САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

Базы данных

Лекция 8
Процедурный SQL

Агафонов Антон Александрович
к.т.н., доцент кафедры ГИиИБ

Самара

План лекции

- Процедурные расширения SQL
  - Хранимые процедуры
  - Функции
  - Управляющие инструкции SQL
  - Триггеры
  - Курсоры

Декларативность SQL

Найти минимальное значение цены

```sql
SELECT MIN(price) FROM products;
```

- Нет переменных
- Нет цикла
- Нет сравнений
- …

Процедурные расширения

SQL/PSM (SQL/Persistent Stored Modules – постоянно хранимые модули) стандартизирует процедурное расширение для SQL:

- Управление потоком выполнения
- Обработка условий
- Обработка флагов состояний
- Курсоры
- Локальные переменные
- Поддержка постоянных подпрограмм языков баз данных

Процедурные языки в СУБД:

- **PL/SQL** в Oracle Database
- **SQL/PSM** в MySQL
- **Transact-SQL** в Microsoft SQL Server
- **PL/pgSQL** в PostgreSQL
- и т.д.

Процедурные расширения

Хранимые процедуры / функции / триггеры – объекты базы данных, представляющие собой набор SQL-инструкций, который компилируется и хранится как самостоятельный исполняемый код в системном каталоге БД.

Хранимые процедуры / функции / триггеры:

- подпрограммы, заранее подготовленные разработчиком БД
- компилируются и хранятся в системном каталоге БД
- поддерживают стандартные операции работы с БД
- поддерживают работы с локальными переменными, циклы, ветвления

Хранимые процедуры и функции могут иметь входные параметры и возвращаемые значения.

Способ вызова:

- Хранимые процедуры – через **CALL** (**EXECUTE**)
- Функции – через **SELECT**
- Триггеры – запускаются СУБД автоматически по определенному событию

Достоинства хранимых процедур / функций

Достоинства:

- Повышение производительности: хранимые процедуры хранятся в скомпилированном и оптимизированном виде. Как следствие выполнение хранимой процедуры происходит быстрее, чем запуск аналогичного кода динамического SQL.
- Снижение объема передаваемых данных: для вызова хранимой процедуры достаточно указать ее имя и значения параметров, а не передавать полный текст запроса.
- Обеспечение безопасности данных: код, выполняемый хранимой процедурой, известен, в то время как сформированный злоумышленником SQL-запрос может содержать вредоносные команды.
- Повторное использование кода.

Создание хранимой процедуры в MySQL

```
CREATE PROCEDURE [IF NOT EXISTS] sp_name ([proc_parameter [,...]])
  [characteristic ...] routine_body

proc_parameter:
  [ IN | OUT | INOUT ] param_name type

type:
  валидный тип данных MySQL

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

routine_body:
  валидный оператор SQL
```


Пример создания хранимой процедуры в MySQL

Хранимая процедура поиска самой дорогой цены по производителю

```
DELIMITER $$
CREATE PROCEDURE find_max_price (IN p_manufacturer VARCHAR(45),
                                 OUT p_max_price DECIMAL(10, 2))
BEGIN
    SELECT MAX(price) INTO p_max_price
    FROM products
    WHERE manufacturer = p_manufacturer;
END$$
DELIMITER ;

CALL find_max_price('Apple', @max_price);
SELECT @max_price;
```

Результат запроса

	@max_price
	115000.00

Создание функции в MySQL

```
CREATE FUNCTION [IF NOT EXISTS] sp_name ([func_parameter [,...]])
  RETURNS type
  [characteristic ...] routine_body

func_parameter:
  param_name type

type:
  валидный тип данных MySQL

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

routine_body:
  валидный оператор SQL
```

Пример создания функции в MySQL

Функция поиска числа заказов для продукта

```
DELIMITER $$
CREATE FUNCTION get_orders_count(p_product_id INT) RETURNS INT
    READS SQL DATA
BEGIN
    DECLARE count INT DEFAULT 0;
    SELECT COUNT(*) INTO count
    FROM orders
    WHERE product_id = p_product_id;
    RETURN count;
END$$
DELIMITER ;
```

Пример использования функции в MySQL

Вывод информации о товарах и количестве заказов

```
SELECT *, get_orders_count(product_id) AS orders_count
FROM products
ORDER BY orders_count DESC;
```

