



TEMA4

DISEÑO FÍSICO. DDL

Bases de Datos
CFGs DAW

Raquel Torres
raquel.torres@ceedcv.es
Versión:181111.1510


Licencia




Reconocimiento - NoComercial - CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

 Interesante

Revisiones

-

ÍNDICE DE CONTENIDO

1. Introducción.....	4
2. Creación de una base de datos.....	5
3. Creación de tablas.....	6
3.1 Nombre de las tablas.....	6
3.2 El nombre de cada columna.....	7
3.3 El tipo de dato de cada columna.....	7
3.4 Restricciones.....	8
3.5 Creación de tablas. Comando CREATE TABLE.....	8
3.6 Practicando la creación de tablas.....	11
3.6.1 MySQL.....	12
3.6.2 Oracle.....	13
4. Creación de restricciones en tablas.....	14
4.1 Prohibir nulos (valor no nulo).....	15
4.2 Valores únicos (clave alternativa).....	15
4.3 Clave primaria.....	16
4.4 Clave secundaria o foránea.....	17
4.5 Restricciones de validación.....	19
4.6 Practicando con restricciones.....	20
4.6.1 MySQL.....	20
4.6.2 Oracle.....	21
5. Modificación de tablas.....	22
5.1 Cambiar de nombre a una tabla.....	22
5.2 Borrar contenido de tablas.....	22
5.3 Añadir columnas.....	22
5.4 Borrar columnas.....	23
5.5 Modificar columna.....	23
5.6 Renombrar columna.....	24
5.7 Valor por defecto.....	24
5.8 Añadir restricciones.....	24
5.9 Borrar restricciones.....	24
5.10 Desactivar restricciones.....	25
5.11 Activar restricciones.....	26
5.12 Cambiar de nombre a las restricciones.....	26
5.13 Practicando la modificación de tablas.....	26
5.13.1 MySQL.....	27
5.13.2 Oracle.....	29
6. Borrado de tablas.....	31
6.1 Probando a borrar tablas.....	31
6.1.1 MySQL.....	32
6.1.2 Oracle.....	33
7. Sinónimos.....	33
7.1 Creación de sinónimos.....	34
7.2 Borrado de sinónimos.....	35
8. Consultar el Diccionario.....	35
8.1 Consultar tablas del usuario.....	35
8.2 Obtener la lista de las columnas de las tablas.....	37
8.3 Orden DESCRIBE.....	38
8.4 Consulta de sinónimos.....	39

UD4 DISEÑO FÍSICO. DDL

1. INTRODUCCIÓN

Bien, llegó el momento de implementar físicamente en un SGBD las tablas que hemos conseguido en nuestro diseño de la base de datos.



El **DDL** (Data Definition Language) es la parte del lenguaje SQL que realiza la función de definición de datos del SGBD.

Fundamentalmente se encarga de la creación, modificación y eliminación de los objetos de la base de datos (es decir de los metadatos). Por supuesto es el encargado de la creación de las tablas.

Recuerda que en Oracle, cada usuario de una base de datos posee un esquema. El esquema suele tener el mismo nombre que el usuario y sirve para almacenar los objetos de esquema, es decir los objetos que posee el usuario. Sin embargo en MySQL sí tenemos la posibilidad de crear diferentes Databases (también se pueden denominar Schema).

Esos **objetos** pueden ser: **tablas, vistas, índices** y otros objetos relacionados con la definición de la base de datos. Los objetos son manipulados y creados por los usuarios. En principio sólo los administradores y los usuarios propietarios pueden acceder a cada objeto, salvo que se modifiquen los privilegios del objeto para permitir el acceso a otros usuarios.

Hay que tener en cuenta que ninguna instrucción DDL puede ser anulada por una instrucción ROLLBACK (la instrucción ROLLBACK está relacionada con el uso de transacciones que se comentarán más adelante) por lo que hay que tener mucha precaución a la hora de utilizarlas. Es decir:



Las **instrucciones DDL** generan acciones que **no se pueden deshacer** (salvo que dispongamos de alguna copia de seguridad).

2. CREACIÓN DE UNA BASE DE DATOS

Antes de continuar, recuerda que hemos comentado que en MySQL sí tenemos la posibilidad de crear diferentes Databases (también se pueden denominar Schema), mientras que en Oracle lo hacemos a partir de esquemas de usuarios. Por tanto, veamos cómo se crea una BBDD en MySQL.

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nombre_base_de_datos  
[especificación_create [, especificación_create] ...]  
  
especificación_create:  
[DEFAULT] CHARACTER SET juego_de_caracteres |  
[DEFAULT] COLLATE nombre_de_colación
```

El comando es **CREATE**, después se puede especificar **DATABASE** o **SCHEMA**, es indiferente utilizar cualquiera de las dos para crear la base de datos. Cuando se pueden elegir un elemento entre varios éstos van entre llaves { } y separados por un pipe |.

Por otro lado los elementos que van entre corchetes son opcionales, pueden ir o no, por ejemplo es opcional poner **IF NOT EXIST**, es decir que cree la base de datos si no existe ya.

Por último, puedes observar que la especificación del CREATE viene desarrollada después y que puede ser una especificación del conjunto de caracteres a utilizar o bien la forma de tratar ese conjunto de caracteres.

Por ejemplo si utilizamos **CHARACTER SET utf8**, estamos diciendo que se emplee utf8 para almacenar los caracteres. Si indicamos **COLLATE utf8_spanish_ci** quiere decir que se va a utilizar el alfabeto español para las ordenaciones y las comparaciones en nuestra base de datos. Esto es importante para que al ordenar alfabéticamente la ñ y las letras acentuadas vayan en su sitio y para que al comparar datos acentuados y otros no acentuados sean considerados iguales (é es igual a e) .

Veamos un ejemplo:

```
mysql> CREATE DATABASE prueba1 CHARACTER SET utf8 COLLATE utf8_spanish_ci;  
Query OK, 1 row affected (0.52 sec)  
mysql>
```

MySQL nos indica que el comando se ha ejecutado correctamente (Query OK).

Si intentamos crear de nuevo la misma base de datos se producirá un error.

```
mysql> CREATE DATABASE prueba1 CHARACTER SET utf8 COLLATE utf8_spanish_ci;
ERROR 1007 (HY000): Can't create database 'prueba1'; database exists
mysql>
```

Nos indica que no se ha podido crear la base de datos porque ya existe.

Si queremos evitar este error debemos emplear la clausula opcional IF NOT EXISTS.

```
mysql> CREATE DATABASE IF NOT EXISTS prueba1
-> CHARACTER SET utf8 COLLATE utf8_spanish_ci;
Query OK, 1 row affected, 1 warning (0.00 sec)
```

En este caso la ejecución ha sido correcta y nos muestra un aviso (1 warning) indicando que no se ha creado la base de datos.

Podemos utilizar el comando **show databases** para ver las bases de datos que existen, y que efectivamente se ha creado nuestra tabla prueba1.

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| prueba1 |
| test |
+-----+
5 rows in set (0.00 sec)
```

Por último, para cambiarnos a una base de datos concreta utilizaremos el comando **use** seguido del nombre de la base de datos que vamos a utilizar.

```
mysql> use prueba1;
Database changed
mysql>
```

3. CREACIÓN DE TABLAS

3.1 Nombre de las tablas

Deben cumplir las siguientes reglas: (Estas son reglas de Oracle, pero también son una buena práctica en otros SGBD)

- Deben comenzar con una letra
- No deben tener más de 30 caracteres
- Sólo se permiten utilizar letras del alfabeto (inglés), números o el signo de subrayado (también el signo \$ y #, pero esos se utilizan de manera especial por lo que no son recomendados)
- No puede haber dos tablas con el mismo nombre para el mismo esquema (pueden coincidir

los nombres si están en distintos esquemas)

- No puede coincidir con el nombre de una palabra reservada SQL (por ejemplo no se puede llamar SELECT a una tabla)
- En el caso de que el nombre tenga espacios en blanco o caracteres nacionales (permitido sólo en algunas bases de datos), entonces se suele entrecomillar con comillas dobles. En el estándar SQL 99 (respetado por Oracle) se pueden utilizar comillas dobles al poner el nombre de la tabla, a fin de hacerla sensible a las mayúsculas (se diferenciará entre “FACTURAS” y “facturas”).

3.2 El nombre de cada columna

Ha de ser autodescriptivo. Si partimos de un diseño relacional, ya dispondremos del nombre adecuado de cada columna.

3.3 El tipo de dato de cada columna

Debe ser uno de los tipos permitidos por Oracle o MySQL. Si en el diseño relacional no nos han proporcionado información, tendremos que elegirlo nosotros. Es conveniente dedicar algo de tiempo a pensar bien el tipo y tamaño de cada columna.

