LENGUAJE ESTRUCTURADO DE CONSULTAS SQL

FAVA - Formación en Ambientes Virtuales de Aprendizaje.

SENA - Servicio Nacional de Aprendizaje.

Estructura de contenidos

Pág.

Introducción 4
Mapa de contenido 5
1. SQL 6
1.1 Definición 6
1.2 Historia 6
1.3 Características 7
1.3.1 Lenguaje de definición de datos – DDL 8
1.3.1.1 Creación de la base de datos 8
1.3.1.2 Creación de las tablas 8
1.3.1.3 Modificación de las tablas 10
1.3.1.4 Eliminación de tablas y base de datos 11
1.3.2 Lenguaje de manipulación de datos DML 12
1.3.2.1 Inserción de datos 12
1.3.2.2 Modificación de datos 12
1.3.2.3 Eliminación de registros 13
1.3.2.4. Consulta de datos 14
1.3.3 Alias de columnas 15
1.3.4 La cláusula WHERE 16
1.3.5 Las expresiones 16
1.3.5.1 Tipos de operadores 16
1.3.5.2 Otros elementos del lenguaje 18
1.3.6 Predicados 18
1.3.6.1 BETWEEN 21
1.3.6.2 Like 21
1.3.6.3 Pertenencia a un conjunto (IN) 22
1.3.7 La unión de tablas UNION 23
1.3.8 Cláusula Order By 24
2. Consultar datos, provenientes de diferentes tablas 25

2.1 Producto cartesiano 26
2.1.1 El producto cartesiano CROSS JOIN 27
2.1.2 La composición interna INNER JOIN 29
2.1.3 Combinaciones Externas La Composición externa LEFT, RIGHT y FULL OUTER JOIN 30
2.1.4 Autocombinaciones – SELF JOIN 33
2.2. Funciones de agregado 34
2.3. La cláusula GROUP BY 35
2.4 La cláusula HAVING 36
3. Subconsultas 37
Glosario 39
Bibliografía 40
Control del documento 41

Introducción

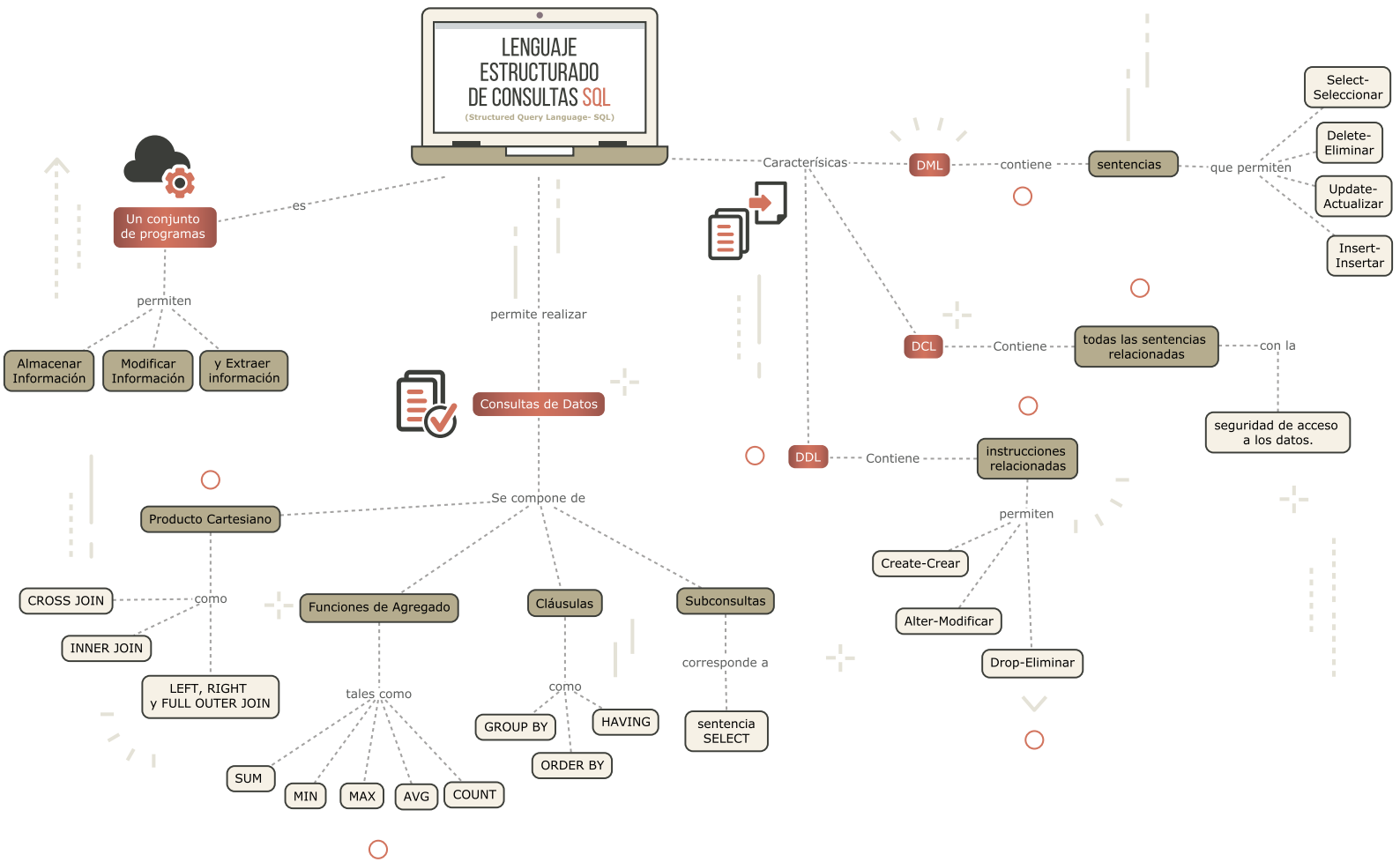
Un Sistema de Gestión de Bases de Datos (SGBD), es un grupo de programas que permiten modificar, almacenar, extraer, manipular y consultar determinada información dentro de una base de datos por el usuario.

Dentro de estos sistemas existe una interfaz que facilita a los usuarios la realización de diferentes actividades sobre los datos. Es por ello, que para manipular datos se utiliza una herramienta denominada Lenguaje Estructurado de Consultas (SQL) por sus siglas en inglés: Structured Query Language.

En este material de estudio se mencionan las características principales de este lenguaje y la sintaxis para construir estructuras y objetos de datos, insertar datos, modificarlos, borrar información y la estructura de consulta básica.

Las consultas pueden tener diferentes grados de complejidad, desde las que extraen la información de una tabla o complementando la información desde diferentes tablas, la incorporación de funciones de tipo estadístico sobre los datos y la generación de grupos de datos para crear niveles de resumen, las cuales son implementadas en los sistemas de información a través de las consultas, informes o reportes a generar.

Mapa de contenido



Desarrollo de contenidos

1. SQL

1.1 Definición

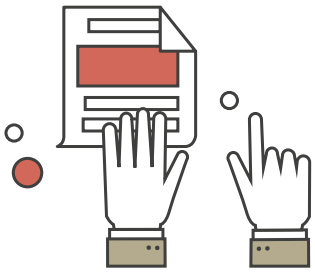
El lenguaje estructurado de consultas, más comúnmente conocido como SQL por sus siglas en inglés (Structured Query Language), es el lenguaje por el cual se puede acceder a las bases de datos relacionales. Utiliza instrucciones que se denominan “sentencias”, para construir el modelo relacional, consultar datos, eliminar o agregar nuevos datos, modificar los existentes y en general actividades de administración sobre los diferentes objetos que componen las bases de datos.

1.2 Historia

Creado en los laboratorios de IBM en la década de los 70, se denominó inicialmente SEQUEL (Structured English Query Language), el cual se basaba en la propuesta de modelo relacional proporcionada por Codd, posteriormente este lenguaje fue liberado y se estandarizó como el lenguaje a ser usado por los diferentes sistemas de gestión de base de datos que se crearon. En 1986 fue estandarizado por el ANSI (American National Standards Institute), lo que comúnmente se conoce como “SQL-86” y posteriormente adoptado por la ISO.

