edix
Digital Workers

UF3-CONSULTAS BÁSICAS SQL

UF-3 FORMULACIÓN DE CONSULTAS BÁSICAS

- SQL aparece como el **lenguaje más usado en los entornos de SGBD** para la comunicación con bases de datos relacionales. Está difundido **en casi todos los sistemas** y se conecta con muchos de los **lenguajes de programación**. Por lo tanto estamos ante un **estándar de facto** para el trabajo con **bases de datos** en los sistemas informáticos.

- **Objetivos:**

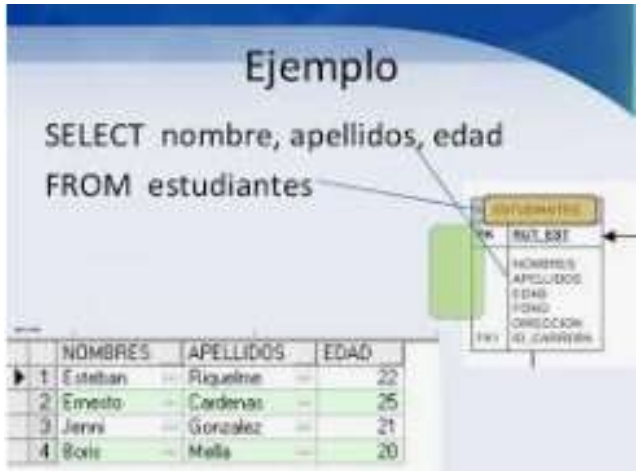
  - Conocer las **características** fundamentales y la **estructura** de SQL.
  - Manejar la **creación de bases de datos** y cuáles son los **tipos de datos** que se pueden almacenar en ellas.
  - Como crean las **claves** y se manejan las **primarias** y las **foráneas** o **externas**.
  - Como se realiza la **gestión de registros en tablas**.
  - Conocer los principios de **realización de consultas** de datos en SQL.
  - Manejar la **selección de columnas**.
  - Trabajar con la **selección de filas y su ordenación**

UF-3 FORMULACIÓN DE CONSULTAS BÁSICAS

- En esta unidad vamos a estudiar el **lenguaje SQL**. Este lenguaje es fundamental para poder realizar las consultas sobre la base de datos. Dominar este lenguaje es muy importante para acceder a la información que nos interesa de una manera rápida y eficaz. Para ello es muy importante haber creado un buen diseño de la base de datos para que dichas consultas no se compliquen.

- Dentro de todas las sentencias de las que disponemos en SQL, contamos con la sentencia SELECT que nos permite interrogar los datos que hemos introducido en una base de datos. Este comando es tan versátil, que nos permitirá elegir qué tablas consultar, qué campos o en qué orden deben de salir los datos que deben cumplir una serie de filtros que le hayamos indicado.

- Dominar esta sentencia es **dominar la base de datos** y disponer de la información necesaria en unas pocas sentencias. A lo largo de la unidad se van a repasar las diferentes sentencias SQL, haciendo hincapié en la sentencia SELECT.



UF-3 FORMULACIÓN DE CONSULTAS BÁSICAS

- **SQL** (Structured Query Language) es un **lenguaje de consulta**, estructurado para la **definición, manipulación y control** de bases de datos relacionales.

- Fue **estandarizado** por el **ANSI** (Instituto Americano de Normalización) y el **ISO** (Organismo Internacional de Normalización) desde 1986, por lo que es el lenguaje "de facto" utilizado en los actuales sistemas de gestión de bases de datos.

- SQL es un **lenguaje declarativo** que está basado en el **álgebra relacional** y se orienta al **manejo de conjuntos**. Permite especificar **diversos tipos de operaciones** en las bases de datos que lo implementan.

- Una de sus características es el **manejo del álgebra y el cálculo relacional** que permiten efectuar **consultas** con el fin de recuperar de forma sencilla **información de interés** de bases de datos, así como **hacer cambios** en ellas.

- Esto permite que el lenguaje sea capaz de **acceder al conjunto de datos** requerido usando solo una o varias operaciones compuestas.

- Estas operaciones **permiten definir qué datos** se quieren encontrar y **no cómo encontrarlos**, ya que la implementación se encarga de hacerlo.

❑ Evolución del lenguaje SQL:

- **Años 70:** El laboratorio de investigación de IBM trabaja en el proyecto **System R** cuyo objetivo era la **implementación de un SGDB relacional**. Paralelamente y como apoyo a este proyecto se desarrolla un **lenguaje de acceso a datos:** el **SEQUEL** (Structured English Query Language), que fue el predecesor de **SQL**.
- **Años 80:** Una vez finalizado el proyecto System R, el lenguaje SQL **comienza a utilizarse** tanto en IBM como en otras empresas alcanzando una gran popularidad. En 1986 el ANSI estandarizo el lenguaje, denominándolo SQL86.
- **1989:** (SQL89 o SQL1): revisión.
- **1992:** (SQL 92 o SQL2): revisado y ampliado.
- **1999** (SQL2000): añadido características orientadas a objetos, disparadores, consultas recursivas.
- **2003:** (SQL2003): añadido soporte XML.
- **2008:**(SQL2008): ampliado con diferentes sentencias.

- **Características de SQL**
  - **Es un lenguaje de "alto nivel":** Permite la **abstracción** con respecto a los archivos de las bases de datos con las que opera.
  - **Es un lenguaje declarativo:** No es necesario un conjunto de instrucciones y estados (programación imperativa) para solucionar el problema, **solo se describe el problema** pero no las instrucciones para solucionarlo. Esto se realiza por mecanismo internos.
  - **Orientado a manejar conjuntos de registros:** Lo que explota la **flexibilidad** y potencia de las bases de datos.
  - **Esquemas relacionales:** Incluye sentencias para **definición de esquemas** relacionales.
  - **Consultas:** Incluye sentencias para **consultas basadas en algebra relacional**.
  - **Restricciones:** Incluye sentencias para **restricciones de integridad** que deben cumplir los datos.
  - **Transacciones:** Contiene sentencias para el **control de transacciones**.
  - **Integración:** Permite la **interacción con otros lenguajes**: c, c++, java, php, etc..
  - **Seguridad:** Contiene **sentencias para especificar derechos de acceso**.
- **Tipos de órdenes SQL:** Un lenguaje para manejar bases de datos se compone de un **conjunto o repertorio de instrucciones**, al igual que en un lenguaje de programación habitual. En particular, en los lenguajes sobre bases de datos relacionales se distinguen estos sublenguajes principales:
  - Lenguaje de Definición de Datos (**DDL**)
  - Lenguaje de Manipulación de Datos (**DML**)
  - Lenguaje de Control de Datos (**DCL**)
  - Control de Transacciones (**TCL**)

- **Características de SQL**
  - **Es un lenguaje de "alto nivel":** Permite la **abstracción** con respecto a los archivos de las bases de datos con las que opera.
  - **Es un lenguaje declarativo:** No es necesario un conjunto de instrucciones y estados (programación imperativa) para solucionar el problema, **solo se describe el problema** pero no las instrucciones para solucionarlo. Esto se realiza por mecanismo internos.
  - **Orientado a manejar conjuntos de registros:** Lo que explota la **flexibilidad** y potencia de las bases de datos.
  - **Esquemas relacionales:** Incluye sentencias para **definición de esquemas** relacionales.
  - **Consultas:** Incluye sentencias para **consultas basadas en algebra relacional**.
  - **Restricciones:** Incluye sentencias para **restricciones de integridad** que deben cumplir los datos.
  - **Transacciones:** Contiene sentencias para el **control de transacciones**.
  - **Integración:** Permite la **interacción con otros lenguajes**: c, c++, java, php, etc..
  - **Seguridad:** Contiene **sentencias para especificar derechos de acceso**.
- **Tipos de órdenes SQL:** Un lenguaje para manejar bases de datos se compone de un **conjunto o repertorio de instrucciones**, al igual que en un lenguaje de programación habitual. En particular, en los lenguajes sobre bases de datos relacionales se distinguen estos sublenguajes principales:
  - Lenguaje de Definición de Datos (**DDL**)
  - Lenguaje de Manipulación de Datos (**DML**)
  - Lenguaje de Control de Datos (**DCL**)
  - Control de Transacciones (**TCL**)

❑ Tipos de órdenes SQL:

- **Lenguaje de Definición de Datos (DDL):** El **DDL** 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. Por supuesto es el encargado de la creación de las tablas.

- **Lenguaje de Manipulación de Datos (DML):** Es el lenguaje de manipulación de datos, que incluye las instrucciones para consultar la base de datos, así como para insertar o eliminar tuplas y **modificar** valores de datos. Este lenguaje es el utilizado para la fase de explotación o de trabajo útil de la base de datos, y es empleado por los programadores y usuarios finales.

- **Lenguaje de Control de Datos (DCL):** Es un Lenguaje de Control de Datos que incluye una serie de comandos SQL que ayudan al administrador a **controlar** el acceso a los datos contenidos en la Base de Datos.

- **Control de Transacciones (TCL):** Es un lenguaje de programación para el control de transacciones en una base de datos.

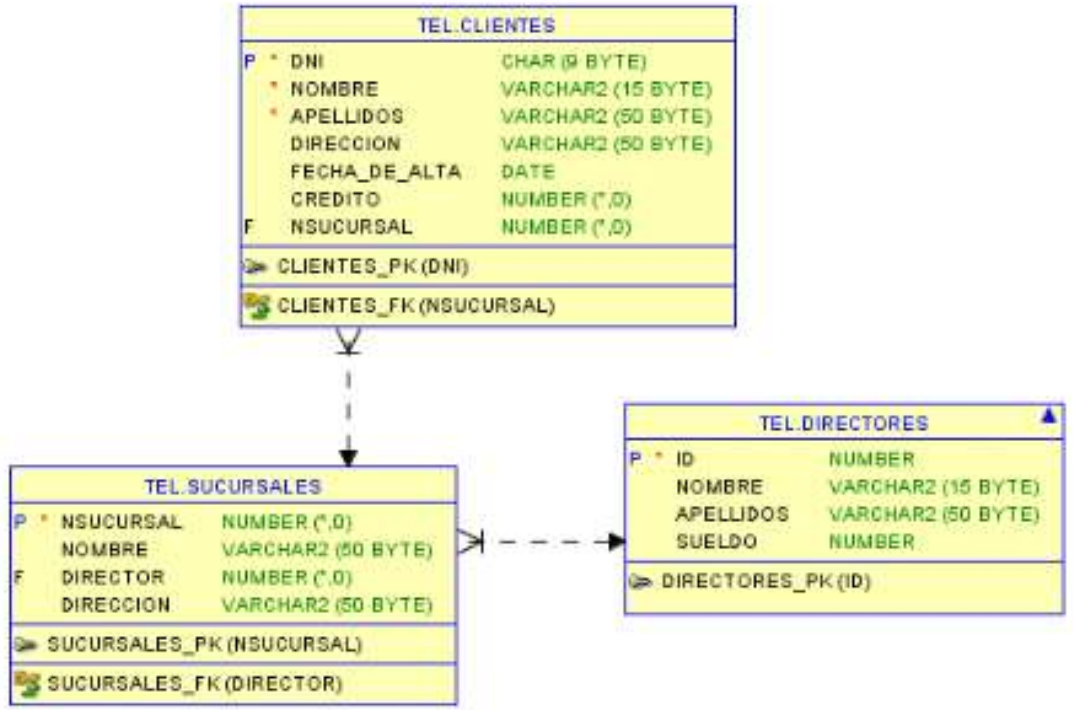
- **Entorno de trabajo:** Para el uso del lenguaje SQL, al igual que para otros lenguajes es necesario disponer de un entorno de trabajo. Un usuario de una base de datos bajo SQL debe disponer de:
  - **Conexión con la base de datos.**
  - **Acceso a la base de datos.** Debes de estar como usuario autorizado en la base de datos.
  - **Entornos.** Se tratan de programas diseñados para realizar operaciones sobre la base de datos. La utilidad más básica es un terminal de texto en el que introducir órdenes SQL, pero existen entornos como:
    - **MySQL Workbench** para MySql: http://itt.telefonicaed.com/phpmyadmin/; va en versión libre en la 8.0.14.
    - **SQL Developer** para Oracle: https://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html ; va en versión 18.4

