Transacciones

¡Comencemos!

¿Qué aprenderás?

- Identificar qué son las transacciones y qué utilidad nos genera al momento de gestionar una base de datos.
- Integrar las transacciones en sentencias SQL para la realización por lote de consultas.
- Cargar una base de datos utilizando dump.

Introducción

Hasta el momento hemos creado un par de tablas en nuestra base de datos, sus registros se han cargado de manera manual y a través del formato .csv, con los que realizamos distintos tipos de consultas para obtener información de una forma personalizada. Sin embargo, te has preguntado: ¿Qué pasa si al momento de realizar una consulta ya sea para recuperar información, crear nuevos registros, entre otros, surge algún inconveniente y obtenemos un error? La respuesta rápida y sencilla, es que no se realiza el cambio o consulta que estamos procesando y en muchos casos eso puede ser un problema. Para darle una solución a este inconveniente surgen las transacciones en Oracle Database, las cuales sirven como simuladores de consultas que nos permiten procesar sentencias SQL sin perjudicar de forma permanente la base de datos a menos que le indiquemos que así sea.

Las transacciones son usadas a menudo para hacer pruebas en una base de datos cuya información es de suma importancia y está enlazada a un producto que ya fue lanzado a producción, no obstante, tienen un uso aún más práctico relacionado con procedimientos que requieren de varias consultas y que se necesita que el 100% de estas sean realizadas con éxito, de lo contrario no se realice nada para no perjudicar la consistencia de la base de datos.

La lógica de las transacciones así como lo indica su nombre son utilizadas en la programación de software en instituciones tan importantes como son los bancos, los cuales realizan sus transacciones bancarias basadas en un todo o nada, pues no tendría sentido que se descuente el dinero de una cuenta A y no se sume a una cuenta B, por esto y mucho más, es importante que aprendas a usar las transacciones en SQL y es lo que verás en este capítulo.

Transacciones

Las transacciones son secuencias de instrucciones ordenadas, las cuáles pueden ser indicadas de forma manual o pueden ser aplicadas automáticamente. Estas realizan cambios en las bases de datos a la hora de aplicar comandos de manipulación de columnas y registros, su importancia recae en el control que nos ofrece sobre los cambios permanentes en la base de datos.

Propiedades de las transacciones:

- **Atomicidad:** Todas las operaciones realizadas en la transacción deben ser completadas. En el caso que ocurra un fallo, esta transacción es abortada y devuelve todo al estado previo a la transacción.
- **Consistencia:** La base de datos cambiará solamente cuando la transacción se haya realizado.
- **Aislamiento:** Las transacciones pueden ocurrir independientes una u otra.
- **Durabilidad:** El resultado de la transacción persiste a pesar de que el sistema falle.

Una transacción empaqueta varios pasos en una operación, de forma que se completen todos o ninguno, cuidando la integridad de la información, de la siguiente manera:

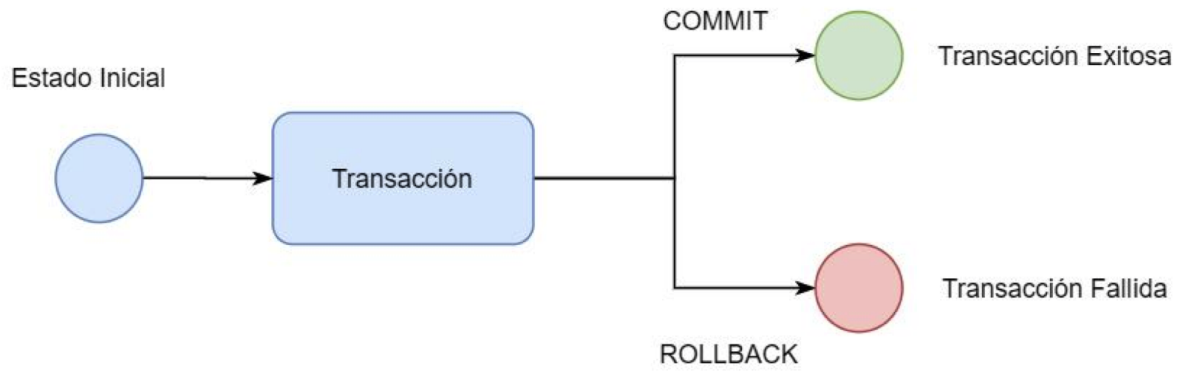

Imagen 1. Diagrama de flujo de las transacciones.

