



## Lenguaje SQL

### 1.- INTRODUCCION

El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado.

#### 1.1. 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.

#### 1.2 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

### 1.3 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

### 1.4 Operadores Lógicos

Operador	Uso
AND	Es el "y" lógico. Evalúa 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 verdad si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

### 1.5 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

## 1.6 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

## 2. 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 (por ejemplo Recordset). Este conjunto de registros puede ser modificable.

### 2.1 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;

**SELECT** Expresión **FROM** Tabla;

Expresión:

1. Constante
2. Variable o Campo
3. Función Standard
4. Combinación de las anteriores enlazadas con operadores.



## 2.2 Ordenar los registros

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula **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 más 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 cláusula (**ASC** -se toma este valor por defecto) ó descendente (**DESC**)

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

## 2.3 Consultas con Predicado

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

Predicado	Descripción
*	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 (NO)	Omite los registros duplicados basándose en la totalidad del registro y no sólo en los campos seleccionados. (Teoría)

\*

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 es 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; (En algunos SGBD no funciona, probarlo)  
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.



**SELECT DISTINCT** Apellido **FROM** Empleados;

Si la tabla empleados contiene dos registros: Antonio López y Marta López el ejemplo del predicado **DISTINCT** devuelve un único registro con el valor López en el campo Apellido ya que busca no duplicados en dicho campo.

**SELECT DISTINCT** Apellido Nombre **FROM** Empleados;

Este último ejemplo devuelve dos registros con el valor López en el apellido y los nombres ya que se busca no duplicados en el registro completo.

## 2.4 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 DISTINCT** Apellido **AS** Empleado **FROM** Empleados;

**SELECT DISTINCT** Empleado = Apellido **FROM** Empleados

## 3. Criterios de Selección (ACCESS)

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 unas condiciones preestablecidas.

Antes de comenzar el desarrollo de este capítulo hay que recalcar tres detalles de 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 se puede establecer condiciones de búsqueda en los campos memo y; la tercera y última hace referencia a las fechas. Las fechas se deben escribir siempre en formato mm-dd-aa en donde mm representa el mes, dd el día y aa el año, hay que prestar atención a los separadores -no sirve la separación habitual de la barra (/), hay que utilizar el guión (-) y además la fecha debe ir encerrada entre almohadillas (#). Por ejemplo si deseamos referirnos al día 3 de Septiembre de 1995 deberemos hacerlo de la siguiente forma; #09-03-95# ó #9-3-95#.

### 3.1 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(Curiosidad):

<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

Null and Verdad = Null

Null or Verdad = Verdad

Null and Falso =Falso

Null or Falso = Null

Importante **la Función** / **Not Is Null** (Verificar la sintaxis en el SGBD)

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');
```

### 3.2 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)  
  
(NO)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) (Para Access)
```

### 3.3 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.

**Motor Jet:**

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'

### SQL Server:

%	Cualquier cadena de cero o más caracteres.	WHERE title <b>LIKE</b> '%computer%' busca todos los títulos de libros que contengan la palabra 'computer' en cualquier parte del título.
_ (subrayado)	Cualquier carácter individual	WHERE au_fname <b>LIKE</b> '_ean' busca todos los nombres de cuatro letras que finalicen con ean (Dean, Sean, etc.).
[ ]	Cualquier carácter individual de intervalo ([a-f]) o del conjunto ([abcdef]) especificado.	WHERE au_lname <b>LIKE</b> '[C-P]arsen' busca apellidos de autores que terminen con arsen y comiencen con cualquier carácter individual entre C y P, por ejemplo Carsen, Larsen, Karsen, etc.
[^]	Cualquier carácter individual que no se encuentre en el intervalo ([^a-f]) o el conjunto ([^abcdef]) especificado.	WHERE au_lname <b>LIKE</b> 'de[^l]%' busca todos los apellidos de autores que comienzan con de y en los que la siguiente letra no sea l.

Consultarlo en Internet o en la ayuda del software correspondiente.

