El objetivo de este anexo es facilitar a aquellos lectores que desconocen el lenguaje *Structure Query Language* (SQL) una referencia rápida de las principales instrucciones o sentencias que se utilizan en él, lo que les permitirá comprender cómo podemos preparar la información de las bases de datos para poder construir las tablas de nuestro *datawarehouse* y comenzar a explotar la información con alguna de las herramientas de *Business Intelligence* disponibles. Evidentemente, hay manuales y libros mucho más extensos para aquellos lectores que quieran profundizar más en este tema.

Algunas veces, en los proyectos relacionados con las Tecnologías de la Información y de la Comunicación, los usuarios de negocio no comprenden la dificultad o facilidad de llevarlos a cabo técnicamente. Este anexo pretende despejar esa duda en lo que se refiere a las bases de datos relacionales y al acceso a los datos que contiene.

Desde el punto de vista del autor, conocer el lenguaje SQL y el modelo relacional ayuda a comprender las posibilidades de acceder a la información, lo que sin duda facilitará el entendimiento entre los distintos participantes en los proyectos de *Business Intelligence*.

Este anexo no está recomendado para aquellos lectores a los que no les interesen los aspectos más técnicos del acceso físico a los datos residentes en una base de datos relacional.

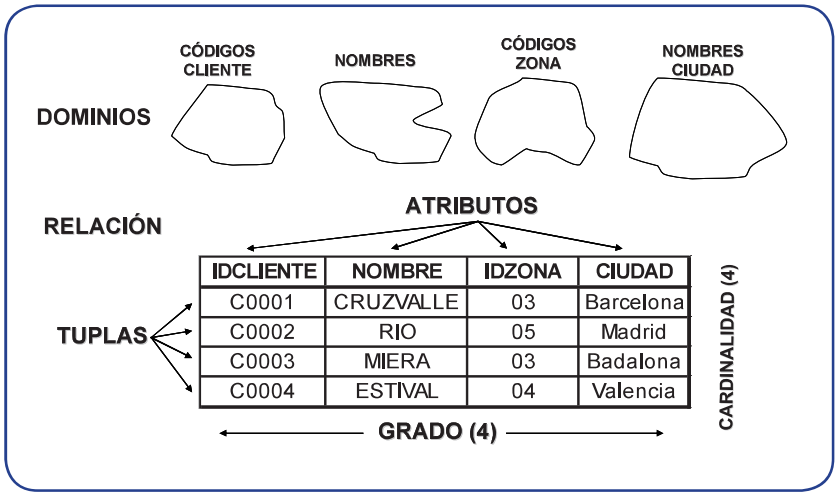
La mayoría de las aplicaciones actuales, facturación, gestión, aplicaciones *web*, utilizan una base de datos. La mayoría de ellas se basan en el modelo relacional. SQL es un lenguaje utilizado por las bases de datos relacionales (RDBMS). Utilizando SQL se puede recuperar o modificar la información independientemente de la base de datos en la que esté.

Las bases de datos relacionales permiten a los usuarios acceder a los datos bien mediante interfases que permiten escribir las sentencias SQL que veremos a continuación, o bien mediante interfases gráficos

que son conocidos como *query-by-example* (QBE)[128]. Estas utilidades usualmente generan el código de las sentencias SQL y se pueden consultar y modificar antes de ejecutar la consulta a la base de datos.

Bases de datos relacionales

Una base de datos relacional es una base de datos que es percibida por el usuario como una colección de tablas. Cada tabla está formada por filas (registros o tuplas) y columnas (atributos o campos). Las tablas están compuestas por registros o tuplas y cada uno de los registros tiene distintos atributos o campos.



Las tablas o relaciones deben cumplir varios requisitos:

128 Una traducción posible es "consultas por ejemplos".

- No existen registros repetidos.
- El orden de los registros no es significativo.
- El orden de los atributos no es significativo.
- Los valores de los atributos son atómicos.

Codd propuso las bases de datos relacionales en 1970, pero fue posteriormente, en 1974, cuando Chanberlin y Boyce publicaron un artículo proponiendo un lenguaje estructurado que llamaron SEQUEL, que es el origen del SQL.

El modelo relacional fue desarrollado posteriormente por C.J. Date y H. Darven, entre otros.

Las bases de datos relacionales[129] tienen al menos dos características:

- La información está integrada, generalmente en un solo lugar de la base de datos. Los usuarios de los distintos procesos acceden a la misma información en una sola ubicación, lo que minimiza las redundancias de información.
- La información es relacional, está interrelacionada con otras informaciones en otras partes de la base de datos.

