Consultas SQL sobre una tabla

Sintaxis de la sentencia SELECT

Comentarios en SQL

- Para escribir comentarios en nuestras sentencias SQL podemos hacerlo de diferentes formas:
  - Precediendo – en la línea
  - Precediendo # en la línea
  - Cuando comentamos un bloque de líneas utilizamos /* para abrir comentario y */` para cerrarlo.

Setencia Select

- Para empezar con consultas sencillas podemos simplificar la definición anterior y quedarnos con la siguiente:

```
SELECT [DISTINCT] select_expr [, select_expr ...]
[FROM table_references]
[WHERE where_condition]
[GROUP BY {col_name | expr | position} [ASC | DESC], ... [WITH ROLLUP]]
[HAVING where_condition]
[ORDER BY {col_name | expr | position} [ASC | DESC], ...]
[LIMIT {[offset,] row_COUNT | row_COUNT OFFSET offset}]
```

El **orden en que se ejecuta cada una de las cláusulas** es el siguiente:

- Cláusula FROM.
- Cláusula WHERE (Es opcional, puede ser que no aparezca).
- Cláusula GROUP BY (Es opcional, puede ser que no aparezca).
- Cláusula HAVING (Es opcional, puede ser que no aparezca).
- Cláusula SELECT.
- Cláusula ORDER BY (Es opcional, puede ser que no aparezca).
- Cláusula LIMIT (Es opcional, puede ser que no aparezca).

Clausula SELECT

- Nos permite indicar cuáles serán las columnas que tendrá la tabla de resultados de la consulta que estamos realizando. Las opciones que podemos indicar son las siguientes:
  - El **nombre de una columna** de la tabla sobre la que estamos realizando la consulta. Será una columna de la tabla que aparece en la cláusula FROM.

    Para indicar todas las columnas utilizamos *
  - Una **constante** que aparecerá en todas las filas de la tabla resultado.
  - Una **expresión** que nos permite calcular nuevos valores.
- Hay que tener en cuenta que el resultado de una consulta siempre será una tabla de datos, que puede tener una o varias columnas y ninguna, una o varias filas.

El nombre de una columna

- Especificamos el nombre de la columna de la que queremos obtener la información, si queremos mas de una especificaremos las columnas separadas por comas.
- No es case sensitive.
- El resultado de la consulta SQL mostrará las columnas que haya solicitado, siguiendo el orden en el que se hayan indicado.

Columnas Calculadas

- Es posible realizar cálculos aritméticos entre columnas, o calcular sobre una columna para hayar nuevos valores.
- Por ejemplo, si queremos calcular el precio de un producto aplicándole el IVA pondremos:

```
SELECT nombre, precio - (precio * 0.21)  FROM producto;
```

- Por ejemplo, si hubiese una columna en productos con el beneficio que sacamos por cada producto la sentencia sería:

```
SELECT nombre, precio + beneficio FROM producto;
```

Nota: Fijaros en el nombre de las columnas de los campos calculados.

ALIAS

- Con la palabra reservada **AS** podemos crear alias para las columnas.
- Esto puede ser útil cuando estamos calculando nuevas columnas a partir de valores de las columnas actuales
- Si el nuevo nombre que estamos creando para el alias contiene espacios en blanco es necesario usar comillas simples.
- Repetimos las consultas anteriores utilizando alias

```
SELECT nombre, precio - (precio * 0.21) as "Precio Iva"  FROM producto;
```

- Para campos calculados los alias no se pueden utilizar en la cláusula Select.

Funciones de MySQL en la cláusula SELECT

- Es posible hacer uso de funciones específicas de MySQL en la cláusula SELECT.
- Existen varias funciones de:
  - Cadena
  - Numéricas
  - Fecha

Funciones de Cadena

Función	Descripción
CONCAT	Concatena cadenas
CONCAT_WS	Concatena cadenas con un separador
LOWER	Devuelve una cadena en minúscula
UPPER	Devuelve una cadena en mayúscula
SUBSTR	Devuelve una subcadena

CONCAT () suma dos o más expresiones juntas. Dentro de los paréntesis se introducen las cadenas a concatenar separadas por comas.
CONCAT_WS(*separator*, *expression1*, *expression2*, *expression3*,...); separator es la cadena con la que se van a unir las demas expresiones.
LOWER(expresión)
UPPER(expresion) equivalente a UCASE(expresión).
SUBSTR(expresión, poscomienzo, numerocaracters).

