

SQL – DML II

Oracle 21c XE

En este capítulo se verá el lenguaje SQL enfocado a la manipulación de datos en una base de datos. Un subconjunto de SQL, el DML (*Data Manipulation Language*) se ocupa de la ejecución de consultas a una base de datos y las operaciones de gestión de datos como inserción borrado y actualización de filas en tablas. Este capítulo se centra en esto último, veremos la actualizaciones de los datos en una base de datos.



SQL DML II - Oracle 21c XE by Rafael Lozano is licensed under a [Creative Commons Reconocimiento-NoComercial-CompartirIgual 3.0 España License](https://creativecommons.org/licenses/by-nc-sa/3.0/es/).

Tabla de contenido

1	Introducción.....	1
2	Actualizaciones de datos.....	2
2.1	Sentencia INSERT.....	2
2.2	Sentencia UPDATE.....	5
2.3	Sentencia DELETE.....	8
2.4	Sentencia TRUNCATE.....	11
3	Carga masiva de datos.....	11
3.1	Exportación / Importación de datos.....	12
3.1.1	Exportar datos.....	12
3.1.2	Importar datos.....	18
3.2	Carga desde archivo CSV.....	19
3.3	SQL Loader.....	23
3.3.1	Fichero de control.....	25
3.3.2	Fichero de parámetros.....	26
3.3.3	Ejecución de SQL Loader.....	27
4	Transacciones.....	27
4.1	Control de transacciones.....	28
5	Bloqueos.....	32
5.1	Bloqueo automático.....	34
5.1.1	Bloqueo DML.....	34
5.1.2	Bloqueo DDL.....	36
5.2	Bloqueo manual.....	37
5.2.1	Sentencia LOCK TABLE.....	37
6	Bibliografía.....	41

SQL – DML II

Oracle 21c EE

1 Introducción

SQL es el lenguaje declarativo de alto nivel con el cual todos las aplicaciones y usuarios acceden a los datos en una BD Oracle. Aunque algunas herramientas y aplicaciones enmascarán el uso de SQL, todas las tareas sobre una base de datos se realizan utilizando SQL. Cualquier otro método de acceso a los datos que elude la seguridad integrada de Oracle y potencialmente compromete la seguridad e integridad de los datos.

SQL suministra una interfaz a una base de datos relacional como Oracle. SQL unifica tareas como las siguientes:

- ✓ Crear, modificar y eliminar objetos.
- ✓ Insertar, actualizar y borrar filas de tablas.
- ✓ Consulta de datos.
- ✓ Controlar el acceso a la base de datos y sus objetos.
- ✓ Garantizar la consistencia e integridad de la base de datos.

SQL puede utilizarse interactivamente, lo que significa que las sentencias se introducen manualmente en un programa. Las sentencias SQL también se pueden escribir dentro de un programa escrito en un lenguaje como C o Java.

Oracle SQL incluye muchas extensiones del estándar SQL ANSI/ISO. Las herramientas y aplicaciones Oracle suministran sentencias adicionales. Las herramientas SQL*Plus, SQL Developer y Oracle Enterprise Manager te permiten ejecutar sentencias estándar ANSI/ISO SQL y cualquier otra sentencia adicional o funciones disponibles para esas herramientas.

Las sentencias SQL se dividen en varias categorías entre las que se encuentran:

- ✓ Lenguaje de definición de datos (DDL) para definir los objetos de la BD.
- ✓ Lenguaje de manipulación de datos (DML) para acceso a los objetos de la BD.
- ✓ Control de transacciones.
- ✓ Control del acceso a datos

El DML (*Data Manipulation Language*) es el conjunto de sentencias que está orientadas a la consulta, y manejo de datos de los objetos creados. El DML es un subconjunto muy pequeño dentro de SQL, pero es el más importante, ya que su conocimiento y manejo con soltura es imprescindible para el acceso a las BD. Básicamente consta de cuatro sentencias: SELECT, INSERT, DELETE, UPDATE.

Las sentencias DML acceden y manipulan los datos existentes en las tablas. En el entorno SQL*Plus podemos introducir una sentencia DML detrás del prompt `SQL>` y el resultado se mostrará a continuación. En el entorno SQL Developer, podemos introducir una sentencia SQL en una hoja de trabajo y veremos el resultado en la parte inferior.

2 Actualizaciones de datos

SQL es un lenguaje completo de manipulación de datos que se utiliza no solamente para consultas, sino también para modificar y actualizar los datos de la base de datos. Comparadas con la complejidad de la sentencia `SELECT`, que soporta las consultas en SQL, las sentencias SQL que modifican los contenidos de la base de datos son extremadamente sencillas. Sin embargo, el SGBD debe proteger la integridad de los datos almacenados durante los cambios, asegurándose que sólo se introduzcan datos válidos y que la base de datos permanezca autoconsistente, incluso en caso de fallos del sistema. El SGBD debe coordinar también las actualizaciones simultáneas por parte de múltiples usuarios, asegurándose que los usuarios y sus modificaciones no interfieran unos con otros. Para modificar los contenidos de una base de datos hay tres sentencias:

- ✓ `INSERT`, que añade nuevas filas de datos a una tabla.
- ✓ `DELETE`, que elimina filas de datos de una tabla.
- ✓ `UPDATE`, que modifica datos existentes en la base de datos.

2.1 Sentencia INSERT

La unidad de datos más pequeña que puede añadirse a una base de datos es una fila completa de una tabla. En general, un SGBD basado en SQL proporciona tres maneras de añadir nuevas filas de datos a una base de datos:

- ✓ Una sentencia `INSERT` de una fila añade una única nueva fila de datos a una tabla.
- ✓ Una sentencia `INSERT` multifila extrae filas de datos de otra parte de la base y las añade a una tabla.

- ✓ Una utilidad de carga masiva añade datos a una tabla desde un fichero externo a la base de datos. Se utiliza habitualmente para cargar inicialmente la base de datos o para incorporar datos transferidos desde otro sistema informático.

La sintaxis de la sentencia `INSERT` que añade una fila simple es la siguiente

```
INSERT INTO tabla [(lista de columnas)]  
VALUES ( lista expresiones | DEFAULT );
```

Donde:

- ✓ `INSERT INTO tabla` .- Tabla en la que se va a insertar la nueva fila.
- ✓ `(lista de columnas)` .- Lista de columnas de la tabla separadas por coma donde se introducirán los valores de la sentencia especificados en la cláusula `VALUES`.
- ✓ `VALUES (lista expresiones | DEFAULT)` .- Lista de valores separados por coma correspondientes a las columnas especificadas en la sentencia. Habrá tantos valores como columnas y deberán ser del mismo tipo estableciendo una correspondencia en orden, es decir, el primer valor se introduce en la primera columna, el segundo en la segunda, así sucesivamente. Un valor puede ser `DEFAULT` indicando que tendrá el valor por defecto especificado en la definición de la tabla.

Los nombres de las columnas detrás del nombre de tabla son opcionales y si no se ponen se supondrá todos los campos de la tabla en su orden original. Si se ponen, se podrán indicar cualquier número de columnas en cualquier orden. La lista de valores son los datos que se insertarán en la tabla. Los tipos de datos deben coincidir con la definición dada en tabla.

Se puede utilizar la sentencia `INSERT` con SQL interactivo para añadir filas a una tabla que crece muy raramente. En la práctica, sin embargo, los datos referentes a un nuevo cliente o un nuevo pedido son casi siempre añadidos a una base de datos mediante una aplicación de usuario orientada a formularios. Cuando la entrada de datos está completa, la aplicación inserta la nueva fila de datos utilizando SQL interactivo o programado, sin embargo, la sentencia `INSERT` es la misma.

El propósito de la lista de columnas en la sentencia `INSERT` es hacer corresponder los valores de datos en la cláusula `VALUES` con las columnas que van a recibirlos. La lista de valores y la lista de columnas deben contener el mismo número de elementos, y el tipo de dato de cada valor debe ser compatible con el tipo de dato de la columna correspondiente, o en caso contrario se producirá un error. El estándar ANSI/ISO exige nombres de columna sin cualificar en la lista de columnas, pero muchas implementaciones permiten nombres cualificados. Naturalmente no puede existir ambigüedad alguna en los nombres de columna, ya que deben referenciar todas a columnas de la tabla destino. Por ejemplo

```
# Insertar un nuevo artículo con referencia CONF0006  
# con descripción 'Tortas de aceite', precio de venta 2,5€  
# y categoría CONF.
```

```
INSERT INTO articulo (referencia, descripcion, pvp, categoria )
VALUES ('CONF0006', 'Tortas de aceite', 2.5, 'CONF');
```

Si hacemos una consulta del artículo recién creado veremos que las columnas `dto_venta`, `und_vendidas`, `und_disponibles` y `fecha_disponible` tienen el valor `NULL` ya que no se definió valor por defecto para ninguna de ellas al crear la tabla. Sin embargo, la columna `tipo_iva` tiene el valor `'N'` que es el valor por defecto definido al crear la tabla.

Las columnas que no se incluyan en la cláusula `INTO`, se inicializarán con `NULL`, siempre que la definición de la tabla lo permita y salvo que se haya definido un valor por defecto. En el ejemplo anterior no hemos indicado valor para la columna `dto_venta` y su valor es `NULL`. Se puede hacer más explícita la asignación del valor `NULL` incluyendo las columnas en la lista y especificando la palabra clave `NULL` en los valores. Esta sentencia `INSERT` tiene exactamente el mismo efecto que la anterior.

```
INSERT INTO articulo
(referencia, descripcion, pvp, dto_venta, categoria )
VALUES ('CONF0006', 'Tortas de aceite', 2.5, NULL, 'CONF');
```

Podemos omitir la lista de columnas si vamos a especificar valores para todas las columnas de la tabla. Veamos el siguiente ejemplo

```
-- Insertar un nuevo artículo con referencia CONF0007
-- con descripción 'Tortas de canela', precio de venta 3,5€,
-- descuento de venta 10%, und_vendidas 500,
-- unidades disponibles 500, sin fecha de disponibilidad
-- categoría CONF e IVA normal.
INSERT INTO articulo
VALUES ('CONF0007', 'Tortas de canela', 3.5, 0.1, 500, 500, NULL,
'CONF', 'N');
```

Como vemos en estos ejemplos hemos utilizado literales para indicar los valores de las columnas, pero se pueden emplear expresiones cuyo resultado es del mismo tipo de datos que la columna que va a recibir los datos. Por ejemplo

```
-- Insertar un nuevo artículo con referencia CONF0008
-- con descripción 'Tortas de anís', precio de venta 3,15€,
-- descuento de venta 15%, unidades vendidas 500,
-- las unidades disponibles son la mitad de las
-- unidades vendidas, su fecha disponibilidad es la fecha
-- del sistema, categoría CONF e IVA normal.
INSERT INTO articulo
VALUES ('CONF0008', 'Tortas de anís', 3.15, 0.15, 500, 500/2,
SYSDATE, 'CONF', 'N');
```

La segunda forma de la sentencia `INSERT` añade múltiples filas de datos a su tabla destino. En esta forma la sentencia `INSERT`, los valores de datos para las nuevas filas no son especificados explícitamente dentro del texto de la sentencia. En su lugar, la fuente de las nuevas filas es una consulta de base de datos, especificada en la sentencia. La sintaxis es

```
INSERT INTO tabla [( lista de columnas )]  
sentencia SELECT
```

La adición de filas cuyos valores provienen de la propia base de datos puede parecer extraña al principio, pero es muy útil en algunas situaciones especiales. Por ejemplo, supongamos que se quiere incluir en la tabla `facturas` los envíos del mes de marzo. La sentencia que haría esto es:

```
INSERT INTO factura (nfactura, fecha, nenvio)  
SELECT SFACTURA.NEXTVAL, SYSDATE, nenvio  
FROM nenvio  
WHERE fecha BETWEEN '01/03/2023' AND '31/03/2023';
```

El estándar SQL ANSI/ISO especifica varias restricciones sobre la consulta que aparece dentro de la sentencia `INSERT` multifila:

- ✓ La consulta no puede contener una cláusula `ORDER BY`. No es útil ordenar los resultados de la consulta de ningún modo, ya que van a ser insertados en una tabla que es, como todas las tablas, desordenada.
- ✓ La tabla destino de la sentencia `INSERT` no puede aparecer en la cláusula `FROM` de la consulta o de ninguna subconsulta que ésta contenga. Esto prohíbe insertar parte de una tabla en sí misma.
- ✓ Los resultados de la consulta deben contener el mismo número de columnas que la lista de columnas de la sentencia `INSERT` (o de la tabla destino entera, si se ha omitido la lista de columnas), y los tipos de datos deben ser compatibles, columna a columna.

2.2 Sentencia UPDATE

La unidad más pequeña de datos que puede modificarse en una base de datos es una única columna de una única fila. La sentencia `UPDATE` modifica los valores de una o más columnas en las filas seleccionadas de una tabla única. Su sintaxis es:

```
UPDATE tabla  
SET columna = { expresión | subconsulta | DEFAULT }, ...  
SET ( columna1, columna2, ...) = subconsulta, ...  
[WHERE condición];
```

Donde:

- ✓ `tabla` .- Tabla que se va a actualizar.
- ✓ `SET columna = { expresión | subconsulta | DEFAULT }, ...` .- Indica el conjunto de pares columna y nuevo valor que se va a asignar a la columna. El nuevo valor puede ser una expresión SQL, el resultado de una subconsulta o el valor por defecto respectivamente. Si se emplea una subconsulta solo puede producir como resultado una fila con una columna.

