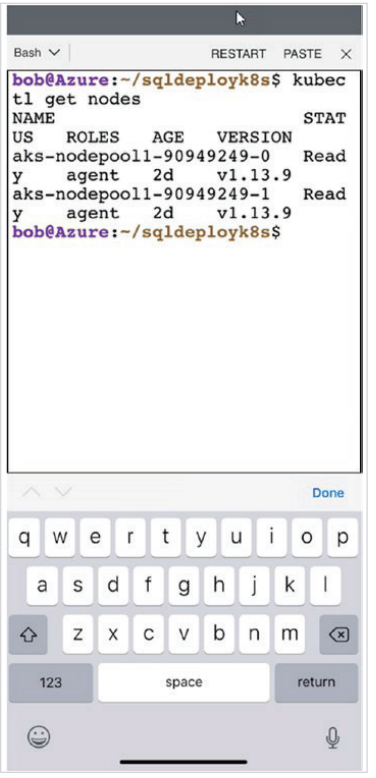


Рис. 8.2. Мобильный пользователь k8s

Я помню, как сидел рядом с пассажиром, который играл в Candy Crush на своем телефоне. Он увидел, что я делаю, и спросил: «Что это за игра, в которую вы играете?» Я ответил: «Я развертываю кластер Kubernetes с помощью Cloud Shell». Мой сосед вернулся к своей игре, очевидно, подумав, что этот чудак играет в странную «игру Kubernetes».

Примеры сценариев для развертывания можно найти в каталоге **ch8_sql_on_k8s\deploy**. Убедитесь, что вы установили разрешения на выполнение сценариев оболочки с помощью команды `chmod u+x <script>`. Кроме того, если вы хотите запустить сценарии в облачной оболочке Azure, прочитайте доку-

ментацию по загрузке этих сценариев, используемых в качестве примера (https://docs.microsoft.com/en-us/azure/cloud-shell/persisting-shell-storage#transfer-local-files-to-cloud-shell). Помните, что большим преимуществом облачной оболочки Azure является то, что для нее требуется только браузер, а все необходимые инструменты, такие как kubectl, az и sqlcmd, уже установлены.

Ниже приведены примеры развертывания SQL Server в AKS.

1. Создайте пространство имен, используя следующую команду или сценарий **step1_create_namespace.sh**:

```
kubectl create namespace mssql
```

   В результате выполнения этой команды вы должны получить следующее сообщение, указывающее, что пространство имен было создано:

```
namespace/mssql created
```

   Проверить, что пространство имен было действительно создано, можно с помощью следующей команды:

```
kubectl get namespaces
```

   Результат, который я получил для своего кластера, приведен ниже. Для кластера k8s вы можете получить другие пространства имен.

```
NAME          STATUS          AGE
default       Active          2d10h
kube-public   Active          2d10h
kube-system   Active          2d10h
mssql         Active          56s
```

2. Я хочу, чтобы все мои объекты создавались в новом пространстве имен mssql. Я могу явно указывать пространство имен при создании объектов или установить контекст по умолчанию для нового пространства имен, выполнив следующие команды (или используя сценарий **step2_setcontext.sh**). В приведенных здесь командах подставьте свой кластер и имя пользователя (вы можете получить эти данные, выполнив команду `kubectl config view`).

```
kubectl config set-context mssql --namespace=mssql
--cluster=bwsqlaks --user=clusterUser_bwaks_bwsqlaks
kubectl config use-context mssql
```

   Если эта команда выполнена успешно, вы должны получить примерно такой результат:

```
Context "mssql" created.
Switched to context "mssql".
```

Чтобы проверить, правильный ли контекст вы используете, или просмотреть его в любое время, вы можете выполнить следующую команду:

```
kubectl config current-context
```

3. Теперь давайте создадим службу балансировки нагрузки, которая будет использоваться для модуля SQL Server. Я должен сделать одно пояснение о балансировщике нагрузки и облачных сервисах, таких как Azure. Azure фактически предоставляет вам общедоступную конечную IP-точку, которая не изменится, поэтому вы можете привязать ее к внутреннему IP-адресу объекта pod независимо от того, изменяется ли он. Я видел сценарии, в которых требуется некоторое время для создания службы балансировки нагрузки. Поэтому я рекомендую после создания не удалять их, если только вы не планируете выполнять тестовые сценарии, аналогичные демонстрации из этой главы.

   Чтобы создать балансировщик нагрузки, выполните следующую команду или сценарий **step3_create_service.sh**:

   ```
   kubectl apply -f sqlloadbalancer.yaml --record
   ```

   Ниже приведен пример использования файла YAML для декларативного доступа к серверу API k8s. Действия, которые выполняются с помощью команды kubectl apply, – это отправка команд API на сервер API, точно так же, как если бы вы написали код с использованием API напрямую (да, теперь вы программист k8s).

   Давайте посмотрим на файл **sqlloadbalancer.yaml**, чтобы понять его формат:

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: mssql-service
   spec:
     selector:
       app: mssql
     ports:
       - protocol: TCP
         port: 31433
         targetPort: 1433
     type: LoadBalancer
   ```

   Протокол использует метки и их значения. Одна из ссылок, которую вы можете использовать для определения точного набора меток и значений для различных объектов k8s, находится по адресу https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/. Лич-

но я смотрю на примеры, копирую их и модифицирую для своих собственных сценариев.

Давайте использовать этот файл YAML для наглядного пояснения некоторых из значений.

```
apiVersion: v1
```

Каждый файл YAML содержит поле apiVersion. Оно сообщает серверу API, какую «версию» API вы используете для различных применений k8. Обычно вам следует использовать версию v1, но для некоторых новых концепций k8s может потребоваться «бета» или другие версии. Подробнее о версиях API читайте по ссылке https://kubernetes.io/docs/reference/using-api/api-overview/#api-versioning.

```
kind: Service
```

Это поле сообщает серверу API, с каким объектом вы взаимодействуете. В данном случае это объект Service, который поможет нам развернуть балансировщик нагрузки. Вы можете прочитать больше о параметре Service для k8s по ссылке https://kubernetes.io/docs/concepts/services-networking/service/.

```
metadata:
  name: mssql-service
```

Это название сервиса. Оно используется, чтобы управлять объектом, а также связывать его с другим объектом, например с объектом pod.

```
spec:
  selector:
    app: mssql
  ports:
    - protocol: TCP
      port: 31433
      targetPort: 1433
  type: LoadBalancer
```

Метка spec позволяет вам дать подробное описание объекта Service. Метка selector дает возможность задать данные для группировки и идентификации объекта. В этом случае использование метки **app:mssql** позволяет управлять и просматривать объекты в k8s. Я покажу вам пример использования этой метки в данном упражнении. Метка app:mssql также важна для LoadBalancer, поскольку она связывает LoadBalancer с любым объектом pod, использующим такую же метку (в нашем случае это будет SQL Server).

Раздел port позволяет сопоставить внешний порт, через который будет осуществляться подключение снаружи, с портом внутри объекта pod. Этот раздел необходимо использовать при развертывании SQL Server, поскольку, как рассказывалось в главе 7 о контейнерах, у вас может быть только один SQL Server, прослушивающий порт 1433 на уровне хост-узла. В этом примере, когда приложения хотят подключиться к SQL Server, они будут использовать IP-адрес, указанный в разделе Service, и порт 31433. Ниже в этом примере я покажу вам хитрый способ подключения к Service для SQL Server.

Метка type – это тип службы (Service), в данном случае это сервис LoadBalancer, реализованный провайдером облака. Различные типы служб, которые можно использовать, можно найти по ссылке https://kubernetes.io/docs/concepts/services-networking/service/#publishingservices-service-types.

Когда вы запускаете kubectl с параметром apply и файлом YAML, выполнение часто происходит *асинхронно*. Это означает, что команда kubectl вернет результат немедленно, но операция, объявленная в файле YAML, будет запланирована сервером API в фоновом режиме.

В данном случае, когда вы выполните команду kubectl apply для этой службы, ваш результат должен выглядеть следующим образом:

```
service/mssql-service created
```

Этот результат фактически означает, что создание службы mssql было *запланировано*. Как узнать, готова ли служба к работе? Есть несколько способов. Сначала вы можете запустить следующую команду:

```
kubectl get service
```

Ваш результат может выглядеть так:

```
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP
    PORT(S)           AGE
mssql-service   LoadBalancer   10.0.150.233     <pending>
    31433:32010/TCP   61s
```

CLUSTER-IP – это внутренний IP-адрес в кластере k8s. EXTERNAL-IP будет статическим внешним IP-адресом, который вы можете использовать для подключения к SQL Server. Обратите внимание, что порт имеет значение 31433:32010. Хотя SQL Server прослушивает порт 1433 в контейнере, порт 32010 сопоставлен с портом 1433 в кластере. Порт 31433 сопоставлен с портом 32010, что позволяет подключаться к <EXTERNAL-IP>,<31433> для подключения к SQL Server, независимо от того, где именно объект pod с SQL Server находится в кластере k8s.

Обратите внимание, что значение параметра EXTERNAL-IP равно <pending>, когда вы смотрите на него сразу после выполнения команды kubectl apply. Вы не сможете использовать LoadBalancer, пока это значение не станет реальным IP-адресом, а это может занять несколько минут.

4. Теперь, когда создание LoadBalancer запланировано, давайте создадим объект secret для хранения пароля sa для SQL Server. Выполните следующую команду или сценарий **step4_create_secret.sh**:

```
kubectl create secret generic mssql-secret --from-literal=
SA_PASSWORD=»Sql2019isfast»
```

После выполнения этой команды вы должны получить следующие результаты (объект secret должен быть создан):

```
secret/mssql-secret created
```

Примечание. В главе 7 я упомянул, что в версии SQL Server 2019 будет поддерживаться аутентификация Active Directory для контейнеров SQL Server. Как только эта возможность будет реализована, мы сможем также поддерживать аутентификацию AD для SQL Server в k8s.

5. Следующим шагом является создание хранилища для баз данных SQL Server с использованием концепции **Persistent Volume Claim (PVC)**. PVC похоже на том, определенный поверх системы дисков, который мы можем использовать для сопоставления с каталогами баз данных для объекта pod с SQL Server.

Используйте следующую команду для создания PVC, который будет применяться для объекта pod с SQL Server, или выполните сценарий **step5_create_storage.sh**:

```
kubectl apply -f storage.yaml
```

Команда должна выполниться очень быстро, и в результате вы должны получить следующее сообщение о том, что запланировано создание PVC в фоновом режиме:

```
persistentvolumeclaim/mssql-data created
```

Пока PVC создается (это может произойти очень быстро), давайте исследуем структуру файла **storage.yaml**, чтобы посмотреть, что происходит за кулисами:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```
  name: mssql-data
  annotations:
    volume.beta.kubernetes.io/storage-class: azure-disk
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
    storage: 8Gi
```

Метаданные интересны тем, что метка annotations связывает PVC с диском, называемым классом хранилищ (storage class). Как я узнал, что мне нужно использовать значение azure-disk для этого параметра? Ответ на этот вопрос заключается в том, что при создании кластера AKS вы автоматически получаете диск Azure, размещенный в хранилище класса Premium, с классом хранилищ, называемым azure-disk. Вы можете создавать другие устройства для хранения данных, но это стандартный диск, созданный AKS. Вы можете узнать больше о хранилищах, используемых в AKS, перейдя по ссылке https://docs.microsoft.com/en-us/azure/aks/concepts-storage. Если вы не используете k8s, значит, вам уже нужно было создать классы хранилищ до выполнения этого шага. Вы можете узнать, какие классы хранилищ доступны, спросив об этом вашего администратора k8s или выполнив следующую команду:

```
kubectl get StorageClass
```

Вы можете узнать больше о свойствах azure-disk, выполнив следующую команду:

```
kubectl describe StorageClass azure-disk
```

Раздел spec файла storage.yaml описывает, какой доступ разрешен для PVC, а также указывает размер тома. Значение параметра AccessModes ReadWriteOnce означает, что том предназначен для чтения/записи и в любой момент доступ к тому разрешен только одному объекту pod/узлу. Это не означает, что объект pod нельзя переместить на другой узел и получить доступ к тому (это будет фундаментальной концепцией HA, как вы узнаете позже в данной главе). Это означает лишь то, что два объекта pod или два узла не могут одновременно получить доступ к одному и тому же тому. Такой подход оправдан для базы данных SQL Server. 8Gi означает объем хранилища 8 Гб.

Вы можете узнать, было ли создано PVC, выполнив следующую команду:

```
kubectl describe PersistentVolumeClaims mssql-data
```

Успешное создание PVC (см. сообщение в конце выведенного результата) выглядит следующим образом:

```
Type      Reason                  Age    From
                            Message
----      ------                  ----   ----
                            -------
Normal    ProvisioningSucceeded   9m16s  persistentvolumecontroller
Successfully provisioned volume pvc-b8c9225e-c038-11e9-b5fa-
c6f80bad26d8 using kubernetes.io/azure-disk
```

Я сталкивался с ситуациями, в которых при выполнении этой команды происходили некоторые временные задержки или выдавались ошибки, но в конечном итоге PVC создавалось.

6. Теперь у вас есть служба балансировки нагрузки, объект secret для хранения пароля sa и хранилище. Пришло время собрать все это воедино и создать объект pod, который будет содержать один контейнер для SQL Server. Запустите следующую команду или сценарий **step6_deploy_sql2019.sh** (параметр --record предоставляет дополнительную информацию о том, как выполняется развертывание):

```
kubectl apply -f sql2019deployment.yaml --record
```

Вы должны получить следующий результат, который указывает, что *развертывание* было запланировано:

```
deployment.apps/mssql-deployment created
```

Задание *развертывания* позволит нам создать объект pod с концепцией *ReplicaSet* для SQL Server. Я подробно расскажу о ReplicaSet позже, когда буду демонстрировать встроенные возможности HA в k8s.

Чтобы увидеть состояние задания развертывания и объектов, связанных с объектом pod, выполните следующую команду:

```
kubectl get all
```

В результате выполнения данной команды вы получите все объекты в текущем контексте (пространство имен mssql), включая состояние объекта pod, LoadBalancer и развертывания.

Когда вы запустите эту команду, то можете получить такие результаты:

```
NAME                                  READY   STATUS              RESTARTS  AGE
pod/mssql-deployment-7bb4c5f5d7-rpw45 0/1     ContainerCreating   0         4s

NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP
```

```
PORT(S)           AGE
service/mssql-service    LoadBalancer    10.0.150.233    13.77.103.119
31433:32010/TCP   55m

NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/mssql-deployment    0/1      1             0            4s

NAME                                          DESIRED  CURRENT     READY      AGE
replicaset.apps/mssql-deployment-7bb4c5f5d7   1        1           0          4s
```

Первая строка полученного результата содержит статус создания объекта pod. Значение ContainerCreating параметра STATUS, относящегося к объекту pod, означает, что контейнер, определенный для объекта pod, находится в процессе создания. Если вы создаете объект pod с образом контейнера SQL Server, это может занять больше времени, так как образ контейнера должен быть загружен в локальный реестр Docker k8s.

Значение параметра STATUS для развертывания (deployment) показывает, когда развертывание выполняется успешно. Значение параметра STATUS для LoadBalancer не зависит от развертывания. Это означает, что вам нужно подождать, пока для объекта pod будет установлено состояние **Running**, LoadBalancer будет иметь корректный адрес EXTERNAL-IP, а для развертывания будет указан статус AVAILABLE = 1, прежде чем вы сможете подключиться к SQL Server. Состояние реплики должно соответствовать состоянию объекта pod или развертывания.

Пока объект pod еще создается, давайте внимательно изучим различные фрагменты содержимого файла **sql2019deployment.yaml**.

```
kind: Deployment
metadata:
  name: mssql-deployment
```

Этот раздел сообщает API-серверу, что мы создаем задание развертывания объекта pod и имя объекта.

Остальная часть файла YAML – это спецификация развертывания. Давайте подробно разберем ее.

```
replicas: 1
  selector:
    matchLabels:
      app: mssql
  strategy:
    type: Recreate
```

Этот фрагмент определяет количество реплик, необходимых для объекта pod (в данном случае оно равно единице) и для повторного

создания объекта pod при необходимости каких-либо обновлений. Я объясню значение данных терминов позже в этой главе. Параметр matchLabels должен связать метку развертывания с меткой объекта pod (оба значения – mssql).

Следующий большой раздел файла называется шаблоном, он определяет более подробную информацию о контейнерах для объекта pod и о томах, используемых для хранения данных.

Вы можете увидеть в шаблоне, что мы используем другую метку для приложения mssql, что дает нам возможность управлять объектами в процессе развертывания, просматривать их или управлять ими с помощью этой метки.

Далее следует спецификация объекта pod, содержащая подробную информацию о контейнере, который необходимо развернуть внутри объекта pod.

```
spec:
        terminationGracePeriodSeconds: 10
        containers:
        - name: mssql
          image: mcr.microsoft.com/mssql/rhel/server:2019-latest
          env:
          - name: MSSQL_PID
            value: "Developer"
          - name: ACCEPT_EULA
            value: "Y"
          - name: MSSQL_SA_PASSWORD
            valueFrom:
            secretKeyRef:
              name: mssql-secret
              key: SA_PASSWORD
        volumeMounts:
        - name: mssqldb
          mountPath: /var/opt/mssql
```

В спецификации вы можете увидеть несколько фрагментов, которые выглядят знакомо, если вы используете контейнеры. Вы замечаете там образ, положенный в основу контейнера, и переменные среды, используемые для передачи контейнеру SQL Server. Обратите внимание, как пароль sa сопоставлен с объектом secret, который вы уже создали.

Значение параметра **terminationGracePeriodSeconds** определяет, сколько времени k8s отведет контейнеру на завершение работы, в случае если контейнер будет необходимо остановить. Для SQL Server может потребоваться значение, большее (или меньшее) использо-

ванного в приведенном примере, однако выбранное мной значение 10 секунд позволяет корректно завершить работу. SQL Server обеспечивает согласованность данных независимо от того, корректно ли завершилась его работа.

Запись **volumeMount** – это имя mssqldb, которое привязывается к каталогу для хранения всех баз данных SQL Server. Имя mssqldb определяется прямо под следующим фрагментом спецификации как параметр развертывания:

```
volumes:
        - name: mssqldb
          persistentVolumeClaim:
          claimName: mssql-data
```

Здесь было выполнено сопоставление с созданным PVC. Теперь, когда контейнер SQL Server запущен в объекте pod, все системные и пользовательские базы данных по умолчанию будут находиться в постоянном хранилище PVC. Вы увидите, насколько это важно, когда перейдете к другим разделам этой главы, посвященным темам высокой доступности и обновлений SQL Server.

Вы можете снова выполнить команду `kubectl get all`, чтобы увидеть, все ли готово для работы с SQL Server.

В вашем арсенале также имеется несколько других интересных команд kubectl для проверки состояния объекта pod и SQL Server. Чтобы увидеть полный список их параметров, вы можете выполнить команду `kubectl help` или обратиться к справочной документации по kubectl по ссылке https://kubernetes.io/docs/reference/kubectl/overview.

7. Выполните следующую команду, чтобы просмотреть журналы объекта pod (и размещенного в нем контейнера). Журнал, который вы увидите, выполнив ее, – это журнал ошибок (ERRORLOG) SQL Server. Вы также можете использовать сценарий **step7_getlogs.sh**:

```
kubectl logs -l app=mssql --tail=100000
```

Обычно для команды `kubectl logs` требуется имя объекта pod, но вы можете использовать метку mssql, а не искать имя объекта pod.

В результате выполнения этой команды на ваш экран должны быть выведены данные журнала ошибок SQL Server (ERRORLOG).

8. Вы также можете просмотреть подробное представление событий, связанных с операциями в k8s, которые вы выполнили в этом пространстве имен, выполнив следующую команду или сценарий **step8_getevents.sh**:

```
kubectl get events
```

В результате вы должны получить хронологию событий, которые произошли в этом пространстве имен. Если все команды были выполнены без ошибок, ваш результат должен выглядеть примерно так:

```
LAST SEEN        TYPE      REASON                    KIND
    MESSAGE
29m              Normal    ProvisioningSucceeded     PersistentVolumeClaim
    Successfully provisioned volume pvc-c18b530f-c040-11e9-b5fac6f80bad26d8
    using kubernetes.io/azure-disk
25m              Normal    Scheduled                 Pod
    Successfully assigned mssql/mssql-deployment-7b6565d684-8r7cctoaks-
    nodepool1-90949249-0
25m              Normal    SuccessfulAttachVolume    Pod
    AttachVolume.Attach succeeded for volume “pvc-c18b530f-c040-11e9-
    b5fa-c6f80bad26d8”
24m              Normal    Pulled                    Pod
    Container image "mcr.microsoft.com/mssql/rhel/server:2019-latest"
    already present on machine
24m              Normal    Created                   Pod
    Created container
24m              Normal    Started                   Pod
    Started container
25m              Normal    SuccessfulCreate          ReplicaSet
    Created pod: mssql-deployment-7b6565d684-8r7cc
25m              Normal    ScalingReplicaSet         Deployment
    Scaled up replica set mssql-deployment-7b6565d684 to 1
30m              Normal    EnsuringLoadBalancer      Service
    Ensuring load balancer
29m              Normal    EnsuredLoadBalancer       Service
    Ensured load balancer
```

9. Вы можете также получить более подробную информацию о развертывании, выполнив следующую команду или запустив сценарий **step9_describe_deployment.sh**:

```
kubectl describe deployment mssql-deployment
```

В результате вы получите все сведения о выполненном развертывании, включая самые последние события, связанные с ним.

10. Вы можете получить более подробную информацию о развернутом объекте pod, выполнив следующую команду или сценарий **step10_describe_pod.sh**:

```
kubectl describe pod -l app=mssql
```

Вот где использование метки mssql снова пригодится. Вы получите более подробную информацию об объекте pod, контейнерах внутри него и событиях, связанных с ним. Все, что вам нужно запомнить, – это метка mssql.

Эта команда вместе с `kubectl get events` может быть полезна для устранения ошибок при развертывании объекта pod.

11. Теперь (наконец) пришло время подключиться к SQL Server, работающему в контейнере, размещенном в объекте pod. Вот прием, который я узнал от Энтони Ночентино (Anthony Nocentino), известного эксперта по Linux, Containers и k8s и технического рецензента моей книги *Pro SQL Server в Linux*. Выполните следующую команду или сценарий **step11_testsql.sh**, чтобы подключиться к SQL Server через LoadBalancer:

```
SERVERIP=$(kubectl get service | grep mssql-service |
awk {'print $4'})
PORT=31433
sqlcmd -Usa -PSql2019isfast -S$SERVERIP,$PORT -Q"SELECT@@version"
```

В результате вы должны получить версию установленного SQL Server. Если вы внимательно посмотрите на эту команду, она динамически извлекает EXTERNAL-IP в LoadBalancer и находит IP-адрес, который будет использоваться как часть строки соединения. Вы, однако, можете пойти дальше и добавить синтаксический разбор, чтобы получить порт, выведенный при выполнении команды `kubectl get service`.

В этом разделе приведено множество подробных описаний выполняемых действий, потому что я хотел рассказать вам, что происходит за сценой k8s, и показать, как можно использовать YAML в качестве API для интерфейса программирования k8s.

Советы по k8s

Прежде чем перейти к следующему разделу, позвольте мне дать вам несколько практических советов по использованию других ресурсов с k8s. Поскольку у вас есть развернутый объект pod с SQL Server, мы будем использовать его.

Расширение k8s

В разделе «Подготовительные шаги для использования примеров, иллюстрирующих применение SQL Server и Kubernetes» этой главы я рассмотрел расширение Kubernetes для Visual Studio Code и показал, как установить его для Azure Data Studio (ADS).

Позвольте мне показать вам несколько примеров использования этого расширения с вашим развернутым объектом pod с SQL Server.

Во-первых, одним из преимуществ использования расширения k8s, а также описаний в YAML являются подсказки по использованию файлов YAML. Найдите файл sql2019deployment.yaml в каталоге **ch8_sql_on_k8s\deploy**. Используйте проводник ADS, чтобы найти файл.

Наведите указатель мыши на любой фрагмент файла YAML и просмотрите подсказки о различных ключевых словах, содержащихся в файле. Например, на рис. 8.3 показана подсказка для описания terminationGracePeriodSeconds.

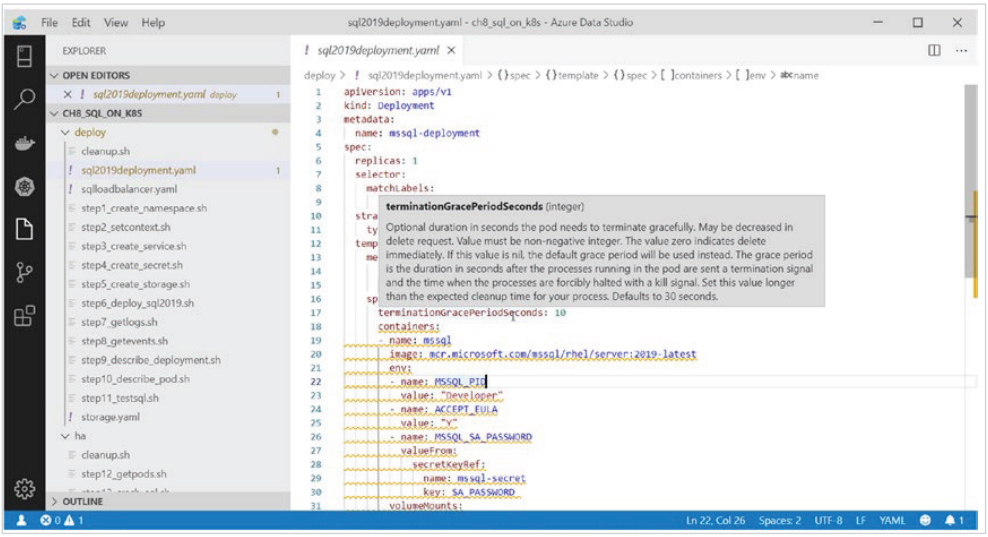


Рис. 8.3. Использование расширения k8s для изучения структуры файла YAML

Расширение ADS также включает в себя «живой» проводник для просмотра ресурсов k8s. Я использовал его для подключения к своему кластеру AKS (если вы работаете с AKS, вам потребуется ввести данные учетной записи для входа в систему, когда будете использовать этот инструмент). После подключения я мог просматривать объекты и даже выполнять некоторые операции.

Поскольку мой объект pod был развернут в пространстве имен, мне сначала нужно было изменить контекст на это пространство имен, как показано на рис. 8.4.

Одна замечательная вещь, которую я могу сделать, – это подключиться к работающему объекту pod и запустить оболочку Bash для просмотра журнала ошибок (ERRORLOG). Я сделал это буквально при помощи нескольких щелчков мыши – сначала я нашел свой объект pod в проводнике k8s, затем щелкнул правой кнопкой мыши, чтобы выбрать **Terminal**, как показано на рис. 8.5.

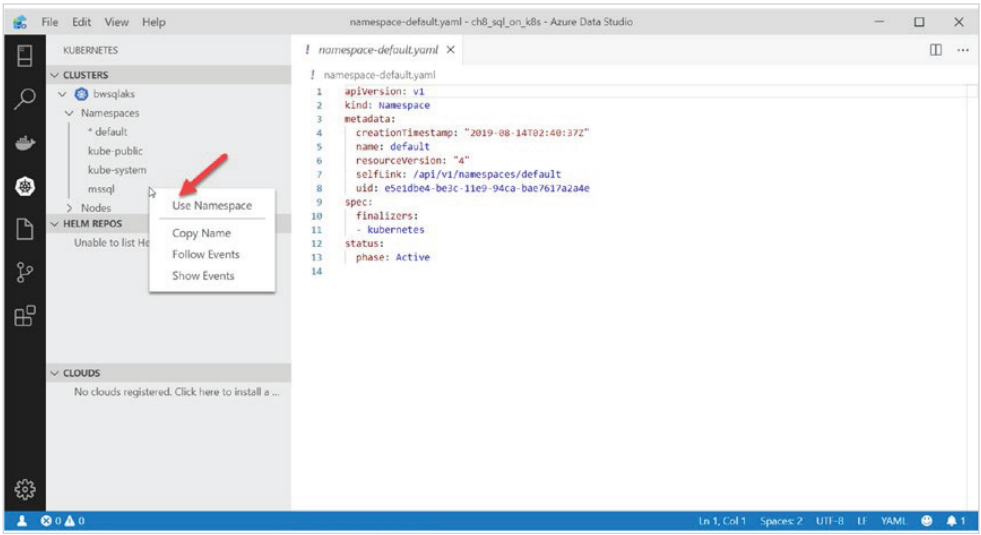


Рис. 8.4. Настройка пространства имен с расширением k8s

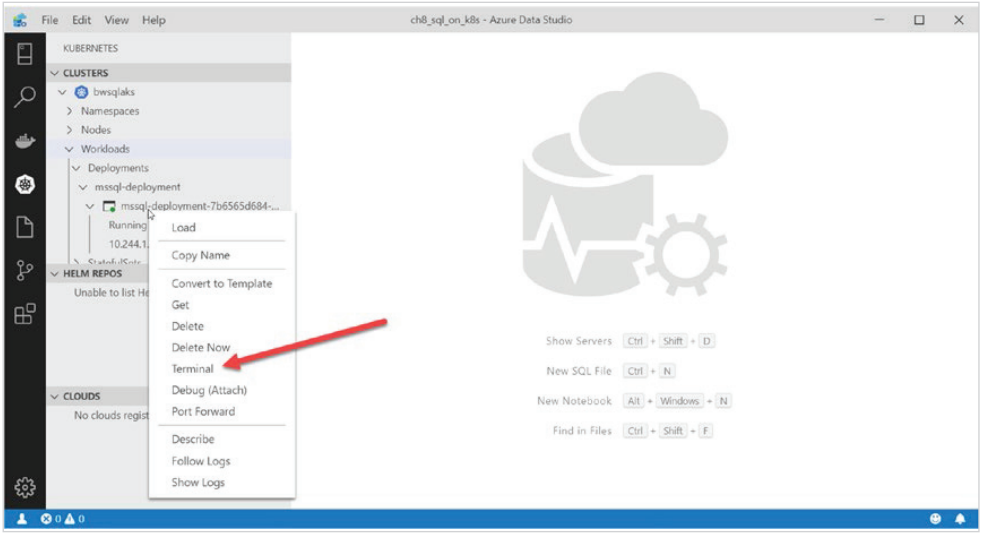


Рис. 8.5. Запуск сеанса работы с терминалом в объекте pod k8s

На экране моего компьютера появился терминал ADS, и теперь я находился в оболочке Bash в контейнере SQL объекта pod. Затем я смог перейти в /var/opt/mssql/log и вывести содержимое журнала ERRORLOG, как показано на рис. 8.6.

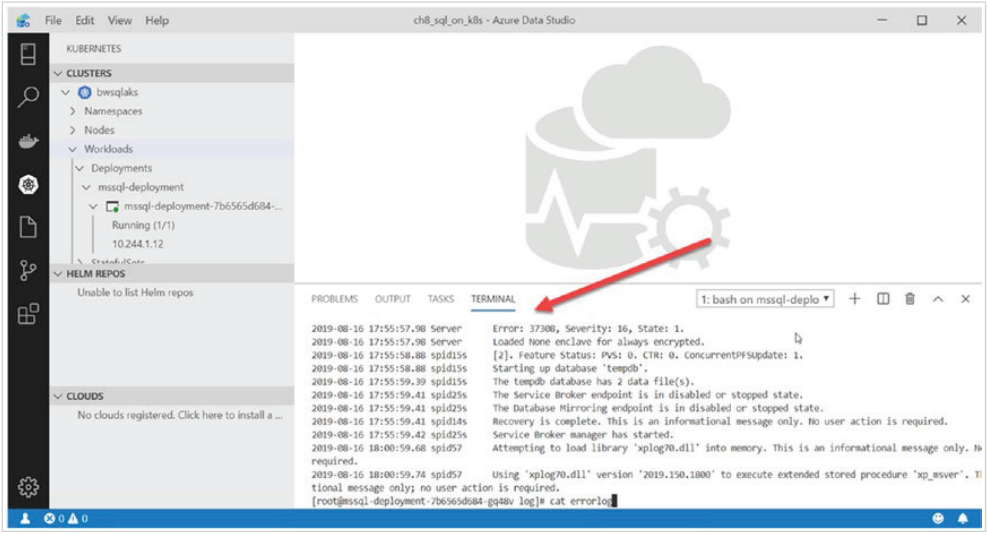


Рис. 8.6. Просмотр журнала ошибок (ERRORLOG) для объекта pod с SQL Server в k8s

Для завершения сеанса работы с терминалом я ввел команду exit. Еще одна интересная вещь, которую я обнаружил, – это возможность реинжиниринга выполненного развертывания, чтобы просмотреть содержимое файла YAML, использованного для этого развертывания. Щелкнув правой кнопкой мыши объект, полученный в результате развертывания, я выбрал команду **Convert to Template (Преобразовать в шаблон)**, как показано на рис. 8.7.

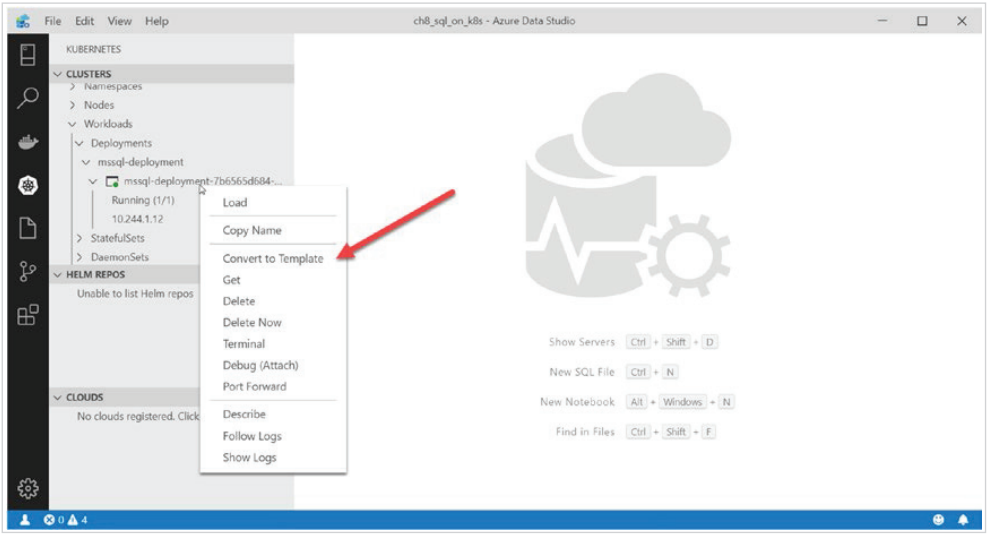


Рис. 8.7. Реинжиниринг развертывания K8S

Я ввел имя моего файла YAML, и полученный файл YAML был открыт в редакторе.

Я уверен, что есть и другие замечательные возможности расширения k8s, которые я еще не исследовал, и я буду продолжать открывать их, глубже знакомясь с k8s.

Другие команды kubectl

В kubectl есть много других команд, о которых стоит рассказать. Ниже приведен список этих команд.

- **kubectl top** – эта команда на основе узла или объекта pod отображает метрики для памяти и ЦП. Это может быть полезно, например, чтобы увидеть, сколько памяти использует объект pod или сколько памяти осталось на узле.
- **kubectl cp** – эту команду можно использовать для копирования файла в файловую систему контейнера для объекта pod. Точно так же, как команду docker cp, вы можете использовать эту команду для копирования файла резервной копии базы данных SQL Server в слой контейнера, доступный для записи.

Например, для выполненного вами примера по развертыванию SQL Server в контейнере предположим, что mssql-deploy-7b6565d684-92l8s – это имя объекта pod в пространстве имен mssql, и вы загрузили образец базы данных WideWorldImporters (https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImporters-Full.bak) в ваш локальный каталог. Следующая команда скопирует файл резервной копии в контейнер SQL Server, чтобы базу данных можно было восстановить:

```
kubectl cp ./WideWorldImporters-Full.bak mssql/mssql-deployment-
7b6565d684-92l8s:/var/opt/mssql
```

- **kubectl exec** – эта команда позволяет вам выполнить программу в пространстве имен контейнера в объекте pod. Эта команда во многом похожа на docker exec для запуска оболочки Bash для контейнера или утилиты sqlcmd в SQL Server, поскольку она является частью контейнера SQL Server.

Основываясь на примере, который я только что показал вам для копирования в контейнер резервной копии БД WideWorldImporters, для восстановления резервной копии можно использовать следующую команду:

```
kubectl exec mssql-deployment-7b6565d684-92l8s -- /opt/mssql-tools/bin/sqlcmd
-S localhost -U SA -P "Sql2019isfast" -Q “RESTORE DATABASE WideWorldImporters
FROM DISK = '/var/opt/mssql/WideWorldImporters-Full.bak' WITH MOVE 'WWI_
Primary' TO '/var/opt/mssql/data/WideWorldImporters.mdf', MOVE 'WWI_UserData'
TO '/var/opt/mssql/data/WideWorldImporters_userdata.ndf', MOVE 'WWI_Log' TO
```

```
'/var/opt/mssql/data/WideWorldImporters.ldf', MOVE 'WWI_InMemory_Data_1' TO
'/var/opt/mssql/data/WideWorldImporters_InMemory_Data_1'"
```

Мне потребовалось некоторое время, чтобы разобраться в синтаксисе данной команды; обратите внимание, что вы не указываете пространство имен, поэтому вы должны находиться в контексте пространства имен для вашего объекта pod. Также обратите внимание на использование символов -- перед указанием /opt/mssql/bin/sqlcmd; они применяются для разделения аргументов для kubectl и аргументов для программы (в данном случае это sqlcmd).

- **kubectl version** – эта команда выводит версию kubectl. Я видел ситуации, когда у пользователей были проблемы с kubectl, потому что используемая версия kubectl была старше и несовместима с версией кластера k8s. Эта команда выводит версии как клиента, так и сервера. Подробнее о совместимости версий читайте по ссылке https://kubernetes.io/docs/setup/release/version-skew-policy/.
- **kubectl explain** – эта команда выводит документацию с информацией об объектах k8s. Например, команда, приведенная ниже, позволяет получить дополнительную информацию о требованиях к файлу YAML для ReplicaSet:

  ```
  kubectl explain ReplicaSet
  ```

- **kubectl cluster-info dump** – осторожно куберинфо (это используемый термин? Если нет, я только что придумал его). Эта команда выведет огромный объем данных диагностики. Используйте синтаксис `--output-directory` для создания набора диагностических файлов. Обязательно используйте параметр `--all-namespaces` для диагностики всех пространств имен. Эта команда создает дамп практически любого файла журнала, который является частью кластера k8s, включая объекты pod. Я действительно не смог найти какой-либо документации о том, что находится в журналах, но поскольку я активно осваиваю k8s, то, вероятно, узнаю об этом больше.

Панель мониторинга K8S

Панель мониторинга Kubernetes отображает визуальную информацию о кластере k8s. Вы можете прочитать все о панели мониторинга, перейдя по ссылке https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/.

Для AKS вы можете прочитать, как вывести панель мониторинга k8s для вашего кластера, по ссылке https://docs.microsoft.com/en-us/azure/aks/kubernetes-dashboard. После того как я выполнил действия, описанные на этой странице, чтобы запустить панель мониторинга, в моем браузере появился пользовательский интерфейс. Затем я просто выбрал пространство имен mssql, и на моем экране появилась панель, приведенная на рис. 8.8.

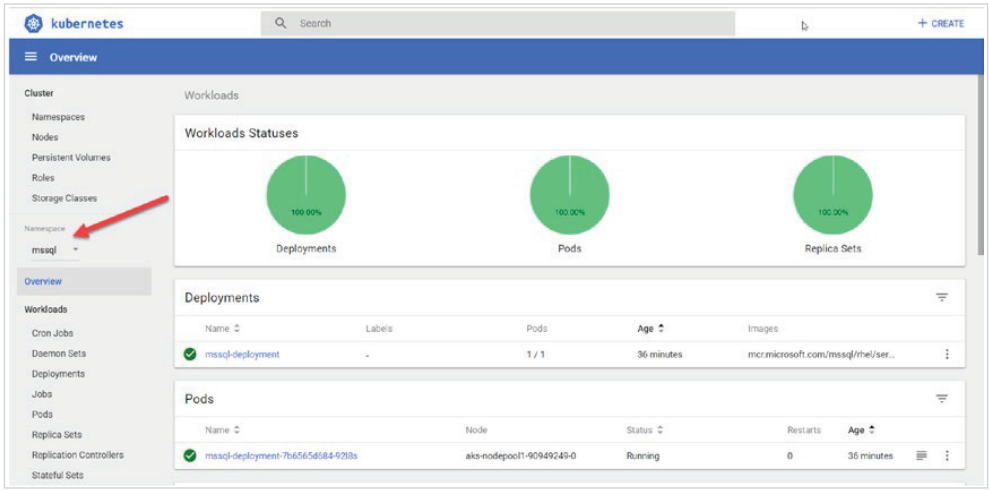


Рис. 8.8. Панель мониторинга K8S

Метрики и журналы в AKS

Использование AKS имеет свои преимущества, поскольку оно предоставляет возможности управляемой платформы k8s. Арсенал средств управления включает в себя встроенные метрики, инструменты визуализации и возможности просмотра журналов на портале Azure. На рис. 8.9 показаны некоторые возможности визуализации, доступные мне при помощи портала Azure с моим кластером AKS.

