SQL

Básico

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 DLL 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 DLL:

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

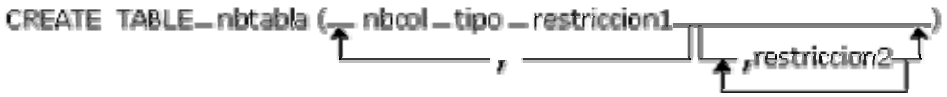
Funciones de Agregado

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.

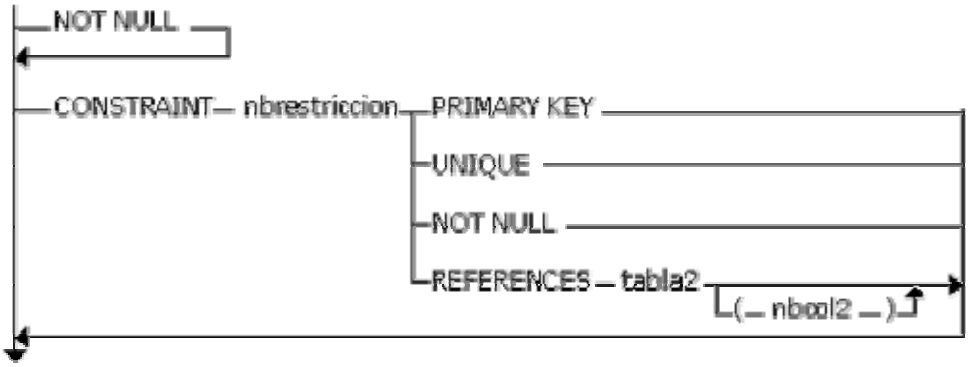
Función	Descripción
AVG	Utilizada para calcular el promedio de los valores de un campo determinado
COUNT	Utilizada para devolver el número de registros de la selección
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado
MAX	Utilizada para devolver el valor más alto de un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

DDL. El lenguaje de definición de datos.

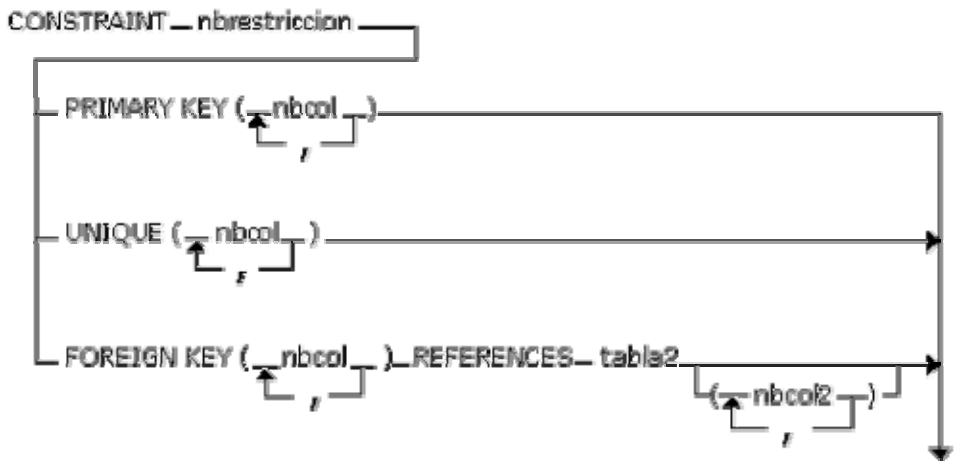
CREATE TABLE.



nbtabla: **n**om**b**re de la **tab**la que estamos definiendo
nbcol: **n**om**b**re de la **col**umna que estamos definiendo
tipo: **tipo de dato** de la columna, todos los datos almacenados en la columna deberán ser de ese tipo.
restricción1: es una restricción que aparece **dentro de la definición de la columna** después del tipo de dato y **afecta** a **una columna**, la que se está definiendo.



