Руководство пользователя платформы «РЕД КВАНТ»

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

РЕД КВАНТ — это ориентированная на память распределенная база данных, кэширующая и вычислительная платформа для разработки приложений, интенсивно использующих данные, масштабируемые до сотен миллионов транзакций в секунду и петабайт данных в памяти. Эта ориентированная на память платформа следующего поколения может функционировать как DataGrid в памяти или может быть развернута как полнофункциональное постоянное хранилище данных с поддержкой транзакций SQL и ACID.

Основной принцип платформы РЕД КВАНТ заключается в том, чтобы хранить весь набор данных в памяти и на диске, повышая производительность приложений за счет возможностей кэширования и параллельной обработки данных. Такой принцип был определен как решение проблем производительности традиционных баз данных на основе дисков, поскольку платформа позволяет загружать и выполнять все необходимые наборы данных в памяти. Этот принцип устраняет значительное количество дисковых операций ввода-вывода, которые препятствуют транзакциям и создают «затор» в производительности традиционных систем баз данных.

Помимо определения, упомянутого выше, РЕД КВАНТ также можетиспользоваться как:

1. Полная In-memory база данных: РЕД КВАНТ может быть развернута как распределенная база данных с поддержкой SQL поверх нереляционной модели данных, всякий раз, когда отключено сохранение на собственном диске.

2. Мультимодельная база данных: под капотом РЕД КВАНТ хранит данные в хранилище «ключ-значение». РЕД КВАНТ хранит данные на специальных страницах памяти, начиная с версии 2.x, где вся память разбивается на страницы, внутри которых хранятся записи типа «ключ-значение». РЕД КВАНТ предоставляет SQL для моделирования и доступа к данным, что позволяет платформе действовать как мультимодельная база данных.

3. Транзакционная база данных: РЕД КВАНТ поддерживает транзакции на уровне API «ключ-значение», которые могут охватывать разные разделы на разных серверах.

4. Платформа микросервисов: благодаря возможности хранить данные и предоставлять сервисы по обработке данных с одного и того же вычислительного ресурса, РЕД КВАНТ является первым в области разработки отказоустойчивых, масштабируемых микросервисов.

5. Фабрики in-memory данных: РЕД КВАНТ также предоставляет такие функции, как управление большими данными, кластеризация веб-сеансов, Spring кэш, и может использоваться в качестве реализации для диспетчера кластеров.

Учитывая эти серьезные обещания, лучшими вариантами использования РЕД КВАНТ являются следующие:

1. Обработка больших объемов ACID-транзакций.
2. Кэш как сервис (CaaS).
3. Кэширование базы данных.
4. Обнаружение мошенничества в режиме онлайн.

5. Complex event processing (CEP) - Обработка сложных событий для IoT-проектов.
6. Аналитика в реальном времени.
7. Бизнес-приложения HTAP.
8. Быстрая обработка данных или реализация лямбда-архитектуры.

Таким образом, РЕД КВАНТ может хранить и обрабатывать большие объемы данных в памяти и на диске благодаря своей надежной архитектуре памяти. Он может перезагружать свое состояние всякий раз, когда перезапускается узел РЕД КВАНТ, либо с жесткого диска, либо по сети из постоянного хранилища. Это все еще база данных в памяти, несмотря на запись на диск, потому что диск используется только как журнал, для надежности доступный только для добавления. РЕД КВАНТ имеет следующие ключевые особенности:

1. Compute Grid.
2. Service Grid.
3. SQL Grid.
4. Ускоритель для работы с большими данными.
5. Streaming grid.
6. Машинное обучение.
7. Долговременная память.
8. Стороннее постоянное хранилище.
9. Поддержка ORM (Hibernate OGM и Spring Data).

На рисунке 1 показаны основные функции платформы РЕД КВАНТ.

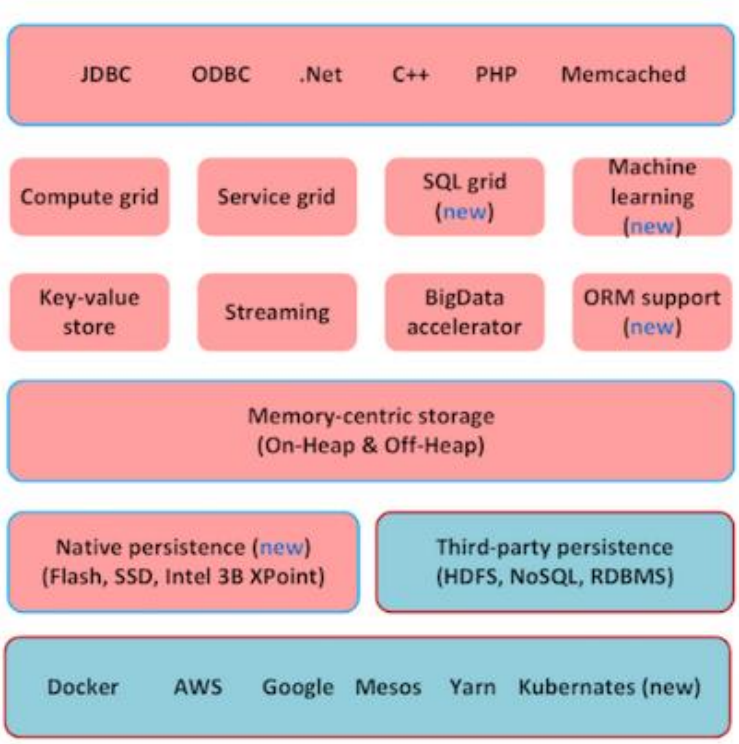

Рисунок 1 - Функции платформы РЕД КВАНТ

Основные технологии РЕД КВАНТ:

1. Написан на Java и C++.
2. Интегрируется со Spring.
3. База данных на основе H2 и Apache Calcite SQL движков.
4. Apache Lucene (полнотекстовый поиск).
5. Поддерживает JDBC, ODBC, .Net, C++, PHP и т. д.

Основные возможности, которые предоставляет РЕД КВАНТ:

- **Эластичность**. Кластер РЕД КВАНТ может увеличиваться горизонтально за счет добавления новых узлов. Иногда это называется масштабированием или горизонтальным масштабированием. РЕД КВАНТ масштабируется до тысяч машин на обычном оборудовании.
- **Надежность**. РЕД КВАНТ предоставляет надежные механизмы для предотвращения потери данных за счет устойчивости на основе дисков. У диска есть два существенных преимущества: они постоянны (при отключении питания данные не потеряются) и имеют меньшую стоимость за гигабайт, чем оперативная память. РЕД КВАНТ постоянно записывает транзакции в файл, доступный только для добавления (WAL) и периодически сбрасывает изменения на диск.
- **Поддержка SQL**. РЕД КВАНТ специально разработан для выполнения SQL на большом объеме данных. РЕД КВАНТ поставляется с совместимой с ANSI-99, горизонтально масштабируемой и отказоустойчивой распределенной базой данных SQL.

- **Децентрализованность.** Нет единой точки отказа или SPOF; каждый узел в кластере идентичен и хранит несколько копий данных.
- **Отказоустойчивость**. Данные автоматически реплицируются на несколько узлов с помощью конфигурации резервной копии. Вышедшие из строя узлы могут быть заменены без простоев.
- **Кэш как сервис (CaaS)**. РЕД КВАНТ поддерживает кэширование как сервис во всей организации, что позволяет нескольким приложениям из разных отделов получать доступ к управляемому кэшу в памяти вместо медленных баз данных на дисках.
- **Кэш 2-го уровня**. РЕД КВАНТ — идеальный уровень кэширования для использования в качестве распределенного кэша 2-го уровня в Hibernate или MyBatis. Этот кэш не ограничивается одним сеансом, а распределяется между сеансами, поэтому данные доступны для всего приложения, а не только для текущего пользователя. Кэш 2-го уровня может значительно повысить производительность приложений, поскольку часто используемые данные могут храниться в памяти на уровне приложений.
- **Совместное использование состояния в памяти между приложениями Spark**. Ориентированная на память архитектура РЕД КВАНТ обеспечивает эффективное совместное использование RDD с IgniteContext и IgniteRDD для совместного использования RDD между приложениями Spark. Ignite RDD позволяет легко обмениваться состояниями в памяти между различными заданиями или приложениями Spark. Любое приложение Spark может поместить данные в кэш РЕД КВАНТ с помощью общих RDD Ignite в памяти, которые позже будут доступны другому приложению Spark. Начиная с версии 2.4 РЕД КВАНТ также поддерживает современный API DataFrame Apache Spark. Можно обмениваться данными и состоянием между заданиями Spark с поддержкой Spark DataFrame, записывая и считывая фреймы данных в РЕД КВАНТ или из него.
- **Распределенные вычисления**. РЕД КВАНТ предоставляет набор простых API-интерфейсов, которые позволяют пользователю распределять вычисления и обработку данных между несколькими узлами кластера для повышения производительности. Распределенные сервисы РЕД КВАНТ очень полезны для разработки и реализации микросервисной архитектуры.
- **Потоковая передача**. РЕД КВАНТ позволяет обрабатывать непрерывные бесконечные потоки данных масштабируемым и отказоустойчивым способом в памяти, а не анализировать данные после их сохранения в базе данных.
- **Стороннее постоянное хранилище**. РЕД КВАНТ может сохранять записи кэша в СУБД, даже в NoSQL, например MongoDB или Cassandra.
- **Поддержка плагинов**. Встроенная система подключаемых модулей РЕД КВАНТ позволила третьим сторонам расширить основные функциональные возможности РЕД КВАНТ. Пользователь может реализовать такие функции, как аутентификация и авторизация в РЕД КВАНТ, используя систему плагинов.

1 ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ РЕД КВАНТ

Варианты использования базы данных в памяти широки и разнообразны. Любое приложение, которое может получить преимущество от повышения производительности, потенциально может извлечь выгоду от использования базы данных в оперативной памяти. Рассмотрим следующую традиционную трехуровневую архитектуру приложения, представленную на рисунке 2.

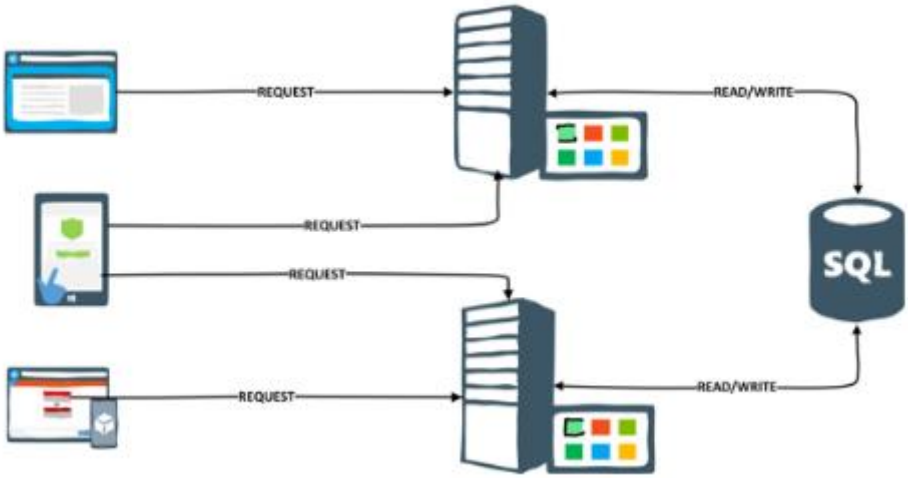

Рисунок 2 - Традиционная архитектура приложения

В традиционной архитектуре приложений используются хранилища данных с синхронными операциями чтения-записи. Это полезно для обеспечения согласованности и долговечности данных, но это может легко стать узким местом, если в очереди находится много транзакций.

База данных в памяти предназначена не только для кэширования и хранения наборов данных; она может предложить гораздо больше. База данных в памяти может использоваться для действий в режиме реального времени. Обработка событий в реальном времени и отказоустойчивость переводят базу данных в памяти на другой уровень. В этой главе выделено несколько примеров использования, демонстрирующих, как использовать РЕД КВАНТ для поддержки высокопроизводительных приложений реального времени, не затрагивая скорость или безопасность ценных данных.

1.1 КЭШИРОВАНИЕ ДЛЯ БЫСТРОГО ДОСТУПА К ДАННЫМ

Производительность веб-приложения варьируетсяв соответствии с ожиданиями пользователей. Исследования показывают, что скорость имеет значение в любом контексте. Потребность скорости загрузки веб-сайтов и мобильных приложений растет с каждым годом. База данных РЕД КВАНТ в памяти может служить уровнем кэширования для существующего приложения, обеспечивая быстрый вызов часто используемых данных. Он находится между серверами приложений и хранилищем данных. РЕД КВАНТ как In-memory data grid использует кэш часто используемых клиентом данных в активной памяти, а затем при необходимости может обращаться к постоянному хранилищу и даже асинхронно отправлять и получать обновления из постоянного хранилища. Архитектура приложения для кэширования часто используемых данных показана на рисунке 3.

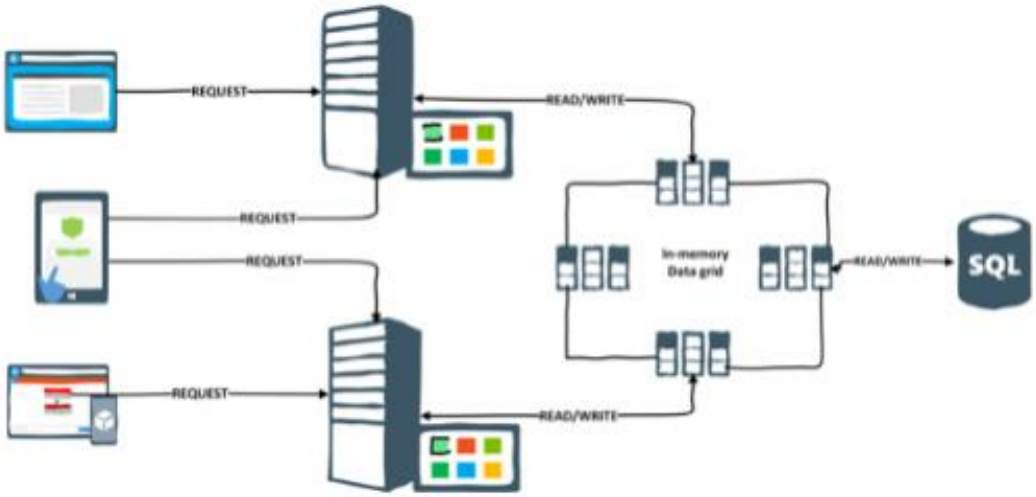

Рисунок 3 - Архитектура приложения для кэширования часто считываемых данных

Часто используемые данные перемещаются ближе к конечной точке приложения с помощью РЕД КВАНТ в качестве уровня кэширования. Такой подход сокращает время отклика при доступе к данным. РЕД КВАНТ динамически загружает данные в память при первом доступе к ним клиентского приложения. Обычно клиентскому приложению не требуется реализовывать какой-либо конкретный код в памяти. Большая часть библиотеки объектно-реляционного сопоставления (ORM) уже поддерживает РЕД КВАНТ в качестве кэша 2-го уровня, например Hibernate или MyBatis.

Часто на уровне кэширования не требуется персистентности, и кэши обычно распределяются по узлам кластера. Это позволяет получить низкую задержку до 200 микросекунд времени отклика.

Упомянутые выше подходы применимы не только к веб-приложению, но также могут использоваться для кэширования данных в механизме BPM или ESB (сервисная шина предприятия). Часто каждый механизм BPM использует базу данных для хранения контекста бизнес-процесса, который используется для запуска экземпляров бизнес-процесса. Кэширование такой информации в памяти может повысить производительность механизма BPM. Аналогичным образом, использование кэширования в памяти также является простым способом передачи данных или обмена состоянием из одного сервиса в другой в любом приложении ESB.

1.2 HTAP

HTAP расшифровывается как Гибридная транзакционно-аналитическая обработка - новый термин, введенный Gartner несколько лет назад. По сути, это единая система, которая поддерживает как OLTP (оперативную), так и OLAP (аналитическую) обработку. Данные сохраняются один раз в памяти и поэтому мгновенно доступны для аналитики.

OLTP + OLAP = HTAP

Исторически сложилось так, что каждое предприятие поддерживает многоуровневую архитектуру базы данных для обработки транзакций и аналитики для принятия бизнес-решения. Как правило, традиционное хранилище данных или OLAP-система извлекает данные из различных OLTP с помощью инструмента ETL (извлечение, преобразование и загрузка) с

заданным интервалом времени в день или в час. На рисунке 4 представлена высокоуровневая архитектура системы хранилища данных. Этот традиционный подход имеет несколько накладных расходов:

1. Вся система очень дорога в обслуживании.
2. Объем хранилища данных увеличивается с каждым днем, потому что каждый бизнес создает огромные объемы данных, и успех компании зависит от того, насколько эффективно эти данные используются. Хранение данных становится главной проблемой, поскольку в системе OLAP хранится огромное количество исторических данных.
3. Аналитика данных в режиме реального времени очень труднодостижима. Обычно передача данных из OLTP в OLAP-систему через ETL занимает много времени. Текущие бизнес-тенденции предполагают, что задержка данных продолжительностью даже в день может оказать значительное влияние на весь бизнес.

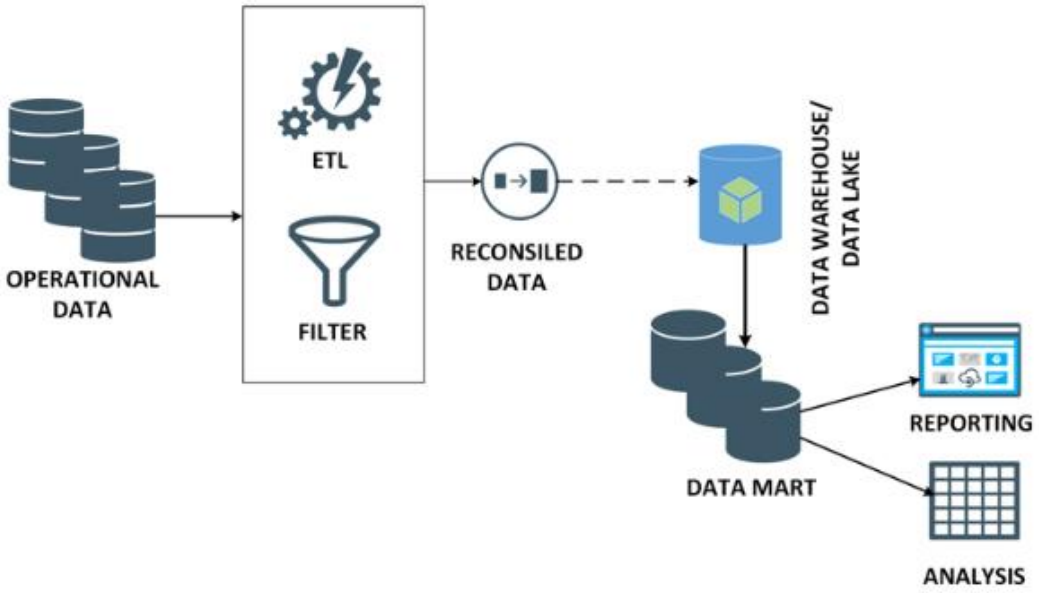

Рисунок 4 - Высокоуровневая архитектура системы хранилища данных

Все вышеперечисленные факторы обусловливают крайнюю необходимость перепроектирования системы OLTP/OLAP и объединения их в новую базу данных. Число баз данных в памяти и NoSQL растет; самое время для поиска нового решения, подходящего как для OLTP, так и для OLAP-сценариев. OLTP системы должны быть способны выполнять большой объем транзакций, а OLAP-система должна быть способна выполнять полезный анализ одних и тех же данных, но обе операции не должны мешать друг другу. База данных в памяти, такая как РЕД КВАНТ, может записывать огромные объемы данных на уровень в памяти, а с другой стороны, может сохранять данные на диск или в любую СУБД NoSQL/RDBMS для аналитической обработки в режиме реального времени. При таком подходе стадия ETL полностью исчезает, что устраняет барьер для проведения аналитики данных в режиме реального времени.

Как было упомянуто ранее, РЕД КВАНТ обеспечивает встроенную сохраняемость данных с высокой транзакционной активностью на диске, а также обеспечивает ANSI SQL для

запроса к той же базе данных. Теоретически, можно создать свою систему классов HTAP на основе РЕД КВАНТ для любого малого или среднего бизнеса с этими двумя функциями. В этом случае нет какой-либо отдельной OLAP-системы для запроса данных. Большинство современных инструментов бизнес-аналитики, таких как Tableau, Qlik или Pentahoh, имеют анализ данных в памяти, который является улучшением модели данных OLAP (рисунок 5).

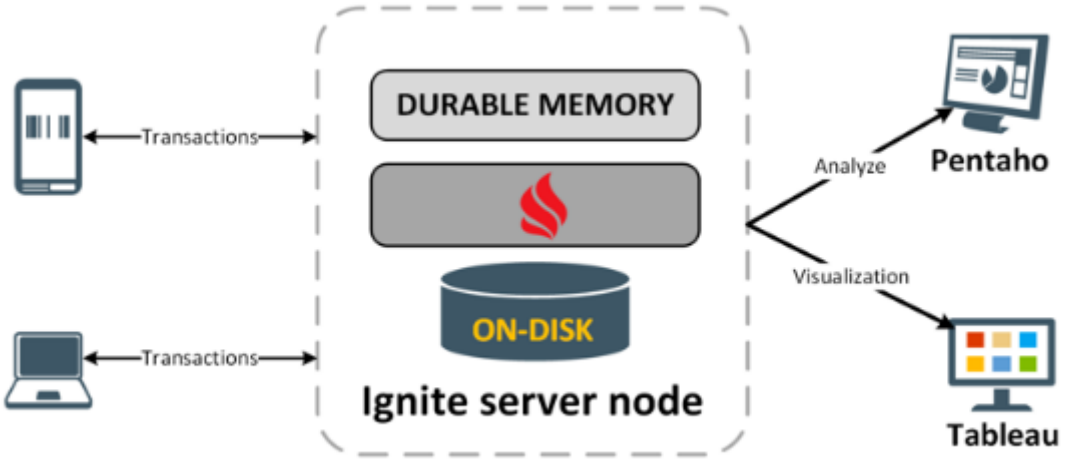

Рисунок 5 - Архитектура гибридной транзакционно-аналитической обработки

Можно подключиться к кластеру РЕД КВАНТ из tableau и анализировать данные различными способами с помощью драйвера Ignite ODBC. Однако также можно использовать Pentaho для подключения к кластеру РЕД КВАНТ через API JDBC для эффективного извлечения, преобразования, визуализации и анализа данных. Следует отметить, что иногда этот подход также называют NewSQL.

Система HTAP усложняется всякий раз, когда необходим интеллектуальный анализ данных или куб OLAP для эффективного запроса данных. Работа с кубами OLAP часто вызывает больше проблем и требует более тщательного архитектурного проектирования. В таком сценарии РЕД КВАНТ не может справиться в одиночку. Однако можно комбинировать РЕД КВАНТ с другой базой данных NoSQL, такой как Apache Cassandra. Сторонняя функция сохранения РЕД КВАНТ становится основой этой архитектуры. В целом, РЕД КВАНТ представляет собой OLTP-систему с полноценными транзакциями, которая хранит данные в памяти и во внешней базе данных NoSQL (такой как Cassandra) одновременно, как показано на рисунке 6.

Примечание. OLAP-куб представляет собой многомерную структуру данных, которая включает в себя факты, меры (то, что нужно считать или добавить) и измерения (атрибуты данных).

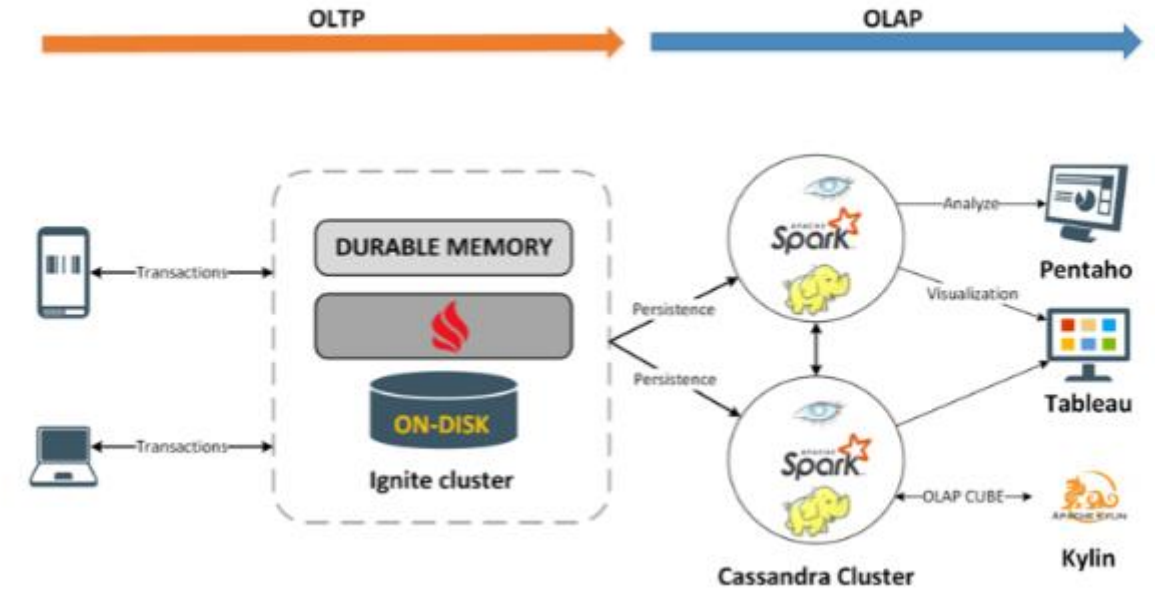

Рисунок 6 - РЕД КВАНТ

Основным преимуществом вышеописанной архитектуры является отсутствие этапов ETL. Данные немедленно становятся доступны в кластере Cassandra после совершения транзакций в кластере РЕД КВАНТ. Считывать данные с узла Cassandra можно даже во время сбоя узла РЕД КВАНТ. Вторая причина, по которой следует рассмотреть возможность использования отдельного кластера для сохранения данных, заключается в том, что можно импортировать данные из других внешних источников, а не из кластера РЕД КВАНТ. Можно использовать стандартное оборудование для создания data lake или витрин данных в зависимости от конкретных обстоятельств с помощью стека Cassandra и Spark. Такая система OLAP может существовать и развиваться независимо, обуславливаясь размером данных.

1.3 Обработка больших объемов транзакций

Рост облачных вычислений и мобильных приложений на предприятии и за его пределами обусловливает потребность в надежной архитектуре базы данных, которая может управлять транзакциями в реальном времени и предоставлять одновременный доступ к данным для большого числа пользователей. Покупки в интернет-магазине; обработка кредитных или дебетовых карт всегда требует чрезвычайно высокопроизводительного управления данными. В подобных случаях РЕД КВАНТ может обрабатывать большое количество одновременных транзакций, включающих петабайты операционных данных, совершая транзакции в памяти. По сравнению с традиционной системой управления реляционными базами данных РЕД КВАНТ также обеспечивает масштабируемую архитектуру, способную эластично масштабироваться по мере увеличения спроса. Такой подход сокращает время отклика и может сократить время транзакций с нескольких секунд до долей секунды.

Кроме того, встроенная персистентность РЕД КВАНТ обеспечивает полную устойчивость совершенных транзакций в обмен на повышенную производительность во время

выполнения транзакционных рабочих нагрузок. Надежность обеспечивается с помощью журнала транзакций (WAL — журнал упреждающей записи), который сохраняется на соответствующем устройстве (например, на жестком диске или SSD). Каждая транзакция, изменяющая данные, записывается в журнал, и журнал используется для восстановления состояния узла РЕД КВАНТ после сбоя питания или перезапуска базы данных/системы.

1.4 Быстрая обработка данных

В настоящее время большие данные — это одна из главных тем разговоров; однако вопрос не только в объеме. Большие данные можно описать такими характеристиками, как: объем, скорость, разнообразие и достоверность. Скорость передачи данных или потоки данных являются одной из основных проблем больших данных. Движение данных является массовым и непрерывным в больших данных. Данные показывают, что происходит с бизнесом прямо сейчас. Возможности воспользоваться преимуществами «горячих» данных выше, чем когда-либо, при использовании IoT в качестве непрерывного источника данных. Эти данные в режиме реального времени могут помочь предприятиям принимать ценные решения, обеспечивающие стратегические конкурентные преимущества по сравнению с другими. Обработку данных в движении часто называют быстрыми данными.

Быстрые данные = большие данные + обработка событий

Быстрые данные требуют различных методов обработки. Обработка данных может осуществляться на основе правил или потоковой обработки событий. РЕД КВАНТ предоставляет несколько способов для обработки непрерывных бесконечных потоков данных в масштабируемой и отказоустойчивой in-memory конфигурации. Один из них — Ignite RDD; реализация собственного RDD Spark и API DataFrame, который разделяет состояние RDD между другими заданиями Spark. Высокоуровневая архитектура использования IgniteRDD со Spark показана на рисунке 7.

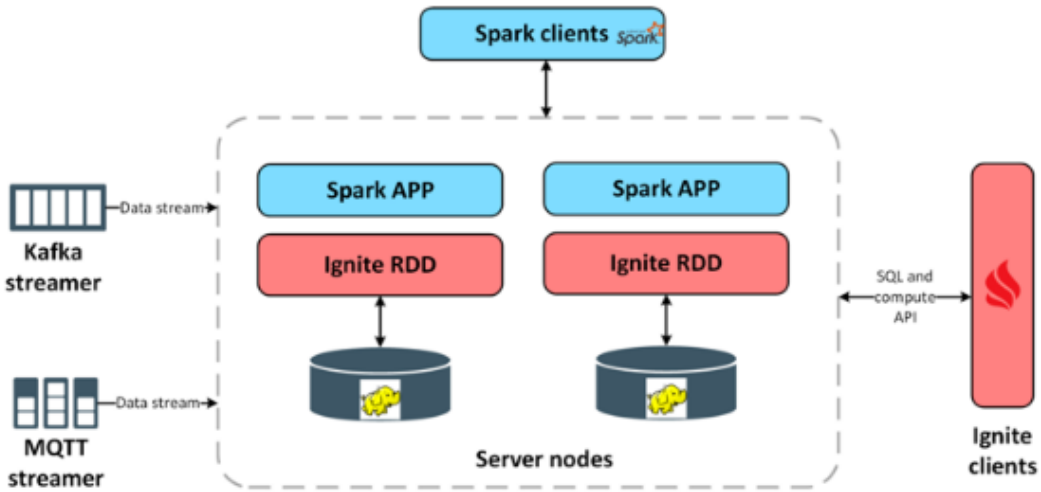

Рисунок 7 - Высокоуровневая архитектура использования IgniteRDD с Spark

Такой дизайн не только позволяет сопоставлять отношения и обнаруживать значимые закономерности в потоках данных, но также помогает обрабатывать их быстрее и намного эффективнее. In-memory data grid РЕД КВАНТ может управлять огромным объемом входящих данных и отправлять push-уведомления бизнес-приложению при возникновении изменений на

сервере. Возможность непрерывных запросов РЕД КВАНТ позволяет системам быстро получать доступ к значительному объему бесконечных входящих данных и выполнять действия.

1.5 Лямбда-архитектура

Назначение этой архитектуры обработки данных аналогично быстрой обработке данных, но с небольшой разницей. Лямбда-архитектура, предложенная для обработки огромных объемов данных, использует преимущества как пакетной, так и потоковой обработки там, где быстрые данные относятся только к потоковой обработке. Лямбда-архитектура описывает систему, состоящую из трех разных уровней: пакетной обработки, скоростной обработки (в реальном времени) и уровня обслуживания для ответа на запросы. Высокоуровневая модель лямбда-архитектуры представлена на рисунке 8.

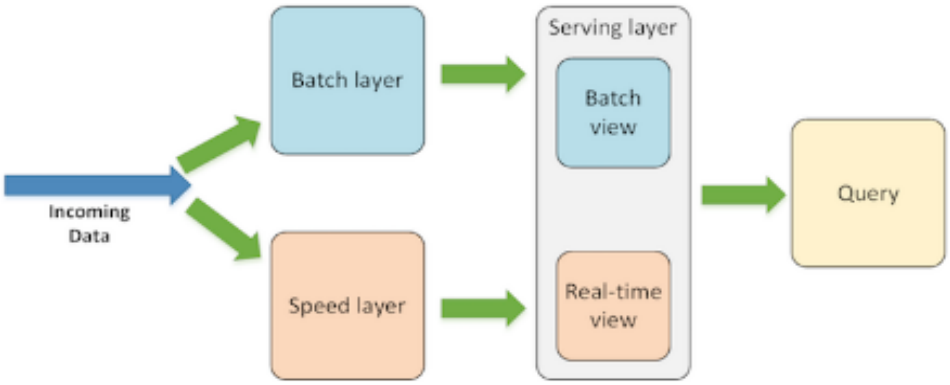

Рисунок 8 - Лямбда-архитектура

Лямбда-архитектура обладает следующими характеристиками:

1. Все данные отправляются параллельно как на пакетном, так и на скоростном уровне.
2. Мастер-данные неизменны.
3. Пакетный уровень предварительно вычисляет функции запроса с нуля; результат называется пакетным представлением.
4. Скоростной уровень компенсирует высокую задержку обновлений пакетных представлений.
5. Запрос разрешается путем получения результатов как из пакетного представления, так и из представления в реальном времени.

Поскольку лямбда-архитектура не зависит от базовых решений для каждого из уровней, существуют различные способы ее реализации. Каждый уровень требует определенных функций базовой реализации, которые могут быть решены с помощью множества различных технологий стека:

1. Пакетный уровень: однократная запись, многократное массовое чтение.
2. Скоростной уровень: случайное чтение-запись, инкрементное вычисление.
3. Уровень обслуживания: случайное чтение, пакетное вычисление и запись.

Например, одна из возможных реализаций лямбда-архитектуры может выглядеть следующим образом со следующими технологиями стека (рисунок 9):

1. Пакетный уровень: Hadoop, Spark
2. Скоростной уровень: Strom или РЕД КВАНТ
3. Уровень обслуживания: Hive или РЕД КВАНТ
4. Источник ввода: Кафка

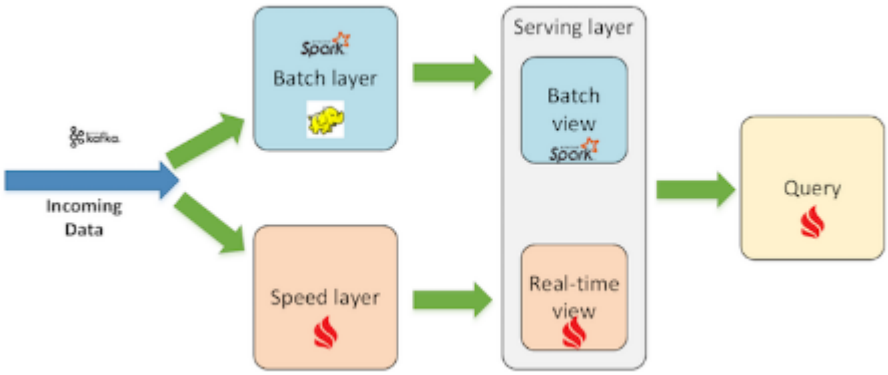

Рисунок 9 - Пример реализации лямбда-архитектуры

Пользовательскую логику для входящих данных в режиме совместного использования можно выполнять с помощью РЕД КВАНТ StreamAPI, например StreamTransformer и StreamVisitor. Также возможно изменить данные или добавить к ним любую логику предварительной обработки, прежде чем помещать данные в кэши. Кроме того, непрерывный запрос РЕД КВАНТ может помочь выполнять запросы, а затем получать уведомления об изменениях данных, попадающих в запрос. Приведенный выше дизайн является одним из возможных способов реализации лямбда-архитектуры, для базовых требований можно использовать Apache Storm или другой стек.

1.6 Надежное веб-ускорение

Веб-приложение можно легко масштабировать, добавляя в кластер дополнительные серверы приложений. Балансировка нагрузки между серверами приложений обычно обеспечивается за счет добавления устройств (аппаратных или программных), таких как балансировщик нагрузки. Однако веб-приложения, использующие веб-сеансы или предоставляющие службы с отслеживанием состояния, создают новую проблему. Балансировщик нагрузки переключает текущего пользователя на новый сервер приложений, который создает новый веб-сеанс, когда один из серверов приложений выходит из строя и текущий веб-сеанс теряется. Это переключение вызывает несколько проблем, в том числе:

1. Если данные сеанса пользователя были утеряны, то все несохраненные текущие данные также будут потеряны.

2. Снижение производительности веб-приложения. Создание Sticky connection в балансировщике нагрузки является операцией с интенсивным использованием ЦП.

Решение этой проблемы заключается в обеспечении кластеризации веб-сеансов. Функция In-memory data grid РЕД КВАНТ обеспечивает кластеризацию веб-сеансов, что может обеспечить отказоустойчивость веб-приложения и повысить его производительность. Есть возможность обмениваться веб-сеансами между веб-приложениями через кэши РЕД КВАНТ без изменения кода логики приложения (рисунок 10).

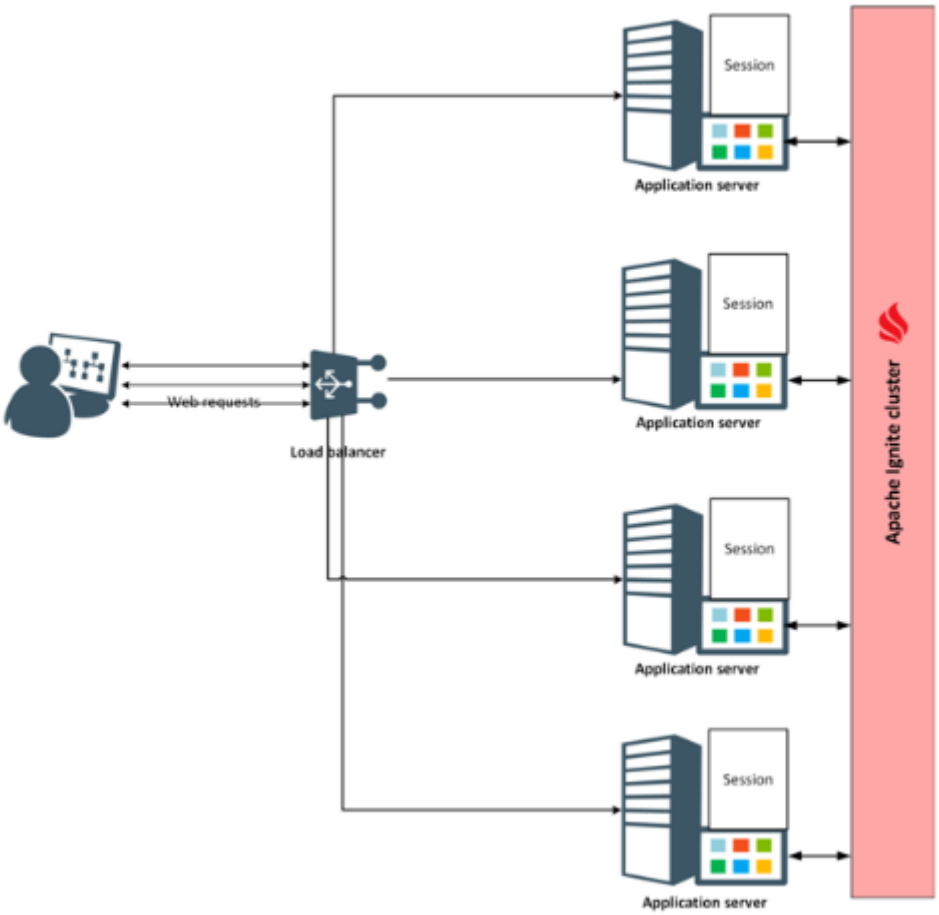

Рисунок 10 - Кластеризация веб-сеансов

Приведенный выше подход использует кэши РЕД КВАНТ в качестве распределенного хранилища веб-сеансов, совместно используемого всеми экземплярами сервера приложений. В случае сбоя сервера приложений балансировщик нагрузки перенаправляет запрос пользователя на новый сервер приложений, который имеет доступ к сеансу пользователя через распределенные кэши РЕД КВАНТ. Такой подход обеспечивает высочайший уровень доступности системы и качества обслуживания клиентов.

1.7 Микросервисы в распределенном режиме

Микросервисная архитектура обладает рядом преимуществ и обеспечивает уровень модульности, которого чрезвычайно сложно достичь с помощью монолитного приложения. Идея микросервисной архитектуры заключается в разработке небольшого подмножества взаимосвязанных приложений вместо создания одного большого приложения. Каждый микросервис имеет собственную многоуровневую архитектуру, аналогичную монолитному приложению (рисунок 11).

РЕД КВАНТ может предоставлять независимые узлы кэширования соответствующим микросервисам в одном и том же распределенном кластере и дает несколько преимуществ по сравнению с традиционными подходами:

1. Бизнес-логика выполняется вместе с данными в том же узле. Эта стратегия сокращает избыточное преобразование данных между сервером приложений и базой данных.
2. Работает как легковесный поток Java и может использовать те же ресурсы JVM.
3. Может обеспечить автоматическую отказоустойчивость сервиса.
4. Может сократить время выхода организаций на рынок для новых услуг.

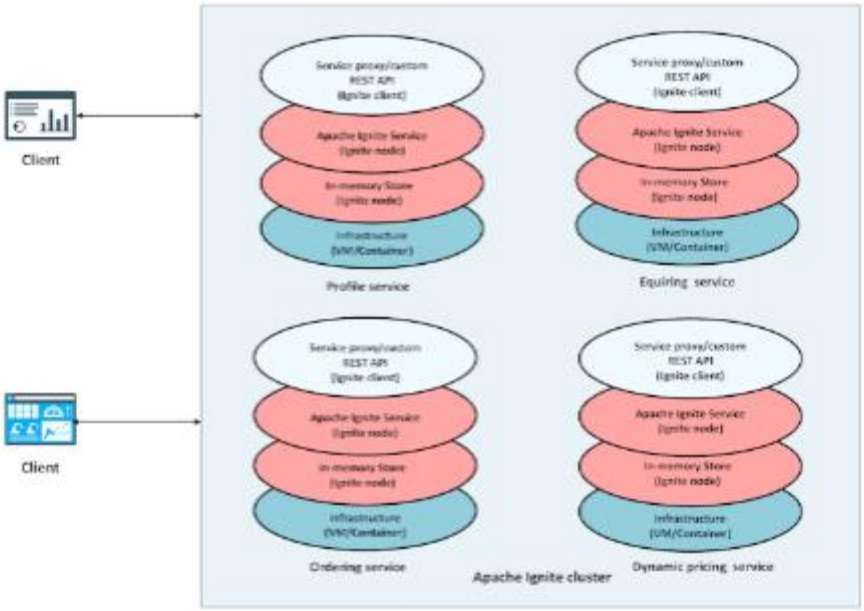

Рисунок 11 - Микросервисная архитектура

Службы, работающие в кластере в оперативной памяти, намного быстрее, чем сервер приложений на диске. Микросервисы РЕД КВАНТ основаны на сервисном grid, что обеспечивает платформу для автоматического развёртывания любого количества экземпляров распределённого сервиса в кластере.

1.8 КЭШ КАК СЕРВИС

Как правило, каждая организация поддерживает несколько различных информационных систем, таких как ERP, CRM или портал для обслуживания клиентов. В такой системе используется отдельный механизм кэширования и уровни для многократного ускорения работы приложения. По мере увеличения количества систем или сервисов эти отдельные уровни кэширования также увеличиваются, и в итоге получается N-ное количество неоднородных кэширующих кластеров.

РЕД КВАНТ может предоставить общий уровень кэширования для всей организации, что позволяет нескольким приложениям получать доступ к управляемому кэшу в памяти (рисунок 12). Уровень кэширования можно изолировать, отделив его от приложений.

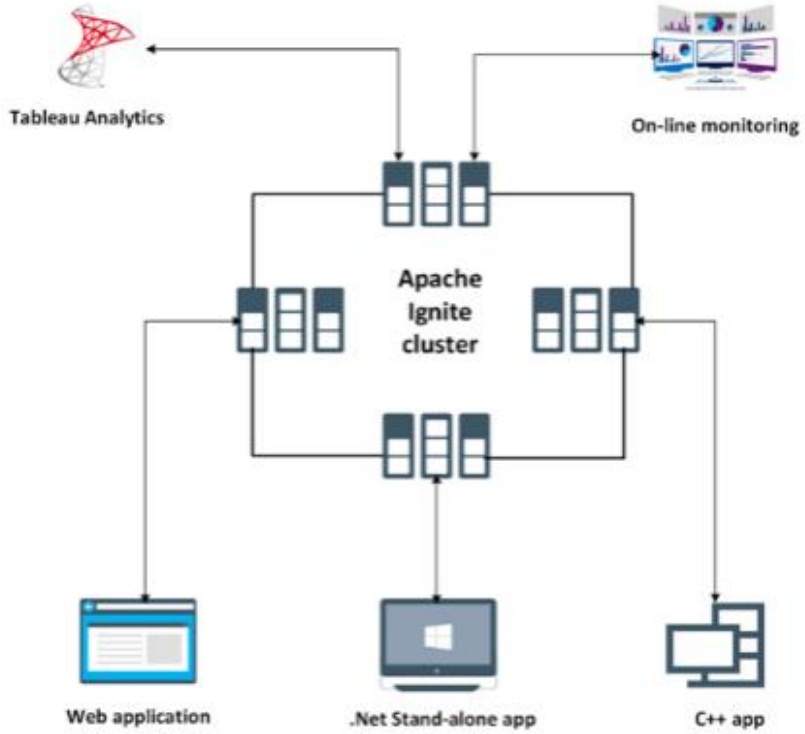

Рисунок 12 - Схема общего уровня кэширования

Любые приложения (Java, .Net, C++) в организации могут хранить и считывать данные из общего уровня кэширования. Нет необходимости создавать и развертывать локальную или частную инфраструктуру кэширования для каждого приложения, используя In-memory data grid в качестве сервиса. Приложения могут использовать РЕД КВАНТ в качестве кэш-памяти, выполнять запись в свою базу данных или загружать данные из базы данных в кэш. Это устраняет сложность управления сотней или более отдельных инфраструктур кэширования в рамках одной организации.

1.9 Ускорение работы с большими данными

Hadoop широко используется благодаря своей способности экономично хранить и анализировать большие наборы данных и уже давно перешагнул порог зарождающейся технологии. Однако накладные расходы на пакетное планирование и хранение данных на дисках сделали его непригодным для использования при анализе данных в реальном времени в производственной среде. Одним из основных факторов, ограничивающих масштабирование производительности Hadoop и Map/Reduce, является тот факт, что Hadoop использует файловую систему, которая создает множество файлов ввода-вывода (I/O). Альтернативой является хранение необходимых распределенных наборов данных в памяти или запуск функции Map/Reduce в памяти с данными (рисунок 13), что может устранить задержку ввода-вывода.

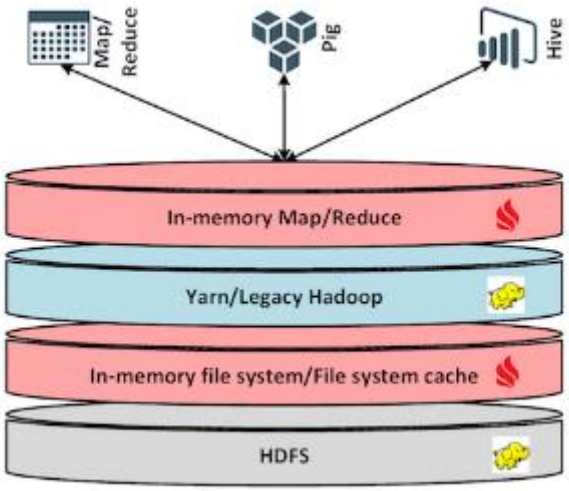

Рисунок 13 - Структура РЕД КВАНТ Hadoop

РЕД КВАНТ предлагает набор полезных компонентов, позволяющих выполнять задания Hadoop в памяти и операции с файловой системой. Ускоритель РЕД КВАНТ Hadoop может автоматически развертывать все необходимые исполняемые программы и библиотеки для выполнения Map/Reduce на JVM, что значительно сокращает время запуска до миллисекунд. Это ускоряет работу, позволяя избежать задержек при доступе к вторичному хранилищу. Кроме того, пары ключ-значение, размещенные в data grid, могут быть эффективно считаны в механизм выполнения, чтобы минимизировать время доступа, поскольку механизм выполнения интегрирован с In-memory data grid. Этот подход удобен, когда есть существующее приложение Hadoop Map/Reduce, и необходимо повысить производительность выполнения Map/Reduce без изменения кода.

1.10 МАШИННОЕ ОБУЧЕНИЕ В ПАМЯТИ

Для тех (особенно специалистов по данным), кто работает над моделью обучения на большом объеме данных, возникают две основные проблемы: перемещение данных (после обучения) из одной среды в другую и проблема масштабируемости среды. Хотя Apache Spark, Mahout или TensorFlow помогают обучить модель, они не помогут развернуть модель в производственной среде. Запуск алгоритмов машинного обучения (ML) или глубокого обучения (DL) с использованием базы данных в памяти решает следующие проблемы:

1. Запуск ML или DL на основе актуальных данных.

2. Использование локальности данных на каждом узле кластера с помощью совместно распределенной обработки для получения чрезвычайно высокой производительности алгоритмов.

РЕД КВАНТ (начиная с версии 2.0) предоставляет набор библиотек для построения прогностической модели машинного обучения, позволяющей максимально эффективно использовать базу данных в памяти, предоставляет набор алгоритмов машинного обучения или компонентов машинного обучения, таких как линейная регрессия, кластеризация K-средних, деревья решений и т. д. для производственного использования (рисунок 14).

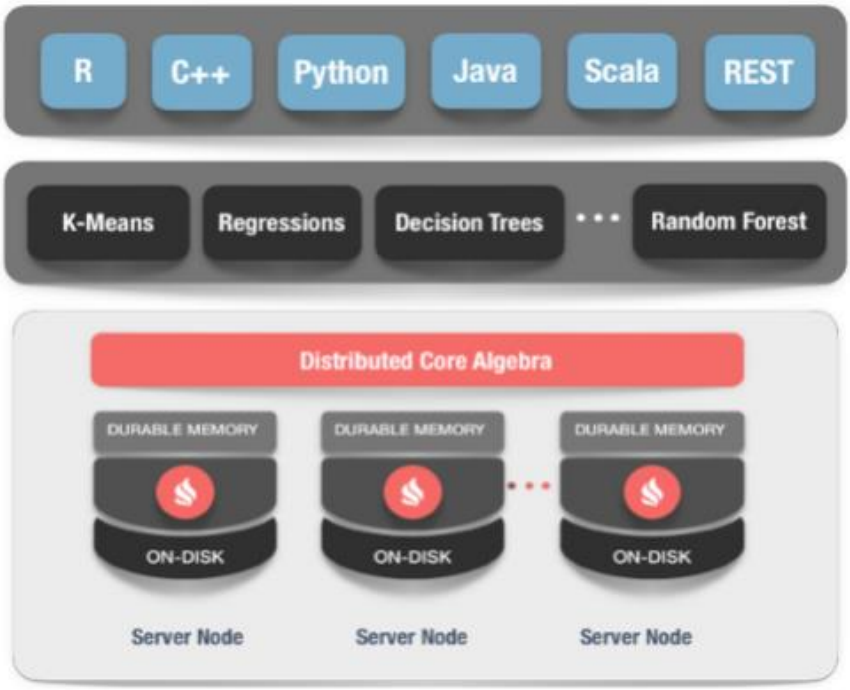

Рисунок 14 - Доступные наборы алгоритмов машинного обучения РЕД КВАНТ

Кроме того, РЕД КВАНТ предоставляет два основных компонента для разработки и запуска алгоритмов ML/DL поверх распределенной базы данных:

1. Распределенная базовая алгебра. Набор библиотек как основа алгоритмов ML/DL. Можно разработать собственную реализацию любых алгоритмов на основе базовой алгебры API РЕД КВАНТ.

2. Предварительно подготовленные алгоритмы ML/DL. Набор предварительно подготовленных алгоритмов ML/DL и примеров, поставляемых с каждым дистрибутивом РЕД КВАНТ. Таким образом, можно создавать и запускать существующие примеры, изучать их результаты и продолжать разработку своего приложения.

1.11 Геопространственная память

Геопространственные технологии, такие как ГИС и GPS, являются неотъемлемой частью нашей повседневной жизни. Геопространственное слово указывает на данные, которые имеют географический компонент. Это означает, что к записи в наборе данных прикреплена информация о местоположении в виде координат, адреса, города или почтового индекса. Большинство из них сформировано системой ГИС, где географические данные хранятся послойно и интегрируются с географическими программами. Другие геопространственные данные могут быть получены из данных GPS, спутниковых изображений и т. д.

Хотя существует множество разновидностей баз данных, которые поддерживают и подходят для геопространственных данных, не все базы данных в памяти обеспечивают геопространственную поддержку. Если необходима поддержка геопространственных данных в масштабе и с высокой пропускной способностью, РЕД КВАНТ является первоклассным решением для хранения и запроса геопространственных данных. РЕД КВАНТ предоставляет частичную реализацию набора топологий JTS: набор API-интерфейсов пространственных

предикатов и функций для обработки геометрии. Геопространственная библиотека РЕД КВАНТ поддерживает следующие возможности:

1. Типы данных: поддерживает точки, координаты и т. д.

2. Функции измерения: многоугольник (полигон), длина, площадь и т. д.

3. Индексирование: параллельное и без блокировки.

4. Пространственные объединения: кэши могут быть объединены с помощью их взаимосвязей.

1.12 Управление кластером

В качестве последнего варианта использования функция in-memory data grid РЕД КВАНТ может использоваться для управления кластером приложения. Обычно менеджер кластера поддерживает различные функции, такие как:

1. Создание высокодоступной среды для продолжения работы и предоставления ресурсов пользователям в случае сбоя.

2. Обнаружение и группировка узлов в кластере. Например, всякий раз, когда узел присоединяется к кластеру или покидает его.

3. Поддержка распределенной карты.

4. Поддержка распределенных блокировок и счетчиков в кластере.

Итак, РЕД КВАНТ рекомендуется использовать в качестве реализации менеджера кластера, если разрабатывается платформа или фреймворк с указанными выше требованиями. Несмотря на то, что Apache Zookeeper лидирует на рынке обслуживания кластеров, следует обратить внимание на РЕД КВАНТ, потому что РЕД КВАНТ мал и очень прост в использовании. Некоторые платформы приложений, такие как Eclipse Vert.x, для разработки реактивных приложений используют РЕД КВАНТ как реализацию своего менеджера кластеров.

2 МОДЕЛЬ ДАННЫХ

Хорошо спроектированная модель данных может повысить производительность приложения, более эффективно использовать ресурсы и помочь в достижении бизнес-целей. При разработке модели данных важно понимать, как данные распределяются в кластере РЕД КВАНТ и какие способы доступа к данным можно использовать.

Чтобы понять, как данные хранятся и используются в РЕД КВАНТ, полезно провести различие между физической организацией данных в кластере и логическим представлением данных, то есть тем, как пользователи будут просматривать данные в приложениях.

На физическом уровне каждая запись данных (либо запись кэша, либо строка таблицы) хранится в виде двоичного объекта, а весь набор данных делится на более мелкие наборы, называемые разделами. Разделы равномерно распределены между всеми узлами. Способ разделения данных на разделы и разбиения на узлы контролируется affinity функцией.

На логическом уровне данные должны быть представлены так, чтобы с ними было легко работать и чтобы конечным пользователям было удобно использовать их в приложениях. РЕД КВАНТ предоставляет два различных логических представления данных: кэш-память «ключ-значение» и таблицы SQL (схема). Хотя эти два представления могут показаться разными, на самом деле они эквивалентны и могут представлять один и тот же набор данных.

Стоит иметь в виду, что в РЕД КВАНТ концепции таблицы SQL и кэша «ключ-значение» являются двумя эквивалентными представлениями одной и той же (внутренней) структуры данных. Вы можете получить доступ к своим данным, используя либо API ключ-значение, либо операторы SQL, либо и то, и другое.

2.1 Кэш «ключ-значение» в сравнении с таблицей SQL

Кэш — это набор пар ключ-значение, к которым можно получить доступ через API ключ-значение. Таблица SQL в РЕД КВАНТ соответствует понятию таблиц в традиционных СУБД с некоторыми дополнительными ограничениями; например, каждая таблица SQL должна иметь первичный ключ.

Таблицу с первичным ключом можно представить в виде кэша «ключ-значение», в котором столбец первичного ключа служит ключом, а остальные столбцы таблицы представляют собой поля объекта (значение) (рисунок 15).

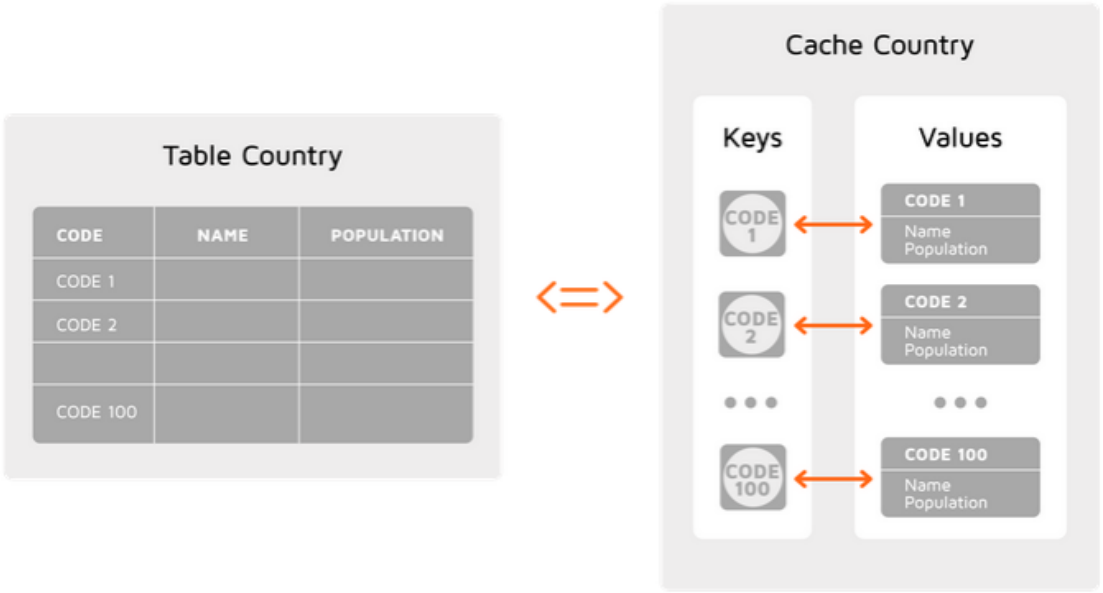

Рисунок 15 - Таблица SQL

Разница между этими двумя представлениями заключается в способе доступа к данным. Кэш «ключ-значение» позволяет работать с объектами через поддерживаемые языки программирования. Таблицы SQL поддерживают традиционный синтаксис SQL и могут помочь, например, при миграции из существующей базы данных. Вы можете комбинировать два подхода и использовать один из них (или оба) в зависимости от варианта использования.

Кэш API поддерживает следующие функции:

- Поддержка спецификации JCache (JSR 107);
- ACID транзакции;
- Continuous Query;
- События.

Даже после того, как будет настроен и запущен кластер, можно динамически создавать кэши «ключ-значение» и таблицы SQL.

В этой главе описаны важные компоненты модели распределения данных РЕД КВАНТ, в том числе партиционирование и коллокация, а также два различных интерфейса, которые можно использовать для доступа к данным (API с «ключ-значение» и SQL).

2.2 Двоичный формат объекта

РЕД КВАНТ хранит записи данных в определенном формате, называемом двоичными объектами. Этот формат сериализации дает несколько преимуществ:

- Прочитать произвольное поле из сериализованного объекта возможно без полной десериализации объекта. Это полностью устраняет необходимость развертывания классов ключей и значений на пути к классам узла сервера.

- Возможность добавлять или удалять поля из объектов того же типа. Учитывая, что серверные узлы не имеют определений классов моделей, эта возможность позволяет динамически изменять структуру объекта и даже позволяет сосуществовать нескольким клиентам с разными версиями определений классов.
- Позволяет создавать новые объекты на основе имени типа, вообще не имея определений классов, что позволяет создавать динамические типы.
- Обеспечивает беспрепятственное взаимодействие между платформами Java, .NET и C++.

Двоичные объекты можно использовать только тогда, когда используется двоичный маршаллер по умолчанию (т. е. в конфигурации не задан другой маршаллер).

Дополнительные сведения о настройке и использовании двоичных объектов см. пункт Работа с бинарными объектами.

2.3 Разделение данных

Разделение данных — это метод разделения больших наборов данных на более мелкие фрагменты и их сбалансированного распределения между всеми серверными узлами. *Разделение данных подробно см. пункт* Разделение данных.

2.4 Разделение данных

Разделение контролируется affinity-функцией. Affinity-функция определяет соответствие между ключами и разделами. Каждый раздел идентифицируется номером из ограниченного набора (по умолчанию от 0 до 1023). Набор разделов распределяется между доступными в данный момент серверными узлами. Таким образом, каждый ключ сопоставляется с определенным узлом и хранится на этом узле (рисунок 16). Когда количество узлов в кластере изменяется, разделы перераспределяются между новым набором узлов посредством процесса, называемого ребалансировкой.

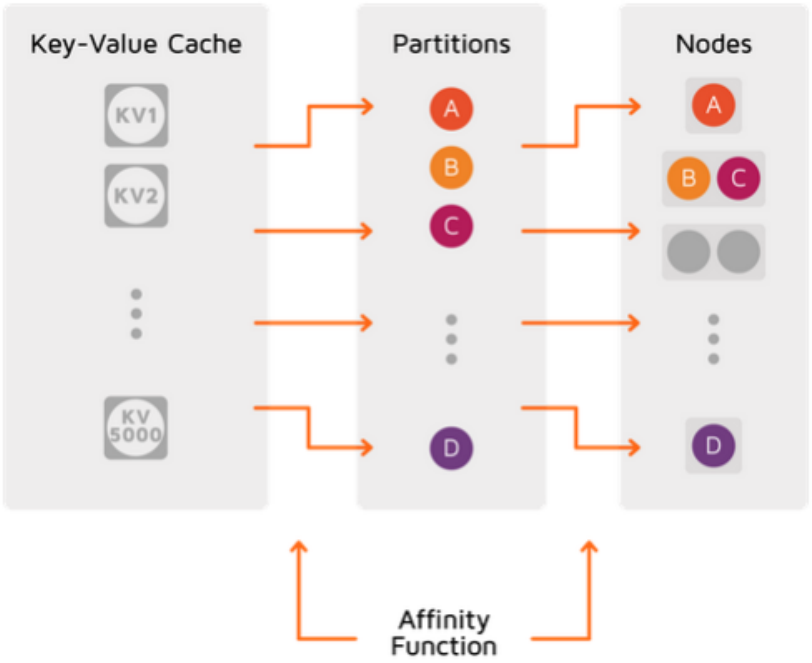

Рисунок 16 - Affinity-функция

Affinity-функция принимает в качестве аргумента ключ привязки. Affinity-ключом может быть любое поле объектов, хранящихся в кэше (любой столбец в таблице SQL). Если affinity-ключ не указан, используется ключ по умолчанию (в таблице SQL это столбец PRIMARY KEY-первичный ключ).

Партиционирование повышает производительность за счет распределения операций чтения и записи. Кроме того, можно спроектировать модель данных таким образом, чтобы записи данных, которые используются вместе, хранились вместе (т. е. в одном разделе). Когда запрашиваются эти данные, сканируется только небольшое количество разделов. Этот метод называется affinity-коллокацией.

Партиционирование помогает добиться линейной масштабируемости практически в любом масштабе. Можно добавлять дополнительные узлы в кластер по мере роста набора данных, РЕД КВАНТ гарантирует, что данные распределяются «равно» между всеми узлами.

Affinty-функция управляет сопоставлением записей данных с партициями, а партиций – с узлами. Affinity-функция по умолчанию реализует алгоритм рандеву-хэширования или алгоритм HRW (https://en.wikipedia.org/wiki/Rendezvous_hashing). Это допускает небольшое расхождение в сопоставлении разделов с узлами (т. е. некоторые узлы могут отвечать за несколько большее количество разделов, чем другие). Однако affinity-функция гарантирует, что при изменении топологии разделы переносятся только на новый присоединившийся узел или с ушедшего узла. Между остальными узлами обмен данными не происходит.

2.4.1 Партиционированный/Реплицированный режим

При создании кэша или таблицы SQL можно выбрать партиционированный или реплицированный режим работы кэша. Эти два режима предназначены для разных сценариев использования и обеспечивают различные преимущества в производительности и доступности.

В партиционированном режиме все разделы поровну распределяются между всеми серверными узлами. Этот режим является наиболее масштабируемым режимом распределенного кэша и позволяет хранить столько данных, сколько умещается в общей памяти (ОЗУ и диске), доступной на всех узлах. По сути, чем больше узлов, тем больше данных можно хранить.

В отличие от реплицированного режима, где обновления обходятся дорого, поскольку необходимо обновлять каждый узел в кластере, в партиционированном режиме обновления становятся дешевыми, поскольку для каждого ключа необходимо обновлять только один основной узел (и, возможно, 1 или несколько резервных узлов). Однако чтение несколько дороже, потому что данные кэшируются только на определенных узлах.

Партиционированные кэши идеально подходят для больших наборов данных и частых обновлений.

На рисунке 17 представлено распределение партиционированного кэша. По сути, есть ключ A, назначенный узлу, работающему в JVM1, ключ B, назначенный узлу, работающему в JVM3, и т. д.

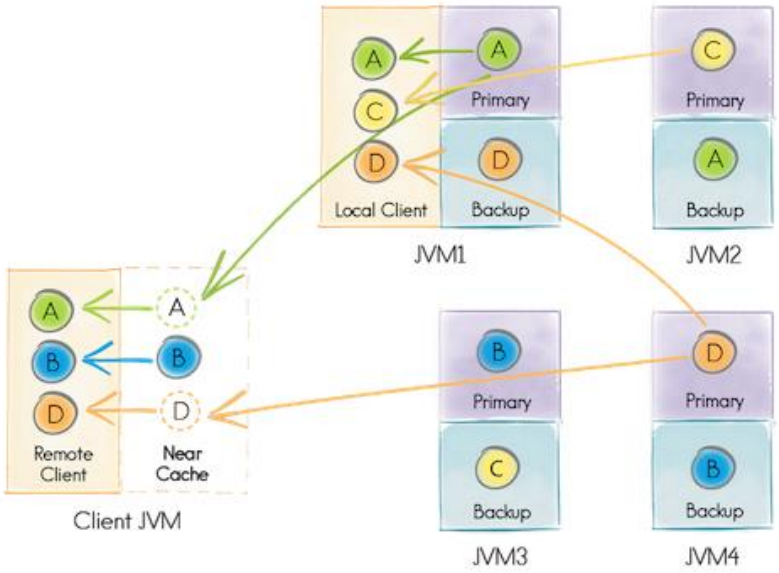

Рисунок 17 - Партиционированный кэш

В режиме репликации все данные (каждый раздел) реплицируются на каждый узел в кластере. Этот режим кэширования обеспечивает максимальную доступность данных, поскольку они доступны на каждом узле. Однако каждое обновление данных должно распространяться на все остальные узлы, что может повлиять на производительность и масштабируемость.

Реплицированные кэши идеальны, когда наборы данных малы, а обновления происходят нечасто.

На рисунке 18 узел, работающий в JVM1, является основным узлом для ключа A, но он также хранит резервные копии для всех остальных ключей (B, C, D).

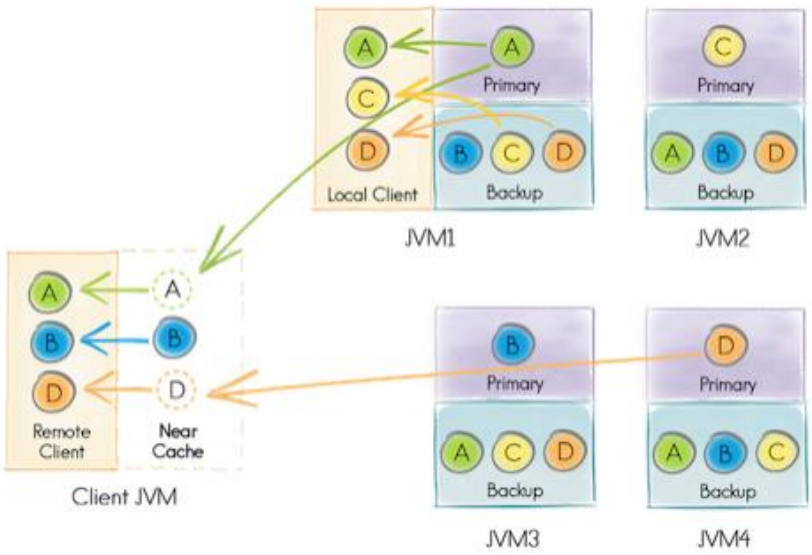

Рисунок 18 - Реплицированный кэш

Поскольку одни и те же данные хранятся на всех узлах кластера, размер реплицированного кэша ограничен объемом памяти (ОЗУ и диска), доступным на узле. Этот режим идеально подходит для сценариев, в которых операции чтения кэша происходят намного чаще, чем записи кэша, а наборы данных небольшие. Если система выполняет поиск в кэше более 80% времени, следует рассмотреть возможность использования режима репликационного кэширования.