Los tipos de datos más habituales que pueden tener los campos de nuestras tablas los podemos ver en las siguientes tablas:

Tipos numéricos:

MySQL	Size	Oracle
BIGINT	8 Bytes	NUMBER (19,0)
BIT	approximately (M+7)/8 Bytes	RAW
DECIMAL(M,D)	M+2 bytes if D > 0, M+1 bytes if D = 0 (D+2, if M < D)	FLOAT(24), BINARY_FLOAT
DOUBLE	8 Bytes	FLOAT(24), BINARY_FLOAT, BINARY_DOUBLE
DOUBLE PRECISION	8 Bytes	FLOAT(24), BINARY_DOUBLE
FLOAT(25<=X<=53)	8 Bytes	FLOAT(24), BINARY_FLOAT
FLOAT(X<=24)	4 Bytes	FLOAT, BINARY_FLOAT
INT	4 Bytes	NUMBER (10,0)
INTEGER	4 Bytes	NUMBER (10,0)
MEDIUMINT	3 Bytes	NUMBER (7,0)
NUMERIC	M+2 bytes if D > 0, M+1 bytes if D = 0 (D+2, if M < D)	NUMBER
REAL	8 Bytes	FLOAT(24), BINARY_FLOAT
SMALLINT	2 Bytes	NUMBER(5,0)
TINYINT	1 Byte	NUMBER(3,0)

Tipos relativos a fechas y horas:

MySQL	Size	Oracle
DATE	3 Bytes	DATE
DATETIME	8 Bytes	DATE
TIMESTAMP	4 Bytes	DATE
TIME	3 Bytes	DATE
YEAR	1 Byte	NUMBER

Tipos String:

MySQL	Size	Oracle
BLOB	L + 2 Bytes whereas $L < 2^{16}$	RAW, BLOB
CHAR(m)	M Bytes, $0 \leq M \leq 255$	CHAR
ENUM (VALUE1, VALUE2, ...)	1 or 2 Bytes depending on the number of enum. values (65535 values max)	
LONGBLOB	L + 4 Bytes whereas $L < 2^{32}$	RAW, BLOB
LONGTEXT	L + 4 Bytes whereas $L < 2^{32}$	RAW, CLOB
MEDIUMBLOB	L + 3 Bytes whereas $L < 2^{24}$	RAW, BLOB
MEDIUMTEXT	L + 3 Bytes whereas $L < 2^{24}$	RAW, CLOB
SET (VALUE1, VALUE2, ...)	1, 2, 3, 4 or 8 Bytes depending on the number of set members (64 members maximum)	
TEXT	L + 2 Bytes whereas $L < 2^{16}$	VARCHAR2, CLOB
TINYBLOB	L + 1 Bytes whereas $L < 2^8$	RAW, BLOB
TINYTEXT	L + 1 Bytes whereas $L < 2^8$	VARCHAR2
VARCHAR(m)	L+1 Bytes whereas $L \leq M$ and $0 \leq M \leq 255$ before MySQL 5.0.3 ($0 \leq M \leq 65535$ in MySQL 5.0.3 and later; effective maximum length is 65,532 bytes)	VARCHAR2, CLOB

Aunque hay una gran cantidad de opciones ya verás como la mayoría de los campos de las tablas que vas a crear serán de tipo INTEGER, DOUBLE y VARCHAR, en este último deberemos indicar su tamaño entre paréntesis, es decir el número máximo de caracteres que tendrá nuestro campo.

3.4 Restricciones

Estarán indicadas en el diseño relacional. Sin embargo, es posible que resulte interesante pensar en si resulta conveniente que ciertas columnas tengan algún valor por defecto.

3.5 Creación de tablas. Comando CREATE TABLE

Los tres comandos SQL que se estudian en esta unidad son **CREATE TABLE**, **ALTER TABLE** y **DROP TABLE**, pertenecientes al DDL. Estos comandos permiten respectivamente crear y modificar la definición de una tabla y eliminarla de la base de datos.



Para la creación de tablas con SQL se utiliza el comando **CREATE TABLE**

Este comando tiene una sintaxis más compleja de la que aquí se expone, pero se van a obviar aquellos detalles que son más complicados.

```
CREATE TABLE [esquema.]nombre_tabla (  
  nombre_columna tipo_datos [ restricción_columna ... ] ...  
  [{nombre_columna tipo_datos [ restricción_columna ... ]...  
  | restricción_tabla}]  
  ...  
  ) tablespace nombre_tablespace;
```

Donde *restricción_columna* tiene la sintaxis:

```
[CONSTRAINT nombre_restricción  
{[NOT] NULL  
| {UNIQUE | PRIMARY KEY}  
| REFERENCES nombre_tabla [(nombre_columna)]  
[ON DELETE {SET NULL|CASCADE}]  
| CHECK (condición)  
}
```

y *restricción_tabla* tiene la sintaxis:

```
[CONSTRAINT nombre_restricción  
{{UNIQUE | PRIMARY KEY} (nombre_columna [,nombre_columna] ...)  
| FOREIGN KEY (nombre_columna [,nombre_columna] ...)  
REFERENCES nombre_tabla [(nombre_columna  
[,nombre_columna] ...)]  
[ON DELETE {SET NULL|CASCADE}]  
| CHECK (condición)  
}
```

Aclaración: Como se puede observar, se han utilizado las mayúsculas para las palabras reservadas de Oracle (cláusulas, tipos y restricciones) mientras que se han usado minúsculas para describir la información introducida por el usuario. Esto es así por cuestiones de legibilidad. Sin embargo, hay que tener en cuenta que Oracle NO hace distinción entre mayúsculas y minúsculas (salvo cuando un texto va entre comillas).



Aunque Oracle lo soporta, **no se recomienda utilizar acentos ni caracteres**

especiales a la hora de identificar el nombre de un objeto (tabla, columna, restricción, etc.) para evitar futuros problemas a la hora de referenciarlos.

El significado de las distintas opciones es:

- **UNIQUE:** impide que se introduzcan valores repetidos para ese atributo. No se puede utilizar junto con PRIMARY KEY.
- **NOT NULL:** evita que se introduzcan tuplas con valor NULL para ese atributo.
- **PRIMARY KEY:** establece ese atributo como la clave primaria de la tabla.
- **PRIMARY KEY *lista_columnas*:** sirve para establecer como clave primaria un conjunto de atributos.
- **FOREIGN KEY:** define una clave externa de la tabla respecto de otra tabla.

Esta restricción especifica una columna o una lista de columnas como de clave externa de una tabla referenciada.

La tabla referenciada se denomina tabla padre de la tabla que hace la referencia llamada tabla hija.

En otras palabras, **no se puede definir una restricción de integridad referencial que se refiere a una tabla antes de que dicha tabla haya sido creada.**

Es importante resaltar que una clave externa debe referenciar a una clave primaria completa de la tabla padre, y nunca a un subconjunto de los atributos que forman esta clave primaria.

En la definición de una tabla pueden aparecer varias cláusulas FOREIGN KEY, tantas como claves externas tenga la tabla, sin embargo sólo puede existir una clave primaria, si bien esta clave primaria puede estar formada por varios atributos.

- **CHECK (*condición*):** permite establecer condiciones que deben cumplir los valores introducidos en ese atributo. La condición puede ser cualquier expresión válida que sea cierta o falsa. Puede contener funciones, atributos (de esa tabla) y literales.

Si un CHECK se especifica como una restricción de columna, la condición sólo se puede referir a esa columna.

Si el CHECK se especifica como restricción de tabla, la condición puede afectar a todas las columnas de la tabla.

Sólo se permiten condiciones simples, por ejemplo, no está permitido referirse a columnas de otras tablas o formular subconsultas dentro de un CHECK. Además las funciones SYSDATE y USER no se pueden utilizar dentro de la condición. En principio están permitidas comparaciones simples de atributos y operadores lógicos (AND, OR y NOT).

- **ON DELETE CASCADE|SET NULL:** especifica que se mantenga automáticamente la integridad referencial borrando o poniendo a nulos los valores de la clave externa correspondientes a un valor borrado de la tabla referenciada (tabla padre).

Si se omite esta opción no se permitirá borrar valores de una tabla que sean referenciados como clave externa en otras tablas.

- **CONSTRAINT *nombre_restricción*:** establece un nombre determinado para la restricción de integridad, lo cual permite buscar en el Diccionario de Datos de la base de datos con

posterioridad y fácilmente las restricciones introducidas para una determinada tabla.

3.6 Practicando la creación de tablas

Vamos a crear una tabla de ejemplo *Empleados* y otra *Departamentos* teniendo en cuenta que todo empleado va a pertenecer a un departamento. Ahora vamos a utilizar pocos campos, simplemente para ir probando y ver el comportamiento de las bases de datos. Más adelante crearemos ejemplos más complejos, más cercanos a la realidad.

Supongamos que tenemos que crear las siguientes tablas:

Tabla Departamentos.

<u>CodDpto</u>	Nombre	Ubicación
INF	Informática	Planta sótano U3
ADM	Administración	Planta quinta U2
COM	Comercial	Planta tercera U3
CONT	Contabilidad	Planta quinta U1
ALM	Almacén	Planta baja U1

Tabla Empleados.

