Triggers

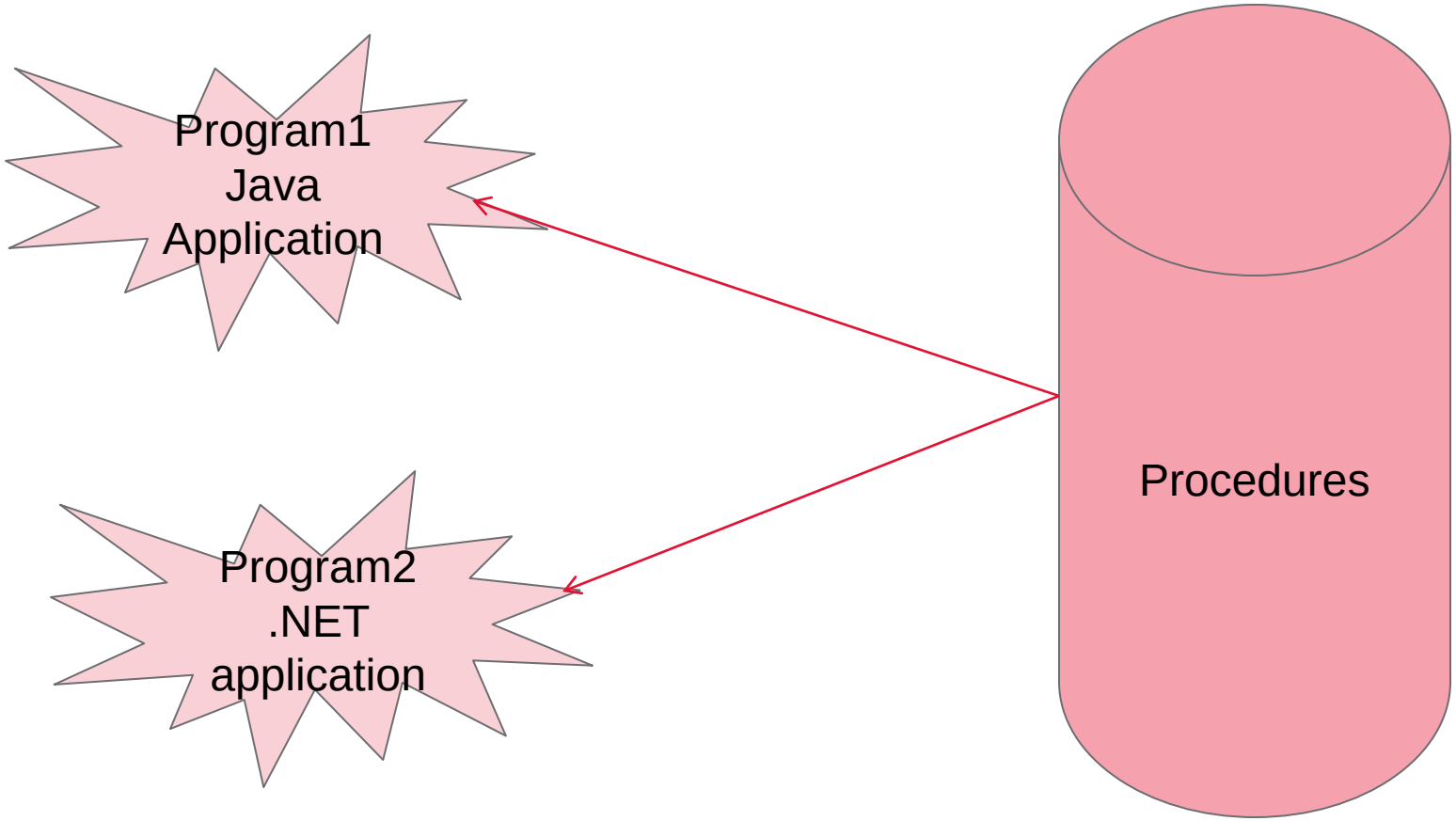
Objectives

- After completing this lesion, you should be able to do the following:
  - Describe different types of triggers
  - Describe database triggers and their use
  - Create Database triggers
  - Describe database trigger firing rules
  - Remove database triggers

