

UF5-Administración de Estructura de datos

Profesor RAÚL SALGADO VILAS





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ Ciertas sentencias nos permiten acceder a los elementos de la base de datos para conocer su contenido, así como permitir aplicar cambios en estructuras ya creadas.
- ❑ En esta unidad vamos a revisar la aplicación de dichas sentencias y como se van traduciendo en la estructura de una base de datos.
- ❑ Objetivos:
 - Modificar las características de las tablas una vez creadas.
 - Crear vistas
 - Acceder a elementos de la base de datos
 - Controlar las transacciones





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

❑ **ALTER:** Las tablas de la base de datos se pueden alterar una vez creadas con la sentencia ALTER. Se puede:

- Renombrar el nombre de una tabla
- Añadir, borrar y modificar las columnas de la tabla.
- Añadir, borrar y modificar restricciones.
- Habilitar y deshabilitar restricciones.

❑ La sintaxis es:

```
ALTER TABLE nombretabla {[RENAME TO nuevonombre]  
[ADD (columna) ] [MODIFY (columna [,columna] ...) ] [DROP COLUMN (columna [,columna] ...) ]  
[ADD CONSTRAINT restricción] [DROP CONSTRAINT restricción] [RENAME CONSTRAINT restricción TO nuevo_nombre]  
[DISABLE CONSTRAINT restricción]  
[ENABLE CONSTRAINT restricción]};
```





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ **Cambiar el nombre a una tabla:** La sentencia ALTER TABLE nos permite cambiar el nombre a una tabla. Para ello se utiliza RENAME TO seguido del nombre de la nueva tabla; no importa que la tabla forme parte de una relación porque el SGBD cambia automáticamente dicha relación para que recupere el nombre de la nueva tabla.
 - Ejemplo: Cambiar el nombre de la tabla directores: **ALTER TABLE directores RENAME TO direc;**
- ❑ **Añadir columnas:** La sentencia ALTER TABLE nos permite añadir nuevas columnas a nuestra tabla. Para ello se utiliza ADD seguido del nombre de la columna y su tipo de datos. Hay que tener en cuenta varios factores a la hora de añadirla:
 - Si la columna no está definida como NOT NULL, se le puede añadir en cualquier momento.
 - Si la columna está definida como NOT NULL, cabe la posibilidad de seguir los siguientes pasos:
 - ✓ En primer lugar se añade la columna sin especificar NOT NULL.
 - ✓ Después se da valor a la columna para cada una de las filas.
 - ✓ Finalmente, se modifica la columna a NOT NULL.
 - Ejemplo: Queremos añadir el campo población a las sucursales: **ALTER TABLE sucursales ADD (poblacion VARCHAR(25));**





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

❑ **Añadir columnas:** La sentencia ALTER TABLE nos permite añadir nuevas columnas a nuestra tabla. Para ello se utiliza ADD seguido del nombre de la columna y su tipo de datos. Hay que tener en cuenta varios factores a la hora de añadirla:

- Si la columna no está definida como NOT NULL, se le puede añadir en cualquier momento.
- Si la columna está definida como NOT NULL, cabe la posibilidad de seguir los siguientes pasos:
- En primer lugar se añade la columna sin especificar NOT NULL.
- Después se da valor a la columna para cada una de las filas.
- Finalmente, se modifica la columna a NOT NULL.
- Ejemplo: Queremos añadir el campo población a las sucursales

ALTER TABLE sucursales ADD (poblacion VARCHAR(25));





