## **Лабораторная работа № 4 Создание хранимых процедур**

**Теоретические сведения**

На практике часто требуется повторять последователь­ность одинаковых запросов. Хранимые процедуры позволяют объединить последова­тельность таких запросов и сохранить их на сервере. После этого клиентам достаточно послать один за­прос на выполнение хранимой процедуры.

Хранимые процедуры обладают следующими преимуществами.

* *Повторное использование кода* – после создания хранимой процедуры ее можно вызывать из любых приложений и SQL-запросов.
* *Сокращение сетевого трафика* – вместо нескольких запросов экономнее послать серверу запрос на выполнение хранимой процедуры и сразу получить ответ.
* *Безопасность* – действия не приведут к нарушению целостности данных, т.к. для выполнения хранимой про­цедуры пользователь должен иметь привилегию.
* *Простота доступа* – хранимые процедуры позволяют инкапсулировать сложный код и оформить его в виде простого вызова.
* *Выполнение бизнес-логики* – хранимые процедуры позволяют перенести код сохранения целостности БД из прикладной программы на сервер БД. Бизнес-логика в виде храни­мых процедур не зависит от языка разработки приложения.

**Создание хранимых процедур**. Реализуется оператором

*CREATE PROCEDURE имя\_процедуры ( [ параметр [, ... ] ] )*

*[характеристика ...] тело\_процедуры*

В скобках передается необязательный список параметров, перечисленных через запятую. Каждый параметр позволяет передать в процедуру (из процедуры) входные данные (результат работы) и имеет следующий синтаксис:

*[ IN | OUT | INOUT ] имя\_параметра тип*

Ключевые слова *in, out, inout* задают направление передачи данных:

* *in* – данные передаются строго внутрь хранимой процедуры; если параметру с данным модификатором присваивается новое значение, при выхо­де из процедуры оно не сохраняется и параметр принимает значение, которое он имел до вызова;
* *out* – данные передаются строго из хранимой процедуры, если параметр имеет какое-то начальное значение, то внутри хранимой процедуры это значение во внимание не принимается;
* *inout* – значение этого параметра как принимается во внимание внутри процеду­ры, так и сохраняет свое значение при выходе из нее.

Список аргументов, заключенных в круглые скобки, присутствует всегда. Если аргументы отсутствуют, следует использовать пустой список. Если ни один из модификаторов не указан, считается, что параметр объявлен с ключевым словом *in*.

Телом процедуры является составной оператор *begin ... end*, внут­ри которого могут располагаться другие операторы:

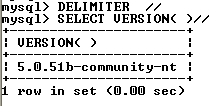
*[ label: ] BEGIN*

*statements*

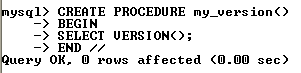
*END [ label ]*

Оператор, начинающийся с необязательной метки *label* (любое уникальное имя), может заканчиваться выражением *end label*. Внутри составного оператора *begin ... end* может находиться другой составной оператор. Если хранимая процедура содержит один запрос, то составной оператор можно не использовать.

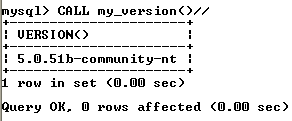
При работе с хранимыми процедурами символ точки с запятой в конце за­проса воспринимается консольным клиентом как сигнал к отправке запроса на сервер. Поэтому следует переопределить разделитель запросов – например, вместо точки с запятой использовать последовательность // :



Пример создания простейшей хранимой процедуры:

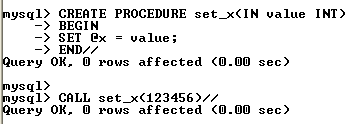


Что­бы вызвать хранимую процедуру, необходимо применить оператор *call*, после кото­рого помещается имя процедуры и ее параметры в круглых скобках:

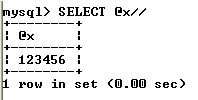
****

Рекомендуется избегать использования названий хранимых процедур, совпадающих с именами встроенных функций MySQL. В теле хранимой процедуры можно использовать многострочный комментарий, который начинается с последовательности /\* и заканчивается последо­вательностью \*/ .