Introduction

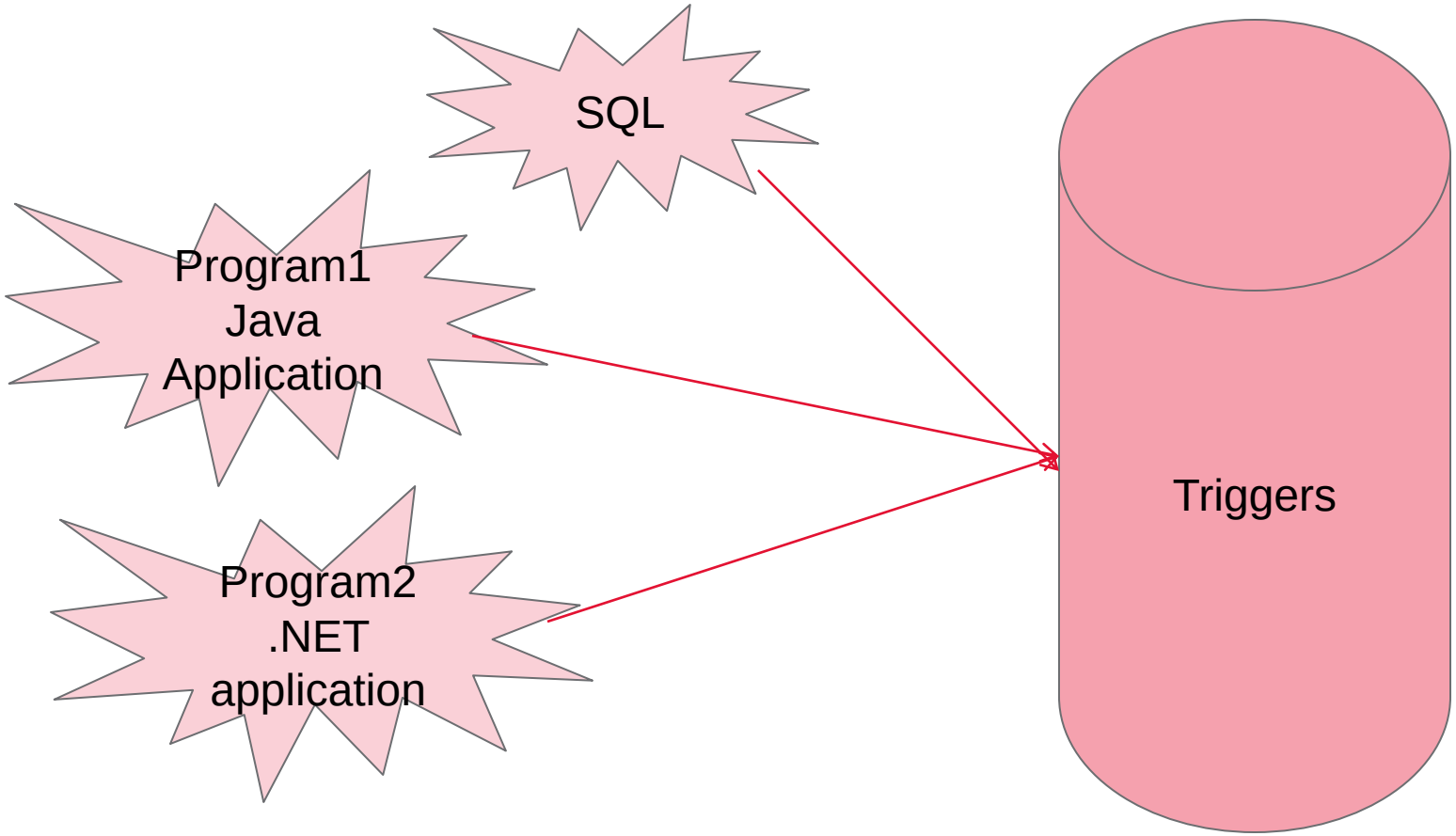
- Oracle allows you to define procedures that are implicitly executed when an INSERT, UPDATE, or DELETE statement is issued against the associated table.
- These procedures are called database triggers. Triggers are similar to stored procedures
- A trigger can include SQL and PL/SQL statements to execute as a unit and can invoke stored procedures.
- Procedures and triggers differ in the way that they are invoked. While a procedure is explicitly executed by a user, application, where triggers are implicitly fired (executed) by Oracle when a triggering INSERT, UPDATE, or DELETE statement is issued, no matter which user is connected or which application is being used.