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ **Modificar columnas:** En este caso utilizamos MODIFY para modificar las características de una o más columnas existentes en la tabla. Al modificar una columna de una tabla se han de tener en cuenta estos aspectos:
- Se puede aumentar la longitud de una columna en cualquier momento.
 - Al disminuir la longitud de una columna que tiene datos no se puede dar menor tamaño que el máximo valor almacenado.
 - Es posible aumentar o disminuir el número de posiciones decimales en una columna de tipo NUMBER.
 - Si se quiere cambiar el tipo de datos de la columna ésta debe de estar vacía.
 - Se puede indicar que la columna sea NOT NULL.
 - Ejemplo: Cambiar el tamaño del campo población que acabamos de añadir a la tabla para que tenga un tamaño de 15=>**ALTER TABLE sucursales MODIFY (POBLACION VARCHAR(15));**
 - Ejemplo: Modificar el campo nombre de la tabla sucursales para no permitir nulos=> **ALTER TABLE sucursales MODIFY (nombre NOT NULL);**
 - Ejemplo: Aunque no tenga mucho sentido, pero para ver que es posible que se puede cambiar el tipo de datos. Cambiar el tipo de datos de apellidos a Integer de la tabla directores. Se podrá cambiar sólo si no tenemos datos en el campo apellidos, de lo contrario nos daría error=>**ALTER TABLE directores MODIFY (apellidos INTEGER);**



- 



UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ Hay que tener en cuenta que cuando creamos nuevas restricciones y hay datos ya introducidos en la tabla que no las cumple, nos dará error. Por ello primero habrá que modificar los datos de las tablas para que se ajuste a la nueva restricción y luego crearla.
 - Si quisiéramos ahora quitar esa restricción, habría simplemente que indicarle el nombre que le hemos dado y utilizar DROP=> **ALTER TABLE directores DROP CONSTRAINT direc_sueldo;**
- ❑ **Cambiar nombre a las restricciones:** Existe la posibilidad de cambiar el nombre a las restricciones. Con esta opción es posible cambiar el nombre de las restricciones que el sistema da por defecto o cambiar otras que se hayan nombrado.
 - Ejemplo: Cambiar el nombre de la restricción de la clave primaria de sucursales por sucursales_pk. En este caso el nombre por defecto que tenía dicha constraint era sys_c008776=>**ALTER TABLE sucursales RENAME CONSTRAINT sys_c008776 TO sucursales_pk;**





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ☐ **Activar y desactivar restricciones:** Por defecto, las restricciones se activan al crearlas. Se pueden desactivar añadiendo la cláusula **DISABLE** al final de la restricción. El siguiente ejemplo añade la misma restricción anterior, pero la desactiva:
 - **ALTER TABLE directores ADD CONSTRAINT direc_sueldo CHECK (sueldo >=1200) DISABLE;**
 - Se ha creado la misma restricción, pero sin activar. Por lo tanto ahora podría insertar un nuevo director que tuviera un sueldo inferior a 1200 Euros y no daría error.
- ☐ Para desactivar una restricción existente usamos la orden: **ALTER TABLE nombretabla DISABLE CONSTRAINT nombre restricción ...**
- ☐ Para activar una restricción existente usamos la orden: **ALTER TABLE nombretabla ENABLE CONSTRAINT nombre restricción ...**
- ☐ Así por ejemplo si quisiéramos activar la restricción creada lo haríamos de la siguiente manera: **ALTER TABLE directores ENABLE CONSTRAINT direc_sueldo;** Al activarla nos dará error porque tenemos insertado un director con sueldo menor a 1200 Euros.



❏ DROP: Borrar tablas

- Con el tiempo la estructura de una base de datos crecerá y se modificará. Nuevas tablas serán creadas para representar nuevas entidades, y algunas tablas antiguas ya no serán necesarias. Se puede eliminar de la base de datos una tabla que ya no es necesaria con la sentencia DROP TABLE=> **DROP TABLE table_name;**
- Cuando la sentencia DROP TABLE suprime una tabla de la base de datos, su definición y todos sus contenidos se pierden. No hay manera de recuperar los datos y habría que utilizar una nueva sentencia CREATE TABLE para volver a crear la definición de la tabla. Debido a sus serias consecuencias, debe utilizarse la sentencia DROP TABLE con mucho cuidado.
- La supresión de una tabla suprimirá los datos y los índices asociados. La supresión no será posible si la tabla es referenciada por una clave externa. De esta manera no podemos borrar la tabla directores porque está referenciada por sucursales y si lo intentamos nos dará un error.