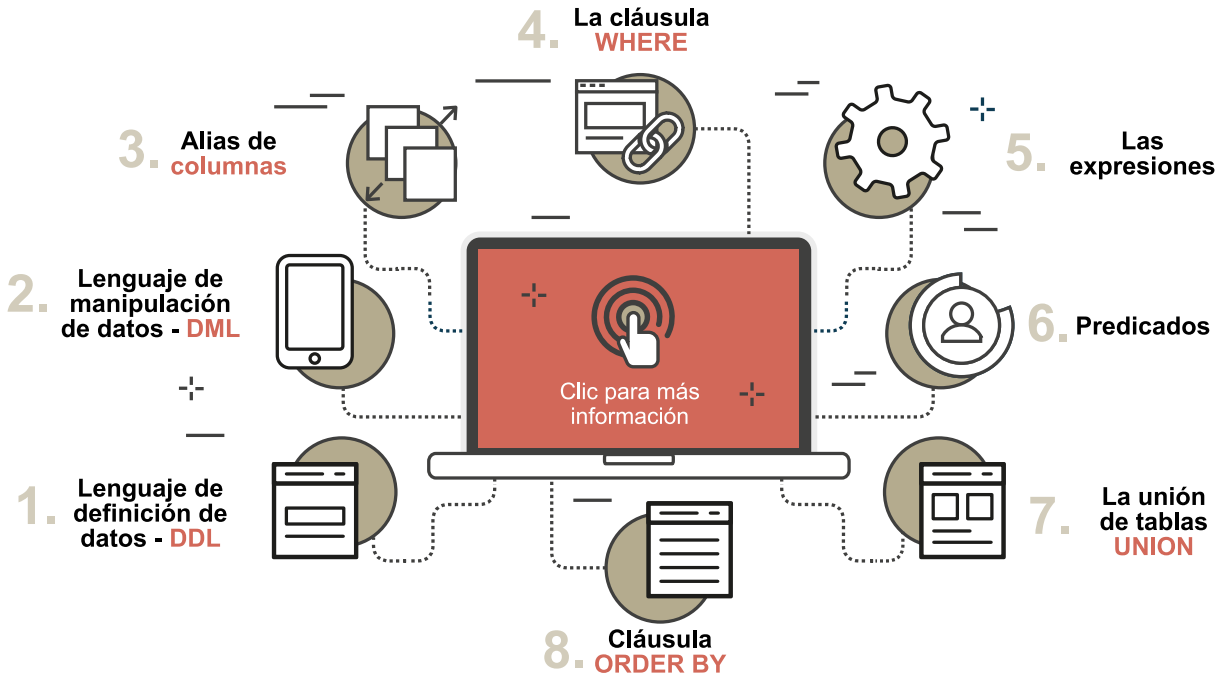
Con el paso del tiempo ha sufrido la adición de nuevas características relacionadas con la programación orientada a objetos, uso de XML y diferentes elementos que le han permitido seguir vigente en el procesamiento de datos.

1.3 Características



El SQL se comporta como un lenguaje de “alto nivel”, con una estructura declarativa de sentencias que posee una sintaxis particular y estándar. Se han clasificado sus instrucciones de acuerdo con su funcionalidad generando las siguientes categorías:

- **DDL (Lenguaje de definición de datos):** conocido por sus siglas en inglés (Data Definition Language), contiene todas las instrucciones relacionadas con la creación, modificación y eliminación de objetos de la base de datos, como son la base de datos, tabla, índices, vistas, procedimientos, funciones, triggers, entre otros.
- **DML (Lenguaje de manipulación de datos):** se identifica por las siglas DML (Data Manipulation Language), en el cual se encuentran las sentencias que permiten manipular los datos a partir de sentencias de inserción, modificación, eliminación y consulta de registros de datos.
- **DCL (Lenguaje de control de datos):** se identifica por las siglas DCL (Data Control Language), y contiene todas las sentencias relacionadas con la seguridad de acceso a los datos.



1.3.1 Lenguaje de definición de datos – DDL

El lenguaje de definición de datos puede tener algunas variaciones respecto al SMBD (Sistema Manejador de Base de Datos) que se utilice para manipularlo, sin embargo existe una codificación estándar que permite identificar una serie de características del lenguaje y las sentencias utilizadas.

A continuación se describe la construcción de bases de datos, tablas, la modificación y eliminación de las mismas.

1.3.1.1 Creación de la base de datos

En el DDL (acciones sobre la definición de la base de datos), se tienen 3 elementos básicos:

- CREATE (Crear).
- ALTER (Modificar).
- DROP (Eliminar).

La base de datos es la estructura de almacenamiento principal, la sentencia que se utiliza para su construcción es **CREATE**. Su sintaxis se muestra en el siguiente ejemplo para crear una base de datos llamada misena, se haría lo siguiente:

```
CREATE DATABASE misena;
```

1.3.1.2 Creación de las tablas

Una vez que se ha construido la base de datos, se deben construir las tablas en su interior, es decir que debe estar predefinido o seleccionado ese espacio de trabajo. Para construir las tablas que conforman las bases de datos, se utiliza la sentencia **CREATE TABLE**. Con esta sentencia se crea la estructura de la tabla, la cual permite definir las columnas que tendrá y ciertas restricciones que deben cumplir esas columnas.

Ejemplo para crear una tabla llamada Programa:

```
CREATE TABLE Programa (.....);
```

Es necesario hacer claridad respecto a las posibles restricciones que se manejan para las columnas de una tabla. Las restricciones son también conocidas como **constraints**, estas representan características particulares que tiene una columna y determinan entre

otras, reglas sobre los contenidos, tipos de datos, límites, relación de la columna con otras columnas, otros registros de la misma tabla o en otras tablas.

A continuación se presenta a manera de resumen las restricciones que se pueden asignar a las columnas en una tabla, el nivel identifica si afecta a nivel de registro (fila).

Nivel	Constraint – Palabra Clave	Descripción
Registro	**NOT NULL**	La columna o campo deberá contener obligatoriamente un valor. Los campos que se encuentran sin ningún tipo de información se denominan campos nulos y se identifican como NULL.
Registro	**PRIMARY KEY**	Esta columna es la que identifica en forma única cada fila de la tabla, se denomina como clave principal o llave primaria.
Columna	**UNIQUE**	Identifica una columna que no puede tener valores repetidos en toda la tabla.
Registro	**DEFAULT**	Permite definir un valor preestablecido para esa columna, es decir que si no se insertan datos en ese campo, el sistema almacenará el valor que por defecto se definió.
Registro	**CHECK**	Corresponde a condiciones que se pueden definir para aceptar o no datos en los campos. Estas reglas pueden referirse a la misma columna, p.e. (edad > 18) o a otras columnas de la misma tabla.
Tabla	**FOREIGN KEY**	Define la llave foránea de la tabla que puede ser un campo o una combinación de ellos y representa el enlace o relación con otras tablas. El valor que se almacena en esta columna debe estar contenido en otra tabla.

```
CREATE TABLE <Nombre_de_la_Tabla>
(<Nombre_de_la_columna1>
<Tipo de dato><Restricción>,
<Nombre_de_la_columna2>
<Tipo de dato><Restricción>,
<Nombre_de_la_columnan>
<Tipo de dato><Restricción>)
```

Consideraciones

Los nombres de la tabla y columnas, deben conservar las mismas características de un identificador, es decir, no espacios en blanco, de preferencia usar los nemotécnicos definidos en la organización, nombres significativos, no iniciar con números, entre otros. Los tipos de datos están directamente relacionados con el SMBD que se utilice, para los ejemplos que se presentan a continuación se utilizarán tipos de datos y expresiones propias de MySql.