2.4.2 Резервное копирование разделов

По умолчанию РЕД КВАНТ хранит одну копию каждого раздела (одну копию всего набора данных). В этом случае, если один или несколько узлов становятся недоступными, теряется доступ к разделам, хранящимся на этих узлах. Чтобы избежать этого, можно настроить РЕД КВАНТ на сохранение резервных копий каждого раздела.

По умолчанию резервное копирование отключено.

Резервные копии настраиваются для каждого кэша (таблицы). Если настроить 2 резервные копии, кластер будет поддерживать 3 копии каждого раздела. Один из разделов называется основным, а два других — резервными. В более широком смысле узел, имеющий первичный раздел, называется первичным узлом для ключей, хранящихся в этом разделе. Узел с резервными разделами называется резервным узлом.

Когда узел с основным разделом для некоторого ключа покидает кластер, РЕД КВАНТ запускает процесс обмена картами разделов (PME - Partition Map Exchange). PME помечает один из резервных разделов (если они настроены) для ключа как первичный.

Резервные разделы повышают доступность данных, а в некоторых случаях и скорость операций чтения, поскольку РЕД КВАНТ считывает данные из резервных копий разделов, если они доступны на локальном узле (это поведение по умолчанию, которое можно отключить). Однако они также увеличивают потребление памяти или размер постоянного хранилища (если они включены).

Резервные разделы можно настроить только в режиме партиционирования.

2.4.3 Обмен картами разделов

Обмен картами разделов (PME - Partition Map Exchange) — это процесс обмена информацией о распределении разделов (карте разделов) в кластере, чтобы каждый узел знал, где искать определенные ключи. PME требуется всякий раз, когда изменяется распределение разделов для любого кэша, например, когда в топологию добавляются новые узлы или старые узлы покидают топологию (будь то по запросу пользователя или из-за сбоя).

Примеры событий, запускающих PME, включают (но не ограничиваются ими):

- Новый узел присоединяется/покидает топологию.
- Новый кэш запускается/останавливается.
- Создается индекс.

Когда происходит одно из событий, инициирующих PME, кластер ожидает завершения всех текущих транзакций, а затем запускает PME. Во время PME новые транзакции откладываются до завершения процесса.

Процесс PME работает следующим образом: узел-координатор запрашивает у всех узлов информацию о разделах, которыми они владеют. Каждый узел отправляет эту информацию координатору. Как только узел-координатор получает сообщения от всех узлов, он объединяет информацию в полную карту разделов и отправляет ее всем узлам. Когда координатор получил подтверждающие сообщения от всех узлов, PME считается выполненным.

Подробнее см. пункт Ребаланс данных.

Может случиться так, что на протяжении жизненного цикла кластера некоторые разделы данных будут потеряны из-за отказа основного узла и резервных узлов, на которых хранились копии разделов. Такая ситуация приводит к частичной потере данных и требует решения в соответствии с вариантом использования. Подробную информацию о политиках потери разделов см. в разделе Политика потери разделов руководства администратора.

2.5 Распределение данных

Во многих случаях выгодно размещать разные записи вместе, если к ним часто обращаются. Таким образом, запросы к нескольким записям выполняются на одном узле (где хранятся объекты). Эта концепция известна как affinity-колокация.

Записи назначаются разделам affinity-функцией. Объекты с одинаковыми affinity-ключами помещаются в одни и те же разделы. Это позволяет спроектировать модель данных таким образом, чтобы связанные записи хранились вместе. «Связанные» здесь относятся к объектам, которые находятся в отношениях родитель-потомок, или к объектам, которые часто запрашиваются вместе.

Например, предположим, что есть объекты Person и Company, и у каждого человека есть поле companyId, которое указывает компанию, в которой работает человек. Указание Person.companyId и Company.ID в качестве affinity-ключей, гарантирует, что все лица,

работающие в одной и той же компании, хранятся на одном узле, где также хранится объект Company. Запросы людей, работающих в конкретной компании, обрабатываются на одном узле.

2.5.1 НАСТРОЙКА AFFINITY-КЛЮЧА

Если affinity-ключ не указан явно, ключ кэша будет использоваться как affinity-ключ по умолчанию. Если создаются кэши в виде таблиц SQL с помощью SQL-запросов, PRIMARY KEY является affinity-ключом по умолчанию.

Если необходимо объединить данные из двух кэшей с помощью другого поля, придется использовать сложный объект в качестве ключа. Этот объект обычно содержит поле, которое однозначно идентифицирует объект в этом кэше, и поле, которое необходимо использовать для колокации.

Существует несколько способов настроить пользовательское affinity-поле в пользовательском ключе, которые описаны ниже.

Следующий пример иллюстрирует, как можно колоцировать объекты «Person» с объектами «Company» с помощью пользовательского класса ключей и аннотации @AffinityKeyMapped.

```
public class AffinityCollocationExample {


  static class Person {
    private int id;
    private String companyId;
    private String name;

    public Person(int id, String companyId, String name) {
      this.id = id;
      this.companyId = companyId;
      this.name = name;
    }

    public int getId() {
      return id;
    }
  }

  static class PersonKey {
    private int id;

    @AffinityKeyMapped
    private String companyId;

    public PersonKey(int id, String companyId) {
      this.id = id;
      this.companyId = companyId;
    }
  }

  static class Company {
    private String id;
```

```
        private String name;

        public Company(String id, String name) {
          this.id = id;
          this.name = name;
        }

        public String getId() {
          return id;
        }
      }

      public void configureAffinityKeyWithAnnotation() {
        CacheConfiguration<PersonKey, Person> personCfg = new CacheConfiguration<PersonKey,
Person>("persons");
        personCfg.setBackups(1);

        CacheConfiguration<String, Company> companyCfg = new CacheConfiguration<>("companies");
        companyCfg.setBackups(1);

        try (Ignite ignite = Ignition.start()) {
          IgniteCache<PersonKey, Person> personCache = ignite.getOrCreateCache(personCfg);
          IgniteCache<String, Company> companyCache = ignite.getOrCreateCache(companyCfg);

          Company c1 = new Company("company1", "My company");
          Person p1 = new Person(1, c1.getId(), "John");

          // Both the p1 and c1 objects will be cached on the same node
          personCache.put(new PersonKey(p1.getId(), c1.getId()), p1);
          companyCache.put("company1", c1);

          // Get the person object
          p1 = personCache.get(new PersonKey(1, "company1"));
        }
      }

    }
```

Также можно настроить поле affinity-ключа в конфигурации кэша, используя класс CacheKeyConfiguration.

```
    public void configureAffinityKeyWithCacheKeyConfiguration() {

      CacheConfiguration<PersonKey, Person> personCfg = new CacheConfiguration<PersonKey,
Person>("persons");
      personCfg.setBackups(1);

      // Configure the affinity key
      personCfg.setKeyConfiguration(new CacheKeyConfiguration("Person", "companyId"));

      CacheConfiguration<String, Company> companyCfg = new CacheConfiguration<String,
Company>("companies");
      companyCfg.setBackups(1);

      Ignite ignite = Ignition.start();

      IgniteCache<PersonKey, Person> personCache = ignite.getOrCreateCache(personCfg);
      IgniteCache<String, Company> companyCache = ignite.getOrCreateCache(companyCfg);
```

```
        Company c1 = new Company("company1", "My company");
        Person p1 = new Person(1, c1.getId(), "John");

        // Both the p1 and c1 objects will be cached on the same node
        personCache.put(new PersonKey(1, c1.getId()), p1);
        companyCache.put(c1.getId(), c1);

        // Get the person object
        p1 = personCache.get(new PersonKey(1, "company1"));
    }
```

Вместо определения пользовательского класса ключа можно использовать класс AffinityKey, который разработан специально для использования пользовательского Affinity-соответствия.

```
    public void configureAffinitKeyWithAffinityKeyClass() {

        CacheConfiguration<AffinityKey<Integer>, Person> personCfg = new
CacheConfiguration<AffinityKey<Integer>, Person>(
                "persons");
        personCfg.setBackups(1);

        CacheConfiguration<String, Company> companyCfg = new CacheConfiguration<String,
Company>("companies");
        companyCfg.setBackups(1);

        Ignite ignite = Ignition.start();

        IgniteCache<AffinityKey<Integer>, Person> personCache = ignite.getOrCreateCache(personCfg);
        IgniteCache<String, Company> companyCache = ignite.getOrCreateCache(companyCfg);

        Company c1 = new Company("company1", "My company");
        Person p1 = new Person(1, c1.getId(), "John");

        // Both the p1 and c1 objects will be cached on the same node
        personCache.put(new AffinityKey<Integer>(p1.getId(), c1.getId()), p1);
        companyCache.put(c1.getId(), c1);

        // Get the person object
        p1 = personCache.get(new AffinityKey(1, "company1"));
    }
```

2.6 СЕРИАЛИЗАЦИЯ

Бинарный маршаллер — это компонент РЕД КВАНТ, отвечающий за сериализацию данных. Он имеет преимущества:

- Позволяет считывать произвольное поле из сериализованной формы объекта без полной десериализации объекта. Это преимущество полностью устраняет необходимость развертывания классов ключей и значений кэша в пути к классам серверного узла.
- Позволяет добавлять и удалять поля из объектов того же типа. Учитывая, что серверные узлы не имеют определений классов моделей, эта возможность позволяет динамически

изменять структуру объекта и даже позволяет сосуществовать нескольким клиентам с разными версиями определений классов.

- Позволяет создавать новые объекты на основе имени типа, не имея определений классов, что позволяет создавать динамические типы.

Двоичные объекты можно использовать только тогда, когда используется двоичный маршаллер по умолчанию (т. е. никакой другой маршаллер явно не задан для конфигурации).

Примечание: Существует несколько ограничений, налагаемых реализацией формата BinaryObject:

- Внутри РЕД КВАНТ не записывает имена полей и типов, а использует хэш имени в нижнем регистре для идентификации поля или типа. Это означает, что поля или типы с одинаковым хэшем имени не разрешены. Несмотря на то, что сериализация не будет работать из коробки в случае противоречия хэшей, РЕД КВАНТ предоставляет способ разрешить эту коллизию на уровне конфигурации.

- По той же причине формат BinaryObject не допускает одинаковых имен полей на разных уровнях иерархии классов.

- Если класс реализует интерфейс Externalizable, РЕД КВАНТ будет использовать OptimizedMarshaller вместо двоичного. OptimizedMarshaller использует методы writeExternal() и readExternal() для сериализации и десериализации объектов этого класса, что требует добавления классов Externalizable объектов в путь к классам серверных узлов.

Фасад IgniteBinary, который можно получить из экземпляра РЕД КВАНТ, содержит все необходимые методы для работы с бинарными объектами.

Примечание: Автоматический расчет хэш-кода и реализация Equals

Есть несколько ограничений, налагаемых реализацией формата BinaryObject:

Если объект может быть сереализован в бинарную форму, то РЕД КВАНТ вычислит его хэш-код при сериализации и запишет в результирующий бинарный массив. Кроме того, РЕД КВАНТ предоставляет собственную реализацию метода equals для нужд сравнения бинарных объектов. Это означает, что не нужно переопределять методы GetHashCode и Equals пользовательских ключей и значений, чтобы их можно было использовать в РЕД КВАНТ, если только они не могут быть сериализованы в двоичную форму. Например, объекты типа Externalizable не могут быть сериализованы в двоичную форму и требуют реализации методов hashCode и equals вручную. Дополнительные сведения см. в примечании с ограничениями выше.

2.6.1 Настройка бинарных объектов

В подавляющем большинстве случаев нет необходимости дополнительно настраивать бинарные объекты.

Однако в случае, когда нужно переопределить вычисление идентификаторов типов и полей по умолчанию или подключить BinarySerializer, объект BinaryConfiguration должен быть определен в IgniteConfiguration. Этот объект позволяет указать глобальный name mapper,

глобальный маппер идентификаторов и глобальный двоичный сериализатор, а также мапперы и сериализаторы для каждого типа. Wildcards поддерживаются для конфигурации каждого типа, и в этом случае предоставленная конфигурация будет применяться ко всем типам, которые соответствуют шаблону имени типа.

```
<bean id="ignite.cfg" class="org.apache.ignite.configuration.IgniteConfiguration">


 <property name="binaryConfiguration">
  <bean class="org.apache.ignite.configuration.BinaryConfiguration">

   <property name="nameMapper" ref="globalNameMapper"/>
   <property name="idMapper" ref="globalIdMapper"/>

   <property name="typeConfigurations">
    <list>
     <bean class="org.apache.ignite.binary.BinaryTypeConfiguration">
      <property name="typeName" value="org.apache.ignite.examples.*"/>
      <property name="serializer" ref="exampleSerializer"/>
     </bean>
    </list>
   </property>
  </bean>
 </property>
</bean>
```

2.6.2 Бинарный объект API

По умолчанию РЕД КВАНТ работает с десериализованными значениями, поскольку это наиболее распространенный вариант использования. Чтобы включить обработку BinaryObject, пользователю необходимо получить экземпляр IgniteCache с помощью метода withKeepBinary(). Если этот флаг включен, он гарантирует, что объекты, возвращенные из кэша, будут в формате BinaryObject, когда это возможно. То же самое относится к значениям, передаваемым в EntryProcessor и CacheInterceptor.

Примечание: Зависимость от платформы

Следует обратить внимание, что не все типы будут представлены как BinaryObject, если включен флаг withKeepBinary(). Существует набор типов «платформы», который включает в себя примитивные типы, String, UUID, Date, Timestamp, BigDecimal, Collections, Maps и массивы, которые никогда не будут представлены как BinaryObject.

В приведенном ниже примере тип ключа Integer не изменяется, поскольку это зависит от платформы.

```
// Create a regular Person object and put it to the cache.

Person person = buildPerson(personId);
ignite.cache("myCache").put(personId, person);

// Get an instance of binary-enabled cache.
IgniteCache<Integer, BinaryObject> binaryCache = ignite.cache("myCache").withKeepBinary();

// Get the above person object in the BinaryObject format.
BinaryObject binaryPerson = binaryCache.get(personId);
```

Экземпляры BinaryObject неизменяемы. Экземпляр BinaryObjectBuilder должен использоваться для обновления полей и создания нового BinaryObject.

Экземпляр BinaryObjectBuilder можно получить из фасада IgniteBinary. Конструктор может быть создан с использованием имени типа, в этом случае возвращаемый конструктор не будет содержать полей, или он может быть создан с использованием существующего BinaryObject, в этом случае возвращаемый конструктор скопирует все поля из данного BinaryObject.

Другой способ получить экземпляр BinaryObjectBuilder — вызвать toBuilder() для существующего экземпляра BinaryObject. Это также скопирует все данные из BinaryObject в созданный билдер.

Ограничения:

- Невозможно изменить типы существующих полей.
- Невозможно изменить порядок значений перечисления или добавить новые константы в начало или середину списка значений перечисления. Однако есть возможность добавить новые константы в конец списка.

Ниже приведен пример использования BinaryObject API для обработки данных на серверных узлах без развертывания пользовательских классов на серверах и без фактической десериализации данных.

```
// The EntryProcessor is to be executed for this key.

int key = 101;

cache.<Integer, BinaryObject>withKeepBinary().invoke(
  key, new CacheEntryProcessor<Integer, BinaryObject, Object>() {
    public Object process(MutableEntry<Integer, BinaryObject> entry,
                          Object... objects) throws EntryProcessorException {
          // Create builder from the old value.
      BinaryObjectBuilder bldr = entry.getValue().toBuilder();

      //Update the field in the builder.
      bldr.setField("name", "Ignite");

      // Set new value to the entry.
      entry.setValue(bldr.build());

      return null;
    }
  });
```

2.6.3 Метаданные типа BinaryObject

Как упоминалось выше, во время выполнения структура двоичного объекта может быть изменена, поэтому также может быть полезно получить информацию о конкретном типе, который хранится в кэше, например, имена полей, имена типов полей и имена affinity-полей. С помощью интерфейса BinaryType РЕД КВАНТ упрощает выполнение этого требования.

Этот интерфейс также представляет более быструю версию геттера значения полей под названием BinaryField. Концепция аналогична рефлексии в Java и позволяет кэшировать определенную информацию о читаемом поле в экземпляре BinaryField, что полезно при чтении одного и того же поля из большой коллекции двоичных объектов.

```
Collection<BinaryObject> persons = getPersons();


BinaryField salary = null;

double total = 0;
int cnt = 0;

for (BinaryObject person : persons) {
  if (salary == null)
    salary = person.type().field("salary");

  total += salary.value(person);
  cnt++;
}

double avg = total / cnt;
```

2.6.4 Бинарный объект и хранилище кэша

Параметр withKeepBinary() в API кэша не влияет на способ передачи пользовательских объектов передаются в CacheStore. Это сделано специально, так как в большинстве случаев единственная реализация CacheStore работает либо с десериализованными классами, либо с представлениями BinaryObject. Для управления способом передачи объектов в хранилище следует использовать флаг storeKeepBinary в CacheConfiguration. Если для этого флага установлено значение false, в хранилище будут передаваться десериализованные значения, в противном случае будут использоваться представления BinaryObject.

Ниже приведен пример псевдокода реализации хранилища, работающего с BinaryObject:

```
public class CacheExampleBinaryStore extends CacheStoreAdapter<Integer, BinaryObject> {

  @IgniteInstanceResource
  private Ignite ignite;

  /** {@inheritDoc} */
  @Override public BinaryObject load(Integer key) {
    IgniteBinary binary = ignite.binary();

    List<?> rs = loadRow(key);

    BinaryObjectBuilder bldr = binary.builder("Person");

    for (int i = 0; i < rs.size(); i++)
      bldr.setField(name(i), rs.get(i));

    return bldr.build();
  }

  /** {@inheritDoc} */
  @Override public void write(Cache.Entry<? extends Integer, ? extends BinaryObject> entry) {
```

```
        BinaryObject obj = entry.getValue();

        BinaryType type = obj.type();

        Collection<String> fields = type.fieldNames();

        List<Object> row = new ArrayList<>(fields.size());

        for (String fieldName : fields)
          row.add(obj.field(fieldName));

        saveRow(entry.getKey(), row);
      }
    }
```

2.6.5 Маппер двоичных имен и маппер двоичных идентификаторов

Внутренне РЕД КВАНТ никогда не записывает полные строки для имен полей или типов. Вместо этого, из соображений производительности, РЕД КВАНТ записывает целочисленные хэш-коды для имен типов и полей. Тестирование показало, что конфликты хэш-кодов для имен типов или имен полей внутри одного типа практически отсутствуют, и для повышения производительности можно безопасно работать с хэш-кодами. В тех случаях, когда хэш-коды для разных типов или полей действительно конфликтуют, BinaryNameMapper и BinaryIdMapper поддерживают переопределение автоматически сгенерированных идентификаторов хэш-кодов для имен типов и полей.

BinaryNameMapper — сопоставляет имена типов/классов и полей с разными именами. BinaryIdMapper — сопоставляет данные из типа BinaryNameMapper и имени поля с идентификатором, который будет использоваться РЕД КВАНТ во внутренних компонентах.

РЕД КВАНТ предоставляет следующую готовую реализацию мапперов:

- BinaryBasicNameMapper — базовая реализация BinaryNameMapper, которая возвращает полное или простое имя данного класса в зависимости от значения свойства setSimpleName (boolean useSimpleName).

- BinaryBasicIdMapper — базовая реализация BinaryIdMapper. Если его свойство конфигурации setLowerCase (boolean isLowerCase) установлено как false, будет возвращен хэш-код заданного типа или имени поля. Если для свойства установлено значение true, будет возвращен хэш-код заданного типа или имени поля в нижнем регистре.

Если используются клиенты Java или .NET и не указываются мапперы в BinaryConfiguration, РЕД КВАНТ будет использовать BinaryBasicNameMapper, для свойства simpleName будет установлено значение false, а для BinaryBasicIdMapper и свойства lowerCase будет установлено значение true.

Если используются клиент C++ и не указываются мапперы в BinaryConfiguration, РЕД КВАНТ будет использовать BinaryBasicNameMapper, для свойства simpleName будет установлено значение true, а для BinaryBasicIdMapper и свойства lowerCase будет установлено значение true.

По умолчанию ничего настраивать не нужно, если используется Java, .NET или C++. При необходимости совместимости платформ мапперы должны быть настроены в случае сложного преобразования имен.

2.7 РАБОТА С БИНАРНЫМИ ОБЪЕКТАМИ

В РЕД КВАНТ данные хранятся в бинарном формате и десериализуются в объекты каждый раз, когда вызываются методы кэширования. Однако с бинарными объектами можно работать напрямую, избегая десериализации.

Бинарный объект — это оболочка хранящегося в кэше двоичного представления записи. Каждый бинарный объект имеет метод field(name), который возвращает значение заданного поля, и метод type(), который извлекает информацию о типе объекта. Бинарные объекты полезны, когда необходимо работать только с некоторыми полями объектов и не нужно десериализовать весь объект.

Для работы с бинарными объектами не нужно иметь определение класса, и можно динамически изменять структуру объектов без перезапуска кластера.

Формат бинарных объектов универсален для всех поддерживаемых платформ, т. е. Java, .NET и C++. Можно запустить кластер Java, затем подключиться к нему из клиентов .NET или C++ и использовать двоичные объекты на этих платформах без необходимости определять классы на стороне клиента.

Внимание:

Есть несколько ограничений, налагаемых реализацией формата бинарных объектов:

- Внутренне тип и поля бинарного объекта идентифицируются по их идентификаторам. Идентификаторы рассчитываются как хэш-коды соответствующих строковых имен. Следовательно, поля или типы с одинаковым хэшем имени не допускаются. Однако можно предоставить свою собственную реализацию генерации идентификатора через конфигурацию.

- По той же причине формат двоичных объектов не допускает одинаковых имен полей на разных уровнях иерархии классов.

- Если класс реализует внешний интерфейс Externalizable, РЕД КВАНТ использует OptimizedMarshaller вместо бинарного. OptimizedMarshaller использует методы writeExternal() и readExternal() для сериализации и десериализации объектов; поэтому класс должен быть добавлен в путь к классам серверных узлов.

2.7.1 ВКЛЮЧЕНИЕ БИНАРНОГО РЕЖИМА ДЛЯ КЭШЕЙ

По умолчанию, когда запрашиваются записи из кэша, они возвращаются в десериализованном формате. Для работы с бинарным форматом необходимо получить экземпляр кэша с помощью метода withKeepBinary(). Этот экземпляр возвращает объекты в двоичном формате (когда это возможно).

```
// Create a regular Person object and put it into the cache.

Person person = new Person(1, "FirstPerson");
ignite.cache("personCache").put(1, person);
```

```
// Get an instance of binary-enabled cache.
IgniteCache<Integer, BinaryObject> binaryCache = ignite.cache("personCache").withKeepBinary();
BinaryObject binaryPerson = binaryCache.get(1);
```

Следует обратить внимание, что не все объекты преобразуются в формат бинарных объектов. Следующие классы никогда не конвертируются (например, метод toBinary(Object) возвращает исходный объект, а экземпляры этих классов сохраняются без изменений):

- Все примитивы (byte, int и т. д.) и их классы-оболочки (Byte, Integer и т. д.);
- Массивы примитивов (byte[], int[], …);
- Строка и массив строк;
- UUID и массив UUID;
- Дата и массив дат;
- Timestamp и массив Timestamp;
- Перечисления и массив перечислений;
- Словари (Maps), коллекции и массивы объектов (но объекты внутри них переконвертируются, если они бинарные).

2.7.2 Создание и изменение двоичных объектов

Экземпляры бинарных объектов неизменяемыми. Чтобы обновить поля или создать новый двоичный объект, нужно воспользоваться конструктором двоичных объектов. Конструктор бинарных объектов — это служебный класс, который позволяет изменять поля бинарных объектов, не имея определения класса объектов.

Ограничения:

- Изменить типы существующих полей нельзя.
- Изменить порядок значений перечисления или добавить новые константы в начало или в середину списка значений перечисления нельзя. Однако можно добавить новые константы в конец списка.

Получить экземпляр конструктора двоичных объектов для определенного типа можно следующим образом:

```
BinaryObjectBuilder builder = ignite.binary().builder("org.apache.ignite.snippets.Person");
builder.setField("id", 2L);
builder.setField("name", "SecondPerson");
binaryCache.put(2, builder.build());
```

Конструкторы, созданные таким образом, не содержат полей. Добавить поля можно, вызвав метод setField(…).

Также конструктор бинарных объектов можно получить из существующего бинарного объекта через метод toBuilder(). В этом случае все значения полей копируются из бинарного объекта в конструктор.

В следующем примере используется обработчик записей для обновления объекта на серверном узле без необходимости развертывания класса объекта на этом узле и без полной десериализации объекта.

```
// The EntryProcessor is to be executed for this key.

int key = 1;
ignite.cache("personCache").<Integer, BinaryObject>withKeepBinary().invoke(key, (entry, arguments) -> {
  // Create a builder from the old value.
  BinaryObjectBuilder bldr = entry.getValue().toBuilder();

  //Update the field in the builder.
  bldr.setField("name", "Ignite");

  // Set new value to the entry.
  entry.setValue(bldr.build());

  return null;
});
```

2.7.3 Бинарный тип и бинарные поля

Бинарные объекты содержат информацию о типе объектов, которые они представляют. Информация о типе включает имена полей, типы полей и имя affinity-поля.

Тип каждого поля представлен объектом BinaryField. После получения объект BinaryField можно повторно использовать несколько раз, если нужно прочитать одно и то же поле из каждого объекта в коллекции. Повторное использование объекта BinaryField выполняется быстрее, чем чтение значения поля непосредственно из каждого двоичного объекта. Ниже приведен пример использования бинарного поля.

```
Collection<BinaryObject> persons = getPersons();


BinaryField salary = null;
double total = 0;
int count = 0;

for (BinaryObject person : persons) {
  if (salary == null) {
    salary = person.type().field("salary");
  }

  total += (float) salary.value(person);
  count++;
}

double avg = total / count;
```

2.7.4 Рекомендации по настройке бинарных объектов

РЕД КВАНТ хранит схему для каждого двоичного объекта заданного типа, в которой указаны поля, присутствующие в объекте, а также их порядок и типы. Схемы реплицируются на все узлы кластера. Бинарные объекты с одинаковыми полями, но в разном порядке,

считаются объектами, имеющими разные схемы. Рекомендуется всегда добавлять поля в бинарные объекты в одном и том же порядке.

Пустое поле обычно занимает пять байтов для хранения — четыре байта для идентификатора поля плюс один байт для длины поля. С точки зрения памяти предпочтительнее не включать поле, чем включать нулевое поле. Однако, если не включить поле, РЕД КВАНТ создаст новую схему для этого объекта, и эта схема будет отлична от схемы объектов, которые включают это поле. Если есть несколько полей, для которых в случайных комбинациях задано значение null, РЕД КВАНТ поддерживает разные схемы бинарных объектов для каждой комбинации, и heap может быть исчерпан из-за общего размера схем бинарных объектов. Лучше иметь несколько схем для бинарных объектов с одним и тем же набором полей одного типа, расположенных в одном и том же порядке, и при создании бинарного объекта выбрать одну из них, указав тот же набор полей, даже с нулевым значением. Это еще одна причина, по которой для пустого поля необходимо указать тип.

Также в объект можно вложить бинарные объекты, если есть необязательное или опциональное подмножество полей, которые но либо все отсутствуют, либо все присутствуют. Можно поместить их в отдельный BinaryObject, который либо хранится в поле родительского объекта, либо устанавливается как null.

Если есть большое количество полей, которые являются необязательными в любых комбинациях и очень часто пусты, их можно сохранить в поле словаря. Тогда будет несколько фиксированных полей в объекте значения и один словарь для дополнительных свойств.

2.7.5 НАСТРОЙКА ДВОИЧНЫХ ОБЪЕКТОВ

В подавляющем большинстве случаев нет необходимости настраивать бинарные объекты. Однако, если нужно изменить генерацию идентификаторов типа и поля или подключить собственный сериализатор, сделать это можно через конфигурацию.

Тип и поля бинарного объекта идентифицируются их идентификаторами. Идентификаторы рассчитываются как хэш-коды соответствующих строковых имен и хранятся в каждом двоичном объекте. Собственную реализацию генерации ID можно определить в конфигурации.

Преобразование имени в идентификатор выполняется в два этапа. Сначала имя типа (имя класса) или имя поля преобразуется словарем имен, затем маппер идентификаторов вычисляет идентификаторы. Можно указать глобальный словарь имен, глобальный словарь идентификаторов и глобальный двоичный сериализатор, а также словари и сериализаторы для каждого типа. Wildcards поддерживаются для конфигурации каждого типа, и в этом случае предоставленная конфигурация применяется ко всем типам, которые соответствуют шаблону имени типа.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <property name="binaryConfiguration">
    <bean class="org.apache.ignite.configuration.BinaryConfiguration">
      <property name="nameMapper" ref="globalNameMapper"/>
      <property name="idMapper" ref="globalIdMapper"/>
      <property name="typeConfigurations">
```

```
            <list>
              <bean class="org.apache.ignite.binary.BinaryTypeConfiguration">
                <property name="typeName" value="org.apache.ignite.examples.*"/>
                <property name="serializer" ref="exampleSerializer"/>
              </bean>
            </list>
          </property>
        </bean>
    </property>

</bean>
```

3 РЕБАЛАНС ДАННЫХ

Когда новый узел присоединяется к кластеру, некоторые разделы перемещаются на новый узел, чтобы данные оставались равномерно распределенными в кластере. Этот процесс называется ребалансировкой данных.

Если существующий узел навсегда покидает кластер, а резервные копии не настроены, хранящиеся на этом узле разделы теряются. Когда резервное копирование настроено, одна из резервных копий потерянных разделов становится основным разделом, и инициируется процесс ребалансировки.

Внимание: Ребалансировка данных инициируется изменениями в базовой топологии. В in-memory кластерах по умолчанию ребалансировка начинается немедленно, когда узел покидает кластер или присоединяется к нему (базовая топология изменяется автоматически). В кластерах с персистентностью базовая топология должна быть изменена вручную (поведение по умолчанию) или может быть изменена автоматически, если включена автоматическая корректировка базовой топологии.

Ребалансировка настраивается для каждого кэша.

3.1.1 НАСТРОЙКА РЕЖИМА РЕБАЛАНСИРОВКИ

РЕД КВАНТ поддерживает как синхронную, так и асинхронную ребалансировку. В синхронном режиме любые операции с данными кэша блокируются до завершения ребалансировки. В асинхронном режиме процесс ребалансировки выполняется асинхронно. Также можно отключить ребалансировку для определенного кэша.

Чтобы изменить режим ребалансировки, необходимо установить одно из следующих значений в конфигурации кэша:

- SYNC — режим синхронной ребалансировки. В этом режиме любой вызов общедоступного API кэша блокируется до завершения ребалансировки.
- ASYNC — режим асинхронной ребалансировки. Распределенные кэши доступны сразу и загружают все необходимые данные с других доступных узлов кластера в фоновом режиме.
- NONE — в этом режиме ребалансировка не выполняется, что означает, что кэши либо загружаются по запросу или требованию из постоянного хранилища при каждом доступе к данным, либо заполняются явно.

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util.xsd">

  <bean class="org.apache.ignite.configuration.IgniteConfiguration" id="ignite.cfg">
    <property name="cacheConfiguration">
      <list>
        <bean class="org.apache.ignite.configuration.CacheConfiguration">
          <property name="name" value="mycache"/>
          <!-- enable synchronous rebalance mode -->
```

```
            <property name="rebalanceMode" value="SYNC"/>
          </bean>
        </list>
      </property>
    </bean>
  </beans>
```

3.1.2 НАСТРОЙКА ПУЛА ПОТОКОВ РЕБАЛАНСИРОВКИ

По умолчанию ребалансировка выполняется в один поток на каждом узле. Это означает, что в каждый момент времени только один поток используется для передачи пакетов от одного узла к другому или для обработки пакетов, поступающих с remote-узла.

Можно увеличить количество потоков, которые берутся из пула системных потоков и используются для ребалансировки. Системный поток берется из пула каждый раз, когда узлу необходимо отправить пакет данных на удаленный узел или обработать пакет, пришедший с удаленного узла. Поток освобождается после обработки пакета.

```
    <beans xmlns="http://www.springframework.org/schema/beans"
xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="  http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util.xsd">

      <bean class="org.apache.ignite.configuration.IgniteConfiguration" id="ignite.cfg">

        <property name="rebalanceThreadPoolSize" value="4"/>

        <property name="cacheConfiguration">
          <list>
            <bean class="org.apache.ignite.configuration.CacheConfiguration">
              <property name="name" value="mycache"/>
            </bean>
          </list>
        </property>
      </bean>
    </beans>
```

Внимание: Пул системных потоков широко используется внутри всех операций, связанных с кэшем (put, get и т. д.), механизма SQL и других модулей. Установка большого значения для размера пула потоков перебалансировки может значительно повысить производительность перебалансировки за счет снижения пропускной способности.

3.1.3 РЕБАЛАНСИРОВКА РЕГУЛИРОВАНИЯ СООБЩЕНИЙ

Когда данные передаются с одного узла на другой, весь набор данных разбивается на пакеты, и каждый пакет отправляется в отдельном сообщении. Можно настроить размер пакета и время ожидания узла между сообщениями.

```
    <beans xmlns="http://www.springframework.org/schema/beans"
xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="  http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util.xsd">
```

```
  <bean class="org.apache.ignite.configuration.IgniteConfiguration" id="ignite.cfg">
    <property name="cacheConfiguration">
      <list>
        <bean class="org.apache.ignite.configuration.CacheConfiguration">
          <property name="name" value="mycache"/>
          <!-- Set batch size. -->
          <property name="rebalanceBatchSize" value="#{2 * 1024 * 1024}"/>
          <!-- Set throttle interval. -->
          <property name="rebalanceThrottle" value="100"/>
        </bean>
      </list>
    </property>
  </bean>
</beans>
```

В таблице 1 перечислены свойства CacheConfiguration, связанные с ребалансировкой.

Внимание! rebalanceDelay и связанные API устарели и будут удалены в следующих релизах.

Таблица 1 - Свойства CacheConfiguration

Свойство	Описание	Значение по умолчанию
rebalanceDelay	Задержка в миллисекундах перед началом процесса ребалансировки после присоединения узла к топологии или выхода из нее. Задержка ребалансировки полезна, если планируется перезапускать узлы или запускать несколько узлов одновременно или один за другим и нет необходимости перераспределять и ребалансировать данные до тех пор, пока не будут запущены все узлы.	0 (без задержки)
rebalanceBatchSize	Размер в байтах одного сообщения ребалансировки. Алгоритм ребалансировки разбивает данные на каждом узле на несколько пакетов перед отправкой на другие узлы.	512 КБ
rebalanceThrottle	Размер пакета и throttle interval узла между сообщениями.	0 (троттлинг отключен)
rebalanceOrder	Порядок выполнения ребалансировки. Порядок ребалансировки может быть установлен на ненулевое значение только для кэшей с режимами ребалансировки SYNC или ASYNC. Сначала выполняется ребалансировка кэшей с меньшим порядком ребалансировки. По умолчанию порядок ребалансировки не назначен.	0

Свойство	Описание	Значение по умолчанию
rebalanceTimeout	Тайм-аут ожидания сообщений о ребалансировке при обмене ими между узлами.	10 секунд

4 ПОТОКОВАЯ РАБОТА С ДАННЫМИ

РЕД КВАНТ предоставляет API потоковой работы с данными, который можно использовать для передачи больших объемов непрерывных потоков данных в кластер РЕД КВАНТ. API потоковой работы с данными спроектирован масштабируемым и отказоустойчивым и обеспечивает семантику доставки «хотя бы один раз».

Данные передаются потоком в кэш через data streamer, связанный с кэшем. Data streamer автоматически буферизует данные и группирует их в пакеты для повышения производительности, а также отправляют параллельно нескольким узлам.

API потоковой работы с данными предоставляет следующие функции (Рисунок 19):

- Добавляемые в data streamer данные автоматически разделяются и распределяются между узлами.
- Возможна параллельная обработка данных с учетом их колокации.
- Клиенты могут выполнять параллельные SQL-запросы к данным по мере их поступления.

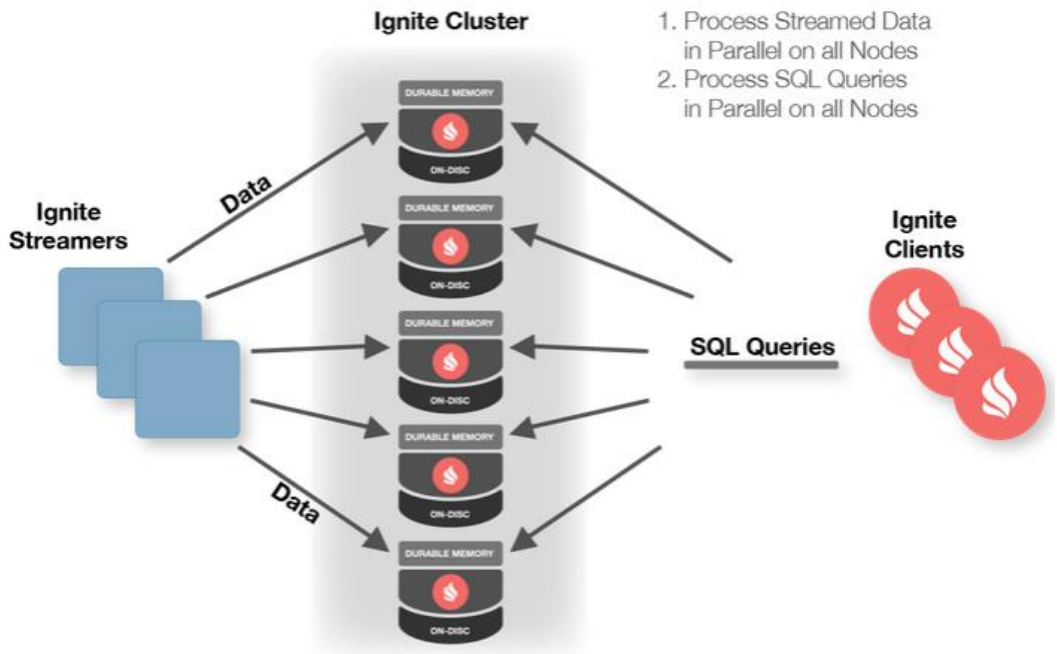

Рисунок 19 - Потоковая передача данных

Data streamer связан с определенным кэшем и предоставляет интерфейс для потоковой передачи данных в кэш.

Обычно при получении data streamer использует один из методов для потоковой передачи данных в кэш, а РЕД КВАНТ заботится о партиционировании и колокации данных во избежание ненужного перемещения данных, объединяя записи в пакеты в соответствии с правилами партиционирования.

Получить data streamer для определенного кэша можно следующим образом:

```
// Get the data streamer reference and stream data.
try (IgniteDataStreamer<Integer, String> stmr = ignite.dataStreamer("myCache")) {
```

```
    // Stream entries.
    for (int i = 0; i < 100000; i++)
        stmr.addData(i, Integer.toString(i));
}
System.out.println("dataStreamerExample output:" + cache.get(99999));
```

В версии РЕД КВАНТ для Java data streamer является реализацией интерфейса IgniteDataStreamer. IgniteDataStreamer предоставляет ряд методов addData(...) для добавления пар «ключ-значение» в кэши.

4.1.1 Перезапись существующих ключей

По умолчанию data streamer не перезаписывают существующие данные и пропускают записи, которые уже находятся в кэше. Это поведение можно изменить, задав для свойства data streamer allowOverwrite значение true.

```
stmr.allowOverwrite(true);
```

Примечание: Если для параметра allowOverwrite установлено значение false (по умолчанию), обновления не распространяются на внешнее хранилище (если таковое используется).

4.1.2 Обработка данных

В тех случаях, когда нужно выполнить пользовательскую логику перед добавлением новых данных, можно использовать Stream receiver. Stream receiver используется для сколлоцированной обработки данных перед их сохранением в кэше. Реализованная в Stream receiver логика выполняется на узле, где должны храниться данные.

Примечание: Stream receiver не помещает данные в кэш автоматически. Для этого необходимо явно вызвать один из методов put(…).

```
try (IgniteDataStreamer<Integer, String> stmr = ignite.dataStreamer("myCache")) {
    stmr.allowOverwrite(true);
    stmr.receiver((StreamReceiver<Integer, String>) (cache, entries) -> entries.forEach(entry -> {
        // do something with the entry
        cache.put(entry.getKey(), entry.getValue());
    }));
}
```

Внимание! Определения классов Stream receiver, которые должны выполняться на remote узлах, должны быть доступны на узлах. Этого можно достичь двумя способами:

- Добавить классы в classpath узлов;
- Включить загрузку однорангового класса.

4.1.2.1 Stream Transformer

Stream Transformer — это удобная реализация stream receiver, который обновляет данные в стриме. Stream Transformer используют функцию колокации и обновляют данные на узле, где они будут храниться.

В приведенном ниже примере используется Stream Transformer для увеличения счетчика каждого отдельного слова, найденного в текстовом потоке.

```
String[] text = { "hello", "world", "hello", "Ignite" };
CacheConfiguration<String, Long> cfg = new CacheConfiguration<>("wordCountCache");
IgniteCache<String, Long> stmCache = ignite.getOrCreateCache(cfg);

try (IgniteDataStreamer<String, Long> stmr = ignite.dataStreamer(stmCache.getName())) {
  // Allow data updates.
  stmr.allowOverwrite(true);

  // Configure data transformation to count instances of the same word.
  stmr.receiver(StreamTransformer.from((e, arg) -> {
    // Get current count.
    Long val = e.getValue();

    // Increment count by 1.
    e.setValue(val == null ? 1L : val + 1);

    return null;
  }));

  // Stream words into the streamer cache.
  for (String word : text)
    stmr.addData(word, 1L);
}
```

4.1.2.2 Stream Visitor

Stream Visitor — это еще одна реализация stream receiver, которая посещает каждую пару «ключ-значение» в стриме. Stream Visitor не обновляет кэш. Если пару необходимо сохранить в кэше, один из методов put(…) должен вызываться явно.

В приведенном ниже примере есть 2 кэша: «marketData» и «instruments». Полученные рыночные данные помещаются в стример для кэша «marketData». Stream Visitor для стримера «marketData» вызывается на элементе кластера, сопоставленного с конкретным рыночным символом. При получении отдельных рыночных тиков он обновляет кэш «инструмента» последней рыночной ценой.

Следует обратить внимание, что кэш «marketData» не обновляется, оставаясь пустым. Он просто используется для колоцированной обработки рыночных данных непосредственно на узле кластера, где хранятся данные.

```
static class Instrument {

  final String symbol;
  Double latest;
  Double high;
  Double low;

  public Instrument(String symbol) {
    this.symbol = symbol;
  }

}

static Map<String, Double> getMarketData() {
  //populate market data somehow
```

```
    return new HashMap<>();
}

@Test
void streamVisitorExample() {
    try (Ignite ignite = Ignition.start()) {
        CacheConfiguration<String, Double> mrktDataCfg = new CacheConfiguration<>("marketData");
        CacheConfiguration<String, Instrument> instCfg = new CacheConfiguration<>("instruments");

        // Cache for market data ticks streamed into the system.
        IgniteCache<String, Double> mrktData = ignite.getOrCreateCache(mrktDataCfg);

        // Cache for financial instruments.
        IgniteCache<String, Instrument> instCache = ignite.getOrCreateCache(instCfg);

        try (IgniteDataStreamer<String, Double> mktStmr = ignite.dataStreamer("marketData")) {
            // Note that we do not populate the 'marketData' cache (it remains empty).
            // Instead we update the 'instruments' cache based on the latest market price.
            mktStmr.receiver(StreamVisitor.from((cache, e) -> {
                String symbol = e.getKey();
                Double tick = e.getValue();

                Instrument inst = instCache.get(symbol);

                if (inst == null)
                    inst = new Instrument(symbol);

                // Update instrument price based on the latest market tick.
                inst.high = Math.max(inst.high, tick);
                inst.low = Math.min(inst.low, tick);
                inst.latest = tick;

                // Update the instrument cache.
                instCache.put(symbol, inst);
            }));

            // Stream market data into the cluster.
            Map<String, Double> marketData = getMarketData();
            for (Map.Entry<String, Double> tick : marketData.entrySet())
                mktStmr.addData(tick);
        }
    }
}
```

4.1.3 НАСТРОЙКА РАЗМЕРА ПУЛА ПОТОКОВ DATA STREAMER

Пул потоков Data Streamer предназначен для обработки сообщений, поступающих от Data Streamers.

Размер пула по умолчанию — max(8, общее количество ядер). Для изменения размера пула необходимо использовать IgniteConfiguration.setDataStreamerThreadPoolSize(…).

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
    <property name="dataStreamerThreadPoolSize" value="10"/>

    <!-- other properties -->
```

```
</bean>
```

5 РАБОТА С БАЗОЙ ДАННЫХ

РЕД КВАНТ может использоваться как база данных в памяти или полнофункциональная распределенная база данных с надежностью диска и высокой согласованностью. Хранилище РЕД КВАНТ, ориентированное на память, помогает хранить данные и индексы как в памяти, так и на диске в одной и той же структуре данных. Оно позволяет выполнять SQL-запросы к данным, хранящимся в памяти и на диске, и обеспечивает оптимальную производительность при минимальных затратах на инфраструктуру. Механизм сохранения данных РЕД КВАНТ называется Native Persistence и является опциональным. Когда он включен, РЕД КВАНТ может хранить больше данных, чем может поместиться в доступной памяти, и действовать как полноценная распределенная база данных SQL. С другой стороны, когда весь набор данных и индексы помещаются в память, а Native Persistence отключен, РЕД КВАНТ функционирует как база данных в памяти, поддерживающая SQL, вместе со всеми существующими API для случаев использования только в памяти.

Кроме того, РЕД КВАНТ поддерживает Spring Data, что позволяет абстрагировать базовое хранилище данных от прикладного уровня. Spring Data также позволяет использовать готовые CRUD-операции для доступа к базе данных вместо написания стандартного кода. Кроме того, можно использовать Hibernate OGM, который обеспечивает поддержку JPA (Java Persistence API) для базы данных РЕД КВАНТ.

Еще одной выдающейся обязательной функцией для любой системы корпоративного класса является транзакция. Поддержка транзакций ACID гарантирует, что после внесения изменений в набор данных они будут корректными и ничего не будет потеряно. Современному предприятию часто требуется транзакционная система с низкой задержкой и аналитическими возможностями и с возможностью проведения транзакций. Этот тип гибридной транзакционной обработки часто называют HTAP (гибридная транзакционная и аналитическая обработка). HTAP позволяет обрабатывать анализ самых последних данных в режиме реального времени. В HTAP данные не нужно перемещать из рабочей базы данных в отдельные витрины данных. В такой системе поддержка SQL играет жизненно важную роль. Де-факто SQL — это естественный язык для анализа данных и продуктивный язык для написания запросов. Поэтому в этой главе будут рассмотрены следующие темы:

1. Возможности запросов РЕД КВАНТ.

2. Интеграция Spring Data.

3. Использование Hibernate OGM с РЕД КВАНТ.

4. Распределенные транзакции.

5. Персистентность РЕД КВАНТ (собственная персистентность (Native Persistence) и сторонняя).

Однако, прежде чем начать, может быть полезно знать, что:

• Каждое приложение, представленное в этой главе, можно загрузить из репозитория GitHub.

• Не нужно реконструировать каждое приложение, если в этом нет необходимости.

5.1 Таблицы и индексы

Под капотом РЕД КВАНТ хранит данные в кэше. Таблицы и индексы, созданные в РЕД КВАНТ, представляют собой что-то вроде метаданных над записями кэша, которые позволяют выполнять SQL-запросы к данным, хранящимся в кэшах РЕД КВАНТ. Настроить таблицы и индексы в РЕД КВАНТ можно несколькими способами:

1. DDL-операторы. РЕД КВАНТ поддерживает операторы Data Definition Language (DDL для создания и удаления таблиц и индексов SQL во время выполнения. На данный момент РЕД КВАНТ предоставляет следующие операторы DDL:

- CREATE TABLE. Создать новую таблицу и лежащий в её основе кэш РЕД КВАНТ. Кэш с уникальным именем будет создан в следующем формате: SQL_{ИМЯ_СХЕМЫ}_{ТАБЛИЦА}, где ИМЯ_СХЕМЫ всегда будет PUBLIC. Например, если создать таблицу с именем EMP, будет также создан кэш с именем SQL_PUBLIC_EMP, и можно будет использовать кэш для работы с ней, используя «ключ-значение», вычисления или другие API, доступные в РЕД КВАНТ. Стоит отметить, что имя кэша, созданного оператором DDL, можно изменить.
- CREATE INDEX. Создать индекс для указанной таблицы. Индексы хранятся во внутренних структурах данных B+tree. B+tree распределяется по кластеру вместе с фактическими данными. Узел кластера хранит часть индекса для данных, которыми он владеет.
- ALTER TABLE. Изменить структуру существующей таблицы. Можно добавить или удалить несколько столбцов из ранее созданной таблицы.
- DROP INDEX. Удалить индекс для указанной таблицы.
- DROP TABLE. Удалить указанную таблицу и лежащий в её основе кэш. Эта команда удаляет существующую таблицу в РЕД КВАНТ. Её распределенный кэш со всеми данными в нем также будет уничтожен.
- CREATE USER. Создать пользователя с заданным именем и паролем.
- ALTER USER. Изменить существующий пароль пользователя.
- DROP USER. Удалить существующего пользователя.

2. На основе аннотаций. В дополнение к командам DDL также можно аннотировать поля записей кэша для SQL-запросов. Индексы, а также запрашиваемые поля также можно настроить из кода Java с использованием аннотации @QuerySqlField. Все поля в классе Java, которые будут задействованы в запросах SQL, должны иметь эту аннотацию.

```
public class Employee implements Serializable {
/** Indexed field. Will be visible for SQL engine. */
@QuerySqlField(index = true)
private long empno;

/** Queryable field. Will be visible for SQL engine. */
@QuerySqlField
private String ename;
```

```
/** Will NOT be visible for SQL engine. */
private int deptno;

/**
* Indexed field sorted in descending order.
* Will be visible for SQL engine.
*/
@QuerySqlField(index = true, descending = true)
privatefloatsal;
}
```

В приведенном выше коде и *emno*, и *sal* являются индексированными полями. Если индексировать поле не нужно, но в SQL-запросе оно необходимо, то его также необходимо аннотировать, опустив параметр index = true. Такое поле называется запрашиваемым полем. Например, *ename* определено выше как запрашиваемое поле. Поле *deptno* не является ни запрашиваемым, ни индексируемым полем и не будет доступно из SQL-запросов в РЕД КВАНТ.

3. Конфигурация на основе сущности запроса. Также можно использовать XML-файл Spring для настройки индексов и запрашиваемых полей для кэшей. В этом подходе необходимо использовать класс QueryEntity, который представляет собой описание записи кэша (состоящей из ключа и значения) с точки зрения того, как она должна быть проиндексирована и может быть запрошена. Все концепции, которые обсуждались как часть описанной выше конфигурации на основе аннотаций, также действительны для подхода на основе QueryEntity. Кроме того, поля, настроенные с помощью аннотации @QuerySqlField, внутренне превращаются в сущности запроса. В приведенном ниже примере показано, как определить поле запроса и индексы с помощью конфигурации Spring XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<bean class="org.apache.ignite.configuration.CacheConfiguration">
  <property name="name" value="EMP" />
  <!--Configure query entities-->
  <property name="queryEntities">
    <list>
      <bean class="org.apache.ignite.cache.QueryEntity">
        <!--Setting indexed type's key class-->
        <property name="keyType" value="java.lang.Long" />
        <!--Key field name to be used in INSERT and SELECT queries-->
        <property name="keyFieldName" value="empno"/>
        <!--Setting indexed type's value class-→
        <property name="valueType" value="com.blu.imdg.Empoyee"/>
        <!--Defining fields that will be either indexed or queryable.
            Indexed fields are added to 'indexes' list below.-->
        <property name="fields">
          <map>
            <entry key="ename" value="java.lang.String"/>
            <entry key="sal" value="java.lang.Long " />
          </map>
        </property>
        <!--Defining indexed fields.-->
        <property name="indexes">
          <list>
            <!--Single field (aka. column) index-->
            <bean class="org.apache.ignite.cache.QueryIndex">
```

```
            <constructor-arg value="ename" />
          </bean>
        </list>
      </property>
    </bean>
  </list>
 </property>
</bean>
```

5.2 Запросы к БД

Существует множество способов запросить хранящиеся в кэшах данные после создания необходимых таблиц или настройки кэшей для запроса данных. РЕД КВАНТ предоставляет несколько элегантных способов, таких как «запросы сканирования» (Scan queries), SQL-запросы (ANSI-99) и текстовые запросы для извлечения данных из кэшей РЕД КВАНТ. Но возникает вопрос - почему РЕД КВАНТ предоставляет API, такие как SQL-запросы к хранилищам «ключ-значение»? Потому что получение данных с помощью ключа имеет некоторые ограничения: нужно обрабатывать запросы данных вручную на уровне приложения, и отсутствие интеграции с инструментами анализа данных. Поэтому для устранения этих недостатков РЕД КВАНТ предоставляет декларативный язык программирования для запроса данных, такой как SQL. Таким образом, SQL стал универсальным интерфейсом для извлечения и анализа данных.

Однако, есть несколько моментов, которые нужно учитывать при запросах РЕД КВАНТ:

1. Внутренне РЕД КВАНТ хранит данные в структуре «ключ-значение» в кэшах.

2. РЕД КВАНТ загружает данные в память перед выполнением запросов.

3. РЕД КВАНТ запускает механизм in-memory базы данных на основе H2 на каждом узле, чтобы выполнить запрос.

4. Можно писать SQL-подобные запросы для запроса данных по интересующим полям или для извлечения всего объекта.

5. Можно использовать как собственный API-интерфейс РЕД КВАНТ SQL, так и драйверы JDBC или ODBC для выполнения запросов SQL.

Обобщая классификацию, обратиться к базе данных РЕД КВАНТ можно следующими способами:

1. SQL-запросы: SQL-запросы через JDBC/ODBC.

2. РЕД КВАНТ SQL API: специальные или собственные SQL API (SQLQuery и SqlFieldsQuery).

3. Запросы кэша: «запросы сканирования» (Scan queries) на основе предикатов и запросы текстового поиска.

5.3 Уровень хранения

In-memory подходы позволяют достичь молниеносной скорости за счет размещения рабочего набора данных в системной памяти. Когда все данные хранятся в памяти, отпадает необходимость решать вопросы, возникающие при использовании традиционных вращающихся HDD жёстких дисков. Это означает, например, что нет необходимости поддерживать дополнительные кэшированные копии данных и управлять синхронизацией между ними. Однако у этого подхода есть и обратная сторона: поскольку данные находятся только в памяти, они не сохранятся, если весь кластер будет остановлен. Следовательно, эти типы хранилищ данных вообще не считаются постоянными.

В большинстве случаев не следует хранить в памяти весь набор данных для вашего приложения, чаще всего вам следует хранить относительно небольшое или активное подмножество данных для повышения производительности приложения. Остальные данные следует хранить где-нибудь на недорогих дисках или лентах для архивирования. Существуют два основных требования к хранению базы данных в оперативной памяти:

- Постоянный носитель - для хранения совершенных транзакций. Таким образом поддерживается надежность и возможность восстановления, если базу данных в памяти необходимо перезагрузить в память.

- Постоянное хранилище - для хранения резервной копии всей базы данных в оперативной памяти.

Постоянным хранилищем или носителем может быть любая распределенная или локальная файловая система, SAN, база данных NoSQL или даже СУБД, такая как Postgres или Oracle. РЕД КВАНТ предоставляет два элегантных способа сохранения данных:

1. Сохранение в сторонней базе данных. РЕД КВАНТ предоставляет API для подключения постоянных хранилищ данных, таких как СУБД или базы данных NoSQL, такие как Mongo DB или Cassandra, для хранения данных. Чаще всего узким местом в СУБД будет слой хранения, и горизонтально масштабировать систему не выйдет.

2. Собственная персистентность РЕД КВАНТ. РЕД КВАНТ предоставляет дисковые хранилища, совместимые с ACID и SQL, которые прозрачно интегрируются с долговременной памятью РЕД КВАНТ в качестве дополнительного дискового уровня для хранения данных и индексов на SSD, флэш-памяти, 3D XPoint и других типах энергонезависимых систем хранения.

5.3.1 Собственная персистентность

Собственная персистентность РЕД КВАНТ использует новую надежную архитектуру памяти, которая позволяет хранить и обрабатывать данные и индексировать как в памяти, так и на диске. Всякий раз, когда эта функция включена, РЕД КВАНТ сохраняет расширенный набор данных на диске и подмножество данных в оперативной памяти в зависимости от ее емкости. Если в оперативной памяти отсутствует подмножество данных или индекс, долговременная память берет их с диска, как показано на рисунке 20.

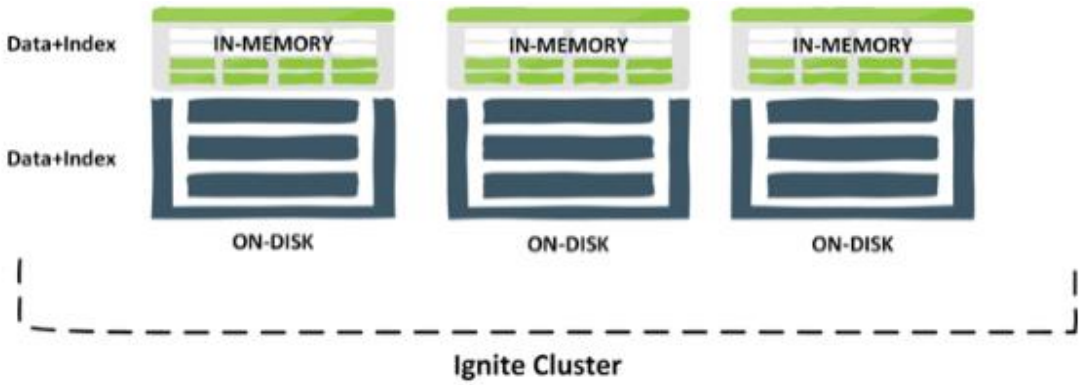

Рисунок 20 - Структура памяти РЕД КВАНТ с собственной сохраняемостью

Данные также могут храниться в центральном дисковом хранилище, к которому подключены все узлы РЕД КВАНТ.

В этом пункте представлен пример запуска узла РЕД КВАНТ с включенной собственной персистентностью и описано, как данные хранятся в файловой системе. Предварительные условия проекта в песочнице:

Таблица 2 - Предварительные условия

Имя	**Описание**
РЕД КВАНТ	>=2.6.0
JVM	1.8
OS	*nix

Шаг 1. Необходимо загрузить бинарный дистрибутив РЕД КВАНТ и разархивировать его в песочнице. Этот шаг можно пропустить, если уже есть установленная копия РЕД КВАНТ в системе.

Шаг 2. Необходимо передать экземпляр DataStorageConfiguration в конфигурацию узла кластера, чтобы включить собственную персистентность РЕД КВАНТ. РЕД КВАНТ DataStorageConfiguration выделяет одну расширяемую область данных с настройками по умолчанию. Он сопоставит настроенные в приложении кэши с этой областью данных. Персистентность для области данных можно включить с помощью флага DataRegionConfiguration.setPersistenceEnabled (boolean). Предварительно настроенную конфигурацию DataStorageConfiguration можно найти в файле example-persistent-store.xml, который поставляется с дистрибутивом РЕД КВАНТ.

Шаг 3. Необходимо запустить следующую команду из каталога IGNITE_HOME.

```
ignite.sh$IGNITE_HOME/examples/config/persistentstore/example-persistent-store.xml
```

Она должна запустить узел РЕД КВАНТ и создать новую папку с именем WORK в каталоге IGNITE_HOME. Сообщения журнала на консоли сервера РЕД КВАНТ также должны подтверждать, что сервер запущен с включенной персистентностью, как показано на рисунке 21.

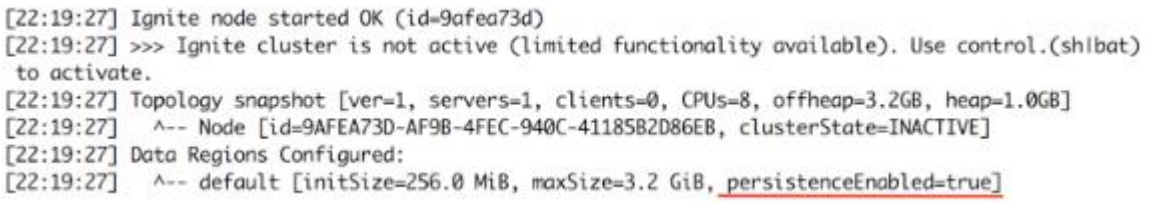

Рисунок 21 - Сообщение журнала о запуске сервера с включенной персистентностью

РЕД КВАНТ также предупредил, что кластер еще не активирован, и нужно активировать его с помощью скрипта control.sh.

Шаг 4. Теперь узел РЕД КВАНТ запущен и работает с включенной персистентностью в НЕАКТИВНОМ режиме. Необходимо активировать кластер с помощью скрипта control.sh|bat, поставляемого с дистрибутивом РЕД КВАНТ. Ввести следующую команду в любом отдельном терминале.

```
control.sh --activate
```

Если команда выполнена успешно, в консоли будет соответствующее сообщение (рисунок 22).

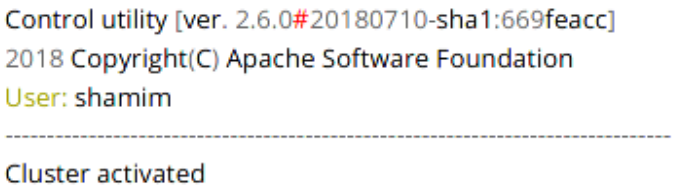

Рисунок 22 - Сообщение об успешной активации кластера

В этот момент также можно использовать команду --state для проверки текущего состояния кластера. Команда --state должна вернуть сообщение о том, что кластер активирован. Подробную информацию об управлении и мониторинге кластера РЕД КВАНТ см. в главах 139 Управление кластером и 155 Мониторинг работы кластера.

Шаг 5. Теперь необходимо создать таблицу и заполнить ее данными. Для подключения к кластеру можно воспользоваться инструментом SQLLINE. Чтобы запустить инструмент SQLLINE, необходимо выполнить следующую команду:

```
sqlline.sh --color=true--verbose=true-u jdbc:ignite:thin://127.0.0.1/
```

Для создания таблицы EMP можно воспользоваться следующим DDL скриптом следующим образом:

```
CREATE TABLE IF NOT EXISTS EMP
(
    empno LONG,
    ename VARCHAR,
    job VARCHAR,
    mgr INTEGER,
    hiredate DATE,
    sal  LONG,
    comm  LONG,
    deptno LONG,
    CONSTRAINT pk_emp PRIMARY KEY (empno)
) WITH "template=partitioned,CACHE_NAME=EMPcache";
```

Вставить несколько строк в таблицу EMP можно следующим образом:

```
INSERT INTO emp
        (empno,
        ename,
        job,
        mgr,
        hiredate,
        sal,
        comm,
        deptno)
VALUES(7839,
        'KING',
        'PRESIDENT',
        NULL,
        To_date('17-11-1981','dd-mm-yyyy'),
        5000,
        NULL,
        10);

INSERT INTO emp
        (empno,
        ename,
        job,
        mgr,
        hiredate,
        sal,
        comm,
        deptno)
VALUES (7698,
        'BLAKE',
        'MANAGER',
        7839,
        To_date('1-5-1981','dd-mm-yyyy'),
        2850,
        NULL,
        30);

INSERT INTO emp
        (empno,
        ename,
        job,
        mgr,
        hiredate,
        sal,
        comm,
        deptno)
VALUES (7782,
        'CLARK',
        'MANAGER',
        7839,
```

```
To_date('9-6-1981','dd-mm-yyyy'),
2450,
NULL,
10);
```

Шаг 6. Далее необходимо воспользоваться инструментом командной строки ignitevisor для сканирования EMPCache. Для этого нужно использовать команду thecache-scan в ignitevisor. Результат в консоли представлен на рисунке 23. Все три элемента в кэше:

```
visor> cache -scan
Time of the snapshot: 2018-12-20 20:38:04
+=============================================================+
| # |    Name(@)    |    Mode     | Size (Heap / Off-heap) |
+=============================================================+
| 0 | EMPcache(@c0) | PARTITIONED | min: 3 (0 / 3)         |
|   |               |             | avg: 3.00 (0.00 / 3.00) |
|   |               |             | max: 3 (0 / 3)         |
+-------------------------------------------------------------+

Choose cache number ('c' to cancel) [c]: 0
Entries in  cache: EMPcache
+==========================================================================
==========================================================================+
|   Key Class    | Key |              Value Class              |
                      Value                                                |
+==========================================================================
==========================================================================+
| java.lang.Long | 7698 | o.a.i.i.binary.BinaryObjectImpl | SQL_PUBLIC_EMP_0bd87eaf_b503_4b06_83b6_25be1cdb893c
 [hash=-1417502761, ENAME=BLAKE, JOB=MANAGER, MGR=7839, HIREDATE=1981-05-01, SAL=2850, COMM=null, DEPTNO=30] |
| java.lang.Long | 7782 | o.a.i.i.binary.BinaryObjectImpl | SQL_PUBLIC_EMP_0bd87eaf_b503_4b06_83b6_25be1cdb893c
 [hash=1453774085, ENAME=CLARK, JOB=MANAGER, MGR=7839, HIREDATE=1981-06-09, SAL=2450, COMM=null, DEPTNO=10]  |
| java.lang.Long | 7839 | o.a.i.i.binary.BinaryObjectImpl | SQL_PUBLIC_EMP_0bd87eaf_b503_4b06_83b6_25be1cdb893c
 [hash=1848897643, ENAME=KING, JOB=PRESIDENT, MGR=null, HIREDATE=1981-11-17, SAL=5000, COMM=null, DEPTNO=10] |
```

Рисунок 23 - Результат сканирования EMPCache

Шаг 7. Теперь можно узнать, что происходит внутри. Необходимо запустить следующую команду из каталога IGNITE_HOME/work:

```
du -h .
```

Команда должна вернуть что-то подобное тому, что показано на рисунке 24.

```
shamim:work shamim$ du -h .
8.0K    ./binary_meta/node00-46ef2525-eef7-4cd1-9b14-b81a1eb9cd1b
8.0K    ./binary_meta
144K    ./db/node00-46ef2525-eef7-4cd1-9b14-b81a1eb9cd1b/cache-EMPcache
 32K    ./db/node00-46ef2525-eef7-4cd1-9b14-b81a1eb9cd1b/cache-ignite-sys-cache
 40K    ./db/node00-46ef2525-eef7-4cd1-9b14-b81a1eb9cd1b/cp
 60K    ./db/node00-46ef2525-eef7-4cd1-9b14-b81a1eb9cd1b/metastorage
280K    ./db/node00-46ef2525-eef7-4cd1-9b14-b81a1eb9cd1b
  0B    ./db/wal/archive/node00-46ef2525-eef7-4cd1-9b14-b81a1eb9cd1b
  0B    ./db/wal/archive
 64M    ./db/wal/node00-46ef2525-eef7-4cd1-9b14-b81a1eb9cd1b
 64M    ./db/wal
 64M    ./db
 92K    ./log
8.0K    ./marshaller
 64M    .
```

Рисунок 24 - Проверка происходящего под капотом

Если пройти по директории db/node00-46ef2525-eef7-4cd1-9b14-b81a1eb9cd1b, можно найти отдельную папку для каждого кэша. Папка с именем cache-EMPcache должна содержать все записи кэша (3 элемента), которые только что были добавлены.

```
shamim:cache-EMPcache shamim$ ls -la
total 288
-rw-r--r--  1 shamim  staff   5211 22:41 cache_data.dat
-rw-r--r--  1 shamim  staff  40960 20:35 index.bin
-rw-r--r--  1 shamim  staff  32768 20:35 part-530.bin
-rw-r--r--  1 shamim  staff  32768 20:35 part-614.bin
-rw-r--r--  1 shamim  staff  32768 20:35 part-671.bin
```

Рисунок 25 - Просмотр содержимого директории

Файл index.bin является индексным файлом записей кэша, и каждый элемент кэша получает свой отдельный раздел или файл страницы (*.bin). Архитектура РЕД КВАНТ — это архитектура, основанная на страницах, а страница — это наименьшая единица данных с фиксированным размером. Страницы хранятся вне heap Java и организованы в оперативной памяти. Когда выделенная память исчерпывается и данные помещаются в постоянное хранилище, это происходит последовательно страница за страницей. Все страницы занимают около 33 Кб памяти. Всякий раз, когда РЕД КВАНТ нужно загрузить данные с диска, он просто загружает кэш файловой системы, и это происходит очень быстро.

