**APARTADO 1**

1.

*Código:*

CREATE TABLE cuentas (

numero number primary key,

saldo number not null

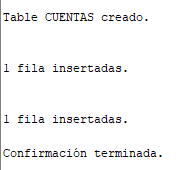
);

INSERT INTO cuentas VALUES (123, 400);

INSERT INTO cuentas VALUES (456, 300);

COMMIT;

Output:



2. Hemos creado T2

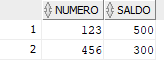
3. En ambos T ejecutamos

SET AUTOCOMMIT OFF;

4.

UPDATE cuentas SET saldo = saldo + 100 WHERE numero = 123;

Consulta de datos desde T1



5. Consultamos la tabla desde T2



6. Hacemos COMMIT en T1

Código:

COMMIT;

Output:



7. Consultamos la tabla en T2



**APARTADO 2**

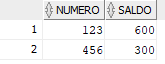
1.

Código:

UPDATE cuentas SET saldo = saldo + 100 WHERE numero = 123;

Output:





(Tabla en T1)

2. Si tratamos de aumentar el saldo desde T2, el programa se queda pillado porque hay un bloqueo a nivel de fila, que evita que dos transacciones accedan al mismo recurso a la vez.

Código:

UPDATE cuentas SET saldo = saldo + 200 WHERE numero = 123;

Output:



Esta barra se queda todo el rato cargando

3. Hacemos commit en t1.

Código:

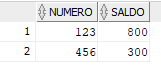
COMMIT;

Output:



Al haber confirmado los cambios en T1, Oracle desbloquea la fila y la transacción T2 termina:



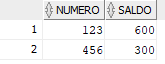


(Tabla en T2)

Ahora el saldo de 123 toma el valor 800: a los 500 que teníamos inicialmente se le suman los 100 que se han confirmado desde T1 y los 200 que le hemos añadido desde T2.

4.

Si consultamos desde T1, el saldo sigue siendo 600.



5. Confirmamos la transacción T2

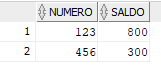
Código:

COMMIT;

Output:



6. Si volvemos a consultar desde T1, veremos que se ha actualizado la fila



**APARTADO 3**

1.

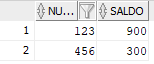
Aumentamos 100e el saldo de 123 desde T1.

Código:

UPDATE cuentas SET saldo = saldo + 100 WHERE numero = 123;

Output:





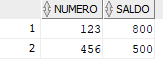
2. Aumentamos el saldo en 200e a la cuenta 456 desde T2.

Código:

UPDATE cuentas SET saldo = saldo + 200 WHERE numero = 456;

Output:





3. Aumentamos 300e el saldo de 456 desde T1.

Código:

UPDATE cuentas SET saldo = saldo + 300 WHERE numero = 456;

Output:



La fila está bloqueada porque se han realizado cambios desde T2 y no se han commiteado.

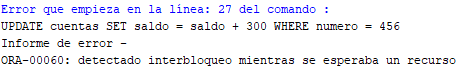
4. Aumentamos el saldo en 400e a la cuenta 123 desde T2.

Código:

UPDATE cuentas SET saldo = saldo + 400 WHERE numero = 123;

Output:

Desde T1:



Desde T2:



Se ha producido un interbloqueo porque T1 está esperando datos bloqueados por T2 y viceversa. T2 permanece bloqueada porque, aunque Oracle ha abortado solo una de las operaciones que ha provocado el interbloqueo, pero estamos modificando dos saldos.

5. Confirmamos los cambios en T1.

Código:

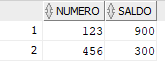
COMMIT;

Output:

Desde T1:



Desde T2:



Como vemos, Oracle ha matado la transacción T2, solo se mantienen los cambios de T1.

**APARTADO 4**

1.

En T1,

Código:

ALTER SESSION SET ISOLATION\_LEVEL = SERIALIZABLE;

Output:



2.

En T1

Código:

SELECT SUM(saldo) FROM cuentas;

Output:



3.

