INFORMATORIO

Parte 1: Introducción

Qué es el lenguaje SQL, qué veremos en el curso.

1.1.- Qué es SQL

Structured Query Language no es más que un **lenguaje estándar de comunicación** con bases de datos. Hablamos por tanto de un lenguaje normalizado que nos permite trabajar con cualquier tipo de lenguaje en combinación con cualquier tipo de base de datos (SQL Server, MySQL...).

El hecho de que sea estándar no quiere decir que sea idéntico para cada base de datos.

En efecto, determinadas bases de datos implementan funciones específicas que no tienen necesariamente que funcionar en otras.

Aparte de esta **universalidad**, el SQL posee otras dos características muy apreciadas. Por una parte, presenta una potencia y versatilidad notables que contrasta, por otra, con su accesibilidad de aprendizaje.

1.2.- Tipos de campo

Como sabemos una base de datos está compuesta de **tablas** donde almacenamos registros catalogados en función de distintos **campos** (características).

Un aspecto previo a considerar es la *naturaleza de los valores* que introducimos en esos campos. Dado que una base de datos trabaja con todo tipo de informaciones, es importante especificar qué tipo de valor le estamos introduciendo de manera a, por un lado, facilitar la búsqueda posteriormente y por otro, optimizar los recursos de memoria.

Cada base de datos introduce tipos de valores de campo que no necesariamente están presentes en otras.

Sin embargo, existe un conjunto de tipos que están representados en la totalidad de estas bases. Estos tipos comunes son los siguientes:

Alfanuméricos	Contienen cifras y letras.
Numéricos	Existen de varios tipos, principalmente, enteros (sin decimales) y reales (con decimales).
Booleanos	Poseen dos formas: Verdadero y falso (Sí o No)
Fechas	Almacenan fechas facilitando posteriormente su explotación. Almacenar fechas de esta forma posibilita ordenar los registros por fechas o calcular los días entre una fecha y otra...
Memos	Son campos alfanuméricos de longitud ilimitada. Presentan el inconveniente de no poder ser indexados (veremos más adelante lo que esto quiere decir).
Autoincrementables	Son campos numéricos enteros que incrementan en una unidad su valor para cada registro incorporado. Su utilidad resulta más que evidente: Servir de identificador ya que resultan exclusivos de un registro.

1.3.- Tipos de datos SQL

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

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*1038 a -1.401298*10-45 para valores negativos, 1.401298*10- 45 a 3.402823*1038 para valores positivos, y 0.

1.4.- Tipos de sentencias SQL y sus componentes sintácticos

Pasamos a describir los tipos de sentencias sql que podemos encontrarnos y sus componentes sintácticos.

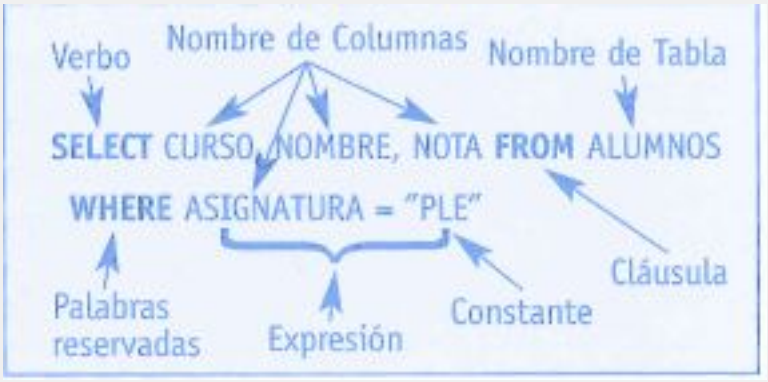
En SQL tenemos bastantes sentencias que se pueden utilizar para realizar diversas tareas.

