Лекция 12. Многопользовательские базы данных. Транзакции

До сих пор все рассмотренные нами примеры были примерами одиночных SQL-выражений. В данной лекции рассматриваются требования и среда, необходимые для совместного выполнения нескольких SQL-выражений.

Многопользовательские базы данных

Системы управления базами данных разрешают обращаться к данным и изменять их не только одному пользователю, но и нескольким одновременно. Если каждый пользователь выполняет запросы, как это происходит с хранилищем данных в течение обычных рабочих часов, для сервера БД это не создает больших проблем. Однако если некоторые пользователи добавляют и/или изменяют данные, серверу приходится сохранять довольно много промежуточных результатов.

Например, создается отчет, представляющий доступный остаток всех текущих счетов, открытых в отделении. При этом одновременно с выполнением отчета происходит следующее:

• Служащий отделения обрабатывает вклад для одного из клиентов.

• Клиент заканчивает снимать деньги через банкомат в операционном зале.

• Банковское приложение, выполняющееся в конце каждого месяца, начисляет процент по счетам.

Блокировка

Следовательно, пока создается отчет, несколько пользователей изменяют данные. Так, какие цифры должны появиться в отчете? Ответ отчасти зависит от того, как сервер реализовывает ***блокировку*** (*locking*) – механизм управления одновременным использованием ресурсов данных.

Существуют две основные **стратегии блокировки**:

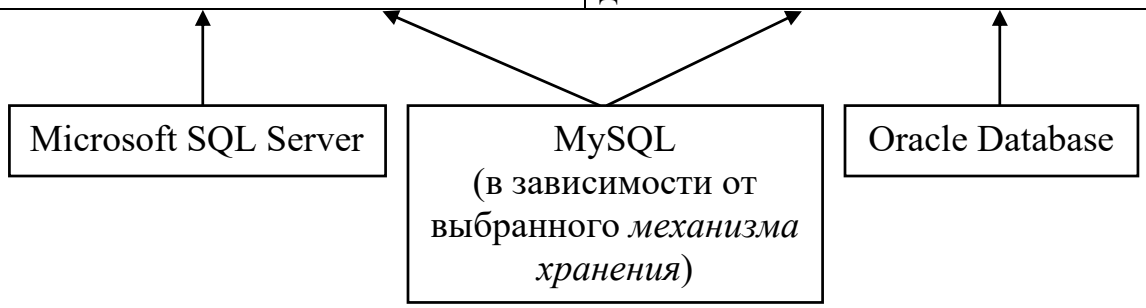
1) Пользователи, осуществляющие запись в БД, должны запрашивать и получать от сервера ***блокировку записи*** (*write lock*) для изменения данных. А пользователи, считывающие данные из БД, должны запрашивать и получать от сервера ***блокировку чтения*** (*read lock*) для осуществления запросов к данным. В то время как чтение может осуществляться одновременно несколькими пользователями, для каждой таблицы (или ее части)

одновременно выдается только одна блокировка записи, и запросы на чтение блокируются до тех пор, пока не будет снята блокировка записи.

2) Пользователи, осуществляющие запись в БД, для изменения данных должны запрашивать и получать от сервера ***блокировку записи***, но пользователи, считывающие данные, для запроса данных не нуждаются ни в каком типе блокировки. Вместо этого сервер гарантирует, что читатель видит непротиворечивое представление данных (данные представляются неизменными, даже несмотря на то, что другие пользователи могут их модифицировать), начиная с момента начала запроса до его завершения. Этот подход известен как ***контроль версий*** (*versioning*).

У обеих стратегий есть свои достоинства и недостатки.

Первый подход	Второй подход
Время ожидания может оказаться длительным, если одновременно поступило много запросов на чтение и запись.	Могут возникнуть проблемы в случае длительных запросов, поскольку происходит изменение данных.



Уровни (детализации) блокировки

Также есть ряд различных стратегий блокировки ресурса. Блокирование может выполняться на одном из трех разных уровней, или с одной из трех ***детализаций*** (*granularities*):

- *Блокирование таблицы* – предотвращает одновременное изменение несколькими пользователями данных одной таблицы.

