

PRÁCTICA 2

APARTADO 1

a) Hacer pruebas repetibles y comprobables

Para ver los efectos de los mecanismos de Oracle necesitamos simular que las transacciones trabajen durante un periodo de tiempo controlado, pudiendo pararlas y arrancarlas en los momentos precisos para provocar concurrencia a las tablas y situaciones concretas.

Por ello vamos a hacer:

Crear una secuencia `sec_T`, cuyo valor mínimo es 0 y máximo es 1. La uso como un semáforo. Cada vez que sumo 1: si estaba en 1 vuelve a 0. Creamos una secuencia para cada transacción (T) a simular: `sec_T1`, `sec_T2`, etc. NO uses `plsql` dinámico en esta práctica.

```
CREATE SEQUENCE sec_T
START WITH 0
INCREMENT BY 1
MINVALUE 0
MAXVALUE 1
CYCLE NOCACHE;
```

Un procedimiento simulador *trabajando_T* (*X* *núm.de segundos*). Para cada T a simular hacemos un procedimiento distinto: *trabajando_T1* (*X*) , *trabajando_T2* (*X*), etc. El pseudocódigo es este:

Pasos dentro de un bucle:

- Llama al proc. *hector.dormir* (*núm.segundos*) para detener el proceso durante esos segundos(ej.:5):
(no necesitas ver el contenido, está en la cuenta hector)
(si usas tu portátil necesitas crear el proc *dormir*: ver archivo *crear_sinonimos.pdf*).
- Comprueba si debe terminar de trabajar: si ve que no, vuelve al principio del bucle y repite el ciclo.
- La forma de indicarle que termine es usando la secuencia *sec_T*. Cuando la modifiquemos desde otra T. (otra copia del SQLdeveloper), terminará el bucle y el procedimiento.
- Para facilitar el seguimiento: lo último que hace el procedimiento será dar el mensaje “he terminado de trabajar” junto con el número de la transacción donde estaba (ver *que-num-trans.sql*).

```
create or replace
PROCEDURE TRABAJANDO_T1 (numSegundos INTEGER) AS
LN_TMP INTEGER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Comienzo');
    LN_TMP := SEC_T.NEXTVAL;
    WHILE LN_TMP != SEC_T.NEXTVAL
    LOOP
        LN_TMP := SEC_T.NEXTVAL;
        hector.dormir(numSegundos);
        DBMS_OUTPUT.PUT_LINE('Sigo dormido');
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('He terminado de trabajar');
END TRABAJANDO_T1;
```

b) Probar el procedimiento *trabajando_T1* del siguiente modo:

- Hacer un procedimiento *probarMiT1*, que incluye estos pasos:
- Empezar una T con INSERTs de tres COMPRAS: formato igual que en *BDejemplo.sql*
- Parar la T, llamando a *trabajando_T1 (5)*. (se dormirá hasta que le demos la orden)
- Continuar la T con otros INSERTs de otras tres COMPRAS
- Después parar la T de nuevo, poniendo una 2ª llamada a *trabajando_T1 (5)* (se vuelve a dormir hasta otra orden).
- Ejecutar *probarMiT1*, y, desde otra transacción (abre otro sqlDeveloper) hacer una modificación de la secuencia para provocar que *probarMiT1* continúe hasta la 2ª llamada. Si volvemos a hacer otra modificación a la secuencia, continuará y terminará.
- Comprueba que ha insertado lo esperado.

```
CREATE OR REPLACE PROCEDURE PROBARMIT1 AS
BEGIN
    INSERT INTO Compras VALUES ('00000005', '50000400',1,
0501,'tienda1',50);
    INSERT INTO Compras VALUES ('00000005', '50000030',1,
0501,'tienda1',5);
    INSERT INTO Compras VALUES ('00000005', '50000400',2,
0502,'tienda1',500);
    TRABAJANDO_T1(5);
    INSERT INTO Compras VALUES ('00000005', '50000400',1,
0501,'tienda1',50);
    INSERT INTO Compras VALUES ('00000005', '50000030',1,
0501,'tienda1',5);
    INSERT INTO Compras VALUES ('00000005', '50000400',2,
0502,'tienda1',500);
    TRABAJANDO_T1(5);
END PROBARMIT1;
```

c) Probar el procedimiento *trabajando_T1* con dos Ts: *Así probaremos los otros Apartados de la Prác.*

