El lenguaje SQL

Carme Martín Escofet

Índice

Introducción

El SQL es el lenguaje estándar ANSI/ISO de definición, manipulación y control de bases de datos relacionales. Es un lenguaje declarativo: sólo hay que indicar qué se quiere hacer. En cambio, en los lenguajes procedimentales es necesario especificar cómo hay que hacer cualquier acción sobre la base de datos. El SQL es un lenguaje muy parecido al lenguaje natural; concretamente, se parece al inglés, y es muy expresivo. Por estas razones, y como lenguaje estándar, el SQL es un lenguaje con el que se puede acceder a todos los sistemas relacionales comerciales.

Recordad que el álgebra relacional, que hemos visto en la unidad "El modelo relacional y el álgebra relacional", es un lenguaje procedimental.

Empezamos con una breve explicación de la forma en que el SQL ha llegado a ser el lenguaje estándar de las bases de datos relacionales:

1) Al principio de los años setenta, los laboratorios de investigación Santa Teresa de IBM empezaron a trabajar en el proyecto System R. El objetivo de este proyecto era implementar un prototipo de SGBD relacional; por lo tanto, también necesitaban investigar en el campo de los lenguajes de bases de datos relacionales. A mediados de los años setenta, el proyecto de IBM dio como resultado un primer lenguaje denominado SEQUEL (*Structured English Query Language*), que por razones legales se denominó más adelante *SQL* (*Structured Query Language*). Al final de la década de los setenta y al principio de la de los ochenta, una vez finalizado el proyecto System R, IBM y otras empresas empezaron a utilizar el SQL en sus SGBD relacionales, con lo que este lenguaje adquirió una gran popularidad.

2) En 1982, ANSI (*American National Standards Institute*) encargó a uno de sus comités (X3H2) la definición de un lenguaje de bases de datos relacionales. Este comité, después de evaluar diferentes lenguajes, y ante la aceptación comercial del SQL, eligió un lenguaje estándar que estaba basado en éste prácticamente en su totalidad. El SQL se convirtió oficialmente en el lenguaje estándar de ANSI en el año 1986, y de ISO (*International Standards Organization*) en 1987. También ha sido adoptado como lenguaje estándar por FIPS (*Federal Information Processing Standard*), Unix X/Open y SAA (*Systems Application Architecture*) de IBM.

3) En el año 1989, el estándar fue objeto de una revisión y una ampliación que dieron lugar al lenguaje que se conoce con el nombre de SQL1 o SQL89. En el año 1992 el estándar volvió a ser revisado y ampliado considerablemente para cubrir carencias de la versión anterior. Esta nueva versión del SQL, que se conoce con el nombre de SQL2 o SQL92, es la que nosotros presentaremos en esta unidad didáctica.

Como veremos más adelante, aunque aparezca sólo la sigla SQL, siempre nos estaremos refiriendo al SQL92, ya que éste tiene como subconjunto el SQL89; por lo tanto, todo lo que era válido en el caso del SQL89 lo continuará siendo en el SQL92. a

De hecho, se pueden distinguir tres niveles dentro del SQL92:

1) El nivel introductorio (*entry*), que incluye el SQL89 y las definiciones de *clave primaria* y *clave foránea* al crear una

tabla.

El concepto de *clave primaria* y su importancia en una relación o tabla se ha visto en la unidad "El modelo relacional y el álgebra relacional" de este curso.

2) El nivel intermedio (*intermediate*), que, además del SQL89, añade algunas ampliaciones del SQL92.

3) El nivel completo (*full*), que ya tiene todas las ampliaciones del SQL92.

El modelo relacional tiene como estructura de almacenamiento de los datos las relaciones. La intensión o esquema de una relación consiste en el nombre que hemos dado a la relación y un conjunto de atributos. La extensión de una relación es un conjunto de tuplas. Al trabajar con SQL, esta nomenclatura cambia, como podemos apreciar en la siguiente figura:

El modelo relacional se ha presentado en la unidad "El modelo relacional y el álgebra relacional" de este curso.

- Hablaremos de tablas en lugar de relaciones.
- Hablaremos de columnas en lugar de atributos.
- Hablaremos de filas en lugar de tuplas.

Sin embargo, a pesar de que la nomenclatura utilizada sea diferente, los conceptos son los mismos.

Con el SQL se puede definir, manipular y controlar una base de datos relacional. A continuación veremos, aunque sólo en un nivel introductorio, cómo se pueden realizar estas acciones: a

1) Sería necesario crear una tabla que contuviese los datos de los productos de nuestra empresa:

```
CREATE TABLE productos
   (codigo_producto INTEGER,
   nombre_producto CHAR(20),
   tipo CHAR(20),
   descripcion CHAR(50),
   precio REAL,
   PRIMARY KEY (codigo_producto));
```

2) Insertar un producto en la tabla creada anteriormente:

```
INSERT INTO productos
VALUES (1250, 'LENA', 'Mesa', 'Diseño Juan Pi. Año 1920.', 25000);
```

3) Consultar qué productos de nuestra empresa son sillas:

```
SELECT codigo_producto, nombre_producto
FROM productos
WHERE tipo = 'Silla';
```

4) Dejar acceder a uno de nuestros vendedores a la información de la tabla `productos`:

```
GRANT SELECT ON productos TO jmontserrat;
```

Y muchas más cosas que iremos viendo punto por punto en los siguientes apartados.

Fijémonos en la estructura de todo lo que hemos hecho hasta ahora con SQL. Las operaciones de SQL reciben el nombre de sentencias y están formadas por diferentes partes que denominamos cláusulas, tal y como podemos apreciar en el siguiente ejemplo:

```
SELECT codigo_producto, nombre_producto, tipo
FROM productos
WHERE precio > 1000;
```

Esta consulta muestra el código, el nombre y el tipo de los productos que cuestan más de 1.000 euros.

Los tres primeros apartados de este módulo tratan sobre un tipo de SQL denominado SQL interactivo, que permite acceder directamente a una base de datos relacional:

a) En el primer apartado definiremos las denominadas sentencias de definición, donde crearemos la base de datos, las tablas que la compondrán y los dominios, las aserciones y las vistas que queramos.

b) En el segundo aprenderemos a manipular la base de datos, ya sea introduciendo, modificando o borrando valores en las filas de las tablas, o bien haciendo consultas.

c) En el tercero veremos las sentencias de control, que aseguran un buen uso de la base de datos.

Sin embargo, muchas veces querremos acceder a la base de datos desde una aplicación hecha en un lenguaje de programación cualquiera, que nos ofrece mucha más potencia fuera del entorno de las bases de datos. Para utilizar SQL desde un lenguaje de programación necesitaremos sentencias especiales que nos permitan distinguir entre las instrucciones del lenguaje de programación y las sentencias de SQL. La idea es que trabajando básicamente con un lenguaje de programación anfitrión se puede cobijar SQL como si fuese un huésped. Por este motivo, este tipo de SQL se conoce con el nombre de SQL hospedado. Para trabajar con SQL hospedado necesitamos un precompilador que separe las sentencias del lenguaje de programación de las del lenguaje de bases de datos. Una alternativa a esta forma de trabajar son las rutinas SQL/CLI* (*SQL/Call-Level Interface*), que resolviendo también el problema de acceder a SQL desde un lenguaje de programación, no necesitan precompilador.

* Las rutinas SQL/CLI se añadieron al estándar SQL92 en 1995.

Introduciremos SQL hospedado y el concepto de SQL/CLI en el apartado 4 de esta unidad didáctica.

Antes de empezar a conocer el lenguaje, es necesario añadir un último comentario. Aunque SQL es el lenguaje estándar para bases de datos relacionales y ha sido ampliamente aceptado por los sistemas relacionales comerciales, no ha sido capaz de reflejar toda la teoría del modelo relacional establecida por E.F. Codd; esto lo iremos viendo a medida que profundicemos en el lenguaje.

Encontraréis la teoría del modelo relacional de E.F. Codd en la unidad “El modelo relacional y el álgebra relacional” de este curso.

Los sistemas relacionales comerciales y los investigadores de bases de datos son una referencia muy importante para mantener el estándar actualizado. En estos momentos ya se dispone de una nueva versión de SQL92 que se denomina SQL: 1999 o SQL3. SQL: 1999 tiene a SQL92 como subconjunto, e incorpora nuevas prestaciones de gran interés. En informática, en general, y particularmente en bases de datos, es necesario estar siempre al día, y por eso es muy importante tener el hábito de leer publicaciones periódicas que nos informen y nos mantengan al corriente de las novedades. a

Objetivos

Una vez finalizado el estudio de los materiales didácticos de esta unidad, dispondréis de las herramientas indispensables para alcanzar los siguientes objetivos:

1. Conocer el lenguaje estándar ANSI/ISO SQL92.

2. Definir una base de datos relacional, incluyendo dominios, aserciones y vistas.

3. Saber introducir, borrar y modificar datos.

4. Ser capaz de plantear cualquier tipo de consulta a la base de datos.

5. Saber utilizar sentencias de control.

6. Conocer los principios básicos de la utilización del SQL desde un lenguaje de programación.

Sentencias de definición

Para poder trabajar con bases de datos relacionales, lo primero que tenemos que hacer es definirlas. Veremos las órdenes del estándar SQL92 para crear y borrar una base de datos relacional y para insertar, borrar y modificar las diferentes tablas que la componen.

En este apartado también veremos cómo se definen los dominios, las aserciones (restricciones) y las vistas. a

Vistas

Una vista en el modelo relacional no es sino una tabla virtual derivada de las tablas reales de nuestra base de datos, un esquema externo puede ser un conjunto de vistas.

La sencillez y la homogeneidad del SQL92 hacen que:

1) Para crear bases de datos, tablas, dominios, aserciones y vistas se utilice la sentencia **CREATE**.

2) Para modificar tablas y dominios se utilice la sentencia **ALTER**.

3) Para borrar bases de datos, tablas, dominios, aserciones y vistas se utilice la sentencia **DROP**.

La adecuación de estas sentencias a cada caso nos dará diferencias que iremos perfilando al hacer la descripción individual de cada una.

Para ilustrar la aplicación de las sentencias de SQL que veremos, utilizaremos una base de datos de ejemplo muy sencilla de una pequeña empresa con sede en Barcelona, Girona, Lleida y Tarragona, que se encarga de desarrollar proyectos informáticos. La información que nos interesará almacenar de esta empresa, que denominaremos *BDUOC*, será la siguiente:

1) Sobre los empleados que trabajan en la empresa, querremos saber su código de empleado, el nombre y apellido, el sueldo, el nombre y la ciudad de su departamento y el número de proyecto al que están asignados.

2) Sobre los diferentes departamentos en los que está estructurada la empresa, nos interesa conocer su nombre, la ciudad

donde se encuentran y el teléfono. Será necesario tener en cuenta que un departamento con el mismo nombre puede estar en ciudades diferentes, y que en una misma ciudad puede haber departamentos con nombres diferentes.

3) Sobre los proyectos informáticos que se desarrollan, querremos saber su código, el nombre, el precio, la fecha de inicio, la fecha prevista de finalización, la fecha real de finalización y el código de cliente para quien se desarrolla.

4) Sobre los clientes para quien trabaja la empresa, querremos saber el código de cliente, el nombre, el NIF, la dirección, la ciudad y el teléfono.

Creación y borrado de una base de datos relacional

El estándar SQL92 no dispone de ninguna sentencia de creación de bases de datos. La idea es que una base de datos no es más que un conjunto de tablas y, por lo tanto, las sentencias que nos ofrece el SQL92 se concentran en la creación, la modificación y el borrado de estas tablas.

La instrucción
CREATE DATABASE

Muchos de los sistemas relacionales comerciales (como ocurre en el caso de Informix, DB2, SQL Server y otros) han incorporado sentencias de creación
de bases de datos con la siguiente sintaxis:

CREATE DATABASE

En cambio, disponemos de una sentencia más potente que la de creación de bases de datos: la sentencia de creación de esquemas denominada **CREATE SCHEMA**. Con la creación de esquemas podemos agrupar un conjunto de elementos de la base de datos que son propiedad de un usuario. La sintaxis de esta sentencia es la que tenéis a continuación:

```
CREATE SCHEMA {[nombre_esquema]} |[AUTHORIZATION usuario]}
          [lista_de_elementos_del_esquema];
```

La nomenclatura utilizada en la sentencia es la siguiente: a

- Las palabras en negrita son palabras reservadas del lenguaje: a

- La notación [...] quiere decir que lo que hay entre los corchetes se podría poner o no.

- La notación {A| ... |B} quiere decir que tenemos que elegir entre todas las opciones que hay entre las llaves, pero debemos poner una obligatoriamente.

La sentencia de creación de esquemas hace que varias tablas (`lista_de_ele-mentos_del_esquema`) se puedan agrupar bajo un mismo nombre (`nom-bre_esquema`) y que tengan un propietario (`usuario`). Aunque todos los parámetros de la sentencia `CREATE SCHEMA` son opcionales, como mínimo se debe dar o bien el nombre del esquema, o bien el nombre del usuario propietario de la base de datos. Si sólo especificamos el usuario, éste será el nombre del esquema.

