1. Introducción

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 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

1.3 Cláusulas

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

Comando	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
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.

Comando	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
DISTINCROW	Omite los registros duplicados basandose en la totalidad del registro y no sólo en los campos seleccionados.

ALL:

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

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

TOP:

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

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

Si no se incluye la cláusula **ORDER BY**, la consulta devolverá un conjunto arbitrario de 25 registros de la tabla Estudiantes .El predicado **TOP** no elige entre valores iguales. En el ejemplo anterior, si la nota media número 25 y la 26 son iguales, la consulta devolverá 26 registros. Se puede utilizar la palabra reservada **PERCENT** para devolver un cierto porcentaje de registros que caen al principio o al final de un rango especificado por la cláusula **ORDER BY**. Supongamos que en lugar de los 25 primeros estudiantes deseamos el 10 por ciento del curso:

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

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

DISTINCT:

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

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

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

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

DISTINCTROW:

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

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

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

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 DISTINCTROW Apellido AS Empleado
FROM Empleados;
```

2.5 Recuperar Información de una base de Datos Externa

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

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

En donde c:\databases\gestion.mdb es la base de datos que contiene la tabla Empleados.

3. Criterios de Selección

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

Antes de comenzar el desarrollo de este capítulo 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 se posible 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:

<expresión1>	**Operador**	**<expresión2>**	**Resultado**
Verdad	**AND**	Falso	Falso
Verdad	**AND**	Verdad	Verdad
Falso	**AND**	Verdad	Falso
Falso	**AND**	Falso	Falso

Verdad	**OR**	Falso	Verdad
Verdad	**OR**	Verdad	Verdad
Falso	**OR**	Verdad	Verdad
Falso	**OR**	Falso	Falso
Verdad	**XOR**	Verdad	Falso
Verdad	**XOR**	Falso	Verdad
Falso	**XOR**	Verdad	Verdad
Falso	**XOR**	Falso	Falso
Verdad	**Eqv**	Verdad	Verdad
Verdad	**Eqv**	Falso	Falso
Falso	**Eqv**	Verdad	Falso
Falso	**Eqv**	Falso	Verdad
Verdad	**Imp**	Verdad	Verdad
Verdad	**Imp**	Falso	Falso
Verdad	**Imp**	Null	Null
Falso	**Imp**	Verdad	Verdad
Falso	**Imp**	Falso	Verdad
Falso	**Imp**	Null	Verdad
Null	**Imp**	Verdad	Verdad
Null	**Imp**	Falso	Null
Null	**Imp**	Null	Null

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

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

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

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)

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)
```

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.

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'

3.4 El Operador In

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los indicados 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 apartados 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 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');
```

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 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 especifico de una consulta. Su sintaxis es:

SumP(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:

Var(expr)

VarP(expr)

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

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

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

5. Consultas de Actualización

Las consultas de actualización 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 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 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 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 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 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 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_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.
LONGBYNARY	Según se necesite	De cero 1 gigabyte. Utilizado para objetos OLE.
TEXT	1 byte por caracter	De cero a 255 caracteres.

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

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

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

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

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

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

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

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

8. Consultas de Referencias Cruzadas

Una consulta de referencias cruzadas es aquella que nos permite visualizar los datos en filas y en columnas, estilo tabla, por ejemplo:

Producto / Año	**1996**	**1997**
Pantalones	1.250	3.000
Camisas	8.560	1.253
Zapatos	4.369	2.563

Si tenemos una tabla de productos y otra tabla de pedidos, podemos visualizar en total de productos pedidos por año para un artículo determinado, tal y como se visualiza en la tabla anterior.

La sintaxis para este tipo de consulta es la siguiente:

```
TRANSFORM función agregada instrucción
select PIVOT campo pivot
[IN (valor1[, valor2[, ...]])]
```

En donde:

función agregada: Es una función **SQL** agregada que opera sobre los datos seleccionados.

instrucción select: Es una instrucción **SELECT**.

campo pivot: Es el campo o expresión que desea utilizar para crear las cabeceras de la columna en el resultado de la consulta.

valor1, valor2: Son valores fijos utilizados para crear las cabeceras de la columna.

Para resumir datos utilizando una consulta de referencia cruzada, se seleccionan los valores de los campos o expresiones especificadas como cabeceras de columnas de tal forma que pueden verse los datos en un formato más compacto que con una consulta de selección.

TRANSFORM es opcional pero si se incluye es la primera instrucción de una cadena **SQL**. Precede a la instrucción **SELECT** que especifica los campos utilizados como encabezados de fila y una cláusula **GROUP BY** que especifica el agrupamiento de las filas. Opcionalmente puede incluir otras cláusulas como por ejemplo **WHERE**, que especifica una selección adicional o un criterio de ordenación .

Los valores devueltos en campo pivot se utilizan como encabezados de columna en el resultado de la consulta. Por ejemplo, al utilizar las cifras de ventas en el mes de la venta como pivot en una consulta de referencia cruzada se crearían 12 columnas. Puede restringir el campo pivot para crear encabezados a partir de los valores fijos (**valor1, valor2**) listados en la cláusula opcional **IN**.

También puede incluir valores fijos, para los que no existen datos, para crear columnas adicionales.

Ejemplos:

```
TRANSFORM Sum(Cantidad) AS Ventas SELECT
Producto, Cantidad FROM
Pedidos WHERE Fecha Between #01-01-98# And #12-31-98# GROUP BY Producto
ORDER BY Producto PIVOT DatePart("m", Fecha);
```

Crea una consulta de tabla de referencias cruzadas que muestra las ventas de productos por mes para un año específico. Los meses aparecen de izquierda a derecha como columnas y los nombres de los productos aparecen de arriba hacia abajo como filas.

```
TRANSFORM Sum(Cantidad) AS Ventas SELECT
Compania FROM Pedidos
WHERE Fecha Between #01-01-98# And #12-31-98# GROUP BY Compania
ORDER BY Compania PIVOT "Trimestre " & DatePart("q",Fecha) In
('Trimestre1',
'Trimestre2', 'Trimestre 3', 'Trimestre 4');
```

Crea una consulta de tabla de referencias cruzadas que muestra las ventas de productos por trimestre de cada proveedor en el año indicado. Los trimestres aparecen de izquierda a derecha como columnas y los nombres de los proveedores aparecen de arriba hacia abajo como filas.

Un caso práctico:

Se trata de resolver el siguiente problema: tenemos una tabla de productos con dos campos, el código y el nombre del producto, tenemos otra tabla de pedidos en la que anotamos el código del producto, la fecha del pedido y la cantidad pedida. Deseamos consultar los totales de producto por año, calculando la media anual de ventas.

Estructura y datos de las tablas:

1. Artículos:

ID	Nombre
1	Zapatos
2	Pantalones
3	Blusas

2. Pedidos:

Id	Fecha	Cantidad
1	11/11/1996	250
2	11/11/1996	125
	11/11/1996	520
1	12/10/1996	50
2	04/05/1996	250
	05/08/1996	100
1	01/01/1997	40

2	02/08/1997	60
	05/10/1997	70
1	12/12/1997	8
2	15/12/1997	520
	17/10/1997	1250

Para resolver la consulta planteamos la siguiente consulta:

```
TRANSFORM Sum(Pedidos.Cantidad) AS Resultado
SELECT Nombre AS Producto,
Pedidos.Id     AS     Código,     Sum(Pedidos.Cantidad)     AS     TOTAL,
Avg(Pedidos.Cantidad)
AS Media FROM Pedidos INNER JOIN Artículos ON Pedidos.Id = Artículos.Id
GROUP BY Pedidos.Id, Artículos.Nombre PIVOT Year(Fecha);
```

y obtenemos el siguiente resultado:

Producto	**Código**	**TOTAL**	**Media**	**1996**	**1997**
Zapatatos	1	48	87	00	48
Pantalones	2	955	238,75	75	580
Blusas		1940	485	620	1320

Comentarios a la consulta:

La cláusula **TRANSFORM** indica el valor que deseamos visualizar en las columnas que realmente pertenecen a la consulta, en este caso **1996** y **1997**, puesto que las demás columnas son opcionales.

SELECT especifica el nombre de las columnas opcionales que deseamos visualizar, en este caso **Producto, Código, Total y Media**, indicando el nombre del campo que deseamos mostrar en cada columna o el valor de la misma. Si incluimos una función de cálculo el resultado se hará en base a los datos de la fila actual y no al total de los datos.

FROM especifica el origen de los datos. La primera tabla que debe figurar es aquella de donde deseamos extraer los datos, esta tabla debe contener al menos tres campos, uno para los títulos de la fila, otros para los títulos de la columna y otro para calcular el valor de las celdas.

En este caso en concreto se deseaba visualizar el nombre del producto, como el tabla de pedidos sólo figuraba el código del mismo se añadió una nueva columna en la cláusula select llamada Producto que se corresponda con el campo Nombre de la tabla de artículos. Para vincular el código del artículo de la tabla de pedidos con el nombre del misma de la tabla artículos se insertó la cláusula **INNER JOIN**.

La cláusula **GROUP BY** especifica el agrupamiento de los registros, contrariamente a los manuales de instrucción esta cláusula no es opcional ya que debe figurar siempre y debemos agrupar los registros por el campo del cual extraemos la información. En este caso existen dos campos del cual extraemos la información: pedidos.cantidad y artículos.nombre, por ellos agrupamos por los campos.

Para finalizar la cláusula **PIVOT** indica el nombre de las columnas no opcionales, en este caso 1996 y 1997 y como vamos a el dato que aparecerá en las columnas, en este caso empleamos el año en que se produjo el pedido, extrayéndolo del campo pedidos.fecha.

Otras posibilidades de fecha de la cláusula pivot son las siguientes:

1. Para agrupamiento por Trimestres

PIVOT "Tri " & DatePart("q",[Fecha]);

2. Para agrupamiento por meses (sin tener en cuenta el año)

PIVOT Format([Fecha],"mmm") In ("Ene", "Feb", "Mar", "Abr", "May", "Jun", "Jul", "Ago", "Sep", "Oct", "Nov", "Dic");

3. Para agrupar por días

PIVOT Format([Fecha],"Short Date")

9. Consultas de Unión Internas

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

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

En donde:

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

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

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

Se puede utilizar una operación **INNER JOIN** en cualquier cláusula **FROM**. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones Equi son las más comunes; éstas combinan los registros de dos tablas siempre que haya

concordancia de valores en un campo común a ambas tablas. Se puede utilizar **INNER JOIN** con las tablas *Departamentos y Empleados* para seleccionar todos los empleados de cada departamento. Por el contrario, para seleccionar todos los departamentos (incluso si alguno de ellos no tiene ningún empleado asignado) se emplea **LEFT JOIN** o todos los empleados (incluso si alguno no está asignado a ningún departamento), en este caso **RIGHT JOIN**.

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

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

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

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

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

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

También puede anidar instrucciones JOIN utilizando la siguiente sintaxis:

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

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

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 *Pedido*s, 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.

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.

11. Estructuras de las Tablas

11.1 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:

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

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	Es 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.

```
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 longitud 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.

11.2 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.
unicoN	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 indexado.

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

Tipo de Indice	Descripción
UNIQUE	Genera un índice 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 .

11.3 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 una tabla ya definida en la siguiente:

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

En donde:

Parte	Descripción
índice	Es el nombre del índice a crear.
tabla	Es el nombre de una tabla existente en la que se creará el índice.
campo	Es el nombre del campo o lista de campos que constituyen el índice.
ASC|DESC	Indica el orden de los valores de los campos ASC indica un orden ascendente (valor predeterminado) y DESC un orden descendente.
UNIQUE	Indica que el índice no puede contener valores duplicados.
DISALLOW NULL	Prohíbe 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 índice que sea "Clave Principal". Si un índice es clave principal implica 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 el campo *ID* no contenga valores nulos ni repetidos.

11.4 Modificar el Diseño de una Tabla

Modifica el diseño de una tabla ya existente, se pueden 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	Es 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 utiliza para agregar un índice de multicampos o de un único campo.
DROP COLUMN	Se utiliza 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 índice 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 índice de la tabla *Pedidos*.

12. Consultas con Parámetros

Las consultas con parámetros son aquellas cuyas condiciones de búsqueda se definen mediante parámetros. Si se ejecutan directamente desde la base de datos donde han sido definidas aparecerá un mensaje solicitando el valor de cada uno de los parámetros. Si deseamos ejecutarlas desde una aplicación hay que asignar primero el valor de los parámetros y después ejecutarlas. Su sintaxis es la siguiente:

```
PARAMETERS nombre1 tipo1, nombre2 tipo2,
... , nombreN tipoN Consulta
```

Parte	Descripción
nombre	Es el nombre del parámetro
tipo	Es el tipo de datos del parámetro
consulta	Una consulta SQL

Puede utilizar nombre pero no tipo de datos en una cláusula **WHERE** o **HAVING**.

```
PARAMETERS Precio_Minimo Currency, Fecha_Inicio
DateTime;
SELECT IDPedido, Cantidad FROM Pedidos WHERE Precio > Precio_Minimo
AND FechaPedido >= Fecha_Inicio;
```