Componentes del SQL

Sentencias[130] SQL

Para crear las tablas y los índices son necesarias las sentencias DDL (*Data Definition Language*). Algunos de las sentencias son:

129 Adaptado del artículo "Relational Data Models in Enterprise-Level Information Systems" de Mark J. Cotteleer, Harvard Business School, 2002.

130 Sentencia o cláusula es la traducción de la terminología inglesa "*command*", coloquialmente es muy habitual el uso del término castellanizado "*comando*". A lo largo del anexo utilizaremos indistintamente sentencia o cláusula.

- CREATE TABLE: Crear tablas.
- CREATE INDEX: Crear índices (para ordenar las tablas).
- DROP TABLE: Eliminar tablas.
- DROP INDEX: Eliminar índices.
- ALTER TABLE: Modificar la estructura de las tablas.

Las sentencias que nos permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos son los que constituyen el DML (*Data Manipulation Language*). DML soporta las siguientes acciones:

- SELECT: Consultar registros que satisfagan un criterio determinado.
- INSERT: Utilizado para cargar datos en una única operación.
- UPDATE: Utilizado para modificar los valores de los campos y registros especificados.
- DELETE: Utilizado para eliminar registros de una tabla.

Nos centraremos en las sentencias DML, ya que son los que nos permiten hacer las consultas para crear las tablas del *datawarehouse*. La mayoría de los motores de bases de datos tienen asistentes para crear o modificar las tablas, sin necesidad de utilizar directamente las sentencias DDL.

Sentencia SELECT simple

La sentencia SELECT es la más utilizada en SQL. La sentencia SELECT sirve para recuperar un conjunto de registros. La forma más simple de la sentencia SELECT[131] es:

```
SELECT *
FROM <nombre de la tabla>
```

Nos devuelve todas las filas o registros de la tabla, con todos sus atributos (*).

Para comprender mejor el uso de las sentencias supongamos las siguientes tablas[132]:

Libros

ISBN	Titulo	CodEdit	Precio
0-103-45678-9	Iliad	1	25,00
0-11-345678-9	Moby Dick	3	49,00
0-12-333433-3	On Liberty	1	25,00
0-12-345678-9	Jane Eyre	3	49,00
0-123-45678-0	Ulysses	2	34,00
0-321-32132-1	Balloon	3	34,00
0-55-123456-9	Main Street	3	22,95
0-555-55555-9	MacBeth	2	12,00
0-91-045678-5	Hamlet	2	20,00
0-91-335678-7	Fairie Queene	1	15,00
0-99-777777-7	King Lear	2	49,00
0-99-999999-9	Emma	1	20,00
1-1111-1111-1	C++	1	29,95
1-22-233700-0	Visual Basic	1	25,00
0-201-96426-0	The SQL Standard	5	39,95
1-56592-297-2	Access Database	5	24,95

131 Pueden haber pequeñas diferencias entre las sentencias SQL utilizadas por los distintos motores de bases de datos, por lo que se deberían consultar los manuales de dichas aplicaciones para adaptar las sentencias. Por ejemplo, en muchos motores de bases de datos al final de cada sentencia SQL se debe añadir un punto y coma (;) que hemos omitido en los ejemplos.

132 Información de las tablas obtenida del libro "Access Database: Design & Programming" Steven Roman, O'Reilly, 1997.

Editorial

CodEdit	NomEdit	TelEdit
1	Big House	123-456-7890
2	Alpha Press	999-999-9999
3	Small House	714-000-0000
4	Beta Press	234-523-3459

El resultado de:

```
SELECT *
FROM Editorial
```

sería:

CodEdit	NomEdit	TelEdit
1	Big House	123-456-7890
2	Alpha Press	999-999-9999
3	Small House	714-000-0000
4	Beta Press	234-523-3459

Si queremos especificar los atributos que nos devuelve deberíamos especificar cada uno de los atributos de la siguiente forma:

```
SELECT atributo 1, ..., atributo n
FROM <nombre de la tabla>
```

El resultado de:

```
SELECT CodEdit, NomEdit
FROM Editorial
```

será:

CodEdit	NomEdit
1	Big House
2	Alpha Press
3	Small House
4	Beta Press

Cuando no queremos recuperar todos los registros de la tabla, sino sólo aquellos que cumplen una condición, debemos utilizar la cláusula WHERE. El resultado esta restringido a un conjunto de condiciones.

```
SELECT atributo 1, …, atributo n
FROM <nombre de la tabla>
WHERE atributo 1= 'valor'
```

Imaginemos que tenemos una tabla de Proveedores, de los cuales tenemos su código, nombre, teléfono y población. Queremos recuperar los proveedores y los teléfonos de los que tienen su domicilio en Zaragoza. La sentencia SELECT debería ser:

```
SELECT nombre, teléfono
FROM Proveedores
WHERE población = 'Zaragoza'
```

La cláusula SELECT anterior solo nos recuperará el nombre y el teléfono de los proveedores que tienen su población en Zaragoza. Se debe notar que para Zaragoza en la cláusula WHERE lo ponemos entre '', también conocido como delimitador de cadenas.

