# PARTE II
# LENGUAJE SQL. CONSULTA DE DATOS

**TEMA 5. CONSULTAS SENCILLAS ....3**
**1 - Consulta de los datos. ....3**
**2 – Consultas sencillas ....3**
**2.1 – Formato mínimo de selección ....3**
**2.2 - Consultas de todas las filas ....4**
**2.3 – Ejemplos de consultas sencillas ....4**
**2.4 - Utilización de alias ....5**
**2.5 – Ejemplos de utilización de alias ....6**
**3 - Condiciones de selección. ....8**
**3.1 – Formato de selección con consultas ....8**
**3.2 – Ejemplos de condiciones de selección ....8**
**4 - Ordenación. ....10**
**4.1 – Formato de selección con ordenación ....10**
**4.2 – Ejemplos de ordenación ....11**
**5 - Selección con limitación de filas ....12**
**5.1 – Formato de selección con limitación de filas ....12**
**5.2 – Ejemplos de limitación de filas ....13**
**6 - Resumen (1) del formato de selección ....14**

**TEMA 6. CONSULTAS CON AGRUPAMIENTO Y FUNCIONES DE GRUPOS ....15**
**1- Consultas de selección con agrupamientos ....15**
**1.1 – Formato de la consulta de selección con agrupamiento ....15**
**1.2 – Consideraciones de consultas con cláusulas de agrupamiento ....16**
**2- Funciones de grupo ....16**
**2.1- Funciones de grupo ....16**
**2. 2 – Ejemplos con funciones de grupo ....17**
**3- Resumen (2) del formato de selección ....21**

**TEMA 7. SUBCONSULTAS ....22**
**1 - ¿Qué es una subconsulta?. ....22**
**1.1 - Formato de una subconsulta ....23**
**2 – Subconsultas que devuelven una sola expresión ....24**
**2.1 - Formato de las subconsultas que devuelven una sola expresión ....24**
**2.2 - Valores de retorno y condiciones de selección. ....25**
**3 – Subconsultas que devuelven más de una expresión. ....32**
**3.1 – Formato de consultas que devuelven más de una expresión ....32**
**3.2 - Valores de retorno y condiciones de selección ....32**
**3.3 – Ejemplos de consultas que devuelven más de una expresión ....33**
**4 - Subconsultas en la selección de grupos. ....34**
**4.1 – Formato de subconsultas en la cláusula HAVING ....34**
**4.2 – Ejemplos de subconsultas en la cláusula HAVING ....34**
**5 - Subconsultas anidadas. ....35**
**5.1 – Anidación de subconsultas ....35**
**5.2 – Ejemplos de subconsultas anidadas ....35**
**6 – Subconsultas correlacionadas ....36**
**6.1 – Correlación entre consultas ....37**
**6.2 – Ejemplos de subconsultas correlacionadas ....38**

**TEMA 8. CONSULTAS MULTITABLA....39**
**1 - Multiplicaciones de tablas....39**
**1.1 – Formato de la multiplicación de tablas....39**
**1.2 – Ejemplos de la multiplicación de tablas....40**
**2 - Composiciones o combinaciones simples (JOIN)....41**
**2.1 – Formato de la combinación de tablas....41**
**3- Composición o combinación natural (INNER JOIN)....42**
**3.1 – Formato de la combinación natural de tablas....43**
**3.2 – Ejemplos de combinación natural de tablas....43**
**4 - Composiciones o combinaciones basadas en desigualdad....44**
**4.1 – Formato de combinaciones basadas en desigualdad....44**
**4.2 – Ejemplos de combinaciones basadas en desigualdad....44**
**5 - Composiciones o combinaciones de una tabla consigo misma....45**
**5.1 - Formato de una combinación de una tabla consigo misma....45**
**5.2 - Ejemplos de una combinación de una tabla consigo misma....45**
**6 - Composiciones o combinaciones externas (OUTER JOIN)....46**
**6.1 – Formato de combinaciones externas....47**
**6.2 – Ejemplos de combinaciones externas....49**
**7 - Composiciones y subconsultas....50**
**8 - Formato completo de las consultas....51**

**TEMA 9. CONSULTAS DENTRO DE OTRAS INSTRUCCIONES....52**
**1 - Creación de una tabla a partir de una selección de otra tabla....52**
**1.1 – Formato para la creación de una tabla a partir de una selección de otra tabla....52**
**1.2 – Ejemplos de la creación de una tabla a partir de una selección de otra tabla....53**
**2 - Actualización de una tabla a partir de una subconsulta....53**
**2.1- Inserciones con subconsultas....53**
**2.2 – Modificaciones con subconsultas....55**
**2.3 – Eliminaciones con subconsultas....55**
**3 - Vistas....57**
**3.1 - ¿Qué son las vistas y para qué sirven?....57**
**3.2 - Creación y utilización de vistas....58**
**3.3 - Eliminación de vistas....63**

# TEMA 5. CONSULTAS SENCILLAS

## 1 - Consulta de los datos.

Realizar una consulta en SQL consiste en recuperar u obtener aquellos datos que, almacenados en filas y columnas de una o varias tablas de una base de datos, cumplen unas determinadas especificaciones. Para realizar cualquier consulta se utiliza la sentencia SELECT.

Las primeras consultas van a ser escritas con un formato inicial de la sentencia SELECT, que se irá completando durante este tema y en temas siguientes.

Vamos a comenzar a utilizar las sentencias del lenguaje SQL. En cada apartado iremos escribiendo los formatos de las sentencias con lo que hemos visto hasta ese momento.

## 2 – Consultas sencillas

Corresponden al formato mínimo de la instrucción de selección. Las cláusulas que aparecen en ellas, SELECT y FROM, son obligatorias.

La consulta más sencilla consiste en recuperar una o varias columnas o expresiones relacionadas de una tabla.

### 2.1 – Formato mínimo de selección

Formato inicial de la sentencia SELECT

**SELECT [ALL/DISTINCT]** ***ExpresionColumna*** **[,** ***ExpresionColumna*****....]**
**FROM** ***NombreTabla*****;**

*Notación: ALL y DISTINCT aparecen entre corchetes porque son opcionales y separados por la barra porque hay que elegir entre ambos. La segunda Expresión de columna aparece entre corchetes por es opcional y seguida de puntos suspensivos porque puede repetirse las veces que se quiera.*

donde ***ExpresionColumna...*** es un conjunto de nombres de columnas, literales o constantes, operadores, funciones y paréntesis.
***NombreTabla.....*** es el nombre de la tabla de la que queremos seleccionar filas.
**ALL.....** obtiene los valores de todos los elementos seleccionados en todas las filas, aunque sean repetidos.
es la opción por defecto y no suele escribirse

**DISTINCT.....** obtiene los valores no repetidos de todos los elementos.

En caso de que queramos mostrar varias expresiones estas irán separadas por comas.

Este es el formato mínimo con las cláusulas SELECT y FROM que son siempre obligatorias. A este formato mínimo le iremos añadiendo otras cláusulas opcionales en los siguientes apartados y siguientes temas.

## 2.2 - Consultas de todas las filas

En lugar de las expresiones puede parecer el carácter ***** que indica que deben seleccionares todas las columnas de la tabla. No se ha escrito para no complicar el formato inicial y hacerlo más fácil de comprender. El formato sería:

**SELECT { * | [ALL/DISTINCT]** ***ExpresionColumna*** **[,*ExpresionColumna]*....] }**
**FROM** ***NombreTabla***

*Notación: hay que escoger obligatoriamente una de las opciones, entre las de expresiones de columnas y el asterisco *****. Por eso aparecen entre llaves y separadas por una barra las posibles opciones.*

## 2.3 – Ejemplos de consultas sencillas

1. Obtener todos los empleados de la tabla `empleados` con todos sus datos.

```
mysql> SELECT *
    -> FROM empleados;
+--------+---------+-----------+----------+-----------+---------+----------+--------+
| EMP_NO | APELLIDO | OFICIO    | DIRECTOR | FECHA_ALTA | SALARIO | COMISION |DEP_NO  |
+--------+---------+-----------+----------+-----------+---------+----------+--------+
|   7499 | ALONSO   | VENDEDOR  |     7698 | 1981-02-23 | 1400.00 |   400.00 |    30  |
|   7521 | LOPEZ    | EMPLEADO  |     7782 | 1981-05-08 | 1350.50 |     NULL |    10  |
|   7654 | MARTIN   | VENDEDOR  |     7698 | 1981-09-28 | 1500.00 |  1600.00 |    30  |
|   7698 | GARRIDO  | DIRECTOR  |     7839 | 1981-05-01 | 3850.12 |     NULL |    30  |
|   7782 | MARTINEZ | DIRECTOR  |     7839 | 1981-06-09 | 2450.00 |     NULL |    10  |
|   7839 | REY      | PRESIDENTE|     NULL | 1981-11-17 | 6000.00 |     NULL |    10  |
|   7844 | CALVO    | VENDEDOR  |     7698 | 1981-09-08 | 1800.00 |     0.00 |    30  |
|   7876 | GIL      | ANALISTA  |     7782 | 1982-05-06 | 3350.00 |     NULL |    20  |
|   7900 | JIMENEZ  | EMPLEADO  |     7782 | 1983-03-24 | 1400.00 |     NULL |    20  |
+--------+---------+-----------+----------+-----------+---------+----------+--------+
9 rows in set (0.03 sec)
```