Результат запроса

	product_id	product_name	manufacturer	product_count	price	orders_count
	1	iPhone 13	Apple	1	115000.00	2
	3	Galaxy S22	Samsung	4	88000.00	1
	2	iPhone 12	Apple	7	79900.00	0
	4	Galaxy S21	Samsung	3	67000.00	0
	5	P50	Huawei	1	115000.00	0
	6	12 Pro	Xiaomi	3	115000.00	0

Управляющие конструкции. Переменные

Локальные переменные – используются в коде хранимых процедур, функций, триггеров.
Объявление переменных:

```
DECLARE var_name [, var_name, …] type [DEFAULT value]
DECLARE count INT 0;
```

Присваивание значения:

```
SET variable = expr [, variable = expr] ...
SET count = 10;
```

Пользовательские переменные – предназначены для решения локальных задач с применением интерактивного SQL, они доступны в рамках отдельной сессии.

```
SET @start = 1;
SELECT @finish := @start + 10;
```

Системные переменные – играют глобальную роль и в основном связаны с работой сервера.

```
SELECT @@version;
```

Управляющие конструкции. Оператор IF…THEN…ELSE

Условный оператор:

```
IF УСЛОВИЕ1 THEN
{… выполняется, когда УСЛОВИЕ1 принимает TRUE …}
[ ELSEIF УСЛОВИЕ2 THEN
{… выполняется, когда УСЛОВИЕ2 принимает TRUE …} ]
[ ELSE
{… выполняется, когда УСЛОВИЕ1 и УСЛОВИЕ2 принимают FALSE …} ]
END IF;
```

Управляющие конструкции. Оператор IF…THEN…ELSE

Функция вывода текстового описания уровня дохода

```
DELIMITER $$
CREATE FUNCTION get_income_level ( monthly_value INT ) RETURNS VARCHAR(20)
    DETERMINISTIC
BEGIN
    DECLARE income_level VARCHAR(20);
    IF monthly_value <= 3000 THEN
      SET income_level = 'Low Income';
    ELSEIF monthly_value > 3000 AND monthly_value <= 6000 THEN
      SET income_level = 'Avg Income';
    ELSE
      SET income_level = 'High Income';
    END IF;
    RETURN income_level;
END$$
DELIMITER ;
```

Функция вывода текстового описания уровня дохода

```
SELECT get_income_level(500);
```

	get_income_level(500)
	Low Income

```
SELECT get_income_level(3001);
```

	get_income_level(3001)
	Avg Income

```
SELECT get_income_level(10000) AS income_level;
```

	income_level
	High Income

Управляющие конструкции. Оператор CASE

Оператор-селектор **CASE**:

```
CASE выражение
WHEN значение_1 THEN
{… выполняется код, когда выражение равно значение_1 …}
[ WHEN значение_2 THEN
{… выполняется код, когда выражение равно значение_2 …} ]
[ WHEN значение_n THEN
{… выполняется код, когда выражение равно значение_n …} ]
[ ELSE
{… выполнится код, если выражение не будет равно значению …} ]
END CASE;
```

Управляющие конструкции. Оператор IF…THEN…ELSE

Функция преобразования номера дня недели в текст

```
DELIMITER //
CREATE FUNCTION fn_weekday(weekdaynum TINYINT) RETURNS VARCHAR(12)
    DETERMINISTIC
BEGIN
    CASE weekdaynum
        WHEN 1 THEN RETURN 'Понедельник';
        WHEN 2 THEN RETURN 'Вторник';
        WHEN 3 THEN RETURN 'Среда';
        WHEN 4 THEN RETURN 'Четверг';
        WHEN 5 THEN RETURN 'Пятница';
        WHEN 6 THEN RETURN 'Суббота';
        WHEN 7 THEN RETURN 'Воскресенье';
        ELSE RETURN NULL;
    END CASE;
END; //
DELIMITER ;
```

Управляющие конструкции. Оператор CASE

Оператор-селектор **CASE**:

```
CASE
WHEN условие_1 THEN
{… выполняется код, когда условие_1 будет иметь значение TRUE …}
[ WHEN условие_2 THEN
{… выполняется код, когда условие_2 будет иметь значение TRUE …} ]
[ WHEN условие_n THEN
{… выполняется код, когда условие_n будет иметь значение TRUE …} ]
[ ELSE
{… выполнится код, когда все условия будут иметь значения FALSE …} ]
END CASE;
```