- ✓ `SET (columna1, columna2, ...) = subconsulta, ...` .- Indica un conjunto de columnas separadas por coma cuyos valores se actualizan con el resultado de la subconsulta. Esta debe producir una única fila con tantas columnas como se haya especificado para actualizar.
- ✓ `[WHERE condición]` .- Condición que tienen que cumplir las filas para ser actualizadas.

La tabla destino a actualizar se indica en la sentencia, y es necesario disponer de permiso para actualizar la tabla además de cada una de las columnas individuales que serán modificadas. La cláusula `WHERE` selecciona las filas de la tabla a modificar. La cláusula `SET` especifica qué columnas que se van a actualizar y calcula los nuevos valores.

El nuevo valor a especificar puede ser:

- ✓ Una expresión SQL del mismo tipo de datos que la columna a actualizar.
- ✓ Una subconsulta. Si se especifica una única columna entonces la subconsulta solo puede devolver un valor. Si se especifican múltiples columnas entonces la subconsulta debe retornar tantos valores como columnas haya. Si la consulta no devuelve ninguna fila, entonces se asigna `NULL` a la columna .
- ✓ La palabra clave `DEFAULT` para indicar que se asigne a la columna el valor por defecto que se definió para la columna cuando se creó la tabla. Si no se especificó un valor por defecto a la columna entonces se le asigna el valor `NULL`.

Veamos un ejemplo.

```
-- Modificar la dirección de envío y la forma de envío del
-- envío del cliente con nif 30000001A y fecha 1/1/2023.
UPDATE envio
SET id_dir_env = 2, forma_envio = '04'
WHERE nif = '30000001A' AND fecha = '2/1/2023';
```

La cláusula `WHERE` es exactamente la misma que se utilizaría en una sentencia `DELETE` o `SELECT` para identificar la fila. De hecho, las condiciones de búsqueda que pueden aparecer en la cláusula `WHERE` de una sentencia `UPDATE` son exactamente las mismas que las disponibles en las sentencias `SELECT` y `DELETE`.

Al igual que la sentencia `DELETE`, la sentencia `UPDATE` puede actualizar varias filas de una vez con la condición de búsqueda adecuada, como en este ejemplo:

```
-- Aumentar en un 5% el descuento del artículo CARN0001
-- en todos los pedidos.
UPDATE lpedido
SET dto = dto + 0.05 WHERE referencia = 'CARN0001';
```

En este caso, la cláusula `WHERE` selecciona varias filas de la tabla `lpedido`, y el valor de la columna `dto` se modifica en todas ellas. Conceptualmente, SQL procesa la sentencia `UPDATE` al recorrer la tabla `lpedido` fila a fila, actualizando aquellas filas para las cuales la condición de búsqueda produce un resultado `TRUE` y omitiendo aquellas para las cuales la

condición de búsqueda produce un resultado **FALSE** o **NULL**.

La cláusula **SET** en la sentencia **UPDATE** es una lista de asignaciones separadas por comas. Cada asignación identifica una columna destino a actualizar y especifica cómo calcular el nuevo valor para la columna destino. Cada columna destino debería aparecer solamente una vez en la lista; no debería haber dos asignaciones para la misma columna destino. La especificación ANSI/ISO exige nombres sin cualificar para las columnas destino, pero algunas implementaciones SQL permiten la implementación de nombres de columna cualificados. No puede existir ambigüedad alguna en los nombres de columna, ya que deben referirse a columnas de la tabla destino.

La expresión en cada asignación puede ser cualquier expresión SQL que genere un valor del tipo de dato apropiado para la columna destino. La expresión debe ser calculable basada en los valores de la fila actualmente en actualización en la tabla destino. No pueden incluirse funciones de columna.

Si una expresión en la lista de asignación referencia a una de las consultas de la tabla destino, el valor utilizado para calcular la expresión es el valor de esa columna en la fila actual antes de que se aplique ninguna actualización. Lo mismo es aplicable a las referencias de columna que se producen en la cláusula **WHERE**. Por ejemplo, consideremos esta sentencia **UPDATE**

```
UPDATE articulo
SET und_disponibles = 400, und_vendidas = und_disponibles
WHERE und_disponibles < 400;
```

Antes de la actualización, el artículo **CARN0001** tenía un valor de **und_disponibles** de 350 y un valor de **und_vendidas** de 375. Después de la actualización, su fila tiene un valor de **und_vendidas** de 350, y no de 400. El orden de las asignaciones en la cláusula **SET** no importa; las asignaciones pueden especificarse en cualquier orden.

La cláusula **WHERE** en la sentencia **UPDATE** es opcional. Si se omite la cláusula **WHERE**, entonces se actualizan todas las filas de la tabla destino, como en este ejemplo

```
UPDATE articulo_proveedor
SET dto_compra = dto_compra + 0.1;
```

A diferencia de la sentencia **DELETE**, en donde la cláusula **WHERE** casi nunca se omite, la sentencia **UPDATE** sin una cláusula **WHERE** realiza una función útil. Básicamente efectúa una actualización masiva de toda la tabla, como demuestra el ejemplo anterior.

Al igual que con la sentencia **DELETE**, las subconsultas pueden jugar un papel importante en la sentencia **UPDATE** ya que permiten seleccionar filas a actualizar en base a información contenida en otras tablas. He aquí varios ejemplos de sentencias **UPDATE** que utilizan subconsultas:

```
-- Actualiza la tabla cliente añadiendo 5000 € a sus ventas,
-- para los clientes que han hecho pedidos en el mes de
-- septiembre
```

```

UPDATE cliente
SET ventas = ventas + 5000
WHERE nif IN (SELECT cliente
              FROM pedido WHERE MONTH(fecha) = 9 );

-- Aumentar en 10000 las ventas de los clientes si el total
-- de los pedidos de cada uno de ellos supera sus ventas.
UPDATE cliente
SET ventas = ventas + 10000
WHERE ventas > ( SELECT SUM(total_pedido) FROM pedido
                WHERE pedido.nif = cliente.nif );

-- Aumentar un 10% las unidades vendidas de los artículos
-- que tengan más de tres líneas de pedido en los seis
-- primeros meses del año
UPDATE articulo
SET und_vendidas = und_vendidas + und_vendidas * 0.1
WHERE 3 < ( SELECT COUNT(*)
            FROM lpedido
            WHERE lpedido.referencia = articulo.referencia
            AND npedido IN ( SELECT lpedido FROM pedido
                            WHERE fecha BETWEEN '01/01/2023'
                            AND '30/06/2023' )
          );

-- Actualiza la tabla pedido calculando el total_importe con
-- la suma de los importes de las líneas de pedido
-- correspondientes a cada pedido.
UPDATE pedido
SET total_importe = (SELECT SUM(unidades * (precio - precio *
                                dto))
                    FROM lpedido
                    WHERE pedido.npedido = lpedido.npedido );

```

Como en la sentencia **SELECT**, las subconsultas en la cláusula **WHERE** de la sentencia **UPDATE** pueden anidarse a cualquier nivel, y pueden contener referencias externas a la tabla destino de la sentencia **UPDATE**. La columna **npedido** en la subconsulta del ejemplo anterior es una referencia externa; se refiere a la columna **npedido** en la fila de la tabla **pedido** actualmente comprobada por la sentencia **UPDATE**.

La misma restricción que se aplicaba a la sentencia **DELETE** es aplicable aquí: la tabla destino no puede aparecer en la cláusula **FROM** de ninguna subconsulta a ningún nivel de anidación. Esto impide que las subconsultas referencien la tabla destino (algunas de cuyas filas pueden ya haber sido actualizadas). Cualesquiera referencias a la tabla destino en las subconsultas son por tanto referencias externas a la fila de la tabla destino que actualmente está siendo comprobada por la cláusula **WHERE** de la sentencia **UPDATE**.

2.3 Sentencia DELETE

La unidad más pequeña de datos que puede ser suprimida de una base de datos relacional es una única fila. La sentencia **DELETE** elimina filas seleccionadas de datos de una única tabla. Su sintaxis es:

```
DELETE [FROM] tabla [WHERE condición];
```

Donde:

- ✓ `[FROM] tabla` .- Tabla donde se van a eliminar las filas.
- ✓ `[WHERE condición]` .- Condición que tienen que cumplir las filas para ser borradas

La cláusula `FROM` especifica la tabla destino que contiene las filas. La cláusula `WHERE` especifica la condición que cumplen las filas de la tabla que van a ser suprimidas. Es equivalente al `SELECT`, pero en vez de mostrar las filas que cumplan la condición, las elimina. Por ejemplo:

```
DELETE FROM cliente WHERE nif = 1;
```

La cláusula `WHERE` de este ejemplo identifica una sola fila de la tabla `cliente`, que SQL elimina de la tabla. La cláusula `WHERE` tiene un aspecto familiar, es exactamente la misma cláusula `WHERE` que se especificaría en una sentencia `SELECT` para recuperar la misma fila de la tabla. Las condiciones de búsqueda que pueden especificarse en la cláusula `WHERE` de la sentencia `DELETE` son las mismas disponibles en la cláusula `WHERE` de la sentencia `SELECT`, descrita en secciones anteriores.

Recordemos que las condiciones de búsqueda en la cláusula `WHERE` de una sentencia `SELECT` pueden especificar una sola fila o un conjunto entero de filas, dependiendo de la condición de búsqueda específica. Lo mismo es aplicable a la cláusula `WHERE` en una sentencia `DELETE`. Supongamos, por ejemplo, que se quieren borrar todos los pedidos del cliente 1. He aquí la sentencia de supresión que elimina los pedidos de la tabla Pedidos:

```
DELETE FROM pedido WHERE nif = 1;
```

En este caso, la cláusula `WHERE` selecciona varias filas de la tabla `pedido`, y SQL elimina todas las filas seleccionadas de la tabla. Conceptualmente, SQL aplica la cláusula `WHERE` a cada una de las filas de la tabla `pedido`, suprimiendo aquellas para las cuales la condición de búsqueda produce un resultado `TRUE` y reteniendo aquellas para las cuales la condición de búsqueda produce un resultado `FALSE` o `NULL`. Otro ejemplo

```
-- Eliminar todos los pedidos anteriores al 20/8/2022
DELETE FROM pedido
WHERE fecha < '20/8/2022';
```

Si se omite la cláusula `WHERE` se borrarán todas las filas de la tabla. Por ejemplo:

```
DELETE FROM pedido;
```

Aunque esta sentencia `DELETE` produce una tabla vacía, no borra la tabla `pedido` de la base de datos. La definición de la tabla `pedido` y sus columnas siguen estando almacenadas en la base de datos. La tabla aún existe, y nuevas filas pueden ser insertadas en la tabla `pedido` con la sentencia `INSERT`. Para cancelar la definición de la tabla de la base de datos, debe utilizarse la sentencia `DROP TABLE` descrita en un capítulo posterior.

Debido al daño potencial que puede producir una sentencia **DELETE** tal como ésta, es importante especificar siempre una condición de búsqueda y tener cuidado de que se seleccionan realmente las filas que se desean. Cuando se utiliza SQL interactivo, es buena idea utilizar primero la cláusula **WHERE** en una sentencia **SELECT** para visualizar las filas seleccionadas, asegurarse de que son las que realmente se desea suprimir, y sólo entonces utilizar la cláusula **WHERE** en una sentencia **DELETE**.

Las sentencias **DELETE** con condiciones de búsqueda simples, como las de los ejemplos anteriores, seleccionan las filas a suprimir basándose únicamente en los propios contenidos de las filas. A veces, la selección de las filas debe efectuarse en base a datos contenidos en otras tablas. Por ejemplo, supongamos que se desea suprimir todos los pedidos del cliente con nombre *Javier* y apellidos *González López*. Sin saber su **nif**, no se pueden determinar los pedidos consultando únicamente la tabla **pedido**. El modo de manejar la petición es con una de las condiciones de búsqueda subconsulta. He aquí una forma válida de la sentencia **DELETE** que realiza la petición:

```
DELETE FROM pedido
WHERE nif = ( SELECT nif
              FROM cliente
              WHERE nombre = 'Javier'
              AND apellidos = 'González Gómez' );
```

La consulta halla el **nif** del cliente *Javier González Gómez*, y la cláusula **WHERE** selecciona entonces los pedidos con un valor correspondiente. Como muestra este ejemplo, las subconsultas pueden jugar un papel importante en la sentencia **DELETE**, ya que permiten suprimir filas en base a información contenida en otras tablas. He aquí otro ejemplo de sentencias **DELETE** que utiliza condición de búsqueda subconsulta:

```
-- Borrar los artículos devueltos por el cliente
-- Javier González Gómez
DELETE FROM ldevolucion
WHERE npedido IN ( SELECT npedido FROM pedido
                   WHERE nif = (SELECT nif
                                FROM cliente
                                WHERE nombre = 'Javier'
                                AND apellidos = 'González Gómez')
                   );
```

Las subconsultas en la cláusula **WHERE** pueden anidarse al igual que sucedía en la cláusula **WHERE** de la sentencia **SELECT**. También pueden contener referencias externas a la tabla destino de la sentencia **DELETE**. A este respecto, la cláusula **FROM** de la sentencia **DELETE** funciona igual que la cláusula **FROM** de la sentencia **SELECT**. He aquí un ejemplo de una petición de supresión que requiere una subconsulta con una referencia externa:

```
-- Borrar los clientes que no han hecho
-- pedido desde el 10 de septiembre.
DELETE FROM cliente
WHERE NOT EXISTS ( SELECT * FROM pedido
                   WHERE pedido.nif = cliente.nif
```

```
AND fecha > '10/9/2022' );
```

Conceptualmente, esta sentencia `DELETE` opera recorriendo la tabla `cliente`, fila a fila, y comprobando la condición de búsqueda. Para cada cliente, la subconsulta selecciona los pedidos hechos por ese cliente después de la fecha de corte. La referencia a la columna `nif` en la subconsulta es una referencia externa al `nif` de cliente en la fila de la tabla `cliente` que actualmente está siendo comprobada por la sentencia `DELETE`.