2. Obtener los números de empleados, los apellidos y el número de departamento de todos los empleados de la tabla `empleados`.

```
mysql> SELECT emp_no, apellido, dep_no
    ->  FROM empleados;
+--------+----------+--------+
| emp_no | apellido | dep_no |
+--------+----------+--------+
|   7499 | ALONSO   |     30 |
|   7521 | LOPEZ    |     10 |
|   7654 | MARTIN   |     30 |
```

```
|   7698 | GARRIDO  |     30 |
|   7782 | MARTINEZ |     10 |
|   7839 | REY      |     10 |
|   7844 | CALVO    |     30 |
|   7876 | GIL      |     20 |
|   7900 | JIMENEZ  |     20 |
+--------+----------+--------+
9 rows in set (0.00 sec)
```

3. Obtener los departamentos diferentes que hay en la tabla `empleados`

```
mysql> SELECT DISTINCT dep_no
    ->  FROM empleados;

+--------+
| dep_no |
+--------+
|     10 |
|     20 |
|     30 |
+--------+
3 rows in set (0.00 sec)
```

4- Obtener los diferentes oficios que hay en cada departamento de la tabla `empleados`

```
mysql> SELECT DISTINCT oficio, dep_no
    -> FROM empleados;

+------------+--------+
| oficio     | dep_no |
+------------+--------+
| VENDEDOR   |     30 |
| EMPLEADO   |     10 |
| DIRECTOR   |     30 |
| DIRECTOR   |     10 |
| PRESIDENTE |     10 |
| ANALISTA   |     20 |
| EMPLEADO   |     20 |
+------------+--------+
7 rows in set (0.00 sec)
```

## 2.4 - Utilización de alias

SELECT [ALL/DISTINCT] *ExpresionColumna* **[AS*] AliasColumna]**
[, *ExpresionColumna* **[AS] *AliasColumna]***....]
FROM *NombreTabla* ***[AliasTabla]*** **;**

*Notación: los alias de columna y de tabla aparecen entre corchetes porque son opcionales.*

donde ***AliasTabla.....*** SQL permite asignar otro nombre a la misma tabla, dentro de la misma consulta.

***AliasColumna.....*** se escribe detrás de la expresión de columna, separado de ella al menos por un espacio. Puede ir entre comillas dobles o sin comillas (sino lleva espacios en blanco y sí solo es una palabra)
La cláusula AS que asigna el alias puede omitirse

Usar alias para una tabla es opcional cuando su finalidad consiste en simplificar su nombre original, y obligatorio en consultas cuya sintaxis lo requiera (más adelante lo utilizaremos)

Usar alias par las columnas puede ser necesario porque los títulos o cabeceras que muestra la salida de una consulta para las columnas seleccionadas, se corresponden con los nombres de las columnas de las tablas o las expresiones y esto no siempre es muy visual. Para mejorar su legibilidad y estética se utilizan los alias de columna. También puede ser utilizado, en algunos casos, para renombrar la columna y utilizar este nombre posteriormente

## 2.5 – Ejemplos de utilización de alias

**Ejemplos de alias de tabla**
1 - Mostrar el apellido y la fecha de alta de todos los empleados.

```
mysql> SELECT apellido, fecha_alta
    ->  FROM empleados emp;

+----------+------------+
| apellido | fecha_alta |
+----------+------------+
| ALONSO   | 1981-02-23 |
| LOPEZ    | 1981-05-08 |
| MARTIN   | 1981-09-28 |
| GARRIDO  | 1981-05-01 |
| MARTINEZ | 1981-06-09 |
| REY      | 1981-11-17 |
| CALVO    | 1981-09-08 |
| GIL      | 1982-05-06 |
| JIMENEZ  | 1983-03-24 |
+----------+------------+
9 rows in set (0.02 sec)
```

*Nota: de momento no podemos ver la utilidad del alias de tabla, pero más adelante veremos su necesidad.*

**Ejemplos de alias de columna.**
1 - Obtener el salario total anual (14 pagas) de los empleados de la empresa mostrando el mensaje Salario total
Vemos, en este ejemplo, las diferentes posibilidades para escribir el alias.
a) Con la cláusula AS

```
mysql> SELECT salario*14  AS Salario_total
FROM empleados;

+---------------+
| Salario_total |
+---------------+
|      18907.00 |
|      19600.00 |
|      19600.00 |
```

```
|      21000.00 |
|      25200.00 |
|      34300.00 |
|      46900.00 |
|      53901.68 |
|      84000.00 |
+---------------+
9 rows in set (0.00 sec)
```

b) Con comillas dobles (admite el carácter blanco en el alias)

```
mysql> SELECT salario*14  "Salario total"
    ->  FROM empleados;

+---------------+
| Salario total |
+---------------+
|      19600.00 |
|      18907.00 |
|      21000.00 |
|      53901.68 |
|      34300.00 |
|      84000.00 |
|      25200.00 |
|      46900.00 |
|      19600.00 |
+---------------+
9 rows in set (0.00 sec)
```

c) Con comillas simples (admite el carácter blanco en el alias)

```
mysql> SELECT salario*14  'Salario total'
    ->  FROM empleados;
+---------------+
| Salario total |
+---------------+
|      19600.00 |
|      18907.00 |
|      21000.00 |
|      53901.68 |
|      34300.00 |
|      84000.00 |
|      25200.00 |
|      46900.00 |
|      19600.00 |
+---------------+
9 rows in set (0.00 sec)
```

2 - Mostrar el número de empleado, el apellido y el departamento de los empleados de la empresa.

```
mysql> SELECT emp_no "Nº_Empleado", apellido "Apellido",
              dep_no  "Departamento"
    ->  FROM empleados;

+-------------+----------+--------------+
```

```
| Nº_Empleado | Apellido | Departamento |
+-------------+----------+--------------+
|        7499 | ALONSO   |           30 |
|        7521 | LOPEZ    |           10 |
|        7654 | MARTIN   |           30 |
|        7698 | GARRIDO  |           30 |
|        7782 | MARTINEZ |           10 |
|        7839 | REY      |           10 |
|        7844 | CALVO    |           30 |
|        7876 | GIL      |           20 |
|        7900 | JIMENEZ  |           20 |
+-------------+----------+--------------+
9 rows in set (0.00 sec)
```

# 3 - Condiciones de selección.

Para seleccionar las filas de la tabla sobre las que realizar una consulta, la cláusula `WHERE` permite incorporar una condición de selección a la sentencia `SELECT`.
Muestra todas aquellas filas para las que el resultado de evaluar la condición de selección es VERDADERO

## 3.1 – Formato de selección con consultas

Formato de consulta con condición de selección

SELECT [ALL/DISTINCT] *ExpresionColumna* [,*ExpresionColumna*....]
FROM *NombreTabla*
**[WHERE *CondicionSeleccion*];**

*Notación: la cláusula WHERE es opcional, por eso aparece toda ella entre corchetes*

donde ***CondicionSeleccion.....*** es una expresión (conjunto de nombres de columnas, literales, operadores, funciones y paréntesis) cuyo resultado es *VERDADERO/FALSO/NULO*

El funcionamiento es el siguiente: para cada fila se evalúa la condición de selección y si el resultado es VERDADERO se visualizan las expresiones indicadas.

## 3.2 – Ejemplos de condiciones de selección

**1.** Seleccionar aquellos empleados cuyo apellido empiece por 'M' y tengan un salario entre 1000 y 200 euros.

```
mysql> SELECT emp_no "Nº Empleado", apellido "Apellido", dep_no
“Departamento”
    -> FROM empleados
    -> WHERE apellido LIKE 'M%' AND salario BETWEEN 1000 AND 2000;

+-------------+----------+--------------+
```

```
| Nº Empleado | Apellido | Departamento |
+-------------+----------+--------------+
|        7654 | MARTIN   |           30 |
+-------------+----------+--------------+
1 row in set (0.01 sec)
```

El operador `LIKE` usado con `'%'` indica que puede sustituirse por cualquier grupo de caracteres

**2.** Seleccionar aquellos empleados cuyo apellido incluya una `'A'` en el segundo carácter.

```
mysql> SELECT emp_no "Nº Empleado", apellido "Apellido", dep_no
"Departamento"
    -> FROM empleados
    -> WHERE (apellido LIKE '_A%') ;

+-------------+----------+--------------+
| Nº Empleado | Apellido | Departamento |
+-------------+----------+--------------+
|        7654 | MARTIN   |           30 |
|        7698 | GARRIDO  |           30 |
|        7782 | MARTINEZ |           10 |
|        7844 | CALVO    |           30 |
+-------------+----------+--------------+
4 rows in set (0.00 sec)
```

El operador `LIKE` usado con `'_'` indica que ocupa la posición de un carácter.

