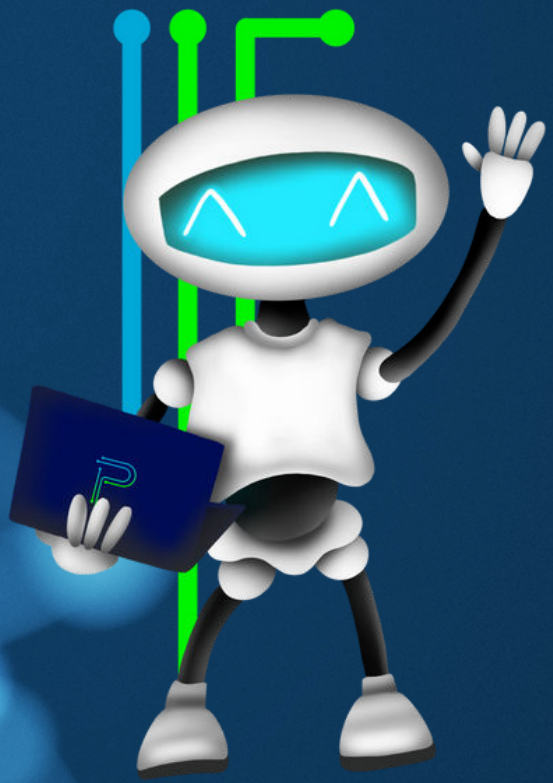
pontia

GLOSARIO

Listado con sentencias clave para SQL



Sentencias básicas

Funciones de agregación

Funciones de cadena

Funciones de fecha

Joins

Operadores lógicos

Definición de datos

Manipulación de datos

Otras funciones

SENTENCIAS BÁSICAS

FROM

Es una cláusula en SQL que especifica la fuente de los datos que se van a seleccionar con una sentencia **"SELECT"**. La sintaxis básica es la siguiente:

```
SELECT column1, column2, ...
FROM table_name;
```

Donde "**table_name**" es el nombre de la tabla de la que desea extraer los datos y "**column1, column2, etc.**" son las columnas específicas que desea seleccionar.

Esta sentencia es una parte esencial de una sentencia SELECT, ya que especifica la fuente de los datos que se van a seleccionar. Es posible seleccionar datos de una o más tablas, así como de vistas, subconsultas o unión de múltiples tablas mediante la cláusula FROM.

Es importante destacar que esta cláusula es independiente de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos.

GROUP BY

Es una cláusula en SQL que se utiliza para agrupar los registros de una tabla o consulta según una o más columnas específicas. La cláusula "**GROUP BY"** permite agrupar los datos de una consulta SELECT y realizar operaciones matemáticas y estadísticas sobre ellos, como sumas, promedios, máximos, mínimos, etc.

La sintaxis básica es la siguiente:

```
SELECT column1, SUM(column2), ...
FROM table
GROUP BY column1;
```

Donde **“table”** es el nombre de la tabla que se quiere agrupar, "**column1"** es la columna por la que se desea agrupar los datos, y “**SUM(column2)”** es una función de agregación que se utiliza para realizar una operación matemática sobre los datos agrupados.

Es importante destacar que las funciones de agregación como **SUM, AVG, MAX, MIN, etc.** Sólo pueden aplicarse a las columnas incluidas en la cláusula GROUP BY, si se han agrupado los datos de la consulta.

Esta sentencia independiente de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos.

HAVING

Es una cláusula en SQL que se utiliza para filtrar los resultados de una consulta agrupada por la cláusula **"GROUP BY"**. La cláusula "**HAVING**" permite aplicar una condición a los grupos de datos generados por la cláusula GROUP BY y devolver sólo aquellos grupos que cumplen con la condición especificada.

La sintaxis básica es la siguiente:

```
SELECT column1, SUM(column2), ...
FROM table
GROUP BY column1
HAVING SUM(column2) > 100;
```

Donde "**table**" es el nombre de la tabla que se quiere agrupar, "**column1**" es la columna por la que se desea agrupar los datos, "**SUM(column2)**" es una función de agregación que se utiliza para realizar una operación matemática sobre los datos agrupados, y **"HAVING SUM(column2) > 100"** es la condición que se aplica a los grupos de datos generados por la cláusula GROUP BY.

Es importante destacar que esta cláusula, sólo puede utilizarse después de una cláusula GROUP BY y que las condiciones especificadas en la cláusula HAVING sólo pueden aplicarse a las funciones de agregación incluidas en la consulta.

La cláusula "HAVING" es independiente de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos.

ORDER BY

Es una cláusula en SQL que se utiliza para ordenar el conjunto de resultados en "orden ascendente o descendente" basado en una o más columnas. Se utiliza típicamente en conjunto con una sentencia "**SELECT**" para ordenar los datos devueltos por la consulta.