❏ DROP: Borrar índices

- Con la sentencia DROP también podemos borrar índices creados en una tabla. La sintaxis sería: **DROP INDEX nombre_indice;**
- Para averiguar el nombre del índice de una determinada tabla podemos acceder al diccionario de datos. Esto se verá más adelante en la unidad.



UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ **TRUNCATE:** Permite suprimir todas las filas de una tabla y liberar el espacio ocupado para otros usos sin que desaparezca la definición de la tabla de la base de datos.
 - Es una orden del lenguaje de definición de datos o DDL y no genera información de retroceso; es decir, una sentencia TRUNCATE no se puede anular.
 - Por eso, la eliminación de filas con la orden TRUNCATE es más rápida que con DELETE. Su formato es: **TRUNCATE TABLE nombretabla;**
 - Por ejemplo si quisiéramos borrar la tabla clientes: **TRUNCATE TABLE clientes;**
- ❑ No se puede truncar una tabla cuya clave primaria sea referenciada por la clave ajena de otra tabla. Antes de truncar la tabla hay que desactivar la restricción.
- ❑ **TRUNCATE Ó DELETE:** Una duda bastante habitual es cuál es la diferencia entre hacer un TRUNCATE TABLE y un DELETE FROM TABLE. Pues bien hay diferencias y se muestran en la siguiente tabla:
 - ✓ **Truncate table:**
 - ❖ Es una operación DDL.
 - ❖ No permite el borrado selectivo. TRUNCATE TABLE elimina todo el contenido de la tabla.
 - ❖ No se puede ejecutar, si tiene tablas relacionadas, aun si no existiesen registros en la tabla que contiene la FK.
 - ❖ Es la forma más rápida de eliminar el contenido de una tabla.
 - ❖ No se activa ningún trigger al ejecutarse.





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

✓ **Delete From:**

- ❖ Es una operación DML
- ❖ Permite el borrado selectivo, mediante la cláusula WHERE.
- ❖ Se puede ejecutar si hay FK asociadas a la tabla, pero siempre y cuando no tenga registros asociados o la FK este deshabilitada.
- ❖ Es más lenta.
- ❖ Puede activarse el trigger de ON DELETE y poder determinar que registros están siendo eliminados.

- ❑ Como podemos ver, TRUNCATE TABLE es bastante más restrictivo que DELETE, al punto que en muchas situaciones, aun si queremos eliminar todo el contenido de la tabla, no podemos hacerlo, sin embargo gana en rapidez porque no va registro a registro tal y como hace DELETE, sino que lo que hace es liberar su contenido directamente.
- ❑ Los Triggers o Disparadores son objetos que se asocian con tablas y se almacenan en la base de datos. Su nombre se deriva por el comportamiento que presentan en su funcionamiento, ya que se ejecutan cuando sucede algún evento sobre las tablas a las que se encuentra asociado. Los eventos que hacen que se ejecute un trigger son las operaciones de inserción (INSERT), borrado (DELETE) o actualización (UPDATE), ya que modifican los datos de una tabla.





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ **Vistas:** A veces, para obtener datos de varias tablas hemos de construir una sentencia SELECT compleja y, si en otro momento necesitamos realizar esa misma consulta, tenemos que construir de nuevo la sentencia SELECT.
- ❑ Sería muy cómodo obtener los datos de una consulta compleja con una simple sentencia SELECT. Pues bien, las vistas solucionan este problema: mediante una consulta simple de una vista cabe la posibilidad de obtener datos de una consulta compleja.
- ❑ **Una vista es una tabla lógica que permite acceder a la información de una o de varias tablas.** No contiene información por sí misma, sino que su información está basada en la que contienen otras tablas, llamadas tablas base, y siempre refleja los datos de estas tablas.
- ❑ **Si se suprime una tabla, la vista asociada se invalida. Las vistas tienen la misma estructura que una tabla: filas y columnas, y se tratan de forma semejante a una tabla.**