La creación de esquemas puede hacer mucho más que agrupar tablas, porque `lista_de_elementos_del_esquema` puede, además de tablas, ser también dominios, vistas, privilegios y restricciones, entre otras cosas.

Para borrar una base de datos encontramos el mismo problema que para crearla. El estándar SQL92 sólo nos ofrece la sentencia de borrado de esquemas **`DROP SCHEMA`**, que presenta la siguiente sintaxis:

```
DROP SCHEMA nombre_esquema {RESTRICT|CASCADE};
```

La sentencia DROP DATABASE

Muchos de los sistemas relacionales comerciales (como por ejemplo Informix, DB2, SQL Server y otros) han incorporado sentencias

de borrado de bases de datos con la siguiente sintaxis:

`DROP DATABASE`

Donde tenemos lo siguiente:

- La opción de borrado de esquemas **`RESTRICT`** hace que el esquema sólo se pueda borrar si no contiene ningún elemento.

- La opción **`CASCADE`** borra el esquema aunque no esté completamente vacío.

Creación de tablas

Como ya hemos visto, la estructura de almacenamiento de los datos del modelo relacional son las tablas. Para crear una tabla, es necesario utilizar la sentencia **CREATE TABLE**. Veamos su formato:

```
CREATE TABLE nombre_tabla
            ( definición_columna
            [, definición_columna...]
            [, restricciones_tabla]
            );
```

Recordad que las tablas se han estudiado en la unidad "El modelo relacional y el álgebra relacional" de este curso.

Donde **definición_columna** es:

```
nombre_columna {tipo_datos|dominio} [def_defecto] [restric_col]
```

El proceso que hay que seguir para crear una tabla es el siguiente: a

1) Lo primero que tenemos que hacer es decidir qué nombre queremos poner a la tabla (`nombre_tabla`).

2) Después, iremos dando el nombre de cada uno de los atributos que formarán las columnas de la tabla (`nombre_columna`).

3) A cada una de las columnas le asignaremos un tipo de datos predefinido o bien un dominio definido por el usuario. También podremos dar definiciones por defecto y restricciones de columna.

4) Una vez definidas las columnas, sólo nos quedará dar las restricciones de tabla.

Tipos de datos

Para cada columna tenemos que elegir entre algún dominio definido por el usuario o alguno de los tipos de datos predefinidos que se describen a continuación:

Tipos de datos predefinidos	
Tipos de datos	Descripción
`CHARACTER (longitud)`	Cadenas de caracteres de longitud fija.
`CHARACTER VARYING (longitud)`	Cadenas de caracteres de longitud variable.
Tipos de datos predefinidos	
Tipos de datos	Descripción
`BIT (longitud)`	Cadenas de bits de longitud fija.

BIT VARYING (longitud)	Cadenas de bits de longitud variables.
NUMERIC (precisión, escala)	Número decimales con tantos dígitos como indique la precisión y tantos decimales como indique la escala.
DECIMAL (precisión, escala)	Número decimales con tantos dígitos como indique la precisión y tantos decimales como indique la escala.
INTEGER	Números enteros.
SMALLINT	Números enteros pequeños.
REAL	Números con coma flotante con precisión predefinida.
FLOAT (precisión)	Números con coma flotante con la precisión especificada.
DOUBLE PRECISION	Números con coma flotante con más precisión predefinida que la del tipo REAL.
DATE	Fechas. Están compuestas de: YEAR año, MONTH mes, DAY día.
TIME	Horas. Están compuestas de HOUR hora, MINUT minutos, SECOND segundos.
TIMESTAMP	Fechas y horas. Están compuestas de YEAR año, MONTH mes, DAY día, HOUR hora, MINUT minutos, SECOND segundos.

Los tipos de datos NUMERIC y DECIMAL

NUMERIC y DECIMAL se describen igual, y es posible utilizar tanto el uno como el otro para definir números decimales.

El tratamiento del tiempo

El estándar SQL92 define la siguiente nomenclatura para trabajar con el tiempo:

`YEAR`
(0001..9999)

`MONTH`
(01..12)

`DAY`
(01..31)

`HOUR`
(00..23)

`MINUT`
(00..59)

`SECOND`
(00..59.precisión)

De todos modos, los sistemas relacionales comerciales disponen de diferentes formatos, entre los cuales podemos elegir cuando tenemos que trabajar con columnas temporales.

Recordad que las correspondencias entre los tipos de datos y los dominios predefinidos del modelo relacional se han visto en el subapartado 2.2 de la unidad "El modelo relacional y el álgebra relacional" de este curso.

Ejemplos de asignaciones de columnas

Veamos algunos ejemplos de asignaciones de columnas en los tipos de datos predefinidos `DATE`, `TIME` y `TIMESTAMP`:

- La columna `fecha_nacimiento` podría ser del tipo `DATE` y podría tener como valor '1978-12-25'.
- La columna `inicio_partido` podría ser del tipo `TIME` y podría tener como valor '17:15:00.000000'.
- La columna `entrada_trabajo` podría ser de tipo `TIMESTAMP` y podría tener como valor '1998-7-8 9:30:05'.

Creación, modificación y borrado de dominios

Además de los dominios dados por el tipo de datos predefinidos, el SQL92 nos ofrece la posibilidad de trabajar con dominios definidos por el usuario.

Dominios definidos por el usuario

Aunque el SQL92 nos ofrece la sentencia `CREATE DOMAIN`, hay pocos sistemas relacionales comerciales que nos permitan utilizarla.

Para crear un dominio es necesario utilizar la sentencia **`CREATE DOMAIN`**:

```
CREATE DOMAIN nombre dominio [AS] tipos_datos
             [def_defecto] [restricciones_dominio];
```

donde **`restricciones_dominio`** tiene el siguiente formato:

```
[CONSTRAINT nombre_restricción] CHECK (condiciones)
```

Explicaremos la construcción de condiciones más adelante, en el subapartado 2.5 cuando hablemos de cómo se hacen consultas a una base de datos. Veremos `def_defecto` en el subapartado 1.2.3 de esta unidad.

Creación de un dominio en BDUOC

Si quisiéramos definir un dominio para las ciudades donde se encuentran los departamentos de la empresa BDUOC, haríamos:

```
CREATE DOMAIN dom_ciudades AS CHAR (20)
CONSTRAINT ciudades_validas
CHECK (VALUE IN ('Barcelona', 'Tarragona', 'Lleida', 'Girona'));
```

De este modo, cuando definimos la columna `ciudades` dentro de la tabla departamentos no se tendrá que decir que es de tipo `CHAR (20)`, sino de tipo `dom_ciudades`. Esto nos debería asegurar, según el modelo relacional, que sólo haremos operaciones sobre la columna `ciudades` con otras columnas que tengan este mismo dominio definido por el usuario; sin embargo, el SQL92 no nos ofrece herramientas para asegurar que las comparaciones que hacemos sean entre los mismos dominios definidos por el usuario.

Por ejemplo, si tenemos una columna con los nombres de los empleados definida sobre el tipo de datos `CHAR (20)`, el SQL nos permite compararla con la columna `ciudades`, aunque semánticamente no tenga sentido. En cambio, según el modelo relacional, esta comparación no se debería haber permitido.

Para borrar un dominio definido por el usuario es preciso utilizar la sentencia **`DROP DOMAIN`**, que tiene este formato:

```
DROP DOMAIN nombre_dominio {RESTRICT|CASCADE};
```

En este caso, tenemos que:

- La opción de borrado de dominios **RESTRICT** hace que el dominio sólo se pueda borrar si no se utiliza en ningún sitio.

- La opción **CASCADE** borra el dominio aunque esté referenciado, y pone el tipo de datos del dominio allí donde se utilizaba.

Borrar un dominio de BDUOC

Si quisiéramos borrar el dominio que hemos creado antes para las ciudades donde se encuentran los departamentos de la empresa BDUOC, haríamos:

```
DROP DOMAIN dom_ciudades RESTRICT;
```

En este caso nos deberíamos asegurar de que ninguna columna está definida sobre `dom_ciu-dades` antes de borrar el dominio.

Para modificar un dominio semántico es necesario utilizar la sentencia **ALTER DOMAIN**. Veamos su formato:

```
ALTER DOMAIN nombre_dominio {acción_modificar_dominio|
                             acción_modif_restricción_dominio};
```

Donde tenemos lo siguiente:

- **acción_modificar_dominio** puede ser:

```
{SET def_defecto|DROP DEFAULT}
```

- **acción_modif_restricción_dominio** puede ser:

```
{ADD restricciones_dominio|DROP CONSTRAINT nombre_restricción}
```

Modificar un dominio en BDUOC

Si quisiéramos añadir una nueva ciudad (Mataró) al dominio que hemos creado antes para las ciudades donde se encuentran los departamentos de la empresa BDUOC, haríamos:

```
ALTER DOMAIN dom_ciudades DROP CONSTRAINT ciudades_validas;
```

Con esto hemos eliminado la restricción de dominio antigua. Y ahora tenemos que introducir la nueva restricción:

```
ALTER_DOMAIN dom_ciudades ADD CONSTRAINT ciudades_validas
CHECK (VALUE IN ('Barcelona', 'Tarragona', 'Lleida', 'Girona', 'Mataro'));
```

Definiciones por defecto

Ya hemos visto en otros módulos la importancia de los valores nulos y su inevitable aparición como valores de las bases de datos.

La opción **`def_defecto`** nos permite especificar qué nomenclatura queremos dar a nuestros valores por omisión.

Por ejemplo, para un empleado que todavía no se ha decidido cuánto ganará, podemos elegir que, de momento, tenga un sueldo de 0 euros (`DEFAULT 0.0`), o bien que tenga un sueldo con un valor nulo (`DEFAULT NULL`).

Sin embargo, hay que tener en cuenta que si elegimos la opción **`DEFAULT NULL`**, la columna para la que daremos la definición por defecto de valor nulo debería admitir valores nulos.

La opción **`DEFAULT`** tiene el siguiente formato:

`DEFAULT` `(literal|función|`**`NULL`**`)`

La posibilidad más utilizada y la opción por defecto, si no especificamos nada, es la palabra reservada **`NULL`**. Sin embargo, también podemos definir nuestro propio literal, o bien recurrir a una de las funciones que aparecen en la tabla siguiente:

Función	Descripción
`{USER|CURRENT_USER}`	Identificador del usuario actual
`SESSION_USER`	Identificador del usuario de esta sesión
`SYSTEM_USER`	Identificador del usuario del sistema operativo
`CURRENT_DATE`	Fecha actual
`CURRENT_TIME`	Hora actual
`CURRENT_TIMESTAMP`	Fecha y hora actuales

Restricciones de columna

En cada una de las columnas de la tabla, una vez les hemos dado un nombre y hemos definido su dominio, podemos imponer ciertas restricciones que siempre se tendrán que cumplir. Las restricciones que se pueden dar son las que aparecen en la tabla que tenemos a continuación:

Restricciones de columna	
Restricción	Descripción
`NOT NULL`	La columna no puede tener valores nulos.
`UNIQUE`	La columna no puede tener valores repetidos. Es una clave alternativa.
`PRIMARY KEY`	La columna no puede tener valores repetidos ni nulos. Es la clave primaria.
`REFERENCES tabla [(columna)]`	La columna es la clave foránea de la columna de la tabla especificada.
`CHECK (condiciones)`	La columna debe cumplir las condiciones especificadas.

Restricciones de tabla

Una vez hemos dado un nombre, hemos definido una tabla y hemos impuesto ciertas restricciones para cada una de las columnas, podemos aplicar restricciones sobre toda la tabla, que siempre se deberán cumplir. Las restricciones que se pueden dar son las siguientes:

Restricciones de tabla	
Restricción	Descripción
`UNIQUE (columna [, columna...])`	El conjunto de las columnas especificadas no puede tener valores repetidos. Es una clave alternativa.

Restricciones de tabla	
Restricción	Descripción
`PRIMARY KEY (columna [, columna...])`	El conjunto de las columnas especificadas no puede tener valores nulos ni repetidos. Es una clave primaria.
`FOREIGN KEY (columna [, columna...]) REFERENCES tabla [(columna2 [, columna2...])]`	El conjunto de las columnas especificadas es una clave foránea que referencia la clave primaria formada por el conjunto de las columnas2 de la tabla dada. Si las columnas y las columnas2 se denominan exactamente igual, entonces no sería necesario poner columnas2.
`CHECK (condiciones)`	La tabla debe cumplir las condiciones especificadas.

Modificación y borrado de claves primarias con claves foráneas que hacen referencia a éstas

En otra unidad de este curso hemos visto tres políticas aplicables a los casos de borrado y modificación de filas que tienen una clave primaria referenciada por claves foráneas. Estas políticas eran la restricción, la actualización en cascada y la anulación.

Para recordar las políticas que se pueden aplicar a los casos de borrado y modificación de las filas, consultad los subapartados 4.3.1, 4.3.2 y 4.3.3 de la unidad "El modelo relacional y el álgebra relacional" de este curso.