Podríamos utilizar en lugar del operador igual (=) los operadores de comparación:

- <> distinto que
- < menor que
- > mayor que
- <= menor o igual que
- >= mayor o igual que

Además, disponemos de operadores específicos como LIKE, IN, BETWEEN que describiremos a continuación.

Si quisiéramos recuperar aquellos proveedores cuya ciudad comienza por la letra Z, deberíamos utilizar el operador LIKE:

```
SELECT nombre, teléfono
FROM Proveedores
WHERE población LIKE 'Z*'
```

Z* significa que comience por Z seguida de cualquier conjunto de caracteres (*).

Si quisiéramos aquellas que comienzan con cualquier letra comprendida entre la letra N y la Z, deberíamos cambiar por '[N-Z]*'

Si lo que queremos es que sólo nos muestre los proveedores de tres poblaciones concretas, deberemos utilizar el operador IN:

```
SELECT nombre, teléfono
FROM Proveedores
WHERE población IN ('Madrid', 'Barcelona', 'Zaragoza')
```


Nos devolverá aquellos que estén en cualquiera de las tres ciudades. Si lo que queremos obtener son aquellos proveedores que no están en ninguna de las tres ciudades, deberíamos utilizar el operador NOT IN.

Podemos también recuperar aquellos registros que se encuentran en un intervalo de valores mediante el operador BETWEEN. La sentencia sería:

```
SELECT atributo 1, …, atributo n
FROM <nombre de la tabla>
WHERE atributo 1 BETWEEN  'valor1' AND 'valor2'
```

El operador BETWEEN admite el operador NOT: En ese caso nos devolvería los registros que no están incluidos en el intervalo.

Se puede dar el caso de que tengamos algún atributo que no contenga ningún valor. Para localizar los registros en los que un atributo no tiene valor, debemos utilizar el operador IS NULL: Este operador también admite el operador NOT. La sintaxis sería:

```
SELECT atributo 1, …, atributo n
FROM <nombre de la tabla>
WHERE atributo 1 IS NULL
```

Nos podría interesar ordenar los registros. Para ello, la sentencia SQL debería incluir la cláusula ORDER BY:

```
SELECT atributo 1, …, atributo n
FROM <nombre de la tabla>
WHERE atributo 1= 'valor'
ORDER BY lista de atributos ASC | DES
```

Por defecto el orden es ascendente (ASC); deberemos incluir DES para que sea descendente. En la cláusula ORDER BY podemos utilizar, en lugar del nombre del atributo de la tabla, la posición que ocupa en la cláusula SELECT.

En el ejemplo anterior podríamos ordenar el resultado de nuestra consulta de proveedores alfabéticamente por el atributo nombre, en este caso optamos por utilizar ORDER BY 1, que significa por le primer atributo de la sentencia SELECT:

```
SELECT nombre, teléfono
FROM Proveedores
WHERE población= 'Zaragoza'
ORDER BY 1
```

Podríamos ordenar la consulta por más campos, tan sólo los tendríamos que añadir a la cláusula ORDER BY separándolos mediante comas, por ejemplo: ORDER BY 1, teléfono.

En SQL disponemos de los operadores lógicos:

- AND: 'y' lógico
- OR: 'o' lógico
- XOR: 'o' exclusivo lógico
- NOT: 'no' lógico

Estos operadores nos permiten enlazar las condiciones de las expresiones de la siguiente forma:

<expresión 1> Operador <expresión 2>

Por ejemplo, si queremos seleccionar aquellos proveedores cuya población es Zaragoza y su número de teléfono finaliza por 3, la sentencia SQL completa sería:

```
SELECT nombre, teléfono
FROM Proveedores
WHERE población= 'Zaragoza' AND teléfono LIKE '*3'
```


Sentencia SELECT con predicado

El predicado se incluye en la cláusula SELECT antes del primer atributo a recuperar. Los posibles predicados son:

- ALL: Devuelve todos los campos de la tabla.
- TOP: Devuelve un determinado número de registros de la tabla.
- DISTINCT: Omite los registros cuyos campos seleccionados coincidan totalmente.
- DISTINCTROW: Omite los registros duplicados basándose en la totalidad del registro y no sólo en los campos seleccionados.