How Procedures Are Used



Procedure should be called explicitly into the application program. So procedure should be invoked explicitly.
Load on the application is going to increase.

How Triggers Are Used



Even though different applications are running on the database, Triggers are going to fire implicitly. No need to call them through application.

Trigger - Levels

- These Triggers are written at three different levels
  - Schema
  - Table
  - Row

    - A Schema level trigger is a trigger which is written at database level or user level.. These triggers can be written by DBA only.

    - Any user can write table and ROW level triggers. Table level triggers are ment for providing security at object (Table) level.

    - Row level triggers are ment for validations

Triggers

- A trigger:
  - is a PL/SQL block or a PL/SQL procedure associated with a table, view, schema, or the database.
  - Executes implicitly whenever a particular event takes place

    - Can be either
      - Application trigger: Fires whenever an event occurs with a particular application.
      - Database trigger: Fires whenever a data event (such as DML) or system event (such as logon or shutdown) occurs on a schema or database.

Types of Triggers

- ROW Triggers and Statement Triggers
- BEFORE or AFTER Triggers
- INSTRAD-OF triggers
- Triggers on System events and User Events

Row Triggers

- A row trigger is fired each time the table is affected by 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 a triggering statement affects no rows, a row trigger is not executed at all.

- A Row trigger is fired once for each row affected by the command. These triggers are used to check for the validity of the data in the triggering statements and rows affected.

Statement Triggers

- A statement trigger is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects (even no rows are affected).

- For example, if a DELETE statement deletes several rows from a table, a statement level DELETE trigger is fired only once, regardless of how many rows are deleted from the table.

BEFORE and AFTER Triggers

- When defining a trigger, you can specify the trigger timing- whether the trigger action is to be executed before or after the triggering statement. BEFORE and AFTER apply to both statement and row triggers.

- BEFORE and AFTER triggers fired by DML statements can be defined only on tables, not on views. However, triggers on the on the base table(s) of a view fired if an INSERT, UPDATE, or DELETE statement is issued against the view.

- BEFORE and AFTER triggers fired by DDL statements can be defined only on the database or a schema, not on particular tables.

When to use BEFORE triggers?

- For example, when we are inserting a row in to the CUSTOMER table(CID, CNAME, address), If we trying to place new customer details, we need to validate and check for existing customer :

- This validation should be done before inserting a new row into the table. In such situations, we have to write BEFORE triggers.

When to use AFTER Triggers?

- Consider a Banking scenario : Where we are maintaining Customer details and their day to day operations.

BANK MASTER

Accno	Account_holder_name	Balance

Bank transaction

Transaction_no	transaction_date	Accno	Transaction_type	Amount

In this scenario, if we want to update the account balance, when customer makes the transaction, your trigger should fire to update the balance only after customer transaction is successful.

INSTEAD-OF Triggers

- These triggers provide a transparent way of modifying views that cannot be modified directly through SQL DML statements. These triggers are called INSTEAD-OF triggers because, unlike other types of triggers, Oracle fires the trigger instead of executing the triggering statement

Guidelines for Designing Triggers

- Design triggers to:
  - Perform related actions
  - Centralize global operations
- Do not design triggers :
  - Where functionality is already built into the Oracle server
  - That duplicates other triggers
- Create stored procedures and invoke them in a trigger, if the PL/SQL code is very length.
- The excessive use of triggers can result in complex interdependencies, which may be difficult to maintain in large applications.