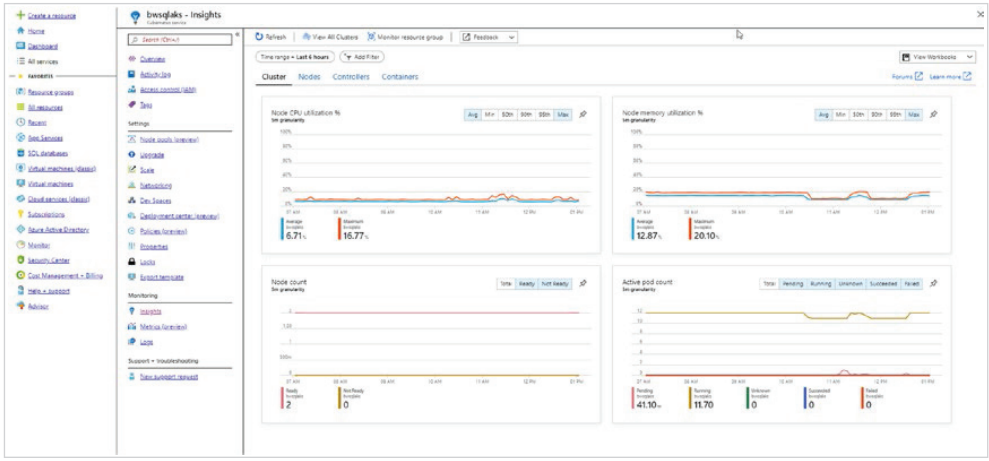


Рис. 8.9. Данные, представленные в графическом виде, на портале Azure для AKS

Следующий шаг, который я намерен предпринять в этой главе, – показать, как встроенные средства, обеспечивающие высокую доступность (HA), работают с k8s и как это применяется к SQL Server. Если вы собирае-

тесь использовать примеры в следующем разделе, не удаляйте те ресурсы, которые вы уже настроили. Сценарий **cleanup.sh** можно использовать для удаления всех ресурсов, поэтому вы можете использовать его, если не планируете выполнять на практике примеры из следующего раздела.

Высокая доступность SQL Server на k8s

Одним из самых замечательных аспектов k8s является встроенный набор функций для обеспечения высокой доступности. Представьте себе высокую доступность SQL Server без программного обеспечения для кластеризации, которое вам необходимо устанавливать и поддерживать!

Я уже упоминал термин **ReplicaSet** ранее в этой главе, и теперь пришло время поговорить о его значении.

Когда вы выполнили развертывание SQL Server в примере из предыдущего раздела, в файле YAML содержалось следующее объявление:

```
replicas: 1
```

Это объявление указывает k8s следующее: k8s всегда *должен пытаться гарантировать*, что один экземпляр контейнера в объекте pod (который в этом случае содержит контейнер SQL Server) всегда работает. Если в работе контейнера происходит сбой, то k8s перезапустит контейнер. Если объект pod прекращает существовать, то k8s создает новый объект pod; если узел прекращает работать, k8s разворачивает новый объект pod на новом узле, если таковой существует (и доступны ресурсы, позволяющие это сделать).

В SQL Server, когда вы объединяете ReplicaSet с LoadBalancer и постоянным хранилищем, все эти компоненты естественным образом обеспечивают высокую доступность совместно используемого хранилища. На рис. 8.10 представлено визуальное представление развертывания, которое вы выполнили в предыдущем разделе.

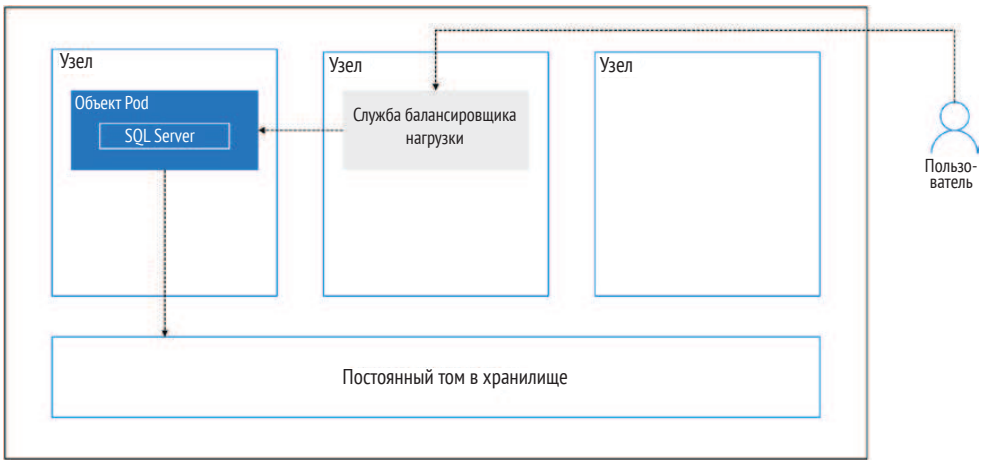


Рис. 8.10. Высокая доступность для SQL Server и k8s

В этом примере пользователь подключается к балансировщику нагрузки, который связан с объектом pod, в котором находится контейнер SQL Server (на самом деле балансировщик нагрузки не размещен на узле, но ради простоты мы изобразили его таким образом). Если произойдет аварийное завершение работы контейнера SQL Server, вы увидите только небольшое отклонение параметров нагрузки, поскольку k8s переключается на другой контейнер в объекте pod.

Подумайте, что произойдет, если объект pod внезапно станет недоступен, как показано на рис. 8.11.

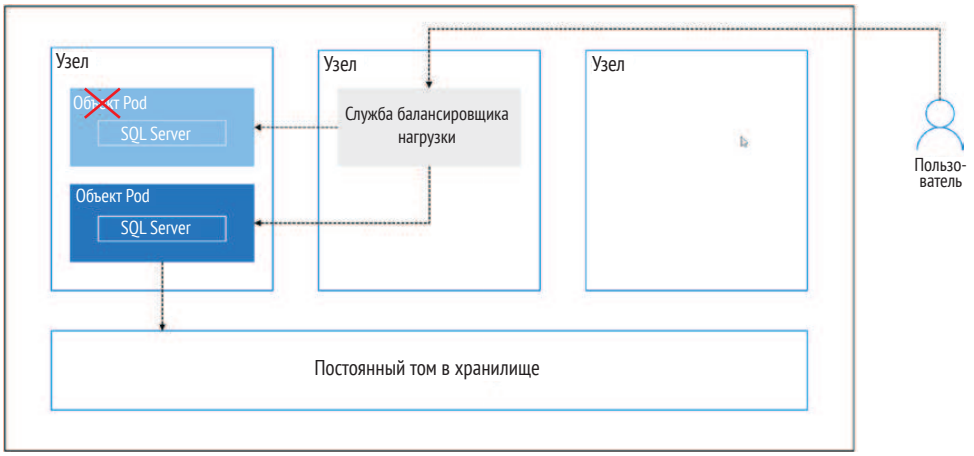


Рис. 8.11. Внезапное отключение объекта pod в k8s

В этом сценарии k8s создает и запускает другой объект pod (скорее всего, на том же узле), который запускает новый контейнер. Но обратите внимание, контейнер по-прежнему указывает на PVC, который привязан к системным и пользовательским базам данных. Для SQL Server он просто видит существующие системные и пользовательские базы данных и запускается. Новый объект pod будет иметь новый внутренний IP-адрес, но балансировщик нагрузки автоматически перенаправляется на этот новый адрес. С точки зрения приложения, это просто повторная попытка соединения, и все происходит незаметно для пользователя.

Что происходит, если узел (это может быть виртуальная машина) дает сбой, показано на рис. 8.12.

k8s обнаружит эту проблему и запустит новый объект pod на новом узле. Даже если новый объект pod имеет новый внутренний IP-адрес, балансировщик нагрузки будет перенаправлен на новый объект pod. Этот сценарий аналогичен тому, как работает экземпляр отказоустойчивого кластера Always On (Always On Failover Cluster Instance), за исключением того, что вам не нужно устанавливать никакого специального программного обеспечения для кластеризации.

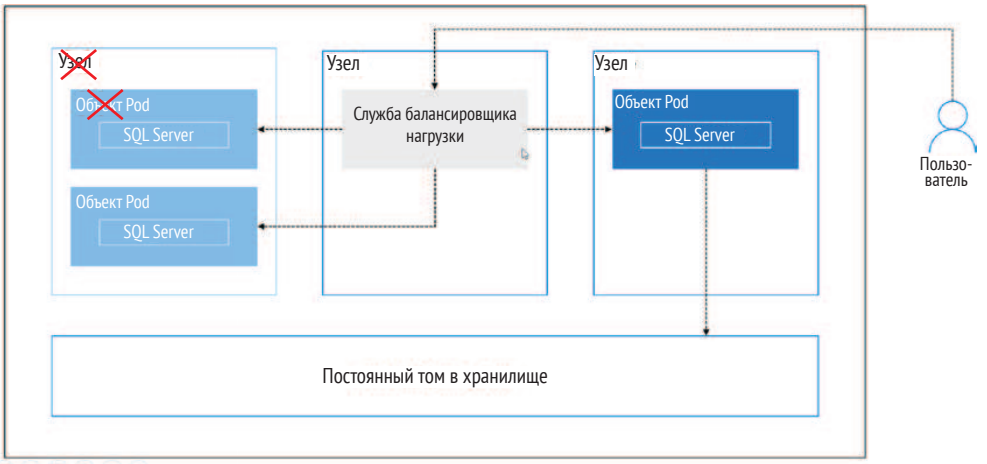


Рис. 8.12. Сбой узла в k8s

Давайте продолжим выполнять практические задания, используя пример из предыдущей главы, чтобы увидеть, как это работает. Все сценарии для этого примера размещены в каталоге **ch8_sql_on_k8s\ha**.

1. Запустите следующую команду или **step12_getpods.sh**, чтобы увидеть имя модуля, IP-адрес и узел, на котором работает объект pod:

```
kubectl get pods -o wide
```

Полученный результат должен выглядеть примерно так:

```
NAME                                READY        STATUS        RESTARTS
    AGE    IP            NODE                        NOMINATED NODE
    READINESS GATES
mssql-deployment-7b6565d684-8r7cc   1/1          Running       0
    91m    10.244.1.11   aks-nodepool1-90949249-0    <none>        <none>
```

2. Давайте смоделируем сбой контейнера, завершив работу SQL Server. Выполните следующую команду или сценарий **step13_crash_sql.sh**, чтобы завершить работу SQL Server и тем самым остановить контейнер:

```
SERVERIP=$(kubectl get service | grep mssql-service |
awk {'print $4'})
PORT=31433
sqlcmd -Usa -PSql2019isfast -S$SERVERIP,$PORT -Q"SHUTDOWN WITH NOWAIT"
```

3. Запустите следующую команду или сценарий **step14_getpods.sh**, чтобы увидеть все то же самое:

```
kubectl get pods -o wide
```

Вы должны получить результат, схожий с тем, который получили раньше, поскольку контейнер был перезапущен в том же объекте pod на том же узле:

```
NAME                                READY         STATUS         RESTARTS
    AGE    IP             NODE                          NOMINATED NODE
    READINESS GATES
mssql-deployment-7b6565d684-8r7cc   1/1           Running        1
    91m    10.244.1.11    aks-nodepool1-90949249-0     <none>        <none>
```

Выполните следующую команду, чтобы просмотреть последовательность событий:

```
kubectl get events
```

Ваш результат должен выглядеть примерно так:

```
LAST SEEN    TYPE      REASON      KIND      MESSAGE
16s          Normal    Pulled      Pod       Container image
"mcr.microsoft.com/mssql/rhel/server:2019-latest" already
present on machine
16s          Normal    Created     Pod       Created container
16s          Normal    Started     Pod       Started container
```

4. Попробуйте подключиться к SQL Server и убедитесь, что он работает, выполнив следующую команду или сценарий **step15_testsql.sh**:

```
SERVERIP=$(kubectl get service | grep mssql-service |
awk {'print $4'})
PORT=31433
sqlcmd -Usa -PSql2019isfast -S$SERVERIP,$PORT -Q"SELECT@@version"
```

Вы должны увидеть, что можете подключиться к SQL Server, и в результате выполнения команды должна быть выведена версия SQL Server.

5. Сэмулируйте сбой объекта pod с помощью следующей команды или сценария **step16_pod_failure.sh**:

```
kubectl delete pod -l app=mssql
```

В этом примере мы можем не указывать имя объекта pod, а воспользоваться тем, что мы связали метку, которую легко запомнить, с объектом pod.

Вы должны увидеть сообщение, подобное следующему:

```
pod "mssql-deployment-7b6565d684-8r7cc" deleted
```

6. Проверьте состояние объекта pod, включая IP-адрес, выполнив следующую команду или сценарий **step17_getpods.sh**:

```
kubectl get pods -o wide
```

Из полученного результата видно, что объект pod теперь работает на том же узле (его создание можно было запланировать на новом узле) с новым именем и новым IP-адресом:

```
NAME                                READY    STATUS    RESTARTS
    AGE     IP            NODE                        NOMINATED NODE
    READINESS GATES
mssql-deployment-7b6565d684-gq48v    1/1      Running   0
    2m55s   10.244.1.12   aks-nodepool1-90949249-0   <none>        <none>
```

Проверьте последовательность событий, используя следующую команду:

```
kubectl get events
```

Полученный вами результат покажет последовательность завершения работы объекта pod и создания нового. Примерный вид результата выполнения команды приведен ниже.

```
LAST SEEN        TYPE        REASON            KIND        MESSAGE
6m53s            Normal      Pulled            Pod         Container image
"mcr.microsoft.com/mssql/rhel/server:2019-latest" already present on machine
6m53s            Normal      Created           Pod         Created container
6m53s            Normal      Started           Pod         Started container
39s              Normal      Killing           Pod         Killing container
with id docker://mssql:Need to kill Pod
40s              Normal      Scheduled         Pod         Successfully
assigned mssql/mssql-deployment-7b6565d684-gq48v to aks-nodepool1-90949249-0
34s              Normal      Pulled            Pod         Container image
"mcr.microsoft.com/mssql/rhel/server:2019-latest" already present on machine
34s              Normal      Created           Pod         Created container
34s              Normal      Started           Pod         Started container
40s              Normal      SuccessfulCreate  ReplicaSet  Created pod:
mssql-deployment-7b6565d684-gq48v
```

7. Попробуйте подключиться к SQL Server с помощью службы LoadBalancer, выполнив следующую команду или сценарий **step18_testsql.sh**:

```
SERVERIP=$(kubectl get service | grep mssql-service |
awk {'print $4'})
PORT=31433
```

```
sqlcmd -Usa -PSql2019isfast -S$SERVERIP,$PORT -Q"SELECT
@@version"
```

Поскольку мы используем LoadBalancer, привязанный к объекту pod, в параметрах соединения не выполняется никаких изменений, даже если объект pod имеет новый внутренний IP-адрес.

8. Очистите все ресурсы, выполнив следующие команды или сценарий **cleanup.sh**:

```
kubectl delete namespace mssql
kubectl config delete-context mssql
kubectl config use-context bwsqlaks
```

Теперь вы познакомились с основными возможностями высокой доступности SQL Server, работающего в k8s. Для сценария, в котором узел больше не доступен, для имитации настоящего сбоя узла потребуется прямой доступ к виртуальным машинам, поддерживающим узел, и останов узла. Однако вы можете увидеть, как k8s будет автоматически планировать развертывание SQL Server на основе определения ReplicaSet, выполнив следующую команду:

```
kubectl drain <nodename>
```

Вы можете вернуть свой узел в рабочий режим для планирования (однако это не означает, что объекты pod будут перемещены на этот узел), выполнив следующую команду:

```
kubectl uncordon <nodename>
```

Теперь давайте рассмотрим, как вы можете обновить SQL Server в k8s подобно тому, как вы обновляли контейнер в главе 7.

Обновление SQL Server на k8s

В главе 7 вы узнали, как обновить контейнер SQL Server, «переключая» контейнеры, работающие с постоянным томом. Работающий контейнер останавливается, и запускается новый контейнер с новой сборкой CU SQL Server, указывающий на тот же том, который сопоставлен с каталогом, содержащим системные и пользовательские базы данных.

В k8s вы можете получить такой же результат. Только в этот раз k8s сделает всю работу за вас, учитывая правильные объявления параметров. Давайте вернемся к следующему разделу файла sql2019deployment.yaml из первого упражнения этой главы:

```
spec:
  replicas: 1
  selector:
```

```
    matchLabels:
      app: mssql
  strategy:
    type: Recreate
```

Обратите внимание на фрагмент `strategy: type: Recreate`. **Recreate** объявляет k8s, что если развернутые объекты обновлены, необходимо остановить и заново создать контейнер. Другой вариант для объявления `strategy: type:` – **RollingUpdate**. Мы не можем использовать его с SQL Server, если у нас отсутствует определенное синхронизированное взаимодействие с несколькими контейнерами SQL Server. Мы поговорим об этом понятии, однако, в последнем разделе данной главы, где речь пойдет о группах доступности Always On (Always On Availability Groups) и k8s.

Одним из способов обновления развертывания является обновление образа контейнера, запущенного в объекте pod. Для SQL Server это может означать обновление с использованием нового накопительного обновления; оно выполняется аналогично тому примеру, в котором я показал вам, как переключиться на контейнер с новым образом в главе 7. А поскольку мы используем постоянный том, новый контейнер распознает систему и пользовательскую базу данных, когда мы будем использовать обновленный образ. k8s предоставляет метод, позволяющий выполнить все это *при помощи одной команды*. Также возможно выполнить откат, так как k8s отслеживает обновление для развернутых объектов как *новую версию*.

Давайте рассмотрим эти возможности на практическом примере. Все сценарии для данного примера можно найти в каталоге **ch8_sql_on_k8s\update**. Если вы выполняли практические задания из предыдущих примеров, обязательно очистите все существующие ресурсы с помощью сценария **cleanup.sh**, находящегося либо в каталоге **ha**, либо в каталоге **deploy**.

На момент написания этой книги мы еще не поставили накопительное обновление для SQL Server 2019, поэтому в этих примерах я буду использовать SQL Server 2017. Однако как только мы начнем поставлять сборки CU, вы можете использовать те же приемы для SQL Server 2019.

1. Сначала нам нужно развернуть объект pod SQL Server аналогично тому, как мы проделали это в первом примере данной главы. Чтобы не выполнять по одному все необходимые для этого шаги, я создал один сценарий, который делает все необходимое, под названием **step1_deploysql.sh**.

   Этот сценарий выполняет следующие команды:

   ```
   kubectl create namespace mssql
   kubectl config set-context mssql --namespace=mssql
   --cluster=bwsqlaks --user=clusterUser_bwaks_bwsqlaks
   kubectl config use-context mssql
   kubectl apply -f sqlloadbalancer.yaml
   ```

```
kubectl create secret generic mssql --from-literal=SA_
PASSWORD="Sql2017isfast"
kubectl apply -f storage.yaml
kubectl apply -f sql2017deployment.yaml
```

Файлы **storage.yaml** и **sqlloadbalancer.yaml** идентичны файлам, использованным в первом примере этой главы. Файл **sql2017deployment.yaml** также почти не изменился, за исключением следующего раздела:

```
image: mcr.microsoft.com/mssql/server:2017-CU10-ubuntu
```

Это означает, что наш новый объект pod с контейнером будет использовать образ SQL Server 2017 CU10 для Ubuntu. Если этот образ не находится на узле, развернутом для объекта pod, k8s сначала должен будет извлечь образ.

Примечание. На сегодняшний день я не нашел ни одного простого способа предварительного извлечения образов SQL Server на всех узлах k8s, за исключением запуска объектов pod с использованием этих образов и последующего удаления объектов pod (образ SQL Server при этом остается в кеше на локальном узле). Существуют и другие методы, и один из них заключается в том, чтобы получить доступ администратора для входа на виртуальные машины узла Linux и непосредственного использования Docker для извлечения образов.

Используйте те же методы, что и в первом примере, чтобы убедиться, что объект pod и развернутые образы работают. Например, выполните следующую команду:

```
kubectl get all
```

Прежде чем вы сможете продолжать выполнять дальнейшие шаги, вам необходимо убедиться, что в столбце состояния объекта pod, которое будет указано в возвращаемом результате выполнения команды, указано Running и для LoadBalancer указан правильный адрес External-IP.

2. Теперь мы хотим обновить развернутую конфигурацию, изменив образ с помощью следующей команды или сценария **step2_updatesql.sh**:

```
kubectl --record deployment set image mssql-deployment
mssql=mcr.microsoft.com/mssql/server:2017-latest-ubuntu
```

k8s выполнит все необходимые действия без участия пользователя, чтобы остановить текущий контейнер и запустить новый (используя те же аргументы) с новым образом.

3. Используйте следующую команду или сценарий **step3_checkstatus.sh**, чтобы следить за ходом обновления. Эта команда не будет выполнена до тех пор, пока не будет завершено обновление контейнера с новым образом и контейнер снова не запустится:

```
kubectl rollout status deployment mssql-deployment
kubectl rollout history deployment mssql-deployment
```

Когда новое развертывание будет завершено, ваш результат будет выглядеть примерно так:

```
Waiting for deployment "mssql-deployment" rollout to finish:
0 out of 1 new replicas have been updated...
Waiting for deployment "mssql-deployment" rollout to finish:
0 out of 1 new replicas have been updated...
Waiting for deployment "mssql-deployment" rollout to finish:
0 out of 1 new replicas have been updated...
Waiting for deployment "mssql-deployment" rollout to finish:
0 of 1 updated replicas are available...
deployment "mssql-deployment" successfully rolled out
deployment.extensions/mssql-deployment
REVISION CHANGE-CAUSE
1        <none>
2        kubectl deployment set image mssql-deployment mssql=mcr.
microsoft.com/mssql/server:2017-latest-ubuntu --record=true
```

4. Вы можете убедиться, что ваш объект pod снова активен, с помощью этой команды или сценария **step4_getpods.sh**:

```
kubectl get pods -o wide
```

Статус объекта pod должен отображаться как Running.

5. SQL Server распознает существующие системные и пользовательские базы данных, но должен будет выполнить все необходимые шаги для обновления до новой сборки CU. Поэтому если вы попытаетесь подключиться к SQL Server слишком быстро, то можете получить следующую ошибку:

```
Sqlcmd: Error: Microsoft ODBC Driver 17 for SQL Server : Login
failed for user ‘sa’. Reason: Server is in script upgrade mode.
Only administrator can connect at this time.
```

Попробуйте несколько раз выполнить следующую команду или сценарий **step5_testsql.sh**, пока не увидите, что запущена новая версия SQL Server:

```
SERVERIP=$(kubectl get service | grep mssql-service |
awk {'print $4'})
PORT=31433
sqlcmd -Usa -PSql2017isfast -S$SERVERIP,$PORT -Q"SELECT@@version"
```

6. Как и в примере с контейнерами в главе 7, вы можете выполнить откат, восстановив предыдущий контейнер. k8s позволяет сделать это, изменив номер версии. Выполните следующую команду для отката к предыдущей сборке CU или используйте сценарий **step6_rollbacksql.sh**:

```
kubectl rollout undo deployment mssql-deployment --to-revision=1
```

7. Выполните следующую команду или сценарий **step7_getpods.sh**, чтобы убедиться, что состояние объекта pod снова вернулось в Running:

```
kubectl get pods -o wide
```

8. После запуска объекта pod выполните следующую команду (или используйте сценарий **step8_testsql.sh**), чтобы убедиться, что был выполнен откат до SQL 2017 CU10:

```
SERVERIP=$(kubectl get service | grep mssql-service |
awk {'print $4'})
PORT=31433
sqlcmd -Usa -PSql2017isfast -S$SERVERIP,$PORT -Q"SELECT@@version"
```

Вы успешно обновили контейнер SQL Server и выполнили откат изменений, используя встроенные возможности k8s для обновления образов из работающего контейнера.

Используйте сценарий **cleanup.sh** для очистки всех ресурсов, которые вы развернули, выполняя практическое задание из этого примера.

Использование Helm Charts

Процесс развертывания объекта pod с контейнером в k8s довольно прост, но, как вы видели на предыдущих примерах, для его завершения требуется выполнить много шагов. Было бы неплохо развернуть контейнер, такой как SQL Server, в объекте pod, аналогично тому, как выполняется установка из диспетчера пакетов (например, yum на RHEL).

Такую возможность предоставляет Helm. Вы можете прочитать о том, как использовать Helm Charts, перейдя по ссылке https://helm.sh/.

Helm Chart для SQL Server 2017 для Linux доступен по ссылке https://github.com/helm/charts/tree/master/stable/mssql-linux.

Когда вы установите Helm в своем кластере k8s, то сможете развернуть SQL Server, используя всего лишь одну команду, аналогичную приведенной ниже:

```
helm install --name sql-server stable/mssql-linux --set acceptEula.value=Y
--set sapassword=Sql2019isfast --set edition.value=Developer
```

Примеры, приведенные по ссылке https://github.com/helm/charts/tree/master/stable/mssql-linux, показывают, как настроить установку с сохранением объектов и как подключиться к работающему объекту pod, используя встроенный LoadBalancer.

Я думаю, что в арсенале Helm есть прекрасные возможности для упрощения работы k8s с SQL Server, поэтому будет интересно продолжить изучение этой технологии в сочетании с k8s для развертывания SQL Server.

Группы доступности SQL Server в k8s

Встроенное решение высокой доступности для k8s хорошо соответствует требованиям к доступности SQL Server. Однако при использовании этого подхода для обеспечения высокой доступности вашей базы данных возникает несколько проблем.

- Если k8s должен развернуть новый объект pod и контейнер, это фактически означает перезапуск SQL Server. Для всех системных и пользовательских баз данных должно выполняться полное восстановление. В зависимости от того, как была завершена работа контейнера (и если вы не используете нашу новую возможность ускоренного восстановления базы данных), это может привести к увеличению ожидаемого времени запуска нового контейнера SQL Server (даже если объект pod находится в состоянии Running). Работающий объект pod означает, что процесс sqlservr запущен, но это не значит, что SQL Server в действительности доступен.
- Вторая проблема – это время, которое может потребоваться для извлечения нового образа SQL Server. Если образы SQL Server предварительно не извлекаются на узле, где для SQL Server создается новый объект pod, это может привести к задержке запуска объекта pod и временной недоступности SQL Server.
- k8s *знает* только о состоянии контейнера, объекта pod или узла. Однако k8s ничего не знает о работоспособности программы, запущенной в контейнере. Контейнер SQL Server может быть запущен, но недоступен (либо база данных недоступна) из-за проблемы непосредственно с самим SQL Server.

  Мы создали группы доступности Always On (Always On Availability Groups), чтобы уменьшить время простоя для обеспечения доступности в случае сбоя. Частью этой технологии является распознавание условий перехвата управления при отказе за пределами работоспособности хост-узла для SQL Server. Вы можете прочитать больше об этих условиях аварийного переключения по ссылке https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/flexible-auto-

matic-failover-policy-availability-group. Кроме того, мы добавили состояние отработки отказа для баз данных в группах доступности, о которых вы можете прочитать подробнее по ссылке https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/sql-server-always-on-database-health-detection-failover-option?view=sql-server-ver15.

- Четвертая проблема заключается в том, что если у вас имеется один объект pod для sqlservr, отсутствует понятие реплики. Только один объект pod за один раз может получить доступ к системной и пользовательской базам данных. Было бы неплохо, чтобы в решении высокой доступности было задействовано более одного SQL Server, дабы другие экземпляры (реплики) могли считывать копии данных и все контейнеры не полагались бы на одно хранилище PVC.

  Поэтому для нас имеет смысл найти способ объединить встроенную HA k8s с технологией отработки отказа SQL Server, реализуемой при помощи групп доступности. И мы сделали это во время предварительного пользовательского тестирования SQL Server 2019. Вы можете прочитать историю о том, как мы это сделали и как это работает, в следующей заметке моего коллеги Сураба Агарвала (Sourabh Agarwal) по ссылке https://cloudblogs.microsoft.com/sqlserver/2018/12/10/availability-groups-on-kubernetes-in-sql-server-2019-preview/.

  Методология заключается в том, что мы используем концепцию StatefulSet для k8s (подробнее см. https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/) для развертывания реплик группы доступности. Мы также будем использовать концепцию *оператора* для организации развертывания группы доступности (Availability Group) и выявления сценариев отработки отказа.

  Кроме того, мы разработали это решение для использования служб LoadBalancer как для первичной, так и для вторичной реплики. Таким образом, приложение может подключиться к первичной реплике с помощью основного LoadBalancer, независимо от того, какая реплика была первичной. При этом другое приложение (в частности, приложение для создания отчетов) может подключаться к одной или нескольким вторичным репликам для чтения и использовать k8 для правильной балансировки нагрузки соединений. Мы также создали новый контейнер, называемый *AG Agent*, который размещен внутри объекта pod SQL Server, чтобы помочь обнаруживать и координировать сценарии отработки отказа для SQL Server. В сочетании с концепцией под названием k8s ConfigMap (дополнительную информацию вы можете получить по ссылке https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/) AG Agent и оператор помогли бы интегрировать решения, направленные на обеспечение высокой отказоустойчивости, с кластером k8s для сценариев, выходящих за рамки отслеживания состояния k8s (контейнер, узел или объект pod).

Все эти возможности основаны на компонентах, которые мы создали во время подготовки к выпуску SQL Server 2019 для предварительного пользовательского тестирования; мы объявили, что группы доступности для k8s не войдут в версию SQL Server 2019. Однако группы доступности являются частью решений HADR для кластеров больших данных, о которых будет рассказываться в главе 10.

Я разговаривал с Россом Монстером (Ross Monster), ведущим разработчиком этой функции. Он сказал мне, что все еще намерен развивать эту функцию в будущем. Росс сказал мне, что основная идея ее развития такова, чтобы в конечном итоге использовать концепцию оператора, концепцию агента AG и концепцию StatefulSet, однако общий дизайн может измениться. После развертывания группы доступности (Availability Group, AG) AG будет вести себя так же, как AG, не использующая k8s; это, в частности, обеспечит вам возможность читать вторичные реплики и иметь аналогичные возможности отработки отказа. Опять же, одно из больших преимуществ k8s с AG заключается в том, что вам не потребуется устанавливать и обслуживать программное обеспечение отказоустойчивого кластера.

Если вы хотите выполнить практическое упражнение с использованием предварительной версии AG с k8s, обратите внимание на модуль 5 SQL Server 2019 в OpenShift lab (см. https://github.com/microsoft/sqlworkshops/blob/master/SQLonOpenShift/sqlonopenshift/05_Operator.md).

Росс объяснил, что планируется включить в число возможностей SQL Server 2019 с k8s сценарий непрерывного обновления. Таким образом, вместо того чтобы выполнять переключение контейнеров вручную, что в конечном итоге увеличивает время простоев, мы могли бы потенциально представить сценарий работы практически без простоев для обновления множества контейнеров SQL Server в группе доступности. Этот дивный новый мир будет очень похож на уже реализованную нами возможность, которую Red Hat продемонстрировал на конференции Red Hat Summit в мае 2019 года. Вы можете просмотреть видео этой демонстрации, размещенное по ссылке www.pscp.tv/RedHatOfficial/1vAGRWYPjngJl, познакомиться с этим новым миром операторов и увидеть, как обеспечивается практически нулевое время простоя SQL Server.

Резюме

Я уверен, что контейнеры и Kubernetes – это будущее распределенных и масштабируемых вычислений. И мы создали SQL Server, который станет частью этого будущего. В этой главе вы изучили достаточно основных понятий Kubernetes (k8s), чтобы разобраться, как развернуть SQL Server в кластере k8s. Вы также смогли увидеть мощь технологий высокой доступности (HA), реализованных в k8s, и посмотрели, как это можно использо-

вать в SQL Server. Как вы узнали из главы 7, посвященной контейнерам, вы можете использовать возможности k8s для обновления контейнера SQL Server в объекте pod – установки нового накопительного обновления (CU) и отката выполненных изменений при необходимости. Я кратко познакомил вас с Helm Charts, которые представляют новый метод развертывания объектов pod и контейнеров в k8 с использованием подхода управления пакетами.

Наконец, я дал вам представление о будущем SQL Server с k8s, поскольку мы интегрируем группы доступности с k8s, чтобы получить решение высокой доступности для SQL Server, включая все возможности групп доступности, которые имеются на сегодняшний день. Главы 7 и 8 являются важными основами для выполнения практических упражнений в главе 10, посвященной кластерам больших данных SQL Server.

Глава 9

Виртуализация данных в SQL Server

Виртуализация данных – одна из самых захватывающих возможностей в SQL Server 2019. В этой главе вы узнаете больше о том, как благодаря технологии Polybase в SQL Server 2019 появилась виртуализация данных. Эта глава наряду с главами 6, 7 и 8 закладывает основы для изучения кластеров больших данных SQL Server, которым посвящена глава 10.

Что такое Polybase?

Polybase – это инновационная технология, представленная в SQL Server 2016 и усовершенствованная в SQL Server 2019, позволяющая решить проблему перемещения данных. Перемещение данных обычно включает использование дорогостоящих и сложных процессов извлечения, преобразования и загрузки данных (Extract, Transform, and Load, ETL) из других источников данных в SQL Server. Polybase решает эту проблему, используя концепцию *виртуализации данных* (Data Virtualization). Виртуализация данных – термин, который я буду обсуждать и разъяснять по мере продвижения вперед. В этой главе я подробно расскажу о виртуализации данных.

Здесь я поведаю вам историю возникновения Polybase и покажу, как она реализует виртуализацию данных. Я расскажу о том, как Polybase устроена «внутри», и о типичном сценарии работы с Polybase с использованием внешних таблиц. И как и в большинстве глав этой книги, мы будем использовать примеры, чтобы показать вам детали того, как применять Polybase для ваших потребностей в виртуализации данных.

Вы можете использовать нашу документацию в качестве руководства по Polybase. Эта документация опубликована по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-guide.

История Polybase

Примерно в 2011 году доктор Дэвид ДеВитт (Dr. David DeWitt) и его команда создали новый проект под названием Polybase (веб-сайт этого проекта размещен по ссылке http://gsl.azurewebsites.net/Projects/Polybase.aspx). В эту команду входили Римма Нехме (Rimma Nehme), ныне известная бла-

годаря проекту Azure CosmosDB, и Алан Хэлверсон (Alan Halverson) из лаборатории Jim Gray Systems Lab, входящей в состав подразделения Microsoft Research. Целью данного проекта было создание нового способа доступа к данным в системах Hadoop, в котором отсутствовала бы необходимость программирования заданий MapReduce (подробнее о MapReduce, который очень популярен среди тех, кто использует Hadoop, см. https://en.wikipedia.org/wiki/MapReduce).

Я подробно расспросил Дэвида, который сейчас работает в MIT, об истории Polybase. Я спросил его, что натолкнуло его на мысль создать новый способ использования MapReduce. Он дал мне ссылку на заметку в блоге, которую написали он и Майкл Стоунбрейкер (Michael Stonebraker). Вы можете прочитать эту заметку, перейдя по ссылке https://homes.cs.washington.edu/~billhowe/mapreduce_a_major_step_backwards.html. В ней названо несколько причин, в силу которых MapReduce представляет собой ужасный подход к решению проблемы доступа к данным.

Впоследствии Дэвид и его команда создали проект Polybase для использования технологии Parallel Data Warehouse (PDW) в платформах Microsoft для доступа к большим данным в системах Hadoop. PDW, теперь называемый Analytics Platform System (APS), является предшественником облачного хранилища данных Azure SQL Data Warehouse. Как говорит Дэвид: «…мы могли бы подключить PDW к HDFS и использовать параллельный запрос PDW, чтобы дать нашим клиентам возможность применять стандартный SQL вместо MapReduce. Это даст клиентам возможность получать доступ к своим реляционным данным и внешним таблицам, хранящимся в HDFS, используя только один запрос».

Команда написала статью, где рассказала об этой технологии. Статья доступна по ссылке http://gsl.azurewebsites.net/portals/0/users/projects/polybase/polybasesigmod2013.pdf. Этот документ был опубликован в материалах конференции ACM SIGMOD 2013 года. Polybase впервые появилась как функция в PDW в середине 2012 года, и она до сих пор существует и используется.

На рис. 9.1 представлена визуальная схема оригинальной концепции Polybase.

Перенесемся к моменту подготовки к выпуску SQL Server 2016. Если Polybase может использоваться через SQL в PDW, почему бы не использовать эту технологию с SQL Server? В SQL Server 2016 мы добавили поддержку Polybase для доступа к данным в системах Hadoop с помощью T-SQL. Я часто называю эту функцию «классическая Polybase» (это мой собственный термин, а не официальный термин Microsoft). Вы можете использовать T-SQL для создания так называемой внешней таблицы для отображения файлов HDFS, а затем выполнить запрос к этой внешней таблице, как и к любой другой таблице. «Запрос» будет преобразован в Java-программу MapReduce для запуска в целевой системе Hadoop.

Я присоединился к команде разработчиков сразу после выхода версии SQL Server 2016 и никогда не видел, чтобы Polybase пользовалась популяр-

ностью у наших пользователей. Я точно не знаю, почему так произошло, но, возможно, одна из причин может заключаться в том, что пользователи должны были установить Java – обычно это пакет Java Runtime Environment (JRE), выпускаемый Oracle, – на тот же компьютер, что и SQL Server. Возможно также, что в 2016 году пользователи SQL Server просто не были готовы к интеграции с Hadoop, а пользователи Hadoop хотели дистанцироваться от реляционных баз данных.

В 2016 году Microsoft приобрела компанию под названием Metanautix, о которой я упоминал во вступительной части данной книги. Благодаря этому приобретению появилась технология ODBC для доступа к различным источникам данных, таким как SQL Server, Oracle, Teradata и MongoDB. Трэвис Райт (Travis Wright) и Слава Окс (Slava Oks) увидели преимущество, которое можно получить, применяя эти технологии, и поэтому они расширили возможности Polybase в SQL Server 2019, чтобы пользователи могли применять внешние таблицы для доступа не только к Hadoop, но также и к SQL Server, Oracle, Teradata и MongoDB. И что еще сильнее поразит вас, мы добавили поддержку доступа к любому источнику данных с помощью драйвера ODBC на ваш выбор. Я называю эту новую возможность **Polybase++** (опять же, это мой термин, а не официальный термин, используемый Microsoft).

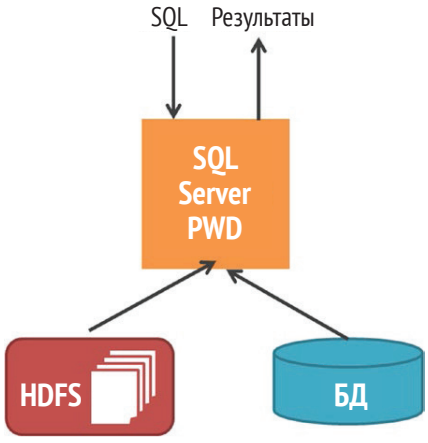


Рис. 9.1. Оригинальная концепция Polybase, разработанная Jim Grey Systems Lab

Что такое виртуализация данных?

Я не затратил много времени на погружение в тему Polybase для SQL Server 2016, поскольку был знаком с ее основными концепциями. Когда же я начал участвовать в обсуждениях и подготовке обучающих материалов по SQL Server 2019, то, услышав термин «виртуализация данных» (я думаю, что впервые услышал этот термин от Трэвиса Райта), был вынужден глубже погрузиться в Polybase.

Существует много различных определений виртуализации данных, и вы можете прочитать «официальное» определение в Википедии по ссылке https://en.wikipedia.org/wiki/Data_virtualization. Мне нравится следующая формулировка, в которой говорится: «В отличие от традиционного процесса извлечения, преобразования, загрузки (Extract, Transform, Load – ETL), данные остаются на месте, и доступ к исходной системе для данных предоставляется в режиме реального времени. Этот подход снижает риск возникновения ошибок данных, исключает лишнюю нагрузку на систему, появляющуюся вследствие перемещений данных, которые могут никогда не использоваться, и не пытается навязать единую модель данных».