Шаг 8. Теперь необходимо перезапустить узел РЕД КВАНТ и проверить кэш EMPCache с помощью интерфейса командной строки ignitevisor. В кэше нет данных (рисунок 26).

```
visor> cache -scan
(wrn) <visor> No caches found
```

Рисунок 26 - Проверка кэша

Далее необходимо воспользоваться инструментом РЕД КВАНТ SQLLINE для подключения к узлу. Затем нужно выполнить следующую инструкцию SELECT, чтобы получить список сотрудников, не принадлежащих к отделу 30:

```
SELECT*
FROM emp
WHERE deptno NOT IN(30);
```

РЕД КВАНТ может выполнять SQL-запросы к данным, которые находятся как в памяти, так и на диске. Всякий раз, когда возникает какой-либо запрос на чтение, РЕД КВАНТ сначала проверяет данные в памяти. Если набор данных не существует в памяти, РЕД КВАНТ немедленно извлекает записи кэша с диска и загружает их в память. Следует обратить внимание, что все записи в памяти находятся вне heap.

Основным преимуществом собственной персистентности РЕД КВАНТ является надежность данных. Также может быть удобно выполнять резервное копирование для восстановления данных. Кроме того, можно реплицировать данные из одного кластера в другой в режиме реального времени, используя собственную персистентность РЕД КВАНТ. Можно воспользоваться любым основанным на дисках стандартным средством репликации

данных, чтобы скопировать измененный набор данных из основного центра обработки данных в резервный или другой кластер РЕД КВАНТ.

5.3.2 Сохранение в сторонней базе данных (MongoDB)

РЕД КВАНТ использует стратегии кэширования со сквозным чтением и сквозной записью для сохранения данных в сторонних базах данных. РЕД КВАНТ предоставляет готовую реализацию для чтения и записи сведений базы данных из любой СУБД и Apache Cassandra. РЕД КВАНТ реализует интерфейс JCache для CacheLoader и CacheWriter, которые используются для сквозного чтения и записи на любые базовые постоянные носители. РЕД КВАНТ предоставляет интерфейс org.apache.ignite.cache.store.CacheStoreinterface, который расширяет интерфейсы CacheLoader и CacheWriter.

Примечание: Для других баз данных NoSQL, таких как MongoDB или Neo4J, РЕД КВАНТ предоставляет API для реализации пользовательского хранилища кэша. Можно самостоятельно реализовать интерфейс org.apache.ignite.cache.store.CacheStore для хранения записей кэша на любых внешних ресурсах, таких как локальная файловая система, SAN/NAS или Hadoop HDFS.

Кроме того, РЕД КВАНТ CacheStore полностью транзакционный и автоматически вливается в текущую транзакцию кэша. Однако при использовании сквозной записи есть компромисс. Для каждой отдельной транзакции РЕД КВАНТ пытается сохранить или обновить данные в постоянном хранилище или в базе данных на диске, поэтому общая продолжительность времени обновления кэша может быть относительно высокой. Здесь РЕД КВАНТ аналогичен базе данных на диске.

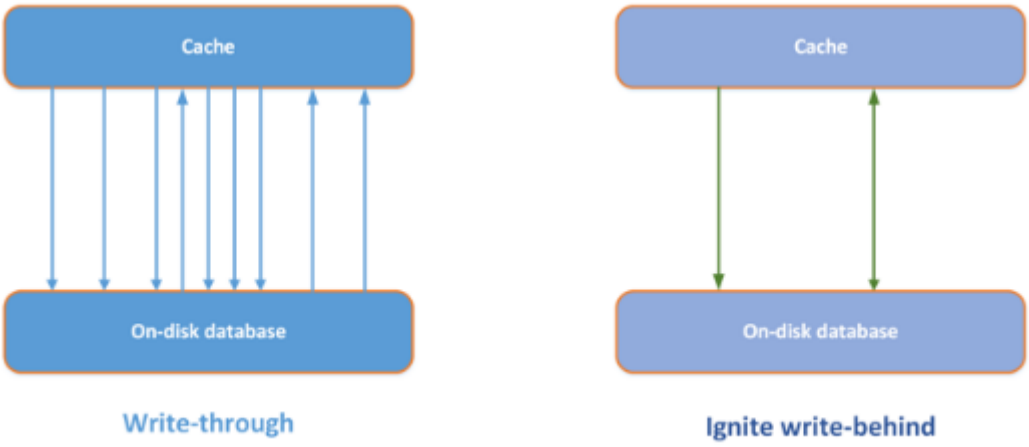

Рисунок 27 - Асинхронное обновление с отложенной записью

Интенсивное обновление кэша может привести к чрезвычайно высокой нагрузке на базу данных на диске. В таких случаях для асинхронного обновления РЕД КВАНТ предлагает опцию массового обновления, называемую отложенной записью (рисунок 27). Основная идея этого подхода — агрегировать обновления и асинхронно сбрасывать их в постоянное хранилище. РЕД КВАНТ может сбрасывать агрегированные данные по разным критериям:

- Основанные на времени: максимальное время, в течение которого запись данных может находиться в очереди.

- Основанные на размере очереди события: очередь сбрасывается, когда ее размер достигает определенного размера.

- Комбинация обоих критериев.

Внимание: При использовании отложенной записи в базовое хранилище будет записано только последнее обновление записи. Если запись кэша с ключом key1 последовательно обновляется значениями value1, value2 и value3 соответственно, то в постоянное хранилище будет передан только один запрос на сохранение для пары (key1, value3).

В дополнение к методу хранения и загрузки JCache интерфейс РЕД КВАНТ CacheStore предоставляет возможность массового хранения и загрузки кэшей из постоянного хранилища. Кроме того, РЕД КВАНТ предлагает удобный адаптер для хранилища кэша с именем CacheStoreAdapter, который реализует интерфейс CacheStore и предоставляет реализацию по умолчанию для массовых операций. Методы абстрактного класса CacheStoreAdapter, которые необходимо реализовать для работы с постоянным хранилищем, представлены в таблице 3.

Таблица 3 - Методы абстрактного класса CacheStoreAdapter

Название метода	Описание
load(), write(), delete()	Методы, унаследованные от интерфейсов JCache.CacheLoader и JCache.CacheWrite, которые позволяют помещать, получать или удалять записи кэша из постоянного хранилища. Эти методы используются для обеспечения сквозного чтения и сквозной записи при работе с отдельными записями кэша.
loadAll(), writeAll(), deleteAll()	Методы, унаследованные от интерфейсов JCache.CacheLoader и JCache.CacheWriter, позволяют выполнять массовые операции. Эти методы используются для обеспечения сквозной записи и сквозного чтения при работе с несколькими записями кэша. Эти методы должны быть реализованы для пакетных операций в постоянном хранилище, чтобы повысить производительность.
loadCache()	Метод, унаследованный от интерфейса Ignite.CacheStore и используемый для загрузки всех значений из соответствующего постоянного хранилища. Обычно он используется для горячей загрузки кэша при запуске, но также может вызываться в любой момент после запуска кэша.
sessionEnd()	Необязательный метод, имеющий пустую реализацию по умолчанию для завершения транзакций. РЕД КВАНТ использует этот метод для хранения сеанса, который может охватывать более одной операции сохранения кэша. Этот метод полезен при работе с транзакцией.

Класс CacheConfiguration определяет конфигурацию кэша РЕД КВАНТ и предлагает набор методов для включения кэширования сквозной записи/чтения и отложенной записи, представленных в таблице 4.

Таблица 4 - Методы класса CacheConfiguration

Название метода	Описание	Значение по умолчанию
setCacheStoreFactory()	Устанавливает фабрику хранилища кэша для постоянного ханилища. Фабрика хранилища кэша должна быть реализацией абстрактного класса CacheStoreAdapter.	
setReadThrough(boolean)	Включает персистентность со сквозным чтением.	False
setWriteThrough(boolean)	Включает персистентность со сквозной записью.	False
setWriteBehindEnabled(boolean)	Включает персистентность с отложенной записью.	False
setWriteBehindFlushSize(int)	Максимальный размер кэша отложенной записи. Если размер кэша превышает это значение, все кэшированные элементы сбрасываются в хранилище кэша, а кэш записи очищается. Если это значение равно 0, то сброс выполняется в соответствии с интервалом частоты сброса. Установить размер и частоту сброса равными 0 нельзя.	10240
setWriteBehindFlushFrequency(long	Частота, с которой кэш отложенной записи сбрасывается в хранилище кэша, в миллисекундах. Это значение определяет максимальный интервал времени между добавлением/удалением объекта из кэша и моментом применения соответствующей операции к хранилищу кэша. Если это значение равно 0, то сброс выполняется в соответствии с размером сброса. Установить размер и частоту сброса равными 0 нельзя.	5 секунд
setWriteBehindFlushThreadCount(int)	Количество потоков, которые будут выполнять сброс кэша.	1

Название метода	Описание	Значение по умолчанию
setWriteBehindBatchSize(int)	Максимальный размер пакета для операций сохранения в кэше с отложенной записью.	512

5.4 Транзакции

Чтобы включить поддержку транзакций для определенного кэша, необходимо установить для параметра atomicityMode значение TRANSACTIONAL в конфигурации кэша.

Транзакции позволяют группировать несколько операций кэша с одним или несколькими ключами в одну атомарную транзакцию. Эти операции выполняются без каких-либо других чередующихся операций над указанными ключами, и либо все завершаются успешно, либо все терпят неудачу. Частичного выполнения операций нет.

Включить транзакции для определенного кэша можно в конфигурации кэша.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  <property name="transactionConfiguration">
    <bean class="org.apache.ignite.configuration.TransactionConfiguration">
      <!--Set the timeout to 20 seconds-->
      <property name="TxTimeoutOnPartitionMapExchange" value="20000"/>
    </bean>
  </property>
</bean>
```

API «ключ-значение» предоставляет интерфейс для запуска и завершения транзакций, а также для получения метрик, связанных с транзакциями. Интерфейс можно получить из экземпляра РЕД КВАНТ.

```
Ignite ignite = Ignition.ignite();

IgniteTransactions transactions = ignite.transactions();

try (Transaction tx = transactions.txStart()) {
  Integer hello = cache.get("Hello");

  if (hello == 1)
    cache.put("Hello", 11);

  cache.put("World", 22);

  tx.commit();
}
```

5.4.1 Режимы параллелизма и уровни изоляции

Кэши с транзакционным режимом атомарности (TRANSACTIONAL) поддерживают как оптимистический (OPTIMISTIC), так и пессимистический (PESSIMISTIC) режим параллелизма для транзакций. Режим параллелизма определяет, когда будет осуществляться блокировка транзакции начального уровня: во время доступа к данным или на этапе

подготовки. Блокировка предотвращает одновременный доступ к объекту. Например, когда происходит попытка обновить элемент списка задач с помощью пессимистической блокировки, сервер блокирует объект до тех пор, пока не завершится или не отменится транзакция, и никакая другая транзакция или операция не сможет обновить ту же запись. Независимо от режима параллелизма, используемого в транзакции, существует момент времени, когда все записи, включенные в транзакцию, блокируются перед завершением.

Уровень изоляции определяет, как параллельные транзакции «видят» и обрабатывают операции с одними и теми же ключами. РЕД КВАНТ поддерживает уровни изоляции READ_COMMITTED, REPEATABLE_READ и SERIALIZABLE.

Разрешены все комбинации режимов параллелизма и уровней изоляции. Ниже приведено описание поведения системы и гарантий, предоставляемых каждой комбинацией параллелизма и изоляции.

В пессимистических транзакциях блокировки устанавливаются во время первого доступа для чтения или записи (в зависимости от уровня изоляции) и удерживаются транзакцией до ее завершения или отката. В этом режиме блокировки сначала устанавливаются на основных узлах, а затем на этапе подготовки передаются на резервные узлы. Для режима параллелизма PESSIMISTIC можно настроить следующие уровни изоляции:

- READ_COMMITTED — данные считываются без блокировки и никогда не кэшируются в самой транзакции. Данные могут быть считаны с резервного узла, если это разрешено в конфигурации кэша. В этом режиме изоляции могут быть так называемые неповторяемые чтения, потому что параллельная транзакция может изменить данные, когда данные дважды считываются в транзакции. Блокировка устанавливается только во время первого доступа для записи (включая вызов EntryProcessor). Это означает, что запись, которая была прочитана во время транзакции, может иметь другое значение к моменту завершения транзакции. В этом случае исключение не выдается.
- REPEATABLE_READ — возникает блокировка записи, и данные извлекаются из основного узла при первом доступе для чтения или записи и сохраняются в локальном словаре транзакций. Все последовательные обращения к одним и тем же данным являются локальными и возвращают последнее прочитанное или обновленное значение транзакции. Это означает, что никакие другие параллельные транзакции не могут вносить изменения в заблокированные данные. Таким образом транзакции осуществляют повторяемые чтения.
- SERIALIZABLE — в режиме PESSIMISTIC этот уровень изоляции работает так же, как REPEATABLE_READ.

Следует отметить, что в режиме PESSIMISTIC важен порядок блокировки. Более того, блокировки возникают последовательно и точно в указанном порядке.

Внимание: Если получена хотя бы одна пессимистическая блокировка транзакции, изменить топологию кэша невозможно до тех пор, пока транзакция не будет завершена или отменена. Поэтому следует избегать удержания блокировок транзакций в течение длительного периода времени.

В транзакциях OPTIMISTIC блокировки записей устанавливаются на основных узлах во время первой фазы 2PC, на этапе подготовки, а затем передаются на резервные узлы и освобождаются после завершения транзакции. При откате транзакции без попытки её завершить блокировки никогда не будут получены. В режиме параллелизма OPTIMISTIC можно настроить следующие уровни изоляции:

- READ_COMMITTED — изменения, которые должны быть применены к кэшу, собираются на исходном узле и применяются при завершении транзакции. Данные транзакции считываются без блокировки и никогда не кэшируются в транзакции. Данные могут быть считаны с резервного узла, если это разрешено в конфигурации кэша. На этом уровне изоляции могут быть так называемые неповторяемые чтения, потому что при считывании данных в транзакции дважды параллельная транзакция может изменить данные. Эта комбинация режимов не проверяет, было ли значение записи изменено с момента первого доступа для чтения или записи, и никогда не вызывает оптимистичное исключение.
- REPEATABLE_READ — транзакции на этом уровне изоляции работают аналогично транзакциям OPTIMISTIC READ_COMMITTED с одним отличием: прочитанные значения кэшируются на исходном узле, и все последующие чтения гарантированно будут локальными. Эта комбинация режимов не проверяет, было ли значение записи изменено с момента первого доступа для чтения или записи, и никогда не вызывает оптимистичное исключение.
- SERIALIZABLE — сохраняет версию записи при первом доступе для чтения. РЕД КВАНТ расценивает транзакцию как ошибочную на этапе завершения, если механизм РЕД КВАНТ обнаруживает, что была изменена хотя бы одна из записей, используемая как часть инициированной транзакции. Иными словами, это означает, что если РЕД КВАНТ обнаруживает конфликт на этапе завершения транзакции, он завершает транзакцию с ошибкой, вызывая исключение TransactionOptimisticException и откатывая любые сделанные изменения. Следует убедиться, что это исключение обработано, и повторить транзакцию.

```
CacheConfiguration<Integer, String> cfg = new CacheConfiguration<>();
cfg.setAtomicityMode(CacheAtomicityMode.TRANSACTIONAL);
cfg.setName("myCache");
IgniteCache<Integer, String> cache = ignite.getOrCreateCache(cfg);

// Re-try the transaction a limited number of times.
int retryCount = 10;
int retries = 0;

// Start a transaction in the optimistic mode with the serializable isolation
// level.
while (retries < retryCount) {
   retries++;
   try (Transaction tx = ignite.transactions().txStart(TransactionConcurrency.OPTIMISTIC,
        TransactionIsolation.SERIALIZABLE)) {
      // modify cache entries as part of this transaction.
      cache.put(1, "foo");
```

```
        cache.put(2, "bar");
        // commit the transaction
        tx.commit();

        // the transaction succeeded. Leave the while loop.
        break;
      } catch (TransactionOptimisticException e) {
        // Transaction has failed. Retry.
      }
    }
```

Еще один важный момент, на который следует обратить внимание, заключается в том, что транзакция завершается ошибкой, даже если запись была прочитана без изменения (cache.put(…)), поскольку значение записи может быть важно для логики инициированной транзакции.

Следует обратить внимание, что порядок ключей важен для транзакций READ_COMMITTED и REPEATABLE_READ, поскольку в этих режимах блокировки по-прежнему устанавливаются последовательно.

Для достижения полной согласованности чтения в режиме PESSIMISTIC необходимо установить блокировки чтения. Это означает, что полная согласованность между чтениями в режиме PESSIMISTIC может быть достигнута только с транзакциями PESSIMISTIC REPEATABLE_READ (или SERIALIZABLE).

При использовании транзакций OPTIMISTIC полная согласованность чтения может быть достигнута за счет исключения потенциальных конфликтов между операциями чтения. Такое поведение обеспечивается режимом OPTIMISTIC SERIALIZABLE. Однако, следует обратить внимание, что до тех пор, пока не произойдет завершение транзакции, все еще возможно прочитать несогласованное состояние транзакции, поэтому логика транзакции должна защищать от этого. Только на этапе совершения, в случае любого конфликта, создается исключение TransactionOptimisticException, позволяющее повторить транзакцию.

Внимание: Если не используются транзакции PESSIMISTIC REPEATABLE_READ или SERIALIZABLE или транзакции OPTIMISTIC SERIALIZABLE, то можно увидеть частичное состояние транзакции. Это означает, что если одна транзакция обновляет объекты A и B, то другая транзакция может увидеть новое значение для A и старое значение для B.

5.4.2 Обнаружение взаимоблокировки

Одно из основных правил, которое необходимо соблюдать при работе с распределенными транзакциями, заключается в том, что блокировки для участвующих в транзакции ключей должны быть получены в одном и том же порядке. Нарушение этого правила может привести к распределенной взаимоблокировке.

РЕД КВАНТ не избегает распределенных взаимоблокировок, а имеет встроенную функциональность, упрощающую отладку и исправление таких ситуаций.

В приведенном ниже фрагменте кода транзакция была запущена с тайм-аутом. Если тайм-аут истекает, процедура обнаружения взаимоблокировки пытается найти возможную

взаимоблокировку, которая могла вызвать тайм-аут. По истечении времени ожидания создается исключение TransactionTimeoutException, которое передается в код приложения в качестве причины CacheException независимо от взаимоблокировки. Однако при обнаружении взаимоблокировки причиной возвращенного TransactionTimeoutException будет TransactionDeadlockException (по крайней мере, для одной транзакции, вовлеченной в взаимоблокировку).

```
CacheConfiguration<Integer, String> cfg = new CacheConfiguration<>();
cfg.setAtomicityMode(CacheAtomicityMode.TRANSACTIONAL);
cfg.setName("myCache");
IgniteCache<Integer, String> cache = ignite.getOrCreateCache(cfg);

try (Transaction tx = ignite.transactions().txStart(TransactionConcurrency.PESSIMISTIC,
      TransactionIsolation.READ_COMMITTED, 300, 0)) {
   cache.put(1, "1");
   cache.put(2, "1");

   tx.commit();
} catch (CacheException e) {
   if (e.getCause() instanceof TransactionTimeoutException
         && e.getCause().getCause() instanceof TransactionDeadlockException)

      System.out.println(e.getCause().getCause().getMessage());
}
```

Сообщение TransactionDeadlockException содержит полезную информацию, которая может помочь вам найти причину взаимоблокировки.

Обнаружение взаимоблокировки — это многоэтапная процедура, которая может выполняться много раз в зависимости от количества узлов в кластере, ключей и транзакций, вовлеченных в возможную взаимоблокировку. Инициатором обнаружения взаимоблокировок является узел, на котором транзакция была запущена и завершилась с исключением TransactionTimeoutException. С помощью передачи запросов и ответов с другими удаленными узлами этот узел проверяет, произошла ли взаимоблокировка, а затем подготавливает отчет о взаимоблокировке, который предоставляется с TransactionDeadlockException. Каждое такое сообщение (запрос/ответ) называется итерацией.

Поскольку транзакция не откатывается до тех пор, пока не будет завершена процедура обнаружения взаимоблокировки, при необходимости предсказуемого времени отката транзакции имеет смысл настроить указанные ниже параметры.

- IgniteSystemProperties.IGNITE_TX_DEADLOCK_DETECTION_MAX_ITERS — указывает максимальное количество итераций для процедуры обнаружения взаимоблокировок. Если значение этого свойства меньше или равно нулю, обнаружение взаимоблокировок отключено (по умолчанию 1000);
- IgniteSystemProperties.IGNITE_TX_DEADLOCK_DETECTION_TIMEOUT — указывает время ожидания для механизма обнаружения взаимоблокировок (по умолчанию 1 минута).

Следует обратить внимание, что если итераций слишком мало, можно получить неполный отчет о взаимоблокировке.

5.4.3 Транзакции без взаимоблокировок

Для транзакций OPTIMISTIC SERIALIZABLE блокировки не устанавливаются последовательно. В этом режиме доступ к ключам возможен в любом порядке, поскольку блокировки транзакций устанавливаются параллельно с дополнительной проверкой, позволяющей РЕД КВАНТ избежать взаимоблокировок.

Прежде чем описать, как работают блокировки в транзакциях SERIALIZABLE, необходимо ввести некоторые понятия. В РЕД КВАНТ каждой транзакции присваивается сопоставимая версия, называемая XidVersion. После завершения транзакции каждой записи, записанной в транзакции, назначается новая сопоставимая версия с именем EntryVersion. Транзакция OPTIMISTIC SERIALIZABLE с версией XidVersionA завершается с ошибкой TransactionOptimisticException, если:

- Существует текущая пессимистическая или несериализуемая оптимистическая транзакция, удерживающая блокировку записи транзакции SERIALIZABLE.
- Существует еще одна текущая оптимистическая транзакция SERIALIZABLE с версией XidVersionB, так что XidVersionB > XidVersionA, и эта транзакция блокирует запись транзакции SERIALIZABLE.
- К тому времени, когда транзакция OPTIMISTIC SERIALIZABLE получает все необходимые блокировки, существует запись с текущей версией, отличной от наблюдаемой версии до завершения транзакции.

Примечание: В среде с высокой степенью параллелизма оптимистичная блокировка может привести к высокой частоте сбоев транзакций, а пессимистическая блокировка может привести к взаимоблокировкам, если блокировки запрашиваются транзакциями в другом порядке.

Однако в бесконфликтной среде оптимистичная сериализуемая блокировка может обеспечить лучшую производительность для больших транзакций, поскольку количество сетевых подключений зависит только от количества узлов, которые охватывает транзакция, и не зависит от количества ключей в транзакции.

5.4.4 Обработка неудачных транзакций

Транзакция может завершиться неудачно со следующими исключениями, представленными в таблице 5.

Таблица 5 - Исключения при неудачных транзакциях

Исключение	Описание	Решение
CacheException, вызванное TransactionTimeoutException	TransactionTimeoutException генерируется, если время ожидания транзакции истекло.	Чтобы устранить это исключение, необходимо увеличить время ожидания или сократить транзакцию.

Исключение	Описание	Решение
CacheException, вызванное TransactionTimeoutException, которое вызвано TransactionDeadlockException	Это исключение возникает, если оптимистическая транзакция по какой-либо причине завершается неудачно. В большинстве случаев это исключение возникает, когда данные, которые транзакция пыталась обновить, изменялись одновременно.	Необходимо повторить транзакцию.
TransactionOptimisticException	Это исключение возникает, если оптимистичная транзакция по какой-либо причине завершается неудачей. В большинстве сценариев это исключение возникает, когда данные, которые транзакция пыталась обновить, изменялись одновременно.	Необходимо повторить транзакцию.
TransactionRollbackException	Это исключение возникает, когда транзакция откатывается (автоматически или вручную). В этом случае данные непротиворечивы.	Поскольку данные находятся в согласованном состоянии, можно повторить транзакцию.
TransactionHeuristicException	Маловероятное исключение, возникающее из-за неожиданной внутренней проблемы или проблемы со связью. Исключение существует для сообщения о проблемных сценариях, которые не были предусмотрены транзакционной подсистемой и не были обработаны ею должным образом.	Данные могут не остаться согласованными, если возникает исключение. Необходимо перезагрузить данные и сообщить об этом сообществу разработчиков РЕД КВАНТ.

5.4.5 Прекращение длительных транзакций

Некоторые события кластера запускают процесс обмена словарями разделов и ребалансировку данных в кластере РЕД КВАНТ для обеспечения равномерного распределения данных по всему кластеру. Примером одного из таких событий является событие изменения топологии кластера, которое происходит всякий раз, когда новый узел присоединяется к кластеру или существующий покидает его. Кроме того, каждый раз, когда создается новый кэш или таблица SQL, запускается обмен словарями разделов.

Когда начинается обмен словарями разделов, РЕД КВАНТ получает глобальную блокировку на определенном этапе. Блокировка не может быть получена, пока параллельно выполняются незавершенные транзакции. Эти транзакции препятствуют продвижению процесса обмена словарями разделов, тем самым блокируя некоторые операции, такие как процесс присоединения нового узла.

Чтобы установить максимальное время, в течение которого длительные транзакции могут блокировать обмен словарями разделов, используется метод TransactionConfiguration.setTxTimeoutOnPartitionMapExchange(…). По истечении тайм-аута все незавершенные транзакции откатываются, позволяя продолжить обмен словарями разделов.

В этом примере показано, как настроить тайм-аут:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  <property name="transactionConfiguration">
    <bean class="org.apache.ignite.configuration.TransactionConfiguration">
      <!--Set the timeout to 20 seconds-->
      <property name="TxTimeoutOnPartitionMapExchange" value="20000"/>
    </bean>
  </property>

</bean>
```

5.5 JDBC ДРАЙВЕР

РЕД КВАНТ поставляется с драйверами JDBC, которые позволяют непосредственно со стороны JDBC обрабатывать распределенные данные с помощью стандартных операторов SQL, таких как SELECT, INSERT, UPDATE или DELETE.

В настоящее время РЕД КВАНТ поддерживает два драйвера: легковесный и простой в использовании тонкий драйвер JDBC, и клиентский драйвер JDBC, который взаимодействует с кластером посредством клиентского узла.

5.5.1 ТОНКИЙ ДРАЙВЕР JDBC

Тонкий драйвер JDBC — это облегченный драйвер по умолчанию, предоставляемый РЕД КВАНТ. Чтобы начать использовать драйвер, просто необходимо добавить ignite-core-2.13.0.jar в путь к классам приложения.

Драйвер подключается к одному из узлов кластера и перенаправляет на него все запросы для окончательного выполнения. Узел обрабатывает распределение запросов и агрегирование результатов. Затем результат отправляется обратно в клиентское приложение.

Строка подключения JDBC может быть отформатирована с использованием одного из двух шаблонов: URL-запрос или точка с запятой:

```
// URL query pattern
jdbc:ignite:thin://<hostAndPortRange0>[,<hostAndPortRange1>]...[,<hostAndPortRangeN>][/schema][?<params>]

hostAndPortRange := host[:port_from[..port_to]]

params := param1=value1[&param2=value2]...[&paramN=valueN]

// Semicolon pattern
```

jdbc:ignite:thin://<hostAndPortRange0>[,<hostAndPortRange1>]...[,<hostAndPortRangeN>][;schema=<schema_name>][;param1=value1]...[;paramN=valueN]

• host является обязательным и определяет хост узла кластера, к которому необходимо подключиться.

• port_from — это начало диапазона портов, используемого для открытия соединения. 10800 используется по умолчанию, если этот параметр отсутствует.

• port_to не является обязательным. По умолчанию устанавливается значение port_from, если этот параметр отсутствует.

• schema — это имя схемы для доступа. По умолчанию используется PUBLIC. Это имя должно соответствовать стандарту SQL ANSI-99. Идентификаторы без кавычек не чувствительны к регистру. Идентификаторы в кавычках чувствительны к регистру. Когда используется формат точки с запятой, схема может быть определена как параметр с именем схемы.

• <params> необязательны.

Задав несколько конечных точек подключения в строке подключения, можно включить автоматическую отработку отказа, если текущее подключение разорвано. Драйвер JDBC случайным образом выбирает адрес из списка для подключения. В случае сбоя соединения драйвер JDBC выбирает другой адрес из списка до тех пор, пока соединение не будет восстановлено. Драйвер прекращает повторное подключение и выдает исключение, если все конечные точки недоступны.

В приведенном ниже примере показано, как передать три адреса через строку подключения:

```
// Register JDBC Driver.
Class.forName("org.apache.ignite.IgniteJdbcThinDriver");

// Open the JDBC connection passing several connection endpoints.
Connection conn = DriverManager
    .getConnection("jdbc:ignite:thin://192.168.0.50:101,192.188.5.40:101,192.168.10.230:101");
```

Осведомленность о разделах — это функция, позволяющая драйверу JDBC «осведомляться» о распределении разделов в кластере. Это позволяет драйверу выбирать узлы, которым принадлежат запрашиваемые данные, и отправлять запрос непосредственно на эти узлы (если адреса узлов указаны в конфигурации драйвера). Осведомленность о разделах может повысить среднюю производительность запросов, использующих affinity-ключ.

Без осведомленности о разделах драйвер JDBC подключается к одному узлу, и все запросы выполняются через этот узел. Если данные размещены на другом узле, запрос должен быть перенаправлен внутри кластера, что добавляет дополнительный сетевой переход. Осведомленность о разделах устраняет этот переход, отправляя запрос на правильный узел.

Чтобы использовать функцию осведомленности о разделах, необходимо указать адреса всех серверных узлов в свойствах соединения. Драйвер будет направлять запросы к узлам, которые хранят данные, запрошенные запросом.

Внимание: В настоящее время необходимо указать адреса всех серверных узлов в свойствах соединения, поскольку драйвер не загружает их автоматически после открытия соединения. Это также означает, что если к кластеру присоединится новый серверный узел, рекомендуется переподключить драйвер и добавить адрес узла в свойства подключения. В противном случае драйвер не сможет отправлять прямые запросы к этому узлу.

Чтобы включить поддержку разделов, необходимо добавить параметр partitionAwareness=true в строку подключения и указать конечные точки нескольких серверных узлов:

```
Class.forName("org.apache.ignite.IgniteJdbcThinDriver");

Connection conn = DriverManager
    .getConnection("jdbc:ignite:thin://192.168.0.50,192.188.5.40,192.168.10.230?partitionAwareness=true");
```

5.5.2 Клиентский драйвер JDBC

Клиентский драйвер JDBC взаимодействует с кластером посредством клиентского узла.

Клиентский драйвер JDBC подключается к кластеру с помощью подключения к узлу lclient. Необходимо предоставить полную XML-конфигурацию Spring как часть строки подключения JDBC и скопировать все JAR-файлы, упомянутые ниже, в путь к классам приложения или инструмента SQL:

• Все файлы JAR в каталоге {IGNITE_HOME}\libs.

• Все файлы JAR в каталогах {IGNITE_HOME}\ignite-indexing и {IGNITE_HOME}\ignite-spring.

Сам драйвер более надежен и может не поддерживать последние SQL функции РЕД КВАНТ. Однако, поскольку внутри он использует соединение с клиентским узлом, он может выполнять и распределять запросы и агрегировать их результаты непосредственно со стороны приложения.

URL-адрес подключения JDBC имеет следующий шаблон:

```
jdbc:ignite:cfg://[<params>@]<config_url>
```

Где:

• <config_url> является обязательным и должен представлять допустимый URL-адрес, указывающий на файл конфигурации для клиентского узла. Этот узел будет запущен клиентским драйвером РЕД КВАНТ JDBC, когда он (драйвер JDBC) попытается установить соединение с кластером.

• <params> является необязательным и имеет следующий формат:

```
param1=value1:param2=value2:...:paramN=valueN
```

Имя класса драйвера — org.apache.ignite.IgniteJdbcDriver. Пример открытия JDBC с кластером РЕД КВАНТ:

```
// Registering the JDBC driver.
Class.forName("org.apache.ignite.IgniteJdbcDriver");
```

```
// Opening JDBC connection (cache name is not specified, which means that we use default cache).
Connection conn = DriverManager.getConnection("jdbc:ignite:cfg://config/ignite-jdbc.xml");
```

Клиентский драйвер JDBC поддерживает параметры, указанные в таблице 6.

Таблица 6 - Поддерживаемые параметры

Параметр	**Описание**	**Значение по умолчанию**
cache	Имя кэша. Если он не определен, то будет использоваться кэш по умолчанию. Обратите внимание, что имя кэша чувствительно к регистру.	Нет
nodeId	ID узла, на котором будет выполняться запрос. Полезно для запросов через локальные кэши.	Нет
local	Запрос будет выполняться только на локальном узле. Этот параметр необходимо использовать с параметром nodeId, чтобы ограничить набор данных указанным узлом.	false
collocated	Флаг, который используется в целях оптимизации. Всякий раз, когда РЕД КВАНТ выполняет распределенный запрос, он отправляет подзапросы отдельным членам кластера. Если заранее известно, что элементы выборки запроса расположены вместе на одном узле, РЕД КВАНТ может значительно повысить производительность и оптимизировать работу с сетью.	false
distributedJoins	Позволяет использовать распределенные объединения для не сколоцированных данных.	false
streaming	Включает режим массовой загрузки данных через операторы INSERT для этого подключения. Подробнее см. в разделе Режим потоковой передачи.	false
streamingAllowOverwrite	Указывает РЕД КВАНТ перезаписывать значения существующих ключей при дублировании, а не пропускать их. Подробнее см. в разделе Режим потоковой передачи.	false
streamingFlushFrequency	Время ожидания в миллисекундах, которое Data streamer должен использовать для сброса данных. По	0

Параметр	Описание	Значение по умолчанию
	умолчанию данные сбрасываются при закрытии соединения. Подробнее см. в разделе Режим потоковой передачи.	
streamingPerNodeBufferSize	Размер буфера Data streamer на узел. Подробнее см. в разделе Режим потоковой передачи.	1024
streamingPerNodeParallelOperations	Число параллельных операций Data streamer на узел. Подробнее см. в разделе Режим потоковой передачи.	16
transactionsAllowed	В настоящее время ACID транзакции поддерживаются, но только на уровне API «ключ-значение». На уровне SQL РЕД КВАНТ поддерживает атомарную, но не транзакционную согласованность. Это означает, что при попытке использовать эту функцию будет выброшено исключение «Транзакции не поддерживаются». В случаях, когда нужен транзакционный синтаксис (даже без транзакционной семантики), для предотвращения возникновения исключений необходимо установить для этого параметра значение true. Например, некоторые BI tools могут вызвать транзакционное поведение.	false
multipleStatementsAllowed	Драйвер JDBC сможет обрабатывать несколько операторов SQL одновременно, возвращая несколько объектов ResultSet. Если параметр отключен, запрос с несколькими операторами завершается ошибкой.	false
lazy	Ленивое выполнение запросов. По умолчанию РЕД КВАНТ пытается извлечь весь набор результатов запроса в память и отправить его клиенту. Для малых и средних наборов результатов это обеспечивает оптимальную производительность и минимизирует продолжительность внутренних блокировок базы данных, тем самым увеличивая параллелизм. Однако, если результирующий набор слишком велик для размещения в доступной памяти, это может привести	false

Параметр	Описание	Значение по умолчанию
	к чрезмерным паузам GC (сборщика мусора) и даже ошибкам OutOfMemoryError. Этот флаг используется, чтобы указать РЕД КВАНТ лениво извлекать набор результатов, тем самым сводя к минимуму потребление памяти за счет умеренного падения производительности.	
skipReducerOnUpdate	Включает функцию обновления на стороне сервера. Когда РЕД КВАНТ выполняет операцию DML (Data Manipulation Language — язык манипулирования данными), он сначала извлекает все затронутые промежуточные строки для анализа инициатором запроса (Reducer), а затем подготавливает пакеты обновленных значений для отправки на удаленные узлы. Такой подход может повлиять на производительность и перегрузить сеть, если операция DML должна перемещать по ней много записей. Необходимо использовать этот флаг в качестве подсказки для РЕД КВАНТ, чтобы выполнять анализ всех промежуточных строк и обновления «на месте» на соответствующих удаленных узлах данных. По умолчанию установлено значение false, что означает, что промежуточные результаты сначала будут получены инициатором запроса.	false

5.5.2.1 Режим потоковой передачи

Можно добавлять данные в кластер в потоковом режиме (массовом режиме) с помощью драйвера JDBC. В этом режиме драйвер создает экземпляр IgniteDataStreamer внутри и передает ему данные. Чтобы активировать этот режим, нужно добавить для параметра потоковой передачи значение true в строку подключения JDBC:

```
// Register JDBC driver.
Class.forName("org.apache.ignite.IgniteJdbcDriver");

// Opening connection in the streaming mode.
```

```
Connection conn = DriverManager.getConnection("jdbc:ignite:cfg://streaming=true@file:///etc/config/ignite-jdbc.xml");
```

В настоящее время потоковый режим поддерживается только для операций INSERT. Это полезно в тех случаях, когда необходимо добиться быстрой предварительной загрузки данных в кэш. Драйвер JDBC определяет несколько параметров соединения, влияющих на поведение режима потоковой передачи. Эти параметры перечислены в таблице параметров выше.

Внимание: Необходимо убедиться, что указан целевой кэш для потоковой передачи в качестве аргумента параметра cache= в строке подключения JDBC. Если кэш не указан или не соответствует таблице, используемой в потоковых операторах DML, обновления будут игнорироваться.

Параметры охватывают почти все настройки общего IgniteDataStreamer и позволяют настроить Data streamer в соответствии с потребностями. Для получения дополнительной информации о том, как настроить стример, см. «Потоковая передача данных».

Примечание: По умолчанию данные сбрасываются при закрытии соединения или при наличии параметра streamingPerNodeBufferSize. Если нужно сбрасывать данные чаще, необходимо настроить параметр streamingFlushFrequency.

```
// Register JDBC driver.
  Class.forName("org.apache.ignite.IgniteJdbcDriver");

  // Opening a connection in the streaming mode and time based flushing set.
  Connection conn = DriverManager.getConnection("jdbc:ignite:cfg://streaming=true:streamingFlushFrequency=1000@file:///etc/config/ignite-jdbc.xml");

  PreparedStatement stmt = conn.prepareStatement(
    "INSERT INTO Person(_key, name, age) VALUES(CAST(? as BIGINT), ?, ?)");

  // Adding the data.
  for (int i = 1; i < 100000; i++) {
      // Inserting a Person object with a Long key.
      stmt.setInt(1, i);
      stmt.setString(2, "John Smith");
      stmt.setInt(3, 25);

      stmt.execute();
  }

  conn.close();

  // Beyond this point, all data is guaranteed to be flushed into the cache.
```

5.6 ODBC ДРАЙВЕР

РЕД КВАНТ включает в себя драйвер ODBC, который позволяет выбирать и изменять данные, хранящиеся в распределенном кэше, с помощью стандартных запросов SQL и собственного API ODBC.

Драйвер ODBC рассматривается как динамическая библиотека для Windows и Linux (shared object). Приложение не загружает его напрямую. Вместо этого оно использует API диспетчера драйверов, который при необходимости загружает и выгружает драйверы ODBC.

Внутри драйвер ODBC использует TCP для подключения к кластеру. Параметры подключения на стороне кластера можно настроить с помощью свойства IgniteConfiguration.clientConnectorConfiguration.

```
<bean id="ignite.cfg" class="org.apache.ignite.configuration.IgniteConfiguration">
  <property name="clientConnectorConfiguration">
    <bean class="org.apache.ignite.configuration.ClientConnectorConfiguration"/>
  </property>
</bean>
```

Конфигурация клиентского коннектора поддерживает свойства, представленные в таблице 7.

Таблица 7 - Свойства, поддерживаемые конфигурацией клиентского коннектора

Параметр	Описание	Значение по умолчанию
host	Имя хоста или IP-адрес для привязки. Если установлено значение null, выполняется привязка к локальному хосту.	null
port	TCP-порт для привязки. Если указанный порт уже используется, РЕД КВАНТ попытается найти другой доступный порт, используя свойство portRange.	10800
portRange	Определяет количество портов, к которым необходимо выполнить привязку. Например, если для порта установлено значение 10800, а для portRange установлено значение 100, то сервер будет последовательно пытаться выполнить привязку к любому порту из [10800, 10900], пока не найдет свободный порт.	100

Параметр	Описание	Значение по умолчанию
maxOpenCursorsPerConnection	Максимальное количество курсоров, которые можно открыть одновременно для одного соединения.	128
threadPoolSize	Количество потоков обработки запросов в пуле потоков.	MAX(8, ядер ЦП)
socketSendBufferSize	Размер буфера для отправки сокета TCP. При значении 0 используется системное значение по умолчанию.	0
socketReceiveBufferSize	Размер буфера для приема сокета TCP. При значении 0 используется системное значение по умолчанию.	0
tcpNoDelay	Необходимо ли использовать параметр TCP_NODELAY.	true
idleTimeout	Тайм-аут простоя для клиентских подключений. Клиенты будут автоматически отключены от сервера после бездействия в течение настроенного времени ожидания. Если для этого параметра задано нулевое или отрицательное значение, тайм-аут простоя будет отключен.	0
isOdbcEnabled	Включен ли доступ через ODBC.	true
isThinClientEnabled	Необходимо ли использовать доступ через тонкий клиент.	true

Эти параметры можно изменить следующим образом:

```
        <bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <!-- Enabling ODBC. -->
  <property name="clientConnectorConfiguration">
    <bean class="org.apache.ignite.configuration.ClientConnectorConfiguration">
      <property name="host" value="127.0.0.1"/>
      <property name="port" value="10800"/>
      <property name="portRange" value="5"/>
      <property name="maxOpenCursorsPerConnection" value="512"/>
      <property name="socketSendBufferSize" value="65536"/>
      <property name="socketReceiveBufferSize" value="131072"/>
      <property name="threadPoolSize" value="4"/>
    </bean>
  </property>
</bean>
```

5.6.1 Сборка драйвера ODBC

РЕД КВАНТ поставляется с готовыми установщиками как для 32-, так и для 64-разрядных версий драйвера для Windows. Поэтому, если необходимо просто установить

драйвер ODBC в Windows, можно сразу перейти к разделу «Установка драйвера ODBC» для получения инструкций по установке.

Для Linux потребуется создать драйвер ODBC перед его установкой. Поэтому, если используется Linux или все еще есть желание самостоятельно собрать драйвер для Windows, см. следующие пункты.

5.6.1.1 Сборка на Windows

Для начала необходимо установить следующие зависимости:

• MS Visual C (10.0 и выше), g (4.4.0 и выше);

• OpenSSL (32-разрядная или 64-разрядная версии);

• CMake 3.6+;

• Набор инструментов WiX и добавить его в %Path%.

Затем необходимо выполнить следующее:

1. Перейти в папку %IGNITE_HOME%\platforms\cpp.
2. Собрать драйверы и установщики, выполнив следующие действия:

- для 64-разрядной версии:

```
mkdir cmake-build-release-64

cmake .. -DWITH_CORE=OFF -DWITH_ODBC=ON -DWITH_ODBC_MSI=ON -
DCMAKE_GENERATOR_PLATFORM=x64 -DOPENSSL_ROOT_DIR=<openssl 64-bit install dir> -
DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=..\install\amd64

cmake --build . --target install --config Release
```

- для 32-разрядной версии:

```
mkdir cmake-build-release-32

cmake .. -DWITH_CORE=OFF -DWITH_ODBC=ON -DWITH_ODBC_MSI=ON -
DCMAKE_GENERATOR_PLATFORM=Win32 -DOPENSSL_ROOT_DIR=<openssl 32-bit install dir> -
DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=..\install\x86
cmake --build . --target install --config Release
```

В результате файлы ignite-odbc-amd64.msi и ignite-odbc-x86.msi должны появиться в каталогах %IGNITE_HOME%\platforms\cpp\install\amd64\bin и %IGNITE_HOME%\platforms\cpp\install\x86\bin соответственно.

5.6.1.2 Сборка на Linux

В операционной системе на базе Linux потребуется установить диспетчер драйверов ODBC, чтобы иметь возможность создавать и использовать драйвер РЕД КВАНТ ODBC. Драйвер ODBC был протестирован с UnixODBC.

Необходимо установить следующие пакеты:

- Компилятор C++

- cmake 3.6+
- openssl, включая файлы заголовков
- unixODBC

Ниже представлен пример по установке дистрибутива Ubuntu 18.04/20.04:

```
sudo apt-get install -y build-essential cmake unixodbc-dev libssl-dev
```

5.6.1.3 Сборка драйвера ODBC

1. Необходимо создать каталог сборки для cmake, назвав его ${CPP_BUILD_DIR}.

2. (Необязательно) Выбрать префикс каталога установки (по умолчанию /usr/local) назвав его ${CPP_INSTALL_DIR}.

3. Необходимо собрать и установить драйвер, выполнив следующие команды:

```
cd ${CPP_BUILD_DIR}

cmake -DCMAKE_BUILD_TYPE=Release -DWITH_CORE=OFF -DWITH_ODBC=ON
${IGNITE_HOME}/platforms/cpp -DCMAKE_INSTALL_PREFIX=${CPP_INSTALL_DIR}
make
sudo make install
```

После завершения процесса сборки можно узнать, где находится драйвер ODBC, выполнив следующую команду:

```
whereis libignite-odbc
```

Путь должен выглядеть примерно так: /usr/local/lib/libignite-odbc.so

5.6.2 Установка драйвера ODBC

Для использования драйвера ODBC необходимо зарегистрировать его в системе, чтобы диспетчер драйверов ODBC мог его найти.

5.6.2.1 Установка в Windows

Для 32-разрядной версии Windows следует использовать 32-разрядную версию драйвера. Для 64-битной Windows можно использовать как 64-битный драйвер, так и 32-битный. Можно установить как 32-разрядные, так и 64-разрядные драйверы в 64-разрядной версии Windows, чтобы иметь возможность использовать драйвер как в 32-разрядных, так и в 64-разрядных приложениях.

5.6.2.1.1 Установка с помощью установщиков

Примечание: Сначала необходимо установить распространяемый пакет Microsoft Visual C++ 2010 для 32-разрядной или 64-разрядной версии.

Это самый простой способ, и его следует использовать по умолчанию. Необходимо просто запустить установщик нужной версии драйвера и следовать инструкциям:

32-разрядная программа установки: %IGNITE_HOME%\platforms\cpp\bin\odbc\ignite-odbc-x86.msi

64-разрядная программа установки: %IGNITE_HOME%\platforms\cpp\bin\odbc\ignite-odbc-amd64.msi

5.6.2.1.2 Установка вручную

Чтобы установить драйвер ODBC в Windows вручную, сначала нужно выбрать каталог в файловой системе, где будет находиться драйвер или драйверы. После того, как было выбрано место, нужно поместить туда драйвер и убедиться, что все зависимости драйвера также могут быть разрешены, то есть их можно найти либо в %PATH%, либо в том же каталоге, где находится DLL драйвера.

После этого необходимо воспользоваться одним из скриптов установки из следующего каталога %IGNITE_HOME%/platforms/cpp/odbc/install. Следует обратить внимание, что для выполнения этих скриптов могут потребоваться права администратора ОС.

```
install_x86 <absolute_path_to_32_bit_driver>
```

5.6.2.2 Установка в Linux

Чтобы иметь возможность собирать и устанавливать драйвер ODBC в Linux, необходимо сначала установить диспетчер драйверов ODBC. Драйвер ODBC был протестирован с UnixODBC.

После сборки драйвера и выполнения команды make install драйвер ODBC (libignite-odbc.so), будет помещен в папку /usr/local/lib. Чтобы установить его в качестве драйвера ODBC в диспетчере драйверов и иметь возможность его использовать, необходимо выполнить следующие действия:

- Убедиться, что компоновщик может найти все зависимости драйвера ODBC. Это можно проверить с помощью команды ldd. Например, драйвер ODBC находится в каталоге /usr/local/lib:

```
ldd /usr/local/lib/libignite-odbc.so
```

Если есть неразрешенные ссылки на другие библиотеки, можно добавить каталоги с этими библиотеками в LD_LIBRARY_PATH.

- Отредактировать файл ${IGNITE_HOME}/platforms/cpp/odbc/install/ignite-odbc-install.ini и убедиться, что параметр Driver в разделе РЕД КВАНТ указывает на расположение libignite-odbc.so.

- Чтобы установить драйвер ODBC, необходимо воспользоваться следующей командой:

```
odbcinst -i -d -f ${IGNITE_HOME}/platforms/cpp/odbc/install/ignite-odbc-install.ini
```

Для выполнения этой команды могут потребоваться привилегии root.

Теперь драйвер РЕД КВАНТ ODBC установлен и готов к использованию. Можно подключиться к нему и использовать его так же, как любой другой драйвер ODBC.

5.7 MVCC

Кэши с режимом атомарности TRANSACTIONAL_SNAPSHOT поддерживают транзакции SQL, а также транзакции «ключ-значение» и позволяют управлять многоверсионным конкурентным доступом (MVCC) для обоих типов транзакций.

Многоверсионное управление конкурентным доступом (MVCC) - это метод контроля согласованности данных, к которым одновременно обращаются несколько пользователей. MVCC реализует защиту гарантию изоляции зафиксированных данных, что значит, что каждая транзакция всегда видит согласованный моментальный снимок данных.

Каждая транзакция получает согласованный снимок данных при запуске и может просматривать и изменять данные только в этом снимке. Когда транзакция обновляет запись, РЕД КВАНТ проверяет, не была ли запись обновлена другими транзакциями, и создает новую версию записи. Новая версия становится видимой для других транзакций только в том случае, если эта транзакция успешно завершается. Если запись была обновлена, текущая транзакция завершается с исключением (см. пункт Параллельные обновления для получения информации о том, как обрабатывать конфликты обновлений).

Моментальные снимки - это не физические снимки, а логические снимки, которые генерируются координатором MVCC: узлом кластера, который координирует транзакционную активность в кластере. Координатор отслеживает все активные транзакции и получает уведомление о завершении каждой транзакции. Все операции с кэшем с включенным MVCC запрашивают моментальный снимок данных у координатора.

5.7.1 Включение MVCC

Чтобы включить MVCC для кэша, необходимо воспользоваться режимом атомарности TRANSACTIONAL_SNAPSHOT в конфигурации кэша. Если создается таблица с помощью команды CREATE TABLE, нужно указать режим атомарности в качестве параметра в части WITH команды:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">


<property name="cacheConfiguration">
  <bean class="org.apache.ignite.configuration.CacheConfiguration">
    <property name="name" value="myCache"/>
    <property name="atomicityMode" value="TRANSACTIONAL_SNAPSHOT"/>
  </bean>
</property>

</bean>
```

Примечание: Режим TRANSACTIONAL_SNAPSHOT поддерживает только режим параллелизма по умолчанию (PESSIMISTIC) и уровень изоляции по умолчанию (REPEATABLE_READ). Дополнительные сведения см. в разделе Режимы параллелизма и уровни изоляции.

5.7.2 Параллельные обновления

Если запись считывается, а затем обновляется в рамках одной транзакции, возможно, что в промежутке между двумя операциями может быть обработана другая транзакция, которая обновит запись первой. В таком случае возникает исключение, которое выбрасывается, когда первая транзакция пытается обновить запись, после чего транзакция помечается как «Только откат». Необходимо повторить транзакцию.

Способ определения конфликта обновления:

• При использовании Java transaction API выдается исключение CacheException с сообщением «Не удается сериализовать транзакцию из-за конфликта записи (транзакция помечена для отката)», а флаг Transaction.rollbackOnly имеет значение true.

• Когда транзакции SQL выполняются через драйвер JDBC или ODBC, возвращается код ошибки SQLSTATE:40001.

```
for(int i = 1; i <=5 ; i++) {

  try (Transaction tx = Ignition.ignite().transactions().txStart()) {
    System.out.println("attempt #" + i + ", value: " + cache.get(1));
    try {
      cache.put(1, "new value");
      tx.commit();
      System.out.println("attempt #" + i + " succeeded");
      break;
    } catch (CacheException e) {
      if (!tx.isRollbackOnly()) {
        // Transaction was not marked as "rollback only",
        // so it's not a concurrent update issue.
        // Process the exception here.
          break;
      }
    }
  }
}
```

5.7.3 Ограничения

5.7.3.1 Транзакции с перекрестным кэшированием

Режим TRANSACTIONAL_SNAPSHOT включается для каждого кэша и не допускает использование кэшей с разными режимами атомарности в рамках одной транзакции. Как следствие, если необходимо охватить несколько таблиц в одной транзакции SQL, все таблицы должны быть созданы в режиме TRANSACTIONAL_SNAPSHOT.

5.7.3.2 Вложенные транзакции

РЕД КВАНТ поддерживает три режима обработки вложенных SQL-транзакций. Они могут быть включены с помощью параметра подключения JDBC/ODBC.

```
jdbc:ignite:thin://127.0.0.1/?nestedTransactionsMode=ФИКСАЦИЯ
```

Когда вложенная транзакция возникает внутри другой транзакции, параметр nestedTransactionsMode определяет поведение системы:

- ERROR — при обнаружении вложенной транзакции выдается сообщение об ошибке, и внешняя (та, в которой появилась вложенная транзакция) транзакция откатывается. Это поведение по умолчанию.
- COMMIT — внешняя транзакция завершается; вложенная транзакция запускается и завершается, когда встречается ее оператор COMMIT. Остальные операторы внешней транзакции выполняются как неявные транзакции.
- IGNORE — Этот режим использовать нельзя. Начало вложенной транзакции игнорируется, операторы внутри вложенной транзакции будут выполняться как часть внешней транзакции, и все изменения будут завершены вместе с завешением вложенной транзакции. Последующие операторы внешней транзакции будут выполняться как неявные транзакции.

5.7.3.3 Continuous Queries

Если используются Continuous Queries с кэшем с включенным MVCC, существует несколько ограничений, о которых следует знать:

- Когда получено событие обновления, последующие чтения обновленного ключа могут возвращать старое значение в течение некоторого периода времени, прежде чем координатор MVCC узнает об обновлении. Это связано с тем, что событие обновления отправляется с узла, на котором обновляется ключ, как только он обновляется. В таком случае координатор MVCC может не сразу узнать об этом обновлении, и, следовательно, последующие чтения могут возвращать устаревшую информацию в течение этого периода времени.
- Существует ограничение на количество ключей на узел, которые может обновлять одна транзакция при использовании Continuous Queries. Обновленные значения хранятся в памяти, поэтому если обновлений слишком много, узлу может не хватить оперативной памяти для хранения всех объектов. Чтобы избежать ошибок OutOfMemory, каждой транзакции разрешено обновлять не более 20 000 ключей (значение по умолчанию) на одном узле. Если это значение будет превышено, транзакция выдаст исключение и будет отменена. Это число можно изменить, указав системное свойство IGNITE_MVCC_TX_SIZE_CACHING_THRESHOLD.

5.7.3.4 Другие ограничения

Следующие функции не поддерживаются для кэшей с включенным MVCC:

- Expiry Policies;
- События;
- Перехватчики кэша;
- Внешнее хранилище;
- On-heap кэширование;
- Явные блокировки;

- Методы localEvict() и localPeek().

6 SQL

РЕД КВАНТ разработан как горизонтально масштабируемая и отказоустойчивая распределенная база данных SQL, совместимой с ANSI-99. В зависимости от варианта использования распределение обеспечивается либо путем разделения данных между узлами кластера, либо путем полной репликации.

В качестве базы данных SQL РЕД КВАНТ поддерживает все команды DML, включая запросы SELECT, UPDATE, INSERT и DELETE, а также реализует подмножество команд DDL, актуальных для распределенных систем.

С РЕД КВАНТ можно взаимодействовать так же, как и с любым другим хранилищем с поддержкой SQL, подключаясь с помощью драйверов JDBC или ODBC как из внешних инструментов, так и из приложений. Разработчики Java, .NET и C++ могут использовать встроенные API-интерфейсы SQL.

Внутренне таблицы SQL имеют ту же структуру данных, что и кэши «ключ-значение». Это означает, что можно изменить распределение разделов данных и использовать методы affinity-колокации для повышения производительности.

Механизм SQL РЕД КВАНТ по умолчанию использует базу данных на основе H2 для парсинга и оптимизации запросов и создания планов выполнения, но также можно включить механизм SQL на основе Apache Calcite для выполнения запросов.

Внимание: Механизм SQL на основе Apache Calcite является экспериментальной функцией. Дополнительные сведения см. в пункте SQL движок на базе Apache Calcite.

Запросы к партиционированным таблицам выполняются распределенным образом:

- Запрос парсится и разбивается на несколько запросов «map» и один запрос «reduce».
- Все запросы к словарю выполняются на всех узлах, где находятся требуемые данные.
- Все узлы предоставляют наборы результатов локального выполнения инициатору запроса, который, в свою очередь, объединит предоставленные наборы результатов в конечные результаты.

Запрос можно принудительно обработать локально, т.е. на подмножестве данных, которые хранятся на узле, где выполняется запрос.

Если запрос выполняется по реплицированной таблице, он будет выполняться по локальным данным.

Запросы к партиционированным таблицам выполняются распределенным образом. Однако можно принудительно выполнить локальный запрос над партиционированной таблицей.

6.1 Схемы

РЕД КВАНТ имеет ряд схем по умолчанию и поддерживает создание пользовательских схем.

По умолчанию доступны две схемы:

• Схема SYS, которая содержит ряд системных представлений с информацией об узлах кластера. Создавать таблицы в этой схеме нельзя.

• Схема PUBLIC, которая используется по умолчанию, если схема не указана.

Пользовательские схемы создаются в следующих случаях:

• Пользовательские схемы можно указать в конфигурации кластера. См. Пользовательские схемы.

• РЕД КВАНТ создает схему для каждого кэша, созданного с помощью одного из программных интерфейсов или конфигурации XML. См. Имена кэша и схемы.

6.1.1 Схема PUBLIC

Схема PUBLIC используется по умолчанию всякий раз, когда требуется схема, но она не указана. Например, когда выполняется подключение к кластеру через JDBC без явно заданной схемы, будет выполнено подключение к схеме PUBLIC.

6.1.2 Пользовательские схемы

Пользовательские схемы можно задать с помощью свойства sqlSchemas IgniteConfiguration. Можно указать список схем в конфигурации перед запуском кластера, а затем создавать объекты в этих схемах во время выполнения. Ниже приведен пример конфигурации с двумя пользовательскими схемами.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
<property name="sqlConfiguration">
   <bean class="org.apache.ignite.configuration.SqlConfiguration">
     <property name="sqlSchemas">
       <list>
         <value>MY_SCHEMA</value>
         <value>MY_SECOND_SCHEMA</value>
       </list>
     </property>
   </bean>
</property>
</bean>
```

Чтобы подключиться к определенной схеме, например, через драйвер JDBC, нужно указать имя схемы в строке подключения:

```
jdbc:ignite:thin://127.0.0.1/MY_SCHEMA
```

6.1.3 Имена кэша и схемы

Когда создается кэш с запрашиваемыми полями, можно управлять кэшированными данными с помощью SQL API. В терминах SQL каждому такому кэшу соответствует отдельная схема, имя которой совпадает с именем кэша. Точно так же, когда создается таблица с помощью оператора DDL, можно получить к ней доступ как к кэшу «ключ-значение» через программные интерфейсы, поддерживаемые РЕД КВАНТ. Имя соответствующего кэша можно задать, указав параметр CACHE_NAME в части WITH оператора CREATE TABLE.

```
CREATE TABLE City (
  ID INT(11),
  Name CHAR(35),
  CountryCode CHAR(3),
  District CHAR(20),
  Population INT(11),
  PRIMARY KEY (ID, CountryCode)
) WITH "backups=1, CACHE_NAME=City";
```

Подробнее см. **Приложение B. DDL**.

Если этот параметр не используется, имя кэша определяется в следующем формате (заглавными буквами):

```
SQL_<SCHEMA_NAME>_<TABLE_NAME>
```

6.2 Индексы

В дополнение к обычным командам DDL, таким как CREATE/DROP INDEX, разработчики могут использовать API-интерфейсы SQL РЕД КВАНТ для определения индексов.

Примечание: Возможности индексирования предоставляются модулем 'ignite-indexing'. Если РЕД КВАНТ запускается из кода Java, необходимо добавить этот модуль в путь к классам.

РЕД КВАНТ автоматически создает индексы для каждого поля первичного ключа и affinity-ключа. Когда определяется индекс для поля в объекте-значении, РЕД КВАНТ создает составной индекс, состоящий из индексированного поля и первичного ключа кэша. В терминах SQL это означает, что индекс будет состоять из двух столбцов: столбца, который необходимо проиндексировать, и столбца первичного ключа.

6.2.1 Создание индексов с помощью SQL

См. **Приложение B. DDL**.

6.2.2 Настройка индексов с помощью аннотаций

Индексы, а также запрашиваемые поля можно настроить из кода с помощью аннотации @QuerySqlField. В приведенном ниже примере механизм РЕД КВАНТ SQL создаст индексы для полей id и salary.

```
public class Person implements Serializable {

/** Indexed field. Will be visible to the SQL engine. */
@QuerySqlField(index = true)
private long id;

/** Queryable field. Will be visible to the SQL engine. */
@QuerySqlField
private String name;

/** Will NOT be visible to the SQL engine. */
private int age;
```

```
    /**
     * Indexed field sorted in descending order. Will be visible to the SQL engine.
     */
    @QuerySqlField(index = true, descending = true)
    private float salary;
}
```

Имя типа используется в качестве имени таблицы в запросах SQL. В этом случае имя таблицы будет Person (использование и определение имени схемы описано в пункте Схемы).

И id, и salary являются индексируемыми полями. id будет отсортирован в порядке возрастания (по умолчанию), а зарплата в порядке убывания.

Если индексировать поле не нужно, но использовать его в SQL-запросах все же необходимо, то поле необходимо аннотировать без параметра index=true. Такое поле называется запрашиваемым полем. В приведенном выше примере имя определено как запрашиваемое поле.

Поле age не является ни запрашиваемым, ни индексированным полем, поэтому оно не будет доступно из SQL-запросов.

Когда определяются индексированные поля, необходимо зарегистрировать индексированные типы.

Примечание: Необходимо использовать команды CREATE/DROP INDEX, если нужно управлять индексами или сделать новые поля объекта видимыми для механизма SQL во время выполнения.

Дополнительные сведения см. также в **Приложение B. DDL**.

Поля вложенных объектов также можно индексировать и запрашивать с помощью аннотаций. Например, можно рассмотреть объект Person, который имеет объект Address в качестве поля:

```
public class Person {

    /** Indexed field. Will be visible for SQL engine. */
    @QuerySqlField(index = true)
    private long id;

    /** Queryable field. Will be visible for SQL engine. */
    @QuerySqlField
    private String name;

    /** Will NOT be visible for SQL engine. */
    private int age;

    /** Indexed field. Will be visible for SQL engine. */
    @QuerySqlField(index = true)
    private Address address;
}
```

Где структура класса Address может выглядеть следующим образом:

```
public class Address {
```

```
    /** Indexed field. Will be visible for SQL engine. */
    @QuerySqlField (index = true)
    private String street;

    /** Indexed field. Will be visible for SQL engine. */
    @QuerySqlField(index = true)
    private int zip;
}
```

В приведенном выше примере аннотация @QuerySqlField(index = true) указана для всех полей класса Address, а также для объекта Address класса Person.

Это позволяет выполнять SQL-запросы, подобные следующим:

```
QueryCursor<List<?>> cursor = personCache.query(new SqlFieldsQuery( "select * from Person where street = 'street1'"));
```

Следует отметить, что не нужно указывать address.street в условии WHERE SQL-запроса. Это связано с тем, что поля класса Address преобразуются в плоскую коллекцию в таблице Person, что позволяет напрямую обращаться к полям Address в запросах.

Внимание! Если создаются индексы для вложенных объектов, выполнять операторы UPDATE или INSERT для таблицы невозможно.

После определения индексируемых и запрашиваемых полей их необходимо зарегистрировать в механизме SQL вместе с типами объектов, которым они принадлежат.

Чтобы указать, какие типы следует индексировать, необходимо передать соответствующие пары «ключ-значение» в метод CacheConfiguration.setIndexedTypes(), как показано в примере ниже.

```
// Preparing configuration.

CacheConfiguration<Long, Person> ccfg = new CacheConfiguration<>();

// Registering indexed type.
ccfg.setIndexedTypes(Long.class, Person.class);
```

Этот метод принимает только пары типов: один для класса ключей, а другой для класса значений. Примитивы передаются как упакованные (boxed) типы.

Примечание: В дополнение ко всем полям, отмеченным аннотацией @QuerySqlField, каждая таблица будет иметь два специальных предопределенных поля: _key и _val, которые представляют собой ссылки на целые объекты ключей и значений. Это полезно, например, когда один из них имеет примитивный тип, который необходимо отфильтровать его по значению. Для этого нужно выполнить запрос вида: SELECT * FROM Person WHERE _key = 100.

Примечание: Поскольку РЕД КВАНТ поддерживает двоичные объекты, нет необходимости добавлять классы индексированных типов в путь к классам узлов кластера. Механизм запросов SQL может обнаруживать значения индексированных и запрашиваемых полей, избегая десериализации объекта.

Чтобы настроить индекс с несколькими полями, который может ускорить запросы со сложными условиями, можно использовать аннотацию @QuerySqlField.Group. Можно

добавить несколько аннотаций @QuerySqlField.Group в orderedGroups, если необходимо, чтобы поле было частью более чем одной группы.

Например, в указанном ниже классе Person есть поле age, которое принадлежит индексированной группе с именем age_salary_idx с порядком группировки «0» и порядком сортировки по убыванию. Также, в той же группе есть поле salary с порядком группировки «3» и порядком сортировки по возрастанию. Кроме того, поле salary представляет собой индекс с одним столбцом (параметр index = true указывается в дополнение к объявлению orderedGroups). Порядок группировки order не обязательно должен быть конкретным числом. Он нужен только для сортировки полей внутри определенной группы.

```
public class Person implements Serializable {

  /** Indexed in a group index with "salary". */
  @QuerySqlField(orderedGroups = { @QuerySqlField.Group(name = "age_salary_idx", order = 0, descending =
true) })

  private int age;

  /** Indexed separately and in a group index with "age". */
  @QuerySqlField(index = true, orderedGroups = { @QuerySqlField.Group(name = "age_salary_idx", order =
3) })
  private double salary;
}
```

Примечание: Аннотирование поля с помощью @QuerySqlField.Group за пределами @QuerySqlField(orderedGroups={…}) не будет иметь никакого эффекта.

6.2.3 Настройка индексов с использованием сущностей запроса

Индексы и запрашиваемые поля также можно настроить с помощью класса org.apache.ignite.cache.QueryEntity, который удобен для конфигурации на основе Spring XML.

Все концепции, которые рассматриваются как часть конфигурации на основе указанных выше аннотаций, также применимы для подхода на основе QueryEntity. Кроме того, типы, поля которых настроены с помощью аннотации @QuerySqlField и зарегистрированы с помощью метода CacheConfiguration.setIndexedTypes(), внутренне преобразуются в сущности запросов.