El SQL nos ofrece la posibilidad de especificar, al definir una clave foránea, qué política queremos seguir. Veamos su formato:

```
CREATE TABLE nombre_tabla
            ( definición_columna
            [, definición_columna. . .]
            [, restricciones_tabla]
            );
```

Donde una de las restricciones de tabla era la definición de claves foráneas, que tiene el siguiente formato:

```
FOREIGN KEY clave_secundaria REFERENCES tabla [(clave_primaria)]
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
```

Donde `NO ACTION` corresponde a la política de restricción; `CASCADE`, a la actualización en cascada, y `SET NULL` sería la anulación. `SET DEFAULT` se podría considerar una variante de `SET NULL`, donde en lugar de valores nulos se puede poner el valor especificado por defecto.

Aserciones

Una aserción es una restricción general que hace referencia a una o más columnas de más de una tabla. Para definir una aserción se utiliza la sentencia **`CREATE ASSERTION`**, y tiene el siguiente formato:

```
CREATE ASSERTION nombre_aserción CHECK (condiciones);
```

Crear una aserción en BDUOC

Creamos una aserción sobre la base de datos BDUOC que nos asegure que no hay ningún empleado con un sueldo superior a 80.000 asignado al proyecto SALSA:

```
CREATE ASSERTION restriccion1 CHECK (NOT EXISTS (SELECT *
                              FROM proyectos p, empleados e
                              WHERE p.codigo_proyec =
                              = e.num_proyec and e.sueldo > 8.0E+4
                              and p.nom_proj = 'SALSA') );
```

Para borrar una aserción es necesario utilizar la sentencia **DROP ASSERTION**, que presenta este formato:

```
DROP ASSERTION nombre_aserción;
```

Borrar una aserción en BDUOC

Por ejemplo, para borrar la aserción `restriccion1`, utilizaríamos la sentencia `DROP ASSERTION` de la forma siguiente:

```
DROP ASSERTION restriccion1;
```

Modificación y borrado de tablas

Para modificar una tabla es preciso utilizar la sentencia **ALTER TABLE**. Veamos su formato:

```
ALTER TABLE nombre_tabla {acción_modificar_columna|
                          acción_modif_restricción_tabla};
```

En este caso, tenemos que:

- **acción_modificar_columna** puede ser:

```
{ADD [COLUMN] columna def_columna |
ALTER [COLUMN] columna {SET def_defecto|DROP DEFAULT}|
DROP [COLUMN ] columna {RESTRICT|CASCADE}}
```

- **acción_modif_restricción_tabla** puede ser:

```
{ADD restricción|
DROP CONSTRAINT restricción {RESTRICT|CASCADE}}
```

Si queremos modificar una tabla es que queremos realizar una de las siguientes operaciones: a

1) Añadirle una columna (`ADD columna`).

2) Modificar las definiciones por defecto de la columna (`ALTER columna`).

3) Borrar la columna (`DROP columna`).

4) Añadir alguna nueva restricción de tabla (`ADD restricción`).

5) Borrar alguna restricción de tabla (`DROPCONSTRAINT restricción`).

Para borrar una tabla es preciso utilizar la sentencia **`DROP TABLE`**:

```
DROP TABLE nombre_tabla {RESTRICT|CASCADE};
```

En este caso tenemos que:

- Si utilizamos la opción **`RESTRICT`**, la tabla no se borrará si está referenciada, por ejemplo, por alguna vista.

- Si usamos la opción **`CASCADE`**, todo lo que referencie a la tabla se borrará con ésta.

Creación y borrado de vistas

Como hemos observado, la arquitectura ANSI/SPARC distingue tres niveles, que se describen en el esquema conceptual, el esquema interno y los esquemas externos. Hasta ahora, mientras creábamos las tablas de la base de datos, íbamos describiendo el esquema conceptual. Para describir los diferentes esquemas externos utilizamos el concepto de vista del SQL.

Los tres niveles de la arquitectura ANSI/SPARC se han estudiado en el subapartado 4.1 de la unidad “Introducción a las bases de datos” de este curso.

Para crear una vista es necesario utilizar la sentencia **`CREATE VIEW`**. Veamos su formato:

```
CREATE VIEW nombre_vista [(lista_columnas)] AS (consulta)
          [WITH CHECK OPTION];
```

Lo primero que tenemos que hacer para crear una vista es decidir qué nombre le queremos poner (`nombre_vista`). Si queremos cambiar el nombre de las columnas, o bien poner nombre a alguna que en principio no tenía, lo podemos hacer en `lista_columnas`. Y ya sólo nos quedará definir la consulta que formará nuestra vista.

Por lo que respecta a la construcción de consultas, consultad el subapartado 2.5 de esta unidad didáctica.

Las vistas no existen realmente como un conjunto de valores almacenados en la base de datos, sino que son tablas ficticias, denominadas *derivadas* (no materializadas). Se construyen a partir de tablas reales (materializadas) almacenadas en la base de datos, y conocidas con el nombre de *tablas básicas* (o tablas de base). La no-existencia real de las vistas hace que puedan ser actualizables o no. a

Creación de una vista en BDUOC

Creamos una vista sobre la base de datos BDUOC que nos dé para cada cliente el número de proyectos que tiene encargados el cliente en cuestión.

```
CREATE VIEW proyectos_por_cliente (codigo_cli, numero_proyectos) AS
(SELECT c.codigo_cli, COUNT(*)
 FROM proyectos p, clientes c
 WHERE p.codigo_cliente = c.codigo_cli
 GROUP BY c.codigo_cli);
```

Si tuviésemos las siguientes extensiones:

- Tabla `clientes`:

clientes					
codigo_cli	**nombre_cli**	**nif**	**direccion**	**ciudad**	**telefono**
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67

- Tabla `proyectos`:

proyectos						
codigo_proyec	**nombre_proyec**	**precio**	**fecha_inicio**	**fecha_prev_fin**	**fecha_fin**	**codigo_cliente**
1	GESCOM	1,0E+6	1-1-98	1-1-99	NULL	10
2	PESCI	2,0E+6	1-10-96	31-3-98	1-5-98	10
proyectos						
codigo_proyec	**nombre_proyec**	**precio**	**fecha_inicio**	**fecha_prev_fin**	**fecha_fin**	**codigo_cliente**
3	SALSA	1,0E+6	10-2-98	1-2-99	NULL	20
4	TINELL	4,0E+6	1-1-97	1-12-99	NULL	30

Y mirásemos la extensión de la vista `proyectos_por_clientes`, veríamos lo que encontramos en el margen.

En las vistas, además de hacer consultas, podemos insertar, modificar y borrar filas.

proyectos_por_clientes	
codigo_cli	**numero_proyectos**
10	2
20	1
30	1

Actualización de vistas en BDUOC

Si alguien insertase en la vista `proyectos_por_cliente`, los valores para un nuevo cliente 60 con tres proyectos encargados, encontraríamos que estos tres proyectos tendrían que figurar realmente en la tabla `proyectos` y, por lo tanto, el SGBD los debería insertar con la información que tenemos, que es prácticamente inexistente. Veamos gráficamente cómo quedarían las tablas después de esta hipotética actualización, que no llegaremos a hacer nunca, ya que iría en contra de la teoría del modelo relacional:

- Tabla `clientes`

clientes					
codigo_cli	**nombre_cli**	**nif**	**direccion**	**ciudad**	**telefono**
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67
60	NULL	NULL	NULL	NULL	NULL

- Tabla proyectos:

proyectos						
codigo_proyec	**nombre_proyec**	**precio**	**fecha_inicio**	**fecha_prev_fin**	**fecha_fin**	**codigo_cliente**
1	GESCOM	1,0E+6	1-1-98	1-1-99	NULL	10
2	PESCI	2,0E+6	1-10-96	31-3-98	1-5-98	10
3	SALSA	1,0E+6	10-2-98	1-2-99	NULL	20
NULL	NULL	NULL	NULL	NULL	NULL	60

NULL	NULL	NULL	NULL	NULL	NULL	60
NULL	NULL	NULL	NULL	NULL	NULL	60

El SGBD no puede actualizar la tabla básica `clientes` si sólo sabe la clave primaria, y todavía menos la tabla básica `proyectos` sin la clave primaria; por lo tanto, esta vista no sería actualizable.

En cambio, si definimos una vista para saber los clientes que tenemos en Barcelona o en Girona, haríamos:

```
CREATE VIEW clientes_Barcelona_Girona AS
(SELECT *
 FROM clientes
 WHERE ciudad IN (‘Barcelona’, ‘Girona’))
WHITH CHECK OPTION;
```

Si queremos asegurarnos de que se cumpla la condición de la cláusula `WHERE`, debemos poner la opción **`WHITH CHECK OPTION`**. Si no lo hiciésemos, podría ocurrir que alguien incluyese en la vista `clientes_Barcelona_Girona` a un cliente nuevo con el código 70, de nombre JMB, con el NIF 36.788.224-C, la dirección en NULL, la ciudad Lleida y el teléfono NULL.

Si consultásemos la extensión de la vista `clientes_Barcelona_Girona`, veríamos:

clientes_Barcelona_Girona					
codigo_cli	**nombre_cli**	**nif**	**direccion**	**ciudad**	**telefono**
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21

Esta vista sí podría ser actualizable. Podríamos insertar un nuevo cliente con código 50, de nombre CEA, con el NIF 38.226.777-D, con la dirección París 44, la ciudad Barcelona y el teléfono 93.422.60.77. Después de esta actualización, en la tabla básica `clientes` encontraríamos, efectivamente:

clientes					
codigo_cli	**nombre_cli**	**nif**	**direccion**	**ciudad**	**telefono**
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67
50	CEA	38.226.777-D	París, 44	Barcelona	93.442.60.77

Para borrar una vista es preciso utilizar la sentencia **`DROP VIEW`**, que presenta el formato:

```
DROP VIEW nombre_vista (RESTRICT|CASCADE);
```

Si utilizamos la opción **RESTRICT**, la vista no se borrará si está referenciada, por ejemplo, por otra vista. En cambio, si ponemos la opción **CASCADE**, todo lo que referencie a la vista se borrará con ésta.

Borrar una vista en BDUOC

Para borrar la vista `clientes_Barcelona_Girona`, haríamos lo siguiente:

```
DROP VIEW clientes_Barcelona_Girona RESTRICT;
```

Definición de la base de datos relacional BDUOC

Veamos cómo se crearía la base de datos BDUOC, utilizando, por ejemplo, un SGBD relacional que disponga de la sentencia `CREATE DATABASE`:

```
CREATE DATABASE bduoc;
CREATE TABLE clientes
   (codigo_cli INTEGER,
   nombre_cli CHAR(30) NOT NULL,
   nif CHAR (12),
   direccion CHAR (30),
   ciudad CHAR (20),
   telefono CHAR (12),
   PRIMARY KEY (codigo_cli),
   UNIQUE(nif)
 );

CREATE TABLE departamentos
   (nombre_dep CHAR(20) PRIMARY KEY,*
   ciudad_dep CHAR(20),
   telefono INTEGER DEFAULT NULL,
   PRIMARY KEY (nombre_dep, ciudad_dep)
 );

CREATE TABLE proyectos
   (codigo_proyec INTEGER,
   nombre_proyec CHAR(20),
   precio REAL,
   fecha_inicio DATE,
   fecha_prev_fin DATE,
   fecha_fin DATE DEFAULT NULL,
   codigo_cliente INTEGER,
   PRIMARY KEY (codigo_proyec),
   FOREIGN KEY codigo_cliente REFERENCES clientes (codigo_cli),
   CHECK (fecha_inicio < fecha_prev_fin),
   CHECK (fecha_inicio < fecha_fin)
 );

CREATE TABLE empleados
```

```
  (codigo_empl INTEGER,
  nombre_empl CHAR (20),
  apellido_empl CHAR(20),
  sueldo REAL CHECK (sueldo > 7000),
  nombre_dep CHAR(20)
  ciudad_dep CHAR(20),
  num_proyec INTEGER,
  PRIMARY KEY (codigo_empl),
  FOREIGN KEY (nombre_dep, ciudad_dep) REFERENCES
  departamentos (nombre_dep, ciudad_dep),
  FOREIGN KEY (num_proyec) REFERENCES proyectos (codigo_proyec)
 );

COMMIT;
```

Orden de creación

Antes de crear una tabla con una o más claves foráneas, se deben haber creado las tablas que tienen como clave primaria las referenciadas por las foráneas.

* Tenemos que elegir restricción de tabla porque la clave primaria está compuesta por más de un atributo.

La sentencia `COMMIT` se explica en el subapartado 3.1 de esta unidad didáctica.

Al crear una tabla vemos que muchas restricciones se pueden imponer de dos formas: como restricciones de columna o como restricciones de tabla. Por ejemplo, cuando queremos decir cuál es la clave primaria de una tabla, tenemos las dos posibilidades. Esto se debe a la flexibilidad del SQL:

- En el caso de que la restricción haga referencia a un solo atributo, podemos elegir la posibilidad que más nos guste.

