



Universidad Autónoma de Zacatecas

ACADEMIC UNIT OF ELECTRICAL ENGINEERING

Software Engineering Academic Program

Group: 5B - Semester: 2022-5^o

Practice Number: 06

Practice Name: DML

DATE: 22/SEPTEMBER/2022

Professor:

Aldonso Becerra Sánchez.

Student:

Cristian Omar Alvarado Rodríguez.

Índice

1. Introduction	3
2. Practice objective	4
3. Developing	4
4. Pre-assessment	57
5. Conclusion	57

DML

September 22th, 2022

1. Introduction

The DML (Data Modification Language) is one of the fundamental parts of the SQL language. It is formed by the instructions capable of modifying (add, change or delete) the data of the tables.

The set of DML statements that are executed consecutively is called a transaction. The interesting thing about transactions is that we can cancel them, since they form a logical unit of work that until they are accepted, their results will not be final.

In all DML statements, the only data returned by the system is the number of rows that have been modified by executing the statement.

The elements used to manipulate the data are the following:

- SELECT**, this statement is used to query the data.
- INSERT**, with this instruction we can insert the values in a database.
- UPDATE**, used to modify the values of one or more records.
- DELETE** is used to remove rows from a table.

2. Practice objective

Apply Oracle DML statements for several context situations.

3. Developing

Activity 1: Write the section that describes the **Work developed** in the following activities.

Read all the choices carefully because there might be more than one correct answer. Choose all the correct answers for each question. **Explain the reason for your answer.**

DESCRIBE EACH DATA MANIPULATION LANGUAGE (DML) STATEMENT

1. Which of the following commands can be rolled back?

R = DELETE, INSERT, UPDATE AND MERGE.

Explanation: This is because the insert, delete, update and merge statements are DML statements which can be undone if transaction is still open.

2. How can you change the primary key value of a row? (Choose the best answer.)

R = C) You cannot change the primary key value.

Explanation: This is because a value of a primary key is always unique.

3. If an UPDATE or DELETE command has a WHERE clause that gives it a scope of several rows, what will happen if there is an error part way through execution? The command is one of several in a multistatement transaction. (Choose the best answer.)

R = B) Whatever work the command had done before hitting the error will be rolled back, but work done already by the transaction will remain.

INSERT ROWS INTO A TABLE

4. If a table T1 has four numeric columns, C1, C2, C3, and C4, which of these statements will succeed? (Choose the best answer.)

R = A) insert into T1 values (1,2,3,null);

Explanation: In columns c1, c2 and c3 if the data (numbers: 1,2,3) were inserted, however in column c4 a null value would be inserted if the column allows it; that is, if it has a not null constraint.

5. Study the result of this SELECT statement:

SQL : select * from t1;

C1 C2 C3 C4

1 2 3 4

5 6 7 8

If you issue this statement:

insert into t1 (c1,c2) values(select c1,c2 from t1);

why will it fail? (Choose the best answer.)

R = A) Because the VALUES keyword is not used with a subquery.

Explanation: When a subquery is performed in an insert, the word values is not used.

6. Consider this statement:

insert into regions (region_id,region_name) values ((select max(region_id)+1 from regions), 'Spain');

What will the result be? (Choose the best answer.)

R = D) The statement has a syntax error because you cannot use the VALUES keyword with a subquery.

Explanation: When a subquery is performed in an insert, the word values is not used.

UPDATE ROWS IN A TABLE

7. You want to insert a row and then update it. What sequence of steps should you follow? (Choose the best answer.)

R = D) INSERT, UPDATE, COMMIT

Explanation: First the record is inserted, then the update is performed and finally the confirmation is performed (the transaction is closed).

8. If you issue this command:

`update employees set salary=salary * 1.3;`

what will be the result? (Choose the best answer.)

R = A) Every row will have SALARY incremented by 30 percent, unless SALARY was NULL.

DELETE ROWS FROM A TABLE

9. How can you delete the values from one column of every row in a table? (Choose the best answer.)

R = C) Use the DELETE COLUMN command.

D) Use the TRUNCATE COLUMN command.

Explanation: Both the DELETE statement and the TRUNCATE statement delete a column from the table, the difference is that DELETE can be reversed and the TRUNCATE statement cannot because it is a DDL statement.

10. Which of these commands will remove every row in a table? (Choose one or more

correct answers.)

R = B) A TRUNCATE command.

D) A DELETE command with no WHERE clause.

Explanation: The TRUNCATE statement removes all data from the table and the DELETE statement without a where clause also removes all data from a table.

CONTROL TRANSACTIONS

11. User JOHN updates some rows and asks user MICHAEL to log in and check the changes before he commits them. Which of the following statements is true? (Choose the best answer.)

R = B) MICHAEL will not be able to see the changes.

Explanation: MICHAEL will not be able to see the changes because JOHN has an open transaction so MICHAEL will only be able to see the changes until JOHN commits (closes the transaction).

12. User JOHN updates some rows but does not commit the changes. User MICHAEL queries the rows that JOHN updated. Which of the following statements is true? (Choose the best answer.)

R = C) MICHAEL will see the old versions of the rows.

Explanation: MICHAEL will only be able to see the rows as they were originally because JOHN has an open transaction and has not committed the changes.

13. Which of these commands will terminate a transaction? (Choose three correct answers.)

R = TRUNCATE, ROLLBACK AND COMMIT.

Explanation: The TRUNCATE statement closes a transaction because the TRUNCATE statement is a DML statement and the COMMIT and ROLLBACK statements will also close a transaction.

Activity 2: Insert a dataset for the scheme SALES carried out in practice 4, activity 2. Take into account the following instructions:

- Use sequences as possible for primary key values (using pseudo-columns CURRVAL and NEXTVAL).
- Insert a representative set of values for each table.
- Use SYSDATE as possible.

Figure 1 shows the insertion of records in the PRODUCT table

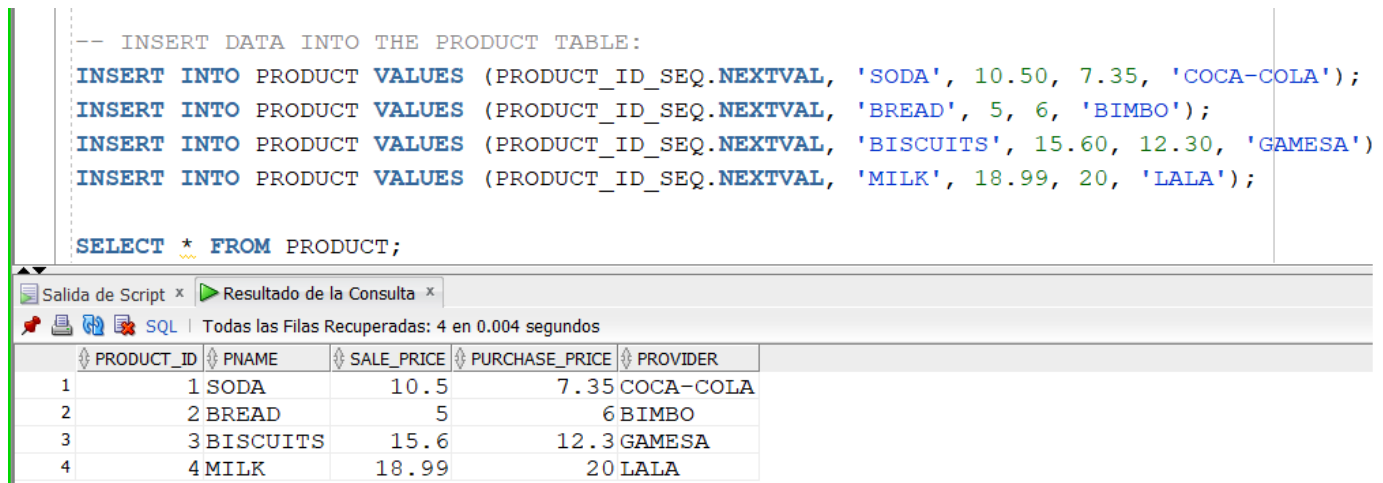


Figura 1: *Inserting data into the PRODUCT table.*

In figure 2 you can see the insertion of some records in the CHANNEL table.


```
-- INSERT DATA INTO THE CHANNEL TABLE:
INSERT INTO CHANNEL VALUES (CHANNEL_ID_SEQ.NEXTVAL, 'INTERNET');
INSERT INTO CHANNEL VALUES (CHANNEL_ID_SEQ.NEXTVAL, 'IN SHOP');
INSERT INTO CHANNEL VALUES (CHANNEL_ID_SEQ.NEXTVAL, 'TELEPHONE');

SELECT * FROM CHANNEL;
```

Salida de Script x Resultado de la Consulta x

SQL | Todas las Filas Recuperadas: 3 en 0.011 segundos

	CHANNEL_ID	CNAME
1	100	INTERNET
2	105	IN SHOP
3	110	TELEPHONE

Figura 2: Inserting data into the CHANNEL table.

Figure 3 shows the insertion of records in the EMPLOYEE table.

```
-- INSERT DATA INTO THE EMPLOYEE TABLE:
INSERT INTO EMPLOYEE VALUES (EMPLOYEE_ID_SEQ.NEXTVAL, 'JUAN', 'RAMIREZ', NULL,
                              '20/09/1978', 'MALE', 'MARIA RAMIREZ');

INSERT INTO EMPLOYEE VALUES (EMPLOYEE_ID_SEQ.NEXTVAL, 'ANA', 'GARCIA', 100,
                              '12/02/1989', 'FEMALE', 'MARIO GARCIA');

INSERT INTO EMPLOYEE VALUES (EMPLOYEE_ID_SEQ.NEXTVAL, 'OMAR', 'ALVARADO', 100,
                              '16/12/2002', 'MALE', 'PEDRO ALVARADO');

INSERT INTO EMPLOYEE VALUES (EMPLOYEE_ID_SEQ.NEXTVAL, 'MARIA', 'TORRES', 101,
                              '05/06/2000', 'FEMALE', 'NANCY HERNANDEZ');

INSERT INTO EMPLOYEE VALUES (EMPLOYEE_ID_SEQ.NEXTVAL, 'CARLOS', 'OLVERA', 100,
                              '08/11/1990', 'MALE', 'ALEX OLVERA');

SELECT * FROM EMPLOYEE;

DELETE FROM EMPLOYEE WHERE EMP_ID = 102;
```

Salida de Script x Resultado de la Consulta x

SQL | Todas las Filas Recuperadas: 5 en 0.005 segundos

	EMP_ID	EMP_NAME	EMP_LASTN	BOOS_ID	DATE_OF_BIRTH	GENDER	BENEFICIARIES
1	100	JUAN	RAMIREZ	(null)	20/09/78	MALE	MARIA RAMIREZ
2	101	ANA	GARCIA	100	12/02/89	FEMALE	MARIO GARCIA
3	103	OMAR	ALVARADO	100	16/12/02	MALE	PEDRO ALVARADO
4	104	MARIA	TORRES	101	05/06/00	FEMALE	NANCY HERNANDEZ
5	105	CARLOS	OLVERA	100	08/11/90	MALE	ALEX OLVERA

Figura 3: Inserting data into the EMPLOYEE table.

Figure 4 shows the insertion of records in the SHOP table.

```
-- INSERT DATA INTO THE SHOP TABLE:
INSERT INTO SHOP VALUES (SHOP_ID_SEQ.NEXTVAL, 'ZACATECAS NUM 102', 100);
INSERT INTO SHOP VALUES (SHOP_ID_SEQ.NEXTVAL, 'FRESNILLO, ZAC NUM 02', 103);
INSERT INTO SHOP VALUES (SHOP_ID_SEQ.NEXTVAL, 'ZACATECAS NUM 34', 101);
INSERT INTO SHOP VALUES (SHOP_ID_SEQ.NEXTVAL, 'JEREZ, ZAC NUM 07', 104);
INSERT INTO SHOP VALUES (SHOP_ID_SEQ.NEXTVAL, 'TEPITO, CDMX NUM 90', 105);

SELECT * FROM SHOP;
```

SHOP_ID	ADDRESS	MANGER
1	10 ZACATECAS NUM 102	100
2	20 FRESNILLO, ZAC NUM 02	103
3	30 ZACATECAS NUM 34	101
4	50 JEREZ, ZAC NUM 07	104
5	60 TEPITO, CDMX NUM 90	105

Figura 4: Inserting data into the SHOP table.

Figure 5 shows the insertion of records in the SALES table.

```
-- INSERT DATA INTO THE SALE TABLE:
INSERT INTO SALES VALUES (SALE_ID_SEQ.NEXTVAL, 100, 3, 30, 5, 101, SYSDATE);
INSERT INTO SALES VALUES (SALE_ID_SEQ.NEXTVAL, 105, 2, 50, 5, NULL, SYSDATE);
INSERT INTO SALES VALUES (SALE_ID_SEQ.NEXTVAL, 105, 1, 30, 5, 100, SYSDATE);
INSERT INTO SALES VALUES (SALE_ID_SEQ.NEXTVAL, 100, 4, 60, 5, 101, SYSDATE);
INSERT INTO SALES VALUES (SALE_ID_SEQ.NEXTVAL, 110, 1, 20, 5, 105, SYSDATE);

SELECT * FROM SALES;
```

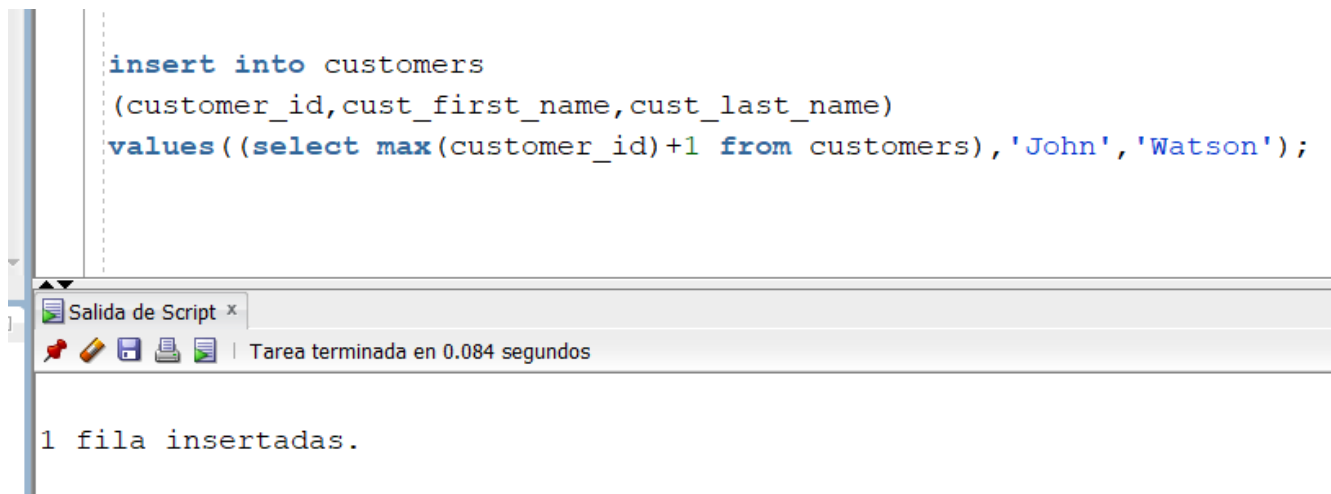
SALE_ID	CHANNEL_ID	PRODUCT_ID	SHOP_ID	QUANTITY	EMP_ID	SALE_DATE
1	4	100	3	30	5	101 21/09/22
2	5	105	2	50	5 (null)	21/09/22
3	6	105	1	30	5	100 21/09/22
4	7	100	4	60	5	101 21/09/22
5	8	110	1	20	5	105 21/09/22

Figura 5: Inserting data into the SALES table.

Activity 3: Carry out this exercise in the OE schema (copy and paste screen results). Analyze the results for each sentence.

1. Insert a customer into CUSTOMERS, using a function to generate a unique customer number

What the statement shown in **figure 6** does is that it inserts a record in the CUSTOMERS table. In this sentence a subquery is used to obtain the value of the primary key of the record; what this subquery does is obtain the largest id (the id of the last record) and add one to it, the value returned by that subquery will be the id of the new record to insert.



```

insert into customers
(customer_id,cust_first_name,cust_last_name)
values((select max(customer_id)+1 from customers),'John','Watson');
  
```

Salida de Script x

Tarea terminada en 0.084 segundos

1 fila insertadas.

Figura 6: Insertion of a record in the CUSTOMERS table.