SENTENCIA	DESCRIPCIÓN
DML **Manipulación de datos** **SELECT** **INSERT** **DELETE** **UPDATE**	 Recupera datos de la base de datos. Añade nuevas filas de datos a la base de datos. Suprime filas de datos de la base de datos. Modifica datos existentes en la base de datos
DDL **Definición de datos** **CREATE TABLE** **DROP TABLE** **ALTER TABLE** **CREATE VIEW** **DROP VIEW** **CREATE INDEX** **DROP INDEX**	 Añade una nueva tabla a la base de datos. Suprime una tabla de la base de datos. Modifica la estructura de una tabla existente. Añade una nueva vista a la base de datos. Suprime una vista de la base de datos. Construye un índice para una columna. Suprime el índice para una columna.
DCL **Control de acceso** **GRANT** **REVOKE** **Control de transacciones** **COMMIT** **ROLLBACK**	 Concede privilegios de acceso a usuarios. Suprime privilegios de acceso a usuarios Finaliza la transacción actual. Aborata la transacción actual.

1.4.1.- Componentes sintácticos

La mayoría de **sentencias SQL** tienen la misma estructura.

Todas comienzan por un verbo (select, insert, update, create), a continuación le siguen una o más cláusulas que nos dicen los datos con los que vamos a operar (from, where), algunas de estas son opcionales y otras obligatorias como es el caso del from.



INFORMATORIO

Parte 2: Inserción y modificación de datos

Vemos como insertar, modificar o borrar datos en nuestras tablas SQL.

2.1.- Creación de tablas

Explicamos la manera de **crear tablas** a partir de **sentencias SQL**. Definimos los **tipos de campos principales** y la **forma de especificar los índices**.

En general, la mayoría de las bases de datos poseen potentes editores de bases que permiten la creación rápida y sencilla de cualquier tipo de tabla con cualquier tipo de formato.

Sin embargo, una vez la base de datos está alojada en el servidor, puede darse el caso de que necesitemos introducir una nueva tabla ya sea con carácter temporal (para gestionar un carrito de compra por ejemplo) o bien permanente por necesidades concretas de nuestra aplicación.

En estos casos, podemos, a partir de una sentencia SQL, crear la tabla con el formato que deseemos lo cual nos puede ahorrar más de un quebradero de cabeza.

Este tipo de sentencias son especialmente útiles para bases de datos como Mysql, las cuales trabajan directamente con comandos SQL y no por medio de editores.

Para crear una tabla debemos especificar diversos datos: El nombre que le queremos asignar, los nombres de los campos y sus características. Además, puede ser necesario especificar cuáles de estos campos van a ser índices y de qué tipo van a serlo.

La sintaxis de creación puede variar ligeramente de una base de datos a otra ya que los tipos de campo aceptados no están completamente estandarizados.

2.1.1.- Sintaxis

```
Create Table nombre_tabla
(
nombre_campo_1 tipo_1
nombre_campo_2 tipo_2
nombre_campo_n tipo_n
Key(campo_x,...)
)
```

Pongamos ahora como ejemplo la creación una tabla de pedidos:

```
Create Table pedidos
(
id_pedido INT(4) NOT NULL AUTO_INCREMENT,
id_cliente INT(4) NOT NULL,
id_articulo INT(4)NOT NULL,

fecha DATE,
cantidad INT(4),
total INT(4), KEY(id_pedido,id_cliente,id_articulo)

)
```

En este caso creamos los campos id los cuales son considerados de tipo entero de una longitud especificada por el número entre paréntesis. Para `id_pedido` requerimos que dicho campo se incremente automáticamente (`AUTO_INCREMENT`) de una unidad a cada introducción de un nuevo registro para, de esta forma, automatizar su creación. Por otra parte, para evitar un mensaje de error, es necesario requerir que los campos que van a ser definidos como índices no puedan ser nulos (`NOT NULL`).

El campo fecha es almacenado con formato de fecha (**DATE**) para permitir su correcta explotación a partir de las funciones previstas a tal efecto.
Finalmente, definimos los índices enumerándolos entre paréntesis precedidos de la palabra **KEY o INDEX**.
Del mismo modo podríamos crear la tabla de artículos con una sentencia como ésta:

```
Create Table articulos
(
id_articulo INT(4) NOT NULL AUTO_INCREMENT,
titulo VARCHAR(50),
autor VARCHAR(25),
editorial VARCHAR(25),
precio REAL,
KEY(id_articulo)
)
```

En este caso puede verse que los campos alfanuméricos son introducidos de la misma forma que los numéricos. Volvemos a recordar que en tablas que tienen campos comunes es de vital importancia definir estos campos de la misma forma para el buen funcionamiento de la base.