Automáticamente algunas bases de datos, están configuradas para que se ejecuten estas transacciones sin manipular este flujo, sin embargo, no es así en oracle 11g, ya que nuestras transacciones se ejecutan en nuestro usuario hasta que cerremos la sesión.

Un ejemplo de esta situación, es cuando ingresamos un registro en Sql plus, no veremos este nuevo campo ingresado en ninguna otra aplicación hasta que hagamos el commit. Hibernate, el cual es muy utilizado en aplicaciones con Spring, viene con el auto commit por defecto. Para realizar estas tareas de confirmación existen los siguientes comandos:

Comando	Descripción
BEGIN	El sistema permite que se ejecuten todas las sentencias SQL que necesitemos.
COMMIT	Guarda los cambios de la transacción.
ROLLBACK	Retrocede los cambios realizados.
SAVEPOINT	Guarda el punto de partida al cual volver a la hora de aplicar ROLLBACK.
SET TRANSACTION	Le asigna nombre a la transacción.

Tabla 1. Comandos usados en las transacciones

Estos comandos sólo pueden ser usados con las operaciones INSERT, UPDATE y DELETE, ya que son aquellos que manipulan toda la tabla y hacen este proceso automáticamente.

La sintaxis de estos comandos es la siguiente:

- COMMIT;
- SAVEPOINT nombre_savepoint;
- ROLLBACK [TO nombre_savepoint];

Lo que está entre corchetes es de carácter opcional, por lo que podemos decirle con ROLLBACK a qué punto volver. Este volverá al último punto guardado o por defecto al estado inicial de la transacción.

En el siguiente código te muestro la sintaxis que puedes ocupar para iniciar una transacción.

```
SET TRANSACTION [READ ONLY|WRITE][NAME nombre_transaccion];
```

Como puedes notar, existen varias opciones, podemos usar READ ONLY para solamente leer la base de datos, READ WRITE para leer y escribir sobre ella, y poder nombrar la transacción con el comando NAME.

Ejemplo de una transacción

En el siguiente ejemplo, usamos la tabla autos, que trabajamos en guías anteriores.

Es importante iniciar sesión con el usuario que se ha creado, no con system. Adicionalmente, la sentencia transaction, está más orientada a la ejecución de bloques, por lo que es más aplicable a PLSQL (una forma de programación estructurada), los puntos importantes o lo que se puede destacar de esta característica es la capacidad de volver atrás en caso de un borrado ingreso o actualización no deseada. Es una forma de trabajo defensivo.

En algunos motores una transacción comienza con un Begin Transaction, no es así para el caso de Oracle, para que comience una transacción, **debe anteriormente existir un commit**, con ello, la transacción se activa al momento de realizar la primera modificación en una tabla.