- **Programadores.** Para los usuarios que quieran acceder a traves de lenguajes de programación a la base de datos deben de disponer de los entornos de desarrollo adecuados y conocer la api de cada lenguaje para poder acceder a ella.

- **Útiles de administración del sistema.** El DBA (Administrador de Bases de datos) utilizará para su trabajo algunas herramientas que están vedadas al resto de usuarios, como programas que evalúen el rendimiento del sistema o aplicaciones para la gestión de usuarios.

Bases de datos de ejemplo: Entidades bancarias y clientes

- A lo largo de la unidad se va a trabajar con numerosos ejemplos que en algunos casos va a hacer referencia a un modelo de datos ya definido.

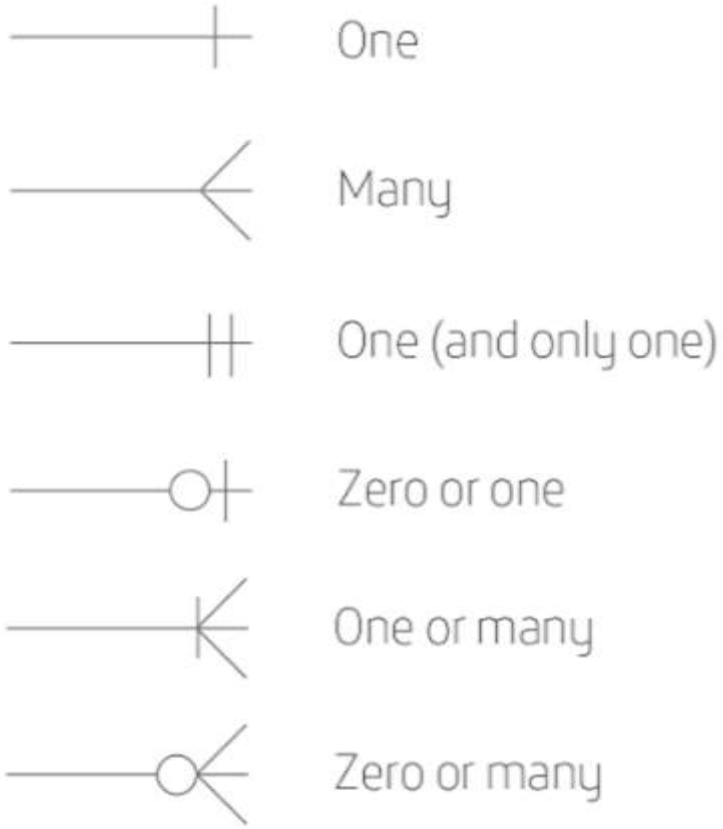


Bases de datos de ejemplo: Entidades bancarias y clientes

- Se trata de un modelo de datos de una entidad bancaria muy simple donde se va a almacenar información de clientes, sucursales y directores. Un cliente pertenece a una sucursal y cada sucursal tiene asociado un director.
  - **Clientes:** Clientes de las sucursales bancarias.
  - **Sucursales:** Sucursales bancarias
  - **Directores:** Directores de Sucursales

- La mayoría de entornos para trabajar con Base de datos disponen de utilidades para generar los diagramas.
- Cada uno de ellos representan las características de las tablas y sus relaciones con notaciones parecidas, pero es necesario consultar la documentación para conocer en detalle el significado de cada uno de los elementos que representan.
- Por ejemplo, para la tabla clientes tenemos:
  - P (CLIENTES_PK): Primary KEY
  - F: (CLIENTES_FK (NSUCURSAL): Foreign KEY

Bases de datos de ejemplo: Entidades bancarias y clientes



UF-3 FORMULACIÓN DE CONSULTAS BÁSICAS

Bases de datos de ejemplo: Entidades bancarias y clientes

SUCURSALES

NSUCURSAL	NOMBRE	DIRECTOR	DIRECCION
1001	Sucursal Centro	12	Avd. del Estilo, 45
1002	Sucursal Oeste	15	Avd. Mediterráneo, 14
1003	Sucursal Este	20	Calle Martínez, 45
1004	Sucursal Norte	(null)	Calle Especias, 23

CLIENTES

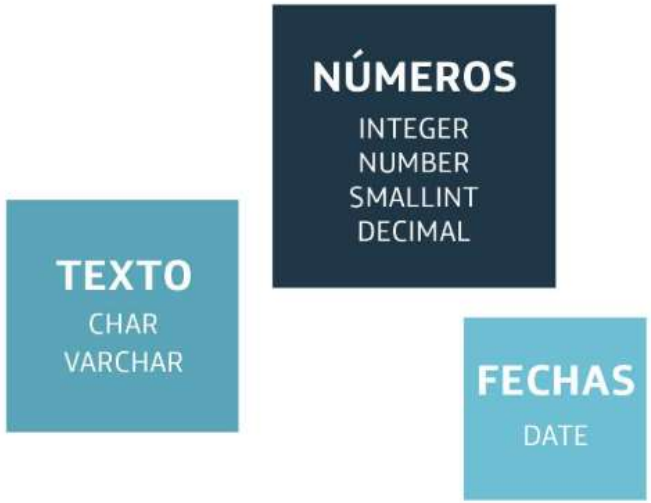
DNI	NOMBRE	APELLIDOS	DIRECCION	FECHA_DE_ALTA	CREDITO	NSUCURSAL
30515454K	Ana	Martín Martín	Calle del Socorro, 1	08/01/18	1500	1001
33358796A	Marta	López Ruiz	Calle Martinez, 76	14/09/17	600	1001
78458784B	Antonio	Castillo Mentas	Calle Soles, 14	(null)	500	1002

DIRECTORES

ID	NOMBRE	APELLIDOS	SUELDO
12	Alberto	Pérez Martín	1800
15	Antonio	López López	1500
20	Silvia	Martín Martín	1300
13	Pedro	García Martín	1900

Bases de datos de ejemplo: Entidades bancarias y clientes

- **Creación de la base de datos:** Lo primero a realizar es crear la base de datos en el SGBD. Para ello disponemos de la sentencia CREATE DATABASE cuya sintaxis es la siguiente:
  - CREATE DATABASE database_name; Si llamamos a la base de datos tel, tendríamos que ejecutar en el SGBD la siguiente sentencia: CREATE DATABASE tel;
- **Tipos de datos:** Antes de pasar a la creación de tablas es necesario conocer los tipos de datos con los que se pueden trabajar en un SGBD. Los **tipos de datos** más utilizados se definen a continuación.



Tipos de datos Oracle =>Bases de datos de ejemplo: Entidades bancarias y clientes

BLOB	Permite almacenar datos binarios no estructurados	Admiten hasta 8 terabytes
CLOB	Almacena datos de tipo carácter	Admiten hasta 8 terabytes
NCLOB	Almacena datos de tipo carácter	Admiten hasta 8 terabytes. Guarda los datos según el juego de caracteres Unicode nacional.
BFILE	Almacena datos binarios no estructurados en archivos del sistema operativo, fuera de la base de datos. Una columna BFILE almacena un localizador del archivo a uno externo que contiene los datos	Admiten hasta 8 terabytes.

Tipos de datos Oracle =>Bases de datos de ejemplo: Entidades bancarias y clientes

NUMBER	Almacena números fijos y en punto flotante	Se admiten hasta 38 dígitos de precisión. Para declarar un tipo de datos NUMBER en un CREATE es suficiente con: nombre_columna NUMBER. Opcionalmente se le puede indicar la **precisión** (número total de dígitos) y la **escala** (número de dígitos a la derecha de la coma, decimales, los cogerá de la precisión indicada): **nombre_columna NUMBER (precision, escala)** Si no se indica la precisión se tomará en función del número a guardar, si no se indica la escala se tomará escala cero. Para no indicar la precisión y sí la escala podemos utilizar: **nombre_columna NUMBER (*, escala)** Para introducir números que no estén en el formato estándar de Oracle se puede utilizar la función TO_NUMBER.
FLOAT	Almacena tipos de datos numéricos en punto flotante	Es un tipo NUMBER que sólo almacena números en punto flotante

Tipos de datos Oracle =>Bases de datos de ejemplo: Entidades bancarias y clientes

DATE	Almacena un punto en el tiempo (fecha y hora)	El tipo de datos DATE almacena el año (incluyendo el siglo), el mes, el día, las horas, los minutos y los segundos (después de medianoche). Oracle utiliza su propio formato interno para almacenar fechas.
TIMESTAMP	Almacena datos de tipo hora, fraccionando los segundos	
TIMESTAMP WITH TIME ZONE	Almacena datos de tipo hora incluyendo la zona horaria (explícita), fraccionando los segundos	
TIMESTAMP WITH LOCAL TIME ZONE	Almacena datos de tipo hora incluyendo la zona horaria local (relativa), fraccionando los segundos	Cuando se usa un SELECT para mostrar los datos de este tipo, el valor de la hora será ajustado a la zona horaria de la sesión actual
XMLType	Tipo de datos abstracto. En realidad, se trata de un CLOB.	Se asocia a un esquema XML para la definición de su estructura.