Управляющие конструкции. Оператор CASE

Функция вывода текстового описания уровня дохода

```
DELIMITER //
CREATE FUNCTION get_income_level_case ( monthly_value INT ) RETURNS VARCHAR(20)
    DETERMINISTIC
BEGIN
    DECLARE income_level VARCHAR(20);
    CASE
        WHEN monthly_value <= 3000 THEN
            SET income_level = 'Low Income';
        WHEN monthly_value > 3000 AND monthly_value <= 6000 THEN
            SET income_level = 'Avg Income';
        ELSE
            SET income_level = 'High Income';
    END CASE;
    RETURN income_level;
END; //
DELIMITER ;
```

Управляющие конструкции. Цикл WHILE

Цикл с предусловием **WHILE**:

```
[метка:] WHILE условие_цикла DO
операторы_цикла
END WHILE [метка]
```

Пример использования цикла с предусловием

```
DELIMITER //
CREATE FUNCTION calc_cost (value INT) RETURNS INT
    NO SQL
BEGIN
   DECLARE cost INT;
   SET cost = 0;
   WHILE cost <= 3000 DO
     SET cost = cost + value;
   END WHILE;
   RETURN cost;
END; //
DELIMITER ;
```

Управляющие конструкции. Цикл REPEAT

Цикл с постусловием **REPEAT**:

```
[метка:] REPEAT
операторы_цикла
UNTIL условие_цикла
END REPEAT [метка]
```

Построение таблицы синусов в цикле REPEAT

```
DELIMITER //
CREATE PROCEDURE pr_sin_table()
BEGIN
    DECLARE d int DEFAULT 0; # переменная для хранения градусов
    DECLARE r float DEFAULT 0; # переменная для хранения радиан
# ---- создаем временную таблицу с 3 столбцами ----------
    DROP TEMPORARY TABLE IF EXISTS sin_table;
    CREATE TEMPORARY TABLE sin_table (deg int, rad float, sin_value float);
# ---- цикл заполнения таблицы данными -------
    REPEAT
        INSERT INTO sin_table VALUES (d, r, sin(r));
        SET r = r + pi() / 180;
        SET d = d + 1;
    UNTIL r>=pi()
    END REPEAT;
# ----------------------------------------------
    SELECT * FROM sin_table; # результат – выборка из таблицы синусов
    DROP TEMPORARY TABLE sin_table;
END; //
DELIMITER ;
```

Управляющие конструкции. Цикл LOOP

Петлевой цикл **LOOP**:

```
[метка:] LOOP
операторы_цикла
END LOOP [метка]
```

Пример использования петлевого цикла

```
CREATE FUNCTION calc_cost (value INT) RETURNS INT NO SQL
BEGIN
   DECLARE cost INT;
   SET cost = 0;
   label1: LOOP
     SET cost = cost + value;
     IF cost < 3000 THEN
       ITERATE label1;
     END IF;
     LEAVE label1;
   END LOOP label1;
   RETURN cost;
END;
```

Модификация процедур и функций

Изменение процедур и функций

```
ALTER PROCEDURE имя_процедуры …;
ALTER FUNCTION имя_функции …;
```

Удаление процедур и функций

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] имя_процедуры(функции);
```

Создание функции в PostgreSQL

```
CREATE [ OR REPLACE ] FUNCTION
    имя ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента [ { DEFAULT | = }
выражение_по_умолчанию ] [, ...] ] )
    [ RETURNS тип_результата | RETURNS TABLE ( имя_столбца тип_столбца [, ...] ) ]
  { LANGUAGE имя_языка
    | TRANSFORM { FOR TYPE имя_типа } [, ... ]
    | WINDOW
    | { IMMUTABLE | STABLE | VOLATILE }
    | [ NOT ] LEAKPROOF
    | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
    | { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER }
    | PARALLEL { UNSAFE | RESTRICTED | SAFE }
    | COST стоимость_выполнения
    | ROWS строк_в_результате
    | SUPPORT вспомогательная_функция
    | SET параметр_конфигурации { TO значение | = значение | FROM CURRENT }
    | AS 'определение'
    | AS 'объектный_файл', 'объектный_символ'
    | тело_sql
  } ...
```


Пример создания функции в PostgreSQL

Пример использования цикла с предусловием

```
CREATE FUNCTION calc_cost(val INT) RETURNS INT
    LANGUAGE plpgsql
AS $$
DECLARE
    res INTEGER;
BEGIN
    res = 0;
    WHILE res <= 3000 LOOP
        res = res + val;
    END LOOP;
    RETURN res;
END $$;
```

Пример создания функции в PostgreSQL

Пример функции проверки товара в наличии из БД pagila