314	841	Ali	Boyer	1000	d	SWITZER...	1400	Ali.Boyer@WILLET.COM
315	844	Alice	Julius	1000	d	SWITZER...	700	Alice.Julius@BITTERN.COM
316	847	Ally	Streep	1000	d	SWITZER...	5000	Ally.Streep@PIPIT.COM
317	927	Bryan	Belushi	1000	hi	INDIA	2300	Bryan.Belushi@TOWHEE.COM
318	931	Buster	Edwards	1000	hi	INDIA	900	Buster.Edwards@KINGLET.COM
319	981	Daniel	Gueney	1000	zhs	CHINA	200	Daniel.Gueney@REDPOLL.COM
320	982	John	Watson	(null)	(null)	(null)	(null)	(null)

Figura 7: Insertion of a record in the CUSTOMERS table.

2. Give him a credit limit equal to the average credit limit:

In the statement shown in **Figure 8**, it can be seen how the Watson customer's credit_limit is updated, the update is performed through a sub-query which obtains the average of the credit_limit of all the customers.

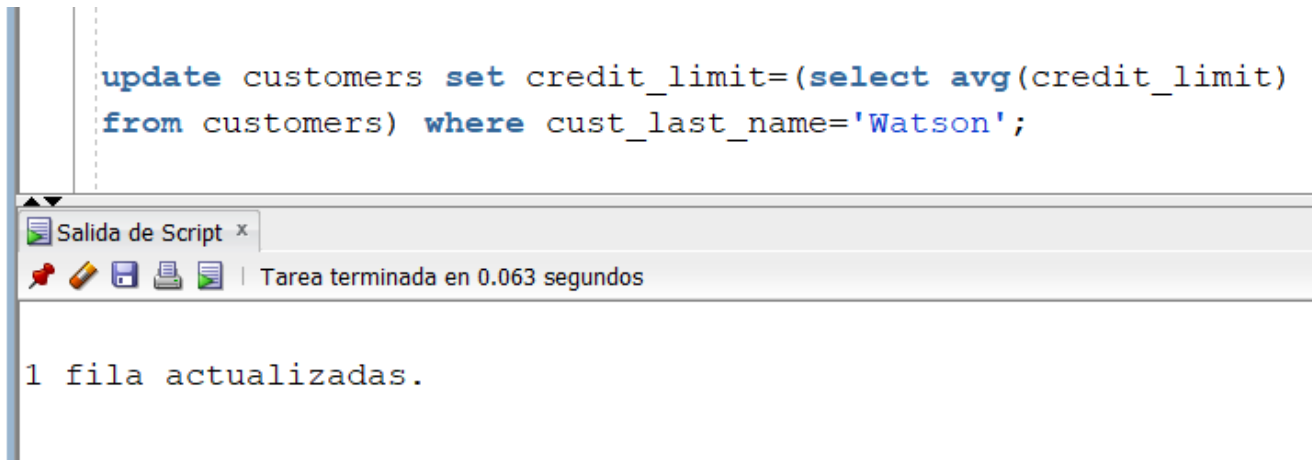
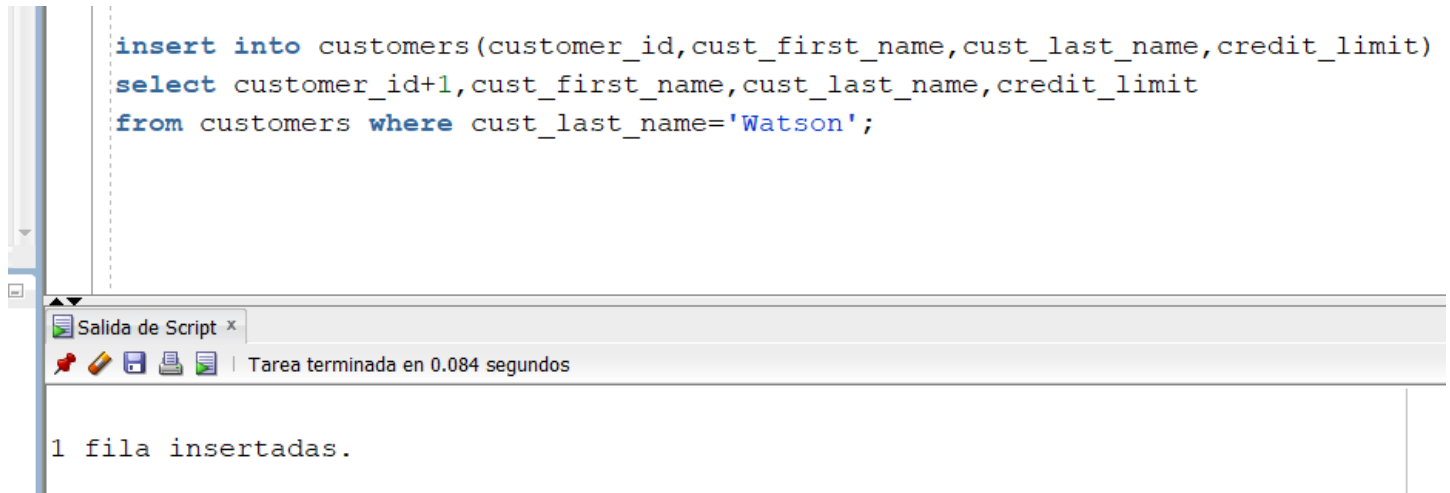


Figura 8: Updating a record in the CUSTOMER table.

316	847	Ally	Streep	1000	d	SWITZER...	5000	Ally.Sti
317	927	Bryan	Belushi	1000	hi	INDIA	2300	Bryan.Be
318	931	Buster	Edwards	1000	hi	INDIA	900	Buster.F
319	981	Daniel	Gueney	1000	zhs	CHINA	200	Daniel.C
320	982	John	Watson	(null)	(null)	(null)	1894.67	(null)

Figura 9: Updating a record in the CUSTOMER table.

3. Create another customer using the customer just created, but make sure the CUSTOMER_ID is unique; See figure 10 and 11.



```
insert into customers(customer_id,cust_first_name,cust_last_name,credit_limit)
select customer_id+1,cust_first_name,cust_last_name,credit_limit
from customers where cust_last_name='Watson';
```

Salida de Script x

Tarea terminada en 0.084 segundos

1 fila insertadas.

Figura 10: *Insertion of a record in the CUSTOMERS table.*

931	Buster	Edwards	1000	hi	INDIA	900	Buster.Edwards@KINGLET.COM
981	Daniel	Gueney	1000	zhs	CHINA	200	Daniel.Gueney@REDPOLL.COM
982	John	Watson	(null)	(null)	(null)	1894.67	(null)
983	John	Watson	(null)	(null)	(null)	1894.67	(null)

Figura 11: *Insertion of a record in the CUSTOMERS table.*

4. Change the name of the second entered customer: See **figures 12 and 13**.

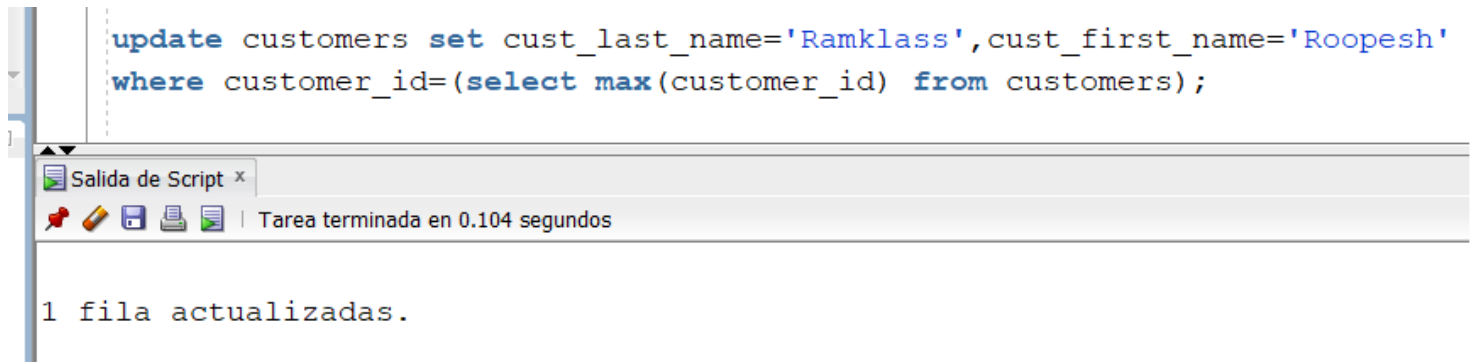


Figura 12: *Update of a record of the CUSTOMERS table.*

319	981 Daniel	Gueney	1000 zhs
320	982 John	Watson	(null) (null)
321	983 Roopesh	Ramklass	(null) (null)

Figura 13: *Update of a record of the CUSTOMERS table.*

5. Commit this transaction:

With the COMMIT statement we close the transaction that we had open.

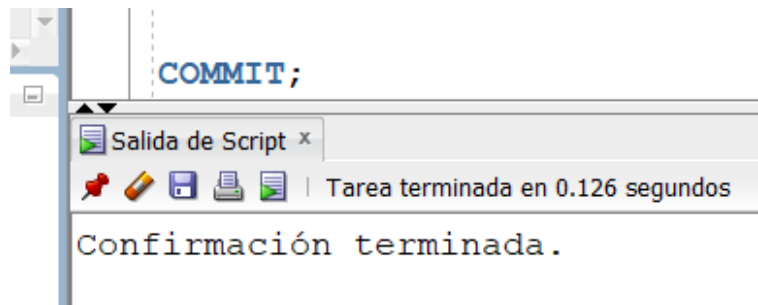


Figura 14: *Confirm the changes.*

6. Determine the CUSTOMER.IDs of the two new customers and lock the rows: See **figure 15**.

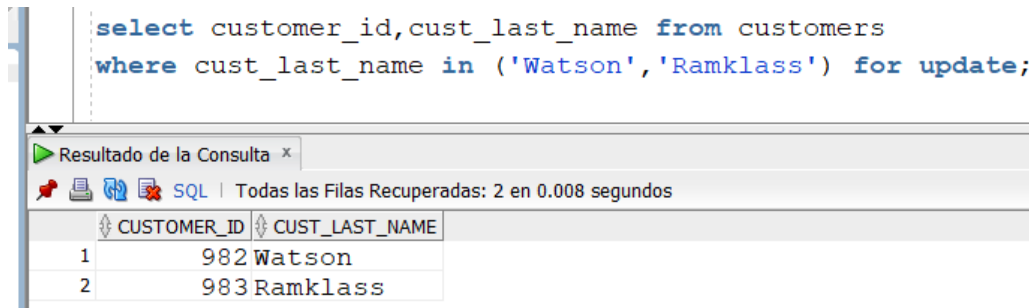


Figura 15: *Query and data block.*

7. From another session connected to the OE schema, attempt to select the locked rows, **What happened?**

The data is consulted normally from another session of the OE schema, only that the data cannot be modified because the data is locked in the other session.

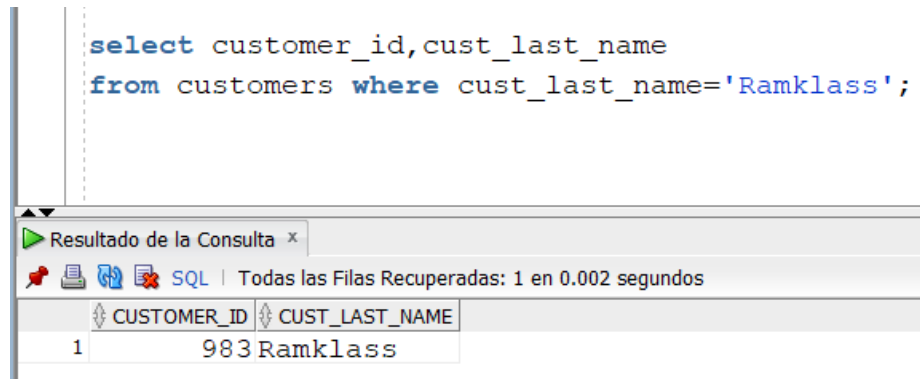


Figura 16: *Query data from another session.*

8. In this second session connected to the OE schema, attempt to update one of the locked rows:

The record cannot be updated (the data is locked) the execution of the query waits for the transaction to close in the other session, See **figure 17**.

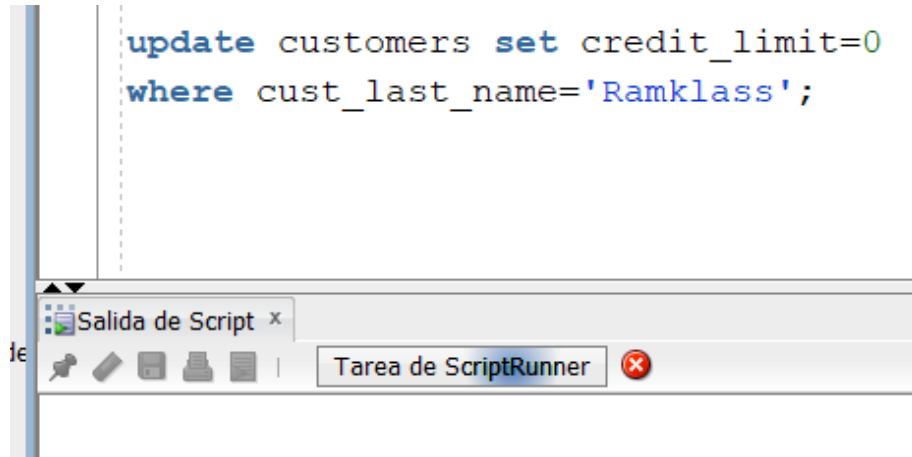


Figura 17: *Updating data from another session.*

9. This command will hang. In the first session, release the locks by issuing a commit:

In the first session the transaction is closed and in the second session the waiting record is updated, since the lock is released.

Session 1:

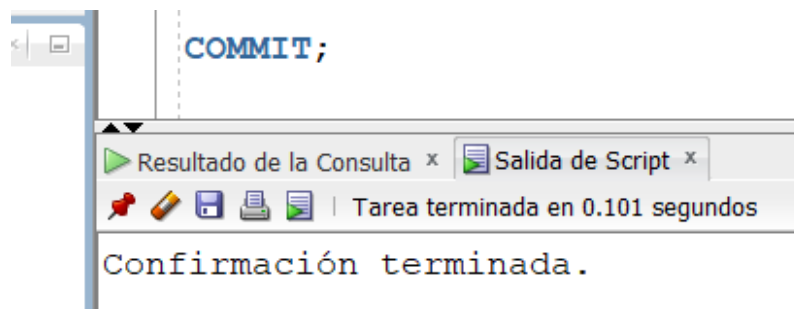
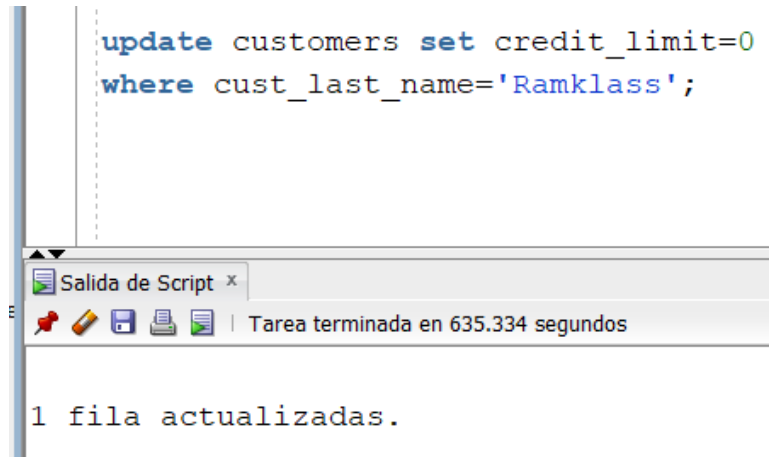


Figura 18: *Confirmation of changes.*

Session 2:



```
update customers set credit_limit=0
where cust_last_name='Ramklass';
```

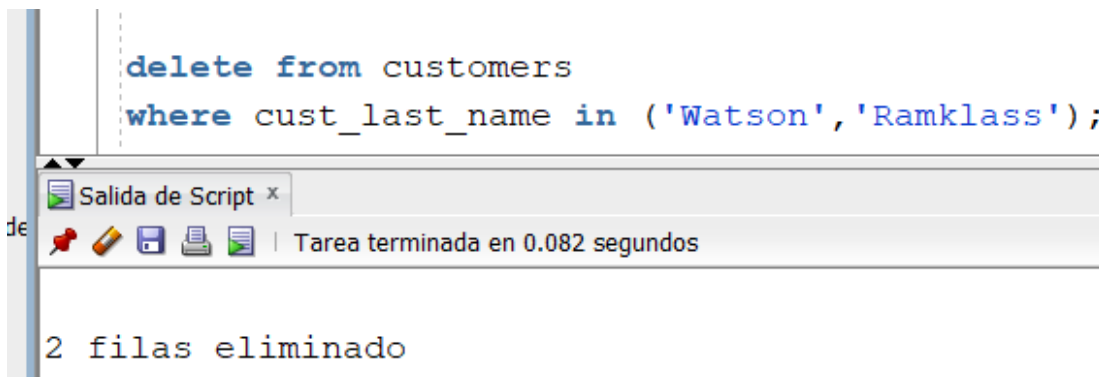
Salida de Script x

Tarea terminada en 635.334 segundos

1 fila actualizadas.