2.4 Sentencia TRUNCATE

La sentencia `TRUNCATE` elimina todas las filas de una tabla, pero manteniendo la tabla y su estructura intacta. Equivale a ejecutar la sentencia `DELETE FROM <tabla>`. La diferencia con la anterior es que `TRUNCATE` no genera datos *undo* por lo que no se pueden deshacer sus efectos, y por tanto, es más rápido que `DELETE`. Debemos tener mucho cuidado en el uso de esta sentencia, ya que los datos borrados no pueden recuperarse incluso si intentamos deshacer los efectos de la ejecución de esta sentencia dentro de una transacción con `ROLLBACK`. La sintaxis de la sentencia `TRUNCATE` es

```
TRUNCATE TABLE <tabla>
[ {DROP [ALL] | REUSE} STORAGE ]
```

Donde:

- ✓ `TRUNCATE TABLE <tabla>` .- Tabla donde se van a eliminar todas las filas.
- ✓ `[DROP [ALL] STORAGE]` .- Se libera todo el espacio que ocupaban las filas, quedando a disposición para otros objetos del esquema. Si se emplea `ALL` se libera también el espacio indicado en la cláusula `MINEXTENTS` cuando se creó la tabla.
- ✓ `[REUSE STORAGE]` .- Se retiene todo el espacio que ocupaban las filas en la tabla, que podría utilizarse si se vuelven a introducir nuevas filas en la tabla.

3 Carga masiva de datos

La tercera manera de añadir filas a las tablas de una base de datos es mediante una utilidad de carga masiva. Los datos a insertar en una base de datos son con frecuencia extraídos de otro sistema informático o recolectados de otros lugares y almacenados en un archivo de texto. Para cargar los datos en una tabla, se podría escribir un programa con un bucle que leyera cada registro del archivo y utilizara la sentencia `INSERT` de una fila para añadir la fila a la tabla. Sin embargo, la carga computacional de hacer que el DBMS ejecute repetidamente sentencias `INSERT` de una fila puede ser bastante alta. Si insertar una sola fila tarda medio segundo para una carga de sistema típica, éste es un rendimiento probablemente aceptable para un programa interactivo. Pero este rendimiento rápidamente pasa a ser inaceptable cuando se aplica a la tarea de cargar 50.000 filas de datos de una vez. En este caso, la carga de los datos requeriría más de seis horas.

Por esta razón, todos los productos DBMS comerciales incluyen una capacidad de carga masiva que carga los datos desde un archivo a una tabla a alta velocidad. El estándar

SQL ANSI/ISO no considera esta función, y suele ser suministrada como un programa de utilidad autónomo en lugar de formar parte del lenguaje SQL. La utilidad que cada SGBD suministra dispone de un conjunto ligeramente diferente de características, funciones y órdenes.

En el caso de ORACLE se dispone de SQL*Loader para cargar datos desde un fichero de texto.

3.1 Exportación / Importación de datos

Esta sección describe como exportar e importar a una BD Oracle. Podemos exportar e importar los metadatos, es decir, las definiciones de objeto de la BD, los datos o ambos. Los datos pueden exportarse para posteriormente ser importados en otra BD Oracle o en una BD diferente de Oracle. También, se pueden importar a una BD Oracle datos exportados en otra BD no Oracle, siempre y cuando tengan un formato adecuado.

SQL Developer provee herramientas para exportar e importar datos y metadatos de una BD Oracle. Estas herramientas permiten exportar los datos y metadatos en varios formatos, como SQL, CSV, Excel, etc.

3.1.1 Exportar datos

Para exportar los datos de una BD Oracle con intención de importarlos en otra BD Oracle seguimos el siguiente proceso:

1. Arrancar SQL Developer y abrir una conexión con la BD.
2. Seleccionar la opción de menú *Herramientas* ➔ *Exportación de Base de Datos...* Comienza un asistente de exportación. Seleccionamos la conexión de la BD que empleará para la exportación.
3. Activar *Exportar DDL* si queremos que se exporten los metadatos. Esto hará que en el fichero de exportación se incluyan sentencias **CREATE TABLE** para crear la tabla. Así, podríamos importarlos sin necesidad de crear las tablas primero, sino que en el propio fichero de importación tendríamos las sentencias para crear las tablas. En las casillas de esta sección indicaremos las opciones que deseamos que aparezcan en las sentencias **CREATE TABLE**.

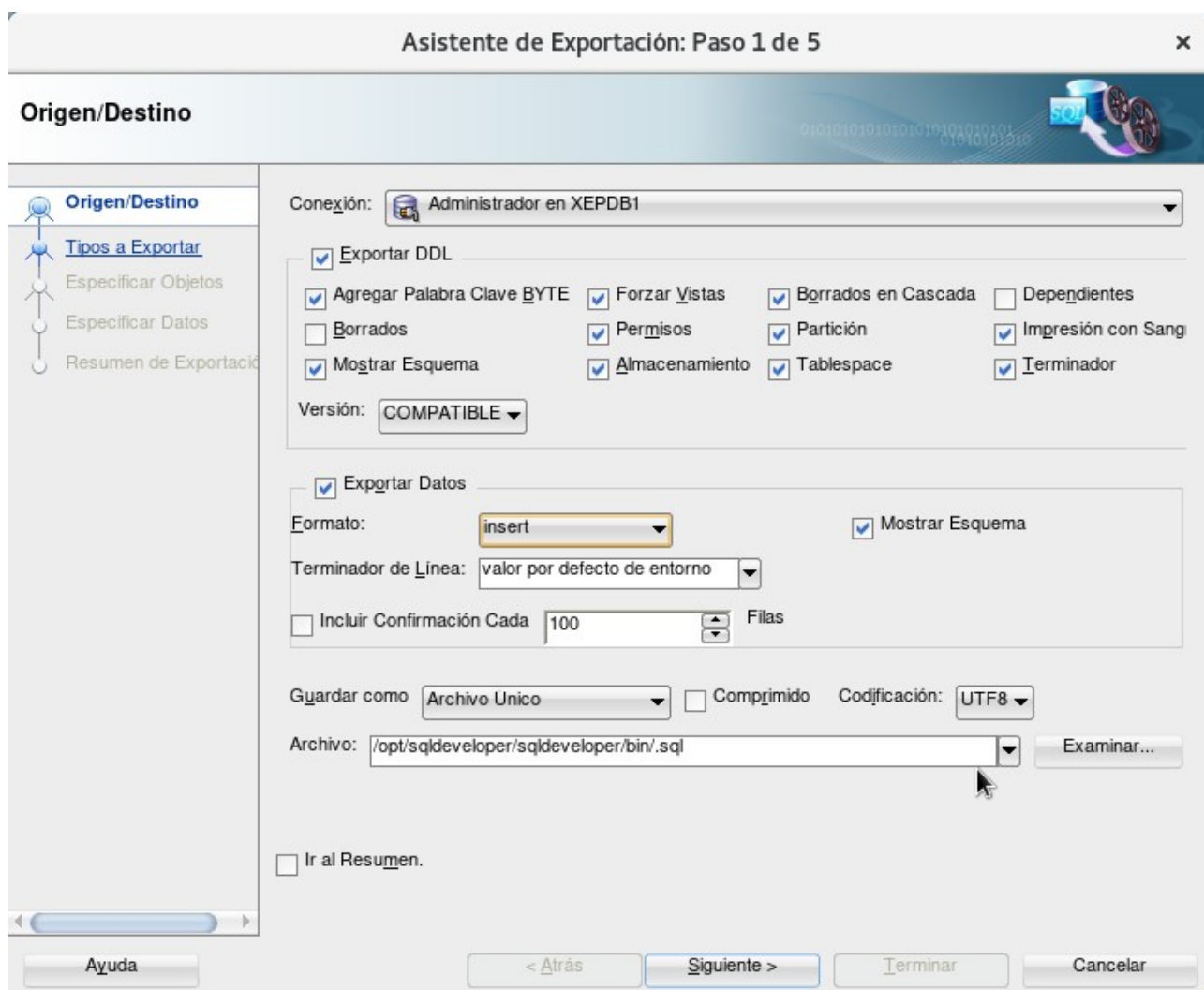


Figura 1.- Exportación de base de datos en SQLDeveloper

4. Activamos *Exportar datos* para que incluya también los datos en el fichero de exportación. En la lista Formato debemos indicar el formato de la exportación. Disponemos de las siguientes opciones:
 - CSV.- Los datos se exportan en formato .CSV donde cada valor de columna se separa del siguiente con una coma y los valores cadena de caracteres se encierra entre comillas dobles u otro carácter a elegir. En este formato conviene hacer la exportación en archivos separados y la definición de datos se haría en formato SQL.
 - Delimited.- Igual que el anterior pero podemos elegir el carácter delimitador de valores de columna.
 - Excel.- Disponemos de varias versiones de excel para exportar los datos a formato de hoja de cálculo.
 - insert .- Archivo SQL. Se incluyen sentencias INSERT por cada fila de cada tabla. Se pueden exportar todas las tabla, definición y datos, en el mismo archivo con

extensión SQL.

- Fixed.- Archivo de texto con ancho de columna fijo. Cada línea del archivo es una fila de la tabla donde todas las columnas tienen el mismo ancho (la de la columna mayor) y los valores encerrados entre comillas dobles. En este caso cada tabla tendrá dos archivos: uno con extensión SQL para la creación de la tabla (si hemos activado Exportar DDL) y otro con extensión TXT con los datos de la tabla.
- HTML.- Los datos se exportan en una página web. Este formato no es adecuado para una importación.
- JSON.- Es un archivo donde los datos se exportan en forma de objetos JSON de Javascript.
- LOADER.- Incluye los datos en un archivo con formato compatible con la utilidad SQL*Loader para importar datos.
- PDF.- Los datos se exportan a un fichero PDF, el cual no es adecuado para su importación.
- Text.- Los datos se exportan en formato .TSV donde cada valor de columna se separa del siguiente con un tabulador y los valores cadena de caracteres se encierra entre comillas dobles u otro carácter a elegir. En este formato conviene hacer la exportación en archivos separados y la definición de datos se haría en formato SQL.
- XML.- Los datos se exportan en formato XML.

Dependiendo del formato elegido las opciones posteriores pueden variar. Establecemos en cada caso lo que sea necesaria. Podemos encontrar las siguientes:

- Cabecera.- Si queremos que incluya una fila de cabecera al archivo con los rótulos de los nombres de columna.
- Delimitador.- Si elegimos el formato delimitado hay que indicar el carácter que separa un campo del siguiente. En CSV está fijado a coma y en TSV a tabulador. En ambos casos no se puede cambiar.
- Terminador de línea.- El formato texto de Windows y Linux utilizan diferente carácter para indicar un fin de línea en un archivo de texto. Dependiendo del destino de la exportación, o el origen de la importación, elegiremos uno u otro.
- Cierre a la izquierda y la derecha.- Carácter para encerrar cada valor de columna en cada fila. Por defecto es comillas dobles en ambos casos lo que significa que las columnas de tipo cadena de caracteres van a tener sus valores encerrados entre comillas dobles.
- Nombre de hoja de trabajo de datos.- Nombre de la hoja de cálculo. Solo para exportación a formato Excel.
- Incluir confirmación cada ... filas.- En el formato SQL (insert) se puede incluir una

sentencia COMMIT cada *n* filas indicadas de sentencia INSERT.

- Título, asunto y palabras clave.- Estos datos se insertan dentro de un archivo PDF.

Una vez establecido el formato de exportación hay que indicar en qué archivo o archivos se guardan. Para ello se indican las siguientes opciones:

- Guardar como.- Aquí indicamos si queremos la exportación en uno o varios archivos. Dependiendo del formato de exportación se pueden incluir la definición de datos y los datos en un solo archivo o, por otro lado, también puede ser necesario separar la definición de datos y los datos de cada tabla en archivos independientes.
- Comprimido.- Si activamos esta casilla los archivos con los datos se comprimen.
- Codificación.- Juego de caracteres del archivo o archivos resultado de la exportación. Este dato es importante cuando exportamos en un archivo de texto, como SQL, CSV, TSV, etc. ya que cuando hagamos la importación deberemos tener los datos en el juego de caracteres de la base de datos de destino si no queremos incluir caracteres extraños en los valores de las columnas al exportar en un juego de caracteres e importar en otro distinto.
- Archivo.- Nombre del archivo resultado de la exportación.