**3.** Seleccionar los empleados existentes en los departamentos 10 y 30.

```
mysql> SELECT emp_no "Nº Empleado", apellido "apellido", dep_no
"Departamento"
    -> FROM empleados
    -> WHERE dep_no=10 OR dep_no=30;
+-------------+----------+--------------+
| Nº Empleado | apellido | Departamento |
+-------------+----------+--------------+
|        7499 | ALONSO   |           30 |
|        7521 | LOPEZ    |           10 |
|        7654 | MARTIN   |           30 |
|        7698 | GARRIDO  |           30 |
|        7782 | MARTINEZ |           10 |
|        7839 | REY      |           10 |
|        7844 | CALVO    |           30 |
+-------------+----------+--------------+
7 rows in set (0.02 sec)
```

También puede hacerse utilizando el operador IN: El operador `IN` comprueba si una determinada expresión toma alguno de los valores indicados entre paréntesis.

```
mysql> SELECT emp_no "Nº EMPLEADO", apellido, dep_no Departamento
    ->  FROM empleados
    ->  WHERE dep_no IN(10,30);
```

**4.** Seleccionar los empleados que tienen de oficio ANALISTA

```
mysql> SELECT emp_no, apellido, oficio
    -> FROM empleados
    -> WHERE oficio = 'ANALISTA';
+--------+----------+----------+
| emp_no | apellido | oficio   |
+--------+----------+----------+
|   7876 | GIL      | ANALISTA |
+--------+----------+----------+
1 row in set (0.00 sec)
```

Aunque el campo oficio está grabado en mayúsculas obtenemos el mismo resultado si lo escribimos en minúsculas (MySQL no diferencia entre ambas)

```
mysql> SELECT emp_no, apellido, oficio
    -> FROM empleados
    -> WHERE oficio = 'analista';
+--------+----------+----------+
| emp_no | apellido | oficio   |
+--------+----------+----------+
|   7876 | GIL      | ANALISTA |
+--------+----------+----------+
1 row in set (0.00 sec)
```

*NOTA: MySql no diferencia mayúsculas de minúsculas pero otros gestores de bases de datos sí. Por lo que si se trabaja con otro gestor debe tenerse en cuenta esa posibilidad a la hora de escribir las sentencias.*

# 4 - Ordenación.

Para obtener la salida de una consulta clasificada por algún criterio o especificación, la sentencia SELECT dispone de la cláusula ORDER BY para ordenar.

## 4.1 – Formato de selección con ordenación

Formato de consulta con ordenación

SELECT [ALL/DISTINCT] *ExpresionColumna* [,*ExpresionColumna*....]
FROM *NombreTabla*
[WHERE *CondicionSeleccion*]
**[ORDER BY *{ExpresionColumna|Posicion}* [ASC|DESC]**
**[,*{ExpresionColumna|Posicion}* [ASC|DESC].....] ] ;**

*Notación: la cláusula ORDER BY es opcional, por eso aparece toda ella entre corchetes. Dentro de ella expresión de columna y posición van entre llaves porque hay que elegir una de ellas y ASC y DESC entre corchetes porque son opcionales*

donde **ASC|DESC.....** ASC (ascendente) o DESC (descendente) indica la forma de ordenación para esa expresión. Por omisión es ASC.

**ExpresionColumna.....** conjunto de nombres de columna con literales, operadores y/o funciones. También admite alias.

**Posicion.....** si queremos ordenar por expresiones que se muestran en el select la ExpresionColumna puede ser sustituida por el número**, Posicion,** que corresponde al número de orden que ocupa en la lista de expresiones visualizadas en la select.

Si existe más de una expresión por la que ordenar estas aparecen separadas por comas y el orden en que se realizan las clasificaciones es de izquierda a derecha, es decir, a igualdad de la expresión más a la izquierda ordena por la siguiente expresión y así sucesivamente.

## 4.2 – Ejemplos de ordenación

1. Obtener relación alfabética de todos los empleados con todos sus datos.

```
mysql> SELECT dep_no, apellido, salario
    ->  FROM empleados
    ->  ORDER BY apellido;

+--------+----------+---------+
| dep_no | apellido | salario |
+--------+----------+---------+
|     30 | ALONSO   | 1400.00 |
|     30 | CALVO    | 1800.00 |
|     30 | GARRIDO  | 3850.12 |
|     20 | GIL      | 3350.00 |
|     20 | JIMENEZ  | 1400.00 |
|     10 | LOPEZ    | 1350.50 |
|     30 | MARTIN   | 1500.00 |
|     10 | MARTINEZ | 2450.00 |
|     10 | REY      | 6000.00 |
+--------+----------+---------+
9 rows in set (0.00 sec)
```

2. Obtener clasificación alfabética de empleados por departamentos.

```
mysql> SELECT dep_no, apellido, salario
    ->  FROM empleados
    ->  ORDER BY dep_no, apellido;
+--------+----------+---------+
| dep_no | apellido | salario |
+--------+----------+---------+
|     10 | LOPEZ    | 1350.50 |
|     10 | MARTINEZ | 2450.00 |
|     10 | REY      | 6000.00 |
|     20 | GIL      | 3350.00 |
|     20 | JIMENEZ  | 1400.00 |
|     30 | ALONSO   | 1400.00 |
|     30 | CALVO    | 1800.00 |
|     30 | GARRIDO  | 3850.12 |
|     30 | MARTIN   | 1500.00 |
+--------+----------+---------+
9 rows in set (0.00 sec)
```

O también podemos escribir:

```
mysql> SELECT dep_no, apellido, salario
    ->  FROM empleados
    ->  ORDER BY 1,2;
```

3. Obtener los datos de los empleados clasificados por oficios y en orden descendente de salarios.

```
mysql> SELECT emp_no, apellido, oficio, salario
    ->  FROM empleados
->  ORDER BY oficio, salario DESC;
+--------+----------+------------+---------+
| emp_no | apellido | oficio     | salario |
+--------+----------+------------+---------+
|   7876 | GIL      | ANALISTA   | 3350.00 |
|   7698 | GARRIDO  | DIRECTOR   | 3850.12 |
|   7782 | MARTINEZ | DIRECTOR   | 2450.00 |
|   7900 | JIMENEZ  | EMPLEADO   | 1400.00 |
|   7521 | LOPEZ    | EMPLEADO   | 1350.50 |
|   7839 | REY      | PRESIDENTE | 6000.00 |
|   7844 | CALVO    | VENDEDOR   | 1800.00 |
|   7654 | MARTIN   | VENDEDOR   | 1500.00 |
|   7499 | ALONSO   | VENDEDOR   | 1400.00 |
+--------+----------+------------+---------+
9 rows in set (0.00 sec)
```

# 5 - Selección con limitación de filas

Nos va a permitir limitar el número de filas que se visualicen como resultado de una sentencia select.

## 5.1 – Formato de selección con limitación de filas

Formato de consulta con limitación de las filas que se visualizan dentro de las filas seleccionadas con el WHERE.

SELECT [ALL/DISTINCT] *ExpresionColumna* [,*ExpresionColumna*....]
FROM *NombreTabla*
[WHERE *CondicionSeleccion*]
[ORDER BY *{ExpresionColumna|Posicion}* [ASC|DESC]
[,*{ExpresionColumna|Posicion}* [ASC|DESC].....] ]
**[ LIMIT [m, ] n ] ;**

*Notación: la cláusula LIMIT es opcional, por eso aparece toda ella entre corchetes. Dentro de ella también lo es indicar el valor de m*

donde **m**..... es el número de fila por el que se comienza la visualización. Las filas se empiezan a numerar por 0.
Es opcional y en caso de omitirse se supone el valor 0 (1ª fila)
**n**..... indica el número de filas que se quieren visualizar.

