# Stored Procedures and Triggers

# Content

1. Trigger Function Format
2. Creating a Trigger Function
3. Testing the Trigger Function

# Trigger

- A *trigger* defines an operation that is performed when a specific event occurs on a table:
  - inserts a new record / updates an existing record, or deletes a record.
- The function executed as a result of a trigger is called a *trigger function*.

# 1. Trigger Function Format

- Looks similar to the stored procedure function (same CREATE OR REPLACE FUNCTION command)
- 2 two things:
  - Trigger functions do not use input arguments in the function, but rather are passed arguments from a trigger event
  - Trigger functions have access to special variables from the database engine

# CREATE TRIGGER command

CREATE TRIGGER *name* { BEFORE | AFTER | INSTEAD OF} {*event* [OR ... ] } ON *table*
[ FOR [ EACH ] { ROW | STATEMENT }]
[ WHEN ( *condition* ) ]
EXECUTE PROCEDURE *function* (*arguments*)

WHEN (condition) that determines whether the trigger function will actually be executed

**BEFORE, AFTER** can be used for tables and views

**INSTEAD OF** can be only used for views at row-level

file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/sql-createtrigger.html

# CREATE TRIGGER command - Explain

- Can occur either before or after the *event* occurs (INSERT, UPDATE,DELETE, TRUNCATE on the *table)*
  - multiple events can be specified using OR
  - UPDATE events, it is possible to specify a list of columns using this syntax: UPDATE OF *column_name1* [, *column_name2* ... ]
  - INSTEAD OF UPDATE events do not support lists of columns.
- Fire triggers:
  - ROW: for each row that is affected by the event
  - STATEMENT: for each statement that triggers the event, no matter how many rows are returned (even if no rows are returned)

# CREATE TRIGGER command - Explain

- CREATE TRIGGER check_update

  BEFORE UPDATE ON accounts

  FOR EACH ROW

  WHEN (OLD.balance IS DISTINCT FROM NEW.balance)

  EXECUTE PROCEDURE check_account_update();

file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/sql-createtrigger.html

# CREATE TRIGGER command

- *function:* execute when the trigger is fired is declared
  - the arguments in the CREATE TRIGGER command are passed using the TG_ARGV special variable

- When a trigger function is called, the database engine passes a group of special variables to the trigger function →define the environment
  - how the function was called
  - what data is present when the trigger was fired
  - when the trigger was fired
  - …

| Special Variable | Description |
|---|---|
| NEW | The record column data values present in the `INSERT` or `UPDATE` command |
| OLD | The record column data values present in the table before an `UPDATE` or `DELETE` command is executed |
| TG_NAME | The name of the called trigger |
| TG_WHEN | When the trigger was fired, either `BEFORE` or `AFTER` the SQL command |
| TG_LEVEL | The trigger definition, either `ROW` or `STATEMENT` |
| TG_OP | The event that fired the trigger, either `INSERT`, `UPDATE`, or `DELETE` |
| TG_RELID | The OID of the table that fired the trigger |
| TG_RELNAME | The name of the table that fired the trigger |
| TG_NARGS | The number of arguments in the `CREATE TRIGGER` command |
| TG_ARGV[] | An array containing the arguments used in the `CREATE TRIGGER` command |

file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/plpgsql-trigger.html

# 2. Creating a Trigger Function

- Can use the pgAdmin III program to create trigger functions:
  - right-click the Trigger Functions object →select New Trigger Function
  - Set the Language textbox to plpgsql
  - A trigger function updates table records → VOLATILE function
  - Parameter tab, NOT allowed to define arguments.
  - Definition textbox →enter the function code

# Example

- A trigger function that automatically updates the ***inventory*** every time an order is entered into the ***orderlines*** table

```
CREATE OR REPLACE FUNCTION tg_updateinventory()  RETURNS trigger AS
$$DECLARE
    oldquan int4;
    oldsales int4;
BEGIN
    SELECT into oldquan, oldsales "quan_in_stock", "sales"
    FROM inventory
    WHERE prod_id = NEW.prod_id;
    IF TG_OP = 'INSERT' THEN
            update inventory
                    set quan_in_stock = oldquan - NEW.quantity,
                                sales = oldsales + NEW.quantity
            where prod_id = NEW.prod_id;
    END IF;
    RETURN NEW;
END;
$$  LANGUAGE plpgsql VOLATILE;
```