Figura 19: *Updating a record.*

10. The second session will now complete its update. In the second session, delete the two rows: See figure 20.



```
delete from customers
where cust_last_name in ('Watson','Ramklass');
```

Salida de Script x

Tarea terminada en 0.082 segundos

2 filas eliminado

Figura 20: *Deletion of records.*

11. In the first session, attempt to truncate the CUSTOMERS table:

The table cannot be deleted because the table is in use in another session that session has an open transaction. See **figure 21**.

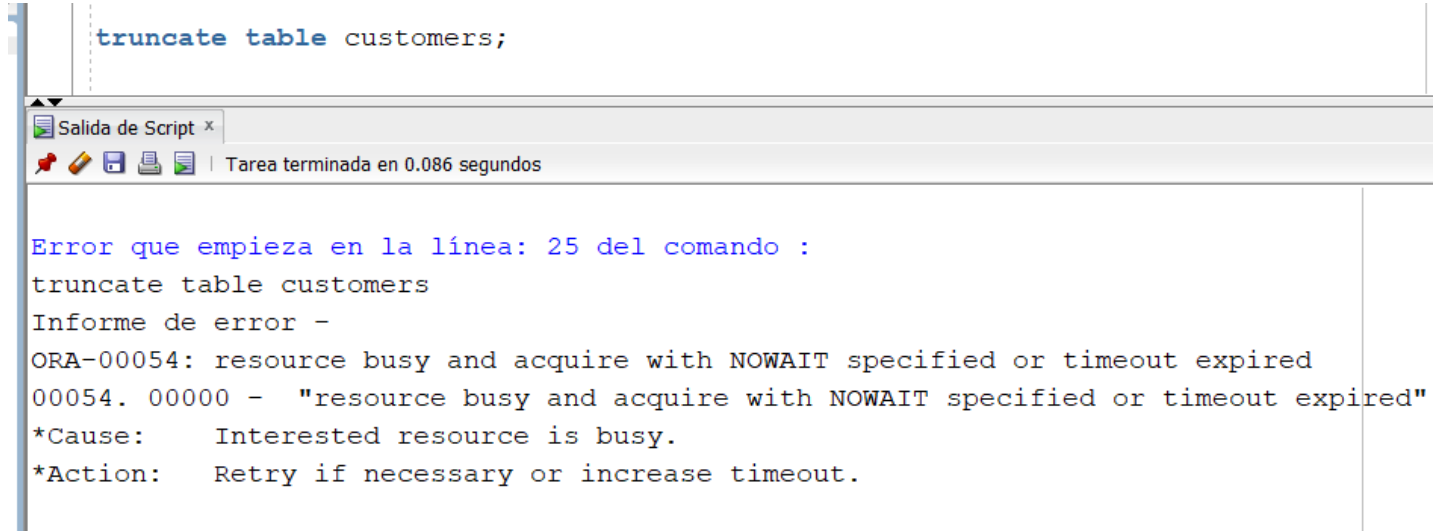


Figura 21: *Attempt to delete table CUSTOMERS.*

12. This will fail because there is a transaction in progress against the table, which will block all DDL commands. In the second session, commit the transaction:

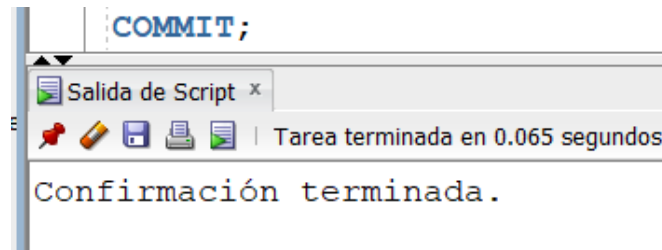


Figura 22: *Confirmation of changes.*

13. The CUSTOMERS table will now be back in the state it was in at the start of the exercise. Confirm this by checking the value of the highest CUSTOMER_ID: see **figure 23**.

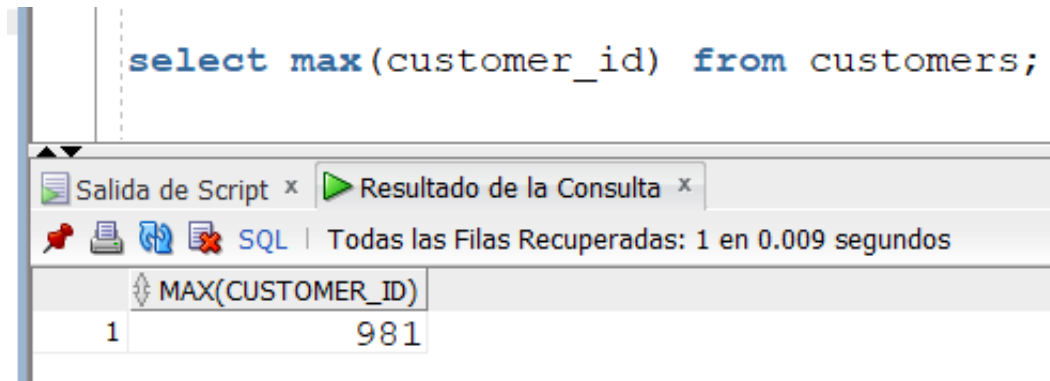
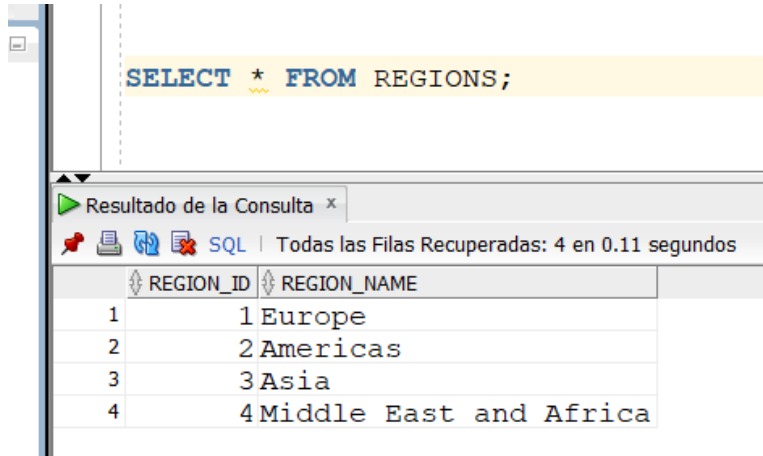


Figura 23: *Query the largest id.*

Activity 4: Copy and paste screen results. Analyze the results for each sentence.

In this section, use various techniques to insert rows into a table. Follow the next instructions:

1. Connect to the HR schema.
2. Query the REGIONS table, to check what values are already in use for the REGION_ID column: This exercise assumes that values above 100 are not in use. If they are, adjust the values suggested below to avoid primary key conflicts. See **figure 24**.



```
SELECT * FROM REGIONS;
```

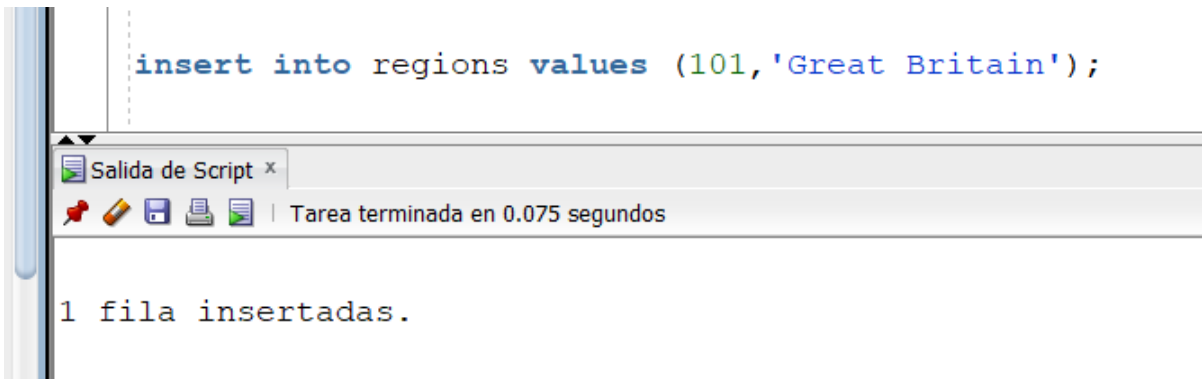
Resultado de la Consulta x

SQL | Todas las Filas Recuperadas: 4 en 0.11 segundos

REGION_ID	REGION_NAME
1	1 Europe
2	2 Americas
3	3 Asia
4	4 Middle East and Africa

Figura 24: *Data query to the REGIONS table.*

3. Insert a row into the REGIONS table, providing the values in line, See **figure 25**:



```
insert into regions values (101, 'Great Britain');
```

Salida de Script x

Tarea terminada en 0.075 segundos

1 fila insertadas.

Figura 25: *Insertion of a record in the REGIONS table.*

4. Insertion of a record in the REGIONS table. When prompted, give the values 102 for the number, Australasia for the name. Note the use of quotes around the string.

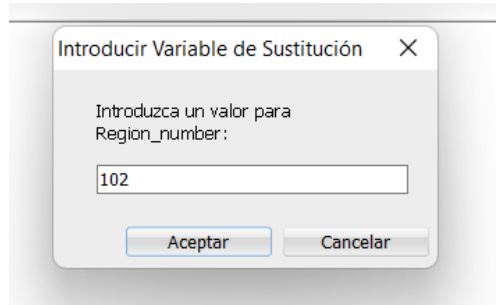


Figura 26: *Insertion of a record in the REGIONS table.*

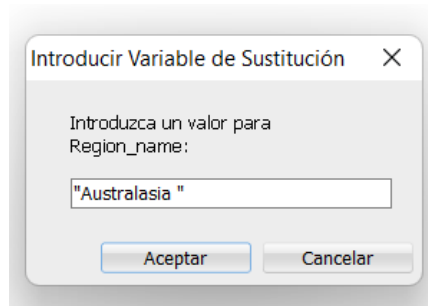


Figura 27: *Insertion of a record in the REGIONS table.*

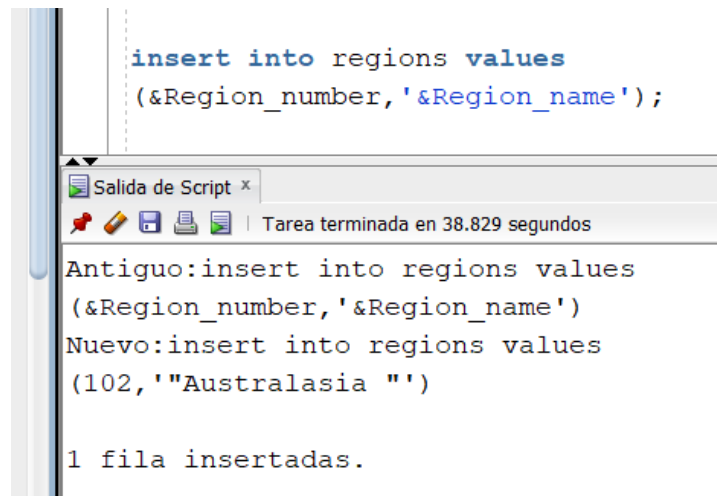


Figura 28: *Insertion of a record in the REGIONS table.*

5. Insert a row into the REGIONS table, calculating the REGION_ID to be one higher than the current high value. This will need a scalar subquery:

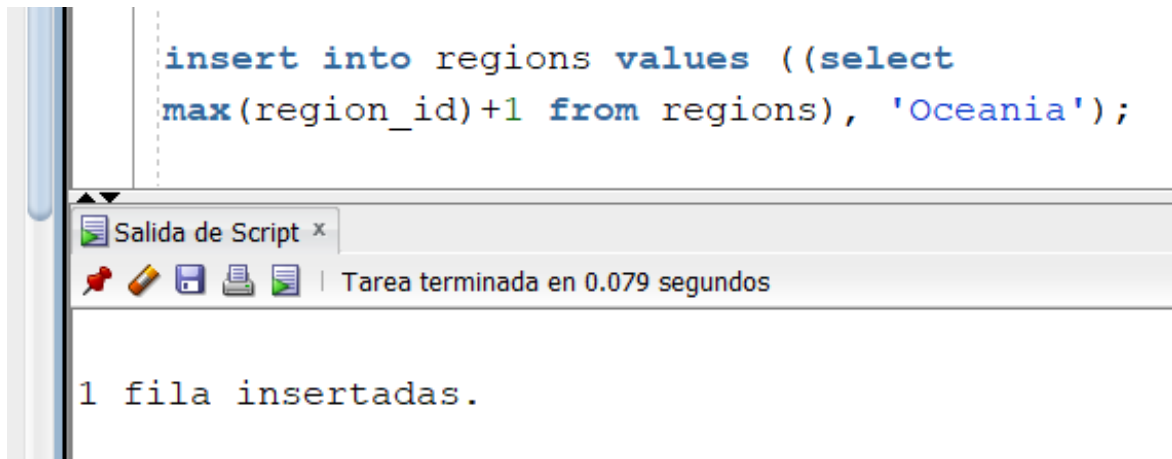


Figura 29: Insertion of a record in the REGIONS table.

6. Confirm the insertion of the rows:



Figura 30: Query of data from the REGIONS table.

7. Commit the insertions:

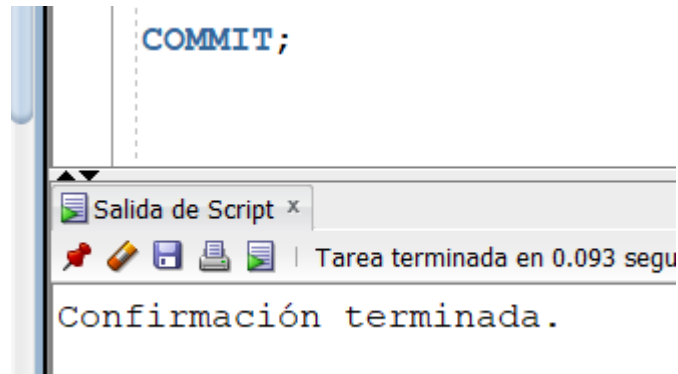


Figura 31: *Confirmation of changes.*

Activity 5: Copy and paste screen results. Analyze the results for each sentence.