Ejemplo

De acuerdo con la base de datos misena, se procede a crear las respectivas tablas, para crear la tabla “Aprendiz”, se debe tener previamente definida la estructura de datos, producto de los procesos de modelamiento o normalización vista anteriormente:

Nombre del campo	Tipo de Dato	Longitud	Modificador
Id_Aprendiz	Char	10	Primary Key, Not Null
Nom_Aprendiz	VarChar	50	Not Null
Ape_Aprendiz	VarChar	50	Not Null

```
Create table Aprendiz (
Id_Aprendiz Char (10) Primary Key not null,
Nom_Aprendiz Varchar  (50) not null,
Ape_Aprendiz Varchar (50) not null )
```

1.3.1.3 Modificación de las tablas

El comando utilizado es **ALTER TABLE**, permite realizar cambios a una tabla ya creada. Las posibles modificaciones a realizar son: adición de nuevas columnas, eliminación de columnas, cambio de tipo de dato, adición de constraints (restricciones) a las columnas previamente creadas o la eliminación de restricciones.

Sin embargo, para proceder a incorporar el cambio el SMBD realiza una serie de verificaciones para que estos cambios no afecten los datos previamente registrados (si ya existieran) o entren en contravención con reglas anteriores. Por ejemplo, para que una columna pueda modificarse y convertirse en llave primaria, debe tener la restricción de not null y los datos (si los hubiese) no podrían tener valores repetidos.

En esta sentencia se utilizan expresiones que determinan el tipo de cambio a incorporar, de la siguiente forma:

- **ADD (COLUMN o CONSTRAINT):** para agregar columnas o restricciones.
- **DROP (COLUMN o CONSTRAINT):** para eliminar columnas o restricciones.
- **ALTER / MODIFY:** para modificar columnas existentes.

Sintaxis:

```
ALTER TABLE <Nombre_de_la_tabla>
ADD (COLUMN) <Nombre_de_la_columna> <tipo de dato>
<Restricción>
ADD(CONSTRAINT)<Nombre_restricción> <tipo_restricción>
<texto_restricción>
DROP(COLUMN) <Nombre_ de_la_columna>
DROP (CONSTRAINT) <Nombre_de_la_restricción>
```

Ejemplo

Se requiere agregar una columna a la tabla Medico, para almacenar el registro del programa del aprendiz, el cual es un número de tipo entero y no puede repetirse.

```
Alter table Aprendiz
Add Column Reg_Programa
Int
Alter table Aprendiz
Add Constraint UNIQUE ( Reg_Programa)
```

1.3.1.4 Eliminación de tablas y base de datos

La sentencia **DROP TABLE** permite eliminar una tabla o una base de datos, siempre y cuando se tengan permisos sobre el objeto, no se encuentre abierta o siendo accesada por algún usuario o si al eliminarla se infringe alguna regla. El caso más común está relacionado con las llaves foráneas, cuando el contenido de una tabla es referenciada por otra a través de las llaves foráneas.

Ejemplo para eliminar una base de datos	Ejemplo para eliminar una tabla
DROP DATABASE misena;	DROP TABLE aprendiz;
Permite borrar, eliminar la base de datos misena.	Elimina la tabla aprendiz.

1.3.2 Lenguaje de manipulación de datos DML

Este lenguaje se utiliza para “manipular” los datos de una base de datos, es decir: insertar, borrar, modificar y consultar los registros de las tablas que conforman una base de datos.

Las sentencias DML son:

- Insert.
- Update.
- Delete.
- Select.

1.3.2.1 Inserción de datos

La sentencia **INSERT** se utiliza para agregar los registros a una tabla, es decir que se agregan filas completas de datos a la tabla, previa a la inserción se realiza un proceso de verificación de las restricciones presentes en cada campo, cuando el campo es una llave primaria, el valor a insertar no debe ser nulo o repetido y así sucesivamente con cada dato a insertar. La fila siempre es agregada al final de la tabla y el valor de cada campo debe coincidir con el tipo de dato establecido para cada columna.

Sintaxis	Ejemplo:
INSERT INTO <Nombre_de_la_tabla> (<Nombre_columna1>, <Nombre_columna2> <Nombre_columnaN>) VALUES (valor1, valor2, valorN)	En la tabla Aprendiz se van a insertar los siguientes datos Juan José Teatín Rincón, con identificación 1932208547 y registro del programa 754632. La sentencia de inserción sería: INSERT INTO Aprendiz(Id_Aprendiz, Nomb_Aprendiz, Ape_Aprendiz, Reg_Programa)VALUES (1932208547, ‘Juan José’,‘Teatín Rincón’, 754632)

Los datos de tipo alfanumérico, así como las fechas, generalmente se escriben entre comillas simples o dobles dependiendo del SGDB a usar.

1.3.2.2 Modificación de datos

La sentencia **UPDATE** se utiliza para realizar modificaciones sobre los datos que se encuentran en los campos de una tabla. El sistema realiza una validación de la integridad de los campos, verificando que los nuevos datos no infrinjan ninguna de las restricciones

asociadas a los campos. Se debe tener especial cuidado en proporcionar adecuadamente la condición que determina sobre cuál o cuáles de los registros deben aplicarse los cambios.

Sintaxis	Ejemplo:
UPDATE <Nombre_de_la_tabla> SET <Nombre_columna a cambiar valor> = <Nuevo_Valor> WHERE <condición>	Se desea modificar la dirección de residencia del aprendiz José Teatín, su nueva dirección es: “Calle 55 Nro. 20-15”. La sentencia de modificación de datos entonces sería: UPDATE Aprendiz SET Dir_Aprendiz = “Calle 55 Nro. 20-15”

De acuerdo con la anterior instrucción, si se dejara así, se modificarían todas las direcciones de los médicos, de otro modo al agregar la condición se especifican los registros sobre los cuales se debe hacer el cambio. Para que esto no suceda se requiere realizar la siguiente instrucción:

```
UPDATE Aprendiz
SET Dir_Aprendiz = “Calle 55 Nro. 20-15”
WHERE Nom_Aprendiz = “José” AND Ape_Aprendiz = “Teatín”
```

Más adelante en este material encontrará los aspectos más relevantes a considerar para construir los filtros o condiciones de las sentencias SQL.

1.3.2.3 Eliminación de registros

La sentencia **DELETE** se utiliza para borrar filas de datos de una tabla. El sistema realiza una validación de la integridad referencial antes de ejecutar la acción. Así como con la modificación se debe tener especial cuidado en proporcionar adecuadamente la condición que determine cual o cuales de los registros deben ser borrados.

Sintaxis	Ejemplo:
DELETE FROM <Nombre_de_la_tabla> WHERE <condición>	Se desea retirar de la base de datos al aprendiz José Teatín, con registro del programa 11854632. La sentencia de modificación de datos entonces sería: DELETE FROM Aprendiz WHERE nom_aprendiz = “José” AND ape_aprendiz = “Teatín” AND reg_programa = 11854632

Aunque no es necesario proporcionar todos los datos, se debe asegurar que la condición filtra únicamente el o los registros que se desean eliminar.

1.3.2.4. Consulta de datos

Con la sentencia **SELECT** se visualiza la información de la base de datos. Los datos que se presentan corresponden a una o más filas de una tabla o también a una o más filas de una o más tablas.

La sintaxis básica es:

Sintaxis	Ejemplo:
SELECT <Nombre_columna> o <lista de columnas> FROM<Nombre_de_la_tabla>	Se desea consultar los nombres y apellidos de todos los aprendices, la sentencia para hacer esta consulta sería: SELECT Nom_Aprendiz, Ape_Aprendiz FROM Aprendiz Esta instrucción, puede ir acompañada de las siguientes clausulas: WHERE <condición> GROUP BY <Nombre_columna1>, … HAVING <condición> ORDER BY <Nombre_columna> <Modo de ordenamiento>

Antes de realizar cualquier consulta a la base de datos, es muy importante tener claro cuál o cuáles son los datos que se requiere visualizar y de qué tabla o tablas se van a extraer.