- En el caso de la tabla `departamentos`, tenemos que elegir por fuerza la opción de restricciones de tabla, porque la clave primaria está compuesta por más de un atributo.

En general, lo pondremos todo como restricciones de tabla, excepto `NOT NULL` y `CHECK` cuando haga referencia a una sola

columna. a

Sentencias de manipulación

Una vez creada la base de datos con sus tablas, debemos poder insertar, modificar y borrar los valores de las filas de las tablas. Para poder hacer esto, el SQL92 nos ofrece las siguientes sentencias: **INSERT** para insertar, **UPDATE** para modificar y **DELETE** para borrar. Una vez hemos insertado valores en nuestras tablas, tenemos que poder consultarlos. La sentencia para hacer consultas a una base de datos con el SQL92 es **SELECT FROM**. Veamos a continuación estas sentencias. a

Inserción de filas en una tabla

Antes de poder consultar los datos de una base de datos, es preciso introducirlos con la sentencia **INSER TINTO VALUES**, que tiene el formato:

```
INSERT INTO nombre_tabla [(columnas)]
{VALUES ({v1|DEFAULT|NULL}, ..., {vn/DEFAULT/NULL})|<consulta>};
```

Inserción de múltiples filas

Para insertar más de una fila con una sola sentencia, tenemos que obtener los valores como resultado de una consulta realizada en una
o más tablas.

Los valores `v1`, `v2`, ..., `vn` se deben corresponder exactamente con las columnas que hemos dicho que tendríamos con el `CREATE TABLE` y deben estar en el mismo orden, a menos que las volvamos a poner a continuación del nombre de la tabla. En este último caso, los valores se deben disponer de forma coherente con el nuevo orden que hemos impuesto. Podría darse el caso de que quisiéramos que algunos valores para insertar fuesen valores por omisión, definidos previamente con la opción `DEFAULT`. Entonces pondríamos la palabra reservada `DEFAULT`. Si se trata de introducir valores nulos, también podemos utilizar la palabra reservada `NULL`.

Inserción de una fila en BDUOC

La forma de insertar a un cliente en la tabla `clientes` de la base de datos de BDUOC es:

```
INSERT INTO clientes
VALUES (10, 'ECIGSA', '37.248.573-C', 'ARAGON 242', 'Barcelona', DEFAULT);
```

o bien:

```
INSERT INTO clientes(nif, nombre_cli, codigo_cli, telefono, direccion,
ciudad)
VALUES ('37.248.573-C', 'ECIGSA', 10, DEFAULT, 'ARAGON 242', 'Barcelona');
```

Borrado de filas de una tabla

Para borrar valores de algunas filas de una tabla podemos utilizar la sentencia **DELETE FROM WHERE**. Su formato es el siguiente:

```
DELETE FROM nombre_tabla
[WHERE condiciones];
```

En cambio, si lo que quisiéramos conseguir es borrar todas las filas de una tabla, entonces sólo tendríamos que poner la sentencia **DELETE FROM**, sin WHERE.

Borrar todas las filas de una tabla en BDUOC

Podemos dejar la tabla `proyectos` sin ninguna fila:

```
DELETE FROM proyectos;
```

En nuestra base de datos, borrar los proyectos del cliente 2 se haría de la forma que mostramos a continuación:

```
DELETE FROM proyectos
WHERE codigo_cliente = 2;
```

Borrado de múltiples filas

Notemos que el cliente con el código 2 podría tener más de un proyecto contratado y, por lo tanto, se borraría más de una fila con una sola sentencia.

Modificación de filas de una tabla

Si quisiéramos modificar los valores de algunas filas de una tabla, tendríamos que utilizar la sentencia **UPDATE SET WHERE**. A continuación presentamos su formato:

```
UPDATE nombre_tabla
SET columna = {expresión|DEFAULT|NULL}
    [, columna = {expr|DEFAULT|NULL} ...]
WHERE condiciones;
```

Modificación de múltiples filas

Notemos que el proyecto número 2 podría tener a más de un empleado asignado y, por lo tanto, se modificaría la columna `sueldo`, de más de una fila

con una sola sentencia.

Modificación de los valores de algunas filas en BDUOC

Supongamos que queremos incrementar el sueldo de todos los empleados del proyecto 2 en 1.000 euros. La modificación a ejecutar sería:

```
UPDATE empleados
SET sueldo = sueldo + 1000
WHERE num_proyec = 2;
```

Introducción de filas en la base de datos relacional BDUOC

Antes de empezar a hacer consultas a la base de datos BDUOC, habremos introducido unas cuantas filas en sus tablas con la sentencia `INSERT INTO`. De esta forma, podremos ver reflejado el resultado de las consultas que iremos haciendo, a partir de este momento, sobre cada extensión; esto lo podemos observar en las tablas correspondientes a cada extensión, que presentamos a continuación:

- Tabla `departamentos`:

departamentos		
nombre_dep	**ciudad_dep**	**telefono**
DIR	Barcelona	93.422.60.70
DIR	Girona	972.23.89.70
DIS	Lleida	973.23.50.40
DIS	Barcelona	93.224.85.23
PROG	Tarragona	977.33.38.52
PROG	Girona	972.23.50.91

- Tabla `clientes`:

clientes					
codigo_cli	nombre_cli	nif	direccion	ciudad	telefono
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67
40	JGM	38.782.345-B	Rosellón 44	Tarragona	977.33.71.43

- Tabla `empleados`:

empleados						
codigo_empleado	nombre_empl	apellido_empl	sueldo	nombre_dep	ciudad_dep	num_proyec
1	María	Puig	1,0E+5	DIR	Girona	1
2	Pedro	Mas	9,0E+4	DIR	Barcelona	4
3	Ana	Ros	7,0E+4	DIS	Lleida	3
4	Jorge	Roca	7,0E+4	DIS	Barcelona	4
5	Clara	Blanc	4,0E+4	PROG	Tarragona	1
6	Laura	Tort	3,0E+4	PROG	Tarragona	3
7	Rogelio	Salt	4,0E+4	NULL	NULL	4
8	Sergio	Grau	3,0E+4	PROG	Tarragona	NULL

- Tabla `proyectos`:

proyectos						
codigo_proyec	nombre_proyec	precio	fecha_inicio	fecha_prev_fin	fecha_fin	codigo_cliente
1	GESCOM	1,0E+6	1-1-98	1-1-99	NULL	10
2	PESCI	2,0E+6	1-10-96	31-3-98	1-5-98	10

3	SALSA	1,0E+6	10-2-98	1-2-99	NULL	20
4	TINELL	4,0E+6	1-1-97	1-12-99	NULL	30

Consultas a una base de datos relacional

Para hacer consultas sobre una tabla con el SQL es preciso utilizar la sentencia **SELECT FROM**, que tiene el siguiente formato:

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

La opción **AS** nos permite renombrar las columnas que queremos seleccionar o las tablas que queremos consultar que en este caso, es sólo una. Dicho de otro modo, nos permite la definición de alias. Fijémonos en que la palabra clave AS es opcional, y es bastante habitual poner sólo un espacio en blanco en lugar de toda la palabra.

Consultas a BDUOC

A continuación presentamos un ejemplo de consulta a la base de datos BDUOC para conocer todos los datos que aparece en la tabla `clientes`:

```
SELECT *
FROM clientes;
```

El * después de **SELECT** indica que queremos ver todos los atributos que aparecen en la tabla.

La respuesta a esta consulta sería:

codigo_cli	nombre_cli	nif	direccion	ciudad	telefono
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67
40	JGM	38.782.345-B	Rosellón 44	Tarragona	977.33.71.43

Si hubiésemos querido ver sólo el código, el nombre, la dirección y la ciudad, habríamos hecho:

```
SELECT codigo_cli, nombre_cli, direccion, ciudad
FROM clientes;
```

Y habríamos obtenido la respuesta siguiente:

codigo_cli	nombre_cli	direccion	ciudad
10	ECIGSA	Aragón 11	Barcelona
20	CME	Valencia 22	Girona
30	ACME	Mallorca 33	Lleida
40	JGM	Rosellón 44	Tarragona

Con la sentencia `SELECT FROM` podemos seleccionar columnas de una tabla, pero para seleccionar filas de una tabla es preciso añadirle la cláusula `WHERE`. El formato es:

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE condiciones;
```

La cláusula WHERE nos permite obtener las filas que cumplen la condición especificada en la consulta.

Consultas a BDUOC seleccionando filas

Veamos un ejemplo en el que pedimos "los códigos de los empleados que trabajan en el proyecto número 4":

```
SELECT codigo_empl
FROM empleados
WHERE num_proyec = 4;
```

codigo_empl
2
4
7

La respuesta a esta consulta sería la que podéis ver en el margen.

Para definir las condiciones en la cláusula WHERE, podemos utilizar alguno de los operadores de los que dispone el SQL, que son los siguientes: a

Operadores de comparación	
=	Igual

Operadores lógicos	
NOT	Para la negación de condiciones

<	Menor
>	Mayor
<=	Menor o igual
>=	Mayor o igual
< >	Diferente

AND	Para la conjunción de condiciones
OR	Para la disyunción de condiciones

Si queremos que en una consulta nos aparezcan las filas resultantes sin repeticiones, es preciso poner la palabra clave **DISTINCT** inmediatamente después de SELECT. También podríamos explicitar que lo queremos todo, incluso con repeticiones, poniendo **ALL** (opción por defecto) en lugar de DISTINCT. El formato de DISTINCT es:

```
SELECT DISTINCT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
[WHERE condiciones];
```

Consulta a BDUOC seleccionando filas sin repeticiones

Por ejemplo, si quisiéramos ver qué sueldos se están pagando en nuestra empresa, podríamos hacer:

```
SELECT DISTINCT sueldo
FROM empleados;
```

sueldo
3,0E+4
4,0E+4
7,0E+4
9,0E+4
1,0E+5

La respuesta a esta consulta, sin repeticiones, sería la que aparece en el margen.

Funciones de agregración

El SQL nos ofrece las siguientes funciones de agregación para efectuar varias operaciones sobre los datos de una base de datos:

Funciones de agregación	
Función	Descripción
COUNT	Nos da el número total de filas seleccionadas

SUM	Suma los valores de una columna
MIN	Nos da el valor mínimo de una columna
MAX	Nos da el valor máximo de una columna
AVG	Calcula el valor medio de una columna

En general, las funciones de agregación se aplican a una columna, excepto la función de agregación COUNT, que normalmente se aplica a todas las columnas de la tabla o tablas seleccionadas. Por lo tanto, COUNT (*) contará todas las filas de la tabla o las tablas que cumplan las condiciones. Si se utilizase COUNT(distinct columna), sólo contaría los valores que no fuesen nulos ni repetidos, y si se utilizase COUNT (columna), sólo contaría los valores que no fuesen nulos.

Ejemplo de utilización de la función COUNT (*)

Veamos un ejemplo de uso de la función COUNT, que aparece en la cláusula SELECT, para hacer la consulta "¿Cuántos departamentos están ubicados en la ciudad de Lleida?":

```
SELECT COUNT(*) AS numero_dep
FROM departamentos
WHERE ciudad_dep = 'Lleida';
```

La respuesta a esta consulta sería la que aparece reflejada en la tabla que encontraréis en el margen.

numero_dep
1

Veremos ejemplos de las demás funciones de agregación en los siguientes apartados. a

Subconsultas

Una subconsulta es una consulta incluida dentro de una cláusula WHERE o HAVING de otra consulta. En ocasiones, para expresar ciertas condiciones no hay más remedio que obtener el valor que buscamos como resultado de una consulta.

Veremos la cláusula HAVING en el subapartado 2.5.5 de esta unidad didáctica.

Subconsulta en BDUOC

Si quisiéramos saber los códigos y los nombres de los proyectos de precio más elevado, en primer lugar tendríamos que encontrar los proyectos que tienen el precio más elevado. Lo haríamos de la forma siguiente:

```
SELECT codigo_proyec, nombre_proyec
```

```
FROM proyectos
WHERE precio = (SELECT MAX(precio)
                FROM proyectos);
```

codigo_proyec	nombre_proyec
4	TINELL

El resultado de la consulta anterior sería lo que puede verse al margen.

Los proyectos de precio más bajo

Si en lugar de los códigos y los nombres de proyectos de precio más alto hubiésemos querido saber los de precio más bajo, habríamos aplicado la función de agregación `MIN`.

Otros predicados

1) Predicado **`BETWEEN`**

Para expresar una condición que quiere encontrar un valor entre unos límites concretos, podemos utilizar `BETWEEN`:

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna BETWEEN límite1 AND límite2;
```

Ejemplo de uso del predicado `BETWEEN`

Un ejemplo en el que se pide "Los códigos de los empleados que ganan entre 20.000 y 50.000 euros anuales" sería:

```
SELECT codigo_empl
FROM empleados
WHERE sueldo BETWEEN 2.0E+4 and 5.0E+4;
```

codigo_empl
5
6
7
8

La respuesta a esta consulta sería la que se ve en el margen.

