E-Business

E-Marketing

Page No.

Date: / /

PL/SQL

Unit-6

Database Triggers:-

Oracle engine allows us to define the procedures that are implicitly executed that means executed by Oracle engine itself when an Insert Delete or Update operations are executed in a table based on some conditions. These procedures are called database triggers. So, database triggers are Event-Condition-Based Actions that means actions that are executed automatically at the ocurance of an event and satisfying the given condition.

Uses of Database Triggers:-

Some of the common uses of a database trigger are:-

1. A trigger can permit DML statements against a table only if they are issued during regular business hours or on a predermined week days. That means triggers are helpful to impliment the business rules and to ensure the accuray and

consistency of data.

2. A trigger can also be used to keep the log records of various database related operation along with the operations and time on which those operations were ~~perm~~ performed. Log information is useful for database auditing.

3. It can be used to ~~stark~~ invalid transactions
   ↳ prevent

4. It also help to enforce complex security authorization.

5. It can be used to perform database operations and validate data against different business rules.

6. It can be used to generate primary key automatically.

Triggers Vs Procedures

1. Triggers do not accept parameters whereas procedure can accept 0 or any number of parameter.
2. A trigger is executed implicitly by the Oracle engine itself when certain modification on data tables are performed and a condition is being satisfied, whereas to execute a procedure it has to be explicitly called by user

PL/SQL

Elective-I

Page No.
Date: / /

Database Triggers Vs Declarative Integrity Constraints

Triggers as well as declarative integrity constraints can be used to validate the input data. However, both have some significant differences and some of them are:-

1. A declarative integrity constraints is a statement about a database that is always true.
2. A constraints applies to existing data in the table and any statement that manipulate the table.
2. Triggers also enforce integrity constraints but during the insert, update and delete operations.
2. Integrity constraints are always performed during the database operations whereas triggers are performed only when certain conditions are satisfied.
5. A trigger enforces some business rules on data during a execution of transaction which cannot be enforced by declarative integrity constraints.

Page No.
Date: / /

Types of Triggers.

While defining a trigger, the number of times the trigger action is to be executed can be specified and according to this concept there are two types of trigger as:-

(1) Row Trigger:-

A row trigger is fired each time a row in the table is affected by the triggering statement. For example:-

If an update statement updates multiple rows of a table, a row trigger is fired once for each row affected by the update statement.

If the triggering statement doesnot affect any row, the trigger is not executed at all.

(2) Statement Trigger:-

A statement trigger is fired once on behalf of triggering statement, independent of number of rows affected by the statements (Even if no rows are affected).

Page No.
Date: / /

Before Vs After Triggers:-

When defining a trigger it is necessary to specify the trigger timing, that means specifying when the triggering actions are to be executed in reference to the triggering statement. According to the timing there are two types of triggers as:-

1. Before trigger:-

Before trigger executes the trigger action before the triggering statement. These types of triggers are commonly used in the following situations:-

a.) Before triggers are used when the trigger action should determine weather or not the triggering statement should be allowed to complete By using a before trigger users can eliminate unnecessary processing of the triggering statement.

(b.) Before triggers are used to derive specific column values before completing a triggering insert or update statement.

(6) After trigger:-

After trigger executes the trigger action after the triggering statement is executed. It is commonly used in the following situations:

- When the triggering statement should complete its action before executing the trigger action.
- If a before trigger is already present then an after trigger can perform different actions on the same triggering statement.

We can create different types of triggers by the combination of above triggers as:-

(1) Before Row Trigger:- Before modifing each row affected by the triggering statement, the trigger action is executed for each row independently.

(2) Before Statement Trigger:- Before executing the triggering statement, the trigger action is executed only once.

Page No.
Date: / /

(3) After Row Trigger:- After modifying each row affected by the triggering statement and applying appropriate integrity constraints, the trigger action is executed for each row.

(4) After Statement Trigger:- After executing the triggering statement the trigger action is executed only once.

PL/SQL

Page No.
Date: / /

Creating Triggers :-

Syntax:

```
CREATE OR REPLACE TRIGGER Trigger-
                                Name
[BEFORE, AFTER] [INSERT, UPDATE,
                                DELETE]
ON TableName
[REFERENCING [OLD AS old, NEW AS
                                new]]
[FOR EACH ROW [WHEN Condition]])
DECLARE
    Variable declarations;
BEGIN
    ---------
EXCEPTION
    ---------
END;
```

(सुन कुठे Capital हवं त्यो Keyword No)

The keywords and parameters in this syn
are:

- OR REPLACE
  ↳ It recreates the table if already exists.
- BEFORE
  ↳ It indicates that the ORACLE engine fires the trigger before executing the triggering statement.
- AFTER
  ↳ It indicates that the ORACLE engine fires the trigger after executing the triggering statement.
- INSERT, UPDATE, DELETE
  ↳ It indicates that ORACLE engine fires the trigger whenever a DELETE statement removes a row from the Table, INSERT statement insert a new row or a UPDATE statement changes the value of the columns.
- ON:
  ↳ It specifies the name of the table for which the trigger is to be created.

- REFFERENCING
  ↳ It defines the co-relation between the old values and new values

- FOR EACH ROW
  ↳ It indicates the trigger as a row level trigger. If this clause is ignored then the trigger will be as statement trigger.

- WHEN
  ↳ It specifies the trigger restriction that contains a SQL condition that must be satisfied to fire the trigger.

Deleting a Trigger:

Syntax:

```
DROP TRIGGER TriggerName;
```

Generating a Primary Key Using a Database Trigger

In a multi user environment, to allow data entry ~~operatio~~ operators to create and enter a primary key value is complex because different users may enter the same value for primary key. To avoid this problem we can use a trigger to create a primary key automatically. Few simple but effective approaches that can be used to generate primary key are:-

1.) Generating primary key using sequence.
2.) Generating primary key using max function.
3.) Generating primary key using LOOKUP Table

PL/SQL

(i) Using sequence

The ORACLE engine provides an object called a sequence that generates numeric values starting with a given value and with an increment as specified by the user. Once the sequence is created, a database trigger can be written such that it retrives the next number from the sequence and uses the same number as primary key value.

For example:

A sequence with a trigger to generate the account number as primary key for account table.

Sequence creation

```
CREATE SEQUENCE accno_seq
INCREMENT BY 1
START WITH 1;
```

Database Trigger Creation :

Page No.
Date: / /

```
CREATE OR REPLACE TRIGGER accno_gen
  BEFORE INSERT
  ON Account
  FOR EACH ROW
DECLARE
  ac_pk_val Account.Accno%TYPE;
BEGIN
    SELECT accno_seq.NEXTVAL
      INTO ac_pk_val FROM DUAL;
    :NEW.accno := ac_pk_val;
```

(Table field)

(*.) Define a trigger to generate the primary key on account table using the max function.

```
CREATE OR REPLACE TRIGGER accno_gen
  BEFORE / INSERT
    ON Account
    FOR EACH ROW
DECLARE
    ac_pk_val Account.Accno%TYPE;
    max_val Account.Accno%TYPE;
BEGIN
    SELECT MAX(Accno) INTO max_val
    FROM Account;
  ac_pk_val := max_val + 1;
```

```
:NEW.acno:= ac_pk_val;
END;
```