Veamos cada uno de ellos en detalle:

El predicado ALL es por defecto. Las siguientes sentencias son equivalentes:

```
SELECT ALL * FROM Tabla
SELECT * FROM Tabla
```

El predicado TOP recupera o un número determinado de registros o un porcentaje, en caso de que utilicemos la palabra reservada PERCENT. El uso del predicado TOP tiene más sentido cuando utilizamos la cláusula ORDER BY, puesto que, si no, nos devolverá un número de registros o un porcentaje sin ningún criterio: El criterio será el número de registros de la tabla. La sintaxis es:

```
SELECT TOP n atributo 1, …, atributo n
FROM Tabla
ORDER BY atributo
```

o

```
SELECT TOP n PERCENT atributo 1, …, atributo n
FROM Tabla
ORDER BY atributo
```


Imaginemos que tenemos una tabla Ventas con las ventas acumuladas de los clientes, con: código de cliente, nombre, ventasacum. La sentencia:

```
SELECT TOP 20 PERCENT nombre, ventasacum
FROM Ventas
ORDER BY ventasacum
```

Nos devolvería la lista de clientes que acumulan el 20% de las ventas.

Los predicados DISTINCT y DISTINCTROW actúan de la misma forma, pero el primero tiene sólo en cuenta los atributos que se devuelven en la consulta, mientras que el segundo tiene en cuenta todos los atribu-

tos de la tabla, independientemente de que estén en la consulta o no. Estos predicados nos devuelven aquellos registros sin duplicados.

Según nuestro ejemplo de los proveedores, la sentencia

```
SELECT DISTINCT población
FROM Proveedores
```

nos devolvería la lista de poblaciones en las cuales residen nuestros proveedores sin duplicados, es decir, si tenemos dos proveedores en Zaragoza, el valor Zaragoza tan sólo aparecería una vez.

En el caso de que utilizáramos el predicado DISTINCTROW, si tuviéramos dos proveedores en Zaragoza con distintos nombres aparecería Zaragoza tantas veces como proveedores distintos tuviéramos.

Sentencia SELECT con funciones agregadas

Las funciones de agregado se utilizan, dentro de una cláusula SELECT, sobre los registros que seleccionaremos para devolver un único valor.

- AVG: Se utiliza para calcular la media aritmética de los valores de un atributo determinado.
- COUNT: Se utiliza para devolver el número de registros de la selección.
- SUM: Se utiliza para devolver la suma de todos los valores de un atributo determinado.
- MAX: Se utiliza para devolver el valor más alto de un atributo determinado.
- MIN: Se utiliza para devolver el valor más bajo de un atributo determinado.

La sintaxis es del tipo:

```
SELECT Función agregada(atributo)
FROM Tabla
```

Por ejemplo si utilizamos la tabla Ventas, con las ventas acumuladas de los clientes, con: código de cliente, nombre, ventasacum. La sentencia:

```
SELECT SUM (ventasacum)
FROM Ventas
```

Nos devolverá el importe total de las ventas.

Las sentencias SELECT pueden incorporar cláusulas WHERE, lo que implicará que la función de agregado se aplicará tan sólo sobre el conjunto de registros que cumplen la cláusula WHERE.

La función AVG se puede aplicar directamente sobre un campo, o bien sobre una expresión que realiza un cálculo sobre los valores contenidos en el campo.

El resultado que obtendremos es la media calculada, es decir la de la suma de los valores del campo dividido por el número de registros, sin tener en cuenta aquellos que el valor del campo es nulo.

La función COUNT puede contar el total de registros de la tabla si utilizamos la forma COUNT (*) o bien los registros según un atributo, en caso de que en alguno de los atributos el valor del mismo sea nulo no lo tendrá en cuenta.

Al aplicar la función COUNT nos puede interesar hacerlo sobre varios atributos a la vez, para ello deberemos utilizar el operador concatenación '&'. La sintaxis es:

```
SELECT COUNT (atributo 1& atributo 2)
FROM Tabla
```

En este caso COUNT cuenta un registro sólo si al menos uno de los dos atributos no es nulo.

La función SUM nos devuelve la suma de los valores de un atributo o de una operación entre atributos.

Las operaciones que podemos hacer entre atributos con valores numéricos son las aritméticas:

- suma (+)
- resta (-)
- división (/)
- multiplicación (*)

Podemos ejecutar por ejemplo la siguiente sentencia SQL:

```
SELECT SUM(atributo 1*atributo 2)
FROM Tabla
```

El resultado que obtendremos será la suma de multiplicar el atributo 1 por el atributo 2 de cada uno de los registros de la Tabla.

Con los atributos que contienen textos podemos utilizar el operador de concatenación &.


```
SELECT atributo 1&atributo 2
FROM Tabla
```

Si disponemos de una tabla Nombres en la disponemos del nombre y del primer apellido los podríamos concatenar, la sentencia es:

```
SELECT apellido&" ,"&nombre
FROM Nombres
```

Obtendríamos los apellidos seguidos de un espacio en blanco, una coma y el nombre, el formato sería: 'apellido, nombre'.