- Ahora simula dos Ts concurrentes: falta hacer otro procedimiento *probarMIT2* (el mismo contenido que el *probarMIT1*), que llame a un nuevo *trabajando_T2 (5)* que tenga una nueva secuencia *sec_T2*. Este procedimiento se ejecuta en otra copia nueva del sqlDeveloper. Las ordenes de continuar se las damos desde una tercera copia del sqlDeveloper(será una T3):

```
CREATE OR REPLACE PROCEDURE PROBARMIT2 AS
BEGIN
    INSERT INTO Compras VALUES ('00000005', '50000400',1,
0501,'tienda1',50);
    INSERT INTO Compras VALUES ('00000005', '50000030',1,
0501,'tienda1',5);
    INSERT INTO Compras VALUES ('00000005', '50000400',2,
0502,'tienda1',500);
    TRABAJANDO_T2(5);
    INSERT INTO Compras VALUES ('00000005', '50000400',1,
0501,'tienda1',50);
    INSERT INTO Compras VALUES ('00000005', '50000030',1,
0501,'tienda1',5);
    INSERT INTO Compras VALUES ('00000005', '50000400',2,
0502,'tienda1',500);
    TRABAJANDO_T2(5);
END PROBARMIT2;
```

```
CREATE SEQUENCE sec_T2
START WITH 0
INCREMENT BY 1
MINVALUE 0
MAXVALUE 1
CYCLE NOCACHE;
```

- Alterna las ordenes de continuar de la T1 y de la T2 hasta que terminen ambas.
- Consulta qué filas de la tabla ve cada T antes de confirmar.
- Haz un *commit* a mano en la 1ª T.
- Comprueba ahora qué ve la 2ª de la tabla.
- Haz un *commit* a mano en la 2ª T.
- Comprueba ahora si en la tabla de pedidos están las filas esperadas.

Muestra de la tabla sin modificar:

```
SET SERVEROUTPUT ON size 1000000;
SELECT * FROM COMPRAS;
```

Resultado de la Consulta x

Todas las Filas Recuperadas: 12 en 0,012 segundos

DNI	NUMT	NUMF	FECHA	TIENDA	IMPORTE
1 00000003	30000002	200	501	sesion2:una	1
2 00000005	50000400	1	501	tienda1	50
3 00000005	50000030	1	501	tienda1	5
4 00000005	50000400	2	502	tienda1	500
5 00000005	50000400	3	501	tienda2	5000
6 00000005	50000003	1	501	tienda8	50000
7 00000003	30000002	1	501	tienda7	3
8 00000003	30000300	1	501	tienda7	30
9 00000003	30000020	1	501	tienda7	300
10 00000003	30000020	2	501	tienda7	3000
11 00000003	30000020	3	501	tienda8	30000
12 00000004	40000200	1	501	tienda7	4

Ejecutamos SEC_T1.NEXTVALUE FROM DUAL;

```
SET SERVEROUTPUT ON size 1000000;
SELECT SEC_T.NEXTVAL FROM DUAL;
SELECT * FROM COMPRAS;
```