Рассмотрим хранимые процедуры с параметрами. Создадим и вызовем процедуру, которая присваивает поль­зовательской переменной *@х* новое значение:

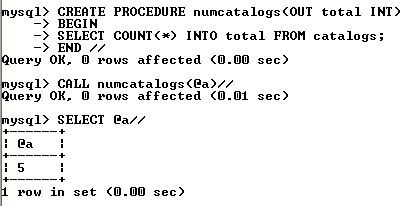


Через параметр *value* процедуре передается числовое значение 123456, которое она присваивает пользовательской переменной *@х*. Моди­фикатор *in* сообщает, что данные передаются внутрь функции. Проверим корректность работы процедуры:

****

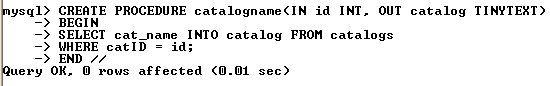
В отличие от пользовательской переменной *@х*, которая является глобальной и дос­тупна как внутри хранимой процедуры *set\_x ( )*, так и вне ее, параметры процедуры являются локальными и доступны для использования только внутри нее.

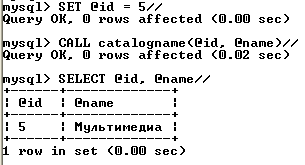
Создадим процедуру *numcatalogs()*, которая подсчитывает число записей в таблице *catalogs* базы данных *book*:



Хранимая процедура *numcatalogs()* имеет один целочисленный параметр *total*, в котором сохраняется число записей в таблице *catalogs*. Осуществляется это при помощи оператора *select ... into ... from*. В качестве параметра функции *numcatalogs()* передает­ся пользовательская переменная *@а*.

Создадим хранимую процедуру *catalogname()*, которая будет возвращать по первич­ному ключу *catID* название каталога *cat\_name*. Для этого потребует­ся определить параметр *id* с атрибутом *in*, и *catalog* с атрибутом *OUT*.

****

****

**Операторы управления потоком данных**. Хранимые процедуры позволяют реализовать сложную логику с помощью операторов ветвления и циклов. Вне хранимых процедур эти операторы применять нельзя. Ветвление программы по условию позволяет реализовать оператор:

*IF лог\_выражение THEN оператор*

*[ELSEIF лог\_выражение THEN оператор] ...*

*[ELSE оператор]*

*END IF ;*

Логическое выражение может принимать два значения:

* 0 (ложь);
* значение, отличное от нуля (истина).

Если логическое выражение истинно, то выполняется опера­тор в блоке *THEN*, иначе выполняется список операторов в блоке *else* (если блок *else* имеется). В логических выражениях можно использовать операторы сравнения ( = , >, >=, <> , <, <= ). Логические выражения можно комбинировать с помощью операторов && (И), а также || (ИЛИ). Если в блоках *if, elseif* и *else* – два или более операторов, необходимо использовать составной оператор *begin ... end*.

Множественный выбор позволяет осуществить оператор:

*CASE выражение*

*WHEN значение THEN оператор*

*[WHEN значение THEN оператор] ...*

*[ELSE оператор]*

*END CASE ;*

Выражение сравнивается со значениями. Как только найдено соответствие, выполняется соответствующий оператор. Если соответствия не найдены, выполняется оператор, размещенный после ключевого слова *else* (если оно при­сутствует).

В MySQL имеется несколько операторов, позволяющих реализовать циклы. Первый оператор цикла имеет следующий синтаксис:

*[ label: ] WHILE условие DO*

*операторы*

*END WHILE [ label ] ;*

Операторы выполняются в цикле, пока истинно условие. При каждой итерации условие прове­ряется, и если при очередной проверке оно будет ложным (0), цикл завершится. Если условие ложно с самого нача­ла, то цикл не выполнится ни разу. Если в цикле выполняется более одного оператора, не обязательно заключать их в блок *begin ... end*, т. к. эту функцию выполняет сам оператор *while*.

Досрочный выход из цикла обеспечивает оператор:

*LEAVE label ;*

Оператор прекращает выполнение блока, помеченного меткой *label* (например, прекращает выполнение цикла по достижении критического числа итераций).

Досрочное прекращение цикла также обеспечивает опе­ратор

*ITERATE label ;*

В отличие от оператора *leave* оператор *iterate* не прекращает выполнение цикла, он лишь выполняет досрочное прекращение текущей итерации. Оператор *leave* эквивалентен оператору *BREAK*, а оператор *iterate* эквивалентен оператору *continue* в С-подобных языках про­граммирования.