- *Блокирование страницы* – предотвращает одновременное изменение несколькими пользователями данных одной страницы таблицы (страница – сегмент памяти, обычно от 2 до 16 Кбайт).

- *Блокирование строки* – предотвращает одновременное изменение несколькими пользователями одной строки таблицы.

У этих подходов тоже есть свои плюсы и минусы. При блокировке всей таблицы возникает очень мало промежуточных результатов, но по мере роста числа пользователей такая блокировка очень быстро приводит к недопустимому времени ожидания. С другой стороны, в случае блокировки строки сохраняется намного больше промежуточных результатов, но такая блокировка позволяет многим пользователям вносить изменения в одну таблицу, если это касается разных строк. **Microsoft SQL Server** использует блокировки таблицы, страницы и строки, **Oracle Database** – блокировку строки, а **MySQL** может блокировать таблицу, страницу или строку.

Возвращаясь к отчету: данные, появляющиеся на его страницах, будут отражать состояние БД или на момент начала создания отчета (если сервер использует контроль версий), или на момент осуществления сервером блокировки чтения (если сервер использует блокировки и чтения, и записи).

Транзакции

Чтобы несколько пользователей могли осуществлять доступ к одним и тем же данным (параллельный доступ), применяются *транзакции* (*transactions*).

Транзакция – механизм группировки нескольких SQL-выражений, позволяющий успешно выполниться *всем* или *ни одному* из них. Например, если клиент пытается перевести 500 долларов со сберегательного счета на текущий, он немного расстроится, если деньги будут успешно сняты с первого счета, но не внесены на второй. Какой бы ни была причина сбоя (сервер был выключен для проведения работ по техническому обслуживанию, истекло время ожидания запроса на блокировку страницы таблицы **account** и др.), клиент захочет вернуть свои 500 долларов.

Чтобы защититься от ошибок такого рода, программа, обрабатывающая запрос на перевод, сначала начинает транзакцию, затем выполняет SQL-выражения, необходимые для перемещения денег со сберегательного счета на текущий, и, если все проходит успешно, завершает транзакцию, формируя команду ***commit***. Однако, если происходит что-то непредвиденное, программа выдает команду ***rollback***, которая указывает серверу отменить все изменения, внесенные с момента начала транзакции.

Запуск транзакции

Серверы БД обрабатывают создание транзакций одним из двух возможных способов:

• Активная транзакция всегда присутствует для каждого сеанса работы с БД, поэтому нет ни необходимости, ни способа для явного начала транзакции. По завершении транзакции сервер автоматически начинает новую транзакцию для сеанса пользователя.

• Если транзакция не начата явно, отдельные SQL-выражения фиксируются автоматически независимо друг от друга. Чтобы начать транзакцию, сначала нужно запустить на выполнение команду.

Oracle Database использует первый подход, а **Microsoft SQL Server** и **MySQL** – второй. Одно из преимуществ подхода Oracle к обработке транзакций в том, что даже в случае одиночной SQL-команды есть возможность сделать откат, если пользователя не удовлетворяет результат или он изменил свое мнение. Таким образом, если вы забудете вставить блок *where* в выражение *delete*, останется возможность отменить неверные действия (как, например, осознание того, что вы не хотели удалять все 125 000 строк своей таблицы). Однако при работе с MySQL и SQL Server, как только нажата клавиша *Enter*, изменения, осуществленные SQL-выражением, становятся постоянными (и тогда только администратор БД сможет восстановить исходные данные из резервной копии или какими-либо иными средствами).

Стандарт SQL включает команду ***start transaction***, предназначенную для явного начала транзакции. MySQL соответствует этому стандарту, а пользователи SQL Server должны вызывать команду ***begin transaction***. Для обоих серверов, пока транзакция не начата явно, все операции выполняются в *режиме автоматической фиксации* (*autocommit mode*), т.е. сервер автоматически фиксирует отдельные выражения.