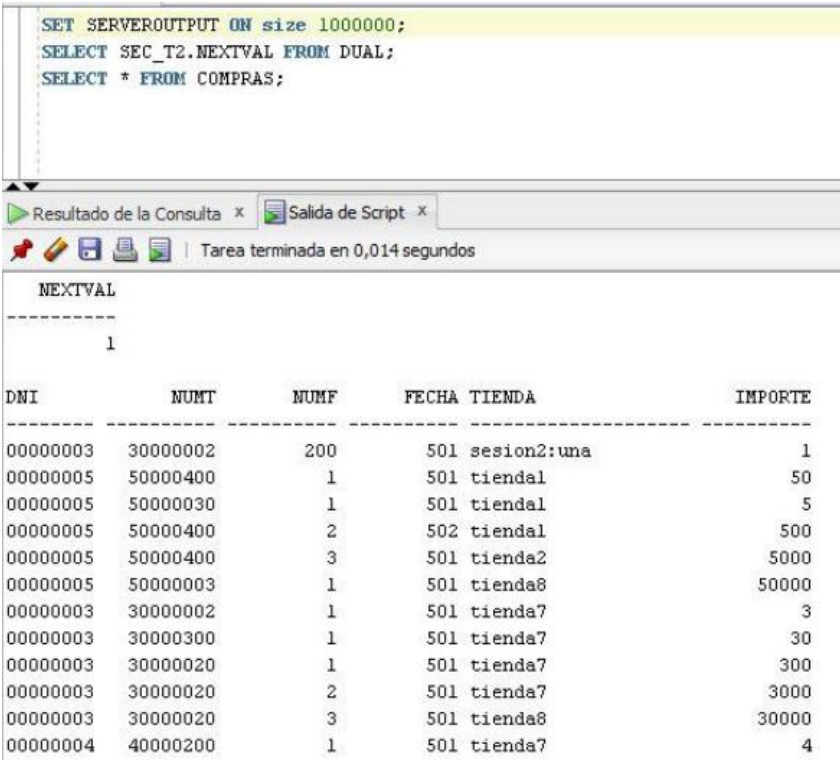
Resultado de la Consulta x Salida de Script x

Tarea terminada en 0,027 segundos

NEXTVAL
1

DNI	NUMT	NUMF	FECHA	TIENDA	IMPORTE
00000003	30000002	200	501	sesion2:una	1
00000005	50000400	1	501	tienda1	50
00000005	50000030	1	501	tienda1	5
00000005	50000400	2	502	tienda1	500
00000005	50000400	3	501	tienda2	5000
00000005	50000003	1	501	tienda8	50000
00000003	30000002	1	501	tienda7	3
00000003	30000300	1	501	tienda7	30
00000003	30000020	1	501	tienda7	300
00000003	30000020	2	501	tienda7	3000
00000003	30000020	3	501	tienda8	30000
00000004	40000200	1	501	tienda7	4

Ejecutamos SEC_T2.NEXTVALUE FROM DUAL;



```
SET SERVEROUTPUT ON size 1000000;
SELECT SEC_T2.NEXTVAL FROM DUAL;
SELECT * FROM COMPRAS;
```