Hemos hablado y de la palabra reservada commit, además la veremos aplicada al siguiente ejemplo. Por ahora sepamos que un commit marca un estado, como un check point, en el siguiente punto de esta lectura, detallaremos más profundamente sobre esta palabra reservada.

```
-- seleccionamos el auto que queremos modificar
SELECT * FROM AUTOS WHERE id = 5;

ID|MARCA  |MODELO|ANIO|COLOR |PRECIO  |
--|-------|------|----|------|--------|
 5|ferrari|Sport |2002|Blanco|50000000|
-- primer commit para que inicie la transacción.
COMMIT;
-- Ingresamos la transacción.
SET TRANSACTION NAME 'car_update';
```

```
-- output: Transaction NAME correcto.

-- actualizamos el registro a 2003
UPDATE AUTOS SET ANIO = 2003 WHERE ID = 5;
-- output: 1 fila actualizadas.
-- creamos el punto de guardado
SAVEPOINT after_car_update;
-- output: Savepoint creado.
-- actualizamos el registro a 2003
UPDATE AUTOS SET ANIO = 2020 WHERE ID = 5;
-- output: 1 fila actualizadas.

-- vemos el cambio
SELECT * FROM AUTOS where id = 5;
ID|MARCA  |MODELO|ANIO|COLOR |PRECIO  |
--|-------|------|----|------|--------|
 5|ferrari|Sport |2020|Blanco|50000000|

-- volvemos al punto guardado, antes de la actualización.
ROLLBACK TO SAVEPOINT after_car_update;
-- 1 fila actualizadas.
-- Rollback terminado.

-- vemos que tenemos el estado desde el punto de guardado.
SELECT * FROM AUTOS where id = 5;

ID|MARCA  |MODELO|ANIO|COLOR |PRECIO  |
--|-------|------|----|------|--------|
 5|ferrari|Sport |2003|Blanco|50000000|
commit;
```

Vemos que gracias a instrucciones como `ROLLBACK` podemos volver a un estado anterior de consultas que se han ejecutado. Lo que podría significar por ejemplo, que todos los clientes de un banco, no tengan sus cuentas en cero por una mala instrucción ejecutada por error.

COMMIT

Un ejemplo muy sencillo de la utilidad de las transacciones, es pensar en el funcionamiento de las cuentas de los bancos. Al hacer una transferencia, por ejemplo, ¿Cómo nos aseguramos que el dinero que se resta de una cuenta, se sume en la siguiente? ¿Cómo controlamos que efectivamente se mantenga la integridad de los datos en caso de fallas?

Las transacciones nos permiten esto. Pensemos en la siguiente tabla:

```
CREATE TABLE cuentas
  (
     numero_cuenta NUMBER NOT NULL PRIMARY KEY,
     balance       NUMBER(11, 2) CHECK (balance >= 0.00)
  -- check valida la condición que el monto sea mayor a cero
  );
```

Nota la palabra reservada “CHECK” escrita luego de definir el tipo de dato para la propiedad “balance”, esto setea una restricción a esa columna que devolverá un error cuando intente actualizar o ingresar un valor que no cumpla con la condición.

Ahora con el siguiente código procedemos con el registro de 2 cuentas con un saldo de $1000 cada una.

```
INSERT INTO cuentas
  (numero_cuenta, balance)
VALUES
  (1, 1000);

INSERT INTO cuentas
  (numero_cuenta, balance)
VALUES
  (2, 1000);
```

```
NUMERO_CUENTA|BALANCE|
-------------|-------|
            1|   1000|
            2|   1000|
```

Ahora si quisiéramos hacer una transferencia de $1000 desde nuestra cuenta 1 a la cuenta 2, una forma de asegurarnos que el monto de nuestro balance disminuya en $1000 y el de la segunda cuenta aumenta en la misma cifra, podría escribirse de la siguiente manera:

```
COMMIT;
SET TRANSACTION NAME 'cuentas_update';
UPDATE cuentas SET balance = balance - 1000 WHERE numero_cuenta = 1;
-- esta consulta se ejecutará sin problemas.
UPDATE cuentas SET balance = balance + 1000 WHERE numero_cuenta = 2;
COMMIT;
```

```
select * from cuentas;

NUMERO_CUENTA|BALANCE|
-------------|-------|
            1|      0|
            2|   2000|
```

Llegando justo a cero, ya no se podrá realizar otra operación, por que sería menos a cero.

