@its_anaehm

Guía No. 5

Libro: Learning SQL, Master SQL Fundamentals - Capítulos 4, 5 y 8

Temas: Filtrado, Canslulas de varias tablas, Agrupaciones y agregados

Preguntas

1.- ¿Cómo consultar datos en una base de datos?

Para realizar consultas sobre las tablas de las bases de datos disponemos de la instrucción **SELECT**. Con ella podemos consultar una o varias tablas. Es sin duda el comando más versátil del lenguaje SQL.

Existen muchas cláusulas asociadas a la sentencia SELECT (GROUP BY, ORDER, HAVING, UNION). También es una de las instrucciones en la que con más frecuencia los motores de bases de datos incorporan cláusulas adicionales al estándar.

El resultado de una consulta SELECT nos devuelve una tabla lógica. Es decir, los resultados son una relación de datos, que tiene filas/registros, con una serie de campos/columnas. Igual que cualquier tabla de la base de datos. Sin embargo esta tabla está en memoria mientras la utilicemos, y luego se descarta. Cada vez que ejecutamos la consulta se vuelve a calcular el resultado.

SELECT Permite seleccionar las columnas que se van a mostrar y en el orden en que lo van a hacer. Simplemente es la instrucción que la base de datos interpreta como que vamos a solicitar información.

ALL / DISTINCT *ALL* es el valor predeterminado, especifica que el conjunto de resultados puede incluir filas duplicadas. Por regla general nunca se utiliza.

DISTINCT especifica que el conjunto de resultados sólo puede incluir filas únicas. Es decir, si al realizar una consulta hay registros exactamente iguales que aparecen más de una vez, éstos se eliminan. Muy útil en muchas ocasiones.

Nombres de campos Se debe especificar una lista de nombres de campos de la tabla que nos interesan y que por tanto queremos devolver. Normalmente habrá más de uno, en cuyo caso separamos cada nombre de los demás mediante comas.

Se puede anteponer el nombre de la tabla al nombre de las columnas, utilizando el formato Tabla.Columna. Además de nombres de columnas, en esta lista se pueden poner constantes, expresiones aritméticas, y funciones, para obtener campos calculados de manera dinámica.

Si queremos que nos devuelva todos los campos de la tabla utilizamos el comodín "*" (asterisco).

Los nombres indicados deben coincidir exactamente con los nombre de los campos de la tabla, pero si queremos que en nuestra tabla lógica de resultados tengan un nombre diferente podemos utilizar:

AS Permite renombrar columnas si lo utilizamos en la cláusula SELECT, o renombrar tablas si lo utilizamos en la cláusula FROM. Es opcional. Con ello podremos crear diversos alias de columnas y tablas. Enseguida veremos un ejemplo.

FROM Esta cláusula permite indicar las tablas o vistas de las cuales vamos a obtener la información. De momento veremos ejemplos para obtener información de una sola tabla.

Como se ha indicado anteriormente, también se pueden renombrar las tablas usando la instrucción "AS".

WHERE Especifica la condición de filtro de las filas devueltas. Se utiliza cuando no se desea que se devuelvan todas las filas de una tabla, sino sólo las que cumplen ciertas condiciones. Lo habitual es utilizar esta cláusula en la mayoría de las consultas.

Condiciones Son expresiones lógicas a comprobar para la condición de filtro, que tras su resolución devuelven para cada fila TRUE o FALSE, en función de que se cumplan o no. Se puede utilizar cualquier expresión lógica y en ella utilizar diversos operadores como:

Símbolo	Significado
>	Mayor
>=	Mayor o igual
<	Menor
<=	Menor o igual
=	Igual
<> o !=	Distinto

IS [NOT] NULL (para comprobar si el valor de una columna es o no es nula, es decir, si contiene o no contiene algún valor) Se dice que una columna de una fila es NULL si está completamente vacía. Hay que tener en cuenta que si se ha introducido cualquier dato, incluso en un campo alfanumérico si se introduce una cadena en blanco o un cero en un campo numérico, deja de ser NULL.

