

Лекция №6

Транзакции, ACID. Временные таблицы, управляющие конструкции, циклы

Сегодня мы узнаем:

Понятие транзакции, свойства ACID.

Понятие временных таблиц

Оператор IF

Оператор While

Термины лекции

Транзакция — это набор последовательных операций с базой данных, соединенных в одну логическую единицу.

Изоляция — это свойство транзакции, которое позволяет скрывать изменения, внесенные одной операцией транзакции при возникновении явления *race condition*

Процедура - это подпрограмма (например, подпрограмма) на обычном языке сценариев, хранящаяся в базе данных.

ACID - это набор из четырех требований к транзакционной системе, обеспечивающих максимально надежную и предсказуемую работу.

Оглавление

Тизер	1
Термины лекции	1
Понятие транзакции, свойства ACID.	2
Переменные	6
Оператор IF	10
Цикл WHILE	12
Итоги	15

Понятие транзакции, свойства ACID.

От корректного функционирования базы данных (БД) может зависеть не только скорость, но и надежность приложения. Для глубокого погружения в задачи специалисту, как правило, нужно освоить работу с транзакциями – об этом и пойдет речь ниже. Рассмотрим виды и свойства транзакций, а также постараемся понять, как устроен этот механизм.

Транзакция — это некий набор связанных операций с базой данных. В первом приближении это действительно так. Однако, пока определение неполное. Не хватает самого главного, а именно — этот набор операций должен представлять единую логическую систему с данными. Например, давайте представим такую ситуацию: у каждого человека есть карта, с помощью которой он может совершать определенные действия, будь то онлайн-покупка, перевод денежных средств с карты на карту, оплата счетов и т.д. Какие операции происходят в базе данных при совершении перевода денежных средств с одного лицевого счета на другой? В этой ситуации необходимо выполнить два запроса к базе данных:

1. С первого лицевого счета происходит списание N-ой суммы денежных средств.

2. На второй лицевой счет идет зачисление этой же суммы.

В данном случае эти две операции связаны и составляют единую логическую систему работы с данными. Теперь можно дать полное определение транзакции.

Транзакция — это набор последовательных операций с базой данных, соединенных в одну логическую единицу.

Виды транзакций

Транзакции делят на два вида:

Неявные транзакции, которые предусмотрены на уровне базы данных. Например, БД задает отдельную инструкцию INSERT, UPDATE или DELETE как единицу транзакции.

Явные транзакции — их начало и конец явно обозначаются такими инструкциями, как BEGIN TRANSACTION, COMMIT или ROLLBACK.

Свойства транзакции

Выделяют так называемые «магические» свойства транзакции, которые описываются аббревиатурой «ACID». Каждая буква аббревиатуры означает одно из свойств, о которых мы поговорим ниже.

Atomicity или атомарность (A)

Вернемся к предыдущему примеру с переводом денежных средств между двумя лицевыми счетами. Мы установили, что эти 2 операции, которые взаимодействуют с базой данных, являются операциями транзакции. А какие проблемы могут возникнуть,

если мы просто выполним эти операции последовательно, отправив два запроса к БД? Первый запрос выполнится успешно. С первого лицевого счета будет списана N-ая сумма денежных средств. Однако, в случае той или иной технической ошибки во время выполнения второго запроса может случиться так, что денежные средства с одного лицевого счета уйдут, а на другой счет не поступят. В этой ситуации речь идет о проблеме потери данных. В целях снижения этого риска транзакции обладают таким свойством, как атомарность (atomicity), неделимость: либо будут выполнены все действия транзакции, либо никакие.

Consistency или согласованность (C)

Согласованность означает, что если до выполнения транзакции данные в БД находятся в некоем состоянии «good state»*, то они будут в этом же состоянии и после выполнения транзакции.

*Иными словами, выполняется некий набор условий. Примеры: в таблице countries не должно быть двух строк с названием страны «Российская Федерация»; возраст человека не может быть больше 150 лет.

На самом деле ни одна база данных не может гарантировать свойство согласованности. А всё потому, что поддержание консистентности — это прерогатива приложения, а не БД. База данных лишь предоставляет инструменты для выполнения данного свойства транзакции, например, уникальные ключи, внешние ключи и т.д.