Las funciones MAX y MIN nos devuelven el valor máximo o mínimo de un conjunto de valores de un atributo en una consulta.

Las funciones agregadas no pueden utilizarse como parte de la cláusula WHERE.

Cuando utilizamos funciones agregadas es útil el uso de alias. Para asignar alias utilizamos la palabra reservada AS. La sintaxis es:

```
SELECT COUNT(*) AS Total
FROM Tabla
```

En el ejemplo contamos los registros de la tabla y lo llamamos Total.

El uso de alias también es útil cuando creamos nuevos campos, por ejemplo:

```
SELECT atributo 1*(1+ atributo 2) AS nuevo_atributo
FROM Tabla
```

o

```
SELECT atributo 1& atributo 2 AS nuevo_atributo
FROM Tabla
```

Debemos señalar que en la primera sentencia SELECT hemos utilizado los delimitadores de precedencia ( ) para indicar el orden en las operaciones, sumará 1 al valor del atributo 2 y posteriormente lo multiplicará por el valor del atributo 1, o bien multiplicará el valor del atributo 1 por la suma de 1 más el valor del atributo 2. Obviamente el resultado es el mismo, dependerá del motor de la base de datos.

Sentencia SELECT con grupos.

Tiene por objeto organizar agrupaciones por los valores de un atributo en lugar de la tabla entera. La cláusula es GROUP BY. Normalmente utilizamos GROUP BY cuando queremos aplicar funciones de agregado, pero también lo podemos utilizar para conocer la existencia de los grupos. Su sintaxis es:

```
SELECT atributos
FROM tabla
WHERE criterio
GROUP BY atributo del grupo
```

Los valores nulos en el GROUP BY se tienen en cuenta y se agrupan en un grupo.

En el ejemplo de los proveedores la sentencia SQL:

```
SELECT COUNT(*), población
FROM Proveedores
GROUP BY población
```

Nos contaría el número de proveedores que tenemos en cada una de las ciudades, nos devolvería el nombre de la ciudad y el número de proveedores.

Como hemos comentado anteriormente, no podemos utilizar funciones de agregado en cláusulas WHERE, para ello tenemos la cláusula HAVING.

Si, por ejemplo, sólo queremos aquellos grupos en que hay más de cuatro registros, la sentencia SELECT sería:


```
SELECT COUNT(*), población
FROM Proveedores
GROUP BY población
HAVING COUNT(*)>4
```

En la sentencia anterior podríamos incluir la cláusula WHERE, con la que restringiríamos los registros antes de agruparlos.

Sentencia SELECT completa.

Una sentencia SELECT completa tendría la siguiente sintaxis:

```
SELECT [predicado] Lista de Atributos | *
[INTO Nueva tabla]
FROM Lista de tablas
[WHERE CondiciónSelecciónFilas]
[GROUP BY Lista de Atributos]
```

```
[HAVING CondiciónSelecciónGrupos]
[ORDER BY Lista de Atributos];
```

Las cláusulas entre [ ] son opcionales, y el símbolo | indica que debemos escoger entre una opción o la otra.

Aparte de las cláusulas que hemos visto en los apartados anteriores hemos añadido la cláusula INTO, que nos permite guardar el resultado en una nueva tabla con el nombre que le indiquemos. Por defecto, la sentencia SELECT tan sólo nos muestra los resultados por pantalla, pero no los guarda físicamente. Cuando nos interese esta opción deberemos utilizar INTO.

Consultas de referencia cruzada.

Las consultas de referencia cruzada se utilizan para visualizar la información en filas y columnas, lo que facilita su interpretación. La sintaxis es:

```
TRANSFORM Atributo
SELECT [predicado] Atributo de la fila [,Atributo]...
FROM Tabla
[WHERE CondiciónSelecciónFilas]
GROUP BY Atributo de la fila [,Atributo]...
PIVOT Atributo de la columna
```

Veamos un ejemplo que muestre la utilidad de las consultas de referencia cruzada utilizando la tabla Libros[133] que mostramos a continuación:

133 Información de las tablas obtenida del libro "Access Database: Design & Programming" Steven Roman, O'Reilly, 1997.

ISBN	Titulo	NomAutor	NomEdit	Precio
1-1111-1111-1	C++	Roman	Big House	$29.95
0-99-999999-9	Emma	Austen	Big House	$20.00
0-91-335678-7	Fairie Queene	Spencer	Big House	$15.00
0-91-045678-5	Hamlet	Shakespeare	Alpha Press	$20.00
0-103-45678-9	Iliad	Homer	Big House	$25.00
0-12-345678-6	Jane Eyre	Austen	Small House	$49.00
0-99-777777-7	King Lear	Shakespeare	Alpha Press	$49.00
0-555-55555-9	Macbeth	Shakespeare	Alpha Press	$12.00
0-11-345678-9	Moby Dick	Melville	Small House	$49.00
0-12-333433-3	On Liberty	Mill	Big House	$25.00
0-321-32132-1	Balloon	Sleepy	Small House	$34.00
0-321-32132-1	Balloon	Snoopy	Small House	$34.00
0-321-32132-1	Balloon	Grumpy	Small House	$34.00
0-55-123456-9	Main Street	Jones	Small House	$22.95
0-55-123456-9	Main Street	Smith	Small House	$22.95
0-123-45678-0	Ulysses	Joyce	Alpha Press	$34.00
1-22-233700-0	Visual Basic	Roman	Big House	$25.00

Si queremos saber cuántos libros han escrito los autores para cada una de las editoriales, utilizando la sentencia SELECT su sintaxis sería:

```
SELECT NomAutor,NomEdit,COUNT(*) AS TOTAL
FROM Libros
GROUP BY NomAutor,NomEdit
ORDER BY 3 DESC
```

El resultado que obtendríamos sería:

NOMAUTOR	NOMEDIT	TOTAL
Shakespeare	Alpha Press	3
Roman	Big House	2
Spencer	Big House	1
Snoopy	Small House	1
Smith	Small House	1
Sleepy	Small House	1
Mill	Big House	1
Melville	Small House	1
Joyce	Alpha Press	1
Jones	Small House	1
Homer	Big House	1
Grumpy	Small House	1
Austen	Small House	1
Austen	Big House	1

Mediante el uso de la cláusula TRANSFORM la sintaxis es:


```
TRANSFORM Count(*) AS TOTAL
SELECT NomAutor
FROM Libros
GROUP BY NomAutor
PIVOT NomEdit
```

El resultado obtenido sería:

NomAutor	Alpha Press	Big House	Small House
Austen		1	1
Grumpy			1
Homer		1	
Jones			1
Joyce	1		
Melville			1
Mill		1	
Roman		2	
Shakespeare	3		
Sleepy			1
Smith			1
Snoopy			1
Spencer		1	

Como podemos ver, tenemos en cada una de las filas los autores, en las columnas los editores, y en el interior de la tabla los libros que cada autor ha escrito para cada editor.

Las consultas de referencias cruzadas nos facilitan la interpretación de la información.

Operaciones con JOIN

Hasta ahora hemos utilizado ejemplos sobre tablas únicas, la situación habitual en toda base de datos relacional es que dispongamos de más de una tabla, JOIN nos permite hacer consultas sobre distintas tablas que están relacionadas a través de los atributos (más específicamente a través de las claves externas).

En el ejemplo vamos a utilizar las tablas: Libros y Editoriales, presentadas al inicio del anexo en las sentencia SELECT simple.

Si analizamos las dos tablas veremos que el campo CodEdit es común, y además es el que nos permitirá hacer consultas sobre las dos tablas.

Existen distintos tipos de JOIN, que veremos a continuación:

En los equi[134]-joins combinamos los datos de dos o más tablas, basándonos en la relación establecida en la cláusula WHERE. Utilizando las tablas del ejemplo, podríamos contar cuantos libros tenemos de la Editorial Big House:

```
SELECT COUNT(*)
FROM Libros, Editorial
WHERE Libros.CodEdit = Editorial.CodEdit AND NomEdit =
'Big House'
```

134 Los joins de tipo equi combinan los registros de dos tablas siempre que haya concordancia de valores en los campo comunes por los que relacionamos las tablas.

Debemos señalar que cuando los atributos tienen el mismo nombre en las dos tablas los precedemos por el nombre de la tabla más un punto (nombretabla.nombreatributo). Como veremos más adelante, la cláusula JOIN dentro de la cláusula FROM nos permitirá hacer las consultas sobre las distintas tablas obteniendo el mismo resultado que utilizando las de la cláusula WHERE, tal como recomienda la American National Standards Institute (ANSI). En la mayoría de los motores de bases de datos esta forma es más eficiente.