In this section, use various techniques to update rows in a table. Follow the next instructions:

1. Connect to the HR schema.
2. Update a single row, identified by primary key: This statement should return the message “1 row updated

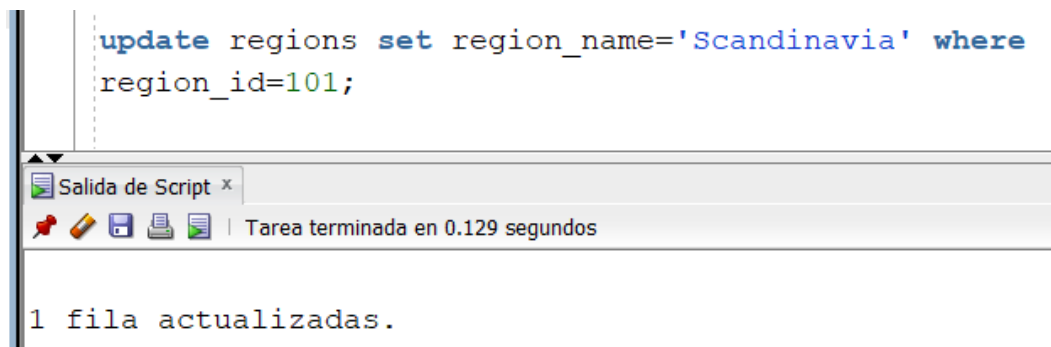
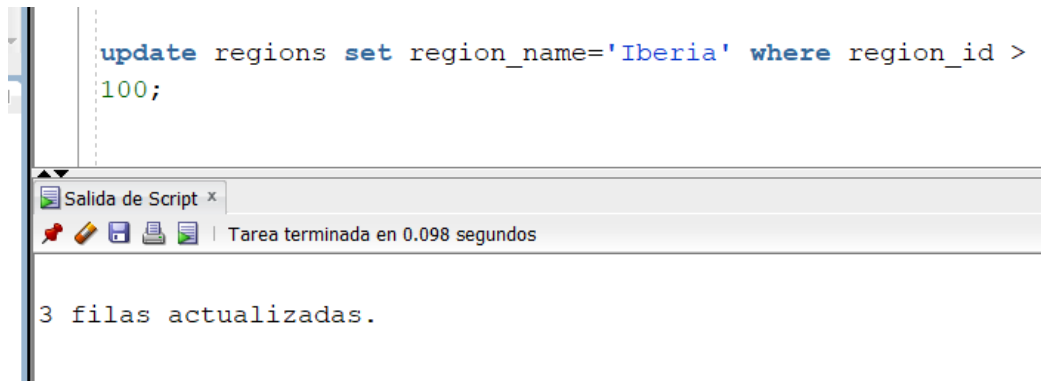


Figura 32: *Updating a record in the REGIONS table.*

3. Update a set of rows, using a nonequality predicate. The statement in **figure 33** should return the message "3 rows updated".



```
update regions set region_name='Iberia' where region_id > 100;
```

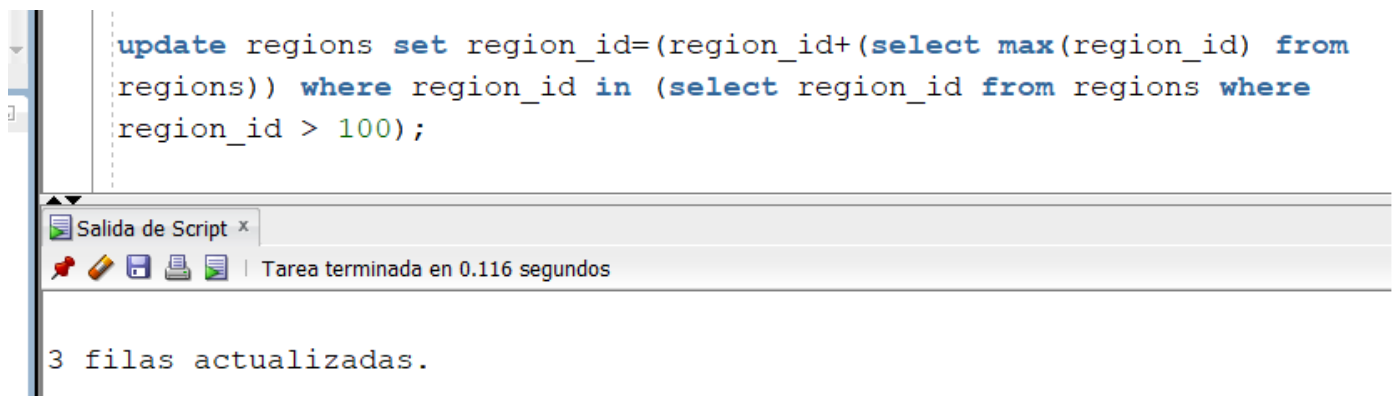
Salida de Script x

Tarea terminada en 0.098 segundos

3 filas actualizadas.

Figura 33: *Updating a set of records from the REGIONS table.*

4. Update a set of rows, using subqueries to select the rows and to provide values. The statement in **figure 34** should return the message "3 rows updated".



```
update regions set region_id=(region_id+(select max(region_id) from regions)) where region_id in (select region_id from regions where region_id > 100);
```

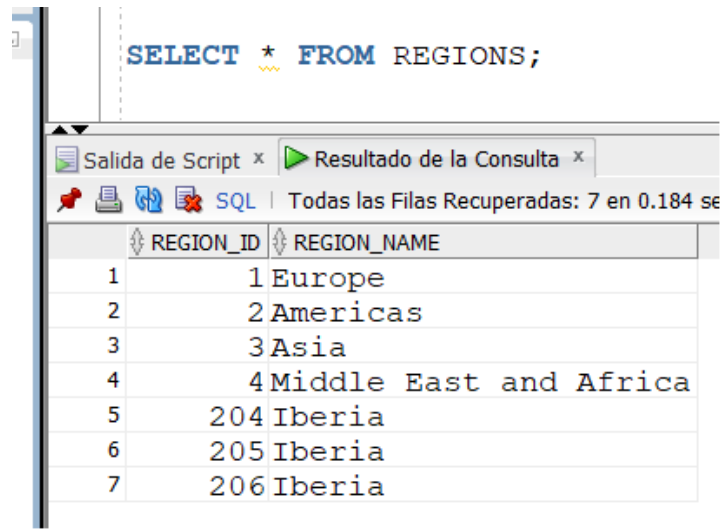
Salida de Script x

Tarea terminada en 0.116 segundos

3 filas actualizadas.

Figura 34: *Updating a set of records from the REGIONS table.*

5. Confirm the state of the rows: see **figure 35**.



The screenshot shows the SQL Developer interface. The query editor contains the SQL statement `SELECT * FROM REGIONS;`. Below the editor, the 'Resultado de la Consulta' (Query Result) tab is active, displaying the results of the query. The status bar indicates 'Todas las Filas Recuperadas: 7 en 0.184 se' (All rows recovered: 7 in 0.184 seconds). The results are shown in a table with two columns: REGION_ID and REGION_NAME.

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa
204	Iberia
205	Iberia
206	Iberia

Figura 35: Consulting the registers of *REGIONS*.

6. Commit the changes made: see **figure 36**.

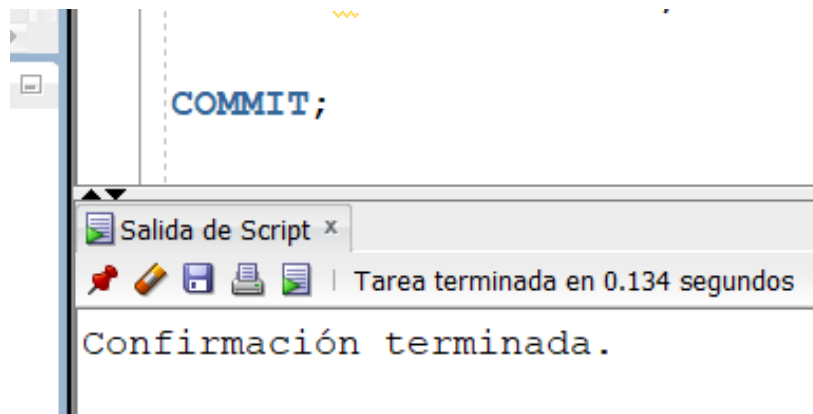


Figura 36: Confirm the changes.

Activity 6: Copy and paste screen results. Analyze the results for each sentence.

In this section, use various techniques to delete rows in a table. Follow the next instructions:

If not, adjust the values as necessary.

1. Connect to the HR schema using SQL Developer.
2. Remove one row, using the equality predicate on the primary key: This should return the message “1 row deleted.”

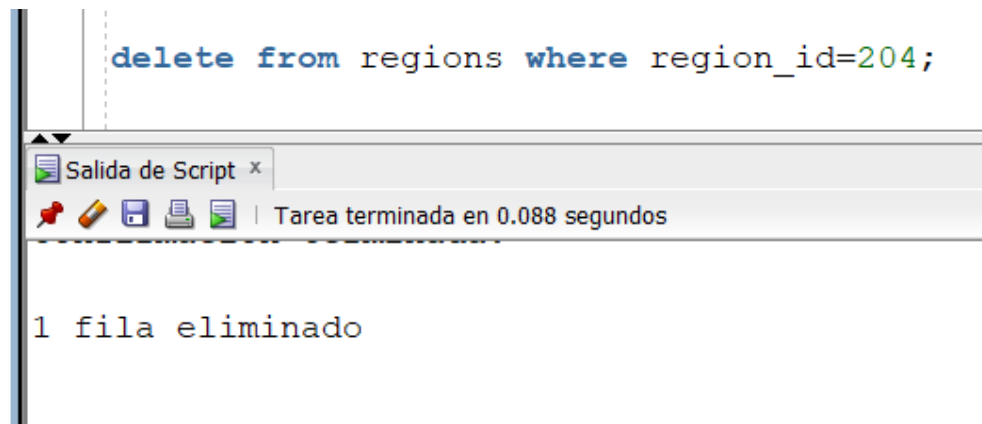


Figura 37: *Deleting a record.*

3. Attempt to remove every row in the table by omitting a WHERE clause: This will fail, due to a constraint violation, see **figure 38**.

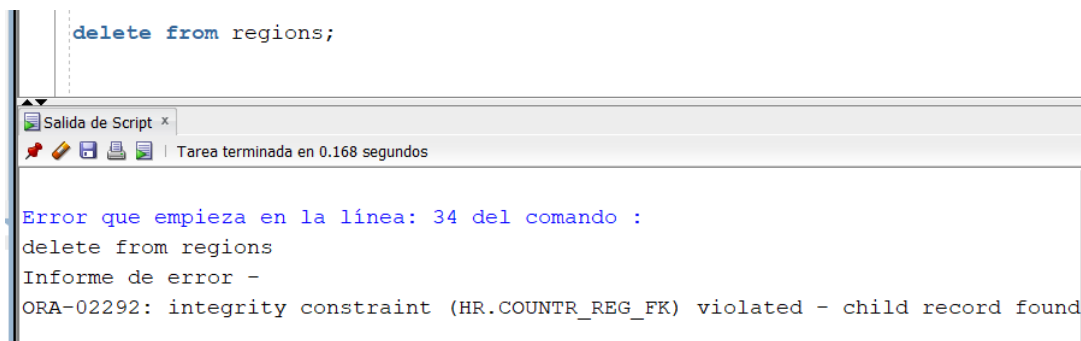


Figura 38: *Deletion of records.*

4. Remove rows with the row selection based on a subquery, This will return the message “2 rows deleted.” see **figure 39**.

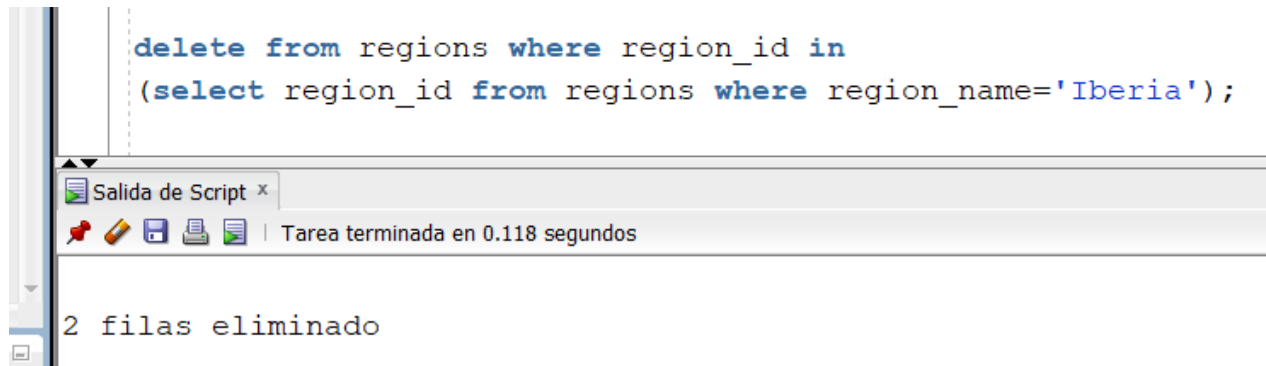


Figura 39: *Elimination of records from the REGIONS table.*

5. Confirm that the REGIONS table now contains just the original four rows: see **figure 40**.

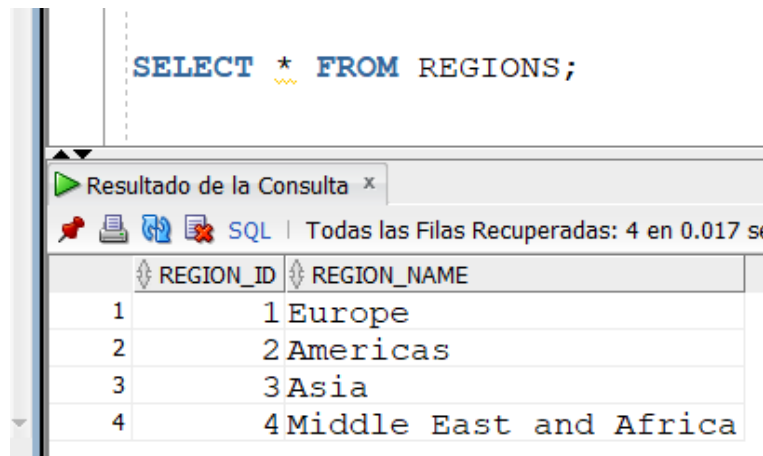
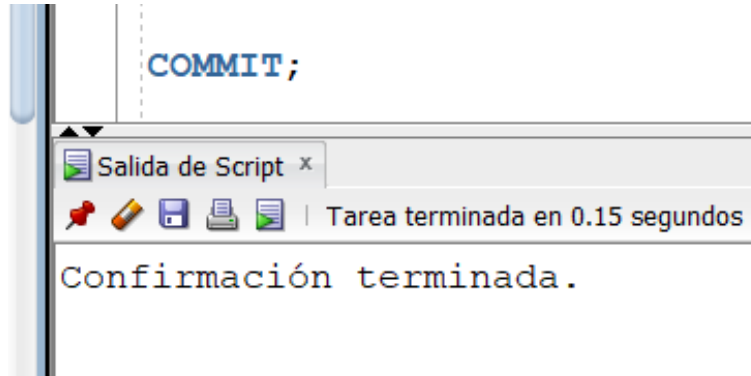


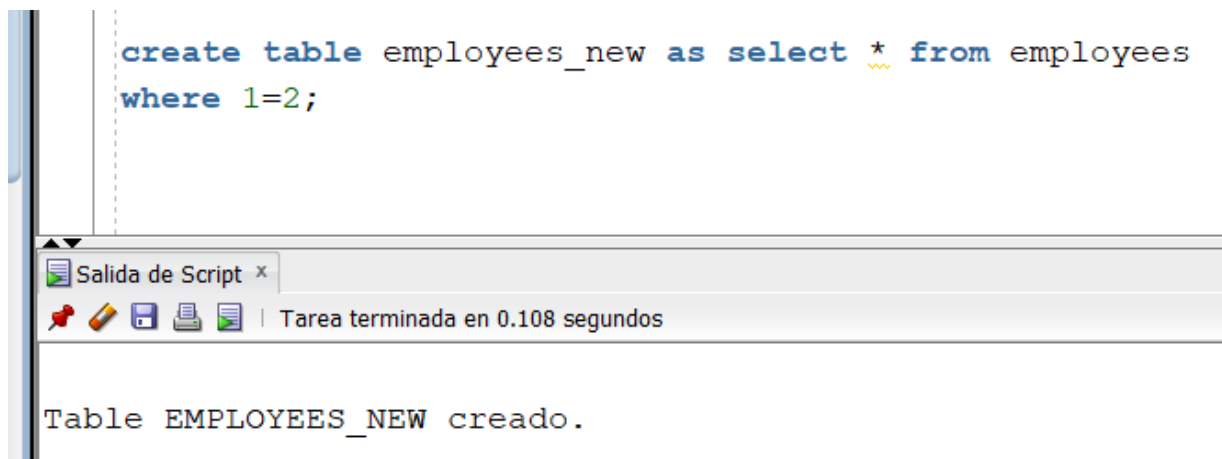
Figura 40: *Querying data from the REGIONS table.*

6. Commit the deletions: see **figure 41**, (The transaction is closed).

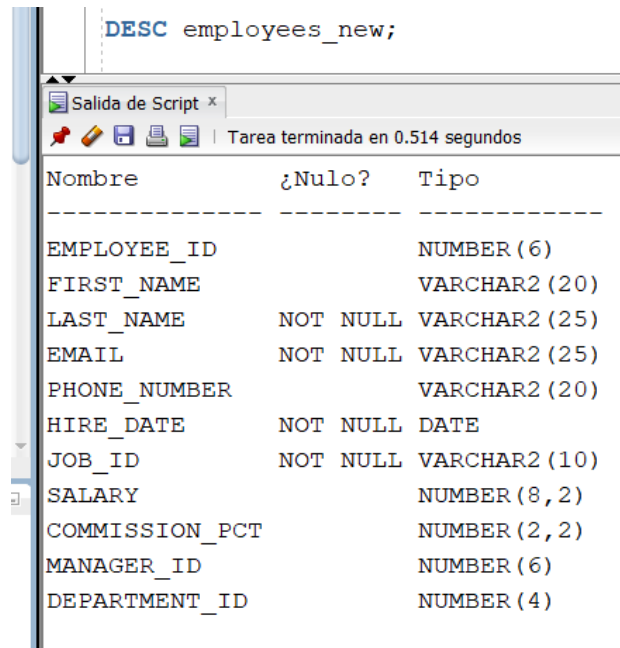
Figura 41: *Confirmation of changes.*

Activity 7: The MERGE command is often ignored, because it does nothing that cannot be done with INSERT, UPDATE, and DELETE. It is, however, very powerful, in that with one pass through the data it can carry out all three operations. This can improve performance dramatically. Here is a simple example, do not forget to include and analyze the results:

1. Create a new table employees2:

Figura 42: *Creation of the employees new table.*

2. Describe the new table: see **figure 43**,

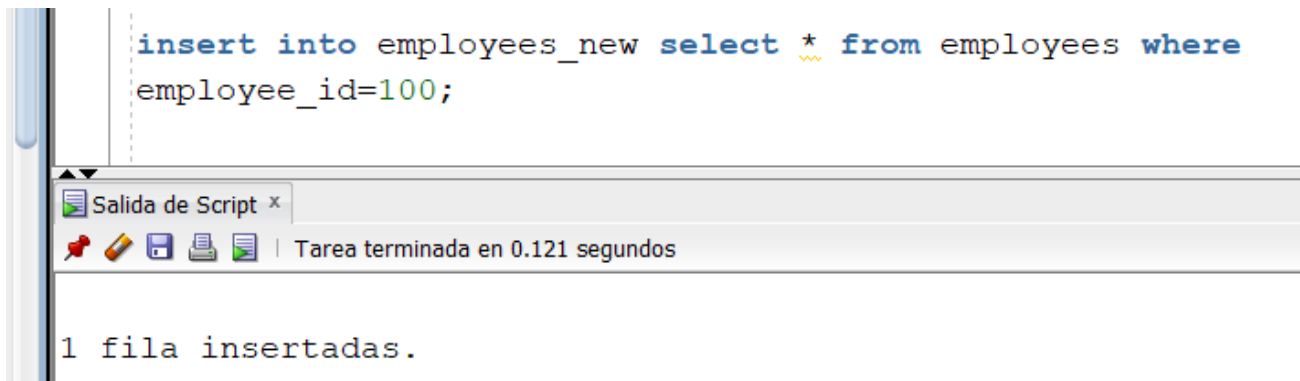


```
DESC employees_new;
```

Nombre	¿Nulo?	Tipo
EMPLOYEE_ID		NUMBER (6)
FIRST_NAME		VARCHAR2 (20)
LAST_NAME	NOT NULL	VARCHAR2 (25)
EMAIL	NOT NULL	VARCHAR2 (25)
PHONE_NUMBER		VARCHAR2 (20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2 (10)
SALARY		NUMBER (8,2)
COMMISSION_PCT		NUMBER (2,2)
MANAGER_ID		NUMBER (6)
DEPARTMENT_ID		NUMBER (4)

Figura 43: *Structure of the employees_newtable.*

3. Insert a new row into table employees2: see **figure 44**.



```
insert into employees_new select * from employees where  
employee_id=100;
```

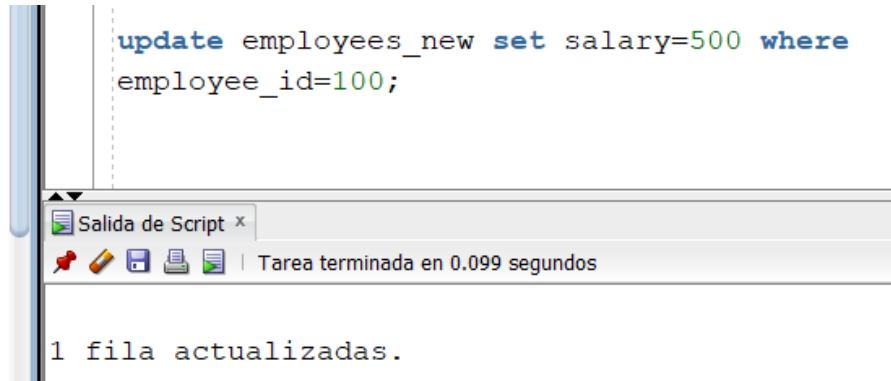
Salida de Script x

Tarea terminada en 0.121 segundos

1 fila insertadas.

Figura 44: *Insertion of a record.*

4. Update the salary of the inserted row: see **figure 45**.



```
update employees_new set salary=500 where
employee_id=100;
```

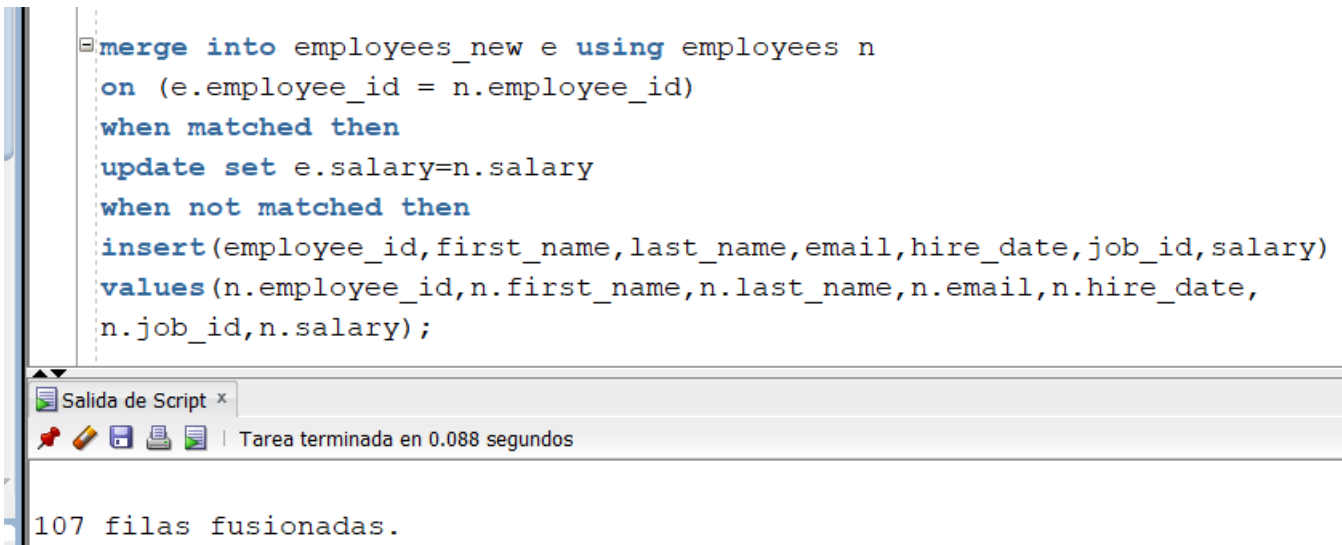
Salida de Script x

Tarea terminada en 0.099 segundos

1 fila actualizadas.

Figura 45: *Updating a record in the employees_newtable.*

5. Execute the merge operation: see **figure 46**.



```
merge into employees_new e using employees n
on (e.employee_id = n.employee_id)
when matched then
update set e.salary=n.salary
when not matched then
insert(employee_id,first_name,last_name,email,hire_date,job_id,salary)
values(n.employee_id,n.first_name,n.last_name,n.email,n.hire_date,
n.job_id,n.salary);
```

Salida de Script x

Tarea terminada en 0.088 segundos

107 filas fusionadas.

Figura 46: *Applying a MARGE operation.*

6. View the final data in employees2: see **figure 47**.

```
select * from employees_new;
```

Resultado de la Consulta x

Todas las Filas Recuperadas: 107 en 0.029 segundos

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING	515.123.4567	17/06/03	AD PRES	24000	(null)	(null)	90
2	101	Neena	Kochhar	NKOCHHAR	(null)	21/09/05	AD VP	17000	(null)	(null)	(null)
3	102	Lex	De Haan	LDEHAAN	(null)	13/01/01	AD VP	17000	(null)	(null)	(null)
4	103	Alexander	Hunold	AHUNOLD	(null)	03/01/06	IT PROG	9000	(null)	(null)	(null)
5	104	Bruce	Ernst	BERNST	(null)	21/05/07	IT PROG	6000	(null)	(null)	(null)
6	105	David	Austin	DAUSTIN	(null)	25/06/05	IT PROG	4800	(null)	(null)	(null)
7	106	Valli	Pataballa	VPATABAL	(null)	05/02/06	IT PROG	4800	(null)	(null)	(null)
8	107	Diana	Lorentz	DLORENTZ	(null)	07/02/07	IT PROG	4200	(null)	(null)	(null)
9	108	Nancy	Greenberg	NGREENBE	(null)	17/08/02	FI MGR	12008	(null)	(null)	(null)
10	109	Daniel	Faviet	DFAVIET	(null)	16/08/02	FI ACCOUNT	9000	(null)	(null)	(null)
11	110	John	Chen	JCHEN	(null)	28/09/05	FI ACCOUNT	8200	(null)	(null)	(null)
12	111	Ismael	Sciarra	ISCIARRA	(null)	30/09/05	FI ACCOUNT	7700	(null)	(null)	(null)
13	112	Jose Manuel	Urman	JMURMAN	(null)	07/03/06	FI ACCOUNT	7800	(null)	(null)	(null)
14	113	Luis	Popp	LPOPP	(null)	07/12/07	FI ACCOUNT	6900	(null)	(null)	(null)
15	114	Den	Raphaely	DRAPHEAL	(null)	07/12/02	PU MAN	11000	(null)	(null)	(null)
16	115	Alexander	Khoo	AKHOO	(null)	18/05/03	PU CLERK	3100	(null)	(null)	(null)
17	116	Shelli	Baida	SBAIDA	(null)	24/12/05	PU CLERK	2900	(null)	(null)	(null)
18	117	Sigal	Tobias	STOBIAS	(null)	24/07/05	PU CLERK	2800	(null)	(null)	(null)
19	118	Guy	Himuro	GHIMURO	(null)	15/11/06	PU CLERK	2600	(null)	(null)	(null)
20	119	Karen	Colmenares	KCOLMENA	(null)	10/08/07	PU CLERK	2500	(null)	(null)	(null)
21	120	Matthew	Weiss	MWEISS	(null)	18/07/04	ST MAN	8000	(null)	(null)	(null)
22	121	Adam	Fripp	AFRIPP	(null)	10/04/05	ST MAN	8200	(null)	(null)	(null)
23	122	Payam	Kaufling	PKAUFLIN	(null)	01/05/03	ST MAN	7900	(null)	(null)	(null)
24	123	Shanta	Vollman	SVOLLMAN	(null)	10/10/05	ST MAN	6500	(null)	(null)	(null)
25	124	Kevin	Mourgos	KMOURGOS	(null)	16/11/07	ST MAN	5800	(null)	(null)	(null)
26	125	Julia	Nayer	JNAYER	(null)	16/07/05	ST CLERK	3200	(null)	(null)	(null)
27	126	Irene	Mikkilineni	IMIKKILI	(null)	28/09/06	ST CLERK	2700	(null)	(null)	(null)

Figura 47: *Data query, employees_{new}table.*

The statement in Figure 46 uses the contents of an EMPLOYEES table to update or insert rows in EMPLOYEES2. The situation could be that EMPLOYEES2 is a table of all staff and EMPLOYEES is a table with rows for new staff and salary changes for existing staff. The command will go through EMPLOYEES2, and for each row, try to find a row in EMPLOYEES with the same EMPLOYEE_ID. If a row is found, its SALARY column will be updated with the value of the row in NEW_EMPLOYEES. If no such row exists, one will be inserted.

Activity 8: Propose a solution to the following scenarios:

a) Transactions, like constraints, are business rules: a technique whereby the database can enforce rules developed by business analysts. If the “logical unit of work” is huge, such as an accounting suite period rollover, should this actually be implemented as one transaction?

R = Not this because many users could make simultaneous changes to the database.

b) Being able to do DML operations, look at the result, then roll back and try them again can be very useful. But is it really a good idea?

R = It's really not a bad idea because if changes are made to the database we can always roll back and when the transaction is still open, this can help not affect the database always and when the transaction is not closed.

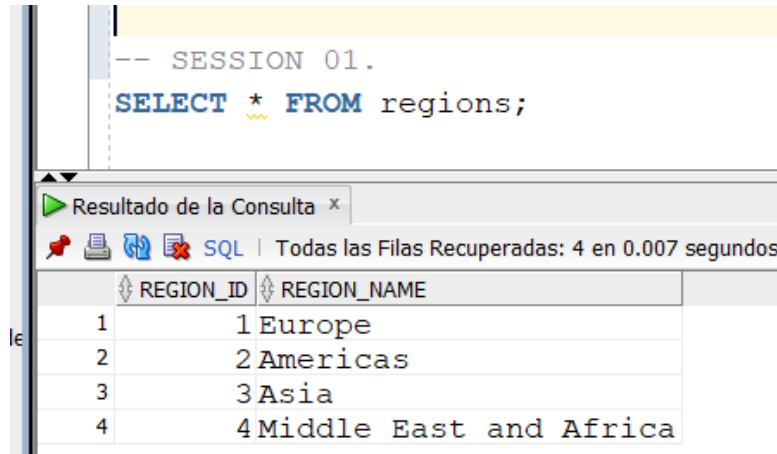
Activity 9: In this exercise, demonstrate the use of transaction control statements and transaction isolation. Connect to the human resources scheme with two sessions at the same time. This can be two sessions of SQL Developer. The following table in **Figure 48** lists the steps to follow in each session.

Step	In your first session	In your second session
1	<code>select * from regions;</code>	<code>select * from regions;</code>
Both sessions see the same data.		
2	<code>insert into regions values(100, 'UK');</code>	<code>insert into regions values(101, 'GB');</code>
3	<code>select * from regions;</code>	<code>select * from regions;</code>
Both sessions see different results: the original data, plus their own change.		
4	<code>commit;</code>	
5	<code>select * from regions;</code>	<code>select * from regions;</code>
One transaction has been published to the world, the other is still visible to only one session.		
6	<code>rollback;</code>	<code>rollback;</code>
7	<code>select * from regions;</code>	<code>select * from regions;</code>
The committed transaction was not reversed because it has already been committed, but the uncommitted one is now completely gone, having been terminated by rolling back the change.		
8	<code>delete from regions where region_id=100;</code>	<code>delete from regions where region_id=101;</code>
9	<code>select * from regions;</code>	<code>select * from regions;</code>
Each deleted row is still visible in the session that did not delete it, until you do the following:		
10	<code>commit;</code>	<code>commit;</code>
11	<code>select * from regions;</code>	<code>select * from regions;</code>
With all transactions terminated, both sessions see a consistent view of the table.		

Figura 48: *Table.*

1. In the first point in both sessions, the data from the REGIONS table are correctly consulted, see figures 49 and 50.

Session 1:

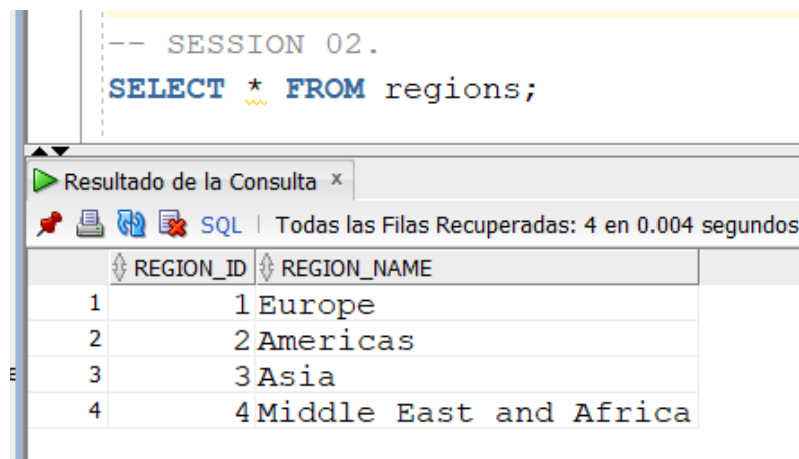


```
-- SESSION 01.  
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa

Figura 49: Querying the data from the REGIONS table.

Session 2:



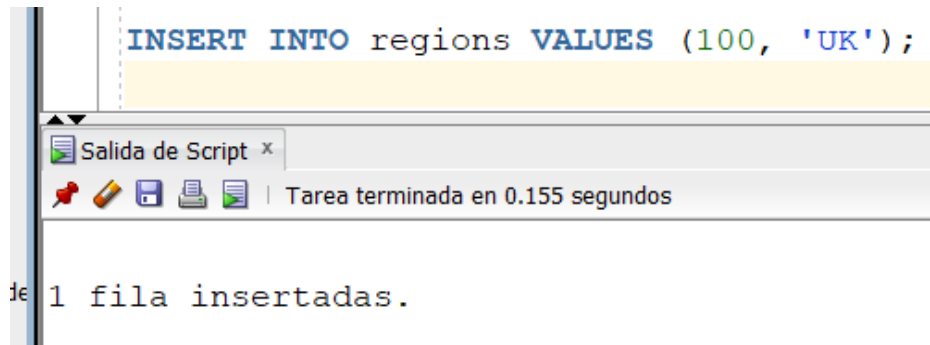
```
-- SESSION 02.  
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa

Figura 50: Querying the data from the REGIONS table.