<u>DNI</u>	Nombre	Especialidad	FechaAlta	Dpto ▲
12345678A	Alberto Gil	Contable	10/12/2010	CONT
23456789B	Mariano Sanz	Informática	04/10/2011	INF
34567890C	Iván Gómez	Ventas	20/07/2012	COM
45678901D	Ana Silván	Informática	25/11/2012	INF
56789012E	María Cuadrado	Ventas	02/04/2013	COM
67890123A	Roberto Milán	Logística	05/02/2010	ALM

Para la creación de la tabla *Departamentos* utilizaremos la siguiente sintaxis:

```
CREATE TABLE departamentos (
  CodDpto VARCHAR(10) PRIMARY KEY,
  Nombre VARCHAR(30) NOT NULL,
  Ubicacion VARCHAR(30)
);
```

Podemos probar a escribirlo directamente en la línea de comandos, pero es poco práctico, será muy fácil cometer un error y tener que volver a escribirlo todo, por ello es aconsejable escribirlo con un editor de texto y después ejecutarlo como un script, o bien escribirlo sobre la pestaña SQL de phpMyAdmin o en la hoja de trabajo de SQL_Developer. En la teoría vamos a hacerlo por línea de comando, dejo para vosotr@s la comprobación en el entorno gráfico de todo aquello que veamos por línea de comando.

Prestad atención a cómo hemos establecido el campo *CodDpto* como clave primaria con PRIMARY KEY y cómo hemos indicado que el campo *Nombre* no se puede quedar sin valor con NOT NULL.

3.6.1 MySQL

Para crear estas tablas utilizaremos la base de datos *prueba1* creada en un punto anterior. El primer paso será usar esa base de datos por si no estamos en ella.

```
mysql> use prueba1;
Database changed
mysql>
```

Una vez que estamos en la base de datos correspondiente, escribiremos y ejecutaremos nuestro script con el comando **source** (el script se llama DEPARTAMENTOS.SQL y está almacenado en la ruta C:\SRC).

```
mysql> SOURCE C:\SRC\DEPARTAMENTOS.SQL
Query OK, 0 rows affected (1.72 sec)
```

Podemos comprobar que se ha creado la tabla utilizando el comando **show tables**.

```
mysql> show tables;
+-----+
| Tables_in_prueba1 |
+-----+
| departamentos     |
+-----+
1 row in set (0.00 sec)
```

También podemos ver la composición de una tabla de la base de datos con el comando **desc** o **describe** y el nombre de la tabla.

```
mysql> DESC DEPARTAMENTOS;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| CodDpto | varchar(10) | NO   | PRI | NULL    |       |
| Nombre  | varchar(30) | NO   |     | NULL    |       |
| Ubicacion | varchar(30) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.41 sec)
```

Bien, pues ya tenemos la tabla preparada y hemos comprobado que se ha creado correctamente.

MySQL puede utilizar varios tipos de tabla para trabajar, para este curso nos interesa trabajar con **InnoDB**. Si has instalado la versión que hemos aconsejado en este curso, por defecto las tablas se crearán de este tipo. Si no es así debemos especificarlo en la creación de la tabla añadiendo entre el paréntesis de cierre y el punto y coma la expresión **ENGINE = InnoDB** quedando la instrucción:

```
CREATE TABLE departamentos (
    CodDpto VARCHAR(10) PRIMARY KEY,
    Nombre VARCHAR(30) NOT NULL,
    Ubicacion VARCHAR(30)
) ENGINE=InnoDB;
```

Podemos saber cómo está creada nuestra tabla con la instrucción:

```
SHOW CREATE TABLE nombre_tabla;
```

Por ejemplo:

```
mysql> show create table departamentos;
+-----+
| Table           | Create Table                               |
+-----+-----+
| departamentos | CREATE TABLE `departamentos` (
  `CodDpto` varchar(10) collate utf8_spanish_ci NOT NULL,
  `Nombre` varchar(30) collate utf8_spanish_ci NOT NULL,
  `Ubicacion` varchar(30) collate utf8_spanish_ci default NULL,
  PRIMARY KEY (`CodDpto`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_spanish_ci |
```

Puedes ver en la última línea el tipo de tabla, el conjunto de caracteres que utiliza y su colación.

3.6.2 Oracle

Para hacer esta prueba en Oracle vamos a utilizar el *usuario_prueba* que habíamos creado anteriormente (en nuestro caso *alumno/alumno*). Primero nos conectamos con dicho usuario.

```
SQL> conn usuario_prueba/clave_prueba
Connected.
```

Después ejecutamos el script de creación de la tabla. Recuerda que en Oracle el script se ejecuta con una @

```
SQL> @ c:\src\departamentos.sql
Table created.
```

Para ver las tablas que tenemos creadas debemos utilizar el comando **select** para seleccionar el nombre de las tablas (**table_name**) de una tabla especial llamada **user_tables** donde se almacena la información de las tablas que creamos.

```
SQL> select table_name from user_tables;
TABLE_NAME
-----
DEPARTAMENTOS
```

Por último para ver la estructura de la tabla utilizaremos el comando **desc** o **describe** que ya hemos empleado con MySQL.

```
SQL> desc departamentos;
Name
-----
CODDPTO
NOMBRE
UBICACION
SQL>
```

Null?

Type

NOT NULL VARCHAR2(10)
NOT NULL VARCHAR2(30)
VARCHAR2(30)

Vamos a estudiar las restricciones antes de crear la tabla *Empleados*.

4. CREACIÓN DE RESTRICCIONES EN TABLAS



Una **restricción** es una condición de obligado cumplimiento para una o más columnas de la tabla.

A cada restricción se le pone un **nombre**, en el caso de no poner un nombre (algo poco recomendable) entonces el propio Oracle le coloca el nombre que es un mnemotécnico con el nombre de tabla, columna y tipo de restricción.

Los **nombres** de restricción **no se pueden repetir para el mismo esquema**, debemos de buscar **nombres únicos**. Es buena idea incluir de algún modo el nombre de la tabla, los campos involucrados y el tipo de restricción en el nombre de la misma. Por ejemplo *pieza_id_pk* podría indicar que el campo *id* de la tabla *pieza* tiene una clave principal (PRIMARY KEY).

NOTACIÓN NOMBRE RESTRICCIONES

Desde la empresa Oracle se aconseja la siguiente regla a la hora de poner nombre a las restricciones:

- Tres letras para el nombre de la tabla
- Carácter de subrayado
- Tres letras con la columna afectada por la restricción
- Carácter de subrayado
- Dos letras con la abreviatura del tipo de restricción. La abreviatura puede ser:
 - NN.** NOT NULL.
 - PK.** PRIMARY KEY
 - UK.** UNIQUE
 - FK.** FOREIGN KEY
 - CK.** CHECK (validación)

Por ejemplo para hacer que la clave principal de la tabla Alumnos sea el código del alumno, el nombre de la restricción podría ser: *alu_cod_pk*

4.1 Prohibir nulos (valor no nulo)



La restricción **NOT NULL** permite prohibir los nulos en una determinada tabla. Eso obliga a que la columna tenga que tener obligatoriamente un valor para que sea almacenado el registro.

Se puede especificar durante la creación (o modificación) del campo añadiendo la palabra NOT NULL tras el tipo:

```
CREATE TABLE cliente (  
  dni VARCHAR2(9) NOT NULL);
```

En ese caso el nombre lo coloca la propia base de datos (en el caso de Oracle el nombre sería algo como SY002341 por ejemplo). Para especificar nosotros el nombre se usa:

```
CREATE TABLE cliente(  
  dni VARCHAR2(9) CONSTRAINT cli_dni_nn NOT NULL);
```

4.2 Valores únicos (clave alternativa)



Las restricciones de tipo **UNIQUE** obligan a que el contenido de una o más columnas no puedan repetir valores.

Nuevamente hay dos formas de colocar esta restricción. Si no se especifica el nombre, este lo pondrá el sistema:

```
CREATE TABLE cliente(  
  dni VARCHAR2(9) UNIQUE);
```

O podemos indicar el nombre de la restricción de la siguiente manera:

```
CREATE TABLE cliente (  
  dni VARCHAR2(9) CONSTRAINT cli_dni_uk UNIQUE);
```

Si la restricción se refiere a varios campos, la forma sería:

```
CREATE TABLE alquiler (  
  dni VARCHAR2(9),  
  cod_pelicula NUMBER(5),  
  CONSTRAINT alq_dnicod_uk UNIQUE(dni,cod_pelicula));
```

La coma tras la definición del campo `cod_pelicula` hace que la restricción sea independiente de ese campo. Eso obliga a que, tras `UNIQUE` se indique la lista de campos. Incluso para un solo campo se puede colocar la restricción al final de la lista en lugar de definirlo a continuación del nombre y tipo de la columna.

4.3 Clave primaria



La **clave primaria (PRIMARY KEY)** de una tabla la forman la columna o columnas que identifican a cada registro de la misma. La clave primaria hace que los campos que la forman sean **NOT NULL** (sin posibilidad de quedar vacíos) y que los valores de los campos sean de tipo **UNIQUE** (sin posibilidad de repetición).

Si la clave está formada por un solo campo basta con definirla de la siguiente manera:

```
CREATE TABLE cliente (  
  dni VARCHAR(9) PRIMARY KEY,  
  nombre VARCHAR(50));
```

O, poniendo un nombre a la restricción:

```
CREATE TABLE cliente (  
  dni VARCHAR(9) CONSTRAINT cli_dni_pk PRIMARY KEY,  
  nombre VARCHAR(50));
```

Si la clave está formada por más de un campo:

```
CREATE TABLE alquiler (  
  dni VARCHAR(9),  
  cod_pelicula NUMBER(5),  
  CONSTRAINT alq_pk PRIMARY KEY(dni,cod_pelicula));
```

En este caso no podemos ponerlo de la primera forma.