LIKE: Para la comparación de un modelo. Para ello utiliza los caracteres comodín especiales: "%" y "_". Con el primero indicamos que en su lugar puede ir cualquier cadena de caracteres, y con el segundo que puede ir cualquier carácter individual (un solo carácter). Con la combinación de estos caracteres podremos obtener múltiples patrones de búsqueda. Por ejemplo:

- El nombre empieza por A: Nombre LIKE 'A%'
- El nombre acaba por A: Nombre LIKE '%A'
- El nombre contiene la letra A: Nombre LIKE '%A%'
- El nombre empieza por A y después contiene un solo carácter cualquiera: Nombre LIKE 'A_'
- El nombre empieza una A, después cualquier carácter, luego una E y al final cualquier cadena de caracteres: Nombre LIKE 'A_E%'

BETWEEN: para un intervalo de valores. Por ejemplo: Clientes entre el 30 y el 100: CodCliente BETWEEN 30 AND 100 Clientes nacidos entre 1970 y 1979: FechaNac BETWEEN '19700101' AND '19791231' IN( ): para especificar una relación de valores concretos. Por ejemplo: Ventas de los Clientes 10, 15, 30 y 75: CodCliente IN(10, 15, 30, 75)

Por supuesto es posible combinar varias condiciones simples de los operadores anteriores utilizando los operadores lógicos OR, AND y NOT, así como el uso de paréntesis para controlar la prioridad de los

operadores (como en matemáticas). Por ejemplo: … (Cliente = 100 AND Provincia = 30) OR Ventas > 1000 … que sería para los clientes de las provincias 100 y 30 o cualquier cliente cuyas ventas superen 1000.

ORDER BY Define el orden de las filas del conjunto de resultados. Se especifica el campo o campos (separados por comas) por los cuales queremos ordenar los resultados.

ASC / DESC ASC es el valor predeterminado, especifica que la columna indicad en la cláusula ORDER BY se ordenará de forma ascendente, o sea, de menor a mayor. Si por el contrario se especifica DESC se ordenará de forma descendente (de mayor a menor).

2.- ¿Cómo agrupar registros?

La cláusula GROUP BY permite organizar las filas de una consulta en grupos. Los grupos están determinadospor las columnas que se especifican en la cláusula GROUP BY.

A continuación, se ilustra la sintaxis de la cláusula GROUP BY:

```
SELECT select_list FROM table_name
GROUP BY    column_name1,   column_name2 ,...;
```

En esta consulta, la cláusula GROUP BY regresa un grupo para cada combinación de los valores en las columnas enumeradas en la cláusula GROUP BY.

3.- ¿Cómo agrupar registros para sumarizar?

Sum

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

```
Sum(expr)
```

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

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

4.- ¿Cómo agrupar registros para contabilizar?

Count

Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente:

```
Count(expr)
```

En donde expr contiene el nombre del campo que desea contar. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). Puede contar cualquier tipo de datos incluso texto.

Aunque expr puede realizar un cálculo sobre un campo, Count simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función Count no cuenta los registros que tienen campos null a menos que expr sea el carácter comodín asterisco (*). *Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos null. Count(*)* es considerablemente más rápida que Count(Campo). No se debe poner el asterisco entre dobles comillas ('*').

```
SELECT Count(*) AS Total FROM Pedidos;
```

5.- ¿Cómo agrupar registros para promediar?

AVG

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta. Su sintaxis es la siguiente

```
Avg(expr)
```

En donde expr representa el campo que contiene los datos numéricos para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por Avg es la media aritmética (la suma de los valores dividido por el número de valores). La función Avg no incluye ningún campo Null en el cálculo.

```
SELECT Avg(Gastos) AS Promedio FROM Pedidos WHERE Gastos > 100;
```

6.- ¿Cómo trabajar con campos numéricos?