Sintaxis:

```
SELECT columna1, columna2, ...
FROM nombre_tabla
ORDER BY columna1 [ASC|DESC], columna2 [ASC|DESC], ...
```

Aquí, **"columna1, columna2, etc"**. son las columnas que se utilizan para ordenar el conjunto de resultados, y **"nombre_tabla"** es el nombre de la tabla que contiene los datos. La palabra clave "**ASC**" se utiliza para ordenar los datos en orden ascendente (que es el valor predeterminado), mientras que la palabra clave **"DESC"** se utiliza para ordenar los datos en orden descendente.

Por ejemplo, la siguiente consulta ordena los empleados en la tabla empleados por su salario en orden descendente:

```
SELECT id_empleado, nombre, apellido, salario
FROM empleados
ORDER BY salario DESC;
```

SELECT

Es una sentencia SQL que se utiliza para seleccionar y extraer datos de una o más tablas de una base de datos.

La sintaxis básica de esta sentencia es la siguiente:

```
SELECT column1, column2, ...
FROM table_name;
```

Donde **"column1, column2, etc".** son las columnas específicas que desea seleccionar y "**table_name**" es el nombre de la tabla de la que desea extraer los datos. La sentencia SELECT devuelve un conjunto de resultados que contiene una fila para cada registro que cumple con la consulta.

La sentencia SELECT es una de las sentencias más importantes y utilizadas en SQL, ya que se utiliza para realizar la mayoría de las consultas a la base de datos. Además, permite realizar una variedad de operaciones en los datos, como filtrar, ordenar y agrupar los resultados, así como combinar datos de múltiples tablas.

Es importante destacar que esta sentencia es independiente de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos.

WHERE

Es una cláusula en SQL que se utiliza para especificar las condiciones que deben cumplir los registros seleccionados con una sentencia "**SELECT**".

La sintaxis básica de esta cláusula es la siguiente:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Donde "**condition**" es una expresión que define las condiciones que deben cumplir los registros para ser incluidos en el resultado de la consulta.

La cláusula "WHERE" es una parte esencial de una sentencia "SELECT", ya que permite filtrar los datos seleccionados para obtener sólo aquellos que cumplen con ciertas condiciones. Por ejemplo, es posible utilizar la cláusula "WHERE" para seleccionar solo los registros que tienen un valor determinado en una columna específica, o que se encuentran en un rango determinado de fechas.

Es importante destacar que esta cláusula es independiente de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos.

FUNCIONES DE AGREGACIÓN

AVG

Es una función de agregación en SQL que se utiliza para calcular el promedio (media aritmética) de los valores en una columna en una tabla. La función "**AVG**" devuelve un número que representa el promedio de los valores en una columna.

La sintaxis básica de la función es la siguiente:

```
SELECT AVG(column)
FROM table;
```

Donde "**table**" es el nombre de la tabla en la que se desea calcular el promedio de los valores en una columna y "**column**" es el nombre de la columna en la que se desea calcular el promedio.

También es posible calcular el promedio de los valores en una columna que cumplen con una determinada condición, por ejemplo:

```
SELECT AVG(column)
FROM table
WHERE condition;
```

Donde “**condition**” es la condición que se debe cumplir para calcular el promedio de los valores en una columna.

La función “**AVG**” es independiente de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos.

COUNT

Es una función de agregación en SQL que se utiliza para contar el número de filas en una tabla que cumplen con una determinada condición. La función "**COUNT**" devuelve un número entero que representa el número de filas en una tabla.

La sintaxis básica de la función es la siguiente:

```
SELECT COUNT(*)
FROM table;
```

Donde “**table**” es el nombre de la tabla en la que se desea contar el número de filas. El asterisco **(*)** dentro de paréntesis es una representación de todas las columnas de la tabla.

También es posible contar el número de filas en una tabla que cumplen con una determinada condición, por ejemplo:

```
SELECT COUNT(*)
FROM table
WHERE column = value;
```

Donde “**value**” es el valor que se desea comparar con los valores de la columna (column) en la tabla (table).

MAX

Es una función de agregación en SQL que se utiliza para calcular el valor máximo en una columna en una tabla. La función "**MAX**" devuelve el valor más grande en una columna.

La sintaxis básica de la función es la siguiente:

```
SELECT MAX(column)
FROM table;
```

Donde "**table**" es el nombre de la tabla en la que se desea calcular el valor máximo en una columna y "**column**" es el nombre de la columna en la que se desea calcular el valor máximo.

También es posible calcular el valor máximo en una columna que cumplen con una determinada condición, por ejemplo:

```
SELECT MAX(column)
FROM table
WHERE condition;
```

Donde "**condition**" es la condición que se debe cumplir para calcular el valor máximo en una columna.

La función MAX es independiente de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos.

MIN

Es una función de agregación en SQL que se utiliza para calcular el valor mínimo en una columna en una tabla. La función "**MIN**" devuelve el valor más pequeño en una columna.

La sintaxis básica de la función es la siguiente:

```
SELECT MIN(column)
FROM table;
```

Donde "**table**" es el nombre de la tabla en la que se desea calcular el valor mínimo en una columna y "**column**" es el nombre de la columna en la que se desea calcular el valor mínimo.

