Для примера возьмем базу данных компании среднего размера, которая занимается продажами товаров через Web. В любой момент времени на сайте компании может быть до пятидесяти посетителей. При этом для отображения одной страницы требуется выполнение одного или нескольких запросов. Например, web-приложение должно на одной странице собрать информацию о содержимом Корзины клиента, и еще получить информацию о просматриваемом товаре. Иногда два пользователя могут одновременно выполнять запросы, и система должна обслуживать их независимо друг от друга. Подобное одновременное выполнение называется параллелизмом.

Если выполнение параллельных процессов не контролировать, это может привести к искажению данных. Когда два или более потоков пытаются записать информацию в один и тот же файл, результат будет непредсказуем. Решить проблему можно путем наложения ограничения на количество потоков, одновременно обращающихся к файлу. Web-приложения, которые хранят свои данные в обычных файлах, при обращении к файлу блокируют его с помощью системных вызовов.
В РСУБД механизм блокировки организован на более абстрактном уровне, в виде транзакций. При этом транзакцией называется последовательность SQL-инструкций, выполняемая полностью или не выполняемая вообще. Сервер изолирует одновременные потоки друг от друга, ограничивая тем самым их доступ к измененным данным. По мере завершения транзакций сервер выполняет их так, будто они происходят последовательно, а не параллельно.
Транзакции являются относительно новым понятием в MySQL. Их можно применять лишь к таблицам определенных типов, но в то же время в MySQL поддерживаются табличные блокировки и другие функциональные возможности, позволяющие имитировать транзакции.

Параллельные запросы

Параллелизм — это одна из сложнейших проблем для СУБД. MySQL является многопоточной программой, поэтому должна справляться с множественными запросами на подключение. Но, помимо проблемы планирования, существует еще и проблема одновременного доступа к данным.
Представим себе двух агентов по продажам, пытающихся выяснить количество единиц одного и того же товара на складе. Первый работающий агент вводит инструкцию SELECT и определяет, что на складе осталось 150 единиц продукции. Второй менеджер получает те же самые данные. Первый агент оформляет покупку 30 единиц товара, поэтому он вводит инструкцию UPDATE, устанавливая объем запасов в 120 единиц. Тем временем второй агент, ничего не зная о работе первого, также оформляет продажу еще 10 единиц товара. Казалось бы, логично предположить, что количество товара на складе должно быть равно 110. Однако второй агент реально увеличил количество товара на складе на 20, и теперь информация в БД является неправильной.
В программировании проблема параллельного доступа решается с помощью механизма блокировок. Существует центральный системный сервис, контролирующий работу

потоков и управляющий блокировкой ресурсов. Если какой-то поток захватил ресурс, поставив на него блокировку, остальные потоки, пытающиеся обратиться к этому же ресурсу, будут ждать его освобождения. Блокирование ресурсов ведет к ощутимому снижению производительности, поэтому блокировки осуществляются на разных уровнях, что позволяет точнее определить область действия допустимых и недопустимых операций.
В СУБД MySQL блокировки реализуются без прямого вмешательства со стороны пользователя. Одиночные запросы выполняются в атомарном режиме, в котором каждый запрос представляет собой отдельно взятую транзакцию. Тогда описанную проблему несогласованного заказа можно решить, объединив несколько операций в одном запросе. Например:

```
UPDATE ITEM
SET INVENT = INVENT — 30
WHERE ID = 3
```

Конечно, не всякую последовательность операций можно выразить в виде одного запроса — для обновления двух таблиц необходимо два запроса. В таких случаях нужно явно указывать начало и конец каждой транзакции.