В приведенном ниже примере показано, как определить индекс отдельного поля, групповые индексы и запрашиваемые поля.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration" id="ignite.cfg">

  <property name="cacheConfiguration">
    <bean class="org.apache.ignite.configuration.CacheConfiguration">
      <property name="name" value="Person"/>
      <!-- Configure query entities -->
      <property name="queryEntities">
        <list>
          <bean class="org.apache.ignite.cache.QueryEntity">
            <!-- Setting  the type of the key -->
            <property name="keyType" value="java.lang.Long"/>

            <property name="keyFieldName" value="id"/>
```

```
                <!-- Setting type of the value -->
                <property name="valueType" value="org.apache.ignite.examples.Person"/>

                <!-- Defining fields that will be either indexed or queryable.
                    Indexed fields are added to the 'indexes' list below.-->
                <property name="fields">
                  <map>
                    <entry key="id" value="java.lang.Long"/>
                    <entry key="name" value="java.lang.String"/>
                    <entry key="salary" value="java.lang.Float "/>
                  </map>
                </property>
                <!-- Defining indexed fields.-->
                <property name="indexes">
                  <list>
                    <!-- Single field (aka. column) index -->
                    <bean class="org.apache.ignite.cache.QueryIndex">
                      <constructor-arg value="name"/>
                    </bean>
                    <!-- Group index. -->
                    <bean class="org.apache.ignite.cache.QueryIndex">
                      <constructor-arg>
                        <list>
                          <value>id</value>
                          <value>salary</value>
                        </list>
                      </constructor-arg>
                      <constructor-arg value="SORTED"/>
                    </bean>
                  </list>
                </property>
              </bean>
            </list>
          </property>
        </bean>
      </property>
    </bean>
```

Краткое имя valueType используется в качестве имени таблицы в SQL-запросах. В этом случае имя таблицы будет Person (использование и определение имени схемы объясняется в пункте Схемы).

После определения QueryEntity можно выполнить SQL-запрос следующим образом:

```
SqlFieldsQuery qry = new SqlFieldsQuery("SELECT id, name FROM Person" + "WHERE id > 1500 LIMIT 10");
```

Примечание: Если нужно управлять индексами или сделать новые поля объекта видимыми для механизма SQL во время выполнения, необходимо использовать команду CREATE/DROP INDEX

6.2.4 НАСТРОЙКА INLINE SIZE ИНДЕКСА

Правильный inline size индекса может помочь ускорить запросы к проиндексированным полям.

В большинстве случаев нужно будет только установить inline size для индексов полей переменной длины, таких как строки или массивы. Значение по умолчанию — 10.

Можно изменить значение по умолчанию любым из следующих способов:

• Установить inline size для каждого индекса индивидуально;

• Установить свойство CacheConfiguration.sqlIndexMaxInlineSize для всех индексов в данном кэше;

• Установить системное свойство IGNITE_MAX_INDEX_PAYLOAD_SIZE для всех индексов в кластере.

Настройки применяются в порядке, указанном выше.

Также можно настроить inline size для каждого индекса отдельно, что перезапишет значение по умолчанию. Чтобы задать inline size индекса для пользовательского индекса, нужно использовать один из следующих методов. Во всех случаях значение задается в байтах.

• При использовании аннотаций:

```
@QuerySqlField(index = true, inlineSize = 13)

private String country;
```

• При использовании QueryEntity:

```
QueryIndex idx = new QueryIndex("country");

idx.setInlineSize(13);
queryEntity.setIndexes(Arrays.asList(idx));
```

• Если индексы создаются с помощью команды CREATE INDEX, для установки inline size можно использовать параметр INLINE_SIZE.

```
create index country_idx on Person (country) INLINE_SIZE 13;
```

6.2.5 Пользовательские ключи

Если для первичных ключей используются только предопределенные типы данных SQL, выполнять дополнительные манипуляции с конфигурацией схемы SQL не нужно. Эти типы данных определяются константой GridQueryProcessor.SQL_TYPES, как указано ниже.

К предопределенным типам данных SQL относятся:

• все примитивы и их оболочки, кроме char и Character;

• String;

• BigDecimal;

• byte[];

• java.util.Date, java.sql.Date, java.sql.Timestamp;

• java.util.UUID.

При решении ввести собственный сложный ключ и ссылаться на его поля из операторов DML, необходимо:

• Определить эти поля в QueryEntity тем же образом, как задаются поля для объекта value.

• Использовать новый параметр конфигурации QueryEntity.setKeyFields(..), чтобы отличать ключевые поля от полей значений.

Пример ниже показывает, как это сделать.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <property name="cacheConfiguration">
    <bean class="org.apache.ignite.configuration.CacheConfiguration">
      <property name="name" value="personCache"/>
      <!-- Configure query entities -->
      <property name="queryEntities">
        <list>
          <bean class="org.apache.ignite.cache.QueryEntity">
            <!-- Registering key's class. -->
            <property name="keyType" value="CustomKey"/>
            <!-- Registering value's class. -->
            <property name="valueType" value="org.apache.ignite.examples.Person"/>
            <!-- Defining all the fields that will be accessible from DML. -->
            <property name="fields">
              <map>
                <entry key="firstName" value="java.lang.String"/>
                <entry key="lastName" value="java.lang.String"/>
                <entry key="intKeyField" value="java.lang.Integer"/>
                <entry key="strKeyField" value="java.lang.String"/>
              </map>
            </property>
            <!-- Defining the subset of key's fields -->
            <property name="keyFields">
              <set>
                <value>intKeyField</value>
                <value>strKeyField</value>
              </set>
            </property>
          </bean>
        </list>
      </property>
    </bean>
  </property>
</bean>
```

Примечание: Если пользовательский ключ можно сериализовать в двоичную форму, Ingnite вычисляет его хэш-код и автоматически реализует метод equals().

Однако, если тип ключа Externalizable (является экстернализуемым, то есть расширяет сериализуемый) и его нельзя сериализовать в двоичную форму, необходимо реализовать методы hashCode и equals вручную. См. пункт Работа с бинарными объектами.

6.3 API ДЛЯ РАБОТЫ С SQL ЗАПРОСАМИ

Помимо использования драйвера JDBC, разработчики Java могут использовать API SQL РЕД КВАНТ для запроса и изменения данных, хранящихся в РЕД КВАНТ.

Класс SqlFieldsQuery — это интерфейс для выполнения запросов SQL и навигации по результатам. SqlFieldsQuery выполняется с помощью метода IgniteCache.query(SqlFieldsQuery), который возвращает курсор запроса.

6.3.1 НАСТРОЙКА ЗАПРАШИВАЕМЫХ ПОЛЕЙ

Если нужно запросить кэш с помощью операторов SQL, то необходимо определить, какие поля объектов-значений доступны для запроса. Запрашиваемые поля — это поля модели данных, которые механизм SQL может «видеть» и запрашивать.

Примечание: Если таблицы создаются с помощью инструментов JDBC или SQL, определять запрашиваемые поля не нужно.

Примечание: Возможности индексирования предоставляются модулем 'ignite-indexing'. Если РЕД КВАНТ запускается из кода Java, необходимо добавить этот модуль в путь к классам приложения.

В Java запрашиваемые поля можно настроить двумя способами:

- с помощью использования аннотаций;
- путем определения объектов запроса.

Чтобы сделать определенные поля запрашиваемыми, необходимо указать аннотацию @QuerySqlField для полей в определении класса значения и вызвать CacheConfiguration.setIndexedTypes(…).

```
class Person implements Serializable {

  /** Indexed field. Will be visible to the SQL engine. */
  @QuerySqlField(index = true)
  private long id;

  /** Queryable field. Will be visible to the SQL engine. */
  @QuerySqlField
  private String name;

  /** Will NOT be visible to the SQL engine. */
  private int age;

  /**
   * Indexed field sorted in descending order. Will be visible to the SQL engine.
   */
  @QuerySqlField(index = true, descending = true)
  private float salary;
}

public static void main(String[] args) {
  Ignite ignite = Ignition.start();
  CacheConfiguration<Long, Person> personCacheCfg = new CacheConfiguration<Long, Person>();
  personCacheCfg.setName("Person");
```

```
    personCacheCfg.setIndexedTypes(Long.class, Person.class);
    IgniteCache<Long, Person> cache = ignite.createCache(personCacheCfg);
}
```

Обязательно нужно вызвать CacheConfiguration.setIndexedTypes(…), чтобы механизм SQL узнал об аннотированных полях.

Запрашиваемые поля можно определить с помощью класса QueryEntity. Сущноссти запроса можно настроить с помощью конфигурации XML.

6.3.2 Запросы

Чтобы выполнить запрос на выборку в кэше, нужно создать объект SqlFieldsQuery, передающий строку запроса конструктору, и запустить cache.query(…). Следует отметить, что в следующем примере кэш Person должен быть настроен так, чтобы он был виден механизму SQL.

```
IgniteCache<Long, Person> cache = ignite.cache("Person");


SqlFieldsQuery sql = new SqlFieldsQuery(
      "select concat(firstName, ' ', lastName) from Person");

// Iterate over the result set.
try (QueryCursor<List<?>> cursor = cache.query(sql)) {
   for (List<?> row : cursor)
      System.out.println("personName=" + row.get(0));
}
```

SqlFieldsQuery возвращает курсор, который перебирает результаты, соответствующие SQL-запросу.

Чтобы принудительно выполнить запрос локально, необходимо использовать SqlFieldsQuery.setLocal(true). В этом случае запрос выполняется для данных, хранящихся на узле, где выполняется запрос. Это означает, что результаты запроса почти всегда неполные. Использовать локальный режим необходимо только при понимании сути этого ограничения.

Запросы SELECT, используемые в операторах INSERT и MERGE, а также запросы SELECT, сгенерированные операциями UPDATE и DELETE, распределяются и выполняются либо в сколоцированном, либо в не сколоцированном распределенном режиме.

Однако, если есть подзапрос, который выполняется как часть условия WHERE, то он может выполняться только в режиме колокации.

Например, есть следующий запрос:

```
DELETE FROM Person WHERE id IN

(SELECT personId FROM Salary s WHERE s.amount > 2000);
```

Механизм SQL генерирует запрос SELECT для получения списка записей, подлежащих удалению. Запрос распределяется и выполняется по всему кластеру и выглядит следующим образом:

```
SELECT _key, _val FROM Person WHERE id IN
(SELECT personId FROM Salary s WHERE s.amount > 2000);
```

Однако подзапрос из условия IN (SELECT personId FROM Salary…) не распределяется и выполняется по локальному набору данных, доступному на узле.

6.3.3 Вставка, обновление, удаление и слияние

С помощью SqlFieldsQuery можно выполнять команды DML для изменения данных:

- вставка:

```
IgniteCache<Long, Person> cache = ignite.cache("personCache");

cache.query(
    new SqlFieldsQuery("INSERT INTO Person(id, firstName, lastName) VALUES(?, ?, ?)")
        .setArgs(1L, "John", "Smith"))
    .getAll();
```

- обновление:

```
IgniteCache<Long, Person> cache = ignite.cache("personCache");

cache.query(new SqlFieldsQuery("UPDATE Person set lastName = ? " + "WHERE id >= ?")
    .setArgs("Jones", 2L)).getAll();
```

- удаление:

```
IgniteCache<Long, Person> cache = ignite.cache("personCache");

cache.query(new SqlFieldsQuery("DELETE FROM Person " + "WHERE id >= ?").setArgs(2L))
    .getAll();
```

- слияние:

```
IgniteCache<Long, Person> cache = ignite.cache("personCache");

cache.query(new SqlFieldsQuery("MERGE INTO Person(id, firstName, lastName)"
    + " values (1, 'John', 'Smith'), (5, 'Mary', 'Jones')")).getAll();
```

При использовании SqlFieldsQuery для выполнения операторов DDL необходимо вызвать getAll() для курсора, возвращенного из метода query(…).

6.3.4 Указание схемы

По умолчанию любой оператор SELECT, выполняемый через SqlFieldsQuery, осуществляется в соответствии со схемой PUBLIC. Однако если участвующая в запросе таблица находится в другой схеме, можно указать схему, вызвав SqlFieldsQuery.setSchema(…). В этом случае оператор выполняется в заданной схеме.

```
SqlFieldsQuery sql = new SqlFieldsQuery("select name from City").setSchema("PERSON");
```

В качестве альтернативы можно определить схему в запросе:

```
SqlFieldsQuery sql = new SqlFieldsQuery("select name from Person.City");
```

6.3.5 Создание таблиц

Любой поддерживаемый оператор DDL можно передать в SqlFieldsQuery и выполнить его в кэше, как показано ниже.

```
IgniteCache<Long, Person> cache = ignite

    .getOrCreateCache(new CacheConfiguration<Long, Person>().setName("Person"));

// Creating City table.
cache.query(new SqlFieldsQuery(
    "CREATE TABLE City (id int primary key, name varchar, region varchar)")).getAll();
```

С точки зрения схемы SQL в результате выполнения кода создаются следующие таблицы:

- Таблица «Person» в схеме «Person» (если она не была создана ранее).
- Таблица «City» в схеме «Person».

Чтобы запросить таблицу «City», нужно использовать такие запросы, как `select * from Person.City` или `new SqlFieldsQuery("select * from City").setSchema("PERSON")` (следует обратить внимание на верхний регистр).

6.3.6 Отмена запросов

Есть два способа отменить длительные запросы.

Первый подход заключается в предотвращении выполнения неуправляемых запросов путем установки времени ожидания выполнения запроса.

```
SqlFieldsQuery query = new SqlFieldsQuery("SELECT * from Person");


// Setting query execution timeout
query.setTimeout(10_000, TimeUnit.SECONDS);
```

Второй подход — остановить запрос с помощью QueryCursor.close().

```
SqlFieldsQuery query = new SqlFieldsQuery("SELECT * FROM Person");


// Executing the query
QueryCursor<List<?>> cursor = cache.query(query);

// Halting the query that might be still in progress.
cursor.close();
```

6.4 Распределённые join’ы

Распределённые join’ы — это запрос SQL с условием join, которое объединяет две или более партиционированных таблиц. Если таблицы объединяются по столбцу партиционирования (affinity-ключ), такой join называется сколоцированным join’ом. В противном случае это называется не сколоцированным join’ом.

Сколоцированные join’ы более эффективны, поскольку они могут быть эффективно распределены между узлами кластера.

По умолчанию РЕД КВАНТ расценивает каждый запрос с join’ом как сколоцированный join и выполняет его соответствующим образом.

Внимание: Если запрос не является сколоцированным, необходимо включить режим выполнения запроса без колокации, установив SqlFieldsQuery.setDistributedJoins(true); в противном случае результаты выполнения запроса могут быть неверными.

Внимание! Если к таблицам часто применяется join, рекомендуется партицировать их по тому столбцу, по которому таблицы объединяются join’ом.

Не сколоцированные join’ы должны быть зарезервированы для случаев, когда невозможно использовать сколоцированные join’ы.

6.4.1 Сколоцированные join’ы

На рисунке 28 показана процедура выполнения сколоцированного join’а. Сколоцированный join (Q) отправляется на все узлы, на которых хранятся данные, соответствующие условию запроса. Затем запрос выполняется по локальному набору данных на каждом узле (E(Q)). Результаты (R) агрегируются на узле, инициировавшем запрос (клиентском узле).

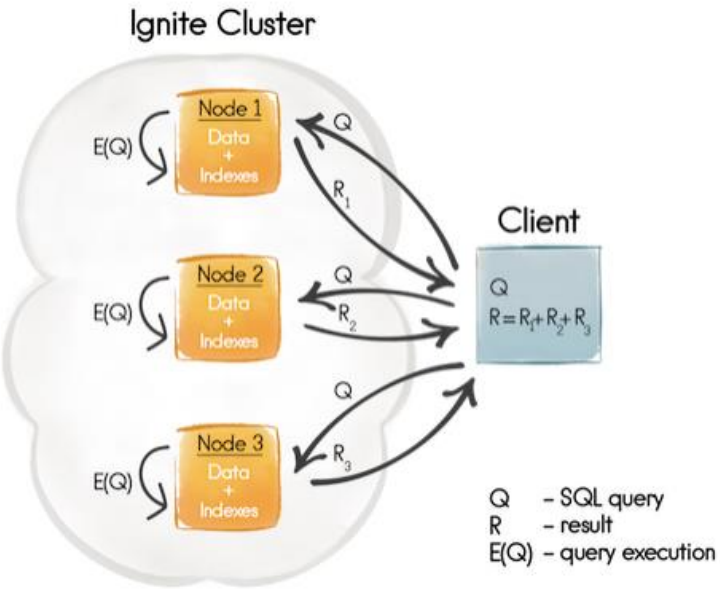

Рисунок 28 - Сколоцированный join

6.4.2 Не сколоцированные join’ы

Если запрос выполняется в режиме без колокации, механизм SQL выполняет его локально на всех узлах, в которых хранятся данные, соответствующие условию запроса. Но поскольку данные не сколоцированы, каждый узел будет запрашивать (локально) отсутствующие данные у других узлов, отправляя либо широковещательные, либо одноадресные запросы. Этот процесс изображен на рисунке 29.

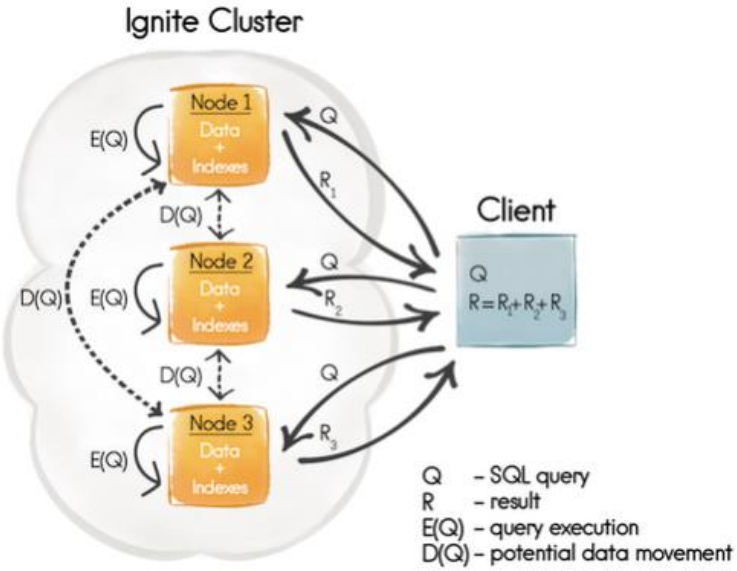

Рисунок 29 - Не сколоцированный join

Если join выполняется по первичному ключу или affinity-ключу, узлы отправляют одноадресные запросы, поскольку в этом случае узлы знают расположение недостающих данных. В противном случае узлы отправляют широковещательные запросы. По соображениям производительности как широковещательные, так и одноадресные запросы объединяются в пакеты.

Необходимо включить режим выполнения запроса без колокации, задав параметр JDBC/ODBC или с помощью вызова SqlFieldsQuery.setDistributedJoins(true), если используется SQL API»

Внимание: Если для столбца из реплицированной таблицы используется не сколоцированный join, этот столбец должен иметь индекс. В противном случае будет получено исключение.

6.5 Пользовательские SQL функции

Механизм SQL может расширить набор SQL функций, определенный спецификацией ANSI-99, путем добавления пользовательских SQL функций, написанных на Java.

Пользовательская SQL функция — это общедоступный статический метод, отмеченный аннотацией @QuerySqlFunction.

```
static class SqlFunctions {

   @QuerySqlFunction
   public static int sqr(int x) {
      return x * x;
   }
}
```

Класс, которому принадлежит пользовательская SQL функция, должен быть зарегистрирован в CacheConfiguration. Для этого необходимо использовать метод setSqlFunctionClasses(…).

```
// Preparing a cache configuration.

CacheConfiguration cfg = new CacheConfiguration("myCache");

// Registering the class that contains custom SQL functions.
cfg.setSqlFunctionClasses(SqlFunctions.class);

IgniteCache cache = ignite.createCache(cfg);
```

После того, как будет развернут кэш с указанной выше конфигурацией, можно вызывать пользовательскую функцию из SQL-запросов:

```
// Preparing the query that uses the custom defined 'sqr' function.

SqlFieldsQuery query = new SqlFieldsQuery("SELECT name FROM myCache WHERE sqr(size) > 100");

// Executing the query.
cache.query(query).getAll();
```

Примечание: Классы, зарегистрированные с помощью CacheConfiguration.setSqlFunctionClasses(…), должны быть добавлены в путь к классам на все узлы, где могут выполняться пользовательские функции. В противном случае при попытке выполнить пользовательскую функцию возникнет ошибка ClassNotFoundException.

6.6 SQL СТАТИСТИКА

РЕД КВАНТ может вычислять статистику по запросу и использовать ее для построения оптимального плана SQL-запроса. Это позволяет значительно ускорить выполнение SQL-запроса.

Без статистики планировщик выполнения SQL-запросов пытается предсказать избирательность условий запроса, используя только общие эвристики. Чтобы получить лучшие планы, необходимо убедиться, что использование статистики включено, и настроить сбор статистики для таблиц, участвующих в запросе.

Статистика проверяется и обновляется каждый раз после одного из следующих действий:

- Запуск узла;
- Изменение топологии;
- Изменение конфигурации.

Узел проверяет разделы и собирает по каждому из них статистику, которую можно использовать при оптимизации SQL-запросов.

6.6.1 НАСТРОЙКА СТАТИСТИКИ

По умолчанию функция статистики включена.

Статистика хранится локально, в то время как параметры конфигурации статистики хранятся по всему кластеру.

Чтобы включить или отключить статистику при использовании кластера, нужно выполнить следующую команду, указав значения ON, OFF или NO_UPDATE:

```
control.sh --property set --name 'statistics.usage.state' --val 'ON'
```

Чтобы увидеть состояние использования статистики, нужно выполнить следующую команду:

```
control.sh --property get --name 'statistics.usage.state'
```

6.6.2 Переопределение статистики

Собранные значения можно переопределить, указав дополнительные параметры в команде ANALYZE. Указанные значения переопределяют собранные ранее значения на каждом узле в STATISTICS_LOCAL_DATA (эти данные используются оптимизатором запросов SQL), но не в STATISTICS_PARTITION_DATA (сохраняет реальную статистику по разделам). Затем переопределенные значения используются оптимизатором запросов SQL.

Каждая команда ANALYZE переопределяет все подобные значения для своих целей. Например, если значение TOTAL уже переопределено и необходимо переопределить значение DISTINCT, следует использовать оба параметра в одной команде ANALYZE. Чтобы установить разные значения для разных столбцов, нужно использовать несколько команд ANALYZE следующим образом:

```
ANALYZE MY_TABLE(COL_A) WITH 'DISTINCT=5,NULLS=6';
ANALYZE MY_TABLE(COL_B) WITH 'DISTINCT=500,NULLS=1000,TOTAL=10000';
```

6.6.3 Устаревание статистики

Каждый раздел имеет специальный счетчик для отслеживания общего количества измененных (вставленных, удаленных или обновленных) строк. Если общее количество измененных строк превышает MAX_CHANGED_PARTITION_ROWS_PERCENT, раздел анализируется снова. После этого узел снова агрегирует статистику, чтобы получить новую статистику.

Чтобы настроить параметр MAX_CHANGED_PARTITION_ROWS_PERCENT, нужно еще раз запустить команду ANALYZE с нужным значением параметра.

По умолчанию используется параметр DEFAULT_OBSOLESCENCE_MAX_PERCENT = 15.

Эти параметры применяются для всех указанных целей.

Примечание: Поскольку статистика агрегируется путем полного сканирования каждого раздела, рекомендуется отключать функцию устаревания статистики при работе с малым количеством изменяющихся строк. Это особенно актуально в случае работы с большими объемами данных, когда полное сканирование может привести к падению производительности.

Чтобы сэкономить ресурсы ЦП при отслеживании устаревания, необходимо использовать состояние NO_UPDATE:

```
control.sh --property set --name 'statistics.usage.state' --val 'NO_UPDATE'
```

6.6.4 Получение лучшего плана выполнения с использованием статистики

Указанные ниже шаги демонстрируют пример получения оптимизированного плана выполнения для базового запроса.

1. Создать таблицу и вставить в нее данные:

```
CREATE TABLE statistics_test(col1 int PRIMARY KEY, col2 varchar, col3 date);

INSERT INTO statistics_test(col1, col2, col3) VALUES(1, 'val1', '2019-01-01');
INSERT INTO statistics_test(col1, col2, col3) VALUES(2, 'val2', '2019-03-01');
INSERT INTO statistics_test(col1, col2, col3) VALUES(3, 'val3', '2019-06-01');
INSERT INTO statistics_test(col1, col2, col3) VALUES(4, 'val4', '2019-09-01');
INSERT INTO statistics_test(col1, col2, col3) VALUES(5, 'val5', '2019-12-01');
INSERT INTO statistics_test(col1, col2, col3) VALUES(6, 'val6', '2020-02-01');
INSERT INTO statistics_test(col1, col2, col3) VALUES(7, 'val7', '2020-05-01');
INSERT INTO statistics_test(col1, col2, col3) VALUES(8, 'val8', '2020-08-01');
INSERT INTO statistics_test(col1, col2, col3) VALUES(9, 'val9', '2020-11-01');
```

2. Создать индексы для каждого столбца:

```
CREATE INDEX st_col1 ON statistics_test(col1);
CREATE INDEX st_col2 ON statistics_test(col2);
CREATE INDEX st_col3 ON statistics_test(col3);
```

3. Получить план выполнения базового запроса:

Следует отметить, что значение col2 меньше максимального значения в таблице, а значение col3 выше максимального. Таким образом, весьма вероятно, что второе условие не возвращает результата, что повышает его селективность. Поэтому база данных должна использовать индекс st_col3.

```
EXPLAIN SELECT * FROM statistics_test WHERE col2 > 'val2' AND col3 > '2020-12-01'

SELECT
"__Z0"."COL1" AS "__C0_0",
"__Z0"."COL2" AS "__C0_1",
"__Z0"."COL3" AS "__C0_2"
FROM "PUBLIC"."STATISTICS_TEST" "__Z0"
/* PUBLIC.ST_COL2: COL2 > 'val2' */
WHERE ("__Z0"."COL2" > 'val2')
AND ("__Z0"."COL3" > DATE '2020-12-01')
```

Без собранной статистики в базе данных недостаточно информации для выбора правильного индекса (поскольку оба индекса имеют одинаковую избирательность с точки зрения планировщика). Эта проблема исправлена ниже.

4. Собрать статистику для таблицы Statistics_test:

```
ANALYZE statistics_test;
```

5. Снова получить план выполнения и убедиться, что выбран индекс st_col3:

```
EXPLAIN SELECT * FROM statistics_test WHERE col2 > 'val2' AND col3 > '2020-12-01'

SELECT
"__Z0"."COL1" AS "__C0_0",
```

```
"__Z0"."COL2" AS "__C0_1",
"__Z0"."COL3" AS "__C0_2"
FROM "PUBLIC"."STATISTICS_TEST" "__Z0"
/* PUBLIC.ST_COL3: COL3 > DATE '2020-12-01' */
WHERE ("__Z0"."COL2" > 'val2')
AND ("__Z0"."COL3" > DATE '2020-12-01')
```

6.7 SQL ДВИЖОК НА БАЗЕ APACHE CALCITE

Начиная с версии 2.13, РЕД КВАНТ включает в себя новый движок SQL, основанный на платформе Apache Calcite.

Apache Calcite — это фреймворк динамического управления данными, который в основном служит посредником между приложениями, одним или несколькими хранилищами данных и механизмами обработки данных.

Текущий движок SQL на основе H2 имеет ряд фундаментальных ограничений выполнения запросов в распределенной среде. Чтобы устранить эти ограничения, был реализован новый движок SQL. Новый движок использует инструменты, предоставляемые Apache Calcite, для разбора и планирования запросов. Он также имеет новый flow выполнения запросов.

Внимание! Механизм запросов на основе Calcite в настоящее время находится в состоянии бета-тестирования.

6.7.1 БИБЛИОТЕКИ МОДУЛЕЙ CALCITE

Чтобы использовать движок на основе Calcite, нужно убедиться, что библиотеки модулей Calcite находятся в пути к классам.

Внимание! В настоящее время часть функциональности модуля ignite-indexing используется повторно. Это означает, что модуль ignite-indexing также должен присутствовать в пути к классам.

При запуске отдельного узла нужно переместить папки optional/ignite-calcite и optional/ignite-slf4j в папку libs перед запуском скрипта ignite.{sh|bat}. В этом случае содержимое папки модуля добавляется в путь к классам.

Если для управления зависимостями проекта используется Maven, можно добавить зависимость модуля Calcite следующим образом: заменить ${ignite.version} актуальной версией РЕД КВАНТ:

```
<dependency>

  <groupId>org.apache.ignite</groupId>
  <artifactId>ignite-calcite</artifactId>
  <version>${ignite.version}</version>
</dependency>
```

6.7.2 НАСТРОЙКА ДВИЖКОВ ЗАПРОСОВ

Чтобы включить движок, нужно добавить явный экземпляр CalciteQueryEngineConfiguration в свойство SqlConfiguration.QueryEnginesConfiguration.

Ниже приведен пример конфигурации двух настроенных механизмов запросов (на основе H2 и на основе Calcite), где движок на основе Calcite выбран в качестве движка по умолчанию:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <property name="sqlConfiguration">
    <bean class="org.apache.ignite.configuration.SqlConfiguration">
      <property name="queryEnginesConfiguration">
        <list>
          <bean class="org.apache.ignite.indexing.IndexingQueryEngineConfiguration">
            <property name="default" value="false"/>
          </bean>
          <bean class="org.apache.ignite.calcite.CalciteQueryEngineConfiguration">
            <property name="default" value="true"/>
          </bean>
        </list>
      </property>
    </bean>
  </property>
  ...
</bean>
```

6.7.3 Маршрутизация запросов к Query Engine

Обычно все запросы направляются в настроенный по умолчанию движок запросов. Если с помощью queryEnginesConfiguration настроено более одного движка, можно использовать другой движок вместо настроенного по умолчанию для отдельных запросов или для всего соединения.

Чтобы выбрать движок запросов для подключения JDBC, нужно использовать параметр соединения queryEngine:

```
jdbc:ignite:thin://127.0.0.1:10800?queryEngine=calcite
```

Чтобы настроить движок запросов для подключения ODBC, нужно использовать свойство QUERY_ENGINE:

```
[IGNITE_CALCITE]

DRIVER={РЕД КВАНТ};
SERVER=127.0.0.1;
PORT=10800;
SCHEMA=PUBLIC;
QUERY_ENGINE=CALCITE
```

Чтобы выбрать определенный движок для выполнения индивидуальных запросов, нужно использовать подсказку QUERY_ENGINE:

```
SELECT /*+ QUERY_ENGINE('calcite') */ fld FROM table;
```

6.7.4 Справочник по SQL

Операторы языка определения данных (DDL) совместимы со старым движком на основе H2. Описание синтаксиса DDL можно найти **Приложение B. DDL**.

Новый движок SQL в основном наследует синтаксис запросов языка управления данными (DML) из среды Apache Calcite.

В большинстве случаев синтаксис запросов совместим со старым движком SQL. Но существуют некоторые различия между диалектами DML в движке на основе H2 и движке на основе Calcite.

Движок SQL на основе Calcite в настоящее время поддерживает типы функций, представленные в таблице 8.

Таблица 8 - Типы функций, поддерживаемые движком SQL на основе Calcite

Группа	Список функций
Функции агрегирования	COUNT, SUM, AVG, MIN, MAX, ANY_VALUE
Строковые функции	UPPER, LOWER, INITCAP, TO_BASE64, FROM_BASE64, MD5, SHA1, SUBSTRING, LEFT, RIGHT, REPLACE, TRANSLATE, CHR, CHAR_LENGTH, CHARACTER_LENGTH, LENGTH, CONCAT, OVERLAY, POSITION, ASCII, REPEAT, SPACE, STRCMP, SOUNDEX, DIFFERENCE, REVERSE, TRIM, LTRIM, RTRIM, REGEXP_REPLACE
Математические функции	MOD, EXP, POWER, LN, LOG10, ABS, RAND, RAND_INTEGER, ACOS, ASIN, ATAN, ATAN2, SQRT, CBRT, COS, COSH, COT, DEGREES, RADIANS, ROUND, SIGN, SIN, SINH, TAN, TANH, TRUNCATE, PI
Функции даты и времени	EXTRACT, FLOOR, CEIL, TIMESTAMPADD, TIMESTAMPDIFF, LAST_DATE, DAYNAME, MONTHNAME, DAYOFMONTH, DAYOFWEEK, DAYOFYEAR, YEAR, QUARTER, MONTH, WEEK, HOUR, MINUTE, SECOND, TIMESTAMP_SECONDS, TIMESTAMP_MILLIS, TIMESTAMP_MICROS, UNIX_SECONDS, UNIX_MILLIS, UNIX_MICROS, UNIX_DATE, DATE_FROM_UNIX_DATE, DATE, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_DATE, LOCALTIME, LOCALTIMESTAMP
Функции XML	EXTRACTVALUE, XMLTRANSFORM, EXTRACT, EXISTSNODE
Функции JSON	JSON_VALUE, JSON_QUERY, JSON_TYPE, JSON_EXISTS, JSON_DEPTH, JSON_KEYS, JSON_PRETTY, JSON_LENGTH, JSON_REMOVE, JSON_STORAGE_SIZE, JSON_OBJECT, JSON_ARRAY
Прочие функции	ROW, CAST, COALESCE, NVL, NULLIF, CASE, DECODE, LEAST, GREATEST, COMPRESS, OCTET_LENGTH, TYPEOF, QUERY_ENGINE

Типы данных, поддерживаемые движком SQL на основе Calcite представлены в таблице 9.

Таблица 9 - Типы данных, поддерживаемые механизмом SQL на основе Calcite

Тип данных	Сопоставленный с классом Java
BOOLEAN	java.lang.Boolean
DECIMAL	java.math.BigDecimal
DOUBLE	java.lang.Double
REAL/FLOAT	java.lang.Float
INT	java.lang.Integer
BIGINT	java.lang.Long
SMALLINT	java.lang.Short
TINYINT	java.lang.Byte
CHAR/VARCHAR	java.lang.String
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
INTERVAL YEAR TO MONTH	java.time.Period
INTERVAL DAY TO SECOND	java.time.Duration
BINARY/VARBINARY	byte[]
UUID	java.util.UUID
OTHER	java.lang.Object

6.8 Continuous Queries

Continuous query — это запрос, который отслеживает изменения данных, происходящие в кэше. После запуска continuous query будут приходить уведомления обо всех изменениях данных, которые попадают в фильтр запроса.

Все события обновления передаются локальному слушателю (listener), который должен быть зарегистрирован в запросе. Реализация continuous query гарантирует однократную доставку события локальному слушателю.

Также можно указать удаленный фильтр, чтобы сузить диапазон записей, которые отслеживаются на наличие обновлений.

Внимание: Continuous queries имеют ряд функциональных ограничений при использовании кэшей с включенным MVCC.

6.8.1 Локальный слушатель

Когда кэш изменяется (запись вставляется, обновляется или удаляется), событие отправляется локальному слушателю continuous query, чтобы приложение могло реагировать

соответствующим образом. Локальный слушатель выполняется на узле, который инициировал запрос.

Следует отметить, что при запуске без локального слушателя, continuous query вызовет исключение.

```
IgniteCache<Integer, String> cache = ignite.getOrCreateCache("myCache");


ContinuousQuery<Integer, String> query = new ContinuousQuery<>();

query.setLocalListener(new CacheEntryUpdatedListener<Integer, String>() {

  @Override
  public void onUpdated(Iterable<CacheEntryEvent<? extends Integer, ? extends String>> events)
    throws CacheEntryListenerException {
    // react to the update events here
  }
});

cache.query(query);
```

6.8.2 Инициирующий запрос

Можно указать инициирующий запрос, который будет выполняться перед тем, как continuous query будет зарегистрирован в кластере и перед тем, как начнут поступать обновления. Чтобы установить инициирующий запрос, нужно использовать метод ContinuousQuery.setInitialQuery(…).

Как и «сканирующие запросы» (Scan queries), continuous query выполняется с помощью метода query(), который возвращает курсор. Когда задан инициирующий запрос, этот курсор можно использовать для перебора результатов инициирующего запроса.

```
IgniteCache<Integer, String> cache = ignite.getOrCreateCache("myCache");


ContinuousQuery<Integer, String> query = new ContinuousQuery<>();

// Setting an optional initial query.
// The query will return entries for the keys greater than 10.
query.setInitialQuery(new ScanQuery<>((k, v) -> k > 10));

//mandatory local listener
query.setLocalListener(events -> {
});

try (QueryCursor<Cache.Entry<Integer, String>> cursor = cache.query(query)) {
  // Iterating over the entries returned by the initial query
  for (Cache.Entry<Integer, String> e : cursor)
    System.out.println("key=" + e.getKey() + ", val=" + e.getValue());
}
```

6.8.3 Удаленный фильтр

Этот фильтр выполняется для каждого обновленного ключа и определяет, следует ли передавать обновление на локальный слушатель запроса. Если фильтр возвращает true, локальный слушатель получает уведомление об обновлении.

Во избежание избыточности фильтр выполняется как для основной, так и для резервной версии (если настроены резервные копии) ключа. По этой причине удаленный фильтр можно использовать в качестве удаленного слушателя событий обновления.

```
ContinuousQuery<Integer, String> qry = new ContinuousQuery<>();


qry.setLocalListener(events ->
  events.forEach(event -> System.out.format("Entry: key=[%s] value=[%s]\n", event.getKey(), event.getValue()))
);

qry.setRemoteFilterFactory(new Factory<CacheEntryEventFilter<Integer, String>>() {
  @Override
  public CacheEntryEventFilter<Integer, String> create() {
    return new CacheEntryEventFilter<Integer, String>() {
      @Override
      public boolean evaluate(CacheEntryEvent<? extends Integer, ? extends String> e) {
        System.out.format("the value for key [%s] was updated from [%s] to [%s]\n", e.getKey(),
e.getOldValue(), e.getValue());
        return true;
      }
    };
  }
});
```

Примечание: Чтобы использовать удаленные фильтры, необходимо убедиться, что определения классов фильтров доступны на серверных узлах. Это можно сделать двумя способами:

- Добавить классы в путь к классам каждого серверного узла;
- Включить загрузку однорангового класса.

6.8.4 Удаленный преобразователь

По умолчанию continuous queries отправляют весь обновленный объект локальному слушателю. Это может привести к чрезмерному использованию сети, особенно если объект очень большой. Более того, приложениям часто требуется только подмножество полей объекта.

Чтобы решить эти проблемы, можно использовать continuous query с преобразователем. Преобразователь — это функция, которая выполняется на удаленных узлах для каждого обновляемого объекта и отправляет обратно только результаты преобразования.

```
IgniteCache<Integer, Person> cache = ignite.getOrCreateCache("myCache");


// Create a new continuous query with a transformer.
ContinuousQueryWithTransformer<Integer, Person, String> qry = new ContinuousQueryWithTransformer<>();

// Factory to create transformers.
```

```
Factory factory = FactoryBuilder.factoryOf(
   // Return one field of a complex object.
   // Only this field will be sent over to the local listener.
   (IgniteClosure<CacheEntryEvent, String>)
      event -> ((Person)event.getValue()).getName()
);

qry.setRemoteTransformerFactory(factory);

// Listener that will receive transformed data.
qry.setLocalListener(names -> {
   for (String name : names)
      System.out.println("New person name: " + name);
});
```

Перед тем как использовать преобразователи, необходимо убедиться, что определения классов преобразователей доступны на узлах сервера. Это можно сделать двумя способами:

- Добавить классы в путь к классам каждого узла сервера;
- Включить загрузку однорангового класса.

6.8.5 Гарантия доставки событий

Continuous queries обеспечивают однократную семантику доставки событий локальным слушателям клиентов.

Как основной, так и резервный узлы поддерживают очередь обновлений, в которой хранятся события, которые обрабатываются continuous queries на стороне сервера, но еще не доставлены клиентам. Например, по какой-либо причине произошел сбой основного узла или изменилась топология кластера. В этом случае каждый узел резервного копирования сбрасывает содержимое своей очереди обновлений на клиент, гарантируя, что каждое событие будет доставлено на локальный слушатель клиента.

РЕД КВАНТ управляет специальным счетчиком обновлений для каждого раздела, что помогает избежать дублирования уведомлений. Как только запись в каком-либо разделе обновляется, счетчик для этого раздела увеличивается как на основном, так и на резервном узлах. Значение этого счетчика также отправляется клиенту вместе с уведомлением о событии. Таким образом, клиент может пропускать уже обработанные события. Как только клиент подтверждает, что событие получено, основной и резервный узлы удаляют запись для этого события из своих очередей резервного копирования.

7 РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛЕНИЯ

Как правило, распределенные вычисления — это система, предназначенная для распределения прикладных задач по кластеру компьютеров и параллельной координации их задач. Основной принцип распределенных вычислений заключается в разделении задачи на несколько частей и их параллельном выполнении на разных узлах кластера. На рисунке 30 представлено очень высокоуровневое представление типичной распределенной системы, где каждый компьютер имеет собственную локальную память, а обмен информацией может осуществляться только в обход сообщений от одного узла к другому, используя доступные каналы связи.

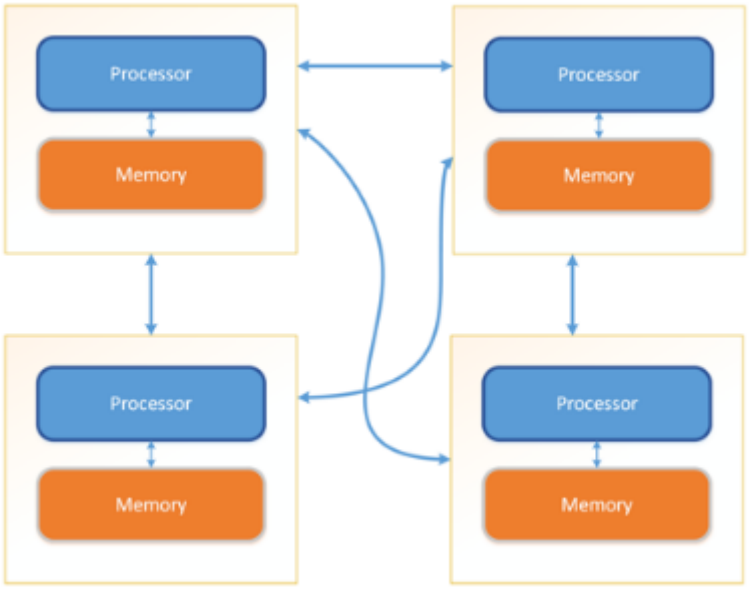

Рисунок 30 - Структура распределенной системы

Главное преимущество заключается в том, что вычисления будут выполняться быстрее, поскольку теперь он может использовать ресурсы из локального узла (локальные данные), а также параллельно со всех узлов грида. РЕД КВАНТ как вычислительный грид предлагает несколько преимуществ и отличий по сравнению с классическими подходами к созданию распределенной вычислительной системы. Некоторые из них:

• Классический подход требует распределения ресурсов вручную и распределения задач по кластерам.

• Необходимо разработать систему мониторинга запущенных в кластере задач и перезапускать их классическим способом в случае ошибок.

• Очень сложно использовать локальность данных с вычислительной задачей.

РЕД КВАНТ предоставляет набор простых API-интерфейсов, которые позволяют пользователю распределять вычисления и обработку данных между несколькими узлами в кластере для повышения производительности. Ключевые особенности распределенных вычислений РЕД КВАНТ включают в себя:

1. Локальность данных: задачи могут быть привязаны к локальным данным.

2. Fork-join: для фреймворка fork-join есть стандартный механизм, позволяющий надежно передавать данные между узлами и перезапускать задачу в случае сбоя.

3. Автоматическое развертывание: нет необходимости устанавливать код приложения/задачи на сервер, достаточно подключиться к серверу, и любой новый код будет автоматически принят. Пользователи обычно могут просто выполнить задачу с одного узла грида, и по мере выполнения задачи она проникает в грид. Все классы и ресурсы также автоматически развертываются.

4. Checkpoint: задача может сохранять свои состояния в checkpoint и восстанавливать их из checkpoint в случае сбоя. Эта функция очень полезна для долго выполняющейся задачи.

5. Отказоустойчивость и восстановление: предоставляет встроенную отказоустойчивость и восстановление для всех задач в кластере.

6. Масштабируемость: масштабируемость до произвольного числа узлов обработки.

7. Асинхронная связь: API поддерживает модель асинхронной связи для обеспечения высокого уровня параллелизма.

8. Балансировка нагрузки: предоставляет набор алгоритмов, таких как циклический, случайный или адаптивный, для балансировки нагрузки в кластере.

9. Сеанс задачи: внутри сессии раздельные job’ы, выполняющие одну и ту же задачу, могут видеть друг друга и взаимодействовать.

На рисунке 31 схематично представлены распределенные вычисления РЕД КВАНТ.

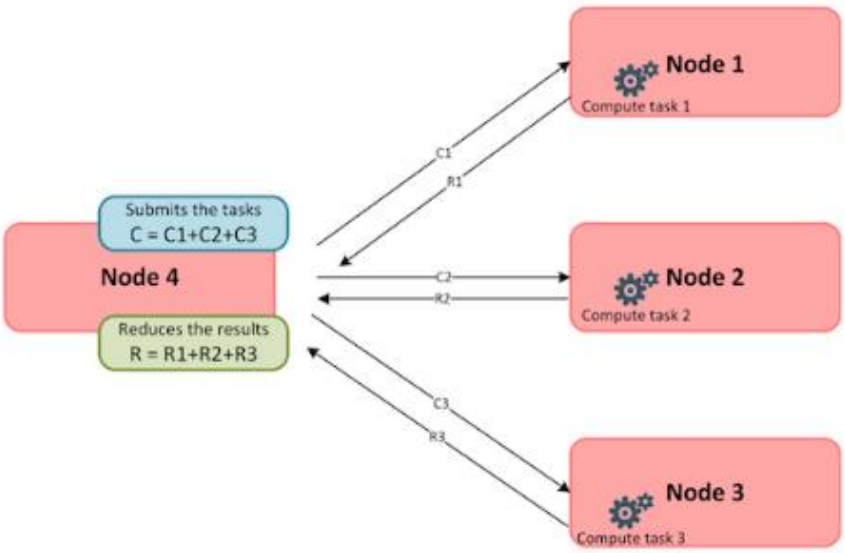

Рисунок 31 - Распредленные вычисления РЕД КВАНТ.

РЕД КВАНТ предлагает два различных подхода к распределенным вычислениям: вычислительная задача и сервисная задача.

Примечание: Основное различие между вычислительной задачей и сервисной задачей заключается в том, что сервисная задача — это продолжительный процесс, всегда доступный независимо от изменений топологии или сбоев. Распределенный сервис также может быть

доступен удаленно через прокси-сервер службы. Вычислительная задача может выполнять очень короткие оперативные задачи и возвращать немедленный результат выполнения задач.

Например, расчет кэшбэка для клиента банка — идеальный пример вычислительной задачи. Каждый месяц выполняются вычислительные задачи в кластере РЕД КВАНТ для расчета кэшбэка для клиента интернет-банка. Чтобы вернуть последние 10 транзакций для отображения в профиле клиента банка можно развернуть распределенный сервис в кластере РЕД КВАНТ. Распределенные сервисы РЕД КВАНТ очень полезны для разработки и реализации микросервисной архитектуры.

7.1 Грид для вычислений

РЕД КВАНТ предоставляет API для сбалансированного и отказоустойчивого распределения вычислений между узлами кластера. Можно отправлять отдельные задачи на выполнение, а также реализовать шаблон MapReduce с автоматическим разделением задач. API обеспечивает детальный контроль над стратегией распределения заданий.

Основной точкой входа для выполнения распределенных вычислений является вычислительный интерфейс, который можно получить из экземпляра РЕД КВАНТ.

```
Ignite ignite = Ignition.start();

IgniteCompute compute = ignite.compute();
```

Вычислительный интерфейс предоставляет методы для распределения различных типов задач по узлам кластера и выполнения сколоцированных вычислений.

Каждый экземпляр вычислительного интерфейса связан с набором узлов, на которых выполняются задачи. При вызове без аргументов функция ignite.compute() возвращает интерфейс вычислений, связанный со всеми узлами сервера. Чтобы получить экземпляр для определенного подмножества узлов, необходимо использовать Ignite.compute(ClusterGroup group). В следующем примере вычислительный интерфейс привязан только к удаленным узлам, то есть ко всем узлам, кроме того, на котором выполняется этот код.

```
Ignite ignite = Ignition.start();


IgniteCompute compute = ignite.compute(ignite.cluster().forRemotes());
```

7.1.1 Выполнение задач

РЕД КВАНТ предоставляет три интерфейса, которые могут быть реализованы для представления задачи и выполнения через интерфейс вычислений:

- IgniteRunnable — расширение java.lang.Runnable, которое можно использовать для реализации вычислений, не имеющих входных параметров и не возвращающих результата.
- IgniteCallable — расширение java.util.concurrent.Callable, которое возвращает определенное значение.
- IgniteClosure — функциональный интерфейс, который принимает параметр и возвращает значение.

Задачу можно выполнить один раз (на одном из узлов) или транслировать ее на все узлы.

Внимание! Чтобы запускать задачи на удаленных узлах, нужно убедиться, что определения классов задач доступны на узлах. Это можно сделать двумя способами:

- Добавить классы в classpath узлов;
- Включить загрузку однорангового класса.

7.1.1.1 Выполнение работающей задачи

Чтобы запустить выполняемую задачу, нужно использовать метод run(…) вычислительного интерфейса. Задача отправляется на один из узлов, связанных с вычислительным экземпляром.

```
IgniteCompute compute = ignite.compute();


// Iterate through all words and print
// each word on a different cluster node.
for (String word : "Print words on different cluster nodes".split(" ")) {
   compute.run(() -> System.out.println(word));
}
```

7.1.1.2 Выполнение вызываемой задачи

Чтобы выполнить вызываемую задачу, нужно использовать метод call(…) вычислительного интерфейса.

```
Collection<IgniteCallable<Integer>> calls = new ArrayList<>();


// Iterate through all words in the sentence and create callable jobs.
for (String word : "How many characters".split(" "))
   calls.add(word::length);

// Execute the collection of callables on the cluster.
Collection<Integer> res = ignite.compute().call(calls);

// Add all the word lengths received from cluster nodes.
int total = res.stream().mapToInt(Integer::intValue).sum();
```

7.1.1.3 Выполнение IgniteClosure

Чтобы выполнить IgniteClosure, нужно использовать метод apply(…) вычислительного интерфейса. Метод принимает задачу и входной параметр для задачи. Параметр передается заданному IgniteClosure во время выполнения.

```
IgniteCompute compute = ignite.compute();


// Execute closure on all cluster nodes.
Collection<Integer> res = compute.apply(String::length, Arrays.asList("How many characters".split(" ")));

// Add all the word lengths received from cluster nodes.
int total = res.stream().mapToInt(Integer::intValue).sum();
```

7.1.1.4 Трансляция задачи

Метод Broadcast() выполняет задачу на всех узлах, связанных с вычислительным экземпляром.

```
// Limit broadcast to remote nodes only.

IgniteCompute compute = ignite.compute(ignite.cluster().forRemotes());

// Print out hello message on remote nodes in the cluster group.
compute.broadcast(() -> System.out.println("Hello Node: " + ignite.cluster().localNode().id()));
```

7.1.1.5 Асинхронное выполнение

Все методы, описанные в предыдущих разделах, имеют асинхронные аналоги:

- callAsync(…);
- runAsync(…);
- applyAsync(…);
- broadcastAsync(…).

Асинхронные методы возвращают IgniteFuture, представляющий результат операции. В следующем примере набор вызываемых задач выполняется асинхронно.

```
IgniteCompute compute = ignite.compute();


Collection<IgniteCallable<Integer>> calls = new ArrayList<>();

// Iterate through all words in the sentence and create callable jobs.
for (String word : "Count characters using a callable".split(" "))
   calls.add(word::length);

IgniteFuture<Collection<Integer>> future = compute.callAsync(calls);

future.listen(fut -> {
   // Total number of characters.
   int total = fut.get().stream().mapToInt(Integer::intValue).sum();

   System.out.println("Total number of characters: " + total);
```

7.1.2 Время ожидания выполнения задачи

Можно установить тайм-аут для выполнения задачи. Если задача не завершается в течение заданного периода времени, она останавливается, и все созданные этой задачей jobs, отменяются.

Чтобы выполнить задачу с тайм-аутом, нужно использовать метод withTimeout(…) вычислительного интерфейса. Метод возвращает вычислительный интерфейс, который выполняет первую переданную ему задачу в течение ограниченного времени. У последовательных задач нет тайм-аута: нужно вызывать withTimeout(…) для каждой задачи, которая должна иметь тайм-аут.

```
IgniteCompute compute = ignite.compute();
```

```
compute.withTimeout(300_000).run(() -> {
   // your computation
   // ...
});
```

7.1.3 Совместное использование состояния между jobs на локальном узле

Полезно совместно использовать состояние между различными вычислительными jobs, выполняемыми на одном узле. Для этого на каждом узле доступен общий параллельный локальный словарь.

```
IgniteCluster cluster = ignite.cluster();

ConcurrentMap<String, Integer> nodeLocalMap = cluster.nodeLocalMap();
```

Локальные значения узла аналогичны локальным переменным потока в том смысле, что эти значения не распределяются и хранятся только на локальном узле. Локальные данные узла можно использовать для совместного использования состояния между вычислительными jobs. Они также могут использоваться развернутыми сервисами.

В следующем примере job увеличивает локальный счетчик узла каждый раз, когда он выполняется на каком-либо узле. В результате локальный счетчик на каждом узле сообщает, сколько раз job выполнялось на этом узле.

```
IgniteCallable<Long> job = new IgniteCallable<Long>() {

   @IgniteInstanceResource
   private Ignite ignite;

   @Override
   public Long call() {
      // Get a reference to node local.
      ConcurrentMap<String, AtomicLong> nodeLocalMap = ignite.cluster().nodeLocalMap();

      AtomicLong cntr = nodeLocalMap.get("counter");

      if (cntr == null) {
         AtomicLong old = nodeLocalMap.putIfAbsent("counter", cntr = new AtomicLong());

         if (old != null)
            cntr = old;
      }

      return cntr.incrementAndGet();
   }
};
```

7.1.4 Доступ к данным из вычислительных задач

Если вычислительной задаче требуется доступ к данным, хранящимся в кэшах, это можно сделать через экземпляр РЕД КВАНТ:

```
public class MyCallableTask implements IgniteCallable<Integer> {
```

```
    @IgniteInstanceResource
    private Ignite ignite;

    @Override
    public Integer call() throws Exception {

        IgniteCache<Long, Person> cache = ignite.cache("person");

        // Get the data you need
        Person person = cache.get(1L);

        // do with the data what you need to do

        return 1;
    }
}
```

Показанный выше пример может быть не самым эффективным способом. Причина в том, что объект person, соответствующий ключу 1, может находиться не на узле, где выполняется задача. В этом случае объект извлекается через сеть. Этого можно избежать, сколоцировав задачу с данными.

Внимание: Если необходимо использовать объекты ключей и значений внутри задач IgniteCallable и IgniteRunnable, нужно убедиться, что классы ключей и значений развернуты на всех узлах кластера.

7.2 Грид для сервисов

Сервис — это часть функциональности, которую можно развернуть в кластере РЕД КВАНТ и выполнять определенные операции. Может быть несколько экземпляров службы на одном или нескольких узлах.

Сервисы РЕД КВАНТ имеют следующие особенности:

• Балансировка нагрузки - Во всех случаях, кроме развертывания синглтонов сервисов, РЕД КВАНТ автоматически обеспечивает развертывание примерно одинакового количества сервисов на каждом узле в кластере. Всякий раз, когда топология кластера изменяется, РЕД КВАНТ повторно оценивает необходимость развертывания службы и может повторно развернуть уже развернутую службу на другом узле для лучшей балансировки нагрузки.

• Отказоустойчивость — РЕД КВАНТ всегда гарантирует постоянную доступность сервисов и их развертывание в соответствии с заданной конфигурацией, независимо от любых изменений топологии или сбоев узла.

• Горячее развертывание - Можно использовать конфигурацию РЕД КВАНТ DeploymentSpi для повторного развертывания сервисов без перезапуска кластера.

Сервисы РЕД КВАНТ можно использовать для приложений или решений на основе микросервисов.

7.2.1 Реализация сервиса

Сервис реализует интерфейс Service. Интерфейс Service имеет три метода:

• init(): этот метод вызывается РЕД КВАНТ перед развертыванием сервиса (и перед вызовом метода execute());

• execute(): запускает выполнение сервиса;

• cancel(): отменяет выполнение сервиса.

7.2.2 РАЗВЕРТЫВАНИЕ СЕРВИСОВ

Сервис можно развернуть либо программно во время выполнения, либо путем предоставления конфигурации сервиса как части конфигурации узла. В последнем случае сервис развертывается при запуске кластера.

7.2.2.1 Развертывание сервисов во время выполнения

Можно развертывать сервисы во время выполнения через экземпляр IgniteServices, который можно получить из экземпляра РЕД КВАНТ, вызвав метод Ignite.services().

Интерфейс IgniteServices имеет ряд методов для развертывания сервисов:

• deploy(ServiceConfiguration) развертывает сервис, определенный заданной конфигурацией.

• deployNodeSingleton(…) гарантирует, что экземпляр сервиса работает на каждом серверном узле.

• deployClusterSingleton(…) развертывает один экземпляр сервиса для каждого кластера. Если узел кластера, на котором развернут сервис, останавливается, РЕД КВАНТ автоматически повторно развертывает сервис на другом узле.

• deployKeyAffinitySingleton(…) развертывает один экземпляр сервиса на основном узле для заданного ключа кэша.

• deployMultiple(…) развертывает заданное количество экземпляров сервиса.

Это пример развертывания синглтона кластера:

```
Ignite ignite = Ignition.start();

//get the services interface associated with all server nodes
IgniteServices services = ignite.services();

//start a node singleton
services.deployClusterSingleton("myCounterService", new MyCounterServiceImpl());
```

Развертывание синглтона кластера с помощью ServiceConfiguration:

```
Ignite ignite = Ignition.start();

ServiceConfiguration serviceCfg = new ServiceConfiguration();

serviceCfg.setName("myCounterService");
serviceCfg.setMaxPerNodeCount(1);
serviceCfg.setTotalCount(1);
serviceCfg.setService(new MyCounterServiceImpl());
```

```
ignite.services().deploy(serviceCfg);
```

7.2.2.2 Развертывание сервисов при запуске узла

Можно указать свой сервис как часть конфигурации узла и запустить службу вместе с узлом. Если сервис является синглтоном узла, он запускается на каждом узле кластера. Если сервис является синглтоном в кластере, он запускается на первом узле кластера и повторно развертывается на одном из других узлов, если первый узел завершает работу. Сервис должен быть доступен в пути к классам каждого узла.

Ниже приведен пример настройки синглтона сервиса кластера:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">


  <property name="serviceConfiguration">
    <list>
      <bean class="org.apache.ignite.services.ServiceConfiguration">
        <property name="name" value="myCounterService"/>
        <property name="maxPerNodeCount" value="1"/>
        <property name="totalCount" value="1"/>
        <property name="service">
          <bean class="org.apache.ignite.snippets.services.MyCounterServiceImpl"/>
        </property>
      </bean>
    </list>
  </property>

</bean>
```

7.2.3 РАЗВЕРТЫВАНИЕ НА ПОДМНОЖЕСТВЕ УЗЛОВ

При получении интерфейса IgniteServices через вызов ignite.services(), экземпляр IgniteServices связывается со всеми серверными узлами. Это означает, что из множества серверных узлов РЕД КВАНТ выбирает, где развернуть сервис. Можно изменить набор узлов, рассматриваемых для развертывания сервиса, используя различные подходы, описанные ниже.

7.2.3.1 Кластерный синглтон

Кластерный синглтон — это стратегия развертывания, при которой в кластере есть только один экземпляр сервиса, и РЕД КВАНТ гарантирует, что этот экземпляр всегда доступен. В случае сбоя или остановки узла кластера, на котором развернут сервис, РЕД КВАНТ автоматически повторно развертывает экземпляр на другом узле.

7.2.3.2 Кластерная группа

Интерфейс ClusterGroup можно использовать для развертывания сервисов на подмножестве узлов. Если сервис является синглтоном узла, он развертывается на всех узлах из подмножества. Если сервис является синглтоном кластера, он развертывается на одном из узлов подмножества.

```
Ignite ignite = Ignition.start();
```

```
//deploy the service to the nodes that host the cache named "myCache"
ignite.services(ignite.cluster().forCacheNodes("myCache"));
```

7.2.3.3 Фильтр узлов

Можно использовать атрибуты узлов для определения подмножества узлов, предназначенных для развертывания сервиса. Для этого используется фильтр узла. Фильтр узлов — это IgnitePredicate<ClusterNode>, который РЕД КВАНТ вызывает для каждого узла, связанного с интерфейсом IgniteService. Если предикат возвращает true для данного узла, этот узел включается.

Внимание: Класс фильтра узлов должен присутствовать в пути к классам всех узлов.

Пример фильтра узла. Фильтр включает серверные узлы с атрибутом «west.coast.node».

```
public static class ServiceFilter implements IgnitePredicate<ClusterNode> {

  @Override
  public boolean apply(ClusterNode node) {
    // The service will be deployed on the server nodes
    // that have the 'west.coast.node' attribute.
    return !node.isClient() && node.attributes().containsKey("west.coast.node");
  }
}
```

Развертывание сервиса с помощью фильтра узлов:

```
Ignite ignite = Ignition.start();

ServiceConfiguration serviceCfg = new ServiceConfiguration();

// Setting service instance to deploy.
serviceCfg.setService(new MyCounterServiceImpl());
serviceCfg.setName("serviceName");
serviceCfg.setMaxPerNodeCount(1);

// Setting the nodes filter.
serviceCfg.setNodeFilter(new ServiceFilter());

// Getting an instance of IgniteService.
IgniteServices services = ignite.services();

// Deploying the service.
services.deploy(serviceCfg);
```

7.2.3.4 Ключ кэша

Развертывание на основе affinity позволяет развертывать сервис на главном узле для определенного ключа в определенном кэше. Для развертывания на базе affinity необходимо указать нужный кэш и ключ в конфигурации сервиса. Кэш не обязательно должен содержать ключ. Узел определяется affinity-функцией. Если топология кластера изменяется таким образом, что ключ переназначается другому узлу, сервис также повторно развертывается на этом узле.

```
Ignite ignite = Ignition.start();
```

```
// Making sure the cache exists.
ignite.getOrCreateCache("orgCache");

ServiceConfiguration serviceCfg = new ServiceConfiguration();

// Setting service instance to deploy.
serviceCfg.setService(new MyCounterServiceImpl());

// Setting service name.
serviceCfg.setName("serviceName");
serviceCfg.setTotalCount(1);

// Specifying the cache name and key for the affinity based deployment.
serviceCfg.setCacheName("orgCache");
serviceCfg.setAffinityKey(123);

IgniteServices services = ignite.services();

// Deploying the service.
services.deploy(serviceCfg);
```

7.2.4 Доступ к сервисам

Доступ к сервису можно получить во время выполнения через прокси-сервер сервиса. Прокси могут быть sticky или non-sticky. Sticky прокси всегда подключается к одному и тому же узлу кластера для доступа к удаленно развернутому сервису. Non-sticky прокси-сервер распределяет нагрузку вызовов удаленных сервисов между всеми узлами кластера, на которых развернут сервис.

Следующий фрагмент кода получает non-sticky прокси-сервер для сервиса и вызывает его метод:

```
//access the service by name

MyCounterService counterService = ignite.services().serviceProxy("myCounterService",
    MyCounterService.class, false); //non-sticky proxy

//call a service method
counterService.increment();
```

7.2.5 Отмена развертывания сервисов

Чтобы отменить развертывание сервиса, необходимо использовать метод IgniteServices.cancel(serviceName) или IgniteServices.cancelAll().

```
services.cancel("myCounterService");
```

7.2.6 Повторное развертывание сервисов

Если необходимо обновить реализацию сервиса без остановки кластера, это можно сделать через конфигурацию РЕД КВАНТ DeploymentSPI.

Для повторного развертывания сервиса нужно использовать следующую процедуру:

1. Обновить файлы JAR в месте, где хранится сервис (на которое указывает свойство UriDeploymentSpi.uriList). РЕД КВАНТ перезагрузит новые классы после периода обновления, указанного в настройках.
2. Добавить реализацию сервиса в путь класса клиентского узла и запустить клиент.
3. Вызвать метод Ignite.services().cancel() на клиентском узле, чтобы остановить сервис.
4. Развернуть сервис с клиентского узла.
5. Остановить клиентский узел.

Таким образом, не нужно останавливать серверные узлы, поэтому работа кластера не прерывается.

7.2.7 Статистика сервиса

Можно измерить продолжительность методов сервиса. Если необходима эта аналитика, нужно включить статистику сервиса в конфигурации сервиса. Статистика сервиса собирается под названием «Services» в метриках, в системных представлениях и в JMX.

```
Ignite ignite = Ignition.start();

ServiceConfiguration serviceCfg = new ServiceConfiguration();

serviceCfg.setName("myService");
serviceCfg.setMaxPerNodeCount(1);
serviceCfg.setService(new MyCounterServiceImpl());

// Enable service statistics.
serviceCfg.setStatisticsEnabled(true);

ignite.services().deploy(serviceCfg);

// NOTE: work via proxy. Direct references like 'IgniteServices#service()' corrupt the statistics.
MyCounterService svc = ignite.services().serviceProxy("myService", MyCounterService.class, true)
```

Примечание: Следует отметить, что:

1. Прямые ссылки на такие сервисы, как IgniteServices.service(name), искажают статистику. Вместо этого нужно использовать прокси (IgniteServices.serviceProxy(…)).
2. Перегруженные сервисные методы имеют одинаковую метрику по имени метода.
3. Статистика сервиса не учитывает сериализацию и проблемы с сетью.
4. Статистика сервиса замедляет вызовы методов сервиса. Это не проблема для таких реальных задач, как работа с БД или кэшем.

7.3 Пользовательский код

Помимо загрузки одноранговых классов, можно развернуть пользовательский код, настроив UriDeploymentSpi. При таком подходе указывается расположение библиотек в конфигурации узла. РЕД КВАНТ периодически сканирует местоположение и в случае изменения повторно развертывает классы. Местоположение библиотек может быть каталогом

файловой системы или расположением HTTP(S). Когда РЕД КВАНТ обнаруживает, что библиотеки удалены из расположения, классы не развертываются в кластере.

Можно указать несколько местоположений (разных типов), указав как пути к каталогам, так и URL-адреса http(s).

7.3.1 РАЗВЕРТЫВАНИЕ ИЗ ЛОКАЛЬНОГО КАТАЛОГА

Чтобы развернуть библиотеки из каталога файловой системы, нужно добавить путь к каталогу в список URI в конфигурации UriDeploymentSpi. Каталог должен существовать на тех узлах, где он указан, и содержать jar-файлы с классами, которые необходимо развернуть. Следует обратить внимание, что путь должен быть указан с использованием схемы «file://». Можно указать разные каталоги на разных узлах.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  Licensed to the Apache Software Foundation (ASF) under one or more
  contributor license agreements.  See the NOTICE file distributed with
  this work for additional information regarding copyright ownership.
  The ASF licenses this file to You under the Apache License, Version 2.0
  (the "License"); you may not use this file except in compliance with
  the License.  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd       http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util.xsd">
    <bean class="org.apache.ignite.configuration.IgniteConfiguration">
      <property name="deploymentSpi">
        <bean class="org.apache.ignite.spi.deployment.uri.UriDeploymentSpi">
          <property name="temporaryDirectoryPath" value="/tmp/temp_ignite_libs"/>
          <property name="uriList">
            <list>
              <value>file://freq=2000@localhost/home/username/user_libs</value>
            </list>
          </property>
        </bean>
      </property>
    </bean>
</beans>
```

В URL можно передать параметр freq — частота сканирования в миллисекундах (значение по умолчанию = 5000).

7.3.2 Развертывание с URL-адреса

Чтобы развернуть библиотеки из расположения http(s), нужно добавить URL-адрес в список URI в конфигурации UriDeploymentSpi.

РЕД КВАНТ анализирует файл HTML, чтобы найти атрибуты HREF всех тегов <a> на странице. Ссылки должны указывать на файлы JAR, которые необходимо развернуть.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  Licensed to the Apache Software Foundation (ASF) under one or more
  contributor license agreements.  See the NOTICE file distributed with
  this work for additional information regarding copyright ownership.
  The ASF licenses this file to You under the Apache License, Version 2.0
  (the "License"); you may not use this file except in compliance with
  the License.  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd       http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util.xsd">
    <bean class="org.apache.ignite.configuration.IgniteConfiguration">
      <property name="deploymentSpi">
        <bean class="org.apache.ignite.spi.deployment.uri.UriDeploymentSpi">
          <property name="temporaryDirectoryPath" value="/tmp/temp_ignite_libs"/>
          <property name="uriList">
            <list>
              <value>http://username:password;freq=10000@www.mysite.com:110/ignite/user_libs</value>
            </list>
          </property>
        </bean>
      </property>
    </bean>
</beans>
```

В URL можно передать параметр freq — частота сканирования в миллисекундах (значение по умолчанию 300000).

7.4 Обмен сообщениями

Распределенный обмен сообщениями РЕД КВАНТ обеспечивает основанное на темах взаимодействие между всеми узлами в рамках всего кластера. Через указанную тему сообщения могут быть распределены на все узлы или подгруппу узлов, подписавшихся на эту тему.