Resultado de la Consulta x Salida de Script x

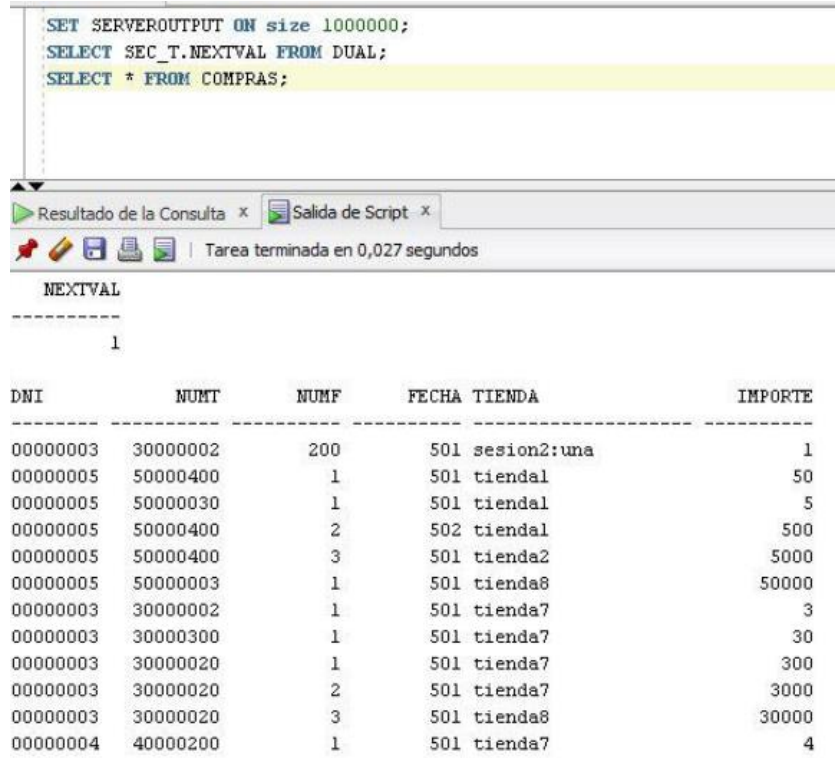
Tarea terminada en 0,014 segundos

NEXTVAL

1

DNI	NUMT	NUMF	FECHA TIENDA	IMPORTE
00000003	30000002	200	501 sesion2:una	1
00000005	50000400	1	501 tienda1	50
00000005	50000030	1	501 tienda1	5
00000005	50000400	2	502 tienda1	500
00000005	50000400	3	501 tienda2	5000
00000005	50000003	1	501 tienda8	50000
00000003	30000002	1	501 tienda7	3
00000003	30000300	1	501 tienda7	30
00000003	30000020	1	501 tienda7	300
00000003	30000020	2	501 tienda7	3000
00000003	30000020	3	501 tienda8	30000
00000004	40000200	1	501 tienda7	4

Ejecutamos SEC_T1.NEXTVALUE FROM DUAL;



```
SET SERVEROUTPUT ON size 1000000;
SELECT SEC_T.NEXTVAL FROM DUAL;
SELECT * FROM COMPRAS;
```

Resultado de la Consulta x Salida de Script x

Tarea terminada en 0,027 segundos

NEXTVAL

1

DNI	NUMT	NUMF	FECHA TIENDA	IMPORTE
00000003	30000002	200	501 sesion2:una	1
00000005	50000400	1	501 tienda1	50
00000005	50000030	1	501 tienda1	5
00000005	50000400	2	502 tienda1	500
00000005	50000400	3	501 tienda2	5000
00000005	50000003	1	501 tienda8	50000
00000003	30000002	1	501 tienda7	3
00000003	30000300	1	501 tienda7	30
00000003	30000020	1	501 tienda7	300
00000003	30000020	2	501 tienda7	3000
00000003	30000020	3	501 tienda8	30000
00000004	40000200	1	501 tienda7	4

Ejecutamos SEC_T2.NEXTVALUE FROM DUAL;

SET SERVEROUTPUT ON size 1000000;
SELECT SEC_T2.NEXTVAL FROM DUAL;
SELECT * FROM COMPRAS;

Resultado de la Consulta x Salida de Script x

Tarea terminada en 0,014 segundos

NEXTVAL					

1					
DNI	NUMT	NUMF	FECHA	TIENDA	IMPORTE

00000003	30000002	200	501	sesion2:una	1
00000005	50000400	1	501	tienda1	50
00000005	50000030	1	501	tienda1	5
00000005	50000400	2	502	tienda1	500
00000005	50000400	3	501	tienda2	5000
00000005	50000003	1	501	tienda8	50000
00000003	30000002	1	501	tienda7	3
00000003	30000300	1	501	tienda7	30
00000003	30000020	1	501	tienda7	300
00000003	30000020	2	501	tienda7	3000
00000003	30000020	3	501	tienda8	30000
00000004	40000200	1	501	tienda7	4

Hacemos COMMIT en la 1ª

SET SERVEROUTPUT ON size 1000000;
SELECT * FROM COMPRAS;

