# Лабораторная работа 22. Представления, хранимые процедуры, функции, триггеры MySQL

http://www.zoonman.ru/library/mysql\_sr\_and\_t.htm

Цель: изучение создания представлений, хранимых процедур, функций, триггеров в MySQL.

Для выполнения заданий должны быть привилегии create\_routine\_priv, alter\_routine\_priv, execute\_priv, trigger\_priv, show\_view\_priv, create\_view\_priv. Если их нет, надо остановить службу MySQL, затем в папке MySQL Server в файле my.ini временно убрать знак «#» перед строкой «skip-grant-tables» и запустить mysqld.exe или перезагрузить компьютер с сервером MySQL.

Теоретическая часть

Хранимые процедуры представляют собой набор команд SQL, которые могут компилироваться и храниться на сервере. Таким образом, вместо того, чтобы хранить часто используемый запрос, клиенты могут ссылаться на соответствующую хранимую процедуру. Это обеспечивает лучшую производительность, поскольку данный запрос должен анализироваться только однажды и уменьшается трафик между сервером и клиентом. Концептуальный уровень можно также повысить за счет создания на сервере библиотеки функций.

Триггер представляет собой хранимую процедуру, которая активизируется при наступлении определенного события. Например, можно задать хранимую процедуру, которая срабатывает каждый раз при удалении записи из транзакционной таблицы - таким образом, обеспечивается автоматическое удаление соответствующего заказчика из таблицы заказчиков, когда все его транзакции удаляются.

Хранимые программы (процедуры и функции) поддерживаются в MySQL 5.0. Хранимые процедуры – набор SQL-выражений, который может быть сохранен на сервере. Как только это сделано, клиенту уже не нужно повторно передавать запрос, а требуется просто вызвать хранимую программу.

Это может быть полезным тогда, когда:

* многочисленные приложения клиента написаны в разных языках или работают на других платформах, но нужно использовать ту же базу данных операций
* безопасность на 1 месте

 Хранимые процедуры и функции (подпрограммы) могут обеспечить лучшую производительность потому, что меньше информации требуется для пересылки между клиентом и сервером. Выбор увеличивает нагрузку на сервер БД, но снижает затраты на стороне клиента. Используйте это, если много клиентских машин (таких как Веб-серверы) обслуживаются одной или несколькими БД.

Хранимые подпрограммы также позволяют вам использовать библиотеки функций, хранимые в БД сервера. Эта возможность представлена для многих современных языков программирования, которые позволяют вызывать их непосредственно (например, используя классы).

MySQL следует в синтаксисе за SQL:2003 для хранимых процедур, который уже используется в IBM's DB2.

Хранимые процедуры требуют наличия таблицы proc в базе mysql. Эта таблица обычно создается во время установки.

При создании, модификации, удалении хранимых подпрограмм сервер манипулирует с таблицейmysql.proc

Начиная с MySQL 5.0.3 требуются следующие привилегии:

**CREATE ROUTINE** для создания хранимых процедур

**ALTER ROUTINE** необходимы для изменения или удаления процедур. Эта привилегия автоматически назначается создателю  процедуры (функции)

**EXECUTE** привилегия потребуется для выполнения подпрограммы. Тем не менее, автоматически назначается создателю процедуры (функции). Также, по умолчанию, SQL SECURITY параметр для подпрограммы DEFINER , который разрешает пользователям, имеющим доступ к БД вызывать подпрограммы, ассоциированные с этой БД.

Синтаксис хранимых процедур и функций

Хранимая подпрограмма представляет собой процедуру или функцию. Хранимые подпрограммы создаются с помощью выражений CREATE PROCEDURE или CREATE FUNCTION. Хранимая подпрограмма вызывается, используя выражение CALL , причем только возвращающие значение переменные используются в качестве выходных. Функция может быть вызвана подобно любой другой функции и может возвращать скалярную величину. Хранимые подпрограммы могут вызывать другие хранимые подпрограммы.

Начиная с MySQL 5.0.1, загруженная процедура или функция связана с конкретной базой данных. Это имеет несколько смыслов:

* Когда подпрограмма вызывается, то подразумевается, что надо произвести вызов USE db\_name (и отменить использование базы, когда подпрограмма завершилась, и база больше не потребуется)
* Вы можете квалифицировать обычные имена с именем базы данных. Это может быть использовано, чтобы ссылаться на подпрограмму, которая - не в текущей базе данных. Например, для выполнения хранимой процедуры p или функции f которые связаны с БД test, вы можете сказать интерпретатору команд так: CALL test.p() или test.f().
* Когда база данных удалена, все загруженные подпрограммы связанные с ней тоже удаляются. В MySQL 5.0.0, загруженные подпрограммы - глобальные и не связанны с базой данных. Они наследуют по умолчанию базу данных из вызывающего оператора. Если USE db\_name выполнено в пределах подпрограммы, оригинальная текущая БД будет восстановлена после выхода из подпрограммы (Например текущая БД db\_11, делаем вызов подпрограммы, использующей db\_22, после выхода из подпрограммы остается текущей db\_11 )

