<codoacodo/>

FULL STACK Node JS
Clase 24

SQL 3

BA

JOIN y Subconsultas

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 23

Lenguaje y Sublenguajes SQL

- Gestión y manipulación de datos con SQL.
- Gestión y manipulación de datos.
- Sublenguajes DDL y DML.
- Consultas: Estructura consulta SQL. Cláusulas SELECT, FROM, WHERE.
- Alias y literales. ORDER BY.

Clase 24

JOIN y Subconsultas

- JOIN: Inner, Left, Right.
- Funciones de agregación, GROUP BY, HAVING.
- Funciones escalares: Caracteres o cadena, Conversión, Fecha y tiempo, Matemáticas.
- Subconsultas.

Clase 25

Fundamentos de Node

- Introducción a Node. Entorno.

Cláusula JOIN

La cláusula **JOIN** se utiliza para indicar la manera en que se relacionan las tablas, es decir, con qué atributos se está plasmando la relación entre ellas.
Para unir las tablas vamos a necesitar un dato que relacione a ambas tablas, que las una.

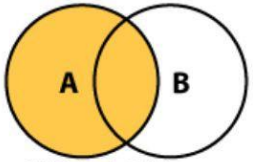
Cláusula JOIN

Las relaciones son importantes para no duplicar datos. *Por ejemplo: si tenemos una tabla "escuelas" y existe un alumno que pertenece a una escuela no debemos repetir los datos de la escuela en la tabla alumnos.*

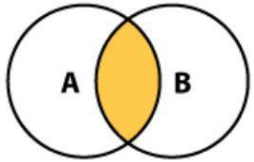
Las tablas deben estar normalizadas, los datos deben estar almacenados en forma eficiente para poder hacer consultas más rápidas, para que la BD no ocupe más espacio del que podría y para que las tablas no tengan tantos campos.

```
SELECT campo1, campo2, ...,campoN FROM tabla1 JOIN tabla2 ON

tabla1.campo1 = tabla2.campo2 JOIN tabla3 ON tabla2.campo3 =

tabla3.campo4
```

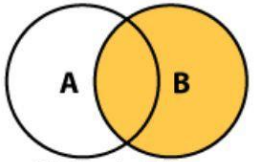
SQL JOINS



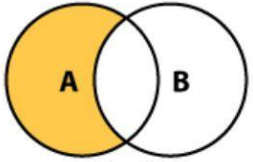

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key=B.Key
```



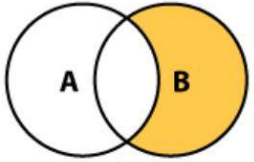

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key=B.Key
```



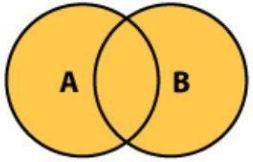

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key=B.Key
```



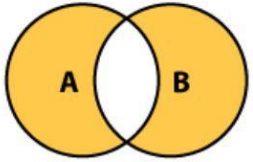

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key=B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key=B.Key
WHERE A.Key IS NULL
```



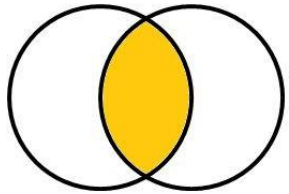

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key=B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key=B.Key
WHERE A.Key IS NULL
OR B.Key IS NULLL
```

INNER JOIN

INNER JOIN traerá solamente los registros que coincidan entre ambas tablas. Por ejemplo: si una escuela no tiene alumnos relacionados, esa consulta no los traerá, del mismo modo si un alumno no tiene asignada una escuela tampoco lo mostrará.

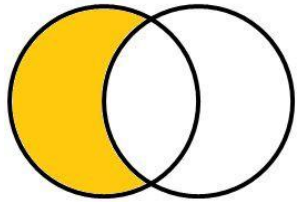