## 5.2 – Ejemplos de limitación de filas

1. Obtener los datos de los 5 empleados con menos salario.

```
mysql> SELECT emp_no, apellido, salario, dep_no
    -> FROM empleados
    -> ORDER BY salario
    -> LIMIT 5;
+--------+----------+---------+--------+
| emp_no | apellido | salario | dep_no |
+--------+----------+---------+--------+
|   7521 | LOPEZ    | 1350.50 |     10 |
|   7499 | ALONSO   | 1400.00 |     30 |
|   7900 | JIMENEZ  | 1400.00 |     20 |
|   7654 | MARTIN   | 1500.00 |     30 |
|   7844 | CALVO    | 1800.00 |     30 |
+--------+----------+---------+--------+
5 rows in set (0.00 sec)
```

2. Obtener clasificación alfabética de empleados según su apellido y mostrar desde el 5º hasta el 7º de la lista

```
mysql> SELECT emp_no, apellido, salario, dep_no
    -> FROM empleados
    -> ORDER BY apellido
    -> LIMIT 4,3;
+--------+----------+---------+--------+
| emp_no | apellido | salario | dep_no |
+--------+----------+---------+--------+
|   7900 | JIMENEZ  | 1400.00 |     20 |
|   7521 | LOPEZ    | 1350.50 |     10 |
|   7654 | MARTIN   | 1500.00 |     30 |
+--------+----------+---------+--------+
5 rows in set (0.00 sec)
```

Si observamos la salida producida al ordenar por apellido comprobamos que se han visualizado 3 filas desde la 5º (Fila 4 empezando por 0)

```
mysql> SELECT emp_no, apellido, salario, dep_no
    -> FROM empleados
    -> ORDER BY apellido
    -> ;
+--------+----------+---------+--------+
| emp_no | apellido | salario | dep_no |
+--------+----------+---------+--------+
|   7499 | ALONSO   | 1400.00 |     30 |Fila 0
|   7844 | CALVO    | 1800.00 |     30 |Fila 1
|   7698 | GARRIDO  | 3850.12 |     30 |Fila 2
|   7876 | GIL      | 3350.00 |     20 |Fila 3
|   7900 | JIMENEZ  | 1400.00 |     20 |Fila 4
|   7521 | LOPEZ    | 1350.50 |     10 |Fila 5
|   7654 | MARTIN   | 1500.00 |     30 |Fila 6
|   7782 | MARTINEZ | 2450.00 |     10 |Fila 7
|   7839 | REY      | 6000.00 |     10 |Fila 8
+--------+----------+---------+--------+
9 rows in set (0.00 sec)
```

# 6 - Resumen (1) del formato de selección

Formato de selección con todas las cláusulas vistas hasta el momento.

```
SELECT { * | [ALL/DISTINCT]  ExpresionColumna [[AS] AliasColumna]
                [, ExpresionColumna [[AS] AliasColumna]....] }
FROM NombreTabla [AliasTabla]
[WHERE CondicionSeleccion]
[ORDER BY {ExpresionColumna|Posicion} [ASC|DESC]
                [,{ExpresionColumna|Posicion} [ASC|DESC].....] ]
[LIMIT [m, ] n ] ;
```

# TEMA 6. CONSULTAS CON AGRUPAMIENTO Y FUNCIONES DE GRUPOS

## 1- Consultas de selección con agrupamientos

SQL permite agrupar las filas de una tabla, seleccionadas en una consulta formando grupos según el contenido de alguna o algunas expresiones, y obtener salidas, calculadas a partir de los grupos formados.

Las salidas obtenidas son los resultados de agrupar y aplicar las funciones a cada uno de los grupos de las filas seleccionada en la tabla.

### 1.1 – Formato de la consulta de selección con agrupamiento

Añadimos a las cláusulas que ya conocemos las de agrupamiento y selección de grupos.

```
SELECT { ExpresionColumna [,ExpresionColumna .....] | * }
FROM NombreTabla
[WHERE CondicionSeleccion]
[GROUP BY ExpresionColumnaAgrupacion|Posición
                    [,ExpresionColumnaAgrupacion|Posicion... ]
        [HAVING CondicionSeleccionGrupos ] ]
[ORDER BY {ExpresionColumna|Posicion| } [ASC|DESC]
            [,{ExpresionColumna|Posicion| } [ASC|DESC].....] ]
[LIMIT [m, ] n ] ;
```

*Notación: la cláusula GROUP BY es opcional, por eso aparece toda ella entre corchetes y en caso de existir debe ir antes de la cláusula ORDER BY. En caso de existir la cláusula GROUP BY dentro de ella puede estar la cláusula HAVING, que es a su vez opcional.*

Donde ***ExpresionColumna.....*** es una expresión que contiene columnas de la tabla, literales, operadores y/o funciones. Además admite alias.
Esta expresión solo puede contener columnas de agrupación y/o funciones de grupo.

***ExpresionColumnaAgrupacion....*** es una expresión que contiene columnas de la tabla, literales, operadores y/o funciones por la que se formarán los grupos. No admite alias.

***Posicion .....*** posición que ocupa le expresión por la que queremos agrupar en la lista de expresiones visualizadas.

***CondicionSelecionGrupos……..*** es una condición, expresión que devuelve el valor *verdadero*

*/falso/nulo,* para seleccionar grupos

Si existe más de una expresión por la que agrupar, estas aparecen separadas por comas y el orden en que se realizan las agrupaciones es de izquierda a derecha. Se forman grupos con la expresión más a la izquierda y a igualdad de la expresión más a la izquierda se agrupa por la siguiente expresión y así sucesivamente.

La cláusula `HAVING` se emplea para controlar qué grupos se seleccionan, una vez realizada la agrupación. Esta asociada a la cláusula `GROUP BY` y no tiene sentido sin ella.

La salida después de utilizar la cláusula `GROUP BY` queda ordenada por las expresiones de agrupación. Las cláusulas `ORDER BY` y `LIMIT` pueden utilizarse después de agrupar, si se quiere agrupar por otro concepto y/o limitar el número de filas. Solo existe una limitación en la cláusula `ORDER BY`, ya que no puede contener funciones de grupo. En caso de que quiera ordenarse por una de estas funciones deberemos hacerlo referenciando la posición que ocupa en la select.

Ni en la expresión de la cláusula `GROUP BY` ni en condición de la cláusula `HAVING` pueden utilizarse los alias.

## 1.2 – Consideraciones de consultas con cláusulas de agrupamiento

El funcionamiento de la sentencia `SELECT` con cláusulas de agrupamiento es el siguiente:

- primero realiza una selección de filas según la cláusula `WHERE`
- forma grupos según la cláusula `GROUP BY`
- hace una selección de grupos según la cláusula `HAVING`.

Es importante tener en cuenta que, en caso de selección con cláusula de agrupación, **solo pueden mostrarse expresiones que contengan columnas de agrupación y/o funciones de grupo**. Así mimo, si se utilizan funciones de agrupación sin la cláusula `GROUP BY` no pueden mostrarse el resto de las filas de la tabla.

La cláusula **`HAVING` actúa como un filtro sobre el resultado de agrupar las filas,** a diferencia de la cláusula `WHERE` que actúa sobre las filas antes de la agrupación.

# 2- Funciones de grupo

Estas funciones de grupo actúan sobre las filas previamente seleccionadas y los grupos que se hayan formado en ellas. El criterio para agrupar suele ser una o varias de las columnas o expresiones de la tabla llamadas columnas o expresiones de agrupación. Si no se especifica ningún criterio, las filas de la tabla seleccionadas en la consulta, formarán un grupo único.

## 2.1- Funciones de grupo

| | |
|---|---|
| **AVG(expr)** | Valor medio de “expr” ignorando los valores nulos |
| **COUNT( { * \| expr } )** | Número de veces que “expr” tiene un valor no nulo. La opción “*” cuenta el número de filas seleccionadas. |
| **MAX(expr)** | Valor máximo de “expr” |
| **MIN(expr)** | Valor mínimo de “expr” |

| | |
|---|---|
| **STDDEV(expr)** | Desviación típica de “expr” sin tener en cuenta los valores nulos |
| **SUM(expr)** | Suma de “expr” |
| **VARIANCE(expr)** | Varianza de “expr” sin tener en cuenta los valores nulos. |

**Descripción de las funciones de grupo**

**AVG(expr)** Calcula el valor medio de la expresión de columna que se indique dentro del paréntesis, teniendo en cuenta que los valores NULL no son incluidos.

**COUNT( { * | expr } )** Tiene dos posibilidades, la primera con un * cuenta el número filas seleccionadas y la segunda con una expresión de columna cuenta el número de veces que la expresión tiene el valor diferente de NULL

**MAX(expr)** Devuelve el valor máximo de la expresión de columna que le acompaña.

**MIN(expr)** Devuelve el valor mínimo de la expresión de columna que le acompaña.

**STDDEV(expr)** Calcula la desviación típica para los valores de la expresión de columna que le acompaña.

**SUM(expr)** Calcula la suma de valores de la expresión de columna indicada dentro del paréntesis

**VARIANCE(expr)** Calcula la varianza para los valores de la expresión de columna que se indique dentro del paréntesis, teniendo en cuenta que los valores `NULL` no son incluidos.

