MySQL

Урок 4

Объединение запросов, хранимые процедуры, триггеры, функции

Для чего нужны view, хранимые процедуры, триггеры и функции.

VIEW

Очень полезная возможность – сохранить запрос, который часто выполняется. Делается это с помощью простой команды.

Синтаксис команды:

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Примеры

```
CREATE VIEW Londonstaff
           AS SELECT *
           FROM Salespeople
           WHERE city = 'London';
```

Обращение происходит как к простой таблице:

```
SELECT * FROM Londonstaff;
```

Можно изменять представление, но изменения будут проходить не в представлении (так как оно не содержит никакой реальной таблицы), а в таблице, на которую это представление указывает.

```
CREATE VIEW Salesown
           AS SELECT snum, sname, city
           FROM Salespeople;
```

```
UPDATE Salesown
             SET city = 'Palo Alto'
             WHERE snum = 1004;
```

Чтобы удалить представление, используется команда:

```
DROP VIEW < view name >
```

Хранимые процедуры и функции

Сперва стоит разобраться, зачем могут понадобиться хранимые процедуры и функции.
Все запросы, которые выполняет MySQL, проходят анализатор и компилируются на стороне сервера. Каждый раз на это может уходить значительное время. Одинаковые запросы будут автоматически кешироваться, что позволит выполнять их, минуя шаги анализа и компиляции.
Если в приложении есть очень сложные логические части, которые включают в себя циклические выборки, очень большие аналитические задачи, их можно выполнить намного быстрее с помощью хранимых процедур и функций.
Не стоит делать все простые операции (скажем, CRUD) в приложении на хранимых процедурах и функциях, так как значительного прироста производительности за счет этого не добиться, но управляемость кода, который вы пишете, очень сильно упадет. Подобным способом решаются только сложные аналитические задачи.
Синтаксис создания хранимых процедур и функций.

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body
CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body
proc_parameter:
    [ IN | OUT | INOUT ] param_name type
func_parameter:
    param_name type
type:
    Any valid MySQL data type
characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
routine_body:
    Valid SQL routine statement
```

Примеры

Для того, чтобы создать процедуру, нужно изменить служебную переменную delimiter, которая служит для обозначения конца команды, т. е. в конце команды можно использовать не «;», а любой другой символ. Это нужно, чтобы внутри хранимой процедуры можно было легко использовать символ «;» и анализатор не думал, что это конец команды.

В процедуре бывает 2 типа параметров – входные и выходные. Во входных параметрах значения передаются в функцию, через выходные параметры значения можно записать, например, в переменную.

Объявим процедуру с выходным параметром.

```
mysql> delimiter //

mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
    -> BEGIN
    ->   SELECT COUNT(*) INTO param1 FROM t;
    -> END//
Query OK, 0 rows affected (0.00 sec)
mysql> delimiter ;
```

Пример вызова процедуры:

```
mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @a;
+------+
| @a   |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

Давайте теперь разберемся, что происходит в примере выше.

1. Мы объявили переменную @a. В MySQL переменные объявляются с помощью символа @.
2. CALL simpleproc(@a); мы вызвали процедуру и записали результат в переменную, так как параметр является выходным.
3. SELECT @a; вывели результат на экран (вывели значение переменной).

Функция может принимать входящие параметры, а также возвращать значения. Этот подход может быть намного проще, не нужно объявлять переменные, результат сразу передается в SELECT.

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
    -> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

В функции определяется, какие параметры она будет принимать, а также возвращать.

1. s CHAR(20) принимает параметр размером до 20 символов.

2. RETURNS CHAR(50) DETERMINISTIC возвращает до 50 символов.
3. SELECT hello('world'); – вызываем функцию и выводим результат на экран.

Как вы видите, синтаксис хранимых процедур и функций довольно прост, не отличается от подхода в любых других языках программирования.

Триггеры

Чтобы в СУБД запускались действия по событиям (например, после удаления данных в определенной таблице), используются триггеры.

Про триггеры можно сказать следующее. Они очень удобны для обработки каких-либо событий в СУБД: будучи установленным один раз, обработчик будет работать и выполнять свою функцию. Но есть и ряд проблем, при которых подобный подход может создать трудности – например, при отладке иногда достаточно сложно понять, что какие-то действия обрабатывает триггер. Поэтому многие считают, что это плохой паттерн, при его использовании отладка очень сложна, особенно если триггеров очень много.

Главный плюс триггеров: они минуют долгую фазу анализа и компиляции запроса.

Из этого можно сделать вывод: для использования триггеров нужно иметь четкую спецификацию ПО, в которой будет описаны места кода, где они используются. Триггеры нельзя использовать бездумно и вешать без предварительного описания, а также не взвесив все за и против данного подхода.

Пример создания триггеров

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```

Практическая работа

Используя БД «Сотрудники» (которую мы получили на предыдущем уроке), давайте создадим VIEW.

```
CREATE VIEW view_managers
   AS
 SELECT d.*, e.first_name, e.last_name
 FROM departments d
  LEFT JOIN dept_manager m
    ON d.dept_no = m.dept_no
  LEFT JOIN employees e
    ON m.emp_no = e.emp_no
```

Создадим функцию:

```
delimiter //
```

```
drop function if exists emp_dept_id //
--
-- returns the department id of a given employee
--
create function emp_dept_id( employee_id int )
returns char(4)
reads sql data
begin
    declare max_date date;
    set max_date = (
        select
            max(from_date)
        from
            dept_emp
        where
            emp_no = employee_id
    );
    set @max_date=max_date;
    return (
        select
            dept_no
        from
            dept_emp
        where
            emp_no = employee_id
            and
            from_date = max_date
            limit 1
    );
end //
```

Практическое задание

1. Создать на основе запросов, которые вы сделали в ДЗ к уроку 3, VIEW.
2. Создать функцию, которая найдет менеджера по имени и фамилии.
3. Создать триггер, который при добавлении нового сотрудника будет выплачивать ему вступительный бонус, занося запись в таблицу salary.

Дополнительные материалы

1. https://www.w3schools.com/sql/

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. http://dev.mysql.com/doc/refman/5.7/en/create-view.html
2. http://www.mysql.ru/docs/gruber/mg20.html
3. http://dev.mysql.com/doc/refman/5.7/en/create-procedure.html
4. http://dev.mysql.com/doc/refman/5.7/en/trigger-syntax.html