ROLLBACK

Con este comando podemos deshacer las transacciones que se hayan ejecutado, revirtiendo los cambios realizados por una transacción hasta el último COMMIT o ROLLBACK ejecutado. Esto permite controlar los flujos de ejecución en nuestras transacciones, de manera que volvamos a un estado anterior, sin alterar los datos almacenados.

Continuando con el ejemplo del banco, en el punto anterior realizamos una transacción en donde transferimos $1000 de la cuenta 1 a la cuenta 2, como verás no obtuvimos ningún error, no obstante podemos verificar que en las transacciones que terminan en error no se altera el estado de nuestros datos. ¿Y cómo lo hacemos? Ejecuta el siguiente código en tu terminal.

```
select * from cuentas;

COMMIT;
SET TRANSACTION NAME 'cuentas_update';
UPDATE cuentas SET balance = balance + 1000 WHERE numero_cuenta = 2;
UPDATE cuentas SET balance = balance - 1000 WHERE numero_cuenta = 1;
select * from cuentas;
ROLLBACK;
```

Todo lo que se ejecute en esta transacción, volverá a su estado inicial de commit. Compruébalo verificando el resultado por ti mismo.

Si por alguna razón, en medio de una transacción se decide que ya no se quiere registrar los cambios (tal vez nos dimos cuenta que estamos actualizando todos los registros de nuestra base y no es lo que buscábamos), **se puede recurrir a la orden ROLLBACK** en lugar de COMMIT y todas las actualizaciones hasta ese punto quedarán canceladas.

SAVEPOINT

Como vimos en un ejemplo anterior, es posible tener un mayor control de las transacciones por medio de puntos de recuperación (**SAVEPOINTS**). Estos permiten seleccionar qué partes de la transacción serán descartadas bajo ciertas condiciones, mientras el resto de las operaciones sí se ejecutan.

Después de definir un punto de recuperación seguido de su nombre representativo, se puede volver a él por medio de ROLLBACK TO. Todos los cambios realizados por la transacción, entre el punto de recuperación y el rollback se descartan.

Probemos esto, con el siguiente código Intentemos registrar una nueva cuenta de número 3 en nuestra tabla “cuentas” con un saldo de $5000 y justo luego guardemos ese punto de la transacción con un SAVEPOINT de nombre “nueva_cuenta”.

```
COMMIT;
SET TRANSACTION NAME 'cuentas_update';
INSERT INTO cuentas(numero_cuenta, balance) VALUES (3, 5000);
SAVEPOINT nueva_cuenta;
```

Esto provocará que te devuelva un error por la terminal puesto que la cuenta 2 no dispone de $3000. ¿Entonces qué pasó? Si consultas la tabla “cuentas” obtendrás lo siguiente.

```
UPDATE cuentas SET balance = balance + 3000 WHERE numero_cuenta = 3;
UPDATE cuentas SET balance = balance - 3000 WHERE numero_cuenta = 2;
-- Justo acá deberás recibir un error
ROLLBACK TO nueva_cuenta;
```

Como puedes notar se registró con éxito la cuenta 3, no obstante su balance sigue siendo $5000 y el balance de la cuenta 2 no se redujo. Esto es porque volvimos al punto guardado, luego de haber hecho la inserción de nuestra nueva cuenta a pesar de no haberse procesado lo que le procedía a esa instrucción.

```
NUMERO_CUENTA|BALANCE|
-------------|-------|
            1|      0|
            2|   2000|
            3|   5000|
```

AUTOCOMMIT

Oracle posee por defecto **autocommit OFF**; pero esto se puede modificar para casos en que quisiéramos que las transacciones se confirmen de inmediato. En resumen el auto-commit **es un modo que realiza COMMIT de manera automática.**

La forma de asignar los valores respectivos al auto-commit son los siguientes:

```
-- para mostrar el estado del autocommit
show autocommit;
-- para activarlo
autocommit off;
-- para desactivarlo
autocommit on;
```

Cargar una base de datos utilizando Data Pump Export

En algunos motores de bases de datos, encontraremos formas simples de exportar o importar estructuras de una base de datos o sus datos. Podemos realizar estas tareas de importar o exportar datos con nuestros clientes gráficos, pero existe una forma de realizar esta tarea mediante comandos por medio de nuestra consola. De esta forma podríamos realizar respaldos fuera de horario, programando tareas.