How triggers are Used?

- Triggers can supplement the standard capabilities of Oracle to provide a highly customized database management system.
- For example,
  - A trigger can restrict DML operations against a table to those issued during regular business hours.
  - A trigger could also restrict DML operations to occur only at certain times during weekdays.
- Other uses of triggers are to
  - Prevent invalid transaction
  - Enforce complex security authorizations
  - Enforce complex business rules
  - Gather statistics on table access
  - Modify table data when DML statements are issued against views

Syntax : *Table Level Trigger*

- Syntax:

```
CREATE OR REPLCE TRIGGER <triggername>
    BEFORE | AFTER INSERT OR UPDATE OR DELETE ON  <table>
   [DECLARE]
   BEGIN
     ST1;
     ST2;
 [EXCEPTION]

 END;
```

ROW Level Trigger

- Syntax:

```
CREATE OR REPLCE TRIGGER <triggername>
 BEFORE | AFTER INSERT OR UPDATE OR DELETE ON  <table>
 FOR EACH ROW
[DECLARE]
BEGIN
  ST1;
  ST2;
[EXCEPTION]

END;
```

Example to create Table level or Statement triggers

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT ON emp
BEGIN
IF TO_CHAR(SYSDATE,'DY') IN('SAT','SUN') OR
TO_CHAR(SYSDATE,'HH24:MI') NOT
BETWEEN '08:00' AND '18:00' THEN
RAISE_APPLICATION_ERROR(-20001,'You may Insert into EMP table only
during
business hours..');
END IF;
END;
/
```

Note: RAISE_APPLICATION_ERROR is a server-side built-in procedure that returns an error to the user and causes the PL/SQL block to fail.

When a database trigger fails, the Oracle server automatically rolls the triggering statement back.

Test the trigger

INSERT INTO EMP (empno, ename) VALUES (101,'ravi');
ERROR at line 1
ORA-20001 You may insert into EMP table only during business hours
ORA-06512 at PLSQL SECURE_emp, LINE 4
Ora-04088 error during execution of trigger "PLSQL SECURE_EMP"

Note: The row might be inserted if you are in a different time zone from the database server. The trigger fires even if your system clock is within these business hours.

Combining Triggering Events

- You can combine several triggering events into one by taking advantage of the special conditional predicates INSERTING, UPDATING and DELETEING within the trigger body.
- Create one trigger to restrict all the data manipulation events on the EMP table to certain business hours, Monday through Friday.

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT OR UPDATE OR DELETE  ON emp
BEGIN
   IF TO_CHAR(SYSDATE,'DY') IN('SAT','SUN')  OR
     TO_CHAR(SYSDATE,'HH24:MI') NOT BETWEEN '08:00' AND '18:00' THEN
        IF INSERTING THEN
          RAISE_APPLICATION_ERROR(-20001,'You may Insert into EMP table
               only   during business hours..');
         ELSIF UPDATING THEN
          RAISE_APPLICATION_ERROR(-20001,'You may UPDATE  EMP table
               only   during business hours..');
         ELSIF DELETING THEN
          RAISE_APPLICATION_ERROR(-20001,'You may DELETE from  EMP
              table only   during business hours..');
         END IF;
   END IF;
END;
/
```

Using OLD and NEW Qualifiers

- Within a ROW trigger, reference the value of a column before and after the data change by prefixing it with the OLD and NEW qualifiers.

Data Operation	Old Value	New Value
INSERT	Null	Inserted value
UPDATE	Value before update	Value after update
DELETE	Value before delete	NULL

- The OLD and NEW qualifiers are available only in ROW triggers
- Prefix these qualifiers a colon (:) in every SQL and PL/SQL statement
- There is no colon (:) prefix if the qualifiers are referenced in the WHEN restricting condition

Note: Row triggers can decrease the performance if you do a lot of updates on larger tables.