Isolation или изоляция (I)

Переходим к самому интересному свойству — изоляции. Представим ситуацию, когда в определенный момент времени с системой работают несколько пользователей. Естественно, операции транзакции в БД выполняются параллельно, чтобы ускорить работу системы. Но у параллельной работы транзакций есть свои подводные камни:

Если операции транзакции взаимодействуют с разным набором непересекающихся данных, все работает корректно.

Но что будет, если две и более операций транзакции в один момент времени начнут работать с одним и тем же набором данных? Возникнет явление, называемое race condition (состояние гонки).

Выделяют несколько эффектов, связанных с этим явлением.

Эффект потерянного обновления возникает, когда несколько

транзакций обновляют одни и те же данные, не учитывая изменений, сделанных другими транзакциями.

Представим, что у клиента банка есть счет, на котором находится 1000 денежных единиц. Транзакции А и В считывают данное значение из БД. Транзакция А должна увеличить данную сумму на 100 денежных единиц, а транзакция В — на 200. Транзакция А увеличивает сумму денежных единиц на счёте на 100 (итого 1100) и записывает значение в БД, транзакция В увеличивает сумму на 200 денежных единиц и записывает в БД (итого 1200). В результате на счете должно оказаться 1300, а по факту имеем 1200 денежных единиц.

Эффект грязного чтения возникает, когда транзакция считывает данные, которые еще не были зафиксированы.

Представим, что транзакция А переводит все деньги клиента на другой счет, но не фиксирует изменения. Транзакция В считывает изменения счёта А, получает 0 денежных единиц на счете и отказывает клиенту в выдаче наличных. Транзакция А прерывается и отменяет перевод между счетами.

Эффект неповторяемого чтения возникает, когда транзакция считывает дважды одну и ту же строку, но каждый раз получает разные результаты.

Например, по правилу согласованности клиент банка не может иметь отрицательный баланс на счёте. Транзакция А хочет уменьшить баланс счета клиента на 200 денежных единиц. Она проверяет текущее значение суммы на счёте — 500 денежных единиц. В это время транзакция В уменьшает сумму на счёте до 0 и фиксирует изменения. Если бы транзакция А повторно проверила сумму, то получила бы 0 денежных единиц, но на основе первоначальных данных она уже приняла решение уменьшить значение, и счет уходит в минус.

Эффект чтения фантомов возникает, когда набор данных соответствует условиям поиска, но изначально не отображается.

Например, правило согласованности запрещает иметь клиенту более 3 лицевых счетов. Для открытия нового счета транзакция А проверяет все счета клиента банка и в результате получает 2 счета. В этот момент транзакция В открывает еще один счет клиенту и фиксирует изменения (3 счета). Если бы транзакция А повторно проверила количество лицевых счетов клиента, то их оказалось бы 3, и по правилу согласованности открытие нового счета было бы невозможно.

Решение

Для устранения данных эффектов на уровне баз данных предусмотрены уровни изоляции, или transaction isolation levels, которые так или иначе реализованы во многих СУБД. Для примера рассмотрим движок InnoDB в СУБД MySQL:

Read uncommitted – это уровень изоляции, при котором каждая транзакция видит незафиксированные изменения другой транзакции. Справляется с эффектом потерянного обновления, но остаются остальные проблемы: эффекты грязного чтения, неповторяемого чтения, чтения фантомов. Все запросы SELECT считывают данные в неблокирующей манере.

Блокирующее чтение (SELECT ... FOR UPDATE, LOCK IN SHARE MODE), UPDATE и DELETE блокирует искомые индексные строки. Таким образом, возможна вставка данных в промежутки между индексами. Промежутки блокируются только при проверках на дублирующиеся и внешние ключи.

Read committed — это уровень изоляции, при котором параллельно исполняющиеся транзакции видят только зафиксированные изменения других транзакций. Справляется с эффектами потерянного обновления и грязного чтения, остаются эффекты неповторяемого чтения и чтения фантомов. Согласованное чтение не накладывает блокировок, однако считывает данные из свежего снэпшота. В остальном ведёт себя так же, как и read uncommitted.

Repeatable read или snapshot isolation — это уровень изоляции, при котором транзакция не видит изменения данных, прочитанные ей ранее, однако способна прочитать новые данные, соответствующие условию поиска. Справляется с эффектами потерянного обновления, грязного чтения, неповторяемого чтения, остается эффект чтения фантомов. Согласованное чтение не накладывает блокировок и считывает данные из снэпшота, который создается при первом чтении в транзакции. Таким образом, одинаковые запросы вернут одинаковый результат.