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ **Vistas simples:** El formato para crear una vista es:

```
CREATE VIEW Nombrevista [(columna [,columna])] AS consulta  
[WITH {CHECK OPTION | READ ONLY}]  
Nombrevista es el nombre que damos a la vista.
```

- ❑ [(columna [,columna])] son los nombres de columnas que va a contener la vista. Si no se ponen, se asumen los nombres de columna devueltos por la consulta.
- ❑ AS consulta determina las columnas y las tablas que aparecerán en la vista.
- ❑ [WITH CHECK OPTION] es la opción de comprobación para una vista. Si se especifica, SQL comprueba automáticamente cada operación INSERT y UPDATE sobre la vista para asegurarse que las filas resultantes satisfagan el criterio de búsqueda de la definición de la vista. Si la fila insertada o modificada no satisface la condición de creación de la vista, la sentencia INSERT o UPDATE falla y no se realiza la operación.
- ❑ [WITH READ ONLY] especifica que sólo se puede hacer SELECT de las filas de la vista.[CONSTRAINT nombrerestriccion]





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ **Creación de vistas:** Una vista muy sencilla es la que podemos realizar sobre una única tabla, indicando ciertos campos que quiero que saque. Así por ejemplo podemos crear una vista que sea sólo el nombre de los directores cuyo sueldo sea mayor de 1200. Se haría de la siguiente manera:

```
CREATE VIEW nombres_direct AS  
(SELECT nombre FROM directores WHERE sueldo>=1200 );
```

- ❑ La vista creada se llama nombres_direct y a partir de ahora se puede usar como si de una tabla se tratase. Se puede consultar, se pueden borrar y actualizar filas, si las columnas a actualizar no son expresiones y se puede insertar, siempre y cuando todas las columnas obligatorias de la tabla asociada estén presentes en la vista. Por lo tanto una sentencia SQL totalmente válida y que accede a la vista puede ser:

```
SELECT * FROM nombres_direct WHERE nombre<>'Antonio' ORDER BY nombre;
```

- ❑ En el ejemplo indicado se muestran los nombres de los directores que cobran más de 1200 euros excluyendo a Antonio y ordenados por nombre. La consulta no se ha hecho a través de la tabla inicial sino a través de la vista, que ya contenía sólo los nombres de los directores con un sueldo mayor a 1200 Euros.





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ☐ **CHECK OPTION y READ ONLY:** Se puede crear una vista de forma que todas las operaciones INSERT, UPDATE y DELETE que se hagan sobre ella satisfagan la condición por la que se creó.
- ☐ Por ejemplo, si tenemos una vista con los clientes cuyo credito sea mayor de 500 Euros, e intentamos insertar un cliente con un credito menor a esos 500 Euros si le indicamos CHECK OPTION, no permitirá la inserción.

/* Sin CHECK OPTION no daría error */

CREATE OR REPLACE VIEW clientes_500 AS

**(SELECT dni, nombre, apellidos, direccion, fecha_de_alta, credito, nsucursal
FROM clientes WHERE credito > 500);**

INSERT INTO clientes_500

VALUES('33358479A', 'Sonia', 'López Ruiz', 'Calle S/N', '01/01/2018', 350, 1002);

/* Con CHECK OPTION daría error */

CREATE OR REPLACE VIEW clientes_500 AS

**(SELECT dni, nombre, apellidos, direccion, fecha_de_alta, credito, nsucursal
FROM clientes WHERE credito > 500) WITH CHECK OPTION;**

INSERT INTO clientes_500

VALUES('33358479A', 'Sonia', 'López Ruiz', 'Calle S/N', '01/01/2018', 350, 1002);

- ☐ La opción **WITH READ ONLY** sólo nos permitirá hacer **SELECT** en la vista. Cualquier operación **INSERT**, **UPDATE** o **DELETE** sobre la vista fallará.