Ключ к виртуализации данных – это концепция отсутствия перемещения данных. Для ясности, «данные» в этом случае не должны перемещаться из источника в их исходном формате. Вместо этого данные извлекаются с помощью запроса или же выполняются запросы к источнику данных.

В рамках общей стратегии SQL Server 2019 по внедрению решения для виртуализации данных мы можем сказать, что SQL Server является превосходным *центром* виртуализации данных. Другими словами, SQL Server 2019 может стать *центром данных* (data hub) для вашей организации.

На рис. 9.2 приведен слайд презентации, часто используемый при обсуждении общей концепции Polybase, виртуализации данных (Data Virtualization) и SQL Server 2019.

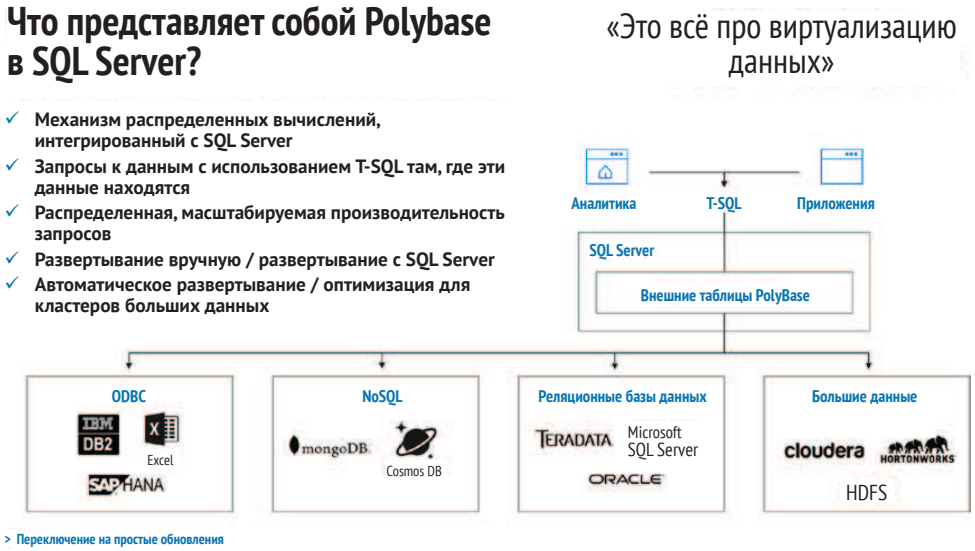


Рис. 9.2. Polybase и виртуализация данных в SQL Server 2019

Посмотрите на значки на этом рисунке. SQL Server 2019 позволяет выполнять запросы T-SQL к внешним таблицам на основе разнообразных источников данных, начиная от HDFS и Oracle и заканчивая CosmosDB и SAP HANA. И наиболее радикальным изменением является то, что вы можете

создавать и выполнять запросы T-SQL ко всем этим ресурсам и присоединять их к локальным таблицам SQL Server или к любой другой внешней таблице, представляющей любой из этих других источников данных.

На данном слайде я попытаюсь упростить определение того, что такое Polybase:

- **механизм распределенных вычислений.**

  Polybase содержит программное обеспечение, изначально присущее первоначальному дизайну PDW, которое интегрировано с SQL Server и содержит собственный механизм распределенных вычислений. Об этом компоненте будет рассказываться в следующем разделе «Как работает Polybase»;

- **запросы к данным с использованием T-SQL там, где эти данные находятся.**

  Это преимущество виртуализации данных – возможность выполнять запросы T-SQL к локальному серверу SQL и запросы к данным из других источников, не перемещая их. Еще один момент, касающийся Polybase для SQL Server 2019: программное обеспечение, необходимое для запросов к SQL Server, Oracle, Teradata и MongoDB, уже включено в состав дистрибутива SQL Server. Никакого дополнительного клиентского программного обеспечения не требуется!

- **распределенная, масштабируемая производительность запросов.**

  Polybase предоставляет нечто большее, чем просто метод «подключения» к другим источникам данных; это возможно благодаря связанным серверам. Поскольку Polybase является интегрированным механизмом распределенных вычислений, он может обеспечить масштабируемую производительность запросов. А концепция, называемая масштабируемыми группами, позволяет распределять запросы к источникам данных, таким как Hadoop, SQL Server и Oracle;

- **развертывание вручную / развертывание с SQL Server.**

  Все, что было сказано о Polybase, уже прозвучало весьма впечатляюще; так неужели здесь нет никаких подводных камней? Что ж, настройка Polybase требует некоторых усилий, особенно если вы хотите настроить масштабируемые группы в Windows. После развертывания Polybase настройка не требуется. При настройке подключений к источникам данных требуется выполнить определенные действия, поскольку Polybase хороша только в том случае, если вы можете получить доступ и подключиться к источникам данных, которые будут использоваться с помощью внешних таблиц;

- **автоматическое развертывание / оптимизация для кластеров больших данных.**

  Как вы узнаете из главы 10, кластеры больших данных SQL Server обеспечат виртуализацию данных с развернутым компонентом

Polybase и развернутым кластером Hadoop с оптимизированным доступом к данным в HDFS.

Как работает Polybase

Я считаю, что понимание того, как работает SQL Server, позволяет вам использовать его наиболее эффективно. Если вы видели мои выступления на различных конференциях, таких как PASS Summit, то знаете, что я обладаю необходимым набором компетенций, чтобы представлять внутренние аспекты функциональности SQL Server. Поэтому, когда меня попросили подготовить несколько сессий на конференции SQL Bits 2019 в Манчестере (Великобритания), я выбрал Polybase в качестве темы выступлений. Я хотел рассказать о том, как Polybase работает внутри компании, особенно о том, какую архитектуру мы создали для доступа к источникам данных, таким как Oracle. У меня был большой опыт работы с SQL Server, поэтому я был очень хорошо знаком с подробностями относительно работы со связанными серверами. Чем же отличается Polybase? Я приведу подробное сравнение этих технологий далее в главе. Хорошим дополнением к этой главе будет мой доклад на SQL Bits на эту тему, который вы можете найти по ссылке https://sqlbits.com/Sessions/Event18/Inside_SQL_Server_2019_Polybase.

Схема работы Polybase

Прежде чем я перейду к описанию всех программных компонентов, которые необходимо развернуть совместно с SQL Server для обеспечения работы с Polybase, вам необходимо познакомиться со схемой работы Polybase.

На рис. 9.3 представлен слайд, который я часто использую, чтобы показать схему работы Polybase с SQL Server.

Далее я объясню каждую из частей этой схемы.

- **Настройка и конфигурирование Polybase**. Более подробно о настройке и конфигурировании Polybase я расскажу в разделе «Подготовительные шаги для использования примеров, иллюстрирующих применение Polybase и SQL Server». Вы также можете прочитать о настройке Polybase для Windows по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-installation и для Linux по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-linux-setup.
- **Настройка аутентификации.** Вам необходимо настроить способ аутентификации для подключения к внешнему источнику данных. Polybase поддерживает только концепцию базовой аутентификации, которая означает, что вы должны хранить определенный тип IDENTITY (или пользователя) и SECRET (пароль или ключ) в SQL Server для доступа к внешнему источнику данных. Это объект, называемый *учетными данными в базе данных* (database scoped credential),

и он зашифрован с помощью объекта SQL Server **MASTER KEY**. Вы можете прочитать об учетных данных в базе данных, перейдя по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/create-database-scoped-credential-transact-sql.

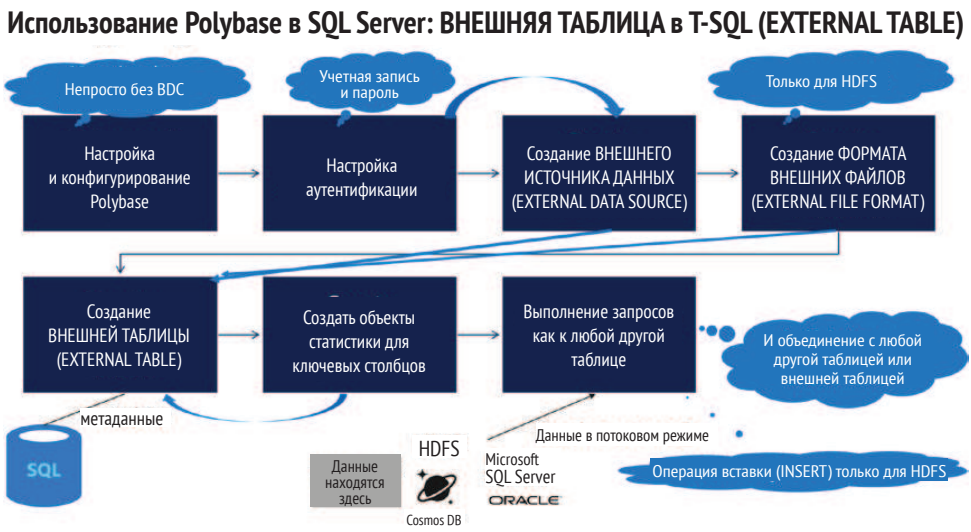


Рис. 9.3. Схема работы Polybase с SQL Server

- **Внешний источник данных (EXTERNAL DATA SOURCE).** Рассматривайте внешний источник данных как объект T-SQL, аналогичный источнику данных ODBC. Создайте его один раз для источника данных, который вы будете использовать для одного или нескольких определений внешних таблиц EXTERNAL TABLE. В примерах из этой главы вы увидите, что вам потребуется информация о подключении к внешнему источнику данных. Вы можете прочитать о внешних источниках данных по адресу https://docs.microsoft.com/en-us/sql/t-sql/statements/create-external-data-source-transact-sql. Значение CREDENTIAL будет именем созданной вами учетной записи в базе данных.
- **Формат внешних файлов (EXTERNAL FILE FORMAT).** Реляционные данные и даже данные noSQL имеют определенную структуру; обычно они представлены в виде столбцов или полей. Данные, хранящиеся в системах Hadoop, как правило, полуструктурированы. Чтобы SQL Server имел доступ к данным в файлах в HDFS, необходимо указать формат, который определяется параметром формат внешних файлов (EXTERNAL FILE FORMAT). Эта спецификация не нужна для таких источников данных, как Oracle. Вы можете прочитать о формате внешних файлов, перейдя по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/create-external-file-format-transact-sql.

- **Внешняя таблица (EXTERNAL TABLE).** Внешние таблицы можно рассматривать как виртуальные таблицы SQL Server (более известные как *представление* (view)). Это означает, что внешние таблицы работают как таблицы SQL Server – метаданные, описывающие таблицы, хранятся в представлениях каталога (catalog views), но сами данные или хранилище внешних таблиц находятся непосредственно в источнике данных. Вы можете прочитать о внешних таблицах, перейдя по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/create-external-table-transact-sql. Параметр DATA_SOURCE при создании внешней таблицы будет именем внешнего источника данных. Для внешних таблиц HDFS вы укажете формат внешнего файла, который создали, используя параметр FILE_FORMAT.
- **Объекты статистики (Statistics)**. Чтобы помочь обработчику запросов и механизму вычислений Polybase создать оптимальный план запроса для внешних таблиц, вы можете определить данные статистики, хранящиеся в SQL Server, на основе столбцов из внешних таблиц. Вы можете прочитать о создании объектов статистики по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/create-statistics-transact-sql.
- **Запрос.** После определения всех этих объектов можно выполнять запросы T-SQL к внешним таблицам и даже выполнять операции их соединения (join) с локальными таблицами SQL Server либо другими внешними таблицами. Ключевая концепция заключается в том, что данные размещены во внешнем источнике данных и не загружаются в SQL Server; а в базе данных SQL Server хранятся лишь метаданные и объекты статистики. Запросы к внешним таблицам доступны только для чтения везде, кроме Hadoop. SQL Server поддерживает процессы получения, внесения и обработки данных для последующего их использования или хранения в своей базе данных или вставку во внешние таблицы на основе Hadoop. Вы можете прочитать о запросах Polybase, перейдя по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-queries.

Архитектура SQL Server 2019 Polybase

Теперь, когда мы рассмотрели объекты и схему работы Polybase для запросов к внешним таблицам, я перейду к описанию программных компонентов, которые обеспечивают эту возможность, прежде чем вы приступите к практическому использованию данной возможности.

> **Примечание**. Я глубоко признателен Стюарту Падли (Stuart Padley), Дэвиду Кризу (David Kryze), Джеймсу Роуланду-Джонсу (James Rowland-Jones) и Калифорнийскому университету за информацию, относящуюся к внутренним компонентам Polybase, которая использовалась в этом разделе главы.

Как работают внешние таблицы

На рис. 9.4 показан первый слайд со схемой, которую я привожу, когда рассказываю о том, как работает Polybase.

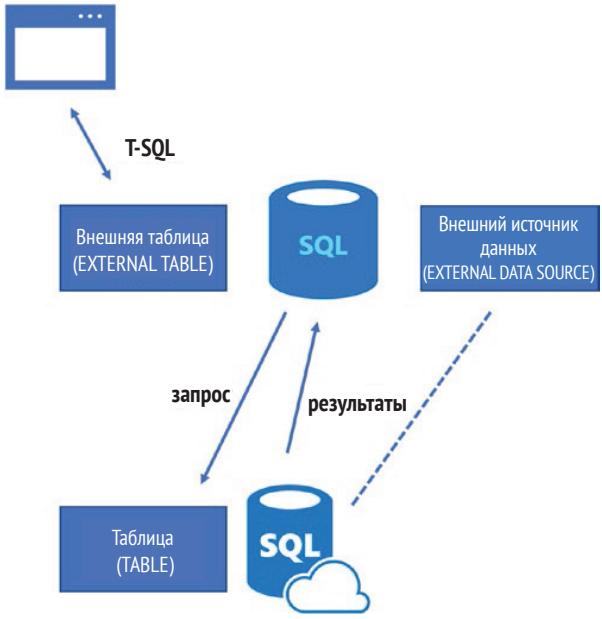


Рис. 9.4. Как работают внешние таблицы

При использовании и изучении Polybase важно понимать, что SQL Server хранит только метаданные для внешних источников данных (EXTERNAL DATA SOURCE) и внешних таблиц (EXTERNAL TABLE), а не сами данные. Пользователи выполняют запросы T-SQL, ссылаясь на внешние таблицы так же, как на таблицы SQL Server. Внешние таблицы связаны с внешними источниками данных, указывающими на место, где в действительности находятся данные. SQL Server как концентратор данных (data hub) принимает запрос к внешней таблице и отправляет новый запрос во внешний источник данных, используя драйвер, соответствующий этому источнику. Результаты отправляются обратно на SQL Server и в конечном итоге отображаются пользователю, выполнившему запрос. Другим аспектом запросов к внешним таблицам является концепция «передачи вниз» (pushdown).

Внешняя фильтрация, или «передача вниз» (pushdown), – это концепция передачи фильтрации данных во внешний источник данных. На рис. 9.4, если внешним источником данных была база данных Azure SQL Database, а в запросе использовалось условие WHERE, определяющее критерии запроса, Polybase попытается отправить запрос в базу данных Azure SQL Database, включая оператор WHERE (это может быть неявное выражение WHERE для всех источников данных), так что определение минимального числа результирующих строк выполняется на внешнем источнике данных.

Противоположный (и менее эффективный) подход заключается в том, чтобы вернуть все строки из внешней таблицы в SQL Server и позволить механизму SQL Server выполнять фильтрацию строк, определяя, какие строки необходимы для получения результата запроса. Вы можете прочитать больше о внешней фильтрации для Polybase по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-pushdown-computation.

Автономный экземпляр Polybase

Давайте углубимся в архитектуру Polybase, представленную на рис. 9.5.

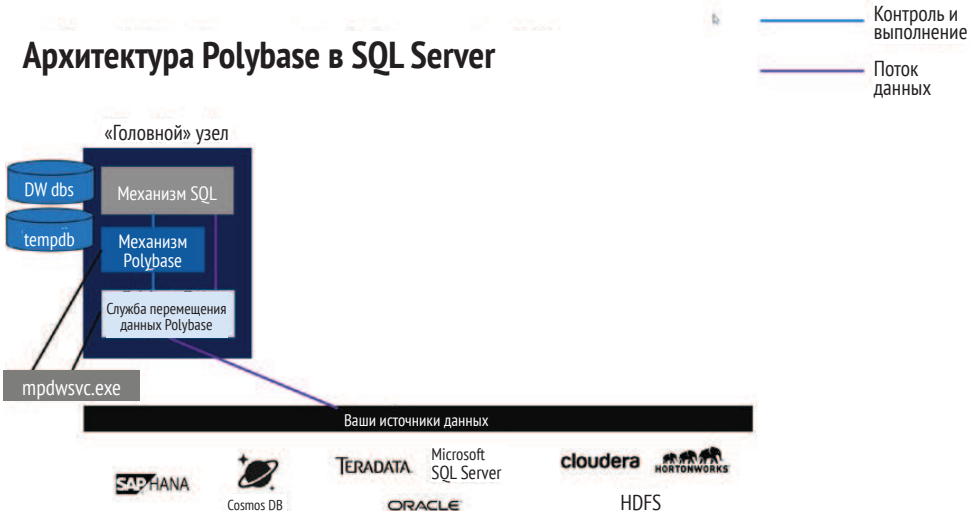


Рис. 9.5. Архитектура головного узла Polybase

Я рассмотрю эту схему подробнее в контексте Windows, а затем расскажу, как мы реализуем ее в Linux.

При развертывании Polybase в Windows вам будет предложено два варианта:

- **Standalone PolyBase-enabled instance (Автономный экземпляр с поддержкой PolyBase).**

  Выберите этот параметр, если хотите, чтобы для Polybase использовался только один экземпляр SQL Server. Все программное обеспечение, необходимое для Polybase, будет установлено на этом экземпляре и будет считаться *головным узлом*;
- **Use the SQL Server instance as part of a PolyBase scale-out group (Используйте экземпляр SQL Server как часть масштабируемой группы PolyBase).**

  Используйте этот вариант, чтобы настроить так называемую масштабируемую группу. Далее я расскажу о масштабируемых группах более подробно.

На рис. 9.5 представлен рабочий сценарий для автономного экземпляра с поддержкой PolyBase. Ниже приведено описание всех элементов этой схемы.

- **Механизм Polybase (Polybase Engine)**. Это служба Windows, которая включает исполняемый модуль mpdwsvc.exe. Обратите внимание на схему, приведенную на рисунке: Polybase Engine отвечает за **контроль и выполнение**. Другими словами, Polybase Engine является координатором для выполнения запросов к внешним таблицам. Механизм SQL Server будет координировать выполняемые действия с механизмом Polybase. Polybase Engine фактически включает исполняемый программный код Polybase в PDW для поддержки внешних таблиц. Исполняемый файл mpdwsvc.exe выполняется службой Windows с использованием параметра **-dweng**. Связь между Polybase Engine и SQL Server осуществляется через локальный именованный канал.
- **Служба перемещения данных Polybase (Polybase Data Movement Service, DMS)**. Как и следует из названия, за данные отвечает Polybase DMS. Это означает, что Polybase DMS будет выполнять запросы к внешним источникам данных и передавать результаты обратно в механизм SQL Server.

  Что интересно, Polybase DMS также реализована с помощью исполняемого файла mpdwsvc.exe, но с другим параметром **-dms**. На сервере головного узла вы должны увидеть два процесса с именем mpdwsvc.exe. Это также означает, что Polybase DMS – программа, которая загружает все драйверы ODBC или запускает код Java для MapReduce для систем Hadoop. Polybase DMS также взаимодействует с механизмом SQL Server и механизмом Polybase через именованные каналы. Служба Polybase DMS будет передавать данные по именованному каналу с помощью механизма SQL Server для отправки результатов из запросов к внешним данным.
- **DW dbs** – для Polybase требуются собственные метаданные. При установке Polybase вы найдете следующие базы данных, установленные на SQL Server: DWConfiguration, DWDiagnostics и DWQueue. Вы должны рассматривать эти базы данных как системные базы данных для Polybase, поэтому они должны быть доступны, чтобы Polybase могла функционировать. Я не буду вдаваться в подробности того, что находится в каждой базе данных, и в официальной документации не приведено никаких сведений об этом. Однако я нашел интересного пользователя, который писал в своем блоге о том, как «нырнуть» внутрь этих баз данных, по ссылке https://36chambers.wordpress.com/2019/04/03/polybase-revealed-the-dwdatabases/.
- **Tempdb** – Polybase может использовать tempdb для промежуточной обработки запросов при выполнении запросов к внешним таблицам.

Кроме того, чтобы обеспечить правильную обработку потоков данных, Polybase создаст таблицы tempdb в качестве «хранилища данных» для потоковой передачи данных (хотя она может никогда не использовать ее). В моей практике использования Polybase я не наблюдал сколько-нибудь значительного применения tempdb. Я просто хочу, чтобы вы знали об использовании базы данных tempdb – поэтому вы не удивитесь, увидев в Polybase активность, связанную с временными таблицами.

В состав поставляемого пакета Polybase также входят ряд представлений каталога (catalog views) и представлений динамического управления (Dynamic Management Views, DMV). Некоторые из них я буду использовать в примерах в данной главе. Список этих представлений каталога и представлений динамического управления можно найти по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-troubleshooting.

Масштабируемая группа Polybase

Если вы решите настроить масштабируемую группу Polybase, можно использовать несколько экземпляров SQL Server для масштабирования обработки запросов; см. рис. 9.6, где представлена конфигурация масштабируемой группы.

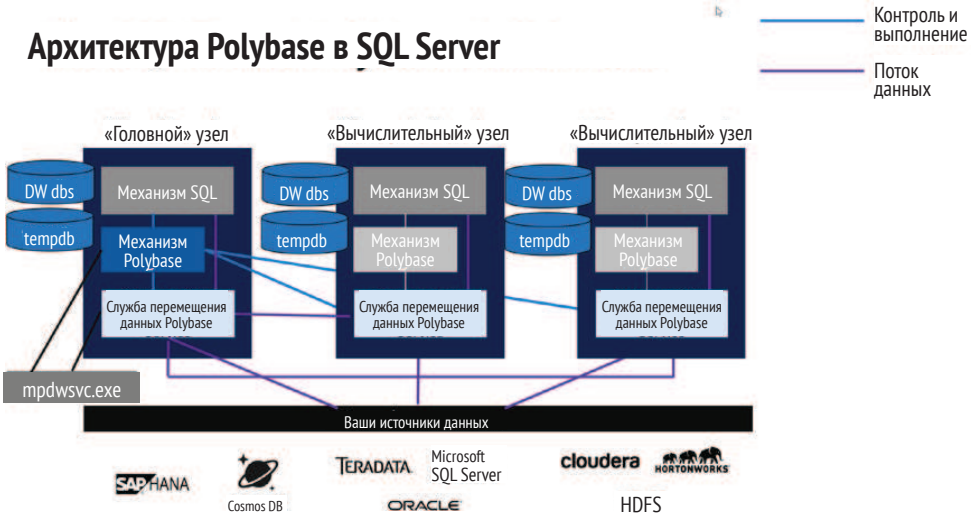


Рис. 9.6. Масштабируемая группа Polybase

Масштабируемые группы Polybase позволяют включить использование Polybase для других экземпляров SQL Server для обработки запросов с горизонтальным масштабированием. Другие экземпляры SQL Server, которые при этом работают с Polybase, называются *вычислительными узлами*. Обратите внимание, что на вычислительных узлах Polybase Engine неакти-

вен. В Windows служба Polybase Engine установлена, но она отключена и не используется. Polybase Engine на головном узле выполняет всю координацию на всех узлах, а службы Polybase DMS отвечают за весь обмен данными на каждом узле. Причина, по которой мы устанавливаем Polybase Engine на всех узлах, заключается в том, что вычислительный узел при необходимости может стать головным узлом (например, если на текущем головном узле возникла проблема).

Использование масштабируемых групп наиболее эффективно, когда SQL Server решает, что применение нескольких экземпляров может ускорить запрос к внешней таблице. Это может быть очень мощным рабочим инструментом для систем Hadoop, и масштабируемые группы создавались с учетом распределенных систем Hadoop. Для других источников данных, таких как SQL Server или Oracle, Polybase может обнаруживать разделы в этих источниках и использовать масштабируемую группу для запросов к каждому разделу в целевом объекте.

Мы называем эту возможность *чтением с возможностью масштабирования*, о котором вы можете прочитать по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-scale-out-groups?view=sql-server-ver15#scale-out-reads.

Обработка запросов и Polybase

Одним из замечательных нововведений Polybase является то, что запросы к внешним таблицам встроены в процессор запросов SQL Server. Это означает, что обработчик запросов SQL Server понимает, когда он работает с внешней таблицей, и создает правильные сценарии плана выполнения для отправки в Polybase Engine, чтобы можно было поддерживать такие операции, как внешняя фильтрация данных (pushdown).

Далее в этой главе я покажу пример того, как выглядит оператор удаленного запроса для запроса к внешней таблице в механизме SQL Server.

Как это работает в Linux?

SQL Server 2019 в Linux поддерживает только автономный экземпляр Polybase (мы будем поддерживать концепцию масштабируемой группы с кластерами больших данных SQL Server, об этом будет рассказываться в главе 10). Кроме того, Polybase для SQL Server 2019 в Linux не поддерживает универсальный коннектор ODBC для источников данных.

Поэтому архитектура Polybase подразумевает внедрение механизма Polybase Engine и службы перемещения данных Polybase (Polybase Data Movement Service) в процесс sqlservr в Linux с использованием SQLPAL (более подробно об SQLPAL рассказывается в главе 6).

В то время когда я писал эту главу, мы завершали работу над выпуском версии SQL Server 2019, но все еще не поддерживали внешние таблицы Hadoop в SQL Server для Linux (за исключением кластеров больших дан-

ных SQL Server). Я ожидаю, что эта функция войдет в официальный выпуск SQL Server 2019; однако в ее основе лежат те же концепции, что и в версии для Windows. Возможно, у нас будет отдельный пакет для Linux для этой возможности; это должно быть отражено в документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-linux-setup.

Чем это отличается от Azure?

В настоящее время Polybase реализована как функция в SQL Server, хранилище данных Azure SQL Data Warehouse и Analytics Platform System (APS, ранее известной как Parallel Data Warehouse). Однако возможности Polybase, предоставляемые для каждого из этих решений, различны.

> **Примечание.** Объект EXTERNAL TABLE существует в базе данных Azure SQL Database, но сам по себе он не является функцией Polybase (на момент выхода версии SQL Server 2019). Внешняя таблица в базе данных Azure SQL Database используется для поддержки эластичных запросов. Вы можете узнать больше об эластичных запросах из документации, опубликованной по ссылке https://docs.microsoft.com/en-us/azure/sql-database/sql-database-elastic-query-getting-started.

Polybase для хранилища данных Azure SQL Data Warehouse использует внешние таблицы для доступа к Hadoop или HDFS с такими источниками, как хранилище BLOB-объектов Azure (Azure Blob Storage) или озеро данных Azure (Azure Data Lake). Такие источники, как SQL Server, Oracle и т. д., не поддерживаются для хранилища данных Azure SQL Data Warehouse. Вы можете прочитать больше об использовании Polybase с хранилищем данных Azure SQL Data Warehouse, перейдя по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/create-external-data-source-transact-sql.

Polybase для APS похож на хранилище данных Azure SQL Data Warehouse, но больше предназначен для предоставления доступа к «локальным» системам Hadoop. Вы можете найти информацию о Polybase для APS по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/create-external-table-transact-sql?view=aps-pdw-2016-au7.

Подготовительные шаги для использования примеров, иллюстрирующих применение Polybase и SQL Server

Теперь перейдем к практическим примерам, размещенным в оставшейся части главы. Сначала я дам вам несколько советов по развертыванию и настройке Polybase, а затем несколько советов, относящихся к практическому использованию примеров.

Настройка и подключение Polybase

Чтобы установить Polybase для Windows, вы можете выполнить действия, описанные в документации по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-installation. Процесс установки очень прост, если выбрать установку автономного экземпляра Polybase. Один из вариантов, который вы должны выбрать, – это использовать ли Java-коннектор для HDFS (**Java connector for HDFS**). Если вы устанавливаете Polybase для поддержки внешних таблиц для HDFS, то вам будет предложен выбор применять по умолчанию Open Java – пакет, который мы поставляем с SQL Server 2019, или установить свою собственную версию. Пакет Open Java, который мы предоставляем, основан на Zulu Java, о которой вы можете прочитать по ссылке https://cloudblogs.microsoft.com/sqlserver/2019/07/24/free-supported-java-in-sql-server-2019-is-now-available/.

После установки Polybase в Windows мы установим серию драйверов ODBC (которые поместим в каталог **binn\Polybase\ODBC Drivers**). Эти драйверы поддерживают встроенные коннекторы для SQL Server, Oracle, Teradata и MongoDB.

После установки компонента Polybase необходимо включить его с помощью sp_configure, как описано в документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-installation?view=sql-server-ver15#enable.

В SQL Server для Linux мы предоставляем отдельный пакет для установки Polybase; о том, как его настроить и использовать, вы можете прочитать по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-linux-setup.

Для групп с горизонтальным масштабированием процесс настройки становится очень интересным. Поскольку в настоящее время масштабируемые группы поддерживаются только в Windows, это единственная конфигурация SQL Server 2019, которую вам нужно поддерживать.

Мой опыт развертывания масштабируемых групп был довольно сложным. Вы можете прочитать инструкцию по развертыванию, подробно описывающую все необходимые шаги, в документации по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/configure-scale-out-groups-windows?view=sql-server-ver15.

Для начала позвольте мне дать вам несколько советов, прежде чем вы приступите к процедуре развертывания:

- вам потребуется домен Windows; поэтому если у вас нет контроллера домена, вам необходимо его настроить;
- все службы Windows для масштабируемой группы Polybase должны использовать одну и ту же учетную запись службы домена. Вы должны настроить эту конфигурацию при установке или с помощью диспетчера конфигурации SQL Server;

- сначала вам придется установить параметры конфигурации на головном узле и вычислительном узле. Когда вы впервые устанавливаете Polybase на все узлы с помощью программы установки и выбираете вариант **Use the SQL Server instance as part of a PolyBase scale-out group (Использовать экземпляр SQL Server как часть масштабируемой группы PolyBase)**, все узлы являются кандидатами для выбора в качестве головного узла. Для правильной работы Polybase вам необходимо выбрать один из ваших компьютеров в качестве головного узла. Затем для других узлов вам нужно запустить хранимую процедуру, чтобы настроить их как вычислительные узлы, перечислив имя сервера и порта головного узла (обратите внимание на порт, который вы выбрали во время установки, потому что он вам здесь понадобится). Процесс присоединения в качестве вычислительного узла описан в документации, доступной по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/configure-scale-out-groups-windows?view=sql-server-ver15#add-other-sql-server-instances-as-compute-nodes;
- вам необходимо включить Polybase с помощью sp_configure на всех узлах и перезапустить SQL Server;
- вам также необходимо перезапустить все службы Polybase на всех узлах. Фактически, если хранимая процедура не делает этого автоматически, вам нужно остановить Polybase Engine на вычислительных узлах. Если все работает хорошо, служба Polybase Engine будет отключена на вычислительных узлах, но вы должны дважды проверить это;
- выполните запрос DMV dm_exec_compute_nodes, чтобы убедиться, что все узлы имеют правильный статус HEAD или COMPUTE. Подробная информация об этом DMV доступна в документации по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-compute-nodes-transact-sql.

Выполнение практических упражнений

Рассмотрим возможный сценарий для компании WideWorldImporters, представленный на рис. 9.7.

В этом примере компания WideWorldImporters (WWI) использует SQL Server 2019, но хочет получить доступ к данным, размещенным в следующих источниках данных:

- **SQL Server 2008R2** – компания имеет систему SQL Server одной из предыдущих версий, в которой хранится архив поставщиков. Они не хотят трогать эту систему, но хотят получить доступ к информации о поставщиках;
- **база данных Azure SQL Database**. Команда компании ищет возможности перехода в облако и создания новой базы данных StockItems с использованием Azure. Структурные подразделения WWI хотят ви-

деть и объединять эти данные StockItem с существующими в базе данных SQL Server 2019, не прерывая работу с базами данных;

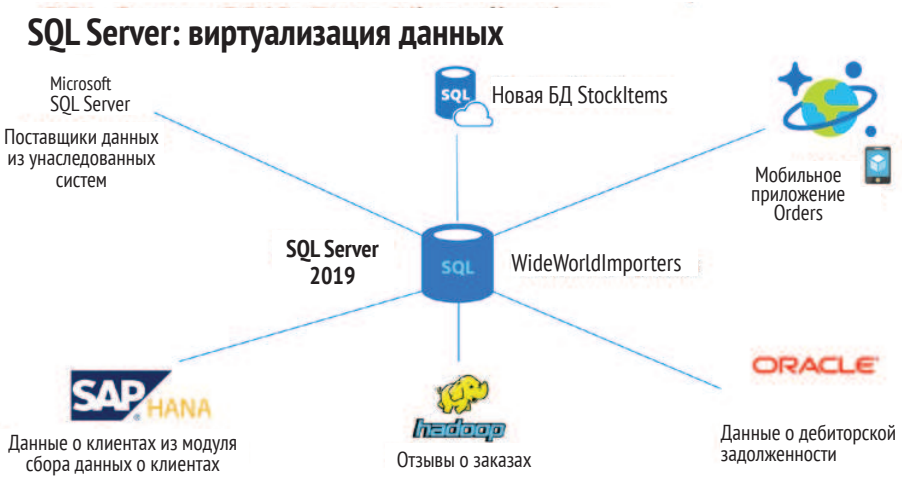


Рис. 9.7. Центр данных SQL Server

- **Azure CosmosDB** – еще одна команда пилотирует мобильное приложение для заказов и экспериментирует с Azure CosmosDB. Команда WWI хочет иметь возможность просматривать эти заказы и объединять данные, связанные с заказами, в локальной базе данных;
- **Oracle** – бухгалтерское программное обеспечение для WWI использует Oracle. Хотя WWI собирается выполнять миграцию этой базы данных на SQL Server, для проекта миграции требуется время. Между тем в WWI знают, что некоторые данные в базе данных SQL Server ссылаются на данные в базе данных дебиторской задолженности. Если они могут получить доступ к Oracle, они хотели бы объединить локальные данные SQL Server с данными дебиторской задолженности в Oracle, пока миграция не будет завершена;
- **Hadoop** – команда WWI создает систему рейтинга на веб-сайте компании, чтобы клиенты могли просматривать данные о заказах. Для ускорения проекта группа разработчиков хранит сводные данные о заказах в полуструктурированном формате с использованием хранилища BLOB-объектов Azure (Azure Blob Storage). Структурные подразделения WWI хотят анализировать эти данные и объединять их с локальными данными SQL Server;
- **SAP HANA** – компания WWI недавно приобрела другую компанию, Vandelay Industries (меня вдохновила вымышленная компания из Seinfeld. См. https://seinfeld.fandom.com/wiki/Vandelay_Industries). Эта компания хранит данные о своих клиентах в SAP HANA. В то время как команда WWI разрабатывает стратегию миграции, они хотят анализировать данные об этих клиентах, не перемещая при этом данных.

Все эти сценарии возможны с использованием Polybase в SQL Server 2019 и внешних таблиц. Пример каждого из них есть в каталоге **ch9_data_virtualization\sqldatahub**.

Использование внешних таблиц

Прежде чем мы перейдем к подробному рассмотрению некоторых примеров из каталога **sqldatahub**, я опишу некий базовый шаблон, который вы, несомненно, обнаружите в этих примерах. Этот шаблон соответствует общему процессу, используемому при работе с внешними таблицами, который я описал в разделе «Схема работы Polybase». Все объекты Polybase находятся в области пользовательской базы данных.

1. Создайте главный ключ (MASTER KEY) в базе данных.
2. Создайте учетную запись уровня базы данных (DATABASE SCOPED CREDENTIAL) для аутентификации во внешнем источнике данных.
3. Создайте внешний источник данных (EXTERNAL DATA SOURCE), чтобы указать местоположение источника данных. Параметр CREDENTIAL будет именем учетной записи уровня базы данных.
4. Создайте формат внешнего файла (EXTERNAL FILE FORMAT) для данных HDFS.
5. Создайте внешнюю таблицу (EXTERNAL TABLE) для сопоставления с целевыми таблицами внешнего источника данных. Свойство DATA_SOURCE будет именем внешнего источника данных. Свойство FILE_FORMAT (только для HDFS) будет наименованием формата внешнего файла.
6. Создайте локальный объект статистики (local statistics) для столбцов для внешней таблицы.
7. Выполните запрос к внешней таблице (EXTERNAL TABLE), включив в запрос соединения с локальными таблицами SQL Server или другими внешними таблицами.

В качестве подсказки вот отличная ссылка на документацию, где описаны все операторы T-SQL, связанные с созданием объектов Polybase: https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-t-sql-objects.

Инструменты и внешние таблицы

Прежде чем мы перейдем к практическим примерам сценариев из **sqldatahub**, я должен рассказать об инструментах поддержки внешних источников данных и внешних таблиц.

SQL Server Management Studio (SSMS) поддерживает создание внешних источников данных и внешних таблиц с помощью шаблонов SSMS. На рис. 9.8 показан пример использования SSMS для создания внешнего источника данных.

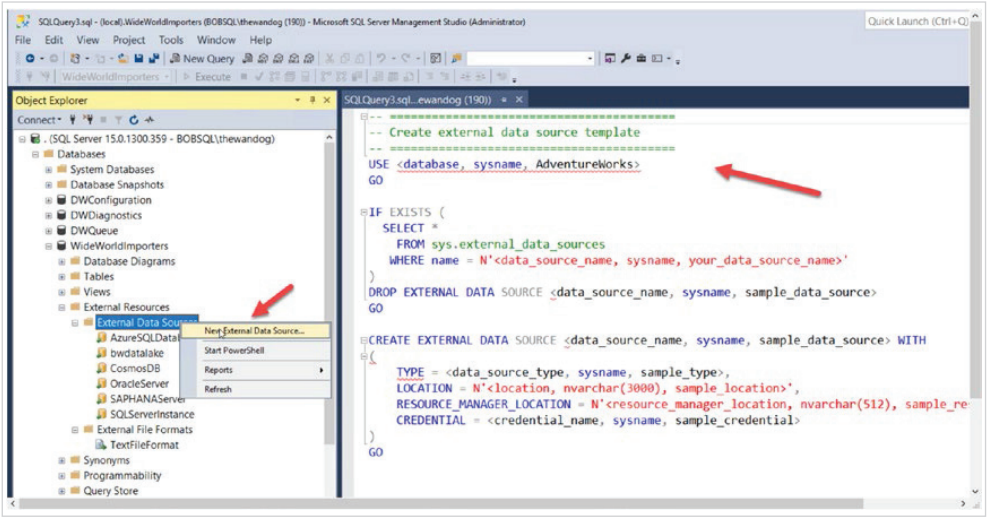


Рис. 9.8. Использование шаблона SSMS для создания внешнего источника данных

То же самое относится и к внешним таблицам.

Создав внешние источники данных и внешние таблицы, вы можете использовать SSMS Object Explorer для просмотра этих ресурсов. На рис. 9.9 показан пример внешних источников данных и форматов файлов, созданных в базе данных WideWorldImporters.

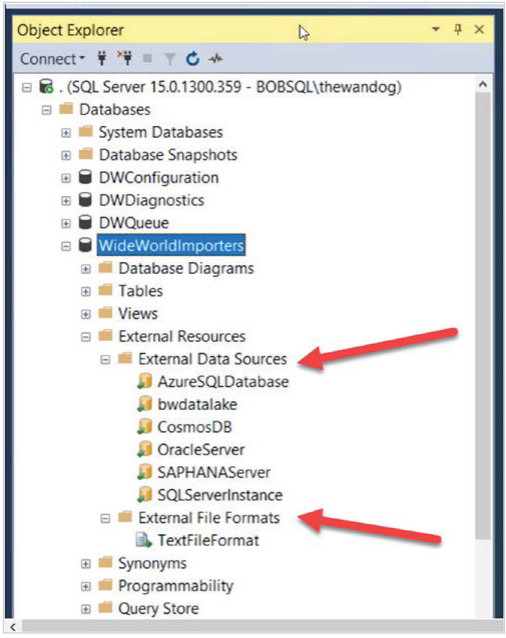


Рис. 9.9. SSMS Object Explorer, используемый для просмотра внешних источников данных и форматов файлов

Azure Data Studio (ADS) также предоставляет мастер для создания внешних таблиц External Table Wizard для источников данных SQL Server и Oracle. Вы можете прочитать об этой возможности по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/data-virtualization.