Транзакции

Транзакция — это совокупность одной или нескольких SQL-инструкций, имеющих начало и конец. В конце транзакции происходит либо ее отмена, либо подтверждение. Отмена транзакции называется откатом (rollback), так как идет последовательная отмена всех сделанных изменений. Подтверждение транзакции называется фиксацией (commit).
Считается, что правильная транзакция обладает следующими свойствами: атомарностью, согласованностью, изолированностью и устойчивостью. Под атомарностью понимается принцип неделимости: все инструкции, составляющие транзакцию, обязательно выполняются все вместе или не выполняется ни одна из них. Никаких промежуточных состояний не существует. База данных поддерживает внутреннюю согласованность за счет сериализации транзакций. Несмотря на кажущуюся одновременность событий, результаты транзакций заносятся в базу данных последовательно. Все транзакции изолируются друг от друга, а изменения, вызываемые каждой из них, становятся доступными лишь после завершения транзакции. Свойство устойчивости означает, что при успешном завершении транзакции в базу данных вносятся постоянные, неотменяемые изменения. Устойчивая база данных способна выдержать внезапную аварию, например, сбой питания, и при этом остаться внутренне согласованной.

Транзакции реализуются путем ведения журнала всех изменений, вносимых в базу данных в ходе каждой транзакции. При откате СУБД сверяется с журналом и отменяет все проведенные инструкции, вложенные в откатываемую транзакцию. Журнал также позволяет легко восстановить согласованное состояние базы данных в случае непредвиденного сбоя. Опять же, ведение журнала транзакций заставляет систему работать медленнее, поэтому в MySQL для типа таблиц по умолчанию MyISAM

транзакции не поддерживаются. Есть в этом и свои минусы, но главным доводом в пользу такой политики разработки является высокая скорость работы программы. Изначально MySQL работала без механизма транзакций. Транзакции появились сравнительно недавно, и только для таблиц расширенного типа: InnoDB, Berkeley DB, Gemini. Но прежде чем переходить на использование таблиц этих типов и отказываться от MyISAM, надо отметить, что применение транзакций не всегда оправдано. Во многих случаях табличных блокировок бывает вполне достаточно. Так что, в отличие от других СУБД, MySQL предоставляет возможность выбора: работать с более медленными таблицами, поддерживающими транзакции, или с более быстрыми, в которых транзакции не используются.

В крупных банковских базах данных транзакции, несомненно, необходимы. В подобных системах множество параллельных потоков могут соперничать за право доступа к информационным ресурсам, и потеря данных при сбое обойдется очень дорого. С другой стороны, информация с web-форумов не требует столь тщательного управления, так как стоимость сообщения, вообще говоря, невелика. Правда, некоторые форумы в Сети по активности своих участников могут быть сравнимы, наверное, со средней банковской системой, разве что дохода это приносит не так много. Впрочем, приобретение дополнительного оборудования для обработки транзакций в этом случае видится малооправданным.
Для Internet-магазина, как и в банке, также требуется целостность базы данных, только интенсивность пользователей здесь значительно ниже. Таким образом, для магазина можно воспользоваться и табличными блокировками. Когда программа создает заказ, она блокирует некоторое количество таблиц, и в это время никто из посетителей магазина также не сможет оформить заказ. Но для небольшого магазина, пусть даже имеющего около 100 заказов в день, весьма маловероятно, что два клиента будут оформлять заказ в один и тот же момент времени. Но при росте активности посетителей и их количества производительность системы будет падать. Также стоит учитывать, что нет нужды блокировать таблицы, ответственные за навигацию по каталогу предлагаемых товаров, при процедуре оформления заказа. Клиентам не придется ожидать разблокирования таблиц, если система была правильно спроектирована.