Creating ROW Trigger

- You can create a BEFORE row trigger in order to prevent the triggering operation from succeeding if a certain condition is violated.

- Write a trigger to restrict the user when they are updating the salary which is less than original salary of the employee.

```
CREATE TRIGGER ALLOW_UPDATE
BEFORE UPDATE ON EMP
FOR EACH ROW
BEGIN
    IF :NEW.SAL < :OLD.SAL THEN
      RAISE_APPLICATION_ERROR(-20002,'NOT A VALID UPDATION');
    END IF;
END;
/
```

Using WHEN clause

```
create trigger allow_update
before update on programmer
for each row
WHEN(new.salary < old. Salary)
begin
   raise_application_error(-20004,'not a valid update');
end;
```

Trigger to gather statistics

- When different users are performing INSERT, UPDATE and DELETE operations on a particular table, We can write a trigger to monitor which user is performing these operations.

- Create a audit_table for monitoring users operations:

Column	Data type
USERNAME	VARCHAR2(20)
TRANSACTION_DATE	TIMESTAMP
KEY_VALUE	NUMBER
TRANSACTION	VARCHAR2(20)

Trigger definition

```
CREATE OR REPLACE TRIGGER ALLOW_UPDATE
BEFORE UPDATE ON EMP
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO AUDIT_TABLE
VALUES(USER,LOCALTIMESTAMP,:NEW.EMPNO,'INSERT');
    ELSIF UPDATING THEN
        INSERT INTO AUDIT_TABLE
VALUES(USER,LOCALTIMESTAMP,:NEW.EMPNO,'UPDATE');
    ELSIF DELETING THEN
        INSERT INTO AUDIT_TABLE
VALUES(USER,LOCALTIMESTAMP,:NEW.EMPNO,'DELETE');
    END IF;

END;
/
```