Блокировка для блокирующего чтения будет зависеть от типа условия:

1. если условие с диапазоном, например, WHERE (id > 7), то блокируется весь диапазон;
2. если уникальное, например, WHERE (id = 7), то блокируется одна индексная запись.

Serializable — это уровень изоляции, при котором каждая

транзакция выполняется так, как будто параллельных транзакций не существует. Справляется со всеми перечисленными выше эффектами. Аналогично repeatable read, но есть интересный момент. Если выключен autocommit (а при явном старте транзакции START TRANSACTION он выключен по умолчанию), то все запросы SELECT превращаются в запросы SELECT ... LOCK IN SHARE MODE

Изоляция — это свойство транзакции, которое позволяет скрывать изменения, внесенные одной операцией транзакции при возникновении явления race condition.

Durability или долговечность (D)

Долговечность означает, что если транзакция выполнена, и даже если в следующий момент произойдет сбой в системе, результат сохранится.

Если вы пользуетесь облачными хранилищами, такими как Amazon S3, то могли заметить, что разные тарифы обещают вам разное количество девяток durability. В контексте облака durability означает сохранность ваших данных и то, как они реплицируются. Чем больше копий ваших данных в разных точках мира, тем выше вероятность их не потерять из-за наводнения, землетрясения или нашествия инопланетян. В контексте «ACID» это обычно означает, что после фиксирования данные записываются в постоянное хранилище.

Пример транзакции

Денежные переводы — отличный пример, показывающий, почему необходимы транзакции. Если при оплате покупки происходит перевод от клиента на счет магазина, то счет клиента должен уменьшиться на эту сумму, а счет магазина — увеличиться на нее же.

По шагам это будет выглядеть так:

Убедиться, что остаток на счете клиента больше 3000 рублей.

Вычесть 3000 рублей со счета клиента.

Добавить 3000 к счету интернет-магазина.

Команды входящие в транзакцию:

```
START TRANSACTION;
```

Команда **START TRANSACTION** начинает транзакцию.

```
SELECT total FROM accounts WHERE user_id = 2;
```

Убеждаемся, что на счету пользователя достаточно средств.

```
UPDATE accounts SET total = total - 3000 WHERE user_id = 2;
```

Снимаем средства со счета пользователя.

```
UPDATE accounts SET total = total + 3000 WHERE user_id IS NULL;
```

Перемещаем денежные средства на счет интернет-магазина.

```
COMMIT;
```

Чтобы изменения вступили в силу, мы должны выполнить команду COMMIT.

Понятие временных таблиц

В MySQL временная таблица — это особый тип таблицы, который позволяет вам сохранить временный набор результатов, который вы можете повторно использовать несколько раз в одном сеансе. Временная таблица очень удобна, когда невозможно запрашивать данные, требующие одного SELECT оператора с JOIN предложениями. В этом случае вы можете использовать временную таблицу для хранения немедленного результата и использовать другой запрос для его обработки.

Временная таблица MySQL имеет следующие специальные функции:

- Временная таблица создается с помощью CREATE TEMPORARY TABLE инструкции. Обратите внимание, что ключевое слово TEMPORARY добавлено между ключевыми словами CREATE и TABLE.
- MySQL автоматически удаляет временную таблицу при завершении сеанса или разрыве соединения. Конечно, вы можете использовать этот DROP TABLE оператор для явного удаления временной таблицы, когда вы ее больше не используете.
- Временная таблица доступна и доступна только клиенту, который ее создает. Разные клиенты могут создавать временные таблицы с одним и тем же именем, не вызывая ошибок, поскольку только клиент, создавший временную таблицу, может ее видеть. Однако в одном сеансе две временные таблицы не могут иметь одно и то же имя.
- Временная таблица может иметь то же имя, что и обычная

таблица в базе данных. Например, если вы создаете временную таблицу с именем `employee` в образце базы данных , существующая `employees` таблица становится недоступной. Каждый запрос, который вы отправляете к `employees` таблице, теперь ссылается на временную таблицу `employees`. Когда вы удаляете `employees` временную таблицу, постоянная `employees` таблица доступна и доступна.

Хотя временная таблица может иметь то же имя, что и постоянная таблица, это не рекомендуется. Потому что это может привести к путанице и потенциально вызвать непредвиденную потерю данных.