El uso de la cláusula INNER JOIN nos permitirá unir dos o más tablas a partir de los atributos comunes y nos devolverá tan sólo los registros que tengan valores iguales en los atributos de cada una de las tablas (equi-join, se basa en la intersección, el resultado será de los que tengan valores comunes en las dos tablas). En nuestro ejemplo la sentencia SELECT sería:

```
SELECT ISBN,Titulo,NomEdit,Precio
FROM LIBROS INNER JOIN EDITORIAL
ON LIBROS.CodEdit = EDITORIAL.CodEdit
```

El resultado que obtendremos es:

ISBN	Titulo	NomEdit	Precio
0-103-45678-9	Iliad	Big House	25,00
0-12-333433-3	On Liberty	Big House	25,00
0-91-335678-7	Fairie Queene	Big House	15,00
0-99-999999-9	Emma	Big House	20,00
1-1111-1111-1	C++	Big House	29,95
1-22-233700-0	Visual Basic	Big House	25,00
0-123-45678-0	Ulysses	Alpha Press	34,00
0-555-55555-9	MacBeth	Alpha Press	12,00
0-91-045678-5	Hamlet	Alpha Press	20,00
0-99-777777-7	King Lear	Alpha Press	49,00
0-11-345678-9	Moby Dick	Small House	49,00
0-12-345678-9	Jane Eyre	Small House	49,00
0-321-32132-1	Balloon	Small House	34,00
0-55-123456-9	Main Street	Small House	22,95

Si nos fijamos en el resultado obtenido, la tabla Libros tenía 16 registros, mientras que la tabla obtenida tan sólo tiene 14. Los registros de Libros en los que el valor del Código de Editorial era 5 no aparecen,

ya que en la tabla Editorial no tenemos ningún registro cuyo valor de código de la editorial sea 5.

Si estuviéramos interesados en obtener todos los registros de la tabla Libros aunque no tengan editorial, en la tabla Editoriales deberemos utilizar la cláusula LEFT JOIN (Se trata de un outer join[135] que se basa en la unión).

Su sintaxis sería:

```
SELECT ISBN, Titulo, NomEdit, Precio
FROM LIBROS LEFT JOIN EDITORIAL
ON Libros.CodEdit = Editorial.CodEdit
```

El resultado que obtendríamos es:

ISBN	Titulo	NomEdit	Precio
0-103-45678-9	Iliad	Big House	25,00
0-12-333433-3	On Liberty	Big House	25,00
0-91-335678-7	Fairie Queene	Big House	15,00
0-99-999999-9	Emma	Big House	20,00
1-1111-1111-1	C++	Big House	29,95
1-22-233700-0	Visual Basic	Big House	25,00
0-123-45678-0	Ulysses	Alpha Press	34,00
0-555-55555-9	MacBeth	Alpha Press	12,00
0-91-045678-5	Hamlet	Alpha Press	20,00
0-99-777777-7	King Lear	Alpha Press	49,00
0-11-345678-9	Moby Dick	Small House	49,00
0-12-345678-9	Jane Eyre	Small House	49,00
0-321-32132-1	Balloon	Small House	34,00
0-55-123456-9	Main Street	Small House	22,95
0-201-96426-0	The SQL Standard		39,95
1-56592-297-2	Access Database		24,95

135 El concepto de OUTER JOIN existe en la mayoría de los motores de bases de datos, pero su sintaxis puede variar de vendedor a vendedor, por lo que se debería consultar el manual de la aplicación particular para comprobar que la sintaxis que estamos utilizando es la adecuada.

Podemos observar que tenemos los 16 registros de la tabla Libros y los dos últimos no tienen nombre de editorial, ya que en la tabla Editoriales no tenemos ninguna con el valor 5 en el código de editorial.

Si por otro lado quisiéramos obtener todos los registros de la tabla Editorial independientemente de que tengan registros en la tabla Libros, deberemos utilizar un RIGHT JOIN (se trata de la otra opción que permiten los outer joins). La sentencia SELECT debería ser:

```
SELECT ISBN,Titulo,NomEdit,Precio
FROM LIBROS RIGHT JOIN  EDITORIAL
ON Libros.CodEdit = Editorial.CodEdit
```

El resultado que obtendríamos es:

ISBN	Titulo	NomEdit	Precio
0-103-45678-9	Iliad	Big House	25,00
0-12-333433-3	On Liberty	Big House	25,00
0-91-335678-7	Fairie Queene	Big House	15,00
0-99-999999-9	Emma	Big House	20,00
1-1111-1111-1	C++	Big House	29,95
1-22-233700-0	Visual Basic	Big House	25,00
0-123-45678-0	Ulysses	Alpha Press	34,00
0-555-55555-9	MacBeth	Alpha Press	12,00
0-91-045678-5	Hamlet	Alpha Press	20,00
0-99-777777-7	King Lear	Alpha Press	49,00
0-11-345678-9	Moby Dick	Small House	49,00
0-12-345678-9	Jane Eyre	Small House	49,00
0-321-32132-1	Balloon	Small House	34,00
0-55-123456-9	Main Street	Small House	22,95
		Beta Press	

Como podemos observar, en el resultado tenemos los 14 registros de los libros que tienen editorial y un último registro de la editorial con código de editorial 4, de la cual no tenemos ningún libro.

