Curso de SQL

Introducción

El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado, utilizado por los diferentes motores de bases de datos para realizar determinadas operaciones sobre los datos o sobre la estructura de los mismos. Pero como sucede con cualquier sistema de normalización hay excepciones para casi todo; de hecho, cada motor de bases de datos tiene sus peculiaridades y lo hace diferente de otro motor, por lo tanto, el lenguaje SQL normalizado (ANSI) no nos servirá para resolver todos los problemas, aunque si se puede asegurar que cualquier sentencia escrita en ANSI será interpretable por cualquier motor de datos.

Breve Historia

La historia de SQL (que se pronuncia deletreando en inglés las letras que lo componen, es decir "ese-cu-ele" y no "siquel" como se oye a menudo) empieza en 1974 con la definición, por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación de IBM, de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional. Este lenguaje se llamaba SEQUEL (Structured English Query Language) y se implementó en un prototipo llamado SEQUEL-XRM entre 1974 y 1975. Las experimentaciones con ese prototipo condujeron, entre 1976 y 1977, a una revisión del lenguaje (SEQUEL/2), que a partir de ese momento cambió de nombre por motivos legales, convirtiéndose en SQL. El prototipo (System R), basado en este lenguaje, se adoptó y utilizó internamente en IBM y lo adoptaron algunos de sus clientes elegidos. Gracias al éxito de este sistema, que no estaba todavía comercializado, también otras compañías empezaron a desarrollar sus productos relacionales basados en SQL. A partir de 1981, IBM comenzó a entregar sus productos relacionales y en 1983 empezó a vender DB2. En el curso de los años ochenta, numerosas compañías (por ejemplo Oracle y Sybase, sólo por citar algunos) comercializaron productos basados en SQL, que se convierte en el estándar industrial de hecho por lo que respecta a las bases de datos relacionales.

En 1986, el ANSI adoptó SQL (sustancialmente adoptó el dialecto SQL de IBM) como estándar para los lenguajes relacionales y en 1987 se transfomó en estándar ISO. Esta versión del estándar va con el nombre de SQL/86. En los años siguientes, éste ha sufrido diversas revisiones que han conducido primero a la versión SQL/89 y, posteriormente, a la actual SQL/92.

El hecho de tener un estándar definido por un lenguaje para bases de datos relacionales abre potencialmente el camino a la intercomunicabilidad entre todos los productos que se basan en él. Desde el punto de vista práctico, por desgracia las cosas fueron de otro modo. Efectivamente, en general cada productor adopta e implementa en la propia base de datos sólo el corazón del lenguaje SQL (el así llamado Entry level o al máximo el Intermediate level), extendiéndolo de manera individual según la propia visión que cada cual tenga del mundo de las bases de datos.

Actualmente, está en marcha un proceso de revisión del lenguaje por parte de los comités ANSI e ISO, que debería terminar en la definición de lo que en este momento se conoce como SQL3. Las características principales de esta nueva encarnación de SQL deberían ser su transformación en un lenguaje stand-alone (mientras ahora se usa como lenguaje hospedado en otros lenguajes) y la introducción de nuevos tipos de datos más complejos que permitan, por ejemplo, el tratamiento de datos multimediales.

Componentes del SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

Comandos

Existen dos tipos de comandos SQL:

- Los DDL que permiten crear y definir nuevas bases de datos, campos e índices.

- Los DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DDL:

Comando	Descripción
CREATE	Utilizado para crear nuevas tablas, campos e índices
DROP	Empleado para eliminar tablas e índices
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Comandos DML:

Comando	Descripción
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
INSERT	Utilizado para cargar lotes de datos en la base de 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 de una base de datos

Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Cláusula	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

Operadores lógicos

Operador	Uso
AND	Es el "y" lógico. Evalua dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de

	verdar si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

Operadores de Comparación

Operador	Uso
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor ó Igual que
>=	Mayor ó Igual que
=	Igual que
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en la comparación de un modelo
In	Utilizado para especificar registros de una base de datos

Funciones de Agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Función	Descripción
AVG	Utilizada para calcular el promedio de los valores de un campo determinado
COUNT	Utilizada para devolver el número de registros de la selección
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado
MAX	Utilizada para devolver el valor más alto de un campo especificado
MIN	Utilizada para devolver el valor más bajo de un campo especificado

Estructuras de las Tablas

Una base de datos en un sistema relacional está compuesta por un conjunto de tablas, que corresponden a las relaciones del modelo relacional. En la terminología usada en SQL no se alude a las relaciones, del mismo modo que no se usa el término atributo, pero sí la palabra columna, y no se habla de tupla, sino de línea.

Creación de Tablas Nuevas

```
CREATE TABLE tabla (campo1 tipo (tamaño) índice1 ,
campo2 tipo (tamaño) índice2 , ...,
índice multicampo , ... )
```

En donde:

Parte	Descripción
tabla	Es el nombre de la tabla que se va a crear.
campo1 campo2	Es el nombre del campo o de los campos que se van a crear en la nueva tabla. La nueva tabla debe contener, al menos, un campo.
tipo	Es el tipo de datos de campo en la nueva tabla. (Ver Tipos de Datos)
tamaño	Es el tamaño del campo sólo se aplica para campos de tipo texto.
índice1 índice2	Es una cláusula `CONSTRAINT` que define el tipo de indice a crear. Esta cláusula en opcional.
índice multicampos	Es una cláusula `CONSTRAINT` que define el tipo de indice multicampos a crear. Un índice multi campo es aquel que está indexado por el contenido de varios campos. Esta cláusula en opcional.

```
CREATE TABLE Empleados (Nombre TEXT (25) , Apellidos TEXT (50));
```

Crea una nueva tabla llamada Empleados con dos campos, uno llamado Nombre de tipo texto y longutid 25 y otro llamado apellidos con longitud 50.

```
CREATE TABLE Empleados (Nombre TEXT (10), Apellidos TEXT, Fecha_Nacimiento DATETIME) CONSTRAINT IndiceGeneral UNIQUE ([Nombre], [Apellidos], [Fecha_Nacimiento]);
```

Crea una nueva tabla llamada Empleados con un campo Nombre de tipo texto y longitud 10, otro con llamado Apellidos de tipo texto y longitud predeterminada (50) y uno más llamado Fecha_Nacimiento de tipo Fecha/Hora. También crea un índice único (no permite valores repetidos) formado por los tres campos.

```
CREATE TABLE Empleados (ID INTEGER CONSTRAINT IndicePrimario PRIMARY,
Nombre TEXT, Apellidos TEXT, Fecha_Nacimiento DATETIME);
```

Crea una tabla llamada Empleados con un campo Texto de longitud predeterminada (50) llamado Nombre y otro igual llamado Apellidos, crea otro campo llamado Fecha_Nacimiento de tipo Fecha/Hora y el campo ID de tipo entero el que establece como clave principal.

La cláusula CONSTRAINT

Se utiliza la cláusula `CONSTRAINT` en las instrucciones `ALTER TABLE` y `CREATE TABLE` para crear o eliminar índices. Existen dos sintaxis para esta cláusula dependiendo si desea Crear ó Eliminar un índice de un único campo o si se trata de un campo multiíndice. Si se utiliza el motor de datos de Microsoft, sólo podrá utilizar esta cláusula con las bases de datos propias de dicho motor.

Para los índices de campos únicos:

```
CONSTRAINT nombre {PRIMARY KEY | UNIQUE | REFERENCES tabla externa
[(campo externo1, campo externo2)]}
```

Para los índices de campos múltiples:

```
CONSTRAINT nombre {PRIMARY KEY (primario1[, primario2 [,
...]]) |
UNIQUE (único1[, único2 [, ...]]) |
FOREIGN KEY (ref1[, ref2 [, ...]]) REFERENCES tabla externa
[(campo externo1
[,campo externo2 [, ...]])]}
```

Parte	Descripción
nombre	Es el nombre del índice que se va a crear.
primarioN	Es el nombre del campo o de los campos que forman el índice primario.
únicoN	Es el nombre del campo o de los campos que forman el índice de clave única.
refN	Es el nombre del campo o de los campos que forman el índice externo (hacen referencia a campos de otra tabla).
tabla externa	Es el nombre de la tabla que contiene el campo o los campos referenciados en refN
campos externos	Es el nombre del campo o de los campos de la tabla externa especificados por ref1, ref2, ..., refN

Si se desea crear un índice para un campo cuando se esta utilizando las instrucciones `ALTER TABLE` o `CREATE TABLE` la cláusula `CONSTRAINT` debe aparecer inmediatamente después de la especificación del campo indexado.

Si se desea crear un índice con múltiples campos cuando se está utilizando las instrucciones `ALTER TABLE` o `CREATE TABLE` la cláusula `CONSTRAINT` debe aparecer fuera de la cláusula de creación de tabla.