Tipos de datos MYSQL =>Bases de datos de ejemplo: Entidades bancarias y clientes

Tipos de datos numéricos

Tipo de dato	Descripción	Rango	Tamaño
BIT [(m)]	Numero entero que indica bits por valor (Valor por defecto - 1)	1 64	m bit
TINYINT [(m)]	Entero de tamaño pequeño con o sin signo.	Rango con signo (-128 127) Rango sin signo (0 255)	1 Byte
BOOL | BOOLEAN	Dato binario 0 Falso | 1 Verdadero	0 1	1 bit
SMALLINT [(m)]	Entero de tamaño pequeño con o sin signo	Rango con signo (-32768 32767) Rango sin signo (0 65535)	2 Bytes
MEDIUMINT [(m)]	Entero de tamaño medio con o sin signo.	Rango con signo (-8388608 8388607) Rango sin signo (0 16777215)	3 Bytes
INTEGER [(m)] INT [(m)]	Entero estándar con o sin signo.	Rango con signo (-2147483648 2147483647) Rango sin signo (0 4294967295)	4 Bytes
BIGINT [(m)]	Entero grande con o sin signo.	Rango con signo (-9223372036854775.808 9223372036854775.807) Rango sin signo (0 18446744073709551615)	8 Bytes
FLOAT [(m)]	Número pequeño en coma flotante de precisión simple o doble.	Rango en precisión simple (0 24) Rango en precisión doble (25 53)	4 u 8 Bytes
FLOAT[(m,d)]	Número pequeño en coma flotante de precisión simple. m – anchura de muestra d – dígitos significativos	Rango -3.402823466E+38 -1.175494351E-38, 0 1.175494351E-38 3.402823466E+38	4 Bytes

Tipos de datos MYSQL =>Bases de datos de ejemplo: Entidades bancarias y clientes

Tipo de dato	Rango soportado	Formato de almacenamiento	Tamaño
DATE	'1000-01-01' a '9999-12-31'	'YYYY-MM-DD'	3 Bytes
DATETIME	'1000-01-01 00:00:00' a '9999-12-31 23:59:59'.	'YYYY-MM-DD HH:MM:SS'	8 Bytes
TIMESTAMP[(m)]	Una marca temporal. El rango es de '1970-01-01 00:00:00' hasta el año 2037. Si damos como valor NULL, tomará la fecha y la hora actual.	'YYYY-MMDD HH:MM:SS'	4 Bytes
TIME	'-838:59:59' a '838:59:59'	'HH:MM:SS'	3 Bytes
YEAR[(2|4)]	1901 a 2155, y 0000. Año formado por dos o cuatro dígitos	'YYYY'	1 Byte

Tipos de datos MYSQL =>Bases de datos de ejemplo: Entidades bancarias y clientes

Tipo de dato	Rango soportado	Formato de almacenamiento	Tamaño
DATE	'1000-01-01' a '9999-12-31'	'YYYY-MM-DD'	3 Bytes
DATETIME	'1000-01-01 00:00:00' a '9999-12-31 23:59:59'.	'YYYY-MM-DD HH:MM:SS'	8 Bytes
TIMESTAMP[(m)]	Una marca temporal. El rango es de '1970-01-01 00:00:00' hasta el año 2037. Si damos como valor NULL, tomará la fecha y la hora actual.	'YYYY-MMDD HH:MM:SS'	4 Bytes
TIME	'-838:59:59' a '838:59:59'	'HH:MM:SS'	3 Bytes
YEAR[(2|4)]	1901 a 2155, y 0000. Año formado por dos o cuatro dígitos	'YYYY'	1 Byte

Tipos de datos MYSQL =>Bases de datos de ejemplo: Entidades bancarias y clientes

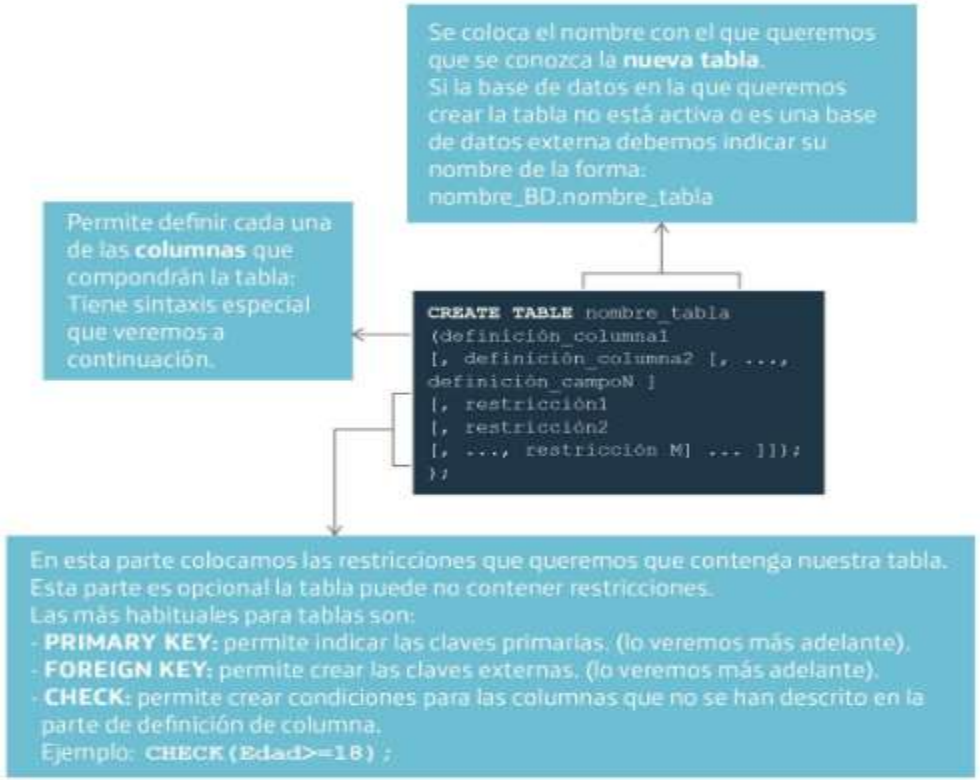
Tipo de dato	Descripción	Tamaño de almacenamiento
CHAR(m)	Una cadena de caracteres de longitud fija que siempre tiene el número necesario de espacios a la derecha para ajustarla a la longitud especificada al almacenarla (m).	m Bytes
VARCHAR(m)	Cadena de caracteres de longitud variable. m representa la longitud de columna máxima.	m +1 Bytes
BINARY(m)	El tipo BINARY es similar al tipo CHAR, pero almacena cadenas de datos binarios	m Bytes
VARBINARY(m)	El tipo VARBINARY es similar al tipo VARCHAR, pero almacena cadenas de caracteres binarias	m+1 Bytes
BLOB [(m)]	Una columna BLOB con longitud máxima de 65,535 (2^16 - 1) bytes.	Longitud +2 Bytes
TEXT[(m)]	Una columna TEXT con longitud máxima de 65,535 (2^16 - 1) caracteres.	Longitud +2 Bytes
TINYBLOB	Una columna BLOB con una longitud máxima de 255 (2^8 - 1) bytes.	Longitud +1 Bytes
TINYTEXT	Una columna TEXT con una longitud máxima de 255 (2^8 - 1) bytes.	Longitud +1 Bytes
MEDIUMBLOB	Una columna BLOB con longitud de 16,777,215 (2^24 - 1) bytes.	Longitud +3 Bytes
MEDIUMTEXT	Una columna TEXT con longitud de 16,777,215 (2^24 - 1) bytes.	Longitud +3 Bytes
LONGBLOB	Una columna BLOB con longitud máxima de 4,294,967,295 o 4GB	Longitud +4 Bytes

Tipos de datos MYSQL =>Bases de datos de ejemplo: Entidades bancarias y clientes

LONGTEXT	Una columna TEXT con longitud máxima de 4,294,967,295 o 4GB (2^32 - 1) bytes.	Longitud +4 Bytes
ENUM('value1','value2',...)	Una enumeración. Un objeto de cadena de caracteres que sólo puede tener un valor, elegido entre 65535 valores distintos.	1 ó dos bytes dependiendo del número de valores
SET('value1','value2',...)	Una enumeración. Un objeto de cadena de caracteres que sólo puede tener un valor, elegido entre 64 valores distintos.	1, 2, 3, 4 ó 8 bytes, dependiendo del número de valores

Bases de datos de ejemplo: Entidades bancarias y clientes

- **Crear una tabla:** Una vez creada la base de datos, el siguiente paso es la creación de las tablas. En este caso el estándar SQL si define claramente una sentencia para la creación de tablas. Hay que tener en cuenta que las restricciones que podamos aplicar a una tabla las podemos hacer a nivel de columna o de tabla, tal y como se muestra a continuación. Para crear una tabla la sentencia en SQL es CREATE TABLE cuya síntesis es:



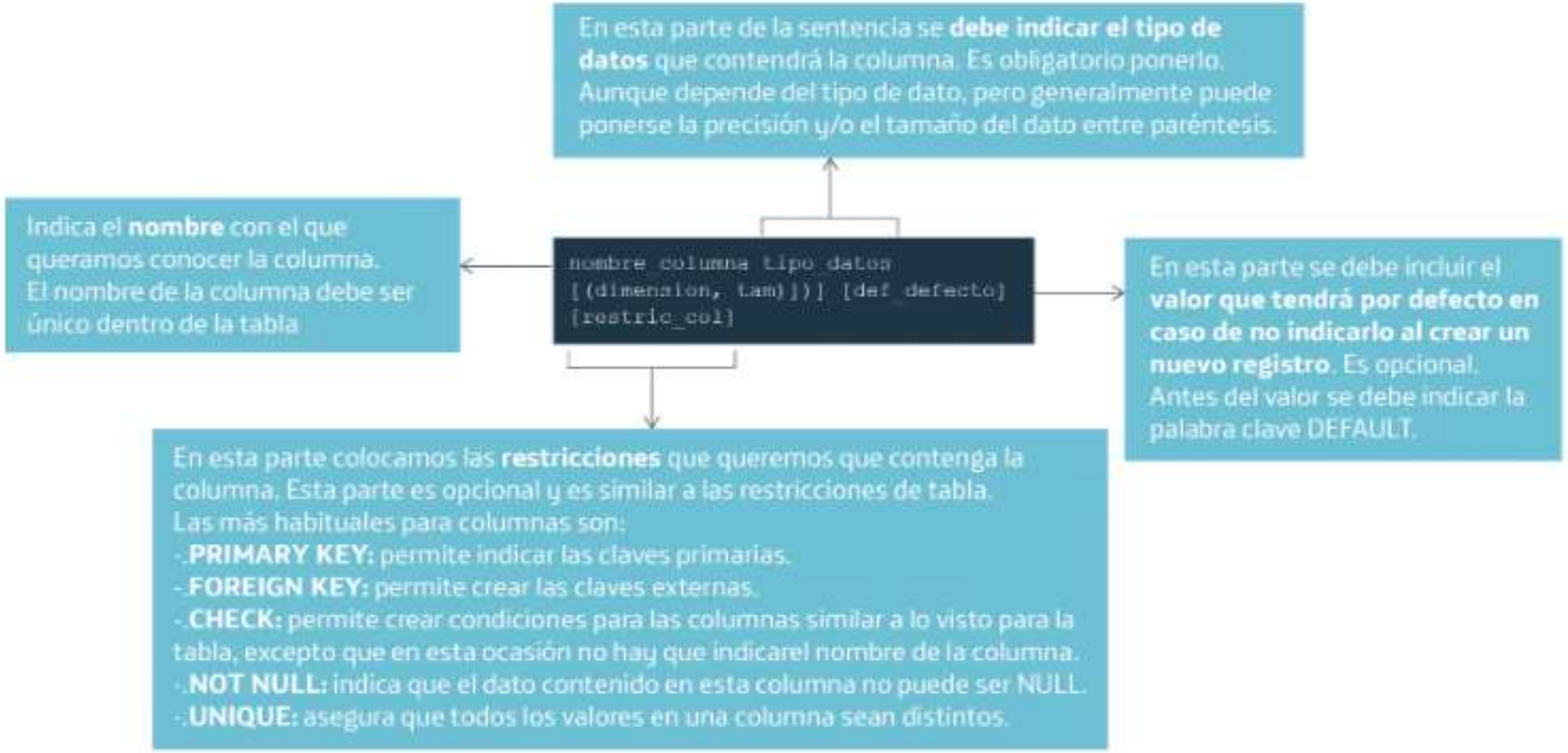
Bases de datos de ejemplo: Entidades bancarias y clientes