Por lo general, las funciones de grupos se utilizan sobre más de un grupo de filas. La cláusula `GROUP BY` establece el criterio o columnas de agrupación y se calculará el valor de la función para cada grupo. Pero también pueden utilizarse sin la cláusula `GROUP BY` y en ese caso estas funciones actúan sobre un único grupo formado por todas las filas seleccionadas.

Estas funciones pueden ser utilizadas con la cláusula `DISTICT`.
Por ejemplo `COUNT(DISTICT(NombreColumna)` cuenta cuantos valores diferentes para esa columna hay entre las filas seleccionadas o agrupadas.

## 2. 2 – Ejemplos con funciones de grupo.

**a) Ejemplos de consulta con funciones de grupo sin criterio de agrupación**

**1.** Obtener la masa salarial mensual de todos los empleados.

```
mysql> SELECT SUM(salario)
    ->  FROM empleados;

+--------------+
| SUM(salario) |
+--------------+
|     23100.62 |
+--------------+
1 row in set (0.00 sec)
```

**2.** Obtener los salarios máximo, mínimo y la diferencia existente entre ambos.

```
mysql> SELECT MAX(salario)"Salario mas alto" ,
    ->    MIN(salario) "Salario mas bajo",
    ->    MAX(salario)- MIN(salario) "Diferencia"
    -> FROM empleados;

+------------------+------------------+------------+
| Salario mas alto | Salario mas bajo | Diferencia |
+------------------+------------------+------------+
|          6000.00 |          1350.50 |    4649.50 |
+------------------+------------------+------------+
1 row in set (0.00 sec)
```

**3.** Obtener la fecha de alta más reciente.

```
mysql> SELECT MAX(fecha_alta)"Fecha alta"
    -> FROM empleados;

+------------+
| Fecha alta |
+------------+
| 1983-03-24 |
+------------+
1 row in set (0.00 sec)
```

**4.** Calcular el salario medio de los empleados.

```
mysql> SELECT AVG(salario) "Salario medio"
    -> FROM empleados;

+---------------+
| Salario medio |
+---------------+
|   2566.735569 |
+---------------+
1 row in set (0.00 sec)
```

A veces hacer la media con la función AVG no da el mismo resultado que hacer la suma y dividirla por el número de filas, `SUM/CONT.` Esto es porque `COUNT` cuenta el número de valores de datos que hay en una columna, sin incluir los valores `NULL,` y por el contrario, `COUNT(*)` cuenta todas las filas de la tabla, sin considerar que en algunas columnas existan valores `NULL`. Sin embargo la función `AVG` si tiene en cuenta las filas con valores `NULL.`
Veamos un ejemplo:

```
mysql> SELECT AVG(comision)
    -> FROM empleados;
+---------------+
| AVG(comision) |
+---------------+
|    666.666667 |
+---------------+
1 row in set (0.00 sec)

mysql> SELECT SUM(comision)/COUNT(*)
    ->  FROM empleados;
+------------------------+
```

```
| SUM(comision)/COUNT(*) |
+------------------------+
|              222.2222 |
+------------------------+
1 row in set (0.00 sec)

mysql> SELECT SUM(comision)/COUNT(comision)
    -> FROM empleados;

+--------------------------------+
| SUM(comision)/COUNT(comision) |
+--------------------------------+
|                      666.6667 |
+--------------------------------+
1 row in set (0.00 sec)
```

**5.** Calcular el salario medio de los empleados que sean ANALISTAS

```
mysql> SELECT AVG(salario) "Salario medio"
    ->  FROM empleados
    ->  WHERE oficio = 'ANALISTA';

+---------------+
| Salario medio |
+---------------+
|   3350.000000 |
+---------------+
1 row in set (0.00 sec)
```

**b) Ejemplos de consulta con funciones de grupo con criterio de agrupación `GROUP BY`**

**1.** Obtener los salarios medios por departamento.

```
mysql> SELECT dep_no "Departamento" , AVG(salario) "Salario medio"
    ->  FROM empleados
    ->  GROUP BY dep_no;

+--------------+---------------+
| Departamento | Salario medio |
+--------------+---------------+
|           10 |   3266.833333 |
|           20 |   2375.000000 |
|           30 |   2137.530029 |
+--------------+---------------+
3 rows in set (0.00 sec)
```

**2.** Obtener cuántos empleados hay en cada oficio

```
mysql> SELECT oficio "Oficio", COUNT(*) "Nº de Empleados"
    -> FROM empleados
    -> GROUP BY oficio;

+------------+-----------------+
| Oficio     | Nº de Empleados |
+------------+-----------------+
| ANALISTA   |               1 |
| DIRECTOR   |               2 |
```

```
| EMPLEADO   |              2 |
| PRESIDENTE |              1 |
| VENDEDOR   |              3 |
+------------+----------------+
5 rows in set (0.00 sec)
```

**c) Ejemplos de consulta con funciones de grupo con criterio de agrupación `GROUP BY` y cláusula `HAVING`**

`SQL` realiza la selección de grupos en el proceso siguiente:

- A partir de la tabla sobre la que se realiza la consulta, la cláusula `WHERE` actúa como un primer filtro que da como resultado una tabla interna cuyas filas cumplen la condición especificada en el `WHERE`.
- La cláusula `GROUP BY` produce la agrupación de las filas de la segunda tabla, dando como resultado una tercera tabla.
- La cláusula `HAVING` actúa filtrando las filas de la tercera tabla, según la condición de selección especificada, dando como resultado la salida de la consulta.

**1.** Seleccionar los oficios que tengan dos o más empleados:

```
mysql> SELECT oficio, COUNT(*)
    ->  FROM empleados
    ->  GROUP BY oficio
    ->  HAVING COUNT(*)>= 2;

+----------+----------+
| oficio   | COUNT(*) |
+----------+----------+
| DIRECTOR |        2 |
| EMPLEADO |        2 |
| VENDEDOR |        3 |
+----------+----------+
3 rows in set (0.00 sec)
```

**2.** Seleccionar los oficios que tengan dos o más empleados, cuyo salario supere los 1400 euros.

```
mysql> SELECT oficio, COUNT(*)
    ->  FROM empleados
    ->  WHERE salario > 1400
    ->  GROUP BY oficio
    ->  HAVING COUNT(*) >= 2;

+----------+----------+
| oficio   | COUNT(*) |
+----------+----------+
| DIRECTOR |        2 |
| VENDEDOR |        2 |
+----------+----------+
2 rows in set (0.00 sec)
```

**d) Ejemplos de consulta con funciones de grupo con criterio de agrupación `GROUP BY`, incluyendo las cláusulas `ORDER BY` y `LIMIT`**

1- Seleccionar los datos del departamento con menor salario medio

```
mysql> SELECT dep_no, AVG (salario)
    -> FROM empleados
    -> GROUP BY dep_no
```

```
    -> ORDER BY 2
    -> LIMIT 1;
+--------+---------------+
| dep_no | AVG (salario) |
+--------+---------------+
|     30 |   2137.530029 |
+--------+---------------+
1 row in set (0.23 sec)
```

2- Seleccionar los datos del departamento con mayor número de empleados.

```
mysql> SELECT dep_no, COUNT(*)
    -> FROM empleados
    -> GROUP BY dep_no
    -> ORDER BY 2 DESC
    -> LIMIT 1;

+--------+----------+
| dep_no | COUNT(*) |
+--------+----------+
|     30 |        4 |
+--------+----------+
1 row in set (0.39 sec)
```

# 3- Resumen (2) del formato de selección

Formato de selección con todas las cláusulas vistas hasta el momento.

**SELECT { * | [ALL/DISTINCT] *ExpresionColumna* [*AliasColumna*]**
**[,*ExpresionColumna* [*AliasColumna*]....] }**
**FROM *NombreTabla [AliasTabla]***
**[WHERE *CondicionSeleccion*]**
**[GROUP BY *ExpresionColumna*|*Posicion [,ExpresionColumna*|*Posicion ...* ]**
**[HAVING *CondicionSelecciónGrupos* ] ]**
**[ORDER BY *{ExpresionColumna*|*Posicion* } [ASC|DESC]**
**[,*{ExpresionColumna*|*Posicion* } [ASC|DESC].....] ]**
**[LIMIT [m, ] n ] ;**

# TEMA 7. SUBCONSULTAS

## 1 - ¿Qué es una subconsulta?.

Una subconsulta en SQL 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” que da el nombre al SQL de Lenguaje de Consultas Estructuradas (Structured Query Language) .

