БАЗЫ ДАННЫХ

Хранимые функции, процедуры и триггеры

Программирование в СУБД

В процессе работы с базой данных (часто в административных целях) возникает необходимость автоматизации рутинных действий, таких как выполнение логики, непосредственно связанной с данными.

Для программирования в СУБД предусмотрены следующие виды подпрограмм:

- Функции
- Процедуры
- Триггеры

Плюсы и минусы

Минусы	**Плюсы**
- Размазывание бизнес-логики - Примитивные возможности SQL - Непереносимость между разными СУБД - Высокая стоимость разработки	- Скорость обработки данных - Сокрытие структуры данных - Снижение вероятности SQL-инъекций - Повышение целостности и непротиворечивости данных

Разделитель

При написании кода хранимых функций, процедур и триггеров довольно часто возникает необходимость в смене разделителя SQL-инструкций, т.к. стандартный разделитель `;` может встречаться внутри кода подпрограмм.

```
DELIMITER //
код вашей подпрограммы;
код вашей подпрограммы;
DELIMITER ;
```

Функции

Функции выполняют частную атомарную логику над небольшими данными (например, значения столбцов). Функции обязательно возвращают значение, поэтому нужно указать его тип.

```
DELIMITER //

CREATE FUNCTION multiply(s INT) RETURNS INTEGER
BEGIN
  SET s = s * 2;
  RETURN s;
END//

DELIMITER ;
```

Вызов функций

Для вызова функции необходимо указать ее имя и передать аргументы:

```
SELECT multiply(3);
```

```
SELECT id, multiply(money) FROM accounts;
```

Процедуры

Процедуры позволяют инкапсулировать работу с данными, при этом они могут обрабатывать большие объёмы данных, производить запросы, выполнять обновление данных и пр.

Процедуры никогда не возвращают значение.

```
DELIMITER //

CREATE PROCEDURE hello()
BEGIN
  SELECT 'Hello world!';
END//

DELIMITER ;
```

Вызов процедур

Для запуска процедуры необходимо вызвать ее через оператор `CALL`:

```
CALL hello();
```

Результатом выполнения процедуры будет либо информация о затронутых строках (вставка, обновление, удаление), либо выборка, произведенная в рамках процедуры.

Аргументы процедур

В отличие от функций аргументы процедуры делятся на 3 типа:

- Входящие: `IN`
- Исходящие: `OUT`
- Входяще-исходящие: `INOUT`

Входящие аргументы

```
DELIMITER //

CREATE PROCEDURE proc_in(IN var1 INT)
BEGIN
  SELECT var1 + 1 AS result;
END//

DELIMITER ;
```

Исходящие аргументы

```
DELIMITER //

CREATE PROCEDURE proc_out(OUT var1 VARCHAR(100))
BEGIN
  SET var1 = 'This is a test';
END//

DELIMITER ;
```

Входяще-исходящие аргументы

```
DELIMITER //

CREATE PROCEDURE proc_out(INOUT var1 INT)
BEGIN
  SET var1 = var1 * 2;
END//

DELIMITER ;
```

Переменные

Внутри подпрограмм можно использовать переменные. Для этого необходимо их объявить:

```
DECLARE varname TYPE [ DEFAULT defaultvalue ];
```

Примеры:

```
DECLARE a, b INT DEFAULT 5;
DECLARE str VARCHAR(50);
DECLARE today TIMESTAMP DEFAULT CURRENT_DATE;
DECLARE v1, v2, v3 TINYINT;
```

Ветвление кода

MySQL поддерживает конструкции IF, CASE, WHILE, LOOP и REPEAT для управления потоками в пределах хранимой процедуры или функции.

Мы рассмотрим, как использовать IF, CASE и WHILE, так как они наиболее часто используются.

Конструкция IF

```
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//
DELIMITER ;
```

Конструкция CASE

```
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//
DELIMITER ;
```

Цикл WHILE

```
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//
DELIMITER ;
```

Триггеры

Триггер – процедура, запускаемая при наступлении определенного события в определенной таблице.

Виды событий:

- BEFORE INSERT - до вставки,
- BEFORE UPDATE - до обновления,
- BEFORE DELETE - до удаления,
- AFTER INSERT - после вставки,
- AFTER UPDATE - после обновления,
- AFTER DELETE - после удаления.

Триггеры

В случае с событиями типа DELETE нужно помнить, что эти события не вызываются при удалении таблицы `DROP TABLE` и очистке всех данных из таблицы `TRUNCATE TABLE`.

Внутри описания триггера можно получить доступ к значениям полей той записи, с которой произошло событие.

Объекты `OLD` и `NEW` обеспечивают доступ к старым (до изменения) и новым (измененным) данным соответственно.

Создание триггеров

Синтаксис объявления триггеров:

```
CREATE TRIGGER trigger_name
BEFORE INSERT
ON `table_name`
FOR EACH ROW
тело_триггера;
```

Пример триггера

```
CREATE TABLE `account` (
  `acct_num` INT, `amount` DECIMAL(10,2)
);
CREATE TRIGGER `ins_sum`
BEFORE INSERT ON `account` FOR EACH ROW
SET @sum = @sum + NEW.amount;


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

Пример триггера

Следующий триггер следит за тем, чтобы значения в столбец amount находились в диапазоне от 0 до 100:

```
DELIMITER //
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//
DELIMITER ;
```