Например, в случае потери соединения с сервером базы данных и автоматического повторного подключения к серверу вы не сможете отличить временную таблицу от постоянной. Затем вы можете выдать `DROP TABLE` инструкцию для удаления постоянной таблицы вместо временной, что не ожидается. Чтобы избежать этой проблемы, вы можете использовать `DROP TEMPORARY TABLE` инструкцию для удаления временной таблицы.

CREATE TEMPORARY TABLE

Синтаксис `CREATE TEMPORARY TABLE` оператора аналогичен синтаксису `CREATE TABLE` оператора, за исключением `TEMPORARY` ключевого слова.

Переменные в MySQL.

Иногда вы хотите передать значение из оператора SQL в другой оператор SQL. Для этого вы сохраняете значение в пользовательской переменной MySQL в первом операторе и ссылаетесь на него в последующих инструкциях. Для создания пользовательской переменной вы используете формат `@variable_name`, в котором `variable_name` состоит из буквенно-цифровых символов. Максимальная длина пользовательской переменной составляет 64 символа с MySQL 5.7.5. Пользовательские переменные не чувствительны к регистру. Это означает, что `@id` и `@ID` то же самое. Вы можете назначить пользовательскую переменную определенным типам данных, таким как целое число, число с плавающей запятой, десятичное число, строка или `NULL`. Пользовательская переменная, определенная одним клиентом, не видна другим клиентам. Другими словами, пользовательская переменная является специфичной для сессии.

Обратите внимание, что пользовательские переменные являются специфичным для MySQL расширением стандарта SQL. Они могут быть недоступны в других системах баз данных.

Есть два способа присвоить значение пользовательской переменной. Первый способ заключается в следующем:

1. SET @variable_name := value;

2. Вы можете использовать либо : =, либо = как оператор присваивания в операторе SET. Например, оператор присваивает число 100 переменной @counter.

SET @counter := 100;

3. Второй способ присвоения значения переменной — использование оператора SELECT. В этом случае вы должны использовать оператор присваивания : =, потому что в операторе SELECT в MySQL рассматривает оператор = как оператор равенства.

SELECT @variable_name := value;

4. Пользовательские переменные не чувствительны к регистру:

@ID и @id - то же самое

Примеры переменных MySQL

Следующий оператор получает самый дорогой продукт в таблице products и назначает цену пользовательской переменной @msrp:

```
1 SELECT
2   @msrp:=MAX(msrp)
3 FROM
4   products;
```

Оператор IF

Функция MySQL IF— это одна из функций потока управления MySQL, которая возвращает значение на основе условия. Функция IF иногда упоминается как IF ELSE и IF THEN ELSE функция.

Синтаксис функции MySQL IF следующий:

IF(expr,if_true_expr,if_false_expr)

Язык кода: SQL (язык структурированных запросов) (sql)

Если значение expr оценивается как TRUE, expr не равно NULL и не expr равно 0, функция возвращает if true expr, в противном случае она возвращает if false expr. Функция IF возвращает

числовое значение или строку, в зависимости от того, как она используется.

Простой пример функции ЕСЛИ

Вы можете использовать функцию ЕСЛИ непосредственно в операторе SELECT без предложений From других, как показано ниже:

```
SELECT IF(1 = 2,'true','false'); -- false
SELECT IF(1 = 1,'true','false'); -- true
```

Отображение N/A вместо NULL с использованием функции MySQL IF

Давайте взглянем на данные в customers таблице в образце базы данных .

В customer's таблице у многих клиентов нет данных о состоянии в state столбце, поэтому, когда мы выбираем клиентов, в столбце состояния отображаются NULL значения, которые не имеют значения для целей отчетности. См. следующий запрос:

Мы можем улучшить вывод, используя IF функцию для возврата N/A, если состояние равно NULL, как следующий запрос:

```
1 SELECT
2     customerNumber,
3     customerName,
4     IF(state IS NULL, 'N/A', state) state,
5     country
6 FROM
7     customers;
```

Функция MySQL IF с агрегатными функциями

MySQL SUM IF – объединение функции ЕСЛИ с функцией СУММ

Функция IF полезна, когда она сочетается с агрегатной функцией. Предположим, если вы хотите узнать, сколько заказов было отправлено и отменено, вы можете использовать функцию ЕСЛИ с агрегатной функцией СУММ в виде следующего запроса.