2. The respective records are inserted in both sessions, see **figures 51 and 52**.

Session 1:



```
INSERT INTO regions VALUES (100, 'UK');
```

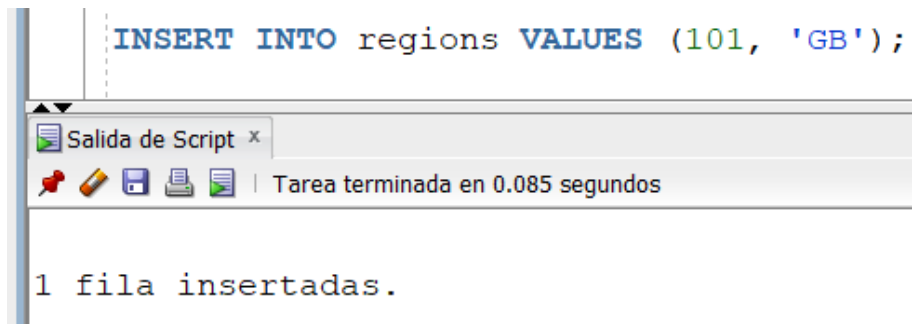
Salida de Script x

Tarea terminada en 0.155 segundos

1 fila insertadas.

Figura 51: *Inserting Records in the REGIONS table.*

Session 2:



```
INSERT INTO regions VALUES (101, 'GB');
```

Salida de Script x

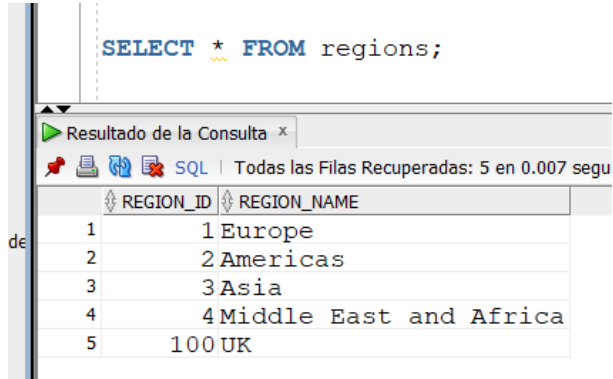
Tarea terminada en 0.085 segundos

1 fila insertadas.

Figura 52: *Inserting Records in the REGIONS table.*

3. Both sessions see different results: the original data, plus your ow3 change, see **figures 53 and 54**.

Session 1:

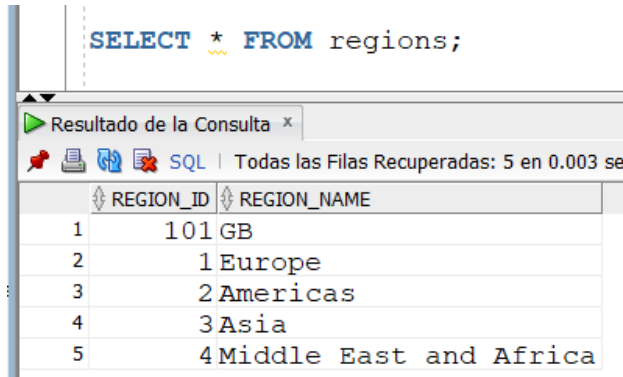


```
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa
100	UK

Figura 53: *Querying Data from the REGIONS table.*

Session 2:



```
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	GB
2	Europe
3	Americas
4	Asia
5	Middle East and Africa

Figura 54: *Querying Data from the REGIONS table.*

4. The transaction is closed in session one, see **figure 55**.

Session 1:

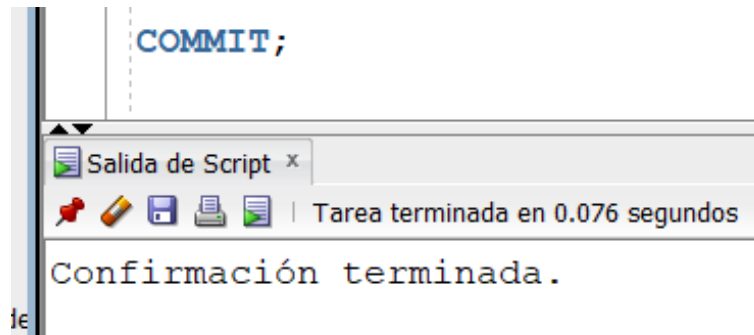


Figura 55: *Confirmation of changes.*

5. One transaction has been published to the world, the other is still visible to a single session. see **figures 56 and 57**.

Session 1:

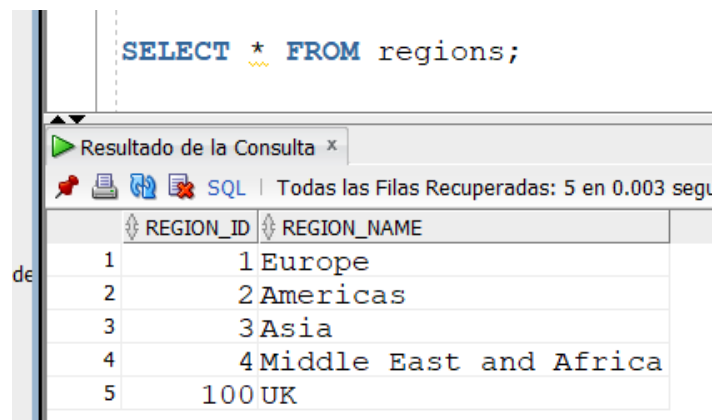
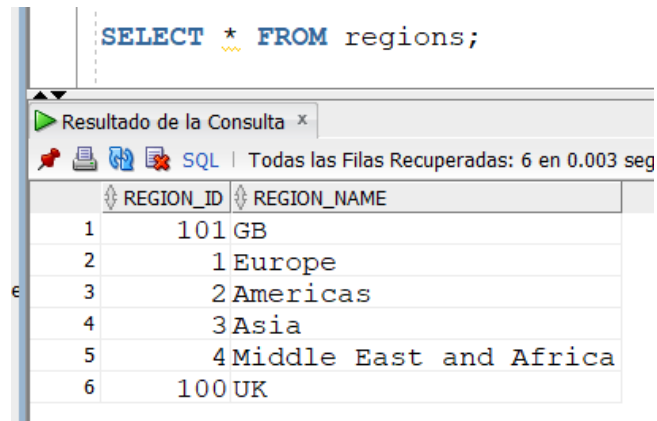


Figura 56: *Querying Data from the REGIONS table.*

Session 2:



```
SELECT * FROM regions;
```

Resultado de la Consulta x

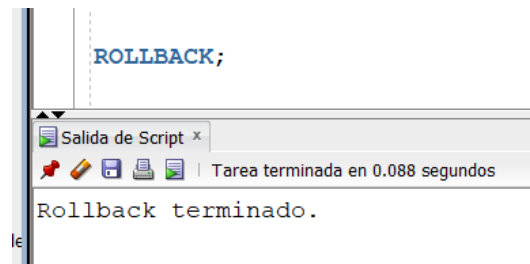
Todas las Filas Recuperadas: 6 en 0.003 seg

REGION_ID	REGION_NAME
1	101 GB
2	1 Europe
3	2 Americas
4	3 Asia
5	4 Middle East and Africa
6	100 UK

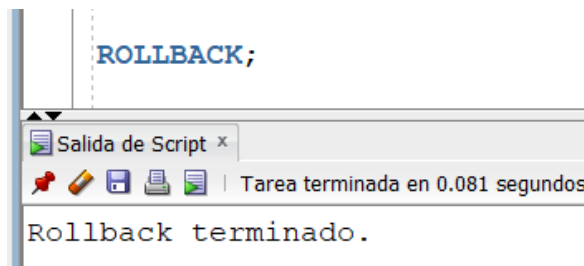
Figura 57: *Querying Data from the REGIONS table.*

6. In both sessions the ROLLBACK is executed correctly, see **figures 58 and 59**.

Session 1:

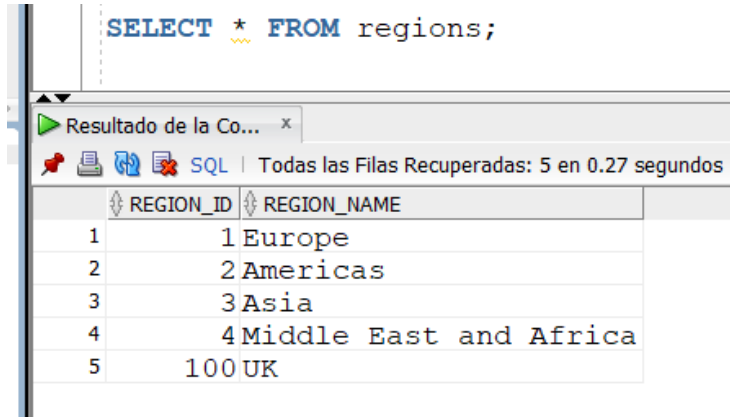
Figura 58: *Execution of a ROLLBACK.*

Session 2:

Figura 59: *Execution of a ROLLBACK.*

7. The committed transaction in session one is not rolled back because it has already been committed, but in session two the uncommitted one is now completely gone as it was canceled by reverting the change, see **figures 60 and 61**.

Session 1:

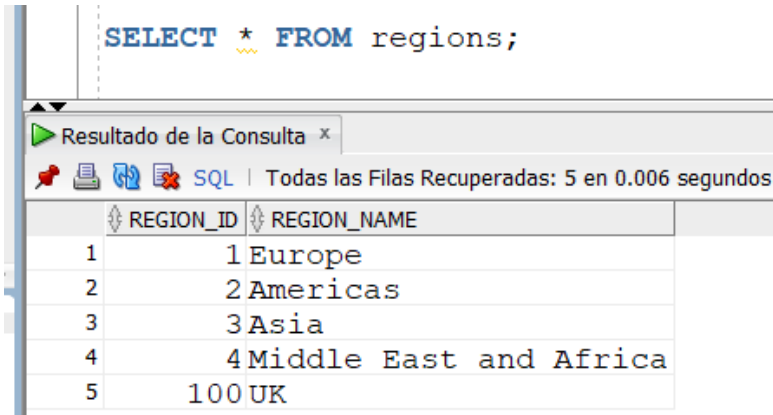


```
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	1 Europe
2	2 Americas
3	3 Asia
4	4 Middle East and Africa
5	100 UK

Figura 60: *Querying data from the REGIONS table.*

Session 2:



```
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	1 Europe
2	2 Americas
3	3 Asia
4	4 Middle East and Africa
5	100 UK

Figura 61: *Querying data from the REGIONS table.*

8. In session one the record with the id equal to 100 was deleted and in session two no record was deleted because the record with the id equal to 101 did not exist in the table, see **figures 62 and 63**.

Session 1:

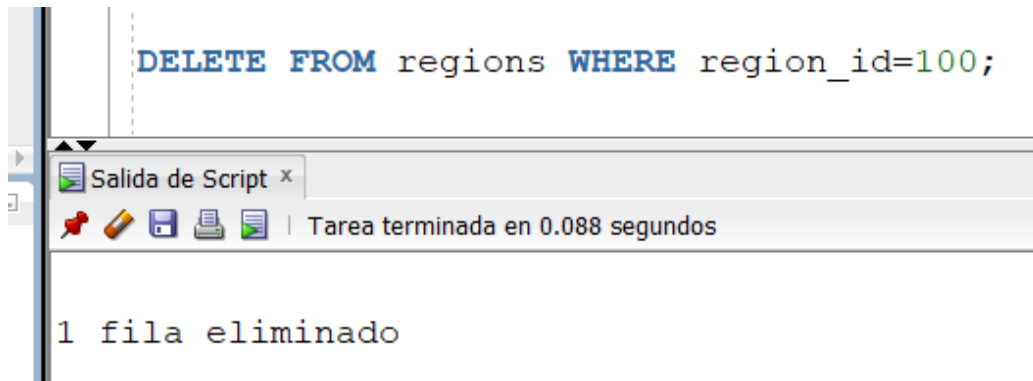


Figura 62: *Delete records.*

Session 2:

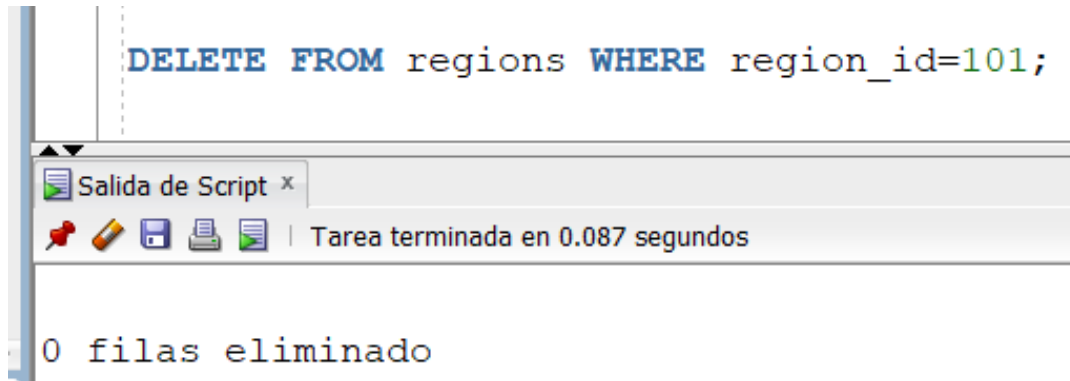
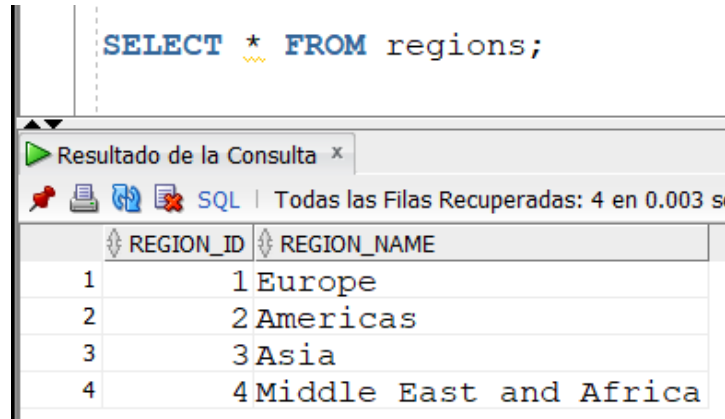


Figura 63: *Delete records.*

9. Each deleted row is still visible in the session that didn't delete it, until you do a COMMIT, see figures 64 and 65.

Session 1:

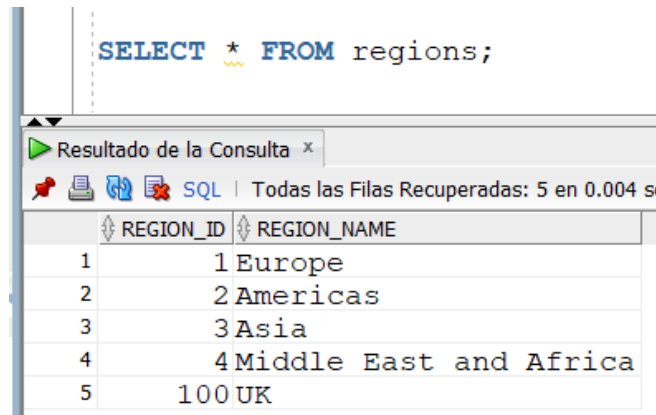


```
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	1 Europe
2	2 Americas
3	3 Asia
4	4 Middle East and Africa

Figura 64: Querying data from the *REGIONS* table.

Session 2:



```
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	1 Europe
2	2 Americas
3	3 Asia
4	4 Middle East and Africa
5	100 UK

Figura 65: Querying data from the *REGIONS* table.