Tipo de Indice	Descripción
UNIQUE	Genera un índece de clave única. Lo que implica que los registros de la tabla no pueden contener el mismo valor en los campos indexados.
PRIMARY KEY	Genera un índice primario el campo o los campos especificados. Todos los campos de la clave principal deben ser únicos y no nulos, cada tabla sólo puede contener una única clave principal.
FOREIGN KEY	Genera un índice externo (toma como valor del índice campos contenidos en otras tablas). Si la clave principal de la tabla externa consta de más de un campo, se debe utilizar una definición de índice de múltiples campos, listando todos los campos de referencia, el nombre de la tabla externa, y los nombres de los campos referenciados en la tabla externa en el mismo orden que los campos de referencia listados. Si los campos referenciados son la clave principal de la tabla externa, no tiene que especificar los campos referenciados, predeterminado por valor, el motor Jet se comporta como si la clave principal de la tabla externa fueran los campos referenciados.

Creación de Indices

Si se utiliza el motor de datos Jet de Microsoft sólo se pueden crear índices en bases de datos del mismo motor. La sintaxis para crear un índice en una tabla ya definida en la siguiente:

```
CREATE [ UNIQUE ] INDEX índice
ON tabla (campo [ASC|DESC][, campo [ASC|DESC], ...])
[WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]
```

En donde:

Parte	Descripción
índice	Es el nombre del índice a crear.
tabla	Es el nombre de una tabla existentes en la que se creará el índice.
campo	Es el nombre del campo o lista de campos que consituyen el índice.
ASC|DESC	Indica el orden de los valores de lso campos ASC indica un orden ascendente (valor predeterminado) y DESC un orden descendente.
UNIQUE	Indica que el indice no puede contener valores duplicados.
DISALLOW NULL	Prohibe valores nulos en el índice
IGNORE NULL	Excluye del índice los valores nulos incluidos en los campos que lo componen.
PRIMARY	Asigna al índice la categoría de clave principal, en cada tabla sólo puede existir un único indice que sea "Clave Principal". Si un índice es clave principal implica que que no puede contener valores nulos ni duplicados.

En el caso de ACCESS, se puede utilizar `CREATE INDEX` para crear un pseudo índice sobre una tabla adjunta en una fuente de datos ODBC tal como SQL Server que no tenga todavía un índice. No necesita permiso o tener acceso a un servidor remoto para crear un pseudo índice, además la base de datos remota no es consciente y no es afectada por el pseudo índice. Se utiliza la misma sintaxis para las tabla adjunta que para las originales. Esto es especialmente útil para crear un índice en una tabla que sería de sólo lectura debido a la falta de un índice.

```
CREATE INDEX MiIndice ON Empleados (Prefijo, Telefono);
```

Crea un índice llamado MiIndice en la tabla empleados con los campos Prefijo y Telefono.

```
CREATE UNIQUE INDEX MiIndice ON Empleados (ID) WITH
DISALLOW NULL;
```

Crea un índice en la tabla Empleados utilizando el campo ID, obligando que el campo ID no contenga valores nulos ni repetidos.

Modificar el Diseño de una Tabla

`ALTER TABLE` modifica el diseño de una tabla ya existente, se puden modificar los campos o los índices existentes. Su sintaxis es:

```
ALTER TABLE tabla {ADD {COLUMN tipo de campo[(tamaño)]
[CONSTRAINT índice]
CONSTRAINT índice multicampo} |
DROP {COLUMN campo I CONSTRAINT nombre del índice} }
```

En donde:

Parte	Descripción
tabla	Es el nombre de la tabla que se desea modificar.
campo	Es el nombre del campo que se va a añadir o eliminar.
tipo	Es el tipo de campo que se va a añadir.
tamaño	El el tamaño del campo que se va a añadir (sólo para campos de texto).
índice	Es el nombre del índice del campo (cuando se crean campos) o el nombre del índice de la tabla que se desea eliminar.
índice multicampo	Es el nombre del índice del campo multicampo (cuando se crean campos) o el nombre del índice de la tabla que se desea eliminar.

Operación	Descripción
ADD COLUMN	Se utiliza para añadir un nuevo campo a la tabla, indicando el nombre, el tipo de campo y opcionalmente el tamaño (para campos de tipo texto).
ADD	Se utliza para agregar un índice de multicampos o de un único campo.
DROP COLUMN	Se utliza para borrar un campo. Se especifica únicamente el nombre del campo.
DROP	Se utiliza para eliminar un índice. Se especifica únicamente el nombre del índice a continuación de la palabra reservada CONSTRAINT.

```
ALTER TABLE Empleados ADD COLUMN Salario CURRENCY;
```
Agrega un campo Salario de tipo Moneda a la tabla Empleados.

```
ALTER TABLE Empleados DROP COLUMN Salario;
```
Elimina el campo Salario de la tabla Empleados.

```
ALTER TABLE Pedidos ADD CONSTRAINT RelacionPedidos FOREIGN
KEY
(ID_Empleado) REFERENCES Empleados (ID_Empleado);
```
Agrega un indice externo a la tabla Pedidos. El índice externo se basa en el campo ID_Empleado y se refiere al campo ID_Empleado de la tabla Empleados. En este ejemplo no es necesario indicar el campo junto al nombre de la tabla en la cláusula REFERENCES, pues ID_Empleado es la clave principal de la tabla Empleados.

```
ALTER TABLE Pedidos DROP CONSTRAINT RelacionPedidos;
```
Elimina el índide de la tabla Pedidos.

Consultas de Selección

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros que se pueden almacenar en un objeto recordset. Este conjunto de registros es modificable.

Consultas Básicas

La sintaxis básica de una consulta de selección es la siguiente:

```
SELECT Campos FROM Tabla;
```

En donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos, por ejemplo:

```
SELECT Nombre, Telefono FROM Clientes;
```

Esta consulta devuelve un recordset con el campo nombre y teléfono de la tabla clientes.

Devolver Literales

En determinadas ocasiones nos puede interesar incluir una columna con un texto fijo en una consulta de selección, por ejemplo, supongamos que tenemos una tabla de empleados y deseamos recuperar las tarifas semanales de los electricistas, podríamos realizar la siguiente consulta:

```
SELECT
    Empleados.Nombre, 'Tarifa semanal: ',
Empleados.TarifaHora * 40
FROM
    Empleados
WHERE
    Empleados.Cargo = 'Electricista'
```

Ordenar los Registros

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la claúsula `ORDER BY Lista de Campos`. En donde `Lista de campos` representa los campos a ordenar. Ejemplo:

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER
BY Nombre;
```

Esta consulta devuelve los campos CodigoPostal, Nombre, Telefono de la tabla Clientes ordenados por el campo Nombre.

Se pueden ordenar los registros por mas de un campo, como por ejemplo:

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER
BY
CodigoPostal, Nombre;
```

Incluso se puede especificar el orden de los registros: ascendente mediante la claúsula `ASC` (se toma este valor por defecto) ó descendente (`DESC`).

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER
BY
CodigoPostal DESC , Nombre ASC;
```

Uso de Indices de las tablas

Si deseamos que la sentencia SQL utilice un índice para mostrar los resultados se puede utilizar la palabra reservada `INDEX` de la siguiente forma:

```
SELECT ... FROM Tabla (INDEX=Indice) ...
```

Normalmente los motores de las bases de datos deciden que indice se debe utilizar para la consulta, para ello utilizan criterios de rendimiento y sobre todo los campos de búsqueda especificados en la cláusula WHERE. Si se desea forzar a no utilizar ningún índice utilizaremos la siguiente sintaxis:

```
SELECT ... FROM Tabla (INDEX=0) ...
```

Consultas con Predicado

El predicado se incluye entre la claúsula y el primer nombre del campo a recuperar, los posibles predicados son:

Predicado	Descripción
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 basandose en la totalidad del registro y no sólo en los campos seleccionados.

ALL

Si no se incluye ninguno de los predicados se asume ALL. El Motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL. No se conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.

