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 procedimento *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.

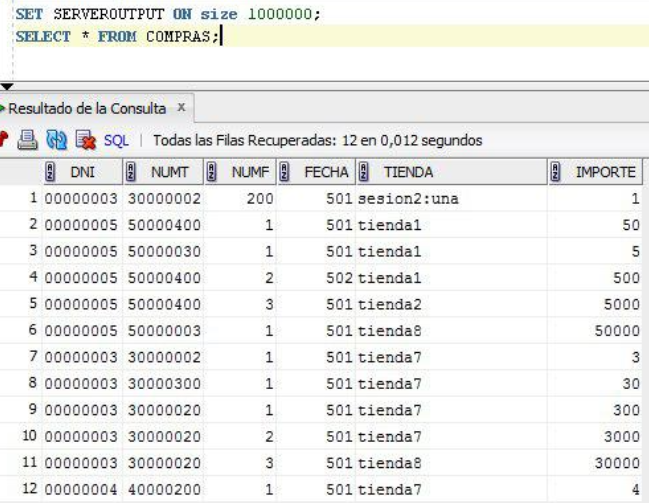
- Haz un *commit* a mano en la 1ª T.

- Comprueba ahora qué ve la 2ª de la tabla.

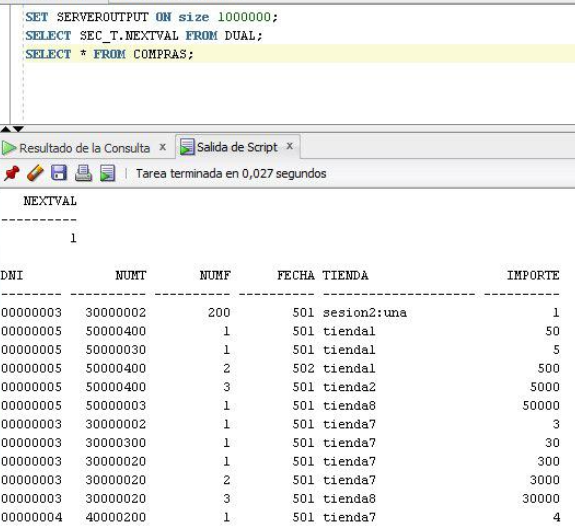
- Haz un *commit* a mano en la 2ª T.

- Comprueba ahora si en la tabla de pedidos están las filas esperadas.

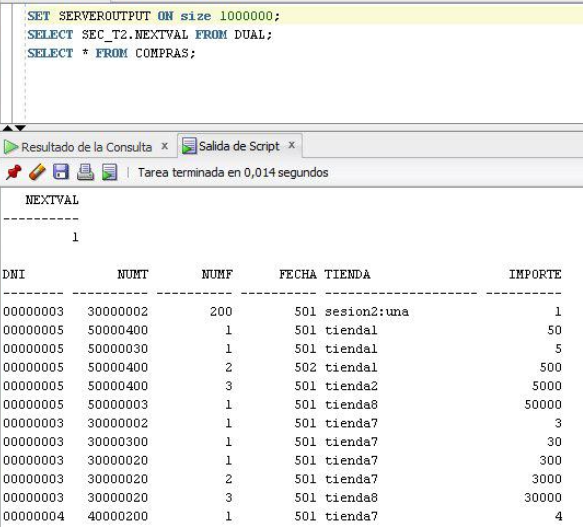
Muestra de la tabla sin modificar:



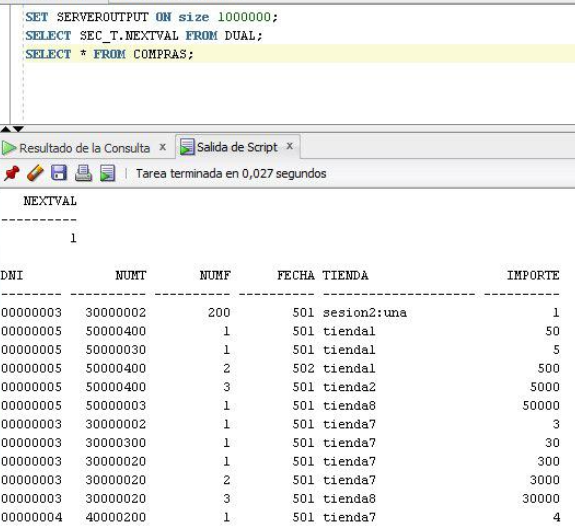
Ejecutamos SEC\_T1.NEXTVALUE FROM DUAL;