В приведенном выше запросе, если статус заказа равен shipped или cancelled , функция ЕСЛИ возвращает 1, в противном случае она возвращает 0. SUM Функция вычисляет общее количество shipped cancelled заказов на основе возвращаемого значения функции.

MySQL COUNT IF – Объединение функции IF с функцией COUNT

Во- первых, мы выбираем отдельный статус заказа в orders таблице, используя следующий запрос.

Во-вторых, мы можем получить количество заказов в каждом статусе, объединившая функцию с COUNT функцией. Поскольку COUNT функция не подсчитывает NULL значения, IF функция возвращает значение, NULL если статус не находится в выбранном статусе, в противном случае она возвращает 1. См. следующий запрос.

По плану нашей лекции далее идет цикл WHILE, но чтобы увидеть примеры использования цикла, попробуем разобраться с еще одной важной темой - процедуры.

Процедура - это подпрограмма (например, подпрограмма) на обычном языке сценариев, хранящаяся в базе данных. В случае MySQL процедуры записываются в MySQL и хранятся в базе данных / сервера MySQL. Процедура MySQL имеет имя, список параметров и операторы SQL.

В MySQL процедура - это хранимая программа, в которую вы можете передавать параметры. Процедура не возвращает значение, как функция.

Создать процедуру (create Procedure)

Как и на других языках программирования, вы можете создавать свои собственные процедуры в MySQL. Рассмотрим подробнее.

Синтаксис

procedure_name - наименование процедуры в MySQL.

parameter - необязательный. Один или несколько параметров передаются в процедуру. При создании процедуры могут быть объявлены три типа параметров:

IN - Параметр может ссылаться на процедуру. Значение параметра не может быть перезаписано процедурой.

OUT - Параметр не может ссылаться на процедуру, но значение параметра может быть перезаписано процедурой.

IN OUT - Параметр может ссылаться на процедуру, и значение параметра может быть перезаписано процедурой.

declaration_section - место в процедуре, где вы объявляете локальные переменные.

executable_section - место в процедуре, в котором вы создаете код процедуры.

Создадим процедуру с именем «Фильтр» без параметров. Он выбирает все записи из таблицы «запись», где в столбце «Страна» в конце значений стоит «ia». Процесс должен быть завершен ключевым словом «END». Мы будем использовать предложение CALL для выполнения хранимой процедуры в

командной строке. После выполнения команды CALL мы получаем следующие результаты. Вы можете видеть, что запрос должен получить только те записи, в которых столбец «Страна» имеет «ia» в конце значений.

Отличие процедуры от функции

Отличие процедур и функций состоит лишь в том, что процедуры не возвращают никаких значений, в отличие от функций. И процедуры и функции могут принимать входные параметры. Но в PostgreSQL нет различий в синтаксисе создания функций и процедур, обе создаются с помощью ключевых слов create function

WHILE

Оператор MySQL WHILE loop используется для выполнения одного или нескольких операторов снова и снова, пока условие истинно. Мы можем использовать цикл, когда нам нужно выполнить задачу с повторением, пока условие истинно.

