

CSE – 302

Database Management Systems

Sessional

TRIGGERING & SEQUENCE

28-04-2019

TRIGGERS

- A database trigger is a stored PL/SQL program unit associated with a specific database
- ORACLE executes (fires) a database trigger automatically when a given SQL operation (like INSERT, UPDATE or DELETE) affects the table
- Unlike a procedure, or a function, which must be invoked explicitly, database triggers are invoked **implicitly**

Event-Condition-Action Model

- A **trigger** is a statement that the system executes automatically as a side effect of a **modification to the database**.
- Two requirements:
 1. Specify when a trigger is to be executed. This is broken up into an *event* that causes the trigger to be checked and a *condition* that must be satisfied for trigger execution to proceed.
 2. Specify the *actions* to be taken when the trigger executes.
- Once we enter a trigger into the database, the database system takes on the responsibility of executing it whenever the specified event occurs and the corresponding condition is satisfied.

TRIGGERS

Database triggers can be used to perform any of the followings

- Audit data modification
- Log events transparently
- Derive column values automatically
- Implement complex security authorizations
- Maintain replicate tables

TRIGGERS

- We can associate up to 12 database triggers with a given table
- When an event occurs, a database trigger is fired, and an predefined PL/SQL block will perform the necessary action.
- Use triggers to guarantee that when a specific operation is performed, related actions are performed.

Some Points

- Do not define triggers that duplicate features already built into Oracle Database. For example, do not define triggers to reject bad data if you can do the same checking through declarative integrity constraints.
- Limit the size of triggers. If the logic for your trigger requires much more than 60 lines of PL/SQL code, it is better to include most of the code in a stored procedure and call the procedure from the trigger.
- Use triggers only for centralized, global operations that should be fired for the triggering statement, regardless of which user or database application issues the statement.

TRIGGERS

```
CREATE [OR REPLACE] TRIGGER trigger_name  
{BEFORE|AFTER| INSTEAD OF} triggering_event  
ON table_name
```

```
[FOR EACH ROW]  
[WHEN condition]
```

```
DECLARE  
    //Declaration statements  
BEGIN  
    //Executable statements  
EXCEPTION  
    //Exception-handling statements  
END;  
/
```

TRIGGERS

- The ***trigger_name*** references the name of the trigger.
- **BEFORE** or **AFTER** specify when the trigger is fired (before or after the triggering event).
- **INSTEAD OF** is used to create a trigger on a view. before and after cannot be used to create a trigger on a view.
- The ***triggering_event*** references a DML statement issued against the table (e.g., INSERT, DELETE, UPDATE).
- The ***table_name*** is the name of the table associated with the trigger.

TRIGGERS

- The clause, **FOR EACH ROW**, specifies a trigger is a row trigger and fires once for each modified row.
- A **WHEN** clause specifies the condition for a trigger to be fired.
- Bear in mind that if you drop a table, all the associated triggers for the table are dropped as well.

TRIGGERS

Triggers may be called BEFORE or AFTER the following events:

- **INSERT, UPDATE and DELETE.**
- The before/after options can be used to specify when the trigger body should be fired with respect to the triggering statement.
- If the user indicates a **BEFORE** option, then Oracle fires the trigger before executing the triggering statement.
- On the other hand, if an **AFTER** is used, Oracle fires the trigger after executing the triggering statement.

TRIGGERS

- A trigger may be a ROW or STATEMENT type. If the statement FOR EACH ROW is present in the CREATE TRIGGER clause of a trigger, the trigger is a row trigger. A row trigger is fired for each row affected by an triggering statement.
- A statement trigger, however, is fired only once for the triggering statement, regardless of the number of rows affected by the triggering statement

Example 01: Create a Trigger

```
CREATE OR REPLACE TRIGGER MyTrigger  
BEFORE DELETE OR INSERT OR UPDATE  
ON CUSTOMER  
BEGIN  
    RAISE_APPLICATION_ERROR (-20343, 'UNSAFE');  
END;
```

Example 02

A trigger that limits the DML actions to the customer table to weekends and from 8.30am to 6.30pm otherwise.

If a user tries to insert/update/delete a row in the CUSTOMER table, a warning message will be prompted.

Example 02 (Oracle 11g)

```
CREATE OR REPLACE TRIGGER MyTrigger
BEFORE DELETE OR INSERT OR UPDATE
ON CUSTOMER
BEGIN
  IF (TO_CHAR(SYSDATE, 'day') IN ('saturday', 'friday'))
    OR (TO_CHAR(SYSDATE, 'hh:mi') NOT BETWEEN '10:30' AND '18:30')
  THEN
    RAISE_APPLICATION_ERROR (-20343, 'UNSAFE');
  END IF;
END;
```