4.4 Clave secundaria o foránea



Una **clave secundaria o foránea (FOREIGN KEY)**, es uno o más campos de una tabla que están relacionados con la clave principal de otra tabla.

La forma de indicar una clave foránea (aplicando una restricción de integridad referencial) es haciendo referencia a la tabla y el campo/s con la que se relaciona a través de la palabra clave **REFERENCES**:

```
CREATE TABLE alquiler (
dni VARCHAR2(9) CONSTRAINT alq_dni_fk REFERENCES clientes(dni),
cod_pelicula NUMBER(5) CONSTRAINT alq_cod_fk REFERENCES peliculas(cod),
CONSTRAINT alq_pk PRIMARY KEY(dni,cod_pelicula));
```

Que significa que el campo *dni* de la tabla *alquiler* se relaciona con el campo *dni* de la tabla *clientes*.

Si el campo al que se hace referencia es la clave principal, se puede obviar el nombre del campo:

```
CREATE TABLE alquiler (
dni VARCHAR2(9) CONSTRAINT alq_dni_fk REFERENCES clientes,
cod_pelicula NUMBER(5) CONSTRAINT alq_cod_fk REFERENCES peliculas,
CONSTRAINT alq_pel_pk PRIMARY KEY(dni,cod_pelicula));
```

En este caso se entiende que los campos hacen referencia a las claves principales de las tablas referenciadas (si la relación la forma más un campo, el orden de los campos debe de ser el mismo).

Se forma una relación entre ambas tablas, que además obliga al cumplimiento de la integridad referencial. Esta integridad obliga a que cualquier *dni* incluido en la tabla *alquiler* tenga que estar obligatoriamente en la tabla de *clientes*. De no ser así el registro no será insertado en la tabla (ocurrirá un error).

Otra forma de crear claves foráneas, en el caso de claves formadas por más de un campo es:

```
CREATE TABLE existencias (
tipo CHAR2(9),
modelo NUMBER(3),
n_almacen NUMBER(1),
cantidad NUMBER(7),
CONSTRAINT exi_tip_mod_fk FOREIGN KEY(tipo,modelo) REFERENCES piezas,
CONSTRAINT exi_nal_fk FOREIGN KEY(n_almacen) REFERENCES almacenes,
CONSTRAINT exi_pk PRIMARY KEY(tipo,modelo,n_almacen));
```

Si la definición de clave secundaria se pone al final, hace falta colocar el texto **FOREIGN KEY** para indicar en qué campos se coloca la restricción de clave foránea. En el ejemplo anterior es absolutamente necesario que la clave principal de la tabla *piezas*, a la que hace referencia la clave foránea, la formen las columnas *tipo* y *modelo* y además que estén en ese orden.

La integridad referencial es una herramienta imprescindible de las bases de datos relacionales. Pero provoca varios problemas. Por ejemplo, si borramos un registro en la tabla principal que está relacionado con uno o varios de la secundaria ocurrirá un error, ya que de permitírse nos borrar el registro ocurrirá fallo de integridad (habrá claves secundarias refiriéndose a una clave principal que ya no existe).

Por ello se nos pueden ofrecer soluciones a añadir tras la cláusula REFERENCES.

En SQL estándar:

ON DELETE SET NULL. Coloca nulos en todas las claves secundarias relacionadas con la borrada.

ON DELETE CASCADE. Borra todos los registros cuya clave secundaria es igual que la clave del registro borrado.

ON DELETE SET DEFAULT. Coloca en el registro relacionado el valor por defecto en la columna relacionada.

ON DELETE NOTHING. No hace nada.

En esas cuatro cláusulas se podría sustituir la palabra **DELETE** por la palabra **UPDATE**, haciendo que el funcionamiento se refiera a cuando se modifica un registro de la tabla principal, en lugar de cuando se borra; en muchas bases de datos se admite el uso tanto de ON DELETE como de ON UPDATE, como en el caso de MySQL, pero:

⚡ En la base de datos **Oracle** sólo se permite utilizar ON DELETE SET NULL u ON DELETE CASCADE. **No se admite** el uso de **ON UPDATE** en ningún caso.

La sintaxis completa para añadir claves foráneas es:

```
CREATE TABLE tabla (lista_de_campos
CONSTRAINT nombreRestriccion FOREIGN KEY (listaCampos)
REFERENCES tabla(clavePrincipalRelacionada)
[ON DELETE|ON UPDATE [[SET NULL|CASCADE|DEFAULT]|NO ACTION]])
```

Si es de un solo campo existe esta alternativa:

```
CREATE TABLE tabla(lista_de_campos tipos propiedades,
nombreCampoClaveSecundaria CONSTRAINT nombreRestriccion
REFERENCES tabla(clavePrincipalRelacionada)
[ON DELETE|ON UPDATE [[SET NULL|CASCADE|DEFAULT]|NO ACTION]]);
```

Ejemplo

```
CREATE TABLE alquiler(
dni VARCHAR(9),
cod_pelicula NUMBER(5),
CONSTRAINT alq_pk PRIMARY KEY(dni,cod_pelicula),
CONSTRAINT alq_dni_fk FOREIGN KEY (dni)
REFERENCES clientes(dni) ON DELETE SET NULL,
CONSTRAINT alq_pel_fk FOREIGN KEY (cod_pelicula)
REFERENCES peliculas(cod) ON DELETE CASCADE );
```

4.5 Restricciones de validación



Son restricciones que dictan una condición que deben cumplir los contenidos de una columna.

Una misma columna puede tener múltiples **CHECK** en su definición (se pondrían varios **CONSTRAINT** seguidos, sin comas).

Ejemplo

```
CREATE TABLE ingresos(
cod NUMBER(5) PRIMARY KEY,
concepto VARCHAR2(40) NOT NULL,
importe NUMBER(11,2),
CONSTRAINT ing_imp_min_ck CHECK (importe>0)
CONSTRAINT ing_imp_max_ck CHECK (importe<8000));
```

En este caso la CHECK prohíbe añadir datos cuyo importe no esté entre 0 y 8000.

Para poder hacer referencia a otras columnas hay que construir la restricción de forma independiente a la columna (es decir al final de la tabla):

```
CREATE TABLE ingresos (
cod NUMBER(5) PRIMARY KEY,
```

```

concepto VARCHAR2(40) NOT NULL,
importe_max NUMBER(11,2),
importe NUMBER(11,2),
CONSTRAINT ing_imp_max_ck CHECK (importe<importe_max));

```

4.6 Practicando con restricciones

Recordemos que teníamos pendiente la creación de la segunda tabla, *Empleados*.

DNI	Nombre	Especialidad	FechaAlta	Dpto ▲
12345678A	Alberto Gil	Contable	10/12/2010	CONT
23456789B	Mariano Sanz	Informática	04/10/2011	INF
34567890C	Iván Gómez	Ventas	20/07/2012	COM
45678901D	Ana Silván	Informática	25/11/2012	INF
56789012E	María Cuadrado	Ventas	02/04/2013	COM
67890123A	Roberto Milán	Logística	05/02/2010	ALM

En nuestra tabla tenemos una clave primaria (DNI) y una clave foránea o extranjera (Dpto).

4.6.1 MySQL

La instrucción para crear la tabla será:

```

CREATE TABLE empleados(
    dni VARCHAR(10) ,
    nombre VARCHAR(30) ,
    especialidad VARCHAR(25) ,
    fechaalta DATE ,
    dpto VARCHAR(10) ,
    PRIMARY KEY (dni) ,
    FOREIGN KEY (dpto) REFERENCES departamentos(CodDpto)
    ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB;

```

En esta instrucción estamos creando la clave principal y la foránea a nivel de tabla (personalmente creo que se ve más claro y además es la única forma de hacerlo para claves compuestas).

Además estamos indicando que para conservar la integridad referencial si se elimina el registro de la tabla *Departamentos* se eliminarán los registros de la tabla *Empleados* (ON DELETE CASCADE) y si se actualiza la clave principal del departamento también se actualizará en los empleados de ese departamento (ON UPDATE CASCADE [Recordemos que esta opción no está disponible en Oracle]).