También es posible calcular el valor mínimo en una columna que cumplen con una determinada condición, por ejemplo.

```
SELECT MIN(column)
FROM table
WHERE condition;
```

Donde "**condition**" es la condición que se debe cumplir para calcular el valor mínimo en una columna.

La función MIN es independiente de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos

SUM

Es una función de agregación en SQL que se utiliza para calcular la suma de los valores numéricos en una columna de una tabla. La función "**SUM**" toma como argumento la columna que se va a sumar y devuelve la suma de los valores de la columna.

La sintaxis básica es la siguiente:

```
SELECT SUM(columna) FROM tabla;
```

Aquí, "**columna**" es la columna de la tabla que se desea sumar, y tabla es el nombre de la tabla que contiene los datos.

Por ejemplo, si se tiene una tabla llamada ventas que contiene las ventas mensuales de una empresa, se puede utilizar la función SUM para calcular el total de ventas para todo el año:

```
SELECT SUM(ventas_mensuales) AS total_ventas
FROM ventas;
```

La función también se puede utilizar en combinación con otras cláusulas SQL como GROUP BY para calcular sumas en subconjuntos de datos. Por ejemplo, se puede utilizar la siguiente consulta para calcular la suma de las **"ventas mensuales"** para cada vendedor:

```
SELECT vendedor, SUM(ventas_mensuales) AS total_ventas
FROM ventas
GROUP BY vendedor;
```

Aquí, la consulta agrupa los datos por vendedor y utiliza la función SUM para calcular la suma de las ventas mensuales para cada vendedor.

FUNCIONES DE CADENA

CONCAT

Es una función en SQL que se utiliza para concatenar dos o más cadenas de texto en una sola cadena. La sintaxis general de la función "**CONCAT**" es la siguiente:

```
CONCAT(string1, string2, ...)
```

Donde "**string1, string2**", etc. son las cadenas de texto que se desean concatenar.

Por ejemplo, la siguiente consulta SQL muestra cómo utilizar la función, para combinar el nombre y el apellido de un empleado en una sola cadena:

```
SELECT CONCAT(nombre, ' ', apellido) AS nombre_completo
FROM empleados;
```

En esta consulta, la función, se utiliza para combinar las columnas nombre y apellido en una sola cadena separada por un espacio. El resultado se mostrará en una nueva columna llamada "nombre_completo".

La función CONCAT es especialmente útil para unir varias columnas o cadenas de texto en una sola cadena, lo que puede ser útil en informes y otras tareas de manipulación de datos.

LOWER

Es una función en SQL que se utiliza para convertir una cadena de texto en minúsculas. La sintaxis general de la función "**LOWER**" es la siguiente:

```
LOWER(string)
```

Donde "**string**" es la cadena de texto que se desea convertir a minúsculas.

Por ejemplo, la siguiente consulta SQL muestra cómo utilizar la función, para convertir un nombre de ciudad en minúsculas:

```
SELECT LOWER(nombre_ciudad) AS ciudad
FROM ciudades;
```

En esta consulta, la función, se utiliza para convertir el valor de la columna "**nombre_ciudad**" en minúsculas. El resultado se mostrará en una nueva columna llamada ciudad.

La función LOWER es útil cuando se necesita comparar cadenas de texto, ya que la comparación sensible a mayúsculas y minúsculas puede producir resultados inesperados. Al convertir las cadenas de texto en minúsculas (o mayúsculas con la función "**UPPER**"), se puede garantizar que las comparaciones de cadenas de texto sean más precisas.

REPLACE

Es una función en SQL que se utiliza para buscar y reemplazar todas las ocurrencias de una cadena de texto en otra cadena. La sintaxis general de la función "**REPLACE**" es la siguiente:

```
REPLACE(string, search_string, replacement_string)
```

Donde "**string**" es la cadena de texto original en la que se va a buscar, "**search_string**" es la cadena de texto que se va a buscar y "**replacement_string**" es la cadena de texto que se utilizará para reemplazar todas las ocurrencias de la cadena de búsqueda.

Por ejemplo, la siguiente consulta SQL muestra cómo utilizar la función para reemplazar todas las ocurrencias de la cadena de búsqueda "**ABC**" en una columna llamada "**descripcion**" con la cadena de reemplazo "**XYZ**":

```
SELECT REPLACE(descripcion, 'ABC', 'XYZ') AS nueva_descripcion
FROM productos;
```

En esta consulta, la función REPLACE se utiliza para buscar y reemplazar todas las ocurrencias de la cadena de búsqueda "ABC" en la columna descripcion con la cadena de reemplazo "XYZ". El resultado se mostrará en una nueva columna llamada "**nueva_descripcion**".

La función REPLACE es especialmente útil para limpiar y normalizar datos, como cambiar el formato de fechas o corregir errores tipográficos en nombres o descripciones.