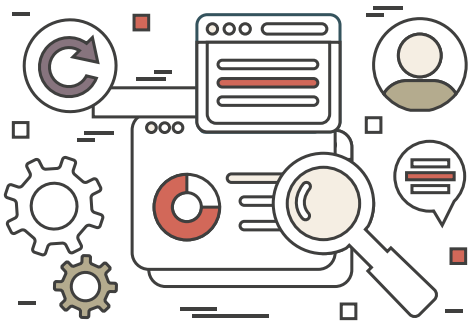
En caso de que se deseara consultar TODOS los campos de la tabla, no es necesario escribir todos los campos, a menos que se desee escribirlo en un orden particular, si no es así se utiliza el comodín * (asterisco) que representa todos los campos. En el ejemplo anterior se requería sólo nombre y apellido, en caso de desear visualizar todos los campos de una tabla, utilizando el comodín *, quedaría la sentencia de la siguiente manera:

```
Select * From Aprendiz;
```

Nota
Cuando existen muchos campos, la utilización de los comodines no se hace muy recomendable porque pueden presentar demoras en la ejecución de la consulta al tener que devolver todos los campos de la tabla.

1.3.3 Alias de columnas

Estos alias se utilizan para ayudar a la visualización del nombre de las columnas al momento de generar el resultado de la consulta, estos nombres son temporales, es decir que sólo se reflejan en salida, bien sea pantalla o impresión, pero no afectan los nombres que los campos tienen en la tabla. Para asignar el alias a un campo se utiliza la cláusula AS (como).



Una consulta, sin alias es:

```
Select Id_Aprendiz, Nom_Aprendiz, Ape_Aprendiz
From Aprendiz
```

Ahora se configura la salida por pantalla a los datos incluidos en la tabla aprendiz, la salida debe ser, la columna Id_Aprendiz con título Identificacion; la columna Nom_Aprendiz, con título Nombre y la columna Ape_Aprendiz con título Apellidos, la sintaxis sería la siguiente:

```
Select Id_Aprendiz as Identificacion,
Nom_Aprendiz as Nombre ,
Ape_Aprendiz as Apellidos
From Aprendiz
```

1.3.4 La cláusula WHERE

Esta cláusula es muy importante porque se utiliza para generar resultados basados en condiciones, ya que filtra verticalmente las filas a presentar.

La sintaxis es:

```
SELECT <Nombre_columna> o <lista de columnas>
FROM <Nombre_de_la_tabla>
WHERE <condición>
```

Cuando se van a comparar dos valores se realizan según las siguientes reglas:

- Solo se pueden comparar dos valores cuando son del mismo tipo.
- Todos los datos de tipo numérico pueden ser comparados unos con otros (por ejemplo un decimal con un entero).
- Se pueden comparar dos valores alfanuméricos, tomando como referencia el código de cada uno de los caracteres que conforman la cadena.

La selección de filas se especifica en la cláusula **WHERE** mediante predicados. Un predicado expresa una condición y su resultado puede ser verdadero, falso o desconocido.

1.3.5 Las expresiones

Una expresión es una combinación de símbolos y operadores que el motor de base de datos de SQL Server evalúa para obtener un único valor. Una expresión simple puede ser una sola constante, variable, columna o función escalar. Los operadores se pueden usar para combinar dos o más expresiones simples y formar una expresión compleja.

Dos expresiones pueden combinarse mediante un operador si ambas tienen tipos de datos admitidos por el operador y se cumple al menos una de estas condiciones:

1.3.5.1 Tipos de operadores

Operadores numéricos:

- suma +
- resta -
- multiplicación *

- división /
- módulo
- (resto de una división) %

Operadores bit a bit: realizan manipulaciones de bits entre dos expresiones de cualquiera de los tipos de datos de la categoría del tipo de datos entero.

- AND &
- OR |
- OR exclusivo ^

Operadores de comparación:

- Igual a =
- Mayor que >
- Menor que <
- Mayor o igual que >=
- Menor o igual que <=
- Distinto de <>
- No es igual a !=
- No menor que !<
- No mayor que !>

Operadores lógicos:

- IN
- LIKE
- BETWEEN
- EXISTS

Operadores de cadenas:

- Concatenación +

Resultados de la expresión: si se combinan dos expresiones mediante operadores de comparación o lógicos, el tipo de datos resultante es booleano y el valor es uno de los siguientes: **TRUE, FALSE o UNKNOWN.**

1.3.5.2 Otros elementos del lenguaje

Comentarios
En este lenguaje también se pueden utilizar comentarios destinados a facilitar la legibilidad del código. En SQL se insertan comentarios con los signos:

/* */ Varias líneas
/* Esto es un comentario en varias líneas
*/-- Una única línea
-- Esto es un comentario en una única línea.

USE. Cambia el contexto de la base de datos al de la base de datos especificada.

USE BDSena
Hace que la base de datos activa pase a ser la base de datos indicada en la instrucción, las consultas que se ejecuten a continuación se harán sobre tablas de esa base de datos si no se indica lo contrario. Es una instrucción útil para asegurarnos de que la consulta se ejecuta sobre la base de datos correcta.

1.3.6 Predicados

En SQL existen unos tipos de predicados, condiciones básicas de búsqueda:

Comparación estándar

- Pertenencia a un intervalo (BETWEEN).
- Coincidencia con patrón (LIKE).
- Pertenencia a un conjunto (IN).
- Test de valor nulo (IS NULL).
- Si contiene (CONTAINS).
- FREETEXT.

Nota
Dentro de este material de aprendizaje se detallarán más adelante los predicados más recurrentes dentro de una consulta de la base de datos.

Comparación estándar
Compara el valor de una expresión con el valor de otra. Para la comparación se pueden emplear = , <> , !=, < , <= , !<, > , >= ,!>

Sintaxis:

<expresion> {=|<>|!=|>|>=|!>|<|<=|!<} <expresion>

<expresion> puede ser:

- Un nombre de columna.
- Una constante.
- Una función (inclusive la función CASE).
- Una variable.
- Una subconsulta escalar.
- Cualquier combinación de nombres de columna, constantes y funciones conectados mediante uno o varios operadores o una subconsulta.

Ejemplo 1

Listar los “buenos” vendedores (los que han rebasado su cuota).

```
SELECT item_emp, nom_emp,  ventas, cuota
FROM empleados
WHERE ventas > cuota
```

item_emp	Nombre	Ventas	Cuota
1	Rosario Badillo	1500000	30000
2	Agustín Rincón	2000000	35000
3	Fredy González	3500000	27500
4	Juan Teatín	1800000	35000
5	José Rincón	5000000	27500
6	Jorge Pérez	2500000	35000
7	Rita Rincón	4000000	3000

Las columnas que aparecen en el **WHERE** no tienen por qué aparecer en la lista de selección, esta instrucción es igual de válida:

Ejemplo 2

```
SELECT item_emp, nom_emp
FROM empleados
WHERE ventas > cuota;
```

Ejemplo 3
Hallar vendedores contratados antes del año 2000.

```
SELECT item_emp, nom_emp, contrato
FROM empleados
WHERE contrato < '01/01/2000'>;
```

item_emp	nombre	contrato
1	Rosario Badillo	1/06/1990
2	Agustín Rincón	1/08/1999
3	Fredy González	1/11/1997

También se puede utilizar funciones, ésta es equivalente a la anterior:

Ejemplo 4

```
SELECT item_emp, nom_emp
FROM empleados
WHERE YEAR(contrato) < 2000;
La función YEAR(fecha) devuelve el año de una fecha.
```

Ejemplo 5
Hallar oficinas cuyas ventas estén por debajo del 80% de la meta.

```
SELECT  oficina
FROM  oficinas
WHERE  ventas < (.8 * meta);
```

Ejemplo 6
Hallar las oficinas dirigidas por el empleado 10.

```
SELECT  oficina
FROM  oficinas
WHERE  dir = 10;
```

1.3.6.1 BETWEEN

Sintaxis:

```
<expresion> [NOT] BETWEEN <expresion2> AND <expresion3>
```

Ejemplo 1
Hallar vendedores cuyas ventas estén entre $2.000.000 y $3.000.000 de pesos.

```
SELECT item_emp, nom_emp, ventas
FROM empleados
WHERE  ventas BETWEEN 2000000 AND  3000000;
```

item_emp	nom_emp	ventas
1	Rosario Badillo	2500000
2	Agustín Rincón	2200000
3	Fredy González	2800000
4	Juan Teatín	2200000
5	José Rincón	2700000
6	Jorge Pérez	2500000
7	Rita Rincón	2500000

Ejemplo 2
La instrucción anterior es equivalente a:

```
SELECT item_emp, nom_emp, ventas
FROM empleados
WHERE  ventas >= 2000000 AND ventas <=3000000;
```

1.3.6.2 Like

Comparación de caracteres: se utiliza el predicado LIKE. Cuando se requiere precisar buscar campos que contengan combinaciones de caracteres que cumplan ciertas condiciones. Para ello se utiliza el predicado LIKE, según el siguiente formato:

```
<Nombre_columna> [NOT] LIKE <constante_alfanumerica>
```

Donde <Nombre_columna> debe ser de tipo alfanumérico.

