1.- INTRODUCCION

El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado, utilizado por el motor de base de datos de Microsoft Jet. SQL se utiliza para crear objetos QueryDef, como el argumento de origen del método OpenRecordSet y como la propiedad RecordSource del control de datos. También se puede utilizar con el método Execute para crear y manipular directamente las bases de datos Jet y crear consultas SQL de paso a través para manipular bases de datos remotas cliente - servidor.

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. 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.

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 recordset. Este conjunto de registros es 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;
```

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

2.2 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;
```

2.3 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

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 y apellidos de los 25 primeros estudiantes:

```
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.

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 cualquiera de estos casos esta 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;
```

3. Criterios de Selección

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

Antes de comenzar hay que recalcar tres detalles de vital importancia:

1. Cada vez que se desee establecer una condición referida a un campo de texto la condición de búsqueda debe ir encerrada entre comillas simples.

2. No se posible establecer condiciones de búsqueda en los campos memo.

3. 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 siguente forma; #09-03-95# ó #9-3-95#.

3.1 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. 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';
```

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 cantidad Between 28000 And 30000;
(Devuelve los pedidos realizados cuya cantidad este entre 28000 y 30000)
```

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 puede utilizar el caracter comodín %.

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.

```
SELECT * FROM Pedidos WHERE Provincia Iike ‘S%’
```

3.4 El Operador In

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

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

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

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 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*;
```

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 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';
```

4.5 Sum

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;
```

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  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 afectaran 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 tipo: 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 indicarlas 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 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;
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

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 Datos	Longitud	Descripción
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

7. 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 tabla1 INNER JOIN tabla2 ON tabla1.campo1 comp tabla2.campo2
```

En donde:

tabla1, tabla2

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.

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.

8. Estructuras de las Tablas

8.1 Creación de Tablas Nuevas

Si se está utilizando el motor de datos de Microsoft para acceder a bases de datos access, sólo se puede emplear esta instrucción para crear bases de datos propias de access. Su sintaxis es:

```
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.

8.2 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 CONTRAINT debe aparecer inmediatamente después de la especificación del campo indexeado.

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 .

8.3 Creación de Índices

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 ua 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.

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 que el campo ID no contenga valores nulos ni repetidos.

8.4 Modificar el Diseño de una Tabla

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.

Sugerencias:

Siempre es bueno intentar hacer las cosas de igual modo para que el mantenimiento y la revisión nos sea una labor lo más sencilla posible. Las siguiente normas son bastante útiles a la hora de elaborar sentecias SQL:

1. Las cláusulas siempre escribirlas con Mayúsculas.
2. Los operadores lógicos de sentencias siempre con Mayúsculas.
3. Las operaciones siempre la primera letra con mayúsculas y el resto en minúsculas.
4. Los operadores lógicos incluidos en otros operadores la primera letra con mayúsculas y el resto con minúculas.
5. Los Nombres de las Tablas, Campos y Consultas, siempre la primera letra con mayúsculas y el resto con minúsculas, en algunos casos es bueno utilizar el carácter "_" para definir mejor el nombre: Detalles_Pedidos.
6. Aunque con el motor Jet se pueden utilizar acentos y espacios en blanco para nombrar los campos, las tablas y las consultas no es bueno utilizarlos porque cuando se exportar tablas a otros sistemas los acentos y los espacios en blanco pueden producir errores innecesarios.
7. Si se utiliza espacios en blanco para los nombres de tablas o consultas cada vez que se hace referencia a ellos en una consulta deben incluir sus nombres entre corchetes. Ejemplo:

```
SELECT [ID de Pedido], [Nombre del Producto], Cantidad FROM [Detalles del Pedido];
```