Уровни изоляции

Теоретически СУБД должна обеспечивать полную изоляцию транзакций. На практике же вводится несколько уровней изоляции, самым высоким из которых является полная изоляция. В СУБД MySQL выбор уровня осуществляется с помощью инструкции SET TRANSACTION. Остановимся на ней более подробно.
В стандарте SQL определены четыре уровня изоляции: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ и SERIALIZABLE. Они перечислены в порядке увеличения изолированности и по очереди решают три основные проблемы, возникающие при использовании транзакций: появление промежуточных данных, появление несогласованных данных и появление строк-призраков.
Промежуточные данные появляются при модификации записи одной транзакцией и одновременном чтении этой записи другой транзакцией. Если первая транзакция будет отменена, то получится, что вторая получила данные, которые "никогда" не

существовали. Именно такое поведение проявляется на уровне транзакций READ UNCOMMITTED.
В режиме READ COMMITTED транзакциям разрешено читать только подтвержденные данные. Однако это также не решает проблему несогласованных данных.
Предположим, в контексте первой транзакции исполняется запрос на определение числа записей в таблице. По завершении этого запроса во второй транзакции проводится удаление и/или добавление записей в таблицу. Если теперь первая транзакция заново выполнит запрос на количество записей в таблице, результат будет отличаться от первоначально полученного значения.
Несогласованность данных исключается на уровне REPEATABLE READ. В этом режиме строки, к которым обращается транзакция для чтения или записи, блокируются. Но здесь возникают проблемы со строками-призраками. Транзакция может заблокировать все записи, с которыми идет работа, но другая транзакция в это время может добавить строки в таблицу. Поэтому, когда между двумя чтениями в одной транзакции другая добавляет строки, возникают так называемые "строки-призраки", так как она внезапно появляется в процессе работы одной транзакции.
Но по стандарту SQL рекомендуется использовать SERIALIZABLE, когда транзакции исполняются принудительно одна за другой. В случае спорного доступа к одной и той же записи одна из транзакций будет объявлена "вне закона" и отменена. Очевидно, что на любом уровне, кроме SERIALIZABLE, появляется риск нарушения целостности. Но в некоторых ситуациях это допустимо.

Выполнение транзакций

По умолчанию MySQL работает в режиме автозавершения транзакций. Учитывая атомарность транзакции, каждый запрос в этом случае считается отдельной транзакцией и потому немедленно подтверждается после завершения. Такое поведение можно отключить инструкцией SET или воспользоваться инструкцией BEGIN. В первом случае каждый последующий запрос входит в неявную транзакцию, во втором — инструкция отмечает старт новой транзакции.
Как уже говорилось выше, транзакции применимы лишь к некоторым типам таблиц. Для стандартного типа по умолчанию MyISAM транзакции не поддерживаются, и попытка обновить таблицу в рамках транзакции все равно приведет к немедленному обновлению данных. И это изменение уже нельзя будет отменить. Если же транзакции используются достаточно часто, то можно сразу изменить тип таблиц по умолчанию в конфигурационном файле MySQL.
Отменяется транзакция с помощью команды ROLLBACK.
В транзакциях разрешается обновлять таблицы, но только с помощью инструкций INSERT и UPDATE. Изменения метаданных, схемы базы данных выполняются вне транзакций. Когда вводится инструкция, вносящая изменение в метаданные, текущая транзакция сразу же завершается. К завершению транзакции приводят следующие инструкции: ALTER TABLE, BEGIN, CREATE INDEX, DROP DATABASE, DROP TABLE, RENAME TABLE, TRUNCATE.

Блокировки

Блокировки — это механизм, применяемый в MySQL для реализации транзакций и обеспечения одновременного доступа к данным. Помимо этого, можно явно