Я расскажу вам, как настроить внешнюю таблицу для Azure SQL Database. Что касается других примеров в **sqldatahub**, я приведу примеры сценариев и объясню несколько моментов для каждого сценария.

Использование внешних таблиц в Azure SQL Database

Один из встроенных коннекторов SQL Server обеспечивает доступ к источникам данных для SQL Server, базы данных Azure SQL Database и хранилища данных Azure SQL Data Warehouse.

Я разместил примеры сценариев на основе шагов из шаблона, которые описал выше, в каталоге **ch9_data_virtualization\sqldatahub\azuredb**. Я создал сценарии для записной книжки T-SQL (T-SQL Notebook) и сценарии T-SQL для создания внешних таблиц и выполнения запроса к ним.

Чтобы использовать эти сценарии, в первую очередь вам необходимо подготовить и получить доступ к Azure SQL Database. Используйте операторы, приведенные в сценарии **createazuredbtable.sql**, для вашей базы данных в Azure SQL Database.

Как только вы выполните все необходимые настройки, давайте рассмотрим каждый шаг сценария и получим результаты, которые можно ожидать при выполнении сценария T-SQL **azuredbexternaltable.sql**.

1. Выполните **шаг 1** из сценария T-SQL, чтобы изменить контекст базы данных и создать главный ключ для шифрования учетных данных в области базы данных:

```
-- Шаг 1. Создайте главный ключ для шифрования учетных данных базы
-- данных.
USE [WideWorldImporters]
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'S0me!nfo'
GO
```

2. Выполните **шаг 2**, чтобы создать учетные данные в области базы данных, защищенной главным ключом. Вам необходимо указать имя учетной записи и пароль сервера для созданной вами базы данных Azure SQL Database:

```
-- Шаг 2. Создайте учетные данные для базы данных, где будут храниться
-- имя пользователя и пароль для базы данных Azure SQL Server.
-- IDENTITY = имя учетной записи
-- SECRET = пароль
CREATE DATABASE SCOPED CREDENTIAL AzureSQLDatabaseCredentials
```

```
WITH IDENTITY = '<login>', SECRET = '<password>'
GO
```

3. Выполните **шаг 3**, чтобы создать внешний источник данных с использованием учетных данных в базе данных для аутентификации с использованием учетных данных CREDENTIAL:

```
-- Шаг 3. Создайте внешний источник данных
-- sqlserver - это ключевое слово, означающее, что источником данных
-- является SQL Server, база данных Azure SQL Database или хранилище
-- данных Azure SQL Data Warehouse.
-- Имя после символов :// является именем сервера Azure SQL Server
-- Database. Ваш SQL Server должен находиться в той же виртуальной
-- сети, что и Azure SQL Server Database, или должен удовлетворять
-- проверкам брандмауэра.
CREATE EXTERNAL DATA SOURCE AzureSQLDatabase
WITH (
LOCATION = 'sqlserver://<azure sql database server URI>',
PUSHDOWN = ON,
CREDENTIAL = AzureSQLDatabaseCredentials
)
GO
```

Есть несколько моментов, на которые стоит обратить внимание в этом сценарии. Синтаксис LOCATION содержит параметры <тип>:<информация о соединении>, где параметр *тип* имеет следующие возможные значения:

- sqlserver;
- oracle;
- teradata;
- mongodb;
- OBDC.

Параметр *тип* будет указывать SQL Server, какой драйвер ODBC использовать для внешнего источника данных. Для SQL Server сведения о соединении для базы данных Azure SQL Database должны быть URL-адресом сервера (например, <сервер>..database.windows.net).

После успешного создания внешнего источника данных вы можете увидеть список созданных источников, используя представление каталога **external_data_sources** в контексте вашей пользовательской базы данных.

Совет. К сожалению, внешний источник данных можно создать без проверки соединения с источником данных. Если вы неправильно введете данные о соединении, то не узнаете об этом, пока не попытаетесь создать внешнюю таблицу. То же самое

относится и к учетным данным для базы данных: если вы неправильно введете имя учетной записи и/или пароль, то не узнаете об этом, пока не попробуете создать внешнюю таблицу.

4. Выполните **шаг 4** сценария, чтобы создать схему для хранения объектов внешней таблицы. Это не обязательно, но мне нравится использовать схемы для организации объектов (это также весьма разумно с точки зрения безопасности):

```
-- Шаг 4. Создайте схему в WideWorldImporters для внешней таблицы.
CREATE SCHEMA azuresqldb
GO
```

5. Выполните **шаг 5**, чтобы создать внешнюю таблицу с использованием внешнего источника данных, указанного в значении параметра DATA_SOURCE:

```
-- Шаг 5: Создайте внешнюю таблицу (EXTERNAL TABLE)
-- Каждый столбец должен соответствовать столбцу в удаленной таблице
-- (remote table).
-- Обратите внимание, что символьные столбцы используют параметры
-- сортировки, совместимые с целевой таблицей.
-- Ключевое слово WITH содержит имя удаленного [database].[Schema].
-- [Table] и внешнего источника базы данных.
CREATE EXTERNAL TABLE azuresqldb.ModernStockItems
(
    [StockItemID] [int] NOT NULL,
    [StockItemName] [nvarchar](100) COLLATE Latin1_General_100_CI_AS
    NOT NULL,
    [SupplierID] [int] NOT NULL,
    [ColorID] [int] NULL,
    [UnitPackageID] [int] NOT NULL,
    [OuterPackageID] [int] NOT NULL,
    [Brand] [nvarchar](50) COLLATE Latin1_General_100_CI_AS NULL,
    [Size] [nvarchar](20) COLLATE Latin1_General_100_CI_AS NULL,
    [LeadTimeDays] [int] NOT NULL,
    [QuantityPerOuter] [int] NOT NULL,
    [IsChillerStock] [bit] NOT NULL,
    [Barcode] [nvarchar](50) COLLATE Latin1_General_100_CI_AS NULL,
    [TaxRate] [decimal](18, 3) NOT NULL,
    [UnitPrice] [decimal](18, 2) NOT NULL,
    [RecommendedRetailPrice] [decimal](18, 2) NULL,
    [TypicalWeightPerUnit] [decimal](18, 3) NOT NULL,
    [LastEditedBy] [int] NOT NULL
)
```

```
  WITH (
  LOCATION='wwiazure.dbo.ModernStockItems',
  DATA_SOURCE=AzureSQLDatabase
)
GO
```

Это важная часть сценария Polybase, поэтому я укажу на несколько особенностей, на которые нужно обратить внимание:

- число столбцов, имена и типы данных в создаваемой внешней таблице должны точно соответствовать таблице из внешнего источника данных, однако в SQL вы можете использовать любое имя как для имен столбцов, так и для имени самой таблицы;
- отображение типов – достаточно сложная тема. У нас есть документация, которая поможет вам определить типы данных SQL Server для сопоставления с соответствующими типами данных внешних источников данных. Эта документация доступна по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-type-mapping;
- ключевое слово LOCATION для внешней таблицы – это способ сопоставления объекта внешнего источника данных. Каждый источник данных имеет размещение (LOCATION), при этом параметры LOCATION могут различаться для разных источников и использоваться для идентификации объекта источника данных. В случае SQL Server или базы данных Azure SQL Database вы должны ссылаться на таблицу, используя соглашение об именовании, состоящее «из трех частей»: <база данных>.<Схема>.<Имя таблицы>;
- при попытке создать внешнюю таблицу выполняется проверка правильного сопоставления столбцов, сопоставления типов и проверки использования любых запрещенных типов (я упомяну об ограничениях ниже, в разделе «Ограничения»);
- после создания внешних таблиц вы можете увидеть список внешних таблиц, используя представление каталога **sys.external_tables**. Представление каталога **sys.objects** выводит внешние таблицы, имеющие тип USER_TABLE. Представление каталога **sys.tables** содержит столбец **is_external**, который вы можете использовать, чтобы определить, какие таблицы являются внешними.

6. Выполните **шаг 6**, чтобы создать локальный объект статистики по ключевым столбцам из внешней таблицы. Это не обязательно, но рекомендуется, чтобы помочь обработчику запросов принимать разумные решения для поддержки таких операций, как вычисления, связанные с фильтрацией данных (pushdown):

```
-- Шаг 6. Создайте локальный объект статистики по столбцам, которые
-- вы будете использовать для фильтров.
```

```
CREATE STATISTICS ModernStockItemsStats ON azuresqldb.
ModernStockItems ([StockItemID]) WITH FULLSCAN
GO
```

7. Выполните **шаг 7**, чтобы увидеть простой пример сканирования всех строк во внешней таблице. В этом примере должна быть возвращена только одна строка, если запрос был выполнен успешно:

```
-- Шаг 7. Просто попробуйте просканировать удаленную таблицу
SELECT * FROM azuresqldb.ModernStockItems
GO
```

Удаленные операторы (remote operators) встроены в обработчик запросов для поддержки запросов к внешним таблицам с использованием службы Polybase. На рис. 9.10 показан фактический план выполнения запроса для шага 7, включая сведения об удаленном операторе.

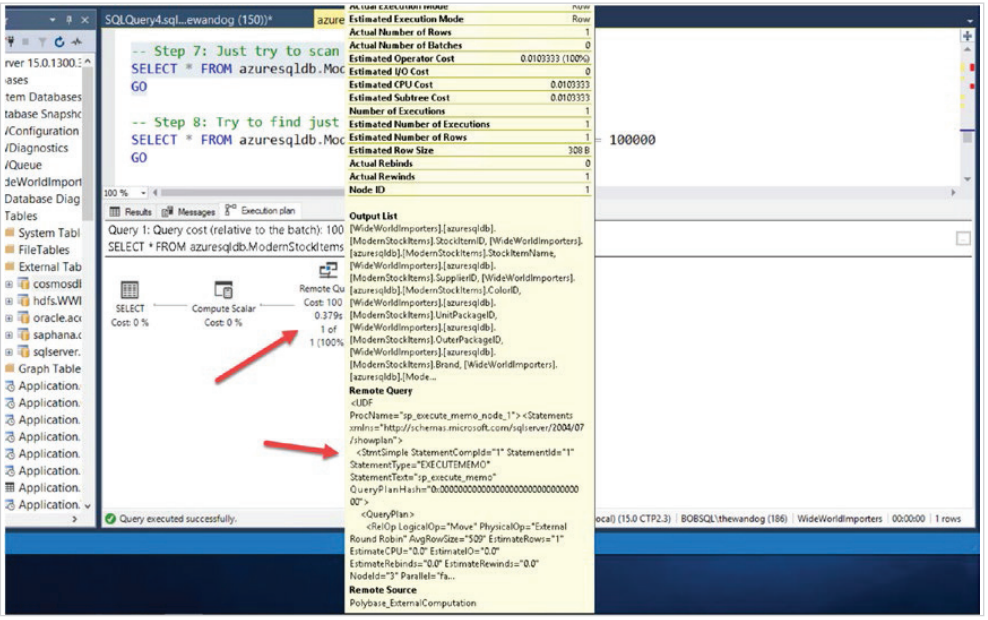


Рис. 9.10. Удаленный оператор для внешних таблиц

Polybase поставляется с серией динамических административных представлений (Dynamic Management Views, DMV), которые можно использовать для анализа выполнения запросов к внешним таблицам.

sys.dm_exec_distributed_requests – подобно **sys.dm_exec_requests**, вы можете найти запросы, относящиеся к Polybase. Что приятно в этом DMV, так это то, что он хранит историю последних запросов, а

не только активных. Значение в столбце **execution_id** является ключом, позволяющим использовать другие DMV для более глубокого анализа выполнения запроса.

sysdm_exec_distributed_request_steps – это DMV будет получать execution_id из sys.dm_exec_distributed_requests и позволит вам посмотреть на конкретные шаги, выполняемые Polybase при обработке запроса к внешней таблице. Для execution_id каждый шаг имеет значение **step_index**.

sys.dm_exec_distributed_sql_requests – это DMV отображает более подробную информацию для каждого step_index в sys.dm_exec_distributed_steps, включая информацию о том, какой вычислительный узел выполняет запрос (это может быть головной и/или вычислительный узел для масштабируемого запроса).

dm_exec_dms_workers – это DMV предоставляет более подробную информацию о выполнении с помощью службы перемещения данных (DMS) Polybase для определенного execution_id и step_index. Позволяет просмотреть данные подключения к внешнему источнику данных через драйверы ODBC, включая информацию о возможных ошибках.

8. Выполните **шаг 8**, чтобы использовать конструкцию WHERE для фильтрации результатов (и, возможно, использовать вычисления, связанные с фильтрацией данных (pushdown) для внешнего источника данных):

```
-- Шаг 8. Попробуйте найти только определенный StockItemID
SELECT * FROM azuresqldb.ModernStockItems WHERE StockItemID = 100000
GO
```

9. Выполните **шаг 9**, чтобы найти все элементы StockItem как в SQL Server 2019, так и в базе данных Azure SQL Database с помощью оператора UNION:

```
-- Шаг 9. Используйте оператор UNION, чтобы найти все элементы
-- StockItem для определенного поставщика как в локальной таблице,
-- так и в таблице Azure.
SELECT msi.StockItemName, msi.Brand, c.ColorName
FROM azuresqldb.ModernStockItems msi
JOIN [Purchasing].[Suppliers] s
ON msi.SupplierID = s.SupplierID
and s.SupplierName = 'Graphic Design Institute'
JOIN [Warehouse].[Colors] c
ON msi.ColorID = c.ColorID
UNION
SELECT si.StockItemName, si.Brand, c.ColorName
```

```
FROM [Warehouse].[StockItems] si
JOIN [Purchasing].[Suppliers] s
ON si.SupplierID = s.SupplierID
and s.SupplierName = 'Graphic Design Institute'
JOIN [Warehouse].[Colors] c
ON si.ColorID = c.ColorID
GO
```

Первая часть UNION включает соединение внешней таблицы и локальной таблицы SQL Server.

Итак, мы рассмотрели пример использования внешней таблицы, применяющей концепцию виртуализации данных (Data Virtualization) при работе с базой данных Azure SQL Database с использованием встроенного коннектора для SQL Server. Далее я приведу краткую информацию для других примеров, размещенных в каталоге sqldatahub.

Использование встроенных коннекторов для внешних таблиц

Есть и другие примеры сценариев, использующих встроенные коннекторы для работы с внешними таблицами. Каждый пример содержит файл readme.md с советами по настройке внешнего источника данных и сценариями для создания объекта источника данных и добавления данных. Все они используют тот же шаблон, что и пример для базы данных Azure SQL Database.

- **ch9_data_virtualization\sqldatahub\cosmosdb** – используйте этот сценарий в качестве демонстрации применения коннектора MongoDB с Azure CosmosDB.
- **ch9_data_virtualization\sqldatahub\oracle** – применяйте этот сценарий в качестве примера использования коннектора Oracle.

Совет. При использовании коннектора Oracle имейте в виду, что значение параметра LOCATION для EXTERNAL TABLE для Oracle учитывает регистр.

- **ch9_data_virtualization\sqldatahub\sql2008r2** – используйте этот сценарий в качестве примера использования коннектора SQL Server для более старой версии SQL Server.

Примечание. В этом примере потребуется использовать обходной путь для SQL Server 2008R2, который не рассматривался во время написания данной главы. Когда мы готовились к выпуску SQL Server 2019, не было ясно, какие из предыдущих версий SQL Server будут поддерживаться, и версия 2008R2 больше не поддерживается.

Использование внешней таблицы с HDFS

Пример использования Polybase с HDFS и хранилищем BLOB-объектов Azure (Azure Blob Storage) можно найти в каталоге **ch9_data_virtualization\sqldatahub\hdfs**. Файл readme.md, содержащийся в этом каталоге, предоставляет дополнительную информацию о том, как выполнить необходимые настройки и как использовать приведенный пример.

Одно большое отличие для внешних источников данных с HDFS – это использование свойства LOCATION с внешним источником данных и применение свойства TYPE.

Вот пример оператора T-SQL для создания внешнего источника данных из сценария, размещенного в каталоге ch9_data_virtualization\sqldatahub\hdfs:

```
CREATE EXTERNAL DATA SOURCE bwdatalake with (
TYPE = HADOOP,
LOCATION ='wasbs://<container>@<azure storage account name>',
CREDENTIAL = AzureStorageCredential
)
GO
```

В отличие от других приведенных примеров, для HADOOP необходимо поле TYPE. Кроме того, свойство LOCATION не имеет атрибута <type>, подобного sqlserver. Это связано с тем, что спецификация TYPE = HADOOP сообщает SQL Server тип коннектора, используемого для HDFS.

Использование внешних таблиц с коннекторами ODBC

Последний пример – демонстрация использования внешней таблицы SAP HANA, применяющей коннектор ODBC. Обратите внимание, что этот пример работает только для версии SQL Server 2019 для Windows. Примеры кода для данного примера размещены в каталоге **ch9_data_virtualization\sqldatahub\saphana**.

В этом примере источник данных отличается от используемых в предыдущих разделах, так как для него требуются данные об источнике данных ODBC и строка подключения. Вот как выглядит создание внешнего источника данных для этого примера:

```
CREATE EXTERNAL DATA SOURCE SAPHANAServer
WITH (
LOCATION = 'odbc://<datasource>',
CONNECTION_OPTIONS = 'Driver={HDBODBC};ServerNode=<server>:<port>',
PUSHDOWN = ON,
CREDENTIAL = SAPHANACredentials
)
GO
```

> **Совет.** Вот важный совет по использованию коннектора данных ODBC с масштабируемыми группами, потому что в этом месте я столкнулся с проблемами при первой настройке этих сценариев. Необходимо установить драйвер ODBC, который вы используете, на каждом узле масштабируемой группы. Если вы этого не сделаете, то можете периодически получать ошибки при выполнении запросов. Это связано с тем, что при наличии масштабируемой группы любой из узлов может использоваться для выполнения запросов к внешнему источнику данных, даже если это не готовый к масштабированию запрос.

Документация, которая поможет вам разобраться с коннекторами ODBC, находится по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-configure-odbc-generic. Я должен сказать вам, что коннекторы ODBC открывают некоторые интересные возможности для использования SQL Server в качестве концентратора данных. Один клиент Microsoft на большом мероприятии спросил меня о возможности применения Polybase с Office 365. Я не знал ответа на этот вопрос и потому спросил: «Есть ли драйвер ODBC для O365?» Оказывается, такой драйвер есть на https://marketplace.visualstudio.com/items?itemName=CDATASOFTWARE.Office365ODBCDriver. Подождите немного. Возможно, вы когда-нибудь увидите, как я создаю демонстрацию, показывающую, как SQL Server выполняет запросы к моей электронной почте!

Обсуждение внешних таблиц

Теперь, когда вы увидели, как работает Polybase, и рассмотрели несколько примеров, я должен сказать, что есть несколько областей, которые следует учитывать при принятии решения о том, нужно ли использовать Polybase с SQL Server 2019.

Дополнительный семантический слой

Я позаимствовал эту концепцию у моего коллеги Трэвиса Райта (Travis Wright). Идея состоит в том, что Polybase позволяет вам определять объекты, используя соглашения об именах, находящиеся под вашим контролем, а не использовать соглашения об именах объектов из внешних источников данных.

Другими словами, вы можете использовать семантику политик и процедур, которые применяете в SQL Server. При создании внешних таблиц вы используете соглашения, схемы и защищаемые объекты SQL Server, находящиеся под вашим контролем. Добавьте это к возможности соединения с локальными таблицами SQL Server, используйте UNION для объединения с локальными таблицами, а затем создавайте представления (views) поверх этих конструкций.

Также помните, что Polybase определяется на уровне пользовательской базы данных, поэтому все объекты защищены и контролируются владельцем пользовательской базы данных.

Внешние таблицы или связанные серверы?

Один из самых частых вопросов, которые мне задают о Polybase и внешних таблицах, заключается в том, отличаются ли они от связанных серверов, которые поддерживались начиная с версии SQL Server 7.0.

Мы представили сравнение технологий в документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-faq?view=sql-server-ver15#polybase-vs-linked-servers.

Наиболее заметным отличием является то, что связанные серверы определяются на уровне экземпляра и используют OLE-DB для доступа к данным из другого источника данных. Polybase определяется на уровне пользовательской базы данных и использует ODBC для доступа к внешним данным.

Ограничения

Ограничение номер один, о котором вам нужно знать, – это то, что Polybase является, по большей части, решением только для чтения данных (исключение состоит в том, что вы можете использовать операторы INSERT для внешних таблиц, созданных на HDFS).

Я также столкнулся с несколькими проблемами с поддерживаемыми типами данных в SQL Server для внешних таблиц. Следующие типы данных не поддерживаются для внешних таблиц (EXTERNAL TABLE) в SQL Server 2019:

- VARCHAR(MAX);
- GEOGRAPHY;
- Computed Columns;
- JSON.

Резюме

Я полагаю, что с добавлением новых встроенных коннекторов и поддержкой ODBC Polybase получит большее распространение, чем ранее, когда она была впервые включена в состав в SQL Server 2016. Возможность доступа и выполнения запросов к данным из многих различных источников данных без перемещения данных – весьма убедительный пример. На самом деле даже после того, как я представил возможности SQL Server 2019 в сентябре 2019 года, это одна из наиболее востребованных функций новой версии SQL Server у наших клиентов. Тот факт, что вы можете выполнять запросы к данным Oracle через SQL Server и не устанавливать специальное программное обеспечение на SQL Server, открыл пользователям глаза.

Polybase может стать частью стратегии перехода с Oracle на SQL Server. Просмотрите эту запись выступления, которое я провел вместе со своим коллегой Амитом Банерджи (Amit Banerjee) на Microsoft Ignite в 2019 году, где Амит показывает, как использовать Polybase с SQL Server 2019 для

стратегии постепенного перехода с Oracle на SQL Server (https://myignite.techcommunity.microsoft.com/sessions/65955).

Важно, чтобы вы прочитали эту главу, прежде чем приступите к главе 10, потому что в главе 10 будет обсуждаться новое решение для SQL Server 2019, которое включает в себя концепцию виртуализации данных и Polybase и известно под названием кластеры больших данных SQL Server (SQL Server Big Data Clusters).

Глава 10

Кластеры больших данных в SQL Server

Если вы помните, в главе 1 я показал основные новые возможности SQL Server 2019. На рис. 10.1 изображена первая основная функция, приведенная в верхнем левом углу рис. 1.3 первой главы.

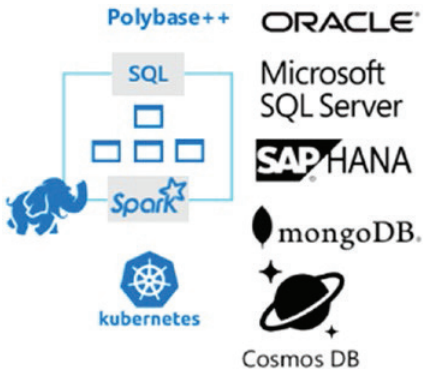


Рис. 10.1. Интеллектуальная обработка данных

Более точным представлением этой инфографики является интеллектуальная обработка всех ваших данных. Это связано с тем, что функциональность, показанная на рис. 10.1, представляет собой нечто большее, чем просто Polybase++. Когда вы прочитаете эту главу, то узнаете больше о том, что я имею в виду под интеллектуальной обработкой всех ваших данных.

На рис. 10.1 в двух словах показано, что стоит за кластерами больших данных (Big Data Clusters, BDC) SQL Server; однако рисунок не полностью раскрывает их потенциал. То, что начиналось как проект Aris, о котором я рассказал в главе 1, стало *продуктом внутри продукта* и одной из самых интересных историй о SQL Server 2019.

Если я начал с обсуждения Project Aris в главе 1 и этой инфографики на рис. 10.1, почему кластеры больших данных SQL Server появились лишь в десятой главе?

Когда я планировал эту книгу, изначально я планировал рассказ о BDC в первой главе, и основной рассказ начался очень хорошо! Однако, поскольку я больше думал о выпуске всей книги, то оставил этот рассказ на потом,

включив его в одну из последних глав. Это имело смысл по следующим причинам:

- мне нужно было познакомить вас с основами SQL Server 2019 для Linux, Containers, Kubernetes и Polybase, чтобы вы смогли понять концепцию BDC. Вот почему эти темы обсуждались в главах 6, 7, 8 и 9;
- вы увидите, что элементом BDC является **ведущий экземпляр SQL Server** (SQL Server Master Instance). Когда разговор зайдет о ведущем экземпляре SQL Server, необходимо, чтобы вы уже были знакомы с другими основными возможностями SQL Server 2019, которые входят в состав этого элемента BDC;
- для создания такой возможности, как BDC, потребовались серьезные усилия многих членов команды, занимавшихся ее проектированием, сборкой и написанием программного кода. Таким образом, это самый большой компонент SQL Server 2019, который был полностью завершен во время наших сборок CTP и предварительных выпусков. Я хотел подождать до заключительной части работы над книгой, чтобы предоставить вам самую свежую и точную информацию о BDC, рассказать, для чего вы можете использовать кластеры больших данных и как они работают.

Из этой главы вы узнаете, каким образом BDC решают некоторые интересные задачи, актуальные для современного профессионала в области обработки данных:

- как я показал в главе 9, посвященной Polybase, профессионалы в области обработки данных должны иметь доступ к источникам данных в своей организации, размещенным вне SQL Server. Они хотели бы иметь возможность доступа к данным из различных источников практически без перемещения данных;
- многие профессионалы в области обработки данных активно интересуются темой больших данных. В следующем разделе я подробнее расскажу о некоторых аспектах, связанных с термином «большие данные»; но когда встречается этот термин, обычно подразумевается система, основанная на Hadoop;
- некоторые организации никогда не вкладывали средства в Hadoop, поэтому они хотели бы получить руководство по развертыванию системы Hadoop или даже автоматизировать развертывание системы Hadoop для хранения неструктурированных или полуструктурированных данных. В таких ситуациях обычно речь идет о данных *большого объема*, в то время как данные, хранящиеся на их SQL Server, считаются данными, имеющими *большую ценность*;
- кроме того, многим организациям требуются более совершенные защита и управление системой Hadoop, аналогичные тому, как это

сегодня реализовано в SQL Server. Им нужна полная экосистема для создания озера данных, которое легко развертывается, защищается и масштабируется и использует лучшие современные технологии как для SQL Server, так и для больших данных;

- организации хотят вкладывать больше средств в машинное обучение (ML) и хотят создавать и развертывать приложения ML, которые являются масштабируемыми и безопасными и работают близко к источникам данных, с которыми работают модели ML. Я слышал, что клиенты говорят, что им нужна комплексная платформа для машинного обучения.

Рисунок 10.2 – это схема, которую мы использовали для обсуждения трех основных аспектов решения, которые пытаются воплотить кластеры больших данных.

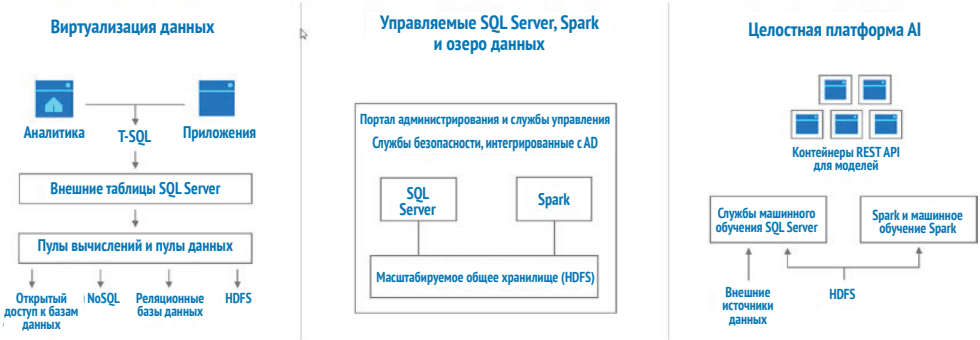


Рис. 10.2. Решения, используемые в кластере больших данных

В этой главе я намерен ответить на следующие вопросы:

- **Почему** мы назвали это решение кластерами больших данных?
- **Какие** возможности вы получаете при развертывании кластера больших данных?
- Как разворачивать кластер больших данных?
- Какова архитектура кластеров больших данных и **как она работает**?
- Как используется кластер больших данных?
- Как используется **машинное обучение** с кластером больших данных?
- Как осуществляются **управление и контроль** кластера больших данных?

Кажется, всего перечисленного хватило бы на отдельную книгу, поэтому я не могу глубоко погрузиться во все эти темы. Тем не менее я раскрою некоторые подробности, которых нет в документации и которые, как я полагаю, вы должны знать. А также приведу практические примеры и советы относительно BDC и расскажу о том, почему BDC является важным решением для SQL Server 2019.

Примечание. В этой главе я иногда буду ссылаться на информацию и примеры, приведенные на семинаре, подготовленном моим коллегой Баком Вуди (Buck Woody) под названием **«Мастерская: кластеры больших данных SQL Server – архитектура»** (Workshop: SQL Server Big Data Clusters – Architecture). Вы можете найти эту информацию по ссылке https://github.com/Microsoft/sqlworkshops/tree/master/sqlserver2019bigdataclusters. Это отличный ресурс, дополняющий данную главу.

Зачем нужны кластеры больших данных, и почему они так называются?

Я уверен, что у нашей команды есть много причин назвать это решение кластерами больших данных SQL Server (SQL Server Big Data Clusters). Для меня же ответ на прозвучавший вопрос очень прост. В этом решении мы внедряем и объединяем три основные технологии:

- **SQL Server** – SQL Server будет центром доступа к данным в кластере. Это полная версия SQL Server, включающая все то, что я описал в этой книге, работающая в контейнере на основе образа ОС Linux;
- **большие данные** – мы внедряем технологии больших данных, такие как HDFS и Spark;
- **кластер.** Мы используем кластер Kubernetes для развертывания и запуска различных контейнеров, чтобы обеспечить единую, полную, целостную систему.

По мере того как вы будете знакомиться со следующими несколькими разделами этой главы, вы узнаете, как мы интегрируем эти технологии.

Я хочу остановиться и сделать акцент на моем понимании термина «*большие данные*» и на том, как мы решали, какие элементы важно включить в это решение для кластеров больших данных. Мой коллега Бак Вуди (Buck Woody) из Microsoft написал отличную заметку в блоге на тему «Большие данные» по ссылке https://buckwoody.wordpress.com/2019/08/26/big-data-is-just-data/.

Мне нравится следующее определение термина «большие данные»: большие данные – это любые данные, которые вы не можете обработать за приемлемое время с помощью имеющихся у вас систем. Для SQL Server это означает, что в вашей организации могут иметься данные, для хранения которых не подходит система управления реляционными базами данных (Relational Database Management System, RDBMS), такая как SQL Server. Подобная ситуация может быть следствием многих разнообразных причин, в числе которых можно назвать объем, структуру, происхождение данных и сложность их преобразования в реляционные таблицы.

Было интересно читать об истории проекта Hadoop. Изначально основатели Hadoop хотели, чтобы файловая система позволяла распределять огромные объемы данных по кластеру, созданному на основе стандарт-

ных (недорогих) аппаратных средств. Они назвали это решение файловой системой Google (Google File System) (в действительности это более сложная история; вы можете прочитать об истории появления Hadoop по ссылке https://en.wikipedia.org/wiki/Apache_Hadoop#History). Целью данного проекта было решить проблему, обозначенную Баком при определении термина «большие данные».

На мой взгляд, используя кластеры больших данных SQL Server (SQL Server Big Data Clusters), мы действительно предлагаем единую систему, которая объединяет возможности хранилищ данных и обработки данных. Мы используем SQL Server для хранения доступа к данным, находящимся в реляционных таблицах. Кроме того, мы разворачиваем кластер распределенной файловой системы Hadoop (Hadoop distributed file system, HDFS), позволяющий хранить данные в неструктурированном или полуструктурированном формате. Ключевым компонентом, который делает эту систему особенной, является то, что они интегрированы. С помощью Polybase вы можете объединять таблицы из SQL Server (и других источников, таких как Oracle, Teradata и MongoDB) с файлами HDFS взаимодополняющим способом, использующим эффективную интеграцию и обеспечивающим масштабируемую производительность.

Есть еще кое-что в этой истории; в следующем разделе я опишу, какую ценность вы получите, разворачивая кластер больших данных SQL Server.

Что входит в состав кластеров больших данных?

Я описал кластер больших данных (Big Data Cluster, BDC) SQL Server как продукт, являющийся частью другого продукта. Это связано с тем, что при развертывании BDC вы получаете множество ценных инструментов и функциональных возможностей, перечисленных ниже.

SQL Server 2019

BDC поставляется с экземпляром SQL Server 2019, работающим в контейнере, использующем образ ОС Linux. Это означает, что все возможности SQL Server 2019 для Linux также включены в BDC. В их число входят аутентификация Active Directory и высокая доступность, с поддержкой групп доступности Always On (Always On Availability Groups).

Polybase

Компонент Polybase для SQL Server с BDC устанавливается и включается автоматически. Это означает, что вы получаете встроенные коннекторы для SQL Server, Oracle, Teradata, MongoDB и Hadoop. Кроме того, BDC поставляется со *специальными коннекторами* для оптимизированного доступа к

файлам HDFS и кешам данных в кластере. Более того, даже несмотря на то, что SQL Server 2019 в Linux не поддерживает группы масштабирования Polybase (Polybase Scale-Out Groups), BDC включает реализацию групп масштабирования Polybase с использованием концепции, называемой пулом вычислений (Compute Pool), о которой я расскажу подробнее в разделе «Архитектура кластера больших данных».

Распределенная файловая система Hadoop (HDFS)

Когда вы будете использовать решение BDC, оно развернет кластер хранения данных HDFS при помощи набора инструментов Apache Hadoop (Apache Hadoop – свободно распространяемый набор утилит, библиотек и фреймворк с открытым исходным кодом). В вашем распоряжении будет несколько разных способов доступа к файлам, хранящимся в кластере HDFS в BDC, включая способ, использующий Polybase для обеспечения возможности работы с данными в SQL Server. Мы также предоставляем метод для подключения собственного внешнего хранилища HDFS к локальному хранилищу HDFS в BDC, концепцию, которую называем *многоуровневой концепцией хранения HDFS* (HDFS Tiering).

Spark

При развертывании BDC устанавливается фреймворк Apache Spark, предоставляющий еще один способ для анализа и обработки данных. Мне нравится определение фреймворка Spark, данное Баком Вуди: «Apache Spark – это аналитический механизм для обработки больших данных. Он может работать с данными, хранящимися в HDFS, и имеет коннекторы для работы с данными в SQL Server». Взаимодействие со Spark будет осуществляться посредством Spark Jobs, которые будут получать данные, размещенные внутри кластера. Более подробно об использовании Spark с BDC я расскажу в разделе «Использование Spark».

Кеш данных

В нашей документации говорится, что мы предоставляем *витрину данных*, и я полагаю, что этот термин корректен с технической точки зрения. Однако для меня это *кеш данных*. Я называю это кешем данных, потому что мы предоставляем специализированный набор экземпляров SQL Server, оптимизированных для хранения результатов запросов к внешним источникам данных Polybase. Представьте сценарий, в котором вы хотите *сохранить набор результатов*, обновляемых еженедельно, для целей отчетности. Эти результаты могут быть получены из запросов Polybase с использованием множества различных источников данных, и наш кеш данных в BDC является идеальным решением для этого. Мы реализуем кеш данных в компоненте, называемом *пулом данных* (Data Pool), о котором я расскажу более подробно в разделе «Архитектура кластера больших данных».

Инструменты и сервисы

Для облегчения развертывания, использования и управления BDC мы предоставляем набор инструментов, доступных как часть решения. Вы увидите, что инструмент Azure Data Studio, с которым вы познакомились в этой книге, станет ключевой частью общего решения BDC, включая поддержку *записных книжек* (Notebooks).

Кроме того, мы развертываем набор контейнеров в качестве *сервисов*, которые помогают координировать и управлять BDC. В документации эти сервисы названы *Контроллером* (Controller), и рассказу о том, как работает контроллер, посвящено несколько разделов этой главы.

Конечные точки

Вам потребуется возможность подключения к BDC для всех типов задач, поэтому мы предоставляем ряд *конечных точек сервисов*. В их число входят конечные точки для подключения к SQL Server, HDFS и Spark, а также нескольких сервисов управления и мониторинга. О конечных точках будет рассказываться на протяжении данной главы.

Развертывание приложений

Кластеры больших данных SQL Server позволяют исполнять программный код посредством операторов T-SQL и Spark Jobs. Службы машинного обучения SQL Server (SQL Server Machine Learning Services) и платформа расширяемости (включая языковые расширения) также позволяют запускать код на R, Python и Java, интегрированный с SQL Server. Поскольку BDC развертывается в кластере Kubernetes, мы хотим предоставить разработчикам удобный способ развертывания приложений в BDC, а также открытый интерфейс для взаимодействия с этими приложениями и обеспечить приложению доступ к источникам данных, подключенным к BDC, таким как таблицы SQL Server и внешние таблицы.

Таким образом, BDC предоставляет концепцию *развертывания приложений* (Application Deployment) для приложений, написанных на R, Python, MLeap и SSIS. Развертывание приложений является ключевой концепцией использования BDC в качестве комплексной платформы машинного обучения. Более подробно о развертывании приложений я расскажу в разделе «Использование кластеров больших данных» далее в этой главе.

Машинное обучение

Я уже говорил, что одним из решений, предоставляемых BDC, является комплексная платформа для машинного обучения. Это становится возможным благодаря таким элементам BDC, как:

- службы машинного обучения SQL Server;
- SparkML;

- MLeap;
- пакеты машинного обучения;
- развертывание приложений.

Я расскажу об этом более подробно в разделе «Машинное обучение и кластеры больших данных» этой главы.

Просмотрите внимательно данный список! Теперь вы понимаете, почему кластеры больших данных SQL Server – это продукт внутри продукта? Наша история становится все интереснее. Продолжайте читать!

> **Примечание.** Семинар Бака Вуди **«Кластеры больших данных SQL Server – архитектура»** (Workshop: SQL Server Big Data Clusters – Architecture) содержит страницу с описанием компонентов BDC в разделе Module 2.0. Используйте этот материал как дополнительный ресурс, позволяющий понять, что находится «внутри» BDC.

Подготовительные шаги для использования обучающих материалов

Прежде чем погружаться в тему развертывания BDC, я должен рассказать, где находятся обучающие материалы, которые буду использовать в этой главе. Вместо того чтобы создавать и рассматривать конкретные примеры и сценарии, я буду использовать несколько примеров из следующих источников:

- **репозиторий GitHub с примерами для SQL Server (SQL Server Samples GitHub Repo).** Несколько примеров, которые я буду использовать и о которых буду рассказывать, размещены по ссылке https://github.com/Microsoft/sql-server-samples/tree/master/samples/features/sql-big-data-cluster;
- **семинар по кластерам больших данных SQL Server 2019 (SQL Server 2019 Big Data Cluster Workshop)** – у Бака Вуди есть несколько замечательных примеров, которые я буду использовать. Они размещены по ссылке https://github.com/Microsoft/sqlworkshops/tree/master/sqlserver2019bigdataclusters.

Для использования этих материалов вам потребуется:

- развернутый кластер больших данных с SQL Server 2019 (SQL Server 2019 Big Data Cluster). Во время написания главы этой книги я использовал версию-кандидат SQL Server 2019, которая очень близка к финальной версии SQL Server 2019. Более подробно о требованиях, включая клиентские инструменты, я расскажу в следующем разделе этой главы «Развертывание кластеров больших данных»;

- клиент Windows, macOS или Linux для развертывания и запуска примеров сценариев или запросов T-SQL. Почти все инструменты, используемые при развертывании и в приведенных примерах, работают на Windows, macOS и Linux. Я также рекомендую установить и использовать Azure Data Studio (ADS) (этот инструмент можно загрузить по ссылке https://docs.microsoft.com/en-us/sql/azure-data-studio/download-azure-data-studio). Для успешного применения кластеров больших данных вам потребуется использовать ADS и записные книжки (notebooks).

Развертывание кластеров больших данных

Я продемонстрирую вам свой опыт развертывания кластеров больших данных SQL Server и затем опишу их компоненты и архитектуру. Когда я писал эту главу, мне было сложно решить, что следует показать сначала – архитектуру или развертывание. Я подумал, что важно сначала развернуть кластер, а затем описать развернутое решение.

> **Примечание.** Все программное обеспечение в BDC развертывается в виде контейнеров в кластере Kubernetes. При работе с BDC предполагается, что вы развертываете свой собственный кластер Kubernetes, но также опционально предоставляются инструменты, помогающие развернуть k8s.