- En la siguiente URL tenéis la sintaxis completa de la sentencia:
  - https://dev.mysql.com/doc/refman/8.0/en/create-table.html

- Para definir una columna dentro de la sentencia CREATE TABLE utilizamos la siguiente sintaxis:

```
CREATE TABLE nombre_tabla
(definición_columna1
[, definición_columna2 [, ...,
definición_campoN] ... ] ]
[, restricción1
[, restricción2 [, ..., restricción M] ...]]);
);
```

Bases de datos de ejemplo: Entidades bancarias y clientes



Bases de datos de ejemplo: Entidades bancarias y clientes

```
CREATE TABLE Clientes
(
  Dni CHAR (9) NOT NULL UNIQUE,
  Nombre VARCHAR(15) NOT NULL,
  Apellidos VARCHAR(50) NOT NULL,
  Direccion VARCHAR(50) DEFAULT '',
  Fecha_de_alta DATE,
  Credito INTEGER CHECK (Credito>0),
  nSucursal INTEGER
);
```

CLIENTES

DNI	NOMBRE	APELLIDOS	DIRECCION	FECHA_DE_ALTA	CREDITO	NSUCURSAL
30515454K	Ana	Martín Martín	Calle del Socorro, 1	08/01/18	1500	1001
33358796A	Marta	López Ruiz	Calle Martinez, 76	14/09/17	600	1001
78458784B	Antonio	Castillo Mentas	Calle Soles, 14	(null)	500	1002

- Como se observa se ha definido el tipo de datos de cada columna y también se le han añadido algunas características más en la propia definición del campo. En concreto:
  - En el campo Dni no se permiten ni nulos ni valores repetidos.
  - En los campos Nombre y Apellidos no se permiten valores nulos.
  - En el campo Dirección el valor por defecto si no introducimos nada es la cadena vacía.
  - En el campo Crédito solo se permiten valores mayores que cero.

Bases de datos de ejemplo: Entidades bancarias y clientes

```
CREATE TABLE Sucursales
(
  nSucursal INTEGER,
  Nombre VARCHAR(50),
  Direccion VARCHAR(50),
  Director INTEGER,
  UNIQUE (nSucursal)
);
```

SUCURSALES

NSUCURSAL	NOMBRE	DIRECTOR	DIRECCION
1001	Sucursal Centro	12	Avd. del Estilo, 45
1002	Sucursal Oeste	15	Avd. Mediterráneo, 14
1003	Sucursal Este	20	Calle Martínez, 45
1004	Sucursal Norte	(null)	Calle Especias, 23

- En el caso de la tabla Sucursales las características adicionales añadidas a parte del tipo de campo son:
  - No permitir valores repetidos en el campo nSucursal
  - Como se puede observar en este caso las restricciones indicadas se han definido después de haber definido el tipo de columna. Esto está permitido.

Bases de datos de ejemplo: Entidades bancarias y clientes

- **La clave primaria es la columna o columnas** de la tabla que identifica a un registro inequívocamente. Por lo tanto no puede haber ningún valor nulo para estas columnas, ni puede haber dos registros con el mismo valor en esta columna.
- Para crear una clave primaria en una columna se utiliza la restricción PRIMARY KEY. Se puede indicar en las definiciones de columna o en las restricciones de la tabla:
- En definición de columna:

nombre_columna tipo_datos [(dimension, tam)])] [def_defecto] PRIMARY KEY
En restricción de tabla

CREATE TABLE nombre_tabla

(definición_columna1

[, definición_columna2 [, …, definición_campoN] …]]

[, PRIMARY KEY

(columna [, columna. . .])]);

UF-3 FORMULACIÓN DE CONSULTAS BÁSICAS

Bases de datos de ejemplo: Entidades bancarias y clientes

- Vamos a ir completando características de las tablas de ejemplo. Más adelante veremos que una vez creada una tabla se pueden alterar sus restricciones, sin necesidad de borrarla. Por ahora iremos creándola de nuevo. Por lo tanto se debe borrar la tabla anterior para probar el siguiente código. Para borrar una tabla se utiliza la sentencia: DROP TABLE nombre_tabla;
- En la tabla clientes es bastante fácil ver que la clave primaria debe ser el DNI ya que es el único campo que debe ser único y por el que localizamos a los clientes inequívocamente (salvo error administrativo).

```
CREATE TABLE Clientes
(
  Dni CHAR (9) PRIMARY KEY,
  Nombre VARCHAR(15) NOT NULL,
  Apellidos VARCHAR(50) NOT NULL,
  Direccion VARCHAR(50) DEFAULT '',
  Fecha_de_alta DATE,
  Credito INTEGER CHECK (Credito>0),
  nSucursal INTEGER
);
```

Bases de datos de ejemplo: Entidades bancarias y clientes

- En cuanto a la tabla sucursales hemos incluido un nº (nSucursal) que identifica a cada sucursal de manera única:

```
CREATE TABLE Sucursales
(
  nSucursal INTEGER,
  Nombre VARCHAR(50),
  Direccion VARCHAR(50),
  Director INTEGER,
  PRIMARY KEY (nsucursal)
);
```

UF-3 FORMULACIÓN DE CONSULTAS BÁSICAS

Bases de datos de ejemplo: Entidades bancarias y clientes

- **Creación de claves externas** La clave externa o foránea es la columna de la tabla que identifica a una clave primaria de otra tabla.
- Por lo tanto el valor que se encuentra en esta columna identifica inequívocamente a un registro de otra tabla. Esta tabla se dice que esta referenciada por esta clave externa. Es posible que en la tabla se repitan valores en esta columna.
- La creación de claves externas en SQL es similar a lo explicado para las claves primarias. Según el SGDB se crea en la definición de columna o en las restricciones de la tabla. La única diferencia es que hay que escribir la tabla y columna a la que se referencia con la clave externa.
- En definición de columna:

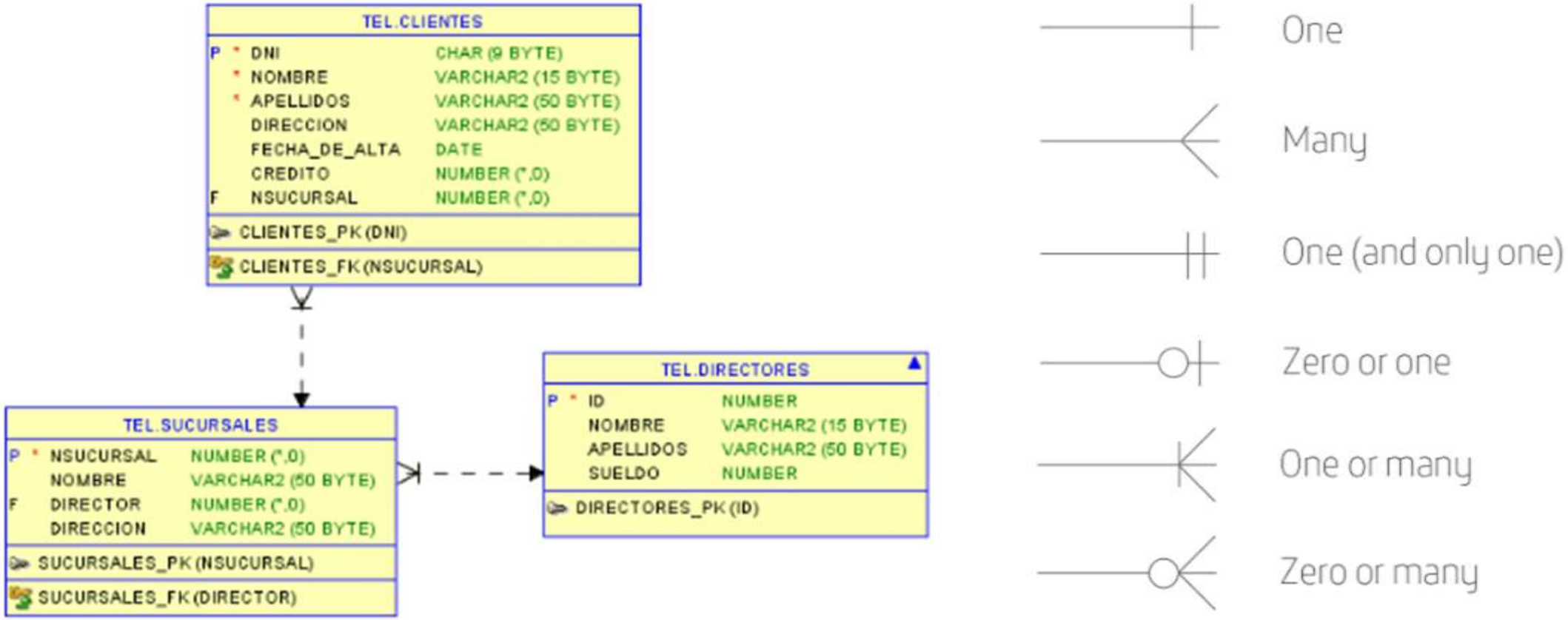
```
nombre_columna tipo_datos [(dimension, tam)])] [def_defecto]
REFERENCES nombre_tabla [(nombre_columna)]
```

- En restricción de tabla:

```
CREATE TABLE nombre_tabla
( definición_columna1
[, definición_columna2 [, …,  definición_campoN ] …]  ]
[, FOREIGN KEY
(columna [, columna. . .])
REFERENCES tabla
[(columna2 [, columna2. . .])]]
```

UF-3 FORMULACIÓN DE CONSULTAS BÁSICAS

Bases de datos de ejemplo: Entidades bancarias y clientes



Bases de datos de ejemplo: Entidades bancarias y clientes

- Con las claves externas podemos indicar las relaciones que tienen las tablas entre si. Un cliente pertenece a una sucursal por lo tanto la columna nSucursal de la tabla clientes es la clave externa que referencia a la tabla Sucursales:

```
CREATE TABLE Clientes
(
  Dni CHAR (9) PRIMARY KEY,
  Nombre VARCHAR(15) NOT NULL,
  Apellidos VARCHAR(50) NOT NULL,
  Direccion VARCHAR(50) DEFAULT '',
  Fecha_de_alta DATE,
  Credito INTEGER CHECK (Credito>0),
  nSucursal INTEGER REFERENCES Sucursales(nSucursal)
);
CREATE TABLE Sucursales
(
 nSucursal INTEGER,
 Nombre VARCHAR(50),
 Direccion VARCHAR(50),
 Director INTEGER,
 PRIMARY KEY (nsucursal),
 FOREIGN KEY (Director) REFERENCES Directores(id)
);
```

- En el primer ejemplo se incluye la clave secundaria en la definición de columna. Como se ve sólo es necesario ponerle la palabra clave REFERENCES y la tabla a la que referencia.