**Синтаксис:**

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

**CREATE FUNCTION** имя\_функции ([параметр\_функции[,...]])
**RETURNS** тип
[характеристика ...] тело\_подпрограммы

параметр\_процедуры:
 [ **IN** | **OUT** | **INOUT** ] имя\_параметра тип
параметр\_функции:
 имя\_параметра тип

тип:
 Любой тип данных MySQL

характеристика:
 LANGUAGE SQL
 | [NOT] DETERMINISTIC
 | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
 | SQL SECURITY { DEFINER | INVOKER }
 | COMMENT 'string'

тело\_подпрограммы:
 Правильное SQL выражение.

Хранимые процедуры

**Задание 1.**

Запустите MySQL Query Browser. Создайте схему bd2 или с другим именем и используйте её, если это возможно, иначе используйте доступную схему. Создайте в используемой схеме таблицу t3 или с другим именем, содержащую столбцы val и summa типа integer. Введите в таблицу строки, содержащие числа. Нажмите ПКМ в используемой схеме и выберите «Create New Procedure / Function». Введите название процедуры, например, «mypr». Откроется новое окно, содержащее текст:

DELIMITER $$

DROP PROCEDURE IF EXISTS `bd2`.`mypr` $$

CREATE PROCEDURE `bd2`.`mypr` ()

BEGIN

END $$

DELIMITER ;

Измените этот текст на следующий и сохраните в файл, например, mypr1.sql.

Скрипт 1 (mypr1.sql):

DELIMITER $$

DROP PROCEDURE IF EXISTS `bd2`.`mypr` $$

CREATE PROCEDURE `bd2`.`mypr` (OUT param1 INTEGER))

BEGIN

 select val1+summa INTO 'param1' from `t3` LIMIT 0,1;

END $$

DELIMITER ;

Нажмите кнопку execute для выполнения. То же сделайте для следующего текста, сохраняя с другим именем, например sp1.sql.

Скрипт 2 (sp1.sql):

DELIMITER $$

DROP PROCEDURE IF EXISTS `bd2`.`sp` $$

CREATE PROCEDURE `bd2`.`sp` (OUT param1 INTEGER))

BEGIN

 select COUNT(\*) INTO 'param1' from `t3`;

END $$

DELIMITER ;

Запустите MySQL с командной строки, как показано ниже (имя пользователя может быть другим).

C:\Program Files\MySQL\MySQL Server 5.5\bin>mysql.exe -u user -p

Enter password: \*\*\*\*\*\*\*\*

Перейдите к используемой БД:

mysql> use bd2

Database changed

Вызовите первую процедуру:

mysql> call mypr(@a);

Query OK, 1 row affected (0.00 sec)

Выведите полученное процедурой значение:

mysql> select@a;

+------+

| @a |

+------+

| 21 |

+------+

1 row in set (0.00 sec)

Вызовите вторую процедуру:

mysql> call sp(@a);

Query OK, 1 row affected (0.00 sec)

Выведите полученное процедурой значение:

mysql> select@a;

+------+

| @a |

+------+

| 3 |

+------+

1 row in set (0.02 sec)

Database changed

mysql> call mpr1(@a);

ERROR 1305 (42000): PROCEDURE bd2.mpr1 does not exist

mysql> call mypr(@a);

Query OK, 1 row affected (0.00 sec)

mysql> select@a;

+------+

| @a |

+------+

| 21 |

+------+

1 row in set (0.00 sec)

mysql> call mypr(@a);

Query OK, 1 row affected (0.00 sec)

mysql> select@a;

+------+

| @a |

+------+

| 21 |

+------+

1 row in set (0.00 sec)

Триггеры

Поддержка триггеров появилась в MySQL начиная с версии 5.0.2.

**Триггер** – поименованный объект БД, который ассоциирован с таблицей и активируемый при наступлении определенного события, события связанного с этой таблицей.

**Задание 2.**

Выполните нижеприведенные коды, первый из которых создает таблицу и INSERT триггер. Триггер суммирует значения, вставляемые в один из столбцов  таблицы.

mysql> CREATE TABLE account (acct\_num INT, amount DECIMAL(10,2));

Query OK, 0 rows affected (0.09 sec)

mysql> CREATE TRIGGER ins\_sum BEFORE INSERT ON account FOR EACH ROW SET @sum = @sum + NEW.amount;

Query OK, 0 rows affected (0.05 sec)

Объявим переменную sum и присвоим ей значение 1. После этого при каждой вставке в таблицу **account** значение этой переменной будет увеличивать согласно вставляемой части.