En T2

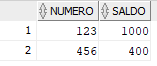
Código:

UPDATE cuentas SET saldo=saldo+100;

COMMIT;

Output:





(Tabla vista en T2)

4.

Código:

SELECT SUM(saldo) FROM cuentas;

Output:



Al haber cambiado la sesión a SERIALIZABLE hemos impuesto el nivel de aislamiento de mayor grado, por lo que las transacciones se ejecutan en serie, una detrás de otra. Se asegura que los datos recuperados por las cosultas incluidas en una transacción son consistentes en el contexto de la transacción. Este nivel de aislamiento no permite la lectura no repetible.

5.

En T1:

Código:

ALTER SESSION SET ISOLATION\_LEVEL = READ COMMITTED;

Output:



6.

En T1:

Código:

SELECT SUM(saldo) FROM cuentas;

Output:



El nivel de aislamiento ahora es READ COMMITED, el nivel de aislamiento más concurrente que garantiza la integridad de lo leído. Con este nivel no se protege de la lectura no repetible. Por eso, al haber comprometido el update de T2, podemos detectarlo desde T1.

7.

En T2:

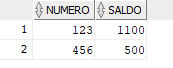
Código:

UPDATE cuentas SET saldo = saldo+100;

COMMIT;

Output:





(Tabla en T2)

8.

En T1.

Código:

SELECT SUM(saldo) FROM cuentas;



Una vez más, al no haber protección frente a la lectura no repetible, y al haber commiteado el cambio en T2, se muestran los cambios realizados. Así, se puede apreciar una clara diferencia entre SERIALIZABLE y READ COMMITED.

**APARTADO 5**

1.

Cerramos las instancias anteriores de SQL-D

2.

Creamos las tablas.

Código:

CREATE TABLE butacas(id number(8) primary key,

evento varchar(30),

fila varchar(10),

columna varchar(10)) ;

CREATE TABLE reservas(id number(8) primary key,

evento varchar(30),

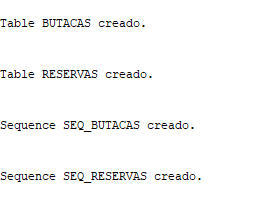
fila varchar(10),

columna varchar(10)) ;

CREATE SEQUENCE Seq\_Butacas INCREMENT BY 1 START WITH 1 NOMAXVALUE;

CREATE SEQUENCE Seq\_Reservas INCREMENT BY 1 START WITH 1 NOMAXVALUE;

Output:



3.

Insertamos algunos valores en butacas.

Código:

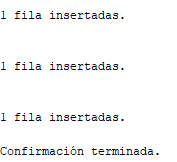
INSERT INTO butacas VALUES (Seq\_Butacas.NEXTVAL,'Circo','1','1');

INSERT INTO butacas VALUES (Seq\_Butacas.NEXTVAL,'Circo','1','2');

INSERT INTO butacas VALUES (Seq\_Butacas.NEXTVAL,'Circo','1','3');

COMMIT;

Output:



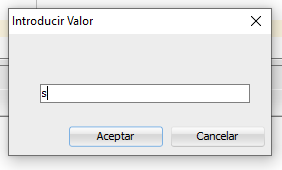
4.

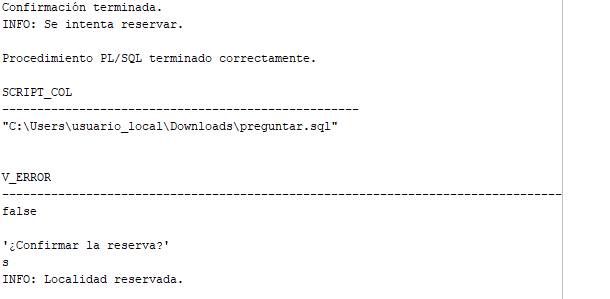
Actualizamos los directorios de los scripts en reservar.sql y lo ejecutamos.

Código:

@ C:\Users\usuario\_local\Downloads\reservar.sql

Output:





Como vemos se ha generado la reserva correctamente.



