Лекция № 5
MySQL - хранимые процедуры и функции

Хранимая подпрограмма (процедура или функция) - это способ инкапсуляции [1] повторяющихся действий. В хранимых подпрограммах можно объявлять переменные, управлять потоками данных, а также применять другие техники программирования.

Приведем все за и против создания подпрограмм:

За

- Разделение логики с другими приложениями. Хранимые процедуры инкапсулируют функциональность; это обеспечивает связность доступа к данным и управления ими между различными приложениями.
- Изоляция пользователей от таблиц базы данных. Это позволяет давать доступ к хранимым процедурам, но не к самим данным таблиц.
- Обеспечивает механизм защиты. В соответствии с предыдущим пунктом, если вы можете получить доступ к данным только через хранимые процедуры, никто другой не сможет испортить ваши данные, например, через команду SQL DELETE.
- Улучшение выполнения, как следствие сокращения сетевого трафика. С помощью хранимых процедур множество запросов могут быть объединены.

Против

- Повышение нагрузки на сервер баз данных в связи с тем, что большая часть работы выполняется на серверной части, а меньшая - на клиентской.
- Миграция с одной СУБД на другую (DB2, SQL Server и др.) может привести к проблемам связанным с отсутствием стандартов на оформление подпрограмм.

Шаг 1: Ставим ограничитель

Ограничитель - это символ или строка символов, который используется для указания клиенту MySQL, что вы завершили написание выражения SQL. По умолчанию ограничителем является символ точки с запятой. Тем не менее, могут возникнуть проблемы, так как в хранимой процедуре может быть несколько выражений, каждое из которых должно заканчиваться точкой с запятой. В этой лекции используется символы “//” в качестве ограничителя.

```
mysql>delimiter //
```

Шаг 2: Как работать с хранимыми процедурами

Приведем пример создания хранимой процедуры

```
delimiter //
create procedure `p2` ()
language sql
```

1 Скрытие внутренней реализации от других компонентов

```
deterministic
sql security definer
begin
select 'Hello World !';
end//
delimiter;
```

Первая строка кода задаёт новый ограничитель "//" вместо ";". Вторая создает заголовок самой процедуры, её имя `p2`. Параметры процедуры в нашем случае отсутствуют (пустые скобки обязательны). Названия хранимых процедур чувствительны к регистру. Внутри хранимой процедуры не может быть выражений, изменяющих саму базу данных.

- language: в целях обеспечения переносимости, по умолчанию указан SQL.
- deterministic: если процедура все время возвращает один и тот же результат, и принимает одни и те же входящие параметры. Это для репликации и процесса регистрации. Значение по умолчанию — not deterministic.
- SQL security: во время вызова идет проверка прав пользователя. invoker - это пользователь, вызывающий хранимую процедуру. definer - это "создатель" процедуры. Значение по умолчанию - definer.
- Comment: в целях документирования, значение по умолчанию - ""

Вызов хранимой процедуры

Чтобы вызвать хранимую процедуру, необходимо напечатать ключевое слово call, а затем название процедуры, а в скобках указать параметры (переменные или значения). Скобки обязательны.

```
call stored_procedure_name (param1, param2, ….)
call procedure1(10 , 'string parameter' , @parameter_var);
```

Изменение хранимой процедуры

В MySQL есть выражение Alter procedure для изменения процедур, но оно подходит для изменения лишь некоторых характеристик. Если вам нужно изменить параметры или тело процедуры, вам следует удалить и создать ее заново.

Удаление хранимой процедуры

```
drop procedure if exists p2;
```

Это простая команда. Выражение if exists отлавливает ошибку в случае, если такой процедуры не существует.

Шаг 3: Параметры

Расмотрим, как можно передавать в хранимую процедуру параметры.

- create procedure proc1 (): пустой список параметров
- create procedure proc1 (in varname data-type): один входящий параметр. Слово IN необязательно, потому что параметры по умолчанию - IN (входящие).
- create procedure proc1 (out varname data-type): один возвращаемый параметр.
- create procedure proc1 (inout varname data-type): один параметр, одновременно входящий и возвращаемый.

Естественно, вы можете задавать несколько параметров разных типов.

Пример параметра in

```
delimiter //
create procedure `proc_in` (in var1 int)
begin
    select var1+2 as result;
end //
```

Пример параметра out

```
create procedure `proc_out` (out var1 varchar(100))
begin
    set var1='This is text';
end //
```

Пример параметра inout

```
create procedure `proc_inout` (inout var1 int)
begin
    set var1=var1 * 2;
end //
```

Шаг 4: Переменные

Переменные можно создавать и сохранять внутри процедур. Для этого нужно объявлять их явно в начале блока begin/end, вместе с их типами данных. После объявления переменной, её можно использовать там же, как обычные переменные сессии, литералы или имена колонок.