Una vez indicado qué exportar, en qué formato y donde hacemos clic en el botón Siguiente.

5. En *Tipos a exportar* indicamos que objetos de la base de datos vamos a exportar. De nuevo tendremos que tener en cuenta el destino de la exportación, ya que si es otra BD Oracle podemos elegirlos todos para hacer una migración completa. Si por el contrario es un SGBD diferente, como MySQL o Postgres SQL, tendremos que exportar las tablas y aquellos objetos también existan en el otro SGBD. Cuando acabemos hacemos clic en el botón *Siguiente*.



Figura 2.- Tipos de objetos

- En *Especificar objetos* indicamos los objetos de BD que vamos a exportar. Si hacemos clic en el botón *Consulta* nos aparecerán todos los objetos del tipo seleccionado en el punto anterior. Iremos seleccionando el que nos interese y haciendo clic en el botón con flecha derecha para incluirlo en la exportación. Con el botón de la doble flecha se incluyen todos los objetos. Cuando terminemos hacemos clic en el botón *Siguiente*.

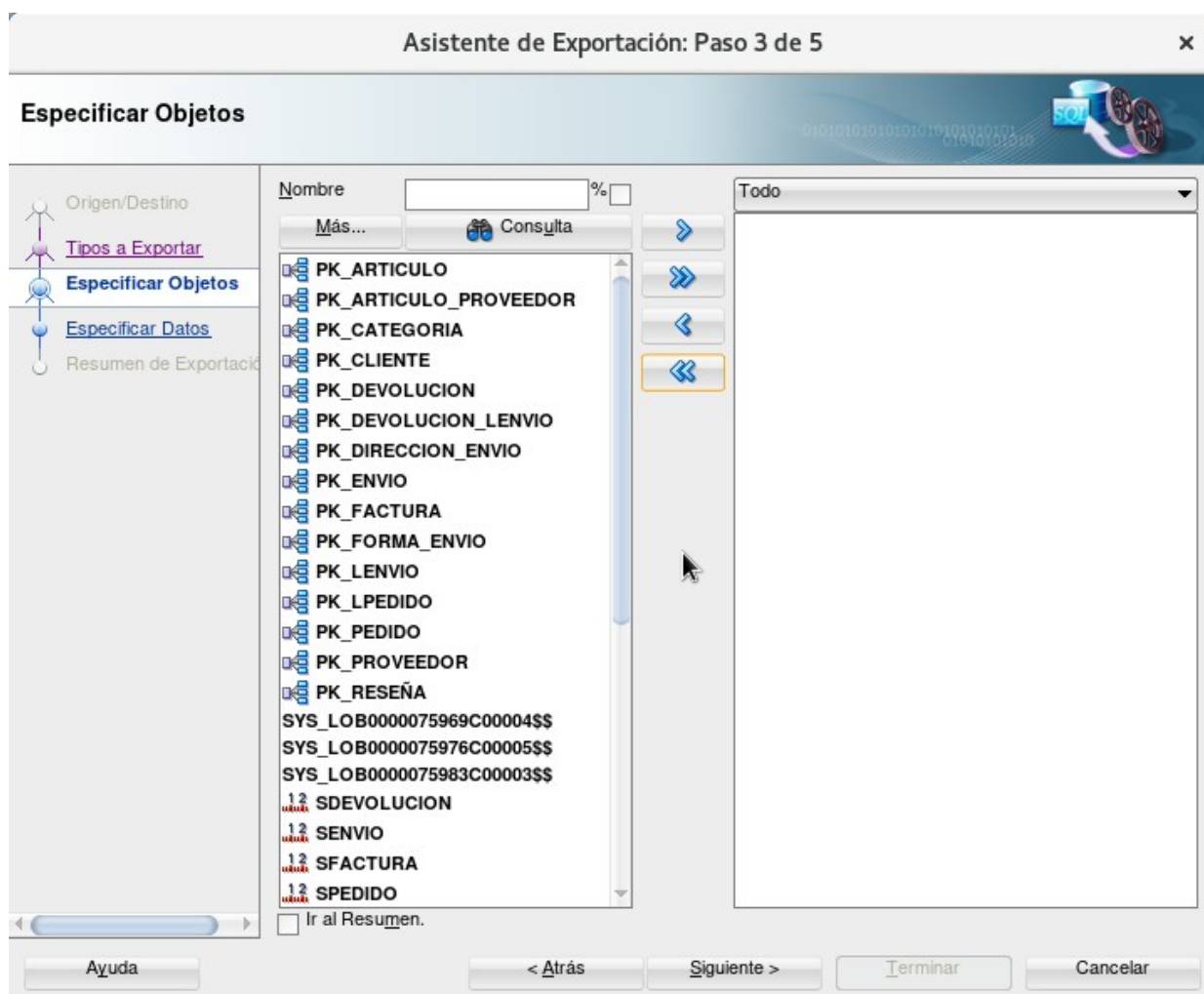


Figura 3.- Especificar objetos

7. En *Especificar datos* podemos elegir qué columnas y filas de las tablas se incluyen en la importación. Por defecto se incluyen todas las columnas y todas las filas, pero en la columna *Columnas* podemos hacer clic y editarla en cada tabla para especificar las columnas a incluir en la exportación. También podemos especificar una condición en la columna *WHERE* de objeto para seleccionar un subconjunto de las filas. Cuando terminemos hacemos clic en el botón *Siguiente*.

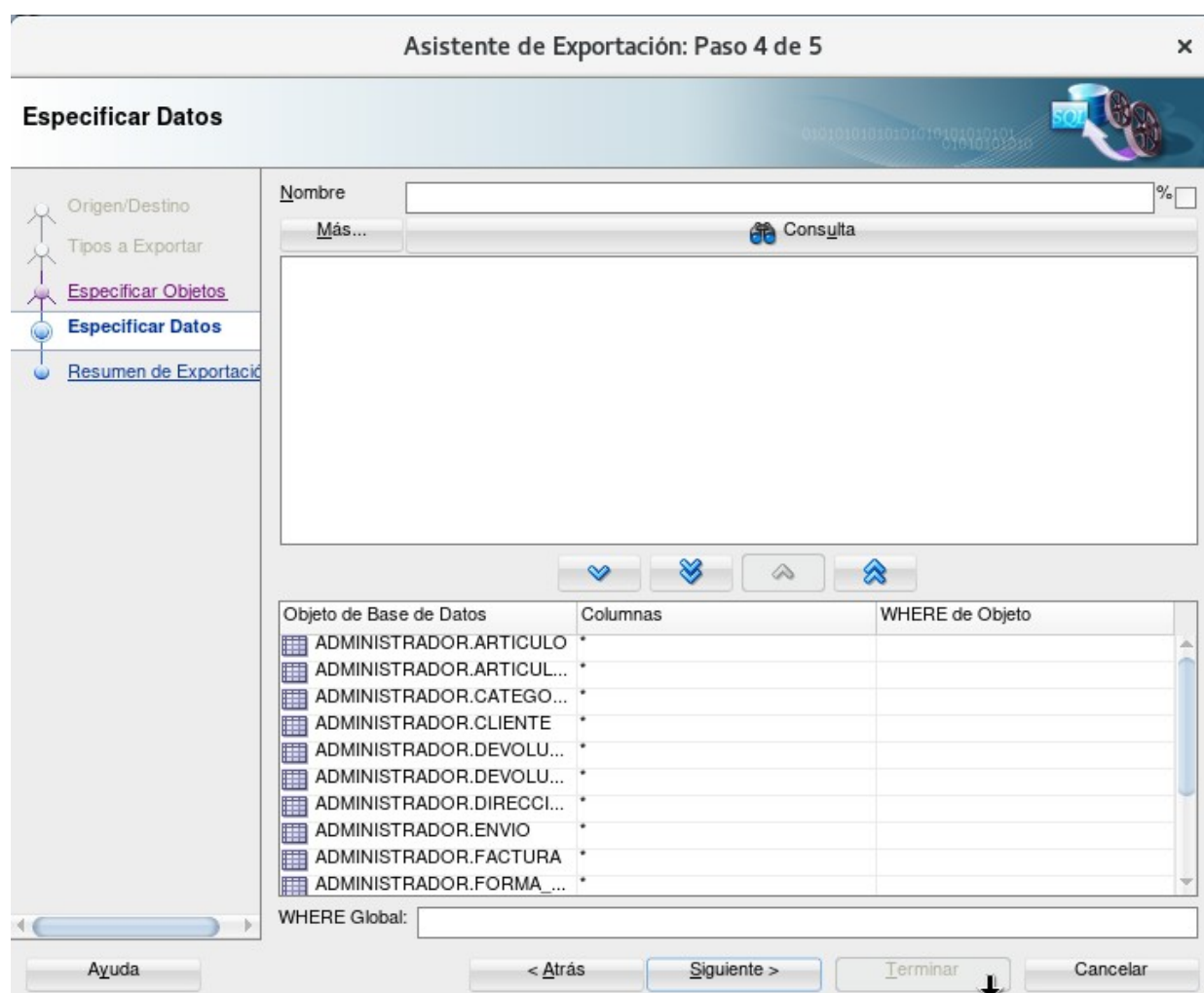


Figura 4.- Especificar datos

8. Nos aparece el último paso del asistente con un resumen de la exportación. Hacemos clic en *Terminar* y comienza la exportación. Cuando finalice se cierra el asistente y tendremos en la carpeta elegida el archivo, o los archivos, con la exportación. Dependiendo del formato es posible que se abra una pestaña en SQL Developer con alguno de los archivos de la exportación.

3.1.2 Importar datos

Si hemos creado una exportación con formato SQL (insert) simplemente tenemos que invocar el archivo o archivos SQL generados en la importación desde el SQL Developer. Si solamente tenemos un archivo la invocación es de dicho archivo. Si tenemos varios archivos debemos seguir el siguiente orden:

1. Invocar los archivos con las definiciones de objetos de BD: primero las tablas, luego los índices y los constraint.
2. Invocar los archivos con las sentencias `INSERT` que introducen los datos en las tablas creadas en el punto anterior.

3.2 Carga desde archivo CSV

Cuando desde un SGBD se realiza una exportación en formato CSV suele incluir varios archivos de texto, uno por tabla, con los datos del archivo. Cada línea del archivo es una fila de una tabla, donde cada valor de columna está encerrado entre comillas dobles y se separa del siguiente con una coma. En este caso, solo disponemos de los datos, pero no de las definiciones de los objetos del esquema, que tendrán que ir en un archivo SQL aparte.

Para importar los datos desde archivos con formato CSV seguimos el siguiente proceso:

1. Ejecutar SQL Developer y abrir una conexión con la BD.
2. Ejecutar los archivos SQL con las definiciones de los objetos: tablas, índices, secuencias, etc si los hubiere. De esta forma tendremos las tablas creadas para poder rellenarlas con los datos de la exportación.
3. En el árbol de conexiones desplegar el icono *Tablas* y sobre una tabla en la que queremos importar datos hacemos clic con el botón derecho del ratón.
4. Seleccionamos *Importar datos...*
5. Aparece un asistente de importación. El primer paso es *Vista previa de datos* en la que elegiremos el formato del archivo de importación en la lista *Formato* y con el botón *Examinar* elegiremos el archivo con los datos de la tabla. Tendremos que establecer las mismas opciones que utilizamos durante la exportación, como el juego de caracteres, caracteres de cierre de cadenas, terminador de línea, etc. Cuando hayamos indicado todos los datos hacemos clic en el botón *Siguiente*.

Asistente de Importación de Datos: Paso 1 de 5

Vista Previa de Datos

Vista Previa de Datos

Método de Importación

Seleccionar Columnas

Definición de Columna

Terminar

Restaurar Estado

Origen: Archivo Local

Archivo: /home/dba/Documentos/Ciente.csv Examinar...

Formato de Archivo

☒ Cabecera Después de Omitir

Omitir Filas: 0

Formato: csv

☒ Límite de Vista Previa de Filas: 100

Codificación: UTF8

Delimitador: ,

Terminador de Línea: estándar: CR LF, CR o LF

Cierre a la Izquierda: "

Cierre a la Derecha: "

Contenido del Archivo

NIF	NOMBRE	APELLIDOS	CLAVE	IBAN	TELEFONO	EMAIL	VENTAS
40000002A	Luis	Lucero Labore	9250E222C4...	ES00 0000 0...	957000000	luisluce@ge...	
40000003A	Luis	Lucero Labore	9250E222C4...	ES00 0000 0...	957000000	luisluce@ge...	
30000001A	Rafael	González G...	9250E222C4...	ES00 1234 5...	957000000	rafagon1111...	1590
30000002A	Javier	Hernández ...	9250E222C4...	ES00 1234 5...	957000001	javierhern22...	1580
30000003A	Maria	Ibarra Ibáñez	9250E222C4...	ES00 1234 5...	957000002	mariaiba333...	1207
30000004A	Lucia	Jiménez Jua...	9250E222C4...	ES00 1234 5...	957000003	lucijim444...	1372
30000005A	Santiago	Klein King	9250E222C4...	ES00 1234 5...	957000004	santiklein555...	1899
30000006A	Sergio	López León	9250E222C4...	ES00 1234 5...	957000005	sergiolop666...	1151
30000007A	Inés	Martínez Mo...	9250E222C4...	ES00 1234 5...	957000006	inesmart777...	1169
30000008A	Irene	Navarro Nog...	9250E222C4...	ES00 1234 5...	957000007	irenenav888...	1137
30000009A	Carlos	Ordóñez Ortiz	9250E222C4...	ES00 1234 5...	957000008	carlosord99...	1173

Ayuda
< Atrás
Siguiente >
Terminar
Cancelar

Figura 5.- Importación de datos desde archivo CSV

- En *Método de Importación* seleccionamos el método que empleará para insertar las filas. Todos excepto *Tabla externa* en área temporal insertan los datos directamente en la tabla. Hacemos clic en el botón *Siguiente*.