Con los datos numéricos la diferencia entre uno y otro tipo de dato es simplemente el rango de valores que puede contener.

Dentro de los datos numéricos, podemos distinguir dos grandes ramas: enteros y decimales.

NUMÉRICOS ENTEROS

Comencemos por conocer las opciones que tenemos para almacenar datos que sean númericos enteros (edades, cantidades, magnitudes sin decimales); poseemos una variedad de opciones:

Tipo de Dato	Bytes	Valor mínimo	Valor máximo

Tipo de Dato	Bytes	Valor mínimo	Valor máximo
TINYINT	1	-128	127
SMALLINT	2	-32768	32767
MEDIUMINT	3	-8388608	8388607
INT	4	-2147483648	2147483647
BIGINT	8	-9223372036854775808	9223372036854775807

Sin Signo:

Tipo de Dato	Bytes	Valor mínimo	Valor máximo
TINYINT	1	0	255
SMALLINT	2	0	65535
MEDIUMINT	3	0	16777215
INT	4	0	4294967295
BIGINT	8	0	18446744073709551615

NÚMEROS FLOTANTES

Estos tipos de datos son necesarios para almacenar precios, salarios, importes de cuentas bancarias, etc. que no son enteros.

Tenemos que tener en cuenta que si bien estos tipos de datos se llaman "de coma flotante", por ser la coma el separador entre la parte entera y la parte decimal, en realidad MySQL los almacena usando un punto como separador.

En esta categoría, disponemos de tres tipos de datos: FLOAT, DOUBLE y DECIMAL.

7.- ¿Cómo consultar información de múltiples tablas relacionadas?

Los JOINs en SQL sirven para combinar filas de dos o más tablas basándose en un campo común entre ellas, devolviendo por tanto datos de diferentes tablas. Un JOIN se produce cuando dos o más tablas se juntan en una sentencia SQL.

Existen más tipos de joins en SQL que los que aquí se explican, como CROSS JOIN, O SELF JOIN, pero no todos ellos están soportados por todos los sistemas de bases de datos. Los más importantes son los siguientes:

1. INNER JOIN: Devuelve todas las filas cuando hay al menos una coincidencia en ambas tablas.
2. LEFT JOIN: Devuelve todas las filas de la tabla de la izquierda, y las filas coincidentes de la tabla de la derecha.
3. RIGHT JOIN: Devuelve todas las filas de la tabla de la derecha, y las filas coincidentes de la tabla de la izquierda.

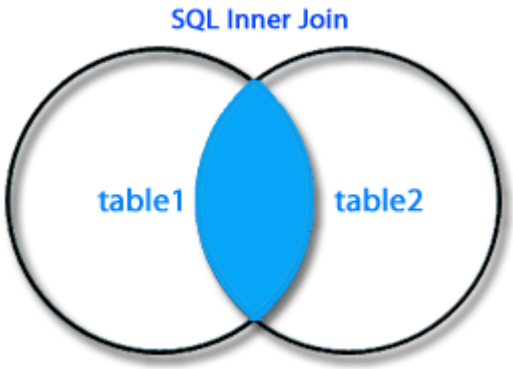
4. OUTER JOIN: Devuelve todas las filas de las dos tablas, la izquierda y la derecha. También se llama FULL OUTER JOIN.