- En el segundo ejemplo tenemos una clave externa referenciando una tabla (Directores) donde tenemos todos los datos de los Directores. En este caso se ha realizado a través de restricción de tabla.

Bases de datos de ejemplo: Entidades bancarias y clientes

- **Integridad referencial:** La integridad referencial es una herramienta imprescindible de las bases de datos relacionales, para mantener la consistencia entre las tablas relacionadas.
- 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 permitírsenos borrar el registro ocurrirá fallo de integridad (habrá claves foráneas refiriéndose a una clave principal que ya no existe).
- Para mantener esa consistencia disponemos de las siguientes opciones a indicar en la cláusula REFERENCES. Son:
  - **ON DELETE SET NULL. Coloca nulos todas las claves foráneas relacionadas con la borrada.**
  - **ON DELETE CASCADE. Borra todos los registros cuya clave foránea es igual que la clave del registro borrado.**
  - **ON DELETE SET DEFAULT. Coloca en el registro relacionado el valor por defecto en la columna foránea**
  - **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 muchos gestores de datos se admite el uso tanto de ON DELETE como de ON UPDATE.

UF-3 FORMULACIÓN DE CONSULTAS BÁSICAS

Bases de datos de ejemplo: Entidades bancarias y clientes

- Ejemplo: En este caso le indicamos a la tabla Sucursales que si se borra de la tabla directores un director relacionado con una sucursal, poner en el campo Director el valor NULL.

```
CREATE TABLE Sucursales
(
  nSucursal INTEGER,
  Nombre VARCHAR(50),
  Direccion VARCHAR(50),
  Director INTEGER,
  PRIMARY KEY (nsucursal),
  FOREIGN KEY (Director) REFERENCES Directores(id)
  ON DELETE SET NULL
);
```

Bases de datos de ejemplo: Entidades bancarias y clientes

- **Nombre de las restricciones:** Las restricciones de clave foránea al igual que otras restricciones que queramos asociar con la tabla pueden tener un nombre asociado. Esto es útil cuando querramos alterar dicha restricción o incluso inhabilitarla. Lo veremos en próximas unidades. La sintaxis sería: CONSTRAINT nombre_constraint restriccion
- Un ejemplo para nombrar la restricción de clave foránea de Sucursales sería:

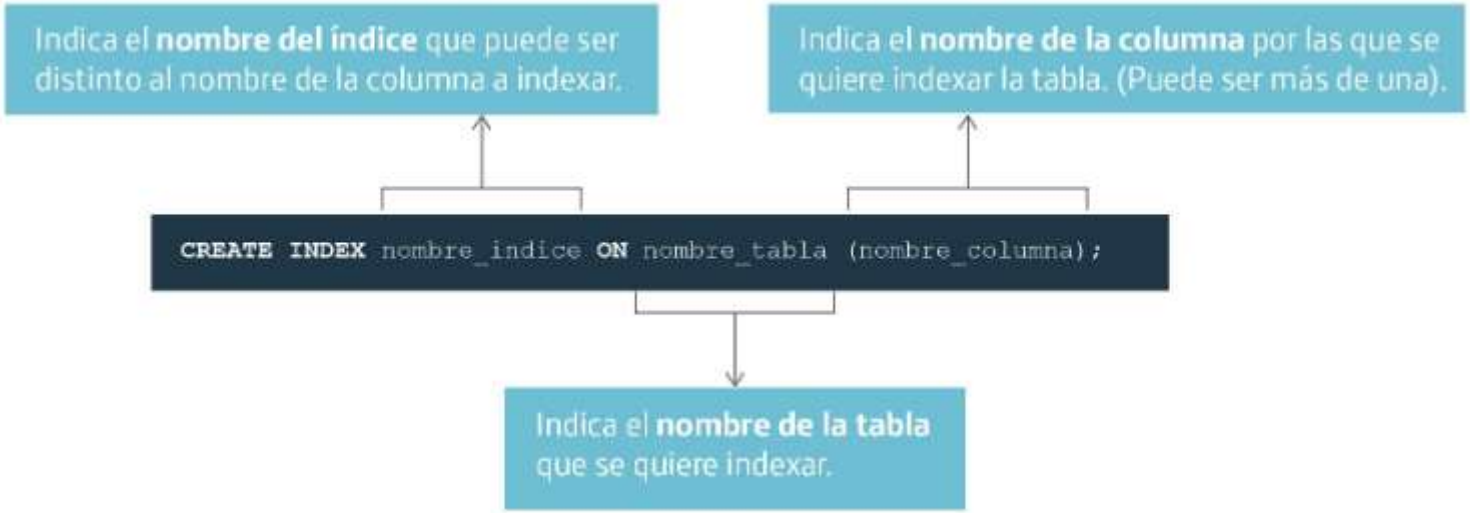
```
CREATE TABLE Sucursales
(
  nSucursal INTEGER,
  Nombre VARCHAR(50),
  Direccion VARCHAR(50),
  Director INTEGER,
  PRIMARY KEY (nsucursal),
  CONSTRAINT Sucursales_Directores_FK FOREIGN KEY (Director)
  REFERENCES Directores(id)
  ON DELETE SET NULL
);
```

- En este caso hemos nombrado a la restricción Sucursales_Directores_FK

UF-3 FORMULACIÓN DE CONSULTAS BÁSICAS

Bases de datos de ejemplo: Entidades bancarias y clientes

- **Creación de índices:** El índice en las bases de datos tiene la misma función que su homónimo en un libro; en el libro se llega a la información requerida más rápido si buscamos en el índice que hojeando el libro. En las bases de datos, los índices permiten ejecutar las operaciones más rápidas ya que en lugar de buscar la información en toda la tabla se busca primero en el índice (más ligero, en principio). El índice no es más que una copia menor de la tabla, guardando solo las columnas indexadas.
- **El índice no es igual que la clave primaria**, no identifica inequívocamente un registro solo se utiliza para saber por dónde buscar de forma más rápida. Por lo que es posible repetir valores en una columna indexada, incluso es posible tener nulos. Para indexar una columna se utiliza la sentencia CREATE INDEX que tiene la sintaxis:



Bases de datos de ejemplo: Entidades bancarias y clientes

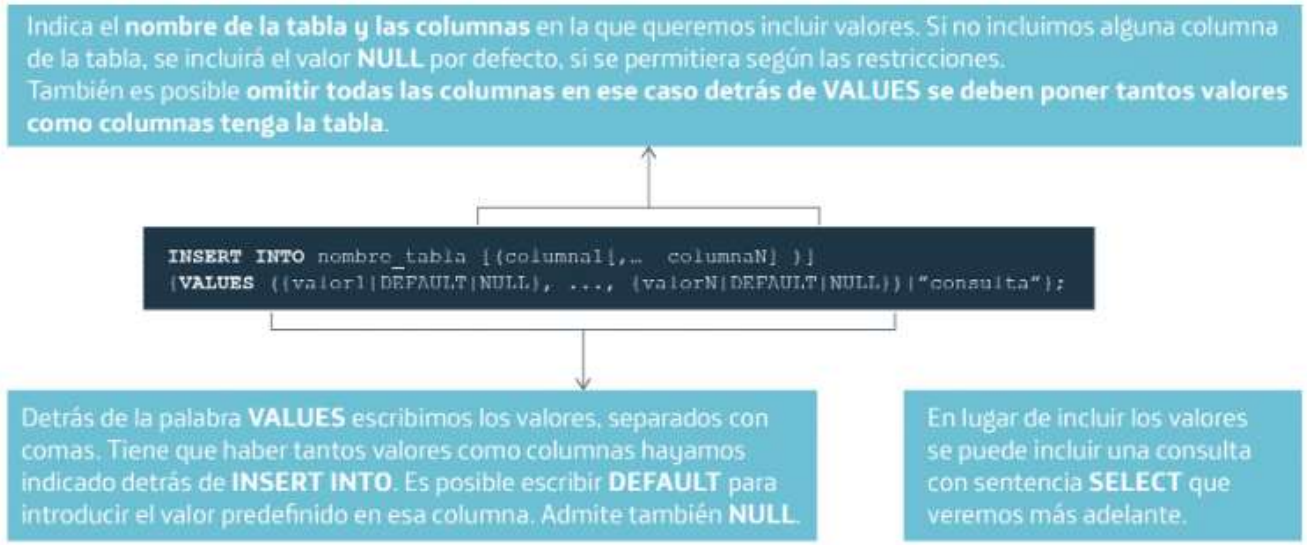
- **Ejemplo de creación de índice:** En el ejemplo que venimos trasladando podemos incluir índices en las columnas de Apellidos y Nombre de la tabla cliente. De esta forma si en alguna ocasión queremos buscar en la tabla un cliente por apellidos o nombre, podemos encontrar más rápidamente al cliente. En cuanto la tabla Sucursales también sería bueno tener índices para los directores.
- Las sentencias a utilizar son:
  - CREATE INDEX indiceCli ON Clientes (Nombre, Apellidos);
  - CREATE INDEX indiceSuc ON Sucursales (Director);

UF-3 FORMULACIÓN DE CONSULTAS BÁSICAS

Bases de datos de ejemplo: Entidades bancarias y clientes

- **Inserción, Modificación, Borrado:** Una vez creada la estructura de la base de datos y tablas, podemos operar con los registros. Las operaciones que nos permiten gestionar los registros son:
  - Inserción
  - Modificación
  - Borrado