```sql
-- Mostrar el legajo, el nombre
-- y el nombre de la escuela de todos los alumnos
SELECT alu.legajo, alu.nombre, esc.nombre
FROM alumnos alu
INNER JOIN escuelas esc ON alu.id_escuela = esc.id;
```

*Dentro del INNER JOIN establecemos que para la tabla **escuelas** el campo **id** está relacionado con el campo el campo **id_escuela** de la tabla alumnos.*

LEFT JOIN

LEFT JOIN tiene como condición que figure en, al menos una tabla. Left indica que va a tomar como tabla principal la de la izquierda. De esa tabla muestra todos los registros, sin importar si tiene registros asociados en la otra tabla.



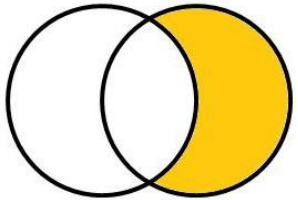
```sql
-- Mostrar TODOS los alumnos con los datos de la escuela (opcional)
SELECT alu.legajo, alu.nombre, esc.nombre
FROM alumnos alu
LEFT JOIN escuelas esc ON alu.id_escuela = esc.id;
```

legajo	nombre	nombre
101	Juan Perez	Normal 1
105	Pedro Go...	Normal 1
106	Martín Bo...	Normal 1
1000	Ramón M...	Gral. San ...
1002	Tomás Smith	Gral. San ...
100	Ramiro Es...	EET Nro 2
1234	Pedro Gó...	EET Nro 2
190	Roberto L...	NULL

El último alumno no tiene escuela asociada o tiene una escuela asociada que no existe.

RIGHT JOIN

RIGHT JOIN tiene como condición que figure en, al menos una tabla. Right indica que va a tomar como tabla principal la de la derecha. De esa tabla muestra todos los registros, sin importar si tiene registros asociados en la otra tabla.



```sql
-- Mostrar TODAS las escuelas con el nombre de cada alumno
SELECT esc.*, alu.nombre
FROM escuelas esc
RIGHT JOIN alumnos alu ON esc.id = alu.id_escuela;
```

id	nombre	localidad	provincia	capacidad	nombre
1	Normal 1	Quilmes	Buenos Aires	250	Juan Perez
1	Normal 1	Quilmes	Buenos Aires	250	Pedro Go...
2	Gral. San ...	San Salvador	Jujuy	100	Ramón M...
2	Gral. San ...	San Salvador	Jujuy	100	Tomás Smith
4	EET Nro 2	Avellaneda	Buenos Aires	500	Ramiro Es...
4	EET Nro 2	Avellaneda	Buenos Aires	500	Pedro Gó...
NULL	NULL	NULL	NULL	NULL	Roberto L...
NULL	NULL	NULL	NULL	NULL	Martín Bo...

Los últimos dos alumnos no tienen escuela asignada o tienen una escuela asignada que no existe.

Funciones de agregación

Las **funciones de agregación** más comunes disponibles en el lenguaje son: SUM, AVG, MAX, MIN, COUNT. La sintaxis del uso de las funciones agregadas es la siguiente:

```
SELECT <lista de campos>, función agregada FROM <tabla1 JOIN tabla2 ON....>
GROUP BY <lista de campos>
[HAVING función agregada <condición>]
```

Reúnen un conjunto de registros para **agrupar los datos** y llevar a cabo la operación en cuestión (suma, promedio, cuenta, etc), de ahí la necesidad de la cláusula **GROUP BY**.

Uso de funciones agregadas

La *lista de campos* en el GROUP BY y en el SELECT es la misma. Si no hay una lista de campos significa que vamos a obtener una suma total, por lo tanto la cláusula GROUP BY tampoco es necesaria.
Estas cláusulas se utilizan conjuntamente con el SELECT ya que siempre van asociadas a una consulta.

Ejemplo: uso de función **SUM**

```
SELECT SUM(cant*pr.precio) FROM pedidos_productos pp
JOIN productos pr ON pp.codproducto=pr.codigo JOIN pedidos p ON
p.nro=pp.codpedido
WHERE month(p.fecha)=1 and year(p.fecha)=2017
```

Uso de funciones agregadas

Si quisiéramos saber cuántas unidades se vendieron de cada producto por mes para el 2017, podríamos formularlo de la siguiente manera:

```
SELECT month(p.fecha) as Mes, SUM(cant) as Cantidad FROM pedidos_productos
pp
JOIN productos pr ON pp.codproducto=pr.codigo JOIN pedidos p ON
p.nro=pp.codpedido
WHERE year(p.fecha)=2017 GROUP BY month(p.fecha)
```

Uso de funciones agregadas

Si hubiéramos querido saber los pedidos cuyo total fuera superior a $ 1.000 podríamos hacer lo siguiente:

```
SELECT p.Nro, SUM(cant*precio) as total FROM pedidos_productos pp
JOIN productos pr ON pp.codproducto=pr.codigo JOIN pedidos p ON
p.nro=pp.codpedido
GROUP BY p.nro
HAVING SUM(cant*precio)>1000
```

¿Cómo funcionan las cláusulas Having y Where?

- **WHERE** opera sobre registros individuales, mientras que **HAVING** lo hace sobre un grupo de registros.
- Con **WHERE** podemos establecer una condición usando ***registros individuales***, aquellos que cumplan con esta condición serán seleccionados (eliminados o actualizados). Con **HAVING** podemos establecer una condición ***sobre un grupo de registros***.
- **HAVING** acostumbra ir acompañado de la cláusula GROUP BY, dado que opera sobre los grupos que nos "retorna" GROUP BY.

¿Cuándo usar HAVING o WHERE? Deberíamos usar HAVING solo cuando se vea implicado el uso de funciones de grupo (AVG, SUM, COUNT, MAX, MIN), debido a que con WHERE no podemos realizar condiciones que impliquen estas funciones.

Funciones escalares

Tras ver las funciones de agregación, vamos a ver las **funciones escalares del lenguaje de consultas**.

Al igual que en cualquier otro lenguaje de programación, SQL dispone de una serie de funciones que nos facilitan la obtención de los resultados deseados. Éstas se pueden utilizar en las cláusulas SELECT, WHERE y ORDER BY que ya hemos estudiado. Otra característica a tener en cuenta es que se pueden anidar, es decir, una función puede llamar a otra función.

Funciones escalares

En el lenguaje SQL estándar existen básicamente **5 tipos de funciones**: **aritméticas**, de **cadenas** de caracteres, de **fechas**, de **conversión**, y **otras funciones** que no se pueden incluir en ninguno de los grupos anteriores.

Nos vamos a limitar a citar algunas de las más significativas, incluyendo una breve descripción de las mismas. +info

Funciones escalares | Aritméticas

Funciones aritméticas: Todas las funciones matemáticas devuelven NULL en caso de error. +info

- ABS(n): Devuelve el valor absoluto de "n".
- ROUND(m, n): Redondea el número "m" con el número de decimales indicado en "n", si no se indica "n" asume cero decimales.
- SQRT(n): Devuelve la raíz cuadrada del parámetro que se le pase.
- POWER(m, n): Devuelve la potencia de "m" elevada al exponente "n".

Funciones escalares | Cadenas

Funciones de cadenas: Para funciones que operan en posiciones de cadena, la primera posición se numera 1. +info

- LOWER(c): Devuelve la cadena "c" con todas las letras convertidas a minúsculas.
- UPPER(c): Devuelve la cadena "c" con todas las letras convertidas a mayúsculas.
- LTRIM(c): Elimina los espacios por la izquierda de la cadena "c".
- RTRIM(c): Elimina los espacios por la derecha de la cadena "c".
- REPLACE(c, b, s): Sustituye en la cadena "c" el valor buscado "b" por el valor indicado en "s".
- LEFT(c, n): Devuelve "n" caracteres por la izquierda de la cadena "c".
- RIGHT(c, n): Devuelve "n" caracteres por la derecha de la cadena "c".
- SUBSTRING(c, m, n): Devuelve una sub-cadena obtenida de la cadena "c", a partir de la posición "m" y tomando "n" caracteres.

Funciones escalares | Fechas

Funciones de manejo de fechas: Las funciones que esperan valores de fecha generalmente aceptan valores de fecha y hora e ignoran la parte de la hora. Las funciones que esperan valores de hora generalmente aceptan valores de fecha y hora e ignoran la parte de la fecha. +info

- YEAR(d): Devuelve el año correspondiente de la fecha “d”.
- MONTH(d): Devuelve el mes de la fecha “d”.
- DAY(d): Devuelve el día del mes de la fecha “d”.
- DATE_ADD(): Agrega valores de tiempo (intervalos) a un valor de fecha.

Funciones escalares | Conversión

Funciones de conversión: Estas funciones suelen ser específicas de cada gestor de datos, ya que cada SGBD (SQL Server, Oracle, MySQL) utiliza nombres diferentes para los distintos tipos de datos (aunque existen similitudes se dan muchas diferencias).

Las funciones de conversión nos permiten cambiar valores de un tipo de datos a otro. Por ejemplo, si tenemos una cadena y sabemos que contiene una fecha, podemos convertirla al tipo de datos fecha.

En **MySQL**, por ejemplo, solucionamos la mayor parte de las conversiones con la función **CAST**. +info

Ejemplo:

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
        -> 38.8, '38.8'
mysql> SELECT 38.8, CONCAT(38.8);
        -> 38.8, '38.8'
```

Subconsultas

Consiste en utilizar los resultados de una consulta dentro de otra, que se considera la principal. Esta posibilidad fue la razón original para la palabra "estructurada" en el nombre Lenguaje de Consultas Estructuradas (*Structured Query Language, SQL*).

```
SELECT numemp, nombre, (SELECT MIN(fechapedido) FROM pedidos WHERE rep = numemp)
FROM empleados;
```

En este ejemplo la consulta principal es **SELECT... FROM empleados**.
La subconsulta es (**SELECT MIN (fechapedido) FROM pedidos WHERE rep = numemp**).
En esta subconsulta tenemos una referencia externa (numemp) es un campo de la tabla empleados (origen de la consulta principal).

Subconsultas

¿Qué pasa cuando se ejecuta la consulta principal?