Ejemplo 1
Las ciudades que empiecen por S.

```
SELECT … FROM…
WHERE ciudad LIKE 'S%'
```

Ejemplo 2
Las referencias que no tengan un 3.

```
SELECT … FROM…
WHERE referencia LIKE '%[^3]%'
```

Ejemplo 3
Las referencias que no tengan un 3, 4 o 5.

```
SELECT … FROM…
WHERE ciudad LIKE '%[3-5]%'
```

1.3.6.3 Pertenencia a un conjunto (IN)

IN sirve para examinar si el valor de la expresión es uno de los valores incluidos en la lista de valores indicados entre paréntesis. La única condición es que todas las expresiones devuelvan el mismo tipo de datos.

```
<expresion> IN (  <exp_valor> [ ,...n ] )
```

Ejemplo
Listar los empleados que trabajan en las oficinas 110, 111 o 114.

```
SELECT ítem_emp, nom_emp, oficina
FROM empleados
WHERE oficina IN (110,111,114);
```

Resultado:

item_emp	nom_emp	oficina
1	Rita Rincón	110
2	Agustín Rincón	111
3	Fredy González	110
4	Juan Teatín	111
5	José Rincón	114

1.3.7 La unión de tablas UNION

La unión de tablas consiste en coger dos tablas y obtener una tabla con las filas de las dos tablas, en el resultado aparecerán las filas de una tabla y, a continuación, las filas de la otra tabla. Para poder realizar la operación, las dos tablas deben tener el mismo esquema (mismo número de columnas y tipos compatibles) y la tabla resultante hereda los encabezados de la primera tabla.

La sintaxis es la siguiente:

```
{< consulta >|(<  consulta >)}

 UNION [ALL]
{<  consulta >|(< consulta >)}

[{UNION  [ALL] {< consulta >|(< consulta >)}}[ ...n ] ]
```

Se pueden combinar varias tablas con el operador **UNION**. Por ejemplo si se tiene otra tabla Casanare con las oficinas nuevas de Casanare:

Ejemplo 1

```
SELECT oficina, ciudad FROM Bogotá
UNION
SELECT oficina, ciudad FROM Santander
UNION
SELECT oficina, ciudad FROM Casanare;
```

En la anterior instrucción se combinan las tres tablas.

Ejemplo 2
Obtener todos los productos cuyo precio exceda de $20.000 pesos o que se haya vendido más de $30.000 pesos del producto en algún pedido.

```
SELECT id_fab, id_producto
FROM  productos
WHERE precio > 20000
UNION
SELECT fab, producto
FROM pedidos
WHERE venta > 30000;
```

1.3.8 Cláusula Order By

Esta cláusula sirve para mostrar las consultas ordenadas por distintos criterios.

Sintaxis:

[ORDER BY {expression_columna|posicion_columna [ASC|DESC]}
[,...n]]
< consulta > representa la especificación de la consulta que devolverá la tabla a combinar.

Ejemplo 1
Se tiene la tabla Bogotá con las nuevas oficinas de Bogotá y otra tabla Santander con las nuevas oficinas de Santander y se desea obtener una tabla con las nuevas oficinas de las dos ciudades pero no de manera ordenada por oficina:

```
SELECT oficina as OFI, ciudad FROM Bogotá
UNION ALL
SELECT oficina,  ciudad FROM Santander;
```

El resultado sería:

OFI	ciudad
11	Bogotá
28	Bogotá
23	Santander

El resultado toma los nombres de columna de la primera consulta y aparecen primero las filas de la primera consulta y después las de la segunda. Si se desea que el resultado aparezca ordenado se puede incluir la cláusula **ORDER BY**, pero después de la última especificación de consulta, y expresion_columna será cualquier columna válida de la primera consulta.

Ejemplo 2

```
SELECT oficina as OFI
, ciudad FROM Bogotá
UNION
SELECT oficina, ciudad FROM Santander
ORDER BY ofi;
```

OFI	ciudad
11	Bogotá
23	Santander
28	Bogotá

Ahora las filas aparecen ordenadas por el número de oficina y se ha utilizado el nombre de columna de la primera consulta. Cuando aparezcan en el resultado varias filas iguales, el sistema por defecto elimina las repeticiones. Si se especifica **ALL**, el sistema devuelve todas las filas resultantes de la unión, incluidas las repetidas. El empleo de **ALL** también hace que la consulta se ejecute más rápidamente ya que el sistema no tiene que eliminar las repeticiones.

2. Consultar datos, provenientes de diferentes tablas

Este tipo de consultas se utiliza frecuentemente y sirve para visualizar datos que están en diferentes tablas, específicamente los datos de aquellas que tienen llaves foráneas. En este tipo de consultas se vuelven a trabajar los alias, solo que en esta ocasión el alias es para las tablas. El uso de alias en los nombres de tablas mejora la legibilidad de las secuencias de comandos, facilita la escritura de combinaciones complejas y simplifica el mantenimiento de las consultas.

Al escribir secuencias de comandos, puede sustituir un nombre de tabla descriptivo largo y complejo por un alias sencillo y abreviado. El alias se utiliza en lugar del nombre completo de la tabla.

Sintaxis:

```
SELECT *
FROM <Nombre_tabla1>AS <alias Tabla1>
INNER JOIN
<Nombre_tabla2>AS <alias Tabla2>
ON <aliasTabla1>.<Campo_llaveprimaria>=
<aliasTabla2>.<Campo_llaveforánea>
```

Ejemplo 1

En este ejemplo se muestran los nombres de los pacientes, la identificación, sexo y el nombre del médico que lo ha atendido, información que se encuentra en las tablas pacientes y médicos.

Esta consulta no utiliza alias en las tablas de la sintaxis de JOIN.

```
SELECT PacNombres, PacApellidos, PacSexo, MedNombres,
MedApellidos
FROM Tblpacientes INNER JOIN TblMedicos
ON TblMedicos.MedIdentificacion = Tblpacientes.
PacMedIdentifica
```

Ejemplo 2
Utilizando alias para las tablas.

```
SELECT PacNombres, PacApellidos, PacSexo, MedNombres,
MedApellidos
FROM Tblpacientes as TP INNER JOIN TblMedicos as TM
ON TM.MedIdentificacion = TP. PacMedIdentifica
```

2.1 Producto cartesiano

Consiste en una nueva tabla formada por las filas que resulten de todas las combinaciones posibles de las filas de la primera tabla con todas las filas de la segunda tabla. El número de filas resultante es el producto de la multiplicación de todas las filas de la primera tabla por la segunda. Por esta razón, es imprescindible adicionar un filtro que corresponda con el vínculo que existe entre las tablas, de otra forma los resultados no serían coherentes con la información.
Cuando se realiza una consulta, se especifican en la cláusula **SELECT** las columnas de cada tabla que se desean visualizar, en la cláusula **FROM** los nombres de las tablas que contienen la información separadas por coma.

Por último en la cláusula **WHERE** una condición en la cual intervienen al menos una columna de cada tabla que representan la conexión entre las dos tablas.