Podría parecer una tarea engorrosa si la comparamos con otros motores y su dump, pero esto se debe a motivos de seguridad, por ejemplo el asignar un directorio para los respaldos, nos asegura al menos un orden. Adicionalmente la exportación se realiza en archivos especiales para oracle.

Lo primero que debemos hacer, es asignar un directorio en nuestro computador, **en mi caso será '/usr/respaldos'**, que es una carpeta que guarda elementos temporalmente, en su caso sean más creativos, podría ser c:\respaldos\base de datos.

Este directorio debe tener permisos de escritura, lectura y ejecución, con un 775, estaría disponible sin problemas.

Debemos con nuestro usuario administrador ingresamos las siguientes instrucciones:

```
-- creamos el direcctorio
CREATE DIRECTORY mi_directorio AS '/usr/respaldos';
-- asignamos los permisos de escritura y lectura
GRANT READ, WRITE ON DIRECTORY mi_directorio TO mawashicars;
commit;
```

Luego desde la consola donde está el servidor, ejecutamos el siguiente comando para exportar nuestra base, donde `mawashicars/1234`, es el usuario/contraseña.

```
expdp mawashicars/1234 directory=mi_directorio dumpfile=laFechaDeHoy.dmp
```

Podemos indicar que solamente queremos solo los metadatos:

```
expdp mawashicars/1234 directory=mi_directorio dumpfile=laFechaDeHoy.dmp
ontent=METADATA_ONLY
```

Como sabemos en qué lugar estarán nuestros respaldos, es posible realizar la operación inversa (es decir, cargar nuestro respaldo).

```
impdp hr DUMPFILE=mi_directorio:laFechaDeHoy.dmp FULL=YES
LOGFILE=dpump_dir2:full_imp.log
```

Para mayor detalle de estas operaciones, verificar el manual oficial.

Estas operaciones, como fueron mencionadas anteriormente forman parte de la estrategia de respaldo de Oracle, el cual presenta mucha resistencia hasta que valoremos su gran utilidad. Para nuestro contexto, lo mejor será importar estructuras SQL desde la comodidad de los clientes gráficos.

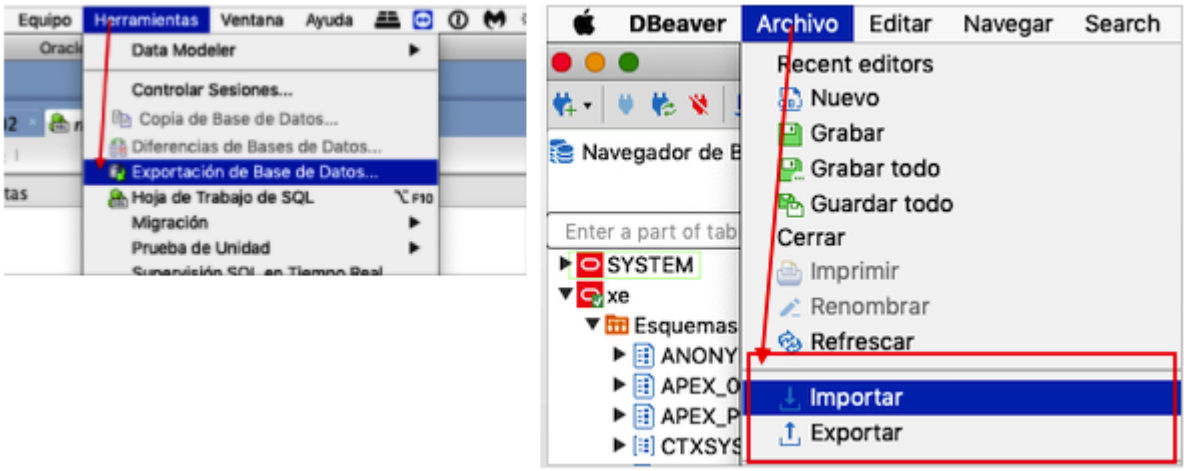

Imagen 2. Tratamiento de carga y exportaciones, desde los clientes Sql Developer y Dbeaver.