10. Changes were confirmed in both sessions, see **figures 66 and 67**.

Session 1:

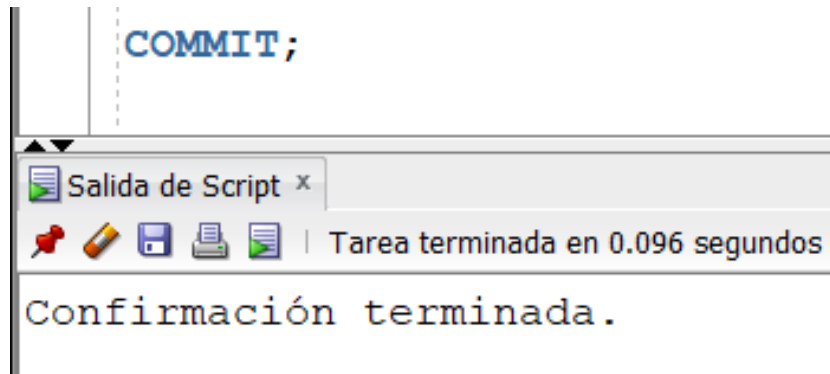


Figura 66: *Confirmation of changes.*

Session 2:

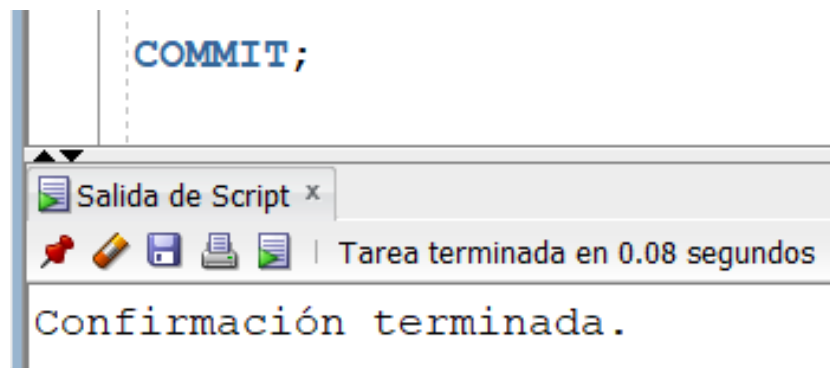
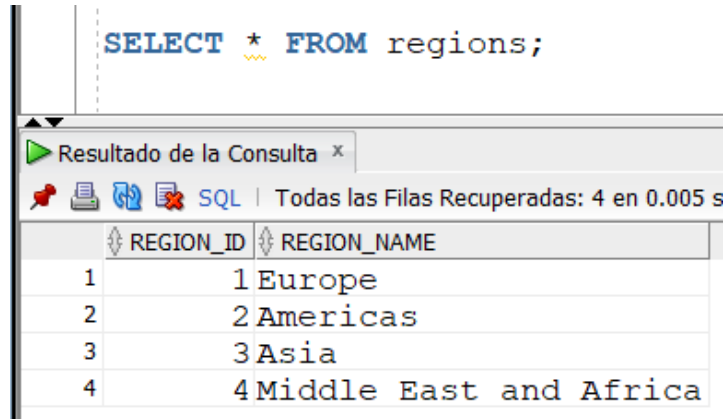


Figura 67: *Confirmation of changes.*

11. With all transactions completed, both sessions are a consistent view of the table, see **figures 68 and 69**.

Session 1:

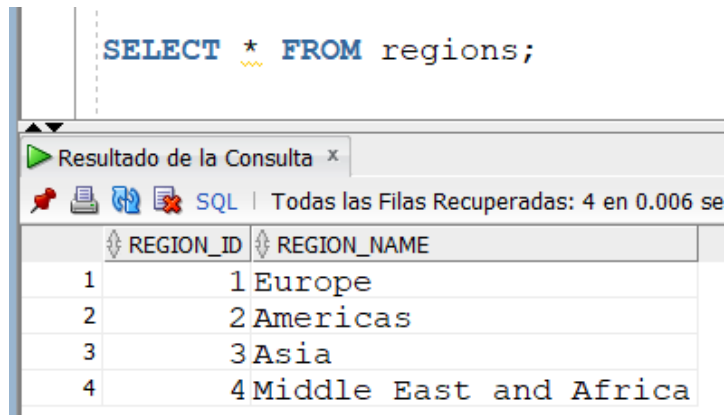


```
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	1 Europe
2	2 Americas
3	3 Asia
4	4 Middle East and Africa

Figura 68: *Querying data from the REGIONS table.*

Session 2:



```
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	1 Europe
2	2 Americas
3	3 Asia
4	4 Middle East and Africa

Figura 69: *Querying data from the REGIONS table.*

Activity 10: Write the section that describes the **Work developed** in the following activities (capture an image for each statement output).

The HR department wants you to create SQL statements to insert, update, and delete employee data. As a prototype, you use the MY_EMPLOYEE table before giving the statements to the HR department.

Note: For all the DML statements, use the Run Script icon (or press [F5]) to execute the query. This way you get to see the feedback messages on the Script Output tab page. For SELECT queries, continue to use the Execute Statement icon or press [F9] to get the formatted output on the Results tab page.

Insert data into the MY_EMPLOYEE table.

1. Create the DDL sentence to build the MY_EMPLOYEE table used in this activity (see **figure 70**). Save this sentences in lab_06_01.sql.

DESCRIBE MY_EMPLOYEE		
Name	Null	Type

ID	NOT NULL	NUMBER(4)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
USERID		VARCHAR2(8)
SALARY		NUMBER(9,2)

Figura 70: *Description of the MY_EMPLOYEE table.*

2. Describe the structure of the MY_EMPLOYEE table to identify the column names, See **figure 71**

```
CREATE TABLE MY_EMPLOYEE (  
    ID NUMBER(4) CONSTRAINT MY_EMPLOYEE_PK PRIMARY KEY,  
    LAST_NAME VARCHAR2(25),  
    FIRST_NAME VARCHAR2(25),  
    USERID VARCHAR2(8),  
    SALARY NUMBER(9,2)  
);  
  
DESC MY_EMPLOYEE;
```

Salida de Script x

Tarea terminada en 0.58 segundos

Table MY_EMPLOYEE creado.

Nombre	¿Nulo?	Tipo
ID	NOT NULL	NUMBER(4)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
USERID		VARCHAR2(8)
SALARY		NUMBER(9,2)

Figura 71: *Creation and description of the MY_EMPLOYEE table.*

3. Create an INSERT statement to add the first row of data to the MY_EMPLOYEE table from the following sample data. Do not list the columns in the INSERT clause. Do not enter all rows yet, see figure 72.

```
INSERT INTO MY_EMPLOYEE (ID, LAST_NAME, FIRST_NAME, USERID, SALARY)
VALUES (1, 'Patel', 'Ralph', 'rpatel', 895);
```

Salida de Script x

Tarea terminada en 0.106 segundos

1 fila insertadas.

Figura 72: Inserting records into the MY_EMPLOYEE table.

4. Populate the MY_EMPLOYEE table with the second row of the sample data from the preceding list. This time, list the columns explicitly in the INSERT clause, see **figure 73**.

```
INSERT INTO MY_EMPLOYEE VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);
```

Salida de Script x

Tarea terminada en 0.106 segundos

1 fila insertadas.

Figura 73: Inserting records into the MY_EMPLOYEE table.

5. Confirm your addition to the table, see **figure 74**.

```
SELECT * FROM MY_EMPLOYEE;
```

Resultado de la Co... x

Todas las Filas Recuperadas: 2 en 0.065 seg

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	895
2	2	Dancs	Betty	bdancs	860

Figura 74: Query data from the MY_EMPLOYEE table.

6. Write an INSERT statement in a dynamic reusable script file to load the remaining rows into the MY_EMPLOYEE table. The script should prompt for all the columns (ID, LAST_NAME, FIRST_NAME, USERID, and SALARY). Save this script to a lab_06_06.sql file, see **figure 75**.

```
INSERT INTO MY_EMPLOYEE (ID, LAST_NAME, FIRST_NAME, USERID, SALARY)
VALUES (&ID, '&LAST_NAME', '&FIRST_NAME', '&USERID', &SALARY);
```

Figura 75: *Inserting records into the MY_EMPLOYEE table.*

7. Populate the table with the next two rows of the sample data listed in step 3 by running the INSERT statement in the script that you created, see **figure 76, 77, 78, 79, 80, 81 and 82**.

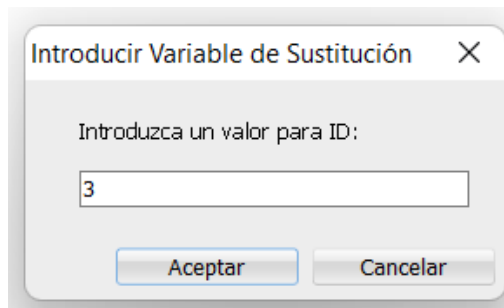


Figura 76: *Inserting records into the MY_EMPLOYEE table.*

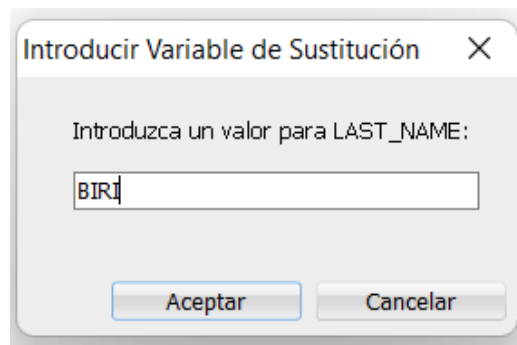


Figura 77: *Inserting records into the MY_EMPLOYEE table.*

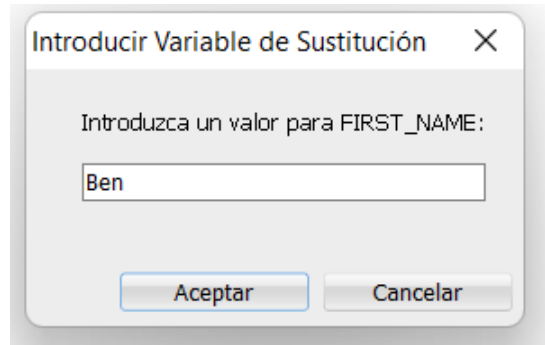


Figura 78: *Inserting records into the MY_EMPLOYEE table.*

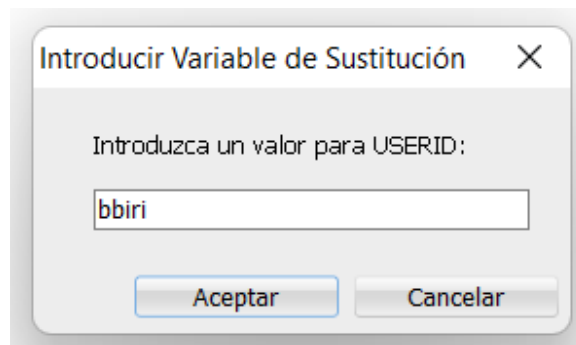


Figura 79: *Inserting records into the MY_EMPLOYEE table.*

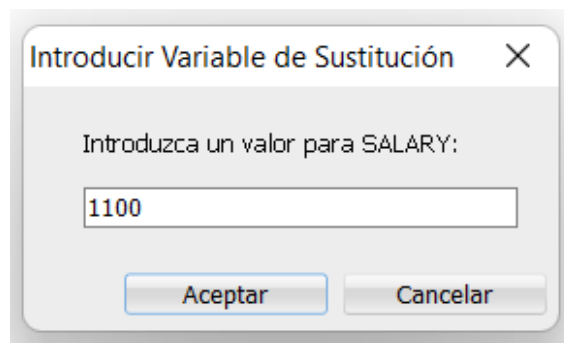
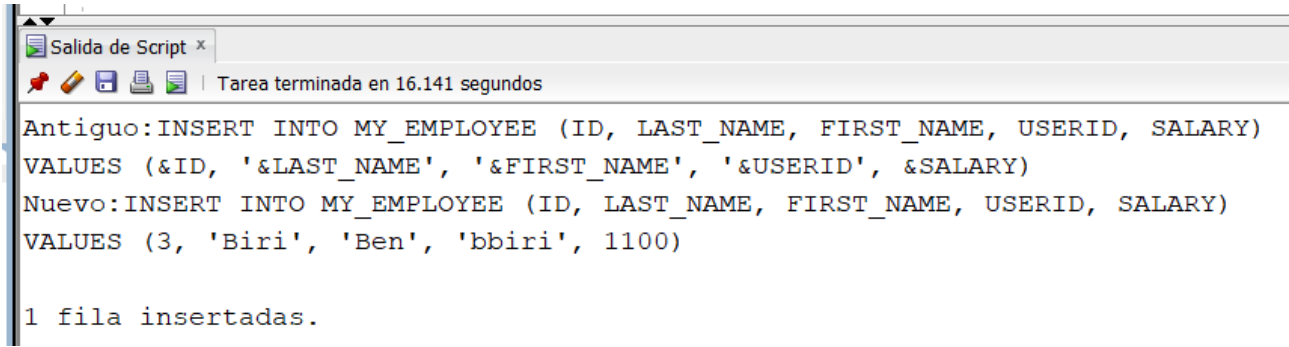


Figura 80: *Inserting records into the MY_EMPLOYEE table.*



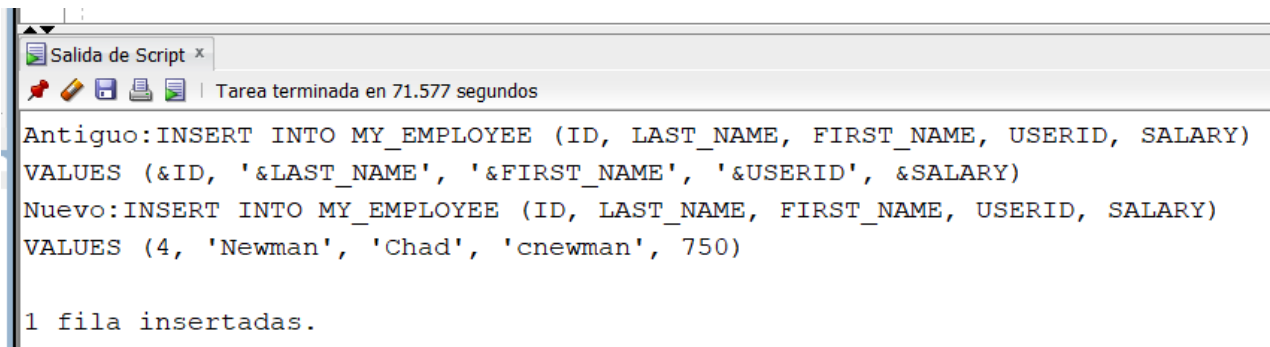
```

Salida de Script x
Tarea terminada en 16.141 segundos

Antiguo:INSERT INTO MY_EMPLOYEE (ID, LAST_NAME, FIRST_NAME, USERID, SALARY)
VALUES (&ID, '&LAST_NAME', '&FIRST_NAME', '&USERID', &SALARY)
Nuevo:INSERT INTO MY_EMPLOYEE (ID, LAST_NAME, FIRST_NAME, USERID, SALARY)
VALUES (3, 'Biri', 'Ben', 'bbiri', 1100)

1 fila insertadas.

```

Figura 81: *Inserting records into the MY_EMPLOYEE table.*


```

Salida de Script x
Tarea terminada en 71.577 segundos

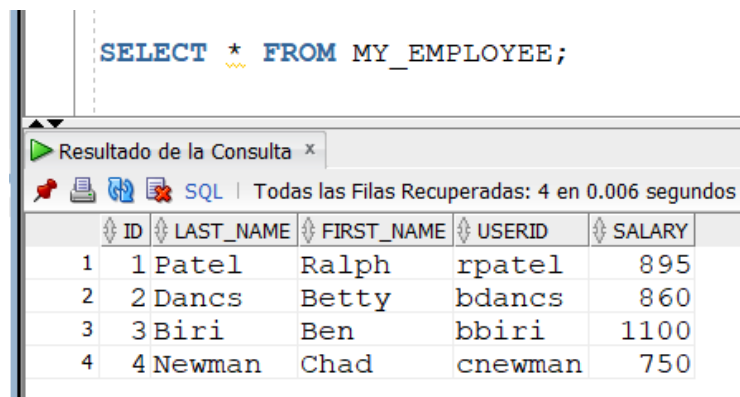
Antiguo:INSERT INTO MY_EMPLOYEE (ID, LAST_NAME, FIRST_NAME, USERID, SALARY)
VALUES (&ID, '&LAST_NAME', '&FIRST_NAME', '&USERID', &SALARY)
Nuevo:INSERT INTO MY_EMPLOYEE (ID, LAST_NAME, FIRST_NAME, USERID, SALARY)
VALUES (4, 'Newman', 'Chad', 'cnewman', 750)

1 fila insertadas.

```

Figura 82: *Inserting records into the MY_EMPLOYEE table.*

8. Confirm your additions to the table, see **figure 83**.



```

SELECT * FROM MY_EMPLOYEE;

```

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750

Figura 83: *Check for changes.*

9. Make the data additions permanent, see **figure 84**.

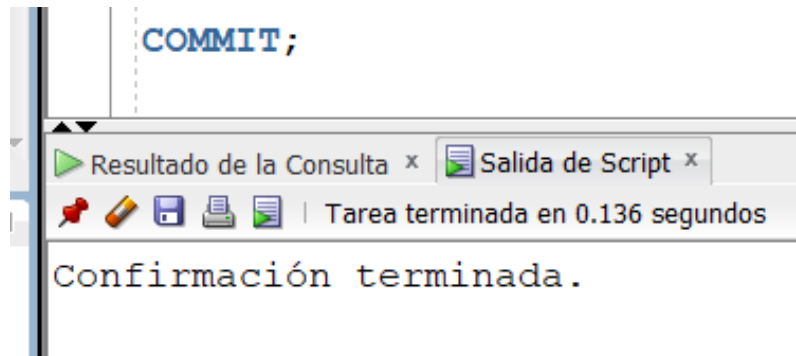


Figura 84: *Confirmation of changes.*

Update and delete data in the MY_EMPLOYEE table.

