Memo: Trigger definition

```
CREATE [OR REPLACE] TRIGGER Trigger_Name
      BEFORE | AFTER
      INSERT | DELETE | UPDATE     |     [INSERT] [[OR] DELETE] [[OR] UPDATE]
      ON Table_Name
      [FOR EACH ROW]
      -- PL/SQL block
```

Exercise #1: Automatic primary key generation

Let be any table TBL, whose primary key *NumPK* is numerical (integer). Define a trigger before insertion that automatically affects a number to the primary key. The first value must be 1.

1. Create table TBL (NumPK) with SQL.

2. Code the trigger (name it *TrigPK*) and create it.

3. Insert several rows into table TBL with NULL (or any) values instead of *NumPK*, which will automatically fire *TrigPK* each time. Then, delete one row and insert one last row.

4. Check whether the result is right by listing table TBL's contents.

Exercise #2: Dynamic domain constraint

Let us consider table BANK_CUSTOMER (CustID, CustName, HusbandWifeID#). We want to code a trigger before insert or update that controls that the name of a customer's husband or wife is the same as the customer's (pretty conservative, isn't it?). A fatal error should interrupt data insertion or update if names differ.

1. With SQL, create the structure of table BANK_CUSTOMER.

2. Code the trigger (name it *TrigBank*) and create it.

3. Insert and update several rows into table BANK_CUSTOMER. Conclusion?

Exercise #3: Automatic usage statistics

We want to store statistics about data modifications (insertions, updates and deletions) in table EMP.

1. With SQL, create the structure of table STATS (ModType, ModNumber, ModLastDate) and populate it as follows.

ModType	ModNumber	ModLastDate
INSERT	0	NULL
UPDATE	0	NULL
DELETE	0	NULL

2. Define a trigger after insert or update or delete on table EMP that automatically updates table STATS. Test its functioning by performing various modifications in table EMP.

Hints:

- Modification type determination (in trigger):
  ```
  IF INSERTING THEN -- insertion
  IF UPDATING THEN  -- update
  IF DELETING THEN  -- deletion
  ```
- System date: `SYSDATE`

3. Test the effect of the presence and absence of clause `FOR EACH ROW` on the trigger's behavior by using a query that updates several rows (e.g., `UPDATE EMP SET SAL = SAL * 1.05`).

Exercise #4: Materialized view refreshing

1. Copy table DEMO_CUSTOMERS of user DARMONT onto your account.

2. With SQL, create a materialized view (i.e., a table) named CUSTNAMES with respect to query `SELECT CUSTOMER_ID, CUST_FIRST_NAME, CUST_LAST_NAME FROM DEMO_CUSTOMERS`.

3. Unlike a classical view, the contents of a materialized views is not recomputed when it is queried. This helps achieve better response times. However, the materialized view must be maintained, i.e., it must be updated when its data source is modified. Write a trigger after insert, update or delete on DEMO_CUSTOMERS that automatically refreshes view CUSTNAMES' data. Every inserted, updated or deleted row from DEMO_CUSTOMERS must also be modified in CUSTNAMES.

4. Test the effects of your trigger by inserting, updating and then deleting a row from table DEMO_CUSTOMERS. Check at each step whether your update is correctly echoed in CUSTNAMES.