Синтаксис объявления переменной выглядит так:

```
declare varname data-type default value;
```

Пример объявления несколько переменных:

```
declare a, b int default 5;
declare str varchar(50);
declare today timestamp default current_date;
```

Работа с переменными

Как только вы объявили переменную, вы можете задать ей значение с помощью команд SET или SELECT:

```
delimiter //
create procedure `var_proc` (IN parameter varchar(20))
begin
declare a, b, int default 5;
declare str varchar(50);
declare today timestamp default curren_date;
declare v1, v2, v3 tinyint;
insert into table1 value (a);
set str='I am a string';
select concat(str, paramstr), today from tabl2 where b>=5;
```

```
end //
```

Шаг 5: Структуры управления потоками

MySQL поддерживает конструкции IF, CASE, ITERATE, LEAVE LOOP, WHILE и REPEAT для управления потоками в пределах хранимой процедуры. Мы рассмотрим, как использовать IF, CASE и WHILE, так как они наиболее часто используются.

Конструкция IF

С помощью конструкции IF, мы можем выполнять задачи, содержащие условия:

```
delimiter //
create procedure `proc_IF (In param1 int)
begin
declare variable1 int;
set variable1=param1+1;
if variable1 = 0 then
    select variable1;
end if;
if param1 = 0 then
    select 'Parameter value = 0';
else
    select 'Parameter value <> 0';
end if;
end //
```

Конструкция CASE

CASE - это еще один метод проверки условий и выбора подходящего решения. Это отличный способ замены множества конструкций IF. Конструкцию можно описать двумя способами, предоставляя гибкость в управлении множеством условных выражений.

```
delimiter //
create procedure `proc_CASE (IN param1 int)
begin
    declare variable1 int;
    set variable1 = param1+1;
    case variable1
        when 0 then
            insert into table1 values (param1);
        when 1 then
            insert into table1 values (variable1);
        else
            insert into table1 values (99);
    end case;
end //
```

Или:

```
delimiter //
create procedure `proc_CASE (IN param1 int)
begin
    declare variable1 int;
    set variable1 = param1+1;
    case
        when variable1 = 0 then
            insert into table1 values (param1);
        when variable1 = 1 then
            insert into table1 values (variable1);
        else
            insert into table1 values (99);
    end case;
end //
```

Конструкция WHILE

Технически, существует три вида циклов: цикл WHILE, цикл LOOP и цикл REPEAT. Вы также можете организовать цикл с помощью техники программирования “Дарта Вейдера”: выражения GOTO. Вот пример цикла:

```
delimiter //
create procedure `proc_WHILE` (IN param1 int)
begin
    declare variable1, variable2 int;
    set variable1 = 0;
    while variable1 < param1 do
        insert into table1 values (param1);
        select count(*) into variable2 from table1;
        set variable1 = variable1 + 1;
    end while;
end //
```

Шаг 6: Курсоры

Курсоры используются для прохождения по набору строк, возвращенному запросом, а также обработки каждой строки.

MySQL поддерживает курсоры в хранимых процедурах. Вот краткий синтаксис создания и использования курсора.

```
declare  cursor_name cursor for select …./*Объявление курсора и его заполнение */
declare continue handler for not found  set var = 1  /*Что делать, когда больше нет записей*/
open cursor_name;                       /*Открыть курсор*/
```

fetch cursor_name into variable [, variable]; /*Назначить значение переменной, равной текущему значению столбца*/

close cursor_name; /*Закрыть курсор*/

Проведем кое-какие простые операции с использованием курсора:

```
declare //
create procedure `proc_CURSOR` (OUT param1 int)
begin
    declare a, b int;
    declare cur1 cursor for select col1 from table1;
    declare continue handler for not found set b = 1;
    open cur1;
    set b = 0;
    set c = 0;
    while b = 0 do
        fetch cur1 into a;
        if b = 0 then
            set c = c+a;
        end if;
    end while;
    set param1 = c;
end //
```

У курсоров есть три свойства, которые вам необходимо понять, чтобы избежать получения неожиданных результатов:

- Не чувствительный: открывшийся однажды курсор не будет отображать изменения в таблице, происшедшие позже. В действительности, MySQL не гарантирует то, что курсор обновится, так что не надейтесь на это.
- Доступен только для чтения: курсоры нельзя изменять.
- Без перемотки: курсор способен проходить только в одном направлении - вперед, вы не сможете пропускать строки, не выбирая их.

Заключение

В этой лекции мы познакомились с основами работы с хранимыми процедурами и с некоторыми специфическими свойствами, связанными с ней. Конечно, нужно будет углубить знания в таких областях, как безопасность, выражения SQL и оптимизация, прежде чем стать настоящим гуру MySQL процедур.

Вы должны подсчитать, какие преимущества даст вам использование хранимых процедур в вашем конкретном приложении, и только потом создавать лишь необходимые процедуры. В общем, их стоит внедрять в проекты в следствие их безопасности, обслуживания кода и общего дизайна. К тому же, не забывайте, что над процедурами

MySQL все еще ведется работа. Ожидайте улучшений, касающихся функциональности и улучшений.