Anteriormente hemos utilizado la cláusula WHERE para seleccionar los datos que deseábamos comparando un valor de una columna con una constante, o un grupo de ellas. Si los valores de dichas constantes son desconocidos, normalmente por proceder de la aplicación de funciones a determinadas columnas de la tabla, tendremos que utilizar subconsultas. Por ejemplo, queremos saber la lista de empleados cuyo salario supere el salario medio. En primer lugar, tendríamos que averiguar el importe del salario medio:

```
mysql> SELECT AVG(salario)"Salario Medio"
    ->  FROM empleados;

+---------------+
| Salario Medio |
+---------------+
|   2566.735569 |
+---------------+
1 row in set (0.00 sec)
```

A continuación, anotarlo en un papel o recordarlo para la siguiente sentencia:

```
mysql> SELECT dep_no "Nº Empleado", apellido, salario
   ->   FROM empleados
   ->   WHERE salario> 2566.73;
+-------------+----------+---------+
| Nº Empleado | apellido | salario |
+-------------+----------+---------+
|          30 | GARRIDO  | 3850.12 |
|          10 | REY      | 6000.00 |
|          20 | GIL      | 3350.00 |
+-------------+----------+---------+
3 rows in set (0.00 sec)
```

Pero además de tener que anotar el resultado de otra consulta para ser utilizado en esta, tiene el problema de que si el número de empleados o el salario de estos cambiase, cambiaría el valor del salario medio y habría que modificar esta segunda consulta reemplazando el valor antiguo por el nuevo valor.

Sería mucho más eficiente utilizar una subconsulta:

```
mysql> SELECT dep_no "Nº Empleado", apellido, salario
    ->  FROM empleados
    ->  WHERE salario > (SELECT AVG(salario)
    ->                   FROM empleados);

+-------------+----------+---------+
| Nº Empleado | apellido | salario |
+-------------+----------+---------+
|          30 | GARRIDO  | 3850.12 |
|          10 | REY      | 6000.00 |
|          20 | GIL      | 3350.00 |
+-------------+----------+---------+
3 rows in set (0.00 sec)
```

La subconsulta (comando SELECT entre paréntesis) se ejecuta primero y, posteriormente, el valor extraído es utilizado en la consulta principal, obteniéndose el resultado deseado.

## 1.1 - Formato de una subconsulta

**( SELECT [ALL/DISTINCT]** ***ExpresionColumna*** **[,*ExpresionColumna .....]**
**FROM** ***NombreTabla [ , NombreTabla ]***
**[WHERE** ***CondicionSeleccion*****]**
**[GROUP BY** ***ExpresionColumnaAgrupacion [, ExpresionColumnaAgrupacion ...*** **]**
**[HAVING** ***CondicionSeleciónGrupos*** **] ] )**

donde el formato de la sentencia SELECT entre paréntesis tiene las siguientes diferencias con la sentencia SELECT de las consultas:

- No tiene sentido la cláusula ORDER BY ya que los resultados de una subconsulta se utilizan internamente y no son visibles al usuario.
- Los nombres de columna que aparecen las expresiones en una subconsulta pueden referirse a columnas de la tabla de la consulta principal y se conocen como *referencias externas.*

Una subconsulta siempre forma parte de la condición de selección en las cláusulas WHERE o HAVING. Cuando incluimos una subconsulta en una sentencia select el funcionamiento es el siguiente: para cada fila de la consulta ejecuta la subconsulta y con ese resultado se evalúa la fila correspondiente de la consulta, mostrándose si el resultado de la evaluación es VERDADERO.

Las subconsultas habitualmente devuelven una sola expresión pero también pueden devolver más de una. La sentencia select que conecta con la subconsulta deberá recoger estos valores en una o varias columnas, según sea la subconsulta, para poder después compararlos.

Vamos a verlo con unos ejemplos
a) Subconsulta que devuelve una sola expresión
Obtener el nombre del departamento donde trabaja GARRIDO

```
mysql> SELECT dnombre
```

```
    -> FROM departamentos
    -> WHERE dep_no = (SELECT dep_no
    ->                   FROM empleados
    ->                   WHERE apellido = 'GARRIDO');

+---------+
| dnombre |
+---------+
| VENTAS  |
+---------+
1 row in set (0.02 sec)
```

La subconsulta devuelve una sola expresión dep_no, que en este caso el valor del departamento de GARRIDO que la consulta compara con el correspondiente en la tabla `departamentos`.

b)Subconsulta que devuelven más de una expresión
Obtener los empleados que tengan el mismo oficio y departamento que ALONSO

```
mysql> SELECT emp_no, apellido, oficio, dep_no
    ->  FROM   empleados
    ->  WHERE (dep_no, oficio) = (SELECT  dep_no, oficio
    ->                              FROM empleados
    ->                             WHERE apellido = 'ALONSO');

+--------+----------+----------+--------+
| emp_no | apellido | oficio   | dep_no |
+--------+----------+----------+--------+
|   7499 | ALONSO   | VENDEDOR |     30 |
|   7654 | MARTIN   | VENDEDOR |     30 |
|   7844 | CALVO    | VENDEDOR |     30 |
+--------+----------+----------+--------+
3 rows in set (0.02 sec)
```

La subconsulta devuelve dos expresiones, dep_no y oficio (en este caso formadas por una columna cada una) correspondientes al departamento y oficio de ALONSO y la consulta lo compara con dos columnas dep_no y oficio, de cada una de las filas de la tabla.

Lo más habitual es el primer caso por ello comenzaremos con las subconsultas que devuelven una sola expresión y posteriormente, en el apartado siguiente, trataremos las que devuelven más de una expresión.

# 2 – Subconsultas que devuelven una sola expresión

## 2.1 - Formato de las subconsultas que devuelven una sola expresión

Es un formato habitual de una sentencia `SELECT` con la particularidad de que solo debe seleccionarse una expresión de columna.

**( SELECT [ALL/DISTINCT]** ***ExpresionColumna***
**FROM** ***NombreTabla [ , NombreTabla ]***
**[WHERE** ***CondicionSeleccion*****]**
**[GROUP BY** ***ExpresionColumnaAgrupacion [,ExpresionColumnaAgrupacion ...*** **]**
**[HAVING** ***CondicionSeleciónGrupos*** **] ] )**

## 2.2 - Valores de retorno y condiciones de selección

Como hemos dicho comenzaremos con la subconsultas que devuelven una sola expresión. El resultado de ejecutar la subconsulta puede devolvernos, en esta expresión, un valor simple o más de un valor. Según el retorno de la subconsulta, el operador de comparación que se utilice en la condición de selección del `WHERE` o `HAVING` deberá ser del tipo apropiado según la tabla siguiente:

| *Retorno de la subconsulta* | *Operador comparativo* |
|---|---|
| Valor simple | De tipo aritmético |
| Más de un valor | De tipo lógico |

### 2.2.1 - Condición de selección con operadores aritméticos de comparación

Se utiliza cuando la subconsulta devuelve un único valor a comparar con una expresión, por lo general formada a partir de la fila obtenida en la consulta principal. Si la comparación resulta cierta (`TRUE`), la condición de selección también lo es. Si la subconsulta no devuelve ninguna fila (`NULL`), la comparación devuelve también el valor `NULL`. Si la condición de comparación resulta falsa (`FALSE`), la condición de selección también lo será.

**Formato para la condición de selección con operadores aritméticos de comparación**

**ExpresionColumna OperadorComparacion (Subconsulta)**

donde **OperadorComparacion** puede ser =,<>,<,>,<=,<=

**Ejemplos de subconsultas con operadores de comparación**

1. Obtener todos los empleados que tienen el mismo oficio que GARRIDO

```
mysql> SELECT emp_no "Nº Empleado", apellido, oficio
    ->  FROM empleados
    ->  WHERE oficio = (SELECT oficio
    ->                    FROM empleados
    ->                    WHERE apellido = 'GARRIDO');

+-------------+----------+----------+
```

```
| Nº Empleado | apellido | oficio   |
+-------------+----------+----------+
|        7698 | GARRIDO  | DIRECTOR |
|        7782 | MARTINEZ | DIRECTOR |
+-------------+----------+----------+
2 rows in set (0.00 sec)
```

La subconsulta devuelve el oficio de GARRIDO que es DIRECTOR y se visualizan los trabajadores de oficio DIRECTOR, entre ellos GARRIDO. Más adelante haremos un ejemplo donde no se visualizará el empleado de la subconsulta.