```
SELECT ALL FROM Empleados;
SELECT * FROM Empleados;
```

TOP

Devuelve un cierto número de registros que entran entre al principio o al final de un rango especificado por una cláusula `ORDER BY`. Supongamos que queremos recuperar los nombres de los 25 primeros estudiantes del curso 1994:

```
SELECT TOP 25 Nombre, Apellido FROM Estudiantes ORDER BY
Nota DESC;
```

Si no se incluye la cláusula `ORDER BY`, la consulta devolverá un conjunto arbitrario de 25 registros de la tabla Estudiantes. El predicado `TOP` no elige entre valores iguales. En el ejemplo anterior, si la nota media número 25 y la 26 son iguales, la consulta devolverá 26 registros. Se puede utilizar la palabra

reservada PERCENT para devolver un cierto porcentaje de registros que caen al principio o al final de un rango especificado por la cláusula ORDER BY. Supongamos que en lugar de los 25 primeros estudiantes deseamos el 10 por ciento del curso:

```
SELECT TOP 10 PERCENT Nombre, Apellido FROM Estudiantes
ORDER BY Nota DESC;
```

El valor que va a continuación de TOP debe ser un Integer sin signo. TOP no afecta a la posible actualización de la consulta.

DISTINCT

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos.

Por ejemplo, varios empleados listados en la tabla Empleados pueden tener el mismo apellido. Si dos registros contienen López en el campo Apellido, la siguiente instrucción SQL devuelve un único registro:

```
SELECT DISTINCT Apellido FROM Empleados;
```

Con otras palabras el predicado DISTINCT devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un contenido diferente. El resultado de una consulta que utiliza DISTINCT no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.

DISTINCTROW

Este predicado no es compatible con ANSI. Que yo sepa a día de hoy sólo funciona con ACCESS.

Devuelve los registros diferentes de una tabla; a diferencia del predicado anterior que sólo se fijaba en el contenido de los campos seleccionados, éste lo hace en el contenido del registro completo independientemente de los campo indicados en la cláusula SELECT.

```
SELECT DISTINCTROW Apellido FROM Empleados;
```

Si la tabla empleados contiene dos registros: Antonio López y Marta López, el ejemplo del predicado DISTINCT devuleve un único registro con el valor López en el campo Apellido ya que busca no duplicados en dicho campo. Este último ejemplo devuelve dos registros con el valor López en el apellido ya que se buscan no duplicados en el registro completo.

Alias

En determinadas circunstancias es necesario asignar un nombre a alguna columna determinada de un conjunto devuelto, otras veces por simple capricho o por otras circunstancias. Para resolver todas ellas tenemos la palabra reservada AS que se encarga de asignar el nombre que deseamos a la columna deseada. Tomado como referencia el ejemplo anterior podemos hacer que la columna devuelta por la consulta, en lugar de llamarse apellido (igual que el campo devuelto) se llame Empleado. En este caso procederíamos de la siguiente forma:

```
SELECT DISTINCTROW Apellido AS Empleado FROM Empleados;
```

AS no es una palabra reservada de ANSI, existen diferentes sistemas de asignar los alias en función del motor de bases de datos. En ORACLE para asignar un alias a un campo hay que hacerlo de la siguiente forma:

```
SELECT Apellido AS "Empleado" FROM Empleados;
```

También podemos asignar alias a las tablas dentro de la consulta de selección, en esta caso hay que tener en cuenta que en todas las referencias que deseemos hacer a dicha tabla se ha de utilizar el alias en lugar del nombre. Esta técnica será de gran utilidad más adelante cuando se estudien las vinculaciones entre tablas. Por ejemplo:

```
SELECT Apellido AS Empleado FROM Empleados AS Trabajadores;
```

Para asignar alias a las tablas en ORACLE y SQL-SERVER los alias se asignan escribiendo el nombre de la tabla, dejando un espacio en blanco y escribiendo el Alias (se asignan dentro de la cláusula `FROM`).

```
SELECT Trabajadores.Apellido AS Empleado FROM Empleados
Trabajadores;
```

Esta nomenclatura `[Tabla].[Campo]` se debe utilizar cuando se está recuperando un campo cuyo nombre se repite en varias de las tablas que se utilizan en la sentencia. No obstante cuando en la sentencia se emplean varias tablas es aconsejable utilizar esta nomenclatura para evitar el trabajo que supone al motor de datos averiguar en que tabla está cada uno de los campos indicados en la cláusula `SELECT`.

Recuperar Información de una base de Datos Externa

Para concluir este capítulo se debe hacer referencia a la recuperación de registros de bases de datos externa. En ocasiones es necesario la recuperación de información que se encuentra contenida en una tabla que no se encuentra en la base de datos que ejecutará la consulta o que en ese momento no se encuentra abierta, esta situación la podemos salvar con la palabra reservada `IN` de la siguiente forma:

```
SELECT DISTINCTROW Apellido AS Empleado FROM Empleados
IN 'c:\databases\gestion.mdb';
```

En donde `c:\databases\gestion.mdb` es la base de datos que contiene la tabla Empleados. Esta técnica es muy sencilla y común en bases de datos de tipo ACCESS en otros sistemas como SQL-SERVER u ORACLE, la cosa es más complicada la tener que existir relaciones de confianza entre los servidores o al ser necesaria la vinculación entre las bases de datos. Este ejemplo recupera la información de una base de datos de SQL-SERVER ubicada en otro servidor (se da por supuesto que los servidores están lincados):

```
SELECT Apellido FROM Servidor1.BaseDatos1.dbo.Empleados
```

Criterios de Selección

En el capítulo anterior se vio la forma de recuperar los registros de las tablas, las formas empleadas devolvían todos los registros de la mencionada tabla. A lo largo de este capítulo se estudiarán las posibilidades de filtrar los registros con el fin de recuperar solamente aquellos que cumplan una condiciones preestablecidas.

Antes de comenzar el desarrollo de este apartado hay que recalcar tres detalles de vital importancia. El primero de ellos es que cada vez que se desee establecer una condición referida a un campo de texto la condición de búsqueda debe ir encerrada entre comillas simples; la segunda es que no es posible establecer condiciones de búsqueda en los campos memo y; la tercera y última hace referencia a las fechas. A día de hoy no he sido capaz de encontrar una sintaxis que funcione en todos los sistemas, por lo que se hace necesario particularizarlas según el banco de datos:

Banco de Datos	Sintaxis	Ejemplo (para grabar la fecha 18 de mayo de 1969)
SQL-SERVER	Fecha = #mm-dd-aaaa#	Fecha = #05-18-1969# ó Fecha = 19690518

ORACLE	Fecha = to_date('YYYYDDMM','aaaammdd',)	Fecha = to_date('YYYYDDMM', '19690518')
ACCESS	Fecha = #mm-dd-aaaa#	Fecha = #05-18-1969#

Referente a los valores lógicos `True` o `False` cabe destacar que no son reconocidos en ORACLE, ni en este sistema de bases de datos ni en SQL-SERVER existen los campos de tipo `"SI/NO"` de ACCESS; en estos sistemas se utilizan los campos `BIT` que permiten almacenar valores de `0` ó `1`. Internamente, ACCESS, almacena en estos campos valores de `0` ó `-1`, así que todo se complica bastante, pero aprovechando la coincidencia del `0` para los valores `FALSE`, se puede utilizar la sintaxis siguiente que funciona en todos los casos: si se desea saber si el campo es falso `"... CAMPO = 0"` y para saber los verdaderos `"CAMPO <> 0"`.

Operadores Lógicos

Los operadores lógicos soportados por SQL son: `AND`, `OR`, `XOR`, `Eqv`, `Imp`, `Is` y `Not`. A excepción de los dos últimos todos poseen la siguiente sintaxis:

```
<expresión1>    operador    <expresión2>
```

En donde `expresión1` y `expresión2` son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico. La tabla adjunta muestra los diferentes posibles resultados:

<expresión1>	Operador	<expresión2>	Resultado
Verdad	AND	Falso	Falso
Verdad	AND	Verdad	Verdad
Falso	AND	Verdad	Falso
Falso	AND	Falso	Falso
Verdad	OR	Falso	Verdad
Verdad	OR	Verdad	Verdad
Falso	OR	Verdad	Verdad
Falso	OR	Falso	Falso
Verdad	XOR	Verdad	Falso
Verdad	XOR	Falso	Verdad
Falso	XOR	Verdad	Verdad
Falso	XOR	Falso	Falso
Verdad	Eqv	Verdad	Verdad
Verdad	Eqv	Falso	Falso
Falso	Eqv	Verdad	Falso
Falso	Eqv	Falso	Verdad
Verdad	Imp	Verdad	Verdad
Verdad	Imp	Falso	Falso

Verdad	Imp	Null	Null
Falso	Imp	Verdad	Verdad
Falso	Imp	Falso	Verdad
Falso	Imp	Null	Verdad
Null	Imp	Verdad	Verdad
Null	Imp	Falso	Null
Null	Imp	Null	Null

Si a cualquiera de las anteriores condiciones le anteponemos el operador `NOT` el resultado de la operación será el contrario al devuelto sin el operador `NOT`.

El último operador denominado `IS` se emplea para comparar dos variables de tipo objeto `<Objeto1> Is <Objeto2>`. Este operador devuelve verdad si los dos objetos son iguales.

```
SELECT * FROM Empleados WHERE Edad > 25 AND Edad < 50;
SELECT * FROM Empleados WHERE (Edad > 25 AND Edad < 50) OR
Sueldo = 100;
SELECT * FROM Empleados WHERE NOT Estado = 'Soltero';
SELECT * FROM Empleados WHERE (Sueldo > 100 AND Sueldo <
500) OR
(Provincia = 'Madrid' AND Estado = 'Casado');
```

Intervalos de Valores

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador Between cuya sintaxis es:

```
campo [Not] Between valor1 And valor2 (la condición Not es
opcional)
```

En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo.