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ **Vistas complejas:** Las vistas complejas contienen consultas que se definen sobre más de una tabla, pueden agrupar filas usando las cláusulas GROUP BY o crear condiciones que afecten a varias tablas.
- ❑ Se pueden crear vistas usando funciones, expresiones en columnas y consultas avanzadas, pero únicamente se podrán consultar estas vistas. Por ejemplo podemos tener una vista con los nombres de las sucursales, junto con los nombres de sus directores y clientes.

```
CREATE VIEW sucurs_detalle(Sucursal,Nom_Dir,Apell_Dir,Nom_cli,Apell) AS  
(SELECT sucursales.nombre, directores.nombre,directores.apellidos, clientes.nombre,clientes.apellidos  
FROM directores JOIN sucursales ON directores.id=sucursales.director  
JOIN clientes ON sucursales.nsucursal=clientes.nsucursal);
```

- ❑ En este caso podemos observar que le hemos indicado a la vista los nombres de los campos, porque entran en conflicto con los campos de las diferentes tablas del SELECT.





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ **Borrar vistas:** Para borrar una vista se usa la sentencia DROP VIEW:

DROP VIEW nombrevista;

- ❑ Por ejemplo, podríamos eliminar la vista que acabamos de crear:

DROP VIEW sucurs_detalle;

- ❑ Si se elimina una tabla a la que hace referencia una vista, la vista no se elimina, hay que eliminarla explícitamente.





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ☐ Acceso a los elementos de la base de datos
- ☐ Todas las bases de datos disponen de posibilidades para consultar el diccionario de datos. La forma de consultar los metadatos es la misma que en el resto de tablas. Es decir existen tablas (en realidad vistas) que en lugar de contener datos, contienen los metadatos.
- ☐ Todas las bases de datos disponen de posibilidades para consultar el diccionario de datos. La forma de consultar los metadatos es la misma que en el resto de tablas. Es decir existen tablas (en realidad vistas) que en lugar de contener datos, contienen los metadatos.
- ☐ Estos metadatos son las tablas, vistas, usuarios, etc. que tenemos en nuestra base de datos. En el caso de SQL estándar, el diccionario de datos es accesible mediante el esquema de información (INFORMATION_SCHEMA), un esquema especial que contiene el conjunto de vistas con el que se pueden consultar los metadatos de la base de datos.
- ☐ En este punto y para ir comprobando la información almacenada con nuestro ejemplo nos vamos a centrar en el SGBD Oracle.





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ Aspectos del diccionario de datos de Oracle: El diccionario de datos está constituido por numerosas tablas y vistas creadas por la base de datos Oracle.
- ❑ Tal y como hemos comentado en el punto anterior la información del diccionario de datos se almacena en vistas. Para acceder a los objetos de mi base de datos tengo que acceder a:
 - Vistas USER: dan información sobre los objetos de la base de datos que somos propietarios.
 - Acceder al diccionario de datos Oracle
 - En la siguiente lista podéis encontrar una descripción de las vistas del diccionario de datos Oracle que se utilizan más a menudo y que corresponden a vistas USER:
 - Usuarios: user_users
 - Roles asignados a roles o usuarios: user_role_privs
 - Tablas, vistas, sinónimos y secuencias: user_catalog.
 - Tablas: user_tables
 - Campos de tablas con restricciones: user_cons_columns.
 - Columnas de las tablas: user_tab_columns.
 - Vistas: user_views.
 - Restricciones de clave primaria, externa, not null, integridad referencial: user_constraints.
 - Índices: user_indexes.
 - Columnas de los índices: user_ind_columns.