Ejemplo de funciones de cadena

- **SELECT** CONCAT(nombre, apellido1, apellido2) **AS** nombre_completo **FROM** alumno;
- **SELECT** CONCAT_WS(' ', nombre, apellido1, apellido2) **AS** nombre_completo **FROM** alumno;

Funciones matemáticas

Función	Descripción
`ABS()`	Devuelve el valor absoluto
`POW(x,y)`	Devuelve el valor de x elevado a y
`SQRT()`	Devuelve la raíz cuadrada
`PI()`	Devuelve el valor del número `PI`
`ROUND()`	Redondea un valor numérico
`TRUNCATE()`	Trunca un valor numérico

Modificadores ALL, DISTINCT y DISTINCTROW

- Los modificadores ALL y DISTINCT indican si se deben incluir o no filas repetidas en el resultado de la consulta.
- **ALL** indica que se deben incluir todas las filas, incluidas las repetidas. Es la opción por defecto, por lo tanto no es necesario indicarla.
- **DISTINCT** elimina las filas repetidas en el resultado de la consulta.
- **DISTINCTROW** es un sinónimo de DISTINCT.

Cláusula ORDER BY

- ORDER BY permite ordenar las filas que se incluyen en el resultado de la consulta. La sintaxis de MySQL es la siguiente:

```
[ORDER BY {col_name | expr | position} [ASC | DESC], ...]
```

Esta cláusula nos permite ordenar el resultado de forma ascendente ASC o descendente DESC, además de permitirnos ordenar por varias columnas estableciendo diferentes niveles de ordenación.

Ejemplos

- Obtener el nombre y los apellidos de todos los alumnos, ordenados por su primer apellido de forma ascendente.

```
SELECT apellido1, apellido2, nombre
FROM alumno
ORDER BY apellido1;
```

```
SELECT apellido1, apellido2, nombre
FROM alumno
ORDER BY apellido1 ASC;
```

nombre	apellido1	apellido2
Carretero	Ortega	Antonio
Domínguez	Hernández	Manuel
Fernández	Ramírez	Cristina
Gutiérrez	Sánchez	Irene
Martínez	López	Paco
Moreno	Ruiz	Daniel
Ramírez	Gea	Pepe
Sáez	Vega	Juan
Sánchez	Pérez	María
Sánchez	Ortega	Lucía

Ejemplos

- Obtener el nombre y los apellidos de todos los alumnos, ordenados por su primer apellido y segundo apellido de forma ascendente.

```
SELECT apellido1, apellido2, nombre
FROM alumno
ORDER BY apellido1, apellido2, nombre;
```

```
SELECT apellido1, apellido2, nombre
FROM alumno
ORDER BY 1, 2, 3;
```

Cláusula LIMIT

- LIMIT permite limitar el número de filas que se incluyen en el resultado de la consulta. La sintaxis de MySQL es la siguiente:

```
[LIMIT {[offset,] row_COUNT | row_COUNT OFFSET offset}]
```

Donde **row_COUNT** es el número de filas que queremos obtener y **offset** el número de filas que nos saltamos antes de empezar a contar. Es decir, la primera fila que se obtiene como resultado es la que está situada en la posición offset + 1.

Ejemplo LIMIT

- Suponga que queremos mostrar en una página web las filas que están almacenadas en la tabla resultados, y queremos mostrar la información en diferentes páginas, donde cada una de las páginas muestra solamente 5 resultados. ¿Qué consulta SQL necesitamos realizar para incluir los primeros 5 resultados de la primera página?

  SELECT * FROM resultado LIMIT 5

- ¿Qué consulta necesitaríamos para mostrar resultados de la segunda página?

  SELECT * FROM resultado LIMIT 5, 5

- ¿Y los resultados de la tercera página?

  SELECT * FROM resultado LIMIT 10, 5

Ejercicios

- Obtener toda la información que contiene la tabla Productos.
- Obtener el nombre de todos los fabricantes de los que disponemos. Fabricante(nombre)
- Obtener el nombre y el precio de todos los productos.
- Añadir a la tabla productos la columna beneficios tipo decimal, ella contendrá el porcentaje de beneficios que obtendremos por cada producto. Dicha columna tomará por defecto 10% (0,1).
- Calcula el precio de venta del producto (precio + beneficio)
- Calcula el precio final de cada uno de los productos teniendo en cuenta que será precio + beneficio y se le deberá aplicar un 21% de IVA.
- Realiza las dos consultas anteriores utilizando alias.
- Calcula el precio final de cada uno de los productos sin decimales.
- Lista todos códigos de departamento que tengan empleados.
- Crea la tabla clientes (Fichero clientes.sql):

CLAUSULA WHERE

- La cláusula WHERE nos permite añadir filtros a nuestras consultas para seleccionar sólo aquellas filas que cumplen una determinada condición.
- Estas condiciones se denominan predicados y el resultado de estas condiciones puede ser verdadero, falso o desconocido.
- Una condición tendrá un resultado desconocido cuando alguno de los valores utilizados tiene el valor NULL.
- Los operandos usados en las condiciones pueden ser nombres de columnas, constantes o expresiones.
- Los operadores que podemos usar en las condiciones pueden ser aritméticos, de comparación, lógicos, etc.