Example 02 (Oracle 10g)

```
CREATE OR REPLACE TRIGGER MyTrigger
BEFORE DELETE OR INSERT OR UPDATE
ON CUSTOMER
BEGIN
  IF (TO_CHAR(SYSDATE, 'day') IN ('saturday', 'friday'))
    OR (TO_CHAR(SYSDATE, 'HH24:MI') NOT BETWEEN '10:30' AND '18:
      30')
  THEN
    RAISE_APPLICATION_ERROR (-20343, 'UNSAFE ');
  END IF;
END;
```

Example 02:

INSERT values in Customer table to test the trigger

```
INSERT INTO CUSTOMER (Cust_id, Cust_name, Cust_dob)
VALUES ('C_115', 'Sazia Binte', to_date('08/16/1994', 'MM/
DD/YYYY'));
```


Example 03

1) Create the **Product** table and **Product_Restock** table

```
CREATE TABLE Product_Restock
```

```
(  
    product_id number(5),  
    product_name varchar2(32),  
    supplier_name varchar2(32),  
    unit_price number(7,2),  
    Amount number(7,0)  
);
```

```
CREATE TABLE Product
```

```
(  
    product_id number(5),  
    product_name varchar2(32),  
    supplier_name varchar2(32),  
    unit_price number(7,2),  
    Amount number(7,0)
```

```
);
```

Example 03

2) Create the **Product_Restock_Trigger** and execute it.

```
CREATE or REPLACE TRIGGER Product_Restock_Trigger
BEFORE UPDATE OF Amount
ON Product
FOR EACH ROW
WHEN (New.AMOUNT < 20)
BEGIN
    dbms_output.put_line('trigger called');
    INSERT INTO Product_Restock VALUES (:old.product_id, :old.
        product_name, :old.supplier_name, :old.unit_price, :old.amount);
END;
/
```

| product_id | product_name | supplier_name | unit_price | Amount |
|------------|--------------|---------------|------------|--------|
| 100 | Soap | Supplier 1 | 25 | 20 |
| 105 | Paracetamol | Supplier 2 | 10 | 25 |

Example 03

3) Lets update the price of a product.

```
UPDATE Product SET Amount = 19  
WHERE Product_id = 100
```

Example 03

```
SQL> UPDATE Product SET Amount = 19  
      WHERE Product_id = 100
```

1 row updated.

```
SQL> SELECT * FROM Product_Restock
```

Example 03

```
SQL> UPDATE Product SET Amount = 19  
      WHERE Product_id = 100
```

1 row updated.

```
SQL> SELECT * FROM Product_Restock
```

| PRODUCT_ID | PRODUCT_NAME | SUPPLIER_NAME | UNIT_PRICE | AMOUNT |
|------------|--------------|---------------|------------|--------|
| 100 | Soap | Supplier 1 | 25 | 20 |

Example 04: Trigger with User Defined Exception

```
CREATE OR REPLACE TRIGGER ACCOUNT_OVERDRAFT
BEFORE UPDATE
ON ACCOUNT

FOR EACH ROW
  WHEN ( NEW.BALANCE < 0 )
DECLARE
  OVERDRAFT_ERROR EXCEPTION;
BEGIN
  RAISE OVERDRAFT_ERROR;
EXCEPTION
  WHEN OVERDRAFT_ERROR
  THEN
    RAISE_APPLICATION_ERROR(-20001,'OVERDRAFT NOT ALLOWED' );
END;
```

Example 04

While updating account balance if balance becomes negative then it raises an application error

Example 04

```
UPDATE account  
SET balance = - 30  
WHERE account_id = 'A-113';
```

Example 04

```
UPDATE account  
SET balance = - 30  
WHERE account_id = 'A-113';
```

ORA-20343: OVERDRAFT NOT ALLOWED

ORA-06512: at "SANJIDA17.ACCOUNT_OVERDRAFT", line 8

ORA-04088: error during execution of trigger 'SANJIDA17.
ACCOUNT_OVERDRAFT'

The previous trigger is used to keep track of all the price changes performed on the product table.

Note that we can specify the **old** and **new** values of an updated row by prefixing the column names with the :OLD and :NEW qualifiers.

ALTER TRIGGERS

- `ALTER TRIGGER trigger_name DISABLE;`
- `ALTER TABLE table_name DISABLE ALL TRIGGERS;`

To enable a trigger, which is disabled, we can use:

`ALTER TRIGGER trigger_name ENABLE;`

ALTER TRIGGERS

- All triggers can be enabled for a specific table by using the following command

```
ALTER TABLE table_name ENABLE ALL  
TRIGGERS;
```

- To drop a trigger:

```
DROP TRIGGER trigger_name
```

TYPES OF PL/SQL TRIGGERS

There are two types of triggers based on the which level it is triggered.

1) Row level trigger - An event is triggered for each row updated, inserted or deleted.

2) Statement level trigger - An event is triggered for each sql statement executed.