Обмен сообщениями РЕД КВАНТ основан на парадигме «издатель-подписчик», в которой издатели и подписчики связаны общей темой. Когда один из узлов отправляет сообщение A для темы T, оно публикуется на всех узлах, подписавшихся на T.

Примечание: Любой новый узел, присоединяющийся к кластеру, автоматически подписывается на все темы, на которые подписаны другие узлы в кластере (или группе кластера).

Функциональность распределенного обмена сообщениями в РЕД КВАНТ доступна через интерфейс IgniteMessaging. Экземпляр IgniteMessaging можно получить следующим образом:

```
Ignite ignite = Ignition.ignite();

// Messaging instance over this cluster.
IgniteMessaging msg = ignite.message();

// Messaging instance over given cluster group (in this case, remote nodes).
IgniteMessaging rmtMsg = ignite.message(ignite.cluster().forRemotes());
```

7.4.1 Публикация сообщений

Методы отправки помогают отправлять/публиковать сообщения с указанной темой всем узлам. Сообщения могут быть отправлены в упорядоченном или неупорядоченном виде.

Метод sendOrdered(…) можно использовать, если необходимо получать сообщения в том порядке, в котором они были отправлены. Параметр timeout передается для указания, как долго сообщение будет оставаться в очереди для ожидания сообщений, которые должны быть отправлены до этого сообщения. При истечении тайм-аута будут проигнорированы все сообщения, которые еще не поступили по данной теме на этот узел.

Методы send(…) не гарантируют порядок сообщений. Это означает, что когда последовательно отправляются сообщение A и сообщение B, не гарантируется, что целевой узел сначала получит A, а затем B.

7.4.2 Подписка на сообщения

Методы прослушивания помогают прослушивать сообщения и подписываться на них. При вызове этих методов слушатель с указанной темой сообщения регистрируется на всех (или подгруппе) узлах для прослушивания новых сообщений. В методах прослушивания передается предикат, который возвращает логическое значение, указывающее слушателю продолжать или прекратить прослушивание новых сообщений.

Метод localListen(…) регистрирует слушателя сообщений с указанной темой только на локальном узле и прослушивает сообщения с любого узла в данной кластерной группе.

Метод remoteListen(…) регистрирует слушателей сообщений с указанной темой на всех узлах в данной кластерной группе и прослушивает сообщения с любого узла в этой кластерной группе.

8 РАСПРЕДЕЛЕННЫЕ СТРУКТУРЫ ДАННЫХ

8.1 Очереди и наборы

В дополнение к предоставлению стандартного хранилища, похожего на словарь «ключ-значение», РЕД КВАНТ также предоставляет реализацию быстрой распределенной блокирующей очереди и распределенного набора.

IgniteQueue и IgniteSet, реализация интерфейса java.util.concurrent.BlockingQueue и java.util.Set соответственно, также поддерживают все операции из интерфейса java.util.Collection. Оба типа могут быть созданы как в режиме колокации, так и в режиме без колокации.

Ниже приведен пример создания:

• распределенной очереди:

```
Ignite ignite = Ignition.start();


IgniteQueue<String> queue = ignite.queue("queueName", // Queue name.
    0, // Queue capacity. 0 for an unbounded queue.
    new CollectionConfiguration() // Collection configuration.
);
```

• набора:

```
Ignite ignite = Ignition.start();


IgniteSet<String> set = ignite.set("setName", // Set name.
    new CollectionConfiguration() // Collection configuration.
);
```

8.1.1 Режимы с колокацией и без

Если планируется создать всего несколько очередей или наборов, содержащих большое количество данных, то необходимо создать их в режиме без колокации. Это гарантирует, что примерно равная часть каждой очереди или набора будет храниться на каждом узле кластера. С другой стороны, если планируется иметь много относительно небольших по размеру (по сравнению со всем кэшем) очередей или наборов, то, скорее всего, они будут созданы в режиме колокации. В этом режиме все элементы очереди или набора будут храниться на одном узле кластера, но каждому узлу будет назначено примерно одинаковое количество очередей/наборов.

Режим без колокации имеет смысл только для партиционированных кэшей и поддерживается только для них.

Сколоцированную очередь и набор можно создать, установив свойство колокации CollectionConfiguration, например:

Очередь:

```
Ignite ignite = Ignition.start();
```

```
CollectionConfiguration colCfg = new CollectionConfiguration();

colCfg.setCollocated(true);

// Create a colocated queue.
IgniteQueue<String> queue = ignite.queue("queueName", 0, colCfg)
```

Набор:

```
Ignite ignite = Ignition.start();


CollectionConfiguration colCfg = new CollectionConfiguration();

colCfg.setCollocated(true);

// Create a colocated set.
IgniteSet<String> set = ignite.set("setName", colCfg);
```

8.1.2 Очереди кэша и балансировка нагрузки

Учитывая, что элементы будут оставаться в очереди до тех пор, пока кто-то их не заберет, и что никакие два узла никогда не должны получать один и тот же элемент из очереди, очереди кэша могут использоваться в качестве альтернативного подхода к распределению работы и балансировке нагрузки в РЕД КВАНТ.

Например, можно добавить вычисления в очередь, такие как экземпляры IgniteRunnable, и заставить потоки на удаленных узлах вызывать метод IgniteQueue.take(), который будет блокироваться, если очередь пуста. Как только метод take() вернет job, поток обработает его и снова вызовет take(), чтобы получить следующее job. При таком подходе потоки на удаленных узлах начнут работать над следующим job только после завершения предыдущего, создавая, таким образом, идеально сбалансированную систему, в которой каждый узел выполняет только то количество job, которое он может обработать, и не больше.

8.1.3 Конфигурация коллекции

Коллекции РЕД КВАНТ можно настроить в API через CollectionConfiguration (см. примеры выше). Можно использовать параметры конфигурации, указанные в таблице 10.

Таблица 10 - Поддерживаемые параметры конфигурации

Параметр конфигурации	Описание	Значение по умолчанию
setCollocated(boolean)	Устанавливает режим колокации.	false
setCacheMode(CacheMode)	Задает базовый режим кэша (СЕКЦИОНИРОВАННЫЙ, РЕПЛИЦИРОВАННЫЙ или ЛОКАЛЬНЫЙ).	PARTITIONED
setAtomicityMode(CacheAtomicityMode)	Устанавливает базовый режим атомарности кэша (ATOMIC или TRANSACTIONAL).	ATOMIC

Параметр конфигурации	Описание	Значение по умолчанию
setOffHeapMaxMemory(long)	Устанавливает максимальный размер памяти off-heap.	0 (неограниченный)
setBackups(int)	Устанавливает количество резервных копий.	0
setNodeFilter(IgnitePredicate< ClusterNode>)	Устанавливает опциональный предикат, указывающий, на каких узлах должны храниться записи.	

8.2 АТОМАРНЫЕ ТИПЫ

РЕД КВАНТ поддерживает распределенный атомарный long и атомарные ссылки, аналогичные java.util.concurrent.atomic.AtomicLong и java.util.concurrent.atomic.AtomicReference соответственно.

Атомарные значения в РЕД КВАНТ распределены по кластеру, что позволяет выполнять атомарные операции (такие как увеличение и получение или сравнение и установка) с одним и тем же глобально видимым значением. Например, можно обновить значение atomic long на одном узле и прочитать его на другом узле.

Функции:

• Получить текущее значение.

• Атомарно изменить текущее значение.

• Атомарно увеличить или уменьшить текущее значение.

• Атомарно сравнить и установить текущее значение с новым значением.

Распределенный atomic long и atomic reference можно получить через интерфейсы IgniteAtomicLong и IgniteAtomicReference соответственно, как показано ниже:

AtomicLong:

```
Ignite ignite = Ignition.start();


IgniteAtomicLong atomicLong = ignite.atomicLong("atomicName", // Atomic long name.
    0, // Initial value.
    true // Create if it does not exist.
);

// Increment atomic long on local node
System.out.println("Incremented value: " + atomicLong.incrementAndGet());
```

AtomicReference:

```
Ignite ignite = Ignition.start();


// Create an AtomicReference
```

```
IgniteAtomicReference<String> ref = ignite.atomicReference("refName", // Reference name.
    "someVal", // Initial value for atomic reference.
    true // Create if it does not exist.
);

// Compare and update the value
ref.compareAndSet("WRONG EXPECTED VALUE", "someNewVal"); // Won't change.
```

Все атомарные операции, предоставляемые IgniteAtomicLong и IgniteAtomicReference, являются синхронными. Время атомарной операции зависит от количества узлов, выполняющих параллельные операции с одним и тем же экземпляром atomic long, интенсивности этих операций и сетевой задержки.

Атомарные значения в РЕД КВАНТ могут быть сконфигурированы с помощью свойства atomicConfiguration в Ignite Configuration.

В таблице 11 перечислены доступные параметры конфигурации.

Таблица 11 - Параметры конфигурации

Параметр конфигурации	**Описание**	**Значение по умолчанию**
setBackups(int)	Количество резервных копий.	0
setCacheMode(CacheMode)	Режим кэширования для всех атомарных типов.	PARTITIONED
setAtomicSequenceReserveSize (int)	Задает количество значений последовательности, зарезервированных для экземпляров IgniteAtomicSequence.	1000

8.3 Счетчики

IgniteCountDownLatch предоставляет функциональные возможности, аналогичные java.util.concurrent.CountDownLatch, и позволяет синхронизировать операции между узлами кластера.

Распределенный CountDownLatch можно создать следующим образом:

```
Ignite ignite = Ignition.start();


IgniteCountDownLatch latch = ignite.countDownLatch("latchName", // Latch name.
    10, // Initial count.
    false, // Auto remove, when counter has reached zero.
    true // Create if it does not exist.
);
```

После выполнения приведенного выше кода все узлы в указанном кэше смогут синхронизироваться с блокировкой с именем latchName. Ниже приведен пример кода такой синхронизации:

```
Ignite ignite = Ignition.start();
```

```
final IgniteCountDownLatch latch = ignite.countDownLatch("latchName", 10, false, true);

// Execute jobs.
for (int i = 0; i < 10; i++)
  // Execute a job on some remote cluster node.
  ignite.compute().run(() -> {
    int newCnt = latch.countDown();

    System.out.println("Counted down: newCnt=" + newCnt);
  });

// Wait for all jobs to complete.
latch.await();
```

8.4 Последовательности

Распределенная атомарная последовательность, предоставляемая интерфейсом IgniteCacheAtomicSequence, похожа на распределенный атомарный long, но ее значение может только увеличиваться. Она также поддерживает резервирование диапазона значений, чтобы избежать дорогостоящих сетевых подключений или обновлений кэша каждый раз, когда последовательность должна предоставить следующее значение. То есть, когда выполняется incrementAndGet() (или любая другая атомарная операция) над атомарной последовательностью, структура данных заранее резервирует диапазон значений, которые гарантированно будут уникальными в кластере для этого экземпляра последовательности.

Пример создания атомарной последовательности:

```
Ignite ignite = Ignition.start();


//create an atomic sequence
IgniteAtomicSequence seq = ignite.atomicSequence("seqName", // Sequence name.
    0, // Initial value for sequence.
    true // Create if it does not exist.
);

// Increment the atomic sequence.
for (int i = 0; i < 20; i++) {
  long currentValue = seq.get();
  long newValue = seq.incrementAndGet();
}
```

Ключевым параметром IgniteAtomicSequence является atomicSequenceReserveSize, который представляет собой количество зарезервированных значений последовательности для каждого узла. Когда узел пытается получить экземпляр IgniteAtomicSequence, ряд значений последовательности будет зарезервирован для этого узла, и последующие увеличения последовательности будут происходить локально без связи с другими узлами, пока не потребуется сделать следующее резервирование.

Значение по умолчанию для atomicSequenceReserveSize — 1000. Этот параметр можно изменить через свойство atomicSequenceReserveSize в AtomicConfiguration.

8.5 Семафоры

Реализация и поведение распределенного семафора в РЕД КВАНТ схожи с концепцией хорошо известного java.util.concurrent.Semaphore. Как и любой другой семафор, он поддерживает набор разрешений, которые берутся с помощью метода acquire() и освобождаются с помощью аналога release(). Семафоры позволяют ограничить доступ к какому-либо логическому или физическому ресурсу или синхронизировать поток выполнения. Единственное отличие состоит в том, что семафор РЕД КВАНТ позволяет выполнять такие действия не только в рамках одной JVM, но и в масштабе всего кластера на многих удаленных узлах.

Распределенный семафор можно создать следующим образом:

```
Ignite ignite = Ignition.start();


IgniteSemaphore semaphore = ignite.semaphore("semName", // Distributed semaphore name.
      20, // Number of permits.
      true, // Release acquired permits if node, that owned them, left topology.
      true // Create if it doesn't exist.
);
```

Как только семафор создан, он может использоваться одновременно несколькими узлами кластера для реализации некоторой распределенной логики или ограничения доступа к распределенному ресурсу, как в следующем примере:

```
Ignite ignite = Ignition.start();


IgniteSemaphore semaphore = ignite.semaphore("semName", // Distributed semaphore name.
      20, // Number of permits.
      true, // Release acquired permits if node, that owned them, left topology.
      true // Create if it doesn't exist.
);

// Acquires a permit, blocking until it's available.
semaphore.acquire();

try {
   // Semaphore permit is acquired. Execute a distributed task.
   ignite.compute().run(() -> {
      System.out.println("Executed on:" + ignite.cluster().localNode().id());

      // Additional logic.
   });
} finally {
   // Releases a permit, returning it to the semaphore.
   semaphore.release();
}
```

8.6 Генераторы

Атомарная последовательность является подходящей и эффективной структурой данных для реализации распределенного генератора идентификаторов. Например, такой генератор можно использовать для создания уникальных первичных ключей для всего кластера.

Пример того, как можно создать атомарную последовательность:

```
Ignite ignite = Ignition.ignite();

IgniteAtomicSequence seq = ignite.atomicSequence(
   "seqName", // Sequence name.
   0,      // Initial value for sequence.
   true    // Create if it does not exist.
);
```

Ниже приведен простой пример использования:

```
Ignite ignite = Ignition.ignite();

// Initialize atomic sequence.
final IgniteAtomicSequence seq = ignite.atomicSequence("seqName", 0, true);

// Increment atomic sequence.
for (int i = 0; i < 20; i++) {
  long currentValue = seq.get();
  long newValue = seq.incrementAndGet();

  ...
}
```

Ключевым параметром IgniteAtomicSequence является atomicSequenceReserveSize, который представляет собой количество значений последовательности, зарезервированных для каждого узла. Когда узел пытается получить экземпляр IgniteAtomicSequence, ряд значений последовательности резервируется для этого узла, и последующие увеличения последовательности будут происходить локально без связи с другими узлами, пока не потребуется сделать следующее резервирование.

Значение по умолчанию для atomicSequenceReserveSize — 1000. Этот параметр можно изменить через свойство atomicSequenceReserveSize в AtomicConfiguration.

8.7 РАСПРЕДЕЛЕННЫЕ БЛОКИРОВКИ

Транзакции РЕД КВАНТ получают распределенные блокировки неявным образом. Однако существуют определенные случаи, когда может потребоваться получить блокировки явно. Метод lock() API IgniteCache возвращает экземпляр java.util.concurrent.locks.Lock, который позволяет определить явные распределенные блокировки для любого заданного ключа. Блокировки также можно получить для коллекции объектов с помощью метода IgniteCache.lockAll().

```
IgniteCache<String, Integer> cache = ignite.cache("myCache");


// Create a lock for the given key
Lock lock = cache.lock("keyLock");
try {
   // Acquire the lock
   lock.lock();

   cache.put("Hello", 11);
```

```
    cache.put("World", 22);
}
finally {
    // Release the lock
    lock.unlock();
}
```

Примечание: В РЕД КВАНТ блокировки поддерживаются только для режима атомарности TRANSACTIONAL, который можно установить с помощью параметра CacheConfiguration.atomicityMode.

Явные блокировки не являются транзакционными и не могут использоваться внутри транзакций (будет выдано исключение). Если нужна явная блокировка внутри транзакций, следует использовать элемент управления параллелизмом TransactionConcurrency.PESSIMISTIC для получения транзакциями явных блокировок соответствующих запросов данных кластера.

9 УПРАВЛЕНИЕ КЛАСТЕРОМ

РЕД КВАНТ предоставляет несколько инструментов для управления кластером. Эти инструменты включают в себя веб-интерфейс или интерфейс командной строки, которые позволяют выполнять различные задачи, такие как запуск/остановка/перезапуск удаленных узлов или управление состояниями кластера (базовая топология). В таблице 12 показаны все встроенные инструменты РЕД КВАНТ для настройки и управления кластером РЕД КВАНТ.

Таблица 12 - Встроенные инструменты РЕД КВАНТ

Наименование	Описание
Ignite Web console	Позволяет настраивать все свойства кластера и управлять Ignitecluster через веб-интерфейс.
Control script	Скрипт командной строки, который позволяет отслеживать и контролировать состояния кластера, включая базовую топологию РЕД КВАНТ.

Примечание: Веб-консоль РЕД КВАНТ также используется для мониторинга функциональных возможностей кластера, таких как показатели кэша, а также использования ЦП и heap памяти.

9.1 УТИЛИТА УПРАВЛЕНИЯ CONTROL.SH

РЕД КВАНТ предоставляет скрипт командной строки — control.sh|bat — который можно использовать для мониторинга и управления кластерами. Скрипт находится в папке /bin/ каталога установки.

Синтаксис управляющего скрипта для Unix и Windows соответственно:

```
control.sh <connection parameters> <command> <arguments>

control.bat <connection parameters> <command> <arguments>
```

9.1.1 ПОДКЛЮЧЕНИЕ К КЛАСТЕРУ

При выполнении без параметров подключения управляющий скрипт пытается подключиться к узлу, работающему на локальном хосте (localhost:11211). Если необходимо подключиться к узлу, работающему на удаленной машине, нужно указать параметры подключения, представленные в таблице 13.

Таблица 13 - Параметры подключения

Параметр	Описание	Значение по умолчанию
--host HOST_OR_IP	Имя хоста или IP-адрес узла.	localhost
--port PORT	Порт для подключения.	11211
--user USER	Имя пользователя.	
--password PASSWORD	Пароль пользователя.	

Параметр	Описание	Значение по умолчанию
--ping-interval PING_INTERVAL	Интервал пинга.	5000
--ping-timeout PING_TIMEOUT	Тайм-аут ответа на запрос.	30000
--ssl-protocol PROTOCOL1, PROTOCOL2…	Список протоколов SSL, которые следует использовать при подключении к кластеру.	TLS
--ssl-cipher-suites CIPHER1,CIPHER2…	Список SSL-шифров.	
--ssl-key-algorithm ALG	Алгоритм SSL-ключа.	SunX509
--keystore-type KEYSTORE_TYPE	Тип хранилища ключей.	JKS
--keystore KEYSTORE_PATH	Путь к хранилищу ключей. Чтобы включить SSL для управляющего скрипта, необходимо указать хранилище ключей.	
--keystore-password KEYSTORE_PWD	Пароль к хранилищу ключей.	
--truststore-type TRUSTSTORE_TYPE	Тип доверенного хранилища.	JKS
--truststore TRUSTSTORE_PATH	Путь к хранилищу доверия.	
--truststore-password TRUSTSTORE_PWD	Пароль к хранилищу доверия.	

9.1.2 Активация, деактивация и управление топологией

Управляющий скрипт можно использовать для активации или деактивации кластера и управления базовой топологией.

Кластер может находиться в одном из трех состояний: активен, только для чтения или неактивен.

Чтобы получить состояние кластера, нужно выполнить следующую команду:

```
control.sh --state
```

Активация устанавливает базовую топологию кластера на набор узлов, доступных на момент активации. Активация требуется только при использовании встроенной персистентности.

Чтобы активировать кластер, нужно выполнить следующую команду:

```
control.sh --set-state ACTIVE
```

Внимание! Деактивация освобождает все ресурсы памяти, включая данные приложения, на всех узлах кластера и отключает общедоступный API кластера. Если есть in-memory кэши, резервные копии которых не поддерживаются постоянным хранилищем (ни собственным постоянным хранилищем, ни внешним хранилищем), данные будут утеряны и нужно будет повторно заполнить эти кэши. Также очищаются не персистентные системные кэши.

Чтобы деактивировать кластер, нужно выполнить следующую команду:

```
control.sh --set-state INACTIVE [--yes]
```

Чтобы получить список узлов, зарегистрированных в базовой топологии, нужно выполнить следующую команду:

```
control.sh --baseline
```

Выходные данные содержат текущую версию топологии, список согласованных идентификаторов узлов, включенных в базовую топологию, и список узлов, присоединившихся к кластеру, но не добавленных в базовую топологию.

Чтобы добавить узел в базовую топологию, нужно выполнить приведенную ниже команду. После добавления узла начинается процесс ребалансировки.

control.sh --baseline add *consistentId1,consistentId2,...* [--yes]

Чтобы удалить узел из базовой топологии, необходимо использовать команду remove. Из базовой топологии можно удалить только автономные узлы: сначала необходимо выключить узел, а затем использовать команду remove. Эта операция запускает процесс ребалансировки, который перераспределяет данные между узлами, которые остаются в базовой топологии.

control.sh --baseline remove *consistentId1,consistentId2,...* [--yes]

Установить базовую топологию можно, либо предоставив список узлов (согласованных идентификаторов), либо указав желаемую версию базовой топологии.

Чтобы установить список узлов в качестве базовой топологии, нужно использовать следующую команду:

control.sh --baseline set *consistentId1,consistentId2,...* [--yes]

Чтобы восстановить определенную версию базовой топологии, нужно использовать следующую команду:

control.sh --baseline version *topologyVersion* [--yes]

Автоматическая настройка базовой топологии относится к автоматическому обновлению базовой топологии после того, как топология остается стабильной в течение определенного периода времени. Для in-memory кластеров автонастройка включена по умолчанию, а время ожидания равно 0. Это означает, что базовая топология изменяется сразу после того, как серверные узлы присоединяются к кластеру или покидают его. Для кластеров с персистентностью автоматическая настройка базовой топологии отключена по умолчанию. Для ее включения нужно использовать следующую команду:

```
control.sh --baseline auto_adjust enable timeout 30000
```

Время ожидания устанавливается в миллисекундах. Базовая топология устанавливается на текущую топологию при истечении заданного количества миллисекунд после последнего события JOIN/LEFT/FAIL. Каждое новое событие JOIN/LEFT/FAIL перезапускает обратный отсчет времени ожидания.

Чтобы отключить автоматическую настройку базовой топологии, нужно использовать следующую команду:

```
control.sh --baseline auto_adjust disable
```

9.1.3 Управление транзакциями

Управляющий скрипт позволяет получить информацию о транзакциях, выполняемых в кластере. Также существует возможность отменить определенные транзакции.

Следующая команда возвращает список транзакций, удовлетворяющих заданному фильтру (или всех транзакций, если фильтр не указан):

```
control.sh --tx <transaction filter> --info
```

Параметры фильтра транзакций перечислены в таблице 14.

Таблица 14 - Параметры фильтра транзакций

Параметр	Описание
--xid XID	Идентификатор транзакции.
--min-duration SECONDS	Минимальное количество секунд, в течение которых выполнялась транзакция.
--min-size SIZE	Минимальный размер транзакции.
--label LABEL	Метка пользователя для транзакций. Можно использовать регулярное выражение.
--servers|--clients	Ограничение области действия либо сервером, либо клиентскими узлами.
--nodes nodeId1,nodeId2…	Список согласованных идентификаторов узлов, с которых необходимо получать транзакции.
--limit NUMBER	Ограничение количества транзакций до заданного значения.
--order DURATION|SIZE|START_TIME	Параметр, используемый для сортировки вывода.

Для отмены транзакций необходимо воспользоваться следующей командой:

```
control.sh --tx <transaction filter> --kill
```

9.1.4 Обнаружение конфликтов в транзакциях

Команда contention определяет, когда несколько транзакций конкурируют за блокировку одного и того же ключа. Команда полезна, если есть длительные или зависшие транзакции.

```
# Reports all keys that are point of contention for at least 5 transactions on all cluster nodes.
```

```
control.sh|bat --cache contention 5

# Reports all keys that are point of contention for at least 5 transactions on specific server node.
control.sh|bat --cache contention 5 f2ea-5f56-11e8-9c2d-fa7a
```

9.1.5 Мониторинг состояния кэша

Одной из наиболее важных команд, предоставляемых control.sh|bat, является --cache list, которая используется для мониторинга кэша. Команда предоставляет список развернутых кэшей и их параметры affinity и распределения, а также распределение в группах кэшей. Также есть команда для просмотра существующих атомарных последовательностей.

```
# Displays a list of all caches

control.sh|bat --cache list .

# Displays a list of caches whose names start with "account-".
control.sh|bat --cache list account-.*

# Displays info about cache group distribution for all caches.
control.sh|bat --cache list . --groups

# Displays info about cache group distribution for the caches whose names start with "account-".
control.sh|bat --cache list account-.* --groups

# Displays info about all atomic sequences.
control.sh|bat --cache list . --seq

# Displays info about the atomic sequnces whose names start with "counter-".
control.sh|bat --cache list counter-.* --seq
```

9.1.6 Уничтожение кэшей

Для уничтожения определенных кэшей можно использовать управляющий скрипт.

```
control.sh|bat --cache destroy --caches cache1,...,cacheN|--destroy-all-caches
```

Параметры:

- --caches cache1,…,cacheN — задает список разделенных запятыми имен кэшей, которые необходимо уничтожить;
- --destroy-all-caches — безвозвратно удаляет все созданные пользователем кэши.

9.1.7 Управление индексами

Приведенные ниже команды позволяют получить конкретную информацию об индексах и запустить процесс перестроения индексов.

Чтобы получить список всех индексов, соответствующих указанным фильтрам, нужно воспользоваться командой:

```
control.sh --cache indexes_list [--node-id nodeId] [--group-name grpRegExp] [--cache-name cacheRegExp] [--index-name idxNameRegExp]
```

Параметры команды:

• --node-id nodeId - идентификатор узла для выполнения job. Если ID не указан, узел выбирается гридом.

• --group-name regExp - регулярное выражение, включающее фильтрацию по имени группы кэша.

• --cache-name regExp - регулярное выражение, включающее фильтрацию по имени кэша.

• --index-name regExp - регулярное выражение, включающее фильтрацию по имени индекса.

9.1.8 Работа со свойствами кластера

Скрипт control.sh|bat предоставляет возможность работы со статистикой SQL.

Для получения полного списка доступных свойств, нужно использовать команду --property list. Эта команда возвращает список всех доступных для работы свойств:

```
control.sh --property list
```

Можно установить значение свойства с помощью команды --property set. Например, чтобы включить или отключить статистику SQL при использовании кластера, нужно указать значения ON, OFF или NO_UPDATE:

```
control.sh --property set --name 'statistics.usage.state' --val 'ON'
```

Также можно получить значение свойства с помощью команды --property get. Например:

```
control.sh --property get --name 'statistics.usage.state'
```

9.2 Visor CMD

Интерфейс командной строки Visor (CMD) — это инструмент командной строки для мониторинга кластеров РЕД КВАНТ. Он предоставляет базовую статистику об узлах кластера, кэшах и вычислительных задачах (рисунок 32). Также он позволяет управлять размером кластера, запуская или останавливая узлы.

Примечание: Скрипт управления — это еще один инструмент командной строки, разработанный сообществом Ignite. Он дополняет и расширяет возможности Visor CMD.

Рисунок 32 - Visor CMD

Для старта Visor CMD РЕД КВАНТ запускает скрипт IGNITE_HOME/bin/ignitevisorcmd.{sh|bat}. Чтобы подключить Visor CMD к кластеру, необходимо воспользоваться командой open.

В таблице 15 представлены команды, которые поддерживаются Visor. Чтобы получить полную информацию о команде, необходимо ввести help "cmd" или ? "cmd".

Таблица 15 - Команды, поддерживаемые Visor

Команда	Псевдоним	Описание
ack		Подтверждает аргументы на всех удаленных узлах.
alert		ОповещАЕТ о событиях, определенных пользователем.
cache		Выводит статистику кэша, очищает кэш, выводит список всех записей из кэша.
close		Отключает консоль Visor CMD от кластера.
config		Выводит конфигурации узлов.
deploy		Копирует файл или папку на удаленный хост.
disco		Выводит журнал изменений топологии.
events		Выводит список событий из узла.

Команда	Псевдоним	Описание
gc		Запускает GC (сборщик мусора) на удаленных узлах.
help	?	Помощь Visor CMD.
kill		Уничтожает или перезапускает узел.
log		Запускает или останавливает ведение журнала событий в масштабе кластера.
mclear		Очищает переменные памяти Visor CMD.
mget		Возвращает переменные памяти Visor CMD.
mlist		Выводит переменные памяти Visor CMD.
node		Выводит статистику узла.
open		Подключает Visor CMD к кластеру.
ping		Пингует узел.
quit		Завершает соединение Visor CMD.
start		Запускает или перезапускает удаленные узлы.
status	!	Выводит подробный статус Visor CMD.
tasks		Выводит статистику выполнения задач.
top		Выводит текущую топологию кластера.
vvm		Открывает VisualVM для узлов в кластере.

9.3 РАБОТА С SQL ПРИ ПОМОЩИ КОНСОЛИ SQLLINE.SH

Инструмент командной строки для подключения к SQL.

РЕД КВАНТ поставляется с инструментом SQLLine — консольной утилитой для подключения к реляционным базам данных и выполнения SQL команд.

Чтобы подключить SQLLine к кластеру, необходимо в каталоге {IGNITE_HOME}/bin запустить sqlline.sh -u jdbc:ignite:thin:[host]. Затем заменить [host] фактическим значением. Например:

```
./sqlline.sh --verbose=true -u jdbc:ignite:thin://127.0.0.1/
```

Чтобы просмотреть различные доступные в SQLLine параметры, нужно использовать параметр -h или help:

```
./sqlline.sh -h
```

```
./sqlline.sh --help
```

Если для кластера включена аутентификация, то для подключения SQLLine к кластеру из каталога {IGNITE_HOME}/bin необходимо запустить `jdbc:ignite:thin://[адрес]:[порт];user=[имя пользователя];password=[пароль]. Затем необходимо заменить [адрес], [порт], [имя пользователя] и [пароль] фактическими значениями. Например:

```
./sqlline.sh --verbose=true -u "jdbc:ignite:thin://127.0.0.1:10800;user=ignite;password=ignite"
```

Если аутентификация не установлена, [имя пользователя] и [пароль] нужно пропустить.

Примечание: При подключении из среды bash нужно обязательно заключать URL-адрес подключения в кавычки " " следующим образом: "jdbc:ignite:thin://[адрес]:[порт];user=[имя пользователя];password=[пароль]"

Список поддерживаемых команд SQLLine приведен в таблице 16.

Таблица 16 - Команды SQLLine

Команда	Описание
!all	Выполнить указанный SQL для всех текущих подключений.
!batch	Запустить или выполнить пакет операторов SQL.
!brief	Включить режим краткого вывода.
!closeall	Закрыть все текущие открытые соединения.
!columns	Показать столбцы таблицы.
!connect	Подключиться к базе данных.
!dbinfo	Вывести список метаданных текущего соединения.
!dropall	Удалить все таблицы в базе данных.
!go	Перейти на другое активное соединение.
!help	Показать справочную информацию.
!history	Показать историю команд.
!indexes	Показать индексы таблицы.
!list	Показать все активные соединения.
!manual	Показать руководство по SQLLine.
!metadata	Вызывать произвольные команды метаданных.
!nickname	Создать понятное имя для подключения (обновление командной строки).
!outputformat	Изменить способ отображения результатов SQL.
!primarykeys	Показать столбцы первичного ключа для таблицы.
!properties	Подключиться к базе данных, определенной в указанном файле свойств.
!quit	Закрыть SQLLine.
!reconnect	Повторно подключиться к текущей базе данных.
!record	Начать запись всех выходных данных команд SQL.
!run	Выполнить скрипт команд.
!script	Сохранить выполненные команды в файл.
!sql	Выполнить SQL-запрос к базе данных.
!tables	Перечислить все таблицы в базе данных.

Команда	Описание
!verbose	Включить режим подробного вывода.

Следует отметить, что приведенный выше список может быть неполным. Можно добавить поддержку дополнительных команд SQLLine.

9.4 GUI клиент для работы с SQL DBeaver

РЕД КВАНТ JDBC и ODBC драйверы позволяют подключаться к кластеру и обрабатывать хранящиеся там данные с помощью DBeaver - бесплатного универсального инструмента для работы с базами данных с открытым исходным кодом для разработчиков и администраторов баз данных.

Для этого необходимо настроить драйвер JDBC или ODBC, и в данном пункте будет показано, как выполнить эти основные шаги настройки для инструмента DBeaver.

9.4.1 Установка и настройка DBeaver

РЕД КВАНТ поставляется с собственной реализацией драйвера JDBC, который DBeaver может использовать для работы с данными, хранящимися в распределенном кластере КВАНТ.

1. Необходимо загрузить и установить DBeaver для выбранной операционной системы.
2. После установки DBeaver необходимо запустить его и выбрать пункт меню Database->Driver Manager для настройки драйвера РЕД КВАНТ JDBC.
3. Указать Apache Ignite в качестве имени базы данных/драйвера и нажать на кнопку New (рисунок 33):

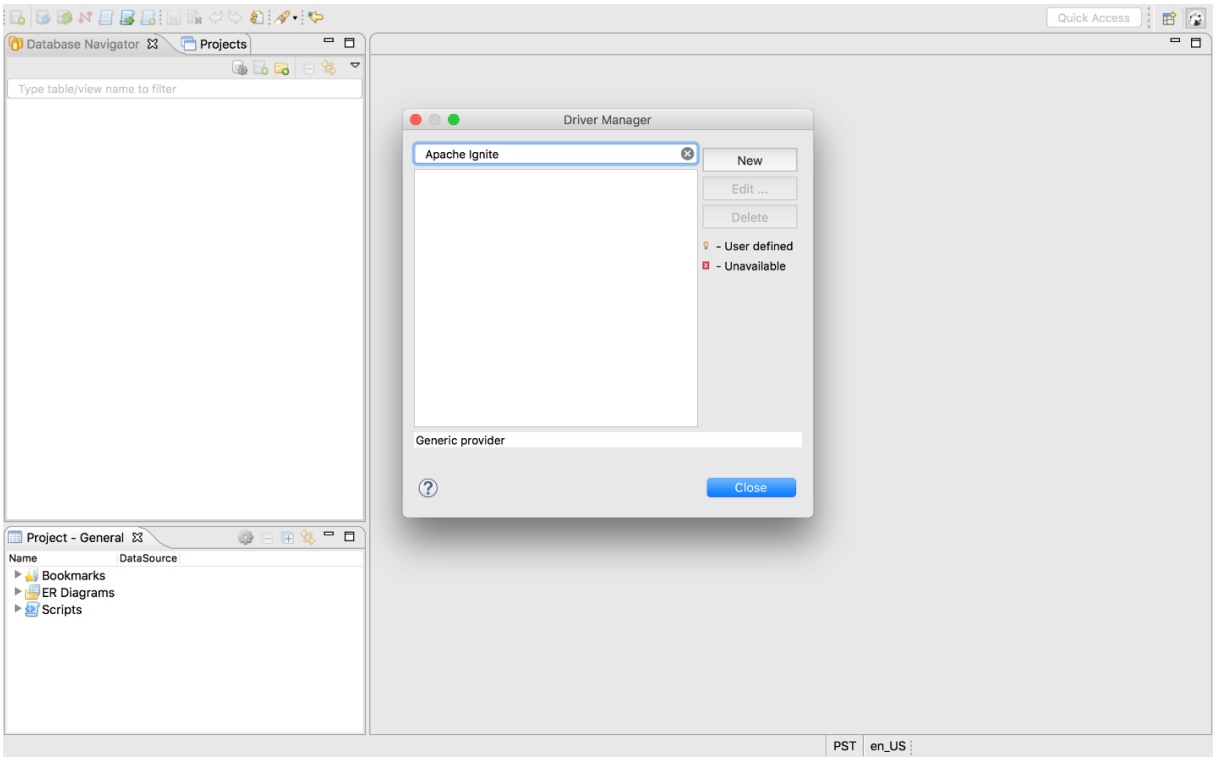

Рисунок 33 - Настройка DBeaver

Далее необходимо заполнить параметры в открывшемся окне следующим образом (рисунок 34):

- Driver Name - пользовательское имя, для упрощения установлено имя Apache Ignite;
- Class Name - нужно указать org.apache.ignite.IgniteJdbcThinDriver;
- URL Template - строка подключения JDBC Ignite, которая используется при настройке; в данном случае это jdbc:ignite:thin://127.0.0.1/;
- Default Port- порт 10800 по умолчанию используется JDBC-драйвером Ignite. См. JDBC драйвер, если нужно изменить порт или изменить строку подключения URL-адреса, указанную выше;
- Libraries вкладка — нужно нажать кнопку Add file и найти файл {apache-ignite-version}/libs/ignite-core-{version}.jar, содержащий драйвер JDBC Ignite. После добавления JAR необходимо нажать Find Class и явно выбрать имя Driver Class в раскрывающемся меню.

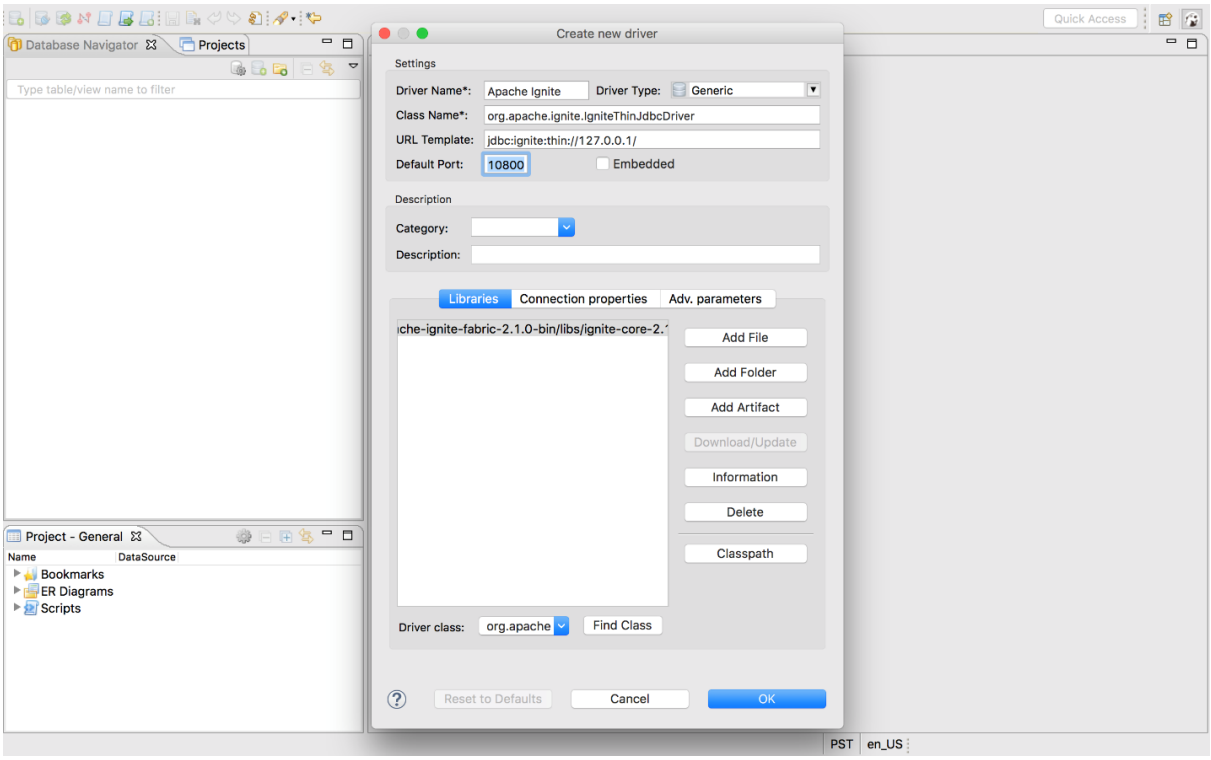

Рисунок 34 - Настройка DBeaver

Далее нужно нажать на кнопку OK, чтобы завершить настройку и закрыть диалоговое окно Driver Manager (рисунок 35). Среди списка драйверов можно увидеть созданный Apache Ignite.

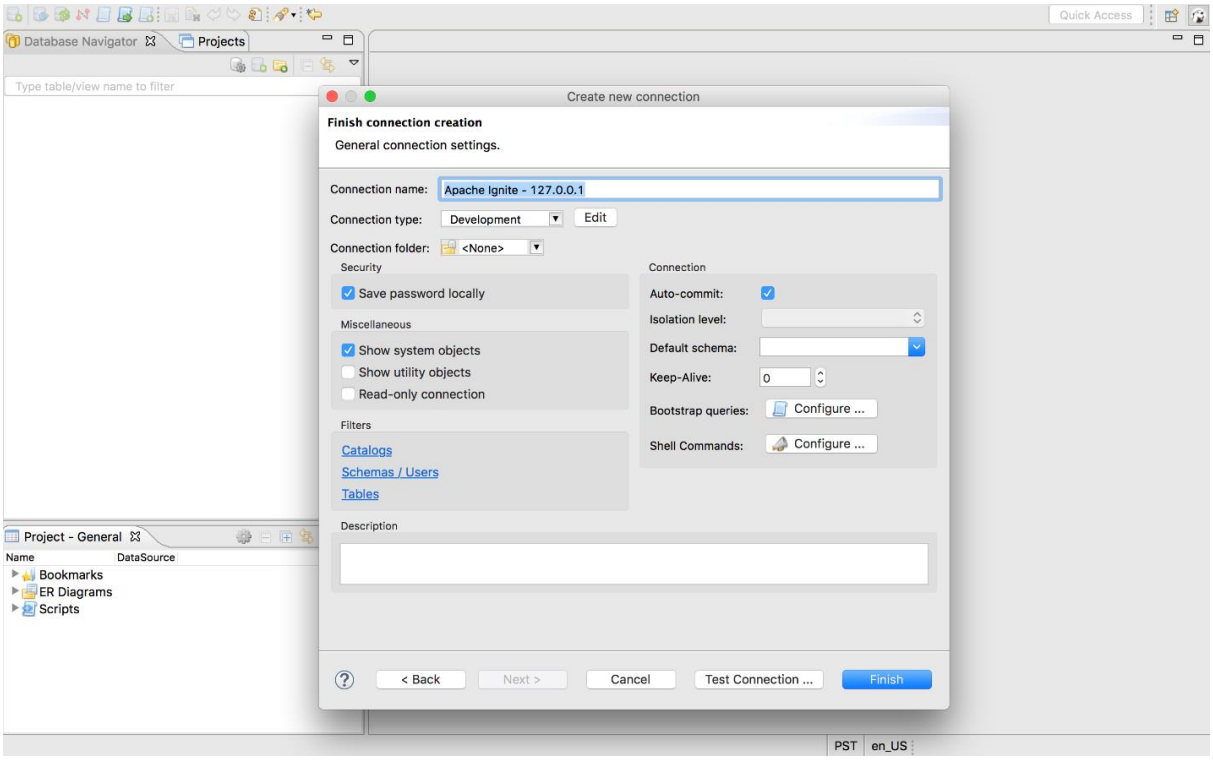

Рисунок 35 - Созданный драйвер Apache Ignite

9.4.2 Подключение к кластеру

Необходимо выбрать пункт меню Database->New Connection, найти в списке Apache Ignite, созданный в предыдущем пункте и нажать на кнопку Next > (рисунок 36).

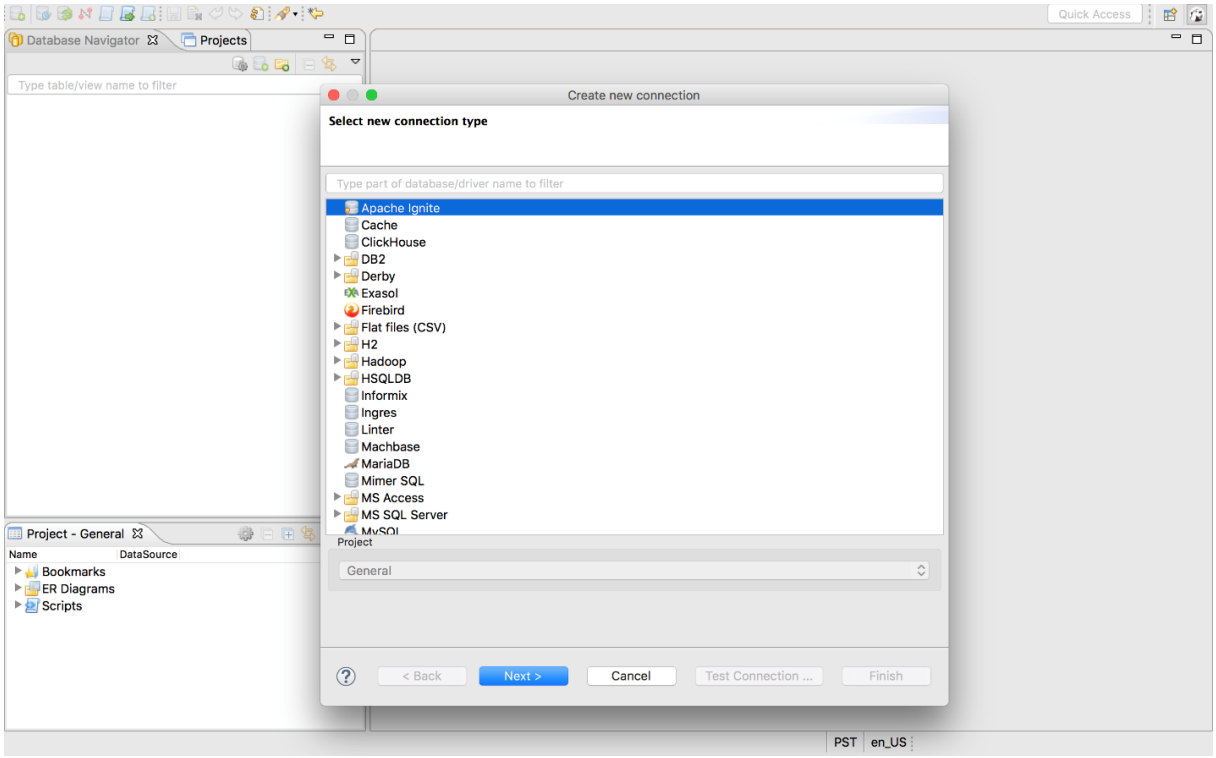

Рисунок 36 - Подключение к кластеру

Следует убедиться, что параметр указывает на ранее настроенную строку подключения, и затем нажать кнопку Test Connection … (рисунок 37), чтобы проверить подключение между DBeaver и кластером Apache Ignite.

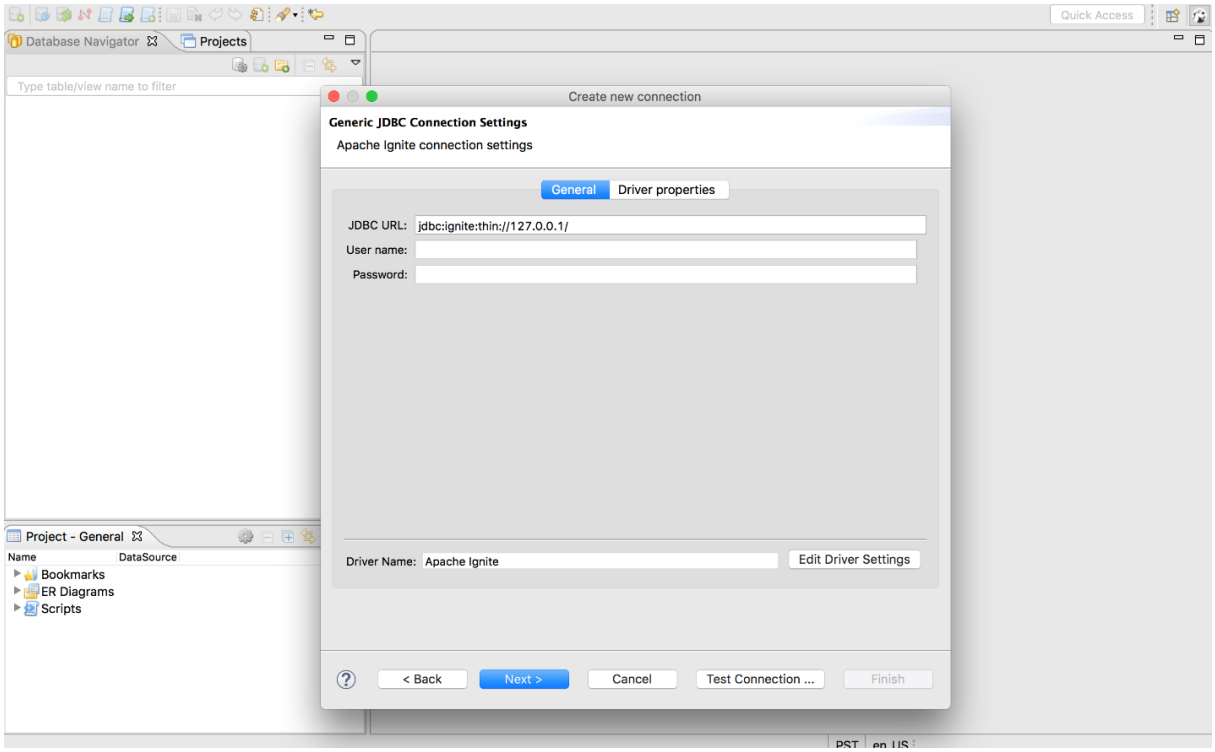

Рисунок 37 - Проверка подключения

Если тест пройден, необходимо нажать Next >.

Если к базе данных нет прямого доступа, можно использовать туннель SSH (рисунок 38), дополнительную информацию о настройке SSH можно найти на странице конфигурации SSH: https://dbeaver.com/docs/wiki/SSH-Configuration/.

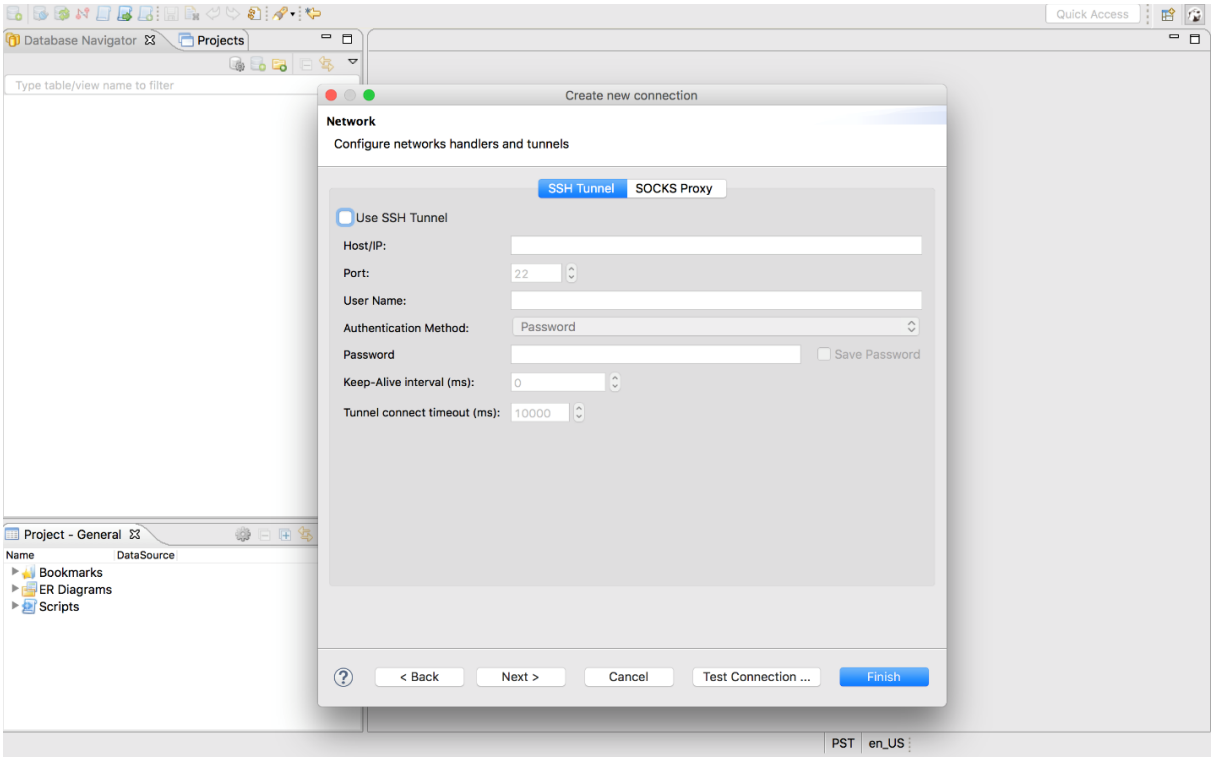

Рисунок 38 - Подключение с помощью туннеля SSH

Подтвердить настройки на последнем экране, нажав на кнопку Finish (рисунок 39).

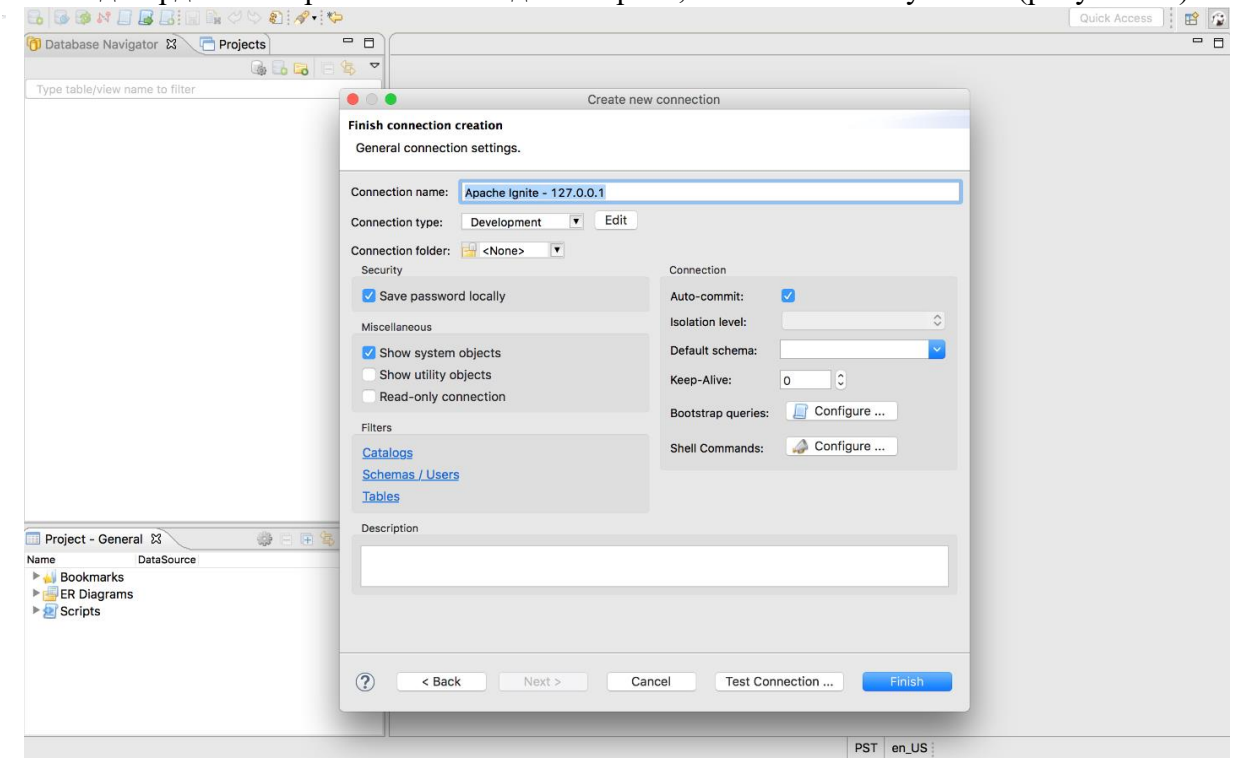

Рисунок 39 - Подтверждение настройки

Apache Ignite появится во вкладке Database Navigator (рисунок 40).

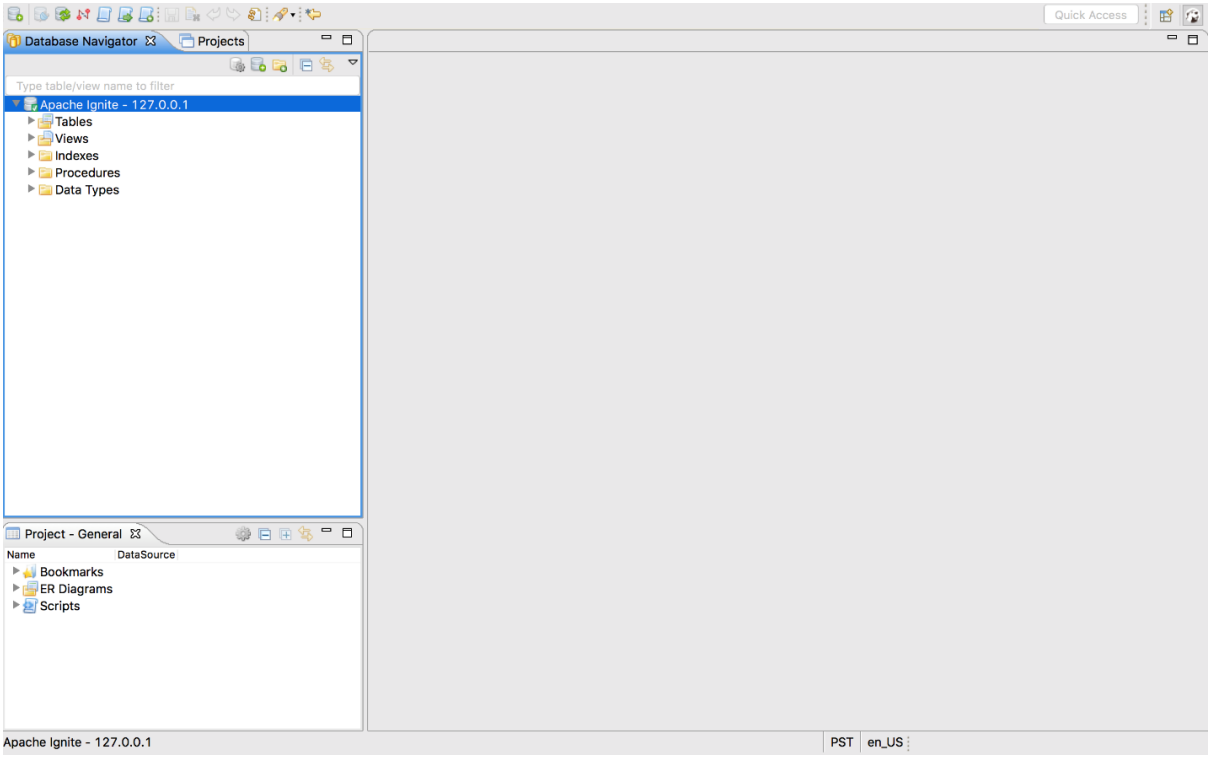

Рисунок 40 - Database Navigator

9.4.3 Запросы к базе данных

Следующим шагом является определение схемы SQL, вставка и запрос некоторых данных с помощью Dbeaver. Для запроса данных следует использовать операторы SQL Data Definition и Data Manipulation, доступные в **Приложение В. DDL** и Приложение Г. DML.

Следует убедиться, что инструмент подключен к кластеру. Нужно щелкнуть правой кнопкой мыши, чтобы увидеть специальное меню и открыть SQL Editor (рисунок 41):

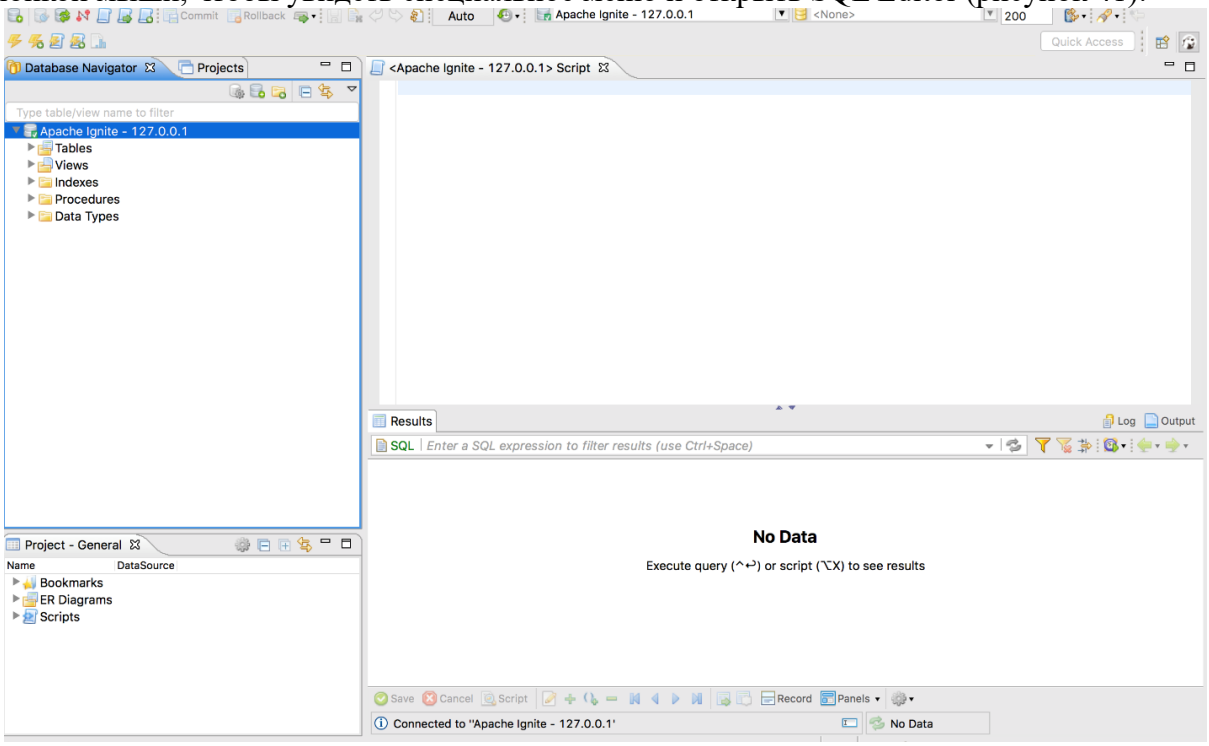

Рисунок 41 - SQL Editor

Далее нужно вставить операторы в окно скрипта DBeaver и щелкнуть пункт меню Execute SQL Statement.

Необходимо обратиться к **Приложение В. DDL** и Приложение Г. DML, которые включают дополнительные операторы, которые можно выполнить для определенной схемы SQL, или создать собственную конфигурацию кластера и начать работать с ней.

10 МОНИТОРИНГ РАБОТЫ КЛАСТЕРА

На этом этапе кластер РЕД КВАНТ уже настроен и работает. Приложения используют кластер РЕД КВАНТ для записи и чтения данных в него и из него. Однако РЕД КВАНТ — это не система «установи и забудь». РЕД КВАНТ — это система на основе JVM, рассчитанная на подход fast fail. Таким образом, для своевременного принятия мер требуется мониторинг.

РЕД КВАНТ построен на JVM, а JVM может использовать JMX (Java Management Extension). Другими словами, используя JMX, можно управлять системой удаленно, собирая метрики (кэша или памяти), включая память, ЦП, потоки или любую другую часть системы, которая была инструментирована в JMX. Инструментарий позволяет приложению или системе предоставлять специфичную для приложения информацию, которая будет собираться внешними инструментами.

JMX был представлен в версии Java 5.0 для управления и мониторинга ресурсов во время выполнения. Используя JMX, можно отслеживать использование памяти, сборку мусора, загруженные классы, количество потоков и т. д. с течением времени.

MBeans или Managed Beans — это особый тип JavaBeans, использующий ресурс приложения или JVM и доступный извне. На рисунке 42 представлена высокоуровневая архитектура JMX.

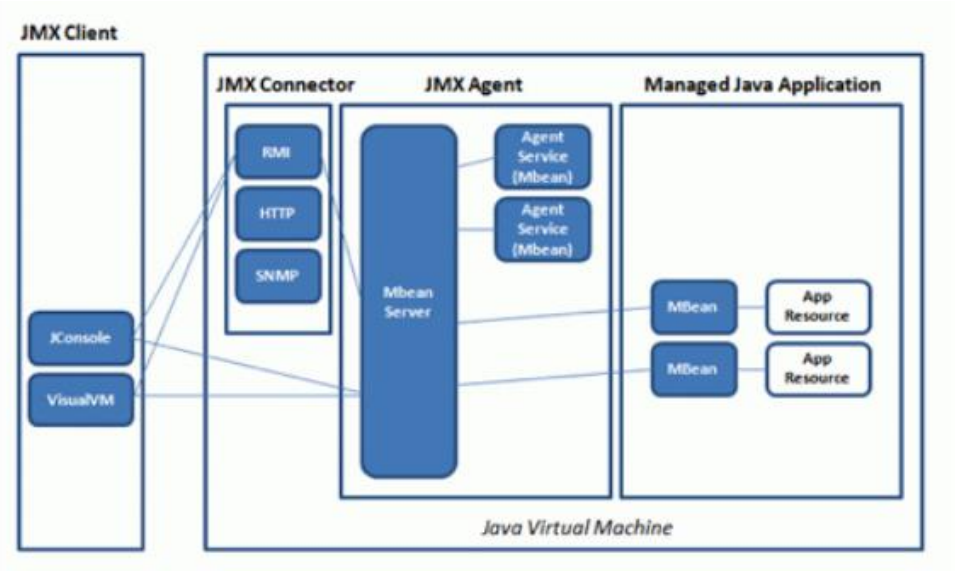

Рисунок 42 - Архитектура JMX

РЕД КВАНТ предоставляет несколько JMX MBeans для сбора и мониторинга метрик кэша и памяти следующим образом:

• CacheMetricsMXBean. Mbean предоставляет доступ к дескриптору кэша.

• CacheGroupMetricsMXBean. Mbean предоставляет метрики для кэшей, связанных с определенной группой CacheGroup.

• DataRegionMetricsMXBean. Mbean предоставляет доступ к DataRegionMetrics локального узла РЕД КВАНТ.

• DataStorageMetricsMXBean. Mbean позволяет отслеживать и настраивать метрики персистентности.

JConsole или VisualVM — это стандартные инструменты, поставляемые с Java, для управления компонентами MBean. Для VisualVM необходимо установить плагин VisualVM-MBeans. VisualVM похож на JConsole, но с более продвинутыми функциями мониторинга, такими как профилирование ЦП и визуализация GC (сборщика мусора).

10.1 Метрики

РЕД КВАНТ предоставляет большое количество метрик, полезных для мониторинга кластера или приложения. Для доступа к этим метрикам через JMX можно использовать JMX и инструмент мониторинга, такой как JConsole. Также можно получить к ним доступ программно.

Далее представлены наиболее полезные метрики, сгруппированные в различные общие категории в зависимости от задачи мониторинга.

10.1.1 Включение JMX для РЕД КВАНТ

По умолчанию автоматическая настройка JMX отключена. Чтобы включить ее, необходимо настроить следующие переменные среды:

• Для control.sh настроить переменную CONTROL_JVM_OPTS.

• Для ignite.sh настроить переменную JVM_OPTS.

Например:

```
JVM_OPTS="-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=${JMX_PORT} \
-Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false"
```

10.1.2 MBean ObjectName

У каждого JMX Mbean есть ObjectName. ObjectName используется для идентификации бина. ObjectName состоит из домена и списка ключевых свойств и может быть представлен в виде строки следующим образом:

```
domain: key1 = value1 , key2 = value2
```

Все метрики РЕД КВАНТ имеют один и тот же домен: org.apache.<classloaderId>, где идентификатор загрузчика классов является необязательным (опускается, если установлено IGNITE_MBEAN_APPEND_CLASS_LOADER_ID=false). Кроме того, у каждой метрики есть два свойства: группа и имя. Например:

```
org.apache:group=SPIs,name=TcpDiscoverySpi
```

Этот MBean предоставляет различные метрики, связанные с обнаружением узлов.

MBean ObjectName можно использовать для идентификации бина в таких инструментах с графическим интерфейсом, как JConsole. Например, JConsole отображает компоненты

MBean в виде древовидной структуры, где все компоненты сначала сгруппированы по домену, а затем по свойству «группа» (рисунок 43).