Muchas son las opciones que se ofrecen al generar tablas. No vamos a tratarlas detalladamente pues sale de lo estrictamente práctico.

2.2.- Estructuras de las tablas en SQL

Una **base de datos** en un sistema relacional está compuesta por un conjunto de tablas, que corresponden a las relaciones del modelo relacional.

En la terminología usada en SQL no se alude a las relaciones, del mismo modo que no se usa el término atributo, pero sí la palabra columna, y no se habla de tupla, sino de línea.

2.2.1.- Creación de Tablas Nuevas

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

En donde:

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 índice a crear. Esta cláusula en opcional.
Índice multicampos	Es una cláusula CONSTRAINT que define el tipo de índice multicampos a crear. Un índice multicampo es aquel que está indexado por el contenido de varios campos. Esta cláusula en opcional.

2.2.2.- 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 [,...])]}
```

En donde:

nombre	Es el nombre del índice que se va a crear.
primario n	Es el nombre del campo o de los campos que forman el índice primario.
único n	Es el nombre del campo o de los campos que forman el índice de clave única.
ref n	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.

Índice	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 estuviera formada por los campos referenciados.

2.2.3.- Creación de Índices

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:

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	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 índice que sea "Clave Principal". Si un índice es clave principal implica que no puede contener valores nulos ni duplicados.

2.2.4.- 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:

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.

2.3.- Añadir un nuevo registro

Los registros pueden ser introducidos a partir de sentencias que emplean la instrucción Insert.

La sintaxis utilizada es la siguiente:

```
Insert Into nombre_tabla (nombre_campo1, nombre_campo2,...) Values
(valor_campo1, valor_campo2...)
```

Un ejemplo sencillo a partir de nuestra tabla modelo es la introducción de un nuevo cliente lo cual se haría con una instrucción de este tipo:

```
Insert Into clientes (nombre, apellidos, direccion, poblacion, codigopostal,
email, pedidos) Values ('Perico', 'Palotes', 'Percebe nº13', 'Lepe',
'123456', 'perico@empresa.com', 33)
```

Como puede verse, los campos no numéricos o booleanos van delimitados por ‘ ’.
También resulta interesante ver que el código postal lo hemos guardado como un campo no numérico. Esto es debido a que en determinados países los códigos postales contienen también letras.

Nota: Si deseamos practicar con una base de datos que está vacía primero debemos crear las tablas que vamos a llenar. Las tablas también se crean con sentencias SQL.
Aunque, de todos modos, puede que sea más cómodo utilizar un programa con interfaz gráfica,

Por supuesto, no es imprescindible rellenar todos los campos del registro. Eso sí, puede ser que determinados campos sean necesarios. Estos campos necesarios pueden ser definidos cuando construimos nuestra tabla mediante la base de datos.

Nota: Si no insertamos uno de los campos en la base de datos se inicializará con el valor por defecto que hayamos definido a la hora de crear la tabla. Si no hay valor por defecto, probablemente se inicialice como NULL (vacío), en caso de que este campo permita valores nulos. Si ese campo no permite valores nulos (eso se define también al crear la tabla) lo más seguro es que la ejecución de la sentencia SQL nos de un error.

Resulta muy interesante, ya veremos más adelante el por qué, el introducir durante la creación de nuestra tabla un campo autoincrementable que nos permita asignar un único número a cada uno de los registros. De este modo, nuestra tabla clientes presentaría para cada registro un número exclusivo del cliente el cual nos será muy útil cuando consultemos varias tablas simultáneamente.

2.4.- Borrar un registro

Para borrar un registro usamos la instrucción **`Delete`**. En este caso debemos especificar cuál o cuáles son los registros que queremos borrar. Es por ello necesario establecer una selección que se llevará a cabo mediante la cláusula **`Where`**.