Salida de Script x Resultado de la Consulta x

Todas las Filas Recuperadas: 18 en 0,002 segundos

DNI	NUMT	NUMF	FECHA	TIENDA	IMPORTE
1 00000003	30000002	200	501	sesion2:una	1
2 00000005	50000030	20	501	tienda1	5
3 00000005	50000400	30	502	tienda1	500
4 00000005	50000400	40	501	tienda1	50
5 00000005	50000030	50	501	tienda1	5
6 00000005	50000400	60	502	tienda1	500
7 00000005	50000400	10	501	tienda1	50
8 00000005	50000400	1	501	tienda1	50
9 00000005	50000030	1	501	tienda1	5
10 00000005	50000400	2	502	tienda1	500
11 00000005	50000400	3	501	tienda2	5000
12 00000005	50000003	1	501	tienda8	50000
13 00000003	30000002	1	501	tienda7	3
14 00000003	30000300	1	501	tienda7	30
15 00000003	30000020	1	501	tienda7	300
16 00000003	30000020	2	501	tienda7	3000
17 00000003	30000020	3	501	tienda8	30000
18 00000004	40000200	1	501	tienda7	4

Hacemos COMMIT en la 2ª

```
SET SERVEROUTPUT ON size 1000000;
SELECT * FROM COMPRAS;
```

	DNI	NUMT	NUMF	FECHA	TIENDA	IMPORTE
1	00000003	30000002	200	501	sesion2:una	1
2	00000005	50000030	20	501	tienda1	5
3	00000005	50000400	30	502	tienda1	500
4	00000005	50000400	40	501	tienda1	50
5	00000005	50000030	50	501	tienda1	5
6	00000005	50000400	60	502	tienda1	500
7	00000005	50000400	10	501	tienda1	50
8	00000005	50000400	70	501	tienda1	50
9	00000005	50000030	80	501	tienda1	5
10	00000005	50000400	90	502	tienda1	500
11	00000005	50000400	100	501	tienda1	50
12	00000005	50000030	110	501	tienda1	5
13	00000005	50000400	120	502	tienda1	500
14	00000005	50000400	1	501	tienda1	50
15	00000005	50000030	1	501	tienda1	5
16	00000005	50000400	2	502	tienda1	500
17	00000005	50000400	3	501	tienda2	5000
18	00000005	50000003	1	501	tienda8	50000
19	00000003	30000002	1	501	tienda7	3
20	00000003	30000300	1	501	tienda7	30
21	00000003	30000020	1	501	tienda7	300
22	00000003	30000020	2	501	tienda7	3000
23	00000003	30000020	3	501	tienda8	30000
24	00000004	40000200	1	501	tienda7	4

d) Repetir el mismo experimento que c), poniendo el nivel de Aislamiento Secuenciable en los dos procedimientos *probarMiT1* y *probarMiT2*. Debe haber diferencias, indica cuáles has encontrado.

e) Hacer un trigger que, cada vez que hagamos un insert en la tabla COMPRAS se active para almacenar los datos de la inserción en una tabla LOGcompra, con los mismos atributos de compras, más un atributo número secuencial que será la clave (obtenido de una secuencia sec_log).

```
CREATE OR REPLACE TRIGGER TRIGGER1
AFTER INSERT ON COMPRAS
FOR EACH ROW
BEGIN
    INSERT INTO LOGcompras VALUES (:NEW.DNI, :NEW.NumT,
:NEW.NumF, :NEW.Fecha, :NEW.Tienda, :NEW.Importe,
:SEC_LOG.NEXTVAL);
END;
```


APARTADO 2. Rollback, checkpoints, bloqueos, transacciones autónomas

a) Para entender cómo funcionan las transacciones autónomas

Ejecuta el procedimiento TRAB_T_1_LINEA_AUTONOMA.sql (instrucciones ejecución al final de ese fichero) que llama al procemiento PONE_LINEA_AUTONOMA.sql

- ¿Qué números de transacción da?

The image consists of two side-by-side screenshots of the Oracle SQL Developer interface, both connected to the 'SRV tania05_UCM - db ABDs' database.