```
SELECT * FROM Pedidos WHERE CodPostal Between 28000 And
28999;
(Devuelve los pedidos realizados en la provincia de Madrid)
SELECT IIf(CodPostal Between 28000 And 28999, 'Provincial',
'Nacional')
FROM Editores;
(Devuelve el valor 'Provincial' si el código postal se
encuentra en el intervalo,
'Nacional' en caso contrario)
```

El Operador Like

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

```
expresión Like modelo
```

En donde expresión es una cadena modelo o campo contra el que se compara expresión. Se puede utilizar el operador `Like` para encontrar valores en los campos que coincidan con el modelo especificado. Por modelo puede especificar un valor completo (Ana María), o se pueden utilizar caracteres comodín como los reconocidos por el sistema operativo para encontrar un rango de valores (`Like An*`).

El operador `Like` se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, si introduce `Like C*` en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra C. En una consulta con parámetros, puede hacer que el usuario escriba el modelo que se va a utilizar.

El ejemplo siguiente devuelve los datos que comienzan con la letra P seguido de cualquier letra entre A y F y de tres dígitos:

```
Like 'P[A-F]###'
```

Este ejemplo devuelve los campos cuyo contenido empiece con una letra de la A a la D seguidas de cualquier cadena.

```
Like '[A-D]*'
```

En la tabla siguiente se muestra cómo utilizar el operador `Like` para comprobar expresiones con diferentes modelos.

Tipo de coincidencia	Modelo Planteado	Coincide	No coincide
Varios caracteres	'a*a'	'aa', 'aBa', 'aBBBa'	'aBC'
Carácter especial	'a[*]a'	'a*a'	'aaa'
Varios caracteres	'ab*'	'abcdefg', 'abc'	'cab', 'aab'
Un solo carácter	'a?a'	'aaa', 'a3a', 'aBa'	'aBBBa'
Un solo dígito	'a#a'	'a0a', 'a1a', 'a2a'	'aaa', 'a10a'
Rango de caracteres	'[a-z]'	'f', 'p', 'j'	'2', '&'
Fuera de un rango	'[!a-z]'	'9', '&', '%'	'b', 'a'
Distinto de un dígito	'[!0-9]'	'A', 'a', '&', '~'	'0', '1', '9'
Combinada	'a[!b-m]#'	'An9', 'az0', 'a99'	'abc', 'aj0'

En determinado motores de bases de datos, esta cláusula, no reconoce el asterisco como carácter comodín y hay que sustituirlo por el carácter tanto por ciento (`%`). Por ejemplo, en SQL-SERVER:

Ejemplo	Descripción
LIKE 'A%'	Todo lo que comience por A
LIKE '_NG'	Todo lo que comience por cualquier carácter y luego siga NG
LIKE '[AF]%'	Todo lo que comience por A ó F
LIKE '[A-F]%'	Todo lo que comience por cualquier letra comprendida entre la A y la F
LIKE '[A^B]%'	Todo lo que comience por A y la segunda letra no sea una B

El Operador In

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los en una lista. Su sintaxis es:

```
expresión [Not] In(valor1, valor2, . . .)

SELECT * FROM Pedidos WHERE Provincia In ('Madrid',
'Barcelona', 'Sevilla');
```

La cláusula WHERE

La cláusula `WHERE` puede usarse para determinar qué registros de las tablas enumeradas en la cláusula `FROM` aparecerán en los resultados de la instrucción `SELECT`. Depués de escribir esta cláusula se deben especificar las condiciones expuestas en los dos primeros apartados de este capítulo. Si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla. `WHERE` es opcional, pero cuando aparece debe ir a continuación de `FROM`.

```
SELECT Apellidos, Salario FROM Empleados WHERE Salario >
21000;

SELECT Id_Producto, Existencias FROM Productos
WHERE Existencias <= Nuevo_Pedido;

SELECT * FROM Pedidos WHERE Fecha_Envio = #5/10/94#;

SELECT Apellidos, Nombre FROM Empleados WHERE Apellidos =
'King';

SELECT Apellidos, Nombre FROM Empleados WHERE Apellidos
Like 'S*';

SELECT Apellidos, Salario FROM Empleados WHERE Salario
Between 200 And 300;

SELECT Apellidos, Salario FROM Empl WHERE Apellidos Between
'Lon' And 'Tol';

SELECT Id_Pedido, Fecha_Pedido FROM Pedidos WHERE
Fecha_Pedido
Between #1-1-94# And  #30-6-94#;

SELECT Apellidos, Nombre, Ciudad FROM Empleados WHERE
Ciudad
In ('Sevilla', 'Los Angeles', 'Barcelona');
```

Agrupamiento de Registros y Funciones Agregadas

La cláusula GROUP BY

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro se crea un valor sumario si se incluye una función SQL agregada, como por ejemplo `Sum` o `Count`, en la instrucción `SELECT`. Su sintaxis es:

```
SELECT campos FROM tabla WHERE criterio GROUP BY campos del grupo
```

`GROUP BY` es opcional. Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción `SELECT`. Los valores Null en los campos `GROUP BY` se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas.

Se utiliza la cláusula `WHERE` para excluir aquellas filas que no desea agrupar, y la cláusula `HAVING` para filtrar los registros una vez agrupados.

A menos que contenga un dato Memo u Objeto OLE , un campo de la lista de campos `GROUP BY` puede referirse a cualquier campo de las tablas que aparecen en la cláusula `FROM`, incluso si el campo no esta incluido en la instrucción `SELECT`, siempre y cuando la instrucción `SELECT` incluya al menos una función SQL agregada.

Todos los campos de la lista de campos de `SELECT` deben o bien incluirse en la cláusula `GROUP BY` o como argumentos de una función SQL agregada.

```
SELECT Id_Familia, Sum(Stock) FROM Productos GROUP BY Id_Familia;
```

Una vez que `GROUP BY` ha combinado los registros, `HAVING` muestra cualquier registro agrupado por la cláusula `GROUP BY` que satisfaga las condiciones de la cláusula `HAVING`.

`HAVING` es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando `GROUP BY`, `HAVING` determina cuales de ellos se van a mostrar.

```
SELECT Id_Familia Sum(Stock) FROM Productos GROUP BY Id_Familia
HAVING Sum(Stock) > 100 AND NombreProducto Like BOS*;
```

AVG (Media Aritmética)

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta. Su sintaxis es la siguiente:

```
Avg(expr)
```

En donde `expr` representa el campo que contiene los datos numéricos para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por `Avg` es la media aritmética (la suma de los valores dividido por el número de valores). La función `Avg` no incluye ningún campo Null en el cálculo.

```
SELECT Avg(Gastos) AS Promedio FROM Pedidos WHERE Gastos > 100;
```

Count (Contar Registros)

Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente:

```
Count(expr)
```

En donde `expr` contiene el nombre del campo que desea contar. Los operandos de `expr` pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). Puede contar cualquier tipo de datos incluso texto.

Aunque `expr` puede realizar un cálculo sobre un campo, `Count` simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función `Count` no cuenta los registros que tienen campos null a menos que `expr` sea el carácter comodín asterisco (*). Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos null. `Count(*)` es considerablemente más rápida que `Count(Campo)`. No se debe poner el asterisco entre dobles comillas ('*').

```
SELECT Count(*) AS Total  FROM Pedidos;
```

Si `expr` identifica a múltiples campos, la función `Count` cuenta un registro sólo si al menos uno de los campos no es Null. Si todos los campos especificados son Null, no se cuenta el registro. Hay que separar los nombres de los campos con ampersand (&).

```
SELECT Count(FechaEnvío & Transporte) AS Total FROM
Pedidos;
```

Podemos hacer que el gestor cuente los datos diferentes de un determinado campo

```
SELECT Count(DISTINCT Localidad) AS Total FROM Pedidos;
```

Max y Min (Valores Máximos y Mínimos)

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo especifico de una consulta. Su sintaxis es:

```
Min(expr)
Max(expr)
```

En donde `expr` es el campo sobre el que se desea realizar el cálculo. Expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

```
SELECT Min(Gastos) AS ElMin FROM Pedidos WHERE Pais =
'España';
SELECT Max(Gastos) AS ElMax FROM Pedidos WHERE Pais  =
'España';
```

StDev y StDevP (Desviación Estándar)

Devuelve estimaciones de la desviación estándar para la población (el total de los registros de la tabla) o una muestra de la población representada (muestra aleatoria) . Su sintaxis es:

```
StDev(expr)
StDevP(expr)
```

En donde `expr` representa el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de `expr` pueden

incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

`StDevP` evalúa una población, y `StDev` evalúa una muestra de la población. Si la consulta contiene menos de dos registros (o ningún registro para `StDevP`), estas funciones devuelven un valor Null (el cual indica que la desviación estándar no puede calcularse).