```
Delete From nombre_tabla Where condiciones_de_selección
```

Nota: Si deseamos practicar con una base de datos que está vacía primero debemos crear las tablas que vamos a llenar. Las tablas también se crean con sentencias SQL.

Si queremos por ejemplo borrar todos los registros de los clientes que se llaman **Perico** lo haríamos del siguiente modo:

```
Delete From clientes Where nombre='Perico'
```

Hay que tener cuidado con esta instrucción ya que si no especificamos una condición con **`Where`**, lo que estamos haciendo es borrar toda la tabla:

```
Delete From clientes
```

2.5.- Actualizar un registro

`Update` es la instrucción que nos sirve para modificar nuestros registros. Como para el caso de `Delete`, necesitamos especificar por medio de Where cuáles son los registros en los que queremos hacer efectivas nuestras modificaciones. Además, obviamente, tendremos que especificar cuáles son los nuevos valores de los campos que deseamos actualizar. La sintaxis es de este tipo:

```
Update nombre_tabla Set nombre_campo1 = valor_campo1, nombre_campo2 =
valor_campo2,...

Where condiciones_de_selección
```

Un ejemplo aplicado:

```
Update clientes Set nombre='José' Where nombre='Pepe'
```

Mediante esta sentencia cambiamos el nombre Pepe por el de José en todos los registros cuyo nombre sea Pepe.
Aquí también hay que ser cuidadoso de no olvidarse de usar Where, de lo contrario, modificaríamos todos los registros de nuestra tabla.

INFORMATORIO

Parte 3: Búsqueda y selección de datos en SQL

Estudiamos a fondo todo lo relacionado con la sentencia select dentro del lenguaja SQL.

Utilizar esta tabla de ejemplo:

CLIENTES
nombre VARCHAR(30)
apellidos VARCHAR(30)
direccion VARCHAR(50)
población VARCHAR(20)
codigopostal lNT(4)
email VARCHAR(60)
pedidos INT(3)

3.1.- Selección de tablas I

Cómo realizar selecciones eficientemente. Ejemplos prácticos.

La selección total o parcial de una tabla se lleva a cabo mediante la instrucción Select.
En dicha selección hay que especificar:

-Los campos que queremos seleccionar
-La tabla en la que hacemos la selección

En nuestra tabla modelo de clientes podríamos hacer por ejemplo una selección del nombre y dirección de los clientes con una instrucción de este tipo:

```
Select nombre, dirección From clientes
```

Si quisiésemos seleccionar todos los campos, es decir, **toda la tabla**, podríamos utilizar el comodín * del siguiente modo:

```
Select * From clientes
```

Resulta también muy útil el filtrar los registros mediante condiciones que vienen expresadas después de la **cláusula Where**.

Si quisiésemos mostrar los clientes de una determinada ciudad usaríamos una expresión como esta:

```
Select * From clientes Where poblacion Like 'Madrid'
```

Además, podríamos **ordenar los resultados** en función de uno o varios de sus campos. Para este ultimo ejemplo los podríamos ordenar por nombre así:

```
Select * From clientes Where poblacion Like 'Madrid' Order By nombre
```

Teniendo en cuenta que puede haber más de un cliente con el mismo nombre, podríamos dar un segundo criterio que podría ser el apellido:

```
Select * From clientes Where poblacion Like 'Madrid' Order By nombre, apellido
```

Si invirtiésemos el orden « nombre,apellido » por « apellido, nombre », el resultado sería distinto. Tendríamos los clientes ordenados por apellido y aquellos que tuviesen apellidos idénticos se subclasificarían por el nombre.

Es posible también **clasificar por orden inverso**. Si por ejemplo quisiésemos ver nuestros clientes por orden de pedidos realizados teniendo a los mayores en primer lugar escribiríamos algo así:

```
Select * From clientes Order By pedidos Desc
```

Una opción interesante es la de efectuar **selecciones sin coincidencia**. Si por ejemplo buscásemos el saber en qué ciudades se encuentran nuestros clientes sin necesidad de que para ello aparezca varias veces la misma ciudad usaríamos una sentencia de esta clase:

```
Select Distinct poblacion From clientes Order By poblacion
```

Así evitaríamos ver repetido Madrid tantas veces como clientes tengamos en esa población.

3.2.- Selección de tablas II

Lista de operadores y ejemplos prácticos para realizar selecciones.

Hemos querido compilar a modo de tabla ciertos operadores que pueden resultar útiles en determinados casos. Estos operadores serán utilizados después de la cláusula Where y pueden ser **combinados hábilmente mediante paréntesis** para optimizar nuestra selección a muy altos niveles.

Operadores matemáticos:

>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	Distinto
=	Igual

Operadores lógicos

- And
- Or
- Not

Otros operadores

Like	Selecciona los registros cuyo valor de campo se asemeje, no teniendo en cuenta mayúsculas y minúsculas.
In y Not In	Da un conjunto de valores para un campo para los cuales la condición de selección es (o no) valida
Is Null y Is Not Null	Selecciona aquellos registros donde el campo especificado esta (o no) vacío.
Between... And	Selecciona los registros comprendidos en un intervalo
Distinct	Selecciona los registros no coincidentes
Desc	Clasifica los registros por orden inverso

Comodines	
*	Sustituye a todos los campos
%	Sustituye a cualquier cosa o nada dentro de una cadena
_	Sustituye un solo carácter dentro de una cadena

Veamos a continuación aplicaciones prácticas de estos operadores.

En esta sentencia seleccionamos todos los clientes de Madrid cuyo nombre no es Pepe. Como puede verse, empleamos **Like** en lugar de **=** simplemente para evitar inconvenientes debido al empleo o no de mayúsculas.

```
Select * From clientes Where poblacion Like 'madrid' And Not nombre Like 'Pepe'
```

Si quisiéramos recoger en una selección a los clientes de nuestra tabla cuyo **apellido comienza por A y cuyo número de pedidos esta comprendido entre 20 y 40**:

```
Select * From clientes Where apellidos like 'A%' And pedidos Between 20 And 40
```

El operador **In**, lo veremos más adelante, es muy práctico para consultas en varias tablas. Para casos en una sola tabla es empleado del siguiente modo:

```
Select * From clientes Where poblacion In ('Madrid','Barcelona','Valencia')
```

De esta forma **seleccionamos aquellos clientes que vivan en esas tres ciudades**.

3.3.- Selección de tablas III

Cómo realizar selecciones sobre varias tablas. Ejemplos prácticos basados en una aplicación de e-comercio.

Una base de datos puede ser considerada como un conjunto de tablas. Estas tablas en muchos casos están relacionadas entre ellas y se complementan unas con otras.

Refiriéndonos a nuestro clásico ejemplo de una base de datos para una aplicación de ecommerce, la tabla clientes de la que hemos estado hablando puede estar perfectamente coordinada con una tabla donde almacenamos los pedidos realizados por cada cliente. Esta tabla de pedidos puede a su vez estar conectada con una tabla donde almacenamos los datos correspondientes a cada artículo del inventario.

De este modo podríamos fácilmente **obtener informaciones contenidas** en esas tres tablas como puede ser la *designación del artículo más popular* en una determinada región donde la designación del artículo sería obtenida de la tabla de artículos, la popularidad (cantidad de veces que ese artículo ha sido vendido) vendría de la tabla de pedidos y la región estaría comprendida obviamente en la tabla clientes.

Este tipo de organización basada en múltiples tablas conectadas nos permite trabajar con tablas mucho más manejables a la vez que nos evita copiar el mismo campo en varios sitios ya que podemos acceder a él a partir de una simple llamada a la tabla que lo contiene.

Supongamos que queremos enviar un mailing a todos aquellos que hayan realizado un pedido ese mismo día. Podríamos escribir algo así:

```
Select clientes.apellidos, clientes.email
From clientes,pedidos
Where pedidos.fecha like '25/02/00' And pedidos.id_cliente= clientes.id_cliente
```

Como puede verse esta vez, después de la cláusula From, introducimos el nombre de las dos tablas de donde sacamos las informaciones. Además, el nombre de cada campo va precedido de la tabla de origen separados ambos por un punto. En los campos que poseen un nombre que solo aparece en una de las tablas, no es necesario especificar su origen aunque a la hora de leer la sentencia puede resultar más claro el precisarlo. En este caso el campo fecha podría haber sido designado como "fecha" en lugar de "pedidos.fecha".

Veamos otro ejemplo más para consolidar estos nuevos conceptos. Esta vez queremos ver el título del libro correspondiente a cada uno de los pedidos realizados:

```
Select pedidos.id_pedido, articulos.titulo
From pedidos, articulos
Where pedidos.id_articulo=articulos.id_articulo
```

En realidad la filosofía continua siendo la misma que para la consulta de una única tabla.

3.4.- Selección de tablas IV

El empleo de funciones para la explotación de los campos numéricos y otras utilidades. Ejemplos prácticos.

Además de los criterios hasta ahora explicados para realizar las consultas en tablas, SQL permite también aplicar un conjunto de funciones predefinidas. Estas funciones, aunque básicas, pueden ayudarnos en algunos momentos a expresar nuestra selección de una manera más simple sin tener que recurrir a operaciones adicionales por parte del script que estemos ejecutando.

Algunas de estas funciones son representadas en la tabla siguiente :

Sum(campo)	Calcula la suma de los registros del campo especificado
Avg(Campo)	Calcula la media de los registros del campo especificado
Count(*)	Nos proporciona el valor del numero de registros que han sido
Max(Campo)	Nos indica cual es el valor máximo del campo
Min(Campo)	Nos indica cual es el valor mínimo del campo

Dado que el campo de la función no existe en la base de datos, sino que lo estamos generando virtualmente, esto puede crear inconvenientes cuando estamos trabajando con nuestros scripts a la hora de tratar su valor y su nombre de campo. Es por ello que el valor de la **función ha de ser recuperada a partir de un alias** que nosotros especificaremos en la sentencia SQL a partir de la instrucción **AS**. La cosa podría quedar así:

```
Select Sum(total) As suma_pedidos
From pedidos
```

A partir de esta sentencia calculamos la suma de los valores de todos los pedidos realizados y almacenamos ese valor en un campo virtual llamado suma_pedidos que podrá ser utilizado como cualquier otro campo por nuestras paginas dinámicas.

Por supuesto, todo lo visto hasta ahora puede ser aplicado en este tipo de funciones de modo que, por ejemplo, podemos establecer condiciones con la cláusula Where construyendo sentencias como esta:

```
Select Sum(cantidad) as suma_articulos
From pedidos Where id_articulo=6
```

Esto nos proporcionaría la cantidad de **ejemplares de un determinado libro que han sido vendidos**.

Otra propiedad interesante de estas funciones es que **permiten realizar operaciones con varios campos dentro de un mismo paréntesis**:

```
Select Avg(total/cantidad)
From pedidos
```

Esta sentencia da como resultado el **precio medio al que se están vendiendo los libros**. Este resultado no tiene por qué coincidir con el del **precio medio de los libros presentes en el inventario**, ya que, puede ser que la gente tenga tendencia a comprar los libros caros o los baratos:

```
Select Avg(precio) as precio_venta
From articulos
```

Una cláusula interesante en el uso de funciones es Group By. Esta cláusula nos permite agrupar registros a los cuales vamos a aplicar la función. Podemos por ejemplo calcular el **dinero gastado por cada cliente**:

```
Select id_cliente, Sum(total) as suma_pedidos
From pedidos
Group By id_cliente
```

O saber el **numero de pedidos que han realizado**:

```
Select id_cliente, Count(*) as numero_pedidos
From pedidos
Group By id_cliente
```

INFORMATORIO

Parte 3: Búsqueda y selección de datos en SQL

Estudiamos a fondo todo lo relacionado con la sentencia select dentro del lenguaja SQL.

3.5.- 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.

Este conjunto de registros puede ser modificable.

3.5.1.- 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, Teléfono
FROM
   Clientes
```

Esta sentencia devuelve un conjunto de resultados con el campo nombre y teléfono de la tabla clientes.

3.5.2.- Devolver Literales

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

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

3.5.3.- Ordenar los registros

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

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

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

Se pueden ordenar los registros por 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 cláusula (ASC - se toma este valor por defecto) ó descendente (DESC)

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

3.5.4.- Uso de Indices de las tablas

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

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

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

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

3.5.5.- Consultas con Predicado

El **predicado** se incluye entre la cláusula 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
DISTINCTOW	Omite los registros duplicados basándose en la totalidad del registro y no sólo en los campos seleccionados.

3.5.6.- 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 y devuelve todos y cada uno de sus campos. No es conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.

```
SELECT ALL
FROM
   Empleados

SELECT *
FROM
   Empleados
```

3.5.7.- 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 de 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 entero sin signo. TOP no afecta a la posible actualización de la consulta.