2) Predicado **IN**

Para comprobar si un valor coincide con los elementos de una lista utilizaremos IN, y para ver si no coincide, `NOT IN`:

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna [NOT] IN (valor1, ..., valorN);
```

Ejemplo de uso del predicado IN

"Queremos saber el nombre de todos los departamentos que se encuentran en las ciudades de Lleida o Tarragona":

```
SELECT nombre_dep, ciudad_dep
FROM departamentos
WHERE ciudad_dep IN ('Lleida', 'Tarragona');
```

nombre_dep	ciudad_dep
DIS	Lleida
PROG	Tarragona

La respuesta sería la que aparece en el margen.

3) Predicado **LIKE**

Para comprobar si una columna de tipo carácter cumple alguna propiedad determinada, podemos usar `LIKE`:

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna LIKE característica;
```

Los patrones del SQL92 para expresar características son los siguientes: a

Otros patrones

Aunque _ y % son los caracteres elegidos por el estándar, cada sistema relacional comercial ofrece diversas variantes.

a) Pondremos un carácter _ para cada carácter individual que queramos considerar.

b) Pondremos un carácter % para expresar una secuencia de caracteres, que puede no estar formada por ninguno.

Ejemplo de uso del predicado `LIKE`

A continuación presentamos un ejemplo en el que buscaremos los nombres de los empleados que empiezan por J, y otro ejemplo en el que obtendremos los proyectos que comienzan por S y tienen cinco letras:

a) Nombres de empleados que empiezan por la letra J:

```
SELECT codigo_empl, nombre_empl
FROM empleados
WHERE nombre_empl LIKE 'J%';
```

Atributos añadidos

Aunque la consulta pide sólo los nombres de empleados añadimos el código para poder diferenciar dos empleados con el mismo nombre.

codigo_empl	nombre_empl
4	Jorge

La respuesta a esta consulta sería la que se muestra en el margen.

b) Proyectos que empiezan por S y tienen cinco letras:

```
SELECT codigo_proyec
FROM proyectos
WHERE nombre_proyec LIKE 'S_ _ _ _';
```

codigo_proyec
3

Y la respuesta a esta otra consulta sería la que aparece en el margen.

4) Predicado **`IS NULL`**

Para comprobar si un valor es nulo utilizaremos `IS NULL`, y para averiguar si no lo es, `IS NOT NULL`. El formato es:

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna IS [NOT] NULL;
```

Ejemplo de uso del predicado `IS NULL`

Un ejemplo de uso de este predicado sería "Queremos saber el código y el nombre de todos los empleados que no están asignados a ningún proyecto":

```
SELECT codigo_empl, nombre_empl
FROM empleados
WHERE num_proyec IS NULL;
```

codigo_empl	nombre_empl
8	Sergio

Obtendríamos la respuesta que tenemos al margen.

5) Predicados **ANY/SOME** y **ALL**

Para ver si una columna cumple que todas sus filas (ALL) o algunas de sus filas (ANY/SOME) satisfagan una condición, podemos hacer:

```
SELECT nombre_columnas_a seleccionar
FROM tabla_a_consultar
WHERE columna operador_comparación {ALL|ANY|SOME}subconsulta;
```

Los predicados ANY/SOME

Podemos elegir cualquiera de los dos predicados para pedir que alguna fila satisfaga una condición.

Ejemplo de uso de los predicados ALL y ANY/SOME

a) Veamos un ejemplo de aplicación de ALL para encontrar los códigos y los nombres de los proyectos en los que los sueldos de todos los empleados asignados son menores que el precio del proyecto:

```
SELECT codigo_proyec, nombre_proyec
FROM proyectos
WHERE precio > ALL (SELECT sueldo
                    FROM empleados
                    WHERE codigo_proyec = num_proyec);
```

codigo_proyec	nombre_proyec
1	GESCOM
2	PESCI
3	SALSA
4	TINELL

Fijémonos en la condición de WHERE de la subconsulta, que nos asegura que los sueldos que observamos son los de los empleados asignados al proyecto de la consulta. La respuesta a esta consulta sería la que aparece en el margen.

b) A continuación, presentamos un ejemplo de ANY/SOME para buscar los códigos y los nombres de los proyectos que tienen algún empleado que gana un sueldo más elevado que el precio del proyecto en el que trabaja.

```
SELECT codigo_proyec, nombre_proyec
FROM proyectos
WHERE precio < ANY (SELECT sueldo
```

```
                FROM empleados
                WHERE codigo_proyec = num_proyec);
```

codigo_proyec	nombre_proyec

La respuesta a esta consulta está vacía, como se ve en el margen.

6) Predicado **EXISTS**

Para comprobar si una subconsulta produce alguna fila de resultados, podemos utilizar la sentencia denominada *test de existencia*: EXISTS. Para comprobar si una subconsulta no produce ninguna fila de resultados, podemos utilizar NOT EXISTS.

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE [NOT] EXISTS subconsulta;
```

Ejemplo de uso del predicado EXISTS

Un ejemplo en el que se buscan los códigos y los nombres de los empleados que están asignados a algún proyecto sería:

```
SELECT codigo_empl, nombre_empl
FROM empleados
WHERE EXISTS (SELECT *
              FROM proyectos
              WHERE codigo_proyec = num_proyec);
```

codigo_empl	nombre_empl
1	María
2	Pedro
3	Ana
4	Jorge
5	Clara
6	Laura
7	Rogelio

La respuesta a esta consulta sería la que se muestra en el margen.

Ordenación de los datos obtenidos en respuestas a consultas

Si se desea que, al hacer una consulta, los datos aparezcan en un orden determinado, es preciso utilizar la cláusula **ORDER BY**

en la sentencia SELECT, que presenta el siguiente formato:

```
SELECT nombre_columnas_a seleccionar
FROM tabla_a_consultar
[WHERE condiciones]
ORDER BY columna_según_la_cual_se_quiere_ordenar [DESC]
        [, col_ordenación [DESC]...];
```

Consulta a BDUOC con respuesta ordenada

Imaginemos que queremos consultar los nombres de los empleados ordenados según el sueldo que ganan, y si ganan el mismo sueldo, ordenados alfabéticamente por el nombre:

```
SELECT codigo_empl, nombre_empl, apellido_empl, sueldo
FROM empleados
ORDER BY sueldo, nombre_empl;
```

Esta consulta daría la respuesta siguiente:

codigo_empl	nombre_empl	apellido_empl	sueldo
6	Laura	Tort	3,0E+4
8	Sergio	Grau	3,0E+4
5	Clara	Blanc	4,0E+4
7	Rogelio	Salt	4,0E+4
3	Ana	Ros	7,0E+4
4	Jorge	Roca	7,0E+4
2	Pedro	Mas	9,0E+4
1	María	Puig	1,0E+5

Si no se especifica nada más, se seguirá un orden ascendente, pero si se desea seguir un orden descendente es necesario añadir DESC detrás de cada factor de ordenación expresado en la cláusula ORDER BY:

```
ORDER BY columna_ordenación [DESC] [, columna [DESC] ...];
```

También se puede explicitar un orden ascendente poniendo la palabra clave **ASC** (opción por defecto).

Consultas con agrupación de filas de una tabla

Las cláusulas siguientes, añadidas a la instrucción SELECT FROM, permiten organizar las filas por grupos:

a) La cláusula **GROUP BY** nos sirve para agrupar filas según las columnas que indique esta cláusula.

b) La cláusula **HAVING** especifica condiciones de búsqueda para grupos de filas; lleva a cabo la misma función que antes cumplía la cláusula WHERE para las filas de toda la tabla, pero ahora las condiciones se aplican a los grupos obtenidos.

Presenta el siguiente formato:

```
SELECT nombre_columnas_a seleccionar
FROM tabla_a_consultar
[WHERE condiciones]
GROUP BY columnas_según_las_cuales_se_quiere_agrupar
[HAVING condiciones_por_grupos]
[ORDER BY columna_ordenación [DESC] [, columna [DESC]...]];
```

Notemos que en las sentencias SQL se van añadiendo cláusulas a medida que la dificultad o la exigencia de la consulta lo requiere. a

Consulta con agrupación de filas en BDUOC

Imaginemos que queremos saber el sueldo medio que ganan los empleados de cada departamento:

```
SELECT nombre_dep, ciudad_dep, AVG(sueldo) AS sueldo_medio
FROM empleados
GROUP BY nombre_dep, ciudad_dep;
```

Factores de agrupación

Los factores de agrupación de la cláusula GROUP BY deben ser, como mínimo, las columnas que figuran en SELECT, exceptuando las columnas afectadas por funciones de agregación.

El resultado de esta consulta sería:

nombre_dep	ciudad_dep	sueldo_medio
DIR	Barcelona	9,0E + 4
DIR	Girona	1,0E + 5

DIS	Lleida	7,0E + 4
DIS	Barcelona	7,0E + 4
PROG	Tarragona	3,3E + 4
NULL	NULL	4,0E + 4

Ejemplo de uso de la función de agregación SUM

Veamos un ejemplo de uso de una función de agregación SUM del SQL que aparece en la cláusula HAVING de GROUP BY: "Queremos saber los códigos de los proyectos en los que la suma de los sueldos de los empleados es mayor que 180.000 euros":

```
SELECT num_proyec
FROM empleados
GROUP BY num_proyec
HAVING SUM (sueldo) >1.8E+5;
```

El resultado de esta consulta sería el que se ve al margen.

num_proyec
4

DISTINCT y GROUP BY

En este ejemplo no es necesario poner DISTINCT, a pesar de que la columna num_proyec no es atributo identificador.
Fijémonos en que en la tabla empleados hemos puesto
que todos los proyectos tienen el mismo código juntos en un mismo grupo y no es posible que aparezcan repetidos.

Consultas a más de una tabla

Muchas veces queremos consultar datos de más de una tabla haciendo combinaciones de columnas de tablas diferentes. En el SQL es posible listar más de una tabla que se quiere consultar especificándolo en la cláusula FROM.

1) Combinación

La combinación consigue crear una sola tabla a partir de las tablas especificadas en la cláusula `FROM`, haciendo coincidir los valores de las columnas relacionadas de estas tablas.

Recordad que la misma operación de combinación, pero del álgebra relacional, se ha visto en el subapartado 5.3.3. de la unidad "El modelo relacional y el álgebra relacional" de este curso.

Ejemplo de combinación en BDUOC

A continuación mostramos un ejemplo con la base de datos BDUOC en el que queremos saber el NIF del cliente y el código y el precio del proyecto que desarrollamos para el cliente número 20:

```
SELECT proyectos.codigo_proyecto, proyectos.precio, clientes.nif
FROM clientes, proyectos
WHERE clientes.codigo_cli = proyectos.codigo_cliente AND clientes.
codigo_cli = 20;
```

El resultado sería:

proyectos.codigo_proyecto	proyectos.precio	clientes.nif
3	1,0E+6	38.123.898-E

Si trabajamos con más de una tabla, puede ocurrir que la tabla resultante tenga dos columnas con el mismo nombre. Por ello es obligatorio especificar a qué tabla corresponden las columnas a las que nos estamos refiriendo, denominando la tabla a la que pertenecen antes de ponerlas (por ejemplo, `clientes.codigo_cli`). Para simplificarlo, se utilizan los alias que, en este caso, se definen en la cláusula `FROM`.

Ejemplo de alias en BDUOC

`c` podría ser el alias de la tabla `clientes`. De este modo, para indicar a qué tabla pertenece `codigo_cli`, sólo haría falta poner: `c.codigo_cli`.

Veamos cómo quedaría la consulta anterior expresada mediante alias, aunque en este ejemplo no serían necesarios, porque todas las columnas de las dos tablas tienen nombres diferentes. Pediremos, además, las columnas `c.codigo_cli` y `p.codigo_cliente`.

```
SELECT p.codigo_proyecto, p.precio, c.nif, p.codigo_cliente, c.codigo_cli
FROM clientes c, proyectos p
WHERE c.codigo_cli = p.codigo_cliente AND c.codigo_cli = 20;
```

Entonces obtendríamos este resultado:

p.codigo_proyec	p.precio	c.nif	p.codigo_client e	c.codigo_cli
3	1,0E+6	38.123.898-E	20	20

Notemos que en `WHERE` necesitamos expresar el vínculo que se establece entre las dos tablas, en este caso `codigo_cli` de `clientes` y `codigo_cliente` de `proyectos`. Expresado en operaciones del álgebra relacional, esto significa que hacemos una combinación en lugar de un

producto cartesiano.

Fijémonos en que, al igual que en álgebra relacional, la operación que acabamos de hacer es una equicombinación (*equi-join*); por lo tanto, nos aparecen dos columnas idénticas: `c.codigo_cli` y `p.codigo_cliente`.

Las operaciones del álgebra relacional se han visto en el apartado 5 de la unidad "El modelo relacional y el álgebra relacional" de este curso.