TRIGGER PRIVILEGE

From system:

GRANT CREATE TRIGGER TO *user_name*

SEQUENCE

SEQUENCE

A Sequence

- ❑ can automatically generate unique numbers
- ❑ can be used to create a primary key value
- ❑ Speed up the efficiency of accessing sequence values when cached in memory

Syntax

```
CREATE SEQUENCE sequence_name  
    [INCREMENT BY n]  
    [START WITH n]  
    [MAXVALUE n]  
    [MINVALUE n]  
    [CACHE n | NOCACHE]  
    [CYCLE | NOCYCLE]
```

Step: 01: Create a Table (Person)

```
CREATE TABLE PERSON  
(  
  NID VARCHAR2(15) CONSTRAINT PERSON_PK PRIMARY KEY,  
  NAME VARCHAR2(50) NOT NULL,  
  BDATE DATE DEFAULT SYSDATE  
) ;|
```

Step: 02: Create a Sequence

```
CREATE SEQUENCE PERSON_NID_SEQ  
INCREMENT BY 1  
START WITH 1000000000000000001  
MAXVALUE 9999999999999999999  
NOCYCLE ;
```

The created sequence –

- Will generate numeric values starting with **1000000000000001**
- The next value is found by adding 1 with previous value
- The maximum value will be **9999999999999999**
- The NOCYCLE option ensures that after the maximum value is generated, the sequence will stop generating value.
- If you want the sequence to generate values again from the starting value, use the CYCLE option instead of NOCYCLE option.

Use the Sequence

- INSERT INTO PERSON
VALUES

```
(PERSON_NID_SEQ.NEXTVAL, 'Ahsan',  
TO_DATE('12-DEC-1980','DD-MON-YYYY')) ;
```

The sequence will provide sequential numbers for the NID column of the PERSON table. The NEXTVAL is used to retrieve next sequential number from the sequence

View the Sequence Number

```
SELECT PERSON_NID_SEQ.CURRVAL  
FROM DUAL ;
```

Example 01:

Create a sequence named
DEPT_DEPTID_SEQ to be used for the
primary key of the DEPARTMENTS table.

```
CREATE SEQUENCE dept_deptid_seq  
  INCREMENT BY 10  
  START WITH 120  
  MAXVALUE 9999  
  NOCACHE  
  NOCYCLE;
```


Example 02:

```
CREATE SEQUENCE Customers_Seq  
START WITH 1000  
INCREMENT BY 1  
NOCACHE  
NOCYCLE;
```

Example 03:

```
CREATE SEQUENCE Account_Sequence  
START WITH 1000  
INCREMENT BY 1  
CACHE 20  
NOCYCLE;
```

SEQUENCE TERMS

MAXVALUE

- Specify the maximum value the sequence can generate. This integer value can have 28 or fewer digits.
- MAXVALUE must be equal to or greater than START WITH and must be greater than MINVALUE.

NOMAXVALUE

- Specify NOMAXVALUE to indicate the maximum value of the data type for an ascending sequence or -1 for a descending sequence. **This is the default.**

ERROR SAMPLE

If the sequence reaches to the Maximum value.



ORA-08004: sequence ACCOUNT_SEQUENCE.NEXTVAL exceeds MAXVALUE and cannot be instantiated

SEQUENCE TERMS

MINVALUE

- Specify the minimum value of the sequence. This integer value can have 28 or fewer digits.
- MINVALUE must be less than or equal to START WITH and must be less than MAXVALUE.

NOMINVALUE

Specify NOMINVALUE to indicate a minimum value of 1 for an ascending sequence or the minimum value of the data type that is associated with the sequence. **This is the default.**

SEQUENCE TERMS

CYCLE

Specify CYCLE to indicate that the sequence continues to generate values after reaching either its maximum or minimum value.

After an ascending sequence reaches its maximum value, it generates its minimum value.

After a descending sequence reaches its minimum, it generates its maximum value.

NOCYCLE

Specify NOCYCLE to indicate that the sequence cannot generate more values after reaching its maximum or minimum value. **This is the default.**

SEQUENCE TERMS

CACHE

Specify how many values of the sequence the database preallocates and keeps in memory for faster access. This integer value can have 28 or fewer digits. The minimum value for this parameter is 2.

NOCACHE

Specify NOCACHE to indicate that values of the sequence are not preallocated.

If you omit both CACHE and NOCACHE, then the database caches **20 sequence numbers** by default

SEQUENCE TERMS

ORDER

Specify ORDER to guarantee that sequence numbers are generated in order of request.

NOORDER

Specify NOORDER if you do not want to guarantee sequence numbers are generated in order of request. **This is the default**

NEXTVAL and CURRVAL Pseudocolumns

- **NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.**
- **CURRVAL obtains the current sequence value.**
- **NEXTVAL must be issued for that sequence before CURRVAL contains a value.**

Modifying a Sequence

Change the increment value, maximum value, minimum value, cycle option, or cache option:

```
ALTER SEQUENCE dept_deptid_seq  
        INCREMENT BY 20  
        MAXVALUE 999999  
        NOCACHE  
        NOCYCLE;
```

Sequence altered.

Thank You