2. Obtener información de los empleados que ganan más que cualquier empleado del departamento 30.

```
mysql> SELECT emp_no "Nº Empleado", apellido, salario,
         dep_no "Nº Departamento"
    ->  FROM empleados
    ->  WHERE salario > (SELECT MAX(salario)
    ->                     FROM empleados
    ->                     WHERE dep_no=30);

+-------------+----------+---------+-----------------+
| Nº Empleado | apellido | salario | Nº Departamento |
+-------------+----------+---------+-----------------+
|        7839 | REY      | 6000.00 |              10 |
+-------------+----------+---------+-----------------+
1 row in set (0.00 sec)
```

3. Visualizar el número de VENDEDORES del departamento VENTAS.

```
mysql> SELECT COUNT(*) "Total Empleados"
    ->  FROM empleados
    ->  WHERE oficio = 'VENDEDOR'
    ->    AND dep_no = (SELECT dep_no
    ->                    FROM departamentos
    ->                    WHERE dnombre = 'VENTAS')
    ->  GROUP BY oficio;
+-----------------+
| Total Empleados |
+-----------------+
|               3 |
+-----------------+
1 row in set (0.00 sec)
```

4. Visualizar la suma de los salarios para cada oficio de los empleados del departamento VENTAS.

```
mysql> SELECT oficio, sum(salario)"Suma salarios"
    ->  FROM empleados
    ->  WHERE dep_no = (SELECT dep_no
    ->                    FROM departamentos
    ->                    WHERE dnombre='VENTAS')
    ->  GROUP BY oficio;
+----------+--------------+
| oficio   | Suma salarios |
```

```
+----------+--------------+
| DIRECTOR |       3850.12 |
| VENDEDOR |       4700.00 |
+----------+--------------+
2 rows in set (0.01 sec)
```

## 2.2.3 - Condición de selección con operadores lógicos.

Se utiliza cuando la subconsulta puede devolver más de una fila a comparar con la fila actual de la consulta principal. En ese caso los operadores aritméticos dan error.

**Formato para la condición de selección con operadores lógicos**

**ExpresionColumna OperadorLogico (Subconsulta)**

Donde **OperadorLogico** puede ser IN, ANY, ALL Y EXISTS

**a) Operador lógico IN**

Comprueba si valores de la fila actual de la consulta principal coincide con alguno de la lista de valores devueltos por la subconsulta. Si el resultado es afirmativo la comparación resulta cierta (`TRUE`).

**Formato para la condición de selección con el operador lógico `IN`**

**ExpresionColumna [NOT] IN (Subconsulta)**

**Ejemplos del operador lógico IN**

1. Listar, en orden alfabético, aquellos empleados que no trabajen ni en Madrid ni en Barcelona.

```
mysql> SELECT emp_no, apellido, dep_no
    -> FROM empleados
    -> WHERE dep_no IN (SELECT dep_no
    ->                   FROM departamentos
    ->                   WHERE localidad NOT LIKE 'MADRID'
    ->                      AND localidad NOT LIKE 'BARCELONA');

+--------+----------+--------+
| emp_no | apellido | dep_no |
+--------+----------+--------+
|   7876 | GIL      |     20 |
|   7900 | JIMENEZ  |     20 |
+--------+----------+--------+
2 rows in set (0.00 sec)
```

La subconsulta selecciona todos los departamentos que no están en Madrid ni en Barcelona, y la consulta principal comprueba, empleado a empleado, si su departamento es uno de los seleccionados en la subconsulta, visualizando sus datos caso de ser cierto.

También podemos resolverla utilizando `NOT IN`

```
mysql> SELECT emp_no, apellido, dep_no
    ->    FROM empleados
    ->    WHERE dep_no NOT IN (SELECT dep_no
    ->                         FROM departamentos
    ->                         WHERE localidad LIKE 'MADRID'
    ->                           OR localidad LIKE 'BARCELONA');
```

2. Listar los nombres de los departamentos que tengan algún empleado con fecha de alta anterior a 1982.

```
mysql> SELECT dep_no "NºDepartamento", dnombre "Departamento"
    ->  FROM departamentos
    ->  WHERE dep_no IN (SELECT DISTINCT(dep_no)
    ->                   FROM empleados
    ->                   WHERE fecha_alta < '1982-01-01');

+----------------+--------------+
| NºDepartamento | Departamento |
+----------------+--------------+
|             10 | CONTABILIDAD |
|             30 | VENTAS       |
+----------------+--------------+
2 rows in set (0.00 sec)
```

En este ejemplo, la subconsulta selecciona todos los empleados cuya fecha de alta sea anterior al año 1982, y la consulta principal compara, departamento a departamento, si coincide con el de alguno de los que hayan sido seleccionados en la subconsulta.

3. Obtener los departamentos y sus nombres, siempre que haya más de dos empleados trabajando en ellos.

```
mysql> SELECT dep_no "NºDepartamento", dnombre "Departamento"
    ->  FROM departamentos
    ->  WHERE dep_no IN (SELECT dep_no
    ->                   FROM empleados
    ->                   GROUP BY dep_no
    ->                   HAVING COUNT(*)>2 );

+----------------+--------------+
| NºDepartamento | Departamento |
+----------------+--------------+
|             10 | CONTABILIDAD |
|             30 | VENTAS       |
+----------------+--------------+
2 rows in set (0.00 sec)
```

La subconsulta selecciona todos los departamentos que tienen más de un empleado trabajando, y la consulta principal comprueba, departamento a departamento, si es uno de los seleccionados en la subconsulta, visualizando sus datos caso de ser cierto.

**b) Operadores lógicos ANY y ALL**

Se utilizan junto a los operadores aritméticos de comparación para ampliar las posibles comprobaciones de valores obtenidos a partir de la fila seleccionada en la consulta principal con valores obtenidos en la subconsulta.
Su uso a menudo es sustituido por el del operador `IN`.

**Formato para los operadores ANY/ALL**

**ExpresionColumna OperadorComparacion {ANY|ALL} (Subconsulta)**

donde **OperadorComparacion** puede ser =,<>,<,>,<=,<

El operador `ANY` con uno de los seis operadores aritméticos compara el valor de la expresión formada a partir de la consulta principal con valores producidos por la subconsulta. Si alguna de las comparaciones individuales produce un resultado verdadero (`TRUE`), el operador `ANY` devuelve un resultado verdadero (`TRUE`).

El operador `ALL` también se utiliza con los operadores aritméticos para comparar un valor de la expresión formada a partir de la consulta principal con cada uno de los valores de datos producidos por la subconsulta. Si todos los resultados de las comparaciones son ciertos (`TRUE`), el operador `ALL` devuelve un valor cierto (`TRUE`).

**Ejemplos de ANY**
1. Visualizar los nombres de los departamentos que tengan empleados trabajando en ellos..

```
mysql> SELECT dep_no "Nº Departamento", dnombre Departamento
    ->  FROM departamentos
    ->  WHERE dep_no = ANY (SELECT dep_no
    ->                      FROM empleados);

+-----------------+---------------+
| Nº Departamento | Departamento  |
+-----------------+---------------+
|              10 | CONTABILIDAD  |
|              20 | INVESTIGACION |
|              30 | VENTAS        |
+-----------------+---------------+
3 rows in set (0.00 sec)
```

Lo mismo con el operador `IN`:

```
mysql> SELECT dep_no "Nº Departamento", dnombre Departamento
    ->  FROM departamentos
    ->  WHERE dep_no IN (SELECT dep_no
    ->                   FROM empleados);

+-----------------+---------------+
| Nº Departamento | Departamento  |
+-----------------+---------------+
|              10 | CONTABILIDAD  |
|              20 | INVESTIGACION |
```

```
|              30 | VENTAS        |
+-----------------+---------------+
3 rows in set (0.00 sec)
```

2. Seleccionar aquellos departamentos en los que al menos exista un empleado con comisión nula.

```
mysql> SELECT dep_no "Nº Departamento", dnombre Departamento
    ->  FROM departamentos
    ->  WHERE dep_no = ANY (SELECT DISTINCT dep_no
    ->                       FROM empleados WHERE comision IS NULL);

+-----------------+--------------+
| Nº Departamento | Departamento |
+-----------------+--------------+
|              30 | VENTAS       |
+-----------------+--------------+
1 row in set (0.00 sec)
```

**Ejemplos del operador ALL**

1. Listar los empleados con mayor salario que todos los del departamento 20

```
mysql> SELECT dep_no "Nº Departamento", apellido, salario
    -> FROM empleados
    -> WHERE salario >  ALL (SELECT salario
    ->                       FROM empleados
    ->                       WHERE dep_no = 20);

+-----------------+----------+---------+
| Nº Departamento | apellido | salario |
+-----------------+----------+---------+
|              30 | GARRIDO  | 3850.12 |
|              10 | REY      | 6000.00 |
+-----------------+----------+---------+
2 rows in set (0.00 sec)
```

2. Listar los departamentos que no tienen empleados

```
mysql> SELECT dep_no "Nº Departamento",
              dnombre "Nombre departamento"
    -> FROM departamentos
    -> WHERE dep_no <>  ALL (SELECT DISTINCT(dep_no)
    ->                       FROM empleados);

+-----------------+---------------------+
| Nº Departamento | Nombre departamento |
+-----------------+---------------------+
|              40 | PRODUCCION          |
+-----------------+---------------------+
1 row in set (0.00 sec)
```

Como vemos <> ALL es equivalente a NOT IN

**c) Operador lógico EXISTS**

Se utiliza cuando la condición de selección consiste exclusivamente en comprobar que la subconsulta

devuelve alguna fila seleccionada según la condición incluida en la propia subconsulta. El operador `EXISTS` no necesita que la subconsulta devuelva alguna columna porque no utiliza ninguna expresión de comparación, justificando así la aceptación del * en el formato de la misma.