Оба сервера, MySQL и SQL Server, позволяют отключать режим автоматической фиксации для отдельных сеансов. В этом случае серверы будут вести себя в отношении транзакций точно так же, как Oracle Database. В SQL Server для отключения режима автоматической фиксации служит следующая команда:

```
SET IMPLICIT_TRANSACTIONS ON
```

MySQL позволяет отключить режим автоматической фиксации так:

```
SET AUTOCOMMIT=0
```

Если режим автоматической фиксации выключен, все SQL-команды выполняются в рамках транзакции, и их фиксацию или откат следует выполнять явно.

Завершение транзакции

Если транзакция запущена – явно посредством команды *start transaction* или неявно сервером БД, – пользователь должен явно завершить ее, чтобы внесенные им изменения стали постоянными. Это делается с помощью команды ***commit***, которая указывает серверу пометить изменения как постоянные и высвободить все ресурсы (т.е. снять блокировку страниц или строк), используемые во время транзакции.

Если решено отменить все изменения, сделанные с момента начала транзакции, необходимо выполнить команду ***rollback***, которая указывает серверу вернуть данные в то состояние, в каком они находились до начала транзакции. После завершения выполнения *rollback* все ресурсы, используемые сеансом, высвобождаются.

Кроме выполнения команды *commit* или *rollback*, возможны еще несколько сценариев завершения транзакции:

• Выключение сервера; в этом случае откат транзакции будет выполнен автоматически при возобновлении работы сервера.

• Выполнение SQL-выражения управления схемой, например, *alter table*, что приведет к фиксации текущей транзакции и запуску новой.

• Выполнение еще одной команды *start transaction*, в результате чего происходит фиксация предыдущей транзакции.

• Преждевременное завершение транзакции сервером, который выявил *взаимоблокировку* (*deadlock*), которая происходит, когда две разные транзакции ожидают ресурсов, удерживаемых другой транзакцией. В этом случае будет выполнен откат транзакции и пользователь получит сообщение об ошибке.

Точки сохранения транзакций

В некоторых случаях может возникнуть проблема, когда требуется откат транзакции, но не хочется отменять *все*, что было сделано в рамках этой транзакции. Для таких ситуаций в транзакции можно установить одну или более ***точек сохранения*** (*savepoints*) и использовать их для отката к определенному месту транзакции, а не откатывать полностью к началу.

Всем точкам сохранения должны быть присвоены имена, что позволит иметь несколько таких точек в одной транзакции. Создать точку сохранения *my_savepoint* можно так:

SAVEPOINT my_savepoint;

Чтобы сделать откат к определенной точке сохранения, используется команда:

ROLLBACK TO SAVEPOINT my_savepoint;

Рассмотрим пример использования точек сохранения:

```
START TRANSACTION;
UPDATE product
SET date_retired = CURRENT_TIMESTAMP( )
WHERE product_cd = 'XYZ';

SAVEPOINT before_close_accounts;

UPDATE account
SET status = 'CLOSED', close_date = CURRENT_TIMESTAMP( ),
   last_activity_date = CURRENT_TIMESTAMP( )
WHERE product_cd = 'XYZ';

ROLLBACK TO SAVEPOINT before_close_accounts;

COMMIT;
```

Результирующий эффект этой транзакции состоит в том, что фиктивный тип счета *XYZ* выходит из обращения, но ни один из счетов не закрывается.

При использовании точек сохранения необходимо помнить следующее:

• Несмотря на название, при создании точки сохранения ничего не сохраняется. Если требуется сделать транзакцию постоянной, необходимо выполнить команду *commit*.

• Если использовать команду *rollback* без указания точки сохранения, все точки сохранения транзакции будут проигнорированы и отменена будет вся транзакция.

Работая с SQL Server, используйте его собственные команды: *save transaction* – для создания точки сохранения и *rollback transaction* – для отката к точке сохранения. За каждой такой командой должно следовать имя точки сохранения.

Литература:

1. Алан Бьюли. Изучаем SQL: пер. с англ. – СПб-М.: Символ, O’Reilly, 2007. – 310 с.