TIPOS DE CONDICIONES

- •Condiciones para comparar valores o expresiones.
- Condiciones para comprobar si un valor está dentro de un rango de valores.
- Condiciones para comprobar si un valor está dentro de un conjunto de valores.
- Condiciones para comparar cadenas con patrones.
- Condiciones para comprobar si una columna tiene valores a NULL.

OPERADORES

Operadores aritméticos

Operador	Descripción
+	Suma
-	Resta
	Multiplicación
/	División
%	Módulo

Operador	Descripción
<	Menor que
<=	Menor o igual
>	Mayor que
>=	Mayor o igual
<>	Distinto
!=	Distinto
=	Igual que

Operadores lógicos

Operador	Descripción
AND	Y lógica
&&	Y lógica
OR	O lógica
||	O lógica
NOT	Negación lógica
!	Negación lógica

Ejemplos

1. Obtener el nombre de todos los alumnos que su primer apellido sea Martínez.
2. Obtener todos los datos del alumno que tiene un id igual a 9.
3. Obtener el nombre y la fecha de nacimiento de todos los alumnos nacieron después del 1 de enero de 1997.
4. Devuelve los datos del alumno cuyo id es igual a 1.
5. Devuelve los datos del alumno cuyo teléfono es igual a 692735409
6. Devuelve un listado de todos los alumnos que son repetidores.
7. Devuelve un listado de todos los alumnos que no son repetidores.
8. Devuelve el listado de los alumnos que han nacido antes del 1 de enero de 1993.
9. Devuelve el listado de los alumnos que han nacido después del 1 de enero de 1994.
10. Devuelve el listado de los alumnos que han nacido después del 1 de enero de 1994 y no son repetidores.
11. Devuelve el listado de todos los alumnos que nacieron en 1998.
12. Devuelve el listado de todos los alumnos que no nacieron en 1998.

Operador BETWEEN

```
expresión [NOT] BETWEEN cota_inferior AND cota_superior
```

- Se utiliza para comprobar si un valor está dentro de un rango de valores.

13. Devuelve los datos de los alumnos que hayan nacido entre el 1 de enero de 1998 y el 31 de mayo de 1998.

Operador IN

- Este operador nos permite comprobar si el valor de una determinada columna está incluido en una lista de valores.

14. Obtener todos los datos de los alumnos que tengan como primer apellido Sánchez, Martínez o Domínguez.
15. Obtener todos los datos de los alumnos que no tengan como primer apellido Sánchez, Martínez o Domínguez

Operador LIKE

- Se utiliza para comparar si una cadena de caracteres coincide con un patrón.
- En el patrón podemos utilizar cualquier carácter alfanumérico, pero hay dos caracteres que tienen un significado especial,
  - %: Este carácter equivale a cualquier conjunto de caracteres.
  - _: Este carácter equivale a cualquier carácter.
  - \: Se utiliza para escapar los caracteres anteriores. Si queremos utilizar otro carácter para escarpar deberáimos indicarlo al lado del patrón con la palabra escape

```
columna [NOT] LIKE patrón
```

Ejemplos de patrones

16. Devuelva un listado de todos los alumnos que su primer apellido empiece por la letra S.
17. Devuelva un listado de todos los alumnos que su primer apellido termine por la letra z.
18. Devuelva un listado de todos los alumnos que tengan un nombre de cuatro caracteres.
19. Devuelve un listado de todos los productos cuyo nombre empieza con estas cuatro letras «A%BC».

Operadores IS e IS NOT

- Estos operadores nos permiten comprobar si el valor de una determinada columna es NULL o no lo es.

20. Obtener la lista de alumnos que tienen un valor NULL en la columna teléfono.
21. Obtener la lista de alumnos que no tienen un valor NULL en la columna teléfono.