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

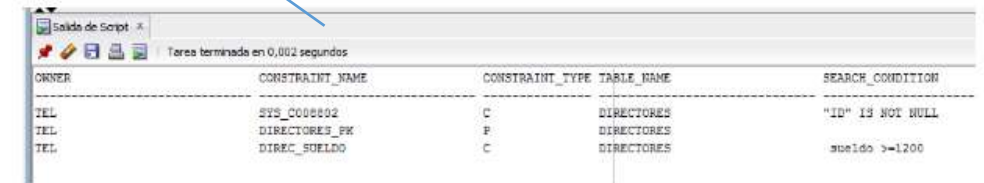


TABLE_NAME
PROPIETARIOS
SUCURSALES
CLIENTES
DIRECTORES
FABRICANTES

- ❑ Ejemplo: Mostrar las tablas que yo he creado. **SELECT * FROM user_tables;**
- ❑ Muestra mucha información, porque le hemos pedido todos los campos, pero entre todos esos campos tenemos el nombre de la tabla. Entre ellas vemos las tablas clientes, sucursales y directores de nuestro ejemplo.
- ❑ Ejemplo: Listar las restricciones que tenemos en la tabla directores. Como las restricciones la tenemos en la tabla user_constraints, debemos listar esta tabla y podemos filtrar por el campo TABLE_NAME, para que sólo nos muestre las de esta tabla en concreto. Para conocer los campos de las tablas, lo más fácil es hacer un select * y visualizarlos. Posteriormente ya conociendo el campo que nos interesa podemos filtrar por él.

SELECT * FROM user_constraints WHERE table_name='DIRECTORES';

- ❑ El resultado sería el siguiente:
 - Podemos ver que nos lista entre otros campos los siguientes:
 - CONSTRAINT_NAME: nombre de la restricción. En este caso tenemos algunos nombres que los ha definido el sistema y otros los creamos nosotros.
 - CONSTRAINT_TYPE: Por ejemplo de tipo CHECK, o PRIMARY KEY.
 - TABLE_NAME: El nombre de la tabla.
 - SEARCH_CONDITION: Condición asociada.
 - De esta manera y tal y como se puede ver en los ejemplos se puede acceder a los diferentes elementos que tengo creados en mi base de datos.



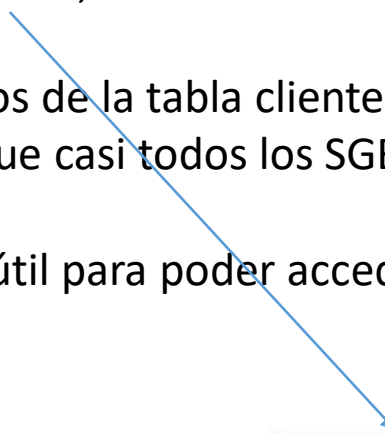
OWNER	CONSTRAINT_NAME	CONSTRAINT_TYPE	TABLE_NAME	SEARCH_CONDITION
TEL	SYS_C008802	C	DIRECTORES	"ID" IS NOT NULL
TEL	DIRECTORES_FK	F	DIRECTORES	
TEL	DIREC_SUELDO	C	DIRECTORES	sueldo >=1200





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ El comando DESCRIBE, permite obtener la estructura de una tabla sin tener que acceder a las vistas user del diccionario de datos.
- ❑ Se puede aplicar tanto a tablas como a vistas, y su sintaxis es muy simple: DESCRIBE { nombre_tabla | nombre_vista }
- ❑ Ejemplo: DESCRIBE clientes;
- ❑ Y aparecerán los campos de la tabla clientes con el tipo de datos. Esta instrucción no es parte del SQL estándar, pero casi es considerada así ya que casi todos los SGBD la utilizan. Un ejemplo del resultado de la orden anterior (en Oracle) sería:
- ❑ Esta sentencia es muy útil para poder acceder de manera rápida a la estructura de la tabla.



```
DESCRIBE clientes
Nombre      Nulo      Tipo
-----
DNI          NOT NULL  CHAR(9)
NOMBRE       NOT NULL  VARCHAR2(15)
APELLIDOS   NOT NULL  VARCHAR2(50)
DIRECCION    VARCHAR2(50)
FECHA_DE_ALTA  DATE
CREDITO      NUMBER(38)
NSUCURSAL    NUMBER(38)
```