Para crear esta tabla escribiremos el contenido en un documento de texto y ejecutaremos el script

correspondiente.

```
mysql> source c:\src\empleados.sql
Query OK, 0 rows affected (0.08 sec)

mysql> desc empleados;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| dni    | varchar(10) | NO | PRI | NULL | |
| nombre | varchar(30) | YES | | NULL | |
| especialidad | varchar(25) | YES | | NULL | |
| fechaalta | date | YES | | NULL | |
| dpto   | varchar(10) | YES | MUL | NULL | |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.02 sec)
```

4.6.2 Oracle

Recuerda que en Oracle no podemos utilizar la cláusula ON UPDATE, por ello la instrucción a utilizar en Oracle será:

```
CREATE TABLE empleados(
    dni VARCHAR(10),
    nombre VARCHAR(30),
    especialidad VARCHAR(25),
    fechaalta DATE,
    dpto VARCHAR(10),
    PRIMARY KEY (dni),
    FOREIGN KEY (dpto) REFERENCES departamentos(CodDpto)
    ON DELETE CASCADE
);
```

Como antes, lo escribiremos en un archivo de texto y ejecutaremos el script. Hasta ahora hemos ejecutado el Script con @ que es el modo abreviado, pero también se pueden ejecutar con **START** seguido el nombre del script, como en este ejemplo:

```
SQL> START C:\SRC\EMPLEADOS.SQL
Table created.
```

Comprobamos el resultado.

```
SQL> DESC EMPLEADOS;
```

Name	Null?	Type
DNI	NOT NULL	VARCHAR2(10)
NOMBRE		VARCHAR2(30)
ESPECIALIDAD		VARCHAR2(25)
FECHAALTA		DATE
DPTO		VARCHAR2(10)

5. MODIFICACIÓN DE TABLAS

5.1 Cambiar de nombre a una tabla

De forma estándar (SQL estándar) se hace:

```
ALTER TABLE nombreViejo RENAME TO nombreNuevo;
```

En Oracle se realiza mediante la orden **RENAME** (que permite el cambio de nombre de cualquier objeto):

```
RENAME nombreViejo TO nombreNuevo;
```

5.2 Borrar contenido de tablas

Oracle dispone de una orden no estándar para eliminar definitivamente los datos de una tabla; es la orden **TRUNCATE TABLE** seguida del nombre de la tabla a borrar. Hace que se elimine el contenido de la tabla, pero no la estructura de la tabla en sí. Incluso borra del archivo de datos el espacio ocupado por la tabla.

```
TRUNCATE TABLE [usuario.]nom_tabla [{DROP | REUSE} STORAGE]
```

5.3 Añadir columnas

```
ALTER TABLE nombreTabla ADD (nombreColumna TipoDatos [Propiedades]  
[,columnaSiguiente tipoDatos [propiedades], ...])
```

Permite añadir nuevas columnas a la tabla. Se deben indicar su tipo de datos y sus propiedades si es necesario (como CREATE TABLE).

En Oracle, las nuevas columnas se añaden al final, no se puede indicar otra posición. En MySQL sí se permite (hay que recordar que el orden de las columnas no importa).

```
ALTER TABLE facturas ADD (fecha DATE);  
ALTER TABLE facturas ADD (fecha DATE) AFTER dni; (sólo MySQL)
```

5.4 Borrar columnas

Elimina la columna indicada de manera irreversible e incluyendo los datos que contenía.

```
ALTER TABLE nombreTabla DROP ([,columnaSiguiente,...]);
```

No se puede eliminar la única columna de una tabla que sólo tiene esa columna (habrá que usar **DROP TABLE**).

```
ALTER TABLE facturas DROP (fecha);
```

Al igual que en el caso anterior, en SQL estándar se puede escribir el texto **COLUMN** tras la palabra **DROP**.

5.5 Modificar columna

Permite cambiar el tipo de datos y propiedades de una determinada columna.

```
ALTER TABLE nombreTabla MODIFY (columna tipo [propiedades] [columnaSiguiente  
tipo [propiedades] ...])
```

Los cambios que se permiten son (en Oracle):

- Incrementar precisión o anchura de los tipos de datos.
- Sólo se puede reducir la anchura si la anchura máxima de un campo si esa columna posee nulos en todos los registros, o todos los valores son tan pequeños como la nueva anchura o no hay registros. Se puede pasar de **CHAR** a **VARCHAR2** y viceversa (si no se modifica la anchura).
- Se puede pasar de **DATE** a **TIMESTAMP** y viceversa.
- Cualquier otro cambio sólo es posible si la tabla está vacía.

Ejemplo:

```
ALTER TABLE facturas MODIFY(fecha TIMESTAMP);
```

En el caso de SQL estándar en lugar de **MODIFY** se emplea **ALTER** (que además opcionalmente puede ir seguida de **COLUMN**). Por ejemplo:

```
ALTER TABLE facturas ALTER COLUMN fecha TIMESTAMP;
```

5.6 Renombrar columna

Permite cambiar el nombre de una columna.

```
ALTER TABLE nombreTabla RENAME COLUMN nombreAntiguo TO nombreNuevo
```

Ejemplo:

```
ALTER TABLE facturas RENAME COLUMN fecha TO fechaYhora;
```

5.7 Valor por defecto

A cada columna se le puede asignar un valor por defecto durante su creación mediante la propiedad **DEFAULT**. Se puede incluir esta propiedad durante la creación o modificación de la tabla (**ALTER TABLE**), añadiendo la palabra **DEFAULT** tras el tipo de datos del campo y colocando detrás el valor que se desea por defecto.

Ejemplo

```
CREATE TABLE articulo (  
    cod NUMBER(7),  
    nombre VARCHAR2(25),  
    precio NUMBER(11,2) DEFAULT 3.5);
```

5.8 Añadir restricciones

Es posible querer añadir restricciones tras haber creado la tabla. En ese caso se utiliza la siguiente sintaxis:

```
ALTER TABLE tabla ADD [CONSTRAINT nombre] tipoDeRestricción(columnas);  
tipoRestricción es el texto CHECK, PRIMARY KEY o FOREIGN KEY.
```

Las restricciones **NOT NULL** deben indicarse mediante **ALTER TABLE ... MODIFY** colocando **NOT NULL** en el campo que se modifica.

5.9 Borrar restricciones

```
ALTER TABLE tabla DROP {PRIMARY KEY | UNIQUE(campos) | CONSTRAINT  
nombreRestricción [CASCADE]}
```

La opción **PRIMARY KEY** elimina una clave principal (también quitará el índice **UNIQUE** sobre las campos que formaban la clave. **UNIQUE** elimina índices únicos. La opción **CONSTRAINT** elimina la restricción indicada.

La opción **CASCADE** hace que se eliminen en cascada las restricciones de integridad que dependen de la restricción eliminada.

Ejemplo:

```
CREATE TABLE curso (  
    cod_curso CHAR(7) PRIMARY KEY,  
    fecha_inicio DATE,  
    fecha_fin DATE,  
    titulo VARCHAR2(60),  
    cod_siguientecurso CHAR(7),  
    CONSTRAINT cur_fec_ck CHECK(fecha_fin>fecha_inicio),  
    CONSTRAINT cur_csi_fk FOREIGN KEY(cod_siguientecurso)  
    REFERENCES curso ON DELETE SET NULL);
```

Tras esa definición de tabla, esta instrucción:

```
ALTER TABLE curso DROP PRIMARY KEY;
```

Produce este error (en Oracle): ORA-02273: a esta clave única/primaria hacen referencia algunas claves ajenas.

Para ello habría que utilizar esta instrucción:

```
ALTER TABLE curso DROP PRIMARY KEY CASCADE;
```

Esa instrucción elimina la restricción de clave secundaria antes de eliminar la principal.

También produce error esta instrucción:

```
ALTER TABLE curso DROP (fecha_inicio);
```

ERROR en línea 1: ORA-12991: se hace referencia a la columna en una restricción de multicolumna.

El error se debe a que no es posible borrar una columna que forma parte de la definición de una instrucción. La solución es utilizar **CASCADE CONSTRAINTS** que elimina las restricciones en las que la columna a borrar estaba implicada:

```
ALTER TABLE curso DROP(fecha_inicio) CASCADE CONSTRAINTS;
```

Esta instrucción elimina la restricción de tipo CHECK en la que aparecía la fecha_inicio y así se puede eliminar la columna.

En SQL estándar sólo se pone **CASCADE** y no **CASCADE CONSTRAINTS**.

5.10 Desactivar restricciones

A veces conviene temporalmente desactivar una restricción para saltarse las reglas que impone. La sintaxis es (en Oracle):

```
ALTER TABLE tabla DISABLE CONSTRAINT nombre [CASCADE]
```

La opción **CASCADE** hace que se desactiven también las restricciones dependientes de la que se desactivó.

5.11 Activar restricciones

Esta sentencia anula la desactivación. Su formato (Oracle) es el siguiente:

```
ALTER TABLE tabla ENABLE CONSTRAINT nombre [CASCADE]
```

Sólo se permite volver a activar si los valores de la tabla cumplen la restricción que se activa. Si hubo desactivado en cascada, habrá que activar cada restricción individualmente.

5.12 Cambiar de nombre a las restricciones

Para hacerlo se utiliza este comando (Oracle):

```
ALTER TABLE table RENAME CONSTRAINT nombreViejo TO nombreNuevo;
```

5.13 Practicando la modificación de tablas

Supongamos que además de las dos tablas con las que ya trabajamos necesitamos crear la tabla *Proyectos* para guardar la información de los proyectos en los que trabaja la empresa. Inicialmente esa tabla se crea con dos campos, el *código del proyecto* y el *nombre del proyecto* pues la empresa aún no tiene muy claro cuál va a ser su tratamiento. Para ello se pide crear la siguiente tabla:

Tabla *Proyectos*.

<u>CodProy</u>	Nombre
MAD20	Repsol, S.A.
TO451	Consejería de Educación
V324	Oceanográfico

(Intenta hacerla sin mirar y después comprueba si ha funcionado correctamente).