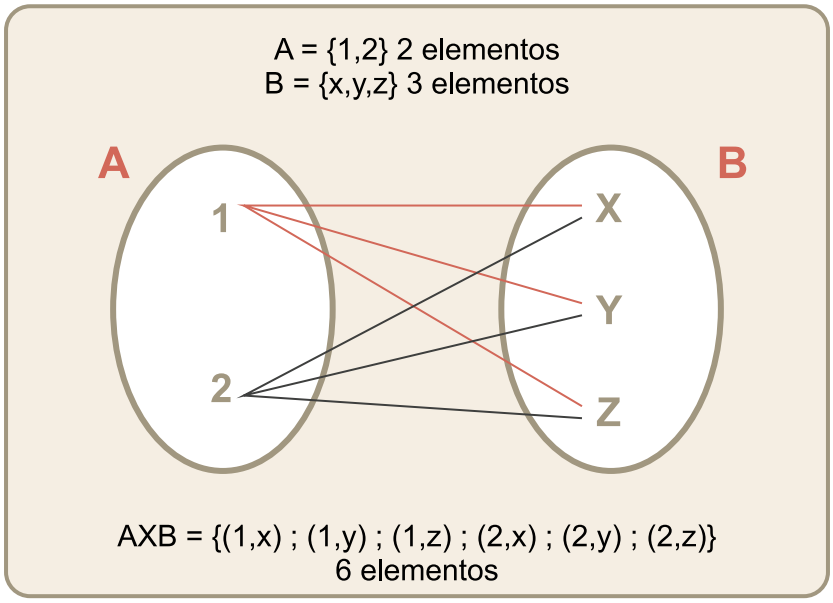
Sintaxis:

```
SELECT *
FROM <Nombre_tabla1> , <Nombre_tabla2>
WHERE <Tabla1>.<Campo> = <Tabla2>.<Campo>
```

Por ejemplo, se desea una lista con los pacientes que trata el medico “Juan Teatín”, si la tabla pacientes cuenta con 3000 pacientes y la tabla de Médicos con 200, la combinación inicial da como resultado 600.000 registros (3000 X 200 = 600000), los cuales deben ser cargados en memoria y posteriormente se aplicarían los filtros de conexión (médicos y

pacientes vinculados) y por último mostraría sólo los que corresponden a “Juan Teatín”, lo cual hace que el proceso consuma más recursos de los necesarios.

La siguiente imagen presenta la relación del producto cartesiano y su perspectiva desde la teoría de conjuntos y el producto cartesiano aplicado a las bases de datos relacionales:



2.1.1 El producto cartesiano CROSS JOIN

En el producto cartesiano se consigue obtener todas las posibles concatenaciones de filas de la primera tabla con filas de la segunda tabla. Se indica escribiendo en la cláusula **FROM** los nombres de las tablas separados por una coma o utilizando el operador **CROSS JOIN.**

```
FROM {<tabla_origen>} [ ,...n ]
  |<tabla_origen> CROSS JOIN <tabla_origen>
```

Tabla_origen puede ser un nombre de una tabla derivada o tabla o de vista o (resultado de una SELECT), en este último caso SELECT tiene que aparecer entre paréntesis y la tabla derivada debe llevar asociado obligatoriamente un alias de tabla. También puede ser una composición de tablas.

Se pueden utilizar hasta 256 orígenes de tabla en una instrucción, aunque el límite varía en función de la memoria disponible y de la complejidad del resto de las expresiones de la consulta. También se puede especificar una variable table como un origen de tabla.

Ejemplo 1

```
SELECT *
FROM aprendiz, centro;
```

Si se ejecuta esta consulta se observará que las filas del resultado están formadas por las columnas de aprendiz y las columnas de centros. En las filas aparece cada aprendiz combinado con el primer centro, luego los mismos aprendices combinados con el segundo centro y así hasta combinar todos los aprendices con todos los centros.

Si se utiliza el **ejemplo 2**

```
SELECT *
FROM aprendiz CROSS JOIN centro;
```

Esta instrucción obtendría lo mismo. Lo más usual sería combinar cada aprendiz con los datos de su centro. Se podría obtener añadiendo a la consulta un **WHERE** para filtrar los registros correctos:

Ejemplo 3

```
SELECT *
FROM aprendiz, centro
WHERE aprendiz.centro=centro.aprendiz;
```

Se necesita cualificar los campos ya que el nombre centro es un campo de aprendiz y de centro por lo que si no lo cualificamos, el sistema arroja error.

Se utiliza la lista de selección *, esto recupera todas las columnas de las dos tablas.

Ejemplo 4

```
SELECT aprendiz.*,ciudad, region
FROM aprendiz, centro
WHERE aprendiz.centro=centro.centro;
```

Recupera todas las columnas de aprendices y las columnas ciudad y región de centros. También se puede combinar una tabla consigo misma, pero en este caso hay que definir un alias de tabla, en al menos una, sino el sistema da error ya que no puede nombrar los campos.

```
SELECT *
FROM centro, centro as cen20;
```

El producto cartesiano no sería el más indicado porque no es la operación más utilizada, ya que normalmente cuando se quiera componer dos tablas se hará con una condición de selección basada en campos de combinación y para este caso es más eficiente el JOIN que se muestra a continuación.

2.1.2 La composición interna INNER JOIN

Una composición interna es aquella en la que los valores de las columnas que se están combinando se comparan mediante un operador de comparación. Es la operación que más se emplea porque lo más usual es juntar los registros de una tabla relacionada con los registros correspondientes en la tabla de referencia (añadir a cada registro los datos de un aprendiz, añadir a cada pedido los datos de su producto, entre otros).

```
FROM
<tabla_origen> INNER JOIN <tabla_origen> ON <condicion_combi>
tabla_origen tiene el mismo significado que en el producto cartesiano.
```

Ejemplo 1

```
SELECT *
FROM empleados INNER JOIN oficinas
ON  empleados.oficina=oficinas.oficina;
```

- Obtiene los empleados combinados con los datos de su oficina.

Ejemplo 2

```
SELECT *
FROM pedido INNER JOIN producto
ON producto = idproducto AND fab =  idfab;
```

- Obtiene los pedidos combinados con los productos correspondientes.

Normalmente la condición de combinación será una igualdad pero se puede utilizar cualquier operador de comparación (<>, >…). Es fácil ver la utilidad de esta instrucción y de hecho se utilizará muy a menudo, pero hay algún caso que no resuelve. En las consultas anteriores, no aparecen las filas que no tienen fila correspondiente en la otra tabla.

Ejemplo 3

```
SELECT item_emp, nom_empleado, empleados.oficina,  ciudad
FROM empleados INNER JOIN oficinas
ON  empleados.oficina=oficinas.oficina;
```

item_emp	nom_emp	oficina	ciudad
1	Rosario Badillo	10	San Gil
2	Agustín Rincón	20	Girón
3	Fredy González	30	Bogotá
4	Juan Teatín	40	Manizales
5	José Rincón	50	Huila
6	Jorge Pérez	60	Cartagena
7	Rita Rincón	70	Bucaramanga

En este ejercicio no aparecen los empleados que no tienen oficina, ni las oficinas que no tienen empleados, porque para que salga la fila, debe de existir una fila de la otra tabla que cumpla la condición.

2.1.3 Combinaciones Externas La Composición externa LEFT, RIGHT y FULL OUTER JOIN

En este tipo de combinaciones existe una tabla de tipo dominante en la consulta y las otras serán subordinadas, los registros de la tabla dominante serán incorporados al resultado final, más los registros provenientes de la intersección de las tablas.

Las combinaciones externas izquierda y derecha combinan filas de dos tablas que cumplen una condición, más las filas de la tabla izquierda o derecha que no la cumplen, tal como se especifique en la cláusula JOIN. Las filas que no cumplen la condición de combinación presentan NULL en el conjunto de resultados. También se pueden utilizar combinaciones externas completas para presentar todas las filas de las tablas combinadas, independientemente de si hay valores que coincidan en las tablas.