FUNCIONES DE AGRUPAMIENTO O AGREGACIÓN

- Estas funciones realizan una operación específica sobre todas las filas de un grupo.
- Ejemplo. Select precio FROM producto----------Select MAX(precio) FROM producto

precio
96.99
130
160.99
195
765
212
255.99
569
454
69.99000000000001
190
40
310

MAX(precio)
765

Las funciones de agregación más comunes son:

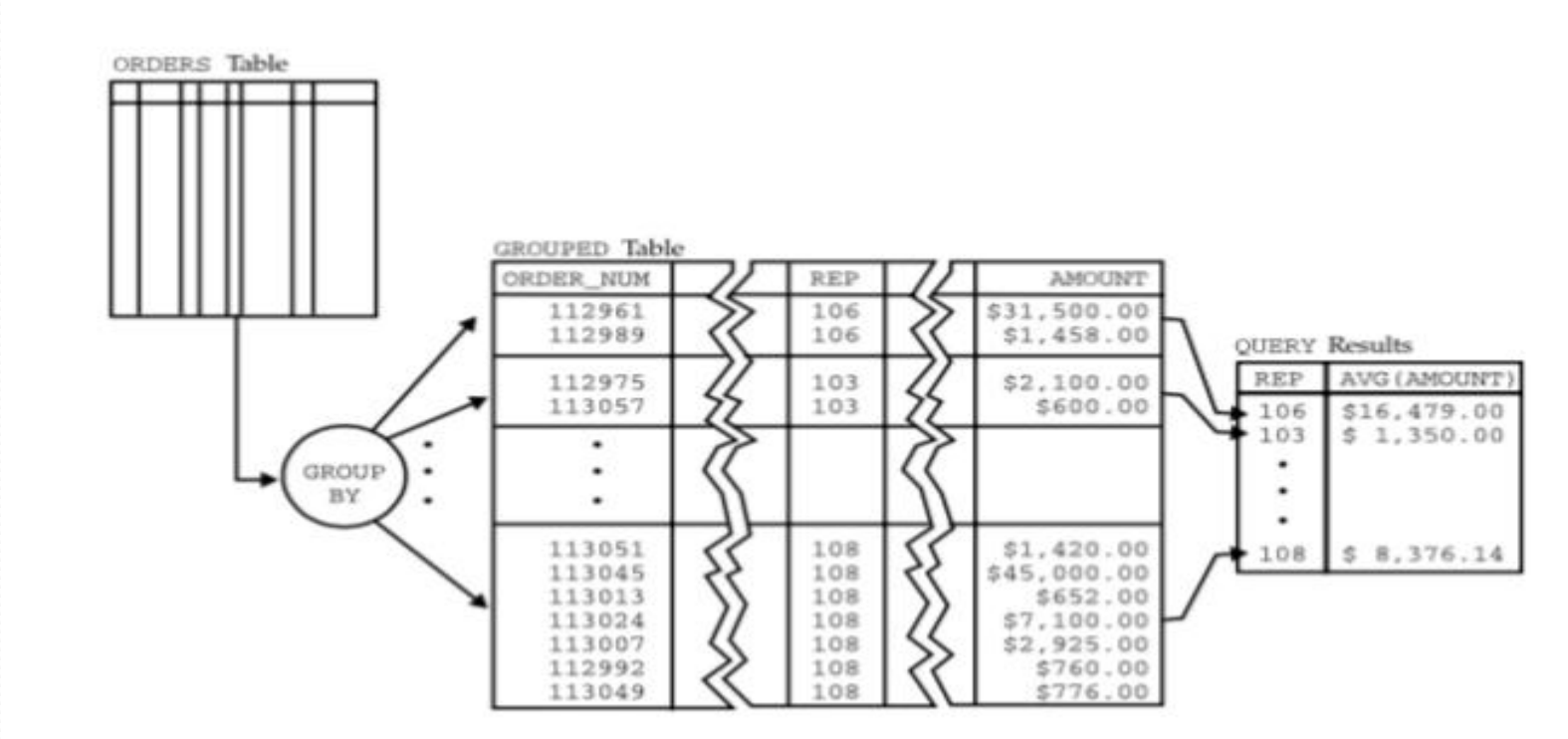
Función	Descripción
MAX(expr)	Valor máximo del grupo
MIN(expr)	Valor mínimo del grupo
AVG(expr)	Valor medio del grupo
SUM(expr)	Suma de todos los valores del grupo
COUNT(*)	Número de filas que tiene el resultado de la consulta
COUNT(columna)	Número de valores no nulos que hay en esa columna

- Las funciones de agregación sólo se pueden usar en las cláusulas SELECT Y HAVING.
- Contar valores distintos **COUNT(DISTINCT columna)**

- COUNT(*): Calcula el número de filas que tiene el resultado de la consulta.
- COUNT(columna): Cuenta el número de valores no nulos que hay en esa columna.

Agrupamiento de filas (GROUP BY)

- La cláusula GROUP BY nos permite crear grupos de filas que tienen los mismos valores en las columnas por las que se desea agrupar.