```
CREATE OR REPLACE FUNCTION inventory_in_stock(p_inventory_id integer) RETURNS boolean
    LANGUAGE 'plpgsql'
AS $$
DECLARE
    v_rentals INTEGER;
    v_out     INTEGER;
BEGIN
    SELECT count(*) INTO v_rentals
    FROM rental
    WHERE inventory_id = p_inventory_id;

    IF v_rentals = 0 THEN
      RETURN TRUE;
    END IF;

    SELECT COUNT(rental_id) INTO v_out
    FROM inventory LEFT JOIN rental USING(inventory_id)
    WHERE inventory.inventory_id = p_inventory_id AND rental.return_date IS NULL;

    IF v_out > 0 THEN
      RETURN FALSE;
    ELSE
      RETURN TRUE;
    END IF;
END $$;
```

Триггеры

Триггер – разновидность хранимой процедуры, вызов которой осуществляется автоматически.

- триггер гарантированно срабатывает только при наступлении определенного события, обычно связанного с модификацией значений в строке таблицы;
- триггер проверяет условия выполнения операции, вызвавшей его срабатывание;
- если условия верны, то триггер выполняет определенные действия (например, разрешает добавить в таблицу новую строку), а если условия ложны – триггер отвергает операцию.

Создание триггера в MySQL

```
CREATE 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
```

Стандарт SQL рекомендует создавать триггеры, выполняющиеся при возникновении следующих событий:

- события **BEFORE INSERT** и **AFTER INSERT** возникают соответственно перед тем, как новая строка попадет в таблицу, и после того, как эта строка будет сохранена;
- события **BEFORE UPDATE** и **AFTER UPDATE** генерируются перед началом и после завершения редактирования данных;
- событие **BEFORE DELETE** предшествует, а **AFTER DELETE** завершает операцию удаления данных из таблицы.

В теле триггера ключевые слова **OLD** и **NEW** позволяют получить доступ к строкам таблицы, действие над которыми активировали триггер:

- в **INSERT**: **NEW**.col_name
- в **UPDATE**: **NEW**.col_name и **OLD**.col_name
- в **DELETE**: **OLD**.col_name

Пример триггера в MySQL

Триггер заполнения даты

```
DELIMITER $$
CREATE TRIGGER orders_BEFORE_INSERT
BEFORE INSERT
ON orders FOR EACH ROW
BEGIN
    SET NEW.created_at = NOW();
END$$
DELIMITER ;
```

Пример триггера в MySQL

Ведение протокола изменений в таблице

```
DELIMITER $$
CREATE TRIGGER products_AFTER_UPDATE
AFTER UPDATE
ON products FOR EACH ROW
BEGIN
    INSERT INTO products_log
    (user_name, record_time, old_product_name, new_product_name)
    VALUES (USER(), NOW(), OLD.product_name, NEW.product_name);
END$$
DELIMITER ;
```

Пример триггера в MySQL

Триггер проверки валидности данных

```
DELIMITER $$
CREATE TRIGGER update_check BEFORE UPDATE ON orders FOR EACH ROW
BEGIN
    IF NEW.product_count < 0 THEN
        SET NEW.product_count = 0;
    ELSEIF NEW.product_count > 100 THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Превышено
количество товаров в заказе';
    END IF;
END;$$
DELIMITER ;
```

Создание триггера в PostgreSQL

```
CREATE [ OR REPLACE ] [ CONSTRAINT ] TRIGGER имя { BEFORE | AFTER | INSTEAD OF } {
событие [ OR ... ] }
    ON имя_таблицы
    [ FROM ссылающаяся_таблица ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
    [ REFERENCING { { OLD | NEW } TABLE [ AS ] имя_переходного_отношения } [ ... ] ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( условие ) ]
    EXECUTE { FUNCTION | PROCEDURE } имя_функции ( аргументы )

событие:
    INSERT
    UPDATE [ OF имя_столбца [, ... ] ]
    DELETE
    TRUNCATE
```

Пример триггера в PostgreSQL

Ведение протокола изменений в таблице: функция-обработчик