TRIM

Es una función en SQL que se utiliza para eliminar espacios en blanco o caracteres especificados de una cadena de texto. La sintaxis general de la función "**TRIM**" es la siguiente:

```
TRIM([BOTH | LEADING | TRAILING] [caracter] FROM string)
```

Donde:

- BOTH, LEADING o TRAILING indican si se deben eliminar los espacios en blanco o caracteres especificados al inicio (LEADING), al final (TRAILING) o en ambos extremos (BOTH) de la cadena de texto. Si no se especifica nada, se asume BOTH.
- Carácter es el carácter o caracteres que se desea eliminar de la cadena de texto. Si no se especifica nada, se eliminan los espacios en blanco.
- String es la cadena de texto que se desea procesar.

Por ejemplo, la siguiente consulta SQL muestra cómo utilizar la función TRIM para eliminar los espacios en blanco de la columna nombre de la tabla clientes:

```
SELECT TRIM(nombre) AS nombre_sin_espacios
FROM clientes;
```

En esta consulta, la función, se utiliza para eliminar todos los espacios en blanco de los valores de la columna nombre. El resultado se mostrará en una nueva columna llamada nombre_sin_espacios.

La función TRIM es útil cuando se desean eliminar espacios en blanco o caracteres no deseados de una cadena de texto antes de realizar búsquedas o comparaciones de cadenas.

UPPER

Es una función en SQL que se utiliza para convertir una cadena de texto a mayúsculas. La sintaxis general de la función "**UPPER**" es la siguiente:

```
UPPER(string)
```

Donde "**string**" es la cadena de texto que se desea convertir a mayúsculas.

Por ejemplo, la siguiente consulta SQL muestra cómo utilizar la función UPPER para convertir a mayúsculas la columna nombre de la tabla clientes:

```
SELECT UPPER(nombre) AS nombre_mayusculas
FROM clientes;
```

En esta consulta, la función, se utiliza para convertir a mayúsculas todos los valores de la columna nombre. El resultado se mostrará en una nueva columna llamada "**nombre_mayusculas**".

La función UPPER es útil cuando se desea realizar búsquedas o filtrar datos sin importar la capitalización de las letras en las cadenas de texto.

FUNCIONES DE FECHA

CURDATE

Es una función en SQL que se utiliza para obtener la fecha actual del sistema en el formato estándar de fecha **YYYY-MM-DD** (año-mes-día). La función **CURDATE()** no requiere parámetros, simplemente se utiliza como una función de SQL en una consulta.

Por ejemplo, la siguiente consulta SQL mostrará todos los registros de una tabla donde la fecha es igual a la fecha actual:

```
SELECT *
FROM tabla
WHERE fecha = CURDATE();
```

En esta consulta, fecha es una columna que contiene valores de fecha y la condición fecha = CURDATE() se utiliza para seleccionar solo los registros donde la fecha es igual a la fecha actual del sistema.

También existe una función relacionada llamada **“NOW()”** que devuelve la fecha y la hora actuales del sistema en formato **“YYYY-MM-DD HH:MI:SS”** (año-mes-día hora:minutos:segundos).

DATE

Es un tipo de dato que se utiliza para almacenar valores de fecha (día, mes, año) sin incluir información de hora. El formato estándar para las fechas en SQL es **YYYY-MM-DD** (año-mes-día).

Los valores de fecha se pueden utilizar en consultas SQL para filtrar y ordenar resultados de búsqueda. También se pueden realizar operaciones aritméticas en las fechas utilizando funciones como **DATEADD y DATEDIFF** para agregar o restar días, semanas, meses, años, etc. de una fecha determinada o para calcular la diferencia entre dos fechas.

Por ejemplo, en la siguiente consulta SQL se muestran los registros de una tabla donde la fecha es posterior al 1 de enero de 2022:

```
SELECT *
FROM tabla
WHERE fecha > '2022-01-01';
```

En esta consulta, fecha es una columna que contiene valores de fecha y la condición fecha > **'2022-01-01'** se utiliza para seleccionar solo los registros donde la fecha es posterior al 1 de enero de 2022.

DATEADD

Es una función en SQL que se utiliza para agregar una cantidad específica de tiempo (días, semanas, meses, años, etc.) a una fecha determinada y devolver la nueva fecha resultante. La sintaxis para la función **DATEADD** es la siguiente:

```
DATEADD(datepart, number, date)
```

Donde "**datepart**" es la unidad de tiempo que se va a agregar (por ejemplo, 'day' para días, '**week**' para semanas, '**month**' para meses, '**year**' para años, etc.), "**number**" es el número de unidades de tiempo que se van a agregar y date es la fecha a la que se le va a agregar la cantidad de tiempo especificada.

Por ejemplo, si deseas agregar 5 días a la fecha '2022-02-10', la sentencia sería la siguiente:

```
SELECT DATEADD(day, 5, '2022-02-10');
```

El resultado de esta sentencia sería la fecha **'2022-02-15'**, que es la fecha resultante de agregar 5 días a la fecha especificada.

También es posible utilizar DATEADD para restar una cantidad específica de tiempo de una fecha, simplemente especificando un número negativo en el segundo parámetro.

