Triggers are used to guarantee that when a specific operation is performed related operations are enforced.

Triggers are created using the CREATE TRIGGER command. When creating triggers use `/` by itself on the last line to denote the end of CREATE TRIGGER statement.

```
CREATE TRIGGER classexample
BEFORE DELETE ON parts
FOR EACH ROW
WHEN (new.part_price > 0)
DECLARE
   /* variables, constants, cursors */
BEGIN
   /* PL/SQL Block
END;

The create statement would fail if there are any errors in the PL/SQL block. You can use SHOW ERRORS to look at the compilation errors. In the above example, classexample is the name of the trigger. Trigger names must be unique with respect to other triggers in the same schema. Trigger names could be the same as the names of other objects such as tables, views and procedures within the same schema.

BEFORE or AFTER options can be used to specify when the trigger body is fired in relation to the triggering statement. AFTER row triggers are slightly more efficient, because the affected data blocks are read only once.

DELETE ON parts is the triggering statement in the above example. DELETE, INSERT, and UPDATE are possible options. You can include more than one option as in DELETE OR UPDATE OR INSERT ON parts. The triggering statement specifies the name of the table. There can be only one such table. You cannot write a trigger for a view. UPDATE triggers can specify a column list as in

BEFORE INSERT, DELETE OR UPDATE OF part_price ON Parts

FOR EACH ROW option The presence or absence of this option determines whether a trigger is a row trigger or statement trigger. In row triggers the trigger body is fired once for each affected row. The absence of this option means that the trigger body will be fired only once for the triggering statement.

The WHEN Clause is an option that can be included in the trigger to specify restriction on the trigger. WHEN clause could specify For each affected row the WHEN clause is evaluated and the trigger body fired if it evaluates to TRUE.
The Trigger Body is a PL/SQL block that can include SQL and PL/SQL statements. Note that you can include only DML statements in the trigger body.

In a trigger body of a row trigger, the code can have access to the new and old column values of the affected row subject to certain limitations. INSERT statement, for example, will have access to the new column values only. DELETE will have access to the old values only. Trigger body fired by an UPDATE statement can have access to both old and new values for both BEFORE and AFTER row triggers.

Only one trigger of each type can exist per table. This allows 12 possible triggers for each table.

```plaintext
BEFORE UPDATE row       AFTER UPDATE row
BEFORE DELETE row        AFTER DELETE row
BEFORE INSERT row        AFTER INSERT row
BEFORE UPDATE statement  BEFORE UPDATE statement
BEFORE INSERT statement  AFTER INSERT statement
BEFORE DELETE statement  AFTER DELETE statement
```

If a trigger body is fired by more than one type of DML statements as in

```plaintext
DELETE OR UPDATE OR INSERT ON parts
```

You can specify different actions in the trigger body for each scenario as in

```plaintext
IF INSERTING THEN... END IF;
```

Lab 1

```sql
CREATE TABLE inventory (part_no VARCHAR2(3) primary key,
                        qty_on_hand integer,
                        reorder_point integer,
                        reorder_quantity integer);

CREATE TABLE pending_orders (part_no VARCHAR2(3),
                              order_quantity integer,
                              date_entered date);

A foreign key reference in pending orders table would cause failure of the trigger

```sql
insert into inventory values ('100', 545, 400, 5000);
insert into inventory values ('101', 234, 200, 2000);
insert into inventory values ('102', 405, 400, 2500);
insert into inventory values ('103', 367, 300, 3000);
insert into inventory values ('104', 222, 100, 8000);
insert into inventory values ('105', 234, 200, 7000);
insert into inventory values ('106', 457, 300, 8000);
insert into inventory values ('107', 456, 300, 9000);
```
CREATE OR REPLACE TRIGGER vijay_reorder
AFTER UPDATE OF qty_on_hand ON Inventory
FOR EACH ROW
WHEN (new.qty_on_hand < new.reorder_point)
DECLARE
  x number;
BEGIN
  SELECT COUNT(*) INTO x
  FROM  pending_orders
  WHERE part_no = :new.part_no;
  IF x = 0 THEN
    INSERT INTO pending_orders
    VALUES (:new.part_no, :new.reorder_quantity,
    sysdate); 
    END IF;
END;
/

Now,
UPDATE inventory SET qty_on_hand = 200 where part_no =
'100'; fires the trigger and inserts a row into the pending
orders table.

Examples of Trigger Applications:

- Auditing
- Preventing invalid transactions
- Enforce referential integrity (across nodes on a distributed application)
- Enforce complex business rules
- Complex security authorizations
- Provide transparent event logging
- Automatically generate derived columns
- Replicate a table synchronously