Второй оператор цикла имеет следующий вид:

*[ label: ] REPEAT*

*операторы UNTIL условие END REPEAT [label] ;*

Условие проверяется не в начале, а в конце оператора цикла. Таким образом, цикл выполняет по крайней мере одну итерацию независимо от условия. Следует отметить, что цикл выполняется, пока условие ложно. Оператор цикла может быть снабжен необязательной меткой *label*, по которой можно осуществлять досрочный выход из цикла при помощи операторов *leave* и *iterate*.

Реализовать бесконечный цикл позволяет оператор

*[ label: ] LOOP*

*операторы END LOOP [ label ] ;*

Цикл *loop* (в отличие от операторов *while* и *repeat*) не имеет условий выхода. По­этому данный цикл должен обязательно иметь в составе оператор *leave*.

Осуществлять безусловный переход позволяет оператор

*GOTO label ;*

Оператор осуществляет переход к оператору, помеченному меткой *label*. Это может быть как оператор *begin*, так и любой из циклов: *while*, *repeat* и *loop*. Кроме того, метка может быть не привязана ни к одному из операторов, а объявлена при помощи оператора

*LABEL label ;*

Использовать оператор *goto* для реализации циклов не рекомендуется, т. к. обычные циклы гораздо нагляднее и проще поддаются модификации, в них сложнее допустить логическую ошибку.

**Удаление хранимых процедур**. Для удаления процедур используется оператор

*DROP PROCEDURE [IF EXISTS] имя\_процедуры ;*

Если удаляемой процедуры с таким именем не существует, оператор возвращает ошибку, которую можно подавить, если использовать необязательное ключевое слово *if exists*.

**Обработчики ошибок**. При выполнении хранимых процедур могут возникать ошибки. MySQL позволяет каждой возникающей в хранимой процедуре ошибке назначить свой обработчик, который в зависимости от ситуации и серьезности ошибки может как прекратить, так и продолжить выполнение процедуры.

Для объявления такого обработчика предназначен оператор

*DECLARE тип\_обработчика HANDLER FOR код\_ошибки [, ... ] выражение;*

Выражение содержит SQL-запрос, который выполняется при срабаты­вании обработчика. Тип обработчика может принимать одно из трех значений:

* *continue* – выполнение текущей операции продолжается после выполнения опе­ратора обработчика;
* *exit* – выполнение составного оператора *begin ... end*, в котором объявлен обработчик, прекращается;
* *undo* – данный вид обработчика в текущей версии не поддерживается.

Обработчик может быть привязан сразу к нескольким ошибкам, для этого их коды следует перечислить через запятую. Код ошибки, для которой будет происходить срабатывание обработчика, может принимать следующие значения:

* *sqlstate [value] значение* – значение *sqlstate* является пятисимвольным кодом ошибки в шестнадцатеричном формате (стандарт в SQL); примеры кодов – *'hy000'*, *'hy001'*, *'42000'* и т. д.; один код обозначает сразу несколько ошибок;
* *sqlwarning* – любое предупреждение MySQL; это ключевое слово позволяет назначить обработчик для всех предупреждений; обрабатываются любые события, для которых код *sqlstate* начинается с 01;
* *not found* – любая ошибка, связанная с невозможностью найти объект (таблицу, процедуру, функцию, столбец и т. п.); обрабатываются любые события, для которых код *sqlstate* начинается с 02;
* *sqlexception* – ошибки, не охваченные ключевыми словами *sqlwarning* и *not found*;
* *mysql\_error\_code* – обычные четырехзначные ошибки MySQL, такие как 1020, 1232, 1324 и т. п.;
* именованное условие (см. ниже).

При указании кода ошибки можно использовать не только целочисленные коды, но и именованные условия, которые объявляются при помощи оператора

*DECLARE именованное условие CONDITION FOR код ошибки;*

Оператор объявляет именованное условие для кода ошибки. Так, для обрабатываемой ошибки 1062 (23000) – дублирование уникального индекса, оператор может выглядеть следующим образом:

*DECLARE 'violation' CONDITION FOR SQLSTATE '23000';*

*DECLARE 'violation' CONDITION FOR 1062;*

Первое объявление охватывает все ошибки со статусом 23000, второй вид ошибок более узкий и включает только дублирование уникального индекса.