Планирование развертывания

Развертывание BDC необходимо спланировать. Далее я расскажу о своем опыте планирования развертывания BDC, поскольку это может помочь вам, когда вы будете самостоятельно планировать и выполнять процедуру развертывания. Если вы планируете развертывание BDC в промышленной среде, то рекомендую вам прочитать раздел «Настройка развертывания для промышленной среды» этой главы.

Определитесь с k8s

Первое решение, которое необходимо принять при развертывании BDC, – это выбор варианта размещения Kubernetes (k8s). BDC поддерживает развертывание на k8s в общедоступном облачном провайдере **Azure Kubernetes Service** (AKS), или на вашем собственном сервере Linux, или на виртуальной машине, где развернут k8s (например, если вы самостоятельно развернули k8s с помощью kubeadm). Я ожидаю, что список других известных поставщиков k8s, которые будут поддерживаться BDC, будет расширяться с выходом SQL Server 2019 и более поздних версий и будет включать Azure Stack, Red Hat OpenShift и другие платформы. На настоящий момент времени, когда я пишу эту книгу, вы можете развернуть BDC при развертывании k8s на Windows Server, но для этого сценария потребуются виртуальные машины Linux, работающие на Windows Server.

Наши инструменты для развертывания BDC создадут серию объектов pod с контейнерами (в большинстве случаев они будут иметь несколько контейнеров) в k8s для поддержки системы BDC. Мы также будем развертывать и использовать другие объекты k8s, такие как Load Balancer, Persistent Volume Claim, ReplicaSet и StatefulSet.

Как только вы определитесь с вариантом размещения k8s, вы можете либо самостоятельно развернуть k8s, либо использовать сценарии, которые мы создали, для совместного развертывания k8s и BDC.

Для любого варианта основным требованием для развертывания BDC в среде dev/test является виртуальная машина (ВМ) Linux или компьютер (в случае если вы развертываете свое решение на AKS, выберите размер виртуальной машины), удовлетворяющая следующим требованиям к ресурсам:

- 64 ГБ ОЗУ;
- 8 процессоров (могут быть логическими объектами);
- для AKS – размер виртуальной машины Azure, поддерживающей не менее 24 дисков;
- если вы планируете развернуть более одного узла BDC, каждый узел (ВМ) должен будет соответствовать этим требованиям к ресурсам.

> **Примечание.** Мы со Славой Окс (Slava Oks) беседовали о необходимости уменьшить требования к ресурсам для «Developer Edition» BDC, для которого не нужно так много оперативной памяти. Я сказал Славе, что в идеале хотел бы развернуть BDC на своем ноутбуке для демонстрации основных возможностей.

В предоставленных нами сценариях и записных книжках по умолчанию выбирается размер виртуальной машины Azure: Standard_L8s_v2, но если вы выберете виртуальную машину Azure для AKS с 64 ГБ, 8 ЦП и 24 дисками, все сценарии должны работать. Подробнее о размере виртуальных машин Azure можно прочитать по ссылке https://docs.microsoft.com/en-us/azure/virtual-machines/linux/sizes-general.

Для моего варианта развертывания я собираюсь использовать AKS и предоставленный нами сценарий, который развернет кластер AKS и BDC за один шаг. Я рекомендую при планировании развертывания обратиться к следующим ресурсам, где вы найдете полезную информацию:

- https://docs.microsoft.com/en-us/sql/big-data-cluster/deploy-get-started;
- https://docs.microsoft.com/en-us/sql/big-data-cluster/deployment-guidance.

Выберите клиентское приложение и загрузите необходимый инструментарий

После того как вы определились со стратегией k8s, вам нужны инструменты для развертывания BDC. Очень важно убедиться, что на клиентском узле установлены все необходимые инструменты, прежде чем вы попы-

таетесь развернуть BDC. В документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/deploy-big-data-tools, представлен список инструментов, которые вам понадобятся. Для своего клиента я выбрал «ноутбук в облаке». Это означает, что я установил Windows 10 на виртуальной машине Azure и использовал все ресурсы этой виртуальной машины для установки инструментария BDC.

Данный инструментарий включает в себя следующие компоненты:

Python – Python является ключевым компонентом, используемым несколькими различными инструментами, и доступен на всех платформах ОС. Инструмент azdata, необходимый для установки BDC, написан на Python. Мне Python был необходим, поскольку я использовал сценарий Python для развертывания AKS и BDC за один шаг. Чтобы установить Python для Windows, я просто загружаю последнюю версию Python с сайта www.python.org/downloads/release/python-374/;

kubectl – как вы узнали из главы 8, kubectl – это инструмент, специально разработанный для отправки запросов на сервер API k8s. Это ваш программный интерфейс для Kubernetes.

Я уже установил kubectl на моем компьютере, работающем под Windows. Я проверил версию kubectl, это была версия 1.14. В примечаниях к документации сказано: «Вы должны использовать kubectl версии 1.10 или более поздней. Кроме того, версия kubectl должна удовлетворять следующему правилу: младший номер версии должен отличаться не более чем на 1 (плюс или минус) от версии вашего кластера Kubernetes». Поскольку я использую AKS, то выполнил команду, чтобы посмотреть, какие версии будут применяться в моем развертывании AKS, и обнаружил, что самая последняя поддерживаемая версия – 1.14.6; так что с моей конфигурацией все должно быть в порядке. Подробнее о проверке наличия поддерживаемых версий AKS в вашем кластере можно узнать по ссылке https://docs.microsoft.com/en-us/azure/aks/supported-kubernetes-versions;

azdata – инструмент, известный под названием mssqlctl в ранних версиях SQL Server 2019, предназначавшихся для предварительного пользовательского тестирования. Он крайне важен для развертывания и управления BDC. Он написан на Python, и вы можете рассматривать его как «kubectl» для BDC.

Чтобы убедиться, что я правильно установил azdata, я просто запустил azdata из командной строки и увидел, как выглядит его интерфейс. Результаты показаны на рис. 10.3.

Вы можете найти полную документацию для azdata, перейдя по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/reference-azdata;

Azure Data Studio (ADS). Этот кросс-платформенный инструмент с открытым исходным кодом можно использовать для выполнения за-

просов, развертывания, управления и навигации по данным в BDC. Хотя SQL Server Management Studio (SSMS) можно использовать для подключения к главному экземпляру SQL Server в BDC, ADS имеет функции и расширения, специально предназначенные для BDC, включая поддержку записных книжек (Notebooks).

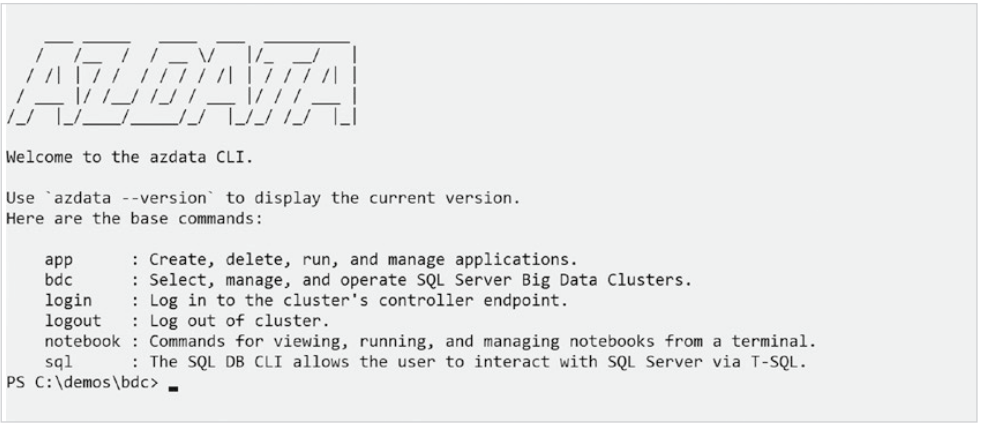


Рис. 10.3. Интерфейс командной строки azdata

Работая над материалами этой главы, я использовал сборку ADS Insiders, опубликованную по ссылке https://github.com/microsoft/azuredatastudio#try-out-the-latest-insiders-build-from-master, но я ожидаю, что к моменту выпуска SQL Server 2019 в вашем распоряжении будет иметься официальная версия ADS, включающая все, что необходимо для BDC. Вы можете получить последнюю версию ADS по ссылке https://docs.microsoft.com/en-us/sql/azure-data-studio/download.

Я также взял последнюю версию расширения SQL Server 2019 для ADS, размещенную по ссылке https://docs.microsoft.com/en-us/sql/azure-data-studio/sql-server-2019-extension, и установил файл vsix. (Вы можете проигнорировать предупреждение о сторонних расширениях, потому что это расширение выпущено Microsoft.) Во время его установки довольно трудно определить, устанавливается оно или установка уже завершена, но подождите несколько минут, и вы увидите в правом нижнем углу сообщение, например «Завершена установка расширения microsoft.sql-vnext»;

az – если вы используете AKS, вам потребуется интерфейс командной строки Azure для входа в Azure, а также для развертывания и управления AKS;

curl – curl означает «URL-адрес клиента». Это популярный инструмент для копирования данных с определенного URL-адреса (в частности, файлов, хранящихся на веб-сайтах). В моем случае он был установлен в Windows 10. Curl – отличный инструмент не только для копи-

рования удаленных сценариев для использования с BDC, но и для копирования данных в кластер BDC HDFS.

Выбор способа развертывания

Теперь, когда вы знаете, какой тип кластера k8s развернете, и загрузили все необходимые инструменты, вы должны выбрать способ развертывания:

- «одношаговый» способ для развертывания AKS и BDC с использованием **Python**, сценарий которого можно найти по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/quickstart-big-data-cluster-deploy?view=sql-server-ver15;
- «одношаговый» сценарий оболочки Bash для развертывания k8s и BDC в кластере k8s с использованием **kubeadm**, сценарий которого можно найти по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/deployment-script-single-node-kubeadm?view=sql-server-ver15;
- создание своего собственного кластера AKS или k8s, с последующим развертыванием BDC с помощью инструмента **azdata**, о котором вы можете прочитать по адресу https://docs.microsoft.com/en-us/sql/big-data-cluster/deployment-guidance;
- использование Azure Data Studio (ADS) для развертывания BDC вместе с новым кластером AKS, в существующем кластере AKS или в существующем кластере k8s, который вы развернули с помощью kubeadm.

На рис. 10.4 показано, как выбрать вариант развертывания BDC в ADS.

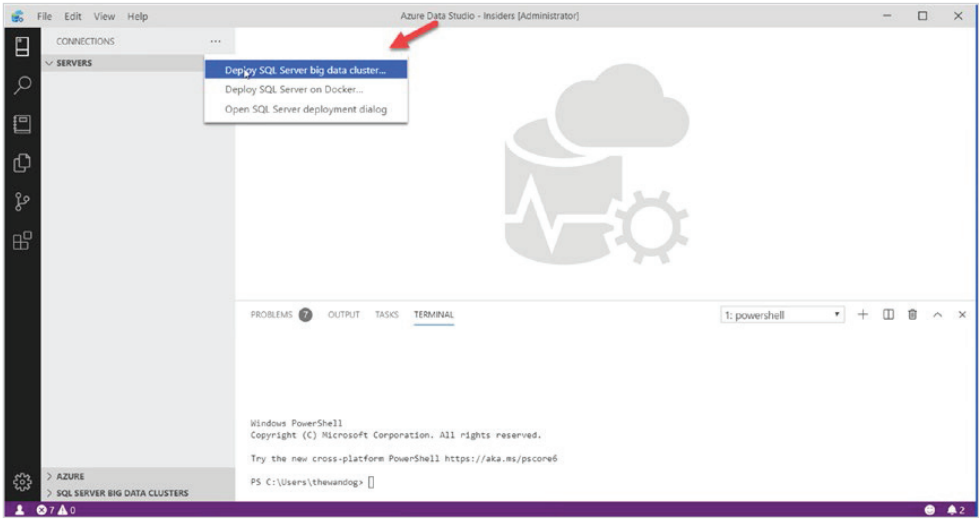


Рис. 10.4. Выбор варианта развертывания BDC в Azure Data Studio

На рис. 10.5 показан выбор метода развертывания.

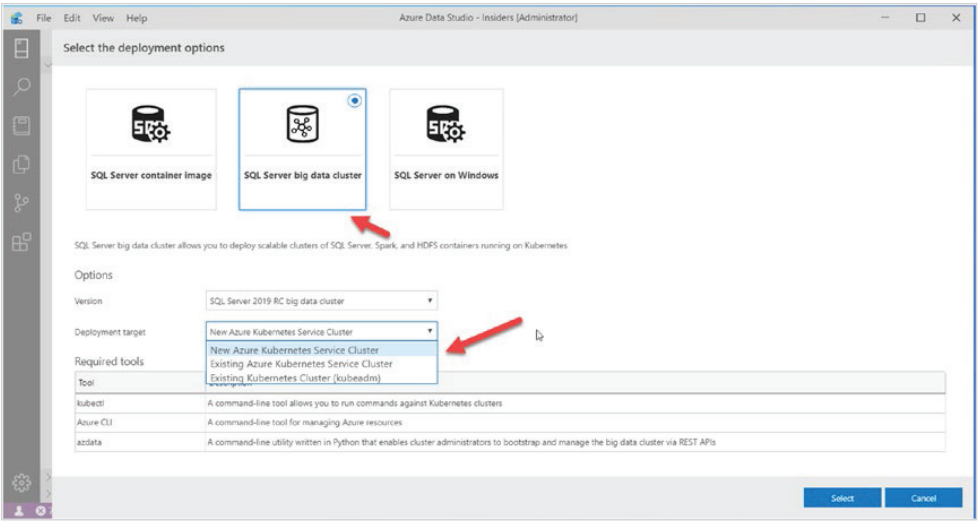


Рис. 10.5. Параметры развертывания для BDC в Azure Data Studio

Автономное развертывание

Я хочу особо упомянуть автономное развертывание (offline deployment). Если вам необходимо развернуть свой кластер автономно, поскольку ваш кластер k8s не подключен к интернету (по крайней мере, когда вам нужно развернуть BDC), то для этого случая мы специально описали в документации, как получить образы наших контейнеров и развернуть BDC на k8s. Вам все еще понадобятся все инструменты, которые я упоминал в этой главе, для развертывания в автономном режиме. Подробная информация доступна по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/deploy-offline.

Развертывание BDC

Чтобы дать вам представление о развертывании BDC, я собираюсь развернуть BDC на AKS и использовать сценарий Python, предоставленный в «одношаговом» способе. Вы можете прочитать подробности о том, как применить этот подход, по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/quickstart-big-data-cluster-deploy.

Я выбрал настройки по умолчанию, за исключением того, что мне нужно было развернуть кластер AKS и BDC в области eastus2.

Используемый сценарий Python по сути является «оберткой» для az и azdata. Он использует выбранные вами параметры (или переменные, или значения по умолчанию) для создания группы ресурсов Azure, кластера AKS и BDC. BDC создается с использованием конфигурации **aks-dev-test**. Это базовая конфигурация для BDC, которая хорошо подходит для сценария разработки или тестирования. О конфигурации рабочих вариантов

развертывания для промышленных сред я расскажу в разделе «Настройка развертывания для промышленной среды» далее в этой главе.

Для развертывания BDC потребуется время. Для решения BDC нужно развернуть множество объектов pod и контейнеров, и этот процесс займет больше времени, если вы также развернете кластер k8s. Я затратил около 20 минут на развертывание с использованием сценария Python в AKS, однако видел случаи, когда это занимало около часа.

Когда вы запустите сценарий Python, то получите примерно такие сообщения:

```
Creating azure resource group: <rgname>
<json details for the resource group>
Creating AKS cluster: <aks cluster name>
<json for the AKS cluster>
Creating SQL Big Data cluster:mssql-cluster
custom\bdc.json created
custom\control.json created
The privacy statement can be viewed at:
https://go.microsoft.com/fwlink/?LinkId=853010

The license terms for SQL Server Big Data Cluster can be viewed at:
https://go.microsoft.com/fwlink/?LinkId=2002534

Cluster deployment documentation can be viewed at:
https://aka.ms/bdc-deploy

NOTE: Cluster creation can take a significant amount of time depending on
configuration, network speed, and the number of nodes in the cluster.

Starting cluster deployment.
Waiting for cluster controller to start.
```

Последнее сообщение `Waiting for cluster controller to start` (Ожидание запуска контроллера кластера) может повторяться несколько раз. Сначала в кластере k8s создается *контроллер*, а затем *служба контроллера* будет использоваться для развертывания оставшейся части BDC.

Далее вы увидите примерно такое сообщение:

```
Cluster controller endpoint is available at <ip address>:<port>
Cluster control plane is ready.
```

А вскоре увидите следующие сообщения:

```
Data pool is ready.
Master pool is ready.
Compute pool is ready.
```

```
Storage pool is ready.
Cluster deployed successfully.
```

Последнее сообщение означает, что AKS и BDC успешно развернуты. Я руководствуюсь принципом «доверяй, но проверяй», поэтому в следующем разделе расскажу о том, как можно убедиться, что развертывание прошло успешно и BDC готов к использованию.

> **Примечание.** Имя для кластера больших данных SQL: `mssql-cluster` становится пространством имен Kubernetes для всех объектов, созданных BDC. Поэтому в моем варианте развертывания **mssql-cluster** является пространством имен k8s.

Проверка выполненного развертывания

Я использовал следующие сценарии проверки работоспособности успешного развертывания AKS и BDC:

- выполните действия, описанные в документации, размещенной по указанной ссылке. В ней описано, как использовать **kubectl** для проверки кластера: https://docs.microsoft.com/en-us/sql/big-data-cluster/quickstart-big-data-cluster-deploy?view=sqlallproductsallversions#inspect-the-cluster;
- войдите в кластер, используя **azdata**, найдите конечную точку контроллера, а затем попробуйте подключиться к SQL Server, чтобы убедиться, что вы можете подключиться к нему. Следуйте инструкциям на странице https://docs.microsoft.com/en-us/sql/big-data-cluster/deployment-guidance?view=sqlallproducts-allversions#endpoints.

  Найдите конечную точку с названием **SQL Server Master Instance Front-End**. Конечной точкой является IP-адрес и порт для подключения к SQL Server.

  Следуйте инструкциям на следующей странице документации, чтобы подключиться к SQL Server в BDC с помощью Azure Data Studio (ADS): https://docs.microsoft.com/en-us/sql/big-data-cluster/connect-to-big-data-cluster.

  Результат моей проверки возможности подключения к моему BDC в ADS показан на рис. 10.6.

> **Примечание.** При развертывании BDC я использовал инсайдерскую ADS и поэтому должен предупредить вас, что некоторые из показанных на рисунке интерфейсов могут измениться после выпуска официальной версии SQL Server 2019.

- Проверьте общее состояние BDC с помощью следующей команды:

```
azdata bdc status show
```

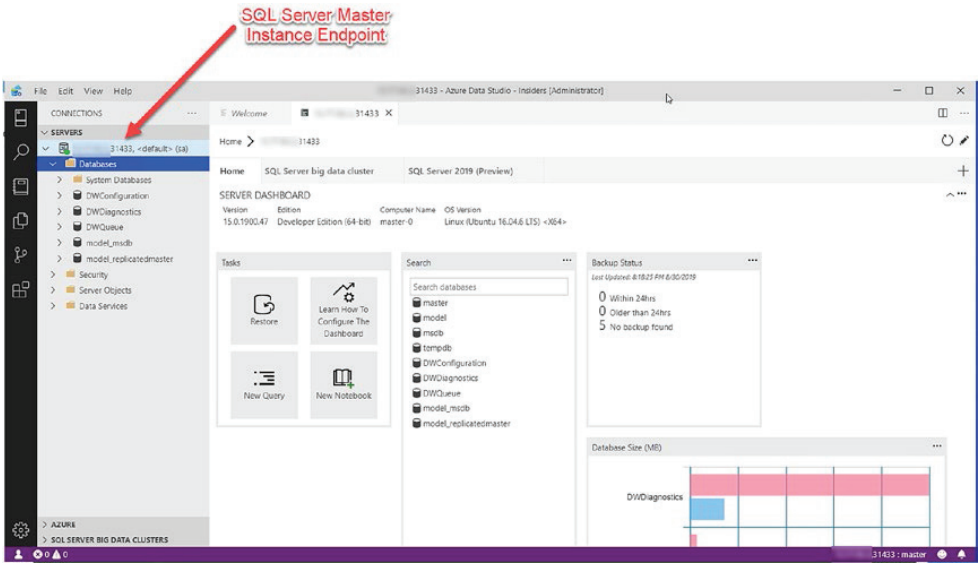


Рис. 10.6. Подключение к SQL Server в BDC после завершения развертывания

Полученный мной для моего кластера BDC результат выглядел так:

```
Mssql-cluster: ready                                                Health Status:
                                                                    healthy
========================================================================================
Services: ready                                                     Health Status:
                                                                    healthy
----------------------------------------------------------------------------------------
Servicename       State       Healthstatus       Details
sql               ready       healthy            -
hdfs              ready       healthy            -
spark             ready       healthy            -
control           ready       healthy            -
gateway           ready       healthy            -
app               ready       healthy            -

Sql Services: ready                                                 Health Status:
                                                                    healthy
----------------------------------------------------------------------------------------
Resourcename      State       Healthstatus       Details
master            ready       healthy            StatefulSet master is healthy
compute-0         ready       healthy            StatefulSet compute-0 is healthy
data-0            ready       healthy            StatefulSet data-0 is healthy
storage-0         ready       healthy            StatefulSet storage-0 is healthy
```

```
Hdfs Services: ready                                          Health Status:
                                                              healthy
-------------------------------------------------------------------------------
Resourcename     State      Healthstatus      Details
nmnode-0         ready      healthy           StatefulSet nmnode-0 is healthy
storage-0        ready      healthy           StatefulSet storage-0 is healthy
sparkhead        ready      healthy           StatefulSet sparkhead is healthy

Spark Services: ready                                         Health Status:
                                                              healthy
-------------------------------------------------------------------------------
Resourcename     State      Healthstatus      Details
sparkhead        ready      healthy           StatefulSet sparkhead is healthy
storage-0        ready      healthy           StatefulSet storage-0 is healthy

Control Services: ready                                       Health Status:
                                                              healthy
-------------------------------------------------------------------------------
Resourcename     State      Healthstatus      Details
controldb        ready      healthy           -
control          ready      healthy           -
metricsdc        ready      healthy           DaemonSet metricsdc is healthy
metricsui        ready      healthy           ReplicaSet metricsui is healthy
metricsdb        ready      healthy           StatefulSet metricsdb is healthy
logsui           ready      healthy           ReplicaSet logsui is healthy
logsdb           ready      healthy           StatefulSet logsdb is healthy
mgmtproxy        ready      healthy           ReplicaSet mgmtproxy is healthy

Gateway Services: ready                                       Health Status:
                                                              healthy
-------------------------------------------------------------------------------
Resourcename     State      Healthstatus      Details
gateway          ready      healthy           StatefulSet gateway is healthy

App Services: ready                                           Health Status:
                                                              healthy
-------------------------------------------------------------------------------
Resourcename     State      Healthstatus       Details
appproxy         ready      healthy            ReplicaSet appproxy is healthy
```

Если не все в порядке, обратитесь к следующей документации для устранения неполадок в кластере: https://docs.microsoft.com/en-us/sql/big-data-cluster/cluster-troubleshooting-commands.

Настройка развертывания для промышленной среды

В моем сценарии развертывания использовалась *конфигурация*, поставляемая с инструментом azdata, предназначенная для кластера, используемого в разработке или тестировании. Конфигурация для azdata определяется в файлах JSON и используется для управления различными типами определений ресурсов в кластере. Вы можете вывести список этих конфигураций с помощью команды

```
azdata bdc config list
```

Файлы JSON, используемые для конфигурации, очень похожи на YAML-файлы Kubernetes, примеры которых вы видели в главе 8. В данном случае JSON-файлы имеют формат, распознаваемый инструментом azdata (так же, как файлы YAML имеют формат, распознаваемый kubectl).

Чтобы узнать, какие параметры можно настроить для развертывания BDC, вы можете выполнить команду, аналогичную приведенной ниже, чтобы посмотреть, как развертывается конфигурация aks-dev-test по умолчанию:

```
azdata bdc config init --source aks-dev-test --target custom
```

Эта команда создает новую папку с именем custom и сохраняет в этом каталоге файлы с именами **bdc.json** и **control.json**. Вы можете внести изменения в эти файлы и выполнить команду, аналогичную следующей, чтобы создать новый BDC с выбранными параметрами конфигурации:

```
azdata bdc create --config-profile custom --accept-eula yes
```

Этот метод описан в документации по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/deployment-guidance?view=sqlallproducts-allversions#customconfig.

Чтобы понять, как вносить изменения в файлы JSON для BDC, вам необходимо познакомиться с архитектурой, о которой будет рассказываться в следующем разделе «Архитектура кластера больших данных». Я ожидаю, что после прочтения раздела, посвященного архитектуре, вы вернетесь к этому разделу, чтобы взглянуть поближе на файлы и методы JSON и изменить их соответствующим образом. Если у вас есть представление о том, что нужно изменить, вы можете использовать следующие рекомендации в нашей документации о том, как вносить изменения в файлы JSON для BDC: https://docs.microsoft.com/en-us/sql/big-data-cluster/deployment-custom-configuration.

Полная документация по конфигурации развертывания BDC, описывающая структуру файлов JSON, находится по адресу https://docs.microsoft.com/en-us/sql/big-data-cluster/reference-deployment-config.

Вам также следует изучить наши сценарии «автоматического развертывания» для Python и bash, чтобы увидеть, как можно создавать k8s и BDC:

- **Python** – https://docs.microsoft.com/en-us/sql/big-data-cluster/quickstart-big-data-cluster-deploy;
- **bash** – https://docs.microsoft.com/en-us/sql/big-data-cluster/deployment-script-single-node-kubeadm.

Эти сценарии предполагают применение одного узла Kubernetes (k8s). Скорее всего, вы захотите использовать несколько узлов в производственном кластере k8s. Затем вы можете решить, как разместить различные компоненты BDC на определенных узлах k8s. Обратитесь к документации по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/deployment-custom-configuration.

Другим важным аспектом настройки BDC для производства является хранилище. Наша документация содержит рекомендации по настройке хранилища BDC в соответствии с конфигурацией хранилища k8s для промышленной среды по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/concept-data-persistence. Каждый объект pod, имеющий хранилище с сохранением состояния в BDC, использует отдельное PVC. Вы можете получить список всех объектов PVC в BDC, выполнив следующую команду:

```
get PersistentVolumeClaim --namespace=mssql-cluster
```

Двумя другими важными аспектами развертывания BDC в промышленной среде являются безопасность и высокая доступность, о которых я расскажу более подробно в разделах «Безопасность» и «Высокая доступность» далее в этой главе.

Архитектура кластера больших данных

Я буду использовать свой развернутый BDC для знакомства с архитектурой кластеров больших данных. Я описал компоненты, которые поставляются с BDC, но это был скорее «список возможностей». Архитектура представляет интерес для читателя, поскольку вы сможете точно узнать, какие объекты pod и контейнеры установили. Знания, почерпнутые из глав 7 и 8, здесь непременно пригодятся.

> **Примечание.** Если вы хотите перейти к использованию BDC, перейдите к следующему разделу «Использование кластеров больших данных». Я считаю этот раздел об архитектуре необязательным, но важным разделом данной главы. Вы всегда можете вернуться к нему, изучив некоторые аспекты использования BDC. Важно знать, что мы создали BDC, так что вам не нужно знать каждую деталь архитектуры. И любая информация, приведенная в этом разделе, может устареть. Я приведу здесь некоторые подробности, скрытые «за кадром», однако они, безусловно, могут измениться со временем.

На рис. 10.7 приведена схема общей архитектуры кластеров больших данных SQL Server (BDC).

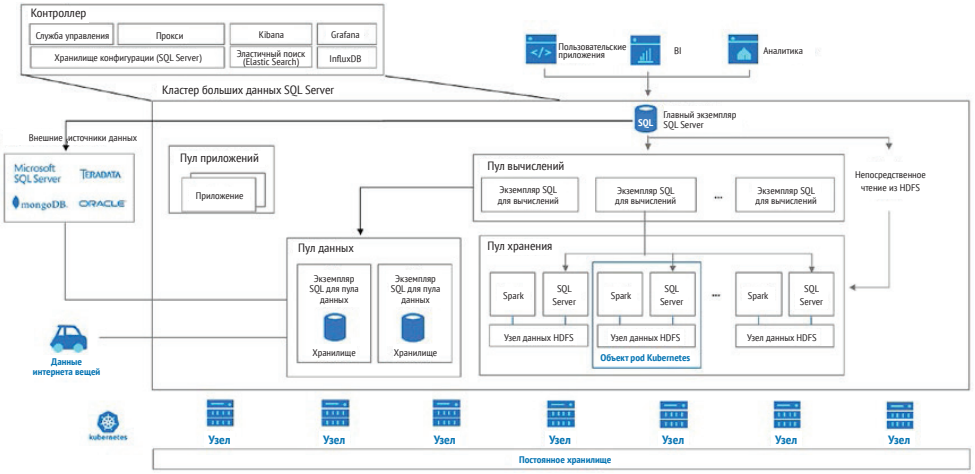


Рис. 10.7. Архитектура кластера больших данных SQL Server

Обратите внимание, что на этой схеме используется термин *пул*. Пул – это логический термин в BDC, который представляет собой набор объектов pod, которые служат определенной цели. Я упоминал некоторые из этих пулов ранее в данной главе, например пул вычислений (Compute Pool) и пул данных (Data Pool). В этом разделе я расскажу более подробно, какие объекты pod и контейнеры входят в состав этих пулов.

Давайте разберем каждый фрагмент рис. 10.7, чтобы составить представление об архитектуре BDC, используя различные команды и визуальные элементы схемы. Один из способов понять архитектуру – это исследовать ее *с точки зрения k8s*.

Когда я запускаю приведенную ниже команду, то получаю список всех объектов pod, развернутых BDC в моем кластере k8s с одним узлом:

```
kubectl get pods --namespace mssql-cluster
```

В результате я получаю следующий список объектов pod и их статусов:

```
NAME                READY   STATUS    RESTARTS   AGE
appproxy-q8zkk      2/2     Running   0          24h
compute-0-0         3/3     Running   0          24h
control-vjwjf       3/3     Running   0          24h
controldb-0         2/2     Running   0          24h
controlwd-l8fmp     1/1     Running   0          24h
data-0-0            3/3     Running   0          24h
data-0-1            3/3     Running   0          24h
gateway-0           2/2     Running   0          24h
```

```
logsdb-0            1/1     Running     0          24h
logsui-f42ln        1/1     Running     0          24h
master-0            3/3     Running     0          24h
metricsdb-0         1/1     Running     0          24h
metricsdc-gtrxn     1/1     Running     0          24h
metricsui-kwh4q     1/1     Running     0          24h
mgmtproxy-nc8tl     2/2     Running     0          24h
nmnode-0-0          2/2     Running     0          24h
sparkhead-0         4/4     Running     0          24h
storage-0-0         4/4     Running     0          24h
storage-0-1         4/4     Running     0          24h
```

Основываясь на этом списке и концепциях, с которыми вы уже успели познакомиться, вы, вероятно, сможете догадаться, где на рис. 10.7 находятся некоторые из объектов pod. Значения в столбце READY показывают, сколько контейнеров запущено в каждом из этих объектов. Нетрудно увидеть, что простой кластер BDC dev/test содержит около 43 контейнеров!

Давайте воспользуемся этим списком для сопоставления объектов pod в кластере k8s с компонентами на рис. 10.7, используя концепцию *пулов*.

Главный экземпляр SQL Server

Главный экземпляр SQL Server (SQL Server Master Instance) представлен в виде объекта pod **master-0**. Основным контейнером, запущенным в этом объекте pod, является контейнер SQL Server Linux. Вы можете использовать следующую команду, чтобы получить подробную информацию о том, как BDC развертывает контейнер SQL Server:

```
kubectl describe pod master-0 --namespace mssql-cluster
```

Одним из важных аспектов организации BDC является использование меток в Kubernetes. Я описал, как использовать метки, в главе 8, посвященной SQL Server и k8s. Посмотрите на следующий фрагмент результатов выполнения предыдущей команды:

```
Labels:         MSSQL_CLUSTER=mssql-cluster
                app=master
                controller-revision-hash=master-7bbc4d95fb
                mssql.microsoft.com/sql-instance=master
                plane=data
                role=master-pool
                statefulset.kubernetes.io/pod-name=master-0
                type=sqlservr
```

Вы можете видеть, как мы применяем некоторые из этих меток для сопоставления с терминами BDC. Например, следующие две метки представляют интерес:

```
plane=data
role=master-pool
```

Если вы выполните следующую команду, то получите все объекты pod в «разрезе данных»:

```
get pods --namespace mssql-cluster -lplane=data
```

Для своего BDC я получил следующий результат:

```
NAME                READY       STATUS        RESTARTS        AGE
appproxy-q8zkk      2/2         Running       0               24h
compute-0-0         3/3         Running       0               24h
data-0-0            3/3         Running       0               24h
data-0-1            3/3         Running       0               24h
master-0            3/3         Running       0               24h
nmnode-0-0          2/2         Running       0               24h
sparkhead-0         4/4         Running       0               24h
storage-0-0         4/4         Running       0               24h
storage-0-1         4/4         Running       0               24h
```

Этот список содержит большинство основных компонентов, представленных на рис. 10.7, за исключением контроллера, о котором я расскажу в следующем разделе главы «Контроллер».

Если вы посмотрите дальше на приведенный выше результат выполнения команды `kubectl description pod`, то увидите структуру контейнера SQL Server, начиная с

```
Containers:
    mssql-server:
```

Если вы вернетесь к главе 8, то увидите, что важными компонентами, задействованными в объекте pod для SQL Server в k8s, были:

- образ контейнера;
- Persistent Volume Claim;
- объект Secret;
- балансировщик нагрузки;
- ReplicaSet.

Результат выполнения команды `kubectl describe`, показанный ранее, содержит все эти компоненты.

Вы можете найти **образ контейнера** для контейнера SQL Server (помните, что я использовал версию SQL Server 2019 Big Data Clusters) в следующем разделе:

```
Image:          mcr.microsoft.com/mssql/bdc/mssql-server-data:2019-RC1-ubuntu
```

Позже в выводимых данных вы увидите список *смонтированных объектов*, которые описывают смонтированные постоянные хранилища для объектов **PersistentVolumeClaim**.

Обратите внимание на следующий фрагмент:

```
/var/opt from data (rw)
```

И на следующий том:

```
Volumes:
  data:
    Type:        PersistentVolumeClaim (a reference to a
                 PersistentVolumeClaim in the same namespace)
    ClaimName: data-master-0
    ReadOnly: false
```

Как вы помните, в главе 8 я показал вам, как сопоставить каталог SQL Server, например `/var/opt`, с объектом PVC.

Вы можете выполнить следующую команду, чтобы просмотреть структуру объекта PVC:

```
describe PersistentVolumeClaim data-master-0 --namespace=mssql-cluster
```

Просмотрев полученный результат, вы можете видеть, что этот объект PVC привязан к классу **StorageClass**, используемому по умолчанию для AKS, и имеет размер 15 ГБ. Это, разумеется, не такой большой объем для хранения данных SQL Server, однако это всего лишь тестовый кластер. Если вам необходимо изменить размер хранилища для реальной конфигурации, то можете прочитать, как это сделать, в нашей документации, опубликованной по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/concept-data-persistence.

Объект **secret** для SQL Server в главе 8 использовался для хранения пароля учетной записи sa для подключения к SQL Server. Развертывание BDC содержит переменную среды с именем **MSSQL_SA_PASSWORD**, которую мне предложил использовать сценарий развертывания, написанный на Python. Для главного экземпляра SQL Server мы создаем файл secret с именем **mssql-sa-password**.

Как вы помните, в главе 8 я показал вам, как создать службу **LoadBalancer** для SQL Server в объекте pod, используемую для подключения к SQL Server. Наш инструмент развертывания BDC создает эту службу для главного экземпляра SQL Server. Чтобы просмотреть объекты для этой службы, вы можете выполнить следующую команду:

```
kubectl get service --namespace=mssql-cluster -lrole=master-pool
```

Полученный результат будет содержать службу **master-svc-external**, внешний IP-адрес и порт.

Последний компонент для объекта pod SQL Server – **ReplicaSet**. В главе 8 я показал вам, как ReplicaSet обеспечивает «базовую высокую доступность (HA)» для k8s для SQL Server. Для BDC мы используем концепцию **StatefulSet**, которая предоставляет функциональность HA, аналогичную ReplicaSet, но с более широкими возможностями. Объекты StatefulSet используются для всех объектов pod в BDC, за исключением контроллера. Объекты StatefulSet позволяют упорядочивать и масштабировать объекты pod и являются ключевым компонентом для обеспечения высокой доступности в BDC. Более подробно о высокой доступности в BDC я расскажу в разделе «Высокая доступность» далее в этой главе.

Если вы посмотрите на результат выполнения команды `kubectl describe`, то увидите следующий фрагмент:

```
Controlled By:        StatefulSet/master
```

Вы можете посмотреть, как мы определяем StatefulSet, выполнив следующую команду:

```
kubectl describe StatefulSet master --namespace=mssql-cluster
```

Обратите внимание, что в объекте pod **master-0** есть два других контейнера:

```
collectd:
fluentbit:
```

Эти контейнеры являются составной частью каждого объекта pod в BDC и используются для сбора данных журналов и метрик, применяемых при управлении и мониторинге BDC.

Документация, в которой вы можете найти информацию о главном экземпляре SQL Server, размещена по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/concept-master-instance.

Более подробно о том, как использовать главный экземпляр SQL Server, я расскажу в разделе «Использование кластеров больших данных» этой главы.

Контроллер

Контроллер – это логический термин, представляющий коллекцию объектов pod и контейнеров. Вы можете получить список объектов pod в контроллере с помощью следующей команды:

```
kubectl get pods --namespace mssql-cluster -lplane=control
```

Вот список объектов pod, полученный для моего развернутого BDC:

```
NAME                READY         STATUS        RESTARTS        AGE
control-vjwjf       3/3           Running       0               38h
controldb-0         2/2           Running       0               38h
```

```
controlwd-l8fmp     1/1     Running     0     38h
gateway-0           2/2     Running     0     38h
logsdb-0            1/1     Running     0     38h
logsui-f42ln        1/1     Running     0     38h
metricsdb-0         1/1     Running     0     38h
metricsdc-gtrxn     1/1     Running     0     38h
metricsui-kwh4q     1/1     Running     0     38h
mgmtproxy-nc8tl     2/2     Running     0     38h
```

Контроллер также называется *уровнем управления*; эта концепция очень похожа на концепцию Kubernetes для уровня управления (https://kubernetes.io/docs/concepts/#kubernetes-control-plane).

На рис. 10.8 показан крупный план компонентов уровня управления для BDC.

Рис. 10.8. Уровень управления BDC

Контроллер можно рассматривать как набор сервисов, которые используются для управления BDC. Одной из задач управления является развертывание, и контроллер используется для развертывания BDC. Как только azdata развертывает контроллер, контроллер «захватывает» и развертывает другие компоненты BDC. Все объекты pod, работающие на уровне управления, используют концепцию k8s ReplicaSet для обеспечения высокой доступности.

Одним из наиболее важных компонентов контроллера является *служба контроллера* (также указанная в качестве службы управления на рис. 10.8). Служба контроллера фактически является сервером API для BDC. Эта служба поддерживает REST API для выполнения всех типов операций для BDC, включая развертывание, управление, виртуализацию данных и многое другое. В практических примерах будет использоваться взаимодействие со службой контроллера с применением нескольких различных методов, включая azdata, внешние таблицы T-SQL и Azure Data Studio (ADS).

На момент написания этой книги отсутствует общедоступная документация по протоколу использования службы контроллера для определенных API. Все API доступны с использованием операторов Azdata, Azure Data Studio (ADS) и T-SQL.

> **Совет.** Azure Data Studio (ADS) может подключаться и взаимодействовать с BDC без azdata. Поэтому примеры REST API для взаимодействия со службой контроллера содержатся в его открытом исходном коде, размещенном по ссылке https://github.com/

microsoft/azuredatastudio. Однако в будущем мы можем их изменить, поэтому я призываю вас обращать особое внимание на этот момент. Кроме того, при использовании данного способа доступа у вас отсутствует возможность установить необходимое программное обеспечение и получить доступ к сертификатам в BDC.

Другие объекты pod, принадлежащие этому уровню управления, реализуют службы, которые поддерживают подключение к различным сервисам (прокси): Kibana и Elasticsearch для ведения журнала, Grafana и InfluxDB для получения значений метрик и мониторинга, а также SQL Server для хранения «метаданных» BDC.

