Функции, процедуры, триггеры, курсоры

Функции

По поведению:

- Функция является детерминированной, если при одном и том же заданном входном значении она всегда возвращает один и тот же результат.
- Функция является недетерминированной, если она может возвращать различные значения при одном и том же заданном входном значении.

Функции

По типу возвращаемого значения:

- Скалярная функция;
- Подставляемая табличная функция;
- Многооператорная табличная функция.

Скалярная функция

Синтаксис:

```
CREATE [OR REPLACE] FUNCTION [ имя-схемы. ] имя-функции ([список-объявлений-параметров ] )
RETURNS скалярный-тип-данных
AS $$
      тело-функции
$$ [ ; ]
```

```
CREATE FUNCTION add_em(integer, integer)
RETURNS integer AS
$$
   SELECT $1 + $2;
$$
LANGUAGE SQL;
```

Пример:

```
CREATE FUNCTION one()
RETURNS integer AS
begin
   SELECT 1 AS result;
end
LANGUAGE SQL;

CREATE FUNCTION getSalary(int)
RETURNS int AS
$$
   SELECT max(Salary) AS result FROM foo
   WHERE fooid = $1;
$$
LANGUAGE SQL;
```

Подставляемая табличная функция

Синтаксис:

```
CREATE FUNCTION [ имя-схемы. ] имя-функции ( [ список-объявлений-параметров ] )
RETURNS SETOF <скаларный тип>
AS $$
[ выражение-выборки ]
$$ [ ; ]
```

```
CREATE TABLE foo (
    fooid int,
    foosubid int,
    fooname text
);
```

Пример:

```
CREATE FUNCTION listchildren(text)
RETURNS SETOF text AS
$$
    SELECT name FROM nodes
   WHERE parent = $1
$$
LANGUAGE SQL;
```

```
CREATE FUNCTION getfoo(int)
RETURNS foo AS
$$
    RETURN QUERY
    SELECT * FROM foo
    WHERE fooid = $1;
$$ LANGUAGE SQL;
```

Многооператорная функция

Синтаксис:

```
CREATE FUNCTION [ имя-схемы. ] имя-функции ( [ список-объявлений-параметров ] )
RETURNS TABLE (определение-таблицы)
AS $$
[ SQL- запрос ]
$$ [ ; ]
```

Пример:

```
CREATE FUNCTION sum_n_product_with_tab (x int)
RETURNS TABLE(sum int, product int) AS
$$
    SELECT x + tab.y, $1 * tab.y
    FROM tab;
$$ LANGUAGE SQL;
```

Триггеры

Триггер - это хранимая процедура особого типа, которая выполняет одну или несколько инструкций в ответ на событие.

По типу события триггеры деляться на два класса:

- DDL-триггеры
- DML-триггеры

DDL-триггеры

Триггер DDL может активироваться, если выполняется такая инструкция, как ALTER SERVER CONFIGURATION, или если происходит удаление таблицы с использованием команды DROP TABLE.

```
CREATE EVENT TRIGGER имя
ON событие
[ WHEN переменная_фильтра
  IN (значение_фильтра
  [, ... ])
[ AND ... ] ]
EXECUTE PROCEDURE имя_функции()
```

Пример триггера:

```
CREATE OR REPLACE FUNCTION
abort_any_command()
RETURNS event_trigger
LANGUAGE plpgsql AS
$$
  BEGIN RAISE EXCEPTION 'команда %
        отключена', tg_tag;
END;
$$;

CREATE EVENT TRIGGER abort_ddl
ON ddl_command_start

EXECUTE PROCEDURE abort_any_command();
```

DML-триггеры

Для поддержания согласованности и точности данных используются декларативные и процедурные методы.