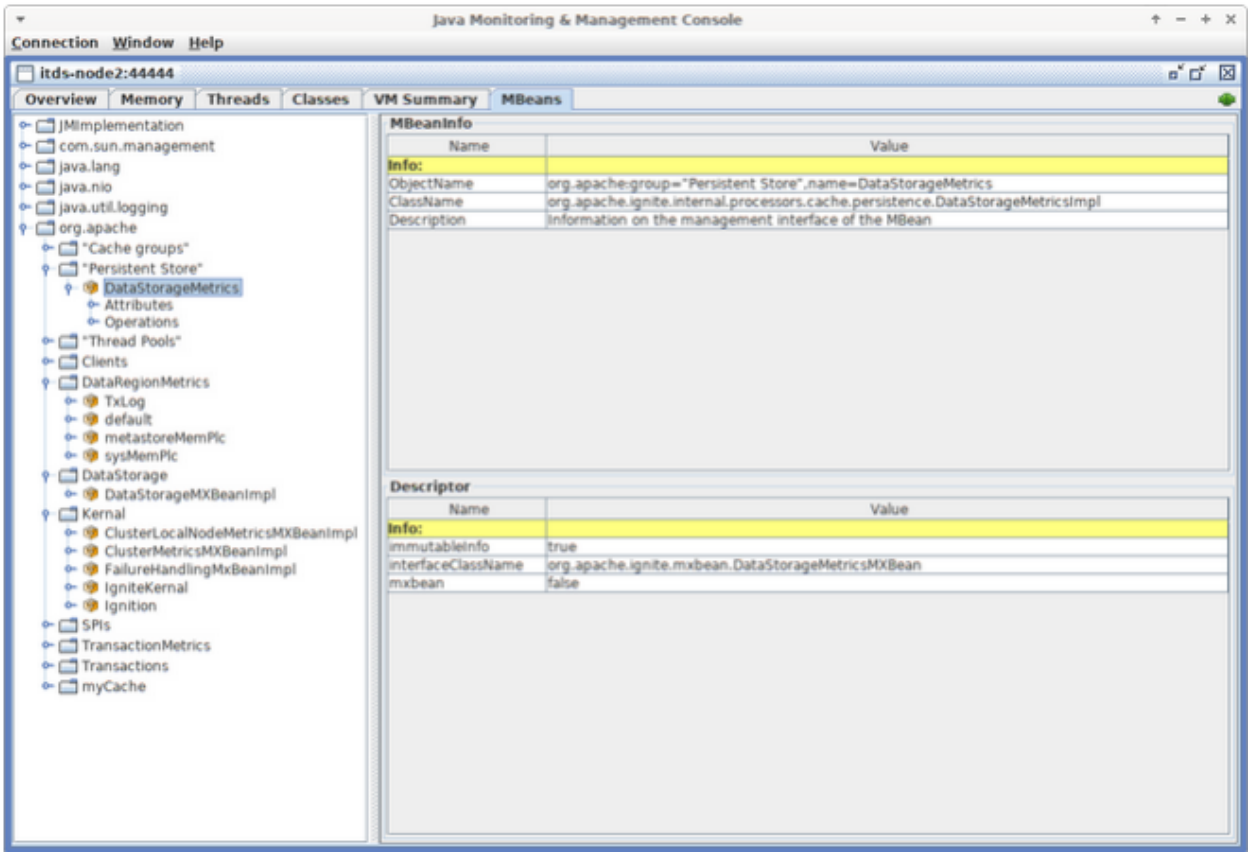

Рисунок 43 - Интерфейс JConsole

10.1.3 Мониторинг количества данных

Если встроенная персистентность не используется (т. е. все данные хранятся в памяти), следует отслеживать использование ОЗУ. Если используется встроенная персистентность, помимо оперативной памяти следует следить за размером хранилища данных на диске.

Размер загруженных на узел данных доступен на разных уровнях агрегации. Можно следить за:

• Общим размером данных, которые узел хранит на диске или в оперативной памяти. Он представляет собой сумму размеров каждой сконфигурированной области данных (в простейшем случае только области данных по умолчанию) плюс размеры системных областей данных.

• Размером определенной области данных на узле. Размер области данных представляет собой сумму размеров всех групп кэша.

• Размером определенного кэша/группы кэша на узле, включая резервные разделы.

Эти метрики могут быть включены/отключены для каждого уровня отдельно и доступны через различные компоненты JMX, перечисленные ниже.

10.1.3.1 Выделенное пространство и фактический размер данных

Не существует способа получить точный размер данных (ни в ОЗУ, ни на диске). Вместо этого есть два варианта его оценки.

Можно получить размер пространства, выделенного для хранения данных. («Пространство» здесь относится либо к пространству в ОЗУ, либо к пространству на диске, в зависимости от того, используется встроенная персистентность или нет.) Когда хранилище заполняется и необходимо добавить больше записей, выделяется пространство. Однако при удалении записей из кэшей пространство не освобождается. Оно повторно используется при необходимости добавления новых записей в хранилище при последующих операциях записи. Поэтому размер выделенного пространства не уменьшается при удалении записей из кэшей. Выделенный размер доступен на уровне хранилища данных, области данных и метрик группы кэша. Метрика называется TotalAllocatedSize.

Также можно получить оценку фактического размера данных, умножив количество используемых страниц данных на коэффициент заполнения. Коэффициент заполнения — это отношение размера данных на странице к размеру страницы, усредненное по всем страницам. Количество используемых страниц и коэффициент заполнения доступны на уровне метрик области данных.

Чтобы получить предполагаемый общий объем данных на узле, нужно сложить предполагаемый размер всех областей данных.

10.1.3.2 Мониторинг использования оперативной памяти

Объем данных в ОЗУ можно отслеживать для каждой области данных с помощью следующих метрик Mbean (таблица 17):

Имя объекта Mbean:

group=DataRegionMetrics,name=<Data Region name>

Таблица 17 - Метрики для мониторинг использования оперативной памяти

Атрибут	Тип	Описание	Область действия
PagesFillFactor	float	Средний размер данных на страницах в отношении к размеру страницы. Если включена встроенная персистентность, эта метрика применима только к постоянному хранилищу (т. е. к страницам на диске).	Узел
TotalUsedPages	long	Количество страниц данных, которые используются в данный момент. Если включена встроенная персистентность, эта метрика применима только к постоянному хранилищу (т. е. к страницам на диске).	Узел
PhysicalMemoryPages	long	Количество выделенных страниц в оперативной памяти.	Узел
PhysicalMemorySize	long	Размер выделенного пространства в оперативной памяти в байтах.	Узел

Если есть несколько областей данных, для получения общего размера данных на узле нужно сложить размеры всех областей данных.

10.1.3.3 Мониторинг размера хранилища

Постоянное хранилище, если оно включено, сохраняет все данные приложения на диске. Общий объем данных, которые каждый узел хранит на диске, состоит из постоянного хранилища (данные приложений), файлов WAL и архивных файлов WAL.

10.1.3.3.1 Размер постоянного хранилища

Чтобы контролировать размер постоянного хранилища на диске, нужно использовать следующие метрики (таблица 18):

Имя объекта Mbean:

```
group="Persistent Store",name=DataStorageMetrics
```

Таблица 18 - Метрики для мониторинга размера постоянного хранилища

Атрибут	Тип	Описание	Область действия
TotalAllocatedSize	long	Размер пространства, выделенного на диске для всего хранилища данных (в байтах). Стоит обратить внимание, что когда встроенная персистентность отключена, эта	Узел

Атрибут	Тип	Описание	Область действия
		метрика показывает общий размер выделенного пространства в оперативной памяти.	
WalTotalSize	long	Общий размер файлов WAL в байтах, включая архивные файлы WAL.	Узел
WalArchiveSegments	int	Количество разделов WAL в архиве.	Узел

В таблице 19 представлены основные операции.

Таблица 19 - Основные операции

Операция	Описание
enableMetrics	Во время выполнения включает сбор метрик, связанных с постоянным хранилищем.
disableMetrics	Отключает сбор метрик.

10.1.3.3.2 Размер области данных

Для каждой настроенной области данных РЕД КВАНТ создает отдельный JMX Bean, который предоставляет конкретную информацию об области. Сбор метрик для областей данных по умолчанию отключен. Его можно включить в конфигурации области данных или через JMX во время выполнения.

Размер области данных на узле включает в себя размер всех принадлежащих узлу разделов (включая резервные разделы) для всех кэшей в этой области данных.

Метрики области данных доступны в следующих компонентах MBean:

Имя объекта Mbean:

group=DataRegionMetrics,name=<Data Region name>

Метрики представлены в таблице 20.

Таблица 20 - Метрики для мониторинга размера области данных

Атрибут	Тип	Описание	Область действия
TotalAllocatedSize	long	Размер пространства, выделенного для этой области данных (в байтах). Следует обратить внимание, что когда встроенная персистентность отключена, эта метрика показывает общий размер выделенного пространства в оперативной памяти.	Узел

Атрибут	Тип	Описание	Область действия
PagesFillFactor	float	Средний объем данных на страницах в соотношении к размеру страницы.	Узел
TotalUsedPages	long	Количество используемых в данный момент страниц данных.	Узел
PhysicalMemoryPages	long		Узел
PhysicalMemorySize	long		Узел

В таблице 21 представлены операции метрики.

Таблица 21 - Операции метрики

Операция	Описание
enableMetrics	Включает сбор показателей для области данных.
disableMetrics	Отключает сбор показателей для области данных.

10.1.3.3.3 Размер группы кэша

Если группы кэша не используются, каждый кэш будет отдельной группой. Для каждой группы кэша существует отдельный компонент JMX. Имя бина соответствует названию группы.

Имя объекта Mbean:

group="Cache groups",name=<Cache group name>

Метрика TotalAllocatedSize (с типом long) измеряет объем пространства, выделенного для группы кэша на этом узле.

10.1.4 Мониторинг Checkpoint операций

Checkpoint может замедлить работу кластера. Можно отслеживать, сколько времени занимает каждая checkpoint операция, чтобы можно было настроить свойства, влияющие на checkpoint. Также можно следить за производительностью диска, чтобы увидеть, не вызвано ли замедление внешними причинами.

Имя объекта Mbean:

group="Persistent Store",name=DataStorageMetrics

Метрики представлены в таблице 22.

Таблица 22 - Метрики для мониторинга checkpoint операций

Атрибут	Тип	Описание	Область действия
DirtyPages	long	Количество страниц в памяти, которые были изменены, но еще не	Узел

		синхронизированы с диском. Они будут записаны на диск во время следующей checkpoint.	
LastCheckpointDuration	long	Время в миллисекундах, которое потребовалось для создания последней checkpoint.	Узел
CheckpointBufferSize	long	Размер буфера checkpoint.	Глобально

10.1.5 Мониторинг ребалансировки

Ребалансировка — это процесс перемещения разделов между узлами кластера, для распределения данных сбалансированным образом. Ребалансировка запускается, когда к кластеру присоединяется новый узел или существующий узел покидает кластер.

Если есть несколько кэшей, они будут последовательно ребалансированы. Существует несколько метрик, представленных в таблице 23, которые можно использовать для отслеживания хода процесса ребалансировки для определенного кэша.

Имя объекта Mbean:

group=<cache name>,name=org.apache.ignite.internal.processors.cache.CacheLocalMetricsMXBeanImpl

Таблица 23 - Метрики для отслеживания процесса перебалансировки

Атрибут	Тип	Описание	Область действия
RebalancingStartTime	long	Эта метрика показывает время начала ребалансировки локальных разделов для кэша. Эта метрика вернет 0, если локальные разделы не участвуют в ребалансировке. Время возвращается в миллисекундах.	Узел
EstimatedRebalancingFinishTime	long	Ожидаемое время завершения процесса ребалансировки.	Узел
KeysToRebalanceLeft	long	Количество ключей на узле, которые еще предстоит ребалансировать. Можно отслеживать этот показатель, чтобы узнать, когда завершится процесс ребалансировки.	Узел

10.1.6 Мониторинг топологии

Топология относится к набору узлов в кластере. Существует ряд метрик, которые предоставляют информацию о топологии кластера. Если топология меняется слишком часто или ее размер отличается от ожидаемого, можно проверить, нет ли проблем с сетью.

Имя объекта Mbean:

group=Kernal,name=ClusterMetricsMXBeanImpl

Метрики представлены в таблице 24.

Таблица 24 - Метрики ClusterMetricsMXBeanImpl

Атрибут	Тип	Описание	Область действия
TotalServerNodes	long	Количество серверных узлов в кластере.	Глобально
TotalClientNodes	long	Количество клиентских узлов в кластере.	Глобально
TotalBaselineNodes	long	Количество узлов, зарегистрированных в базовой топологии. Когда узел выходит из строя, он остается зарегистрированным в базовой топологии, и необходимо удалить его вручную.	Глобально
ActiveBaselineNodes	long	Количество узлов, активных в данный момент в базовой топологии.	Глобально

Имя объекта Mbean:

group=SPIs,name=TcpDiscoverySpi

Метрики представлены в таблице 25.

Таблица 25 - Метрики TcpDiscoverySpi

Атрибут	Тип	Описание
Coordinator	String	Идентификатор узла текущего узла-координатора.
CoordinatorNodeFormatted	String	Подробная информация об узле-координаторе.

10.1.7 Мониторинг кэшей

Связанные с кэшем метрики. Для каждого кэша РЕД КВАНТ создаст два компонента JMX MBean, которые будут отображать метрики кэша. Один MBean показывает информацию о кэше для всего кластера, например, общее количество записей в кэше. Другой компонент MBean показывает локальную информацию о кэше, например, количество записей кэша, расположенных на локальном узле.

Имя объекта глобального кэша Mbean:

group=<Cache_Name>,name="org.apache.ignite.internal.processors.cache.CacheClusterMetricsMXBeanImpl"`

Метрикой является CacheSize с типом long, который обозначает общее количество записей в кэше по всем узлам в пределах кластера.

Имя объекта локального кэша Mbean:

group=<Cache Name>,name="org.apache.ignite.internal.processors.cache.CacheLocalMetricsMXBeanImpl"

Метрикой является CacheSize с типом long, который обозначает количество записей кэша, хранящихся на локальном узле.

10.1.7.1 Мониторинг построения и перестроения индексов

Чтобы получить оценку того, сколько времени потребуется для перестроения индексов кэша, можно использовать одну из метрик, перечисленных ниже:

1. IsIndexRebuildInProgress — сообщает, строятся или перестраиваются индексы в данный момент;
2. IndexBuildCountPartitionsLeft — указывает оставшееся количество разделов (по группам кэша) для перестроения индексов.

Метрика IndexBuildCountPartitionsLeft позволяет оценить только приблизительное количество индексов, оставшихся для перестроения. Для более точной оценки нужно использовать метрику кэша IndexRebuildKeyProcessed:

• org.apache.ignite.mxbean.CacheMetricsMXBean#isIndexRebuildInProgress, чтобы узнать, перестраиваются ли индексы для кэша. В настоящее время доступна только локальная метрика org.apache.ignite.internal.processors.cache.CacheLocalMetricsMXBeanImpl#isIndexRebuildInProgress.

• org.apache.ignite.mxbean.CacheMetricsMXBean#getIndexRebuildKeysProcessed, чтобы узнать количество ключей с перестроенными индексами. Если идет перестроение, указывается количество ключей с индексами, которые перестраиваются в текущий момент. В противном

случае указывается общее количество ключей с перестроенными индексами. Значения сбрасываются перед началом каждого перестроения. В настоящее время доступна только локальная метрика org.apache.ignite.internal.processors.cache.CacheLocalMetricsMXBeanImpl#getIndexRebuildKeysProcessed.

10.1.8 Мониторинг транзакций

Если транзакция охватывает несколько узлов (т. е. если ключи, измененные в результате выполнения транзакции, расположены на нескольких узлах), счетчики будут увеличиваться на каждом узле. Например, счетчик TransactionsCommittedNumber будет увеличиваться на каждом узле, где хранятся ключи, затронутые транзакцией.

Имя объекта Mbean:

group=TransactionMetrics,name=TransactionMetricsMxBeanImpl

Метрики представлены в таблице 26.

Таблица 26 - Метрики мониторинга транзакций

Атрибут	Тип	Описание	Область действия
LockedKeysNumber	long	Количество ключей, заблокированных на узле.	Узел
TransactionsCommittedNumber	long	Количество транзакций, которые были совершены на узле	Узел
TransactionsRolledBackNumber	long	Количество транзакций, для которых был выполнен откат.	Узел
OwnerTransactionsNumber	long	Количество транзакций, инициированных на узле.	Узел
TransactionsHoldingLockNumber	long	Количество открытых транзакций, которые удерживают блокировку хотя бы одного ключа на узле.	Узел

10.1.9 Мониторинг клиентских подключений

Метрики, связанные с подключениями JDBC/ODBC или тонкими клиентами.

Имя объекта Mbean:

group=Clients,name=ClientListenerProcessor

Атрибут Connections с типом java.util.List<String> выводит список строк, где каждая строка содержит информацию о соединении.

10.1.10 МОНИТОРИНГ ОЧЕРЕДЕЙ СООБЩЕНИЙ

Когда очереди пулов потоков растут, это означает, что узел не справляется с нагрузкой, или произошла ошибка при обработке сообщений в очереди. Непрерывный рост размера очереди может привести к ошибкам OOM.

10.1.10.1 Очередь сообщений связи

Очередь исходящих сообщений связи содержит коммуникационные сообщения, ожидающие отправки на другие узлы. Если размер увеличивается, значит есть проблема.

Имя объекта Mbean:

group=SPIs,name=TcpCommunicationSpi

Атрибут OutboundMessagesQueueSize с типом int определяет размер очереди исходящих сообщений связи на узле.

10.1.10.2 Очередь сообщений обнаружения

Очередь сообщений об обнаружении.

Имя объекта Mbean:

group=SPIs,name=TcpDiscoverySpi

Метрики представлены в таблице 27.

Таблица 27 - Метрики очереди сообщений об обнаружении

Атрибут	Тип	Описание	Область действия
MessageWorkerQueueSize	int	Размер очереди сообщений обнаружения, ожидающих отправки другим узлам.	Узел
AvgMessageProcessingTime	long	Среднее время обработки сообщения.	Узел

10.2 СТАТИСТИКА ПРОИЗВОДИТЕЛЬНОСТИ

РЕД КВАНТ предоставляет встроенный инструмент для профилирования кластера.

Можно собрать статистику производительности из кластера, а затем создать отчет о производительности.

10.2.1 СБОР СТАТИСТИКИ

Интерфейс JMX и скрипт управления используются для запуска и остановки сбора статистики.

Каждый узел собирает статистику производительности в двоичный файл. Этот файл находится в каталоге Ignite_work_directory/perf_stat/. Маска имени - node-{nodeId}-{index}.prf.

Файлы статистики производительности используются для построения отчета в автономном режиме.

Узлы используют циклический буфер off-heap для временного хранения сериализованной статистики. При достижении определенного размера поток записи сбрасывает буфер в файл. Если буфер переполняется из-за медленного диска, некоторые статистические данные пропускаются. См. пункт Системные свойства для настройки.

Каждый процесс сбора статистики создает новый файл на узлах. Каждый следующий файл имеет такое же имя с соответствующим индексом. Примеры:

- узел-faedc6c9-3542-4610-ae10-4ff7e0600000.prf
- узел-faedc6c9-3542-4610-ae10-4ff7e0600000-1.prf
- узел-faedc6c9-3542-4610-ae10-4ff7e0600000-2.prf

10.2.2 Построение отчета

РЕД КВАНТ предоставляет инструмент для создания отчета из файлов статистики производительности. Инструмент опубликован в репозитории ignite-extensions как расширение performance-statistics-ext.

Чтобы создать отчет о производительности, необходимо выполнить следующие действия:

1. Остановить сбор статистики и поместить файлы со всех узлов в пустой каталог. Например:

```
/path_to_files/
├── node-162c7147-fef8-4ea2-bd25-8653c41fc7fa.prf
├── node-7b8a7c5c-f3b7-46c3-90da-e66103c00001.prf
└── node-faedc6c9-3542-4610-ae10-4ff7e0600000.prf
```

2. Запустить скрипт из релизного пакета инструмента:

```
performance-statistics-tool/build-report.sh path_to_files
```

Отчет о производительности создается в новом каталоге по пути к файлам статистики производительности: путь_к_файлам/отчет_гггг-ММ-дд_ЧЧ-мм-сс/. Чтобы просмотреть отчет, необходимо открыть в браузере отчет_гггг-ММ-дд_ЧЧ-мм-сс/index.html (report_yyyy-MM-dd_HH-mm-ss/index.html).

10.2.3 Управление

В следующем разделе представлена информация о JMX, скрипте управления и свойствах системы.

10.2.3.1 JMX

Управлять сбором статистики производительности можно через интерфейс PerformanceStatisticsMBean. В таблице 28 представлены основные методы управления сбором статистики.

Таблица 28 - Методы JMX

Метод	Описание
start()	Начать сбор статистики производительности в кластере.
stop()	Остановить сбор статистики производительности в кластере.
rotate()	Чередовать сбор статистики производительности в кластере.
started()	Возвращает True, если запущен сбор статистики производительности.

10.2.3.2 Скрипт управления

Можно управлять сбором статистики производительности с помощью скрипта управления.

```
control.sh --performance-statistics [start|stop|rotate|status]
```

Параметры:

• start — начать сбор статистики производительности в кластере.

• stop — становить сбор статистики производительности в кластере.

• rotate — чередовать сбор статистики в кластере.

• status — получить статус сбора статистики производительности в кластере.

10.2.3.3 Системные свойства

В таблице 29 перечислены системные свойства для настройки сбора статистики.

Таблица 29 - Свойства системы

Свойство	Тип	Значение по умолчанию	Описание
IGNITE_PERF_STAT_FILE_MAX_SIZE	Long	32 Гб	Максимальный размер файла статистики производительности в байтах. Сбор статистики производительности прекращается при превышении размера файла.
IGNITE_PERF_STAT_BUFFER_SIZE	Integer	32 Гб	Статистика производительности с учетом размера off-heap буфера в байтах.
IGNITE_PERF_STAT_FLUSH_SIZE	Integer	8 Мб	Минимальный размер пакета статистики производительности для сброса в байтах.
IGNITE_PERF_STAT_CACHED_STRINGS_THRESHOLD	Integer	1024	Максимальное пороговое значение кэшированных строк статистики производительности. Кэширование строк прекращается при превышении порогового значения.

10.3 Трассировка

Ряд API в РЕД КВАНТ предназначен для отслеживания с помощью OpenCensus. Можно собирать распределенные трассировки различных задач, выполняемых в кластере, и использовать эту информацию для диагностики проблем с задержкой.

Для отслеживания используются следующие API-интерфейсы РЕД КВАНТ:

• Обнаружение;

• Коммуникация;

• Обмен;

• Транзакции;

• SQL-запросы.

Для просмотра трассировок необходимо экспортировать их во внешнюю систему. Можно использовать один из экспортеров OpenCensus или написать свой, но в любом случае придется написать код, который регистрирует экспортер в РЕД КВАНТ. Дополнительные сведения см. в разделе Экспорт трассировок.

10.3.1 НАСТРОЙКА ТРАССИРОВКИ

Необходимо включить трассировку OpenCensus в конфигурации узла. Все узлы в кластере должны использовать одну и ту же конфигурацию трассировки.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">


  <property name="tracingSpi">
    <bean class="org.apache.ignite.spi.tracing.opencensus.OpenCensusTracingSpi"/>
  </property>

</bean>
```

10.3.2 ВКЛЮЧЕНИЕ ВЫБОРКИ ТРАССИРОВКИ

Когда запускается кластер с указанной выше конфигурацией, РЕД КВАНТ не собирает трассировки. Необходимо включить выборку трассировки для определенного API во время выполнения. По желанию можно включать и выключать выборку трассировки, например, только на период устранения неполадок.

Это можно сделать двумя способами:

• через скрипт управления из командной строки;

• программно.

Трассировки собираются с заданной вероятностной частотой выборки. Частота указывается в виде значения от 0,0 до 1,0 включительно: 0 означает отсутствие выборки, 1 означает постоянную выборку.

Когда для частоты выборки установлено значение больше 0, РЕД КВАНТ собирает трассировки. Чтобы отключить сбор трассировки, нужно установить частоту дискретизации на 0.

Способы включения выборки трассировки описаны ниже.

10.3.2.1 Использование скрипта управления

Необходимо перейти в каталог {IGNITE_HOME}/bin установки РЕД КВАНТ. Включить экспериментальные команды в управляющем скрипте:

```
export IGNITE_ENABLE_EXPERIMENTAL_COMMAND=true
```

Включить трассировку для определенного API:

```
./control.sh --tracing-configuration set --scope TX --sampling-rate 1
```

10.3.2.2 Программно

После запуска узла можно включить выборку трассировки следующим образом:

```
Ignite ignite = Ignition.start();


ignite.tracingConfiguration().set(
    new TracingConfigurationCoordinates.Builder(Scope.TX).build(),
    new TracingConfigurationParameters.Builder().withSamplingRate(1).build());
```

Параметр --scope указывает API, который необходимо отслеживать. Для отслеживания используются следующие API:

• DISCOVERY — события обнаружения;

• EXCHANGE — события обмена;

• ОБЩЕНИЕ — коммуникационные события;

• TX — транзакции;

• SQL — SQL-запросы.

--sampling-rate — это вероятностная частота выборки, число от 0 до 1:

• 0 означает отсутствие выборки,

• 1 означает выборка всегда.

10.3.3 Экспорт трассировок

Для просмотра трассировок необходимо экспортировать их во внешнюю систему с помощью одного из доступных экспортеров. OpenCensus поддерживает ряд готовых экспортеров, и можно написать собственный.

В этом разделе представлено, как экспортировать трассировки в Zipkin.

1. Чтобы запустить Zipkin на компьютере, нужно следовать руководству https://zipkin.io/pages/quickstart.html.

2. Зарегистрировать ZipkinTraceExporter в приложении, где запускается РЕД КВАНТ:

```
//register Zipkin exporter

ZipkinTraceExporter.createAndRegister(
    ZipkinExporterConfiguration.builder().setV2Url("http://localhost:9411/api/v2/spans")
        .setServiceName("ignite-cluster").build());
```

```
IgniteConfiguration cfg = new IgniteConfiguration().setClientMode(true)
      .setTracingSpi(new org.apache.ignite.spi.tracing.opencensus.OpenCensusTracingSpi());

Ignite ignite = Ignition.start(cfg);

//enable trace sampling for transactions with 100% sampling rate
ignite.tracingConfiguration().set(
      new TracingConfigurationCoordinates.Builder(Scope.TX).build(),
      new TracingConfigurationParameters.Builder().withSamplingRate(1).build());

//create a transactional cache
IgniteCache<Integer, String> cache = ignite
      .getOrCreateCache(new CacheConfiguration<Integer, String>("myCache")
            .setAtomicityMode(CacheAtomicityMode.TRANSACTIONAL));

IgniteTransactions transactions = ignite.transactions();

// start a transaction
try (Transaction tx = transactions.txStart()) {
   //do some operations
   cache.put(1, "test value");

   System.out.println(cache.get(1));

   cache.put(1, "second value");

   tx.commit();
}

try {
   //This code here is to wait until the trace is exported to Zipkin.
   //If your application doesn't stop here, you don't need this piece of code.
   Thread.sleep(5_000);
} catch (InterruptedException e) {
   e.printStackTrace();
}
```

3. Открыть http://localhost:9411/zipkin в браузере и нажать на значок поиска.

На рисунке 44 представлено, как выглядит трассировка транзакции.

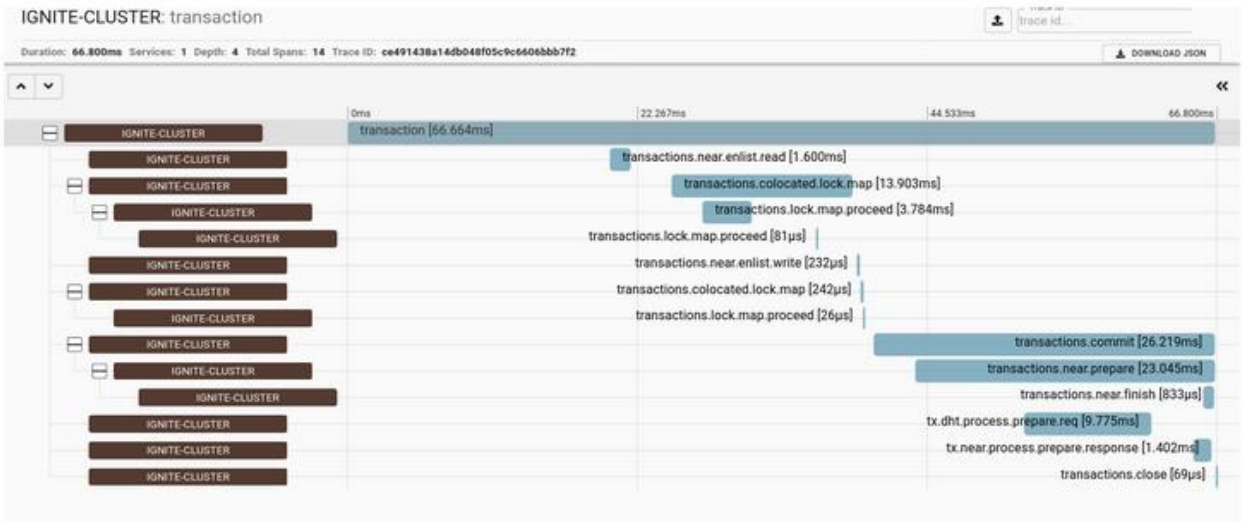

Рисунок 44 - Трассировка транзакции

10.3.4 Анализ данных трассировки

Трассировка — это записанная информация о выполнении определенного события. Каждая трассировка состоит из дерева отрезков (spans). Промежуток — это отдельная единица работы, выполняемая системой для обработки события.

Из-за распределенного характера РЕД КВАНТ операция обычно включает несколько узлов. Таким образом, трассировка может включать в себя промежутки от нескольких узлов. Каждый промежуток всегда содержит информацию об узле, на котором была выполнена соответствующая операция.

На представленном выше изображении трассировки транзакций видно, что трассировка содержит промежутки, связанные со следующими операциями:

- acquire locks (transactions.colocated.lock.map),
- get (transactions.near.enlist.read),
- put (transactions.near.enlist.write),
- commit (transactions.commit) и
- close (transactions.close).

Операция завершения транзакции, в свою очередь, состоит из двух операций: подготовки и завершения.

Можно нажать на каждый диапазон, чтобы просмотреть прикрепленные к нему аннотации и теги (рисунок 45).

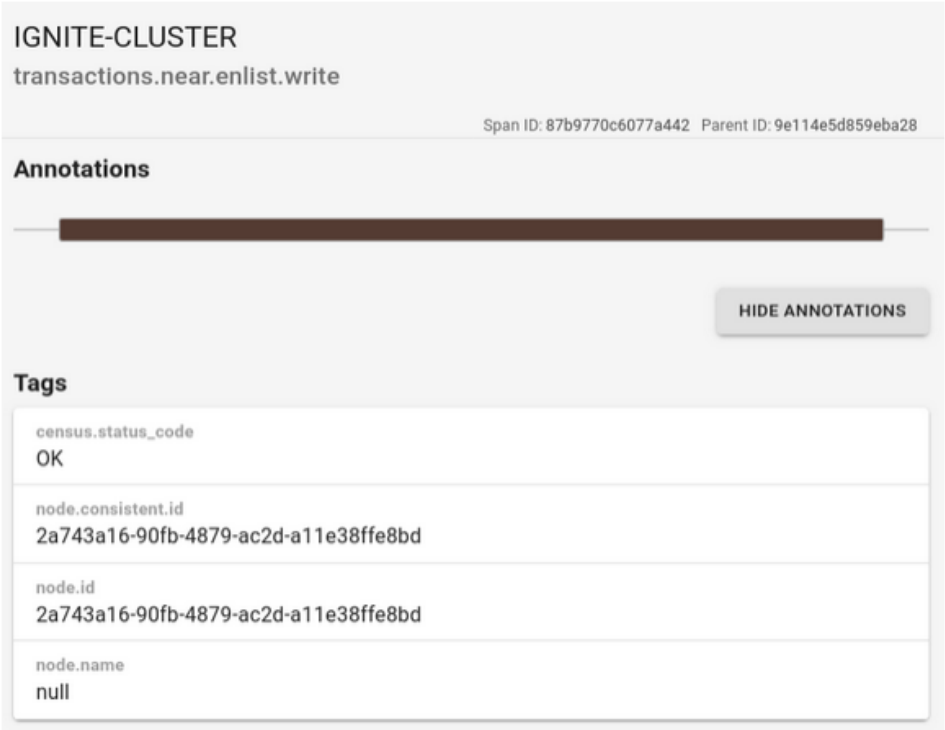

Рисунок 45 - Просмотр данных диапазона трассировки

10.3.5 Отслеживание SQL-запросов

Чтобы включить трассировку SQL-запросов, нужно использовать SQL в качестве значения параметра области (scope) во время настройки выборки трассировки. Если трассировка SQL-запросов включена, выполнение каждого SQL-запроса на любом узле кластера создаст отдельную трассировку.

Внимание! Включение трассировки для SQL-запросов серьезно снижает производительность движка SQL.

В таблице 30 приведены описания, список тегов и аннотации для каждого промежутка, который может быть частью дерева трассировки SQL-запроса.

Примечание: В зависимости от типа SQL-запроса и плана его выполнения некоторые промежутки могут отсутствовать в дереве промежутков SQL-запросов.

Таблица 30 - Виды диапазонов

Имя диапазона	Описание	Теги и аннотации
sql.query	Выполнение SQL-запроса с момента регистрации до освобождения используемых ресурсов на узле-инициаторе запроса	• sql.query.text — текст SQL-запроса • sql.schema — схема SQL
sql.cursor.open	Открытие курсора SQL-запроса	
sql.cursor.close	Закрытие курсора SQL-запроса	
sql.cursor.cancel	Отмена курсора запроса SQL	

Имя диапазона	Описание	Теги и аннотации
sql.query.parse	Разбор SQL-запроса	sql.parser.cache.hit — был ли пропущен синтаксический анализ SQL-запроса из-за кэшированного результата.
sql.query.execute.request	Обработка запроса на выполнение SQL-запроса	sql.query.text — текст SQL-запроса
sql.next.page.request	Обработка запроса на получение следующей страницы результата выполнения локального SQL-запроса	
sql.page.response	Обработка сообщения со страницей результатов выполнения локального SQL-запроса узла	
sql.query.execute	Выполнение запроса движком H2 SQL	sql.query.text — текст SQL-запроса
sql.page.prepare	Чтение строк из курсора и подготовка страницы результатов	sql.page.rows — количество строк, содержащихся на странице результатов.
sql.fail.response	Обработка сообщения, указывающего на сбой выполнения SQL-запроса	
sql.dml.query.execute.request	Обработка запроса на выполнение запроса SQL DML	sql.query.text — текст SQL-запроса
sql.dml.query.response	Обработка результата выполнения запроса SQL DML узлом-инициатором запроса	
sql.query.cancel.request	Обработка запроса на отмену SQL-запроса	
sql.iterator.open	Открытие итератора SQL-запроса	
sql.iterator.close	Закрытие итератора SQL-запроса	
sql.page.fetch	Получение страницы результатов SQL-запроса	sql.page.rows — количество строк, содержащихся на странице результатов.
sql.page.wait	Ожидание получения страницы результатов SQL-запроса от удаленного узла	
sql.index.range.request	Обработка запроса диапазона SQL индексов	• sql.index — имя индекса SQL • sql.table - имя таблицы SQL • sql.index.range.rows — количество строк,

Имя диапазона	Описание	Теги и аннотации
		содержащихся в результате запроса диапазона индексов.
sql.index.range.response	Обработка ответа диапазона SQL индексов	
sql.dml.query.execute	Выполнение запроса SQL DML	
sql.command.query.execute	Выполнение запроса команды SQL, который является либо DDL запросом, либо собственной командой РЕД КВАНТ.	
sql.partitions.reserve	Резервирование разделов данных, используемых для выполнения запроса	Аннотационное сообщение, указывающее на резервирование разделов данных для определенного кэша — «Разделы кэша были зарезервированы [cache=<имя кэша>, partitions=[<номера разделов>]»
sql.cache.update	Обновление кэша в результате выполнения запроса SQL DML	sql.cache.updates — количество записей кэша, которые будут обновлены в результате запроса DML
sql.batch.process	Обработка пакетного обновления SQL	

11 ДОСТУП И БЕЗОПАСНОСТЬ

11.1 Аутентификация

Включить аутентификацию РЕД КВАНТ можно, установив для свойства authenticationEnabled значение true в конфигурации узла. Этот тип аутентификации требует, чтобы постоянное хранилище было включено по крайней мере для одной области данных.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  <property name="dataStorageConfiguration">
    <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
      <property name="defaultDataRegionConfiguration">
        <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
          <property name="persistenceEnabled" value="true"/>
        </bean>
      </property>
    </bean>
  </property>

  <property name="authenticationEnabled" value="true"/>

</bean>
```

На узле, который запускается первым, должна быть включена аутентификация. При запуске РЕД КВАНТ создает учетную запись пользователя с именем «ignite» и паролем «ignite». Эта учетная запись предназначена для создания других учетных записей пользователей для любых нужд. Затем нужно просто удалить учетную запись «ignite».

Управлять пользователями можно с помощью следующих команд SQL:

- CREATE USER — создать пользователя (подробнее см. в **Приложение В. DDL**);
- ALTER USER — изменить пользователя (подробнее см. в **Приложение В. DDL**);
- DROP USER — удалить пользователя (подробнее см. в **Приложение В. DDL**).

11.2 SSL/TLS

Для обеспечения достаточного уровня безопасности рекомендуется, чтобы каждый узел (сервер или клиент) имел свой уникальный сертификат в хранилище ключей узла (включая закрытый ключ). Этому сертификату должны доверять все остальные серверные узлы.

11.2.1 SSL/TLS для узлов

Чтобы включить SSL/TLS для узлов кластера, нужно настроить фабрику SSLContext в конфигурации узла. Можно использовать org.apache.ignite.ssl.SslContextFactory — фабрику по умолчанию, использующую настраиваемое хранилище ключей для инициализации контекста SSL.

Внимание! Необходимо убедиться, что версия JVM устраняет следующую проблему (https://bugs.openjdk.org/browse/JDK-8219658), которая может вызывать взаимоблокировки в соединениях SSL. Если в текущей версии JVM проблема не решена, и нет возможности

обновить ее, нужно установить для параметра TcpDiscoverySpi.soLinger неотрицательное значение.

Ниже приведен пример конфигурации SslContextFactory:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <property name="sslContextFactory">
    <bean class="org.apache.ignite.ssl.SslContextFactory">
      <property name="keyStoreFilePath" value="keystore/node.jks"/>
      <property name="keyStorePassword" value="123456"/>
      <property name="trustStoreFilePath" value="keystore/trust.jks"/>
      <property name="trustStorePassword" value="123456"/>
      <property name="protocol" value="TLSv1.3"/>
    </bean>
  </property>

</bean>
```

Хранилище ключей должно содержать сертификат узла, включая его закрытый ключ. Хранилище доверенных сертификатов должно содержать доверенные сертификаты для всех остальных узлов кластера.

Можно определить другие свойства, такие как алгоритм ключей, тип хранилища ключей или менеджера доверия. См. описание свойств в разделе Свойства SslContextFactory.

После запуска узла в логах можно увидеть следующие сообщения:

```
Security status [authentication=off, tls/ssl=on]
```

11.2.2 SSL/TLS для тонких клиентов и JDBC/ODBC

РЕД КВАНТ использует одни и те же свойства SSL/TLS для всех клиентов, включая тонкие клиенты и соединения JDBC/ODBC. Свойства настраиваются в конфигурации клиентского коннектора. Конфигурация клиентского коннектора определяется с помощью свойства IgniteConfiguration.clientConnectorConfiguration.

Чтобы включить SSL/TLS для клиентских подключений, необходимо задать для свойства sslEnabled значение true и указать SslContextFactory в конфигурации клиентского коннектора. Можно повторно использовать SSLContextFactory, настроенную для узлов, или настроить фабрику SSLContext, которая будет использоваться только для клиентских подключений.

Затем аналогичным образом необходимо настроить SSL на стороне клиента. Для получения подробной информации необходимо обратиться к документации конкретного клиента.

Пример конфигурации, которая устанавливает SslContextFactory для подключения клиента:

```
<property name="clientConnectorConfiguration">
  <bean class="org.apache.ignite.configuration.ClientConnectorConfiguration">
    <property name="sslEnabled" value="true"/>
    <property name="useIgniteSslContextFactory" value="false"/>
    <property name="sslContextFactory">
      <bean class="org.apache.ignite.ssl.SslContextFactory">
```

```
            <property name="keyStoreFilePath" value="/path/to/server.jks"/>
            <property name="keyStorePassword" value="123456"/>
            <property name="trustStoreFilePath" value="/path/to/trust.jks"/>
            <property name="trustStorePassword" value="123456"/>
          </bean>
        </property>
      </bean>
    </property>
```

Если необходимо повторно использовать фабрику SSLContext, настроенную для узлов, нужно только установить для свойства sslEnabled значение true, и ClientConnectorConfiguration будет искать SSLContext, настроенный в IgniteConfiguration:

```
<property name="clientConnectorConfiguration">
  <bean class="org.apache.ignite.configuration.ClientConnectorConfiguration">
    <property name="sslEnabled" value="true"/>
  </bean>
</property>
```

11.2.3 Отключение проверки сертификата

В некоторых случаях полезно отключить проверку сертификата, например, при подключении к серверу с самоподписанным сертификатом. Этого можно добиться с помощью отключенного менеджера доверия, которого можно получить, вызвав метод SslContextFactory.getDisabledTrustManager().

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <property name="sslContextFactory">
    <bean class="org.apache.ignite.ssl.SslContextFactory">
      <property name="keyStoreFilePath" value="keystore/node.jks"/>
      <property name="keyStorePassword" value="123456"/>
      <property name="trustManagers">
        <bean class="org.apache.ignite.ssl.SslContextFactory" factory-method="getDisabledTrustManager"/>
      </property>
    </bean>
  </property>

</bean>
```

11.2.4 Обновление сертификатов

Если срок действия SSL-сертификатов скоро истечет или они были скомпрометированы, можно установить новые сертификаты, не останавливая весь кластер.

Процедура обновления сертификата описана ниже.

1. Прежде всего, нужно убедиться, что новым сертификатам доверяют все узлы кластера. Этот шаг может не понадобиться, если доверенные хранилища содержат корневой сертификат, а новые сертификаты подписаны одним и тем же корневым сертификатом.

Для узлов, где сертификат не является доверенным, необходимо повторить следующую процедуру:

a) Импортировать новый сертификат в доверенное хранилище узла.

b) Перезапустить узел.

c) Повторить эти шаги для всех серверных узлов.

Теперь все узлы доверяют новым сертификатам.

2. Необходимо импортировать новый сертификат (включая закрытый ключ) в хранилище ключей соответствующего узла и удалить старый сертификат. Затем перезапустить узел. Повторить эту процедуру для всех сертификатов, которые необходимо обновить.

11.2.5 Свойства SslContextFactory

SslContextFactory поддерживает свойства, представленные в таблице 31.

Таблица 31 - Свойства SslContextFactory

Свойство	Описание	Значение по умолчанию
keyAlgorithm	Алгоритм управления ключами, который будет использоваться для создания менеджера ключей.	SunX509
keyStoreFilePath	Путь к файлу хранилища ключей. Это обязательный параметр, так как контекст SSL не может быть инициализирован без диспетчера ключей.	N/A
keyStorePassword	Пароль хранилища ключей.	N/A
keyStoreType	Тип хранилища ключей.	JKS
protocol	Протокол для безопасной передачи.	TLS
trustStoreFilePath	Путь к файлу хранилища доверия.	N/A
trustStorePassword	Пароль хранилища доверия.	N/A
trustStoreType	Тип доверенного хранилища.	JKS
trustManagers	Список предварительно настроенных менеджеров доверия.	N/A

11.3 Сквозное шифрование

Сквозное шифрование данных (TDE) позволяет пользователям шифровать свои данные.

Когда включена персистентность РЕД КВАНТ, шифрование может быть включено для каждого кэша/таблицы, и в этом случае будут зашифрованы следующие данные:

- Данные на диске;
- WAL записи.

Если включить шифрование кэша/таблицы, РЕД КВАНТ сгенерирует ключ (называемый ключом шифрования кэша) и будет использовать этот ключ для шифрования/дешифрования данных в кэше. Ключ шифрования кэша хранится в системном кэше, и пользователи не могут получить к нему доступ. Когда ключ шифрования кэша отправляется на другие узлы или

сохраняется на диск (когда узел выходит из строя), он шифруется с помощью мастер-ключа (master key). Мастер-ключ должен быть указан пользователем в конфигурации.

Один и тот же мастер-ключ должен быть указан через конфигурацию на каждом серверном узле. Один из способов убедиться, что используется один и тот же ключ, — скопировать файл JKS с одного узла на другие узлы. Если попытаться включить сквозное шифрование, используя разные ключи, узлы с другим ключом не смогут присоединиться к кластеру (коннект будет отклонен из-за различий в справочниках).

РЕД КВАНТ использует предоставленные JDK алгоритмы шифрования: «AES/CBC/PKCS5Padding» для шифрования записей WAL и «AES/CBC/NoPadding» для шифрования страниц данных на диске.

Сквозное шифрование данных имеет некоторые ограничения, о которых следует знать, прежде чем развертывать его в производственной среде.

1. Шифрование. Нет возможности зашифровать/расшифровать существующие кэши/таблицы.

2. Снимки и восстановление. Шифрование, изменение главного ключа или ключа группы кэша во время операций создания снимков не допускается.

Чтобы включить шифрование в кластере, нужно указать мастер-ключ в конфигурации каждого серверного узла. Пример конфигурации показан ниже.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <!-- We need to configure EncryptionSpi to enable encryption feature. -->
  <property name="encryptionSpi">
    <!-- Using EncryptionSpi implementation based on java keystore. -->
    <bean class="org.apache.ignite.spi.encryption.keystore.KeystoreEncryptionSpi">
      <!-- Path to the keystore file. -->
      <property name="keyStorePath" value="ignite_keystore.jks"/>
      <!-- Password for keystore file. -->
      <property name="keyStorePassword" value="mypassw0rd"/>
      <!-- Name of the key in keystore to be used as a master key. -->
      <property name="masterKeyName" value="ignite.master.key"/>
      <!-- Size of the cache encryption keys in bits. Can be 128, 192, or 256 bits.-->
      <property name="keySize" value="256"/>
    </bean>
  </property>
  <property name="cacheConfiguration">
    <bean class="org.apache.ignite.configuration.CacheConfiguration">
      <property name="name" value="encrypted-cache"/>
      <property name="encryptionEnabled" value="true"/>
    </bean>
  </property>
  <property name="dataStorageConfiguration">
    <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
      <property name="encryptionConfiguration">
        <bean class="org.apache.ignite.configuration.EncryptionConfiguration">
          <!-- Set re-encryption rate limit to 10.3 MB/s. -->
          <property name="reencryptionRateLimit" value="10.3"/>
        </bean>
      </property>
    </bean>
```

```
  </property>
</bean>
```

Когда мастер-ключ настроен, можно включить шифрование кэша следующим образом:

```
<property name="cacheConfiguration">

  <bean class="org.apache.ignite.configuration.CacheConfiguration">
    <property name="name" value="encrypted-cache"/>
    <property name="encryptionEnabled" value="true"/>
  </bean>
</property>
```

11.3.1 Пример генерации мастер-ключа

Хранилище ключей с мастер-ключом можно создать с помощью keytool следующим образом:

```
user:~/tmp:[]$ java -version

java version "1.8.0_161"
Java(TM) SE Runtime Environment (build 1.8.0_161-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.161-b12, mixed mode)

user:~/tmp:[]$ keytool -genseckey \
-alias ignite.master.key \
-keystore ./ignite_keystore.jks \
-storetype PKCS12 \
-keyalg aes \
-storepass mypassw0rd \
-keysize 256

user:~/tmp:[]$ keytool \
-storepass mypassw0rd \
-storetype PKCS12 \
-keystore ./ignite_keystore.jks \
-list

Keystore type: PKCS12
Keystore provider: SunJSSE

Your keystore contains 1 entry

ignite.master.key, 07.11.2018, SecretKeyEntry,
```

11.4 Песочница

РЕД КВАНТ позволяет выполнять пользовательскую логику через различные API, включая вычислительные задачи, фильтры событий, слушателей сообщений. Определяемая пользователем логика может использовать API-интерфейсы Java для получения доступа к ресурсам хоста. Например, он может создавать/обновлять/удалять файлы или системные свойства, открывать сетевые подключения, использовать Java Reflection и другие API, чтобы получить полный контроль над средой хоста. РЕД КВАНТ Sandbox основан на модели Java Sandbox и позволяет ограничить область определяемой пользователем логики, выполняемой через РЕД КВАНТ API.

11.4.1 Активация песочницы

Активация РЕД КВАНТ Sandbox включает настройку экземпляра SecurityManager и создание реализации GridSecurityProcessor.

11.4.1.1 Установка SecurityManager

В связи с тем, что РЕД КВАНТ Sandbox основан на модели Java Sandbox, а SecurityManager является важной частью этой модели, необходимо установить его. SecurityManager отвечает за проверку того, какая политика безопасности действует в данный момент. Он также выполняет проверки контроля доступа. Менеджер безопасности не устанавливается автоматически при запуске приложения. Если РЕД КВАНТ запускается как отдельное приложение, необходимо вызвать виртуальную машину Java с аргументом командной строки -Djava.security.manager (который устанавливает значение свойства java.security.manager). Существует также аргумент командной строки -Djava.security.policy, определяющий, какие файлы политик используются. Если не указать -Djava.security.policy в командной строке, то будут использоваться файлы политик, указанные в файле свойств безопасности.

Примечание: Может оказаться удобным добавление менеджера безопасности и аргументов командной строки политики в скрипт {IGNITE-HOME}/bin/ignite.sh|ignite.bat.

Примечание: У РЕД КВАНТ должно быть достаточно прав для корректной работы. Можно применить самый простой способ, предоставляющий РЕД КВАНТ разрешение java.security.AllPermission, но нужно помнить о принципе безопасности «предоставлять минимальное возможное количество разрешений».

11.4.1.2 Обеспечение реализации GridSecurityProcessor

В настоящее время РЕД КВАНТ не предоставляет готовую реализацию интерфейса GridSecurityProcessor. Но можно реализовать этот интерфейс как часть пользовательского плагина.

Интерфейс GridSecurityProcessor имеет метод sandboxEnabled, который управляет выполнением пользовательского кода внутри песочницы РЕД КВАНТ. По умолчанию этот метод возвращает false, что означает отсутствие песочницы. Если необходимо использовать РЕД КВАНТ Sandbox, переопределенный метод sandboxEnabled должен возвращать значение true.

Если РЕД КВАНТ Sandbox включена, можно увидеть следующую строку трассировки:

```
[INFO] Security status [authentication=on, sandbox=on, tls/ssl=off]
```

11.4.2 Разрешения

Пользовательский код всегда выполняется от имени субъекта безопасности, который инициирует его выполнение. Разрешения песочницы субъекта безопасности определяют действия, которые может выполнять пользовательский код. Песочница РЕД КВАНТ получает эти разрешения с помощью метода SecuritySubject#sandboxPermissions.

Примечание: При работе в песочнице РЕД КВАНТ пользовательский код может использовать общедоступный API РЕД КВАНТ без предоставления каких-либо дополнительных разрешений.

Если субъект безопасности не имеет достаточных разрешений для выполнения чувствительной к безопасности операции, появляется исключение AcccessControlException.

```
// Get compute instance over all nodes in the cluster.

IgniteCompute compute = Ignition.ignite().compute();

compute.broadcast(() -> {
  // If the Ignite Sandbox is turned on, the lambda code is executed with restrictions.

  // You can use the public API of Ignite without granting any permissions.
  Ignition.localIgnite().cache("some.cache").get("key");

  // If the current security subject doesn't have the java.util.PropertyPermission("secret.property", "read")
permission,
  // a java.security.AccessControlException appears here.
  System.getProperty("secret.property");
});
```

В случае доступа к системному свойству, показанному в приведенном выше фрагменте, можно увидеть следующую строку трассировки с исключением:

```
java.security.AccessControlException: access denied ("java.util.PropertyPermission" "secret.property" "read")
```

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

№ версии	*Дата изменения*	*Автор*	*Состав изменений*
1.0	13.03.2023	Антонова Ю.В.	Первоначальная версия документа.

12 ПРИЛОЖЕНИЕ А. БАЗОВЫЕ ОПЕРАЦИИ С КЭШЕМ

1. Получение экземпляра кэша.

Все операции с кэшем выполняются через экземпляр IgniteCache. Можно получить IgniteCache для существующего кэша или создать кэш динамически.

```
Ignite ignite = Ignition.ignite();

// Obtain an instance of the cache named "myCache".
// Note that different caches may have different generics.
IgniteCache<Integer, String> cache = ignite.cache("myCache");
```

2. Динамическое создание кэшей.

Также можно создавать кэш динамически:

```
Ignite ignite = Ignition.ignite();


CacheConfiguration<Integer, String> cfg = new CacheConfiguration<>();

cfg.setName("myNewCache");
cfg.setAtomicityMode(CacheAtomicityMode.TRANSACTIONAL);

// Create a cache with the given name if it does not exist.
IgniteCache<Integer, String> cache = ignite.getOrCreateCache(cfg);
```

Создающие кэш методы выдают исключение org.apache.ignite.IgniteCheckedException при вызове во время изменения базовой топологии.

```
javax.cache.CacheException: class org.apache.ignite.IgniteCheckedException: Failed to start/stop cache, cluster state change is in progress.

        at org.apache.ignite.internal.processors.cache.GridCacheUtils.convertToCacheException(GridCacheUtils.java:1323)
        at org.apache.ignite.internal.IgniteKernal.createCache(IgniteKernal.java:3001)
        at org.apache.ignite.internal.processors.platform.client.cache.ClientCacheCreateWithNameRequest.process(ClientCacheCreateWithNameRequest.java:48)
```

3. Уничтожение кэшей

Чтобы удалить кэш со всех узлов кластера, необходимо вызвать метод destroy().

```
Ignite ignite = Ignition.ignite();

IgniteCache<Long, String> cache = ignite.cache("myCache");

cache.destroy();
```

4. Атомарные операции

Как только получен экземпляр кэша, можно начать выполнять с ним операции get/put.

```
IgniteCache<Integer, String> cache = ignite.cache("myCache");
```

```
// Store keys in the cache (the values will end up on different cache nodes).
for (int i = 0; i < 10; i++)
   cache.put(i, Integer.toString(i));

for (int i = 0; i < 10; i++)
   System.out.println("Got [key=" + i + ", val=" + cache.get(i) + ']');
```

Примечание: Массовые операции, такие как putAll() или removeAll(), выполняются как последовательность атомарных операций и могут частично завершаться ошибкой. В этом случае создается исключение CachePartialUpdateException, содержащее список ключей, для которых не удалось выполнить обновление.

Чтобы обновить набор записей в рамках одной операции, нужно рассмотреть возможность использования транзакций.

Ниже приведены дополнительные примеры основных атомарных операций:

```
// Put-if-absent which returns previous value.

String oldVal = cache.getAndPutIfAbsent(11, "Hello");

// Put-if-absent which returns boolean success flag.
boolean success = cache.putIfAbsent(22, "World");

// Replace-if-exists operation (opposite of getAndPutIfAbsent), returns previous
// value.
oldVal = cache.getAndReplace(11, "New value");

// Replace-if-exists operation (opposite of putIfAbsent), returns boolean
// success flag.
success = cache.replace(22, "Other new value");

// Replace-if-matches operation.
success = cache.replace(22, "Other new value", "Yet-another-new-value");

// Remove-if-matches operation.
success = cache.remove(11, "Hello");
```

5. Асинхронное выполнение

У большинства операций кэширования есть асинхронные аналоги, имена которых имеют суффикс «асинхронный».

```
// a synchronous get

V get(K key);

// an asynchronous get
IgniteFuture<V> getAsync(K key);
```

Асинхронные операции возвращают объект, представляющий результат операции. Можно дождаться завершения операции блокирующим или неблокирующим способом.

Чтобы дождаться результатов неблокирующим образом, нужно зарегистрировать обратный вызов с помощью метода IgniteFuture.listen() или IgniteFuture.chain(). Обратный вызов вызывается, когда операция завершена.

```
IgniteCompute compute = ignite.compute();


// Execute a closure asynchronously.
IgniteFuture<String> fut = compute.callAsync(() -> "Hello World");

// Listen for completion and print out the result.
fut.listen(f -> System.out.println("Job result: " + f.get()));
```

Примечание: Выполнение обратных вызовов и пулы потоков

Если асинхронная операция завершается к моменту передачи обратного вызова методу IgniteFuture.listen() или IgniteFuture.chain(), то обратный вызов выполняется вызывающим потоком синхронно. В противном случае обратный вызов выполняется асинхронно после завершения операции.

Обратные вызовы для асинхронных вычислительных операций вызываются потоками из общедоступного пула РЕД КВАНТ. Вызов синхронного кэширования и вычислительных операций из обратного вызова может привести к взаимоблокировке из-за опустошения пулов. Чтобы добиться вложенного выполнения асинхронных вычислительных операций, можно воспользоваться преимуществами настраиваемых пулов потоков.

Обратные вызовы для асинхронных кэш-операций операций вызываются с помощью ForkJoinPool#commonPool, если другой исполнитель не настроен с помощью IgniteConfiguration#asyncContinuationExecutor.

- Этот исполнитель по умолчанию безопасен для любых операций внутри обратного вызова.
- Поведение по умолчанию было изменено в РЕД КВАНТ 2.11. До этого обратные вызовы операций асинхронного кэша вызывались из системного пула РЕД КВАНТ (т.н. «полосатый пул» (striped pool)).
- Чтобы восстановить предыдущее поведение, нужно использовать IgniteConfiguration.setAsyncContinuationExecutor(Runnable::run).
    - Предыдущее поведение может обеспечить небольшое улучшение производительности, поскольку обратные вызовы выполняются без каких-либо косвенных указаний или планирования.
    - НЕБЕЗОПАСНО: операции с кэшем не могут выполняться, пока системные потоки выполняют обратные вызовы, и возможны взаимоблокировки, если из обратного вызова вызываются другие операции с кэшем.

13 ПРИЛОЖЕНИЕ Б. СТАНДАРТ SQL. ТИПЫ ДАННЫХ

Стандарт SQL

РЕД КВАНТ по умолчанию поддерживает большинство основных функций стандарта ANSI-99. В таблице 32 показано соответствие РЕД КВАНТ стандарту SQL:1999 (Core).

Таблица 32 - Стандарты, поддерживаемые SQL

Идентификатор функции, имя	Поддерживаемые функции
E011 Числовые типы данных	РЕД КВАНТ полностью поддерживает следующие вспомогательные функции: • E011–01 Типы данных INTEGER и SMALLINT (включая все варианты написания); • E011–02 Типы данных REAL, DOUBLE PRECISON и FLOAT; • E011–05 Числовое сравнение; • E011–06 Неявное приведение числовых типов данных. РЕД КВАНТ обеспечивает частичную поддержку следующих вспомогательных функций: • E011–03 Типы данных DECIMAL и NUMERIC. Фиксированный <масштаб> не поддерживается для DEC и NUMERIC, поэтому есть нарушения для: ◦ 7) Если <масштаб> опущен, то <масштаб> 0 (ноль) является неявным (6.1 <тип данных>); ◦ 22) NUMERIC определяет точный числовой тип данных с десятичной точностью и масштабом, указанными параметрами <precision> и <scale>; ◦ 23) DECIMAL задает точный числовой тип данных с десятичной шкалой, заданной <scale>, и определяемой реализацией десятичной точностью, равной или превышающей значение указанного <precision>. • E011–04 Арифметический оператор. См. проблему для функции E011–03.
E021 Типы символьных строк	РЕД КВАНТ полностью поддерживает следующие вспомогательные функции: • E021–03 Символьные литералы; • E021–04 Функция CHARACTER_LENGTH; • E021–05 Функция OCTET_LENGTH; • E021–06 Функция SUBSTRING; • E021–07 Объединение (конкатенация) символов; • E021–08 Функции UPPER и LOWER; • E021–09 Функция TRIM; • E021–10 Неявное приведение типов символьных строк фиксированной и переменной длины; • E021–11 Функция POSITION; • E021–12 Сравнение символов. РЕД КВАНТ обеспечивает частичную поддержку следующих подфункций:

Идентификатор функции, имя	Поддерживаемые функции
	• E021–01 Тип данных CHARACTER (включая все варианты написания). <character string type> ::= CHARACTER [<left paren> <length> <right paren>] | CHAR [<left paren> <length> <right paren>] | CHARACTER VARYING <left paren> <length> <right paren> | CHAR VARYING <left paren> <length> <right paren> | VARCHAR <left paren> <length> <right paren> <length> не поддерживается для типов данных CHARACTER и CHARACTER VARYING. • E021–02 Тип данных CHARACTER VARYING (включая все варианты написания). См. проблему для функции E021–01.
E031 Идентификаторы	РЕД КВАНТ полностью поддерживает следующие вспомогательные функции: • E031–01 Идентификаторы с разделителями; • E031–02 Идентификаторы в нижнем регистре; • E031–03 Подчеркивание в конце.
E051 Базовая спецификация запроса	РЕД КВАНТ полностью поддерживает следующие вспомогательные функции: • E051–01 SELECT DISTINCT; • E051–04 GROUP BY может содержать столбцы, не входящие в <select-list> ; • E051–05 Выбор элементов списка, которые можно переименовать; • E051–06 Условие HAVING; • E051–07 Дополнение * в списке выборки; • E051–08 Корреляционные имена в предложении FROM. РЕД КВАНТ не поддерживает следующие вспомогательные функции: • E051–02 Условие GROUP BY; Нет поддержки ROLLUP, CUBE, GROUPING SETS. • E051–09 Переименование столбцов в предложении FROM.
E061 Основные предикаты и условия поиска	РЕД КВАНТ полностью поддерживает следующие вспомогательные функции: • E061–01 Предикат сравнения; • E061–02 Предикат BETWEEN; • E061–03 Предикат IN со списком значений; • E061–06 Предикат NULL; • E061–08 Предикат EXISTS; • E061–09 Подзапросы в предикате сравнения; • E061–11 Подзапросы в предикате IN; • E061–13 Коррелированные подзапросы; • E061–14 Условия поиска. РЕД КВАНТ обеспечивает частичную поддержку следующих вспомогательных функций: • E061–04 Предикат LIKE; Существует поддержка <символьного предиката>, но <предикат типа октета> не может быть проверен из-за этой проблемы.

Идентификатор функции, имя	Поддерживаемые функции
	• E061–05 Предикат LIKE: предложение ESCAPE; Существует поддержка <предиката типа символа>, но <предикат типа октета> не может быть проверен из-за этой проблемы. • E061–07 Предикат количественного сравнения; Кроме ALL. РЕД КВАНТ не поддерживает следующую вспомогательную функцию: • E061–12 Подзапросы в предикате количественного сравнения.
E071 Основные выражения запроса	РЕД КВАНТ обеспечивает частичную поддержку следующих вспомогательных функций: • E071–01 Оператор таблицы UNION DISTINCT; • E071–02 Оператор таблицы UNION ALL; • E071–03 Оператор таблицы EXCEPT DISTINCT; • E071–05 Столбцы, объединенные с помощью табличных операторов, не обязательно должны иметь точно такой же тип данных; • E071–06 Табличные операторы в подзапросах. Следует отметить, что нерекурсивное условие WITH в H2 и РЕД КВАНТ не поддерживается. Согласно документам H2, рекурсивное условие WITH поддерживается, но в РЕД КВАНТ оно не работает.
E081 Основные привилегии	РЕД КВАНТ не поддерживает следующие вспомогательные функции: • E081–01 Привилегия SELECT на уровне таблицы; • E081–02 Привилегия DELETE; • E081–03 Привилегия INSERT на уровне таблицы; • E081–04 Привилегия UPDATE на уровне таблицы; • E081–05 Привилегия UPDATE на уровне столбца; • E081–06 Привилегия REFERENCES в таблице; • E081–07 Привилегия REFERENCES в столбце; • E081–08 WITH GRANT OPTION; • E081–09 Привилегия USAGE; • E081–10 Привилегия EXECUTE.
E091 Набор функций	РЕД КВАНТ обеспечивает частичную поддержку следующих вспомогательных функций: • E091–01 AVG; • E091–02 COUNT; • E091–03 MAX; • E091–04 MIN; • E091–05 SUM; • E091–06 Дополнение ALL; • E091–07 Дополнение DISTINCT. Следует отметить, что нет поддержки: • GROUPING и ANY (как в H2, так и в РЕД КВАНТ). • функции EVERY и SOME. Есть поддержка в H2, но не работает в РЕД КВАНТ.
E101 Основные операции с данными	РЕД КВАНТ полностью поддерживает следующие вспомогательные функции: • E101–03 Оператор UPDATE с критерием отбора;

Идентификатор функции, имя	Поддерживаемые функции
	• E101–04 Оператор DELETE с критерием отбора. РЕД КВАНТ обеспечивает частичную поддержку следующей вспомогательной функции: • E101–01 Оператор INSERT. Нет поддержки значений DEFAULT в РЕД КВАНТ. Работает в H2.
E111 Однострочный SELECT	РЕД КВАНТ не поддерживает эту функцию.
E121 Базовая поддержка курсора	РЕД КВАНТ не поддерживает следующие вспомогательные функции: • E121–01 DECLARE CURSOR; • E121–02 Столбцы ORDER BY не обязательно должны быть в списке выбора; • E121–03 Выражения значений в предложении ORDER BY; • E121–04 Оператор OPEN; • E121–06 Оператор UPDATE с позиционированием; • E121–07 Оператор DELETE с позиционированием; • E121–08 Оператор CLOSE; • E121–10 Оператор FETCH: неявный NEXT; • E121–17 Указатель WITH HOLD.
E131 Поддержка NULL (NULL вместо значений)	РЕД КВАНТ полностью поддерживает эту функцию.
E141 Основные ограничения целостности	РЕД КВАНТ полностью поддерживает следующую вспомогательную функцию: • E141–01 Ограничения NOT NULL YES. РЕД КВАНТ обеспечивает частичную поддержку следующих вспомогательных функций: • E141–03 Ограничения PRIMARY KEY; • E141–08 NOT NULL распространяется на PRIMARY KEY. РЕД КВАНТ не поддерживает следующие вспомогательные функции: • E141–02 Ограничения UNIQUE для столбцов NOT NULL; • E141–04 Базовое ограничение FOREIGN KEY со значением NO ACTION по умолчанию и для операций удаления со ссылками, и для операций изменения со ссылками; • E141–06 Ограничение CHECK; • E141–07 Значения столбца по умолчанию; • E141–10 Имена во внешнем ключе можно указывать в любом порядке.
E151 Поддержка транзакции	РЕД КВАНТ не поддерживает следующие вспомогательные функции: • E151–01 Оператор COMMIT; • E151–02 Оператор ROLLBACK.
E152 Базовый оператор SET TRANSACTION	РЕД КВАНТ не поддерживает следующие вспомогательные функции: • E152–01 Оператор SET TRANSACTION: условие ISOLATION LEVEL SERIALIZABLE;

Идентификатор функции, имя	Поддерживаемые функции
	• E152–02 Оператор SET TRANSACTION: условия READ ONLY и READ WRITE.
E153 Запросы, изменяющие данные, с подзапросами	РЕД КВАНТ полностью поддерживает эту функцию.
E161 Комментарии SQL, начинающиеся с двух минусов	РЕД КВАНТ полностью поддерживает эту функцию.
E171 Поддержка SQLSTATE	РЕД КВАНТ обеспечивает частичную поддержку этой функции, реализуя подмножество стандартных кодов ошибок и добавляя пользовательские.
E182 Привязка к языку хоста (ранее «Язык модуля»)	РЕД КВАНТ не поддерживает эту функцию.
F021 Основная информационная схема	РЕД КВАНТ не поддерживает следующие вспомогательные функции: • F021–01 Представление COLUMNS; • F021–02 Представление TABLES; • F021–03 Представление VIEWS; • F021–04 TABLE_CONSTRAINTS; • F021–05 Представление REFERENTIAL_CONSTRAINTS; • F021–06 Представление CHECK_CONSTRAINTS.
F031 Основные манипуляции со схемой	РЕД КВАНТ полностью поддерживает следующую функцию: • F031–04 Оператор ALTER TABLE: условие ADD COLUMN. РЕД КВАНТ обеспечивает частичную поддержку следующей вспомогательной функции: • F031–01 Оператор CREATE TABLE создает хранимые основные таблицы. Поддерживается базовый синтаксис. «AS» поддерживается в H2, но не в РЕД КВАНТ. Нет поддержки привилегий (INSERT, SELECT, UPDATE, DELETE). РЕД КВАНТ не поддерживает следующие подфункции: • F031–02 Оператор CREATE VIEW; • F031–03 Оператор GRANT; • F031–13 Оператор DROP TABLE: условие RESTRICT; • F031–16 Оператор DROP VIEW: условие RESTRICT; • F031–19 Оператор REVOKE: условие RESTRICT.
F041 Базовое объединение таблиц	РЕД КВАНТ полностью поддерживает следующие вспомогательные функции: • F041–01 Внутреннее соединение (но не обязательно ключевое слово INNER); • F041–02 Ключевое слово INNER;

Идентификатор функции, имя	Поддерживаемые функции
	• F041–03 LEFT OUTER JOIN; • F041–04 RIGHT OUTER JOIN; • F041–05 Внешние соединения могут быть вложенными; • F041–07 Внутренняя таблица в левом или правом внешнем соединении также может использоваться во внутреннем соединении; • F041–08 Поддерживаются все операторы сравнения (а не только =).
F051 Базовые дата и время	РЕД КВАНТ полностью поддерживает следующие вспомогательные функции: • F051–04 Предикат сравнения для типов данных DATE, TIME и TIMESTAMP; • F051–05 Явный CAST между типами даты и времени и типами символьных строк; • F051–06 CURRENT_DATE; • F051–07 LOCALTIME; • F051–08 LOCALTIMESTAMP. РЕД КВАНТ обеспечивает частичную поддержку следующих вспомогательных функций: • F051–01 Тип данных DATE (включая поддержку литерала DATE). См. IGNITE-7360. • F051–02 Тип данных TIME (включая поддержку литерала TIME) с точностью до долей секунды не менее 0. <precision> неправильно поддерживается для типа данных TIME. • F051–03 Тип данных TIMESTAMP (включая поддержку литерала TIMESTAMP) с точностью до долей секунды не менее 0 и 6. <precision> неправильно поддерживается для типа данных TIME.
F081 UNION и EXCEPT в представлениях	РЕД КВАНТ не поддерживает эту функцию.
F131 Операции группировки	РЕД КВАНТ не поддерживает следующие вспомогательные функции: • F131–01 Условия WHERE, GROUP BY и HAVING поддерживаются в запросах со сгруппированными представлениями; • F131–02 Поддержка нескольких таблиц в запросах со сгруппированными представлениями; • F131–03 Набор функций, поддерживаемых в запросах со сгруппированными представлениями; • F131–04 Подзапросы с условиями GROUP BY и HAVING и сгруппированными представлениями; • F131–05 Однострочный SELECT с условиями GROUP BY и HAVING и сгруппированными представлениями.
F181 Поддержка нескольких модулей	РЕД КВАНТ не поддерживает эту функцию.
F201 Функция CAST	РЕД КВАНТ полностью поддерживает эту функцию.

Идентификатор функции, имя	Поддерживаемые функции
F221 Явные значения по умолчанию	РЕД КВАНТ полностью поддерживает эту функцию.
F261 CASE-выражение	РЕД КВАНТ полностью поддерживает следующие вспомогательные функции: • F261–01 Простой CASE; • F261–02 CASE с поиском; • F261–03 NULLIF; • F261–04 COALESCE.
F311 Оператор определения схемы	РЕД КВАНТ не поддерживает следующие вспомогательные функции: • F311–01 CREATE SCHEMA; • F311–02 CREATE TABLE для хранимых основных таблиц; • F311–03 CREATE VIEW; • F311–04 CREATE VIEW: WITH CHECK OPTION; • F311–05 Оператор GRANT.
F471 Скалярные значения подзапроса	РЕД КВАНТ полностью поддерживает эту функцию.
F481 Расширенный предикат NULL	РЕД КВАНТ полностью поддерживает эту функцию.
F501 Представления возможностей и соответствия	РЕД КВАНТ не поддерживает следующие вспомогательные функции: • F501–01 Представление SQL_FEATURES; • F501–02 Представление SQL_SIZING; • F501–03 Представление SQL_LANGUAGES.
F812 Базовая маркировка	РЕД КВАНТ не поддерживает эту функцию. S011 Различные типы данных. РЕД КВАНТ не поддерживает следующую вспомогательную функцию: • S011–01 Представление USER_DEFINED_TYPES.
T321 Базовые процедуры, вызываемые SQL	РЕД КВАНТ не поддерживает следующие вспомогательные функции: • T321–01 Пользовательские функции без перегрузки; • T321–02 Определяемые пользователем хранимые процедуры без перегрузки; • T321–03 Вызов функции; • T321–04 Оператор CALL; • T321–05 Оператор RETURN; • T321–06 Представление ROUTINES; • T321–07 Представление PARAMETERS.

Типы данных

Таблица 33 содержит список типов данных SQL, доступных в РЕД КВАНТ, таких как строковые, числовые типы и типы даты/времени.

Каждый тип SQL сопоставляется с конкретным типом языка программирования или драйвера, который изначально поддерживается РЕД КВАНТ.

Таблица 33 - Типы данных SQL

Тип данных	Возможные значения	Сопоставлено с:
BOOLEAN	TRUE и FALSE	• Java/JDBC: java.lang.Boolean. • .NET/C#: bool. • C/C++: bool. • ODBC: SQL_BIT.
BIGINT	[-9223372036854775808, 9223372036854775807]	• Java/JDBC: java.lang.Long. • .NET/C#: long. • C/C++: int64_t. • ODBC: SQL_BIGINT.
DECIMAL	Тип данных с фиксированной точностью и масштабом.	• Java/JDBC: java.math.BigDecimal. • .NET/C#: decimal. • C/C++: ignite::Decimal. • ODBC: SQL_DECIMAL.
DOUBLE	Число с плавающей запятой.	• Java/JDBC: java.lang.Double. • .NET/C#: double. • C/C++: double. • ODBC: SQL_DOUBLE.
INT	[-2147483648, 2147483647]	• Java/JDBC: java.lang.Integer. • .NET/C#: int. • C/C++: int32_t. • ODBC: SQL_INTEGER.
REAL	Число одинарной точности с плавающей запятой.	• Java/JDBC: java.lang.Float. • .NET/C#: float. • C/C++: float. • ODBC: SQL_FLOAT.
SMALLINT	[-32768, 32767]	• Java/JDBC: java.lang.Short. • .NET/C#: short. • C/C++: int16_t. • ODBC: SQL_SMALLINT.
TINYINT	[-128, 127]	• Java/JDBC: java.lang.Byte. • .NET/C#: sbyte. • C/C++: int8_t. • ODBC: SQL_TINYINT.
CHAR	Строка в Юникоде. Этот тип поддерживается для совместимости с другими базами данных и старыми приложениями.	• Java/JDBC: java.lang.String. • .NET/C#: string. • C/C++: std::string. • ODBC: SQL_CHAR.
VARCHAR	Строка в Юникоде.	• Java/JDBC: java.lang.String. • .NET/C#: string. • C/C++: std::string. • ODBC: SQL_VARCHAR.

Тип данных	Возможные значения	Сопоставлено с:
DATE	Тип данных даты. Формат — гггг-ММ-дд.	• Java/JDBC: java.sql.Date. • .NET/C#: N/A. • C/C++: ignite::Date. • ODBC: SQL_TYPE_DATE. По возможности лучше использовать тип TIMESTAMP вместо DATE. Тип DATE сериализуется/десериализуется очень неэффективно, что приводит к снижению производительности.
TIME	Тип данных времени. Формат чч:мм:сс.	• Java/JDBC: java.sql.Time. • .NET/C#: N/A. • C/C++: ignite::Time. • ODBC: SQL_TYPE_TIME.
TIMESTAMP	Тип данных метки времени. Формат: гггг-ММ-дд чч:мм:сс[.нннннннн].	• Java/JDBC: java.sql.Timestamp. • .NET/C#: System.DateTime. • C/C++: ignite::Timestamp. • ODBC: SQL_TYPE_TIMESTAMP.
BINARY	Представляет собой массив байтов.	• Java/JDBC: byte[]. • .NET/C#: byte[]. • C/C++: int8_t[]. • ODBC: SQL_BINARY.
GEOMETRY	Тип пространственной геометрии, основанный на библиотеке com.vivids.jts. Обычно представляется в текстовом формате с использованием формата WKT (общеизвестный текст).	• Java/JDBC: Types from the com.vividsolutions.jts package. • .NET/C#: N/A. • C/C++: N/A. • ODBC: N/A.
UUID	Универсальный уникальный идентификатор. Это 128-битное значение.	• Java/JDBC: java.util.UUID. • .NET/C#: System.Guid. • C/C++: ignite::Guid. • ODBC: SQL_GUID.

14 ПРИЛОЖЕНИЕ B. DDL

14.1 CREATE TABLE

Команда создает новый кэш РЕД КВАНТ и определяет поверх него таблицу SQL. Соответствующий кэш хранит данные в виде пар «ключ-значение», а таблица позволяет обрабатывать данные с помощью SQL-запросов.

Таблица будет находиться в схеме, указанной в параметрах подключения. Если схема не указана, по умолчанию будет использоваться PUBLIC.

Команда CREATE TABLE выполняется синхронно. Кроме того, она блокирует выполнение других команд DDL, которые выполняются до завершения выполнения команды CREATE TABLE. На выполнение команд DML это не влияет, и они могут выполняться параллельно.