Left Screenshot:

- Hoja de Trabajo:** Contains a SQL script with the following code:

```
begin
  trab_t_1_linea_autonoma (5);
end;
```
- Salida de Script:** Shows the execution results of the anonymous block. It indicates the transaction started at 7.16.9893 and ended at 7.16.9893. A detailed log follows:

```
Trans. Principal Empieza: 7.16.9893
se ha dormido -> antes: 0 despues: 0 Num.Trans.Secun: 8.3.10216
se ha dormido -> antes: 1 despues: 1 Num.Trans.Secun: 9.1.10109
se ha dormido -> antes: 0 despues: 0 Num.Trans.Secun: 10.13.9940
se ha dormido -> antes: 1 despues: 1 Num.Trans.Secun: 1.0.9777
se ha dormido -> antes: 0 despues: 0 Num.Trans.Secun: 2.25.9836
se ha dormido -> antes: 1 despues: 1 Num.Trans.Secun: 3.24.10055
se ha dormido -> antes: 0 despues: 0 Num.Trans.Secun: 6.27.10138
se ha dormido -> antes: 1 despues: 1 Num.Trans.Secun: 5.28.10151
se ha dormido -> antes: 0 despues: 0 Num.Trans.Secun: 9.25.10112
Trans. Principal TERMINA: 7.16.9893
```
- Mensajes - Log:** Empty.

Right Screenshot:

- Hoja de Trabajo:** Contains a SQL query:

```
SELECT sec_trans_1.NEXTVAL FROM
```
- Resultado de la Consulta:** Shows a single row with the value '1' in the 'NEXTVAL' column.
- Mensajes - Log:** Empty.

- ¿Por qué hay tantas transacciones?

b) Crea una tabla *comprasAcumuladas* (dni, totalcompra) para acumular en cada fila el total de los importes de las compras de un cliente. Rellena la tabla *comprasAcumuladas* con los valores correctos con una instrucción del tipo

```
INSERT INTO comprasAcumuladas (column1, "column2", ...)
SELECT column3, column4, ...
FROM COMPRAS where ... group by ...;
```

```
CREATE TABLE comprasAcumuladas (
    DNI INTEGER,
    totalCompra INTEGER,
    PRIMARY KEY (DNI)
);
```

```
INSERT INTO comprasAcumuladas (DNI, totalcompra)
SELECT compras.DNI, sum(compras.importe)
FROM COMPRAS
GROUP BY compras.dni;
```

c) Crea un procedimiento *actualizaTotal(dni, importe)*, que recibe por parámetro el dni y el importe de una compra. Si existe una fila con ese dni en la tabla *comprasAcumuladas*, entonces suma el importe al totalcompra de ese dni. Y si no existe crea una fila con los datos recibidos.

```
create or replace
PROCEDURE actualizaTotal (DNInew number, Importe number) AS
    var_dni number;
BEGIN

SELECT COUNT(comprasacumuladas.dni) INTO var_dni
FROM comprasacumuladas
WHERE comprasacumuladas.dni = DNInew;

IF (var_dni > 0) THEN

UPDATE comprasacumuladas
SET totalcompra = totalcompra + Importe
WHERE dni = DNInew;

ELSE

INSERT INTO comprasacumuladas VALUES (DNInew, Importe);

END IF;

END actualizaTotal;
```

d) Crea un procedimiento *prac22_d* para que haga lo siguiente:
inicializa estas variables: contador = 0, TOT = 0 y fija = 10.000