**Курсоры**. Если результирующий запрос возвращает одну запись, поместить результаты в про­межуточные переменные можно с помощью оператора *select ... into ... from*. Однако результирующие таблицы чаще содержат несколько записей, и использование такой конструкции приводит к воз­никновению ошибки 1172: «Результат содержит более чем одну строку».

Избежать ошибки можно, добавив предложение *limit 1* или назначив *CoNTiNUE*-обработчик ошибок. Однако такая процедура реализует не то поведение, которое ожидает пользователь. Кроме того, существуют ситуации, когда требуется обработать именно многострочную результирующую таблицу.

Например, пусть требуется вернуть записи одной таблицы, отвечающие определенному условию, и на основании этих записей создать новую таблицу. Решить эту задачу можно с помощью курсоров, которые позволяют в цикле просмотреть каждую строку результирующей таблицы запросов. Работа с курсорами похожа на работу с файлами – сначала открытие курсора, затем чтение и после закрытие.

Работа с курсорами происходит по следующему алгоритму:

1. При помощи инструкции *DECLARE* *курсор CURSOR* *FOR* связывается имя курсора с выполняемым запросом.

2. Оператор *open* выполняет запрос, связанный с курсором, и устанавливает курсор перед первой записью результирующей таблицы.

3. Оператор *fetch* помещает курсор на первую запись результирующей таблицы и извлекает данные из записи в локальные переменные хранимой процедуры. По­вторный вызов оператора *fetch* приводит к перемещению курсора к следующей записи, и так до тех пор, пока записи в результирующей таблице не будут исчерпа­ны. Эту операцию удобно осуществлять в цикле.

4. Оператор *close* прекращает доступ к результирующей таблице и ликвидирует связь между курсором и результирующей таблицей.

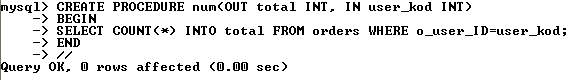
**Практическая работа**

При выполнении лабораторной работы необходимо:

* для заданной предметной области написать две хранимые процедуры и включить их в БД;
* составить отчет по лабораторной работе.

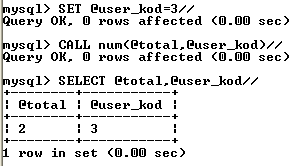
**Пример выполнения работы**

1. Создадим хранимую процедуру, которая выводит число заказов покупателя по вводимому в качестве параметра процедуры коду покупателя.

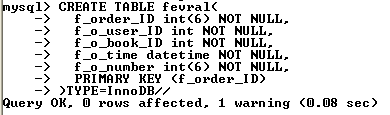


Параметр *total* является выходным, его значение равно числу заказов покупателя, код которого записывается во входной параметр *user\_kod.* Процедура считает все строки, где код клиента совпадает с параметром *user\_kod.*

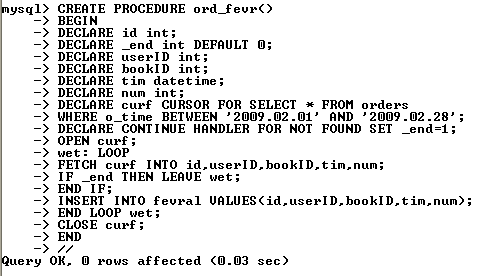
До вызова процедуры присваиваем параметру процедуры значение кода клиента. Затем вызываем процедуру оператором *CALL*. Для вывода результата можно воспользоваться оператором *SELECT*.



2. Создадим хранимую процедуру, которая записывает в новую таблицу *fevral* все заказы, сделанные в феврале 2009 г. Предварительно необходимо создать новую пустую таблицу *fevral* со структурой, аналогичной структуре таблицы *orders.*



Хранимая процедура *ord\_fevr ( )* использует курсор *curf*, который в цикле читает данные из таблицы *orders* и добавляет их в таблицу *fevral*.



Вызов процедуры осуществляется оператором *call*. Для просмотра результата выполнения процедуры используем полную выборку из таблицы *fevral*.