```
CREATE OR REPLACE FUNCTION product_logger_function()
RETURNS trigger AS $body$
BEGIN
   if (TG_OP = 'INSERT') then
       INSERT INTO products_log (user_name, record_time, old_product_name, new_product_name, operation)
       VALUES(CURRENT_USER, CURRENT_TIMESTAMP, null, NEW.product_name, 'INSERT');
       RETURN NEW;
   elsif (TG_OP = 'UPDATE') then
       INSERT INTO products_log (user_name, record_time, old_product_name, new_product_name, operation)
       VALUES(CURRENT_USER, CURRENT_TIMESTAMP, OLD.product_name, NEW.product_name, 'UPDATE');
       RETURN NEW;
   elsif (TG_OP = 'DELETE') then
       INSERT INTO products_log (user_name, record_time, old_product_name, new_product_name, operation)
       VALUES(CURRENT_USER, CURRENT_TIMESTAMP, OLD.product_name, null, 'DELETE');
       RETURN OLD;
   end if;
END;
$body$
LANGUAGE plpgsql
```

Пример триггера в PostgreSQL

Ведение протокола изменений в таблице: триггер

```
CREATE TRIGGER product_logger_trigger
AFTER INSERT OR UPDATE OR DELETE
ON products
FOR EACH ROW
EXECUTE FUNCTION product_logger_function()
```

Курсоры

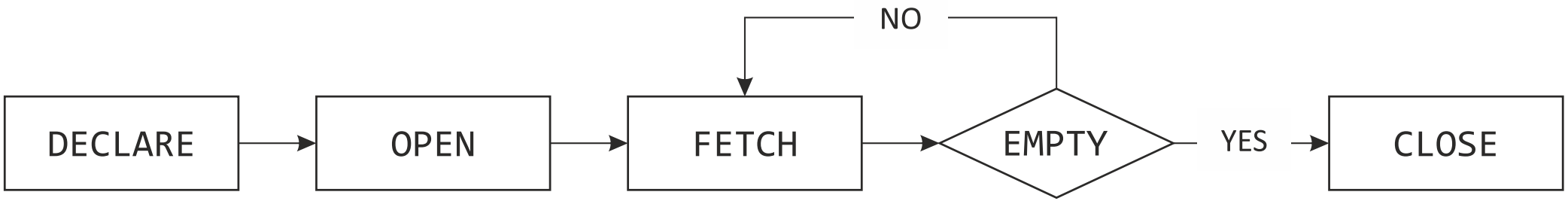
Курсор представляет собой указатель на отдельный кортеж в отношении. С помощью курсора можно последовательно перебирать строки полученного отношения, осуществляя с ними дополнительные операции.

Применение курсора:

- сложная логика запроса (наличие формул, необходимость обращения из запроса к процедурам или функциям, необходимость использования условных операторов и исключений);
- операции по денормализации данных, когда из нескольких отношений следует собрать одно или возвратить результаты в формате текстовой строки, документа XML и т. п.
- создание перекрестного запроса, допустим, выполняющего статистические расчеты по двум и более таблицам, которые позднее группируются в виде таблицы;
- движение по иерархическому дереву.

Для работы с курсорами последовательно применяются 4 команды SQL:

1) объявить курсор (DECLARE CURSOR);
2) открыть курсор (OPEN);
3) считать данные из курсора (FETCH);
4) закрыть курсор (CLOSE).



Курсоры в MySQL

Объявление курсора:

```
DECLARE cursor_name CURSOR FOR SELECT_statement;
```

Открытие курсора:

```
OPEN cursor_name;
```

Получение следующей строки:

```
FETCH cursor_name INTO variables_list;
```

Обработчик события **NOT FOUND** (все строки просмотрены):

```
DECLARE CONTINUE HANDLER FOR NOT FOUND statement;
```

Закрытие курсора:

```
CLOSE cursor_name;
```

Пример курсора в MySQL

Процедура конкатенации товаров в одну строку с использованием курсора

```
DELIMITER $$
CREATE PROCEDURE create_products_list (OUT products_list varchar(4000) )
BEGIN
    DECLARE finished INTEGER DEFAULT 0;
    DECLARE v_product varchar(100) DEFAULT "";

    DECLARE products_cursor CURSOR FOR
    SELECT product_name FROM products;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

    SET products_list = '';
    OPEN products_cursor;
    get_products_label: LOOP
         FETCH products_cursor INTO v_product;
         IF finished = 1 THEN
             LEAVE get_products_label;
         END IF;
         SET products_list = CONCAT(v_product, ";", products_list);
    END LOOP get_products_label;
    CLOSE products_cursor;
END$$
DELIMITER ;
```

САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

БЛАГОДАРЮ ЗА ВНИМАНИЕ

Агафонов А.А.
к.т.н., доцент кафедры ГИиИБ