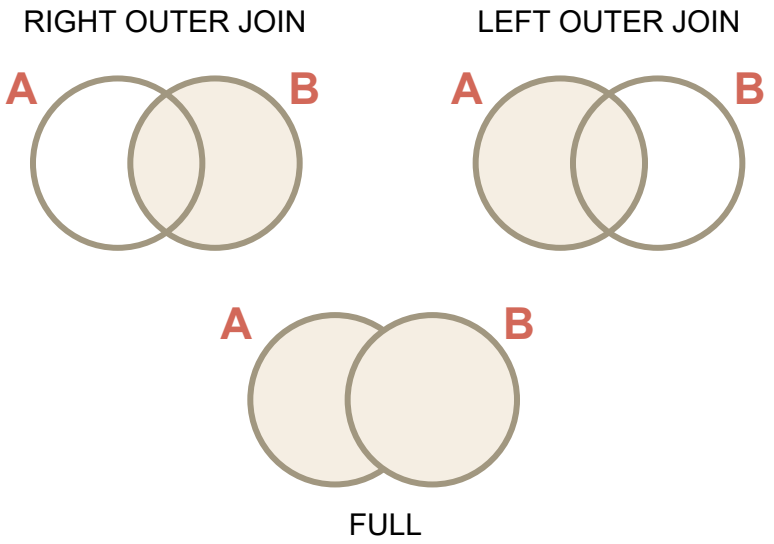
La composición externa se escribe de manera similar al INNER JOIN indicando una condición de combinación pero en el resultado se añaden filas que no cumplen la condición de combinación.

Sintaxis:

```
FROM
<tabla_origen> {LEFT|RIGHT|FULL} [OUTER] JOIN <tabla_origen>
ON <condicion_combi>
```

La palabra OUTER es opcional y no añade ninguna función.
Las palabras LEFT, RIGHT y FULL indican la tabla de la cual se van a añadir las filas sin correspondencia.

La siguiente imagen representa las combinaciones externas:



Ejemplo 1

```
SELECT item_emp, nom_emp, empleados.oficina,  ciudad
FROM empleados LEFT JOIN oficinas
ON  empleados.oficina=oficinas.oficina;
```

item_emp	nom_emp	oficina	ciudad
1	Rosario Badillo	10	San Gil
2	Agustín Rincón	20	Girón
3	Fredy González	30	Bogotá
4	Juan Teatín	40	Manizales
5	José Rincón	50	Huila

6	Jorge Pérez	60	Cartagena
7	Rita Rincón	70	Bucaramanga
8	Guillermo Suarez	NULL	NULL

Ahora sí aparece el empleado 8 que no tiene oficina.

Se obtiene los empleados con su oficina y los empleados (tabla a la izquierda LEFT del JOIN) que no tienen oficina aparecerán también en el resultado con los campos de la tabla oficinas rellenados a NULL.

Ejemplo 2

```
SELECT ítem_emp, nom,emp.oficina,  ciudad, oficinas.oficina
FROM empleados RIGHT JOIN oficinas
ON  empleados.oficina=oficinas.oficina;
```

item_emp	nom_emp	oficina	ciudad	oficina
1	Rosario Badillo	10	San Gil	10
2	Agustín Rincón	20	Girón	20
3	Fredy González	30	Bogotá	30
4	Juan Teatín	40	Manizales	40
5	José Rincón	50	Huila	50
6	Jorge Pérez	60	Cartagena	60
7	Rita Rincón	70	Bucaramanga	70
NULL	NULL	NULL	Bucaramanga	80
NULL	NULL	NULL	Bucaramanga	85
NULL	NULL	NULL	Huila	90

Las oficinas 80,85 y 90 no tienen empleados.
Obtiene los empleados con su oficina y las oficinas (tabla a la derecha RIGHT del JOIN) que no tienen empleados aparecerán también en el resultado con los campos de la tabla empleados rellenados a NULL.

Ejemplo 3

```
SELECT numemp,nombre,empleados.oficina,  ciudad, oficinas.oficina
FROM empleados FULL JOIN oficinas
ON  empleados.oficina=oficinas.oficina;
```

item_emp	nom_emp	oficina	ciudad	oficina
1	Rosario Badillo	10	San Gil	10
2	Agustín Rincón	20	Girón	20
3	Fredy González	30	Bogotá	30
4	Juan Teatín	40	Manizales	40
5	José Rincón	50	Huila	50
6	Jorge Pérez	60	Cartagena	60
7	Rita Rincón	70	Bucaramanga	70
8	Ruby Badilllo	NULL	NULL	NULL
NULL	NULL	NULL	Bucaramanga	80
NULL	NULL	NULL	Bucaramanga	85
NULL	NULL	NULL	Huila	90

Con la anterior instrucción aparecen tanto los empleados sin oficina como las oficinas sin empleados.

2.1.4 Autocombinaciones – SELF JOIN

Es posible que se necesite combinar una tabla consigo misma, esto es lo que se denomina un self-join. Puede ser útil cuando se dese comparar valores de una columna con otra en la misma tabla. Para utilizarlo sólo es necesario colocar alias a la tabla como si se tratase de dos tablas diferentes y utilizar la estructura del JOIN que se requiera.

```
SELECT *
FROM <Nombre_tabla1>AS <alias 1> INNER JOIN
<Nombre_tabla1>AS <alias 2>
ON <alias1>.<Campo>= <alias2>.<Campo>
```

En el ejemplo siguiente se utiliza una autocombinación interna para obtener los nombres de los jefes de los empleados que corresponden a otros empleados en la misma tabla.

Ejemplo 1. Empleados A.

id_jefe	id_emp	nom_emp
15	1	Pedro Badillo
15	2	Juliana Rincón
20	3	Carlos González
20	4	Gustavo Pérez

Empleados B.

id_jefe	id_emp	nom_emp
15	1	Pedro Badillo
15	2	Juliana Rincón
20	3	Carlos González
20	4	Gustavo Pérez
20	5	José Rincón

```
SELECT A.ID_Empl, A.Nombre, B.Nombre as “Jefe”
FROM Empleados as A INNER JOIN Empleados as B
ON A.ID_Jefe=B.ID_Empl
WHERE Id_Jefe IS NOT NULL
```

id_emp	nom_emp	nom_jefe
1	Pedro Badillo	Tatiana Rojas
2	Juliana Rincón	Tatiana Rojas
3	Carlos González	Jesús Castillo
4	Gustavo Pérez	Jesús Castillo
5	José Rincón	Jesús Castillo

Cuando se usan autocombinaciones, cada fila que coincida consigo misma y las parejas que coincidan entre ellas se repiten, por lo que se devuelven filas duplicadas, por tanto se debe utilizar una cláusula WHERE para eliminar dichas filas duplicadas.

2.2. Funciones de agregado

En la lista de selección de una consulta, se incluyen funciones que implementan diferentes fórmulas a aplicar sobre el conjunto de datos de una columna. Las funciones disponibles pueden variar dependiendo del SGBD que se utilice, a continuación se presentan las más comunes:

Expresión	Descripción	Ejemplo
SUM	Calcula la suma del conjunto de datos determinado por la expresión, estos deben ser de tipo numérico y el resultado será del mismo tipo.	SELECT SUM (precio) FROM Ventas/* Calcula el total de las ventas */

MIN	Obtiene el menor de los valores correspondientes con el conjunto de datos determinado por la expresión.	SELECT MIN (precio) FROM Productos/* Obtiene el menor precio de los productos */
MAX	Obtiene el mayor de los valores correspondientes con el conjunto de datos determinado por la expresión.	SELECT MAX (precio) FROM Productos/* Obtiene el precio más alto de los productos*/
AVG	Calcula el valor promedio (media aritmética) del conjunto de datos determinado por la expresión, estos deben ser de tipo numérico y el resultado puede variar.	SELECT AVG (Precio) FROM Ventas /* Calcula el precio promedio de venta de los productos */
COUNT (campo)	Calcula el número de filas (registros) que contiene información en el campo especificado	SELECT COUNT (email) FROM Clientes /* Obtiene el número de clientes que registraron un email */
COUNT(*)	Cuenta el número de registros (filas) resultantes en una consulta.	SELECT COUNT (*) FROM Ventas /* Obtiene el número de ventas que se han hecho */

2.3. La cláusula GROUP BY

Las funciones de agregado hasta ahora sólo se han aplicado al conjunto total de registros de la tabla, produciendo un único resultado, sin embargo en ocasiones es necesario que las funciones se apliquen sobre grupos particulares, produciendo “subtotales”. La cláusula **GROUP BY** se utiliza para identificar los grupos sobre los cuales deben aplicarse las funciones y produce una fila de resultado por cada grupo.