Ahora vamos a modificar la estructura de la tabla que acabamos de crear empleando para ello la instrucción **ALTER TABLE** que comentamos anteriormente.

Después de trabajar con la tabla *Proyectos* se ha llegado a las siguientes conclusiones:

- El campo *nombre* es un poco pequeño y se debe cambiar de 25 caracteres a 30.
- Sería interesante disponer de dos campos *FechaInicio* y *FechaFin* de tipo fecha.

Lo que tiene no planificar bien las cosas, ahora, después de hacernos crear el campo *fechaFin* se han dado cuenta de que no lo necesitan y quieren que lo eliminemos de la tabla.

Para ello utilizaremos la cláusula `DROP COLUMN` de `ALTER TABLE`.

Bien, después de tanto crear y eliminar, el administrador de la BD se ha sentado con el responsable para plantear de una vez para qué quieren utilizar la tabla *proyectos*. Después de algunas conversaciones han llegado a las siguientes conclusiones.

En la tabla *proyectos* se debe añadir un campo más para indicar el departamento al que pertenece (*Dpto*). Además este campo actuará como clave foránea de la tabla *departamentos* con puesta a nulo en caso de eliminación del departamento y (para mysql) con actualización en cascada.

También se quiere añadir un campo llamado *Responsable* para guardar la persona que es o fue responsable de ese proyecto. El valor que se guardará en este campo será el *DNI* del empleado correspondiente, por lo tanto también será una clave foránea de la tabla *empleados*.

Por último se quiere añadir a la tabla *empleados* un campo *CodP* que indicará el proyecto en el que está trabajando actualmente (solo puede trabajar en 1 como mucho, es decir puede haber empleados que no estén trabajando en un proyecto por estar desempeñando otras funciones).

Realicemos todos estos cambios a nuestras tablas!

5.13.1 MySQL

Primero creamos el script para crear la tabla en nuestro editor de texto.

```
CREATE TABLE proyectos (  
    codproy VARCHAR(10),  
    nombre VARCHAR(25),  
    PRIMARY KEY (codproy)  
) ENGINE=InnoDB;
```

Ejecutamos el script.

```
mysql> source c:\src\proyectos.sql  
Query OK, 0 rows affected (0.08 sec)
```

Observamos el resultado.

```
mysql> show tables;
+-----+
| Tables_in_pruebas_curso |
+-----+
| departamentos           |
| empleados               |
| proyectos               |
+-----+
3 rows in set (0.00 sec)
```

Para modificar las características de un campo utilizaremos la cláusula **MODIFY** seguido del nombre del campo y de las nuevas características. Veamos cómo hacerlo. Primero sacamos la estructura de la tabla con el comando **desc** para comprobar el estado actual y después procedemos a modificar el tamaño de la columna *nombre* y volvemos a mostrar la descripción de la tabla para verificar que hemos pasado de 25 a 30 caracteres en el campo.

```
mysql> desc proyectos;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codproy | varchar(10) | NO   | PRI | NULL    |       |
| nombre  | varchar(25) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> alter table proyectos
-> modify nombre varchar(30);
Query OK, 3 rows affected (0.30 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> desc proyectos;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codproy | varchar(10) | NO   | PRI | NULL    |       |
| nombre  | varchar(30) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Ahora vamos añadir los campos de tipo fecha:

```
mysql> desc proyectos;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codproy | varchar(10) | NO   | PRI | NULL    |       |
| nombre  | varchar(30) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)

mysql> alter table proyectos
-> add (fechainicio date, fechafin date);
Query OK, 3 rows affected (0.31 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> desc proyectos;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codproy | varchar(10) | NO   | PRI | NULL    |       |
| nombre  | varchar(30) | YES  |     | NULL    |       |
| fechainicio | date      | YES  |     | NULL    |       |
| fechafin | date      | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)
```

Como podemos observar se han añadido los dos campos indicados. Pero ahora nos toca eliminar el campo *fechafin*:

```
mysql> alter table proyectos
-> drop column fechafin;
Query OK, 3 rows affected (0.26 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

Como hay que realizar varios cambios más, vamos a ir creándolos paso a paso. El primer cambio implica la creación de dos campos y de dos claves foráneas, para ello crearemos un script y lo ejecutaremos:

```
ALTER TABLE proyectos
ADD( dpto varchar(10), responsable varchar(10),
FOREIGN KEY (dpto) REFERENCES departamentos (CodDpto)
ON DELETE SET NULL ON UPDATE CASCADE,
FOREIGN KEY (responsable) REFERENCES empleados (dni)
ON DELETE SET NULL ON UPDATE CASCADE);
```

Ahora crearemos otro script para crear el nuevo campo *codp* en la tabla *empleados*.

```
ALTER TABLE empleados
ADD( codp varchar(10),
FOREIGN KEY (codp) REFERENCES proyectos (codproy)
ON DELETE SET NULL ON UPDATE CASCADE);
```

5.13.2 Oracle

Hacemos las mismas operaciones en Oracle.

```
CREATE TABLE proyectos (
codproy VARCHAR(10),
nombre VARCHAR(25),
PRIMARY KEY (codproy)
);
```

Ejecutamos el script y comprobamos el resultado.

```
SQL> @ c:\src\proyectos.sql
Table created.
SQL> select table_name from user_tables;
TABLE_NAME
-----
DEPARTAMENTOS
EMPLEADOS
PROYECTOS
```

Ahora procedemos a modificar el tamaño del campo *nombre*:

```
SQL> desc proyectos;
Name                                     Null?      Type
-----
CODPROY                                NOT NULL   VARCHAR2(10)
NOMBRE                                  NOT NULL   VARCHAR2(25)

SQL> alter table proyectos
2  modify nombre varchar(30);

Table altered.

SQL> desc proyectos;
Name                                     Null?      Type
-----
CODPROY                                NOT NULL   VARCHAR2(10)
NOMBRE                                  NOT NULL   VARCHAR2(30)
```

Y ahora añadimos los campos de tipo fecha.

```
SQL> alter table proyectos
2  add (fechainicio date, fechafin date);

Table altered.

SQL> desc proyectos;
Name                                     Null?      Type
-----
CODPROY                                NOT NULL   VARCHAR2(10)
NOMBRE                                  NOT NULL   VARCHAR2(30)
FECHAINICIO                            NOT NULL   DATE
FECHAFIN                                NOT NULL   DATE
```

Eliminemos ahora el campo *fechafin*:

```
SQL> alter table proyectos
2  drop column fechafin;

Table altered.
```

Y por último, los cambios finales. El script para la tabla *proyectos* será:

```
ALTER TABLE proyectos
ADD( dpto varchar(10), responsable varchar(10),
FOREIGN KEY (dpto) REFERENCES departamentos (CodDpto)
ON DELETE SET NULL,
FOREIGN KEY (responsable) REFERENCES empleados (dni)
ON DELETE SET NULL);
```

El script para modificar la tabla *empleados* será:

```
ALTER TABLE empleados
ADD( codp varchar(10),
FOREIGN KEY (codp) REFERENCES proyectos (codproy)
ON DELETE SET NULL);
```

Espero que hayas llegado hasta aquí sin tener que mirar cómo hacerlo o mirando poco. Eso quiere decir que vamos por el buen camino.

6. BORRADO DE TABLAS



La orden **DROP TABLE** seguida de un nombre de tabla, permite eliminar la tabla con ese nombre.

Al borrar una tabla:

- Desaparecen todos los datos.
- Cualquier vista y sinónimo referente a la tabla seguirá existiendo, pero ya no funcionará (conviene eliminarlos).
- Lógicamente, sólo se pueden eliminar las tablas sobre las que tenemos permiso de borrado.



Normalmente, el **borrado** de una tabla es **irreversible**, y no hay ninguna petición de confirmación, por lo que conviene ser muy cuidadoso con esta operación.

Para borrar nuestras tablas no es necesario poner nuestro nombre, seguido de un punto, delante del nombre de la tabla. Sin embargo, si queremos borrar la tabla de otro usuario será necesario indicar el usuario al que pertenece la tabla a eliminar.

En Oracle se permite incluir la cláusula **CASCADE CONSTRAINTS** que elimina las restricciones de integridad referencial que remitan a la clave primaria de la tabla borrada. Si no se pone y hay alguna clave ajena que apunta a la clave primaria de la tabla a borrar, el borrado de la tabla dará error.