Контейнер SQL Server для хранения метаданных – это обычный экземпляр SQL Server, но он «скрытый» (private). Другими словами, вы никогда не подключаетесь к этому экземпляру. Контейнер контроллера использует этот SQL Server для чтения важных данных, относящихся к управлению и исправности, а также для обеспечения возможности выполнения запросов HDFS.

Мне нравится узнавать, как все работает, поэтому я использовал следующие методы для запуска оболочки Bash внутри этого специального контейнера SQL Server. Объект pod, в котором размещен этот контейнер, называется **controldb-0**.

Я использовал следующую команду для запуска оболочки Bash и подключения к контейнеру SQL Server:

```
kubectl exec -it controldb-0 --namespace=mssql-cluster -- /bin/bash
```

При помощи этой команды я подключаюсь к первому контейнеру внутри объекта pod – это контейнер SQL Server. Как оказалось, мы создаем этот образ SQL Server на основе нашего основного образа SQL Server, на котором установлен sqlcmd.

Мне нужен пароль sa для работы с sqlcmd, однако это не тот пароль учетной записи sa, который используется для подключения к главному экземпляру SQL Server. Это служебный пароль, используемый только контроллером. Я обнаружил, что мы храним объект secret для пароля sa внутри контейнера в /var/run/secrets/credentials/mssql-sa-password/password. Используя эту строку пароля, я подключился к sqlcmd и обнаружил, что в контейнере установлены следующие базы данных: **health_system**, **controller** и **hive_metastore**. Эти базы данных используются внутри BDC. Это пример контейнера SQL Server, используемого для внутренней функциональности BDC, в отличие от основного экземпляра SQL Server, который используется для обычных целей SQL Server, а также виртуализации данных с HDFS и другими источниками данных.

Пул хранения

В главе 9 я рассказал, как Polybase позволяет вам получать доступ к источникам данных вне SQL Server, включая данные HDFS. При доступе Polybase

к HDFS преобразует запросы T-SQL в Java MapReduce Jobs для доступа к данным HDFS.

BDC развертывает свой собственный кластер HDFS, чтобы вы могли получить доступ к данным HDFS как через Polybase, так и напрямую, используя **Knox Gateway** (https://knox.apache.org/), через контроллер.

На рис. 10.9 показано, как кластер HDFS развернут в BDC в качестве *пула хранения* (Storage Pool).

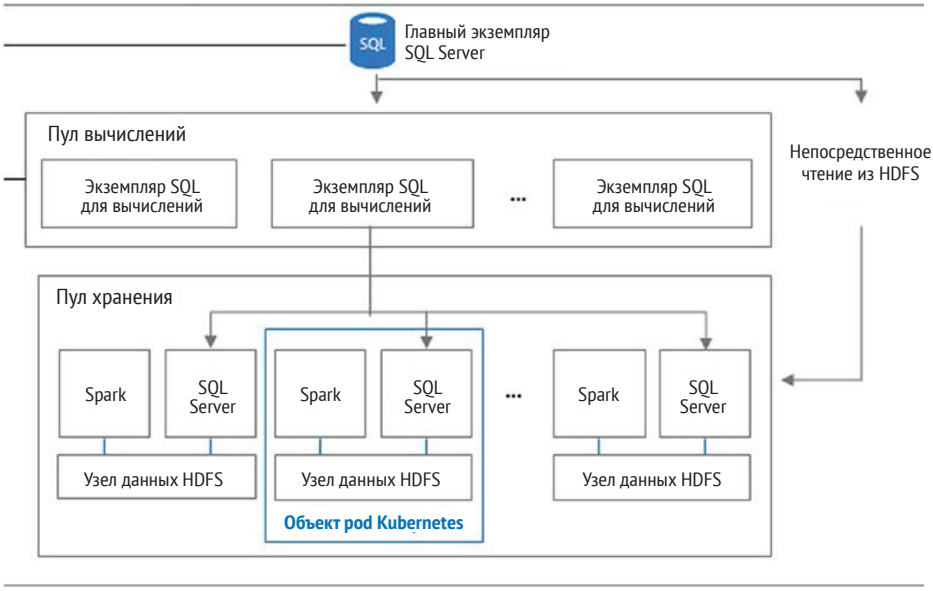


Рис. 10.9. Пул хранения BDC

Пул хранения состоит из одного или нескольких объектов pod k8s. По умолчанию при использовании конфигурации aks-dev-test развертываются два объекта pod пула хранения. Если вы выведете список объектов pod в пуле хранения при помощи команды `kubectl describe`, то увидите, что они связаны меткой `role=storage-pool`. Вы можете масштабировать ваше решение, добавляя объекты pod в пул хранения, используя пользовательскую конфигурацию и указав число реплик.

В моем списке объектов pod, развернутых на BDC, пул хранения представляют следующие объекты pod:

```
storage-0-0        4/4        Running        0        24h
storage-0-1        4/4        Running        0        24h
```

Объекты pod пулов хранения являются частью их собственного множества объектов StatefulSet, поэтому в случае двух реплик в конфигурации BDC вы получаете два объекта pod в одном StatefulSet.

Каждый объект pod в пуле хранения содержит четыре контейнера (предустановлены collectd и fluentbit), включая объект pod для компонен-

тов **Hadoop** и один объект pod для SQL Server. Объект pod для компонентов Hadoop (имя контейнера Hadoop) запускает YARN и HDFS. YARN – это менеджер ресурсов для компонентов Hadoop, запускаемых в кластере, включая Spark Jobs (подробнее о YARN можно прочитать по ссылке https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html). HDFS обеспечивает функциональность распределенной файловой системы Hadoop. BDC также развертывает узел имен HDFS (HDFS Name Node) для хранения метаданных и управления доступом к кластеру HDFS.

Возможности YARN и HDFS включают распределенные вычисления и хранение, что означает, что когда вы взаимодействуете и используете пул хранения через T-SQL и Spark, ваши вычисления и хранение данных являются частью встроенной распределенной системы.

Контейнер SQL Server в системе BDC служит для специальных целей. На рис. 10.9 обратите внимание на коннектор с надписью «Непосредственное чтение из HDFS». Это примечание означает, что контейнер SQL Server в объектах pod пула хранения может считывать данные непосредственно из хранилища HDFS, в том числе из файлов определенного формата, такого как csv и parquet. Вы не подключаетесь к этим контейнерам SQL Server напрямую; они используются внутри BDC для оптимизации доступа к файлам HDFS в кластере BDC. Служба контроллера перенаправляет запросы к внешним таблицам в HDFS в BDC на эти экземпляры SQL Server (эти запросы могут перенаправляться через пул вычислений).

Если у вас есть собственная система HDFS, то можете смонтировать ее в пул хранения, используя концепцию под названием HDFS Tiering. Вы можете прочитать о HDFS Tiering по адресу https://docs.microsoft.com/en-us/sql/big-data-cluster/hdfs-tiering.

Пул вычислений

На рис. 10.9 в предыдущем разделе, посвященном пулу хранения, также показана концепция пула вычислений. Пул вычислений (Compute Pool) – это множество (StatefulSet) объектов pod, которые реализуют группу масштабирования Polybase, о которой рассказывалось в главе 9.

Масштабировать пул вычислений можно, настроив конфигурацию развертывания BDC с помощью счетчика реплик (Replicas count). По умолчанию конфигурация aks-dev-test развертывает только один объект pod пула вычислений (в документации это называется экземпляром).

Если ваша конфигурация BDC включает пул вычислений, все запросы к внешним таблицам в BDC будут использовать пул вычислений. Контроллер перенаправляет все запросы к внешним таблицам в источники данных BDC через пул вычислений.

Для моей конфигурации BDC пул вычислений реализуется указанным ниже объектом pod и использует метку `role=compute-pool`.

```
compute-0-0        3/3        Running        0        43h
```

Пул данных

В пуле данных размещены один или несколько объектов pod для кеширования данных, о котором я говорил в разделе «Что входит в состав кластеров больших данных». По умолчанию конфигурация aks-dev-test для BDC развертывает два объекта pod для пула данных. В моей конфигурации BDC эти объекты pod представлены следующим образом:

```
data-0-0          3/3         Running         0         43h
data-0-1          3/3         Running         0         43h
```

Пул данных состоит из одного или нескольких объектов pod, использующих метку `role=data-pool` в StatefulSet, внутри каждого из которых размещен контейнер SQL Server. Ваш доступ к SQL Server в пуле данных осуществляется через внешние таблицы Polybase из главного экземпляра SQL Server.

При создании внешних таблиц в главном экземпляре SQL Server с использованием внешнего источника данных для пула данных в каждом из объектов pod SQL Server создает отдельную базу данных пула данных, имя которой совпадает с именем области внешних таблиц главного экземпляра SQL Server. Кроме того, мы создаем таблицу с тем же именем, что и имя внешней таблицы.

Это означает, что ваше взаимодействие с пулом данных происходит через внешние таблицы в главном экземпляре SQL Server. На каждом SQL Server в объектах pod пула данных мы создадим базу данных и таблицу, соответствующие вашей внешней таблице. Кроме того, мы автоматически разделяем или сегментируем данные (не используя разделы SQL Server), когда вы добавляете данные в пул данных (по умолчанию используется циклический перебор), и создаем кластеризованный столбцовый индекс для каждой таблицы в каждом объекте pod пула данных, чтобы оптимизировать доступ для чтения. Это означает, что наш пул вычислений может использоваться для доступа к этим данным в масштабируемом режиме через сегменты базы данных. Данные в пуле данных невозможно изменить; вы можете только добавлять (INSERT) данные или выполнять запросы к ним. Поскольку это кеш, это означает, что вы должны удалить внешнюю таблицу и снова заполнить ее данными, когда будете готовы обновить кеш. Контроллер перенаправляет конкретные запросы к внешним таблицам на экземпляры SQL Server в пуле данных (которые могут выполняться в пуле вычислений).

Пул приложений

Пул приложений (Application Pool) – это набор объектов pod, развернутых на основе **приложений** в BDC. На рис. 10.10 показана область BDC для пула приложений.

Рис. 10.10. Пул приложений в BDC

Когда вы используете интерфейсы BDC для создания приложения с помощью файла YAML, служба контроллера будет динамически создавать набор объектов pod ReplicaSet с вашим приложением, работающим в контейнере. В число поддерживаемых типов приложений в настоящее время входят Python, MLeap и SSIS.

Существует еще один объект pod, представляющий прокси-приложения, включая балансировщик нагрузки, который позволяет подключаться к приложению, работающему в пуле, как изнутри BDC, так и из внешней среды через конечную точку службы:

```
appproxy-<id>
```

Вы можете узнать больше об архитектуре развертывания приложений в BDC, перейдя по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/concept-application-deployment.

Использование кластеров больших данных

В этом разделе я рассмотрю различные варианты использования кластеров больших данных (Big Data Clusters, BDC). Первое, что вы захотите сделать, – это **войти в BDC**, используя azdata. Технически вам не нужно входить в систему для доступа к некоторым ресурсам в BDC, но использование azdata для входа дает вам контекст для доступа ко всем конечным точкам службы.

Для входа в BDC необходима конечная точка службы контроллера. Конечная точка службы контроллера – это IP-адрес и порт балансировщика нагрузки (LoadBalancer) для объекта pod службы контроллера. В моем варианте развертывания AKS я смог получить данные о конечной точке службы контроллера с помощью следующей команды:

```
kubectl get svc controller-svc-external -n mssql-cluster
```

Теперь я могу задать для инструмента azdata правильный контекст для использования в различных сценариях, выполнив следующую команду:

```
azdata login --controller-endpoint https://<ip-address-of-controller-svc-
external>:30080 --controller-username admin
```

После этого мне было предложено ввести пароль (это пароль, который я вводил для сценария Python в разделе «Развертывание BDC»). После успешного входа я увидел следующее сообщение:

```
Logged in successfully to `https://<ip-address>:30080`
```

В этом контексте я могу использовать azdata для многих целей. Первое, что я хочу сделать, – это получить список других конечных точек службы для работы с BDC. Для получения списка этих конечных точек я выполню следующую команду:

```
azdata bdc endpoint list -o table
```

Полученный в результате список выглядит следующим образом:

```
       Protocol
------------------------------------------------------ -------------------------
-------------------------- ----------------- -------
Gateway to access HDFS files,  Spark                      https://<knox-ip>:30443
                               gateway            https
Spark Jobs Management and Monitoring Dashboard            https://<knox-ip>:30443/
gateway/default/sparkhistory   spark-history      https
Spark Diagnostics and Monitoring Dashboard                https://<knox-ip>:30443/
gateway/default/yarn           yarn-ui            https
Application Proxy https://<appproxy-ip>:
30778                          app-proxy          https
Management Proxy                                          https://<mgmt-ip>:30777
                               mgmtproxy          https
Log Search Dashboard                                      https://<mgmt-ip:30777/
kibana                         logsui             https
Metrics Dashboard                                         https://<mgmt-ip>:30777/
grafana                        metricsui          https
Cluster Management Service                                https://<cluster-ip>:
30080                          controller         https
SQL Server Master Instance Front-End                      <sql-ip>,31433
                               sql-server-master  tds
HDFS File System Proxy                                    https://<knox-ip>:30443/
gateway/default/webhdfs/v1     webhdfs            https
Proxy for running Spark statements, jobs, applications    https://<knox-ip>:30443/
gateway/default/livy/v1        livy               https
```

Я заменил некоторые имена для представления фактического IP-адреса в моем кластере:

- <**knox-ip**> – это IP-адрес Knox Gateway, который, как вы можете видеть в этом списке, используется для нескольких целей. Knox Gateway применяется для доступа к файлам HDFS (**webhdfs**), запуска Spark Jobs (**livy**), просмотра истории Spark Jobs (**spark-history**) и мониторинга Spark Jobs (**yarn-ui**);
- <**appproxy-ip**> – это IP-адрес, используемый для подключения к приложениям, развернутым в BDC;
- <**sql-ip**> – это IP-адрес для подключения к главному экземпляру SQL Server;
- <**cluster-ip**> – это IP-адрес службы контроллера.

Вы также можете получить все IP-адреса и порты конечной точки, используя kubectl, но только azdata дает вам конкретные данные, такие как доступ к **webhdfs** и **livy**.

Azure Data Studio (ADS) теперь предлагает возможности управления BDC, включая возможность просмотра списка конечных точек.

На рис. 10.11 показан пример списка конечных точек BDC, отображаемого в ADS.

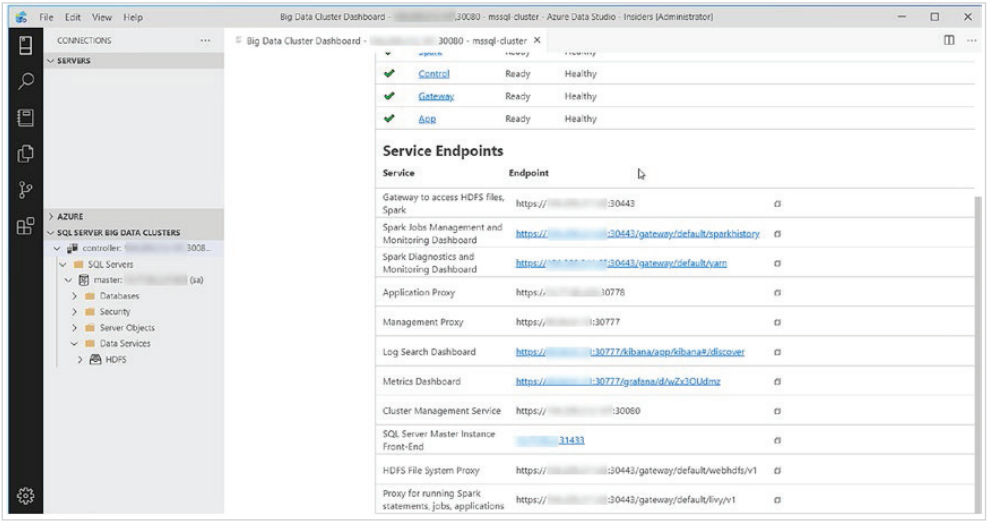


Рис. 10.11. Конечные точки BDC в Azure Data Studio

В документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/concept-security, показаны общие конечные точки BDC. Вы также можете увидеть общие конечные точки BDC на рис. 10.12.

Применение виртуализации данных

Одним из распространенных вариантов применения BDC является доступ к данным из внешних источников данных с использованием Polybase, как упоминалось в главе 9.

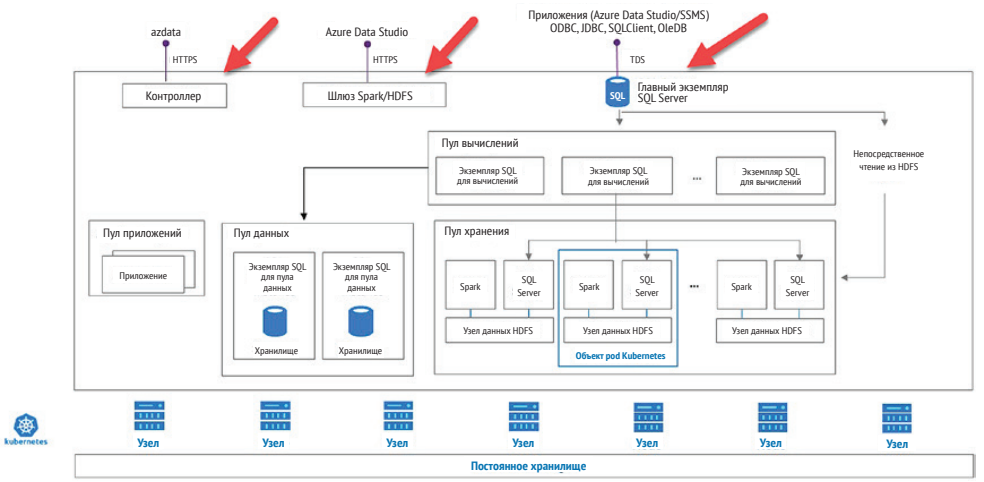


Рис. 10.12. Общие конечные точки BDC

Polybase с BDC обеспечивает те же функциональные возможности, что и Polybase для Linux, включая встроенные коннекторы для SQL Server, Oracle, Teradata, MongoDB и HDFS.

В BDC можно обеспечить поддержку этой функции посредством двух дополнительных встроенных коннекторов, уникальных для BDC:

- **sqlhdfs** – этот коннектор позволяет получить доступ к данным HDFS в пуле хранения;
- **sqldatapool** – этот коннектор позволяет получить доступ к данным, специально хранящимся в пуле данных.

Вот примеры сценариев T-SQL для создания внешних источников данных в вашей базе данных для доступа к этим встроенным коннекторам:

```
IF NOT EXISTS(SELECT * FROM sys.external_data_sources WHERE name =
'SqlDataPool')
    CREATE EXTERNAL DATA SOURCE SqlDataPool
    WITH (LOCATION = 'sqldatapool://controller-svc/default');
IF NOT EXISTS(SELECT * FROM sys.external_data_sources WHERE name =
'SqlStoragePool')
    CREATE EXTERNAL DATA SOURCE SqlStoragePool
    WITH (LOCATION = 'sqlhdfs://controller-svc/default');
```

Обратите внимание, что URI для LOCATION – это конкретное местоположение службы контроллера. Служба контроллера направляет запросы к внешним таблицам на основе этих источников в соответствующий пул через пул вычислений, если он развернут.

В нашей документации есть пример использования внешней таблицы с BDC для доступа к данным Oracle (см. https://docs.microsoft.com/en-us/sql/relational-

databases/polybase/data-virtualization). Вам понадобится установить экземпляр Oracle, чтобы использовать этот пример. Вы также можете использовать примеры, которые я привел в главе 9, размещенные в каталоге **ch9_data_virtualization\sqldatahub**.

> **Примечание.** В этой папке нельзя использовать только два примера: hdfs и saphana. Доступ к данным HDFS в BDC осуществляется через коннектор sqlhdfs. Коннекторы ODBC, которые требуются для работы с SAP HANA, в настоящее время не поддерживаются для BDC.

Я полагаю, вас более заинтересуют примеры, иллюстрирующие доступ к данным через коннекторы **sqlhdfs** и **sqldatapool**.

Я рекомендую сначала загрузить образцы данных для использования BDC, следуя инструкциям на странице документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/tutorial-load-sample-data. Я выполнил действия, описанные в этих инструкциях, и у меня не возникло проблем с загрузкой нужных мне данных. В этом примере вы загрузите каталог csv files в HDFS, используя инструмент curl. В данном примере используется конечная точка **WebHDFS** (https://hadoop.apache.org/docs/r1.0.4/webhdfs.html) объекта Knox Gateway, который называется **прокси файловой системы HDFS (HDFS File System Proxy)**.

После того как вы загрузите данные, можете перейти к обучающим материалам, размещенным по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/tutorial-query-hdfs-storage-pool, чтобы получить доступ к данным HDFS. Вам также может быть интересно попробовать поработать с мастером внешних таблиц, поставляемым с Azure Data Studio. Мастер внешних таблиц обеспечивает альтернативный способ создания внешней таблицы, связанной с данными HDFS в BDC. Обратитесь к документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/data-virtualization-csv.

Возможно, вам потребуется загружать данные непосредственно в HDFS из таких источников, как устройства IOT. В нашей документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/data-ingestion-curl, приводятся примеры того, как напрямую взаимодействовать с HDFS.

Кроме того, Azure Data Studio включает возможность просмотра и работы с файлами в HDFS напрямую, как вы можете видеть на рис. 10.13.

Бак Вуди (Buck Woody) провел семинар под названием «Кластеры больших данных SQL Server – архитектура» (SQL Server Big Data Clusters – Architecture) и создал набор записных книжек (Notebooks), которые можно использовать с ADS, чтобы увидеть, как виртуализация данных работает с BDC. Вы можете просмотреть и загрузить эти записные книжки по ссылке https://github.com/microsoft/sqlworkshops/tree/master/sqlserver2019bigdataclusters/SQL2019BDC/notebooks, используя учебные пособия 00, 01 и 02 для записных книжек, предназначенных для сценариев виртуализации данных.

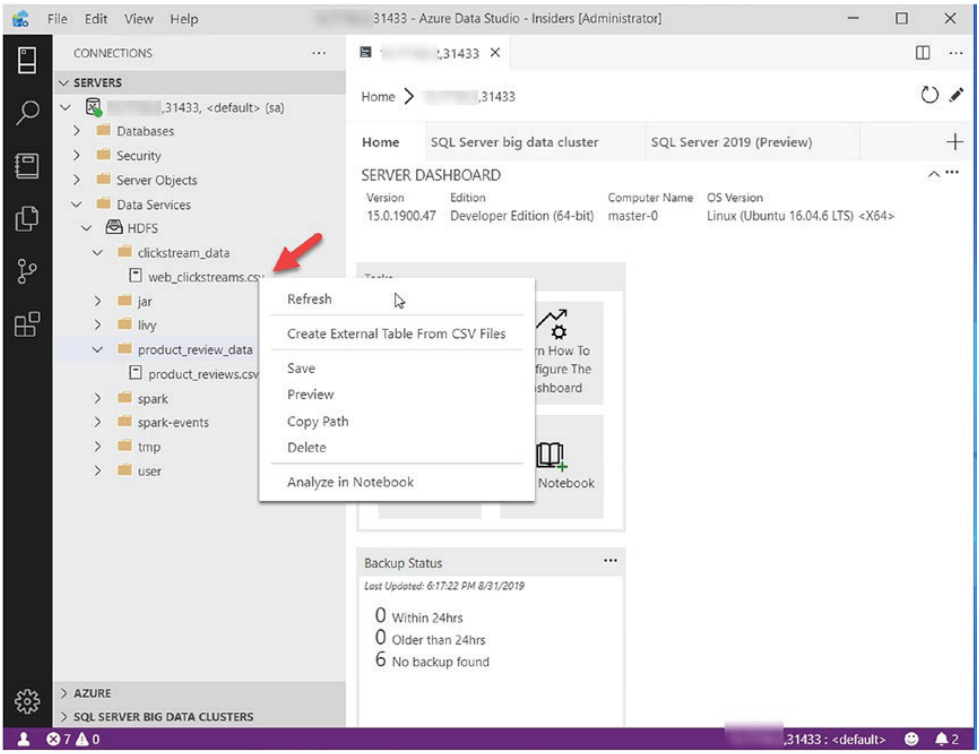


Рис. 10.13. Работа с HDFS в BDC с помощью Azure Data Studio

Использование пула данных

Ранее в этой главе я представил пул данных как кеш данных. Процесс использования пула данных заключается в получении или добавлении данных на основе запросов из других источников данных, которые могут быть таблицами основного экземпляра SQL Server, внешними источниками данных из HDFS или любого другого коннектора Polybase.

Данные автоматически распределяются между объектами pod в пуле данных и оптимизируются для доступа на чтение с кластеризованными столбцовыми индексами.

Я рекомендую вам ознакомиться с примером в нашей документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/tutorial-data-pool-ingest-sql, чтобы получить представление об использовании пула данных.

На семинаре Бака Вуди в занятии 03 показано, как использовать пул данных в BDC (см. https://github.com/microsoft/sqlworkshops/blob/master/sqlserver2019bigdataclusters/SQL2019BDC/notebooks/bdc_tutorial_03.ipynb).

Использование Spark

Spark (https://spark.apache.org/) – это механизм вычислений, который часто применяется в системах Hadoop. BDC автоматически предоставляет возможности для запуска Spark Jobs для различных приложений. Существует несколько способов запуска Spark Jobs в BDC, о которых я расскажу в этом разделе. Вы можете просмотреть некоторые из этих примеров, чтобы увидеть, как Spark работает с BDC. Если у вас нет опыта работы со Spark, вам нужно сначала подумать, почему вы хотите использовать Spark, прежде чем применять Spark Jobs в BDC. Есть несколько очень хороших сценариев, в которых Spark может быть эффективным методом обработки данных в HDFS, поэтому мы включили Spark в состав общего решения BDC. Вы также увидите, что Spark является распространенным решением, используемым в сценариях машинного обучения, о которых я подробнее расскажу в следующем разделе этой главы «Машинное обучение и кластеры больших данных».

Запуск Spark Jobs из Azure Data Studio

Одним из сценариев, в котором Spark может быть полезен, является загрузка данных из HDFS из пула хранения в таблицы и в пул данных в BDC.

Одним из способов запуска задания Spark является использование Azure Data Studio, подключенной напрямую к главному экземпляру SQL Server. На рис. 10.14 показан пример запуска Spark Job из Azure Data Studio (ADS).

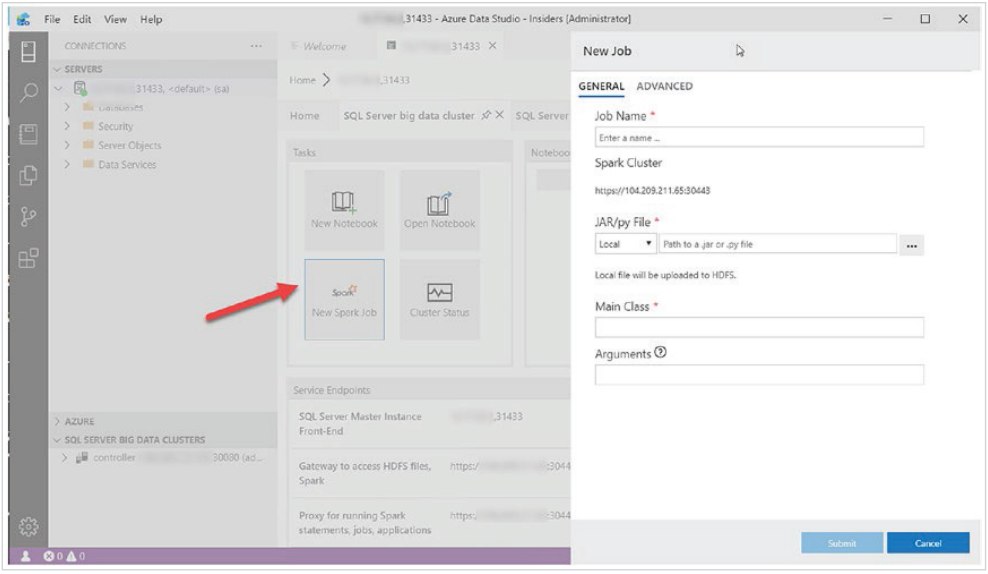


Рис. 10.14. Запуск Spark Job в Azure Data Studio

Дополнительную информацию об отправке Spark Jobs напрямую в ADS можно получить по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/spark-submit-job.

Запуск Spark Jobs из других инструментов

Мы также поддерживаем отправку Spark Jobs в BDC с помощью инструмента **IntelliJ**, о котором вы можете прочитать по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/spark-submit-job-intellij-tool-plugin. Вы также можете отправить Spark Jobs в BDC, используя Visual Studio Code, о котором вы можете прочитать по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/spark-hive-tools-vscode. В обоих этих сценариях вы будете использовать конечную точку **Gateway to access HDFS files, Spark (Шлюз для доступа к файлам HDFS, Spark)** для подключения к BDC и запуска Spark Jobs.

BDC также предоставляет конечную точку REST под названием Livy (https://livy.apache.org/) для отправки Spark Jobs. Конечная точка Livy предоставляется посредством прокси, как часть <knox-ip>, и называется **Proxy for running Spark statements, jobs, applications (Прокси для запуска операторов, исполняемых сценариев (jobs), приложений)**.

Возможно, наиболее распространенный способ использования Spark в контексте BDC – это применение записных книжек (Notebooks) в Azure Data Studio (ADS) . В предыдущих главах данной книги я показывал вам множество примеров использования записных книжек в работе с ADS, использующей *ядро* для SQL. ADS поддерживает ядра и для других языковых сред, в том числе:

- PySpark3;
- PySpark;
- Spark | Scala;
- Spark | R;
- Python.

В любом из этих сценариев вы будете подключаться к главному экземпляру SQL Server с помощью записной книжки. ADS будет обрабатывать отправку Spark Jobs из записной книжки через Knox Gateway, чтобы правильно запустить Spark Job в BDC. Любой код Python или R, который содержится в этих записных книжках, будет работать и на вашем локальном компьютере.

MSSQL Spark Connector

Мы предоставляем и другой метод для запуска Spark Jobs через MSSQL Spark Connector. Этот коннектор взаимодействует с основным экземпляром SQL Server, использует для записи API-интерфейсы массового копирования SQL (SQL Bulk Copy APIs) и имеет знакомый интерфейс JDBC. Вы можете прочитать больше о MSSQL Spark Connector и о том, как использовать его с BDC, по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/spark-mssql-connector.

Развертывание и использование приложений

В разделе «Архитектура кластера больших данных» я описал, как работает пул приложений (Application Pool) в BDC, и дал ссылки на документацию по развертыванию приложений в BDC.

Мы предоставляем «среду выполнения» для приложений, написанных на R и Python, а также приложений, которые поддерживают пакеты MLeap (https://mleap-docs.combust.ml/) и службы SSIS. Разработчик предоставляет код и файл YAML, в которых содержатся инструкции, как запустить приложение, а BDC запустит ReplicaSet контейнеров для выполнения кода приложения.

Развернутые в BDC приложения всегда «работают» как контейнер. Если вы хотите использовать или выполнить код приложения, то можете использовать команду **azdata** с параметром **app**. Вот ссылка на документацию azdata: https://docs.microsoft.com/en-us/sql/big-data-cluster/reference-azdata-app.

BDC также предоставляет другой метод для применения развернутых приложений через веб-интерфейс REST. По умолчанию все развернутые приложения могут использоваться таким образом через протокол **Swagger** (https://swagger.io/).

Лучший способ понять, как все это работает, – это посмотреть примеры, которые вы можете найти, перейдя по ссылке https://github.com/Microsoft/sql-server-samples/tree/master/samples/features/sql-big-data-cluster/app-deploy.

Безопасность

В то время когда я писал эту главу, BDC поддерживал только базовую аутентификацию, то есть учетные записи и пароли. Все конечные точки службы, которые связаны с контроллером, Knox и главным экземпляром SQL Server, требуют ввести имя учетной записи и пароль.

Все взаимодействие между объектами pod внутри кластера происходит по частным каналам связи с использованием объектов secret k8s (которые сами по себе имеют учетные записи и пароли) и самозаверяющих сертификатов.

К тому времени, когда мы выпустим окончательную версию SQL Server 2019, включив в нее все возможности поддержки кластеров больших данных, мы намерены включить поддержку аутентификации Active Directory (AD) в BDC для всех конечных точек служб. Это включает в себя подключение к главному экземпляру SQL Server, службе контроллера (Controller Service) и Knox Gateway.

Я ожидаю, что наша документация, размещенная по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/concept-security, будет содержать подробную информацию о том, как включить BDC в домен, как развернуть BDC, обеспечив все необходимые для AD данные, а также будет включать описание процесса добавления пользователей AD и инструкцию, как войти в BDC с учетной записью AD.

Высокая доступность

Как я уже упоминал в этой главе, объекты pod в BDC развертываются с помощью StatefulSet или ReplicaSet в k8s. Это обеспечивает встроенную высокую доступность (HA) для платформы k8s в случае сбоя контейнера, объекта pod или узла (высокая доступность при сбое узла будет обеспечиваться только при развертывании k8s с несколькими узлами).

Хотя такая поддержка базовой HA полезна для SQL Server, было бы лучше использовать нашу технологию группы доступности Always On (Always On Availability Group, AG), которая включает реплики чтения и мониторинг работоспособности SQL Server.

При развертывании BDC у вас есть возможность включить *hadr*. Включение hadr по умолчанию создаст группу доступности в BDC и включит **системные базы данных** в AG. Для поддержки этого варианта развертывания создается несколько объектов pod в StatefulSet.

Используя эту конфигурацию, мы также создаем конечные точки для подключения к первичной и вторичной репликам AG. Поскольку системные базы данных включены как часть AG, ваше основное подключение – это подключение к основному экземпляру главного экземпляра SQL Server, например прослушивателю AG. Если происходит аварийное переключение, эта конечная точка останется подключенной к той реплике, которая станет основной. Подключения к вторичным репликам (только для чтения) также поддерживаются через отдельную конечную точку.

Документация, в которой рассказывается о том, как включить hadr, и описаны особенности использования этого типа развертывания, доступна по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/deployment-high-availability.

Jupyter Books для кластеров больших данных SQL Server

Jupyter Books (https://jupyter.org/jupyter-book/intro.html) предоставляет механизм для создания коллекции записных книжек. Azure Data Studio (ADS) предоставляет коллекцию записных книжек Jupyter Books, которая помогает отслеживать, управлять и устранять неполадки кластеров больших данных SQL Server. В основу этих записных книжек положены руководства по устранению неполадок (Troubleshooting Guides, TSG), используемые командой разработчиков SQL Server.

На рис. 10.15 показан интерфейс Jupyter Books для кластеров больших данных SQL Server.

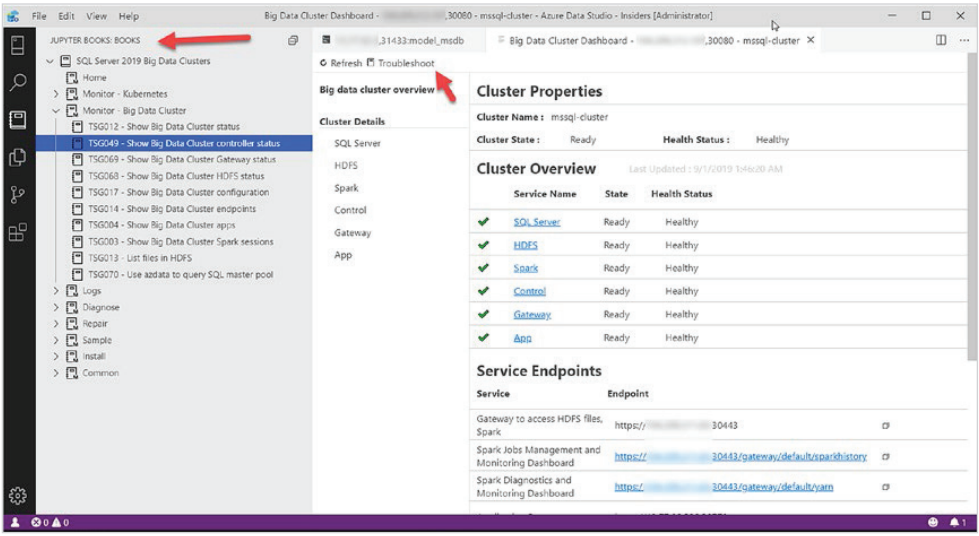


Рис. 10.15. Jupyter Books для кластеров больших данных SQL Server

Машинное обучение и кластеры больших данных

Одним из вариантов применения кластеров больших данных (Big Data Clusters, BDC) SQL Server является комплексная платформа машинного обучения. Рассмотрим упрощенную схему работы машинного обучения на рис. 10.16.

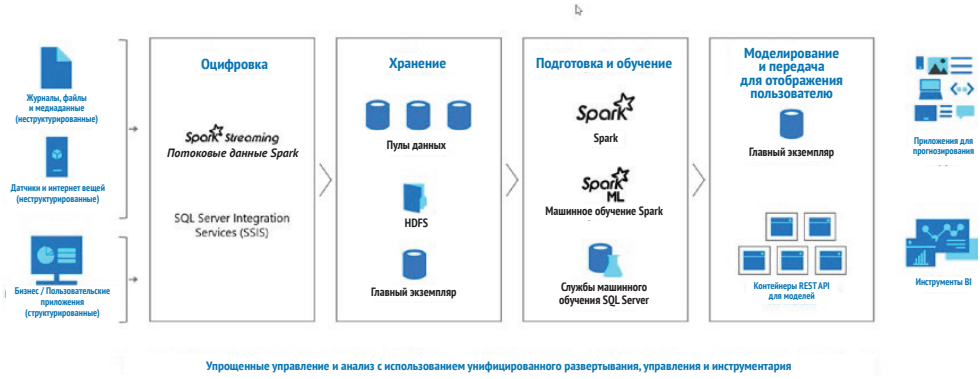


Рис. 10.16. Использование кластеров больших данных в машинном обучении

Вы можете сделать все это, используя BDC! Вы можете получать данные из различных типов источников данных, как структурированных, так и неструктурированных, с помощью Spark и SSIS. Вы можете хранить эти данные в BDC, используя пулы данных, HDFS или даже основной экземпляр SQL Server. Некоторые из ваших данных для моделей машинного обучения могут находиться во внешних источниках данных за пределами BDC, таких как Azure, SQL Server, Oracle, Teradata и MongoDB. BDC предоставляет вам доступ к любым из этих данных с помощью T-SQL.

Теперь вы можете подготовить и обучить свою модель машинного обучения с помощью Spark, SparkML и/или служб машинного обучения SQL Server (SQL Server Machine Learning Services, ML). Затем вы можете разместить свою модель как приложение машинного обучения с использованием SQL Server ML с T-SQL или как приложение, работающее через интерфейс REST, в пуле приложений. Пул приложений предоставляет интересный подход для разработчиков, поскольку в его основу положены код вашего приложения, декларативные файлы YAML и контейнеры. Это означает, что он может использовать модель разработки, основанную на концепции непрерывной интеграции / непрерывной доставки (Continuous Integration / Continuous Delivery, CI/CD).

Пакеты для машинного обучения

Одним из огромных преимуществ для специалистов по исследованию и обработке данных, использующих BDC и SQL Server 2019, являются пакеты для машинного обучения (Machine Learning Packages), входящие в состав поставляемого нами продукта. Я спросил доктора Рони Чаттерджи (Rony Chatterjee), старшего менеджера программы в нашей команде, каким образом могу узнать, какие из пакетов ML установлены. Он показал мне следующий запрос T-SQL, который я мог запустить на SQL Server 2019 или в BDC, чтобы получить список установленных пакетов:

```
EXEC sp_execute_external_script
@language=N'Python',
@script=N'
import pkg_resources
import pandas
OutputDataSet = pandas.DataFrame([(d.project_name, d.version) for d in
pkg_resources.working_set])'
```

Я выполнил этот запрос на своем развернутом BDC и узнал, что в нем было установлено более 160 пакетов машинного обучения!

