Data Science and Database Technology

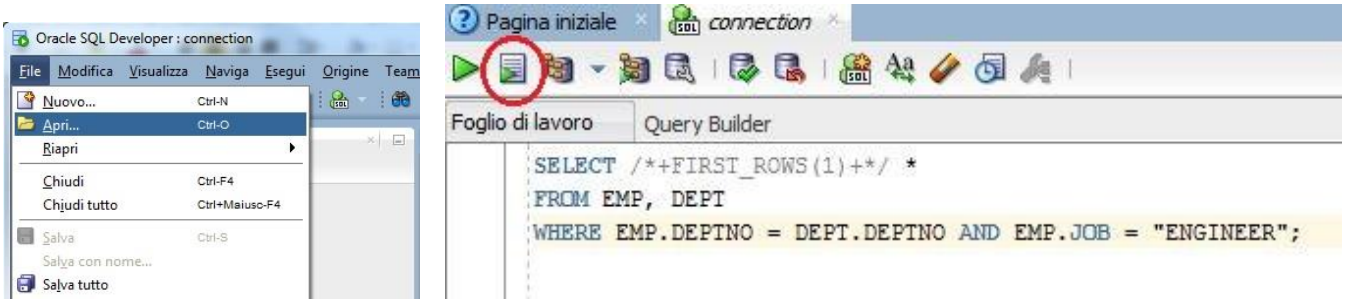
Practice #4 – Oracle Triggers

Available materials

Three scripts with SQL statements are available to create the databases.

The scripts are available at the course website in the `create_db scripts.zip` archive
https://dbdmg.polito.it/wordpress/teaching/data-science-and-database-technology-2020-2021/

The scripts can be loaded clicking on "Open" in the File Menu and selecting the .sql file. To execute the script click on the "Esegui Script" button as shown in the following figure.



Useful SQL statements

Delete a trigger:

```
drop trigger triggerName;
drop trigger "triggerName";
```

Update of an existing trigger (instead of delete and recreate it):

```
CREATE OR REPLACE TRIGGER triggerName …
```

Display defined triggers:

```
select trigger_name, triggering_event, table_name,
status, description, action_type, trigger_body
from user_triggers;
```

Disable a trigger:

```
ALTER TRIGGER triggerName DISABLE;
```

Display trigger errors:

```
select * from USER_ERRORS;
```

Suggestions

Before doing each exercise, load the related tables by executing the scripts script_db_es1.sql, script_db_es2.sql. script_db_es3.sql

To create a trigger, pay attention to the syntax and to the following issues:

- assign a proper name to the variables avoiding keywords like MIN, MAX, …
- declare different variables on different lines and not on the same line delimited by a comma

```
MyVarOne NUMBER;
MyVarTwo NUMBER;
MyVarThree VARCHAR2(16);
```

- terminate the statements with **;** character and assign new values to the variables with **:=**, e.g.

```
UPDATE tablename
```

```
   SET varname=newvalue
   WHERE column=:NEW.attribute;

IF A<3 OR A=3 THEN
   MyVar:='Three';
ELSE
IF A>3 AND A<5 THEN
   MyVar:='Four';
ELSE
   MyVar:='Other';
END IF;
END IF;
```

Before starting this practice we suggest you to delete any existing trigger on the selected account, which could affect the outcome of the exercises. To check for triggers and be able to delete them browse on the "Connessioni" tab on the left.

Remark. Pay attention because copying and pasting SQL code lines from the practice text in pdf format may generate errors due to invalid character encoding or conversion, such as *ora-00911: invalid character.*

Exercise #1

The following relations are given (primary keys are underlined, optional attributes are denoted with *)

IMP (EMPNO, DEPTNO, ENAME, JOB, SAL)
DIP (DEPTNO, DNAME, LOC, MINSAL, MAXSAL)

Write the trigger which manages the update of the DNAME attribute on DIP table. When the DNAME attribute changes from 'ACCOUNTING' to 'SALES', the wage (SAL attribute) for all employees, who work in the corresponding DEPTNO, is increased by 100.

Procedure:

- Create the database using script `create_db1.sql`
- Create the trigger, eventually by means of a script.
- Verify the content of IMP and DIP table.
- Modify the department name 'ACCOUNTING':
  UPDATE DIP set DNAME = 'SALES' where DNAME='ACCOUNTING';
- Verify the content of the IMP and DIP tables.

The following steps should be performed:

- ***write the trigger;***
- ***verify the output generated (result details via Web interface as shown in the figure) where you can observe the obtained result.***

Exercise #2

Let us consider the following database, which collects information about tickets (TICKETS) for flights (FLIGHTS) bought by users who can hold promotional cards (CARDS). For each ticket bought with a promotional card, the card is charged by a credit (in miles) that corresponds to the flight length. The promotional card status depends on the total amount of miles corresponding to the flights bought by the card owner. Specifically, from 0 to 30 miles the card status is SILVER, from 30 to 50 miles GOLD, whereas beyond 50 miles the card status is PREMIUM. Card status changes trigger notifications (NOTIFY) to the card owner.

