ТЕХНОТРЕК

Занятие №6

Хранимые подпрограммы

Дина Сафина

Хранимые подпрограммы

1. Что такое хранимые подпрограммы
2. Синтаксис SQL/PSM
3. Flow Control Statement
4. Курсоры
5. Триггеры
6. Хранимые функции и процедуры
7. События

Persistent Stored Modules –
постоянно хранимые модули

Типы хранимых подпрограмм

- Триггеры
- Хранимые процедуры
- Хранимые функции
- События

Почему это хорошо

- Код к данным, а не данные к коду
- Переиспользование кода
- Скрывает сложность базы данных
- Упрощение версионирования и сопровождения
- Более точное управление привелегиями
- Планы исполнения кэшируются
- Сопровождение средствами базы данных
- Разделение области ответственности между разработчиками приложений и баз данных

Почему это плохо

- У MySQL нет хороших инструментов отладки
- Медленный и примитивный язык
- SQL/PSM слабее T-SQL и PL/SQL
- Разворачивание БД усложняется
- Пагубно отражается на производительности
- Могут возникнуть дополнительные уязвимости
- Возрастает нагрузка на сервер базы данных
- Возможности управления ресурсами слабые
- Трудно профилировать
- Плохо работает с репликами

ТЕХНОТРЕК

Синтаксис SQL/PSM

Строковые литералы

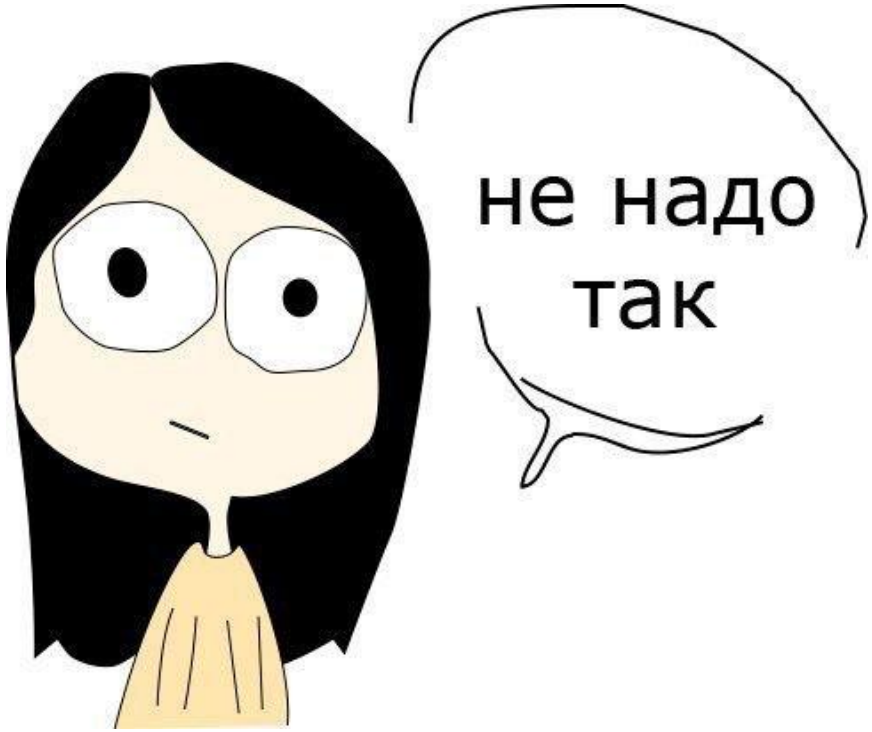
```
mysql> SELECT 'hello', '"hello"', ''"hello"'', 'hel''lo', '\'hello';
+-------+---------+-----------+--------+--------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello";
+-------+---------+-----------+--------+--------+
| hello | 'hello' | ''hello'' | hel"lo | "hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT 'This\nIs\nFour\nLines';
+--------------------+
| This
Is
Four
Lines |
+--------------------+

mysql> SELECT 'disappearing\ backslash';
+------------------------+
| disappearing backslash |
+------------------------+
```

Булевы литералы

```
mysql> SELECT TRUE, true, FALSE, false;
        -> 1, 1, 0, 0
```

Зарезервированные слова

https://dev.mysql.com/doc/refman/8.0/en/keywords.html



Примеры выражений

- expr OR expr
- expr IS [NOT] {TRUE | FALSE | UNKNOWN}
- literal
- identifier
- function_call
- variable
- – expr
- EXISTS (subquery)
- case_expr
- ROW (expr, expr [, expr] ...)