El ejemplo siguiente muestra como utilizar los parámetros en el programa de ***Visual Basic***:

```
Public Sub GeneraConsulta()
Dim SQL As String
Dim Qd As QueryDef
Dim Rs As Recordset


SQL = "PARAMETERS Precio_Minimo Currency, Fecha_Inicio DateTime; "
SQL = SQL & "SELECT IDPedido, Cantidad FROM Pedidos WHERE Precio> "
SQL = SQL & "Precio_Minimo AND FechaPedido >= Fecha_Inicio; "
Set Qd = BaseDatos.CreateQueryDef(MiConsulta, SQL)
Qd.Parameters!Precio_Minimo = 2
Qd.Parameters!FechaInicio = #31/12/95#
```

```
Set Rs = Qd.OpenRecordset()
End Sub
```

Ejemplo:

```
PARAMETERS [Escriba los Apellidos:]
Text; SELECT * FROM Empleados

WHERE [Escriba los Apellidos:] = [Apellidos]; La ejecución desde
la base de datos solicita al usuario los apellidos del empleado y después
muestra los resultados
```

13. Bases de Datos Externas

Para el acceso a bases de datos externas se utiliza la cláusula **IN**. Se puede acceder a base de datos **dBase, Paradox o Btrieve**. Esta cláusula sólo permite la conexión de una base de datos externa a la vez. Una base de datos externa es una base de datos que no sea la activa. Aunque para mejorar los rendimientos es mejor adjuntarlas a la base de datos actual y trabajar con ellas.

Para especificar una base de datos que no pertenece a **Access Basic**, se agrega un punto y coma **(;)** al nombre y se encierra entre comillas simples. También puede utilizar la palabra reservada **DATABASE** para especificar la base de datos externa. Por ejemplo, las líneas siguientes especifican la misma tabla:

```
FROM Tabla IN '[dBASE IV; DATABASE=C:\DBASE\DATOS\VENTAS;]';
FROM Tabla IN 'C:\DBASE\DATOS\VENTAS' 'dBASE IV;'
```

Acceso a una base de datos externa de *Microsoft Access*:

```
SELECT IDCliente FROM Clientes IN MISDATOS.MDB
WHERE IDCliente Like 'A*';
```

En donde **MISDATOS.MDB** es el nombre de una base de datos de *Microsoft Access* que contiene la tabla *Clientes*.

Acceso a una base de datos externa de **dBASE III o IV**:

```
SELECT IDCliente FROM Clientes IN 'C:\DBASE\DATOS\VENTAS'
'dBASE IV';
WHERE IDCliente Like 'A*';
```

Para recuperar datos de una tabla de **dBASE III+** hay que utilizar '**dBASE III+;**' en lugar de '**dBASE IV;**'.

Acceso a una base de datos de **Paradox 3.x o 4.x**:

```
SELECT IDCliente FROM Clientes IN 'C:\PARADOX\DATOS\VENTAS'
'Paradox 4.x;' WHERE IDCliente Like 'A*';
```

Para recuperar datos de una tabla de **Paradox versión 3.x**, hay que sustituir **'Paradox 4.x;'** por **'Paradox 3.x;'**.

Acceso a una base de datos de **Btrieve**:

```
SELECT IDCliente FROM Clientes IN 'C:\BTRIEVE\DATOS\VENTAS\FILE.DDF'
'Btrieve;' WHERE IDCliente Like 'A*';
```

C:\BTRIEVE\DATOS\VENTAS\FILE.DDF es la ruta de acceso y nombre de archivo del archivo de definición de datos de **Btrieve**.

14. Omitir los Permisos de Ejecución

En entornos de bases de datos con permisos de seguridad para grupos de trabajo se puede utilizar la cláusula **WITH OWNERACCESS OPTION** para que el usuario actual adquiera los derechos de propietario a la hora de ejecutar la consulta. Su sintaxis es:

```
instrucción sql WITH OWNERACCESS
OPTION
SELECT Apellido, Nombre, Salario FROM
Empleados ORDER BY Apellido
WITH OWNERACCESS OPTION;
```

Esta opción requiere que esté declarado el acceso al fichero de grupo de trabajo (generalmente **system.mda** ó **system .mdw**) de la base de datos actual.