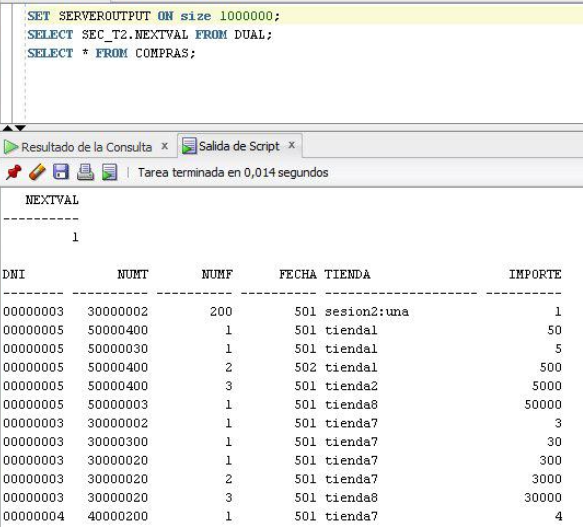
Ejecutamos SEC\_T2.NEXTVALUE FROM DUAL;



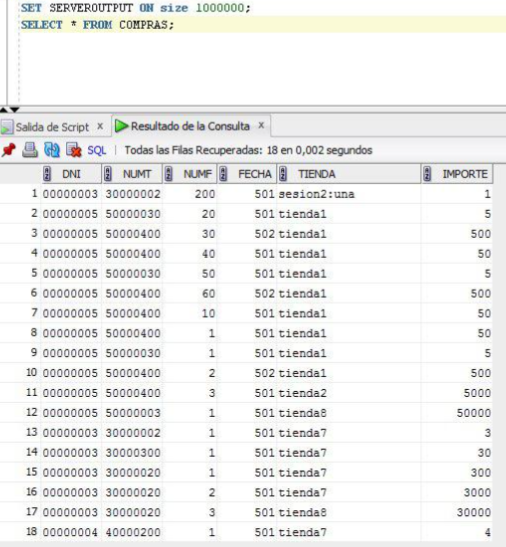
Ejecutamos SEC\_T1.NEXTVALUE FROM DUAL;



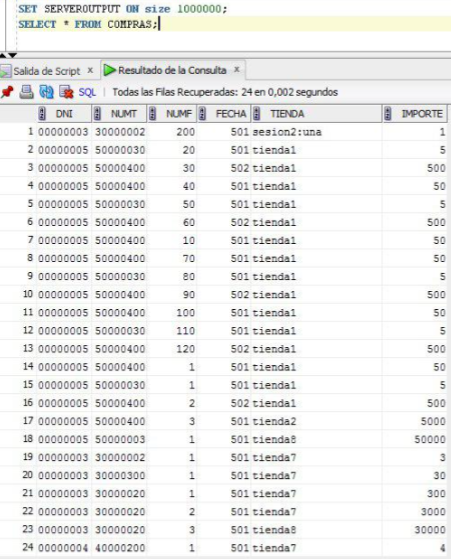
Ejecutamos SEC\_T2.NEXTVALUE FROM DUAL;



Hacemos COMMIT en la 1ª



Hacemos COMMIT en la 2ª



d) Repetir el mismo experimento que c), poniendo el nivel de Aislamiento Secuenciable en los dos procedimentos *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);