Комментарии

```
mysql> SELECT 1+1;     # This comment continues to the end of line
mysql> SELECT 1+1;     -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

```
/*! MySQL-specific code */
```

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

BEGIN ... END

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]
```

- Для составных выражений
- Разделители – точки с запятой
- Могут быть вложенные
- Могут быть именованными
- Из CLI символ окончания команды заменяют

Именованные блоки

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]

[begin_label:] LOOP
    statement_list
END LOOP [end_label]

[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]

[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

Пример именованных блоков

```
1  CREATE PROCEDURE doiterate(p1 INT)
2  BEGIN
3    label1: LOOP
4      SET p1 = p1 + 1;
5      IF p1 < 10 THEN ITERATE label1; END IF;
6      LEAVE label1;
7    END LOOP label1;
8  END;
```

Local Variable DECLARE Syntax

```
1  DECLARE var_name [, var_name] ... type [DEFAULT value]
```

- Определяются внутри блока BEGIN … END
- Могут быть константой или выражением
- Типы общие с типами для таблиц
- Определяются до использования
- Без учёта регистра
- Не называйте переменные как поля таблицы

User-Defined Variables

- ***@var_name***
- Хранится до конца сессии пользователя
- Не чувствительны к регистру

```
SET @var_name = expr [, @var_name = expr] ...
```

```
SET @name = 43;
SET @total_tax = (SELECT SUM(tax) FROM taxable_transactions);
```

```
SET GLOBAL max_connections = 1000;
SET @@global.max_connections = 1000;
```

ТЕХНОТРЕК

Flow Control Statement

Flow Control Statements

- CASE
- IF
- ITERATE, LEAVE & RETURN
- LOOP
- REPEAT
- WHILE

CASE Syntax

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

CASE Example

```
1  DELIMITER |
2
3  CREATE PROCEDURE p()
4    BEGIN
5      DECLARE v INT DEFAULT 1;
6
7      CASE v
8        WHEN 2 THEN SELECT v;
9        WHEN 3 THEN SELECT 0;
10       ELSE
11         BEGIN
12         END;
13     END CASE;
14   END;
15   |
```

IF Syntax

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

IF Example

```
DELIMITER //

CREATE FUNCTION SimpleCompare(n INT, m INT)
  RETURNS VARCHAR(20)

  BEGIN
    DECLARE s VARCHAR(20);

    IF n > m THEN SET s = '>';
    ELSEIF n = m THEN SET s = '=';
    ELSE SET s = '<';
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m);

    RETURN s;
  END //

DELIMITER ;
```

ITERATE, LEAVE & RETURN

```
ITERATE label
```

```
LEAVE label
```

```
RETURN expr
```

LOOP Syntax

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN
      ITERATE label1;
    END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END;
```

REPEAT Syntax

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

REPEAT Example

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT
    ->     SET @x = @x + 1;
    ->   UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

WHILE Syntax

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END;
```

ТЕХНОТРЕК

Курсоры

О курсорах

- Используются только в хранимых подпрограммах
- Могут быть вложенными
- Однонаправленные
- Read Only
- Используют временные таблицы
- Исполняют весь запрос в момент открытия курсора

CURSOR Syntax

```
DECLARE cursor_name CURSOR FOR select_statement
```

```
FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name] ...
```

```
OPEN cursor_name
```

```
CLOSE cursor_name
```