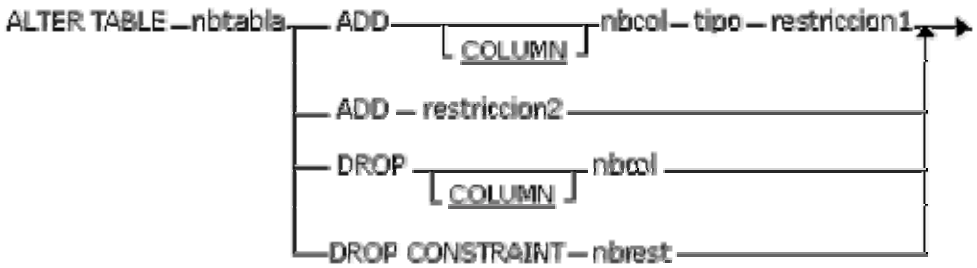
La cláusula **NOT NULL** indica que **la columna no podrá contener un valor nulo**
La cláusula **CONSTRAINT** sirve para definir una **restricción** que se podrá eliminar cuando queramos sin tener que borrar la columna. A cada restricción se le asigna un nombre que se utiliza para identificarla y para poder eliminarla cuando se quiera.
La cláusula **PRIMARY KEY** se utiliza para definir la columna como **clave principal de la tabla**. Esto supone que **la columna no puede contener valores nulos ni pueden haber valores duplicados** en esa columna. En una tabla **no pueden haber varias claves primarias**, por lo que no podemos incluir la cláusula **PRIMARY KEY** más de una vez, en caso contrario la sentencia da un error. No hay que confundir la definición de varias claves principales con la definición de una clave principal compuesta por varias columnas, esto último sí está permitido y se define con una restricción de tipo 2.
La cláusula **UNIQUE** sirve para definir un **índice único** sobre la columna. Un índice único es un índice que **no permite valores duplicados**, es decir que si una columna tiene definida un restricción de **UNIQUE** no podrán haber dos filas con el mismo valor en esa columna. Se suele emplear para que el sistema compruebe el mismo que no se añaden valores que ya existen. Indicado para definir claves alternativas.
La cláusula **NOT NULL** indica que la columna no puede contener valores nulos, cuando queremos indicar que una columna no puede contener el valor nulo lo podemos hacer sin poner la cláusula **CONSTRAINT**, o utilizando una cláusula **CONSTRAINT**.
La última restricción que podemos definir sobre una columna es la de clave ajena, una **clave ajena es una columna** o conjunto de columnas **que contiene un valor que hace referencia a una fila de otra tabla**, en una restricción de tipo 1 se puede definir con la cláusula **REFERENCES**, después de la palabra reservada indicamos a qué tabla hace referencia, opcionalmente podemos indicar entre paréntesis el nombre de la columna donde tiene que buscar el valor de referencia, por defecto coge la clave primaria de la tabla2.
restricción2 se utiliza **para definir una característica que afecta a una columna o a una combinación de columnas** de la tabla que estamos definiendo, **se escribe después de haber definido todas las columnas** de la tabla.



La sintaxis de una restricción de tipo 2 es muy similar a la **CONSTRAINT** de una restricción 1 la diferencia es que ahora tenemos que indicar sobre qué columnas queremos definir la restricción. Se utilizan obligatoriamente las restricciones de tipo 2 cuando la restricción afecta a un grupo de columnas o cuando queremos definir más de una **CONSTRAINT** para una columna (sólo se puede definir una restricción1 en cada columna).
PRIMARY KEY se utiliza para definir la **clave principal** de la tabla.
UNIQUE sirve para definir un **índice único** sobre una columna o sobre una combinación de columnas.
FOREIGN KEY sirve para definir una **clave ajena** sobre una columna o una combinación de columnas. En una restricción 1 se puede definir con la cláusula **REFERENCES**. Para definir una clave ajena en una restricción de tipo 2 debemos empezar por las palabras **FOREIGN KEY** después indicamos entre paréntesis la/s columna/s que es clave ajena, a continuación la palabra reservada **REFERENCES** seguida del nombre de la tabla a la que hace referencia, opcionalmente podemos indicar entre paréntesis el nombre de la/s columna/s donde tiene que buscar el valor de referencia, por defecto coge la clave primaria de la tabla2

ALTER TABLE.

La sentencia ALTER TABLE permite añadir o eliminar columnas y añadir o eliminar restricciones sobre una tabla.