Resumiendo una sentencia SELECT utilizando los distintos JOIN tendría la siguiente sintaxis:

```
SELECT [predicado] Lista de Atributos | *
FROM Tabla1 {INNER|LEFT|RIGHT}  JOIN Tabla2
ON Tabla1.Atributo1 = Tabla2.Atributo1
[{AND|OR ON Tabla1.Atributo2=Tabla2.Atributo2},...];
```

Producto cartesiano

El producto cartesiano combina cada uno de los registros de una tabla con cada uno de los de la otra tabla. La sentencia SELECT sería:

```
SELECT *
FROM A, B
```

El resultado obtenido sería la combinación de todos los registros de la tabla A con cada uno de los registros de la tabla B.

Unión

El operador UNION permite combinar registros con los mismos atributos: El resultado de dos tablas o dos consultas en un nuevo conjunto de registros. La única restricción es que el resultado de la unión debe ser compatible, es decir, el número de atributos en el resultado debe ser el mismo. El operador UNION es útil cuando queremos combinar registros que están almacenados en distintas tablas.

La sintaxis del operador UNION es:

```
Tabla | Sentencia SELECT
UNION
Tabla | Sentencia SELECT
```

Tanto las Tablas como las sentencias SELECT deben tener el mismo número de atributos y los atributos deben ser del mismo tipo de datos, es decir, no podemos unir dos tablas en las que uno de los atributos tiene distinto tipo de datos, por ejemplo números en una y texto en la

otra, a no ser que utilicemos una función para convertir los números en texto.

Sentencia INSERT

La sentencia INSERT se utiliza para añadir registros a las tablas. Podemos insertar registros de uno en uno o añadir varios a la vez.

La sintaxis de la sentencia INSERT para añadir un solo registro es:

```
INSERT
INTO Tabla [(Atributo [,Atributo]…)]
VALUES (valor [,valor]…)
```

Para añadir varios registros es:

```
INSERT
INTO Tabla [(Atributo [,Atributo]…)]
SELECT ...
```


Sentencia UPDATE

Se utiliza para modificar los valores de los atributos de un registro o de un conjunto de registros. Su sintaxis es:

```
UPDATE Tabla
SET  Atributo=Expresión [,Atributo=Expresión]...
[WHERE CondiciónSelecciónFilas]
```

Dependerá de la cláusula WHERE si lo hacemos sobre un registro o sobre el conjunto de registros que cumplan esta cláusula.

Si en una consulta de actualización suprimimos la cláusula WHERE ,todos los registros de la tabla serán actualizados.

Sentencia DELETE

La sentencia DELETE elimina un registro o conjunto de registros de una tabla. Elimina el registro completo, no se puede utilizar para eliminar parte del registro. Su sintaxis es:

```
DELETE *
FROM Tabla
[WHERE CondiciónSelecciónFilas]
```

Dependerá de la cláusula WHERE si lo hacemos sobre un registro o sobre el conjunto de registros que cumplan esta cláusula.

Si en una consulta de actualización suprimimos la cláusula WHERE, todos los registros de la tabla serán borrados.

Consultas anidadas.

En algunos casos es necesario emplear el resultado de una consulta para seleccionar valores de otra. Es lo que llamamos consultas anidadas: hay una subconsulta anidada dentro de una consulta. La subconsulta se encierra entre paréntesis. La subconsulta puede devolver un solo resultado o resultados múltiples. Las consultas anidadas las podemos utilizar dentro de cláusulas SELECT, INSERT, UPDATE, DELETE.

Veamos un ejemplo: Utilizando la tabla Libros que vimos en apartados anteriores, si quisiéramos saber el título de los libros cuyo precio es mayor al precio promedio de los libros de la Editorial Alpha Press, la sentencia SELECT debería ser:

```
SELECT DISTINCT Título
FROM Libros
WHERE Precio >(SELECT AVG(Precio)
FROM Libros
WHERE NomEdit='Alpha Press')
```

En este caso, para enlazar la subconsulta hemos utilizado el operador >. Los operadores de los que disponemos son:

- Cualquier comparador (>,<,=,..). La subconsulta debe mostrar un resultado único.
- Cualquier comparador seguido de ALL, ANY o SOME. La subconsulta debe proporcionar múltiples registros.
- El nombre de un campo + IN/NOT IN. La subconsulta debe proporcionar múltiples registros. Se seleccionan los registros para los que el valor del campo aparezca también en el resultado de la subconsulta. Es equivalente a "=ANY".
- La cláusula EXISTS/NOT EXISTS. La subconsulta debe proporcionar múltiples registros. La condición evaluada es que en la subconsulta se recupere algún registro (EXISTS) o no se recupere ningún registro (NOT EXISTS).