Asistente de Importación de Datos: Paso 2 de 4

Método de Importación

Especifique el método para importar los datos. En el caso del método de tabla externa en área temporal, se creará una tabla externa como tabla intermedia para importar la tabla de destino. Para otros métodos de importación, los datos se importan directamente a la tabla.

Método de Importación: Insertar

☐ Enviar Crear Script a Hoja de Trabajo de SQL

Nombre de la Tabla: CLIENTE

☐ Límite de Importación de Filas: 100

Contenido del Archivo

NIF	NOMBRE	APELLIDOS	CLAVE	IBAN	TELEFONO	EMAIL	VENTAS
40000002A	Luis	Lucero Labore	9250E222C4...	ES00 0000 0...	957000000	luisluce@ge...	
40000003A	Luis	Lucero Labore	9250E222C4...	ES00 0000 0...	957000000	luisluce@ge...	
30000001A	Rafael	González G...	9250E222C4...	ES00 1234 5...	957000000	rafagon1111...	1590
30000002A	Javier	Hernández ...	9250E222C4...	ES00 1234 5...	957000001	javierhern22...	1580
30000003A	Maria	Ibarra Ibáñez	9250E222C4...	ES00 1234 5...	957000002	mariaiba333...	1207
30000004A	Lucia	Jiménez Jua...	9250E222C4...	ES00 1234 5...	957000003	lucijim4444...	1372
30000005A	Santiago	Klein King	9250E222C4...	ES00 1234 5...	957000004	santiklein555...	1899
30000006A	Sergio	López León	9250E222C4...	ES00 1234 5...	957000005	sergiolop666...	1151
30000007A	Inés	Martínez Mo...	9250E222C4...	ES00 1234 5...	957000006	inesmart777...	1169
30000008A	Irene	Navarro Nog...	9250E222C4...	ES00 1234 5...	957000007	irenenav888...	1137
30000009A	Carlos	Ordóñez Ortiz	9250E222C4...	ES00 1234 5...	957000008	carlosord99...	1173
30000010A	César	Pérez Parra	9250E222C4...	ES00 1234 5...	957000009	cesarper000...	991
30000011A	Carmen	Quintana Qu...	9250E222C4...	ES00 1234 5...	957000010	carmenquin1...	1265
30000012A	Carlota	Ruiz Romero	9250E222C4...	ES00 1234 5...	957000011	carlotaruiz22...	1367
30000013A	Teodoro	Sanz Salgado	9250E222C4...	ES00 1234 5...	957000012	teosanz3333...	1271
30000014A	Teófilo	Torre Toledo	9250E222C4...	ES00 1234 5...	957000013	teofitor4444...	1812
30000015A	Daniela	Ugarte Uria	9250E222C4...	ES00 1234 5...	957000014	daniugar555...	857
30000016A	Daniela	Ugarte Uria	9250E222C4...	ES00 1234 5...	957000015	daniugar555...	857

Ayuda
< Atrás
Siguiente >
Terminar
Cancelar

Figura 6.- Método de importación

- En *Selección de columnas* podemos indicar un subconjunto de las columnas de las tablas que se rellenarán los datos. Lo habitual es no indicar ninguna y hacer clic en el botón *Siguiente*.

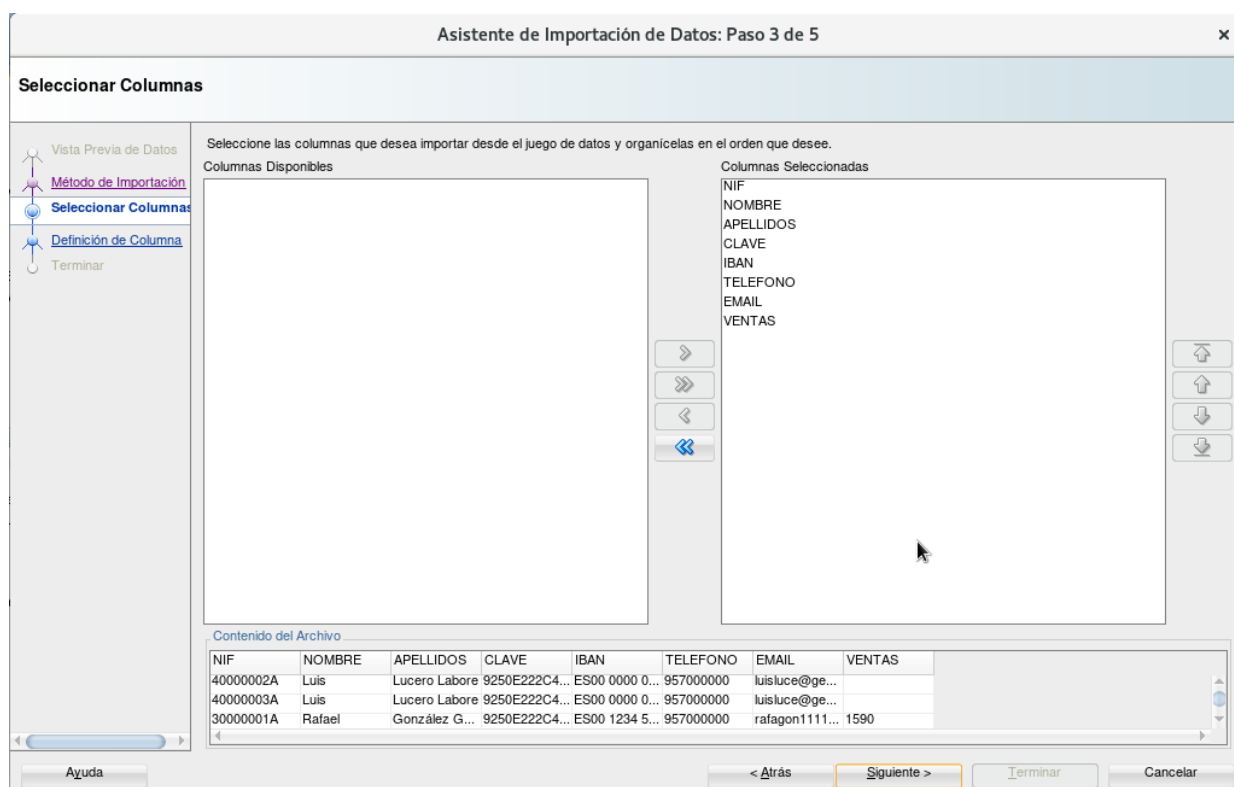


Figura 7.- Selección de columnas

8. En *Definición de columna* indicamos qué columnas del archivo de importación se vincula con qué columna de la tabla. Podemos hacer coincidir el nombre de la cabecera de la columna en el fichero de importación con el nombre de la columna en la tabla. También podemos establecer el vínculo por la posición de cada columna en el fichero. Esto solamente hay que hacerlo si en el fichero de exportación introducimos una fila de cabecera. Si no fue así, podemos hacer clic directamente en el botón Siguiente.

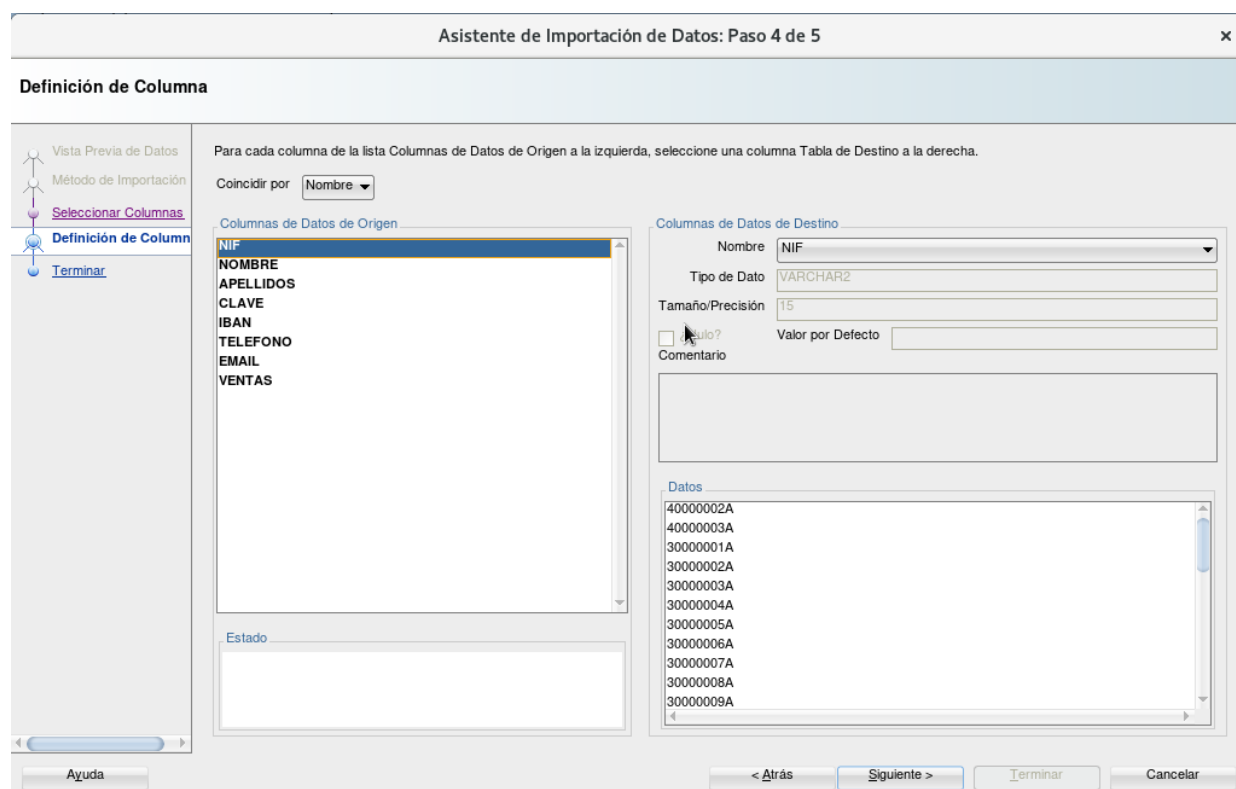


Figura 8.- Definición de columna

9. En el último paso nos presenta un resumen de la importación. Hacemos clic en el botón *Terminar*.

Si no hubo ningún problema informará del fin de la importación. Este proceso hay que repetirlo por cada tabla.

El orden que hay que seguir tanto para invocar los scripts SQL con las definiciones de datos como de las importaciones de los datos en cada tabla depende de las restricciones de clave externa. Primero las tablas referenciadas y luego las tablas con referencias. Para no equivocarnos podemos ver el archivo SQL que se generó en la exportación y que contiene las invocaciones a los diferentes archivos SQL con las definiciones de tabla, restricciones, restricciones referenciales, etc. En este archivo se establece el orden de invocación correcto y podemos hacer las invocaciones desde aquí.

3.3 SQL Loader

SQL Loader es una utilidad que carga datos desde ficheros externos en tablas de la base de datos. Dispone de un potente motor de traducción que permite gran variedad de formato de datos en el fichero de carga. Podemos usar SQL Loader para hacer lo siguiente:

- ✓ Cargar datos desde la red si los ficheros están en un host diferente al de la base de datos.
- ✓ Cargar datos desde múltiples archivos en la misma sesión de carga.
- ✓ Cargar datos en múltiples tablas en la misma sesión de carga.

- ✓ Especificar el conjunto de caracteres de los datos.
- ✓ Carga selectiva de datos basándonos en los valores de los registros.
- ✓ Manipular los datos antes de la carga mediante funciones SQL.
- ✓ Generar valores de clave secuencial única en especificadas columnas.
- ✓ Usar el sistema de archivos del sistema operativo para acceder a los ficheros de datos.
- ✓ Generar informes de errores para ayuda en la solución de problemas.
- ✓ Cargar datos objeto-relacional complejos.
- ✓ Usar ficheros secundarios para la carga de campos LOB y colecciones.

Podemos utilizar SQL*Loader de dos formas: con archivo de control o sin él. Un archivo de control controla el comportamiento de SQL*Loader y uno o más ficheros de datos utilizados en la carga. Mediante el uso de un archivo de control disponemos de un mayor control sobre la operación de carga, lo que es conveniente en situaciones de carga complejas. Para cargas simples podemos utilizar SQL*Loader sin especificar el archivo de control. Esto se conoce como SQL*Loader en modo express.