# Example

- Create a trigger:

```
CREATE TRIGGER tg_after_insert  AFTER INSERT

ON orderlines  FOR EACH ROW

EXECUTE PROCEDURE tg_updateinventory();
```

Others exemples :
file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/plpgsql-trigger.html
http://developer.postgresql.org/pgdocs/postgres/plpgsql-trigger.html

file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/plpgsql.html

# Example

- Lưu ý: RETURN trong trigger function
  - NULL
  - 1 bản ghi có cấu trúc giống bản ghi của bảng trên đó trigger được định nghĩa
- Đối với trigger « AFTER »:
  - RETURN NULL; -- or RETURN NEW; RETURN OLD;
- Đối với trigger « BEFORE »
  - RETURN NULL; : các trigger con (nếu có) và các lệnh insert,update, delete cũng không được thực thi
  - Trigger before-delete : giá trị trả về  không có nghĩa gì chỉ cần không null để lệnh DELETE được thực thi
  - Before update or insert: RETURN NEW;

# 3. Testing the Trigger Function

- Select * from inventory where prod_id = 1;

| prod_id | quan_in_stock | sales |
|---|---|---|
| 1 | 138 | 9 |

- Select * from orderlines where orderid = 1405;
- Insert into orderlines (orderlineid, orderid, prod_id, quantity, orderdate) values (8,1405, 1, 5, '2011-11-02');
- Select * from inventory where prod_id = 1;

| prod_id | quan_in_stock | sales |
|---|---|---|
| 1 | 133 | 14 |

# 4. Bài tập

- Categories(category, categoryname)
- cust_hist(customerid, orderid, prod_id)
- Customers(customerid, firstname, lastname, address1, address2, city, state, zip, country, region, email, phone, creditcardtype, creditcard, creditcardexpiration, username, password, age, income, gender)
- Inventory(prod_id, quan_in_stock,sales)
- Orderlines(orderlineid, orderid, prod_id, quantity, orderdate)
- Orders(orderid, orderdate, customerid, netamount, tax, totalamount)
- Products(prod_id, category, title, actor, price, special, common_prod_id)

# 4. Bài tập

- Xác định và viết các trigger tương ứng để đảm bảo:
  - DL ở cust_hist phải tương ứng với DL ở Orderlines và Orders
    - Trigger **SAU** khi **insert/update/delete** bảng **orders** và bảng **orderlines**
  - Ở bảng Order: totalamount = netamount + tax
    - Trigger **TRƯỚC** khi **insert/update** 1 dòng trong bảng **orders**
  - DL ở bảng Inventory phải được cập nhật hợp lý mỗi khi có sự thay đổi DL ở bảng orderlines:
    - Trigger **SAU** khi **insert/update/delete** 1 dòng trong bảng **orderlines**

- Giả sử có 1 bảng cust_log để lưu thông tin thác tác dữ liệu trên bảng Customers gồm 3 trường (operation: thao thác thực hiện, time: ngày giờ thực hiện, username: người thực hiện )
- Hãy tạo bảng trên
- Xác định và viết các trigger tương ứng để đảm bảo:
  - DL ở cust_log được lưu mỗi khi có thao tác thêm xóa, sửa trên bảng customers.

# Updatable view

- Tạo 1 view cho CSDL dellstore, chứa các thông tin về : (customerid, firstname, lastname, email, phone, orderid, orderlineid, prod_id, title, quantity)
- Hãy tạo trigger để cho view này:
  - Có thể update được số lượng, mã sản phẩm, số email, phone của khách hàng
  - Có thể xóa được: khi 1 dòng ở đây bị xóa thì tương ứng sẽ xóa ở bảng orderlines và có thể ở bảng order(nếu tạo ra order rỗng)
  - Không thể insert