- **Inserción:** Existen dos formas para insertar registros en una tabla:
  - Podemos introducirlo registro a registro o
  - Introducirlo a través de otras tablas existentes realizando una consulta (veremos más adelante como realizar consultas).
- La sintaxis es:



UF-3 FORMULACIÓN DE CONSULTAS BÁSICAS

Bases de datos de ejemplo: Entidades bancarias y clientes

- En este ejemplo vamos a introducir los valores en la tabla Clientes. Pero lo primero que hay que hacer es incluir las sucursales de esos clientes porque ambas tablas están relacionadas a través del número de sucursal y queremos asignarle una sucursal. Sería posible dejarla a NULL, pero no es el caso.
- Lo mismo ocurre con la tabla Directores, es necesario completarla primero. Inserción en la tabla Directores:

```
Insert into DIRECTORES (ID,NOMBRE,APELLIDOS,SUELDO)
values (12,'Alberto','Pérez Martín',1800);
Insert into DIRECTORES (ID,NOMBRE,APELLIDOS,SUELDO)
values (15,'Antonio','López López',1500);
Insert into DIRECTORES (ID,NOMBRE,APELLIDOS,SUELDO)
values (20,'Silvia','Martín Martín',1300);
Insert into DIRECTORES (ID,NOMBRE,APELLIDOS,SUELDO)
values (13,'Pedro','García Martín',1900);
```

- Inserción en la tabla Sucursales

```
Insert into SUCURSALES (NSUCURSAL,NOMBRE,DIRECTOR,DIRECCION)
values (1001,'Sucursal Centro',12,'Avd. del Estilo, 45');
Insert into SUCURSALES (NSUCURSAL,NOMBRE,DIRECTOR,DIRECCION)
values (1002,'Sucursal Oeste',15,'Avd. Mediterráneo, 14');
Insert into SUCURSALES (NSUCURSAL,NOMBRE,DIRECTOR,DIRECCION)
values (1003,'Sucursal Este',20,'Calle Martínez, 45');
Insert into SUCURSALES (NSUCURSAL,NOMBRE,DIRECTOR,DIRECCION)
values (1004,'Sucursal Norte',null,'Calle Especias, 23');
```

Bases de datos de ejemplo: Entidades bancarias y clientes

- Ahora ya podemos insertar los clientes sin problemas.

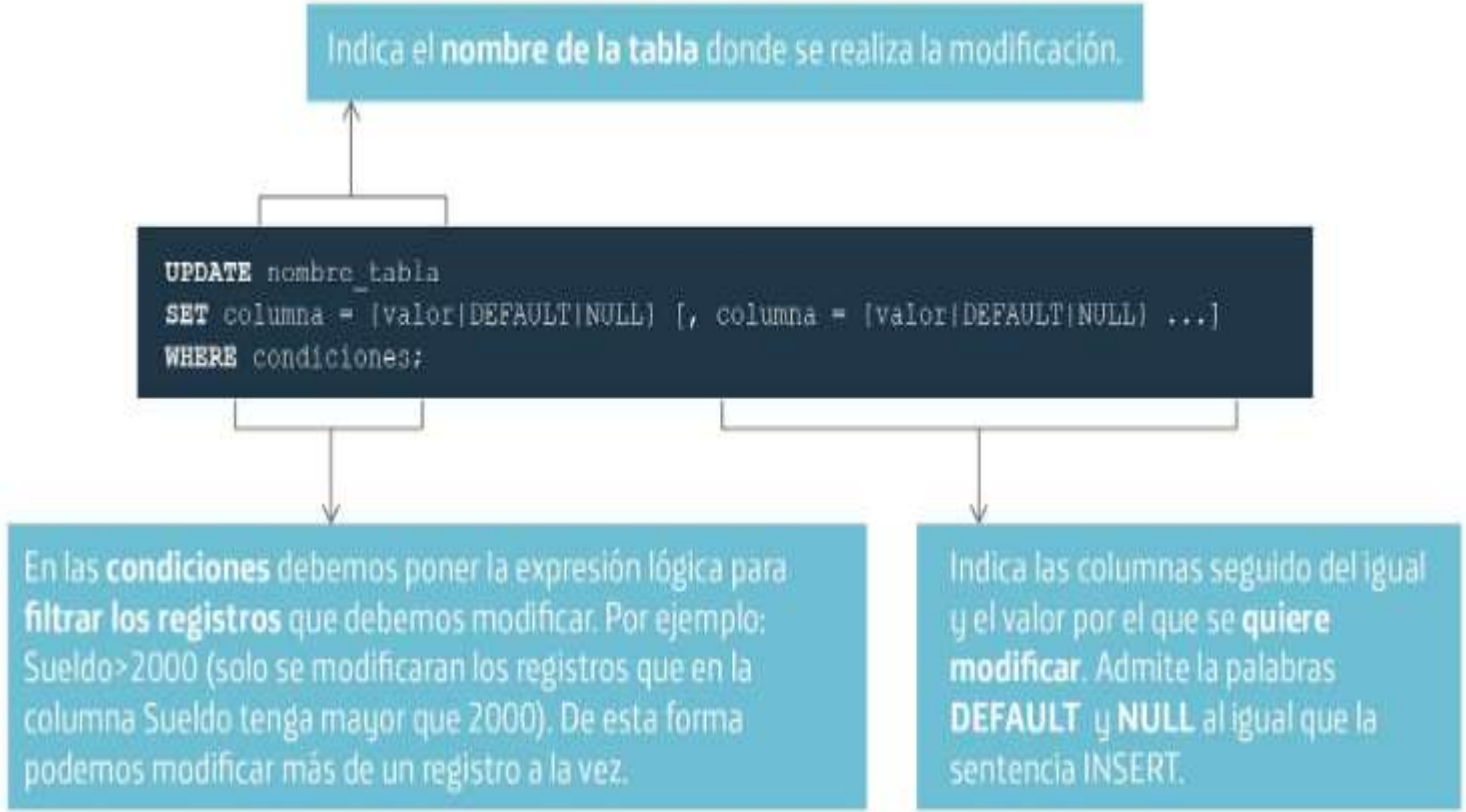
```
Insert into CLIENTES (DNI,NOMBRE,APELLIDOS,DIRECCION,
FECHA_DE_ALTA,CREDITO,NSUCURSAL)
values ('30515454K','Ana','Martín Martín',
'Calle del Socorro, 1','08/01/18',1500,1001);

Insert into CLIENTES (DNI,NOMBRE,APELLIDOS,DIRECCION,
FECHA_DE_ALTA,CREDITO,NSUCURSAL)
values ('33358796A','Marta','López Ruiz',
'Calle Martinez, 76','14/09/17',600,1001);

Insert into CLIENTES (DNI,NOMBRE,APELLIDOS,DIRECCION,
FECHA_DE_ALTA,CREDITO,NSUCURSAL)
values ('78458784B','Antonio','Castillo Mentas',
'Calle Soles, 14',null,500,1002);
```

Bases de datos de ejemplo: Entidades bancarias y clientes

- **Modificación:** Una vez introducido los registros es posible modificarlos a través de la sentencia UPDATE. La sintaxis es:



- Ejemplo: Modificamos los valores en la tabla clientes cambiando el nombre y apellidos de un cliente. Utilizamos para filtrar su clave primaria, por lo tanto solo vamos a modificar un registro:

```
UPDATE Clientes
SET Apellidos = 'Fernández López',
Nombre='Antonio'
WHERE DNI='70912547A';
```

Bases de datos de ejemplo: Entidades bancarias y clientes

- **Borrado:** Para eliminar un registro de una tabla se utiliza la sentencia DELETE. La sintaxis es:

- Ejemplo: Borramos algunos registros de la tabla clientes. Supongamos que queremos borrar los clientes antiguos:

```
DELETE FROM Clientes
WHERE fecha_de_alta <=
'1/1/1984';
```

UF-3 FORMULACIÓN DE CONSULTAS BÁSICAS

- **Consultas en SQL:** Una consulta es la operación que permite separar de una o varias tablas los datos necesarios.
- Hacer consultas y obtener relaciones de datos que se encuentran en tablas diferentes, para crear vistas, para filtrar a la hora de modificar o borrar, etc. Es la sentencia más utilizada.
- Para realizar una consulta se utiliza la sentencia SELECT. Esta sentencia es muy poderosa y ampliamente rica en opciones que permiten realizar todas las operaciones del algebra relacional.
- La sintaxis original resumida es:

```
SELECT nombre_de_columnas
    FROM tabla_origen
    [WHERE condición]
    [GROUP BY nombre_columna_agrupación]
    [HAVING condición_group_by]
    [ORDER BY nombre_columna];
```

- **Selección de columnas:** La sentencia SELECT básica es seleccionar las columnas de una tabla. Se incluyen todas las filas de una tabla.

- **Una opción para este formato básico es incluir la palabra DISTINCT después de SELECT**. Esto permite obtener los registros sin repeticiones. Su sintaxis es:

La clausula **SELECT DISTINCT** tiene la misma sintaxis que SELECT lo único que varia es el resultado que no repite registros con el mismo valor de columna.

```
SELECT DISTINCT nombre_columna [[AS] col_renombrada]
[,nombre_columna [[AS] col_renombrada]...]
FROM tabla_a_consultar [[AS] tabla_renombrada];
```

- En este ejemplo seleccionamos todos los clientes de nuestra empresa y mostramos solo las columnas de DNI (a la que le incluimos un Alias), nombre y apellidos.

```
SELECT DNI AS Identificación, Nombre,
Apellidos
FROM Clientes;
```

- En este otro ejemplo, seleccionamos todos los nombres de la tabla clientes, pero no saldrán los nombres repetidos.

```
SELECT DISTINCT Nombre FROM
Clientes;
```

❑ Selección de filas

En el formato anterior no podíamos filtrar los registros de una tabla, se seleccionaban todos. Para poder filtrar los registros debemos añadir a la sentencia SELECT la cláusula WHERE.