CURSOR Example

```
1  CREATE PROCEDURE curdemo()
2  BEGIN
3    DECLARE done INT DEFAULT FALSE;
4    DECLARE a CHAR(16);
5    DECLARE b, c INT;
6    DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
7    DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
8    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
9
10   OPEN cur1;
11   OPEN cur2;
12
13   read_loop: LOOP
14     FETCH cur1 INTO a, b;
15     FETCH cur2 INTO c;
16     IF done THEN
17       LEAVE read_loop;
18     END IF;
19     IF b < c THEN
20       INSERT INTO test.t3 VALUES (a,b);
21     ELSE
22       INSERT INTO test.t3 VALUES (a,c);
23     END IF;
24   END LOOP;
25
26   CLOSE cur1;
27   CLOSE cur2;
28 END;
```

ТЕХНОТРЕК

Триггеры

Триггеры

Триггер – это хранимая процедура особого типа, которую пользователь не вызывает непосредственно, исполнение которой обусловлено действием по модификации данных.

Триггеры – это плохо

Пожалуйста,
не используйте триггеры!

Почему?

- **Непрозрачная логика**
- Внезапный объём невидимой работы
- Отличный способ получить deadlock
- Если в триггере ошибка, падает исходный запрос
- Трудно отлаживать

Триггеры в MySQL

- На одно событие в таблице может быть несколько триггеров
- Только FOR EACH ROW
- Не атомарны в MyISAM
- ROW_COUNT() = 1 кроме первой строки
- Нельзя использовать команды, явно или неявно влияющие на транзакцию
- Нельзя повесить триггеры на таблицу в INFORMATION_SCHEMA

И зачем они нужны?

- Проверка ограничений
- Поддержание денормализованных таблицы
- Значение create_dttm/change_dttm по умолчанию
- Журналирование обновлений

CREATE TRIGGER Syntax

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

Пример триггера

```
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 (0.01 sec)
```

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----------------------+
| Total amount inserted |
+-----------------------+
|               1852.48 |
+-----------------------+
```

Ещё пример триггера

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
       FOR EACH ROW
       BEGIN
           IF NEW.amount < 0 THEN
               SET NEW.amount = 0;
           ELSEIF NEW.amount > 100 THEN
               SET NEW.amount = 100;
           END IF;
       END;//
mysql> delimiter ;
```

... TRIGGER Syntax

```
SHOW CREATE TRIGGER trigger_name
```

```
SHOW TRIGGERS
    [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

ТЕХНОТРЕК

Хранимые функции и процедуры

Хранимые функции и процедуры

Хранимые функции и процедуры – это набор SQL/PSM-операторов, хранящихся на сервере MySQL

Хранимая функция в отличие от хранимой процедуры возвращает значение и может использоваться в SQL

О хранимых функциях и процедурах

- Параметры нечувствительны к регистру
- Параметры могут быть **IN**, **OUT** и **INOUT**
- По-умолчанию все параметры в процедурах **IN**
- В функция параметры всегда **IN**
- **IN:** не возвращает изменённые значения
- **OUT:** только передаёт значение наружу
- **INOUT:** может изменяться внутри процедуры
- Внутри функции нельзя завершать транзакции

CREATE Syntax

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [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
```

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body
```

ROCEDURE Example

```
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 ;

mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a;
+------+
| @a   |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

FUNCTION Example

```
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)
```

CALL Syntax

```
CALL sp_name([parameter[,...]])
CALL sp_name[()]
```

```
mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+--------------------+------------+
| @version           | @increment |
+--------------------+------------+
| 8.0.3-rc-debug-log |         11 |
+--------------------+------------+
```

ALTER Syntax

```
ALTER PROCEDURE proc_name [characteristic ...]

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

```
ALTER FUNCTION func_name [characteristic ...]

