**OPEN ENDED LAB:**

**Scenario:**

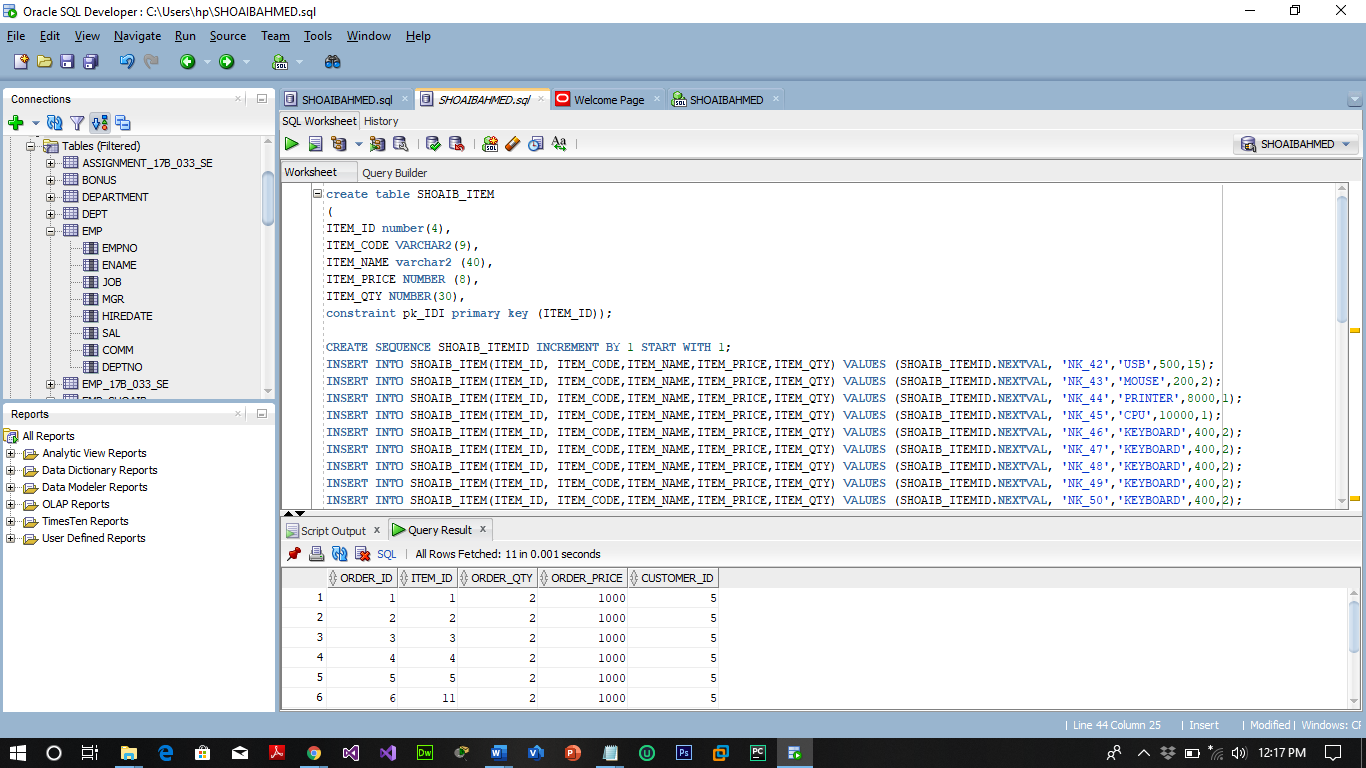
You have been hired by some Super Store to maintain database for their products using SQL Developer environment. Unfortunately, no other software is available for building a GUI and you have to build up manual queries to add new entries or show data available in the database.