DATEDIFF

Es una función en SQL que se utiliza para calcular la diferencia entre dos fechas en términos de una unidad de tiempo específica (días, semanas, meses, años, etc.). La sintaxis para la función DATEDIFF es la siguiente:

```
DATEDIFF(datepart, startdate, enddate)
```

Donde "**datepart**" es la unidad de tiempo que se va a utilizar para calcular la diferencia **(por ejemplo, 'day' para días, 'week' para semanas, 'month' para meses, 'year' para años, etc.)**, "**startdate**" es la fecha de inicio y enddate es la fecha de finalización.

Por ejemplo, si deseas calcular la diferencia en días entre las fechas '2022-02-10' y '2022-02-15', la sentencia sería la siguiente:

```
SELECT DATEDIFF(day, '2022-02-10', '2022-02-15');
```

El resultado de esta sentencia sería el número "**5**", que es la diferencia en días entre las dos fechas especificadas.

También es posible utilizar DATEDIFF para calcular la diferencia en términos de otras unidades de tiempo, simplemente especificando la unidad de tiempo adecuada en el primer parámetro.

DATEPART

Es una función que se utiliza para obtener una parte específica de una fecha y hora. La función toma dos parámetros: la parte de la fecha que se desea obtener **(por ejemplo, año, mes, día, hora, minuto, segundo, etc.)** y la fecha de la que se desea obtener la parte.

La sintaxis general de esta función es la siguiente:

```
DATEPART(part, date)
```

Donde "**part**" es la parte de la fecha que se desea obtener (por ejemplo, (**YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, etc.)** y (**date**) es la fecha de la que se desea obtener la parte.

Por ejemplo, la siguiente consulta SQL mostrará el mes de una fecha específica:

```
SELECT DATEPART(month, '2022-02-10');
```

En esta consulta, la función DATEPART se utiliza para obtener el mes de la fecha **"2022-02-10"**. El resultado de la consulta sería **2**, que es el mes de febrero.

Esta función, se utiliza a menudo en combinación con otras funciones de fecha y hora en SQL Server para realizar cálculos y filtrar datos en consultas.

DAY

Es una función que devuelve el día del mes **(un número entre 1 y 31)** de una fecha dada. La sintaxis para utilizar la función **"DAY"** es la siguiente:

```
DAY(date)
```

Donde (**date**) es la fecha de la cual se desea obtener el día del mes.

Por ejemplo, si deseas obtener el día del mes de la fecha **'2022-02-10'**, la sentencia sería la siguiente:

```
SELECT DAY('2022-02-10');
```

El resultado de esta sentencia sería el número 10, que es el día del mes correspondiente a la fecha especificada.

MONTH

Es una función que devuelve el número del mes **(un número entre 1 y 12)** de una fecha dada. La sintaxis para utilizar la función "**MONTH**" es la siguiente:

```
MONTH(date)
```

Donde (**date**) es la fecha de la cual se desea obtener el número del mes.

Por ejemplo, si deseas obtener el número del mes de la fecha **'2022-02-10'**, la sentencia sería la siguiente

```
SELECT MONTH('2022-02-10');
```

YEAR

La función "**YEAR**" es una función de fecha que devuelve el año de una fecha dada. La sintaxis básica de la función es la siguiente:

```
YEAR(date)
```

Donde "**date**" es el valor de fecha para el que se desea calcular el año. La función devuelve un número entero que representa el año de la fecha dada.

Esta función es útil para clasificar y seleccionar fechas en función del año.

Por ejemplo, puede utilizarse para obtener todas las fechas que se encuentran en un determinado año o para generar informes anuales agrupados por año. Es importante destacar que la función es independiente de la base de datos utilizada y de las configuraciones regionales.

JOINS

FULL OUT JOIN

Es un tipo de operación de unión en SQL que se utiliza para combinar filas de dos o más tablas en una sola tabla resultante. La cláusula **"FULL OUTER JOIN"** se utiliza para seleccionar todos los registros de ambas tablas, tanto aquellos que cumplen la condición de unión como aquellos que no la cumplen.

La sintaxis básica de esta cláusula es la siguiente:

```
SELECT column1, column2, ...
FROM table1
FULL OUTER JOIN table2
ON table1.column = table2.column;
```

Donde "**table1 y table2**" son los nombres de las tablas que se quieren unir, **"column1 y column2"** son las columnas que se desean incluir en el resultado de la consulta. La cláusula "**ON**" se utiliza para especificar la condición de unión, que es una expresión que define cómo deben relacionarse los registros de las dos tablas.

El resultado de una cláusula “FULL OUTER JOIN” es una tabla que contiene todos los registros de ambas tablas, tanto aquellos que cumplen la condición de unión como aquellos que no la cumplen. Si no hay correspondencias para un registro en una de las tablas, las columnas correspondientes en la tabla resultante contendrán valores nulos.

Es importante destacar que la cláusula “FULL OUTER JOIN” es independiente de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos.

INNER JOIN

Es un tipo de operación de unión en SQL que se utiliza para combinar filas de dos o más tablas en una sola tabla resultante. La cláusula **"INNER JOIN"** se utiliza para seleccionar solo aquellos registros que tienen correspondencias en ambas tablas.