Использование примеров

Я считаю, что вы должны познакомиться и попробовать выполнить следующие практические примеры, чтобы увидеть возможности машинного обучения и кластеров больших данных SQL Server (BDC):

- **SparkML** – у нас есть пример использования Spark и Spark ML совместно с BDC для прогнозирования уровней доходов на основе данных прошлых переписей в Соединенных Штатах. Вы можете ознакомиться с этим примером по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/spark-create-machine-learning-model;
- **приложения BDC** – существует несколько примеров приложений ML с использованием Application Deploy, которые можно найти по ссылке https://github.com/microsoft/sql-server-samples/tree/master/samples/features/sql-big-data-cluster/app-deploy;
- **пример Бака Вуди** – мы с Баком Вуди (Buck Woody) весной 2019 года проводили курс обучения для клиента, и Бак привел очень интересный пример задачи, решаемой методами машинного обучения (Machine Learning, ML). Идея заключается в том, что выдуманная компания WideWorldImporters имеет грузовики, которые перевозят чувствительные к температуре продукты. На грузовиках установлены системы охлаждения от батарей. Большая проблема заключается в том, что системы охлаждения на грузовиках могут выйти из строя из-за ограниченного жизненного цикла аккумулятора. Аккумуляторы рассчитаны на три месяца, но во многих случаях они выходят из строя раньше. Компания хочет построить прогностическую модель ML, чтобы определить, когда может потребоваться замена батареи, – на основе динамических параметров грузовика и груза вместо фиксированного трехмесячного цикла.

  У Бака есть специальная записная книжка, которую вы можете использовать, чтобы увидеть этот пример, – она опубликована по ссылке https://github.com/microsoft/sqlworkshops/blob/master/sqlserver2019bigdataclusters/SQL2019BDC/notebooks/bdc_tutorial_05.ipynb. Для применения этого обучающего материала вам необходимо выполнить все практические упражнения, опубликованные в записных книжках по ссылке https://github.com/microsoft/sqlworkshops/tree/master/sqlserver2019bigdataclusters/SQL2019BDC/notebooks.

  Когда мы с Баком проводили данный курс обучения, один из представителей заказчика сказал что-то вроде: «Наконец-то кто-то дал мне практический, наглядный пример машинного обучения, и я понял, как я могу использовать кластеры больших данных, чтобы реализовать его».

Управление и мониторинг кластеров больших данных

Вы можете видеть, что кластеры больших данных Big Data Clusters (BDC) SQL Server включают в себя множество компонентов. Для мониторинга и управления BDC необходимо учитывать множество факторов, включая управление кластером Kubernetes, SQL Server и другими компонентами BDC.

Управление Kubernetes (k8s)

Если вы зададите себе вопрос, что же мы создали с помощью BDC, ответ, скорее всего, будет таков: это *приложение, работающее на Kubernetes*. Несмотря на то что в составе решения BDC для SQL Server 2019 имеются определенные средства, которые могут помочь вам управлять приложением BDC, вы все равно должны быть готовы управлять своим кластером k8s. Для разработки и тестирования BDC это, скорее всего, не представляет проблемы, однако для запуска BDC в рабочей среде вы должны спланировать, каким образом будете управлять кластером k8s и контролировать его независимо от BDC. Я вряд ли преувеличиваю, когда говорю, что очень важно понять, как обеспечить правильное управление и функционирование вашего кластера k8s на должном уровне. От того, насколько хорошо вы управляете вашим кластером, зависит работоспособность вашего решения BDC.

Чтобы погрузиться в тему управления k8s, я рекомендую вам следующие ресурсы:

- ознакомьтесь с нашей документацией по управлению Azure Kubernetes Service (AKS), размещенной по ссылке https://docs.microsoft.com/en-us/azure/aks/best-practices;
- настоятельно рекомендую книгу, в которой также содержится отличная информация о внутреннем устройстве k8s: https://learning.oreilly.com/library/view/managing-kubernetes/9781492033905/.

Я также дал несколько советов и показал несколько методов управления и мониторинга кластера k8s в разделе «Советы по k8s» главы 8.

Управление и мониторинг кластеров больших данных

Помимо стандартного управления и мониторинга главного экземпляра SQL Server с помощью средств диагностики SQL Server, такой как представления динамического управления (Dynamic Management Views, DMV) и расширенные события, мы предоставили ряд инструментов и ресурсов, которые помогут вам управлять и контролировать кластер больших данных (Big Data Cluster, BDC) SQL Server.

- **Панель мониторинга кластера больших данных Azure Data Studio (ADS)**

  ADS поставляется с панелью мониторинга BDC, которая поможет вам оценить работоспособность кластера BDC, включая все его компоненты. На рис. 10.17 показан внешний вид панели мониторинга ADS BDC.

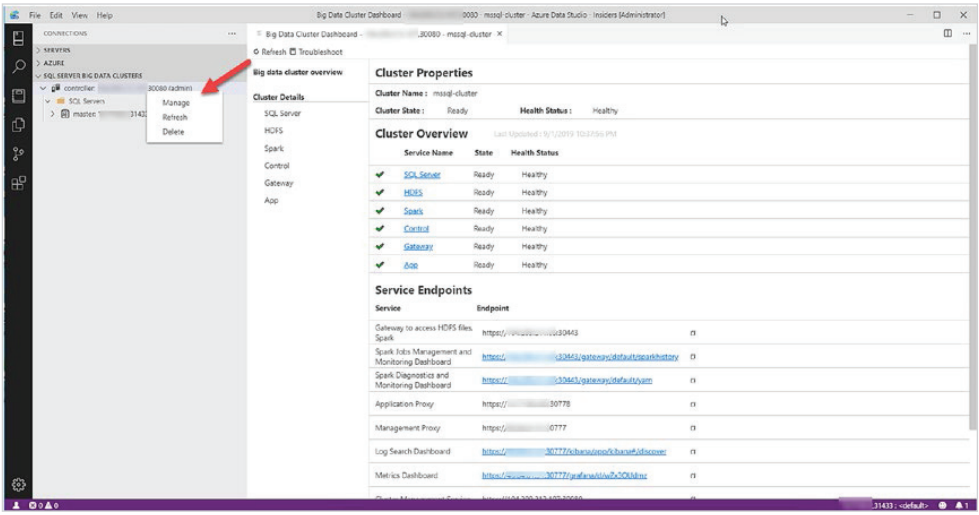


Рис. 10.17. Панель мониторинга кластера больших данных Azure Data Studio

Вы можете щелкнуть один из объектов панели **Cluster Details (Информация о кластере)**, например **SQL Server**, и просмотреть состояние главного экземпляра SQL Server, вычислительных ресурсов, данных и пула хранения. Мы внедрили проверку «жизнеспособности» (https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/) в каждый объект pod в BDC, чтобы отобразить общее состояние работоспособности всех компонентов BDC. Более подробную информацию о панели мониторинга кластера больших данных вы можете найти по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/manage-with-controller-dashboard.

- **Метрики Grafana**

  Используя этот контекст, вы можете перейти к метрикам, отображаемым в панели мониторинга Grafana (https://grafana.com/), управляемой компонентами, размещенными в контроллере. На рис. 10.18 показана панель **Grafana** для главного экземпляра SQL Server.

- **Kibana и Elasticsearch**

  Каждый основной компонент BDC имеет панель мониторинга Grafana и панель визуализации Kibana (https://en.wikipedia.org/wiki/Kibana) и Elasticsearch (www.elastic.co/), а также журналы, используемые для более глубокого анализа и устранения неполадок. На рис. 10.19 показана визуализация в Kibana журнала Elasticsearch для главного экземпляра SQL Server, отображаемая в панели мониторинга кластера больших данных ADS.

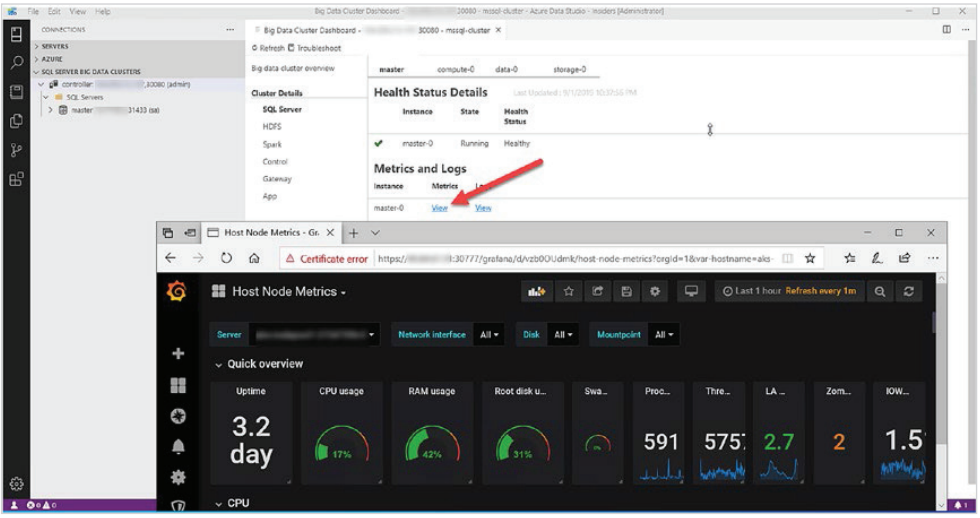


Рис. 10.18. Панель инструментов **Grafana** для кластеров больших данных SQL Server

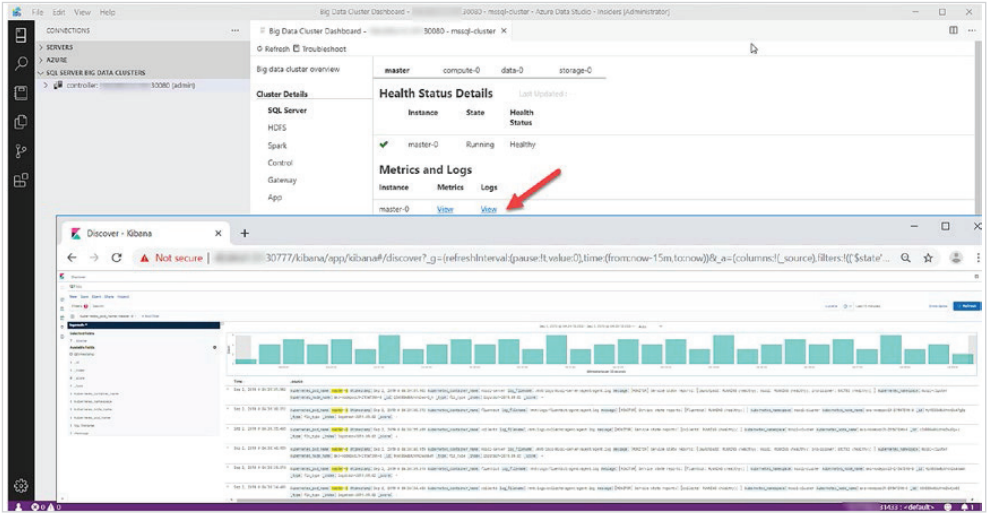


Рис. 10.19. Визуализация данных журналов BDC посредством Kibana и Elasticsearch

- **Использование azdata с SQL Server**

  В то время как Kubernetes позволяет вам взаимодействовать с контейнерами с помощью такой команды, как `kubectl exec`, azdata позволяет вам взаимодействовать с SQL Server, используя параметр sql azdata, как описано в документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/reference-azdata-sql. Это дает вам возможность выполнять команды T-SQL для главного экземпляра

SQL Server, а также получать доступ к «оболочке» sqlcmd. Помните, что инструмент azdata похож на утилиту kubectl для BDC; вы можете ознакомиться с полной документацией по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/reference-azdata.

- **Устранение неполадок Kubernetes (k8s) и BDC**

  Прочитайте мое описание команд k8s в главе 8; однако у нас также имеется еще несколько советов в нашей документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/big-data-cluster/cluster-troubleshooting-commands. Не забудьте также ознакомиться с нашими руководствами по устранению неполадок SQL Server, которые я привел в разделе «Jupyter Books для кластеров больших данных SQL Server» этой главы.

Резюме

В то время как SQL Server 2019 содержит радикальные изменения, кластеры больших данных SQL Server (SQL Server Big Data Clusters) являются революционным решением. Кто бы мог подумать, что продукт, который, по мнению некоторых, представляет собой просто механизм реляционных баз данных, включает в себя полное, целостное решение для создания вашего собственного *озера данных*, *виртуализации данных* и *сквозной платформы машинного обучения* на базе Kubernetes?

Взгляните на перечень технологий, которые мы использовали для кластеров больших данных:

- SQL Server;
- Polybase;
- HDFS;
- Spark;
- Livy;
- Kibana;
- Elasticsearch;
- Grafana;
- InfluxDB;
- записные книжки;
- машинное обучение с поддержкой R и Python;
- расширения Java;
- группы доступности Always On (Always On Availability Groups).

Все это поддерживается платформой управления с нашим «сервером API» или контроллером, функция которого заключается в обеспечении средств для развертывания, управления и обеспечения функционирования кластера больших данных, построенного на Kubernetes.

Это мое мнение, но почему бы просто не привести отзыв клиента, который уже познакомился в общих чертах с этим решением: «Создание и развертывание нашего вертикального AI-решения для клинической радиологии сочетает в себе очень разнообразные парадигмы реализации, форматы данных и нормативные требования. Кластеры больших данных SQL Server 2019 позволили нам приспособить и интегрировать все аспекты в рамках одной общей платформы, предоставляя богатый спектр возможностей для наших исследователей данных с их взглядом вглубь данных, а также для наших инженеров-программистов, которые подключают рабочие процессы, обеспечивают безопасность и масштабируемость. Наши клиенты в области здравоохранения получают выгоду от использования контейнеров благодаря простоте их развертывания и обслуживания, а также они могут легко перемещать наше решение между локальной и облачной средами», – Рене Бальцано (René Balzano), основатель и генеральный директор компании Balzano.

Эта цитата взята из нашего блога, из заметки, которую мы опубликовали, когда выпустили предварительную версию кластеров больших данных для SQL Server 2019 (см. https://cloudblogs.microsoft.com/sqlserver/2019/08/29/sql-server-2019-release-candidate-refresh-with-big-data-clusters/). Я с нетерпением жду, что множество других клиентов остановят свой выбор на этом продукте, убедившись, что кластеры больших данных – это поистине революционное решение, которое необходимо им для развития бизнеса.

Глава 11

Голос клиента и миграция

Я надеюсь, что к тому времени, когда вы дойдете до этой главы, вы сможете оценить большое количество инноваций, появившихся в SQL Server 2019. И если вы прочитали все первые десять глав, у вас, вероятно, возникло чувство «информационной перегрузки». Несколько человек, которые посещали мои презентации, говорили, что чувствуют, как их мозг «плавится». Если вы испытываете похожие ощущения, дойдя до этой главы книги, значит, я выполнил одну из поставленных мною целей. Я хотел, чтобы эта книга была чем-то большим, чем просто обзор SQL Server 2019, потому что любой может почерпнуть необходимые сведения из документации. Я хотел, чтобы моя книга стала полноценным обзором версии SQL Server 2019.

Итак, после того как мы познакомились со всеми этими возможностями для решения современных проблем с данными, осталось ли что-либо, о чем еще можно поговорить? Конечно, да. В заключение я расскажу об «огромной куче функций» (я позаимствовал этот термин у моего коллеги Конора Каннингема (Conor Cunningham)), которые мы добавили в SQL Server 2019 на основе отзывов клиентов. Я также расскажу о методах, инструментах и технологических приемах, которые вы можете использовать при переходе на SQL Server 2019.

Голос клиента

Все, о чем вы прочитали в этой книге, так или иначе зависит от наших клиентов. В этом разделе я покажу вам улучшения для SQL Server 2019, которые были получены непосредственно из отзывов и запросов клиентов, путем эскалации из службы поддержки Microsoft, нашего собственного внутреннего тестирования или взаимодействия инженеров нашей компании непосредственно с клиентами. Если вы никогда не видели канал обратной связи напрямую с командой разработчиков, вы можете найти его по ссылке https://aka.ms/sqlfeedback. В этом разделе я приведу список улучшений, разделив их на три области:

- **производительность.** Повышение производительности ядра СУБД SQL Server, ускоряющее работу для всех или отдельных типов рабочих нагрузок;
- **пользовательский опыт** – это усовершенствования, позволяющие улучшить использование или настройку продукта SQL Server;

- **диагностика** – это усовершенствования, предназначенные для улучшения процесса устранения неполадок или диагностики проблем SQL Server.

Улучшения производительности

Наша команда инженеров всегда стремится повысить производительность ядра базы данных и ищет такие возможности с помощью исследования проблем клиентов и получения отзывов от них, изучения проблем в работе поддержки Microsoft и часто с использованием тестов производительности. Этот опыт и наблюдения были положены в основу следующих изменений в ядре базы данных.

- **Сокращение компиляции для временных таблиц**

  Один из стандартных приемов проектирования с использованием временных таблиц – это создание временной таблицы в одной области и использование ее в другой. Например, вы можете создать временную таблицу в пакете, а затем попытаться использовать временную таблицу в хранимой процедуре, вызываемой пакетом. Это обычно приводит к перекомпиляции хранимой процедуры, которая ссылается на временную таблицу. В SQL Server 2019 по умолчанию мы можем избежать перекомпиляции в данном сценарии. Хотя это улучшение может не сделать рабочую нагрузку значительно быстрее, оно может помочь общему приложению использовать SQL Server, поскольку уменьшение количества перекомпиляций может снизить общую загрузку ЦП SQL Server.

- **Масштабируемость дополнительных контрольных точек**

  Дополнительные контрольные точки – это новый метод по умолчанию для контрольных точек базы данных, о котором можно прочитать по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/logs/database-checkpoints-sql-server. Проведя тестирование производительности и изучив отзывы клиентов, мы обнаружили, что тяжелые модификации данных могут вызывать сбои в механизме SQL Server, что приводит к состоянию, называемому неработающим планировщиком. Обычно мы видели эти проблемы только в больших системах с большим количеством процессоров, что позволяет нам предположить, что это проблема масштабируемости. SQL Server 2019 внес улучшения в ядро базы данных, позволяющие избежать этой проблемы.

- **Параллельные обновления PFS**

  Страницы PFS – это специальные страницы в файле базы данных, которые SQL Server использует для определения свободного места при выделении пространства для объекта. (Вы можете прочитать заметку в блоге Пола Рэндала (Paul Randal), которую он написал, когда

работал в Microsoft, посвященную страницам PFS, по ссылке https://blogs.msdn.microsoft.com/sqlserverstorageengine/2006/07/08/under-the-covers-gam-sgam-and-pfs-pages/.)

Конфликт блокировки страниц на страницах PFS обычно ассоциируется с tempdb, но он также может возникать в пользовательских базах данных, когда существует много параллельных потоков выделения объектов. Это улучшение меняет способ управления параллелизмом с помощью обновлений PFS, так что они могут обновляться в рамках разделяемой кратковременной блокировки страниц, а не эксклюзивной кратковременной блокировки. Это поведение включено по умолчанию во всех базах данных (включая tempdb), начиная с версии SQL Server 2019.

Поскольку клиенты начинают работать с SQL Server 2019, мне очень интересно увидеть эффективность параллелизма TempDB при работе с метаданными TempDB, оптимизированными для выделения памяти (о которых речь шла в главе 2), в сочетании с этим усовершенствованием.

- **Захватывание рабочих потоков**

  Я называю это улучшение «улучшением Славы» в честь Славы Окс (Slava Oks). На протяжении многих лет мы видели, что одним из недостатков планировщика SQLOS является конфликт планировщика для рабочих потоков. Какой совершенной могла бы стать система SQLOS, если бы мы могли динамически менять рабочие потоки для данной задачи, перенаправляя их на другой планировщик в ситуации конкуренции. Неофициально, в плане эксперимента мы начали внедрять такую систему в ограниченном масштабе. SQL Server поддерживает концепцию параллельного повтора операций для восстановления. Донг Цао (Dong Cao) является ведущим разработчиком этой работы и ведет блог о параллельном повторе операций и внутреннем повторе операций на вторичных репликах группы доступности Always On (Always On Availability Group) по ссылке https://docs.microsoft.com/ru-ru/archive/blogs/sql_server_team/sql-server-20162017-availability-group-secondary-replica-redo-model-and-performance. Донг реализовал концепцию захватывания рабочих потоков только для параллельного повтора операций на вторичных репликах. Если наши тесты по-прежнему будут успешными, я жду, когда мы сможем внедрить это изменение в основное планирование в SQLOS для всего ядра для всех типов рабочих нагрузок.

Пользовательский опыт

Мы работали над рядом улучшений, предназначенных для облегчения взаимодействия пользователей с сообщениями об ошибках, и настройкой SQL Server.

- **Подробные уведомления о планируемом усечении данных**

  Знаете ли вы, что является одним из самых частых пожеланий клиентов относительно SQL Server? Это пожелание детализировать следующее сообщение об ошибке:

```
String or binary data would be truncated
```

  Это сообщение об ошибке, под номером 8152, возникает при попытке вставить или обновить данные в столбце, в котором значение, используемое для вставки или обновления, превышает размер целевого столбца. Проблема здесь заключается в отсутствии контекста – в сообщении не указывается имя таблицы, столбца или фрагмента данных, которые будут отсечены.

  В SQL Server 2019 по умолчанию для приложения выдается сообщение об ошибке с номером 2628, которое выглядит следующим образом:

```
String or binary data would be truncated
in table '%.*ls', column '%.*ls'.
Truncated value: '%.*ls'
```

  Об этом улучшении рассказывает Пэм Лахуд (Pam Lahoud) в одном из популярных блогов SQL Tiger Team. Ее заметка, в которой содержатся конкретные примеры, размещена по ссылке https://docs.microsoft.com/ru-ru/archive/blogs/sql_server_team/string-or-binary-data-would-be-truncated-replacing-the-infamous-error-8152.

  В этой заметке в блоге она призывает использовать флаг трассировки 460 в SQL Server 2017, чтобы выводить более подробное сообщение об ошибке.

  SQL Server 2019 также имеет настраиваемый параметр базы данных, позволяющий изменить выдаваемое по умолчанию сообщение об этой ошибке. Об этом вы можете прочитать по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-database-scoped-configuration-transact-sql?#verbose-truncation.

- **Память и параметры, позволяющие установить степень параллелизма во время настройки**

  Два наиболее распространенных параметра конфигурации экземпляров SQL Server, которые нужно изменить после установки, – это *max server memory* (максимальный объем памяти сервера) и *max degree of parallelism* (максимальная степень параллелизма).

  Учитывая, как часто они меняются, мы теперь добавили параметры конфигурации, которые можно использовать во время установки SQL Server в Windows, чтобы задать эти значения.

На рис. 11.1 показан параметр настройки для выбора степени параллелизма.

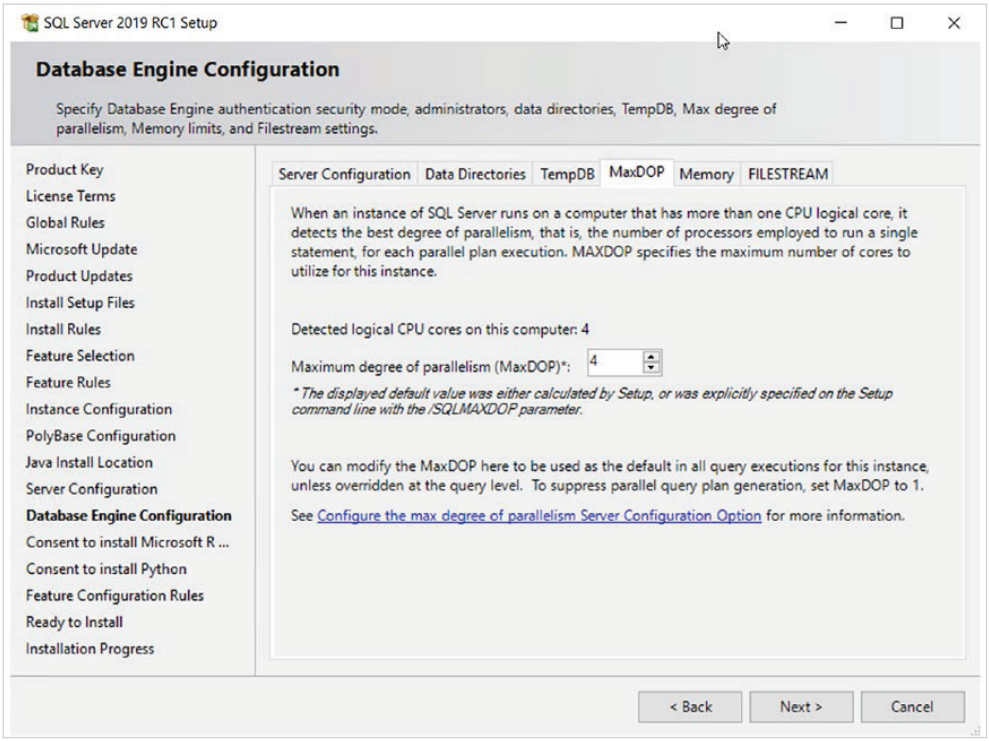


Рис. 11.1. Выбор значения параметра MaxDOP во время установки

На рис. 11.2 показаны параметры конфигурации памяти во время настройки.

Обратите внимание, что на этом рисунке в программе установки SQL Server показано рекомендованное значение **Max Server Memory (MB) (Максимальный объем памяти (Мб))**. Мы не описываем в документации, как получаем конкретное значение этого параметра, и вы можете использовать любое приемлемое значение для вашей среды. Мы обнаружили, что большинство клиентов используют значение, меньшее, чем максимальный объем физической памяти компьютера или виртуальной машины. Я лично считаю, что в рекомендациях программы установки используются алгоритмы, которые вы можете найти в утилитах Tiger Team BPCheck, по ссылке https://github.com/microsoft/tigertoolbox/tree/master/BPCheck.

- **Процент памяти, выделяемой регулятором ресурсов**

  В главе 2 я описал проблему с производительностью, которая может возникнуть из-за выделения больших объемов памяти. Одним из решений, которое может помочь в управлении большими объемами

памяти, является использование регулятора ресурсов с параметром REQUEST_MAX_MEMORY_GRANT_PERCENT. Одна из проблем, связанных с этим параметром, состоит в том, что возможные значения представлены целыми числами от 1 до 100, представляющими процент от максимального значения памяти для SQL Server. Даже 1% от 1 ТБ составляет 10 ГБ, что может быть слишком много для предоставления памяти. SQL Server 2019 теперь допускает указывать для REQUEST_MAX_MEMORY_GRANT_PERCENT значения с плавающей запятой, что означает, что он может принимать значения < 1,0.

Дополнительная информация об использовании REQUEST_MAX_MEMORY_GRANT_PERCENT доступна по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/resource-governor/change-workload-group-settings.

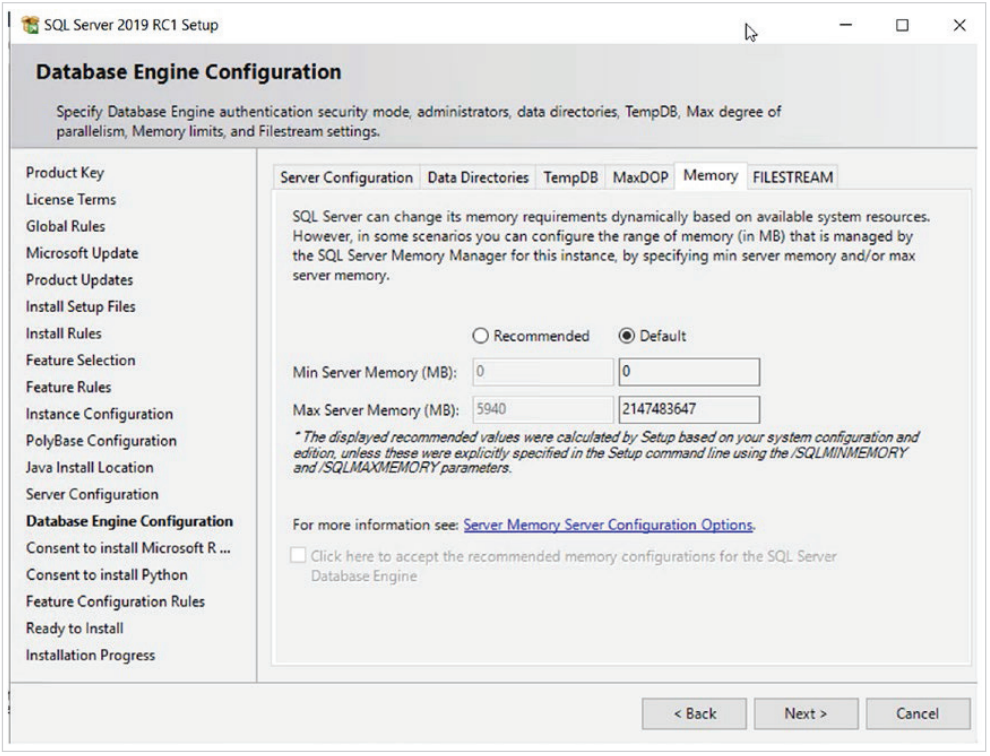


Рис. 11.2. Конфигурация памяти во время настройки

- **Оценка сжатия столбцового индекса**

  В SQL Server имеется системная процедура для оценки экономии при сжатии страниц и строк с названием sp_estimate_data_compression_savings. В SQL Server 2019 эта процедура была улучшена, и теперь она выводит приблизительное сжатие при использовании столбцовых индексов и параметров для архивов столбцовых индексов. Вы мо-

жете узнать больше об использовании этой процедуры, перейдя по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-estimate-data-compression-savings-transact-sql.

Диагностика

Я посвятил диагностике и устранению неисправностей в Microsoft более 20 лет, работая в службе поддержки Microsoft. Я все еще люблю оценивать новые диагностические возможности, добавленные в новые версии этого продукта, даже если они кажутся незначительными. То, что может показаться незначительным одним, может принести огромную пользу другим.

- **Вы тоже можете быть Полом Рэндалом**

  Пол Рэндал (Paul Randal) – мой давний друг. Мы подружились, еще когда он работал в подразделении, разрабатывающем Microsoft SQL Server, и наша дружба продолжалась, когда он занял пост генерального директора SQLskills, где работал вместе со своей женой Кимберли Трипп (Kimberly Tripp). Мы с Полом являемся экспертами в области Microsoft SQL Server – того, как он работает «изнутри», и изучение внутренних элементов страниц базы данных (что иногда называется «взломом страниц») – один из наших важных навыков. Мы оба годами использовали недокументированную и неподдерживаемую команду DBCC PAGE, чтобы «взломать» страницы базы данных. В SQL Server 2019 есть несколько системных объектов, позволяющих изучить заголовки страниц базы данных:

  dm_db_page_info – это системная функция, возвращающая заголовок страницы в виде одной строки, включающей столбцы для каждого поля в заголовке страницы. Дополнительная информация об этой системной функции доступна по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-page-info-transact-sql.

  Входными данными для данной функции является информация, необходимая для идентификации страницы: идентификатор базы данных, идентификатор файла и номер страницы. В некоторых сценариях идентификатор страницы отображается в результирующем наборе данных (столбцов) аналогично тому, как это возвращает DMV. Одним из примеров является **wait_resource** для **dm_exec_requests**. Идентификатор страницы в wait_resource представлен в виде *ресурсной строки страницы*, которая имеет вид <dbid>:<fileid>:<pageid>. Системная функция **fn_PageResCracker** принимает в качестве входных данных ресурсную строку страницы и возвращает в качестве результата db_id, file_id и page_id.

  Таким образом, вы можете выполнить следующий запрос T-SQL:

```
SELECT page_info.*
FROM sys.dm_exec_requests AS d
CROSS APPLY sys.fn_PageResCracker(d.page_resource) AS r
CROSS APPLY sys.dm_db_page_info(r.db_id, r.file_id, r.page_
id,'DETAILED')
AS page_info;
```

Это позволит получить данные заголовка страницы из ресурса страницы. Для сценария параллелизма, такого как ожидание кратковременной блокировки страницы, этот метод может быть полезен, чтобы выяснить, какой объект принадлежит заблокированной странице.

Если вы вернетесь к главе 2, то увидите, что я включил в нее пример использования этого способа проверки, чтобы выяснить, какая таблица была затронута ожиданием кратковременной блокировки страниц tempdb:

```
USE tempdb
GO
SELECT object_name(page_info.object_id), page_info.*
FROM sys.dm_exec_requests AS d
CROSS APPLY sys.fn_PageResCracker(d.page_resource) AS r
CROSS APPLY sys.dm_db_page_info(r.db_id, r.file_id, r.page_
id,'DETAILED')
AS page_info
GO
```

Пэм Лахуд (Pam Lahoud) также написала очень хорошую заметку в блоге об этом небольшом, но очень важном улучшении механизма базы данных, о котором вы можете прочитать по ссылке https://docs.microsoft.com/ru-ru/archive/blogs/sql_server_team/sql-server-2019-ctp-2-0-new-features-introducing-the-page-cracker-aka-sys-dm_db_page_info.

- **Диагностика с использованием данных статистики**

  Данные статистики являются очень важной частью диагностики производительности запросов. Данные статистики можно обновлять с использованием синхронного или асинхронного метода, где синхронный метод означает, что запрос должен ждать обновления статистики, а асинхронный метод означает, что запрос может продолжаться, но статистика будет обновляться в фоновом режиме. Синхронные обновления данных статистики в некоторых случаях могут приводить к тому, что выполнение команды SELECT займет больше времени, чем обычно. SQL Server 2019 предоставляет средство диагностики, дающее детальную информацию об ожидании синхронных обновлений статистики:

WAIT_ON_SYNC_STATISTICS_REFRESH – это новый тип ожидания (wait_type), найденный в dm_os_wait_stats. Он показывает суммированное нарастающим итогом время на уровне экземпляра, затраченное на синхронные операции обновления статистики.

dm_exec_requests – в столбце `command` sys.dm_exec_requests будет показано значение `SELECT (STATMAN)`, если запрос ожидает завершения операции синхронного обновления статистики до продолжения выполнения запроса.

- **Улучшения хранилища запросов (Query Store)**

  Query Store – важная возможность для настройки производительности, тестирования производительности и устранения неполадок. В главе 2 я показал несколько примеров использования хранилища запросов для сравнения различий в производительности запросов с интеллектуальной обработкой запросов.

  Возможности Query Store были расширены, и теперь они включают:

  принудительные планы выполнения запросов для «быстрых» курсоров (fast-forward cursors) и статических курсоров (static cursors) – Query Store уже поддерживает принудительные планы, которые используются для «быстрых» и статических курсоров;

  пользовательская политика отслеживания запросов для Query Store. У некоторых клиентов возникали проблемы с использованием Query Store для определенных типов рабочих нагрузок. В SQL Server 2019 мы добавили больше параметров, чтобы расширить контроль над тем, что происходит в хранилище запросов. Вы можете найти описание этих новых возможностей, объясняемых на примере QUERY_CAPTURE_POLICY, в документации по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-set-options. Лучшее объяснение, как использовать данные параметры, представлено сообществом SQL Server. Я не знаю никого, кто был бы более погружен в тему Query Store, чем Эрин Стеллато (Erin Stellato), и у нее есть отличный блог, где рассказывается об этих новых возможностях: https://www.sqlskills.com/blogs/erin/query-store-in-sql-server-2019-ctp-3-0/.

- **Гранулярный контроль кеша плана**

  В некоторых случаях вы хотите вручную удалить запрос или процедуру из кеша плана выполнения запроса. Один из возможных способов сделать это – принудительная перекомпиляция запроса. Вы можете использовать параметр CLEAR PROCEDURE_CACHE оператора ALTER DATABASE SCOPED CONFIGURATION, чтобы очистить кеш плана для всех запросов и объектов, связанных с базой данных. Начиная с SQL Server 2019 вы можете очистить кеш плана, применяя этот оператор

на основе дескриптора *plan_handle*. Вы можете найти plan_handle, используя DMV, например dm_exec_query_stats.

- **Улучшения DBCC CLONEDATABASE**

  DBCC CLONEDATABASE можно использовать для устранения неполадок, чтобы исследовать схему и собрать статистику из базы данных, исключая пользовательские данные, в новую базу данных. Это может позволить вам изучить примерный план выполнения запросов для базы данных без необходимости копировать все фактические данные.

  SQL Server 2019 расширяет возможности DBCC CLONEDATABASE, собирая статистику для столбцовых индексов, о которой вы можете прочитать подробнее по ссылке https://docs.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-clonedatabase-transact-sql.

Совет. Поскольку DBCC CLONEDATA BASE фиксирует все данные в системных таблицах (то есть метаданные), эта информация включает в себя данные для Query Store. Это означает, что вы можете просматривать данные производительности Query Store в автономном режиме из пользовательской базы данных main.

А как насчет бизнес-аналитики?

Продукт SQL Server поставляется с возможностями Business Intelligence (BI), включая службы аналитики SQL Server (SQL Server Analysis Services, SSAS) и службы отчетов SQL Server (SQL Server Reporting Services, SSRS).

И SSRS, и SSAS являются частью лицензии SQL Server для различных версий SQL Server (SSAS является только частью Enterprise и Standard).

Для SSRS в SQL Server 2019 нет новых функций и улучшений. Важно помнить, что сервер отчетов Power BI, который обновляется отдельно от версии SQL Server, является частью лицензии Enterprise Edition для SQL Server. Подробнее о сервере отчетов Power BI можно узнать по ссылке https://docs.microsoft.com/en-us/power-bi/report-server/get-started#licensing-power-bi-report-server.

SSAS предлагает новые возможности в SQL Server 2019; вы можете прочитать об этих улучшениях по ссылке https://docs.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-ver15?view=sql-server-ver15#analysis-services.

Переход на SQL Server 2019

Когда вы принимаете решение о переходе на SQL Server 2019 для одного, нескольких или всех ваших экземпляров SQL Server, всегда полезно иметь план миграции. В этом разделе я расскажу об инструментах и ресурсах, которые могут быть вам полезны при принятии решений и составлении плана перехода на SQL Server 2019.

Шоу Пэм и Педро

Это не новый сериал, несмотря на название. Он представляет методологию, которую мои коллеги Пэм Лахуд (Pam Lahoud) и Педро Лопес (Pedro Lopes) пропагандируют среди клиентов и в сообществе. Их обучающие выступления посвящены тому, как планировать успешный переход на новые версии SQL Server. Пэм и Педро называют этот процесс «правильной модернизацией». Вместо того чтобы пытаться повторить все то, что Пэм и Педро использовали в качестве примеров, я предлагаю вам просмотреть подготовленные ими видеоматериалы. Вы можете просмотреть одночасовую презентацию Пэм и Педро на эту тему в своем собственном темпе на YouTube: www.youtube.com/watch?v=5RPkuQHcxxs.

Я также включил в эту книгу слайд из своих презентаций о процессе миграции и инструментах, которые вы, возможно, тоже захотите использовать. Этот слайд приведен на рис. 11.3.

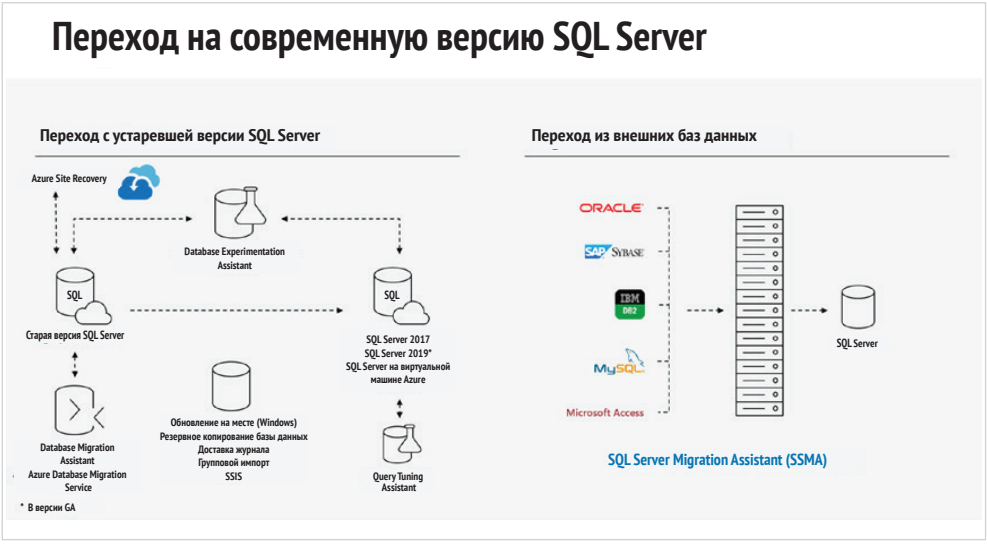


Рис. 11.3. Процесс и инструменты миграции для SQL Server

Обучающие материалы, подготовленные Пэм и Педро, включают обсуждение инструментов миграции. В оставшейся части главы я буду рассматривать этот процесс и инструменты, представленные на рис. 11.3.

Database Migration Assistant

Database Migration Assistant (DMA) – это бесплатный инструмент, который вы можете загрузить и запустить на компьютере Windows, чтобы выполнить оценку текущей конфигурации и кода для существующего экземпляра SQL Server и увидеть любые возможные проблемы, с которыми вы можете столкнуться при переходе на более новые версии. Педро дает пошаговое руководство по этому инструменту в видеоматериале, на

который я ссылался ранее, но позвольте мне также углубиться в некоторые подробности, рассказывая об этом инструменте. Я также расскажу о том, почему, по моему мнению, вам следует его использовать.