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ **Concepto de Transacción:** Una transacción es un conjunto de sentencias SQL que se ejecutan en una base de datos como una única operación, confirmándose o deshaciéndose todo el conjunto de sentencias SQL. La transacción puede quedar finalizada (con las sentencias apropiadas) o implícitamente (terminando la sesión).
- ❑ Una transacción es un conjunto de sentencias SQL que se ejecutan en una base de datos como una única operación, confirmándose o deshaciéndose todo el conjunto de sentencias SQL. La transacción puede quedar finalizada (con las sentencias apropiadas) o implícitamente (terminando la sesión).
- ❑ Supongamos que queremos borrar una fila de una tabla, pero al teclear la orden SQL, se nos olvida la cláusula WHERE y ¡borramos todas las filas de la tabla! => Pues bien, esto no es problema, porque los Gestores de bases de datos permiten dar marcha atrás a un trabajo realizado con las órdenes oportunas.
- ❑ Cuando hacemos transacciones sobre la base de datos, es decir, cuando insertamos, actualizamos y eliminamos datos en las tablas, los cambios no se aplicarán a la base de datos hasta que los confirmemos. Esto significa que, si durante el tiempo que hemos estado realizando transacciones, no hemos hecho ninguna confirmación y de pronto se va la luz, todo el trabajo se habrá perdido, y las tablas estarán en la situación de partida.
- ❑ Cada gestor de base de datos tiene su propia sintaxis para controlar las transacciones, pero el concepto es el mismo en todos ellos. En este caso lo vamos a ver para Oracle.





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

- ❑ Una transacción comienza con la primera instrucción DML que se ejecute y finaliza con alguna de estas circunstancias:
 - Una operación de confirmación o de vuelta atrás.
 - Una instrucción DDL (como ALTER TABLE por ejemplo)
 - Una instrucción DCL
 - El usuario abandona la sesión
 - Caída del sistema
- ❑ RECUERDA: Una transacción es una secuencia de una o más sentencias SQL que juntas forman una unidad de trabajo.





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

❑ **COMMIT:** La instrucción COMMIT hace que los cambios realizados por la transacción sean definitivos, irrevocables. Sólo se debe utilizar si estamos de acuerdo con los cambios, conviene asegurarse mucho antes de realizar el COMMIT ya que las instrucciones ejecutadas pueden afectar a miles de registros.

❑ Además el cierre correcto de la sesión da lugar a un COMMIT, aunque siempre conviene ejecutar explícitamente esta instrucción a fin de asegurarnos de lo que hacemos. Por ejemplo si queremos confirmar el borrado de un cliente lo haríamos de la siguiente manera:

**DELETE FROM clientes WHERE DNI='78458784B';
COMMIT;**

❑ Existe la posibilidad de validar automáticamente las transacciones sin tener que indicarlo de manera explícita. Para eso sirve el parámetro AUTOCOMMIT. El valor de este parámetro se puede mostrar con la orden SHOW:

SHOW AUTOCOMMIT;

❑ OFF es el valor por omisión, de manera que las transacciones (INSERT, UPDATE y DELETE) no son definitivas hasta que no hagamos COMMIT. Si queremos que INSERT, UPDATE Y DELETE tengan un carácter definitivo sin necesidad de realizar la validación COMMIT, hemos de activar el parámetro AUTOCOMMIT con la orden SET:

- SET AUTOCOMMIT ON;
- Ahora, cualquier INSERT, UPDATE y DELETE se validará automáticamente.