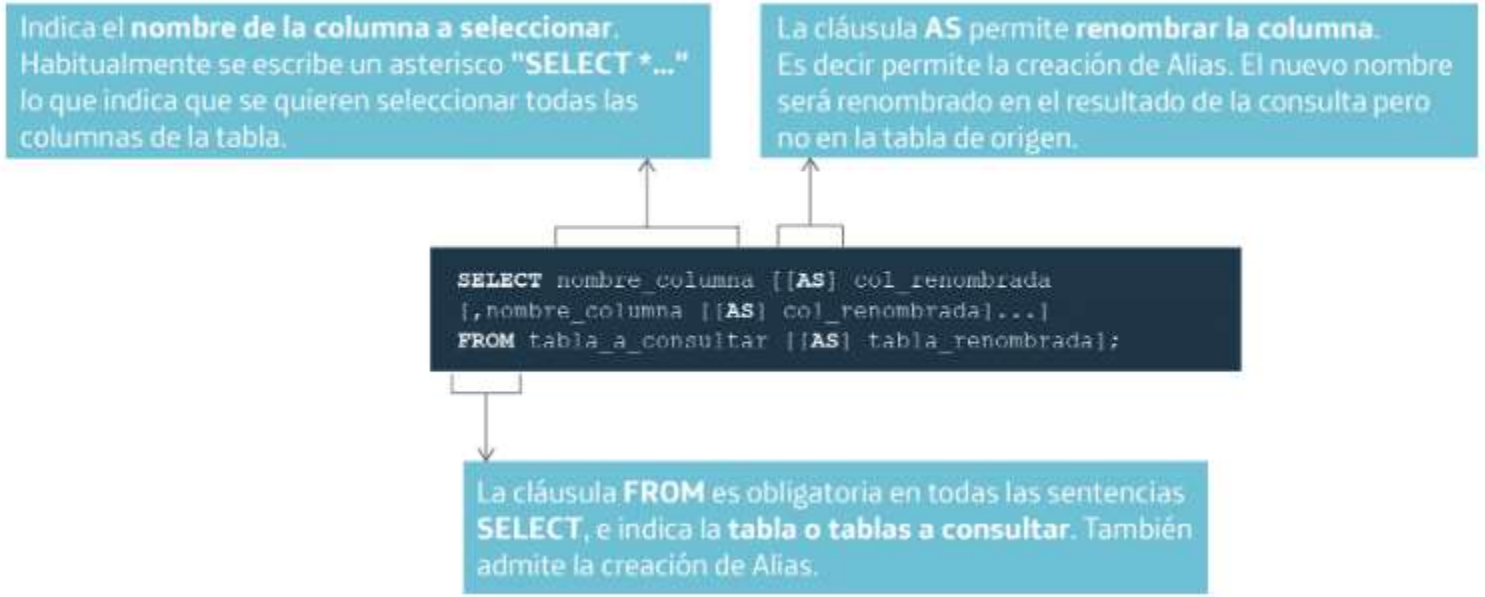
Su sintaxis es:

```
SELECT nombre_columna [,nombre_columna,…]
FROM tabla_a_consultar
WHERE condicion;
```

Hay que tener en cuenta que, al realizar una selección, la tabla generada es temporal.

❑ Selección de columnas

La sentencia **SELECT básica es seleccionar las columnas de una tabla**. Se incluyen todas las filas de una tabla.



❑ Selección de columnas

Una opción para este formato básico es incluir la palabra DISTINCT después de SELECT. Esto permite obtener los registros sin repeticiones. Su sintaxis es:

La clausula **SELECT DISTINCT** tiene la misma sintaxis que SELECT lo único que varia es el resultado que no repite registros con el mismo valor de columna.

```
SELECT DISTINCT nombre_columna [[AS] col_renombrada]
[,nombre_columna [[AS] col_renombrada]...]
FROM tabla_a_consultar [[AS] tabla_renombrada];
```

❑ Ordenación del resultado de una consulta

Es posible que al hacer la consulta los registros obtenidos se **muestren ordenados respecto a una o varias columnas**. La cláusula que permite la ordenación en las consultas es **ORDER BY**:

- **Ordenación del resultado de una consulta:** Ejemplo: Si queremos ordenar los clientes según su fecha de alta, del más antiguo al más nuevo, y dentro de esta clasificación ordenarlo por orden alfabético según apellidos.

```
SELECT DNI AS Identificación, Nombre, Apellidos, fecha_de_alta
 FROM Clientes
 ORDER BY fecha_de_alta  DESC, Apellidos;
```

- Funciones de columna: En ocasiones es necesario contar datos: ¿cuántos clientes hay en Castellón? O también hacer cálculos sobre ellos ¿a cuánto asciende el iva cobrado en la factura 3752? SQL proporciona una serie de funciones que se pueden utilizar en la cláusula SELECT y que actúan sobre los valores de las columnas para realizar diversas operaciones como, por ejemplo, sumarlos, obtener el valor máximo o el valor medio, entre otros. Las funciones de columna más habituales son las que se muestran a continuación:

  - COUNT(*) => cuenta todos los valores: nulos y no nulos.
  - COUNT(columna) => cuenta los valores no nulos.
  - SUM(columna) Suma los valores de la columna.
  - MAX(columna) Obtiene el valor máximo de la columna.
  - MIN(columna) Obtiene el valor mínimo de la columna.
  - AVG(columna) Obtiene la media de los valores de la columna

❑ Instrucción DML (Data Definition Languaje - lenguaje de definición de datos): SELECT

❑ A continuación se muestran algunos ejemplos:

```
SELECT AVG(cant)
FROM lineas_fac; -- cantidad media por línea de factura

SELECT AVG(cant)
FROM lineas_fac -- cantidad media por línea de factura
WHERE codart = 'TLFXK2'; -- del artículo TLFXK2

SELECT SUM(cant) AS suma, COUNT(*) AS lineas
FROM lineas_fac; -- se puede hacer varios cálculos a la vez
```

```
SELECT
SUM(SUELDO)/count(distinct(APELLIDOS)) AS
MEDIASALARIOSFROM EMPLEADOS;
```

- ❑ COUNT(*) Cuenta filas.
- ❑ COUNT(columna) Cuenta el número de valores no nulos en la columna.
- ❑ COUNT(DISTINCT columna) Cuenta el número de valores distintos y no nulos en la columna.

- Tengo la tabla P presentada debajo y se ha ejecutado la siguiente sentencia:

```
SELECT COUNT(*) AS cuenta1, COUNT(color) AS cuenta2,
COUNT(DISTINCT color) AS cuenta3
FROM P;
```

pnum	pnombre	color	peso	ciudad
P1	tuerca	verde	12	París
P2	perno	rojo		Londres
P3	birlo	azul	17	Roma
P4	birlo	rojo	14	Londres
P5	leva		12	París
P6	engrane	rojo	19	París

- La salida saldría:

cuenta1	cuenta2	cuenta3
6	5	3

❑ Instrucción DML (Data Definition Languaje - lenguaje de definición de datos): SELECT

- ❑ Las funciones de columna (SUM, MAX, MIN, AVG) ignoran los nulos, es decir, los nulos no se tienen en cuenta en los cálculos. Según esto ¿coincidirá siempre el valor de media1 y media2 al ejecutar la siguiente sentencia?

```
SELECT AVG(dto) AS media1, SUM(dto)/COUNT(*) AS media2
FROM lineas_fac;
```

- ❑ La respuesta es no. En media1 se devuelve el valor medio de los descuentos no nulos, mientras que en media2 se devuelve el valor medio de los descuentos (interpretándose los descuentos nulos como el descuento cero). Como se ha visto, la función AVG calcula la media de los valores no nulos de una columna. Si la tabla de la cláusula FROM es la de artículos, la media es por artículo; si la tabla de la cláusula FROM es la de facturas, la media es por factura. Cuando se quiere calcular otro tipo de media se debe hacer el cálculo mediante un cociente. Por ejemplo, el número medio de facturas por mes durante el año pasado se obtiene dividiendo el número de facturas del año pasado entre doce meses:

```
SELECT COUNT(*)/12 AS media_mensual
FROM facturas
WHERE EXTRACT(year FROM fecha) = EXTRACT(year FROM CURRENT_DATE) - 1;
```

- **Cláusula GROUP BY**: La cláusula GROUP BY forma grupos con las filas que tienen en común los valores de una o varias columnas. Sobre cada grupo se pueden aplicar las funciones de columna que se han estado utilizando hasta ahora (SUM, MAX, MIN, AVG, COUNT), denominándose ahora funciones de grupo.
- Estas funciones, utilizadas en la cláusula SELECT, se aplican una vez para cada grupo. La siguiente sentencia cuenta cuántas facturas tiene cada cliente el año pasado:

```
SELECT codcli, COUNT(*)
FROM facturas
WHERE EXTRACT(year FROM fecha) = EXTRACT(year FROM CURENT_DATE) - 1
GROUP BY codcli;
```

- El modo en que se ejecuta la sentencia se explica a continuación. Se toma la tabla de facturas (FROM) y se seleccionan las filas que cumplen la restricción (WHERE). A continuación, las facturas se separan en grupos, de modo que en un mismo grupo sólo hay facturas de un mismo cliente (GROUP BY codcli), habiendo tantos grupos como clientes hay con facturas del año pasado.
- Finalmente, de cada grupo se muestra el código del cliente y el número de facturas que hay en el grupo (son las facturas de ese cliente): COUNT(*).

- **Instrucción DML (Data Definition Languaje - lenguaje de definición de datos): SELECT**

- **Cláusula HAVING:** En la cláusula HAVING, que puede aparecer tras GROUP BY, se utilizan las funciones de grupo para hacer restricciones sobre los grupos que se han formado. La sintaxis de la sentencia SELECT, tal y como se ha visto hasta el momento, es la siguiente:

```
SELECT [ DISTINCT ] { * | columna [ , columna ] }
FROM tabla
[ WHERE condición_de_búsqueda ]
[ GROUP BY columna [, columna ]
[ HAVING condición_para_el_grupo ] ]
[ ORDER BY columna [ ASC | DESC ] [ , columna [ ASC | DESC ] ];
```

- En las consultas que utilizan GROUP BY se obtiene una fila por cada uno de los grupos producidos. Para ejecutar la cláusula GROUP BY se parte de las filas de la tabla que cumplen el predicado establecido en la cláusula WHERE y se agrupan en función de los valores comunes en la columna o columnas especificadas. Mediante la cláusula HAVING se realiza una restricción sobre los grupos obtenidos por la cláusula GROUP BY, seleccionándose aquellos que cumplen el predicado establecido en la condición. Es importante destacar que en la condición de la cláusula HAVING sólo pueden aparecer columnas por las que se ha agrupado y funciones de grupo sobre cualquier otra columna de la tabla. Lo mismo ocurre en la cláusula SELECT: sólo se pueden mostrar columnas que aparecen en la cláusula GROUP BY y funciones de grupo sobre cualquier otra columna. Cuando en las cláusulas SELECT o HAVING aparecen columnas que no se han especificado en la cláusula GROUP BY y que tampoco están afectadas por una función de grupo, se produce un error.