```
SELECT StDev(Gastos) AS Desviacion FROM Pedidos WHERE Pais
= 'España';
SELECT StDevP(Gastos) AS Desviacion FROM Pedidos WHERE
Pais= 'España';
```

Sum (Sumar Valores)

Devuelve la suma del conjunto de valores contenido en un campo especifico de una consulta. Su sintaxis es:

```
Sum(expr)
```

En donde `expr` respresenta el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de `expr` pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

```
SELECT Sum(PrecioUnidad * Cantidad) AS Total FROM
DetallePedido;
```

Var y VarP (Varianza)

Devuelve una estimación de la varianza de una población (sobre el total de los registros) o una muestra de la población (muestra aleatoria de registros) sobre los valores de un campo. Su sintaxis es:

```
Var(expr)
VarP(expr)
```

`VarP` evalúa una población, y `Var` evalúa una muestra de la población. `Expr` el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

Si la consulta contiene menos de dos registros, `Var` y `VarP` devuelven Null (esto indica que la varianza no puede calcularse). Puede utilizar `Var` y `VarP` en una expresión de consulta o en una Instrucción SQL.

```
SELECT Var(Gastos) AS Varianza FROM Pedidos WHERE Pais =
'España';
SELECT VarP(Gastos) AS Varianza FROM Pedidos WHERE Pais =
'España';
```

COMPUTE de SQL-SERVER

Esta cláusula añade una fila en el conjunto de datos que se está recuperando, se utiliza para realizar cálculos en campos numéricos. `COMPUTE` actúa siempre sobre un campo o expresión del conjunto de

resultados y esta expresión debe figurar exactamente igual en la cláusula SELECT y siempre se debe ordenar el resultado por la misma o al menos agrupar el resultado. Esta expresión no puede utilizar ningún ALIAS.

```
SELECT IdCliente, Count(IdPedido) FROM Pedidos
GROUP BY IdPedido HAVING Count(IdPedido) > 20 COMPUTE
Sum(Count(IdPedido))

SELECT IdPedido, (PrecioUnidad * Cantidad - Descuento) FROM
[Detalles de Pedidos]
ORDER BY IdPedido
COMPUTE  Sum((PrecioUnidad * Cantidad - Descuento)) //
Calcula el Total
BY IdPedido // Calcula el Subtotal
```

Consultas de Acción

Las consultas de acción son aquellas que no devuelven ningún registro, son las encargadas de acciones como añadir y borrar y modificar registros.

DELETE

Crea una consulta de eliminación que elimina los registros de una o más de las tablas listadas en la cláusula FROM que satisfagan la cláusula WHERE. Esta consulta elimina los registros completos, no es posible eliminar el contenido de algún campo en concreto. Su sintaxis es:

```
DELETE FROM Tabla WHERE criterio
```

Una vez que se han eliminado los registros utilizando una consulta de borrado, no puede deshacer la operación. Si desea saber qué registros se eliminarán, primero examine los resultados de una consulta de selección que utilice el mismo criterio y después ejecute la consulta de borrado. Mantenga copias de seguridad de sus datos en todo momento. Si elimina los registros equivocados podrá recuperarlos desde las copias de seguridad.

```
DELETE * FROM Empleados WHERE Cargo = 'Vendedor';
```

INSERT INTO

Agrega un registro en una tabla. Se la conoce como una consulta de datos añadidos. Esta consulta puede ser de dos tipos: Insertar un único registro ó Insertar en una tabla los registros contenidos en otra tabla.

Insertar un único Registro

En este caso la sintaxis es la siguiente:

```
INSERT INTO Tabla (campo1, campo2, .., campoN)
VALUES (valor1, valor2, ..., valorN)
```

Esta consulta graba en el campo1 el valor1, en el campo2 y valor2 y así sucesivamente. Hay que prestar especial atención a acotar entre comillas simples (') los valores literales (cadenas de caracteres) y las fechas indicarlas en formato mm-dd-aa y entre caracteres de almohadillas (#).

Para seleccionar registros e insertarlos en una tabla nueva

En este caso la sintaxis es la siguiente:

```
    SELECT campo1, campo2, ..., campoN INTO nuevatabla
FROM tablaorigen [WHERE criterios]
```

Se pueden utilizar las consultas de creación de tabla para archivar registros, hacer copias de seguridad de las tablas o hacer copias para exportar a otra base de datos o utilizar en informes que muestren los datos de un periodo de tiempo concreto. Por ejemplo, se podría crear un informe de Ventas mensuales por región ejecutando la misma consulta de creación de tabla cada mes.

Insertar Registros de otra Tabla

En este caso la sintaxis es:

```
INSERT INTO Tabla [IN base_externa] (campo1, campo2, ...,
campoN)
SELECT TablaOrigen.campo1, TablaOrigen.campo2, ...,
TablaOrigen.campoN
FROM TablaOrigen
```

En este caso se seleccionarán los campos 1,2, ..., n de la tabla origen y se grabarán en los campos 1,2,.., n de la Tabla. La condición `SELECT` puede incluir la cláusula `WHERE` para filtrar los registros a copiar. Si Tabla y TablaOrigen poseen la misma estrucutra podemos simplificar la sintaxis a:

```
INSERT INTO Tabla  SELECT TablaOrigen.* FROM TablaOrigen
```

De esta forma los campos de TablaOrigen se grabarán en Tabla, para realizar esta operación es necesario que todos los campos de TablaOrigen estén contenidos con igual nombre en Tabla. Con otras palabras que Tabla posea todos los campos de TablaOrigen (igual nombre e igual tipo).

En este tipo de consulta hay que tener especial atención con los campos contadores o autonuméricos puesto que al insertar un valor en un campo de este tipo se escribe el valor que contenga su campo homólogo en la tabla origen, no incrementandose como le corresponde.

Se puede utilizar la instrucción `INSERT INTO` para agregar un registro único a una tabla, utilizando la sintaxis de la consulta de adición de registro único tal y como se mostró anteriormente. En este caso, su código específica el nombre y el valor de cada campo del registro. Debe especificar cada uno de los campos del registro al que se le va a asignar un valor así como el valor para dicho campo. Cuando no se especifica dicho campo, se inserta el valor predeterminado o Null. Los registros se agregan al final de la tabla.

También se puede utilizar `INSERT INTO` para agregar un conjunto de registros pertenecientes a otra tabla o consulta utilizando la cláusula `SELECT ... FROM` como se mostró anteriormente en la sintaxis de la consulta de adición de múltiples registros. En este caso la cláusula `SELECT` especifica los campos que se van a agregar en la tabla destino especificada.

La tabla destino u origen puede especificar una tabla o una consulta. Si la tabla destino contiene una clave principal, hay que segurarse que es única, y con valores no-Null ; si no es así, no se agregarán los registros. Si se agregan registros a una tabla con un campo Contador , no se debe incluir el campo Contador en la consulta. Se puede emplear la cláusula `IN` para agregar registros a una tabla en otra base de datos.

Se pueden averiguar los registros que se agregarán en la consulta ejecutando primero una consulta de selección que utilice el mismo criterio de selección y ver el resultado. Una consulta de adición copia los registros de una o más tablas en otra. Las tablas que contienen los registros que se van a agregar no se verán afectadas por la consulta de adición. En lugar de agregar registros existentes en otra tabla, se puede especificar los valores de cada campo en un nuevo registro utilizando la cláusula `VALUES`. Si se omite la lista de campos, la cláusula `VALUES` debe incluir un valor para cada campo de la tabla, de otra forma fallará `INSERT`.

```
INSERT INTO Clientes SELECT Clientes_Viejos.* FROM
Clientes_Nuevos;

SELECT Empleados.* INTO Programadores FROM  Empleados
WHERE Categoria = 'Programador'

INSERT INTO Empleados (Nombre, Apellido, Cargo)
VALUES ('Luis', 'Sánchez', 'Becario');

INSERT INTO Empleados SELECT Vendedores.* FROM Vendedores
WHERE Fecha_Contratacion < Now() - 30;
```

UPDATE

Crea una consulta de actualización que cambia los valores de los campos de una tabla especificada basándose en un criterio específico. Su sintaxis es:

```
UPDATE Tabla SET Campo1=Valor1, Campo2=Valor2, ...
CampoN=ValorN
WHERE Criterio;
```

`UPDATE` es especialmente útil cuando se desea cambiar un gran número de registros o cuando éstos se encuentran en múltiples tablas. Puede cambiar varios campos a la vez. El ejemplo siguiente incrementa los valores Cantidad pedidos en un 10 por ciento y los valores Transporte en un 3 por ciento para aquellos que se hayan enviado al Reino Unido:

```
UPDATE Pedidos SET Pedido = Pedidos * 1.1, Transporte =
Transporte * 1.03
WHERE PaisEnvío = 'ES';
```

`UPDATE` no genera ningún resultado. Para saber qué registros se van a cambiar, hay que examinar primero el resultado de una consulta de selección que utilice el mismo criterio y después ejecutar la consulta de actualización.