You should be able to do the following tasks:

1. Create a table having name as **your full name.** 
   1. Table should have the following columns:
      1. Item ID Integer (primary key, not null & generated with sequence)

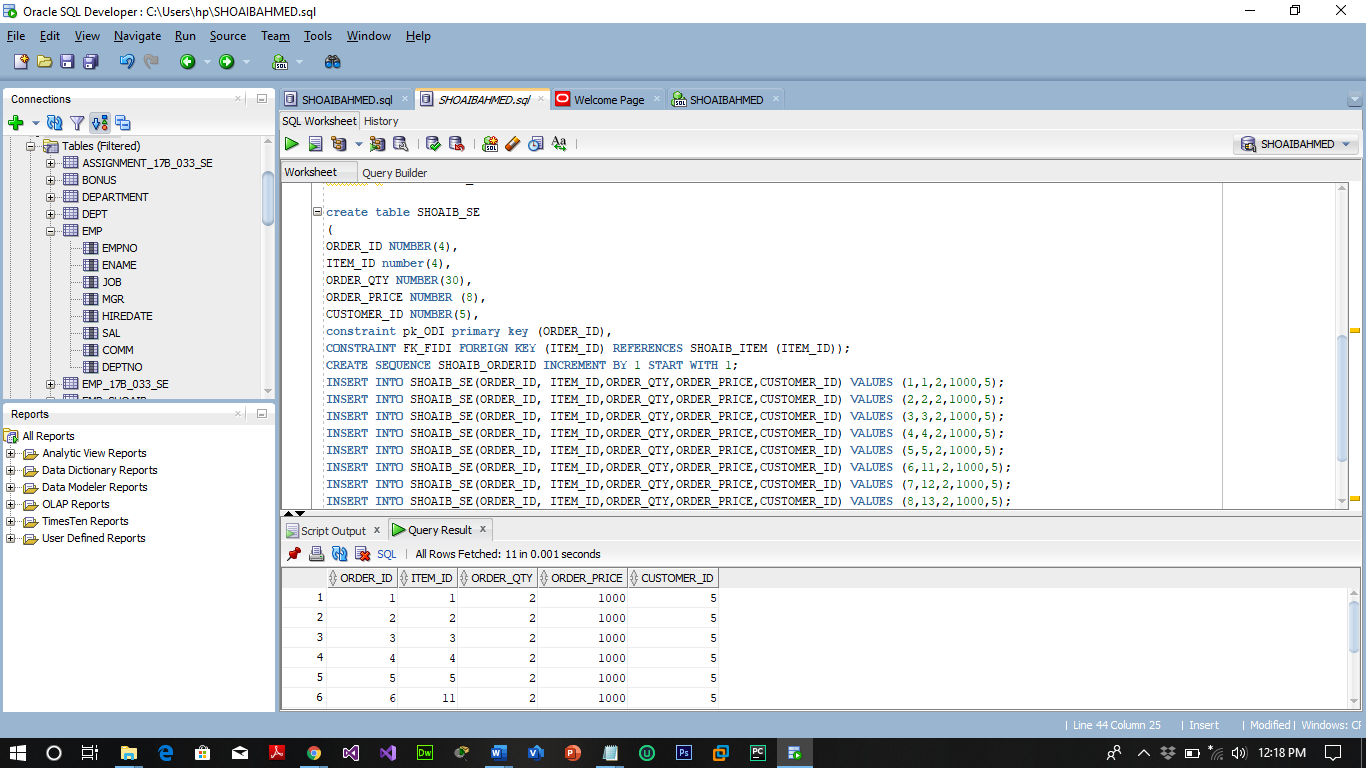
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| It\_ID | It\_Code | It\_Na | It\_Pri | It\_qty |
| 1 | NK\_42 | USB | 500 | 15 |
| …… | …… | …… | …… | …… |

* + 1. Item Code Varchar2
    2. Item Name Varchar2
    3. Item Price Number
    4. Item Qty Number