La forma de expresar la combinación que acabamos de ver pertenece al SQL92 introductorio. Una forma alternativa de realizar la equicombinación anterior, utilizando el SQL92 intermedio o completo, sería la siguiente:

```
SELECT nombre_columnas_a_seleccionar
FROM tabla1 JOIN tabla2
     {ON condiciones|USING (columna [, columna...])}
[WHERE condiciones];
```

Ejemplo anterior con el SQL92 intermedio o completo

El ejemplo que hemos expuesto antes utilizando el SQL92 intermedio o completo sería:

```
SELECT p.codigo_proyecto, p.precio, c.nif, p.codigo_cliente, c.codigo_cli
FROM clientes c JOIN proyectos p ON c.codigo_cli = p.codigo_cliente
WHERE c.codigo_cli = 20;
```

Y obtendríamos el mismo resultado de antes.

La opción `ON`, además de expresar condiciones con la igualdad, en el caso de que las columnas que queramos vincular tengan nombres diferentes, nos ofrece la posibilidad de expresar condiciones con los demás operadores de comparación que no sean el de igualdad. Sería el equivalente a la operación que en álgebra relacional hemos denominado θ-*combinación* (θ-*join*).

Podéis ver la equicombinación y la θ-combinación en el subapartado 5.3.3 de la unidad "El modelo relacional y el álgebra relacional" de este curso.

También podemos utilizar una misma tabla dos veces con alias diferentes, para distinguirlas.

Dos alias para una misma tabla en BDUOC

Si pidiésemos los códigos y los apellidos de los empleados que ganan más que el empleado que tiene por código el número 5, haríamos lo siguiente:

```
SELECT el.codigo_empl, el.apellido_empl
FROM empleados el JOIN empleados e2 ON el.sueldo > e2.sueldo
WHERE e2.codigo_empl = 5;
```

Hemos tomado la tabla `e2` para fijar la fila del empleado con código número 5, de modo que podamos comparar el sueldo de la tabla `e1`, que contiene a todos los empleados, con el sueldo de la tabla `e2`, que contiene sólo al empleado 5.

La respuesta a esta consulta sería:

e1.codigo_empl	e1.apellido_empl
1	Puig
2	Mas
3	Ros
4	Roca

2) Combinación natural

La combinación natural (*natural join*) de dos tablas consiste básicamente, al igual que en el álgebra relacional, en hacer una equicombinación entre columnas del mismo nombre y eliminar las columnas repetidas. La combinación natural, utilizando el SQL92 intermedio o completo, se haría de la forma siguiente:

```
SELECT nombre_columnas_a_seleccionar
FROM tabla1 NATURAL JOIN tabla2
[WHERE condiciones];
```

Combinación natural en BDUOC

Veamos a continuación un ejemplo en el que las columnas para las que se haría la combinación natural se denominan igual en las dos tablas. Ahora queremos saber el código y el nombre de los empleados que están asignados al departamento cuyo teléfono es 977.33.38.52:

```
SELECT codigo_empl, nombre_empl
FROM empleados NATURAL JOIN departamentos
WHERE telefono = '977.333.852';
```

La combinación natural también se podría hacer con la cláusula `USING`, sólo aplicando la palabra reservada `JOIN`:

```
SELECT codigo_empl, nombre_empl
FROM empleados JOIN departamentos USING (nombre_dep, ciudad_dep)
WHERE telefono = '977.333.852';
```

La respuesta que daría sería:

empleados.codigo_empl	empleados.nombre_empl
5	Clara
6	Laura
8	Sergio

3) Combinación interna y externa

Cualquier combinación puede ser interna o externa:

a) La combinación interna (*inner join*) sólo se queda con las filas que tienen valores idénticos en las columnas de las tablas que compara. Esto puede hacer que perdamos alguna fila interesante de alguna de las dos tablas; por ejemplo, porque se

encuentra a NULL en el momento de hacer la combinación. Su formato es el siguiente:

```
SELECT nombre_columnas_a_seleccionar
FROM t1 [NATURAL] [INNER] JOIN t2
       {ON condiciones|
       |USING (columna [,columna...])}
[WHERE condiciones];
```

b) Por ello disponemos de la combinación externa (*outer join*), que nos permite obtener todos los valores de la tabla que hemos puesto a la derecha, los de la tabla que hemos puesto a la izquierda o todos los valores de las dos tablas. Su formato es:

```
SELECT nombre_columnas_a_seleccionar
FROM t1 [NATURAL] [LEFT|RIGHT|FULL] [OUTER] JOIN t2
       {ON condiciones|
       [USING (columna [,columna...])}
[WHERE condiciones];
```

Combinación natural interna en BDUOC

Si quisiéramos vincular con una combinación natural interna las tablas `empleados` y `departamentos` para saber el código y el nombre de todos los empleados y el nombre, la ciudad y el teléfono de todos los departamentos, haríamos:

```
SELECT e.codigo_empl, e.nombre_empl, e.nombre_dep, e.ciudad_dep, d.telefono
FROM empleados e NATURAL JOIN departamentos d;
```

Combinación interna

Aunque en el ejemplo estamos haciendo una combinación natural interna, no es necesario poner la palabra `INNER`, ya que es la opción por defecto.

Y obtendríamos el siguiente resultado:

e.codigo_empl	e.nombre_empl	e.nombre_dep	e.ciudad_dep	d.telefono
1	María	DIR	Girona	972.23.89.70
2	Pedro	DIR	Barcelona	93.422.60.70
3	Ana	DIS	Lleida	973.23.50.40
4	Jorge	DIS	Barcelona	93.224.85.23
5	Clara	PROG	Tarragona	977.33.38.52
6	Laura	PROG	Tarragona	977.33.38.52
8	Sergio	PROG	Tarragona	977.33.38.52

Fijémonos en que en el resultado no aparece el empleado número 7, que no está asignado a ningún departamento, ni el departamento de programación de Girona, que no tiene ningún empleado asignado.

Combinación natural externa a BDUOC

En los ejemplos siguientes veremos cómo varían los resultados que iremos obteniendo según los tipos de combinación externa:

a) Combinación externa izquierda

```
SELECT e.codigo_empl, e.nombre_empl, e.nombre_dep, e.ciudad_dep, d.telefono
FROM empleados e NATURAL LEFT OUTER JOIN departamentos d;
```

El resultado sería el que podemos ver a continuación:

e.codigo_empl	e.nombre_empl	e.nombre_dep	e.ciudad_dep	d.telefono
1	María	DIR	Girona	972.23.89.70
2	Pedro	DIR	Barcelona	93.422.60.70
3	Ana	DIS	Lleida	973.23.50.40
4	Jorge	DIS	Barcelona	93.224.85.23
5	Clara	PROG	Tarragona	977.33.38.52
6	Laura	PROG	Tarragona	977.33.38.52
7	Rogelio	NULL	NULL	NULL
8	Sergio	PROG	Tarragona	977.33.38.52

Combinación externa izquierda

Aquí figura el empleado 7.

b) Combinación externa derecha

```
SELECT e.codigo_empl, e.nombre_empl, e.nombre_dep, e.ciudad_dep, d.telefono
FROM empleados e NATURAL RIGHT OUTER JOIN departamentos d;
```

Obtendríamos este resultado:

e.codigo_empl	e.nombre_empl	e.nombre_dep	e.ciudad_dep	d.telefono
1	María	DIR	Girona	972.23.89.70
2	Pedro	DIR	Barcelona	93.422.60.70
3	Ana	DIS	Lleida	973.23.50.40

4	Jorge	DIS	Barcelona	93.224.85.23
5	Clara	PROG	Tarragona	977.33.38.52
6	Laura	PROG	Tarragona	977.33.38.52
8	Sergio	PROG	Tarragona	977.33.38.52
NULL	NULL	PROG	Girona	972.23.50.91

Combinación externa derecha

Aquí figura el departamento de programación de Girona.

c) Combinación externa plena

```
SELECT e.codigo_empl, e.nombre_empl, e.nombre_dep, e.ciudad_dep, d.telefono
FROM empleados e NATURAL FULL OUTER JOIN departamentos d;
```

Y obtendríamos el siguiente resultado:

e.codigo_empl	e.nombre_empl	e.nombre_dep	e.ciudad_dep	d.telefono
1	María	DIR	Girona	972.23.89.70
2	Pedro	DIR	Barcelona	93.422.60.70
3	Ana	DIS	Lleida	973.23.50.40
4	Jorge	DIS	Barcelona	93.224.85.23
5	Clara	PROG	Tarragona	977.33.38.52
6	Laura	PROG	Tarragona	977.33.38.52
7	Rogelio	NULL	NULL	NULL
8	Sergio	PROG	Tarragona	977.33.38.52
NULL	NULL	PROG	Girona	972.23.50.91

Combinación externa plena

Aquí figura el empleado 7 y el departamento de programación de Girona.

4) Combinaciones con más de dos tablas

Si queremos combinar tres tablas o más con el SQL92 introductorio, sólo tenemos que añadir todas las tablas en el FROM y los vínculos necesarios en el WHERE. Si queremos combinarlas con el SQL92 intermedio o con el completo, tenemos que ir haciendo combinaciones de tablas por pares, y la tabla resultante se convertirá en el primer componente del siguiente par.

Combinaciones con más de dos tablas en BDUOC

Veamos ejemplos de los dos casos, suponiendo que queremos combinar las tablas `empleados`, `proyectos` y `clientes`:

```
SELECT *
FROM empleados, proyectos, clientes
WHERE num_proyec = codigo_proyec AND codigo_cliente = codigo_cli;
```

o bien:

```
SELECT *
FROM (empleados JOIN proyectos ON num_proyec = codigo_proyec)
JOIN clientes ON codigo_cliente = codigo_cli;
```

La unión

La cláusula **UNION** permite unir consultas de dos o más sentencias SELECT FROM. Su formato es:

```
SELECT columnas
FROM tabla
[WHERE condiciones]
UNION [ALL]
SELECT columnas
FROM tabla
[WHERE condiciones];
```

Si ponemos la opción ALL, aparecerán todas las filas obtenidas a causa de la unión. No la pondremos si queremos eliminar las filas repetidas. Lo más importante de la unión es que somos nosotros quienes tenemos que procurar que se efectúe entre columnas definidas sobre dominios compatibles; es decir, que tengan la misma interpretación semántica. Como ya hemos comentado, el SQL92 no nos ofrece herramientas para asegurar la compatibilidad semántica entre columnas. a

Utilización de la unión en BDUOC

Si queremos saber todas las ciudades que hay en nuestra base de datos, podríamos hacer:

```
SELECT ciudad
FROM clientes
UNION
SELECT ciudad_dep
FROM departamentos;
```

ciudad
Barcelona

Girona
Lleida
Tarragona

El resultado de esta consulta sería el que se muestra al margen.

La intersección

Para hacer la intersección entre dos o más sentencias SELECT FROM, podemos utilizar la cláusula **INTERSECT**, cuyo formato es:

```
SELECT columnas
FROM tabla
[WHERE condiciones]
INTERSECT [ALL]
SELECT columnas
FROM tabla
[WHERE condiciones];
```

Si indicamos la opción ALL, aparecerán todas las filas obtenidas a partir de la intersección. No la pondremos si queremos eliminar las filas repetidas. a

Lo más importante de la intersección es que somos nosotros quienes tenemos que vigilar que se haga entre columnas definidas sobre dominios compatibles; es decir, que tengan la misma interpretación semántica.

Utilización de la intersección en BDUOC

Si queremos saber todas las ciudades donde tenemos departamentos en los que podamos encontrar algún cliente, podríamos hacer:

```
SELECT ciudad
FROM clientes
INTERSECT
SELECT ciudad_dep
FROM departamentos;
```

ciudad
Barcelona
Girona
Lleida
Tarragona

El resultado de esta consulta sería el que se muestra al margen.

Sin embargo, la intersección es una de las operaciones del SQL que se puede hacer de más formas diferentes. También

podríamos encontrar la intersección con IN o EXISTS: a

a) Intersección utilizando IN

```
SELECT columnas
FROM tabla
WHERE columna IN (SELECT columna
                  FROM tabla
                  [WHERE condiciones]);
```

b) Intersección utilizando EXISTS

```
SELECT columnas
FROM tabla
WHERE EXISTS (SELECT *
              FROM tabla
              WHERE condiciones);
```

Ejemplo anterior expresado con IN y con EXISTS

El ejemplo que hemos propuesto antes se podría expresar con IN:

```
SELECT c.ciudad
FROM clientes c
WHERE c.ciudad IN (SELECT d.ciudad_dep
                   FROM departamentos d);
```

o también con EXISTS:

```
SELECT c.ciudad
FROM clientes c
WHERE EXISTS (SELECT *
              FROM departamentos d
              WHERE c.ciudad = d.ciudad_dep;
```

La diferencia

Para encontrar la diferencia entre dos o más sentencias SELECT FROM podemos utilizar la cláusula **EXCEPT**, que tiene este formato:

```
SELECT columnas
FROM tabla
[WHERE condiciones]
EXCEPT [ALL]
SELECT columnas
FROM tabla
[WHERE condiciones];
```

Si ponemos la opción ALL aparecerán todas las filas que da la diferencia. No la pondremos si queremos eliminar las filas repetidas.