Agrupamiento de filas (GROUP BY)

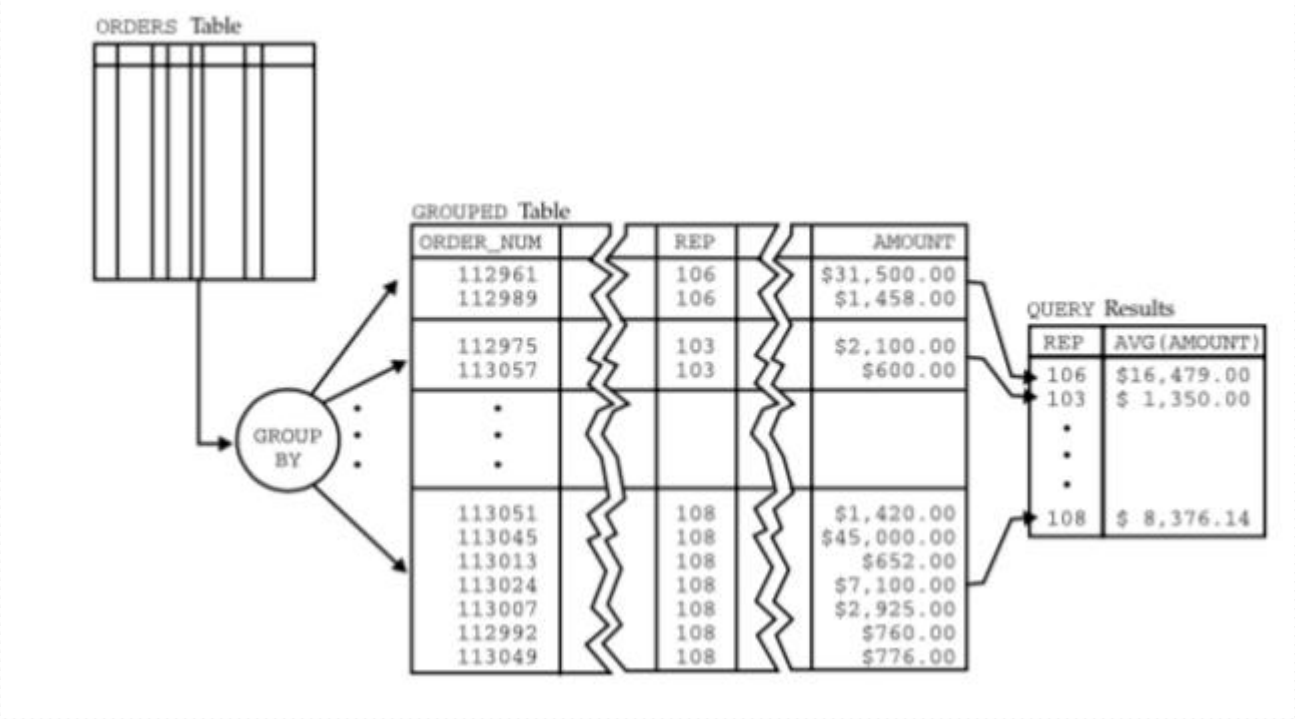
- Visualizar los nombres de todos los fabricantes y la cantidad de productos de estos que tenemos en tienda.

```
Select COUNT(P.codigo), F.nombre
FROM producto as P INNER JOIN  fabricante as F
ON  P.codigo_fabricante=F.codigo
GROUP BY  P.codigo_fabricante
```

COUNT(P.codigo)	nombre
2	Asus
2	Hewlett-Packard
1	Samsung
1	Seagate
2	Crucial
1	Gigabyte
1	fabricante2
1	fabricante3
2	Lenovo

Condición de agrupamiento (HAVING)

- La cláusula HAVING nos permite crear filtros sobre los grupos de filas que tienen los mismos valores en las columnas por las que se desea agrupar.



```
Select COUNT(P.codigo), F.nombre
FROM producto as P INNER JOIN  fabricante as F
ON  P.codigo_fabricante=F.codigo
WHERE  F.nombre LIKE "%s%"
GROUP BY  P.codigo_fabricante
```

COUNT(P.codigo)	nombre
2	Asus
1	Samsung
1	Seagate

```
Select COUNT(P.codigo), F.nombre
FROM producto as P INNER JOIN  fabricante as
F
ON  P.codigo_fabricante=F.codigo
GROUP BY  P.codigo_fabricante
HAVING F.nombre LIKE "%s%"
```

COUNT(P.codigo)	nombre
2	Asus
1	Samsung
1	Seagate