Manos al código - Pongamos a prueba los conocimientos

La pizzeria nacional Hot Cheese, ofrece en su sitio web para el servicio de pizzas a domicilio y apesar de estar funcionando bien los primeros meses ha bajado su clientela por incomodidades en sus usuarios, que realizan pagos electrónicos en la aplicación y posteriormente reciben un correo de disculpas por la empresa diciendo que la pizza que compró ya no está disponible. El dueño de la pizzería ha tomado cartas sobre el asunto y ha solicitado contratar a un programador de bases de datos, para que cree una nueva base de datos aplicando las restricciones para evitar que siga sucediendo esta situación.

Para la solución de este ejercicio deberás realizar lo siguiente:

1. Crear una base de datos llamada “pizzeria”.

2. Conectarse a la base de datos pizzeria.

3. Crear 2 tablas llamadas “ventas” y “pizzas” con la siguiente estructura.
    a. ventas
        i. cliente.
        ii. fecha.
        iii. monto.
        iv. pizza(fk).
    b. pizzas
        i. id(PK).
        ii. stock (verificar que no sea menor a cero).
        iii. costo.
        iv. nombre.

4. Agregar 1 registro a la tabla “pizzas” seteando como stock inicial 0.

5. Realizar una transacción que registre una nueva pizza con un stock positivo mayor a 1.

6. Realizar una transacción que registre una venta con la pizza con stock 0 e intentar actualizar su stock restándole 1.

7. Realizar una transacción que intente registrar 1 venta por cada pizza, guardando un SAVEPOINT luego de la primera venta y volviendo a este punto si surge un error.

8. Exportar la base de datos “pizzeria” como un archivo pizzeria.sql.
9. Eliminar la base de datos “pizzeria”.
10. Importar el archivo pizzeria.sql.

Pasos a seguir

Paso 1: Partimos este ejercicio con la creación de nuestra base de datos, cuyo nombre será “pizzeria.”

```
create user pizzeria identified by 1234;
grant all privileges to pizzeria;
```

Paso 2: Procedemos con la conexión a la base de datos “pizzeria” previamente creada.

Iniciamos la conección desde el cliente gráfico o sqlplus con la siguiente instrucción:

```
sqlplus pizzeria/1234
```

Paso 3: Según el planteamiento del problema, necesitamos crear 2 tablas. Una llamada “ventas” y la otra “pizzas” en donde “pizzas” debe tener una restricción en su atributo “stock” y como clave primaria el id, y para el caso de “venta” una clave foránea en el atributo id también, entendiendo que esta depende de la tabla “pizzas”. Para esto ejecutamos las siguientes instrucciones.

```
CREATE TABLE pizzas
  (
     id     NUMBER,
     stock  NUMBER CHECK (stock >= 0.00),
     costo  NUMBER(11, 2),
     nombre VARCHAR2(20),
     PRIMARY KEY(id)
  );

CREATE TABLE venta
  (
     cliente VARCHAR2(20),
     fecha   DATE,
```

```
    monto   NUMBER(11, 2),
    pizza   NUMBER,
    FOREIGN KEY (pizza) REFERENCES pizzas(id)
  );
```

Paso 4: Para proceder con nuestras pruebas necesitamos agregar 1 registro a la tabla "pizzas", sabiendo que probaremos las transacciones en este ejercicio, en los próximos pasos seteamos como stock inicial 0 recordando que este es el atributo con la restricción.