The following relations are given (primary keys are underlined, optional attributes are denoted with *)

CARDS (CARDNO, NAME, STATUS)
FLIGHTS (FLIGHTID, DEPARTURETIME, DEPARTURECITY, ARRIVALCITY, MILES)
TICKETS (TICKETID, FLIGHTID, FLIGHTDATE, NAME, CARDNO*)
CREDITS (TICKETID, CARDNO, MILES)
NOTIFY (CARDNO, NOTIFYNO, NOTIFYDATE, OLDSTATUS, NEWSTATUS, TOTALMILES)

Write the trigger which manages a new ticket issue (i.e., ticket emission). When a ticket is issued, if it is associated to a *CARDNO* of a frequent-flyer customer (*CARDNO is NOT NULL*) it is necessary to update the covered miles by means of a new insertion in the *CREDITS* table. It is also necessary to verify the customer status and, if required, update it. The initial status of each frequent-flyer customer is "*SILVER*" and it does not change until the customer does not amass a total miles greater than 30.000. When the total miles are included between 30.000 and 50.000 the customer status is updated to "*GOLD*", while if the total miles are greater than 50.000 the customer status changes in "*PREMIUM*" (maximum level). If a customer changes his/her status, it is necessary to insert a new record to the *NOTIFY* table to inform the customer for the status variation. Note that the *NOTIFYNO* attribute is a counter which is increased of one each time a new message is inserted in the *NOTIFY* table for the same customer.

We advice you to write the trigger in 3 different steps.

Step 1: When a ticket is issued, if it is associated to a *CARDNO* of a frequent-flyer customer (*CARDNO is NOT NULL*) it is necessary to update the covered miles by means of a new insertion in the *CREDITS* table.

Step 2: The initial status of each frequent-flyer customer is "*SILVER*" and it does not change until the customer does not amass a total miles greater than 30.000. When the total miles are included between 30.000 and 50.000 the customer status is updated to "*GOLD*", while if the total miles are greater than 50.000 the customer status changes in "*PREMIUM*" (maximum level).

Step 3: If a customer changes his/her status, it is necessary to insert a new record to the *NOTIFY* table to inform the customer for the status variation. Note that the *NOTIFYNO* attribute is a counter which is increased of one each time a new message is inserted in the *NOTIFY* table for the same customer.

Procedure:

1. Create the database using script `create_db2.sql`
2. Create the trigger so as it satisfies requirements specified at step 1
3. Verify trigger correctness by inserting two records in Table TICKETS
   - one record with CARDNO equal to NULL, e.g.
     ```
     INSERT INTO TICKETS (TICKETID, FLIGHTID, FLIGHTDATE, NAME, CARDNO)
     VALUES ('T02', 'RN12K', '01-MAR-07', 'PIPPO', NULL);
     ```
   - another record with CARDNO equal to a value already existing in Table CARDS, e.g.
     ```
     INSERT INTO TICKETS (TICKETID, FLIGHTID, FLIGHTDATE, NAME, CARDNO)
     VALUES ('T03', 'RN12K', '02-APR-07', 'BILL', 50);
     ```
4. Look into the table content and verify its correctness
   ```
   select * from CREDITS;
   select * from TICKETS;
   ```
5. Update the trigger so as it satisfies requirements specified at step 2
6. Verify trigger correctness by inserting two records in Table TICKETS so that user card status changes
   - Insert one or more tickets for very long journeys or for users who have already bought other flights, e.g..
     ```
     INSERT INTO TICKETS (TICKETID, FLIGHTID, FLIGHTDATE, NAME, CARDNO)
     ```

```
VALUES ('T04', 'RN12K', '03-MAY-07', 'BILL', 50);
INSERT INTO TICKETS (TICKETID, FLIGHTID, FLIGHTDATE, NAME, CARDNO)
VALUES ('T05', 'RN12K', '03-MAY-07', 'BILL', 50);
```

7. Look into the table content and verify its correctness
```
select * from CREDITS;
select * from TICKETS;
select * from CARDS;
```
8. Update the trigger so as it satisfies requirements specified at step 3
9. Verify trigger correctness by inserting two records in Table TICKETS so that the card status for another user changes
   - Insert one or more tickets for very long journeys or for users who have already bought other flights, e.g.
```
INSERT INTO TICKETS (TICKETID, FLIGHTID, FLIGHTDATE, NAME, CARDNO)
VALUES ('T06', 'RN12K', '03-MAY-07', 'BILL', 50);
INSERT INTO TICKETS (TICKETID, FLIGHTID, FLIGHTDATE, NAME, CARDNO)
VALUES ('T07', 'RN12K', '03-MAY-07', 'BILL', 50);
```
10. Look into the table content and verify its correctness
```
select * from NOTIFY;
select * from CREDITS;
select * from TICKETS;
select * from CARDS;
```

The following steps should be performed:

- **write the trigger code;**
- **Verify the results achieved at each step by checking the database content changes.**

Exercise #3

The following relations are given (primary keys are underlined, optional attributes are denoted with *)

IMP (EMPNO, ENAME, JOB, SAL)
SUMMARY (JOB, NUM)

In the SUMMARY table, the NUM attribute specifies the number of employees in the IMP table who perform the same job. Write the triggers to guarantee the consistency between the IMP and SUMMARY tables when:

- A new record is inserted in the IMP table
- The value of job in the IMP table is updated

Create the database using script `create_db3.sql`