La salida de SQL*Loader es la carga de los datos en la base de datos, además de un conjunto de archivos.

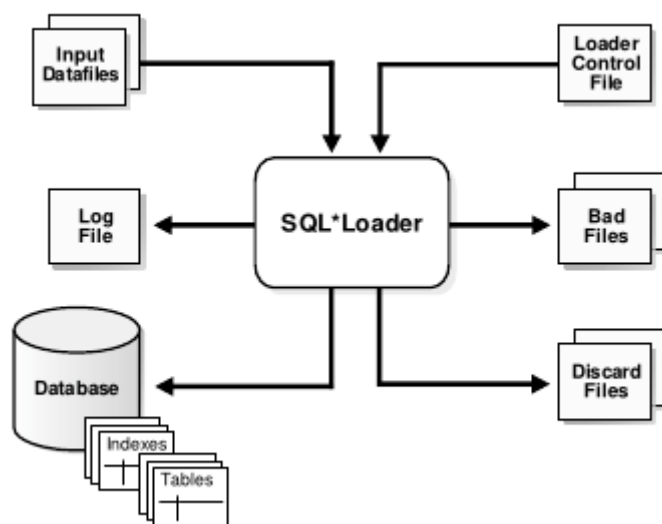


Figura 9.- Diagrama de operación de SQL Loader

Los archivos que genera son:

- ✓ Archivo log → Contiene información sobre la carga de datos realizada, con un resumen de su resultado.
- ✓ Archivo malo → Contiene los registros que han sido rechazados debido a errores. Estos errores podrían incluir tipos de datos erróneos o violaciones de restricciones de

integridad.

- ✓ Archivo de descarte → Contiene registros rechazados ya que fueron filtrados por una sentencia en el archivo de control.

A modo de ejemplo ilustraremos la carga de un fichero en formato CSV en una tabla de la base de datos.

3.3.1 Fichero de control

El fichero de control es un fichero de texto comprensible para SQL*Loader. Este archivo le dice a SQL*Loader donde encontrar los datos, como interpretarlos, donde insertar los datos, etc.

En general, el fichero de control tiene tres secciones:

- ✓ Información de sesión.
- ✓ Información de tabla y lista de campos.
- ✓ Entrada de datos (opcional)

La sintaxis es de formato libre. Las sentencias pueden extenderse sobre múltiples líneas. La sintaxis no es sensible a la capitalización, sin embargo las cadenas encerradas entre comillas simples o dobles se toman como literales, incluyendo la capitalización.

Los comentarios comienzan con dos guiones hasta el final de la línea. La sección opcional se interpreta como datos en lugar de como sintaxis de control, por tanto los comentarios no se admiten en esta sección.

Por ejemplo, el siguiente archivo de control `cliente.txt` se va a emplear para añadir archivos a la tabla `cliente`.

```
LOAD DATA
CHARACTERSET UTF8
INFILE 'cliente.csv'
BADFILE 'cliente.bad'
DISCARDFILE 'cliente.dsc'
INSERT
INTO TABLE cliente
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
( nif,
  nombre,
  apellidos,
  clave,
  iban,
  telefono,
  email,
  ventas
)
```

- ✓ La sentencia `LOAD DATA` le dice a SQL*Loader que es el principio de una nueva

carga de datos.

- ✓ La cláusula `CHARACTERSET UTF8` indica que el archivo con los datos tiene juego de caracteres UTF8.
- ✓ La cláusula `INFILE` especifica el nombre del archivo de datos que queremos cargar.
- ✓ La cláusula `BADFILE` especifica el fichero en el que se incluirán los registros rechazados en la carga.
- ✓ La cláusula `DISCARDFILE` especifica el nombre del archivo con los registros descartados.
- ✓ La cláusula `INSERT` indica que los registros se van a insertar en una tabla vacía. Si la tabla contuviera ya registros habría que indicar la cláusula `APPEND`.
- ✓ La cláusula `INTO TABLE` indica la tabla donde se añadirán los datos.
- ✓ Las cláusulas `FIELDS TERMINATED BY ','` `OPTIONALLY ENCLOSED BY '"'` indican el carácter con el que se separan los campos en el fichero de datos y que algunos tienen sus valores encerrados entre comillas.
- ✓ La cláusula `TRAILING NULLCOLS` indica que trate cualquier columna relativamente posicionada y que no está presente en el registro como columna nula.

A continuación vienen la lista de campos en la tabla a rellenar. Alternativamente, estos campos pueden tener tipo de datos y longitud máxima, aunque para el ejemplo que nos ocupa no es necesario. Para más información consultar la documentación de Oracle en <https://docs.oracle.com/en/database/oracle/oracle-database/21/sutil/index.html>.

3.3.2 Fichero de parámetros

Cuando habitualmente se empleen los mismos parámetros para valores que apenas cambian, puede ser más eficiente especificar los parámetros empleando uno de los dos métodos siguientes:

- ✓ Podemos agruparlos juntos en un fichero de parámetros. Al invocar SQL Loader utilizamos el argumento en línea de comando `PARFILE` para indicar este archivo.
- ✓ Podemos especificar parámetros en el archivo de control mediante la cláusula `OPTIONS`.

Además, podemos indicar parámetros en la línea de comando, los cuales tienen precedencia a los indicados en el archivo de parámetro o en la cláusula `OPTIONS`.

```
USERID="administrador/usuario@localhost/XEPDB1"
CONTROL=cliente.txt
LOG=cliente.log
BAD=cliente.bad
DATA=cliente.csv
```

3.3.3 Ejecución de SQL Loader

Para ejecutar SQL Loader tenemos que invocarlo desde una ventana de terminal. Dispone de un conjunto de argumentos con nombre que nos permiten personalizar la ejecución de la utilidad. Si disponemos del fichero de parámetros visto en la sección anterior podemos indicarlo con el argumento `PARFILE`.

```
sqlldr PARFILE=cliente.par
```

Si por el contrario, queremos utilizar únicamente el fichero de control anterior, podemos invocarlo de la siguiente manera.

```
sqlldr "administrador/usuario@localhost/XEPDB1"  
CONTROL=cliente.txt
```

En nuestro caso el usuario es `administrador` con la clave `usuario`. La instancia de base de datos está en la máquina local y se va a abrir sesión en la PDB con nombre `XEPDB1`.

Con el argumento `CONTROL` indicamos el archivo de control anterior.

4 Transacciones

Una transacción es una unidad de trabajo atómica y lógica que contiene una o más sentencias SQL de actualización de datos que se ejecutan todas, lo cual significa que se aplican a la BD, o ninguna, lo que implica que todos los cambios realizados por cualquier de ellas son deshechos.

Oracle asigna a cada transacción un identificador llamado ID de transacción. Una transacción tiene cuatro propiedades:

- ✓ Atomicidad.- Todas las sentencias de la transacción deben ejecutarse, y si no lo hacen, entonces la BD queda en el estado anterior a la ejecución de la primera de ellas. O se ejecutan todas o ninguna.
- ✓ Consistencia.- Una transacción lleva la BD de un estado consistente a otro estado también consistente.
- ✓ Aislamiento.- El efecto de una transacción no es visible a otras transacciones hasta que no se confirma.
- ✓ Durabilidad.- Los cambios realizados por las transacciones son permanentes. Después de que una transacción se complete, la BD asegura a través de su mecanismo de recuperación que los cambios hechos por la transacción no se pierden.

Toda transacción tiene un comienzo y un final. Una transacción comienza cuando se ejecuta la primera sentencia SQL que genera una llamada a la instancia de BD, lo que incluye sentencias DDL, DML y la sentencia `SET TRANSACTION`.

Una transacción puede acabar por diferentes razones:

- ✓ El usuario o la aplicación invoca la sentencia `COMMIT` o `ROLLBACK` sin una cláusula `SAVEPOINT`. En el primer caso el usuario solicita que los cambios de la transacción se hagan permanentes y visibles a los otros usuarios. En el segundo caso los cambios de la transacción se deshacen. En este caso la transacción termina explícitamente por el usuario
- ✓ El usuario ejecuta una sentencia DDL como `CREATE`, `DROP`, `RENAME`, o `ALTER`. La BD ejecuta una sentencia `COMMIT` implícita antes y después de cada sentencia DDL. Si la transacción actual contiene sentencias DML, Oracle primero confirma la transacción y después ejecuta y confirma la sentencia DDL como una transacción de una única sentencia.
- ✓ El usuario sale normalmente de una herramienta o utilidad de BD Oracle, lo que causa que la actual transacción se confirme. Este comportamiento es configurable, aunque todas las aplicaciones deberían confirmar, o deshacer, explícitamente las transacciones antes de terminar.
- ✓ Una aplicación cliente terminal anormalmente, provocando que la transacción se deshaga implícitamente.

Después del fin de una transacción, la siguiente sentencia SQL ejecutable automáticamente comienza la siguiente transacción.

Durante la transacción, todas las modificaciones que hagamos sobre base de datos, no son definitivas, sino que se realizan sobre un `TABLESPACE` especial que se denomina `TABLESPACE` de `ROLLBACK`, o RBS (*RollBack Segment*). Este `TABLESPACE` tiene reservado un espacio para cada sesión activa en el servidor, y es en ese espacio donde se almacenan todas las modificaciones de cada transacción. Una vez que la transacción se ha finalizado, las modificaciones temporales almacenadas en el RBS, se vuelcan al `TABLESPACE` donde está almacenada la tabla. Esto permite que ciertas modificaciones que se realizan en varias sentencias, se puedan validar todas a la vez, o rechazar todas a la vez.

Cuando tenemos abierta una sesión, los cambios que realizamos no son visibles a otra sesión hasta que se confirmen con `COMMIT`. Este se puede realizar de forma manual, ejecutando la sentencia `COMMIT`; o bien, de forma automática, cuando cerramos la sesión. En una transacción los datos modificados no son visibles por el resto de usuarios hasta que se confirme la transacción.

Si alguna de las tablas afectadas por la transacción tiene triggers, las operaciones que realiza el trigger están dentro del ámbito de la transacción, y son confirmadas o deshechas conjuntamente con la transacción. Durante la ejecución de una transacción, una segunda transacción no podrá ver los cambios realizados por la primera transacción hasta que esta se confirme.

4.1 Control de transacciones

El control de transacciones es la gestión de los cambios realizados por las sentencias

DML y los grupos de sentencias DML dentro de las transacciones. El control de las transacciones implica el uso de las siguientes sentencias:

- ✓ La sentencia **COMMIT** finaliza la actual transacción y hace permanentes todos los cambios realizados durante la transacción. También se borran todos los puntos guardados en la transacción
- ✓ La sentencia **ROLLBACK** deshace el trabajo realizado en la transacción actual, lo que provoca que todos los cambios hechos a los datos desde la última sentencia **COMMIT** o **ROLLBACK** se descarten. La sentencia **ROLLBACK TO SAVEPOINT** deshace los cambios hasta el último punto guardado, pero no finaliza la transacción.
- ✓ La sentencia **SAVEPOINT** identifica un punto dentro de una transacción al que podemos ir cuando hacemos un **ROLLBACK**.