```
UPDATE Empleados SET Grado = 5 WHERE Grado = 2;

UPDATE Productos SET Precio = Precio  * 1.1 WHERE Proveedor
= 8 AND Familia = 3;
```

Si en una consulta de actualización suprimimos la cláusula

`WHERE` todos los registros de la tabla señalada serán actualizados.

```
UPDATE Empleados SET Salario = Salario * 1.1
```

Tipos de datos

Los tipos de datos SQL se clasifican en 13 tipos de datos primarios y de varios sinónimos válidos reconocidos por dichos tipos de datos.

Tipos de datos primarios:

Tipo de	Longitud	Descripción

Datos		
BINARY	1 byte	Para consultas sobre tabla adjunta de productos de bases de datos que definen un tipo de datos Binario.
BIT	1 byte	Valores Si/No ó True/False
BYTE	1 byte	Un valor entero entre 0 y 255.
COUNTER	4 bytes	Un número incrementado automáticamente (de tipo Long)
CURRENCY	8 bytes	Un entero escalable entre 922.337.203.685.477,5808 y 922.337.203.685.477,5807.
DATETIME	8 bytes	Un valor de fecha u hora entre los años 100 y 9999.
SINGLE	4 bytes	Un valor en punto flotante de precisión simple con un rango de $-3.402823*10^{38}$a $-1.401298*10^{-45}$para valores negativos, $1.401298*10^{-45}$a $3.402823*10^{38}$para valores positivos, y 0.
DOUBLE	8 bytes	Un valor en punto flotante de doble precisión con un rango de $-1.79769313486232*10^{308}$a $-4.94065645841247*10^{-324}$para valores negativos, $4.94065645841247*10^{-324}$a $1.79769313486232*10^{308}$para valores positivos, y 0.
SHORT	2 bytes	Un entero corto entre -32,768 y 32,767.
LONG	4 bytes	Un entero largo entre -2,147,483,648 y 2,147,483,647.
LONGTEXT	1 byte por carácter	De cero a un máximo de 1.2 gigabytes.
LONGBINARY	Según se necesite	De cero 1 gigabyte. Utilizado para objetos OLE.
TEXT	1 byte por caracter	De cero a 255 caracteres.

La siguiente tabla recoge los sinonimos de los tipos de datos definidos:

Tipo de Dato	Sinónimos
BINARY	VARBINARY
BIT	BOOLEAN LOGICAL LOGICAL1 YESNO
BYTE	INTEGER1
COUNTER	AUTOINCREMENT
CURRENCY	MONEY

DATETIME	DATE TIME TIMESTAMP
SINGLE	FLOAT4 IEEESINGLE REAL
DOUBLE	FLOAT FLOAT8 IEEEDOUBLE NUMBER NUMERIC
SHORT	INTEGER2 SMALLINT
LONG	INT INTEGER INTEGER4
LONGBINARY	GENERAL OLEOBJECT
LONGTEXT	LONGCHAR MEMO NOTE
TEXT	ALPHANUMERIC CHAR CHARACTER STRING VARCHAR
VARIANT (No Admitido)	VALUE

Subconsultas

Una subconsulta es una instrucción `SELECT` anidada dentro de una instrucción `SELECT`, `SELECT...INTO`, `INSERT...INTO`, `DELETE`, o `UPDATE` o dentro de otra subconsulta.

Puede utilizar tres formas de sintaxis para crear una subconsulta:

```
comparación [ANY | ALL | SOME] (instrucción sql)
expresión [NOT] IN (instrucción sql)
[NOT] EXISTS (instrucción sql)
```

En donde:

comparación
: Es una expresión y un operador de comparación que compara la expresión con el resultado de la subconsulta.

expresión
: Es una expresión por la que se busca el conjunto resultante de la subconsulta.

instrucción sql
: Es una instrucción `SELECT`, que sigue el mismo formato y reglas que cualquier otra instrucción `SELECT`. Debe ir entre paréntesis.

Se puede utilizar una subconsulta en lugar de una expresión en la lista de campos de una instrucción SELECT o en una cláusula WHERE o HAVING. En una subconsulta, se utiliza una instrucción SELECT para proporcionar un conjunto de uno o más valores especificados para evaluar en la expresión de la cláusula WHERE o HAVING.

Se puede utilizar el predicado ANY o SOME, los cuales son sinónimos, para recuperar registros de la consulta principal, que satisfagan la comparación con cualquier otro registro recuperado en la subconsulta. El ejemplo siguiente devuelve todos los productos cuyo precio unitario es mayor que el de cualquier producto vendido con un descuento igual o mayor al 25 por ciento:

```
SELECT * FROM Productos WHERE PrecioUnidad > ANY
(SELECT PrecioUnidad FROM DetallePedido WHERE Descuento >= 0 .25);
```

El predicado ALL se utiliza para recuperar únicamente aquellos registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la subconsulta. Si se cambia ANY por ALL en el ejemplo anterior, la consulta devolverá únicamente aquellos productos cuyo precio unitario sea mayor que el de todos los productos vendidos con un descuento igual o mayor al 25 por ciento. Esto es mucho más restrictivo.

El predicado IN se emplea para recuperar únicamente aquellos registros de la consulta principal para los que algunos registros de la subconsulta contienen un valor igual. El ejemplo siguiente devuelve todos los productos vendidos con un descuento igual o mayor al 25 por ciento:

```
SELECT * FROM Productos WHERE IDProducto IN
(SELECT IDProducto FROM DetallePedido WHERE Descuento >=  0.25);
```

Inversamente se puede utilizar NOT IN para recuperar únicamente aquellos registros de la consulta principal para los que no hay ningún registro de la subconsulta que contenga un valor igual.

El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro. Supongamos que deseamos recuperar todos aquellos clientes que hayan realizado al menos un pedido:

```
SELECT Clientes.Compañía, Clientes.Teléfono FROM Clientes WHERE EXISTS
(SELECT FROM Pedidos WHERE Pedidos.IdPedido = Clientes.IdCliente)
```

Esta consulta es equivalente a esta otra:

```
SELECT Clientes.Compañía, Clientes.Teléfono FROM Clientes WHERE
IdClientes IN
(SELECT Pedidos.IdCliente FROM Pedidos)
```

Se puede utilizar también alias del nombre de la tabla en una subconsulta para referirse a tablas listadas en la cláusula FROM fuera de la subconsulta. El ejemplo siguiente devuelve los nombres de los empleados cuyo salario es igual o mayor que el salario medio de todos los empleados con el mismo título. A la tabla Empleados se le ha dado el alias T1:

```
SELECT Apellido, Nombre, Titulo, Salario FROM Empleados AS T1
WHERE Salario >= (SELECT Avg(Salario) FROM Empleados
WHERE T1.Titulo = Empleados.Titulo) ORDER BY Titulo;
```

En el ejemplo anterior , la palabra reservada AS es opcional. Otros ejemplos:

```
SELECT Apellidos, Nombre, Cargo, Salario FROM Empleados
WHERE Cargo LIKE "Agente Ven*" AND Salario > ALL (SELECT Salario FROM
Empleados WHERE (Cargo LIKE "*Jefe*") OR (Cargo LIKE "*Director*"));
```

Obtiene una lista con el nombre, cargo y salario de todos los agentes de ventas cuyo salario es mayor que el de todos los jefes y directores.

```
SELECT DISTINCTROW NombreProducto, Precio_Unidad FROM Productos
```

```
WHERE (Precio_Unidad = (SELECT Precio_Unidad FROM Productos WHERE
Nombre_Producto = "Almíbar anisado");
```

Obtiene una lista con el nombre y el precio unitario de todos los productos con el mismo precio que el almíbar anisado.

```
SELECT DISTINCTROW Nombre_Contacto, Nombre_Compañia, Cargo_Contacto,
Telefono FROM Clientes WHERE (ID_Cliente IN (SELECT DISTINCTROW
ID_Cliente FROM Pedidos WHERE Fecha_Pedido >= #04/1/93# <#07/1/93#);
```

Obtiene una lista de las compañías y los contactos de todos los clientes que han realizado un pedido en el segundo trimestre de 1993.

```
SELECT Nombre, Apellidos FROM Empleados AS E WHERE EXISTS
(SELECT * FROM Pedidos AS O WHERE O.ID_Empleado = E.ID_Empleado);
```

Selecciona el nombre de todos los empleados que han reservado al menos un pedido.