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

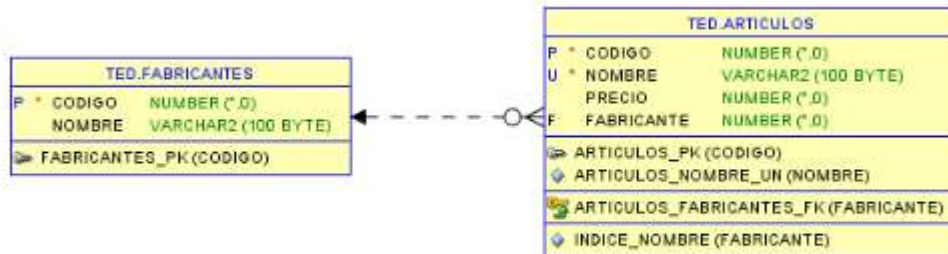
- ❑ **ROLLBACK:** Esta instrucción regresa a la instrucción anterior al inicio de la transacción, normalmente el último COMMIT, la última instrucción DDL o DCL o al inicio de sesión.
- ❑ Anula definitivamente los cambios, por lo que conviene también asegurarse de esta operación.
- ❑ Un abandono de sesión incorrecto o un problema de comunicación o de caída del sistema dan lugar a un ROLLBACK implícito.
- ❑ Ejemplo: Hemos borrado el cliente, pero nos hemos dado cuenta de que no era lo que queríamos. Podemos ejecutar ROLLBACK y el registro borrado lo tendríamos de nuevo.

DELETE FROM clientes WHERE DNI='30515454K';
ROLLBACK;

- ❑ En este caso la tabla se quedaría tal cual estaba, porque hemos borrado el cliente indicado, pero luego le hemos abortado la transacción volviendo a la situación inicial de las tablas.



❑ Partimos del modelo:



❑ Realizar las siguientes tareas:

1. Cambia el nombre de la tabla articulos por artic
2. Añade una nueva columna a la tabla artic que sea su precio_compra.
3. Cambiar el tamaño del nombre del fabricante para que sea de 50.
4. Añade la restricción a la tabla artic para que el precio_compra no pueda ser menor de 0.
5. Añade la restricción de que el nombre del fabricante no se pueda repetir
6. Crea una vista de sólo lectura que contenga los nombres de los artículos, su precio y el nombre del fabricante siempre y cuando el precio sea mayor de 25 Euros.
7. Listar las restricciones que tiene la tabla de artic accediendo al diccionario de datos de ORACLE. Muestra el nombre de la restricción, el tipo y su cometido.
8. Borrar el artículo 5 y luego deshacer esa transacción



UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

❑ Realizar las siguientes tareas:

1. Cambia el nombre de la tabla articulos por artic:

ALTER TABLE articulos RENAME TO artic;

2. Añade una nueva columna a la tabla artic que sea su precio_compra:

ALTER TABLE artic ADD (precio_compra integer);

3. Cambiar el tamaño del nombre del fabricante para que sea de 50:

ALTER TABLE fabricantes MODIFY (nombre VARCHAR(50));

4. Añade la restricción a la tabla artic para que el precio_compra no pueda ser menor de 0:

ALTER TABLE artic ADD CONSTRAINT artic_precio CHECK (precio_compra>0);

5. Añade la restricción de que el nombre del fabricante no se pueda repetir:

ALTER TABLE fabricantes ADD CONSTRAINT fabr_nomb UNIQUE (nombre);





UF-5 ADMINISTRACIÓN DE ESTRUCTURA DE DATOS

❑ Realizar las siguientes tareas:

6. Crea una vista de sólo lectura que contenga los nombres de los artículos, su precio y el nombre del fabricante siempre y cuando el precio sea mayor de 25 Euros:

-- Se crea la vista

```
CREATE VIEW artic_60(nom_artic,precio_artic,nom_fabr) AS  
(SELECT artic.nombre, artic.precio, fabricantes.nombre FROM artic  
JOIN fabricantes  
ON artic.fabricante=fabricantes.codigo  
WHERE precio >60) WITH READ ONLY;
```

-- Se muestra el contenido de la vista

```
SELECT * FROM artic_60;
```

7. Listar las restricciones que tiene la tabla de artic accediendo al diccionario de datos de ORACLE. Muestra el nombre de la restricción, el tipo y su cometido:

```
SELECT CONSTRAINT_NAME,CONSTRAINT_TYPE,  
SEARCH_CONDITION FROM user_constraints WHERE table_name='ARTIC';
```

8. Borrar el artículo 5 y luego deshacer esa transacción:

```
DELETE artic WHERE codigo=1005;  
ROLLBACK;
```