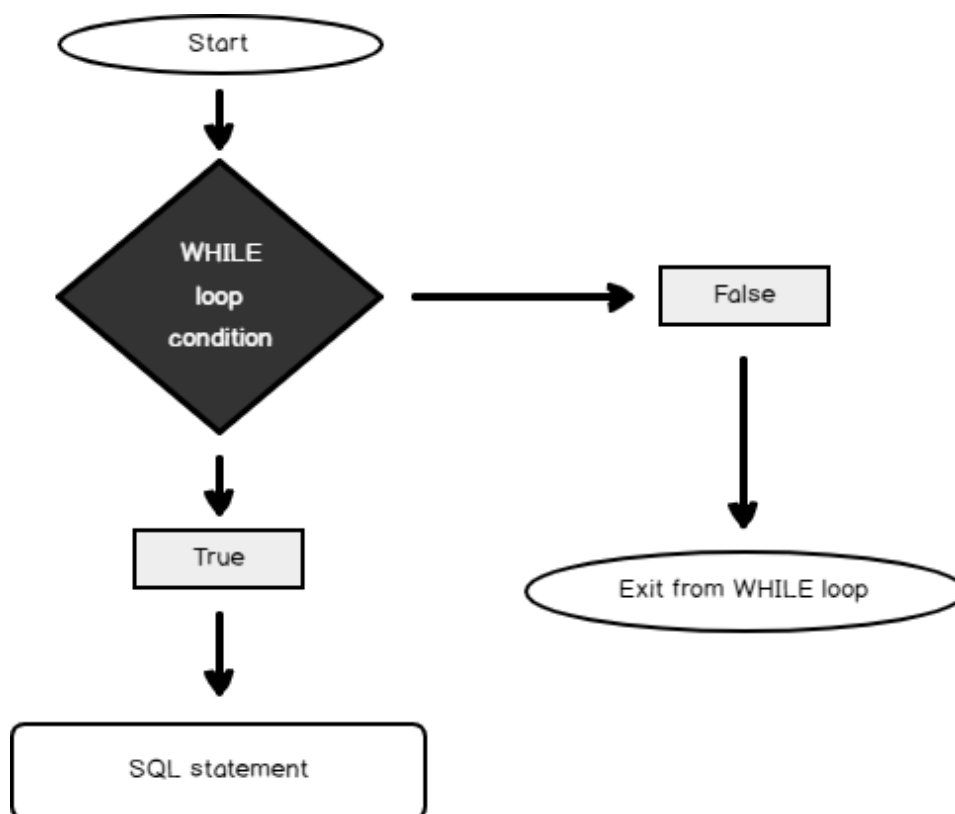
Используйте оператор цикла WHILE в случае, если вы не уверены, сколько раз вы хотели бы, чтобы тело цикла выполнялось. Поскольку условие WHILE оценивается перед входом в цикл, возможно, что тело цикла может не выполняться даже один раз.

label_name - необязательный. Это наименование связано с циклом WHILE.

condition - условие которое проверяется при каждой итерации цикла WHILE. Если condition примет значение TRUE, тело цикла выполняется. Если условие принимает значение FALSE, цикл WHILE прекращается.

statements - код, выполняемый при каждом, проходе через цикл WHILE.

Следующая блок-схема объясняет основную структуру цикла WHILE в SQL:



На приведенной выше визуальной символической диаграмме алгоритм цикла WHILE в MySQL представлен в простом понятии дизайна и значении.

Во-первых, запускается выполнение цикла WHILE, для каждой итерации цикла оценивается определенное условие, затем на основе результата условия WHILE определяется оператор SQL. Когда цикл WHILE приводит к TRUE, операторы потока кода будут выполняться.

Во-вторых, если выполнение цикла WHILE имеет тенденцию давать какой-либо результат FALSE в соответствии с условием поиска, тогда поток кода будет отозван, и цикл WHILE остановит дальнейшую обработку. И предположим, что если какой-либо оператор SQL доступен вне цикла WHILE, то он будет выполнен.

Давайте посмотрим на примере, как это работает. Предположим, мы хотим знать названия, авторов и количество книг, которые поступили в различные поставки. Интересующая нас информация хранится в двух таблицах - Журнал Поставок (magazine_incoming) и Товар (products). Давайте напишем интересующий нас запрос.

А что, если нам необходимо, чтобы результат выводился не в одной таблице, а по каждой поставке отдельно? Конечно, можно написать 3 разных запроса, добавив в каждый еще одно условие:

```

1
2 SELECT magazine_incoming.id_incoming, products.name, products.author,
   magazine_incoming.quantity
3 FROM magazine_incoming, products
4 WHERE magazine_incoming.id_product=products.id_product AND
   magazine_incoming.id_incoming=1;
5
6 SELECT magazine_incoming.id_incoming, products.name, products.author,
   magazine_incoming.quantity
7 FROM magazine_incoming, products
8 WHERE magazine_incoming.id_product=products.id_product AND
   magazine_incoming.id_incoming=2;
9
10 SELECT magazine_incoming.id_incoming, products.name, products.author,
   magazine_incoming.quantity
11 FROM magazine_incoming, products
12 WHERE magazine_incoming.id_product=products.id_product AND
   magazine_incoming.id_incoming=3;

```

Но гораздо короче сделать это можно с помощью цикла WHILE:

```

1 DECLARE i INT DEFAULT 3;
2 WHILE i>0 DO
3     SELECT magazine_incoming.id_incoming, products.name, products.author, magazine_incoming.quantity
4     FROM magazine_incoming, products
5     WHERE magazine_incoming.id_product=products.id_product AND magazine_incoming.id_incoming=i;
6     SET i=i-1;
7 END WHILE;