La sintaxis básica de esta cláusula es la siguiente:

```
SELECT column1, column2, ...
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

Donde **“table1 y table2”** son los nombres de las tablas que se quieren unir, "**column1 y column2"** son las columnas que se desean incluir en el resultado de la consulta. La cláusula ON se utiliza para especificar la condición de unión, que es una expresión que define cómo deben relacionarse los registros de las dos tablas.

El resultado de esta cláusula es una tabla que contiene solo aquellos registros que cumplen con la condición de unión. Por ejemplo, si se quiere combinar los registros de una tabla de clientes con los registros de una tabla de pedidos, una cláusula INNER JOIN permitiría seleccionar solo aquellos clientes que han realizado algún pedido.

Es importante destacar que esta cláusula es independiente de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos.

LEFT JOIN

Es un tipo de operación de unión en SQL que se utiliza para combinar filas de dos o más tablas en una sola tabla resultante. Esta cláusula se utiliza para seleccionar todos los registros de la tabla a la izquierda **(denotada por la cláusula FROM)** y aquellos registros de la tabla a la derecha **(denotada por la cláusula LEFT JOIN)** que cumplan la condición de unión.

La sintaxis básica de una cláusula LEFT JOIN es la siguiente:

```
SELECT column1, column2, ...
FROM table1
LEFT JOIN table2
ON table1.column = table2.column;
```

Donde "**table1 y table2**" son los nombres de las tablas que se quieren unir, y "**column1 y column2"** son las columnas que se desean incluir en el resultado de la consulta. La cláusula "**ON**" se utiliza para especificar la condición de unión, que es una expresión que define cómo deben relacionarse los registros de las dos tablas.

El resultado de esta cláusula, es una tabla que contiene todos los registros de la tabla a la izquierda y aquellos registros de la tabla a la derecha que cumplan la condición de unión. Si no hay correspondencias para un registro en la tabla a la derecha, las columnas correspondientes en la tabla resultante contendrán valores "**nulos**".

Es importante destacar que la cláusula LEFT JOIN es independiente de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos.

RIGHT JOIN

Es un tipo de operación de unión en SQL que se utiliza para combinar filas de dos o más tablas en una sola tabla resultante. Esta cláusula se utiliza para seleccionar todos los registros de la tabla a la derecha **(denotada por la cláusula RIGHT JOIN)** y aquellos registros de la tabla a la izquierda **(denotada por la cláusula FROM)** que cumplan la condición de unión.

La sintaxis básica de esta cláusula es la siguiente:

```
SELECT column1, column2, ...
FROM table1
RIGHT JOIN table2
ON table1.column = table2.column;
```

Donde **“table1 y table2”** son los nombres de las tablas que se quieren unir, y "**column1 y column2"** son las columnas que se desean incluir en el resultado de la consulta. La cláusula "**ON**" se utiliza para especificar la condición de unión, que es una expresión que define cómo deben relacionarse los registros de las dos tablas.

El resultado de una cláusula “RIGHT JOIN” es una tabla que contiene todos los registros de la tabla a la derecha y aquellos registros de la tabla a la izquierda que cumplan la condición de unión. Si no hay correspondencias para un registro en la tabla a la izquierda, las columnas correspondientes en la tabla resultante contendrán valores nulos.

Es importante destacar que la cláusula “RIGHT JOIN” es independiente de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos.

OPERADORES LÓGICOS

AND/OR

Son operadores lógicos en SQL que se utilizan para combinar varias condiciones en una cláusula WHERE. Estos operadores permiten definir múltiples condiciones que deben cumplirse para que un registro sea incluido en el resultado de una consulta SELECT.

La sintaxis básica de los operadores son:

```sql
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2;

SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2;
```

Donde “**condition1** y **condition2**” son expresiones que definen las condiciones que deben cumplirse para incluir un registro en el resultado de la consulta.

El operador "**AND**" es utilizado para combinar dos o más condiciones que deben cumplirse simultáneamente para incluir un registro en el resultado de la consulta. Por ejemplo, se puede utilizar el operador AND para seleccionar solo los registros que tienen un valor determinado en una columna y que se encuentran en un rango determinado de fechas.

El operador "**OR**" es utilizado para combinar dos o más condiciones en las que al menos una de ellas debe cumplirse para incluir un registro en el resultado de la consulta. Por ejemplo, se puede utilizar el operador OR para seleccionar los registros que tienen un valor determinado en una columna o que se encuentran en un rango determinado de fechas.

Es importante destacar que los operadores AND y OR son independientes de la base de datos utilizada y de las configuraciones regionales. Por lo tanto, su sintaxis y funcionamiento son similares en la mayoría de las bases de datos.

DEFINICIÓN DE DATOS

ALTER

Es una sentencia en SQL que se utiliza para modificar la estructura de una tabla existente. Con la sentencia **ALTER**, se pueden realizar diferentes acciones, como agregar o eliminar columnas, modificar la estructura de una columna, agregar o eliminar índices, entre otras acciones.

La sintaxis básica de la sentencia es la siguiente:

```
ALTER TABLE table_name
[ALTER COLUMN column_name column_definition |
ADD COLUMN column_name column_definition |
DROP COLUMN column_name |
ADD INDEX index_name (column_list) |
DROP INDEX index_name];
```

Donde **“table_name”** es el nombre de la tabla que se desea modificar, “**column_name**” es el nombre de la columna que se desea modificar, “**column_definition**” es la nueva definición de la columna, “**index_name**” es el nombre del índice que se desea agregar o eliminar y “**column_list**” es la lista de columnas que se desea incluir en el índice.

Es importante tener en cuenta que la sentencia “ALTER” es dependiente de la base de datos utilizada y que la sintaxis puede variar ligeramente de una base de datos a otra. Sin embargo, en general, la funcionalidad de la sentencia es la misma en la mayoría de las bases de datos.

CREATE

Es una sentencia en SQL que se utiliza para crear una tabla, una base de datos, un índice, una vista u otros objetos en una base de datos. La sintaxis para crear una tabla con la sentencia CREATE es la siguiente:

```
CREATE TABLE table_name (
    column1 data_type,
    column2 data_type,
    column3 data_type,
    ....
```

Donde “**table_name**” es el nombre de la tabla que se desea crear, “column1, column2, column3”, etc. son los nombres de las columnas en la tabla y “**data_type**” es el tipo de datos que se almacenará en cada columna.

Por ejemplo, si deseas crear una tabla llamada "customers" con las columnas "**id**", "**name**" y "**email**", la sentencia sería la siguiente:

```
CREATE TABLE customers (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(100)
```

Además de la creación de tablas, también se pueden crear otros objetos en una base de datos, como por ejemplo, una base de datos nueva:

```
CREATE DATABASE database_name;
```

Donde “**database_name**” es el nombre de la base de datos que se desea crear.

DROP

Es una sentencia en SQL que se utiliza para eliminar una tabla, una columna, un índice, una base de datos o cualquier otro objeto de la base de datos. La sintaxis para eliminar una tabla con la sentencia "**DROP"** es la siguiente:

```
DROP TABLE table_name;
```

Donde “**table_name**” es el nombre de la tabla que se desea eliminar.

Por ejemplo, si deseas eliminar una tabla llamada "**customers**", la sentencia sería la siguiente:

```
DROP TABLE customers;
```

La sentencia “DROP” es una operación irreversible, por lo que debes tener cuidado al utilizarla, ya que una vez que se ha eliminado un objeto de la base de datos, no se puede recuperar.

Además de la eliminación de tablas, también se pueden eliminar otras cosas con la sentencia DROP, como por ejemplo, una columna en una tabla:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Donde "**table_name**" es el nombre de la tabla de la que se desea eliminar la columna y "**column_name**" es el nombre de la columna que se desea eliminar.

MANIPULACIÓN DE DATOS

DELETE

Es una sentencia en SQL que se utiliza para eliminar registros existentes en una tabla. La sintaxis para eliminar datos de una tabla con la sentencia "**DELETE**" es la siguiente:

```
DELETE FROM table_name WHERE condition;
```

Donde "**table_name**" es el nombre de la tabla de la que se desea eliminar los registros y condition es una cláusula que especifica cuáles son los registros que deben ser eliminados.

Por ejemplo, si deseas eliminar todos los registros de la tabla "**customers**" cuyo "**id**" sea igual a "**1**", la sentencia sería la siguiente:

```
DELETE FROM customers WHERE id = 1;
```

Es importante tener precaución al utilizar la sentencia DELETE, ya que los datos eliminados no pueden ser recuperados una vez que se han eliminado. Es por eso que es recomendable hacer una copia de seguridad de los datos antes de utilizar la sentencia.

INSERT

Es una sentencia en SQL que se utiliza para insertar nuevos registros en una tabla existente. La sintaxis para insertar datos en una tabla con la sentencia "**INSERT**" es la siguiente:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Donde "**table_name**" es el nombre de la tabla en la que se desea insertar los datos, **"column1, column2, column3"**, etc. son los nombres de las columnas en la tabla y **"value1, value2, value3"**, etc. son los valores que se desean insertar en cada columna.

Por ejemplo, si deseas insertar un nuevo registro en la tabla "**customers**" con los valores "**1**", **"John Doe"** y "johndoe@email.com", la sentencia sería la siguiente:

```
INSERT INTO customers (id, name, email)
VALUES (1, 'John Doe', 'johndoe@email.com');
```

También es posible insertar datos en una tabla sin especificar las columnas, en cuyo caso se asume que los valores se insertan en el orden en que las columnas están definidas en la tabla:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1_1, value2_1, value3_1, ...),
       (value1_2, value2_2, value3_2, ...),
       ...;
```

UPDATE

Es una instrucción en SQL que se utiliza para actualizar los datos existentes en una o varias filas de una tabla. La sintaxis general de la instrucción "**UPDATE**" es la siguiente:

```
UPDATE nombre_de_tabla
SET columna1 = valor1, columna2 = valor2, ..., columnaN = valorN
WHERE condición;
```

Donde: "n**ombre_de_tabla**" es el nombre de la tabla que se desea actualizar.