Veamos el siguiente ejemplo en el uso de transacciones. Supongamos la tabla empleado con la siguiente definición

```
CREATE TABLE empleado (  
  id          NUMBER(6) PRIMARY KEY,  
  nombre      VARCHAR2(40),  
  categoria   VARCHAR2(3),  
  sueldo_mes   NUMBER(5,2)  
) TABLESPACE users;
```

Supongamos que la tabla almacena los datos de los empleados que pueden ser de tres categorías diferentes: **EJE**(cutivos), **GER**(entes), **AYU**(dantes) y **AUX**(iliares). Vamos a aumentar el sueldo un 10% a los auxiliares, un 7.5% a los ayudantes, un 5% a los ejecutivos y un 2.5% a los gerentes. Las sentencias para actualizar la columna **sueldo_mes** son:

```
-- Actualizamos el sueldo de los gerentes  
UPDATE TABLE empleado  
SET sueldo_mes = sueldo_mes + sueldo_mes * 2.5 / 100  
WHERE categoria = 'GER';  
  
-- Actualizamos el sueldo de los ejecutivos  
UPDATE TABLE empleado  
SET sueldo_mes = sueldo_mes + sueldo_mes * 5 / 100  
WHERE categoria = 'EJE';  
  
-- Actualizamos el sueldo de los ayudantes  
UPDATE TABLE empleado  
SET sueldo_mes = sueldo_mes + sueldo_mes * 7.5 / 100  
WHERE categoria = 'AYU';  
  
-- Actualizamos el sueldo de los auxiliares  
UPDATE TABLE empleado  
SET sueldo_mes = sueldo_mes + sueldo_mes * 10 / 100  
WHERE categoria = 'AUX';
```

Si se produjera un fallo durante la ejecución de alguna de estas sentencias podría ocurrir que algunos empleados tienen su subida de sueldo mientras que otros no. Para

evitarlo incluimos las cuatro sentencias en una transacción y si todo va bien la confirmamos. Quedaría así

```
-- Confirmamos las operaciones de actualización de la actual
-- transacción
-- Al ejecutar COMMIT se confirma la transacción anterior
-- y se crea una nueva
COMMIT

-- Actualizamos el sueldo de los gerentes
UPDATE TABLE empleado
SET sueldo_mes = sueldo_mes + sueldo_mes * 2.5 / 100
WHERE categoria = 'GER';

-- Actualizamos el sueldo de los ejecutivos
UPDATE TABLE empleado
SET sueldo_mes = sueldo_mes + sueldo_mes * 5 / 100
WHERE categoria = 'EJE';

-- Actualizamos el sueldo de los ayudantes
UPDATE TABLE empleado
SET sueldo_mes = sueldo_mes + sueldo_mes * 7.5 / 100
WHERE categoria = 'AYU';

-- Actualizamos el sueldo de los auxiliares
UPDATE TABLE empleado
SET sueldo_mes = sueldo_mes + sueldo_mes * 10 / 100
WHERE categoria = 'AUX';

-- Confirmamos todas las actualizaciones
COMMIT
```

En la anterior transacción hemos supuesto que no ha ocurrido ningún error. Si no fuera así, el último **COMMIT** no habría que ejecutarlo, por el contrario tendríamos que deshacer las operaciones realizadas hasta el momento del error ejecutando **ROLLBACK**.

```
-- Si hubiera ocurrido algún error en alguna de ellas
-- deshacemos la transacción
ROLLBACK
```

Si las sentencias se ejecutan desde una aplicación habría que hacer un control de las operaciones manejando las posibles excepciones que pudieran ocurrir. Por ejemplo, el siguiente fragmento de PL/SQL realiza las mismas operaciones anteriores pero en un procedimiento.

```
DECLARE
    type categorias IS VARRAY(4) OF VARCHAR2(3);
    type subidas IS VARRAY(4) OF NUMBER(3,1);
    categoria categorias;
    subida subidas;
BEGIN
    categoria:= categorias('GER','EJE','AYU','AUX');
```



```

subida:= subidas(2.5, 5, 7.5, 10);
FOR i in 1 .. categoria.count LOOP
    UPDATE TABLE empleado
    SET sueldo_mes = sueldo_mes + sueldo_mes * subida(i) /
100
    WHERE categoria = categoria(i);
END LOOP;
COMMIT;
EXCEPTION
WHEN OTHERS THEN
    dbms_output.put_line('Error en la transaccion:'||SQLERRM);
    dbms_output.put_line('Se deshacen las modificaciones');
    ROLLBACK;
END;

```

También podemos querer recuperarnos desde un punto concreto de la transacción para continuar con las operaciones. Para ello podemos definir un punto dentro de la transacción con la sentencia **SAVEPOINT**.

Siguiente con el ejemplo anterior supongamos que queremos definir puntos dentro de la transacción para recuperarnos cuando cometemos algún error.

```

-- Confirmamos las operaciones de actualización de la actual
-- transacción
-- Al ejecutar COMMIT se confirma la transacción anterior
-- y se crea una nueva
COMMIT

-- Actualizamos el sueldo de los gerentes
UPDATE TABLE empleado
SET sueldo_mes = sueldo_mes + sueldo_mes * 2.5 / 100
WHERE categoria = 'GER';

-- Creamos un punto en la transacción
SAVEPOINT gerentes;

-- Actualizamos el sueldo de los ejecutivos
UPDATE TABLE empleado
SET sueldo_mes = sueldo_mes + sueldo_mes * 5 / 100
WHERE categoria = 'EJE';

-- Creamos un punto en la transacción
SAVEPOINT ejecutivos;

-- Actualizamos el sueldo de los ayudantes
UPDATE TABLE empleado
SET sueldo_mes = sueldo_mes + sueldo_mes * 75 / 100
WHERE categoria = 'AYU';

-- Error, hemos aumentado el sueldo un 75% a los ayudantes
-- Podemos recuperar la BD al estado anterior antes de la última
-- sentencia UPDATE dejando las otras dos sentencias
ROLLBACK TO SAVEPOINT ejecutivos;

```

```
-- Ahora corregimos el error y actualizamos bien el sueldo
-- de los ayudantes
-- Actualizamos el sueldo de los ayudantes
UPDATE TABLE empleado
SET sueldo_mes = sueldo_mes + sueldo_mes * 7.5 / 100
WHERE categoria = 'AYU';

-- Creamos un punto en la transacción
SAVEPOINT ayudantes;

-- Actualizamos el sueldo de los auxiliares
UPDATE TABLE empleado
SET sueldo_mes = sueldo_mes + sueldo_mes * 10 / 100
WHERE categoria = 'AUX';

-- Confirmamos todas las actualizaciones
COMMIT
```

5 Bloqueos

Un bloqueo es un mecanismo que previene las interacciones destructivas entre transacciones. Una interacción entre transacciones que acceden a datos compartidos se considera destructiva cuando actualiza datos o modifica las estructuras de datos subyacentes de forma incorrecta. Los bloqueos juegan un papel crucial en mantener la consistencia y la concurrencia de la base de datos.

La base de datos mantiene varios tipos diferentes de bloqueos, dependiendo de la operación que adquiere el bloqueo. En general, la base de datos utiliza dos tipos de bloqueos: exclusivos y compartidos. Solo un bloqueo exclusivo puede ser obtenido en un recurso como una fila o tabla, pero muchos bloqueos compartidos pueden obtenerse en un recurso simple.

Los bloqueos afectan a la interacción de lectores y escritores. Un lector es una consulta a un recurso mientras que un escritor es una sentencia que modifica un recurso. Las siguientes reglas muestran el comportamiento de Oracle para los lectores y los escritores:

- ✓ Una fila es bloqueada cuando se modifica por un escritor → Cuando una sentencia actualiza una fila, la transacción adquiere el bloqueo para esta única fila. Bloqueando los datos de una tabla a nivel de fila, la base de datos minimiza la contención para los mismos datos. Bajo circunstancias normales, la base de datos no escala un bloqueo de fila al nivel de bloque o tabla.
- ✓ Un escritor de una fila bloquea a un escritor concurrente de la misma fila → Si una transacción está modificando una fila, entonces el bloqueo de fila previene que otra transacción modifique la misma fila simultáneamente.
- ✓ Un lector nunca bloquea a un escritor → Ya que un escritor de una fila no bloquea a un escritor, éste puede modificar la fila. La única excepción es una sentencia `SELECT .. FOR UPDATE`, la cual es un tipo especial de `SELECT` que bloquea la fila

que está leyendo.

- ✓ Un escritor nunca bloquea a un lector → Cuando una fila está siendo cambiada por un escritor, la base de datos utiliza los datos *undo* para proporcionar a los lectores una vista consistente de la fila.

Cuando múltiples usuarios acceden y modifican datos, la base de datos debe proporcionar una forma de prevenir la modificación concurrente de los mismos datos. Los bloqueos logran los siguientes requisitos de base de datos:

- ✓ Consistencia → Los datos que una sesión está viendo o cambiando no pueden ser cambiados por otras sesiones hasta que el usuario termine.
- ✓ Integridad → Los datos y estructuras deben reflejar todos los cambios realizados a los mismos en una secuencia correcta.

Oracle proporciona concurrencia de datos, consistencia e integridad entre transacciones a través de su mecanismo de bloqueos. Los bloqueos ocurren automáticamente y no requieren intervención del usuario.

La necesidad de los bloqueos se ilustra en el siguiente ejemplo con una actualización concurrente de una única fila. En este ejemplo, una aplicación web presenta al usuario final un empleado con un email y un teléfono. Esta aplicación utiliza una sentencia **UPDATE** como la siguiente para modificar los datos.

```
UPDATE empleado
SET email = ?, telefono = ?
WHERE empleado_id = ?
AND email = ? AND telefono = ?
```

En esta sentencia, los valores de email y teléfono en la cláusula **WHERE** son los valores originales y sin modificar para el empleado especificado. Esta actualización garantiza que la fila que la aplicación modifica no fue cambiada antes de leer estos datos. De esta forma, la aplicación evita la pérdida de la actualización si un usuario sobrescribe cambios realizados por otro usuario, haciendo que el segundo usuario pierda la actualización.

Oracle automáticamente obtiene los bloqueos necesarios cuando ejecuta sentencias SQL. Por ejemplo, antes de que la base de datos permita que una sesión modifique datos, la sesión debe primero bloquearlos. El bloqueo da a la sesión control exclusivo sobre los datos para que ninguna otra transacción pueda modificar los datos bloqueados hasta que el bloqueo es liberado.

Ya que el mecanismo de bloqueo de Oracle está fuertemente ligado al control de transacciones, los diseñadores de aplicaciones solo necesitan definir las transacciones adecuadamente y Oracle gestiona automáticamente los bloqueos. El usuario nunca necesita bloquear un recurso explícitamente, aunque Oracle también permite a los usuarios habilitar bloqueos manualmente.

Oracle automáticamente utiliza el nivel aplicable más bajo de restricción para proporcionar el grado más alto de concurrencia de datos asegurando la integridad de los

datos. Cuanto menor es el nivel restrictivo, mayor disponibilidad de los datos para que otros usuarios accedan. En cambio, cuanto mayor nivel restrictivo, más limitadas están las otras transacciones en el tipo de bloqueo que pueden adquirir.

Oracle usa dos modos de bloqueo en una base de datos multiusuario:

- ✓ Modo de bloqueo exclusivo → Este modo previene que el recurso asociado al bloqueo pueda ser compartido. Una transacción obtiene un bloqueo exclusivo cuando modifica los datos. La primera transacción que bloquea un recurso exclusivamente es la única que puede alterar el recurso hasta que el bloqueo exclusivo sea liberado.
- ✓ Modo de bloque compartido → Este modo permite a un recurso asociado al bloqueo ser compartido, dependiendo de las operaciones implicadas. Múltiples usuarios leyendo datos pueden compartir los datos, manteniendo un bloqueo compartido para prevenir el acceso concurrente de un escritor que necesita un bloqueo exclusivo. Múltiples transacciones pueden adquirir bloqueos compartidos en el mismo recurso.

Asumimos que una transacción utiliza una sentencia `SELECT ... FOR UPDATE` para seleccionar una única fila de tabla. La transacción adquiere un bloqueo de fila exclusivo y un bloqueo de tabla compartido de fila. El bloqueo de fila permite a otras sesiones modificar cualquier fila diferente a la fila bloqueada, mientras que el bloqueo de tabla impide que las sesiones modifiquen la estructura de la tabla. Así, la base de datos permite ejecutar tantas sentencias como sea posible.

5.1 Bloqueo automático

Oracle automáticamente bloquea un recurso como parte de una transacción para impedir que otras transacciones hagan algo que requiere acceso exclusivo al mismo recurso. La base de datos automáticamente adquiere diferentes tipos de bloqueo a diferentes niveles de restrictividad dependiendo del recurso y la operación a realizar. Estos son:

5.1.1 Bloqueo DML

También conocido como bloqueo de datos, garantiza la integridad de los datos accedidos concurrentemente por múltiples usuarios.

Por ejemplo, este bloqueo evita que dos clientes compren la última copia de un libro disponible en una tienda online. Los bloqueos DML previenen la interferencia destructiva de operaciones simultáneas y conflictivas.



Un sentencia DML automáticamente adquiere los siguientes tipos de bloqueos: bloqueo de fila (TX) y bloqueo de tabla (TM).

Un bloqueo de fila, también conocido como bloqueo TX es un bloqueo en una fila simple de una tabla. Una transacción adquiere el bloqueo de fila por cada fila modificada por una sentencia `INSERT`, `UPDATE`, `DELETE`, `MERGE`, o `SELECT ... FOR UPDATE`. El bloqueo de fila existe hasta que la transacción termina.

Los bloqueos de fila inicialmente sirven como un mecanismo para impedir que dos

transacciones modifiquen la misma fila. La base de datos siempre bloquea una fila modificada en modo exclusivo para que otras transacciones no puedan modificar la misma fila hasta que la transacción que mantiene el bloqueo termine. Bloqueo de fila proporciona el grano más fino posible de bloqueo y proporciona la mejor concurrencia y rendimiento.

Si una transacción obtiene un bloqueo para una fila, entonces la transacción también adquiere el bloqueo para la tabla que contiene la fila. El bloqueo de tabla previene conflictos en operaciones DDL que podrían sobrescribir los cambios de datos en la transacción actual. La figura siguiente ilustra una actualización en la tercera fila en una tabla. Oracle automáticamente coloca un bloqueo exclusivo en la fila actualizada y un bloqueo subexclusivo en la tabla.

Table EMPLOYEES 						
EMPLOYEE_ID	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	MANAGER_ID	DEPARTMENT_ID
 100	King	SKING	17-JUN-87	AD_PRES		90
101	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	100	90
102	De Hann	LDEHANN	13-JAN-93	AD_VP	100	90
103	Hunold	AHUNOLD	03-JAN-90	IT_PROG	102	60




 Table lock acquired
 Exclusive row lock (TX) acquired
 Row being updated

Figura 10.- Bloqueo de fila y tabla

Un bloqueo de tabla, también conocido como bloqueo TM, se adquiere por una transacción cuando se modifica la tabla con una sentencia `INSERT`, `UPDATE`, `DELETE`, `MERGE`, `SELECT ... FOR UPDATE` o `LOCK TABLE`.