Calling procedure inside a trigger

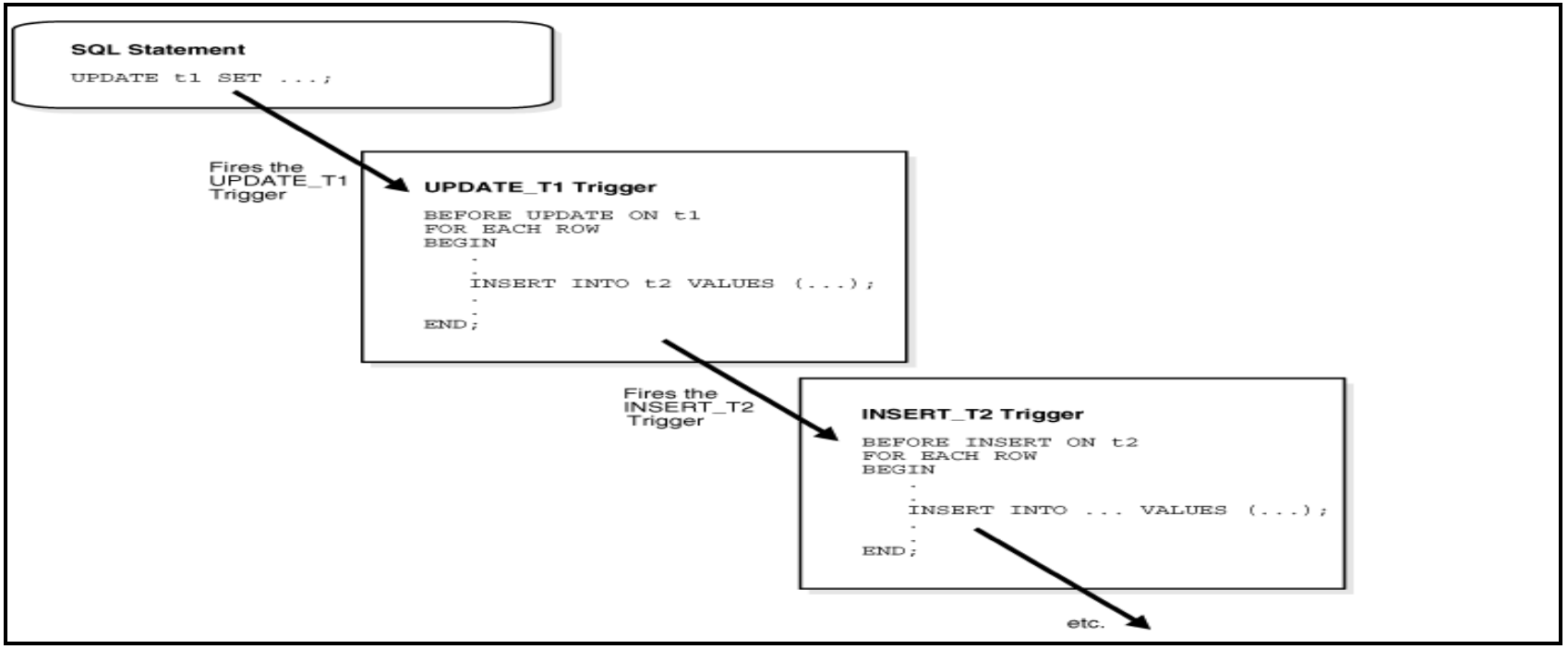
```
CREATE PROCEDURE ADD_ROW(P_EMPNO NUMBER, P_TRANSACTION VARCHAR2)
IS
BEGIN
  INSERT INTO AUDIT_TABLE VALUES(USER,LOCALTIMESTAMP,P_EMPNO,P_TRANSACTION);
END;
/

CREATE OR REPLACE TRIGGER ALLOW_UPDATE
BEFORE INSERT OR UPDATE OR DELETE ON EMP
FOR EACH ROW
BEGIN
      IF INSERTING THEN
            ADD_ROW(:NEW.EMPNO,'INSERT');
      ELSIF UPDATING THEN
            ADD_ROW(:NEW.EMPNO,'UPDATE');
      ELSIF DELETING THEN
            ADD_ROW(:NEW.EMPNO,'DELETE');
      END IF;

END;
/
```

Trigger States

- A trigger can be in either of two states:
  - Enabled. An enabled trigger executes its trigger body if a triggering statement is entered and the trigger restriction (if any) evaluates to TRUE.
  - Disabled. A disabled trigger does not execute its trigger body, even if a triggering statement is entered and the trigger restriction (if any) evaluates to TRUE.
- ALTER TRIGGER TRIGGERNAME ENABLE | DISABLE
- ALTER TABLE TABLENAME DISABLE | ENABLE ALL TRIGGERS
- DROP TRIGGER TRIGGERNAME;

Some Cautionary Notes about Triggers

- Although triggers are useful for customizing a database, use them only when necessary. Excessive use of triggers can result in complex interdependencies, which can be difficult to maintain in a large application.
- For example, when a trigger fires, a SQL statement within its trigger action potentially can fire other triggers, resulting in cascading triggers. This can produce unintended effects.



Triggers Compared with Declarative Integrity Constraints

- You can use both triggers and integrity constraints to define and enforce any type of integrity rule. However, Oracle strongly recommends that you use triggers to constrain data input only in the following situations:

  - To enforce referential integrity when child and parent tables are on different nodes of a distributed database
  - To enforce complex business rules not definable using integrity constraints
  - When a required referential integrity rule cannot be enforced using the following integrity constraints:
    - NOT NULL, UNIQUE
    - PRIMARY KEY
    - FOREIGN KEY
    - CHECK
    - DELETE CASCADE
    - DELETE SET NULL

Exercise

- Write a trigger to access the table in a specified time
- Write a trigger to accept unique values in to a particular column of a table, when we are placing a new record.
- Write a trigger to update stock automatically when customer makes a transaction using below tables
  - Itemmaster (Itemno, itemname, stock)
  - Itemtran(transaction_no,itemno,transaction_date,transaction_type, quantity)