- **Instrucción DML (Data Definition Languaje - lenguaje de definición de datos): SELECT**
- **Cláusula HAVING:** Ejemplo se quiere obtener un listado con los clientes que tienen más de cinco facturas con 16 % de iva, indicando cuántas de ellas tiene cada uno. Para resolver este ejercicio se deben tomar las facturas (tabla FACTURAS) y seleccionar aquellas con 16 % de iva (WHERE). A continuación se deben agrupar las facturas (GROUP BY) de manera que haya un grupo para cada cliente (columna codcli). Una vez formados los grupos, se deben seleccionar aquellos que contengan más de cinco facturas (HAVING). Por último, se debe mostrar (SELECT) el código de cada cliente y su número de facturas:

```
SELECT codcli, COUNT(*) AS facturas
FROM facturas
WHERE iva = 16
GROUP BY codcli
HAVING COUNT(*) > 5;
```

```
SELECT idempleado, COUNT(*) AS empleados
FROM empleados
WHERE sexo='M’
GROUP BY idempleado
HAVING COUNT(*) > 1;
```

- **Instrucción DML (Data Definition Languaje - lenguaje de definición de datos): SELECT**
  - Ejemplo: Se quiere obtener un listado con el número de facturas que hay en cada año, de modo que aparezca primero el año con más facturas. Además, para cada año se debe mostrar el númerode clientes que han hecho compras y en cuántos días del año se han realizado:

```
SELECT EXTRACT(year FROM fecha) AS año, COUNT(*) AS nfacturas,
COUNT(DISTINCT codcli) AS nclientes,
COUNT(DISTINCT codven) AS nvendedores,
COUNT(DISTINCT fecha) AS ndias
FROM facturas
GROUP BY EXTRACT(year FROM fecha)
ORDER BY nfacturas DESC; -- nfacturas es el nombre que se ha
-- dado a COUNT(*) en el SELECT
```

  - Como se ve en el ejemplo, es posible utilizar expresiones en la cláusula GROUP BY. Por otra parte, el ejemplo también muestra cómo en la cláusula ORDER BY se puede hacer referencia a los nombres con que se renombran las expresiones del SELECT. Esto es así porque la cláusula ORDER BY se ejecuta tras el SELECT.

❑ Instrucción DML (Data Definition Languaje - lenguaje de definición de datos): SELECT

- ❑ A continuación se plantean algunas cuestiones que es importante tener en cuenta cuando se realizan agrupaciones.
- ❑ Cuando se utilizan funciones de grupo en la cláusula SELECT sin que haya GROUP BY, el resultado de ejecutar la consulta tiene una sola fila.
- ❑ A diferencia del resto de funciones que proporciona SQL, las funciones de grupo sólo se utilizan en las cláusulas SELECT y HAVING, nunca en la cláusula WHERE; la sentencia SELECT tiene dos cláusulas para realizar restricciones: WHERE y HAVING.
- ❑ Es muy importante saber situar cada restricción en su lugar: las restricciones que se deben realizar a nivel de filas, se sitúan en la cláusula WHERE.
- ❑ Las restricciones que se deben realizar sobre grupos (normalmente involucran funciones de grupo), se sitúan en la cláusula HAVING.
- ❑ El modificador DISTINCT puede ser necesario en la cláusula SELECT de una sentencia que tiene GROUP BY sólo cuando las columnas que se muestren en la cláusula SELECT no sean todas las que aparecen en la cláusula GROUP BY.
- ❑ Una vez formados los grupos mediante la cláusula GROUP BY (son grupos de filas, no hay que olvidarlo), del contenido de cada grupo sólo es posible conocer el valor de las columnas por las que se ha agrupado (ya que dentro del grupo, todas las filas tienen dichos valores en común), por lo que sólo estas columnas son las que pueden aparecer, directamente, en las cláusulas SELECT y HAVING. Además, en estas cláusulas, se pueden incluir funciones de grupo que actúen sobre las columnas que no aparecen en la cláusula GROUP BY.

- **Instrucción DML (Data Definition Languaje - lenguaje de definición de datos): Insercción de datos**
- Una vez creada una tabla podemos introducir **nuevos datos** en ella mediante la sentencia INSERT, como se muestra en los siguientes ejemplos:

```
INSERT INTO facturas (codfac, fecha, codcli, codven, iva, dto ) VALUES (6600, '30/04/2007', 111, 55, 0, NULL);
INSERT INTO lineas_fac (codfac, linea, cant, codart, precio, dto) VALUES (6600, 1, 4, 'L76425', 3.16, 25 );
INSERT INTO lineas_fac (codfac, linea, cant, codart, precio, dto) VALUES (6600, 2, 5, 'B14017', 2.44, 25 );
INSERT INTO lineas_fac (codfac, linea, cant, codart, precio, dto) VALUES (6600, 3, 7, 'L92117', 4.39, 25 );
```

- Mediante estas sentencias se ha introducido la cabecera de una factura y tres de sus líneas. Nótese
- que tanto las cadenas de caracteres como las fechas, se introducen entre comillas simples. Para introducir nulos se utiliza la expresión NULL.
- Algunos SGBD relacionales permiten insertar varias filas en una misma tabla mediante una sola sentencia INSERT, realizando las inserciones de un modo más eficiente que si se hace mediante varias sentencias independientes. Así, la tres inserciones que se han realizado en la tabla LINEAS_FAC también se pueden realizar mediante la siguiente sentencia:

```
INSERT INTO lineas_fac (codfac, linea, cant, codart, precio, dto)
VALUES (6600, 1, 4, 'L76425', 3.16, 25 ), (6600, 2, 5, 'B14017', 2.44, 25 ), (6600, 3, 7, 'L92117', 4.39, 25 );
```

- **Instrucción DML (Data Definition Languaje - lenguaje de definición de datos):** Actualización y eliminación de datos
- Una vez insertados los datos es posible actualizarlos o eliminarlos mediante las sentencias UPDATE y DELETE, respectivamente. Para comprender el funcionamiento de estas dos sentencias es imprescindible conocer bien el funcionamiento de la sentencia SELECT. Esto es así porque para poder actualizar o eliminar datos que se han almacenado es preciso encontrarlos antes. Y por lo tanto, la cláusula de estas sentencias que establece las condiciones de búsqueda de dichos datos (WHERE) se especifica del mismo modo que las condiciones de búsqueda cuando se hace una consulta:

```
UPDATE   facturas
   SET   dto = 0
   WHERE dto IS NULL;
```

```
DELETE FROM facturas
   WHERE codcli = 333;
```

```
UPDATE facturas
   SET codven = 105
   WHERE codven IN ( SELECT codven
                     FROM   vendedores
                     WHERE  codjefe = 105 );
```

```
DELETE FROM facturas
   WHERE iva = ( SELECT MIN(iva)
                 FROM   facturas );
```

- **Instrucción DML (Data Definition Languaje - lenguaje de definición de datos):** Actualización y eliminación de datos
- Una vez insertados los datos es posible actualizarlos o eliminarlos mediante las sentencias UPDATE y DELETE, respectivamente. Para comprender el funcionamiento de estas dos sentencias es imprescindible conocer bien el funcionamiento de la sentencia SELECT. Esto es así porque para poder actualizar o eliminar datos que se han almacenado es preciso encontrarlos antes. Y por lo tanto, la cláusula de estas sentencias que establece las condiciones de búsqueda de dichos datos (WHERE) se especifica del mismo modo que las condiciones de búsqueda cuando se hace una consulta:

```
UPDATE   facturas
   SET   dto = 0
   WHERE dto IS NULL;
```

```
DELETE FROM facturas
   WHERE codcli = 333;
```

```
UPDATE facturas
   SET codven = 105
   WHERE codven IN ( SELECT codven
                     FROM   vendedores
                     WHERE  codjefe = 105 );
```

```
DELETE FROM facturas
   WHERE iva = ( SELECT MIN(iva)
                 FROM   facturas );
```

- **Instrucción DML (Data Definition Languaje - lenguaje de definición de datos):** Subconsultas en INSERT Hay un tipo de consulta, llamada de adición de datos, que permite rellenar datos de una tabla copiando el resultado de una consulta.
- Se hace mediante la instrucción INSERT y ese relleno se basa en una consulta SELECT que poseerá los datos a añadir. Lógicamente el orden de esos campos debe de coincidir con la lista de campos indicada en la instrucción INSERT y la nueva tabla debe estar previamente creada.

```
INSERT INTO tabla (campo1, campo2,...)
SELECT campoCompatibleCampo1, campoCompatibleCampo2,...
FROM lista DeTablas [...otras cláusulas del SELECT...]
```

- Ejemplo: Insertar en una nueva tabla los clientes que tengan un crédito superior a 8200 Euros:

```
INSERT INTO clientes_credito(dni,nombre,apellidos,direccion)
        SELECT dni,nombre,apellidos,direccion
                FROM clientes WHERE credito>8200;

            INSERT INTO empleados1(idempleado,nombre,apellidos,sexo, municipio, fecha_nac, sueldo, idpto)
                    SELECT idempleado,nombre,apellidos,sexo, municipio, fecha_nac, sueldo, idpto
                    FROM empleados
                    WHERE sueldo>1500;
```

- **Instrucción DML (Data Definition Languaje - lenguaje de definición de datos):** Subconsultas en UPDATE
- La instrucción UPDATE permite modificar filas. Es muy habitual el uso de la cláusula WHERE para indicar las filas que se modificarán. Esta cláusula se puede utilizar con las mismas posibilidades que en el caso del SELECT, por lo que es posible utilizar subconsultas.
- Ejemplo: Esta instrucción aumenta un 40% el crédito de los clientes que son de la Sucursal Centro. En primer lugar se averigua qué código tiene esa sucursal y posteriormente se utiliza para actualizar el cliente:

```
UPDATE clientes
SET credito=credito*1.10
WHERE nsucursal= (SELECT nsucursal
                  FROM sucursales
                  WHERE nombre='Sucursal Centro');
```

```
UPDATE empleados
SET sueldo=sueldo*1.20
WHERE sueldo= (SELECT sueldo
               FROM empleados
               WHERE municipio='Madrid');
```

También podemos utilizar subconsultas en la cláusula SET de la instrucción UPDATE. Ejemplo: Actualiza el crédito de los clientes de la sucursal 1002 al máximo de los créditos de la sucursal 1001.

```
UPDATE clientes SET
credito = (SELECT max(credito)
           FROM clientes
           WHERE nsucursal=1001)
WHERE nsucursal=1002;
```

```
UPDATE empleados
SET sueldo = (SELECT max(sueldo)
              FROM empleados)
where municipio='Sevilla';
```

Instrucción DML (Data Definition Languaje - lenguaje de definición de datos): Subconsultas en DELETE

Al igual que en el caso de las instrucciones INSERT o SELECT, **DELETE dispone de cláusula WHERE** y en dicha cláusula podemos utilizar subconsultas.

Ejemplo: Borrar los clientes que sean de la Sucursal Centro.

```
DELETE clientes
WHERE nsucursal IN  (SELECT nsucursal
                     FROM sucursales
                     WHERE nombre='Sucursal Centro');

DELETE from empleados
WHERE municipio IN  (SELECT municipio
                     FROM empleados
                     WHERE municipio='Cordoba');

DELETE from empleados
WHERE municipio IN  (SELECT municipio
                     FROM empleados
                     WHERE municipio='Sevilla’ or municipio='Madrid');
```