Триггеры применяются в следующих случаях:

- если использование методов декларативной целостности данных не отвечает функциональным потребностям приложения;
- если необходимо каскадное изменение через связанные таблицы в базе данных;
- если база данных денормализована и требуется способ автоматизированного обновления избыточных данных в нескольких таблицах;
- если необходимо сверить значение в одной таблице с неидентичным значением в другой таблице;
- если требуется вывод пользовательских сообщений и сложная обработка ошибок.

Классы DML-триггеров

Существуют два класса триггеров:

- INSTEAD OF. Триггеры этого класса выполняются в обход действий, вызывавших их срабатывание, заменяя эти действия. Например, обновление таблицы, в которой есть триггер INSTEAD OF, вызовет срабатывание этого триггера. В результате вместо оператора обновления выполняется код триггера.
- AFTER/BEFORE. Триггеры этого класса исполняются после или до действия, вызвавшего срабатывание триггера. Они считаются классом триггеров по умолчанию.

DML-триггеры

Синтаксис:

```
CREATE [ OR REPLACE ] CONSTRAINT ]
TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
ON table_name [ FROM referenced_table_name ]
[ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
[ REFERENCING { { OLD | NEW } TABLE [ AS ] transition_relation_name } [ ... ] ]
  [
FOR [ EACH ] { ROW | STATEMENT } ]      [ WHEN ( condition ) ]
EXECUTE { FUNCTION | PROCEDURE } function_name ( arguments )
```

DML-триггеры

```
CREATE TRIGGER check_upd
BEFORE UPDATE ON accounts
FOR EACH ROW
EXECUTE FUNCTION
check_account_update();
```

```
CREATE OR REPLACE TRIGGER
check_upd
BEFORE UPDATE OF balance ON accounts

FOR EACH ROW
EXECUTE FUNCTION
check_account_update();
```

```
CREATE TRIGGER log_update
AFTER UPDATE ON accounts
FOR EACH ROW
WHEN (OLD.* IS DISTINCT FROM NEW.*)

EXECUTE FUNCTION
log_account_update();
```

Пример:

```
CREATE OR REPLACE
FUNCTION employee_insert_trigger_fnc()
RETURNS trigger AS
$$
  INSERT INTO "Employee_Audit" (
      "EmployeeId", "LastName",
  "FirstName","UserName" ,
          "EmpAdditionTime")
VALUES(NEW."EmployeeId",NEW."LastName",
  NEW."FirstName", current_user,
current_date);
RETURN NEW;
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER employee_insert_trigger
AFTER INSERT ON "Employee"
FOR EACH ROW
EXECUTE PROCEDURE
employee_insert_trigger_fnc();
```

Курсоры

Операции в реляционной базе данных выполняются над множеством строк. Набор строк, возвращаемый инструкцией SELECT, содержит все строки, которые удовлетворяют условиям, указанным в предложении WHERE.

Курсоры можно классифицировать:

- По области видимости;
- По типу;
- По способу перемещения по курсору;
- По способу распараллеливания курсора

По области видимости

По области видимости имени курсора различают:

- Локальные курсоры (LOCAL). Область курсора локальна по отношению к пакету, хранимой процедуре или триггеру, в которых этот курсор был создан. Курсор неявно освобождается после завершения выполнения пакета, хранимой процедуры или триггера, за исключением случая, когда курсор был передан параметру OUTPUT.
- Глобальные курсоры (GLOBAL). Область курсора является глобальной по отношению к соединению.

По типу

По типу курсора различают:

- Статические курсоры (STATIC). Создается временная копия данных для использования курсором. Все запросы к курсору обращаются к указанной временной таблице в базе данных tempdb, поэтому изменения базовых таблиц не влияют на данные, возвращаемые выборками для данного курсора, а сам курсор не позволяет производить изменения.
- Динамические курсоры (DYNAMIC). Отображают все изменения данных, сделанные в строках результирующего набора при просмотре этого курсора. Значения данных, порядок, а также членство строк в каждой выборке могут меняться. Параметр выборки ABSOLUTE динамическими курсорами не поддерживается.
- Курсоры, управляемые набором ключей (KEYSET). Членство или порядок строк в курсоре не изменяются после его открытия. Набор ключей, однозначно определяющих строки, встроен в таблицу в базе данных tempdb с именем keyset.
- Быстрые последовательные курсоры (FAST_FORWARD). Параметр FAST_FORWARD указывает курсор FORWARD_ONLY, READ_ONLY, для которого включена оптимизация производительности. Параметр FAST_FORWARD не может указываться вместе с параметрами SCROLL или FOR_UPDATE.