15. La Cláusula PROCEDURE

Esta cláusula es poco usual y se utiliza para crear una consulta a la misma vez que se ejecuta, opcionalmente define los parámetros de la misma. Su sintaxis es la siguiente:

```
PROCEDURE NombreConsulta Parámetro1
tipo1, .... , ParámetroN tipon ConsultaSQL
```

En donde:

Parte	Descripción
NombreConsulta	Es el nombre con se guardará la consulta en la base de datos.
Parámetro	Es el nombre de parámetro o de los parámetros de dicha consulta.
tipo	Es el tipo de datos del parámetro
ConsultaSQL	Es la consulta que se desea grabar y ejecutar.

```
PROCEDURE Lista_Categorias; SELECT DISTINCTROW
Nombre_Categoria,
ID_Categoría FROM Categorias ORDER BY Nombre_Categoria;
```

Asigna el nombre Lista_de_categorías a la consulta y la ejecuta.

```
PROCEDURE Resumen Fecha_Inicio DateTime,
Fecha_Final DateTime; SELECT
DISTINCTROW Fecha_Envio, ID_Pedido, Importe_Pedido, Format(Fecha_Envio,
"yyyy")
AS Año FROM Pedidos WHERE Fecha_Envio Between Fecha_Inicio And
Fecha_Final;
Asigna el nombre Resumen a la consulta e incluye
```

16 ANEXOS

16.1 Resolución de Problemas

16.1.1 Buscar Información duplicada en un campo de una tabla.

Para generar este tipo de consultas lo más sencillo es utilizar el asistente de consultas de **Access**, editar la sentencia **SQL** de la consulta y pegarla en nuestro código. No obstante este tipo de consulta se consigue de la siguiente forma:

```
SELECT DISTINCTROW Lista de Campos a Visualizar FROM Tabla
WHERE CampoDeBusqueda In (SELECT CampoDeBusqueda FROM Tabla As psudónimo
GROUP BY CampoDeBusqueda HAVING Count(*)>1 ) ORDER BY CampoDeBusqueda;
```

Un caso práctico, si deseamos localizar aquellos empleados con igual nombre y visualizar su código correspondiente, la consulta sería la siguiente:

```
SELECT DISTINCTROW Empleados.Nombre, Empleados.IdEmpleado
FROM Empleados WHERE Empleados.Nombre In (SELECT Nombre FROM
Empleados As Tmp GROUP BY Nombre HAVING Count(*)>1)
ORDER BY Empleados.Nombre;
```

16.1.2 Recuperar Registros de una tabla que no contengan registros relacionados en otra.

Este tipo de consulta se emplea en situaciones tales como saber que productos no se han vendido en un determinado periodo de tiempo,

```
SELECT DISTINCTROW Productos.IdProducto, Productos.Nombre FROM Productos
LEFT JOIN Pedidos ON Productos.IdProducto = Pedidos.IdProduct WHERE
(Pedidos.IdProducto Is Null) AND (Pedidos.Fecha Between #01-01-98# And
#01-30-98#);
```

La sintaxis es sencilla, se trata de realizar una unión interna entre dos tablas seleccionadas mediante un **LEFT JOIN**, estableciendo como condición que el campo relacionado de la segunda sea **Null**.

16.2 Utlizar SQL desde Visual Basic

Existen dos tipos de consultas **SQL**: las consultas de selección (nos devuelven datos) y las consultas de acción (aquellas que no devuelven ningún registro). Ambas pueden ser tratadas en *Visual Basic* pero de forma diferente.

Las consultas de selección se ejecutan recogiendo la información en un recordset previamente definido mediante la instrucción **openrecordset()**, por ejemplo:

```
Dim SQL as String
Dim RS as recordset

SQL = "SELECT * FROM Empleados;"
Set RS=MiBaseDatos.OpenRecordSet(SQL)
```

Si la consulta de selección se encuentra almacenada en una consulta de la base de datos:

```
Set RS=MiBaseDatos.OpenRecordset("MiConsulta")
```

Las consultas de acción, al no devolver ningún registro, no las podemos asignar a ningún recordset, en este caso la forma de ejecutarlas es mediante los métodos **Execute** y **ExecuteSQL** (para bases de datos **ODBC**), por ejemplo:

```
Dim SQL as string
SQL = "DELETE * FROM Empleados WHERE Categoria = 'Ordenanza';"
MiBaseDatos.Execute SQL
```