3.5.8.- 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.

3.5.9.- ALIAS

En determinadas circunstancias es necesario asignar un nombre a alguna columna determinada de un conjunto devuelto, otras veces por simple capricho o porque estamos recuperando datos de diferentes tablas y resultan tener un campo con igual nombre. 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 DISTINC(Apellido) AS Empleado
FROM Empleados
```

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

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

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

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

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

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

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

3.5.10.- 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 externas. 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
   Apellido AS Empleado
FROM
   Empleados IN'c: databasesgestion.mdb'
```

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

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

Parte 3: Búsqueda y selección de datos en SQL

Estudiamos a fondo todo lo relacionado con la sentencia select dentro del lenguaja SQL.

3.6.- Criterios de selección en SQL

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

Banco de Datos	**Sintaxis**
SQL-SERVER	Fecha = #mm-dd-aaaa#
ORACLE	Fecha = to_date('YYYYDDMM','aaaammdd',)
ACCESS	Fecha = #mm-dd-aaaa#

Referente a los valores lógicos True o False cabe destacar que no son reconocidos en ORACLE, ni en este sistema de bases de datos ni en SQL-SERVER existen los campos de tipo "SI/NO" de ACCESS; en estos sistemas se utilizan los campos BIT que permiten almacenar valores de 0 ó 1. Internamente, ACCESS, almacena en estos campos valores de 0 ó -1, así que todo se complica bastante, pero aprovechando la coincidencia del 0 para los valores FALSE, se puede utilizar la

sintaxis siguiente que funciona en todos los casos: si se desea saber si el campo es falso "... CAMPO = 0" y para saber los verdaderos "CAMPO <> 0".

3.6.1.- Operadores Lógicos

Los operadores lógicos mas comunes en SQL son: AND, OR, Is, 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 para AND y OR:

<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

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')
```

3.6.2.- 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)
```

3.6.3.- 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 una cadena de 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 en SQL Server para comprobar expresiones con diferentes modelos.

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

En determinado motores de bases de datos, esta cláusula, no reconoce el asterisco como carácter comodín y hay que sustituirlo por el carácter tanto por ciento (%).

3.6.4.- El Operador In

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

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

SELECT *
FROM
    Pedidos
WHERE

    Provincia In ('Madrid', 'Barcelona', 'Sevilla')
```

3.6.5.- 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 anteriores. 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
   IdProducto, Existencias
FROM
   Productos
WHERE
    Existencias <= NuevoPedido
```

```
SELECT *
FROM
  Pedidos
WHERE
  FechaEnvio = #05-30-1994#
```

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

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

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

```
SELECT
    Apellidos, Salario
FROM
    Empleados
WHERE
 Apellidos Between 'Lon' And 'Tol'
```

```
SELECT
    IdPedido, FechaPedido
FROM
    Pedidos
WHERE
FechaPedido Between #01-01-1994# And #12-31-1994#
```

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

3.7.- Criterios de selección en SQL II

Seguimos con el group by, avg, sum y con el compute de sql-server.

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
    IdFamilia, Sum(Stock) AS StockActual
FROM
     Productos
GROUP BY

     IdFamilia
```

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
    IdFamilia, Sum(Stock) AS StockActual
FROM
     Productos
GROUP BY
     IdFamilia
HAVING
     StockActual > 100
    AND

    NombreProducto Like BOS*
```

3.7.1.- 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
```

3.7.2.- 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
```

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

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

    Pedidos
```

3.7.3.- 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'
```

3.7.4.- Sum

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

```
Sum(expr)
```

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

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

INFORMATORIO

Parte 3: Búsqueda y selección de datos en SQL

Estudiamos a fondo todo lo relacionado con la sentencia select dentro del lenguaja SQL.

3.8.- Subconsultas en SQL

Defenimos lo que significa subconsulta y mostramos las diferentes subconsultas que se pueden hacer.

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)
expresión [NOT] EXISTS (instrucción sql)

En donde:

Comparación Es una expresión y un operador de comparación que compara la expresión con el resultado de la subconsulta.
Expresión Es una expresión por la que se busca el conjunto resultante de la subconsulta.
Instrucción SQL Es una instrucción SELECT, que sigue el mismo formato y reglas que cualquier otra instrucción SELECT. Debe ir entre paréntesis.