- Se toma el primer empleado y se calcula la subconsulta sustituyendo **numemp** por el valor que tiene en el primer empleado. La subconsulta obtiene la fecha más antigua en los pedidos del **rep = 101**,
- Se toma el segundo empleado y se calcula la subconsulta con **numemp = 102** (**numemp** del segundo empleado)... y así sucesivamente hasta llegar al último empleado.
- Al final obtenemos una lista con el número, nombre y fecha del primer pedido de cada empleado.

Operador EXISTS

El operador **EXISTS** se utiliza para probar la existencia de cualquier registro en una subconsulta. El operador EXISTS devuelve VERDADERO si la subconsulta devuelve uno o más registros.

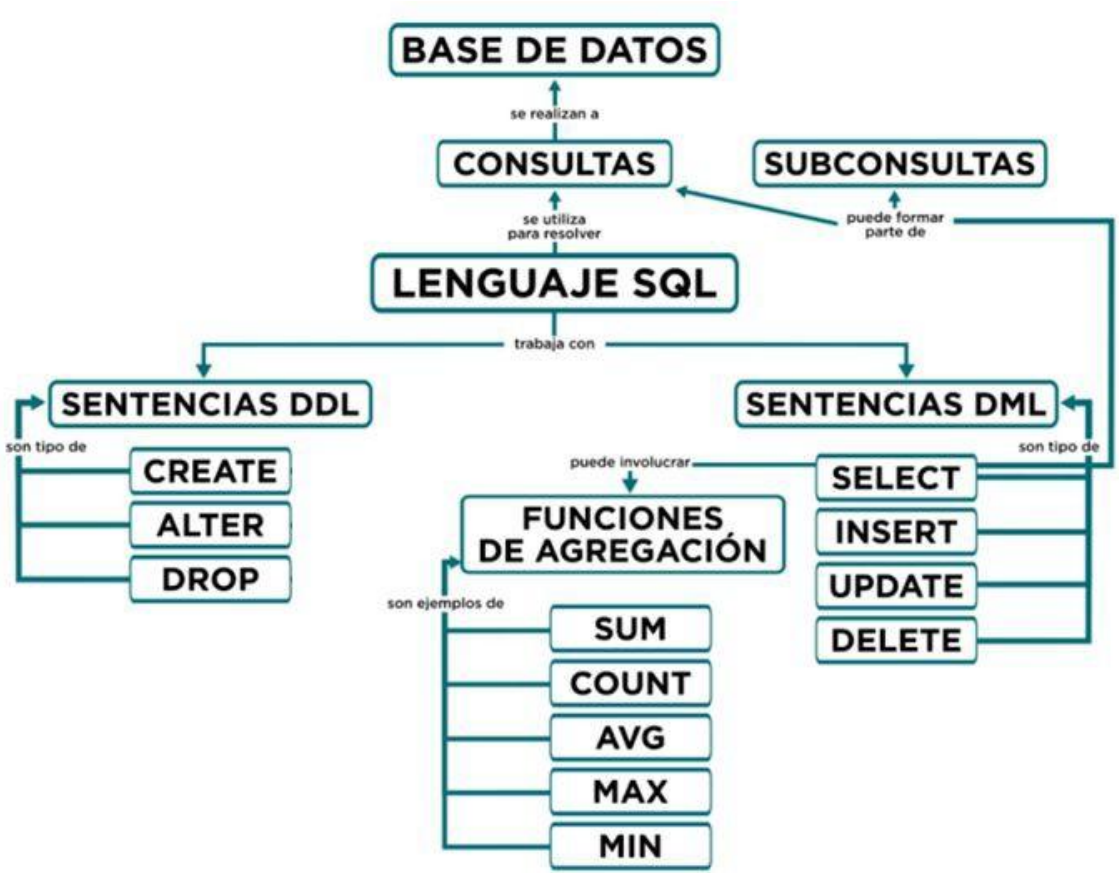
```
SELECT column_name(s) FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

Ejemplo EXISTS

La siguiente instrucción SQL devuelve VERDADERO y enumera los proveedores con un precio de producto inferior a 20:

```
SELECT SupplierName FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE
Products.SupplierID = Suppliers.supplierID AND Price < 20);
```

Síntesis SQL



Material extra

Artículos de interés

Material extra:

- Campusmvp.es: Funciones de agregación
- Base de datos en SQL (curso)
- SQL desde cero (curso)
- SQL HAVING y WHERE: Las diferencias que necesita saber

Videos:

- Curso de Microsoft SQL Server
- La historia completa de las bases de datos SQL (o relacionales)

No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.
- Realizar los Ejercicios de repaso.

Todo en el Aula Virtual.

Muchas gracias por tu atención. Nos vemos pronto