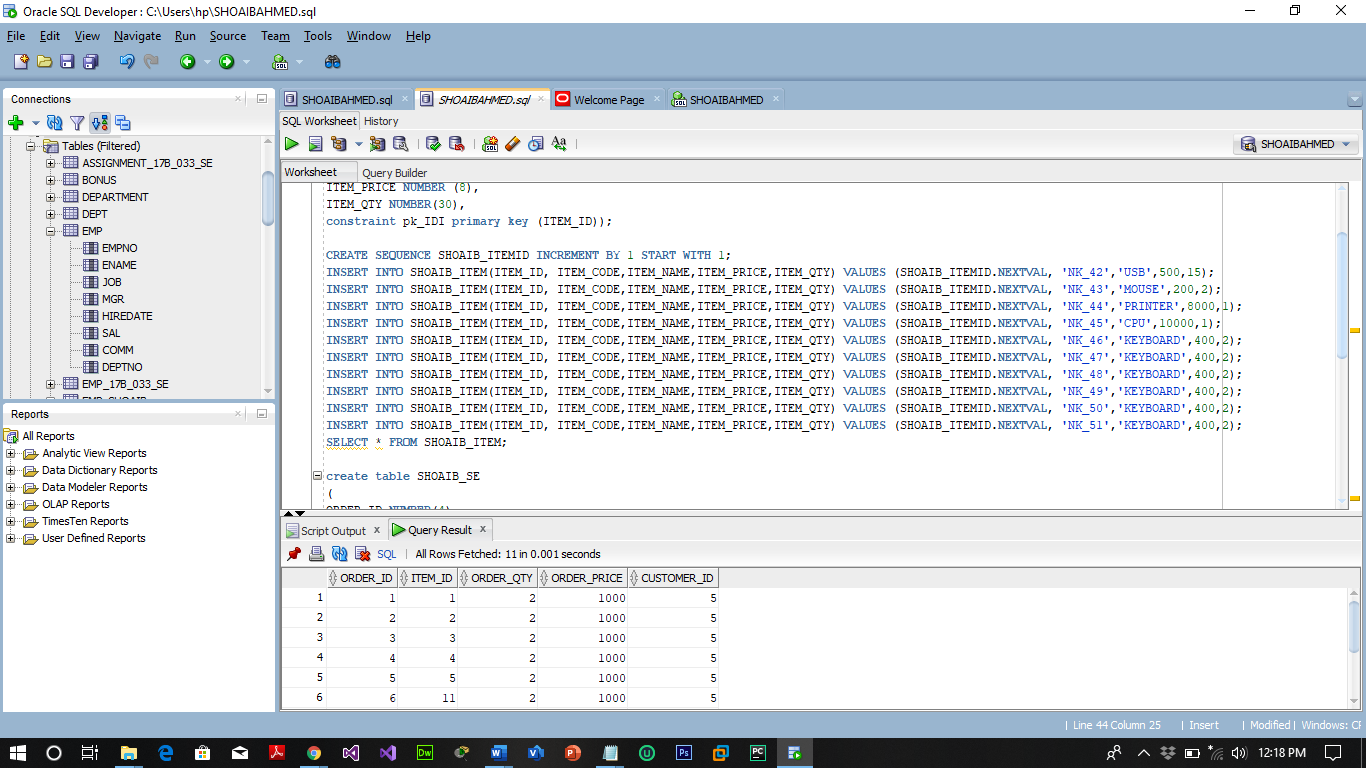


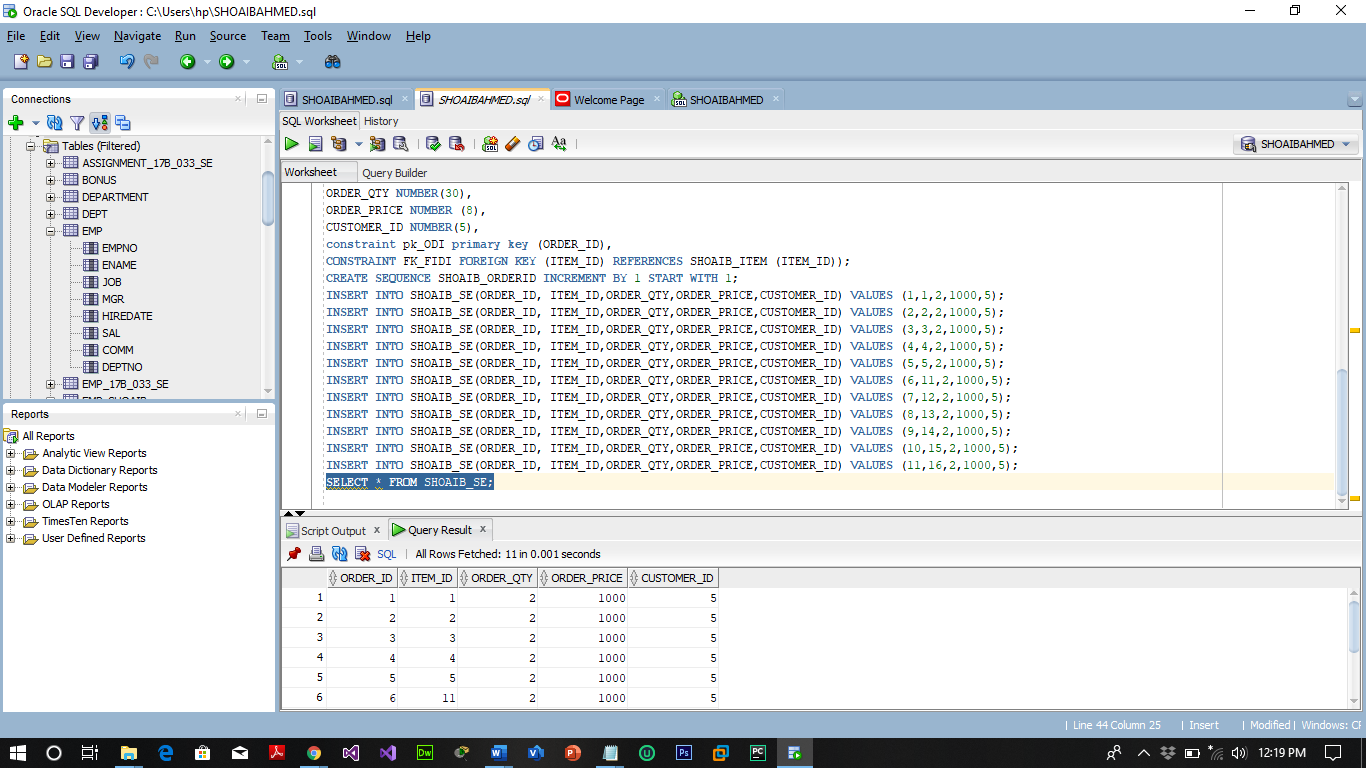
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Or\_ID | IT\_ID | Or\_Qty | Or\_Pri | Cus\_ID |
| 1 | 36 | 2 | 200 | 5 |
| 2 | 43 | 7 | 3500 | 5 |
| 3 | 67 | 5 | 2500 | 11 |
| … | …. | … | …. | …. |

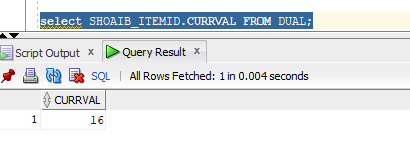
1. Create another table having name as **your first name \_SE**
   1. Table should have the following columns:
      1. Order ID Integer
      2. Item ID Integer (Foreign Key)
      3. Order Qty Integer
      4. Order Price Integer
      5. Customer ID Integer



1. Make some query to insert 10 rows in both tables as order by any customer of a particular item.







1. During the insertion in Second table if order qty is not available or less, a message should be generated telling the user “Item not available!” (**hint:** create a trigger on insert in second table to check the order quantity should be less then item quantity in first table)