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

DROP Syntax

```
1   DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

UDF – User-Defined Functions

- Пишутся на любом языке программирования
- Платформенно-зависимые
- Ошибка в коде может привести к остановке сервера
- Вне транзакционного контекста
- Легко написать агрегатные функции

ТЕХНОТРЕК

События

События

Событие – это хранимый код, выполняемый в нужный момент времени или с заданным интервалом.

О событиях

- Не принимают параметров и не возвращают значения
- Метаинформация: SHOW EVENTS
- Информация о запусках: INFORMATION_SCHEMA.EVENTS
- Используются для перестроения кэшированных и сводных таблиц, мониторинга и диагностики
- Обычно логику помещают в хранимые процедуры, а из Событий вызываются
- Для исключения параллельного запуска можно использовать блокировки
- Текущие события: SHOW PROCESSLIST

CREATE EVENT Syntax

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    EVENT
    [IF NOT EXISTS]
    event_name
    ON SCHEDULE schedule
    [ON COMPLETION [NOT] PRESERVE]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'string']
    DO event_body;

schedule:
    AT timestamp [+ INTERVAL interval] ...
  | EVERY interval
    [STARTS timestamp [+ INTERVAL interval] ...]
    [ENDS timestamp [+ INTERVAL interval] ...]

interval:
    quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
              WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
              DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

SCHEDULE Examples

- CURRENT_TIMESTAMP
- CURRENT_TIMESTAMP + INTERVAL '2:10' MINUTE_SECOND
- CURRENT_TIMESTAMP + INTERVAL 3 WEEK + INTERVAL 2 DAY
- EVERY 6 WEEK
- EVERY 2 WEEK STARTS CURRENT_TIMESTAMP + INTERVAL '6:15' HOUR_MINUTE

CREATE EVENT Example

```
CREATE EVENT myevent
    ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
```

```
CREATE EVENT e_hourly
    ON SCHEDULE
      EVERY 1 HOUR
    COMMENT 'Clears out sessions table each hour.'
    DO
      DELETE FROM site_activity.sessions;
```

```
CREATE EVENT e_call_myproc
    ON SCHEDULE
      AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
    DO CALL myproc(5, 27);
```

CREATE EVENT Example

```
delimiter |

CREATE EVENT e
    ON SCHEDULE
      EVERY 5 SECOND
    DO
      BEGIN
        DECLARE v INTEGER;
        DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN END;

        SET v = 0;

        WHILE v < 5 DO
          INSERT INTO t1 VALUES (0);
          UPDATE t2 SET s1 = s1 + 1;
          SET v = v + 1;
        END WHILE;
    END |

delimiter ;
```

ALTER EVENT Syntax

```
ALTER
    [DEFINER = { user | CURRENT_USER }]
    EVENT event_name
    [ON SCHEDULE schedule]
    [ON COMPLETION [NOT] PRESERVE]
    [RENAME TO new_event_name]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'string']
    [DO event_body]
```

ALTER EVENT Example

```
ALTER EVENT myevent
    ON SCHEDULE
      EVERY 12 HOUR
    STARTS CURRENT_TIMESTAMP + INTERVAL 4 HOUR;
```

```
ALTER EVENT myevent
    ON SCHEDULE
      AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
    DO
      TRUNCATE TABLE myschema.mytable;
```

```
ALTER EVENT myevent
    DISABLE;
```

ALTER EVENT Example

```
1  ALTER EVENT myevent
2      RENAME TO yourevent;
```

```
1  ALTER EVENT olddb.myevent
2      RENAME TO newdb.myevent;
```

DROP EVENT Syntax

```
1  DROP EVENT [IF EXISTS] event_name
```

Литература для чтения

- Бэрон Шварц, Петр Зайцев, Вадим Ткаченко, «MySQL по максимуму»
- Guy Harrison, Steven Feuerstein, «MySQL Stored Procedure Programming»
- Библиотека хранимых процедур: http://mysql-sr-lib.sourceforge.net/
- Библиотека UDF-процедур: https://www.mysqludf.org
- https://dev.mysql.com/doc/refman/8.0/en/

Домашнее задание №6

Разработать скрипты для автоматического заполнения таблиц базы проектов.

В основных таблицах должно быть не менее 100 записей.

Выложить на GitHub.

Срок сдачи

31 октября 2018