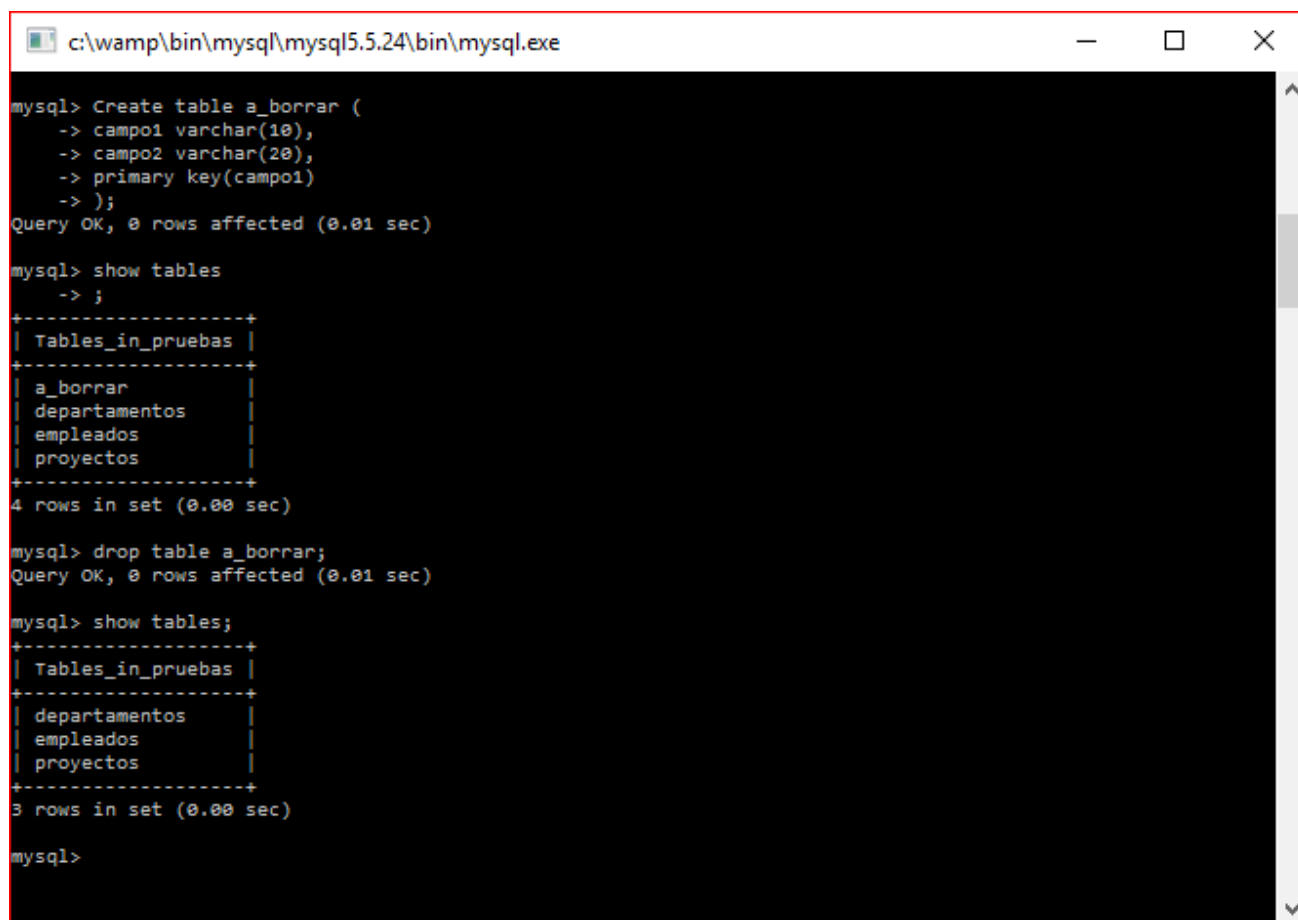
```
DROP TABLE [usuario.]nom_tabla [CASCADE CONSTRAINTS]
```

6.1 Probando a borrar tablas

Para eliminar una tabla completamente en MySQL utilizaremos la sintaxis:

```
DROP TABLE nombre_tabla;
```

Primero vamos a crear una tabla sin dependencias (a *borrar*) para poder borrarla sin problemas:



```
c:\wamp\bin\mysql\mysql5.5.24\bin\mysql.exe

mysql> Create table a_borrar (
  -> campo1 varchar(10),
  -> campo2 varchar(20),
  -> primary key(campo1)
  -> );
Query OK, 0 rows affected (0.01 sec)

mysql> show tables
  -> ;
+-----+
| Tables_in_pruebas |
+-----+
| a_borrar           |
| departamentos      |
| empleados          |
| proyectos          |
+-----+
4 rows in set (0.00 sec)

mysql> drop table a_borrar;
Query OK, 0 rows affected (0.01 sec)

mysql> show tables;
+-----+
| Tables_in_pruebas |
+-----+
| departamentos     |
| empleados         |
| proyectos         |
+-----+
3 rows in set (0.00 sec)

mysql>
```

En Oracle, lo haríamos de la misma manera.

Ya sabemos cómo borrar una tabla. Pero, ¿qué ocurrirá si borramos una tabla que está relacionada con otras y alguno de sus campos es clave foránea en otras tablas? ¿Quedará inconsistente nuestra base de datos?

Veamos cómo se comporta cada una de nuestras bases de datos ante esta circunstancia. Vamos a intentar eliminar la tabla *departamentos* que como ya sabes su clave principal actúa como clave foránea en la tabla *empleados* y en la tabla *proyectos*. ¿Qué ocurrirá?

6.1.1 MySQL

Cuando intentamos eliminar la tabla *departamentos* en MySQL ocurre lo siguiente:

```
mysql> drop table departamentos;
ERROR 1217 (23000): Cannot delete or update a parent row: a foreign key constraint fails
```

Efectivamente, no nos permite borrar la tabla ya que tiene filas que están relacionadas con otros registros en los que actúa como FOREIGN KEY. De esta forma MySQL asegura la integridad de la información de nuestra base de datos.

6.1.2 Oracle

Cuando intentamos eliminar la tabla *departamentos* en Oracle ocurre lo siguiente:

```
SQL> drop table departamentos;  
drop table departamentos  
      *  
ERROR at line 1:  
ORA-02449: unique/primary keys in table referenced by foreign keys
```

Como puedes ver también ocurre un error, indicando que la clave primaria actúa como FOREIGN KEY en otras tablas y por eso no puede ser eliminada. Igual que MySQL está asegurando la integridad de la información almacenada.

Sin embargo, recuerda que Oracle tenía la posibilidad de añadir la cláusula CASCADE CONSTRAINT. Si la utilizamos el resultado será muy diferente. Podremos borrar la tabla *departamentos*, pero además se eliminarán todas las CONSTRAINTS que referenciaban a los campos de dicha tabla, es decir se eliminarán las FOREIGN KEY (la referencia, no el campo) del campo *dpto* en la tabla *empleados* y en la tabla *proyectos*.

```
SQL> select table_name from user_tables;  
TABLE_NAME  
-----  
DEPARTAMENTOS  
EMPLEADOS  
PROYECTOS  
CURRENTE  
  
SQL> drop table departamentos cascade constraint;  
Table dropped.  
  
SQL> select table_name from user_tables;  
TABLE_NAME  
-----  
EMPLEADOS  
PROYECTOS  
CURRENTE
```

(Notad que vosotros no tendréis la tabla CURRENTE, ya que proviene de un ejercicio que no hemos hecho).

7. SINÓNIMOS

Cuando tenemos acceso a las tablas de otro usuario y deseamos consultarlas es necesario teclear el nombre del usuario propietario antes del nombre de la tabla. Es decir, si nosotros tenemos acceso a la tabla *Departamento* del usuario *Pedro* y queremos consultar sus datos, entonces tendremos que teclear la siguiente orden:

```
SELECT * FROM Pedro.Departamento;
```

Sin embargo, mediante el uso de sinónimos, podemos crear un alias para referirnos a dicha tabla sin necesidad de incluir su nombre:

```
SELECT * FROM Tabla_Pedro;
```



En definitiva, un **sinónimo** es un nuevo nombre que se puede dar a una tabla, vista, secuencia, procedimiento, función, paquete, tipo, etc. definido por nosotros.

Con los sinónimos se pueden utilizar dos nombres diferentes para referirse a un mismo objeto. Aunque los podemos utilizar sobre nuestros propios objetos, resultan muy prácticos a la hora de referirnos a objetos de otros usuarios para evitar escribir el nombre de usuario delante de la tabla.

7.1 Creación de sinónimos

La sintaxis para crear sinónimos es la siguiente:

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM sinónimo FOR [usuario.]nom_objeto;
```

Si justo después de **CREATE** añadimos **OR REPLACE**, ocurre lo siguiente: si no había ningún sinónimo creado con dicho nombre entonces se crea, si ya existía, entonces se sustituye el anterior sinónimo por este.

Si especificamos que un sinónimo es **PUBLIC**, entonces dicho sinónimo podrá ser accedido por cualquier usuario. No obstante, cada usuario que quiera utilizar el sinónimo deberá tener los privilegios necesarios para acceder al objeto referido por el sinónimo. Es decir, si un usuario no tiene permisos para acceder a una tabla, aunque haya un sinónimo público para acceder a ella, el usuario seguirá sin poder acceder a dicho objeto.

Hay una serie de consideraciones importantes que hay que tener en cuenta cuando se desea trabajar con sinónimos:

- Para crear sinónimos privados sobre nuestro propio esquema, es necesario disponer del **privilegio CREATE SYNONYM**, el cual no se otorga por defecto.
- Para crear sinónimos públicos es necesario disponer del **privilegio CREATE PUBLIC SYNONYM**.

Ejemplo

Si tenemos una tabla llamada *Departamentos* pero queremos referirnos a ella a través del texto *Depart*, podemos crear un alias:

```
CREATE SYNONYM Depart FOR Departamento;
```

De esta manera podemos consultar el contenido de la tabla de los departamentos mediante su nombre o su sinónimo:

```
SELECT * FROM Departamento;
```

```
SELECT * FROM Depart;
```

Si nuestro usuario no dispone de los privilegios para crear sinónimos, entonces deberíamos acceder al sistema mediante el usuario SYS o SYSTEM y ejecutar la siguiente orden que le otorgaría a usuario los privilegios necesarios:

```
GRANT CREATE SYNONYM TO usuario;
```

7.2 Borrado de sinónimos

De manera similar a como se borran las tablas, se borran los sinónimos. Su sintaxis es:

```
DROP [PUBLIC] SYNONYM [usuario.]sinónimo;
```

Al igual que durante la creación, hay que tener en cuenta lo siguiente a la hora de borrar sinónimos:

- No es necesario disponer de ningún privilegio especial para borrar nuestros propios sinónimos.
- Únicamente los usuarios de tipo DBA y aquellos con el privilegio DROP PUBLIC SYNONYM pueden borrar sinónimos públicos.
- Sólo los usuarios DBA y aquellos con el privilegio DROP ANY SYNONYM pueden borrar los sinónimos de otros usuarios.