¿Y en MySQL? Buscarlo

### 3.4 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');

### 3.5 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. Después de escribir esta cláusula se deben especificar las condiciones expuestas en los partados 3.1 y 3.2. 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 Empleados 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');
```

[\(Moodle\) Funciones estándar del SGBD: En Access](#) <F1> [Información](#)

---

## 4. Agrupamiento de Registros

### 4.1 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 está 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
```

#### 4.2 AVG

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;
```

#### 4.3 Count

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;
```

#### 4.4 Max, Min

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo específico 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';
```

#### 4.5 StDev, StDevP

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';
```

#### 4.6 Sum

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

Sum(expr)

En donde expr representa 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;
```

#### 4.7 Var, VarP

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:

VarP(expr) Var(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';
```

### 5. 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.

#### 5.1 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 Tabla.\* FROM Tabla WHERE criterio

DELETE es especialmente útil cuando se desea eliminar varios registros. En una instrucción DELETE con múltiples tablas, debe incluir el nombre de tabla (Tabla.\*). Si especifica más de una tabla desde la que eliminar registros, todas deben ser tablas de muchos a uno. Si desea eliminar todos los registros de una tabla, eliminar la propia tabla es más eficiente que ejecutar una consulta de borrado.

Se puede utilizar DELETE para eliminar registros de una única tabla o desde varios lados de una relación uno a muchos. Las operaciones de eliminación en cascada en una consulta únicamente eliminan desde varios lados de una relación. Por ejemplo, en la relación entre las tablas Clientes y Pedidos, la tabla Pedidos es la parte de muchos por lo que las operaciones en cascada solo afectarán a la tabla Pedidos. Una consulta de borrado elimina los registros completos, no únicamente los datos en campos específicos. Si desea eliminar valores en un campo especificado, crear una consulta de actualización que cambie los valores a Null.

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';

## 5.2 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.

### 5.2.1 Para 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 indicándolas en formato mm-dd-aa y entre caracteres de almohadillas (#).

### 5.2.2 Para 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 estructura 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 incrementándose 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 especifica 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 asegurarse 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_Viejos;  
INSERT INTO Empleados (Nombre, Apellido, Cargo)  
VALUES ('Luis', 'Sánchez', 'Becario');
```



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

### 5.3 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
```

## 6. Tipos de Datos (Opcional en Access)

**Esta tabla la intentaremos rellenar a lo largo del curso para los SGBD Access, SQL SERVER (Trabajo de clase) y MYSQL**

	TIPO	DESCRIPCIÓN	ALMACENAMIENTO
Numéricos			
Alfanuméricos			
Fechas			



Otros tipos de Datos			

## 7. 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 algún(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.

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.

```
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 DISTINCT 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 DISTINCT Nombre_Contacto, Nombre_Compañia, Cargo_Contacto,  
Telefono FROM Clientes WHERE (ID_Cliente IN (SELECT DISTINCT  
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 DISTINCT Pedidos.Id_Producto, Pedidos.Cantidad,  
(SELECT DISTINCT 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.*

## 9. Consultas de Unión Internas

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 IDCategoría:

```
SELECT Nombre_Categoría, NombreProducto  
FROM Categorías INNER JOIN Productos  
ON Categorías.IDCategoría = Productos.IDCategoría;
```

En el ejemplo anterior, IDCategoría 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, Categorías.IDCategoría.

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;
```

(Intentar escribir el formato de la Inner Join de 2 tablas tanto de Access como de SQL Server)

---

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.

```
Select *      FROM Tabla1 INNER JOIN Tabla2 ON Tabla1.IdT1=Tabla2.IdT2  
  
              INNER JOIN Tabla3 ON Tabla3.N3=
```

## 10. 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*

---

(\*\*\*) Funciones estándar del SGBD: En Access <F1> Información

Funciones aritméticas:



Función	Propósito	Ejemplo probado
ABS		
ROUND		

Funciones de manejo de Fechas:

Función	Propósito	Ejemplo probado
DAY		
MONTH		
YEAR		
DATEPART		
NOW		

Funciones de cadenas de caracteres:

Función	Propósito	Ejemplo probado

Otras Funciones que consideres interesantes:

Función	Propósito	Ejemplo probado