**CREATE OR REPLACE TRIGGER ORDERQTY**

**BEFORE INSERT ON SHOAIB\_SE**

**FOR EACH ROW**

**DECLARE**

**S\_QUANTITY NUMBER(10);**

**BEGIN**

**SELECT ITEM\_QTY INTO S\_QUANTITY FROM SHOAIB\_ITEM WHERE ITEM\_ID=:new.ITEM\_ID;**

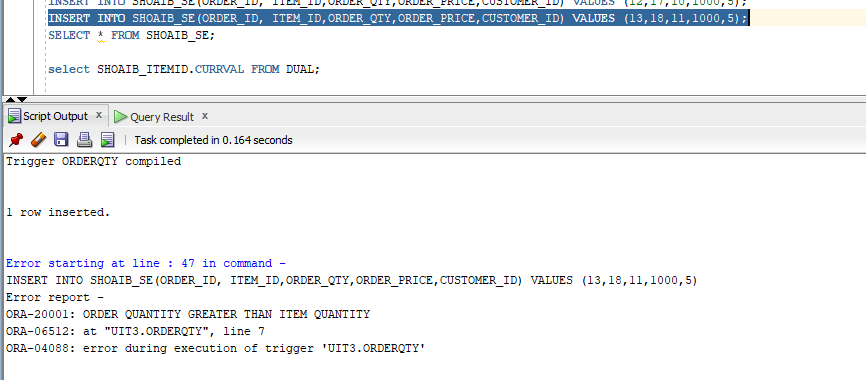
**IF :new.ORDER\_QTY > S\_QUANTITY**

**THEN**

**RAISE\_APPLICATION\_ERROR(-20001,'ORDER QUANTITY GREATER THAN ITEM QUANTITY');**

**end if;**

**END;**



**SCENERIO 2:**

**Tables:**

**Highschooler(ID int, name text, grade int);**

**Friend(ID1 int, ID2 int);**

**Likes(ID1 int, ID2 int);**

1 - Write one or more triggers to maintain symmetry in friend relationships. Specifically, if (A,B) is deleted from Friend, then (B,A) should be deleted too. If (A,B) is inserted into Friend then (B,A) should be inserted too. Don't worry about updates to the Friend table

**QUERY:**

**CREATE TRIGGER FRIEND\_INSERT**

**AFTER INSERT ON friend**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO friend (ID1,ID2) VALUES (:new.id2, :new.id1);**

**END;**

**CREATE TRIGGER FRIEND\_DELETE**

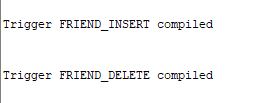
**AFTER DELETE ON friend**

**FOR EACH ROW**

**BEGIN**

**DELETE FROM friend WHERE id1=:old.id2 AND id2=:old.id1;**

**END;**



2 - Write a trigger that automatically deletes students when they graduate, i.e., when their grade is updated to exceed 12. In addition, write a trigger so when a student is moved ahead one grade, then so are all of his or her friends.

**QUERY:**

**CREATE TRIGGER graduate\_trigger**

**AFTER UPDATE on highschooler**

**FOR EACH ROW**

**when (new.grade > 12)**

**BEGIN**

**DELETE FROM highschooler WHERE id=:new.id;**

**END ;**

**CREATE TRIGGER moving\_ahead**

**AFTER UPDATE on highschooler**

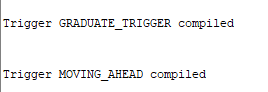
**for EACH ROW**

**WHEN (new.grade = old.grade + 1)**

**BEGIN**

**UPDATE highschooler SET grade = grade+1 WHERE id in (SELECT id2 FROM friend WHERE id1=:new.id);**

**END;**



3 - Write a trigger to enforce the following behavior: If A liked B but is updated to A liking C instead, and B and C were friends, make B and C no longer friends. Don't forget to delete the friendship in both directions, and make sure the trigger only runs when the "liked" (ID2) person is changed but the "liking" (ID1) person is not changed.

CREATE TRIGGER like\_Trigger

AFTER UPDATE ON likes

WHEN (SELECT \* FROM friend WHERE id1=new.id2 AND id2=old.id2

AND new.id1 = old.id1 AND NOT new.id2 = old.id2)

BEGIN

DELETE FROM friend where id1=:new.id2 AND :id2=old.id2);

DELETE FROM friend WHERE id1=:old.id2 AND :id2=new.id2);

END;