- **"Columna1, columna2, ..., columnaN"** son las columnas que se desean actualizar en la tabla.
- **"Valor1, valor2, ..., valorN"** son los nuevos valores que se desean asignar a cada columna.

"**WHERE**" es una cláusula opcional que se utiliza para especificar la condición que deben cumplir las filas que se van a actualizar. Si no se especifica ninguna condición, se actualizarán todas las filas de la tabla.

Por ejemplo, la siguiente consulta SQL muestra cómo utilizar la instrucción UPDATE para cambiar el valor de la columna edad a 30 para el cliente con "**id igual a 1**" en la tabla clientes:

```
UPDATE clientes
SET edad = 30
WHERE id = 1;
```

En esta consulta, la instrucción UPDATE se utiliza para actualizar la columna edad en la tabla clientes y establecer su valor en 30. La condición WHERE se utiliza para especificar que sólo se debe actualizar la fila con id igual a 1.

La instrucción UPDATE es útil cuando se desea actualizar información en una tabla existente, por ejemplo, para corregir errores, actualizar información desactualizada o cambiar valores por otros más relevantes. Es importante tener cuidado al utilizar esta instrucción, ya que puede tener un impacto significativo en la integridad y consistencia de los datos almacenados en la tabla.

OTRAS FUNCIONES

COALESCE

Es una función en SQL que se utiliza para devolver el primer valor no nulo de una lista de expresiones. La sintaxis general de la función **COALESCE** es la siguiente:

```
COALESCE(expresión1, expresión2, ..., expresiónN)
```

Donde: **"expresión1, expresión2, ..., expresiónN"** son las expresiones que se desean evaluar.

La función evalúa cada expresión en orden de izquierda a derecha y devuelve la primera expresión que no sea nula. Si todas las expresiones son nulas, se devuelve un valor nulo.

Por ejemplo, la siguiente consulta SQL muestra cómo utilizar la función COALESCE para devolver el primer valor no nulo de una lista de expresiones en la tabla clientes:

```
SELECT COALESCE(nombre_completo, nombre, apellido) AS nombre
FROM clientes;
```

En esta consulta, la función COALESCE se utiliza para devolver el primer valor no nulo de la columna "**nombre_completo**", la columna nombre y la columna apellido para cada fila de la tabla clientes. Si la columna nombre_completo es nula, la función devolverá el valor de la columna nombre. Si la columna nombre también es nula, la función COALESCE devolverá el valor de la columna apellido. Si todas las columnas son nulas, la función devolverá un valor nulo.

Esta función, es útil cuando se desea seleccionar el primer valor no nulo de una lista de expresiones. Por ejemplo, si una tabla tiene varias columnas que pueden contener información relevante pero no todas las filas tienen valores para todas las columnas, se puede utilizar la función COALESCE para seleccionar el primer valor no nulo de entre varias columnas en función de su orden de prioridad.

IFNULL

Es una función en SQL que toma dos argumentos. Devuelve el primer argumento si no es "**NULL**", de lo contrario devuelve el segundo argumento.

Sintaxis:

```
IFNULL(expresión, valor_alternativo)
```

Aquí, la expresión es cualquier expresión válida en SQL, y "**valor_alternativo**" es el valor que la función devuelve si la expresión se evalúa como NULL.

Por ejemplo, la siguiente consulta devuelve el nombre del departamento y la ciudad en la que se encuentra. Si la ciudad es NULL, la consulta devuelve la cadena "Desconocido".

```
SELECT nombre_departamento, IFNULL(ciudad, 'Desconocido') as ciudad
FROM departamentos;
```

Si estás interesado en aprender sobre SQL y sus sentencias clave, esperamos que este listado te ayude. SQL es un lenguaje de programación muy utilizado en el mundo de la ciencia de datos y el análisis de datos, por lo que es importante tener un buen conocimiento de sus sentencias básicas.

En SQL, las sentencias clave son las que nos permiten realizar las operaciones más comunes en una base de datos, como la selección, la actualización y la eliminación de datos.

Si estás interesado en aprender más sobre SQL y sus sentencias clave, Pontia, escuela de formación online especializada en máster en Data Science y Data Analytics, puede proporcionarte el conocimiento y las herramientas necesarias para dominar este lenguaje de programación. Con nuestros programas de formación en línea, podrás aprender a escribir consultas SQL efectivas y a trabajar con bases de datos de manera eficiente.

¡No esperes más para ampliar tus habilidades en SQL! Pontia está aquí para ayudarte a convertirte en un experto en Data Science y Data Analytics.

¿TE GUSTARÍA DOMINAR MEJOR SQL?

MÁSTER EN DATA ANALYTICS

Aprenderás Matemáticas y Estadística, Python, SQL, Machine learning y Deep learning, Visualización de datos con Tableau y Big Data para convertirte un perfil clave en cualquier empresa tecnológica.

MÁSTER EN DATA SCIENCE

Te formarás en Matemáticas y Estadística, Python, SQL, Excel, Power Bi, Visualización de datos y Big Data para convertirte un perfil clave en la toma de decisiones de cualquier empresa tecnológica.