```
SELECT DISTINCTROW Pedidos.Id_Producto, Pedidos.Cantidad,
(SELECT DISTINCTROW Productos.Nombre FROM Productos WHERE
Productos.Id_Producto = Pedidos.Id_Producto) AS ElProducto FROM
Pedidos WHERE Pedidos.Cantidad > 150 ORDER BY Pedidos.Id_Producto;
```

Recupera el Código del Producto y la Cantidad pedida de la tabla pedidos, extrayendo el nombre del producto de la tabla de productos.

```
SELECT NumVuelo, Plazas FROM Vuelos
WHERE Origen = 'Madrid' AND Exists
(SELECT T1.NumVuelo FROM Vuelos AS T1
 WHERE T1.PlazasLibres > 0 AND T1.NumVuelo=Vuelos.NumVuelo)
```

Recupera números de vuelo y capacidades de aquellos vuelos con destino Madrid y plazas libres.

Supongamos ahora que tenemos una tabla con los identificadores de todos nuestros productos y el stock de cada uno de ellos. En otra tabla se encuentran todos los pedidos que tenemos pendientes de servir. Se trata de averiguar que productos no se podemos servir por falta de stock.

```
SELECT PedidosPendientes.Nombre FROM PedidosPendientes
GROUP BY PedidosPendientes.Nombre
HAVING SUM(PedidosPendientes.Cantidad <
(SELECT Productos.Stock FROM Productos
 WHERE Productos.IdProducto = PedidosPendientes.IdProducto));
```

Supongamos que en nuestra tabla de empleados deseamos buscar todas las mujeres cuya edad sea mayor a la de cualquier hombre:

```
SELECT Empleados.Nombre FROM Empleados
WHERE Sexo = 'M' AND Edad > ANY
(SELECT Empleados.Edad FROM Empleados WHERE Sexo ='H')
```

ó lo que sería lo mismo:

```
SELECT Empleados.Nombre FROM Empleados WHERE Sexo = 'M' AND Edad >
(SELECT Max( Empleados.Edad )FROM Empleados WHERE Sexo ='H')
```

La siguiente tabla muestra algún ejemplo del operador `ANY` y `ALL`

Valor 1	Operador	Valor 2	Resultado
3	> ANY	(2,5,7)	Cierto
3	= ANY	(2,5,7)	Falso
3	= ANY	(2,3,5,7)	Cierto
3	> ANY	(2,5,7)	Falso
3	< ANY	(5,6,7)	Falso

El operacion =ANY es equivalente al operador IN, ambos devuelven el mismo resultado.

Consultas de Unión Internas

Consultas de Combinación entre tablas

Las vinculaciones entre tablas se realiza mediante la cláusula INNER que combina registros de dos tablas siempre que haya concordancia de valores en un campo común. Su sintaxis es:

```
SELECT campos FROM tb1 INNER JOIN tb2 ON tb1.campo1 comp
tb2.campo2
```

En donde:

tb1, tb2
: Son los nombres de las tablas desde las que se combinan los registros.

campo1, campo2
: Son los nombres de los campos que se combinan. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos, pero no tienen que tener el mismo nombre.

comp
: Es cualquier operador de comparación relacional : =, <, >, <=, >=, o <>.

Se puede utilizar una operación INNER JOIN en cualquier cláusula FROM. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones Equi son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas. Se puede utilizar INNER JOIN con las tablas Departamentos y Empleados para seleccionar todos los empleados de cada departamento. Por el contrario, para seleccionar todos los departamentos (incluso si alguno de ellos no tiene ningún empleado asignado) se emplea LEFT JOIN o todos los empleados (incluso si alguno no está asignado a ningún departamento), en este caso RIGHT JOIN.

Si se intenta combinar campos que contengan datos Memo u Objeto OLE, se produce un error. Se pueden combinar dos campos numéricos cualesquiera, incluso si son de diferente tipo de datos. Por ejemplo, puede combinar un campo Numérico para el que la propiedad Size de su objeto Field está establecida como Entero, y un campo Contador.

El ejemplo siguiente muestra cómo podría combinar las tablas Categorías y Productos basándose en el campo IDCategoria:

```
SELECT Nombre_Categoría, NombreProducto
FROM Categorias INNER JOIN Productos
ON Categorias.IDCategoria = Productos.IDCategoria;
```

En el ejemplo anterior, IDCategoria es el campo combinado, pero no está incluido en la salida de la consulta ya que no está incluido en la instrucción SELECT. Para incluir el campo combinado, incluir el nombre del campo en la instrucción SELECT, en este caso, Categorias.IDCategoria.

También se pueden enlazar varias cláusulas ON en una instrucción JOIN, utilizando la sintaxis siguiente:

```
SELECT campos
FROM tabla1 INNER JOIN tabla2
ON tb1.campo1 comp tb2.campo1 AND
ON tb1.campo2 comp tb2.campo2) OR
ON tb1.campo3 comp tb2.campo3)];
```

También puede anidar instrucciones JOIN utilizando la siguiente sintaxis:

```
SELECT campos
FROM tb1 INNER JOIN
(tb2 INNER JOIN [( ]tb3
[INNER JOIN [( ]tablax [INNER JOIN ...)]
ON tb3.campo3 comp tbx.campox)]
ON tb2.campo2 comp tb3.campo3)
ON tb1.campo1 comp tb2.campo2;
```

Un LEFT JOIN o un RIGHT JOIN puede anidarse dentro de un INNER JOIN, pero un INNER JOIN no puede anidarse dentro de un LEFT JOIN o un RIGHT JOIN.

Por ejemplo:

```
SELECT DISTINCTROW Sum([Precio unidad] * [Cantidad]) AS
[Ventas],
[Nombre] & " " & [Apellidos] AS [Nombre completo] FROM
[Detalles de pedidos],
Pedidos, Empleados, Pedidos INNER JOIN [Detalles de
pedidos] ON Pedidos.
[ID de pedido] = [Detalles de pedidos].[ID de pedido],
Empleados INNER JOIN
Pedidos ON Empleados.[ID de empleado] = Pedidos.[ID de
empleado] GROUP BY
[Nombre] & " " & [Apellidos];
```

Crea dos combinaciones equivalentes: una entre las tablas Detalles de pedidos y Pedidos, y la otra entre las tablas Pedidos y Empleados. Esto es necesario ya que la tabla Empleados no contiene datos de ventas y la tabla Detalles de pedidos no contiene datos de los empleados. La consulta produce una lista de empleados y sus ventas totales.

Si empleamos la cláusula INNER en la consulta se seleccionarán sólo aquellos registros de la tabla de la que hayamos escrito a la izquierda de INNER JOIN que contengan al menos un registro de la tabla que hayamos escrito a la derecha. Para solucionar esto tenemos dos cláusulas que sustituyen a la palabra clave INNER, estas cláusulas son LEFT y RIGHT. LEFT toma todos los registros de la tabla de la izquierda aunque no tengan ningún registro en la tabla de la izquierda. RIGHT realiza la misma operación pero al contrario, toma todos los registros de la tabla de la derecha aunque no tenga ningún registro en la tabla de la izquierda.

La sintaxis expuesta anteriormente pertenece a ACCESS, en donde todas las sentencias con la sintaxis funcionan correctamente. Los manuales de SQL-SERVER dicen que esta sintaxis es incorrecta y que hay que añadir la palabra reservada OUTER: LEFT OUTER JOIN y RIGHT OUTER JOIN. En la práctica funciona correctamente de una u otra forma.

No obstante, los INNER JOIN ORACLE no es capaz de interpretarlos, pero existe una sintaxis en formato ANSI para los INNER JOIN que funcionan en todos los sistemas. Tomando como referencia la siguiente sentencia:

```
SELECT Facturas.*, Albaranes.*
FROM Facturas INNER JOIN Albaranes ON Facturas.IdAlbaran =
Albaranes.IdAlbaran
WHERE Facturas.IdCliente = 325
```

La transformación de esta sentencia a formato ANSI sería la siguiente:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes
WHERE Facturas.IdAlbaran = Albaranes.IdAlbaran AND
Facturas.IdCliente = 325
```

Como se puede observar los cambios realizados han sido los siguientes:

1. Todas las tablas que intervienen en la consulta se especifican en la cláusula `FROM`.
2. Las condiciones que vinculan a las tablas se especifican en la cláusula `WHERE` y se vinculan mediante el operador lógico `AND`.

Referente a los `OUTER JOIN`, no funcionan en ORACLE y además no conozco una sintaxis que funcione en los tres sistemas. La sintaxis en ORACLE es igual a la sentencia anterior pero añadiendo los caracteres `(+)` detrás del nombre de la tabla en la que deseamos aceptar valores nulos, esto equivale a un `LEFT JOIN`:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes
WHERE Facturas.IdAlbaran = Albaranes.IdAlbaran (+) AND
Facturas.IdCliente = 325
```

Y esto a un `RIGHT JOIN`:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes
WHERE Facturas.IdAlbaran (+) = Albaranes.IdAlbaran AND
Facturas.IdCliente = 325
```

En SQL-SERVER se puede utilizar una sintaxis parecida, en este caso no se utiliza los caracteres `(+)` sino los caracteres `=*` para el `LEFT JOIN` y `*=` para el `RIGHT JOIN`.