```

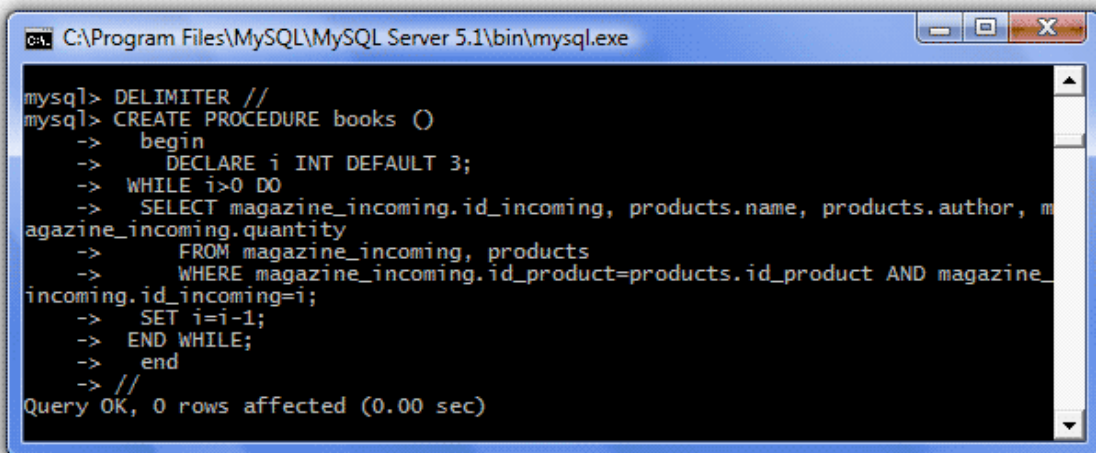
Для проверки попробуем создать процедуру.

Т.е. мы ввели переменную *i*, по умолчанию равную 3, сервер выполнит запрос с *id* поставки равным 3, затем уменьшит *i* на единицу (*SET i=i-1*), убедится, что новое значение переменной *i* положительно (*i>0*) и снова выполнит запрос, но уже с новым значением *id* поставки равным 2. Так будет происходить, пока переменная *i* не получит значение 0, условие станет ложным, и цикл закончит свою работу.

Чтобы убедиться в работоспособности цикла создадим хранимую процедуру *books* и поместим в нее цикл:

Теперь у нас 3 отдельные таблицы (по каждой поставке). Согласитесь, что код с циклом гораздо короче трех отдельных запросов. Но в нашей процедуре есть одно неудобство, мы объявили количество выводимых таблиц значением по умолчанию (*DEFAULT 3*), и нам придется с каждой новой поставкой менять это значение, а значит код процедуры. Гораздо удобнее сделать это число входным параметром. Давайте перепишем нашу процедуру, добавив входной параметр *num*, и,

учитывая, что он не должен быть равен 0:

A screenshot of a Windows command prompt window titled "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe". The window contains a MySQL command-line interface. The user has entered several commands: "mysql> DELIMITER //" to change the statement delimiter, "mysql> CREATE PROCEDURE books ()" to create a new procedure, and then a multi-line procedure definition starting with "begin", followed by a "DECLARE i INT DEFAULT 3;" statement, a "WHILE i>0 DO" loop, a "SELECT" query joining "magazine_incoming" and "products" tables, a "FROM" clause, a "WHERE" clause, a "SET i=i-1;" statement, and finally "END WHILE;" and "end". The prompt returns to "mysql>". At the bottom, it shows "Query OK, 0 rows affected (0.00 sec)".

```
mysql> DELIMITER //  
mysql> CREATE PROCEDURE books ()  
-> begin  
-> DECLARE i INT DEFAULT 3;  
-> WHILE i>0 DO  
-> SELECT magazine_incoming.id_incoming, products.name, products.author, m  
magazine_incoming.quantity  
-> FROM magazine_incoming, products  
-> WHERE magazine_incoming.id_product=products.id_product AND magazine_  
incoming.id_incoming=i;  
-> SET i=i-1;  
-> END WHILE;  
-> end  
-> //  
Query OK, 0 rows affected (0.00 sec)
```

Итоги

Сегодня мы узнали о транзакциях и о требованиях, которые к ним предъявляются. Разобрались с понятием временных таблиц и их отличий от существующих, узнали про создание переменных и присваивание им значений. Узнали про оператор `iF`, задали цикл `WHILE` и научились разбираться с процедурой