Lo más importante de la diferencia es que somos nosotros quienes tenemos que vigilar que se haga entre columnas definidas sobre dominios compatibles. a

Utilización de la diferencia en BDUOC

Si queremos saber los clientes que no nos han contratado ningún proyecto, podríamos hacer:

```
SELECT codigo_cli
FROM clientes
EXCEPT
SELECT codigo_cliente
FROM proyectos;
```

codigo_cli
40

El resultado de esta consulta sería el que se ve en el margen.

La diferencia es, junto con la intersección, una de las operaciones del SQL que se puede realizar de más formas diferentes. También podríamos encontrar la diferencia utilizando `NOT IN` o `NOT EXISTS`: a

a) Diferencia utilizando `NOT IN`:

```
SELECT columnas
FROM tabla
WHERE columna NOT IN (SELECT columna
                      FROM tabla
                      [WHERE condiciones]);
```

b) Diferencia utilizando `NOT EXISTS`:

```
SELECT columnas
FROM tabla
WHERE NOT EXISTS (SELECT *
                  FROM tabla
                  WHERE condiciones);
```

Ejemplo anterior expresado con `NOT IN` y con `NOT EXISTS`

El ejemplo que hemos hecho antes se podría expresar con `NOT IN`:

```
SELECT c.codigo_cli
FROM clientes c
WHERE c.codigo_cli NOT IN (SELECT p.codigo_cliente
                           FROM proyectos p);
```

o también con `NOT EXISTS`

```
SELECT c.codigo_cli
```

```
FROM clientes c
WHERE NOT EXISTS (SELECT *
                  FROM proyectos p
                  WHERE c.codigo_cli = p.codigo_cliente);
```

Sentencias de control

Además de definir y manipular una base de datos relacional, es importante que se establezcan mecanismos de control para resolver problemas de concurrencia de usuarios y garantizar la seguridad de los datos. Para la concurrencia de usuarios utilizaremos el concepto de *transacción*, y para la seguridad veremos cómo se puede autorizar y desautorizar a usuarios a acceder a la base de datos.

Las transacciones

Una transacción es una unidad lógica de trabajo. O informalmente, y trabajando con SQL, un conjunto de sentencias que se ejecutan como si fuesen una sola. En general, las sentencias que forman parte de una transacción se interrelacionan entre sí, y no tiene sentido que se ejecute una sin que se ejecuten las demás.

La mayoría de las transacciones se inician de forma implícita al utilizar alguna sentencia que empieza con `CREATE`, `ALTER`, `DROP`, `SET`, `DECLARE`, `GRANT` o `REVOKE`, aunque existe la sentencia SQL para iniciar transacciones, que es la siguiente:

```
SET TRANSACTION {READ ONLY|READ WRITE};
```

Si queremos actualizar la base de datos utilizaremos la opción `READ WRITE`, y si no la queremos actualizar, elegiremos la opción `READ ONLY`.

Sin embargo, en cambio, una transacción siempre debe acabar explícitamente con alguna de las sentencias siguientes:

```
{COMMIT|ROLLBACK} [WORK];
```

La diferencia entre `COMMIT` y `ROLLBACK` es que mientras la sentencia `COMMIT` confirma todos los cambios producidos contra la BD durante la ejecución de la transacción, la sentencia `ROLLBACK` deshace todos los cambios que se hayan producido en la base de datos y la deja como estaba antes del inicio de nuestra transacción.

La palabra reservada `WORK` sólo sirve para aclarar lo que hace la sentencia, y es totalmente opcional.

Ejemplo de transacción

A continuación proponemos un ejemplo de transacción en el que se quiere disminuir el sueldo de los empleados que han trabajado en el proyecto 3 en

1.000 euros. y aumentar el sueldo de los empleados que han trabajado en el proyecto 1 también en 1.000 euros.

```
SET TRANSACTION READ WRITE;
UPDATE empleados SET sueldo = sueldo – 1000 WHERE num_proyec = 3;
UPDATE empleados SET sueldo = sueldo + 1000 WHERE num_proyec = 1;
COMMIT;
```

Las autorizaciones y desautorizaciones

Todos los privilegios sobre la base de datos los tiene su propietario, pero no es el único que accede a ésta. Por este motivo, el SQL nos ofrece sentencias para autorizar y desautorizar a otros usuarios.

1) Autorizaciones

Para autorizar, el SQL dispone de la siguiente sentencia:

```
GRANT privilegios ON objeto TO usuarios
[WITH GRANT OPTION];
```

Donde tenemos que:

a) **`privilegios`** puede ser:

- `ALL PRIVILEGES`: todos los privilegios sobre el objeto especificado.
- `USAGE`: utilización del objeto especificado; en este caso el dominio.
- `SELECT`: consultas.
- `INSERT` [(`columnas`)]: inserciones. Se puede concretar de qué `columnas`.
- `UPDATE` [(`columnas`)]: modificaciones. Se puede concretar de qué `columnas`.
- `DELETE`: borrados.
- `REFERENCES` [(`columna`)]: referencia del objeto en restricciones de integridad. Se puede concretar de qué `columnas`.

b) **`Objeto`** debe ser:

- `DOMAIN`: dominio

- `TABLE`: tabla.

- `Vista`.

c) Usuarios puede ser todo el mundo: `PUBLIC`, o bien una lista de los identificadores de los usuarios que queremos autorizar.

d) La opción **`WITH GRANT OPTION`** permite que el usuario que autoricemos pueda, a su vez, autorizar a otros usuarios a acceder al objeto con los mismos privilegios con los que ha sido autorizado.

2) Desautorizaciones

Para desautorizar, el SQL dispone de la siguiente sentencia:

```
REVOKE [GRANT OPTION FOR] privilegios ON objeto FROM
usuarios [RESTRICT|CASCADE];
```

Donde tenemos que:

a) `privilegios`, `objeto` y `usuarios` son los mismos que para la sentencia `GRANT`.

b) La opción **`GRANT OPTION FOR`** se utilizaría en el caso de que quisiéramos eliminar el derecho a autorizar (`WITH GRANT OPTION`).

c) Si un usuario al que hemos autorizado ha autorizado a su vez a otros, que al mismo tiempo pueden haber hecho más autorizaciones, la opción **`CASCADE`** hace que queden desautorizados todos a la vez.

d) La opción **`RESTRICT`** no nos permite desautorizar a un usuario si éste ha autorizado a otros.

Sublenguajes especializados

Muchas veces querremos acceder a la base de datos desde una aplicación hecha en un lenguaje de programación cualquiera.

Para utilizar el SQL desde un lenguaje de programación, podemos utilizar el SQL hospedado, y para trabajar con éste necesitamos un precompilador que separe las sentencias del lenguaje de programación de las del lenguaje de bases de datos. Una alternativa muy interesante a esta forma de trabajar son las rutinas SQL/CLI.

El objetivo de este apartado no es explicar con detalle ni el SQL hospedado ni, aún menos, las rutinas SQL/CLI. Sólo introduciremos las ideas básicas del funcionamiento de ambos.

SQL hospedado

Para crear y manipular una base de datos relacional necesitamos SQL. Además, si la tarea que queremos hacer requiere el poder de procesamiento de un lenguaje de programación como Java, C, Cobol, Fortran, Pascal, etc., podemos utilizar el SQL hospedado en el lenguaje de programación elegido. De este modo, podemos utilizar las sentencias del SQL dentro de nuestras aplicaciones, poniendo siempre delante la palabra reservada **`EXEC SQL*`**.

* Puede haber pequeñas diferencias dependiendo del lenguaje de programación concreto que estemos considerando.

Para poder compilar la mezcla de llamadas de SQL y sentencias de programación, antes tenemos que utilizar un precompilador. Un precompilador es una herramienta que separa las sentencias del SQL y las sentencias de programación. Allá donde en el programa fuente haya una sentencia de acceso a la base de datos, se debe insertar una llamada a la interfaz del SGBD. El programa fuente resultante de la precompilación ya está únicamente en el lenguaje de programación, preparado para ser compilado, montado y ejecutado.

En la figura que encontraréis en la página siguiente podéis observar este funcionamiento.

Todas las sentencias de definición, manipulación y control que hemos visto para el SQL se pueden utilizar en el SQL hospedado, pero precedidas de la cláusula `EXEC SQL`. Sólo habrá una excepción: cuando el resultado de una sentencia SQL obtenga más de una fila o haga referencia también a más de una, deberemos trabajar con el concepto de cursor.

Un cursor se tiene que haber declarado antes de su utilización (`EXEC SQL DECLARE nombre_cursor CURSOR FOR`). Para utilizarlo, se debe abrir (`EXEC SQL OPEN nombre_cursor`), ir tomando los datos uno a uno, tratarlos (`EXEC SQL FETCH nombre_cursor INTO`), y finalmente, cerrarlo (`EXEC SQL CLOSE nombre_cursor`).

Las SQL/CLI

Las SQL/CLI (*SQL/Call-Level Interface*), denominadas de forma abreviada CLI, permiten que aplicaciones desarrolladas en un cierto lenguaje de programación (con sólo las herramientas disponibles para este lenguaje y sin el uso de un precompilador) puedan incluir sentencias SQL mediante llamadas a librerías. Estas sentencias SQL se deben interpretar en tiempo de ejecución del programa, a diferencia del SQL hospedado, que requería el uso de un precompilador. a

La interfaz ODBC (*Open Database Connectivity*) define una librería de funciones que permite a las aplicaciones acceder al SGBD utilizando el SQL. Las rutinas SQL/CLI están fuertemente basadas en las características de la interfaz ODBC, y gracias al trabajo desarrollado por SAG-X/Open (*SQL Access Group-X/Open*), fueron añadidas al estándar ANSI/ISO SQL92 en 1995.

Las SQL/CLI son simplemente rutinas que llaman al SGBD para interpretar las sentencias SQL que pide la aplicación. Desde el punto de vista del SGBD, las SQL/CLI se pueden considerar, simplemente, como otras aplicaciones. a

Resumen

En esta unidad hemos presentado las sentencias más utilizadas del lenguaje estándar ANSI/ISO SQL92 de definición, manipulación y control de bases de datos relacionales. Como ya hemos comentado en la introducción, el SQL es un lenguaje muy potente, y esto hace que existan más sentencias y opciones de las que hemos explicado en este módulo. Sin embargo, no es menos cierto que hemos visto más sentencias que las que algunos sistemas relacionales ofrecen actualmente. Hemos intentado seguir con la mayor fidelidad el estándar, incluyendo comentarios sólo cuando en la mayoría de los sistemas relacionales comerciales alguna operación se hacía de forma distinta.

Conociendo el SQL92 podemos trabajar con cualquier sistema relacional comercial; sólo tendremos que dedicar unas cuantas horas a ver qué variaciones se dan con respecto al estándar.

Recordemos cómo será la creación de una base de datos con SQL:

1) En primer lugar, tendremos que dar nombre a la base de datos, con la sentencia `CREATE DATABASE`, si la hay, o con `CREATE SCHEMA`.

2) A continuación definiremos las tablas, los dominios, las aserciones y las vistas que formarán nuestra base de datos.

3) Una vez definidas las tablas, que estarán completamente vacías, se deberán llenar con la sentencia `INSERT INTO`.

Cuando la base de datos tenga un conjunto de filas, la podremos manipular, ya sea actualizando filas o bien haciéndo consultas.

Además, podemos usar todas las sentencias de control que hemos explicado.

Actividad

1. Seguro que siempre habéis querido saber dónde teníais aquella película de vídeo que nunca encontrabais. Por ello os proponemos crear una base de datos para organizar las cintas de vídeo y localizarlas rápidamente cuando os apetezca utilizarlas. Tendréis que crear la base de datos y las tablas; también deberéis decidir las claves primarias e insertar filas.
Para almacenar las cintas de vídeo, tendremos que crear las siguientes tablas:
a) Las cintas: querremos saber su código, la estantería donde se encuentran, el estante y la fila, suponiendo que en un estante haya más de una fila. Tendremos que poner nosotros el código de las cintas, con un rotulador, en el lomo de cada una.
b) Las películas: querremos saber su código, título, director principal (en el caso de que haya más de uno) y el tema. El código de las películas también lo tendremos que escribir nosotros con un rotulador para distinguir películas que tienen el mismo nombre.
c) Los actores: sólo querremos saber de ellos un código, el nombre y el apellido y, si somos aficionados al cine, otros datos que nos pueda interesar almacenar. El código de los actores, que inventaremos nosotros, nos permitirá distinguir entre actores que se llaman igual.
d) Películas que hay en cada cinta: en esta tabla pondremos el código de la cinta y el código de la película. En una cinta puede haber más de una película, y podemos tener una película repetida en más de una cinta; se debe tener en cuenta este hecho en el momento de elegir la clave primaria.
e) Actores que aparecen en las películas: en esta tabla indicaremos el código de la película y el código del actor. En una película puede participar más de un actor y un actor puede aparecer en más de una película; hay que tener presente este hecho cuando se elige la clave primaria.
Esperamos que, además de practicar sentencias de definición, manipulación y control del SQL, esta actividad os resulte muy útil.