**Замечание**. Если значение переменной не инициализировано, то триггер работать не будет.

mysql> set @sum=1;

Query OK, 0 rows affected (0.00 sec)

mysql> insert into account values (25, 70);

Query OK, 1 row affected (0.05 sec)

mysql> select @sum;

+-------+

| @sum |

+-------+

| 71.00 |

+-------+

1 row in set (0.00 sec)

**Синтаксис создания триггера**

**CREATE**
[DEFINER = { имя\_ пользователя | CURRENT\_USER }]
**TRIGGER** имя\_триггера время\_триггера событие\_срабатывания\_триггера
ON имя\_таблицы FOR EACH ROW выражение\_выполняемое\_при\_срабатывании\_триггера

**время\_триггера - о**пределяет время свершения действия триггера. BEFORE означает, что триггер выполнится до завершения события срабатывания триггера, а AFTER означает, что после. Например, при вставке записей (см. пример выше) триггер срабатывал до фактической вставки записи и вычислял сумму. Такой вариант уместен при предварительном вычислении каких-то дополнительных полей в таблице или параллельной вставке в другую таблицу.

**событие\_срабатывания\_триггера:**

* INSERT: т.е. при операциях вставки или аналогичных ей выражениях (INSERT, LOAD DATA, и REPLACE)
* UPDATE: когда сущность (строка) модифицирована
* DELETE: когда запись удаляется (запросы, содержащие выражения DELETE и/или REPLACE)

Представления

Ссылка: http://www.plam.ru/compinet/mysql\_rukovodstvo\_professionala/p151.php

Синтаксис CREATE VIEW

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]

Эта инструкция создает новый view или заменяет существующий, если дано предложение OR REPLACE. Инструкция SELECT *select\_statement* обеспечивает определение view. Инструкция может выбирать из основных таблиц или других views.

Эта инструкция требует привилегии CREATE VIEW для view и некоторой привилегии для каждого столбца, выбранного инструкцией SELECT. Для столбцов, используемых в другом месте в инструкции SELECT, надо иметь привилегию SELECT. Если присутствует предложение OR REPLACE, надо также иметь привилегию DROP для view.

Каждый view принадлежит базе данных. По умолчанию, новый view создан в заданной по умолчанию базе данных. Чтобы явно создавать view в указанной базе данных, можно определить его имя как *db\_name.view\_name*.

mysql> CREATE VIEW test.v AS SELECT \* FROM t;

Основные таблицы и views совместно используют то же самое пространство имен внутри базы данных, так что база данных не может содержать основную таблицу и view, которые имеют то же самое имя.

Views должны иметь уникальные имена столбца без дубликатов, точно так же, как основные таблицы. По умолчанию, имена столбцов, найденных инструкцией SELECT используются для имени столбца view. Чтобы определять явные имена для столбцов view, может быть задано факультативное предложение *column\_list* как список разделяемых запятой идентификаторов. Число имен в *column\_list* должно быть таким же, как число столбцов, найденных командой SELECT.

Столбцы, найденные инструкцией SELECT, могут быть простыми ссылками к столбцам таблицы. Они также могут быть выражениями, которые используют функции, постоянные значения, операторы и т.д.

Неквалифицированная таблица или имя view в операторе SELECT интерпретируется относительно заданной по умолчанию базы данных. view может обратиться к таблицам или view в других базах данных, квалифицируя имя таблицы или view с соответствующим именем базы данных.

view может быть создан из многих видов инструкций SELECT. Он может обратиться к основным таблицам или другим view. Это может использовать объединения, UNION и подзапросы. SELECT не обязан обращаться к каким-либо таблицам.

**Задание 3.**

Выполните следующий пример, который определяет view, который выбирает два столбца из другой таблицы, также как выражение, вычисленное из данных этих столбцов:

mysql> CREATE VIEW v AS SELECT acct\_num, amount, acct\_num\*amount AS value FROM account;

Query OK, 0 rows affected (0.06 sec)

mysql> SELECT \* FROM v;

+----------+--------+---------+

| acct\_num | amount | value |

+----------+--------+---------+

| 25 | 70.00 | 1750.00 |

+----------+--------+---------+

1 row in set (0.00 sec)

Запустите полученное представление в MySQL Query Browser.

**Контрольные вопросы:**

1. Создание и использование представлений в MySQL.

2. Создание и использование триггеров в MySQL.

3. Создание и использование хранимых процедур в MySQL.