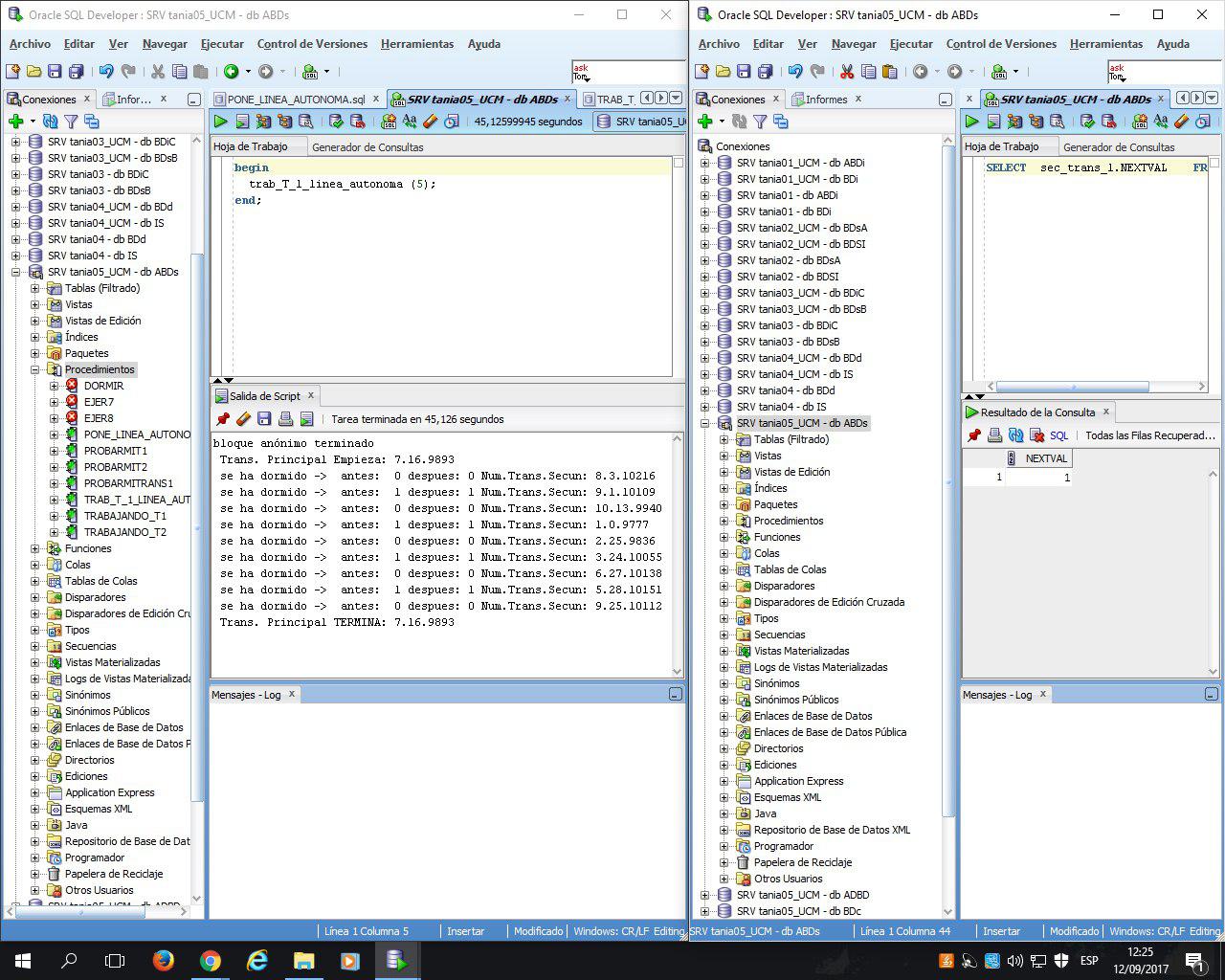
END;

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?



- ¿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 varibles: 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('00000006', 50000400, numF, 501, 'tienda8', 10000);

numF := numF + 1;

actualizatotal(00000006, 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('00000006', 50000400, numF, 501, 'tienda8', 10000);

numF := numF + 1;

actualizatotal(00000006, 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 nohaga 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 commited.

- 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.