Consultas de Autocombinación

La autocombinación se utiliza para unir una tabla consigo misma, comparando valores de dos columnas con el mismo tipo de datos. La sintaxis en la siguiente:

```
SELECT alias1.columna, alias2.columna, ...
FROM tabla1 as alias1, tabla2 as alias2
WHERE alias1.columna = alias2.columna
AND otras condiciones
```

Por ejemplo, para visualizar el número, nombre y puesto de cada empleado, junto con el número, nombre y puesto del supervisor de cada uno de ellos se utilizaría la siguiente sentencia:

```
SELECT t.num_emp, t.nombre, t.puesto, t.num_sup,s.nombre,
s.puesto
FROM empleados AS t, empleados AS s WHERE t.num_sup =
s.num_emp
```

Consultas de Combinaciones no Comunes

La mayoría de las combinaciones están basadas en la igualdad de valores de las columnas que son el criterio de la combinación. Las no comunes se basan en otros operadores de combinación, tales como `NOT`, `BETWEEN`, `<>`, etc.

Por ejemplo, para listar el grado salarial, nombre, salario y puesto de cada empleado ordenando el resultado por grado y salario habría que ejecutar la siguiente sentencia:

```
SELECT grados.grado,empleados.nombre, empleados.salario,
empleados.puesto
FROM empleados, grados
WHERE empleados.salario
BETWEEN grados.salarioinferior AND grados.salariosuperior
ORDER BY grados.grado, empleados.salario
```

Para listar el salario medio dentro de cada grado salarial habría que lanzar esta otra sentencia:

```
SELECT grados.grado, AVG(empleados.salario) FROM empleados,
grados
WHERE empleados.salario
BETWEEN grados.salarioinferior AND grados.salariosuperior
GROUP BY grados.grado
```

CROSS JOIN (SQL-SERVER)

Se utiliza en SQL-SERVER para realizar consultas de unión. Supongamos que tenemos una tabla con todos los autores y otra con todos los libros. Si deseáramos obtener un listado combinar ambas tablas de tal forma que cada autor apareciera junto a cada título, utilizaríamos la siguiente sintaxis:

```
SELECT Autores.Nombre, Libros.Titulo
FROM Autores CROSS JOIN Libros
```

SELF JOIN

`SELF JOIN` es una técnica empleada para conseguir el producto cartesiano de una tabla consigo misma. Su utilización no es muy frecuente, pero pongamos algún ejemplo de su utilización.

Supongamos la siguiente tabla (El campo autor es numérico, aunque para ilustrar el ejemplo utilice el nombre):

Código (Código del libro)	Autor (Nombre del Autor)
B0012	1. Francisco López
B0012	2. Javier Alonso
B0012	3. Marta Rebolledo
C0014	1. Francisco López
C0014	2. Javier Alonso
D0120	2. Javier Alonso
D0120	3. Marta Rebolledo

Queremos obtener, para cada libro, parejas de autores:

```
SELECT A.Codigo, A.Autor, B.Autor FROM Autores A, Autores B
WHERE  A.Codigo = B.Codigo
```

El resultado es el siguiente:

Código	Autor	Autor
B0012	1. Francisco López	1. Francisco López
B0012	1. Francisco López	2. Javier Alonso
B0012	1. Francisco López	3. Marta Rebolledo
B0012	2. Javier Alonso	2. Javier Alonso
B0012	2. Javier Alonso	1. Francisco López
B0012	2. Javier Alonso	3. Marta Rebolledo
B0012	3. Marta Rebolledo	3. Marta Rebolledo
B0012	3. Marta Rebolledo	2. Javier Alonso
B0012	3. Marta Rebolledo	1. Francisco López
C0014	1. Francisco López	1. Francisco López
C0014	1. Francisco López	2. Javier Alonso
C0014	2. Javier Alonso	2. Javier Alonso
C0014	2. Javier Alonso	1. Francisco López
D0120	2. Javier Alonso	2. Javier Alonso
D0120	2. Javier Alonso	3. Marta Rebolledo
D0120	3. Marta Rebolledo	3. Marta Rebolledo
D0120	3. Marta Rebolledo	2. Javier Alonso

Como podemos observar, las parejas de autores se repiten en cada uno de los libros, podemos omitir estas repeticiones de la siguiente forma

```
SELECT A.Codigo, A.Autor, B.Autor FROM Autores A, Autores B
WHERE A.Codigo = B.Codigo AND A.Autor < B.Autor
```

El resultado ahora es el siguiente:

Código	Autor	Autor
B0012	1. Francisco López	2. Javier Alonso
B0012	1. Francisco López	3. Marta Rebolledo
C0014	1. Francisco López	2. Javier Alonso
D0120	2. Javier Alonso	3. Marta Rebolledo

Ahora tenemos un conjunto de resultados en formato Autor - CoAutor.

Si en la tabla de empleados quisiéramos extraer todas las posibles parejas que podemos realizar, utilizaríamos la siguiente sentencia:

```
SELECT Hombres.Nombre, Mujeres.Nombre
FROM Empleados Hombre, Empleados Mujeres
WHERE Hombre.Sexo = 'Hombre' AND  Mujeres.Sexo = 'Mujer'
AND Hombres.Id <>Mujeres.Id
```

Para concluir supongamos la tabla siguiente:

Id	Nombre	SuJefe
1	Marcos	6
2	Lucas	1
3	Ana	2
4	Eva	1
5	Juan	6
6	Antonio	

Queremos obtener un conjunto de resultados con el nombre del empleado y el nombre de su jefe:

```
SELECT Emple.Nombre, Jefes.Nombre FROM Empleados Emple,
Empleados Jefe
WHERE Emple.SuJefe = Jefes.Id
```

Consultas de Unión Externas

Se utiliza la operación UNION para crear una consulta de unión, combinando los resultados de dos o más consultas o tablas independientes. Su sintaxis es:

```
[TABLE] consulta1 UNION [ALL] [TABLE]
consulta2 [UNION [ALL] [TABLE] consultan [ ... ]]
```

En donde:

consulta1, consulta2, consultan

Son instrucciones SELECT, el nombre de una consulta almacenada o el nombre de una tabla almacenada precedido por la palabra clave TABLE.

Puede combinar los resultados de dos o más consultas, tablas e instrucciones SELECT, en cualquier orden, en una única operación UNION. El ejemplo siguiente combina una tabla existente llamada Nuevas Cuentas y una instrucción SELECT:

```
TABLE [Nuevas Cuentas] UNION ALL SELECT * FROM Clientes
WHERE [Cantidad pedidos] > 1000;
```

Si no se indica lo contrario, no se devuelven registros duplicados cuando se utiliza la operación UNION, no obstante puede incluir el predicado ALL para asegurar que se devuelven todos los registros. Esto hace que la consulta se ejecute más rápidamente. Todas las consultas en una operación UNION deben pedir el mismo número de campos, no obstante los campos no tienen porqué tener el mismo tamaño o el mismo tipo de datos.

Se puede utilizar una cláusula GROUP BY y/o HAVING en cada argumento consulta para agrupar los datos devueltos. Puede utilizar una cláusula ORDER BY al final del último argumento consulta para visualizar los datos devueltos en un orden específico.

```
SELECT [Nombre de compañía], Ciudad FROM Proveedores WHERE
País = 'Brasil' UNION SELECT [Nombre de compañía], Ciudad
FROM Clientes
WHERE País = "Brasil"
```

Recupera los nombres y las ciudades de todos proveedores y clientes de Brasil

```
SELECT [Nombre de compañía], Ciudad FROM Proveedores WHERE
País = 'Brasil'
UNION SELECT [Nombre de compañía], Ciudad FROM Clientes
WHERE País =
'Brasil' ORDER BY Ciudad
```

Recupera los nombres y las ciudades de todos proveedores y clientes radicados en Brasil, ordenados por el nombre de la ciudad

```
SELECT [Nombre de compañía], Ciudad FROM Proveedores WHERE
País = 'Brasil'
UNION SELECT [Nombre de compañía], Ciudad FROM Clientes
WHERE País =
'Brasil' UNION SELECT [Apellidos], Ciudad FROM Empleados
WHERE Región =
'América del Sur'
```

Recupera los nombres y las ciudades de todos los proveedores y clientes de brasil y los apellidos y las ciudades de todos los empleados de América del Sur

```
TABLE [Lista de clientes] UNION TABLE [Lista de
proveedores]
```

Recupera los nombres y códigos de todos los proveedores y clientes