```
CREATE TABLE [IF NOT EXISTS] tableName (tableColumn [, tableColumn]...
[, PRIMARY KEY (columnName [,columnName]...)])
[WITH "paramName=paramValue [,paramName=paramValue]..."]

tableColumn := columnName columnType [DEFAULT defaultValue] [PRIMARY KEY]
```

Параметры:

• tableName - имя таблицы.

• tableColumn - имя и тип столбца, который будет создан в новой таблице.

• columnName - имя ранее определенного столбца.

• DEFAULT — указывает значение по умолчанию для столбца. Принимаются только постоянные значения.

• IF NOT EXISTS - создает таблицу, только если таблица с таким именем не существует.

• PRIMARY KEY — указывает первичный ключ для таблицы, которая может состоять из одного или нескольких столбцов.

• WITH — принимает дополнительные параметры, не определенные ANSI-99 SQL:

- TEMPLATE=<имя шаблона кэша> - имя шаблона кэша с учетом регистра. Шаблон — это экземпляр класса CacheConfiguration, зарегистрированный вызовом Ignite.addCacheConfiguration(). Необходимо использовать предопределенные шаблоны TEMPLATE=PARTITIONED или TEMPLATE=REPLICATED для создания кэша с соответствующим режимом репликации. Остальные параметры будут теми, что определены в объекте CacheConfiguration. Если шаблон не указан явно, по умолчанию используется TEMPLATE=PARTITIONED.
- BACKUPS=<количество резервных копий> - устанавливает количество резервных копий разделов. Если ни этот параметр, ни параметр TEMPLATE не установлены, то кэш создается с 0 резервными копиями.

- ATOMICITY=<ATOMIC | TRANSACTIONAL | TRANSACTIONAL_SNAPSHOT> — устанавливает режим атомарности для базового кэша. Если ни этот параметр, ни параметр TEMPLATE не заданы, то кэш создается с включенным режимом ATOMIC. Если указано TRANSACTIONAL_SNAPSHOT, таблица будет поддерживать транзакции.

- WRITE_SYNCHRONIZATION_MODE=<PRIMARY_SYNC | FULL_SYNC | FULL_ASYNC> — устанавливает режим синхронизации записи для базового кэша. Если ни этот параметр, ни параметр TEMPLATE не установлены, то кэш создается с включенным режимом FULL_SYNC.

- CACHE_GROUP=<имя группы> — указывает имя группы, которой принадлежит базовый кэш.

- AFFINITY_KEY=<имя столбца ключа привязки> — указывает имя affinity-ключа, которое является столбцом ограничения PRIMARY KEY.

- CACHE_NAME=<настраиваемое имя нового кэша> - имя базового кэша, созданного командой, или формат SQL_{SCHEMA_NAME}_{TABLE}, если параметр не указан.

- DATA_REGION=<имя существующей области данных> - имя области данных, в которой должны храниться записи таблицы. По умолчанию РЕД КВАНТ хранит все данные в области по умолчанию.

- KEY_TYPE=<настраиваемое имя типа ключа> — устанавливает имя типа пользовательского ключа, который используется из API-интерфейсов «ключ-значение» в РЕД КВАНТ. Имя должно соответствовать классу Java, .NET или C++ или может быть случайным, если вместо пользовательского класса используется BinaryObjects. Количество полей и их типов в типе пользовательского ключа должно соответствовать PRIMARY KEY.

- VALUE_TYPE=<настраиваемое имя типа значения нового кэша> — задает имя типа пользовательского значения, которое используется из пары «ключ-значение» и других не-SQL API в РЕД КВАНТ. Имя должно соответствовать классу Java, .NET или C++ или может быть случайным, если вместо пользовательского класса используется BinaryObjects. Тип значения должен включать все столбцы, определенные в команде CREATE TABLE, за исключением тех, которые перечислены в ограничении PRIMARY KEY.

- WRAP_KEY=<true | false> — этот флаг определяет, должен ли отдельный столбец PRIMARY KEY быть обернут в формат BinaryObjects или нет. По умолчанию для этого флага установлено значение false. Этот флаг не влияет на PRIMARY KEY с несколькими столбцами; такой ключ всегда оборачивается независимо от значения параметра.

- WRAP_VALUE=<true | false> — этот флаг определяет, должно ли отдельное значение столбца примитивного типа быть обернуто в формат BinaryObjects или нет. По умолчанию для этого флага установлено значение true. Этот флаг не влияет на

значение с несколькими столбцами; такое значение всегда упаковывается независимо от значения параметра. Если есть одно значение столбца и не планируется добавлять дополнительные столбцы в таблицу, для этого параметра необходимо установить значение false. Следует отметить, что если для параметра установлено значение false, нельзя использовать команду ALTER TABLE ADD COLUMN для этой конкретной таблицы.

14.2 ALTER TABLE

Изменяет структуру существующей таблицы.

```
ALTER TABLE [IF EXISTS] tableName {alter_specification}


alter_specification:
   ADD [COLUMN] {[IF NOT EXISTS] tableColumn | (tableColumn [,...])}
 | DROP [COLUMN] {[IF EXISTS] columnName | (columnName [,...])}
 | {LOGGING | NOLOGGING}

tableColumn := columnName columnType
```

В настоящее время РЕД КВАНТ поддерживает только добавление и удаление столбцов.

Параметры:

- tableName — имя таблицы.
- tableColumn — имя и тип добавляемого в таблицу столбца.
- columnName — имя добавляемого или удаляемого столбца.
- IF EXISTS — при применении к TABLE — не выдавать ошибку, если таблица с указанным именем таблицы не существует. При применении к COLUMN — не выдавать ошибку, если столбец с указанным именем не существует.
- IF NOT EXISTS — не выдавать ошибку, если столбец с таким именем уже существует.
- LOGGING — включить ведение журнала с упреждающей записью для таблицы. Логирование с упреждающей записью включено по умолчанию. Команда актуальна, только если используется персистентность РЕД КВАНТ.
- NOLOGGING — отключить ведение журнала с упреждающей записью для таблицы. Команда актуальна, только если используется персистентность РЕД КВАНТ.

ALTER TABLE ADD добавляет новый столбец или несколько столбцов в ранее созданную таблицу. После добавления столбца доступ к нему можно получить с помощью команд DML и проиндексировать с помощью оператора CREATE INDEX.

ALTER TABLE DROP удаляет существующий столбец или несколько столбцов из таблицы. После удаления столбца доступ запросов к нему становится невозможен. Следует обратить внимание на следующие примечания и ограничения:

- Команда не удаляет фактические данные из кластера, что означает, что если столбец «имя» удален, значение «имя» все еще сохраняется в кластере. Это ограничение будет устранено в следующих версиях.

- Если столбец был проиндексирован, индекс необходимо удалить вручную с помощью команды DROP INDEX.

- Невозможно удалить столбец, который является первичным ключом или частью такого ключа.

- Невозможно удалить столбец, если он представляет все значение, хранящееся в кластере. Ограничение актуально для примитивных значений. РЕД КВАНТ хранит данные в виде пар «ключ-значение», и все новые столбцы будут принадлежать значению. Невозможно изменить набор столбцов ключа (PRIMARY KEY).

Нацеленные на одну и ту же таблицу команды DDL и DML блокируются на короткое время, пока не будет выполнена инструкция ALTER TABLE.

Примененные этой командой изменения схемы сохраняются на диске, если включена персистентность РЕД КВАНТ. Таким образом, изменения могут сохраняться при полном перезапуске кластера.

Примеры:

1) Добавить столбец в таблицу:

```
ALTER TABLE Person ADD COLUMN city varchar;
```

2) Добавить новый столбец в таблицу только в том случае, если столбец с таким именем не существует:

```
ALTER TABLE City ADD COLUMN IF NOT EXISTS population int;
```

3) Удалить столбец из таблицы:

```
ALTER TABLE Person DROP COLUMN city;
```

4) Удалить столбец только в том случае, если таблица существует:

```
ALTER TABLE IF EXISTS Person DROP COLUMN number;
```

5) Отключить ведение журнала с упреждающей записью:

```
ALTER TABLE Person NOLOGGING
```

14.3 DROP TABLE

Команда DROP TABLE удаляет существующую таблицу. Соответствующий кэш со всеми данными в нем также уничтожается.

```
DROP TABLE [IF EXISTS] tableName
```

Параметры:

- tableName - имя таблицы.

- IF NOT EXISTS - не выдавать ошибку, если таблицы с таким именем не существует.

Нацеленные на одну и ту же таблицу команды DDL и DML блокируются, пока выполняется DROP TABLE. После удаления таблицы все отложенные команды завершатся с соответствующими ошибками.

Примененные этой командой изменения схемы сохраняются на диске, если включена персистентность РЕД КВАНТ. Таким образом, изменения могут сохраняться при полном перезапуске кластера.

Примеры:

Удалить таблицу Person, если она существует:

```
DROP TABLE IF EXISTS "Person";
```

14.4 CREATE INDEX

Создает индекс для указанной таблицы.

```
CREATE [SPATIAL] INDEX [[IF NOT EXISTS] indexName] ON tableName
  (columnName [ASC|DESC] [,...]) [(index_option [...])]
index_option := {INLINE_SIZE size | PARALLEL parallelism_level}
```

Параметры:

- indexName — имя создаваемого индекса.
- ASC — указывает порядок сортировки по возрастанию (это порядок по умолчанию).
- DESC — определяет порядок сортировки по убыванию.
- SPATIAL — создает пространственный индекс. В настоящее время поддерживаются только геометрические типы.
- IF NOT EXISTS — не выдавать ошибку, если индекс с таким именем уже существует. База данных проверяет только имена индексов и не учитывает типы или количество столбцов.
- index_option — дополнительные опции для создания индекса:
  - INLINE_SIZE — указывает inline size индекса в байтах. В зависимости от размера РЕД КВАНТ поместит все проиндексированное значение или его часть непосредственно на страницы индекса, тем самым исключив дополнительные обращения к страницам данных и повысив производительность запросов. Inlining включен по умолчанию, а размер предварительно рассчитывается автоматически на основе структуры таблицы. Чтобы отключить inlining, нужно установить размер равным 0 (не рекомендуется).
  - PARALLEL — указывает количество потоков, которые будут использоваться параллельно для создания индекса. Чем большее число установлено, тем быстрее создается и строится индекс. Если значение превышает количество ЦП, то оно будет

уменьшено до количества ядер. Если параметр не указан, то количество потоков рассчитывается как 25% доступных ядер ЦП.

CREATE INDEX создает новый индекс для указанной таблицы. Обычные индексы хранятся во внутренних структурах данных B+tree. B+tree распространяется по кластеру вместе с фактическими данными. Узел кластера хранит часть индекса для данных, которыми он владеет.

Если CREATE INDEX выполняется на оперативных данных во время выполнения, база данных будет перебирать указанные столбцы, синхронно индексируя их. Остальные команды DDL, нацеленные на ту же таблицу, блокируются до тех пор, пока не будет выполнено CREATE INDEX. Выполнение команды DML не затрагивается и может проводиться параллельно.

Примененные этой командой изменения схемы сохраняются на диске, если включена персистентность РЕД КВАНТ. Таким образом, изменения могут сохраняться при полном перезапуске кластера.

Компромиссы индексов

При выборе индексов для приложения следует учитывать несколько факторов.

- Индексы не бесплатны. Они потребляют память, и каждый индекс необходимо обновлять отдельно, поэтому производительность операций записи может снизиться при создании слишком большого количества индексов. Кроме того, если определено много индексов, оптимизатор может сделать больше ошибок, выбрав неправильный индекс при построении плана выполнения.

Внимание! Индексировать все — плохая стратегия.

- Индексы — это отсортированные структуры данных (B+tree). При определении индекса полей (a,b,c) записи будут отсортированы сначала по a, затем по b и только потом по c.

- Индексы для отдельных полей не лучше, чем групповые индексы для нескольких полей, начинающихся с одного и того же поля (индекс для (a) не лучше, чем (a, b, c)). Поэтому предпочтительнее использовать групповые индексы.

- Если указан параметр INLINE_SIZE, индексы содержат префикс поля данных на страницах B+tree. Это повышает производительность поиска за счет меньшего количества извлечений строк данных, однако существенно увеличивает размер дерева (с умеренным увеличением высоты дерева) и снижает производительность вставки и удаления данных из-за чрезмерного разбиения и слияния страниц. При выборе inline size для дерева рекомендуется учитывать размер страницы: для каждой записи B-tree требуется 16 + inline size на странице (плюс заголовок и дополнительные ссылки для страницы).

Примеры:

1) Создать обычный индекс:

```
CREATE INDEX title_idx ON books (title);
```

2) Создать индекс по убыванию, только если он не существует:

```
CREATE INDEX IF NOT EXISTS name_idx ON persons (firstName DESC);
```

3) Создать составной индекс:

```
CREATE INDEX city_idx ON sales (country, city);
```

4) Создать индекс, определяющий встроенный размер данных:

```
CREATE INDEX fast_city_idx ON sales (country, city) INLINE_SIZE 60;
```

5) Создать геопространственный индекс:

```
CREATE SPATIAL INDEX idx_person_address ON Person (address);
```

14.5 DROP INDEX

DROP INDEX удаляет существующий индекс.

```
DROP INDEX [IF EXISTS] indexName
```

Параметры:

- indexName — имя удаляемого индекса.
- IF EXISTS - не выдавать ошибку, если индекса с указанным именем не существует. База данных проверяет только имена индексов, не учитывая типы или количество столбцов.

Нацеленные на одну и ту же таблицу команды DDL блокируются до тех пор, пока не будет выполнено DROP INDEX. Выполнение команды DML не затрагивается и может проводиться параллельно.

Примененные этой командой изменения схемы сохраняются на диске, если включена персистентность РЕД КВАНТ. Таким образом, изменения могут сохраняться при полном перезапуске кластера.

Пример:

Удалить индекс:

```
DROP INDEX idx_person_name;
```

14.6 CREATE USER

Команда создает пользователя с заданным именем и паролем.

Нового пользователя можно создать только с использованием учетной записи суперпользователя, если включена проверка подлинности для тонких клиентов. РЕД КВАНТ создает учетную запись суперпользователя под именем ignite и паролем ignite при первом запуске кластера. В настоящее время нельзя ни переименовать учетную запись суперпользователя, ни предоставить ее привилегии какой-либо другой учетной записи.

```
CREATE USER userName WITH PASSWORD 'password';
```

Параметры:

- userName - имя нового пользователя. Имя не может быть длиннее 60 байт в кодировке UTF8.

- пароль - новый пароль пользователя. Пустой пароль не допускается.

Чтобы создать имя пользователя с учетом регистра, используйте идентификатор SQL в кавычках (").

Примечание: Свойство нечувствительности к регистру имен пользователей поддерживается только для интерфейсов JDBC и ODBC. Если планируется доступ к РЕД КВАНТ из API Java, .NET или другого языка программирования, то имя пользователя должно быть передано либо заглавными буквами, либо заключено в двойные кавычки ("").

Например, если Test был установлен как имя пользователя, то:

- Можно использовать Test, TEst, TEST и другие комбинации из JDBC и ODBC.

- Можно использовать либо TEST, либо «Test» в качестве имени пользователя из встроенных API SQL РЕД КВАНТ, разработанных для Java, .NET и других языков программирования.

В качестве альтернативы нужно всегда использовать имя пользователя с чувствительностью к регистрам для обеспечения согласованности имени во всех интерфейсах SQL.

Примеры:

1) Создать нового пользователя, используя test в качестве имени и пароля:

```
CREATE USER test WITH PASSWORD 'test';
```

2) Создать имя пользователя с учетом регистра:

```
CREATE USER "TeSt" WITH PASSWORD 'test'
```

14.7 ALTER USER

Команда изменяет существующий пароль пользователя. Пароль может быть обновлен суперпользователем или самим пользователем (подробнее см. CREATE USER).

```
ALTER USER userName WITH PASSWORD 'newPassword';
```

Параметры:

- userName - имя существующего пользователя.

- newPassword — новый пароль для установки учетной записи пользователя.

Примеры:

Обновление пароля пользователя:

```
ALTER USER test WITH PASSWORD 'test123';
```

14.8 DROP USER

Команда удаляет существующего пользователя.

Удалить пользователя может только суперпользователь (подробнее см. CREATE USER).

```
DROP USER userName;
```

Параметры:

• userName - имя удаляемого пользователя.

Примеры:

```
DROP USER test;
```

14.9 ANALYZE

Команда ANALYZE собирает статистику.

```
ANALYZE 'schemaName'.'tableName'(column1, column2);
```

Параметры:

• schemaName — имя схемы, для которой собирается статистика.

• tableName — имя таблицы, для которой собирается статистика.

• (столбец1, столбец2) - названия столбцов, по которым собирается статистика.

Когда команда ANALYZE используется с оператором with, указанные параметры применяются для каждой цели. Например:

```
ANALYZE public.statistics_test, statistics_test2, statistics_test3(col3) WITH 'MAX_CHANGED_PARTITION_ROWS_PERCENT=15,NULLS=0'
```

Возможные параметры:

• MAX_CHANGED_PARTITION_ROWS_PERCENT — максимальный процент устаревших строк в таблице (значение по умолчанию — 15%). Дополнительные сведения см. в Устаревание статистики.

• NULLS — количество нулевых значений в столбце.

• TOTAL — общее количество значений столбца.

• SIZE — средний размер значений столбца (в байтах).

• DISTINCT — количество различных ненулевых значений в столбце.

14.10 REFRESH

Команда обновляет статистику.

```
REFRESH 'schemaName'.'tableName'(column1, column2);
```

Параметры:

• schemaName — имя схемы, для которой необходимо обновить статистику.

• tableName - имя таблицы, для которой необходимо обновить статистику.

• (столбец1, столбец2) - имена столбцов, для которых необходимо обновить статистику.

Пример:

```
REFRESH PRODUCTS, SALE(productId, discount)
```

14.11 DROP STATISTICS

Команда сбрасывает статистику.

```
DROP STATISTICS 'schemaName'.'tableName'(column1, column2);
```

Параметры:

• schemaName — имя схемы, для которой сбрасывается статистика.

• tableName - имя таблицы, для которой сбрасывается статистика.

• (столбец1, столбец2) - имена столбцов, по которым сбрасывается статистика.

Пример:

```
DROP STATISTICS USERS, ORDERS(customerId, productId)
```

15 ПРИЛОЖЕНИЕ Г. DML

Ниже представлены все команды языка обработки данных (DML), поддерживаемые РЕД КВАНТ.

15.1 SELECT

Извлекает данные из таблицы или нескольких таблиц.

Параметры:

• DISTINCT - удаляет повторяющиеся строки из набора результатов.

• GROUP BY - группирует результат по заданному выражению(ям).

• HAVING - фильтрует строки после группировки.

• ORDER BY - сортирует результат по заданным столбцам или выражениям.

• LIMIT и FETCH FIRST/NEXT ROW(S) ONLY - ограничивает количество строк, возвращаемых запросом (без ограничений, если значение null или меньше нуля).

• OFFSET - указывает, сколько строк нужно пропустить.

• UNION, INTERSECT, MINUS, EXPECT - объединяет результат этого запроса с результатами другого запроса.

• tableExpression - присоединяется к таблице. Выражение join не поддерживается для перекрестных и естественных соединений. Естественное соединение - это внутреннее соединение, при котором условие автоматически применяется к столбцам с одинаковыми именами.

• LEFT - LEFT JOIN выполняет соединение, начиная с первой (самой левой) таблицы, а затем с любыми соответствующими вторыми (самыми правыми) записями таблицы.

• RIGHT - RIGHT JOIN выполняет соединение, начиная со второй (самой правой) таблицы, а затем с любыми соответствующими записями первой (самой левой) таблицы.

• OUTER - внешние соединения далее подразделяются на левые внешние соединения, правые внешние соединения и полные внешние соединения, в зависимости от того, какие строки таблицы сохраняются (левые, правые или и те, и другие).

• INNER - для внутреннего соединения требуется, чтобы каждая строка в двух соединенных таблицах имела совпадающие значения столбцов.

• CROSS - CROSS JOIN возвращает декартово произведение строк из таблиц в объединении.

• NATURAL - Естественное соединение является частным случаем эквивалентного соединения.

• ON - значение или условие для присоединения.

Запросы SELECT могут выполняться как для реплицированных, так и для партицированных данных.

Когда запросы выполняются к полностью реплицированным данным, РЕД КВАНТ отправляет запрос на один узел кластера и выполняет его там над локальными данными.

С другой стороны, если запрос выполняется над партицированными данными, то процесс выполнения будет следующим:

• Запрос будет проанализирован и разделен на несколько запросов словаря и один запрос сокращения (reduce).

• Все запросы к словарю выполняются на всех узлах, где находятся требуемые данные.

• Все узлы предоставляют наборы результатов локального выполнения инициатору запроса (reducer), который, в свою очередь, завершит фазу сокращения путем правильного объединения предоставленных наборов результатов.

15.1.1 JOINs

РЕД КВАНТ поддерживает сколоцированные и не сколоцированные распределенные join'ы SQL. Кроме того, если данные находятся в разных таблицах (то есть в кэшах РЕД КВАНТ), РЕД КВАНТ также допускает объединение таблиц.

Join'ы между партицированными и реплицированными наборами данных всегда работают без каких-либо ограничений.

Однако, если выполняется join партицированных наборов данных, то нужно убедиться, что ключи, по которым выполняется join, сколоцированы, либо убедиться, что включен параметр не сколоцированных join'ов для запроса.

15.1.2 Группировка и упорядочивание по оптимизации

Запросы SQL с условиями ORDER BY не требуют загрузки всего набора результатов в узел инициатора запроса (reducer) для завершения сортировки. Вместо этого каждый узел, которому будет адресован запрос, будет сортировать свою часть общего набора результатов, а reducer выполнит слияние в потоковом режиме.

Такая же оптимизация реализована для отсортированных запросов GROUP BY — нет необходимости загружать весь результирующий набор в reducer, чтобы выполнить группировку перед тем, как отдать его приложению. В РЕД КВАНТ частичные наборы результатов с отдельных узлов можно передавать в потоковом режиме, объединять, агрегировать и постепенно возвращать в приложение.

Примеры:

1) Получить все строки из таблицы Person:

```
SELECT * FROM Person;
```

2) Получить все строки в алфавитном порядке:

```
SELECT * FROM Person ORDER BY name;
```

3) Соединить данные, хранящиеся в таблицах Person и City:

```
SELECT p.name, c.name
FROM Person p, City c
WHERE p.city_id = c.id;
```

15.2 INSERT

INSERT добавляет запись или записи в таблицу.

Параметры:

• tableName — имя обновляемой таблицы.

• columnName — имя столбца, который нужно инициализировать значением из предложения VALUES.

Поскольку РЕД КВАНТ хранит все данные в виде пар «ключ-значение», все операторы INSERT в конечном итоге преобразуются в набор операций «ключ-значение».

Если в кэш добавляется одна пара «ключ-значение», то инструкция INSERT будет преобразована в операцию cache.putIfAbsent(…). В случаях, когда вставляется несколько пар «ключ-значение», механизм DML создает EntryProcessor для каждой пары и использует cache.invokeAll(…) для передачи данных в кэш.

Примеры:

1) Добавить запись в таблицу Person:

```
INSERT INTO Person (id, name, city_id) VALUES (1, 'John Doe', 3);
```

2) Заполнить таблицу Person данными, полученными из таблицы Account:

```
INSERT INTO Person(id, name, city_id)
(SELECT a.id + 1000, concat(a.firstName, a.secondName), a.city_id
FROM Account a WHERE a.id > 100 AND a.id < 1000);
```

15.3 UPDATE

UPDATE изменяет существующие записи, хранящиеся в таблице.

Параметры:

• table - имя обновляемой таблицы.

• columnName — имя столбца, который будет обновлен значением из предложения SET.

Поскольку РЕД КВАНТ хранит все данные в виде пар «ключ-значение», все операторы UPDATE в конечном итоге преобразуются в набор операций «ключ-значение».

Движок SQL генерирует и выполняет запрос SELECT на основе выражения UPDATE WHERE и только после этого изменяет существующие значения, удовлетворяющие результату условия.

Модификация выполняется с помощью операции cache.invokeAll(…). Это означает, что как только результат запроса SELECT будет готов, движок SQL подготовит несколько EntryProcessors и выполнит их все с помощью операции cache.invokeAll(…). Пока данные изменяются с помощью EntryProcessors, выполняются дополнительные проверки, чтобы убедиться, что никто не вмешивался между SELECT и фактическим обновлением.

15.3.1 Обновления первичных ключей

РЕД КВАНТ не позволяет обновлять первичный ключ, потому что последний определяет раздел, к которому ключ и его значение принадлежат статически. Хотя раздел со всеми его данными может сменить нескольких владельцев кластера, ключ всегда принадлежит одному разделу. Раздел вычисляется с использованием хеш-функции, применяемой к значению ключа.

Таким образом, если ключ необходимо обновить, его необходимо удалить, а затем вставить.

Примеры:

1) Обновить столбец name для записи:

```
UPDATE Person SET name = 'John Black' WHERE id = 2;
```

2) Обновить таблицу Person данными, взятыми из таблицы Account:

```
UPDATE Person p SET name = (SELECT a.first_name FROM Account a WHERE a.id = p.id)
```

15.4 WITH

Используется для обозначения подзапроса, на него можно ссылаться в других частях SQL-выражения.

Параметры:

• query_name — имя создаваемого подзапроса. Имя, назначенное подзапросу, рассматривается как встроенное представление или таблица.

WITH создает подзапрос. На одну или несколько общих записей таблицы можно ссылаться по имени. Объявления имен столбцов необязательны — имена столбцов будут выводиться из именованных запросов на выборку. Последним действием в операторе WITH может быть select, insert, update, merge, delete или create table.

Пример:

```
WITH cte1 AS (
    SELECT 1 AS FIRST_COLUMN
), cte2 AS (
    SELECT FIRST_COLUMN+1 AS FIRST_COLUMN FROM cte1
)
SELECT sum(FIRST_COLUMN) FROM cte2;
```

15.5 MERGE

MERGE обновляет существующие записи и вставляет новые.

Параметры:

• tableName — имя обновляемой таблицы.

• columnName — имя столбца, который будет инициализирован значением из выражения VALUES.

Поскольку РЕД КВАНТ хранит все данные в виде пар «ключ-значение», все операторы MERGE преобразуются в набор операций «ключ-значение».

MERGE — одна из самых простых операций, поскольку она транслируется в операции cache.put(…) и cache.putAll(…) в зависимости от количества строк, которые необходимо вставить или обновить как часть запроса MERGE.

Примеры:

1) Объединить несколько строк в таблицу Person:

```
MERGE INTO Person(id, name, city_id) VALUES
(1, 'John Smith', 5),
(2, 'Mary Jones', 5);
```

2) Заполнить таблицу Person данными, полученными из таблицы Account:

```
MERGE INTO Person(id, name, city_id)
(SELECT a.id + 1000, concat(a.firstName, a.secondName), a.city_id
FROM Account a WHERE a.id > 100 AND a.id < 1000);
```

15.6 DELETE

DELETE удаляет данные из таблицы.

Параметры:

• tableName — имя таблицы, из которой удаляются данные.

• TOP, LIMIT — указывает количество удаляемых записей (без ограничений, если значение null или меньше нуля).

Поскольку РЕД КВАНТ хранит все данные в виде пар «ключ-значение», все операторы DELETE преобразуются в набор операций «ключ-значение».

Выполнение инструкции DELETE разделено на две фазы и аналогично выполнению инструкции UPDATE.

Во-первых, используя запрос SELECT, движок SQL собирает те ключи, которые удовлетворяют условию WHERE в операторе DELETE. Затем он создает несколько EntryProcessors и выполняет их с помощью cache.invokeAll(…). Во время удаления данных выполняются дополнительные проверки, чтобы убедиться, что между SELECT и фактическим удалением данных никто не вмешивался.

Пример:

Удалить все записи таблицы Person с определенным именем:

```
DELETE FROM Person WHERE name = 'John Doe';
```

16 ПРИЛОЖЕНИЕ Д. SQL КОМАНДЫ

Ниже представлены операционные команды, которые поддерживаются РЕД КВАНТ.

16.1 COPY

Копирует данные из CSV-файла в таблицу SQL.

```
COPY FROM '/path/to/local/file.csv'
INTO tableName (columnName, columnName, ...) FORMAT CSV [CHARSET '<charset-name>']
```

Параметры:

- '/path/to/local/file.csv' - фактический путь к CSV-файлу.
- tableName — имя таблицы, в которую будут скопированы данные.
- columnName - имя столбца, соответствующее столбцам в CSV-файле.

COPY позволяет копировать содержимое файла в локальной файловой системе на сервер и применять его данные к таблице SQL. Внутри COPY считывает содержимое файла в двоичной форме в пакеты данных и отправляет эти пакеты на сервер. Затем содержимое файла анализируется и выполняется в потоковом режиме. Рекомендуется использовать этот режим, если есть данные, сброшенные в файл.

Примечание: В настоящее время COPY поддерживается только драйвером JDBC и может работать только с форматом CSV.

Пример:

COPY можно выполнить так:

```
COPY FROM '/path/to/local/file.csv' INTO city (
ID, Name, CountryCode, District, Population) FORMAT CSV
```

16.2 SET STREAMING

Массовая потоковая передача данных из файла в таблицу SQL.

Используя команду SET, можно выполнять массовую потоковую передачу данных в таблицу SQL в кластере. Когда потоковая передача включена, драйвер JDBC/ODBC будет упаковывать команды в пакеты и отправлять их на сервер (кластер РЕД КВАНТ). На стороне сервера пакет преобразуется в поток команд обновления кэша, которые асинхронно распределяются между серверными узлами. Асинхронное выполнение увеличивает максимальную пропускную способность, поскольку в любой момент времени все узлы кластера заняты загрузкой данных.

Для потоковой передачи данных в кластер нужно подготовить файл с помощью команды SET STREAMING ON, за которой следует команда INSERT для данных, которые необходимо загрузить. Например:

```
SET STREAMING ON;
```

```
INSERT INTO City(ID, Name, CountryCode, District, Population) VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO City(ID, Name, CountryCode, District, Population) VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO City(ID, Name, CountryCode, District, Population) VALUES (3,'Herat','AFG','Herat',186800);
INSERT INTO City(ID, Name, CountryCode, District, Population) VALUES (4,'Mazar-e-
Sharif','AFG','Balkh',127800);
INSERT INTO City(ID, Name, CountryCode, District, Population) VALUES (5,'Amsterdam','NLD','Noord-
Holland',731200);
-- More INSERT commands --
```

Следует отметить, что перед выполнением приведенных выше инструкций нужно создать таблицы в кластере. Перед командой SET STREAMING ON необходимо запустить команды CREATE TABLE или указать команды как часть файла, используемого для вставки данных, например:

```
CREATE TABLE City (

  ID INT(11),
  Name CHAR(35),
  CountryCode CHAR(3),
  District CHAR(20),
  Population INT(11),
  PRIMARY KEY (ID, CountryCode)
) WITH "template=partitioned, backups=1, affinityKey=CountryCode, CACHE_NAME=City,
KEY_TYPE=demo.model.CityKey, VALUE_TYPE=demo.model.City";

SET STREAMING ON;

INSERT INTO City(ID, Name, CountryCode, District, Population) VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO City(ID, Name, CountryCode, District, Population) VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO City(ID, Name, CountryCode, District, Population) VALUES (3,'Herat','AFG','Herat',186800);
INSERT INTO City(ID, Name, CountryCode, District, Population) VALUES (4,'Mazar-e-
Sharif','AFG','Balkh',127800);
INSERT INTO City(ID, Name, CountryCode, District, Population) VALUES (5,'Amsterdam','NLD','Noord-
Holland',731200);
-- More INSERT commands --
```

Примечание: Когда загрузка данных завершена, необходимо закрыть соединение JDBC/ODBC, чтобы все данные были сброшены в кластер.

16.2.1 Ограничения

Хотя потоковый режим позволяет загружать данные намного быстрее, чем другие методы загрузки данных, упомянутые в этом руководстве, он имеет некоторые ограничения:

1. Разрешены только команды INSERT; любая попытка выполнить SELECT или любую другую команду DML или DDL вызовет исключение.

2. Из-за асинхронной природы потокового режима невозможно знать количество обновлений для каждого выполняемого оператора; все команды JDBC/ODBC, возвращающие количество обновлений, будут возвращать 0.

Пример:

В качестве примера можно использовать образец файла world.sql, поставляемый с последним дистрибутивом РЕД КВАНТ. Его можно найти в каталоге {IGNITE_HOME}/examples/sql/. Можно использовать команду запуска из tools/sqlline[SQLLine, window=_blank], как показано ниже:

```
!run /apache_ignite_version/examples/sql/world.sql
```

После выполнения приведенной выше команды и закрытия соединения JDBC все данные будут загружены в кластер и готовы к запросу.

16.3 KILL QUERY

Команда KILL QUERY позволяет отменить выполняющийся запрос. Когда запрос отменяется с помощью команды KILL, все его выполняемые на других узлах части также завершаются.

```
KILL QUERY {ASYNC} 'query_id'
```

Параметры:

• query_id — можно получить через представление SQL_QUERIES.

• ASYNC — это необязательный параметр, который возвращает управление немедленно, не дожидаясь завершения отмены.

16.4 KILL TRANSACTION

Команда KILL TRANSACTION позволяет отменить текущую транзакцию.

```
KILL TRANSACTION 'xid'
```

Параметры:

• xid — идентификатор транзакции, который можно получить через представление TRANSACTIONS.

16.5 KILL SCAN

Команда KILL SCAN позволяет отменить выполняющийся запрос сканирования (scan query).

```
KILL SCAN 'origin_node_id' 'cache_name' query_id
```

Параметры:

• origin_node_id, cache_name, query_id — можно получить через представление SCAN_QUERIES.

Пример:

```
KILL SCAN '6fa749ee-7cf8-4635-be10-36a1c75267a7_54321' 'cache-name' 1
```

16.6 KILL COMPUTE

Команда KILL COMPUTER позволяет отменить запуск компьютера.

```
KILL COMPUTE 'session_id'
```

Параметры:

• session_id — можно получить через представления TASKS или JOBS.

16.7 KILL CONTINUOUS

Команда KILL CONTINUOUS позволяет отменить выполняющийся continuous query.

```
KILL CONTINUOUS 'origin_node_id', 'routine_id'
```

Параметры:

• origin_node_id и procedure_id — можно получить через представление CONTINUOUS_QUERIES.

Пример:

```
KILL CONTINUOUS '6fa749ee-7cf8-4635-be10-36a1c75267a7_54321' '6fa749ee-7cf8-4635-be10-36a1c75267a7_12345'
```

16.8 KILL SERVICE

Команда KILL SERVICE позволяет отменить работающий сервис.

```
KILL SERVICE 'name'
```

Параметры:

• name — соответствует имени, которое было выбрано для сервиса во время развертывания. Его всегда можно найти в представлении SERVICES.

16.9 KILL CONSISTENCY REPAIR/CHECK OPERATIONS

Команда KILL CONSISTENCY позволяет отменить все выполняемые операции исправления/проверки согласованности.

```
./control.sh --kill CONSISTENCY
```

17 ПРИЛОЖЕНИЕ E. SQL ФУНКЦИИ

В таблице 34 представлены основные функции SQL.

Таблица 34 - SQL функции

Функция	Описание	Параметры	Пример
Функции агрегирования			
AVG	AVG ([DISTINCT] expression) Среднее значение. Если строки не выбраны, результатом будет NULL. Агрегатные функции разрешены только в операторах select. Возвращаемое значение имеет тот же тип данных, что и параметр.	• DISTINCT - необязательное ключевое слово. Если присутствует, будет выполнено усреднение уникальных значений.	Расчет среднего возраста игроков: SELECT AVG(age) "AverageAge" FROM Players;
BIT_AND	BIT_AND (expression) Побитовое И всех ненулевых значений. Если строки не выбраны, результатом будет NULL. Агрегатные функции разрешены только в операторах select. Логическая операция И выполняется над каждой парой соответствующих битов двух двоичных выражений одинаковой длины. В каждой паре он возвращает 1, если первый бит равен 1 И второй бит равен 1. В противном случае он возвращает 0.		
BIT_OR	BIT_OR (expression) Побитовое ИЛИ всех ненулевых значений. Если строки не выбраны, результатом будет NULL. Агрегатные функции разрешены только в операторах select. Логическая операция ИЛИ выполняется над каждой парой соответствующих битов двух двоичных выражений одинаковой длины.		

Функция	Описание	Параметры	Пример
	В каждой паре результат равен 1, если первый бит равен 1 ИЛИ второй бит равен 1 ИЛИ оба бита равны 1, иначе результат равен 0.		
COUNT	COUNT (* | [DISTINCT] expression) Количество всех записей или ненулевых значений. Этот метод возвращает значение long. Если записи не выбраны, результат равен 0. Агрегатные функции разрешены только в операторах select.		Подсчитать количество игроков в каждом городе: SELECT city_id, COUNT(*) FROM Players GROUP BY city_id;
GROUP_CONCAT	GROUP_CONCAT([DISTINCT] expression || [expression || [expression ...]] [ORDER BY expression [ASC|DESC], [[ORDER BY expression [ASC|DESC]]] [SEPARATOR expression]) Объединяет строки с помощью разделителя. Разделителем по умолчанию является ',' (без пробела). Этот метод возвращает строку. Если записи не выбраны, результатом будет NULL. Агрегатные функции разрешены только в операторах select. Выражение может представлять собой объединение столбцов и строк с использованием оператора ||, например: column1 || "=" || column2.	• DISTINCT — фильтрует набор результатов для уникальных наборов выражений. • expression — задает выражение, которое может быть именем столбца, результатом другой функции или математической операцией. • ORDER BY — упорядочивает строки по выражению. • SEPARATOR — переопределяет разделитель строк. По умолчанию в качестве разделителя используется запятая ','.	Сгруппировать имена всех игроков в один ряд: SELECT GROUP_CONCAT(name ORDER BY id SEPARATOR ', ') FROM Players;
MAX	MAX (expression) Возвращает наибольшее значение. Если записи не выбраны, результатом будет NULL. Агрегатные функции разрешены только в операторах select. Возвращаемое значение имеет тот же тип данных, что и параметр.	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вернуть рост самого высокого игрока: SELECT MAX(height) FROM Players;
MIN	MIN (expression) Возвращает наименьшее значение. Если записи не выбраны, результатом будет NULL. Агрегатные функции разрешены только в	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вернуть возраст самого молодого игрока: SELECT MIN(age) FROM Players;

Функция	Описание	Параметры	Пример
	операторах select. Возвращаемое значение имеет тот же тип данных, что и параметр.		
SUM	SUM ([DISTINCT] expression) Возвращает сумму всех значений. Если записи не выбраны, результатом будет NULL. Агрегатные функции разрешены только в операторах select. Тип данных возвращаемого значения зависит от данных параметра.	• DISTINCT — накапливает только уникальные значения. • expression — может быть именем столбца, результатом другой функции или математической операции.	Получить общее количество голов, забитых всеми игроками: SELECT SUM(goal) FROM Players;
Числовые функции			
ABS	ABS (expression) Возвращает абсолютное значение выражения.	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вернуть абсолютное значение: SELECT transfer_id, ABS (price) from Transfers;
ACOS	ACOS (expression) Вычисляет арккосинус. Этот метод возвращает значение double.	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вычислить арккосинус: SELECT acos(angle) FROM Triangles;
ASIN	ASIN (expression) Вычисляет арксинус. Этот метод возвращает значение double.	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вычислить арксинус: SELECT asin(angle) FROM Triangles;
ATAN	ATAN (expression) Вычисляет арктангенс. Этот метод возвращает значение double.	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вычислить арктангенс: SELECT atan(angle) FROM Triangles;
COS	COS (expression) Вычисляет тригонометрический косинус. Этот метод возвращает значение double.	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вычислить косинус: SELECT COS(angle) FROM Triangles;
COSH	COSH (expression) Вычисляет гиперболический косинус. Этот метод возвращает значение double.	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вычислить гиперболический косинус: SELECT HCOS(angle) FROM Triangles;
COT	COT (expression)	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вычислить котангенс: SELECT COT(angle) FROM Triangles;

Функция	Описание	Параметры	Пример
	Вычисляет тригонометрический котангенс (1/TAN (УГОЛ)). Этот метод возвращает значение double.		
SIN	SIN (expression) Вычисляет тригонометрический синус. Этот метод возвращает значение double.	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вычислить синус: SELECT SIN(angle) FROM Triangles;
SINH	SINH (expression) Вычисляет гиперболический синус. Этот метод возвращает значение double.	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вычислить гиперболический синус: SELECT SINH(angle) FROM Triangles;
TAN	TAN (expression) Вычисляет тригонометрический тангенс. Этот метод возвращает значение double.	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вычислить тригонометрический тангенс: SELECT TAN(angle) FROM Triangles;
TANH	TANH (expression) Вычисляет гиперболический тангенс. Этот метод возвращает значение double.	• expression — может быть именем столбца, результатом другой функции или математической операции.	Вычислить гиперболический тангенс: SELECT TANH(angle) FROM Triangles;
ATAN2	ATAN2 (y, x) Вычисляет угол при преобразовании прямоугольных координат в полярные координаты. Этот метод возвращает значение double.	• x and y - аргументы.	Вычислить гиперболический тангенс: SELECT ATAN2(X, Y) FROM Triangles;
BITAND	BITAND (y, x) Побитовая операция И. Этот метод возвращает значение long.	• x and y - аргументы.	SELECT BITAND(X, Y) FROM Triangles;
BITGET	BITGET (y, x) Возвращает значение true тогда и только тогда, когда первый параметр имеет бит, установленный в позиции, указанной вторым параметром. Этот метод возвращает логическое значение. Второй параметр имеет нулевой индекс; младший значащий бит имеет позицию 0.	• x and y - аргументы.	Проверить, что 3-й бит равен 1: SELECT BITGET(X, 3) from Triangles;
BITOR	BITOR (y, x)	• x and y - аргументы.	Вычислить ИЛИ между двумя полями:

Функция	Описание	Параметры	Пример
	Побитовая операция ИЛИ. Этот метод возвращает значение long.		SELECT BITGET(X, Y) from Triangles;
BITXOR	BITXOR (y, x) Побитовая операция XOR. Этот метод возвращает значение long.	• x and y - аргументы.	Вычислить XOR между двумя полями: SELECT BITXOR(X, Y) FROM Triangles;
MOD	MOD (y, x) Операция по модулю. Этот метод возвращает значение long.	• x and y - аргументы.	Вычислить модуль между двумя полями: SELECT BITXOR(X, Y) FROM Triangles;
CEILING	CEIL (expression) CEILING (expression) Этот метод возвращает значение double.	• expression — любое допустимое числовое выражение.	Рассчитать максимальную цену для товаров: SELECT item_id, CEILING(price) FROM Items;
DEGREES	DEGREES (expression) Этот метод возвращает значение double.	• expression — любое допустимое числовое выражение.	Преобразовать значение аргумента в градусы: SELECT DEGREES(X) FROM Triangles;
EXP	EXP (expression) Этот метод возвращает значение double.	• expression — любое допустимое числовое выражение.	Вычислить экспоненту: SELECT EXP(X) FROM Triangles;
FLOOR	FLOOR (expression) Этот метод возвращает значение double.	• expression — любое допустимое числовое выражение.	Рассчитать минимальную цену: SELECT FLOOR(X) FROM Items;
LOG	LOG (expression) LN (expression) Этот метод возвращает значение double.	• expression — любое допустимое числовое выражение.	Вычислить логарифм: SELECT LOG(X) from Items;
LOG10	LOG10 (expression) Этот метод возвращает значение double.	• expression — любое допустимое числовое выражение.	Вычислить десятичный логарифм: SELECT LOG(X) FROM Items;
RADIANS	RADIANS (expression) Этот метод возвращает значение double.	• expression — любое допустимое числовое выражение.	Вычислить радианы: SELECT RADIANS(X) FROM Items;
SQRT	SQRT (expression) Этот метод возвращает значение double.	• expression — любое допустимое числовое выражение.	Вычислить квадратный корень: SELECT SQRT(X) FROM Items;
PI	PI (expression) Этот метод возвращает значение double.	• expression — любое допустимое числовое выражение.	Вычислить число ПИ: SELECT PI(X) FROM Items;

Функция	Описание	Параметры	Пример
POWER	POWER (X, Y) Этот метод возвращает значение double.	• expression — любое допустимое числовое выражение.	Вычислить степень числа: SELECT pow(2, n) FROM Rows;
RAND	{RAND | RANDOM} ([expression]) Вызов функции без параметра возвращает следующее псевдослучайное число. Вызов его с параметром запускает генератор случайных чисел сеанса. Этот метод возвращает значение типа double между 0 (включая) и 1 (исключая).	• expression — любое допустимое числовое выражение заполняет генератор случайных чисел сеанса.	Вычислить случайное число для каждой игры: SELECT random() FROM Play;
RANDOM_UUID	{RANDOM_UUID | UUID} () Возвращает новый UUID со 122 псевдослучайными битами.		Вычислить случайный номер для каждого игрока: SELECT UUID(),name FROM Player;
ROUND	ROUND (expression [, precision]) Округляет до определенного количества цифр или до ближайшего long, если количество цифр не задано. Этот метод возвращает числовое значение (того же типа, что и входные данные).	• выражение - любое допустимое числовое выражение. • точность - количество цифр после десятичной дроби для округления до. Округляет до ближайшего значения long, если количество цифр не задано.	Преобразовать возраст каждого игрока в целое число: SELECT name, ROUND(age) FROM Player;
ROUNDMAGIC	ROUNDMAGIC (expression) Эта функция хороша для округления чисел, но она может быть медленной. Он имеет специальную обработку для чисел около 0. Поддерживаются только числа, меньшие или равные +/-1000000000000. Значение внутренне преобразуется в строку, а затем проверяются последние 4 символа. "000 x" становится "0000", а "999x" становится "999999", которое округляется автоматически. Этот метод возвращает значение double.	• expression — любое допустимое числовое выражение.	Округлить возраст каждого игрока: SELECT name, ROUNDMAGIC(AGE/3*3) FROM Player;
SECURE_RAND	SECURE_RAND (int) Генерирует несколько криптографически стойких случайных чисел. Этот метод возвращает байты.	• int - задает количество цифр.	Получить действительно случайное число: SELECT name, SECURE_RAND(10) FROM Player;

Функция	Описание	Параметры	Пример
SIGN	SIGN (expression) Возвращает -1, если значение меньше 0, 0, если равно нулю, и в противном случае 1.	• expression — любое допустимое числовое выражение.	Получить знак для каждого значения: SELECT name, SIGN(VALUE) FROM Player;
ENCRYPT	ENCRYPT (algorithmString , keyBytes , dataBytes) Шифрует данные с помощью ключа. Поддерживаемый алгоритм - AES. Размер блока составляет 16 байт. Этот метод возвращает байты.	• algorithmString - задает поддерживаемый алгоритм AES. • keyBytes - устанавливает ключ. • dataBytes - устанавливает данные.	Зашифровать имя игрока: SELECT ENCRYPT('AES', '00', STRINGTOUTF8(Name)) FROM Player;
DECRYPT	DECRYPT (algorithmString , keyBytes , dataBytes) Расшифровывает данные с помощью ключа. Поддерживаемый алгоритм - AES. Размер блока составляет 16 байт. Этот метод возвращает байты.	• algorithmString - задает поддерживаемый алгоритм AES. • keyBytes - устанавливает ключ. • dataBytes - устанавливает данные.	Расшифровать имена игроков: SELECT DECRYPT('AES', '00', '3fabb4de8f1ee2e97d7793bab2db1116'))) FROM Player;
TRUNCATE	{TRUNC | TRUNCATE} (\{\{numeric, digitsInt} | timestamp | date | timestampString}) Сокращает до нескольких цифр (до следующего значения, более близкого к 0). Этот метод возвращает значение double. При использовании с временной меткой сокращает временную метку до значения даты (day). При использовании с датой сокращает дату до значения даты (дня) за вычетом временной части. При использовании с меткой времени в виде строки сокращает метку времени до значения даты (day).		TRUNCATE(VALUE, 2);
COMPRESS	COMPRESS(dataBytes [, algorithmString]) Сжимает данные, используя указанный алгоритм сжатия. Поддерживаемые алгоритмы: LZF (более быстрое, но более низкое сжатие; по умолчанию) и DEFLATE (более высокое сжатие). Сжатие не всегда уменьшает размер. Очень маленькие объекты и объекты с небольшой избыточностью могут увеличиваться в размерах. Этот метод возвращает байты.	• dataBytes - данные для сжатия. • algorithmString - алгоритм, используемый для сжатия.	COMPRESS(STRINGTOUTF8('Test'))

Функция	Описание	Параметры	Пример
EXPAND	EXPAND(dataBytes) Расшивает данные, которые были сжаты с помощью функции COMPRESS. Этот метод возвращает байты.	• dataBytes - данные для расширения.	UTF8TOSTRING(EXPAND(COMPRESS(STRINGTOUTF8('Test'))))
ZERO	ZERO() Возвращает значение 0. Эту функцию можно использовать, даже если числовые литералы отключены.		ZERO()
Строковые функции			
ASCII	ASCII(string) Возвращает значение ASCII первого символа в строке. Этот метод возвращает значение int.	• string - аргумент.	select ASCII(name) FROM Players;
BIT_LENGTH	BIT_LENGTH(string) Возвращает количество битов в строке. Этот метод возвращает значение long. Для BLOB, CLOB, BYTES и JAVA_OBJECT используется указанная точность объекта. Каждому символу требуется 16 бит.	• string - аргумент.	select BIT_LENGTH(name) FROM Players;
LENGTH	{LENGTH | CHAR_LENGTH | CHARACTER_LENGTH} (string) Возвращает количество символов в строке. Этот метод возвращает значение long. Для BLOB, CLOB, BYTES и JAVA_OBJECT используется указанная точность объекта.	• string - аргумент.	SELECT LENGTH(name) FROM Players;
OCTET_LENGTH	OCTET_LENGTH(string) Возвращает количество байтов в строке. Этот метод возвращает значение long. Для BLOB, CLOB, BYTES и JAVA_OBJECT используется указанная точность объекта. Каждому символу требуется 2 байта.	• string - аргумент.	SELECT OCTET_LENGTH(name) FROM Players;
CHAR	{CHAR | CHR} (int)	• string - аргумент.	SELECT CHAR(65)||name FROM Players;

Функция	Описание	Параметры	Пример
	Возвращает символ, представляющий значение ASCII. Этот метод возвращает строку.		
CONCAT	CONCAT(string, string [,...]) Объединяет строки. В отличие от оператора ||, параметры NULL игнорируются и не приводят к тому, что результат становится нулевым. Этот метод возвращает строку.	• string - аргумент.	SELECT CONCAT(NAME, '!') FROM Players;
CONCAT_WS	CONCAT_WS(separatorString, string, string [,...]) Объединяет строки, разделяя их разделителем. В отличие от оператора ||, параметры NULL игнорируются и не приводят к тому, что результат становится нулевым. Этот метод возвращает строку.	• separatorString - разделитель. • string - аргумент.	SELECT CONCAT_WS(',', NAME, '!') FROM Players;
DIFFERENCE	DIFFERENCE(X, Y) Возвращает разницу между значениями SOUNDEX() двух строк. Этот метод возвращает значение int.	• X, Y - строки для сравнения.	Вычислить разницу в SOUNDEX() для имен двух игроков: select DIFFERENCE(T1.NAME, T2.NAME) FROM players T1, players T2 WHERE T1.ID = 10 AND T2.ID = 11;
HEXTORAW	HEXTORAW(string) Преобразует шестнадцатеричное представление строки в строку. Используется 4 шестнадцатеричных символа на каждый строковый символ.	• string - шестнадцатеричная строка, используемая для преобразования.	Вычислить гармоничность для имен игроков: SELECT HEXTORAW(DATA) FROM Players;
RAWTOHEX	RAWTOHEX(string) Преобразует строку в шестнадцатеричное представление. Используется 4 шестнадцатеричных символа на каждый строковый символ. Этот метод возвращает строку.	• string - строка для преобразования в шестнадцатеричное представление.	Вычислить гармоничность для имен игроков: SELECT RAWTOHEX(DATA) FROM Players;
INSTR	INSTR(string, searchString, [, startInt]) Возвращает расположение строки поиска в строке. Если используется начальная позиция,	• string - любая строка. • searchString - любая строка для поиска. • starting - начальная позиция для поиска.	Проверить, содержит ли строка символ "@": SELECT INSTR(EMAIL,'@') FROM Players;

Функция	Описание	Параметры	Пример
	символы перед ней игнорируются. Если позиция имеет отрицательное значение, возвращается самое правое местоположение. 0 возвращается, если строка поиска не найдена. Эта функция чувствительна к регистру, даже если параметры не учитываются.		
LOWER	{LOWER | LCASE} (string) Преобразует строку в нижний регистр.	• string - аргумент.	SELECT LOWER(NAME) FROM Players;
UPPER	{UPPER | UCASE} (string) Преобразует строку в верхний регистр.	• string - аргумент.	Пример возвращает фамилию в верхнем регистре для каждого игрока: SELECT UPPER(last_name) "LastNameUpperCase" FROM Players;
LEFT	LEFT(string, int) Возвращает крайнее левое количество символов.	• string — аргумент. • int - количество символов для извлечения.	Получить 3 первые буквы имен игроков: SELECT LEFT(NAME, 3) FROM Players;
RIGHT	RIGHT(string, int) Возвращает крайнее правое количество символов.	• string — аргумент. • int - количество символов для извлечения.	Получить 3 последних буквы имен игроков: SELECT RIGHT(NAME, 3) FROM Players;
LOCATE	LOCATE(searchString, string [, startInt]) Возвращает расположение строки поиска в строке. Если используется начальная позиция, символы перед ней игнорируются. Если позиция имеет отрицательное значение, возвращается самое правое местоположение. 0 возвращается, если строка поиска не найдена.		SELECT LOCATE('.', NAME) FROM Players;
POSITION	POSITION(searchString, string) Возвращает расположение строки поиска в строке.		SELECT POSITION('.', NAME) FROM Players;
LPAD	LPAD(string, int[, paddingString]) Слева дополняет строку до указанной длины. Если длина короче строки, она будет сокращена в		SELECT LPAD(AMOUNT, 10, '*') FROM Players;

Функция	Описание	Параметры	Пример
	конце. Если строка заполнения не задана, будут использоваться пробелы.		
RPAD	RPAD(string, int[, paddingString]) Справа дополняет строку до указанной длины. Если длина короче строки, она будет сокращена. Если строка заполнения не задана, будут использоваться пробелы.		SELECT RPAD(TEXT, 10, '-') FROM Players;
LTRIM	LTRIM(string) Удаляет все начальные пробелы из строки.		SELECT LTRIM(NAME) FROM Players;
RTRIM	RTRIM(string) Удаляет все конечные пробелы из строки.		SELECT RTRIM(NAME) FROM Players;
TRIM	TRIM ([{LEADING | TRAILING | BOTH} [string] FROM] string) Удаляет из строки все начальные и конечные пробелы или пробелы на обоих концах. Другие символы также могут быть удалены.		SELECT TRIM(BOTH '_' FROM NAME) FROM Players;
REGEXP_REPLACE	REGEXP_REPLACE(inputString, regexString, replacementString [, flagsString]) Заменяет каждую подстроку, соответствующую регулярному выражению. Если какой-либо параметр равен null (кроме необязательного параметра flagsString), результат равен null.	Значения флагов ограничены «i», «c», «n», «m». Другие символы вызывают исключение. В параметре flagsString можно использовать несколько символов (например: 'im'). Последние флаги переопределяют предыдущие, например, «ic» эквивалентно чувствительному к регистру, соответствующему «c». • 'i' включает сопоставление без учета регистра (Pattern.CASE_INSENSITIVE). • 'c' отключает сопоставление без учета регистра (Pattern.CASE_INSENSITIVE). • 'n' позволяет точке соответствовать символу новой строки (Pattern.DOTALL). • 'm' включает многострочный режим (Pattern.MULTILINE).	SELECT REGEXP_REPLACE(name, 'w+', 'W', 'i') FROM Players;

Функция	Описание	Параметры	Пример
REPEAT	REPEAT(string, int) Возвращает строку, повторяющуюся некоторое количество раз.		SELECT REPEAT(NAME || ' ', 10) FROM Players;
REPLACE	REPLACE(string, searchString [, replacementString]) Заменяет все вхождения строки поиска в указанном тексте другой строкой. Если замена не указана, строка поиска удаляется из исходной строки. Если какой-либо параметр равен null, то результат равен null.		SELECT REPLACE(NAME, ' ') FROM Players;
SOUNDEX	SOUNDEX(string) Возвращает четырехсимвольный код, представляющий SOUNDEX строки. Этот метод возвращает строку.		SELECT SOUNDEX(NAME) FROM Players;
SPACE	SPACE(int) Возвращает строку, состоящую из указанного количества пробелов.		SELECT name, SPACE(80) FROM Players;
STRINGDECODE	STRINGDECODE(string) Преобразует закодированную строку, используя формат кодирования Java string literal. Специальными символами являются \b, \t, \n, \f, \r, \", \, \<octal>, \u<unicode>. Этот метод возвращает строку.		STRINGENCODE(STRINGDECODE('Lines 1\nLine 2'));
STRINGENCODE	STRINGENCODE(string) Кодирует специальные символы в строке, используя формат кодирования строкового литерала Java. Специальными символами являются \b, \t, \n, \f, \r, \", \, \<octal>, \u<unicode>. Этот метод возвращает строку.		STRINGENCODE(STRINGDECODE('Lines 1\nLine 2'))
STRINGTOUTF8	STRINGTOUTF8(string) Кодирует строку в массив байтов, используя формат кодировки UTF8. Этот метод возвращает байты.		SELECT UTF8TOSTRING(STRINGTOUTF8(name)) FROM Players;

Функция	Описание	Параметры	Пример
SUBSTRING	{SUBSTRING | SUBSTR} (string, startInt [, lengthInt]) Возвращает подстроку строки, начиная с указанной позиции. Если начальный индекс отрицательный, то начальный индекс относится к концу строки. Длина не является обязательной. Также поддерживается: SUBSTRING(строка [ОТ начала] [ДЛЯ длины]).		SELECT SUBSTR(name, 2, 5) FROM Players;
UTF8TOSTRING	UTF8TOSTRING(bytes) Декодирует массив байтов в формате UTF8 в строку.		SELECT UTF8TOSTRING(STRINGTOUTF8(name)) FROM Players;
XMLATTR	XMLATTR(nameString, valueString) Создает элемент атрибута XML формы name=value. Значение кодируется в виде XML-текста. Этот метод возвращает строку.		XMLNODE('a', XMLATTR('href', 'http://h2database.com'))
XMLNODE	XMLNODE(elementString [, attributesString [, contentString [, indentBoolean]]]) Создает элемент узла XML. Пустая или нулевая строка атрибута означает, что атрибуты не установлены. Пустая или нулевая строка содержимого означает, что узел пуст. Содержимое имеет отступ по умолчанию, если оно содержит новую строку. Этот метод возвращает строку.		XMLNODE('a', XMLATTR('href', 'http://h2database.com'), 'H2')
XMLCOMMENT	XMLCOMMENT(commentString) Создает XML-комментарий. Два тире (--) преобразуются в - -. Этот метод возвращает строку.		XMLCOMMENT('Test')
XMLCDATA	XMLCDATA(valueString) Создает элемент XML CDATA. Если значение содержит]]>, вместо него создается текстовый элемент XML. Этот метод возвращает строку.		XMLCDATA('data')
XMLSTARTDOC	XMLSTARTDOC()		XMLSTARTDOC()

Функция	Описание	Параметры	Пример
	Возвращает XML-объявление. Результатом всегда является <?xml version=1.0?>.		
XMLTEXT	XMLTEXT(valueString [, escapeNewlineBoolean]) Создает текстовый элемент XML. Если включено, новая строка и перевод строки преобразуются в XML-объект (&#). Этот метод возвращает строку.		XMLSTARTDOC()
TO_CHAR	TO_CHAR(value [, formatString[, nlsParamString]]) Форматирует временную метку, число или текст.		TO_CHAR(TIMESTAMP '2010-01-01 00:00:00', 'DD MON, YYYY')
TRANSLATE	TRANSLATE(value , searchString, replacementString]]) Заменяет последовательность символов в строке другим набором символов.		TRANSLATE('Hello world', 'eo', 'EO')
Функции даты и времени			
CURRENT_DATE	{CURRENT_DATE [()] | CURDATE() | SYSDATE | TODAY} Возвращает текущую дату. При многократном вызове в рамках транзакции эта функция возвращает одно и то же значение.		CURRENT_DATE()
CURRENT_TIME	{CURRENT_TIME [()] | CURTIME()} Возвращает текущее время. При многократном вызове в рамках транзакции эта функция возвращает одно и то же значение.		CURRENT_TIME()
CURRENT_TIMESTAMP	{CURRENT_TIMESTAMP [([int])] | NOW([int])} Возвращает текущую временную метку. Параметр precision для точности в наносекундах является необязательным. Этот метод всегда возвращает одно и то же значение в транзакции.		CURRENT_TIMESTAMP()
DATEADD	{DATEADD| TIMESTAMPADD} (unitString, addIntLong, timestamp) Добавляет единицы (units) к отметке времени. Строка указывает единицу измерения. Нужно		DATEADD('MONTH', 1, DATE '2001-01-31')

Функция	Описание	Параметры	Пример
	использовать отрицательные значения для вычитания единиц. addIntLong может быть длинным значением при работе с миллисекундами, в противном случае его диапазон ограничен значением int. Поддерживаются те же единицы измерения, что и в функции EXTRACT. Метод DATEADD возвращает метку времени. Метод TIMESTAMPADD возвращает тип long.		
DATEDIFF	{DATEDIFF | TIMESTAMPDIFF} (unitString, aTimestamp, bTimestamp) Возвращает количество пересеченных границ единиц между двумя временными метками. Этот метод возвращает значение long. Строка указывает единицу измерения. Поддерживаются те же единицы измерения, что и в функции EXTRACT.		DATEDIFF('YEAR', T1.CREATED, T2.CREATED)
DAYNAME	DAYNAME(date) Возвращает название дня (на английском языке).		DAYNAME(CREATED)
DAY_OF_MONTH	DAY_OF_MONTH(date) Возвращает день месяца (1-31).		DAY_OF_MONTH(CREATED)
DAY_OF_WEEK	DAY_OF_WEEK(date) Возвращает день недели (1 означает воскресенье).		DAY_OF_WEEK(CREATED)
DAY_OF_YEAR	DAY_OF_YEAR(date) Возвращает день года (1-366).		DAY_OF_YEAR(CREATED)
EXTRACT	Возвращает определенное значение из временных меток. Этот метод возвращает значение int.		EXTRACT(SECOND FROM CURRENT_TIMESTAMP)

Функция	Описание	Параметры	Пример
FORMATDATETIME	FORMATDATETIME (timestamp, formatString [,localeString [,timeZoneString]]) Форматирует дату, время или временную метку в виде строки. Наиболее важными символами формата являются: y - год, M - месяц, d - день, H - час, m - минута, s - секунда. Этот метод возвращает строку.		FORMATDATETIME(TIMESTAMP '2001-02-03 04:05:06', 'EEE, d MMM yyyy HH:mm:ss z', 'en', 'GMT')
HOUR	HOUR(timestamp) Возвращает час (0-23) из временной метки.		HOUR(CREATED)
MINUTE	MINUTE(timestamp) Возвращает минуту (0-59) из временной метки.		MINUTE(CREATED)
MONTHNAME	MONTHNAME(date) Возвращает название месяца (на английском языке).		MONTHNAME(CREATED)
PARSEDATETIME	PARSEDATETIME(string, formatString [, localeString [, timeZoneString]]) Анализирует строку и возвращает временную метку. Наиболее важными символами формата являются: y - год, M - месяц, d - день, H - час, m - минута, s - секунда.		PARSEDATETIME('Sat, 3 Feb 2001 03:05:06 GMT', 'EEE, d MMM yyyy HH:mm:ss z', 'en', 'GMT')
QUARTER	QUARTER(timestamp) Возвращает квартал (1-4) из временной метки.		QUARTER(CREATED)
SECOND	SECOND(timestamp) Возвращает секунды (0-59) из метки времени.		SECOND(CREATED)
WEEK	WEEK(timestamp) Возвращает неделю (1-53) из временной метки. Этот метод использует текущую системную локаль.		WEEK(CREATED)
YEAR	YEAR(timestamp) Возвращает год из временной метки.		YEAR(CREATED)

Функция	Описание	Параметры	Пример
Системные функции			
COALESCE	{COALESCE | NVL } (aValue, bValue [,...]) Возвращает первое значение, которое не равно null.		COALESCE(A, B, C)
DECODE	DECODE(value, whenValue, thenValue [,...]) Возвращает первое совпадающее значение. NULL считается соответствующим NULL. Если совпадение найдено не было, то возвращается значение NULL или последний параметр (если количество параметров четное).		DECODE(RAND()>0.5, 0, 'Red', 1, 'Black')
GREATEST	GREATEST(aValue, bValue [,...]) Возвращает наибольшее значение, которое не равно NULL, или NULL, если все значения равны NULL.		GREATEST(1, 2, 3)
IFNULL	IFNULL(aValue, bValue) Возвращает значение 'a', если оно не равно null, в противном случае 'b'.		IFNULL(NULL, '')
LEAST	LEAST(aValue, bValue [,...]) Возвращает наименьшее значение, которое не равно NULL, или NULL, если все значения равны NULL.		LEAST(1, 2, 3)
NULLIF	NULLIF(aValue, bValue) Возвращает NULL, если 'a' равно 'b', в противном случае 'a'.		NULLIF(A, B)
NVL2	NVL2(testValue, aValue, bValue) Если тестовое значение равно null, то возвращается 'b'. В противном случае возвращается 'a'. Типом данных возвращаемого значения является тип данных 'a', если это текстовый тип.		NVL2(X, 'not null', 'null')
CASEWHEN	CASEWHEN (boolean , aValue , bValue)		CASEWHEN(ID=1, 'A', 'B')

Функция	Описание	Параметры	Пример
	Возвращает aValue, если логическое выражение равно true, в противном случае bValue.		
CAST	CAST (value AS dataType) Преобразует значение в другой тип данных. Используются следующие правила преобразования: • При преобразовании числа в логическое значение 0 расценивается как ложное, а любое другое значение - истинным. • При преобразовании логического значения в число значение false равно 0, а значение true равно 1. • При преобразовании числа в число другого типа значение проверяется на переполнение. • При преобразовании числа в двоичный формат количество байтов будет соответствовать точности. • При преобразовании строки в двоичный код она кодируется в шестнадцатеричном формате. • Шестнадцатеричная строка может быть преобразована в двоичную форму, а затем в число. Если прямое преобразование невозможно, значение сначала преобразуется в строку.		CAST(NAME AS INT); CAST(65535 AS BINARY); CAST(CAST('FFFF' AS BINARY) AS INT);
CONVERT	CONVERT (value , dataType) Преобразует значение в другой тип данных.		CONVERT(NAME, INT)
TABLE	TABLE | TABLE_DISTINCT (name dataType = expression) Возвращает результирующий набор. TABLE_DISTINCT удаляет повторяющиеся строки.		SELECT * FROM TABLE(ID INT=(1, 2), NAME VARCHAR=('Hello', 'World'))

18 ПРИЛОЖЕНИЕ Ж. ОПЕРАЦИИ REST API

РЕД КВАНТ предоставляет клиент HTTP REST, который может взаимодействовать с кластером по протоколам HTTP и HTTPS, используя подход REST. REST API можно использовать для выполнения различных операций, таких как чтение/запись из/в кэш, выполнение задач, получение различных метрик и многое другое.

Внутри РЕД КВАНТ использует Jetty для предоставления функций HTTP-сервера. Подробную информацию о настройке jetty см. в разделе Конфигурация.

Чтобы включить HTTP-соединение, необходимо убедиться, что модуль ignite-rest-http включен. Если используется бинарный дистрибутив, нужно скопировать модуль ignite-rest-http из IGNITE_HOME/libs/Optional/ в папку IGNITE_HOME/libs.

Явная настройка не требуется; коннектор запускается автоматически и прослушивает порт 8080. Можно проверить, работает ли он с помощью curl:

```
curl 'http://localhost:8080/ignite?cmd=version'
```

Параметры запроса могут быть предоставлены либо как часть URL, либо в виде данных формы:

```
curl 'http://localhost:8080/ignite?cmd=put&cacheName=myCache' -X POST -H 'Content-Type: application/x-www-form-urlencoded' -d 'key=testKey&val=testValue'
```

18.1 Конфигурация

Параметры HTTP-сервера можно изменить следующим образом:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration" id="ignite.cfg">

  <property name="connectorConfiguration">
    <bean class="org.apache.ignite.configuration.ConnectorConfiguration">
      <property name="jettyPath" value="jetty.xml"/>
    </bean>
  </property>
</bean>
```

В следующей таблице 35 описаны свойства ConnectorConfiguration, связанные с http-сервером.