16.3 Funciones de Visual Basic utilizables en una Instrucción SQL

Función	Sintaxis	Descripción
Now	Variable= **Now**	Devuelve la fecha y la hora actual del sistema
Date	Variable=**Date**	Devuelve la fecha actual del sistema
Time	Variable=**Time**	Devuelve la hora actual del sistema
Year	Variable=**Year(**Fecha**)**	Devuelve los cuatro dígitos correspondientes al año de Fecha
Month	Variable=**Month(**Fecha**)**	Devuelve el número del mes del parámetro fecha.
Day	Variable=**Day(**Fecha**)**	Devuelve el número del día del mes del parámetro fecha.
Weekday	Variable=**Weekday(**Fecha**)**	Devuelve un número entero que representa el día de la semana del parámetro fecha.
Hour	Variable=**Hour(**Hora**)**	Devuelve un número entre 0 y 23 que representa la hora del parámetro Hora.
Minute	Variable=**Minute(**Hora**)**	Devuelve un número entre 0 y 59 que representa los minutos del parámetro hora.
Second	Variable=**Second(**Hora**)**	Devuelve un número entre 0 y 59 que representa los segundos del parámetro hora.

DatePart

Esta función devuelve una parte señalada de una fecha concreta. Su sintaxis es:

```
DatePart(Parte, Fecha, ComienzoSemana,
ComienzoAño)
```

Parte representa a la porción de fecha que se desea obtener, los posibles valores son:

Valor	Descripción
yyyy	Año
q	Trimestre
m	Mes
y	Día del año
d	Día del mes
w	Día de la semana
ww	Semana del año
h	Hora
m	Minutos
s	Segundos

ComienzoSemana indica el primer día de la semana. Los posibles valores son:

Valor	Descripción
0	Utiliza el valor pode efecto del sistema
1	Domingo (Valor predeterminado)
2	Lunes
3	Martes
4	Miércoles
5	Jueves
6	Viernes
7	Sábado

ComienzoAño indica cual es la primera semana del año; los posibles valores son:

Valor	Descripción
0	Valor del sistema
1	Comienza el año el 1 de enero (valor predeterminado).
2	Empieza con la semana que tenga al memos cuatro días en el nuevo año.

3 Empieza con la semana que esté contenida completamente en el nuevo año.

16.4 Evaluar valores antes de ejecutar la Consuta.

Dentro de una sentencia **SQL** podemos emplear la función **iif** para indicar las condiciones de búsqueda. La sintaxis de la función **iif** es la siguiente:

```
iif(Expresion,Valor1,Valor2)
```

En donde *Expresión* es la sentencia que evaluamos; si *Expresión* es verdadera entonces se devuelve *Valor1*, si *Expresión* es falsa se devuelve *Valor2*.

```
SELECT * Total FROM Empleados WHERE Apellido =
iff(TX_Apellido.Text <> '', TX_Apellido.Text, *) ;
```

Supongamos que en un formulario tenemos una casilla de texto llamada *TX_Apellido*. Si cuando ejecutamos esta consulta la casilla contiene algún valor se devuelven todos los empleados cuyo apellido coincida con el texto de la casilla, en caso contrario se devuelven todos los empleados.

```
<preSELECT Fecha, Producto, Cantidad, (iif(CodigoPostal>=28000 And
CodigoPostal <=28999,'Madrid','Nacional')) AS Destino FROM Pedidos;
```

Esta consulta devuelve los campos *Fecha, Nombre del Producto y Cantidad* de la tabla pedidos, añadiendo un campo al final con el valor *Madrid* si el código posta está dentro del intervalo, en caso contrario devuelve *Nacional*.

16.5 Un Pequeño Manual de Estilo

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. En lo que a mi respecta utilizo las siguiente normas a la hora de elaborar sentencias **SQL**:

1. Las cláusulas siempre las escribo 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 minúsculas.

Los Nombres de las Tablas, Campos y Consultas, los escribo siempre la primera letra con mayúsculas y el resto con minúsculas, en algunos casos utilizo el carácter "_" para definir mejor el nombre: Detalles_Pedidos.

Aunque con el motor Jet se pueden utilizar acentos y espacios en blanco para nombrar los campos, las tablas y las consultas no los utilizo porque cuando se exportar tablas a otros sistemas los acentos y los espacios en blanco pueden producir errores innecesarios.

Recuerda siempre que si utilizas espacios en blanco para llamar tablas o consultas cada vez que hagas referencias a ellos en una consulta debes incluir sus nombres entre corchetes.

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