Bucle
Suma 1 a contador
TOT = fija * contador
Inserta en compras una compra por valor de 10.000 para un cliente (el mismo en todo el bucle)
Acumula la cantidad de la compra en tabla *comprasAcumuladas* llamando al proc. *actualizaTotal*.
Si TOT es mayor que 50.000 se sale del bucle
Fin Bucle
Imprime en qué transacción está
hacer un rollback
Vuelve a Imprimir en qué transacción está
¿Puede que no esté en ninguna transacción, ¿por qué?
Fin Procedimiento
¿Qué contiene Compras y ComprasAcumuladas? ¿por qué?
Vacías por el ROLLBACK

```
create or replace
PROCEDURE prac22_d AS
    contador number := 0;
    TOT number := 0;
    fija number := 10000;
    numF number := 0;
BEGIN
    LOOP EXIT WHEN TOT > 50000;
        contador := contador + 1;
        TOT := fija * contador;

        INSERT INTO compras VALUES('00000006', 50000400, numF,
501, 'tienda8', 10000);

        numF := numF + 1;
        actualizatotal(00000006, 10000);

    END LOOP;

    pone_linea_autonoma('Transaccion');
    rollback;
    pone_linea_autonoma('Transaccion despues del rollback');

END prac22_d;
```

Transaccion Num.Trans.Secun: 9.23.10104

Transaccion despues del rollback Num.Trans.Secun: 7.3.9895

e) Crea un procedimiento *prac22_e* para que haga lo siguiente: (sin bucle ahora)
 Inserta en compras una compra por valor de 10.000 para un cliente
 Incluye un savepoint llamado *ch_Punto1*
 Acumula la cantidad de la compra en tabla *comprasAcumuladas* llamando al proc. *actualizaTotal*.
 Imprime en qué transacción está
 hacer un rollback al *ch_Punto1*
 Vuelve a Imprimir en qué transacción está
 ahora sigue en la misma, ¿por qué?
 Haz commit
 Imprime qué valor tiene el totalcompra de ese dni en la tabla *comprasAcumuladas*
 debe tener el valor de antes de ejecutar el procedimiento ¿por qué?
 Comprueba si existe la fila insertada en tabla LOGcompra por el trigger.
 no debe existir, ¿por qué?
 Fin Procedimiento

```
CREATE OR REPLACE PROCEDURE prac22_e AS
  tCompra number;
  tDNI number := 0;
BEGIN

  INSERT INTO compras VALUES('00000006', 50000400, 4, 501,
'tienda8', 10000);
  SAVEPOINT ch_Punto1;
  actualizatotal(00000006, 10000);
  pone_linea_autonoma('Transaccion');
  ROLLBACK TO SAVEPOINT ch_Punto1;
  pone_linea_autonoma('Transaccion despues del rollback');
  COMMIT;

  SELECT totalcompra INTO tCompra
  FROM comprasacumuladas
  WHERE DNI = 00000006;
  DBMS_OUTPUT.PUT_LINE(tCompra);

  SELECT count(DNI) INTO tDNI
  FROM LOGcompras
  WHERE DNI = 00000006;

  IF (tDNI > 0) THEN
    DBMS_OUTPUT.PUT_LINE('Existe en LogCompras');
  END IF;

END prac22_e;
```

f) Modifica lo necesario en prac22_e (renómbralo como prac22_f) y en el *actualizaTotal* para que cumplan las siguientes Restricciones

- En la tabla LOGcompra se quedan todas las compras registradas, *aunque* la transacción principal aborte o haga vuelta atrás (rollback): Debes modificar el trigger.

```
CREATE OR REPLACE TRIGGER TRIGGER1
AFTER INSERT ON COMPRAS
FOR EACH ROW
BEGIN
    INSERT INTO LOGcompras VALUES (:NEW.DNI, :NEW.NumT,
:NEW.NumF, :NEW.Fecha, :NEW.Tienda, :NEW.Importe,
:SEC_LOG.NEXTVAL);
ROLLBACK;
END;
```