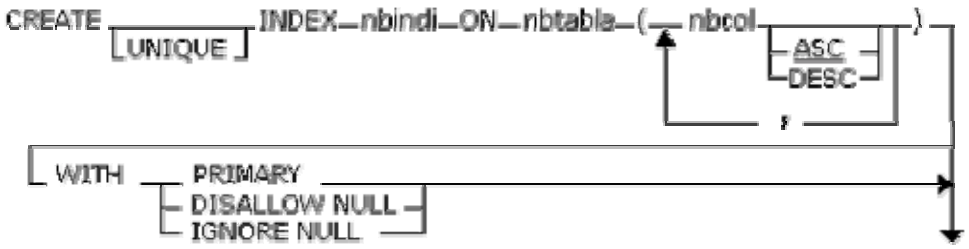


nbtabla: **n**om**b**re de la **tab**la que estamos definiendo
nbcol: **n**om**b**re de la **col**umna que estamos definiendo
tipo: **tipo de dato** de la columna, todos los datos almacenados en la columna deberán ser de ese tipo.
Restricción1 y restriccion2 son restricciones expresadas como en CREATE TABLE.
Nbrest: nombre de restricción.

DROP TABLE

La sentencia DROP TABLE elimina una tabla completa, y por supuesto, cualquier dato que contuviera.
Su sintaxis es muy sencilla: DROP TABLE nombreTabla.

CREATE INDEX



nbindi: **nombre del índice** que estamos definiendo. **En una tabla no pueden haber dos índices con el mismo nombre** de lo contrario da error.
nbtabla: **nombre de la tabla donde definimos el índice**. A continuación entre paréntesis se indica la composición del índice (las columnas que lo forman).
nbcol: **nombre de la columna que indexamos**. Después del nombre de la columna podemos indicar cómo queremos que se ordenen las filas según el índice mediante las cláusulas **ASC**/**DESC**.

ASC: la cláusula **ASC** es la que se asume **por defecto** e indica que el **orden** elegido para el índice es **ascendente**
DESC: indica **orden descendente**
Podemos formar un índice **basado en varias columnas**, en este caso después de indicar la primera columna con su orden, se escribe una coma y la segunda columna también con su orden, así sucesivamente hasta indicar todas las columnas que forman el índice.
Opcionalmente se pueden indicar las cláusulas:

WITH PRIMARY indica que el índice define la **clave principal** de la tabla, si la tabla ya tiene una clave principal, la sentencia **CREATE INDEX** dará error.
WITH DISALLOW NULL indica que **no permite valores nulos** en las columnas que forman el índice.
WITH IGNORE NULL indica que **las filas** que **tengan valores nulos** en las columnas que forman el índice **se ignoran**, no aparecen cuando recuperamos las filas de la tabla utilizando ese índice.

DROP INDEX

La sentencia **DROP INDEX** sirve para **eliminar un índice** de una tabla. Se elimina el índice pero no las columnas que lo forman.
Su sintaxis es: DROP INDEX nombreIndice ON nombreTabla

DML. El lenguaje de manipulación de datos.

SELECT.

Es la orden principal del DML y prácticamente de todo SQL. Es la que se utiliza para realizar Consultas (Query en inglés). Siempre devuelve datos en forma tabular.

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

Para recuperar todos los campos, se puede utilizar un *

SELECT * from 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.

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;

En algunos sistemas, está permitido utilizar en ORDER BY campos no especificados en la lista de selección (Detrás de SELECT, y en otros no).

En Access, se puede utilizar en lugar de un campo, un numero, que indica el número de columna por la que se desea ordenar.

SELECT CodigoPostal, Nombre, Telefono FROM Clientes
ORDER BY 1

Consultas con Predicado

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

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.

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 Apellido

Si no se incluye la cláusula ORDER BY, la consulta devolverá un conjunto arbitrario de 25 registros de la tabla Estudiantes.

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 Apellido DESC;

El valor que va a continuación de TOP debe ser un Integer sin signo.

DISTINCT
Omite los registros que contienen datos duplicados en los campos seleccionados.

SELECT DISTINCT Apellido FROM Empleados;

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 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.

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.

SELECT Apellido AS Empleado FROM Empleados AS Trabajadores;

La 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.

Criterios de Selección. La clausula WHERE

Hasta ahora, se ha visto la forma de recuperar los registros de las tablas, las formas empleadas devolvían todos los registros de la mencionada tabla, con todos o algunos campos.

A partir de ahora, 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. No hay una sintaxis que funcione en todos los sistemas, por lo que se hace necesario particularizarlas según el SGBD.
En el caso concreto de access, las fechas se encierran entre almohadillas, siguiendo el formato #dd/mm/aaaa#

Referente a los valores lógicos True o False cabe destacar que no son reconocidos en ORACLE ni SQL-SERVER ni otros SGBD. ACCESS, almacena internamente 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".