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

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

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

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

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

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

Inversamente se puede utilizar NOT IN para recuperar únicamente aquellos registros de la consulta principal para los que no hay ningún registro de la subconsulta que contenga un valor igual.
El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro. Supongamos que deseamos recuperar todos aquellos clientes que hayan realizado al menos un pedido:

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

1. Lenguaje DDL de SQL

El DDL (Data Definition Language) o Lenguaje de Definición de Datos es la parte de SQL dedicada a la definición de los datos. Las sentencias DDL son las siguientes:

• CREATE: se utiliza para crear objetos como bases de datos, tablas, vistas, índices, triggers y procedimientos almacenados.

• DROP: se utiliza para eliminar los objetos de la base de datos.

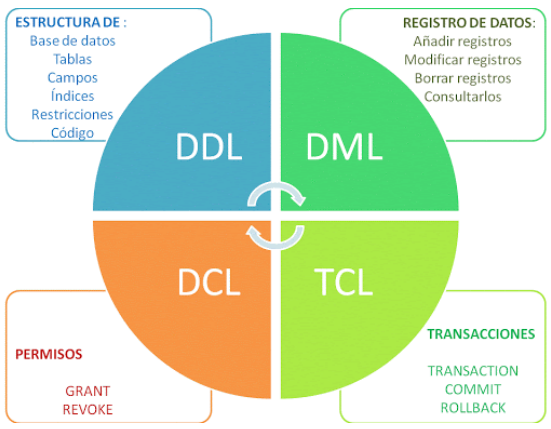
• ALTER: se utiliza para modificar los objetos de la base de datos.

• SHOW: se utiliza para consultar los objetos de la base de datos.

Otras sentencias de utilidad que usaremos en este curso son:

• USE: se utiliza para indicar la base de datos con la que queremos trabajar.

• DESCRIBE: se utiliza para mostrar información sobre la estructura de una tabla.



2. Manipulación de Bases de Datos

2.1 Crear una base de datos

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nombre_base_datos;
```

• DATABASE y SCHEMA son sinónimos.

• IF NOT EXISTS crea la base de datos sólo si no existe una base de datos con el mismo nombre.

Ejemplos:

Si no especificamos el set de caracteres en la creación de la base de datos, se usará `latin1` por defecto.

```
CREATE DATABASE nombre_base_datos;
```

Las bases de datos que vamos a crear durante el curso usarán el set de caracteres `utf8` o `utf8mb4`.

```
CREATE DATABASE nombre_base_datos CHARACTER SET utf8;
```

2.1.1 Conceptos básicos sobre la codificación de caracteres

Unicode es un set de caracteres universal, un estándar en el que se definen todos los caracteres necesarios para la escritura de la mayoría de los idiomas hablados en la actualidad. El estándar Unicode describe las propiedades y algoritmos necesarios para trabajar con los caracteres Unicode y este estándar es gestionado por el consorcio Unicode.

Los formatos de codificación que se pueden usar con Unicode se denominan UTF-8, UTF-16 y UTF-32.

• UTF-8 utiliza 1 byte para representar caracteres en el set ASCII, 2 bytes para caracteres en otros bloques alfabéticos y 3 bytes para el resto del BMP (Basic Multilingual Plane), que incluye la mayoría de los caracteres utilizados frecuentemente. Para los caracteres complementarios se utilizan 4 bytes.

• UTF-16 utiliza 2 bytes para cualquier carácter en el BMP y 4 bytes para los caracteres complementarios.

• UTF-32 emplea 4 bytes para todos los caracteres.

Se recomienda la lectura del artículo Codificación de caracteres: conceptos básicos publicado por la W3C.org

2.2 Eliminar una base de datos

```
DROP {DATABASE | SCHEMA} [IF EXISTS] nombre_base_datos;
```

• DATABASE y SCHEMA son sinónimos.

• IF EXISTS elimina la la base de datos sólo si ya existe.

Ejemplo:

DROP DATABASE nombre_base_datos;

2.3 Modificar una base de datos

```
ALTER {DATABASE | SCHEMA} [nombre_base_datos] alter_specification [,
alter_especification] ...
```