Hasta este punto nuestra tabla "pizzas" va quedando de la siguiente manera.

```
 INSERT INTO pizzas
   (id, stock, costo, nombre)
 VALUES
   (1, 0, 12000, 'Uhlalá');

SELECT * FROM pizzas;

ID|STOCK|COSTO|NOMBRE|
--|-----|-----|------|
 1|    0|12000|Uhlalá|
```

Paso 5: Probemos una primera transacción ingresando una nueva pizza a la tabla "pizzas" y ésta tendrá un stock mayor a 1, por lo que usaremos el siguiente código.

```
INSERT INTO pizzas
  (id, stock, costo, nombre)
VALUES
  (2, 2, 15000, 'Jamón a todo dar');
COMMIT;

SELECT * FROM pizzas;
ID|STOCK|COSTO|NOMBRE          |
--|-----|-----|----------------|
 1|    0|12000|Uhlalá          |
 2|    2|15000|Jamón a todo dar|
```

Paso 6: Es hora de probar nuestra base de datos, por lo que realizaremos una transacción que registre una venta con la pizza con stock 0 e intente actualizar su stock restándole 1. Lo que debería provocar un error.

```
INSERT INTO venta
  (cliente, fecha, monto, pizza)
VALUES
  ('Dexter Gonzalez', to_date('2020-02-02', 'yyyy-mm-dd'), 12000, 1);

UPDATE pizzas SET stock = stock - 1 WHERE id = 1;
COMMIT;
```

Paso 7: Ahora supongamos que se quiere realizar una venta de las 2 pizzas registradas pero solo 1 de ellas tiene stock, por lo que marcaremos un punto de guardado justo después de actualizar el stock de la pizza disponible y volviendo a este punto en el momento que recibimos un error.

```
COMMIT;
SET TRANSACTION NAME 'venta_queries';
INSERT INTO venta
  (cliente, fecha, monto, pizza)
VALUES
  ('Juan Bravo', to_date('2020-02-02', 'yyyy-mm-dd'), 15000, 2);

UPDATE pizzas SET stock = stock - 1 WHERE id = 2;
SAVEPOINT checkpoint;

INSERT INTO venta
  (cliente, fecha, monto, pizza)
VALUES
  ('Utonio Ramirez', to_date('2020-02-02', 'yyyy-mm-dd'), 12000, 1);

UPDATE pizzas SET stock = stock - 1 WHERE id = 1;
-- Acá recibirás un error por intentar rebajar el stock de una pizza
cuyo stock es 0
ROLLBACK checkpoint;
```

Entonces nos vamos al `checkpoint`, volviendo a ese estado.

Paso 8: Supongamos que la pizzería Hot Cheese no le va muy bien y decide guardar un respaldo de sus datos y solicita que se exporte la base de datos como un archivo llamado `pizzeria.dmp`. Para esto realizamos la siguiente instrucción.

```
expdp mawashicars/1234 directory=mi_directorio dumpfile=pizzeria.dmp
```

Paso 9: En el caso que la pizzería cierre sus puertas y solicite eliminar la base de datos "pizzeria" usamos la siguiente sentencia.

```
-- Podemos eliminar el usuario, pero esto significa no recuperarlo para
Oracle
drop user pizzeria cascade;
-- Pero eliminamos el contenido mejor.
DROP TABLE venta;
DROP TABLE pizzas;
```

Paso 10: Y si quisiera aperturar el negocio y recuperar los datos de su último respaldo, deberíamos crear de nuevo la base de datos y posteriormente importar nuestro archivo pizzeria.sql, exportado en el paso 8 tal y como te muestro a continuación.

```
impdp hr DUMPFILE=mi_directorio:pizzeria.dmp FULL=YES
LOGFILE=dpump_dir2:full_imp.log
```

Recuerden que siempre pueden exportar el sql desde el cliente gráfico como se indica en el punto anterior, pero si poseen un servidor configurado para guardar respaldos, entonces respaldamos con los comandos `expdp & impdp.`