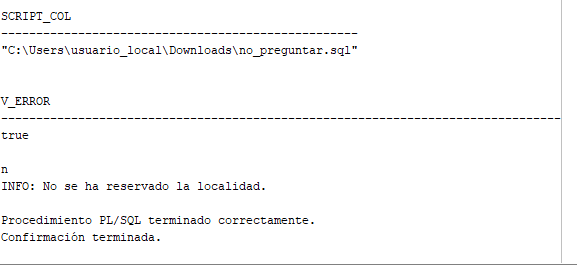
5.

Volvemos a ejecutar el script.

Código:

@ C:\Users\usuario\_local\Downloads\reservar.sql

Output:



Como es obvio, no se permite realizar la reserva puesto que ya se había realizado previamente.

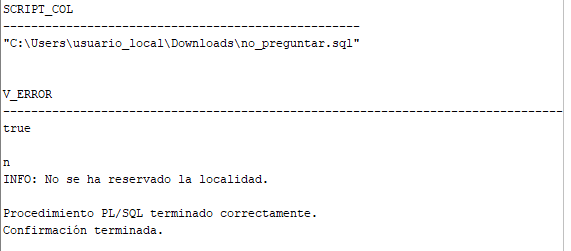
6.

Cambiamos el script para reservar la fila 1, columna 4 para Circo

Input:



Output:



No nos permite reservar porque no existe dicha butaca

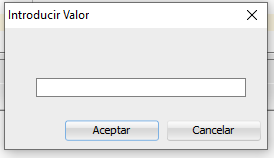
7.

Cambiamos el script para reservar la fila 1, columna 2 para Circo

Input:

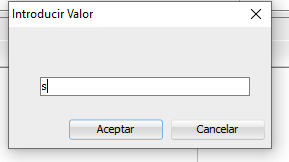


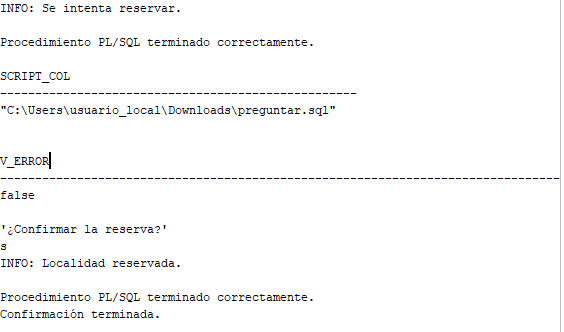
Output:



8.

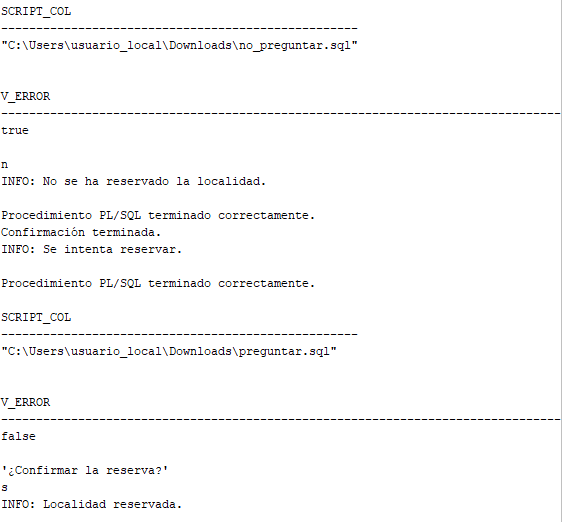
Abrimos una nueva instancia y realizamos la misma reserva, confirmándola.





9.

También nos deja reservar la butaca. Esto es un problema, porque hemos reservado la misma butaca en dos transacciones distintas.



10.

Actualmente se comprueba la disponibilidad de la butaca únicamente al comienzo del script, pero también es necesario hacerlo justo antes de confirmar la reserva para asegurar que nadie ha reservado el asiento en ese periodo de tiempo. Si la respuesta es afirmativa hacemos commit, sino hacemos rollback de la transacción