По способу перемещения по курсору

- Последовательные курсоры (FORWARD_ONLY). Курсор может просматриваться только от первой строки к последней. Поддерживается только параметр выборки FETCH NEXT. Если параметр FORWARD_ONLY указан без ключевых слов STATIC, KEYSET или DYNAMIC, то курсор работает как DYNAMIC.
- Курсоры прокрутки (SCROLL). Перемещение осуществляется по группе записей как вперед, так и назад. В этом случае доступны все параметры выборки (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE). Параметр SCROLL не может указываться вместе с параметром для FAST_FORWARD.

По способу распараллеливания курсора

По способу распараллеливания курсоров различают:

- READ_ONLY. Содержимое курсора можно только считывать.
- SCROLL_LOCKS. При редактировании данной записи вами никто другой вносить в нее изменения не может. Такую блокировку прокрутки иногда еще называют «пессимистической» блокировкой.
- OPTIMISTIC. Означает отсутствие каких бы то ни было блокировок. «Оптимистическая» блокировка предполагает, что даже во время выполнения вами редактирования данных, другие пользователи смогут к ним обращаться.

Курсоры

Синтаксис:

```
DECLARE имя-курсора CURSOR
[ область-видимости-имени-курсора ]
[ возможность-перемещения-по-курсору ]
[ типы-курсоров ]
[ опции-распараллеливания-курсоров ]
[ выявление-ситуаций-с-преобразованием-типа-курсора ]
FOR инструкция_select
[ опция-FOR-UPDATE ]
```

Пример:

```
DECLARE @MyVariable CURSOR;

DECLARE @MyCursor CURSOR
FOR SELECT LastName FROM
AdventureWorks.Person.Contact;

SET @MyVariable = MyCursor;
```

Курсоры

Основой всех операций прокрутки курсора является ключевое слово FETCH. В качестве аргументов оператора FETCH могут выступать:

- NEXT – возвращает строку результата сразу же за текущей строкой и перемещает указатель текущей строки на возвращенную строку. Если инструкция FETCH NEXT выполняет первую выборку в отношении курсора, она возвращает первую строку в результирующем наборе. NEXT является параметром по умолчанию выборки из курсора.
- PRIOR – возвращает строку результата, находящуюся непосредственно перед текущей строкой и перемещает указатель текущей строки на возвращенную строку. Если инструкция FETCH PRIOR выполняет первую выборку из курсора, не возвращается никакая строка и положение курсора остается перед первой строкой.
- FIRST – возвращает первую строку в курсоре и делает ее текущей.
- LAST – возвращает последнюю строку в курсоре, и делает ее текущей.

```
-- Объявляем курсор
DECLARE CursorTest CURSOR GLOBAL SCROLL STATIC
FOR SELECT OrderID, CustomerID FROM CursorTable;

-- Объявляем переменные для хранения
DECLARE @OrderID int;
DECLARE @CustomerID varchar(5);

-- Откроем курсор и запросим первую запись
OPEN CursorTest FETCH NEXT FROM CursorTest INTO
@OrderID, @CustomerID

 -- Обработаем в цикле все записи курсора
WHILE @@FETCH_STATUS=0
BEGIN
      PRINT CONVERT(varchar(5),@OrderID) + ' ' +
@CustomerID
      FETCH NEXT FROM CursorTest INTO @OrderID,
@CustomerID
END

CLOSE CursorTest
```