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');
```

Operadores Lógicos

Los operadores lógicos principales soportados por SQL son: AND, OR y NOT.

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)

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 '*' y '?' como carácter comodín y hay que sustituirlo por el carácter tanto por ciento (%) y subrayado (_) respectivamente.

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');

Funciones agregadas

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).

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;
```

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 o una constante o expresión.

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

SELECT Max(Gastos) AS ElMax 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;
```

Combinación de tablas. Producto cartesiano.

Las vinculaciones entre tablas se realiza mediante la inclusión de dos o más tablas en la claúsula FROM. Estas tablas deben estar relacionadas de alguna manera mediante claves ajenas a claves primarias.

La forma de utilizarla es la siguiente, suponiendo que tabla1 tiene una clave ajena a la tabla tabla2.

```
SELECT campos
  FROM tabla1, tabla2
  WHERE tabla1.claveajena=tabla2.claveprimaria
```

Sin detrimento de utilizar otras comparaciones en la clausula WHERE.

Consultas de Autocombinación

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

Esto ocurre cuando una tabla tiene una clave ajena a sí misma.
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 resumen.

Se pueden obtener subtotales con la **cláusula GROUP BY**. Una consulta con una cláusula GROUP BY se denomina consulta agrupada ya que agrupa los datos de la tabla origen y produce una única fila resumen por cada grupo formado. Las columnas indicadas en el GROUP BY se llaman columnas de agrupación.

```
SELECT SUM(ventas) FROM repventas
```

> Obtiene la suma de las ventas de todos los empleados.

```
SELECT SUM(ventas) FROM repventas
GROUP BY oficina
```

> Se forma un grupo para cada oficina, con las filas de la oficina, y la suma se calcula sobre las filas de cada grupo. El ejemplo anterior obtiene una lista con la suma de las ventas de los empleados de cada oficina.

La consulta quedaría mejor incluyendo en la lista de selección la oficina para saber a qué oficina corresponde la suma de ventas:

```
SELECT oficina,SUM(ventas)
FROM repventas
GROUP BY oficina
```

La columna de agrupación se puede indicar mediante un nombre de columna o cualquier expresión válida basada en una columna pero no se pueden utilizar los alias de campo.

En la lista de selección sólo pueden aparecer : valores constantes, funciones de columna, columnas de agrupación (columnas que aparecen en la cláusula GROUP BY) o cualquier expresión basada en las anteriores.

```
SELECT SUM(importe),rep*10
FROM pedidos
GROUP BY rep*10
```

Se pueden agrupar las filas por varias columnas, en este caso se indican las columnas separadas por una coma y en el orden de mayor a menor agrupación. Se permite incluir en la lista de agrupación hasta 10 columnas.

```
SELECT SUM(ventas)
FROM oficinas
GROUP BY region,ciudad
```

Todas las filas que tienen valor nulo en el campo de agrupación, pasan a formar un único grupo. Es decir, considera el valor nulo como un valor cualquiera a efectos de agrupación.

La cláusula HAVING

La cláusula HAVING nos permite seleccionar filas de la tabla resultante **de una consulta de resumen**.

Para la condición de selección se pueden utilizar los mismos tests de comparación descritos en la cláusula WHERE, también se pueden escribir condiciones compuestas (unidas por los operadores OR, AND, NOT), pero existe una restricción. En la condición de selección sólo pueden aparecer : valores constantes, funciones de columna, columnas de agrupación (columnas que aparecen en la cláusula GROUP BY) o cualquier expresión basada en las anteriores.

```
SELECT oficina
FROM empleados
GROUP BY oficina
HAVING AVG(ventas) > 500000
```

Subconsultas

Una subconsulta es una instrucción SELECT anidada dentro de una instrucción SELECT, DELETE, o UPDATE o dentro de otra subconsulta.
Puede utilizar tres formas de sintaxis para crear una subconsulta:
-Test de comparación de un valor.
La subconsulta devuelve un único valor con el que se hace una comparación simple.

```
SELECT Nombre from empleado where salario=(SELECT min(salario) from empleado).
```

-Test de comparación con un conjunto [ANY | ALL]
La subconsulta devuelve una COLUMNA, con un conjunto de valores. Se comprueba si la condición se cumple para alguno de los valores (ANY) o para todos (ALL).

```
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.

- Test de pertenencia a un conjunto [IN]
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 operacion =ANY es equivalente al operador IN, ambos devuelven el mismo resultado.

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.

Insertar un único Registro:

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 dd/mm/aa y entre caracteres de almohadillas (#).

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