10. Change the last name of employee 3 to Drexler, see **figure 85**.

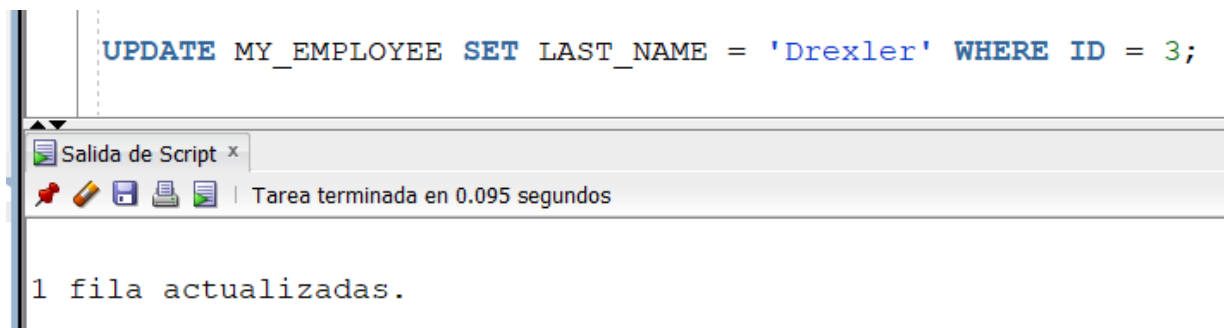


Figura 85: *Updating records in the MY_EMPLOYEE table.*

11. Change the salary to \$1,000 for all employees who have a salary less than \$900, see **figure 86**.

```
UPDATE MY_EMPLOYEE SET SALARY = 1000 WHERE SALARY < 900;
```

Salida de Script x

Tarea terminada en 0.081 segundos

3 filas actualizadas.

Figura 86: Updating records in the MY_EMPLOYEE table.

12. Verify your changes to the table, see **figure 87**.

```
SELECT * FROM MY_EMPLOYEE;
```

Resultado de la Consulta x

Todas las Filas Recuperadas: 4 en 0.005 segundos

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	1000
2	2	Dancs	Betty	bdancs	1000
3	3	Drexler	Ben	bbiri	1100
4	4	Newman	Chad	cnewman	1000

Figura 87: Check for changes.

13. Delete Betty Dancs from the MY_EMPLOYEE table, see **figure 88**.

```
DELETE FROM MY_EMPLOYEE WHERE FIRST_NAME = 'Betty' AND LAST_NAME = 'Dancs';
```

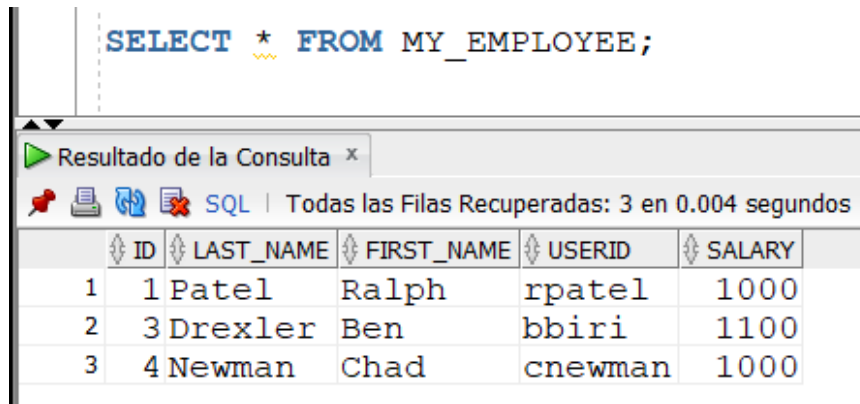
Salida de Script x

Tarea terminada en 0.1 segundos

1 fila eliminado

Figura 88: Delete a record from the MY_EMPLOYEE table.

14. Confirm your changes to the table, see **figure 89**.



	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	1000
2	3	Drexler	Ben	bbiri	1100
3	4	Newman	Chad	cnewman	1000

Figura 89: *Check for changes.*

15. Commit all pending changes, see **figure 90**.

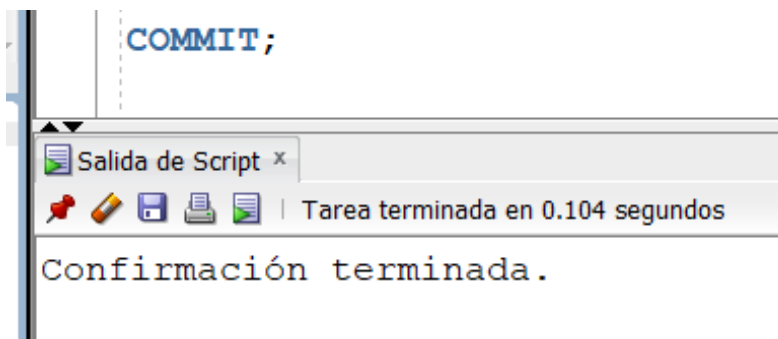
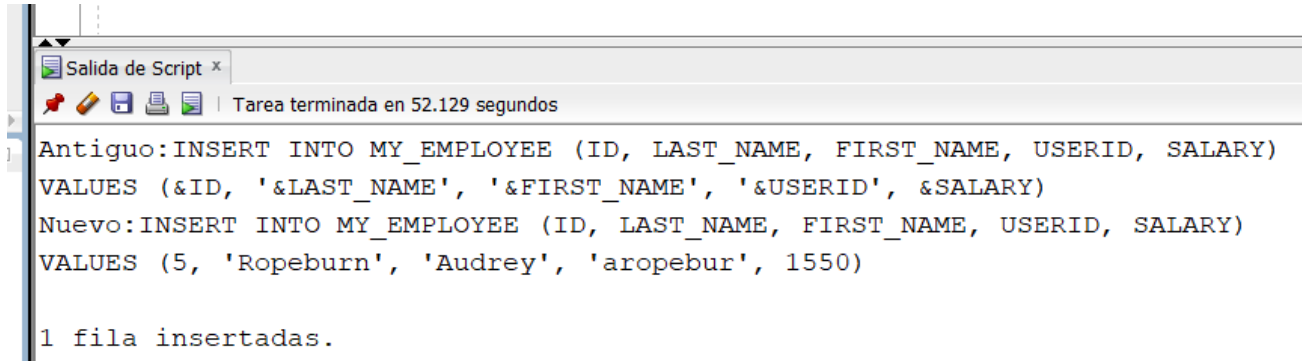


Figura 90: *Confirmation of changes.*

Control data transaction to the MY_EMPLOYEE table.

16. Populate the table with the last row of the sample data listed in step 3 by using the statements in the script that you created in step 6. Run the statements in the script, see **figure 91**.



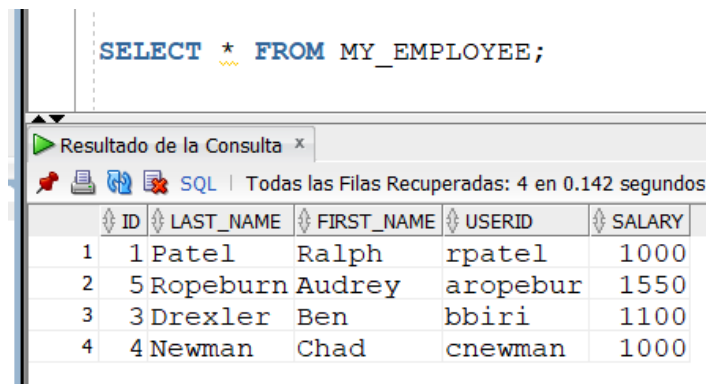
```
Salida de Script x
Tarea terminada en 52.129 segundos

Antiguo:INSERT INTO MY_EMPLOYEE (ID, LAST_NAME, FIRST_NAME, USERID, SALARY)
VALUES (&ID, '&LAST_NAME', '&FIRST_NAME', '&USERID', &SALARY)
Nuevo:INSERT INTO MY_EMPLOYEE (ID, LAST_NAME, FIRST_NAME, USERID, SALARY)
VALUES (5, 'Ropeburn', 'Audrey', 'aropebur', 1550)

1 fila insertadas.
```

Figura 91: *Inserting records into the MY_EMPLOYEE table.*

17. Confirm your addition to the table, see **figure 92**.

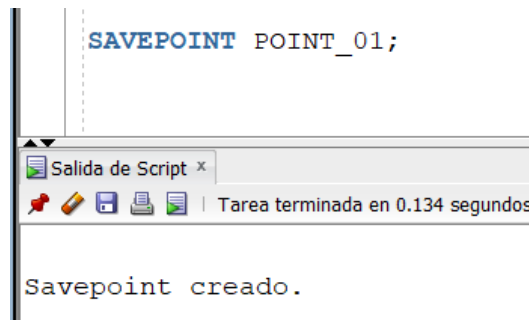


```
SELECT * FROM MY_EMPLOYEE;
```

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	1000
2	Ropeburn	Audrey	aropebur	1550
3	Drexler	Ben	bbiri	1100
4	Newman	Chad	cnewman	1000

Figura 92: *Check for changes.*

18. Mark an intermediate point in the processing of the transaction, see **figure 93**.



```
SAVEPOINT POINT_01;
```

```
Salida de Script x
Tarea terminada en 0.134 segundos

Savepoint creado.
```

Figura 93: *Create a save point.*

19. Delete all the rows from the MY_EMPLOYEE table, see **figure 94**.

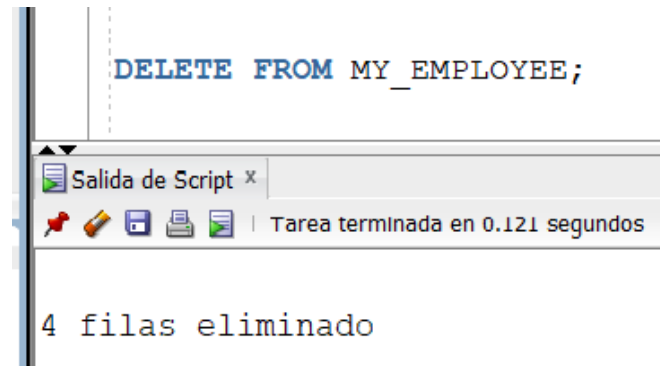


Figura 94: *Deleting the MY_EMPLOYEE table.*

20. Confirm that the table is empty, see **figure 95**.

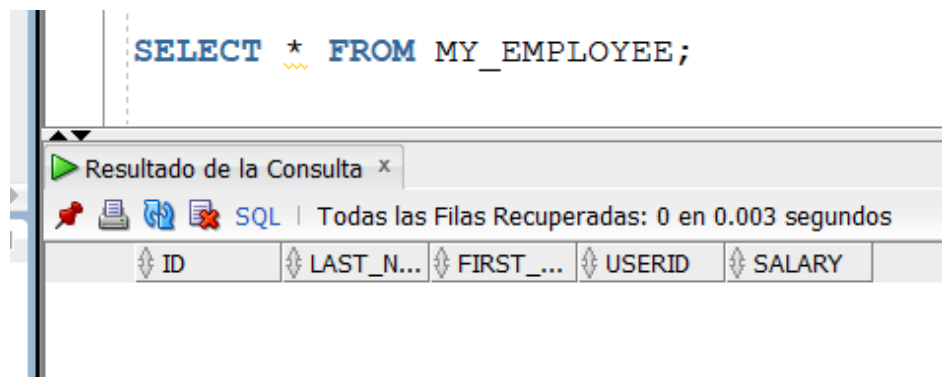
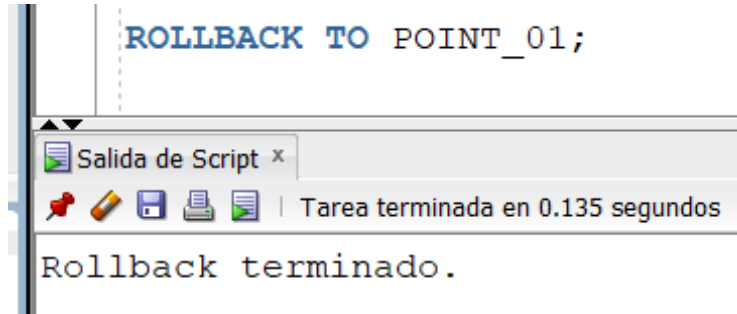
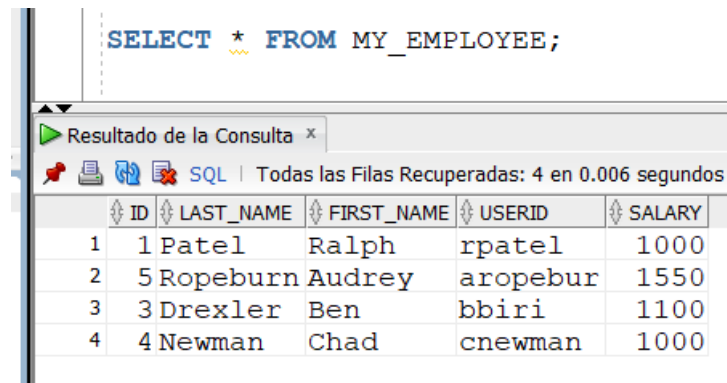


Figura 95: *Check for changes.*

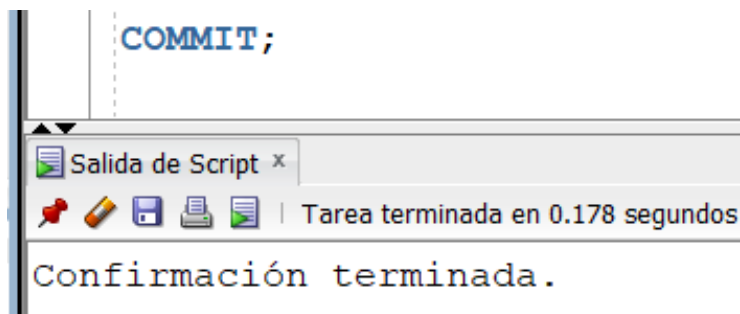
21. Discard the most recent DELETE operation without discarding the earlier INSERT operation, see **figure 96**.

Figura 96: *Reversal of changes*

22. Confirm that the new row is still intact, see **figure 97**.

Figura 97: *Check for changes.*

23. Make the data addition permanent, see **figure 98**.

Figura 98: *Confirmation of changes.*

24. Modify the lab_06_06.sql script such that the USERID is generated automatically by concatenating the first letter of the first name and the first seven characters of the last name. The generated USERID must be in lowercase. Hence, the script should not prompt for the USERID. Save this script to a file named lab_06_24.sql, see **figure 99**.

```
INSERT INTO MY_EMPLOYEE (ID, LAST_NAME, FIRST_NAME, USERID, SALARY)
VALUES (&ID, '&LAST_NAME', '&FIRST_NAME', SUBSTR(FIRST_NAME, 0,1) || ' ' || LAST_NAME, &SALARY);

CREATE OR REPLACE FUNCTION GET_USERID (
    F_NAME VARCHAR2,
    L_NAME VARCHAR2
)
RETURN VARCHAR2
IS
    RET VARCHAR2(50);
BEGIN
    SELECT SUBSTR(FIRST_NAME, 0,1) || ' ' || LAST_NAME INTO RET FROM MY_EMPLOYEE
    WHERE FIRST_NAME = F_NAME AND LAST_NAME = L_NAME;
    RETURN RET;
END GET_USERID;

SELECT GET_USERID(FIRST_NAME, LAST_NAME) FROM MY_EMPLOYEE;
```

Figura 99: *Inserting records into the MY_EMPLOYEE table.*

Note: This point number 24 could not be done because the script developed based on a function that did not work correctly.

25. Run the script, lab_06_24.sql to insert the following record, **it's not over** .

26. Confirm that the new row was added with correct USERID, **it's not over** .

4. Pre-assessment

In this section you will find the Pre-assessment

Criteria to be evaluate	Does it comply? (%)
COMPLIES WITH THE REQUESTED FUNCTIONALITY	YES
HAS THE CORRECT INDENTATION	YES
HAS AN EASY WAY TO ACCESS THE PROVIDED FILES	YES
HAS A REPORT WITH IDC FORMAT	YES
REPORT INFORMATION IS FREE OF SPELLING ERRORS	YES
DELIVERED IN TIME AND FORM	YES
IS FULLY COMPLETED (SPECIFY THE PERCENTAGE COMPLETED)	YES 98 %

5. Conclusion

The Oracle DML statements are transcendental in the handling of SQL statements at the level of both administrator and database programmer, since they allow the data manipulation of database schemes regardless of the platform used to generate it. This kind of statements can provide you data treatment mechanisms during daily programmer's days.

This practice number 6 helped me to practice the uses of the various DML sentences, although there were things that I did not understand one hundred percent, there are others from which I learned different ways of carrying out the sentences.