- En la tabla *comprasAcumuladas* queremos el total de compras reales. Es decir, si aborta o rollback la transacción principal, queremos que se deshaga la actualización de la compra a la que se ha hecho rollback.

```
create or replace
PROCEDURE prac22_e AS
    contador number := 0;
    TOT number := 0;
    fija number := 10000;
    numF number := 0;
BEGIN
    LOOP EXIT WHEN TOT > 50000;
        contador := contador + 1;
        TOT := fija * contador;

        INSERT INTO compras VALUES('00000006', 50000400, numF,
501, 'tienda8', 10000);

        numF := numF + 1;
        actualizatotal(00000006, 10000);

    END LOOP;

    SAVEPOINT salvaUno;
    pone_linea_autonoma('Transaccion');
    rollback TO SAVEPOINT salvaUno;
    pone_linea_autonoma('Transaccion despues del rollback');

END prac22_e;
```

Prueba todas las situaciones posibles (terminen bien todos los procedimientos y trigger, o rollback de cada uno) Quizá necesites hacer captura de excepciones para dar mensajes de lo que pasa y poner rollbacks para las pruebas.

g) Trabajar con Bloqueos EXCLUSIVOS explícitos

Modifica el `prac22_f` (renómbralo como `prac22_g`) para probar desde dos transacciones el efecto de poner un bloqueo exclusivo a la tabla *compras* antes de insertar la compra. Esto requiere, además de poner el bloqueo exclusivo, el parar una transacción mientras ejecutas la otra, puedes copiar del procedimiento `TRAB_T_1_LINEA_AUTONOMA.sql` lo necesario (es como el apartado 1 de esta práctica, puedes usar tu versión si te funciona).

```
create or replace
PROCEDURE prac22_f AS
    contador number := 0;
    TOT number := 0;
    fija number := 10000;
    numF number := 0;
BEGIN
    LOOP EXIT WHEN TOT > 50000;
        contador := contador + 1;
        TOT := fija * contador;

        INSERT INTO compras VALUES('000000006', 50000400, numF,
501, 'tienda8', 10000);

        numF := numF + 1;
        actualizatotal(000000006, 10000);

    END LOOP;

    LOCK TABLE compras IN EXCLUSIVE MODE [NOWAIT] ;
    pone_linea_autonoma('Transaccion');
    rollback;
    pone_linea_autonoma('Transaccion despues del rollback');

END prac22_f;
```

h) Trabajar con Bloqueos LECTURA COMPARTIDA explícitos

Modifica el prac22_g (renómbralo como prac22_h) para probar desde dos transacciones el efecto de poner un bloqueo Lectura Concurrente a la tabla *compras* antes de insertar la compra. Prueba con dos transacciones concurrentes.

¿Qué sucede? ¿porqué?

Describe una situación donde necesitas poner el bloqueo de lectura compartida

```
create or replace
PROCEDURE prac22_h AS
    contador number := 0;
    TOT number := 0;
    fija number := 10000;
    numF number := 0;
BEGIN
    LOOP EXIT WHEN TOT > 50000;
        contador := contador + 1;
        TOT := fija * contador;

        INSERT INTO compras VALUES('000000006', 50000400, numF,
501, 'tienda8', 10000);

        numF := numF + 1;
        actualizatotal(000000006, 10000);

    END LOOP;

    LOCK TABLE compras IN SHARE MODE [NOWAIT] ;
    pone_linea_autonoma('Transaccion');
    rollback;
    pone_linea_autonoma('Transaccion despues del rollback');

END prac22_h;
```

i) Describe las características que provoquen que en una transacción deba ser serializable.

- T's no pierden actualizaciones.
- Se garantizan lecturas repetibles (y L.C.)
- No hay registros fantasma y tampoco resumen incorrecto.
- Porque las modificaciones hechas por T solo la ve T, mientras T no haga COMMIT.
- Si Ti actualiza algo que ya actualizó Tjy Tjestá sin confirmar, entonces Tiaborta.
- Vemos luego qué es un plan secuenciable.

j) Describe las características que provoquen que en una transacción deba ser read committed.

- Lectura Consistente. Y la T no tiene lecturas repetibles.
- Modificaciones hechas por T y otras T's son visibles por T y por otras T's,
 - Solo si las otras han hecho commit.
- Si Ti tiene DML que necesita bloquear filas que tiene otra T, la Ti espera.