запрашивать блокирование таблиц с помощью инструкции LOCK TABLES. Произвольные блокировки создаются функцией GET_LOCK() и работают с любыми таблицами — даже с теми, которые не поддерживают транзакции.
На внутреннем уровне MySQL может блокировать таблицы целиком, если это необходимо. Блокировать можно также строки, столбцы и страницы таблиц (страница — произвольный блок данных, связанный с таблицей). Преимущества и недостатки каждого из этих способов оцениваются по-разному в зависимости от назначения приложения и самой базы данных. Для web-приложений практичней использовать табличное блокирование, а для системы, подверженной частым откатам, — строковое блокирование.
Табличное блокирование производится инструкцией LOCK TABLES. Блокирование может быть жестким и нежестким, блокировка может также распространяться на несколько таблиц. Жесткая блокировка означает монопольный доступ к таблицам со стороны одного-единственного потока, а нежесткая позволяет одновременно считывать данные из таблицы.
Имитировать блокирование на уровне строк можно путем добавления к таблице специального столбца. Значения этого столбца — "заблокировано" и "свободно". Тип столбца можно задать SMALLINT.
Ниже приведен сценарий создания таблицы в базе данных предприятия, в которой содержатся документы определенного типа. С этой таблицей могут одновременно работать несколько сотрудников. Программа в запросе информации о документах запрашивает также и состояние документа — занят он или нет, то есть редактируется ли другим пользователем. При попытке пользователя начать редактирование документа программа сначала помечает запись об этом документе как редактируемую и только потом разрешает редактирование. При завершении редактирования программа возвращает документу статус нередактируемого.

```
CREATE TABLE DOCUMENTS
(ID_DOC INT NOT NULL,
NAME VARCHAR (20),
TOTAL_Q FLOAT (9,2),
LOCK ENUM ('LOCKED', 'UNLOCKED') NOT NULL,
PRIMARY KEY (ID))
```

В следующем листинге показана инструкция, блокирующая строку с идентификатором 2:

```
UPDATE DOCUMENTS
SET LOCK = 'LOCKED'
WHERE ID = 2;
```

Когда СУБД MySQL попытается выполнить эту инструкцию, может оказаться, что столбец LOCK уже содержит значение LOCKED. MySQL не производит обновления строк, если это не приводит к изменению содержащихся в строке данных. Это означает, что попытка заблокировать строку ни к чему не приведет. Сервер сообщит программе, что изменению подверглись ноль строк, и программа поймет, что со строкой уже работают.

Другой механизм блокировки реализуется функциями GET_LOCK() и RELEASE_ LOCK(). Они не связаны с какими-либо ресурсами, не контролируются СУБД и поэтому называются программными блокировками. Контроль над такими блокировками осуществляется программно, у каждой блокировки есть имя, и в конкретный момент времени поток может ставить только одну программную блокировку. С помощью этих функций можно устанавливать блокировки произвольного уровня детализации. Например, перед каждым обновлением строки можно запрашивать блокировку, имя которой будет состоять из имени таблицы и значения первичного ключа.
Типичный пример использования может выглядеть примерно так:

```
SELECT GET_LOCK ('COST ON ITEMS.ID = 3', 60);
UPDATE ITEMS SET COST = 3.142 WHERE ID = 3;
SELECT RELEASE_LOCK ('COST ON ITEMS.ID = 3');
```

В таблице ITEMS запрашиваем блокировку для строки с номером 3, причем имя блокировки выбрано произвольно. Суть такого механизма в том, что все приложения придерживаются единого правила именования блокировок. Здесь нет таких издержек, которые свойственны транзакциям, хотя преимущества, по сути, те же самые.

Последовательности

Последовательность — это специальная инструкция, доступная в некоторых реляционных СУБД включая Oracle. Она представляет собой счетчик, используемый для создания уникальных числовых идентификаторов. Текущее значение счетчика можно извлечь с помощью инструкции SELECT. Это происходит в атомарном режиме, что гарантирует уникальность каждого номера, тем самым гарантируется, что никакие два потока не получат два одинаковых идентификатора.
В MySQL уникальные идентификаторы строк реализуются лучше всего с помощью первичных ключей-счетчиков. Имитация последовательностей может потребоваться при переносе приложений в MySQL. Работа с последовательностями ведется посредством функции LAST_IN-SERT_ID(). Будучи вызванной без аргументов, она возвращает последнее значение счетчика, установленное путем автоматического увеличения или же самой функцией. Если же вызвать функцию с аргументом, она вернет значение аргумента. Но не стоит пытаться извлечь значение счетчика напрямую из таблицы: другой поток мог изменить это значение.