```
DROP SYNONYM Depart;
```

8. CONSULTAR EL DICCIONARIO

8.1 Consultar tablas del usuario

Oracle utiliza diversas vistas para mostrar las tablas de la base de datos. En concreto **USER_TABLES**, que contiene una lista de las tablas del usuario actual (o del esquema actual). Así para sacar la lista de tablas del usuario actual, se haría:

```
SELECT * FROM USER_TABLES;
```

Esta vista obtiene numerosas columnas. En concreto la columna **TABLES_NAME** muestra el nombre de cada tabla.

Otra vista es **ALL_TABLES** que mostrará una lista de **todas las tablas de la base de datos** (no solo del usuario actual), aunque oculta las que el usuario no tiene derecho a ver.

Finalmente **DBA_TABLES** es una tabla que contiene absolutamente **todas las tablas del sistema**; esto es accesible sólo por el usuario administrador (DBA). En el caso de **ALL_TABLES** y de **DBA_TABLES**, la columna **OWNER** indica el nombre del propietario de la tabla.

Un truco, si el resultado de nuestra consulta devuelve muchas columnas y se nos muestra cada fila partida en dos o más, es ampliar el tamaño de las filas y/o página que viene por defecto en Oracle.

Si ejecutamos las órdenes *SHOW LINESIZE* y *SHOW PAGESIZE* nos mostrará el tamaño que tiene por defecto configurado Oracle para mostrar la información. *LINESIZE* es el tamaño de cada fila y *PAGESIZE* el tamaño de la página. Por defecto *LINESIZE* tiene un valor de 80 y *PAGESIZE* de 14.

Para cambiar el tamaño de las líneas podemos ejecutar la orden *SET LINESIZE X* (donde X es el valor que queramos establecer). Si lo que queremos es que nos muestre más registros de los que nos saca inicialmente, podemos aumentar el tamaño de la página con la orden *SET PAGESIZE X* (donde X es el valor que queramos establecer).

Este sería el resultado de una consulta con el tamaño de línea por defecto:

```
SQL> select * from empleados where dpto = 'INF' or dpto = 'CONT';
```

DNI	NOMBRE	ESPECIALIDAD	FECHAALT
12345678A	Alberto Gil	Contable	10/12/10
23456789B	Mariano Sanz	Informática	04/10/11
45678901D	Ana Silván	Informática	25/11/12
78901234G	Rafael Colmenar	Informática	10/06/13

Como se ve, los campos y por ende los registros, se muestran partidos en dos líneas. Tras establecer *LINESIZE* a 200 se mostrarían de esta manera:

```
SQL> SET LINESIZE 200;
SQL> select * from empleados where dpto = 'INF' or dpto = 'CONT';
```

DNI	NOMBRE	ESPECIALIDAD	FECHAALT	DPTO	CODP
12345678A	Alberto Gil	Contable	10/12/10	CONT	MAD20
23456789B	Mariano Sanz	Informática	04/10/11	INF	T0451
45678901D	Ana Silván	Informática	25/11/12	INF	MAD20
78901234G	Rafael Colmenar	Informática	10/06/13	INF	T0451

```
SQL>
```

8.2 Obtener la lista de las columnas de las tablas

Otra posibilidad para poder consultar los datos referentes a las columnas de una tabla, es utilizar el diccionario de datos.

Oracle posee una vista llamada **USER_TAB_COLUMNS** que permite consultar todas las columnas de las tablas del esquema actual. Las vistas **ALL_TAB_COLUMNS** y **DBA_TAB_COLUMNS** muestran los datos del resto de tablas (la primera sólo de las tablas accesibles por el usuario).

En el caso de SQL estándar las columnas son accesibles mediante la vista **INFORMATION_SCHEMA.COLUMNS**. **USER_CONSTRAINTS**: que almacena todas las restricciones de integridad definidas por un usuario concreto.

Existen una serie de vistas creadas por Oracle que contienen información referente a las restricciones definidas en las tablas. Contienen información general las siguientes:

- **USER_CONSTRAINTS**: muestra las definiciones de restricciones de tablas propiedad del usuario.
- **ALL_CONSTRAINTS**: muestra las definiciones de restricciones de las tablas a las que puede acceder el usuario.
- **DBA_CONSTRAINTS**: muestra todas las definiciones de restricciones sobre todas las tablas. La columna **CONSTRAINT_TYPE** de las vistas anteriores puede ser:
 - C: Restricción CHECK
 - P: Restricción PRIMARY KEY
 - R: Restricción FOREIGN KEY
 - U: Restricción UNIQUE

Si queremos información sobre restricciones definidas en las columnas tenemos:

- **USER_CONS_COLUMNS**: describe todas las restricciones de columnas en tablas propiedad del usuario.
- **ALL_CONS_COLUMNS**: describe todas las restricciones de columnas en tablas accesibles por el usuario.
- **DBA_CONS_COLUMNS**: describe todas las columnas en la base de datos que tienen una restricción especificada.

Aquí os dejo una tabla con un resumen de los comandos/vistas más frecuentes:

RESUMEN DICCIONARIO DATOS ORACLE	
SELECT OWNER, TABLE-NAME FROM DBA_TABLES WHERE OWNER = 'usuario';	Para ver las tablas de los usuarios
DESC USER_TABLES	Para ver lo que se guarda en las tablas
SELECT TABLE_NAME FROM USER_TABLES;	Para saber las tablas que tiene el usuario
USER_CONS_COLUMNS DBA_CONS_COLUMNS ALL_CONS_COLUMNS	Vista para ver las restricciones que afectan a las columnas
USER_TABLES	Vista que muestra las tablas propias del usuario activo
ALL_TABLES	Vista que muestra todas las tablas propias del usuario activo
DBA_TABLES	Vista que muestra todas las tablas de la BD's
USER_CONSTRAINTS	Vista que muestra las restricciones del usuario activo
DBA_CONSTRAINTS	Vista que muestra las restricciones de la BD's
ALL_CONSTRAINTS	Vista que muestra todas las restricciones
SELECT * FROM USER_TABLESPACES;	Para ver todos los tablespaces
SELECT CATALOG_ROLE	Para ver entero el diccionario de datos
SESSIONS PRIVS	Información de los privilegios del usuario activo
USER_SYS_PRIVS	Información de los privilegios de sistema del usuario activo
DBA_SYS_PRIVS	Información de los roles y privilegios del sistema del usuario activo
USER_TAB_PRIVS	Información sobre los privilegios de objeto relacionados con el usuario, tanto otorgados como concedidos
USER_TAB_PRIVS MADE	Privilegios concedidos
USER_TAB_PRIVS RECD	Privilegios recibidos
SESSION ROLES	Roles del usuario activo
ROLE_SYS_PRIVS	Privilegios de sistema asignados a los roles
ROLE_TAB_PRIVS	Privilegios sobre objetos asignados a los roles
DBA_ROLES	Todos los roles del sistema
DBA_PROFILES	Todos los perfiles de la BD's
DBA_DATA_FILES	Archivos que componen mi espacio de tabla
USER_FREE_SPACE	Tamaño libre de mi espacio de tabla
DBA_FREE_SPACE	Tamaño libre en todos los tablespaces
DBA_TABLESPACES	Tablespaces del sistema
DBA_TS_QUOTAS	Uso de los tablespaces por los usuarios

8.3 Orden DESCRIBE

El comando **DESCRIBE**, permite obtener la estructura de una tabla.

Ejemplo:

```
DESCRIBE existencias;
```

Y aparecerán los campos de la tabla *existencias*. 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:

Nombre	¿Nulo?	Tipo
N_ALMACEN	NOT NULL	NUMBER(2)
TIPO	NOT NULL	VARCHAR2(2)
MODELO	NOT NULL	NUMBER(2)
CANTIDAD	NOT NULL	NUMBER (7)

8.4 Consulta de sinónimos

Podemos consultar los sinónimos que hemos creado a través de la vista **USER_SYNONYMS**. Basta con ejecutar la siguiente instrucción:

```
SELECT * FROM USER_SYNONYMS;
```