Таблица 35 - Свойства ConnectorConfiguration

Имя параметра	Описание	Необязательное	Значение по умолчанию
setSecretKey(String)	Определяет секретный ключ, используемый для аутентификации клиента. При использовании клиентский запрос должен содержать HTTP-заголовок X-Signature со строкой «[1]:[2]», где [1] - временная метка в миллисекундах, а [2] - Base64 хэша секретного ключа в кодировке SHA1.	Yes	null

Имя параметра	Описание	Необязательное	Значение по умолчанию
setPortRange(int)	Диапазон портов для сервера Jetty. Если порт, указанный в конфигурации Jetty или в системном свойстве IGNITE_JETTY_PORT, уже используется, РЕД КВАНТ итеративно увеличивает порт на 1 и пытается выполнить привязку еще раз, пока не будет превышен указанный диапазон портов.	Yes	100
setJettyPath(String)	Путь к файлу конфигурации Jetty. Должен быть либо абсолютным, либо относительным к IGNITE_HOME. Если путь не указан, РЕД КВАНТ запускает сервер Jetty с простым HTTP-коннектором. Этот коннектор использует системные свойства IGNITE_JETTY_HOST и IGNITE_JETTY_PORT в качестве хоста и порта соответственно. Если IGNITE_JETTY_HOST не указан, по умолчанию используется localhost. Если IGNITE_JETTY_PORT не указан, используется порт 8080.	Yes	null
setMessageInterceptor(…)	Перехватчик преобразует все объекты, которыми обмениваются по протоколу REST. Например, если используется пользовательская сериализация на клиенте, можно написать перехватчик для преобразования двоичных представлений, полученных от клиента, в объекты Java, а затем получить к ним прямой доступ из кода Java.	Yes	null

18.2 Безопасность

Если в кластере настроена проверка подлинности, все приложения, использующие REST API, запрашивают проверку подлинности, предоставляя учетные данные безопасности. Запрос проверки подлинности возвращает токен сеанса, который можно использовать с любой командой в этом сеансе.

Запросить авторизацию можно двумя способами:

1. Можно воспользоваться командой аутентификации с параметрами ignite.login=[user]&ignite.password=[password].

```
https://[host]:[port]/ignite?cmd=authenticate&ignite.login=[user]&ignite.password=[password]
```

2. Можно воспользоваться любой командой REST с параметрами ignite.login=[user]&ignite.password=[password] в пути строки подключения. В примере ниже используется команда версии:

```
http://[host]:[port]/ignite?cmd=version&ignite.login=[user]&ignite.password=[password]
```

В обоих приведенных выше примерах необходимо заменить [host], [port], [user] и [password] фактическими значениями.

Выполнение любой из приведенных выше строк в браузере возвращает ответ с токеном сеанса, который выглядит следующим образом:

```
{"successStatus":0,"error":null,"sessionToken":"EF6013FF590348CE91DEAE9870183BEF","response":true}
```

После получения токена сеанса нужно использовать параметр sessionToken со строкой подключения, как показано в примере ниже:

```
http://[host]:[port]/ignite?cmd=top&sessionToken=[sessionToken]
```

В приведенной выше строке подключения нужно заменить [host], [port] и [sessionToken] фактическими значениями.

18.3 Типы данных

По умолчанию REST API обменивается параметрами запроса в формате String. Кластер работает с параметрами как со строковыми объектами.

Если тип параметра отличается от String, можно использовать keyType или valueType, чтобы указать реальный тип аргумента. REST API поддерживает как типы Java, так и пользовательские типы.

В таблице 36 перечислены типы Java.

Таблица 36 - Типы Java

Тип ключа/значения REST	Соответствующий тип Java
boolean	java.lang.Boolean
byte	java.lang.Byte
short	java.lang.Short
integer	java.lang.Integer
long	java.lang.Long
float	java.lang.Float
double	java.lang.Double
date	java.sql.Date Пример: 2018-01-01
time	java.sql.Time

Тип ключа/значения REST	Соответствующий тип Java
	Пример: 01:01:01
timestamp	java.sql.Timestamp Пример: 2018-02-18%2001:01:01
uuid	java.util.UUID
IgniteUuid	org.apache.ignite.lang.IgniteUuid

В следующем примере показана команда put с keyType=int и valueType=date:

```
http://[host]:[port]/ignite?cmd=put&key=1&val=2018-01-
01&cacheName=myCache&keyType=int&valueType=date
```

Команда get с keyType=int и valueType=date будет выглядеть так:

```
http://[host]:[port]/ignite?cmd=get&key=1&cacheName=myCache&keyType=int&valueType=date
```

Формат JSON используется для обмена сложными пользовательскими объектами по протоколу РЕД КВАНТ REST.

Например, есть класс Person, а ниже представлено JSON-представление экземпляра объекта, который нужно отправить в кластер:

```
{

 "uid": "7e51118b",
 "name": "John Doe",
 "orgId": 5678901,
 "married": false,
 "salary": 156.1
 }
```

Затем нужно использовать этот запрос REST, чтобы поместить объект в кластер, задав для параметра valueType значение Person, а для параметра val — значение объекта JSON:

```
http://[host]:[port]/ignite?cacheName=testCache&cmd=put&keyType=int&key=1&valueType=Person

&val=%7B%0A+++++%22uid%22%3A+%227e51118b%22%2C%0A+++++%22name%22%3A+%22John+Doe%22%
2C%0A+++++%22orgId%22%3A+5678901%2C%0A+++++%22married%22%3A+false%2C%0A+++++%22salary%22%3A+
156.1%0A++%7D&
```

Как только сервер получает запрос, он преобразует объект из JSON во внутренний формат двоичного объекта, следуя приведенной ниже процедуре преобразования:

- Если класс Person существует и доступен в пути к классам сервера, объект JSON преобразуется в экземпляр класса Person.
- Если класс Person недоступен в пути к классам сервера, но есть объект QueryEntity, определяющий Person, то объект JSON преобразуется в двоичный объект этого типа Person;
- В противном случае типы полей объекта JSON преобразуются в соответствии с обычным соглашением JSON.

18.4 Справочник по REST API

18.4.1 Версия

Возвращает версию РЕД КВАНТ.

Запрос:

```
http://host:port/ignite?cmd=version
```

Ответ:

```
{
 "error": "",
 "response": "1.0.0",
 "successStatus": 0
}
```

18.4.2 Состояние кластера

Возвращает текущее состояние кластера.

Запрос:

```
http://host:port/ignite?cmd=state
```

Ответ:

Возвращает true, если кластер активен. Возвращает false, если кластер неактивен.

```
{
 "successStatus":0,
 "error":null,
 "sessionToken":null,
 "response": "ACTIVE_READ_ONLY"
}
```

18.4.3 Изменить состояние кластера

Команда setstate изменяет состояние кластера.

Запрос:

```
http://host:port/ignite?cmd=setstate&state={new_state}
```

Параметры:

- state — тип String. Новое состояние кластера. Одно из значений:
  - ACTIVE: активное состояние,
  - ACTIVE_READ_ONLY: состояние только для чтения,
  - INACTIVE: кластер деактивирован.

Внимание! Деактивация освобождает все ресурсы памяти, включая данные приложения, на всех узлах кластера и отключает общедоступный API кластера. Если есть кэши в памяти, резервные копии которых не поддерживаются постоянным хранилищем (ни

собственным постоянным хранилищем, ни внешним хранилищем), данные будут утеряны и нужно будет повторно заполнить эти кэши. Также очищаются не персистентные системные кэши.

Ответ:


```
{
 "successStatus":0,
 "error":null,
 "sessionToken":null,
 "response":"setstate done"
}
```


18.4.4 Инкремент

Добавляет и получает текущее значение заданного атомарного long.

Запрос:

```
http://host:port/ignite?cmd=incr&cacheName={cacheName}&key={incrKey}&init={initialValue}&delta={delta}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Обозначает имя атомарного long.

• init — необязательный параметр типа long. Обозначает значение при инициализации.

• delta — параметр типа long. Обозначает число, которое будет прибавлено.

Ответ:

Ответ содержит значение после операции.


```
{
 "affinityNodeId": "e05839d5-6648-43e7-a23b-78d7db9390d5",
 "error": "",
 "response": 42,
 "successStatus": 0
}
```


18.4.5 Декремент

Вычитает и получает текущее значение заданного атомарного long.

Запрос:

```
http://host:port/ignite?cmd=decr&cacheName={cacheName}&key={key}&init={init_value}&delta={delta}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Обозначает имя атомарного long.

• init — необязательный параметр типа long. Обозначает значение при инициализации.

• delta — параметр типа long. Обозначает число, которое нужно вычесть.

Ответ:

Ответ содержит значение после операции.


```
{
 "affinityNodeId": "e05839d5-6648-43e7-a23b-78d7db9390d5",
 "error": "",
 "response": -42,
 "successStatus": 0
}
```


18.4.6 Метрики кэша

Показывает метрики для кэша.

Запрос:

```
http://host:port/ignite?cmd=cache&cacheName={cacheName}&destId={nodeId}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "",
 "error": "",
 "response": {
  "hits": 0,
  "misses": 0,
  "reads": 0,
  "writes": 2
 },
 "successStatus": 0
}
```


Параметр response с типом jsonObject обозначает объект JSON, который содержит метрики кэша, такие как время создания, количество прочтений и т. д.

18.4.7 Размер кэша

Получает количество всех записей, кэшированных на всех узлах.

Запрос:

```
http://host:port/ignite?cmd=size&cacheName={cacheName}
```

Ответ:

```
{
 "affinityNodeId": "",
 "error": "",
 "response": 1,
 "successStatus": 0
}
```


18.4.8 МЕТАДАННЫЕ КЭША

Получает метаданные для кэша.

Запрос:

```
http://host:port/ignite?cmd=metadata&cacheName={cacheName}
```

Ответ:


```
{
 "error": "",
 "response": {
  "cacheName": "partitionedCache",
  "types": [
   "Person"
  ],
  "keyClasses": {
   "Person": "java.lang.Integer"
  },
  "valClasses": {
   "Person": "org.apache.ignite.Person"
  },
  "fields": {
   "Person": {
    "_KEY": "java.lang.Integer",
    "_VAL": "org.apache.ignite.Person",
    "ID": "java.lang.Integer",
    "FIRSTNAME": "java.lang.String",
    "LASTNAME": "java.lang.String",
    "SALARY": "double"
   }
  },
  "indexes": {
   "Person": [
    {
     "name": "ID_IDX",
     "fields": [
      "id"
     ],
     "descendings": [],
     "unique": false
    },
    {
     "name": "SALARY_IDX",
     "fields": [
      "salary"
     ],
     "descendings": [],
     "unique": false
```

```
      }
     ]
    }
   },
   "sessionToken": "",
   "successStatus": 0
}
```


18.4.9 Сравнение и обмен

Сохраняет заданную пару «ключ-значение» в кэше только в том случае, если предыдущее значение равно переданному ожидаемому значению.

Запрос:

https://[host]:[port]/ignite?cmd=authenticate&ignite.login=[user]&ignite.password=[password]

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Обозначает ключ для хранения в кэше.

• val — параметр типа string. Значение, связанное с данным ключом.

• val2 — параметр типа string. Обозначает ожидаемое значение.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:

Ответ возвращает true, если значение было заменено, и false в противном случае.


```
{
  "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
  "error": "",
  "response": true,
  "successStatus": 0
}
```


18.4.10 Добавление

Добавляет строку для значения, связанного с ключом.

Запрос:

http://host:port/ignite?cmd=append&key={appendKey}&val={_suffix}&cacheName={cacheName}&destId={nodeId}

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Обозначает ключ для хранения в кэше.

• val — параметр типа string. Значение, которое будет добавлено к текущему значению.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
 "error": "",
 "response": true,
 "successStatus": 0
}
```


18.4.11 Добавление в начало

Добавляет префикс к значению, связанному с данным ключом.

Запрос:

```
http://host:port/ignite?cmd=prepend&key={key}&val={value}&cacheName={cacheName}&destId={nodeId}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Обозначает ключ для хранения в кэше.

• val — параметр типа string. Обозначает строку, которая будет добавлена к текущему значению.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
 "error": "",
 "response": true,
 "successStatus": 0
}
```


18.4.12 Замена

Сохраняет заданную пару «ключ-значение» в кэше, если кэш уже содержит ключ.

Запрос:

```
http://host:port/ignite?cmd=rep&key=repKey&val=newValue&cacheName={cacheName}&destId={nodeId}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Обозначает ключ для хранения в кэше.

• val — параметр типа string. Значение, связанное с данным ключом.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

• exp — необязательный параметр типа long. Обозначает время истечения в миллисекундах для записи. Когда параметр установлен, операция выполняется с ModifiedExpiryPolicy.

Ответ:

Ответ содержит true, если значение было заменено, и false в противном случае.


```
{
 "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
 "error": "",
 "response": true,
 "successStatus": 0
}
```


18.4.13 Получение

Извлекает значение, сопоставленное с указанным ключом, из кэша.

Запрос:

```
http://host:port/ignite?cmd=get&key={getKey}&cacheName={cacheName}&destId={nodeId}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Обозначает ключ для хранения в кэше.

• keyType — необязательный параметр встроенного типа Java. Дополнительные сведения см. в пункте Типы данных.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

18.4.14 Получить все

Извлекает значения, сопоставленные с указанными ключами, из заданного кэша.

Запрос:

```
http://host:port/ignite?cmd=getall&k1={getKey1}&k2={getKey2}&k3={getKey3}&cacheName={cacheName}&destId={nodeId}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• k1…kN — параметр типа string. Ключи, связанные значения которых должны быть возвращены.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "",
 "error": "",
 "response": {
  "key1": "value1",
  "key2": "value2"
 },
 "successStatus": 0
}
```


18.4.15 Получить и удалить

Удаляет из кэша соответствующее данному ключу значение и возвращает предыдущее значение.

Запрос:

```
http://host:port/ignite?cmd=getrmv&cacheName={cacheName}&destId={nodeId}&key={key}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Ключ, соответствующее значение которого должно быть удалено из кэша.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
 "error": "",
 "response": value,
 "successStatus": 0
}
```


18.4.16 Извлечь и поместить

Сохраняет заданную пару «ключ-значение» в кэше и возвращает существующее значение, если таковое имеется.

Запрос:

```
http://host:port/ignite?cmd=getput&key=getKey&val=newVal&cacheName={cacheName}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Ключ, который будет связан со значением.

• val — параметр типа string. Значение, которое должно быть связано с ключом.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:

Ответ содержит предыдущее значение ключа.


```
{
 "affinityNodeId": "2bd7b049-3fa0-4c44-9a6d-b5c7a597ce37",
 "error": "",
 "response": {"name": "bob"},
 "successStatus": 0
}
```


18.4.17 Извлечь и поместить, если отсутствует

Сохраняет заданную пару «ключ-значение» в кэше, только если в кэше не было предыдущего сопоставления для нее. Если кэш ранее содержал значение для данного ключа, то возвращается это значение.

Запрос:

```
http://host:port/ignite?cmd=getputifabs&key=getKey&val=newVal&cacheName={cacheName}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Ключ, который будет связан со значением.

• val — параметр типа string. Значение, которое должно быть связано с ключом.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "2bd7b049-3fa0-4c44-9a6d-b5c7a597ce37",
 "error": "",
 "response": "value",
 "successStatus": 0
}
```

18.4.18 Получить и заменить

Сохраняет заданную пару «ключ-значение» в кэше, только если для нее существует предыдущее сопоставление.

Запрос:

http://host:port/ignite?cmd=getrep&key={key}&val={val}&cacheName={cacheName}&destId={nodeId}

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Ключ для хранения в кэше.

• val — параметр типа string. Значение, связанное с данным ключом.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:

Ответ содержит предыдущее значение, связанное с указанным ключом.

```
{
 "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
 "error": "",
 "response": oldValue,
 "successStatus": 0
}
```

18.4.19 Заменить значение

Заменяет запись для ключа, только если в текущий момент он сопоставлен с заданным значением.

Запрос:

http://host:port/ignite?cmd=repval&key={key}&val={newValue}&val2={oldVal}&cacheName={cacheName}&destId={nodeId}

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Ключ для хранения в кэше.

• val — параметр типа string. Значение, связанное с данным ключом.

• val2 — параметр типа string. Значение, которое, как ожидается, будет связано с указанным ключом.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
 "error": "",
 "response": true,
 "successStatus": 0
}
```


18.4.20 Удаление

Удаляет сопоставление по данному ключу из кэша.

Запрос:

```
http://host:port/ignite?cmd=rmv&key={rmvKey}&cacheName={cacheName}&destId={nodeId}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Ключ, для которого необходимо удалить сопоставление из кэша.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
 "error": "",
 "response": true,
 "successStatus": 0
}
```


18.4.21 Удалить все

Удаляет сопоставления данного ключа из кэша.

Запрос:

```
http://host:port/ignite?cmd=rmvall&k1={rmKey1}&k2={rmKey2}&k3={rmKey3}&cacheName={cacheName}&destId={nodeId}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• k1…kN — параметр типа string. Ключи, сопоставления которых должны быть удалены из кэша.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
 "error": "",
 "response": true,
 "successStatus": 0
}
```


18.4.22 Удалить значение

Удаляет сопоставление для ключа только в том случае, если в данный момент сопоставлено заданному значению.

Запрос:

```
http://host:port/ignite?cmd=rmvval&key={rmvKey}&val={rmvVal}&cacheName={cacheName}&destId={nodeId}
```

Параметры:

- cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.
- key — параметр типа string. Ключ, сопоставление которого должно быть удалено из кэша.
- val — параметр типа string. Значение, которое, как ожидается, будет связано с указанным ключом.
- destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
 "error": "",
 "response": true,
 "successStatus": 0
}
```


18.4.23 Добавление

Сохраняет заданную пару «ключ-значение» в кэше, если в кэше нет ключа.

Запрос:

```
http://host:port/ignite?cmd=add&key=newKey&val=newValue&cacheName={cacheName}&destId={nodeId}
```

Параметры:

- cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.
- key — параметр типа string. Ключ, связанный со значением.
- val — параметр типа string. Значение, связанное с данным ключом.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

• exp — необязательный параметр типа long. Обозначает время истечения в миллисекундах для записи. Когда параметр установлен, операция выполняется с ModifiedExpiryPolicy.

Ответ:


```
{
 "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
 "error": "",
 "response": true,
 "successStatus": 0
}
```


18.4.24 Поместить

Сохраняет заданную пару «ключ-значение» в кэше.

Запрос:

```
http://host:port/ignite?cmd=put&key=newKey&val=newValue&cacheName={cacheName}&destId={nodeId}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Ключ, связанный со значением.

• val — параметр типа string. Значение, связанное с данным ключом.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

• exp — необязательный параметр типа long. Обозначает время истечения в миллисекундах для записи. Когда параметр установлен, операция выполняется с ModifiedExpiryPolicy.

Ответ:


```
{
 "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
 "error": "",
 "response": true,
 "successStatus": 0
}
```


18.4.25 Поместить все

Сохраняет заданные пары «ключ-значение» в кэше.

Запрос:

http://host:port/ignite?cmd=putall&k1={putKey1}&k2={putKey2}&k3={putKey3}&v1={value1}&v2={value2}&v3={value3}&cacheName={cacheName}&destId={nodeId}

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• k1…kN — параметр типа string. Ключ, связанный со значением.

• v1…vN — параметр типа string. Значение, связанное с данным ключом.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "1bcbac4b-3517-43ee-98d0-874b103ecf30",
 "error": "",
 "response": true,
 "successStatus": 0
}
```


18.4.26 ПОМЕСТИТЬ, ЕСЛИ ОТСУТСТВУЕТ

Сохраняет заданную пару «ключ-значение» в кэше, если в кэше нет данного ключа.

Запрос:

http://host:port/ignite?cmd=putifabs&key={getKey}&val={newVal}&cacheName={cacheName}

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Ключ, связанный со значением.

• val — параметр типа string. Значение, связанное с данным ключом.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

• exp — необязательный параметр типа long. Обозначает время истечения в миллисекундах для записи. Когда параметр установлен, операция выполняется с ModifiedExpiryPolicy.

Ответ:

Поле ответа содержит значение true, если запись была помещена, и значение false в противном случае.


```
{
 "affinityNodeId": "2bd7b049-3fa0-4c44-9a6d-b5c7a597ce37",
 "error": "",
```

```
 "response": true,
 "successStatus": 0
}
```


18.4.27 СОДЕРЖИТ КЛЮЧ

Определяет, содержит ли кэш запись для указанного ключа.

Запрос:

```
http://host:port/ignite?cmd=conkey&key={getKey}&cacheName={cacheName}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• key — параметр типа string. Ключ, присутствие которого в этом кэше должно быть проверено.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "2bd7b049-3fa0-4c44-9a6d-b5c7a597ce37",
 "error": "",
 "response": true,
 "successStatus": 0
}
```


18.4.28 СОДЕРЖИТ КЛЮЧИ

Определяет, содержит ли кэш какие-либо записи для указанных ключей.

Запрос:

```
http://host:port/ignite?cmd=conkeys&k1={getKey1}&k2={getKey2}&cacheName={cacheName}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• k1…kN — параметр типа string. Ключ, присутствие которого в этом кэше должно быть проверено.

• destId — необязательный параметр типа string. Обозначает идентификатор узла, для которого должны быть возвращены метрики.

Ответ:


```
{
 "affinityNodeId": "2bd7b049-3fa0-4c44-9a6d-b5c7a597ce37",
 "error": "",
 "response": true,
```

```
 "successStatus": 0
}
```


18.4.29 Получить или создать кэш

Создает кэш с заданным именем, если он не существует.

Запрос:

```
http://host:port/ignite?cmd=getorcreate&cacheName={cacheName}
```

Параметры:

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• backups — необязательный параметр типа int. Количество резервных копий кэша. По умолчанию 0.

• dataRegion — необязательный параметр типа string. Имя области данных, которой должен принадлежать кэш.

• templateName — необязательный параметр типа string. Имя шаблона кэша, зарегистрированного в РЕД КВАНТ, для использования в качестве конфигурации распределенного кэша.

• cacheGroup — необязательный параметр типа string. Имя группы, к которой должен принадлежать кэш.

• writeSynchronizationMode — необязательный параметр типа string. Устанавливает режим синхронизации записи для данного кэша:

- FULL_SYNC
- FULL_ASYNC
- PRIMARY_SYNC.

Ответ:


```
{
 "error": "",
 "response": null,
 "successStatus": 0
}
```


18.4.30 Уничтожить кэш

Уничтожает кэш с заданным именем.

Запрос:

```
http://host:port/ignite?cmd=destcache&cacheName={cacheName}
```

Параметр cacheName — необязательный, типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

Ответ:


```
{
  "error": "",
  "response": null,
  "successStatus": 0
}
```


18.4.31 Узел

Извлекает информацию об узле.

Запрос:

http://host:port/ignite?cmd=node&attr={includeAttributes}&mtr={includeMetrics}&id={nodeId}&caches={includeCaches}

Параметры:

• mtr — необязательный параметр типа boolean. Ответ включает метрики, если этот параметр имеет значение true.

• attr — необязательный параметр типа boolean. Ответ включает атрибуты, если этот параметр имеет значение true.

• ip — параметр типа string. Этот параметр является необязательным, если передается параметр id. Ответ возвращается для узла, у которого есть IP.

• id — параметр типа string. Этот параметр является необязательным, если передается параметр ip. Ответ возвращается для узла, который имеет идентификатор узла.

• caches — необязательный параметр типа boolean. Если установлено значение true, информация кэша, возвращаемая узлом, включает: имя, режим и схему SQL. Если установлено значение false, команда node не возвращает никакой информации о кэше. Значение по умолчанию — true.

Ответ:


```
{
  "error": "",
  "response": {
    "attributes": null,
    "caches": {},
    "consistentId": "127.0.0.1:47500",
    "defaultCacheMode": "REPLICATED",
    "metrics": null,
    "nodeId": "2d0d6510-6fed-4fa3-b813-20f83ac4a1a9",
    "replicaCount": 128,
    "tcpAddresses": ["127.0.0.1"],
    "tcpHostNames": [""],
    "tcpPort": 11211
  },
  "successStatus": 0
}
```

18.4.32 LOG

Показывает логи сервера.

Запрос:

http://host:port/ignite?cmd=log&from={from}&to={to}&path={pathToLogFile}

Параметры:

• from — необязательный параметр типа integer. Номер строки, с которой нужно начать. Параметр является обязательным, если передается to.

• path — необязательный параметр типа string. Путь к файлу журнала. Если не указан, используется значение по умолчанию.

• to — необязательный параметр типа integer. Номер строки, на которой нужно закончить. Параметр является обязательным, если передается from.

Ответ:


```
{
  "error": "",
  "response": ["[14:01:56,626][INFO ][test-runner][GridDiscoveryManager] Topology snapshot [ver=1, nodes=1, CPUs=8, heap=1.8GB]"],
  "successStatus": 0
}
```


18.4.33 Топология

Получает информацию о топологии кластера.

Запрос:

http://host:port/ignite?cmd=top&attr=true&mtr=true&id=c981d2a1-878b-4c67-96f6-70f93a4cd241

Параметры:

• mtr — необязательный параметр типа boolean. Ответ включает метрики, если этот параметр имеет значение true.

• attr — необязательный параметр типа boolean. Ответ включает атрибуты, если этот параметр имеет значение true.

• ip — параметр типа string. Этот параметр является необязательным, если передается параметр id. Ответ возвращается для узла, у которого есть IP.

• id — параметр типа string. Этот параметр является необязательным, если передается параметр ip. Ответ возвращается для узла, который имеет идентификатор узла.

Ответ:


```
{
  "error": "",
  "response": [
    {
      "attributes": {
```

```
      ...
    },
    "caches": [
     {
      name: "",
      mode: "PARTITIONED"
     },
     {
      name: "partitionedCache",
      mode: "PARTITIONED",
      sqlSchema: "partitionedCache"
     }
    ],
    "consistentId": "127.0.0.1:47500",
    "metrics": {
     ...
    },
    "nodeId": "96baebd6-dedc-4a68-84fd-f804ee1ed995",
    "replicaCount": 128,
    "tcpAddresses": ["127.0.0.1"],
    "tcpHostNames": [""],
    "tcpPort": 11211
  },
  {
   "attributes": {
    ...
   },
           "caches": [
    {
     name: "",
     mode: "REPLICATED"
    }
   ],
   "consistentId": "127.0.0.1:47501",
   "metrics": {
    ...
   },
   "nodeId": "2bd7b049-3fa0-4c44-9a6d-b5c7a597ce37",
   "replicaCount": 128,
   "tcpAddresses": ["127.0.0.1"],
   "tcpHostNames": [""],
   "tcpPort": 11212
  }
 ],
 "successStatus": 0
}
```


18.4.34 Выполнить задачу

Выполняет заданную задачу в кластере.

Запрос:

```
http://host:port/ignite?cmd=exe&name=taskName&p1=param1&p2=param2&async=true
```

Параметры:

• name — параметр типа string. Название задачи, которую необходимо выполнить.

• p1…pN — необязательный параметр типа string. Аргумент выполняемой задачи.

• async — необязательный параметр типа boolean. Определяет, выполняется ли задача асинхронно.

Ответ:

Ответ содержит сообщение об ошибке, уникальный идентификатор задачи, статус и результат вычисления.


```
{
 "error": "",
 "response": {
  "error": "",
  "finished": true,
  "id": "~ee2d1688-2605-4613-8a57-6615a8cbcd1b",
  "result": 4
 },
 "successStatus": 0
}
```


18.4.35 Результат задачи

Возвращает результат вычисления для данной задачи.

Запрос:

```
http://host:port/ignite?cmd=res&id={taskId}
```

Параметр id типа string обозначает ID задачи, результат которой необходимо вернуть.

Ответ:

Ответ содержит информацию об ошибках (если есть), идентификатор задачи, статус и результат вычислений.


```
{
 "error": "",
 "response": {
  "error": "",
  "finished": true,
  "id": "69ad0c48941-4689aae0-6b0e-4d52-8758-ce8fe26f497d~4689aae0-6b0e-4d52-8758-ce8fe26f497d",
  "result": 4
 },
 "successStatus": 0
}
```


18.4.36 Выполнение SQL-запроса

Запускает SQL-запрос для кэша.

Запрос:

```
http://host:port/ignite?cmd=qryexe&type={type}&pageSize={pageSize}&cacheName={cacheName}&arg1=1000&arg2=2000&qry={query}
```

Параметры:

• type — параметр типа string. Обозначает тип запрос.

• pageSize — параметр типа number. Обозначает размер страницы для запроса.

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• arg1…argN — параметр типа string. Обозначают аргументы запроса.

• qry — параметр типа string. Кодирование sql-запроса.

• keepBinary — необязательный параметр типа boolean. Нельзя десериализовать двоичные объекты, по умолчанию false.

Ответ:

Объект ответа содержит элементы, возвращенные запросом, флаг, указывающий последнюю страницу, и queryId.


```
{
 "error":"",
 "response":{
  "fieldsMetadata":[],
  "items":[
   {"key":3,"value":{"name":"Jane","id":3,"salary":2000}},
   {"key":0,"value":{"name":"John","id":0,"salary":2000}}],
  "last":true,
  "queryId":0},
 "successStatus":0
}
```


18.4.37 Выполнение запроса полей SQL

Запускает запрос полей SQL для кэша.

Запрос:

```
http://host:port/ignite?cmd=qryfldexe&pageSize=10&cacheName={cacheName}&qry={qry}
```

Параметры:

• pageSize — параметр типа number. Обозначает размер страницы для запроса.

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• arg1…argN — параметр типа string. Обозначают аргументы запроса.

• qry — параметр типа string. Кодирование sql-запроса.

• keepBinary — необязательный параметр типа boolean. Нельзя десериализовать двоичные объекты, по умолчанию false.

Ответ:

Объект ответа содержит элементы, возвращенные запросом, метаданные полей запроса, флаг, указывающий последнюю страницу, и queryId.

```
{
 "error": "",
 "response": {
  "fieldsMetadata": [
   {
    "fieldName": "FIRSTNAME",
    "fieldTypeName": "java.lang.String",
    "schemaName": "person",
    "typeName": "PERSON"
   },
   {
    "fieldName": "LASTNAME",
    "fieldTypeName": "java.lang.String",
    "schemaName": "person",
    "typeName": "PERSON"
   }
  ],
  "items": [["Jane", "Doe" ], ["John", "Doe"]],
  "last": true,
  "queryId": 0
 },
 "successStatus": 0
}
```


18.4.38 Выполнение SQL запроса сканирования (Scan Query)

Выполняет запрос на сканирование кэша.

Запрос:

http://host:port/ignite?cmd=qryscanexe&pageSize={pageSize}&cacheName={cacheName}&className={className}

Параметры:

• pageSize — параметр типа number. Обозначает размер страницы для запроса.

• cacheName — необязательный параметр типа string. Обозначает имя кэша. Если он не указан, используется кэш по умолчанию.

• className — необязательный параметр типа string. Имя класса предикатов для запроса сканирования. Класс должен реализовывать интерфейс IgniteBiPredicate.

• keepBinary — необязательный параметр типа boolean. Нельзя десериализовать двоичные объекты, по умолчанию false.

Ответ:

Объект ответа содержит элементы, возвращенные запросом сканирования, метаданные полей запроса, флаг, указывающий последнюю страницу, и queryId.


```
{
 "error": "",
 "response": {
  "fieldsMetadata": [
   {
```

```
      "fieldName": "key",
      "fieldTypeName": "",
      "schemaName": "",
      "typeName": ""
     },
     {
      "fieldName": "value",
      "fieldTypeName": "",
      "schemaName": "",
      "typeName": ""
     }
    ],
    "items": [
     {
      "key": 1,
      "value": {
       "firstName": "Jane",
       "id": 1,
       "lastName": "Doe",
       "salary": 1000
      }
     },
     {
      "key": 3,
      "value": {
       "firstName": "Jane",
       "id": 3,
       "lastName": "Smith",
       "salary": 2000
      }
     }
    ],
    "last": true,
    "queryId": 0
   },
   "successStatus": 0
}
```


18.4.39 Выборка результатов (Fetch) SQL-запроса

Возвращает следующую страницу для запроса.

Запрос:

```
http://host:port/ignite?cmd=qryfetch&pageSize={pageSize}&qryId={queryId}
```

Параметры:

• pageSize — параметр типа number. Обозначает размер страницы для запроса.

• qryId — параметр типа number. Обозначает идентификатор запроса, возвращаемый из выполнения SQL-запроса (Sql query execute), выполнения запроса полей SQL (sql fields query execute) или SQL команды fetch.

Ответ:

Объект ответа содержит элементы, возвращенные запросом, флаг, указывающий последнюю страницу, и queryId.

```
{

 "error":"",
 "response":{
  "fieldsMetadata":[],
  "items":[["Jane","Doe"],["John","Doe"]],
  "last":true,
  "queryId":0
 },
 "successStatus":0
}
```


18.4.40 Закрытие SQL-запроса

Закрывает ресурсы запроса.

Запрос:

```
http://host:port/ignite?cmd=qrycls&qryId={queryId}
```

Параметр qryId типа number обозначает идентификатор запроса, возвращаемый из выполнения SQL-запроса (Sql query execute), выполнения запроса полей SQL (sql fields query execute) или SQL команды fetch.

Ответ:

Команда возвращает «true», если запрос был успешно закрыт.


```
{

 "error":"",
 "response":true,
 "successStatus":0
}
```

19 ПРИЛОЖЕНИЕ 3. ОСНОВНЫЕ МЕТРИКИ

19.1 Система

Системные метрики, такие как метрики JVM или ЦП, представлены в таблице 37.

Регистрационное имя: sys.

Таблица 37 - Системные метрики

Наименование	Тип	Описание
CpuLoad	double	Загрузка процессора.
CurrentThreadCpuTime	long	ThreadMXBean.getCurrentThreadCpuTime()
CurrentThreadUserTime	long	ThreadMXBean.getCurrentThreadUserTime()
DaemonThreadCount	integer	ThreadMXBean.getDaemonThreadCount()
GcCpuLoad	double	Загрузка процессора сборщиком мусора (GC).
PeakThreadCount	integer	ThreadMXBean.getPeakThreadCount
SystemLoadAverage	java.lang.Double	OperatingSystemMXBean.getSystemLoadAverage()
ThreadCount	integer	ThreadMXBean.getThreadCount
TotalExecutedTasks	long	Общее количество выполненных задач.
TotalStartedThreadCount	long	ThreadMXBean.getTotalStartedThreadCount
UpTime	long	RuntimeMxBean.getUptime()
memory.heap.committed	long	MemoryUsage.getHeapMemoryUsage().getCommitted()
memory.heap.init	long	MemoryUsage.getHeapMemoryUsage().getInit()
memory.heap.used	long	MemoryUsage.getHeapMemoryUsage().getUsed()
memory.nonheap.committed	long	MemoryUsage.getNonHeapMemoryUsage().getCommitted()
memory.nonheap.init	long	MemoryUsage.getNonHeapMemoryUsage().getInit()
memory.nonheap.max	long	MemoryUsage.getNonHeapMemoryUsage().getMax()
memory.nonheap.used	long	MemoryUsage.getNonHeapMemoryUsage().getUsed()

19.2 Кэш

Метрики кэша представлены в таблице 38.

Регистрационное имя: cache.{cache_name}.{near}

Таблица 38 - Метрики кэша

Наименование	Тип	Описание
CacheEvictions	long	Общее количество исключений из кэша.
CacheGets	long	Общее количество попаданий в кэш.
CacheHits	long	Количество запросов на получение, которые были удовлетворены кэшем.
CacheMisses	long	Промах — это неудовлетворенный запрос на получение.
CachePuts	long	Общее количество размещений в кэше.
CacheRemovals	long	Общее количество удалений из кэша.
CacheTxCommits	long	Общее количество совершенных транзакций.
CacheTxRollbacks	long	Общее количество откатов транзакций.
CacheSize	long	Размер локального кэша.
CommitTime	histogram	Время коммита в наносекундах.
CommitTimeTotal	long	Общее время коммитов в наносекундах.
EntryProcessorHits	long	Общее количество обращений к ключам, которые существуют в кэше.
EntryProcessorInvokeTimeNanos	long	Общее время вызовов кэша, инициатором которых является этот узел, в наносекундах.
EntryProcessorMaxInvocationTime	long	На данный момент максимальное время выполнения вызовов кэша, для которых этот узел является инициатором.
EntryProcessorMinInvocationTime	long	На данный момент минимальное время выполнения вызовов кэша, для которых этот узел является инициатором.
EntryProcessorMisses	long	Общее количество обращений к ключам, которых не существует в кэше.
EntryProcessorPuts	long	Общее количество вызовов кэша, вызвавших обновление.
EntryProcessorReadOnlyInvocations	long	Общее количество вызовов кэша, не вызвавших никаких обновлений.
EntryProcessorRemovals	long	Общее количество вызовов кэша, вызвавших удаления.
EstimatedRebalancingKeys	long	Предполагаемое количество ключей для ребалансировки.
GetAllTime	histogram	Время всех получений, в течение которого этот узел является инициатором, в наносекундах.
GetTime	histogram	Время получения , в течение которого этот узел является инициатором, в наносекундах.
GetTimeTotal	long	Общее время получения из кэша, инициатором которого является этот узел, в наносекундах.

Наименование	Тип	Описание
HeapEntriesCount	long	Количество записей в heap.
IndexRebuildKeysProcessed	long	Количество ключей с перестроенными индексами.
IsIndexRebuildInProgress	boolean	True, если выполняется сборка или перестроение индекса.
OffHeapBackupEntriesCount	long	Счетчик записей резервного копирования off-heap.
OffHeapEntriesCount	long	Количество записей off-heap.
OffHeapEvictions	long	Общее количество исключений из памяти off-heap.
OffHeapGets	long	Общее количество запросов на получение к памяти off-heap.
OffHeapHits	long	Количество запросов на получение, которые были удовлетворены памятью off-heap.
OffHeapMisses	long	Промах - это запрос на получение, который не удовлетворяется памятью off-heap.
OffHeapPrimaryEntriesCount	long	Количество первичных записей off-heap.
OffHeapPuts	long	Общее количество запросов на размещение в памяти off-heap.
OffHeapRemovals	long	Общее количество удалений из памяти off-heap.
PutAllTime	histogram	PutAll время, для которого этот узел является инициатором, в наносекундах.
PutTime	histogram	Время помещения, в течение которого этот узел является инициатором, в наносекундах.
PutTimeTotal	long	Общее время помещения кэша, инициатором которого является этот узел, в наносекундах.
QueryCompleted	long	Количество завершенных запросов.
QueryExecuted	long	Количество выполненных запросов.
QueryFailed	long	Количество неудачных запросов.
QueryMaximumTime	long	Максимальное время выполнения запроса.
QueryMinimalTime	long	Минимальное время выполнения запроса.
QuerySumTime	long	Общее время запроса
RebalanceClearingPartitionsLeft	long	Количество разделов, которые необходимо очистить перед фактическим началом ребалансировки.
RebalanceStartTime	long	Время начала ребалансировки.
RebalancedKeys	long	Количество уже ребалансированных ключей.
RebalancingBytesRate	long	Расчетная скорость ребалансировки в байтах.

Наименование	Тип	Описание
RebalancingKeysRate	long	Расчетная скорость ребалансировки в ключах.
RemoveAllTime	histogram	Все время удаления, в течение которого этот узел является инициатором, в наносекундах.
RemoveTime	histogram	Время удаления, в течение которого этот узел является инициатором, в наносекундах.
RemoveTimeTotal	long	Общее время удаления из кэша в наносекундах.
RollbackTime	histogram	Время отката в наносекундах.
RollbackTimeTotal	long	Общее время отката в наносекундах.
TotalRebalancedBytes	long	Количество уже ребалансированных байтов.

19.3 Группы кэша

Метрики групп кэша представлены в таблице 39.

Регистрационное имя: cacheGroups.{group_name}

Таблица 39 - Метрики групп кэша

Наименование	Тип	Описание
AffinityPartitionsAssignmentMap	java.util.Map	Словарь связей affinity-разделов.
Caches	java.util.ArrayList	Список кэшей.
IndexBuildCountPartitionsLeft	long	Количество разделов, которые необходимо обработать для создания или перестройки готовых индексов.
LocalNodeMovingPartitionsCount	integer	Количество разделов с состоянием MOVING для этой группы кэшей, расположенных на этом узле.
LocalNodeOwningPartitionsCount	integer	Количество разделов с состоянием OWNING для этой группы кэшей, расположенных на этом узле.
LocalNodeRentingEntriesCount	long	Количество записей, оставшихся для исключения в разделах RENTING, расположенных на этом узле для этой группы кэша.
LocalNodeRentingPartitionsCount	integer	Количество разделов в состоянии RENTING для этой группы кэшей, расположенных на этом узле.
MaximumNumberOfPartitionCopies	integer	Максимальное количество копий раздела для всех разделов этой группы кэша.
MinimumNumberOfPartitionCopies	integer	Минимальное количество копий раздела для всех разделов этой группы кэша.

Наименование	Тип	Описание
MovingPartitionsAllocationMap	java.util.Map	Словарь распределения разделов с состоянием MOVING в кластере.
OwningPartitionsAllocationMap	java.util.Map	Словарь распределения разделов со статусом OWNING в кластере.
PartitionIds	java.util.ArrayList	Идентификаторы локальных разделов.
SparseStorageSize	long	Место для хранения, выделенное для группы с поправкой на возможную разреженность, в байтах.
StorageSize	long	Место для хранения, выделенное для группы, в байтах.
TotalAllocatedPages	long	Общее количество выделенных страниц в группе кэша.
TotalAllocatedSize	long	Общий размер памяти, выделенной для группы, в байтах.
ReencryptionBytesLeft	long	Количество байтов, оставшихся для повторного шифрования.
ReencryptionFinished	boolean	Флаг указывает, завершено повторное шифрование или нет.

19.4 Транзакции

Метрики транзакций представлены в таблице 40.

Регистрационное имя: tx

Таблица 40 - Метрики транзакций

Наименование	Тип	Описание
AllOwnerTransactions	java.util.HashMap	Словарь транзакций, принадлежащих локальному узлу.
LockedKeysNumber	long	Количество ключей, заблокированных на узле.
OwnerTransactionsNumber	long	Количество активных транзакций, инициатором которых является данный узел.
TransactionsHoldingLockNumber	long	Количество активных транзакций, содержащих по крайней мере одну блокировку ключа.
LastCommitTime	long	Время последнего коммита.

Наименование	Тип	Описание
nodeSystemTimeHistogram	histogram	Системное время транзакций на узле, представленное в виде гистограммы.
nodeUserTimeHistogram	histogram	Пользовательское время транзакций на узле, представленное в виде гистограммы.
LastRollbackTime	long	Время последнего отката.
totalNodeSystemTime	long	Общее системное время транзакций на узле.
totalNodeUserTime	long	Общее время транзакций пользователя на узле.
txCommits	integer	Количество совершенных транзакций.
txRollbacks	integer	Количество откатов транзакций.

19.5 Обмен словарями разделов (PME)

Метрики обмена словарями разделов представлены в таблице 41.

Регистрационное имя: pme

Таблица 41 - Метрики обмена картами разделов

Наименование	Тип	Описание
CacheOperationsBlockedDuration	long	Длительность блокировки текущих PME операций кэша в миллисекундах.
CacheOperationsBlockedDurationHistogram	histogram	Гистограмма длительностей блокировки операций кэша PME в миллисекундах.
Duration	long	Текущая длительность PME в миллисекундах.
DurationHistogram	histogram	Гистограмма длительностей PME в миллисекундах.

19.6 Вычислительные job

Метрики вычислительных заданий представлены в таблице 42.

Регистрационное имя: Compute.jobs

Таблица 42 - Метрики вычислительных заданий

Наименование	Тип	Описание
compute.jobs.Active	long	Количество активных job, выполняемых в данный момент.

compute.jobs.Canceled	long	Количество отмененных job, которые все еще выполняются.
compute.jobs.ExecutionTime	long	Общее время выполнения job.
compute.jobs.Finished	long	Количество выполненных job.
compute.jobs.Rejected	long	Количество job, отклоненных после последней операции разрешения коллизий.
compute.jobs.Started	long	Количество запущенных job.
compute.jobs.Waiting	long	Количество текущих ожидающих выполнения job в очереди.
compute.jobs.WaitingTime	long	Общее время, проведенное job в очереди ожидания.

19.7 Пулы потоков

Метрики пула потоков представлены в таблице 43.

Регистрационное имя: threadPools.{thread_pool_name}

Таблица 43 - Метрики пула потоков

Наименование	Тип	Описание
ActiveCount	long	Приблизительное количество потоков, активно выполняющих задачи.
CompletedTaskCount	long	Приблизительное общее количество задач, которые завершили выполнение.
CorePoolSize	long	Основное количество потоков.
KeepAliveTime	long	Время жизни потока, в течение которого он бездействует, ожидая появления места в основном пуле, прежде чем будет уничтожен.
LargestPoolSize	long	Наибольшее количество потоков, которые когда-либо одновременно находились в пуле.
MaximumPoolSize	long	Максимально допустимое количество потоков.
PoolSize	long	Текущее количество потоков в пуле.
QueueSize	long	Текущий размер очереди выполнения.
RejectedExecutionHandlerClass	string	Имя класса текущего обработчика отказа.
Shutdown	boolean	True, если этот исполнитель был закрыт.

Наименование	Тип	Описание
TaskCount	long	Приблизительное общее количество задач, запланированных для выполнения.
TaskExecutionTime	histogram	Время выполнения задачи в миллисекундах.
Terminated	boolean	True, если все задачи завершены после завершения работы.
Terminating	long	True, если завершается, но еще не завершено.
ThreadFactoryClass	string	Имя класса фабрики потоков, используемой для создания новых потоков.

19.8 I/O ДЛЯ ГУППЫ КЭШЕЙ

Метрики кэш-группы ввода-вывода представлены в таблице 44.

Регистрационное имя: io.statistics.cacheGroups.{group_name}

Таблица 44 - Метрики кэш-группы ввода-вывода

Наименование	Тип	Описание
LOGICAL_READS	long	Количество логических операций чтения.
PHYSICAL_READS	long	Количество физических считываний.
grpId	integer	Идентификатор группы.
name	string	Название индекса.
startTime	long	Время начала сбора статистики.

19.9 Отсортированные индексы

Метрики отсортированных индексов представлены в таблице 45.

Имя регистра: io.statistics.sortedIndexes.{cache_name}.{index_name}

Таблица 45 - Метки отсортированных индексов

Наименование	Тип	Описание
LOGICAL_READS_INNER	long	Количество логических операций чтения для внутреннего узла дерева.
LOGICAL_READS_LEAF	long	Количество логических операций чтения для конечного узла дерева.
PHYSICAL_READS_INNER	long	Количество физических операций чтения для внутреннего узла дерева.
PHYSICAL_READS_LEAF	long	Количество физических операций чтения для конечного узла дерева.
indexName	string	Название индекса.
name	string	Имя кэша.
startTime	long	Время начала сбора статистики.

19.10 Хэш-индексы

Метрики хэш-индексов представлены в таблице 46.

Регистрационное имя: io.statistics.hashIndexes.{cache_name}.{index_name}

Таблица 46 - Метрики хэш-индексов

Наименование	Тип	Описание
LOGICAL_READS_INNER	long	Количество логических операций чтения для внутреннего узла дерева.
LOGICAL_READS_LEAF	long	Количество логических операций чтения для конечного узла дерева.

Наименование	Тип	Описание
PHYSICAL_READS_INNER	long	Количество физических операций чтения для внутреннего узла дерева.
PHYSICAL_READS_LEAF	long	Количество физических чтений для конечного узла дерева.
indexName	string	Название индекса.
name	string	Имя кэша.
startTime	long	Время начала сбора статистики.

19.11 I/O для коммуникации

Метрики коммуникационного ввода-вывода представлены в таблице 47.

Регистрационное имя: io.communication

Таблица 47 - Метрики коммуникационного ввода-вывода

Наименование	Тип	Описание
ActiveSessionsCount	integer	Количество активных TCP-сеансов.
OutboundMessagesQueueSize	integer	Размер очереди исходящих сообщений.
SentMessagesCount	integer	Количество отправленных сообщений.
SentBytesCount	long	Количество отправленных байтов.
ReceivedBytesCount	long	Количество полученных байтов.
ReceivedMessagesCount	integer	Количество полученных сообщений.
RejectedSslSessionsCount	integer	Количество сессий TCP, которые были отклонены из-за ошибок SSL (метрика экспортируется, только если SSL включен).
SslEnabled	boolean	Указывает, включен ли SSL.
SslHandshakeDurationHistogram	histogram	Гистограмма длительности установки соединения SSL в миллисекундах (метрика экспортируется, только если SSL включен).

19.12 РЕД КВАНТ коннектор для тонких клиентов

Метрики РЕД КВАНТ коннектора для тонких клиентов представлены в таблице 48.

Регистрационное имя: client.connector

Таблица 48 - Метрики РЕД КВАНТ коннектора для тонких клиентов

Наименование	Тип	Описание
ActiveSessionsCount	integer	Количество активных TCP-сеансов.
ReceivedBytesCount	long	Количество полученных байтов.
RejectedSslSessionsCount	integer	Подсчитываются сеансы TCP, которые были отклонены из-за ошибок SSL (показатель экспортируется только в том случае, если включен SSL).
RejectedSessionsTimeout	integer	Подсчитываются сеансы TCP, которые были отклонены из-за тайм-аута установки соединения.
RejectedSessionsAuthenticationFailed	integer	Подсчитываются сеансы TCP, которые были отклонены из-за неудачной аутентификации.
RejectedSessionsTotal	integer	Общее количество отклоненных TCP-подключений.
{clientType}.AcceptedSessions	integer	Количество успешно установленных сеансов для типа клиента.
{clientType}.ActiveSessions	integer	Количество активных сеансов для типа клиента.
SentBytesCount	long	Количество отправленных байтов.
SslEnabled	boolean	Указывает, включен ли SSL.
SslHandshakeDurationHistogram	histogram	Гистограмма продолжительности установки соединения SSL в миллисекундах (показатель экспортируется только в том случае, если включен SSL).

19.13 Коннектор клиента РЕД КВАНТ REST

Метрики REST-клиентского коннектора РЕД КВАНТ представлены в таблице 49.

Регистрационное имя: rest.client

Таблица 49 - Метрики REST-клиентского коннектора РЕД КВАНТ

Наименование	Тип	Описание
ActiveSessionsCount	integer	Количество активных TCP-сеансов.
ReceivedBytesCount	long	Количество полученных байтов.
RejectedSslSessionsCount	integer	Количество сессий TCP, которые были отклонены из-за ошибок SSL (метрика экспортируется, только если SSL включен).
SentBytesCount	long	Количество отправленных байтов.
SslEnabled	boolean	Указывает, включен ли SSL.
SslHandshakeDurationHistogram	histogram	Гистограмма длительности установки соединения SSL в миллисекундах (метрика экспортируется, только если SSL включен).

19.14 I/O ДЛЯ ОБНАРУЖЕНИЯ

Метрики обнаружения ввода-вывода представлены в таблице 50.

Регистрационное имя: io.discovery

Таблица 50 - Метрики обнаружения ввода-вывода

Наименование	Тип	Описание
CoordinatorSince	long	Временная метка, с которой локальный узел стал координатором (метрика экспортируется только с серверных узлов).
Coordinator	UUID	Идентификатор координатора (метрика экспортируется только с серверных узлов).
CurrentTopologyVersion	long	Текущая версия топологии.
JoinedNodes	integer	Количество присоединившихся узлов.
LeftNodes	integer	Количество узлов, покинувших кластер.
MessageWorkerQueueSize	integer	Текущий размер очереди обработки сообщений.
PendingMessagesRegistered	integer	Счетчик ожидающих зарегистрированных сообщений.
RejectedSslConnectionsCount	integer	Счетчик TCP-соединений обнаружения, которые были отклонены из-за ошибок SSL.
SslEnabled	boolean	Указывает, включен ли SSL.

Наименование	Тип	Описание
TotalProcessedMessages	integer	Общее количество обработанных сообщений.
TotalReceivedMessages	integer	Общее количество полученных сообщений.

19.15 I/O РЕГИОНА ДАННЫХ

Метрики области ввода-вывода данных представлены в таблице 51.

Регистрационное имя: io.dataregion.{data_region_name}

Таблица 51 - Метрики области ввода-вывода данных

Наименование	Тип	Описание
AllocationRate	long	Скорость распределения (страниц в секунду), усредненная по rateTimeInternal.
CheckpointBufferSize	long	Размер буфера checkpoint в байтах.
DirtyPages	long	Количество страниц в памяти, еще не синхронизированных с постоянным хранилищем.
EmptyDataPages	long	Вычисляет количество пустых страниц данных для области. Учитывает только полностью свободные страницы, которые можно использовать повторно (например, страницы, содержащиеся в корзине повторного использования списка свободных).
EvictionRate	long	Скорость вытеснения (страниц в секунду).
LargeEntriesPagesCount	long	Количество страниц, полностью занятых крупными объектами, превышающими размер страницы.
OffHeapSize	long	Размер off-heap в байтах.
OffheapUsedSize	long	Используемый размер off-heap в байтах.
PagesFillFactor	double	Процент используемого пространства.
PagesRead	long	Количество страниц, прочитанных с момента последнего перезапуска.
PagesReplaceAge	long	Средний период, по истечении которого страницы в памяти заменяются страницами из постоянного хранилища (миллисекунды).

Наименование	Тип	Описание
PagesReplaceRate	long	Скорость, с которой страницы в памяти заменяются страницами из постоянного хранилища (страниц в секунду).
PagesReplaced	long	Количество страниц, замененных с момента последнего перезапуска.
PagesWritten	long	Количество страниц, записанных с момента последнего перезапуска.
PhysicalMemoryPages	long	Количество страниц, находящихся в физической оперативной памяти.
PhysicalMemorySize	long	Возвращает общий размер страниц, загруженных в оперативную память, в байтах.
TotalAllocatedPages	long	Общее количество выделенных страниц.
TotalAllocatedSize	long	Получает общий размер памяти, выделенной в области данных, в байтах.
TotalThrottlingTime	long	Общее время регулирования троттлинга в миллисекундах. РЕД КВАНТ запускает троттлинг потоков, которые генерируют грязные страницы во время текущего checkpoint.
UsedCheckpointBufferSize	long	Получает используемый размер буфера checkpoint в байтах.

19.16 Хранилище данных

Метрики хранения данных представлены в таблице 52.

Регистрационное имя: io.datastorage

Таблица 52 - Метрики хранения данных

Наименование	Тип	Описание
CheckpointBeforeLockHistogram	histogram	Гистограмма действия checkpoint перед блокировкой записи в миллисекундах.
CheckpointFsyncHistogram	histogram	Гистограмма длительности синхронизации checkpoint в миллисекундах.
CheckpointHistogram	histogram	Гистограмма продолжительности checkpoint в миллисекундах.
CheckpointListenersExecuteHistogram	histogram	Гистограмма слушателей выполнения checkpoint при длительности блокировки записи в миллисекундах.

Наименование	Тип	Описание
CheckpointLockHoldHistogram	histogram	Гистограмма продолжительности удержания блокировки checkpoint в миллисекундах.
CheckpointLockWaitHistogram	histogram	Гистограмма времени ожидания блокировки checkpoint в миллисекундах.
CheckpointMarkHistogram	histogram	Гистограмма продолжительности отметки checkpoint в миллисекундах.
CheckpointPagesWriteHistogram	histogram	Гистограмма продолжительности записи страниц checkpoint в миллисекундах.
CheckpointSplitAndSortPagesHistogram	histogram	Гистограмма продолжительности разделения и сортировки страниц checkpoint в миллисекундах.
CheckpointTotalTime	long	Общая длительность checkpoint.
CheckpointWalRecordFsyncHistogram	histogram	Гистограмма длительности синхронизации WAL fsync после логирования ChTotalNodeseckpointRecord в начале checkpoint в миллисекундах.
CheckpointWriteEntryHistogram	histogram	Гистограмма длительности записи буфера записей в файл в миллисекундах.
LastCheckpointBeforeLockDuration	long	Продолжительность действия checkpoint перед блокировкой записи в миллисекундах.
LastCheckpointCopiedOnWritePagesNumber	long	Количество страниц, скопированных во временный буфер checkpoint во время последней checkpoint.
LastCheckpointDataPagesNumber	long	Общее количество страниц данных, записанных во время последней checkpoint.
LastCheckpointDuration	long	Длительность последней checkpoint в миллисекундах.
LastCheckpointFsyncDuration	long	Продолжительность фазы синхронизации последней checkpoint в миллисекундах.
LastCheckpointListenersExecuteDuration	long	Продолжительность блокировки записи слушателей выполнения checkpoint при блокировке записи в миллисекундах.
LastCheckpointLockHoldDuration	long	Продолжительность удержания блокировки checkpoint в миллисекундах.

Наименование	Тип	Описание
LastCheckpointLockWaitDuration	long	Продолжительность ожидания блокировки checkpoint в миллисекундах.
LastCheckpointMarkDuration	long	Длительность checkpoint в миллисекундах.
LastCheckpointPagesWriteDuration	long	Продолжительность записи страниц checkpoint в миллисекундах.
LastCheckpointTotalPagesNumber	long	Общее количество страниц, написанных во время последней checkpoint.
LastCheckpointSplitAndSortPagesDuration	long	Продолжительность разделения и сортировки контрольных страниц последней checkpoint в миллисекундах.
LastCheckpointStart	long	Начальная временная метка последней checkpoint.
LastCheckpointWalRecordFsyncDuration	long	Продолжительность WAL fsync после записи CheckpointRecord в начале последней checkpoint в миллисекундах.
LastCheckpointWriteEntryDuration	long	Длительность записи буфера ввода в файл последней checkpoint в миллисекундах.
SparseStorageSize	long	Выделенное пространство для хранения с поправкой на возможную разреженность в байтах.
StorageSize	long	Выделенное место для хранения в байтах.
WalArchiveSegments	integer	Текущее количество сегментов WAL в архиве WAL.
WalBuffPollSpinsRate	long	Число циклов опроса буфера WAL за последний интервал времени.
WalFsyncTimeDuration	long	Общая продолжительность fsync
WalFsyncTimeNum	long	Общее количество fsync
WalLastRollOverTime	long	Время смены последнего сегмента WAL.
WalLoggingRate	long	Среднее количество записей WAL в секунду, записанных за последний интервал времени.
WalTotalSize	long	Общий размер файлов хранения в байтах.
WalWritingRate	long	Среднее количество байтов в секунду, записанных за последний временной интервал.

19.17 Кластер

Метрики кластера представлены в таблице 53.

Имя регистра: cluster

Таблица 53 - Метрики кластера

Наименование	Тип	Описание
ActiveBaselineNodes	integer	Количество активных базовых узлов.
Rebalanced	boolean	True, если кластер полностью достиг состояния ребалансировки. Следует отметить, что неактивный кластер всегда имеет эту метрику в значении False независимо от реального состояния разделов.
TotalBaselineNodes	integer	Общее количество базовых узлов.
TotalClientNodes	integer	Количество клиентских узлов.
TotalServerNodes	integer	Количество серверных узлов.

19.18 Кэш-процессор

Метрики кэш-процессора представлены в таблице .

Имя регистра: cache

Таблица 54 - Метрики кэш-процессора

Наименование	Тип	Описание
LastDataVer	long	Последняя версия данных на узле.
DataVersionClusterId	integer	Идентификатор кластера версии данных.