Las operaciones DML requieren bloqueos de tabla para impedir operaciones DDL de otras transacciones sobre la misma tabla que podrían tener conflicto con la transacción DML.

Un bloqueo de tabla puede soportar los siguientes modos:

- ✓ Fila compartida (RS) → La transacción ha bloqueado filas en la tabla e intenta actualizarlas con una sentencia `SELECT ... FOR UPDATE`. Un bloqueo de fila compartida es el modo menos restrictivo de bloqueo de tabla ofreciendo un alto grado de concurrencia para la tabla ya que permite consultas para actualizar filas en la misma tabla.
- ✓ Fila exclusiva (RX) → Este bloqueo generalmente indica que una transacción está ejecutando una sentencia `INSERT`, `UPDATE`, `DELETE` o `SELECT ... FOR UPDATE`. Un bloqueo RX permite a las otras transacciones consultar, insertar, actualizar, borrar o bloquear filas concurrentemente en la misma tabla. Por tanto, el bloqueo RX permite a

múltiples transacciones obtener bloqueos RX para la misma tabla.

- ✓ Tabla compartida (S) → Este tipo de bloqueo se consigue con la sentencia `LOCK TABLE` en la tabla. Indica que la transacción intenta realizar actualizaciones de algunas filas en la tabla y previene que otras transacciones ejecuten actualizaciones en la tabla hasta que se libere este bloqueo.
- ✓ Fila compartida exclusiva (SRX) → Este bloqueo también se adquiere con la sentencia `LOCK TABLE` en la tabla. Previene que cualquier transacción realice un bloqueo explícito con la sentencia `LOCK TABLE` hasta que se libere el bloqueo, y también evita cualquier bloqueo en la tabla a nivel de fila.
- ✓ Tabla exclusiva (X) → Este bloqueo es el más restrictivo, prohibiendo a otras transacciones realizar cualquier tipo de sentencia DML, excepto consultas, o colocar cualquier tipo de bloqueo en la tabla.

5.1.2 Bloqueo DDL

Un bloqueo de diccionario de datos protege la definición de un objeto de esquema mientras una operación DDL en ejecución actúa sobre el objeto referido.

Solo objetos individuales de esquema que son modificados o referenciados son bloqueados durante operaciones DDL. La base de datos nunca bloquea el diccionario de datos entero. Oracle adquiere un bloqueo DDL automáticamente como parte de cualquier transacción DDL requerida. Los usuarios no pueden explícitamente requerir bloqueos DDL. Por ejemplo, si un usuario crea un procedimiento almacenado, entonces Oracle adquiere automáticamente el bloqueo DDL para todos los objetos de esquema referenciados en la definición del procedimiento. El bloqueo DDL impide que estos objetos se modifiquen o borren antes de que la compilación del procedimiento almacenado se complete.

Un bloqueo DDL exclusivo impide que otras sesiones obtengan un bloqueo DDL o DML. La mayoría de las operaciones DDL requieren bloqueos DDL exclusivos en un recurso para impedir interferencias destructivas con otras operaciones DDL que podrían modificar o referenciar al mismo objeto de esquema. Por ejemplo, `DROP TABLE` no se permite mientras una sentencia `ALTER TABLE` está añadiendo una columna a la tabla, y viceversa.

Los bloqueos DDL exclusivos duran lo que tarde en ejecutarse una sentencia DDL y realizar la confirmación automática. Durante la adquisición de un bloqueo DDL exclusivo, si otro bloqueo DDL está manteniendo el objeto del esquema por otra operación, entonces la adquisición espera que el bloqueo actual se libere antes de proceder.

Un bloqueo DDL compartido para un recurso impide una interferencia destructiva con operaciones DDL conflictivas, pero permite concurrencia de datos para operaciones DDL similares. Por ejemplo, cuando se ejecuta una sentencia `CREATE PROCEDURE`, la transacción que adquiere el DDL compartido bloquea todas las tablas referenciadas. Otras transacciones pueden concurrentemente crear procedimientos que referencian a las mismas tablas y adquieren bloqueos DDL compartidos concurrentes en las mismas tablas, pero ninguna transacción adquiere un bloqueo DDL exclusivo sobre cualquier tabla referenciada.

Un bloqueo DDL compartido durará la duración de la ejecución de la sentencia DDL y la confirmación automática. Así, una transacción reteniendo un bloqueo DDL compartido se garantiza que la definición de un objeto de esquema referenciado permanece constante durante la transacción.

5.2 Bloqueo manual

Se puede anular manualmente el mecanismo de bloqueo por defecto. Oracle realiza bloqueos automáticos para garantizar la concurrencia e integridad de los datos, y la consistencia en lectura a nivel de sentencia. Sin embargo, la anulación de los bloqueos por defecto es útil en situaciones como las siguientes:

- ✓ Las aplicaciones requieren consistencia en lectura a nivel de transacción o lecturas repetidas ➔ En este caso, las consultas deben producir datos consistentes durante la duración de la transacción, no reflejando cambios realizados por otras transacciones. Podemos lograr consistencia en lectura a nivel de transacción usando el bloqueo explícito, transacciones de solo lectura, transacciones serializables o anulando los bloqueos por defecto.
- ✓ Aplicaciones que requieren que una transacción tenga acceso exclusivo a un recurso para que no tenga que esperar a que se completen otras transacciones ➔ Podemos anular el bloqueo automático a nivel de transacción o sesión. A nivel de sesión, una sesión puede establecer el nivel de aislamiento de transacción requerido con la sentencia `ALTER SESSION`. A nivel de transacción, las transacciones que incluyen las siguientes sentencias SQL anulan los bloqueos por defecto:
 - La sentencia `SET TRANSACTION ISOLATION LEVEL`.
 - La sentencia `LOCK TABLE`, la cual bloquea una tabla o, cuando se usa con vistas, las tablas base.
 - La sentencia `SELECT ... FOR UPDATE`.

Los bloqueos adquiridos por las sentencias anteriores se liberan después de que finalice la transacción o al deshacer una transacción a un punto guardado.

Si los bloqueos por defecto se anulan a cualquier nivel, entonces el administrador de la base de datos o el desarrollador de la aplicación deberían asegurar que el procedimiento de anulación del bloqueo opera correctamente. El procedimiento de bloqueo debe satisfacer los siguientes criterios: se garantiza la integridad de los datos, la concurrencia de datos es aceptable y se impiden los bloqueos muertos o son apropiadamente manejados.

5.2.1 Sentencia LOCK TABLE

Bloquea una o más tablas, particiones de tabla o subparticiones de tabla en un modo especificado. Este bloqueo manual anula el bloqueo automático y permite o deniega acceso a la tabla o vista por otros usuarios mientras dure la operación.

Algunas formas de bloqueo pueden colocarse en la misma tabla a la vez. Otras solo permiten un único bloqueo a la tabla.

Una tabla permanece bloqueada hasta que se confirme la transacción o se deshagan los cambios ([ROLLBACK](#)) completamente o a un punto de guardado anterior al bloqueo de la tabla.

Un bloqueo nunca impide a otros usuarios consultar la tabla. Una consulta nunca coloca un bloqueo sobre la tabla. Los lectores nunca bloquean a los escritores y los escritores nunca bloquean a los lectores.

Para realizar el bloqueo se necesita el privilegio de sistema [LOCK ANY TABLE](#) o debemos tener cualquier privilegio de objeto (excepto el privilegio [READ](#)) en la tabla o vista.

La sintaxis es:

```
LOCK TABLE tabla | vista [, tabla | vista [, ...] ]  
IN modo_bloqueo MODE [NOWAIT | WAIT entero];
```

Donde:

- ✓ [LOCK TABLE tabla | vista](#) → Tabla o vista sobre la que se hace el bloqueo. Si se especifica una vista se bloquean las tablas base de la vista. Pueden especificarse varias separadas por coma.
- ✓ [modo_bloqueo](#) → Indica el modo de bloqueo. Puede ser uno de los siguientes:
 - [ROW SHARE](#) → Permite acceso concurrente a la tabla bloqueada pero prohíbe a los usuarios bloquear la tabla entera.
 - [ROW EXCLUSIVE](#) → Es igual que el anterior, pero también prohíbe bloquear la tabla en modo [SHARE](#).
 - [SHARE](#) → Permite consultas concurrentes pero prohíbe actualizaciones en la tabla bloqueada.
 - [SHARE ROW EXCLUSIVE](#) → Es utilizada para mirar a una tabla completa y permitir a otros mirar filas en la tabla pero prohibiendo a otros bloquear la tabla en modo [SHARE](#) o actualizar filas.
 - [EXCLUSIVE](#) → Permite consultas en la tabla bloqueada pero prohíbe cualquier otra actividad.
- ✓ [NOWAIT](#) → Si queremos que la base de datos devuelva el control inmediatamente si la tabla ya está bloqueada por otro usuario. En este caso, la base de datos devuelve un mensaje indicando que la tabla está bloqueada por otro usuario.
- ✓ [WAIT entero](#) → Indicamos que la sentencia [LOCK TABLE](#) debería esperar el número de segundos especificado con entero para adquirir el bloqueo DML. No hay límite en el valor de entero. Si no especificamos [NOWAIT](#) o [WAIT](#), la base de datos espera indefinidamente hasta que la tabla esté disponible, la bloquea y devuelve el control. Cuando la base de datos está ejecutando sentencias DDL concurrentemente con sentencias DML, puede ocurrir un timeout o un bloqueo muerto. La base de datos detecta algunos timeouts y bloqueos muertos y devuelve un error.

Por ejemplo, supongamos que disponemos del usuario `usuario` al que se le conceden permisos para realizar consultas, inserciones, actualizaciones y borrados de filas en la tabla `cliente` del esquema `administrador`. Con estos privilegios de objeto, el usuario rafa tiene concedido el permiso para ejecutar la sentencia `LOCK TABLE` sobre la tabla cliente.

Comenzamos desde un punto en el que no hay ningún bloqueo pendiente. Podemos consultar los bloqueos pendientes mediante la siguiente sentencia SQL en el diccionario de datos.

```
SELECT B.Owner, B.Object_Name, A.Oracle_Username,  
A.OS_User_Name, A.locked_mode  
FROM V$Locked_Object A, All_Objects B  
WHERE A.Object_ID = B.Object_ID;
```

La sentencia anterior no debe ofrecer ninguna fila en el resultado, indicando que no hay bloqueos pendientes.

Comenzamos añadiendo bloqueos cuando el usuario administrador bloquea en modo `ROW SHARE` con la siguiente sentencia

```
LOCK TABLE cliente IN ROW SHARE MODE NOWAIT;
```

Al consultar los bloqueos aparece la tabla cliente bloqueada por el usuario administrador en modo 2 (`ROW SHARE`). El usuario `usuario` puede insertar filas y actualizar filas en la tabla cliente ya que este modo de bloqueo lo permite. Cuando el usuario rafa ejecuta una sentencia `INSERT` o `UPDATE` aparecerán nuevos bloqueos asociados a estas sentencias. La sentencia `INSERT` provoca un bloqueo en modo 2 (`ROW SHARE`) que permite otros bloqueos similares, pero no en modo 3 (`ROW EXCLUSIVE`). Sin embargo, la sentencia `UPDATE` provoca un bloqueo en modo 3 (`ROW EXCLUSIVE`). Por tanto ningún otro usuario puede bloquear la tabla.

Recordemos que las inserciones, actualizaciones o borrados de filas que una sesión de usuario realiza no son vistas por los demás usuarios hasta que se confirmen.

Cuando usuario confirma la transacción con `COMMIT;` se liberan los bloqueos asociados y solo queda el bloqueo que hizo el usuario administrador con la sentencia `LOCK TABLE` que también se libera cuando este usuario ejecute `COMMIT;`.

Vamos a realizar ahora un bloqueo de mayor nivel restrictivo. Una vez liberados todos los bloqueos, el usuario administrador vuelve a bloquear la tabla en modo `ROW EXCLUSIVE` con la siguiente sentencia:

```
LOCK TABLE cliente IN ROW EXCLUSIVE MODE NOWAIT;
```

Al consultar los bloqueos aparece el del administrador en modo 3 (`ROW EXCLUSIVE`). En esta situación el usuario usuario puede insertar y actualizar filas, al igual que antes.

Una vez liberados los bloqueos anteriores, si el administrador bloquea la tabla en modo `SHARE` con la siguiente sentencia:

```
LOCK TABLE cliente IN SHARE MODE NOWAIT
```

Al consultar los bloqueos aparece el del administrador en modo 4 (SHARE). Si hubiera un bloqueo en modo 3 pendiente, no se permitiría bloquearlo en modo 4. En esta situación, no se permiten las operaciones de actualización en la tabla, las cuales quedarán pendientes hasta que se libere el bloqueo. Si se permiten consultas a la tabla.

6 Bibliografía

ASHDOWN, Lance, KEESLING, Donna, KYTE, Tom, *Oracle Database Concepts, 21c*. ORACLE 2021

KRISHNAMURTHY, Usha, *Oracle Database – SQL Language Reference, 21c*. ORACLE 2022