Ejercicios de autoevaluación

Con la actividad anterior hemos practicado sentencias de definición y control del SQL. Mediante las sentencias de manipulación hemos insertado filas y, si nos hubiésemos equivocado, también habríamos borrado y modificado alguna fila. Con los ejercicios de autoevaluación practicaremos la parte de sentencias de manipulación que no hemos tratado todavía: las consultas. Los ejercicios que proponemos se harán sobre la base de datos relacional BDUOC que ha ido apareciendo a lo largo de esta unidad.

1. Obtened los códigos y los nombres y apellidos de los empleados, ordenados alfabéticamente de forma descendente por apellido y, en caso de repeticiones, por nombre.

2. Consultad el código y el nombre de los proyectos de los clientes que son de Barcelona.

3. Obtened los nombres y las ciudades de los departamentos que trabajan en los proyectos número 3 y número 4.

4. De todos los empleados que perciben un sueldo de entre 50.000 y 80.000 euros, buscad los códigos de empleado y los nombres de los proyectos que tienen asignados.

5. Buscad el nombre, la ciudad y el teléfono de los departamentos donde trabajan los empleados del proyecto GESCOM.

6. Obtened los códigos y los nombres y apellidos de los empleados que trabajan en los proyectos de precio más alto.

7. Averiguad cuál es el sueldo más alto de cada departamento. Concretamente, es necesario dar el nombre y la ciudad del departamento y el sueldo más elevado.

8. Obtened los códigos y los nombres de los clientes que tienen más de un proyecto contratado.

9. Averiguad los códigos y los nombres de los proyectos cuyos empleados asignados tienen un sueldo superior a 30.000 euros.

10. Buscad los nombres y las ciudades de los departamentos que no tienen ningún empleado asignado.

Solucionario

Ejercicios de autoevaluación

1.

```
SELECT apellido_empl, nombre_empl, codigo_empl
FROM empleados
ORDER BY apellido_empl DESC, nombre_empl DESC;
```

2. Con el SQL92 introductorio, la solución sería:

```
SELECT p.codigo_proyec, p.nombre_proyec
FROM proyectos p, clientes c
WHERE c.ciudad = 'Barcelona' and c.codigo_cli = p.codigo_cliente;
```

Con el SQL92 intermedio o con el completo, la solución sería:

```
SELECT p.codigo_proyec, p.nombre_proyec
FROM proyectos p JOIN clientes c ON c.codigo_cli = p.codigo_cliente
WHERE c.ciudad = 'Barcelona';
```

3.

```
SELECT DISTINCT e.nombre_dep, e.ciudad_dep
FROM empleados e
WHERE e.num_proyec IN (3,4);
```

4. Con el SQL92 introductorio, la solución sería:

```
SELECT e.codigo_empl, p.nombre_proyec
FROM empleados e, proyectos p
WHERE e.sueldo BETWEEN 5.0E+4 AND 8.0E+4 and e. num_proyec = p.codigo_proyec;
```

Con el SQL92 intermedio o con el completo, la solución sería:

```
SELECT e.codigo_empl, p.nombre_proyec
FROM empleados e JOIN proyectos p ON e.num_proyec = p.codigo_proyec
WHERE e.sueldo BETWEEN 5.0E+4 AND 8.0E+4;
```

5. Con el SQL92 introductorio, la solución sería:

```
SELECT DISTINCT d.*
FROM departamentos d, empleados e, proyectos p
WHERE p. nombre_proyec = 'GESCOM' and d.nombre_dep = e.nombre_dep AND
d.ciudad_dep = e.ciudad_dep and e. num_proyec = p.codigo_proyec;
```

Con el SQL92 intermedio o con el completo, la solución sería:

```
SELECT DISTINCT d.nombre_dep, d.ciudad_dep, d.telefono
FROM (departamentos dNATURAL JOIN empleados e) JOIN proyectos p ON e.num_proyec = p.codigo_proyec
WHERE p.nombre_proyec = 'GESCOM';
```

6. Con el SQL92 introductorio, la solución sería:

```
SELECT e.codigo_empl, e.nombre_empl, e.apellido_empl
FROM proyectos p, empleados e
WHERE e.num_proyec = p.codigo_proyec and p.precio = (SELECT MAX(p1. precio)
                                                     FROM proyectos p1);
```

Con el SQL92 intermedio o con el completo, la solución sería:

```
SELECT e.codigo_empl, e.nombre_empl, e.apellido_empl
FROM empleados e JOIN proyectos p ON e.num_proyec = p.codigo_proyec
WHERE p.precio = (SELECT MAX(p1.precio)
                  FROM proyectos p1);
```

7.

```
SELECT nombre_dep, ciudad_dep, MAX(sueldo) AS sueldo_maximo
FROM empleados
GROUP BY nombre_dep, ciudad_dep;
```

8. Con el SQL92 introductorio, la solución sería:

```
SELECT c.codigo_cli, c.nombre_cli
FROM proyectos p, clientes c
WHERE c.codigo_cli = p.codigo_cliente
GROUP BY c.codigo_cli, c.nombre_cli
HAVING COUNT(*) > 1;
```

Con el SQL92 intermedio o con el completo, la solución sería:

```
SELECT c.codigo_cli, c.nombre_cli
FROM proyectos p JOIN clientes c ON c.codigo_cliente = p.codigo_cliente
GROUP BY c.codigo_cli, c.nombre cli
HAVING COUNT(*) > 1;
```

9. Con el SQL92 introductorio, la solución sería:

```
SELECT p.codigo_proyec, p.nombre_proyec
FROM proyectos p, empleados e
WHERE e.num_proyec = p.codigo_proyec
GROUP BY p.codigo_proyec, p.nombre_proyec
HAVING MIN(e.sueldo) > 3.0E+4;
```

Con el SQL92 intermedio o con el completo, la solución sería:

```
SELECT p.codigo_proyec, p.nombre_proyec
FROM empleados e JOIN proyectos p ON e.num_proyec = p.codigo_proyec
GROUP BY p.codigo_proyec, p.nombre_proyec
HAVING MIN(e.sueldo)>3.0E+4;
```

10.

```
SELECT d.nombre_dep, d.ciudad_dep
FROM departamentos d
WHERE NOT EXISTS (SELECT *
                  FROM empleados e
                  WHERE e.nombre_dep = d.nombre_dep AND
                  e.ciudad_dep = d.ciudad_dep);
```

o bien:

```
SELECT nombre_dep, ciudad_dep
FROM departamentos
EXCEPT
SELECT nombre_dep, ciudad_dep
FROM empleados;
```

Bibliografía

Bibliografía básica

El SQL92 se define, según lo busquéis en ISO o en ANSI, en cualquiera de los dos documentos siguientes:

Database Language SQL (1992). Document ISO/IEC 9075:1992. International Organization for Standardization (ISO).

Database Language SQL (1992). Document ANSI/X3.135-1992. American National Standards Institute (ANSI).

Date, C.J.; Darwen, H. (1997). *A guide to the SQL Standard* (4.ª ed.). Reading, Massachusetts: Addison-Wesley.
Los libros que contienen la descripción del estándar ANSI/ISO SQL92 son bastante gruesos y pesados de leer. Este libro constituye un resumen del oficial.

Date, C.J. (2001). *Introducción a los sistemas de bases de datos* (7ª edición). Prentice Hall.
Tenéis todavía una versión más resumida de uno de los mismos autores del libro anterior en el capítulo 4 de este libro. Además en el apéndice B podéis encontrar una panorámica de SQL3.

Otros libros traducidos al castellano del SQL92 que os recomendamos son los siguientes:

Groff, J.R.; Weinberg, P.N. (1998). *LAN Times. Guía de SQL.* Osborne: McGraw-Hill.
Os recomendamos la consulta de este libro por su claridad y por los comentarios sobre el modo en el que se utiliza el estándar en los diferentes sistemas relacionales comerciales.

Silberschatz, A.; Korth, H.F.; Sudarshan, S. (1998). *Fundamentos de bases de datos*. (3.ª ed.). Madrid: McGraw-Hill.
Podéis encontrar una lectura rápida, resumida, pero bastante completa del SQL en el capítulo 4 de este libro.

Por último, para profundizar en el estudio de SQL:1999 recomendamos el siguiente libro:

Melton, J.; Simon, A.R. (2001). *SQL:1999. Undestandign Relational Language Components*. Morgan Kaufmann.

Anexos

Anexo 1

Sentencias de definición

1) Creación de esquemas:

```
CREATE SCHEMA {nombre_esquema | AUTHORIZATION usuario}
          [lista_de_elementos_del_esquema];
```

2) Borrado de esquemas:

```
DROP SCHEMA nombre_esquema {RESTRICT|CASCADE};
```

3) Creación de base de datos:

```
CREATE DATABASE nombre_base_de_datos;
```

4) Borrado de bases de datos:

```
DROP DATABASE nombre_base_de_datos;
```

5) Creación de tablas

```
CREATE TABLE nombre_tabla
             (definición_columna
             [, definición_columna...]
             [, restricciones_tabla]
```

```
                                    );
```

Donde tenemos lo siguiente:

- `definición_columna` es:

```
nombre_columna {tipos_datos|dominio} [def_defecto] [restric_col]
```

- Una de las restricciones de la tabla era la definición de claves foráneas:

```
FOREIGN KEY clave_foranea REFERENCES tabla [(clave_primaria)]
[ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
[ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
```

6) Modificación de una tabla:

```
ALTER TABLE nombre_tabla {acción_modificar_columna|
                          acción_modif_restricción_tabla};
```

Donde tenemos lo siguiente:

- `acción_modificar_columna` puede ser:

```
{ADD [COLUMN] columna def_columna|
ALTER [COLUMN] columna {SET def_defecto|DROP DEFAULT}|
DROP [COLUMN] columna {RESTRICT|CASCADE}}
```

- `acción_modif_restriccion_tabla` puede ser:

```
{ADD restricción|
DROP CONSTRAINT restricción {RESTRICT|CASCADE}}
```

7) Borrado de tablas:

```
DROP TABLE nombre_tabla {RESTRICT|CASCADE};
```

8) Creación de dominios:

```
CREATE DOMAIN nombre_dominio [AS] tipo_datos
[def_defecto] [restricciones_dominio];
```

Donde tenemos lo siguiente:

- `def_defecto` tiene el siguiente formato:

```
DEFAULT {literal|función|NULL}
```

- `restricciones_dominio` tiene el siguiente formato:

```
[CONSTRAINT nombre_restriccion] CHECK (condiciones)
```

9) Modificación de un dominio semántico:

```
ALTER DOMAIN nombre_dominio {acción_modificar_dominio|
                             acción_modif_restricción_dominio};
```

Donde tenemos lo siguiente:

- `acción_modificar_dominio` puede ser:

```
{SET def_defecto|DROP DEFAULT}
```

- `acción_modif_restricción_dominio` puede ser:

```
{ADD restricciones_dominio|DROP CONSTRAINT nombre_restricción}
```

10) Borrado de dominios creados por el usuario:

```
DROP DOMAIN nombre_dominio {RESTRICT|CASCADE};
```

11) Definición de una aserción:

```
CREATE ASSERTION nombre_aserción CHECK (condiciones);
```

12) Borrado de una aserción:

```
DROP ASSERTION nombre_aserción;
```

13) Creación de una vista:

```
CREATE VIEW nombre_vista [(lista_columnas)] AS (consulta)
[WITH CHECK OPTION];
```

14) Borrado de una vista:

```
DROP VIEW nombre_vista {RESTRICT|CASCADE};
```

Anexo 2

Sentencias de manipulación

1) Inserción de filas en una tabla:

```
INSERT INTO nombre_tabla [(columnas)]
{VALUES ({v1|DEFAULT|NULL}, ..., {{vn|DEFAULT|NULL})|<consulta>};
```

2) Borrado de filas de una tabla

```
DELETE FROM nombre_tabla
[WHERE condiciones];
```

3) Modificación de filas de una tabla:

```
UPDATE nombre_tabla
SET columna = {expresion|DEFAULT|NULL}
   [, columna = {expr|DEFAULT|NULL} ...]
WHERE condiciones;
```

4) Consultas de una base de datos relacional:

```
SELECT [DISTINCT] nombre_columnas_a_seleccionar
FROM tablas_a_consultar
[WHERE condiciones]
[GROUP BY atributos_según_los_cuales_se_quiere_agrupar]
[HAVING condiciones_por_grupos]
[ORDER BY columna_ordenación [DESC] [, columna [DESC]...]];
```

Anexo 3

Sentencias de control

1) Iniciación de transacciones:

```
SET TRANSACTION {READ ONLY|READ WRITE};
```

2) Finalización de transacciones:

```
{COMMIT|ROLLBACK} [WORK];
```

3) Autorizaciones:

```
GRANT privilegios ON objeto TO usuarios
[WITH GRANT OPTION];
```

4) Desautorizaciones:

```
REVOKE [GRANT OPTION FOR] privilegios ON objeto FROM usuarios
{RESTRICT|CASCADE};
```