DMA – это гораздо больше, чем его предшественник, Database Upgrade Advisor. Ниже перечислены основные возможности DMA.

- Оцените возможные блокирующие факторы для миграции вашего SQL Server (DMA также поддерживает Amazon Relational Database Service (RDS) в качестве источника). Сюда включены параметры, позволяющие увидеть, какие варианты предоставит вам в процессе миграции использование различных параметров уровня совместимости базы данных. Блокирующие факторы могут включать критические изменения, изменения поведения и устаревшие функции. Более подробно об этом я расскажу в разделе «Совместимость баз данных».
- Откройте для себя новые функции в новой версии SQL Server, позволяющие выполнить сравнение новой версии и вашей текущей конфигурации. DMA – достаточно интеллектуальный инструмент, чтобы распознавать функции, которые могут предоставить вам несомненные преимущества. Некоторые примеры рекомендаций по новым функциям включают столбцовые индексы, использование Always Encrypted, прозрачное шифрование данных (Transparent Data Encryption, TDE) и динамическое маскирование данных (Dynamic Data Masking).
- Несмотря на то что DMA не относится конкретно к SQL Server 2019, он также будет оценивать проблемы миграции для пакетов служб SSIS при переходе на базу данных SQL Azure.
- DMA тоже будет поддерживать миграцию вашей базы данных на различные целевые версии, такие как SQL Server 2019. Я рекомендую использовать эту возможность только для миграций очень малого объема или для тестирования миграции вашей базы данных.

Несмотря на то что DMA – это инструмент, созданный для работы в Windows, вы можете использовать этот инструмент для множества различных источников и целей, включая SQL Server для Windows и Linux, Azure SQL Database, Azure SQL Database Managed Instance и SQL Server на виртуальных машинах Azure (Azure Virtual Machine).

На рис. 11.4 показаны возможные целевые конфигурации для оценки и миграции с использованием DMA 4.4.

Есть и несколько других приятных возможностей DMA, о которых вы должны знать:

- DMA является графическим инструментом в Windows, но также имеет интерфейс командной строки (command-line interface, CLI), поэтому вы можете использовать его в сценариях автоматизации.

Вы можете прочитать больше об интерфейсе командной строки, поддерживаемом DMA, по ссылке https://docs.microsoft.com/en-us/sql/dma/dma-commandline;

- если вы хотите выполнить оценку для большого количества экземпляров SQL Server, наша группа по миграции предоставила метод для сохранения информации в базе данных с целью составления отчетов, о котором вы можете прочитать, перейдя по ссылке https://docs.microsoft.com/en-us/sql/dma/dma-consolidatereports;
- используя это хранилище данных оценки, вы можете создавать отчеты Power BI на основе содержащихся в нем данных. Группа миграции также предоставила репозиторий GitHub для отчетов Power BI по ссылке https://docs.microsoft.com/en-us/sql/dma/dma-consolidatereports.

У Педро Лопеса есть демонстрация использования DMA в видеоссылке на YouTube, которую я приводил ранее в этой главе в разделе «Шоу Пэм и Педро».

У нас в Microsoft работает отличная команда инженеров, занимающихся вопросами миграции баз данных, и вы, без сомнения, оцените по достоинству некоторые из их публикаций в блоге на эту тему: https://techcommunity.microsoft.com/t5/microsoft-data-migration/bg-p/MicrosoftDataMigration.

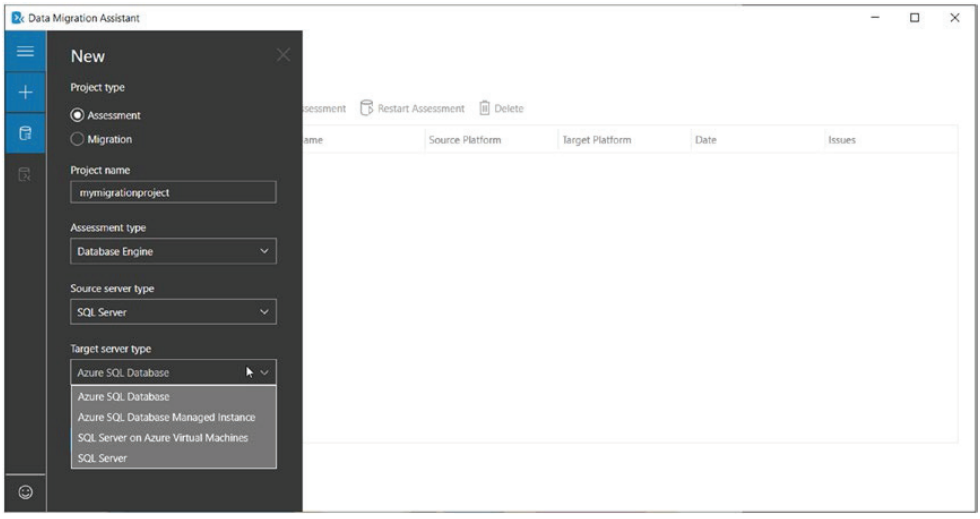


Рис. 11.4. Целевые конфигурации, поддерживаемые DMA

Database Experimentation Assistant

Оценка миграции вашей базы данных и развертывания SQL Server основана на конфигурации вашего SQL Server и баз данных (включая код T-SQL, такой как хранимые процедуры). Однако эта оценка довольно статична и не сможет учесть возможных проблем при миграции вашего SQL Server.

Поэтому одним из важных аспектов миграции является производительность. Выполнение как можно большего объема тестирования производительности приложения SQL Server является одним из наиболее важных аспектов успешной миграции. Database Experimentation Assistant (DEA) может быть очень мощным инструментом для достижения этой цели. Цель состоит в том, чтобы использовать DEA для определения, какие запросы в вашем приложении будут работать лучше, какие – хуже, а производительность каких не изменится для целевой версии SQL Server. Кроме того, DEA может предупредить вас о том, какие запросы могут дать сбой – например, из-за проблемы совместимости.

Документацию по DEA можно найти по ссылке https://docs.microsoft.com/en-us/sql/dea/database-experimentation-assistant-overview; на этой же странице предусмотрена возможность загрузки DEA. Данная утилита предоставляется бесплатно и работает в Windows, но может использоваться и для оценки миграции на SQL Server в Linux.

Для правильного использования DEA может потребоваться до четырех экземпляров SQL Server:

- исходный SQL Server для отслеживания и захвата вашей рабочей нагрузки;
- два целевых сервера SQL для воспроизведения потоков рабочей нагрузки;
- один SQL Server для хранения анализа и запуска отчетов (вам нужна база данных для хранения результатов, чтобы она могла фактически существовать в одном из целевых экземпляров SQL Server).

Основной сценарий использования DEA выглядит следующим образом.

1. Создайте резервную копию базы данных на исходном сервере SQL.
2. Отследите вашу рабочую нагрузку с помощью инструментария DEA, который может использовать SQL Server Trace или Extended Events. Трассировка SQL Server требуется, если вы оцениваете рабочую нагрузку для версии SQL Server, более ранней, чем SQL Server 2012, поскольку в Extended Events отсутствуют необходимые события в SQL Server 2008. DEA поддерживает SQL Server 2005, но для этой версии вам необходимо использовать SQLTrace (вы должны применять SQLTrace для любой версии SQL Server до 2012 года).

   Чтобы наилучшим образом использовать инструмент DEA, необходимо снять профиль вашей рабочей нагрузки, характерный для приложения. Инструмент DEA позволяет исследовать вашу рабочую нагрузку в диапазоне от 5 минут до 3 часов. Вы можете работать с тестовым сервером, на котором можете выполнить трассировку для вашего приложения, или вам может потребоваться запустить его на производственном SQL Server.

3. Подготовьте воспроизводимые трассировки, восстановив резервную копию, созданную на шаге 1 для двух целевых экземпляров SQL Server:

   целевой сервер № 1 – версия SQL Server, которая совпадает с исходным кодом из трассировки, выполненной на шаге 2. Как правило, вы не хотите использовать для этого производственный SQL Server;

   целевой сервер № 2 – новая версия SQL Server, на которую вы переходите, которая может быть SQL Server в Linux.

   Вы должны настроить эти экземпляры SQL Server в средах, максимально приближенных к тем, с которыми вы будете работать, с точки зрения процессора, памяти, скорости диска и конфигурации SQL Server.
4. Используйте инструмент DEA, чтобы воспроизвести трассировку, выполненную на шаге 2, на обоих целевых серверах SQL. Инструмент DEA запросит место для сохранения следа воспроизведения.
5. Используйте инструмент DEA для анализа двух воспроизводимых трассировок, чтобы сравнить производительность или возможные ошибки при выполнении запросов в трассировке. Инструмент DEA выдаст вам запрос о том, какую базу данных SQL Server следует использовать для хранения результатов анализа и местоположения воспроизведенных трассировок, выполненных на шаге 4.

Совет. В более ранних версиях DEA требовалось использовать функцию в SQL Server, которая называется Distributed Replay. Вы все еще можете применять этот метод, но начиная с DEA версии 2.6 вы можете использовать метод воспроизведения **InBuilt**. Метод воспроизведения InBuilt вызывает утилиту ostress.exe (я познакомил вас с этой утилитой в главе 2). Это все еще популярный инструмент в команде и сообществе SQL Server, разработанный моими друзьями из Microsoft, Китом Элмором (Keith Elmore) и Робертом Дорром (Robert Dorr).

Я настоятельно рекомендую вам попробовать поработать с DEA, прежде чем приступать к какой-либо серьезной миграции SQL Server. У Педро Лопеса (Pedro Lopes) есть отличная демонстрация отчетов DEA в видеоролике на YouTube, ссылку на который я давал ранее в этой главе в разделе «Шоу Пэм и Педро».

Вот несколько советов и замечаний по использованию инструмента DEA:

- DEA поставляется с инструментами отчетности для просмотра анализа воспроизведений сохраненного профиля нагрузки. Дополнительную информацию об отчетах DEA вы можете найти по ссылке https://docs.microsoft.com/en-us/sql/dea/database-experimentation-assistant-view-report;

- DEA также имеет интерфейс командной строки (command-line interface, CLI), о котором вы можете прочитать, перейдя по ссылке https://docs.microsoft.com/en-us/sql/dea/database-experimentation-assistant-run-command-prompt.

Обновление до SQL Server 2019

После того как вы выполните все необходимые оценки и будете готовы приступить к обновлению, вам будет необходимо выбрать один из следующих вариантов.

Обновление на месте

Это процесс прямого обновления SQL Server путем запуска программы установки на том же компьютере, где установлен SQL Server. Хотя этот процесс полностью поддерживается Microsoft для экземпляров SQL Server, установленных в корпоративных производственных средах, я не рекомендую использовать его, за исключением случаев непрерывных обновлений, о которых я расскажу позже в этой главе. Это не официальная позиция Microsoft, а моя собственная рекомендация, основанная на многолетнем опыте работы с клиентами в службе поддержки Microsoft. Если вы все же выберете этот вариант, ОБЯЗАТЕЛЬНО убедитесь в том, что у вас созданы и сохранены полные резервные копии баз данных SQL Server предыдущей версии SQL Server, прежде чем выполнять обновление. И более того, я настоятельно рекомендую вам создать полный образ сервера или виртуальной машины, чтобы вы могли быстро выполнить откат к моментальному снимку вашего производственного сервера, если это потребуется.

> **Примечание.** SQL Server для Linux также поддерживает обновление на месте путем переключения хранилища на новую версию SQL Server и запуска обновления менеджера пакетов (например, yum update mssql-server).

Существует вариант обновления на месте, называемый *последовательным обновлением*. Этот метод можно использовать с экземпляром отказоустойчивого кластера Always On (Always On Failover Cluster Instance) или с группами доступности Always On (Always On Availability Groups). Я буду обсуждать данный вариант в разделе «Динамическая миграция».

Не забывайте о возможном сценарии обновления с использованием контейнеров, о чем рассказывалось в главе 7, в разделе «Новый способ обновления SQL Server».

Восстановление базы данных

Это, пожалуй, самый распространенный способ обновления SQL Server (за исключением непрерывного обновления). Механизм SQL Server пони-

мает, как обновить базу данных при ее восстановлении (технически SQL Server знает, как обновить базу данных при ее подключении к сети, – так работает обновление на месте). Поэтому многие клиенты предпочитают устанавливать SQL Server на другой компьютер или виртуальную машину для миграции и затем восстанавливать свои базы данных из более старой версии SQL Server.

Невозможно восстановить обновленную базу данных из новой версии SQL Server, перейдя обратно на более старую версию SQL Server. Например, вы не можете восстановить резервную копию с SQL Server 2019 на SQL Server 2017 (но, как я показал в главе 7, вы можете переключаться между экземплярами с установленными накопительными обновлениями, оставаясь в рамках одной основной версии).

В SQL Server 2019 вы сможете восстановить базы данных из SQL Server 2008 или более поздней версии. Возможно также выполнить восстановление базы данных из SQL Server для Windows в SQL Server 2019 в Linux, поскольку базы данных полностью совместимы на всех платформах. Если у вас есть резервная копия базы данных для версии SQL Server, более ранней, чем SQL Server 2008, вам необходимо выполнить процесс «перехода», восстановив резервную копию в поддерживаемой версии SQL Server (2008), а затем выполнить еще одно резервное копирование, потом произвести восстановление базы данных в SQL Server 2019.

Одним из ключевых факторов, помогающих уменьшить проблемы с миграцией, является то, что при восстановлении базы данных в новой версии SQL Server **уровень совместимости базы данных сохраняется** и остается таким, каким он был установлен для старой версии SQL Server. Более подробно об этой концепции я расскажу в следующем разделе главы «Совместимость баз данных».

SQL Server Integration Services (SSIS) или групповой импорт/экспорт

Другой способ переноса вашей базы данных в SQL Server 2019 – это экспорт и импорт данных с использованием служб интеграции SQL Server (SQL Server Integration Services, SSIS) или других инструментов, позволяющих выполнять операции массового экспорта/импорта.

Я видел, как некоторые клиенты используют этот метод, когда им нужно выполнить преобразования или структурные изменения в общем дизайне или схеме базы данных. Другими словами, вместо того чтобы просто перемещать базу данных «как есть», некоторые клиенты *выполняют перенос приложения* с переходом на новую версию SQL Server. Этот перенос включает в себя изменения в схеме базы данных. Другой метод преобразования, который я видел, заключается в восстановлении базы данных из предыдущей версии и последующем запуске компилирующего кода в новой версии SQL Server для изменения некоторых таблиц или хранимых процедур.

Динамическая миграция

При восстановлении базы данных приложения на некоторое время становятся недоступными для пользователей (это время входит во время простоя системы). Что, если ваши бизнес-требования допускают очень малое время простоя? В этом случае вы можете использовать некоторые из приведенных ниже методов.

- **Последовательное обновление**

  Это один из популярных вариантов, если вы используете экземпляр отказоустойчивого кластера Always On (Always On Failover Cluster Instance) или группы доступности Always On (Always On Availability Groups). Оба этих варианта реализованы в SQL Server 2019, и, в частности, благодаря возможности выполнять последовательное обновление клиенты используют эти решения высокой доступности.

  Always On Failover Cluster Instance поддерживает непрерывное обновление, при условии что версия SQL Server в текущем кластере поддерживает сценарий обновления на месте. Подробнее о развертывании обновлений для экземпляра отказоустойчивого кластера Always On (Always On Failover Cluster Instance) можно прочитать по ссылке https://docs.microsoft.com/en-us/sql/sql-server/failover-clusters/windows/upgrade-a-sql-server-failover-cluster-instance.

  Группы доступности Always On (Always On Availability Groups) поддерживаются в более ранних версиях SQL Server, начиная с SQL Server 2012 и выше, поскольку SQL Server 2012 – это первая версия, в которой эта технология была доступна. Данный вариант последовательного обновления имеет много разнообразных параметров настройки и, вероятно, является оптимальным для сценария оперативной миграции SQL Server в масштабах предприятия. Вы можете прочитать об этом по ссылке https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/upgrading-always-on-availability-group-replica-instances.

  В SQL Server 2017 также включил новую функцию для групп доступности, которая называется *бескластерными* группами доступности (Clusterless Availability Groups). Группе доступности без кластеров не требуется базовое программное обеспечение отказоустойчивой кластеризации. Эта технология также предоставляет возможность выполнить динамическую миграцию, и такой способ может быть одним из лучших вариантов перехода с SQL Server 2017 на SQL Server в Linux. Вы можете прочитать больше об этом варианте по ссылке https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-availability-group-cross-platform.

- **Доставка журналов**

  Более простой метод для динамической миграции – использовать доставку журналов (Log Shipping).

Доставка журналов – это простая технология, которая использует резервные копии базы данных и журнала транзакций SQL Server и восстанавливает их для синхронизации данных со вторичным SQL Server. Вы можете прочитать о доставке журналов по ссылке https://docs.microsoft.com/en-us/sql/database-engine/log-shipping/about-log-shipping-sql-server. Это способ ручного обновления, для которого не требуются технологии постоянной доступности (Always On).

Информация о процессе обновления с помощью доставки журналов находится по ссылке https://docs.microsoft.com/en-ca/sql/database-engine/log-shipping/upgrading-log-shipping-to-sql-server-2016-transact-sql.

- **Репликация**

  Наша документация, размещенная по ссылке https://docs.microsoft.com/en-us/sql/database-engine/install-windows/upgrade-replicated-databases, содержит подробную информацию о том, как обновить полную топологию репликации SQL Server.

  Одна интересная идея (меня на нее вдохновила Амит Банерджи (Amit Banerjee) во время нашей совместной работы) – использовать репликацию SQL Server для оперативной миграции, даже если ваша стратегия развертывания не предполагает использовать репликацию.

  Поскольку подписчик в топологии репликации может быть более новой версией SQL Server, нежели распространитель и издатель, вы можете использовать сервер-подписчик в качестве нового первичного сервера. Другими словами, вы могли бы назначить подписчиком SQL Server 2019 для более старой версии SQL Server в качестве распространителя и издателя. Поскольку ваши приложения и пользователи используют старый сервер-издатель, ваш подписчик SQL Server 2019 будет иметь все последние данные. Когда вы будете готовы к переключению, отключите репликацию и перенаправьте всех пользователей, подключив их к новой базе данных SQL Server 2019 (которая была подписчиком). Это выглядит просто, но такой способ подразумевает некоторое время простоя для настройки и отключения репликации, и это время может оказаться критическим для приложения высокой доступности. Отключение репликации не удаляет данные в базе данных подписчика; это описано в документации, размещенной по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/replication/disable-publishing-and-distribution.

Совместимость баз данных

Совместимость баз данных – это функция, обеспечивающая обратную совместимость при переходе на более новую версию SQL Server. Следующий ресурс содержит терминологию и полную информацию о совместимости баз данных: https://aka.ms/dbcompat.

Прежде всего вам необходимо разобраться в том, какие уровни совместимости базы данных соответствуют уровню, установленному по умолчанию для определенной версии SQL Server. Вы можете просмотреть этот список на странице https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-compatibility-level?view=sql-server-ver15#syntax. Например, уровень совместимости базы данных SQL Server 2012 по умолчанию равен 110.

> **Примечание.** Ранее меня спрашивали, почему уровни dbcompat не совпадают с названием или цифрой версией SQL Server. Это связано с тем, что SQL Server имеет внутренний номер версии, а уровни dbcompat обозначаются в соответствии с этим номером версии. SQL Server 2012 на самом деле является версией 11.x (см. @@VERSION), поэтому уровень dbcompat для этой версии равен 110. Я понимаю, что это сбивает с толку, но мы называем релизы немного иначе, чем версии нашего продукта.

Если вы восстановите резервную копию базы данных, созданную на SQL Server 2012, на SQL Server 2019, SQL Server сохранит уровень совместимости базы данных, равный 110. Цель подобной схемы – гарантировать, что ваше приложение, запросы и взаимодействие сервера с базой данных будут работать точно так же, как они работали на SQL Server 2012. Разница лишь в том, что теперь вы можете воспользоваться новыми функциями в новой версии SQL Server (существует исключение, заключающееся в том, что мы иногда используем совместимость баз данных для включения новых функций).

В одном из предыдущих разделов этой книги, «Database Migration Assistant (DMA)», я упоминал, что инструмент DMA будет определять возможные проблемы миграции: критические изменения, изменения поведения и устаревшие функции. Позвольте мне определить все эти термины в отношении обеспечения обратной совместимости баз данных.

Критические изменения определяются как *изменения поведения*, которые могут привести к получению другого результата. В некоторых случаях от критического изменения можно защититься, установив соответствующий уровень совместимости базы данных в новой версии SQL Server. Однако в других случаях уровень совместимости баз данных не поможет защититься от критического изменения. Вы в замешательстве? Я понимаю ваши чувства. К счастью, в последних версиях SQL Server очень мало критических изменений, которые не защищены. И еще одна хорошая новость заключается в том, что инструмент DMA предназначен для выявления подобных проблем.

Чтобы ознакомиться со списком критических изменений в каждой из версий SQL Server, откройте страницу документации, размещенную по ссылке https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-compatibility-level?#using-compatibility-level-for-backward-compatibility, и переместитесь

по этой странице вниз, до раздела, начинающегося с предложения «Критические изменения, внесенные в данную версию SQL Server…». В этом разделе содержатся ссылки на критические изменения для всех версий SQL Server, включая предыдущие, вплоть до SQL Server 2012.

Существуют некоторые изменения поведения, используемые с определенным уровнем совместимости баз данных и предназначенные для исправления проблемы или добавления новых функций. Одним из примеров этого является возможность интеллектуальной обработки запросов, о которой я говорил в главе 2 и которая сопряжена с уровнем совместимости баз данных 140 или 150.

Устаревшая функциональность защищена соответствующим уровнем совместимости базы данных. Устаревшая функциональность включает функции или поведение, которые мы не собираемся улучшать в будущем. Microsoft может удалить эту функцию в любом новом выпуске. Устаревшая функциональность означает, что мы удалили функцию или поведение из выпуска SQL Server, и потому совместимость базы данных не будет работать. Функциональность, поддержка которой будет прекращена, вначале помечается как устаревшая. К счастью, в последних выпусках мы приняли решение не прекращать поддерживать никаких старых функций. Однако я лично не стал бы полагаться на устаревшую функциональность, особенно в новых проектах. Документация по SQL Server для поиска устаревших функций была переработана и упростилась с выходом последних версий. Воспользуйтесь следующей ссылкой на документацию, чтобы найти устаревшие функции и функции, поддержка которых прекращена в последних выпусках: https://docs.microsoft.com/en-us/sql/database-engine/discontinued-database-engine-functionality-in-sql-server. Чтобы просмотреть документацию, относящуюся к более ранним версиям, вам придется перейти к более старой версии документации. Например, вот ссылка для версии SQL Server 2014: https://docs.microsoft.com/en-us/sql/database-engine/sql-server-database-engine-backward-compatibility?view=sql-server-2014.

Учитывая все вышесказанное, давайте попробуем ответить на вопрос: насколько удобна такая возможность, как уровень совместимости баз данных, для обеспечения обратной совместимости при переходе на новую версию SQL Server? Это достаточно удобно; и Педро Лопес (Pedro Lopes) надеется, что вы будете использовать эту возможность. Он надеется убедить многих независимых поставщиков программного обеспечения (ISV) принять такой же подход. Педро считает, что разработчики приложений должны сертифицировать свои приложения, гарантируя их работоспособность, для определенного уровня совместимости базы данных, а не для определенной версии SQL Server. Если вы внимательно слушали выступление Педро, записанное в видеоролике, ссылку на который я привел в разделе «Шоу Пэм и Педро», то вы помните, что он говорит: «Microsoft поддержит совместимость баз данных» в качестве модели сертификации, обеспечив следующее:

- полную функциональную защиту, если инструмент DMA не обнаружит никаких ошибок.

  Это означает, что для вашего приложения будут отсутствовать критические изменения, если вы выберете соответствующий уровень совместимости баз данных, для которого не будет выявлено никаких потенциальных ошибок;
- защиту структуры плана выполнения запроса на аналогичном оборудовании.

  Это означает, что структура плана выполнения запроса (т. е. операторы и их последовательность) не должна изменяться на оборудовании, аналогичном тому, где вы выполняли запрос в предыдущей версии, если используется тот же уровень совместимости базы данных в новой версии SQL Server.

На сегодняшний день опыт показывает, что эти смелые обещания мы выполняем для своих клиентов.

Вот краткое резюме того, чего вы можете ожидать от обратной совместимости при переходе на SQL Server 2019.

- Если вы используете функцию, поддержка которой прекращается, ваше приложение может перестать работать после миграции.
- Если вы используете устаревшую функцию, все будет в порядке. Но я бы рекомендовал в будущем отказаться от использования этой устаревшей функциональности.
- Начните с уровня совместимости базы данных (dbcompat), который соответствует версии SQL Server, с которой вы выполняли миграцию. Если вы работаете с SQL Server 2012, установите значение dbcompat, равное 110, в качестве отправной точки.

Примечание. Самое большое изменение уровня совместимости базы данных (dbcompat), которое может повлиять на производительность запросов, соответствует значению dbcompat, равному 120. Это связано с тем, что мы внесли в оптимизатор запросов изменение, названное моделью оценки мощности (CE). Если вы перешли на более новую версию SQL Server и вам нужно использовать dbcompat 120 или выше, но возникают проблемы, связанные с моделью CE, то можете отключить режим CE, используя параметр LEGACY_CARDINALITY_ESTIMATION оператора ALTER DATA BASE SCOPED CONFIGURATION.

- Сохраните этот уровень dbcompat для экземпляра SQL Server 2019, работающего в промышленной среде, или выполните тестовую миграцию на SQL Server 2019 и проведите дальнейшее тестирование с новыми уровнями dbcompat, чтобы обнаружить возможные проблемы с вашим приложением. В следующем разделе я расскажу об ин-

струменте под названием **Query Tuning Assistant**, который может вам в этом помочь. Вы можете использовать инструмент DMA, чтобы увидеть оценку конфигурации различных уровней dbcompat для SQL Server 2019 для вашей базы данных.

Несколько других комментариев о совместимости базы данных (dbcompat) и обратной совместимости:

- вы можете увидеть полный список различий в поведении для различных уровней dbcompat, перейдя по этой ссылке на документацию: https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-compatibility-level?view=sql-server-ver15#differences-between-compatibility-level-140-and-level-150. Перемещайтесь по странице вниз, чтобы увидеть различия, вплоть до dbcompat = 90 (SQL Server 2005);
- совместимость базы данных не затрагивает критические изменения на уровне экземпляра SQL Server. Поэтому наша цель состоит в том, чтобы использовать dbcompat в качестве механизма для сертификации приложения. Если приложение использует какую-либо функциональность уровня экземпляра SQL Server, вам необходимо убедиться, что она хорошо протестирована. Функциональные возможности уровня экземпляра могут быть любыми: использование представлений системного каталога, системные динамические административные представления (Dynamic Management Views, DMV), SQL Agent, связанные серверы или операторы T-SQL, которые влияют на весь экземпляр SQL Server и не относятся к области базы данных.

Query Tuning Assistant и Post Migration

В разное время в 2018 году я продолжал получать электронные письма от наших команд от Педро Лопес (Pedro Lopes) о проекте, над которым он работал, под названием TUNA (так я продолжал называть этот проект). На самом деле, как я узнал, проект назывался TunA. TunA расшифровывается как Tuning Assistant.

Если вы вспомните содержание предыдущего раздела, в котором шла речь о рекомендациях по совместимости баз данных, я упомяну идею тестирования вашего приложения на новом уровне совместимости баз данных в рамках нашей миграции.

Давайте вернемся к видео из раздела «Шоу Пэм и Педро» этой главы. На временной отметке приблизительно 40:40 в видео вы увидите, как Педро расскажет о постмиграции (Post Migration).

Здесь речь идет о том, что есть шаги, которые вы можете предпринять для оптимизации рабочей нагрузки после перехода на более новую версию SQL Server, при условии что вы сохранили уровень совместимости исходной базы данных, совпадающий с предыдущей версией SQL Server.

Педро предлагает методологию использования Query Store для сравнения производительности рабочей нагрузки до и после изменения уровня совместимости базы данных. Этот метод взят из документации по хранилищу запросов и применяется в сценарии использования, который называется «**Сохранение стабильности производительности при обновлении до более нового SQL Server**» (Keep performance stability during the upgrade to newer SQL Server), о котором можно прочитать по ссылке https://docs.microsoft.com/en-us/sql/relational-databases/performance/query-store-usage-scenarios.

Теперь вернемся к проекту TunA. Педро и его команда встроили функциональность Query Tuning Assistant (QTA) в инструмент SQL Server Management Studio (SSMS) версии 18.x. QTA позволяет автоматизировать процесс использования хранилища запросов для оценки и устранения проблем с производительностью запросов при переходе на новый уровень совместимости баз данных.

На рис. 11.5 показано, как работает QTA.

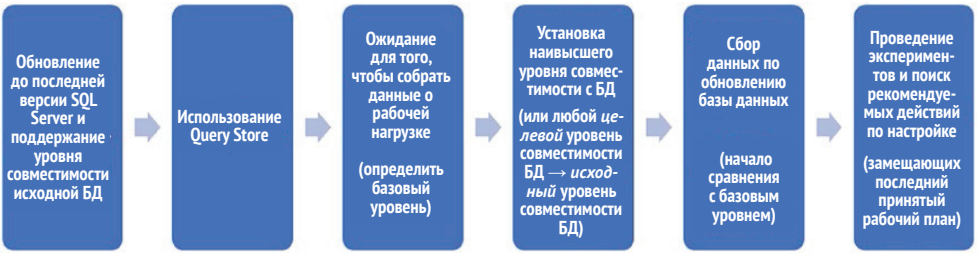


Рис. 11.5. Как работает QTA

Использование QTA совместно с SSMS дает вам подсказки, когда производительность запроса может ухудшиться при новом уровне совместимости базы данных после перехода на более новую версию SQL Server. Выдаваемые рекомендации могут включать изменение запроса – например, использование определенных параметров запроса. Все подсказки основаны на рекомендациях, включенных в документацию. Это «Педро Лопес в коробке».

В этой главе я собирался привести пример для QTA, однако решил, что вы можете просто использовать пример Педро Лопеса, опубликованный на GitHub. Вы можете скачать и использовать этот пример по ссылке https://github.com/microsoft/tigertoolbox/blob/master/Sessions/Winter-Ready-2019/Lab-QTA.md.

Запуск в виртуальной машине Azure

Когда вы принимаете решение о развертывании или миграции на SQL Server 2019, то можете рассмотреть возможность использования SQL Server 2019 в облаке. Одним из вариантов запуска SQL Server 2019 в облаке является виртуальная машина Azure, известная как среда Infrastructure as a Service (IAAS).

Виртуальная машина Azure (VM) позволяет сосредоточиться на развертывании операционной системы (Windows или Linux) и SQL Server и не беспокоиться об аппаратной платформе и инфраструктуре.

Виртуальная машина Azure известна как один из наиболее простых способов перехода на облако для SQL Server, поскольку взаимодействие с SQL Server аналогично виртуальной машине в вашем центре обработки данных или вашей среде.

На самом деле это не на 100% верно, потому что вы должны выбрать некоторые параметры, специфичные для Azure, в том числе размер компьютеров и хранилища, сеть и безопасность. Руководство, в котором рассказывается обо всех этих вариантах, доступно по ссылке https://docs.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/sql-server-on-azure-vm-iaas-what-is-overview.

Мы также предоставляем автоматизацию с использованием виртуальной машины Azure, о которой вы можете не знать. Этот вариант включает в себя автоматическое резервное копирование для SQL Server и автоматическую установку пакетов обновления для операционной системы и SQL Server. Вы можете прочитать об автоматическом резервном копировании, перейдя по ссылке https://docs.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/automated-backup.

Летом 2019 года мы объявили о трех важных новых возможностях в Azure, которые полезны для управления SQL Server на виртуальной машине Azure:

- новый портал Azure, называемый Azure SQL, который упрощает создание и управление ресурсами, связанными с SQL Server, в облаке. На рис. 11.6 показан новый интерфейс портала Azure SQL;

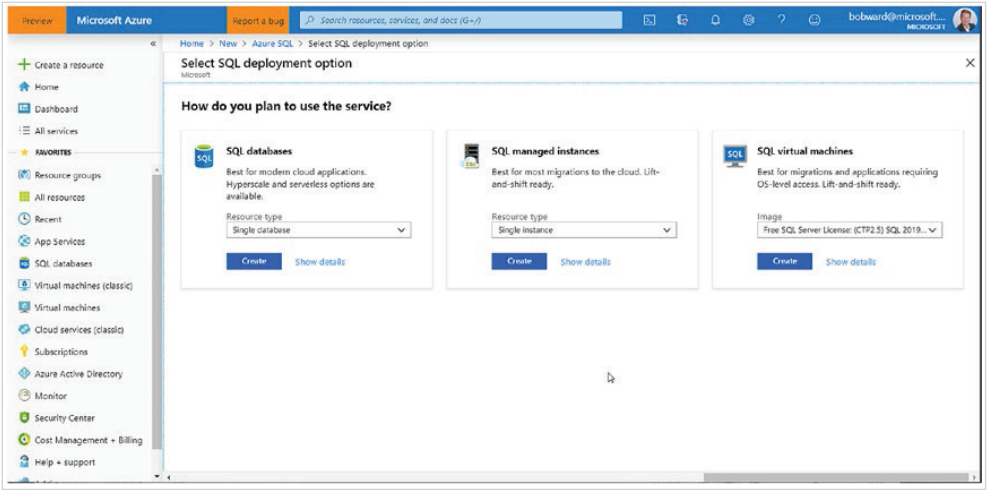


Рис. 11.6. Новый интерфейс портала Azure SQL

- способ регистрации виртуальной машины Azure (Azure Virtual Machine, VM) с SQL Server у нашего поставщика ресурсов. Эта регистрация применяется к сценариям, в которых вы устанавливаете SQL Server на виртуальной машине Azure вместо использования образов галереи. Он разблокирует ваш SQL Server в виртуальной машине Azure, чтобы воспользоваться преимуществами Azure Hybrid Benefits, совместимостью, автоматическим резервным копированием и автоматической установкой обновлений. Вы можете прочитать больше об этой возможности, перейдя по ссылке https://docs.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/sql-vm-resource-provider-register;
- заявление Azure об уровне общей доступности Ultra Disk Storage или Ultra SSD. Ultra SSD обеспечивает высокую скорость работы и чрезвычайно высокую производительность в облаке, что соответствует требованиям, предъявляемым интенсивными нагрузками ввода-вывода SQL Server. Вы можете прочитать об Ultra SSD по ссылке https://azure.microsoft.com/en-us/blog/announcing-the-general-availability-of-azure-ultra-disk-storage/. Если вы хотите посмотреть, как Ultra Disk Storage работает с SQL Server, ознакомьтесь с записью следующего доклада на Microsoft Ignite, сделанного в 2018 году: https://azure.microsoft.com/en-us/resources/videos/ignite-2018-running-high-performance-workloads-in-azure-with-ultra-ssds-the-next-gen-azure-managed-disk/. Если вы переместитесь примерно на 46 минут в этой записи, то можете узнать докладчика, демонстрирующего производительность SQL Server с Ultra Disk Storage.

SQL Server Migration Assistant

Некоторые из читателей этой книги могут не использовать SQL Server или работать с другими системами и иными продуктами баз данных, такими как Oracle, mySQL и DB2. В главе 9 я рассказал, как использовать Polybase для доступа к этим источникам данных. Но что, если вы хотите перенести базу данных и ваше рабочее приложение из этих систем в SQL Server?

У нас есть бесплатный инструмент, который поможет вам перейти на SQL Server (и Azure) из других, «сторонних» решений для баз данных. Этот инструмент называется **SQL Server Migration Assistant**. На правой части рис. 11.3 представлены возможности данного инструмента. Подробнее об этом инструменте, о том, как его развернуть и как успешно использовать для миграции, вы можете прочитать по ссылке https://docs.microsoft.com/en-us/sql/ssma/sql-server-migration-assistant.

На рис. 11.7 показано использование SSMS для перехода с Oracle на SQL Server.

SSMA представляет собой набор инструментов, по одному для каждого типа поддерживаемого стороннего продукта, с которого вы можете выполнить переход. Подобно DMA и DEA, SSMS также имеет интерфейс команд-

ной строки (command-line interface, CLI). Вы можете прочитать больше о синтаксисе SSMS для Oracle по ссылке https://docs.microsoft.com/en-us/sql/ssma/oracle/command-line-options-in-ssma-console-oracletosql.

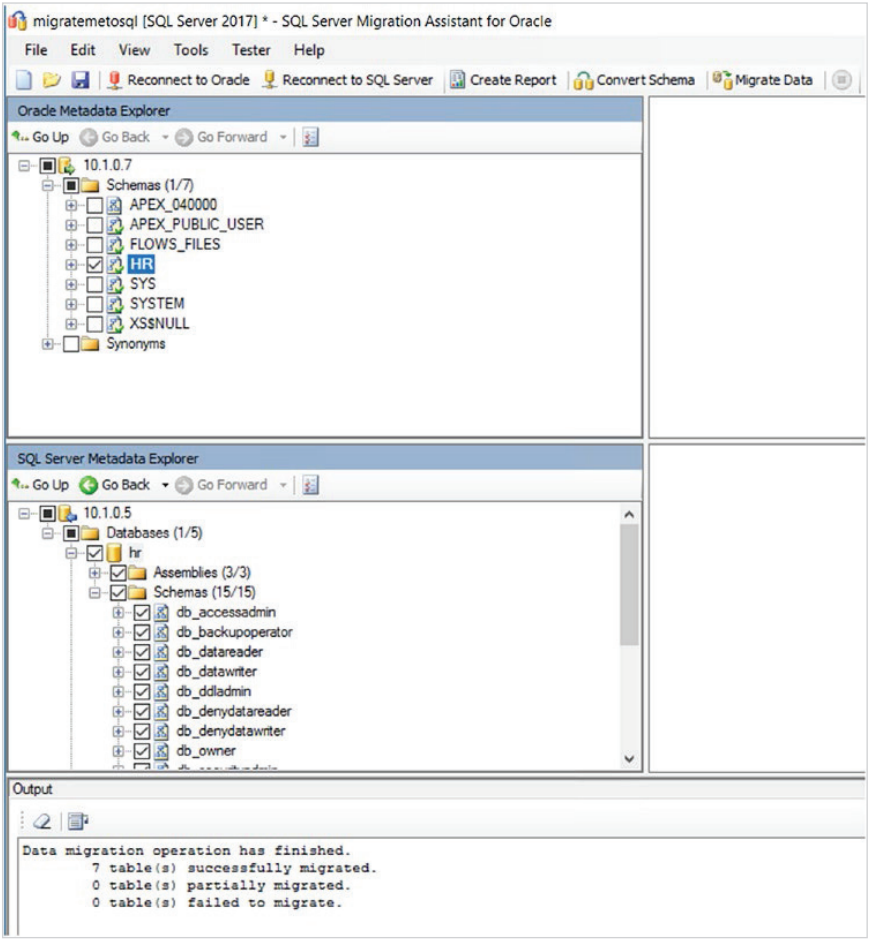


Рис. 11.7. Использование SSMS для перехода с Oracle на SQL Server

Резюме

В этой главе я продемонстрировал многочисленные улучшения SQL Server 2019, в основу которых легли отзывы клиентов и наш опыт работы. Каждое из них может показаться незначительным улучшением, но в совокупности они представляют собой инновационное решение, нацеленное на повышение производительности, улучшение пользовательского опыта и расширение возможностей диагностики SQL Server.

Я также рассказал, как спланировать и выполнить миграцию на SQL Server 2019, какие инструменты и стратегии для этого используются, и

осветил тему совместимости баз данных для функциональной защиты и улучшения производительности.

Эта глава знаменует собой конец нашего совместного путешествия в мир SQL Server 2019. Я надеюсь, что вам так же сильно понравилось читать и изучать SQL Server 2019, как мне понравилось писать о нем. Для многих из вас эта книга станет началом знакомства с SQL Server 2019. Я же надеюсь, что вы будете продолжать слушать, как я рассказываю про SQL Server 2019. Но я также смотрю в будущее, в котором планируется выпуск новой версии, обеспечивающей высокую скорость работы в облаке и предлагающей новые инновационные изменения, которые делают SQL Server поистине уникальным продуктом в отрасли.

Предметный указатель

L

T

U

V

W

А

Б

В

Г

Д

Ж

З

И

K

M

H

O

П

P

C

Т

У

Ф

Х

Ш

Я