Sintaxis:

```
SELECT <Nombre_campos_grupo> , <constantes> ,
<función_de_agregado>
FROM <Nombre_tabla>
WHERE <Condición>
GROUP BY <Nombre_campos_grupo>
```

Ejemplo:

```
Select * From DETALLEPEDIDO
```

Select * From DETALLEPEDIDO

NUMPEDIDO	ITEM	CODPROD	CANTIDAD
12346	1	328	5
12347	2	328	4
12345	1	329	12
12346	3	329	10
12348	1	329	7
12350	1	329	1
12347	1	330	3
12350	2	330	1
12349	1	331	12
12345	2	332	1
12346	2	332	0

SELECT CODPROD, SUM (CANTIDAD)
FROM DETALLEPEDIDO
GROUP BY CODPROD

CODPROD	SUMA CANTIDAD
328	9
329	30
330	4
331	12
332	3

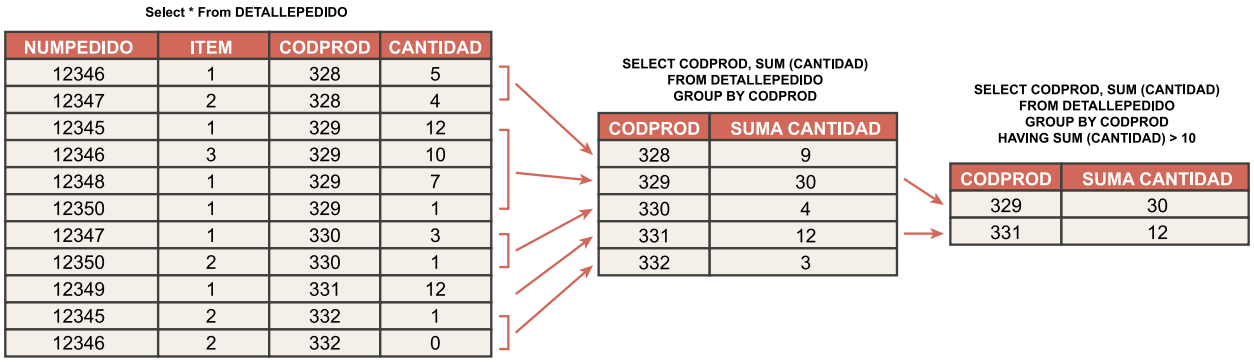
En la consulta de la derecha se generaron grupos por el código del producto obteniendo la cantidad total vendida de cada producto, para esto se aplicó la función SUM y se estableció en la cláusula GROUP BY el campo codprod.

2.4 La cláusula HAVING

En las consultas de resumen se pueden aplicar filtros a través de la cláusula **WHERE**, estos se aplicarían antes de generar los grupos, sin embargo es posible que se requiera aplicar filtros a los resultados después de establecer los grupos y aplicar las funciones. Esto se realiza gracias a la cláusula **HAVING** que actúa directamente sobre los resultados de las funciones de agregado.

Sintaxis:

```
SELECT <Nombre_campos_grupo>, <constantes>,
<función_de_agregado>
FROM <Nombre_tabla>
WHERE <Condición>
GROUP BY <Nombre_campos_grupo>
HAVING <Condición sobre columnas calculadas>
```



3. Subconsultas

Una subconsulta es una sentencia **SELECT** que aparece dentro de otra sentencia **SELECT** y se puede encontrar en la lista de selección, como parte del origen de datos, o como parte del criterio de selección (condición) bien sea a través de la cláusula **WHERE** o en la cláusula **HAVING** de la consulta principal.

La sintaxis de la subconsulta es la misma frente a las consultas vistas, sin embargo se presenta encerrada entre paréntesis, aun así existen algunas restricciones a considerar para su uso:

- No pueden incluir cláusulas de ordenamiento.
- Dependiendo del número de columnas que presente, existen restricciones referentes al lugar donde puede implementarse.
- De preferencia no utilizar campos calculados ya que le bajan el rendimiento a la consulta.

Existen tres tipos de subconsultas:

- Las que devuelven un solo valor, aparecen en la lista de selección de la consulta externa o con un operador de comparación sin modificar.
- Las que generan una columna de valores, aparecen con el operador IN o con un operador de comparación modificado con ANY, SOME o ALL.
- Las que pueden generar cualquier número de columnas y filas, son utilizadas en pruebas de existencia especificadas con EXISTS.

Ejemplo 1

En el ejemplo, se coge el primer empleado (item_emp= 10, por ejemplo) y se calcula la subconsulta sustituyendo item_emp por el valor 10, se calcula la suma de los pedidos del rep = 10, y el resultado se compara con la cuota de ese empleado, y así se repite el proceso con todas las filas de empleados.

El nombre de una columna dentro de la subconsulta se presupone del origen de datos de la subconsulta y, sólo si no se encuentra en ese origen, la considera como columna externa y la busca en el origen de la consulta externa.

Ejemplo 1

```
SELECT oficina, ciudad
FROM oficinas
WHERE meta > (SELECT SUM(ventas)
FROM empleados
WHERE oficina = oficina);
```

La columna oficina se encuentra en los dos orígenes (oficinas y empleados) pero esta consulta no dará error (no se pide cualificar los nombres como pasaría en una composición de tablas), dentro de la subconsulta se considera oficina el campo de la tabla empleados. Con lo que compararía la oficina del empleado con la misma oficina del empleado y eso no es lo que quiere, se desea comparar la oficina del empleado con la oficina de oficinas, se escribirá pues para forzar a que busque la columna en la tabla oficinas.

Ejemplo 2

```
SELECT oficina, ciudad
FROM oficinas
WHERE meta > (SELECT SUM(ventas)
FROM empleados
WHERE oficina = oficinas.oficina);
```

Glosario

ALIAS: nombre con el que se puede remplazar un campo o una tabla de la base de datos, una vez se le ha asignado el alias, en adelante se utilizará como referencia al elemento.

AUTO_INCREMENT: tipo de modificador que se utiliza para ir incrementando automáticamente el valor en cada registro.

DDL: lenguaje de definición de datos.

ENUM: campos cuyo contenido formen parte de una serie de opciones.

MySQL: motor de bases de datos libre, gratuito que está disponible para varios sistemas operativos.

Nemotécnicos: secuencia de datos, nombres, números, y en general para recordar listas de ítems.

NOT NULL: tipo de identificador utilizado para impedir que un campo esté vacío.

PRIMARY KEY: es un campo o a una combinación de campos que identifica de forma única a cada fila de una tabla.

SQL: Structured Query Language: lenguaje de consulta estructurado.

UNIQUE: tipo de identificador que evita la repetición de valores.

USE: palabra utilizada para establecer una base de datos, como base de datos predefinida.

SGBD: sistema de gestión de base de datos es un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos.

Bibliografía

Beaulieu, A. (2010). *Aprende SQL. Segunda Edición.* Grupo Anaya Comercial.

James, R., Weinberg, P. (1991). *Applique SQL.* Osborne. McGraw-Hill.

Ojeda, F. (2005). *SQL*. Anaya Multimedia.

Control del documento

CONSTRUCCIÓN OBJETO DE APRENDIZAJE

SENA

LENGUAJE ESTRUCTURADO DE CONSULTAS	
Centro Industrial de Mantenimiento Integral - CIMI Regional Santander	
Líder línea de producción:	Santiago Lozada Garcés
Asesores pedagógicos:	Rosa Elvia Quintero Guasca Claudia Milena Hernández Naranjo
Líder expertos temáticos:	Rita Rubiela Rincón Badillo
Expertos temáticos:	Magda Milena García G. (V1) Ana Yaqueline Chavarro (V1) Rita Rubiela Rincón B. (V2)
Diseño multimedia:	Eulises Orduz Amezquita
Programador:	Francisco José Lizcano Reyes
Producción de audio:	Víctor Hugo Tabares Carreño