5. INNER JOIN:

INNER JOIN selecciona todas las filas de las dos columnas siempre y cuando haya una coincidencia entre las columnas en ambas tablas. Es el tipo de JOIN más común.

```
SELECT nombreColumna(s)
FROM tabla1
INNER JOIN tabla2
ON tabla1.nombreColumna=table2.nombreColumna;
```

Se ve más claro utilizando una imagen:

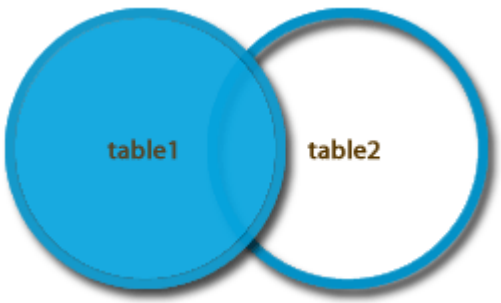


2. LEFT JOIN

LEFT JOIN mantiene todas las filas de la tabla izquierda (la tabla1). Las filas de la tabla derecha se mostrarán si hay una coincidencia con las de la izquierda. Si existen valores en la tabla izquierda pero no en la tabla derecha, ésta mostrará null.

```
SELECT nombreColumna(s)
FROM tabla1
LEFT JOIN tabla2
ON tabla1.nombreColumna=tabla2.nombreColumna;
```

La representación de LEFT JOIN en una imagen es:

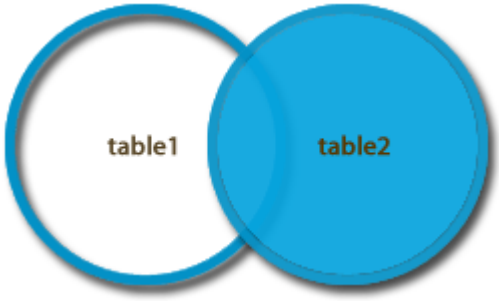


3. RIGHT JOIN

Es igual que LEFT JOIN pero al revés. Ahora se mantienen todas las filas de la tabla derecha (tabla2). Las filas de la tabla izquierda se mostrarán si hay una coincidencia con las de la derecha. Si existen valores en la tabla derecha pero no en la tabla izquierda, ésta se mostrará null.

```
SELECT nombreColumna(s)
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.nombreColumna=tabla2.nombreColumna;
```

La imagen que representa a RIGHT JOIN es:

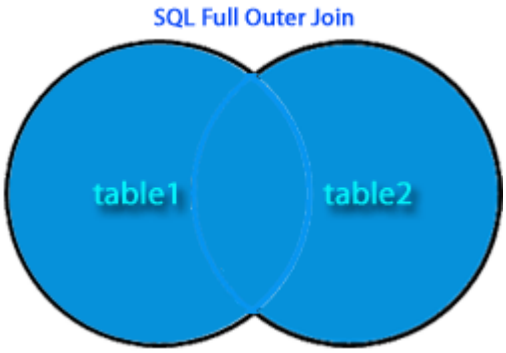


4. OUTER JOIN

OUTER JOIN o FULL OUTER JOIN devuelve todas las filas de la tabla izquierda (tabla1) y de la tabla derecha (tabla2). Combina el resultado de los joins LEFT y RIGHT. Aparecerá null en cada una de las tablas alternativamente cuando no haya una coincidencia.

```
SELECT nombreColumna(s)
FROM tabla1
OUTER JOIN tabla2
ON tabla1.nombreColumna=tabla2.nombreColumna;
```

La imagen que representa el OUTER JOIN es la siguiente:



8.- ¿Cómo usar el filtrado de información para que la data que se extrae cumple con ciertas condiciones usando la cláusula WHERE?

Usando paréntesis

Si su cláusula where incluye tres o más condiciones usando los operadores and y o debe usar paréntesis para dejar clara su intención, tanto para el servidor de la base de datos como para cualquier otra persona que lea su código.

```
WHERE end_date IS NULL
    AND (title = 'Teller' OR start_date < '2007-01-01')
```

Usando el operador NOT Si bien es fácil de manejar para el servidor de la base de datos,normal mente es difícil para una persona evaluar una cláusula where que incluya el operador not, razón por la cual no se encontrará con ella con mucha frecuencia.

```
WHERE end_date IS NULL
    AND NOT (title = 'Teller' OR start_date < '2007-01-01')
```