**Formato para la condición de selección con el operador lógico `EXISTS`**

| |
|---|
| ***ExpresiónColumna* [NOT] EXISTS *(Subconsulta)*** |

Una subconsulta expresada con el operador `EXISTS` también podrá expresarse con el operador `IN`.
Se utiliza ,sobre todo, con consultas correlacionadas que veremos más adelante.

**Ejemplos con el operador lógico EXISTS**
1 - Visualizar los departamentos en los que hay más de un trabajador.

```
mysql> SELECT dep_no, dnombre
    -> FROM departamentos e
    -> WHERE EXISTS ( SELECT *
    ->                  FROM empleados d
    ->                  WHERE e.dep_no = d.dep_no
    ->                  GROUP BY dep_no
    ->                  HAVING COUNT(*) > 1);


+--------+--------------+
| dep_no | dnombre      |
+--------+--------------+
|     10 | CONTABILIDAD |
|     30 | VENTAS       |
+--------+--------------+
2 rows in set (0.00 sec)
```

2. Listar las localidades donde existan departamentos con empleados cuya comisión supere el 10% del salario.

```
mysql> SELECT localidad
    ->  FROM departamentos d
    ->  WHERE EXISTS (SELECT *
    ->                  FROM empleados e
    ->                  WHERE comision>10*salario/100
                              AND e.dep_no=d.dep_no);

+-----------+
| localidad |
+-----------+
| MADRID    |
+-----------+
1 row in set (0.00 sec)
```

*Nota: las tablas de departamentos y de empleados necesitan llevar alias para poder realizar parte de la condición de selección en la subconsulta ya que en ambas existe una columna con el mismo nombre (dep_no). Esto se verá en el apartado posterior de consultas correlacionadas.*

La misma subconsulta podemos expresarla con el operador `IN` de la siguiente manera:

```
mysql> SELECT localidad
    ->  FROM departamentos
    ->  WHERE dep_no IN (SELECT dep_no
    ->                   FROM empleados
    ->                   WHERE comision>10*salario/100);

+-----------+
| localidad |
+-----------+
| MADRID    |
+-----------+
1 row in set (0.00 sec)
```

# 3 – Subconsultas que devuelven más de una expresión.

Hemos dicho que las subconsultas habitualmente de devuelven una sola expresión pero también pueden devolver más de una.

Son útiles, sobre todo, cuando la clave ajena de la tabla de la consulta, que se corresponde con una clave primaria de la tabla de la subconsulta, es compuesta.

## 3.1 – Formato de consultas que devuelven más de una expresión

Es un formato equivalente, pero la select de la subconsulta devuelve más de una expresión.

**(SELECT [ALL/DISTINCT] *ExpresionColumna* [,*ExpresionColumna .....]**
**FROM *NombreTabla [ , NombreTabla ]***
**[WHERE *CondicionSeleccion*]**
**[GROUP BY *ExpresionColumnaAgrupacion [,ExpresionColumnaAgrupacion ...* ]**
**[HAVING *CondicionSelecionGrupos* ] ] )**

## 3.2 - Valores de retorno y condiciones de selección

La condición de selección debe tener en cuenta el número de valores de retorno de la subconsulta y compararlos con el mismo número de expresiones.

**( ExpresionColumna [,ExpresionColumna ....] ) OperadorComparacion (Subconsulta)**

donde **OperadorComparacion** tiene que ser el operador = o bien el operador `IN` dependiendo de si

devuelve un valor simple o puede devolver varios valores.

El número de **ExpresionColumna** en la consulta será igual al número de ExpresionColumna que nos proporcione el select de la subconsulta

## 3.3 – Ejemplos de consultas que devuelven más de una expresión

1- Obtener los empleados que pertenecen al mismo departamento y entraron en la empresa el mismo día que GARRIDO

```
mysql> SELECT emp_no, apellido
    ->  FROM empleados
    ->  WHERE (dep_no,fecha_alta) = (SELECT  dep_no, fecha_alta
    ->                                FROM empleados
    ->                                WHERE apellido='GARRIDO');

+--------+----------+
| emp_no | apellido |
+--------+----------+
|   7698 | GARRIDO  |
+--------+----------+
1 row in set (0.00 sec)
```

2 - Obtener el empleado que pertenece al mismo departamento que JIMENEZ y tiene el máximo salario

```
mysql> SELECT emp_no, apellido
    ->  FROM empleados
    ->  WHERE (dep_no,salario)=(SELECT  dep_no, MAX(salario)
    ->                          FROM empleados
    ->                          WHERE  dep_no= (SELECT dep_no
    ->                                     FROM empleados
    ->                                     WHERE apellido='JIMENEZ')
    ->                          GROUP BY dep_no);

+--------+----------+
| emp_no | apellido |
+--------+----------+
|   7876 | GIL      |
+--------+----------+
1 row in set (0.00 sec)
```

3 - Listar el empleado que tiene el mayor salario de cada departamento

```
 mysql> SELECT dep_no, emp_no, apellido
     -> FROM empleados
     -> WHERE (dep_no, salario) IN (SELECT dep_no, MAX(salario)
     ->                             FROM empleados
     ->                             GROUP BY dep_no)
     -> ORDER BY dep_no;
 +--------+--------+----------+
 | dep_no | emp_no | apellido |
 +--------+--------+----------+
 |     10 |   7839 | REY      |
 |     20 |   7876 | GIL      |
```

```
|     30 |   7698 | GARRIDO  |
+--------+--------+----------+
3 rows in set (0.00 sec)
```

4 - Visualizar los empleados que tienen el mismo jefe y departamento que ALONSO excluido el mismo.

```
mysql> SELECT emp_no, apellido, director, dep_no
    -> FROM empleados
    -> WHERE (director, dep_no) = (SELECT director, dep_no
    ->                               FROM empleados
    ->                               WHERE apellido = 'ALONSO')
    -> AND apellido != 'ALONSO';

+--------+----------+----------+--------+
| emp_no | apellido | director | dep_no |
+--------+----------+----------+--------+
|   7654 | MARTIN   |     7698 |     30 |
|   7844 | CALVO    |     7698 |     30 |
+--------+----------+----------+--------+
2 rows in set (0.00 sec)
```

Ahora para que no suceda lo que en el ejemplo anterior y no se visualice ALONSO, que tiene igual jefe y departamento que el mismo, hemos añadido en la consulta la condición de que el apellido no sea ALONSO

# 4 - Subconsultas en la selección de grupos.

Aunque las subconsultas suelen encontrarse sobre todo en la cláusula `WHERE`, también pueden usarse en la `HAVING` formando parte de la selección del grupo de filas efectuada por dicha cláusula.

## 4.1 – Formato de subconsultas en la cláusula HAVING

Todos los formatos son exactamente iguales a cuando son utilizados en la cláusula `WHERE`

## 4.2 – Ejemplos de subconsultas en la cláusula HAVING

1. Visualizar los departamentos en los que el salario medio de sus empleados sea mayor o igual que la media de todos los salarios de la empresa.

```
mysql> SELECT dep_no "Nº Departamento",AVG(salario)"Salario Medio"
    ->  FROM empleados
    ->  GROUP BY dep_no
    ->  HAVING AVG(salario)>=(SELECT AVG(salario)
    ->                          FROM empleados);

+-----------------+---------------+
| Nº Departamento | Salario Medio |
+-----------------+---------------+
|              10 |   3266.833333 |
+-----------------+---------------+
1 row in set (0.00 sec)
```

2. Visualizar los departamentos que tengan mayor media salarial total (salario + comision) que la mitad de la media salarial total de la empresa.

```
mysql> SELECT dep_no, AVG(salario+IFNULL(comision,0))
    -> FROM empleados
    -> GROUP BY dep_no
    -> HAVING AVG(salario+IFNULL(comision,0)) >
    ->                 (SELECT AVG(salario+IFNULL(comision,0))/2
    ->                       FROM empleados);

+--------+---------------------------------+
| dep_no | AVG(salario+IFNULL(comision,0)) |
+--------+---------------------------------+
|     10 |                     3266.833333 |
|     20 |                     2375.000000 |
|     30 |                     2637.530029 |
+--------+---------------------------------+
3 rows in set (0.03 sec)
```

3. Visualizar el departamento con menos presupuesto asignado para pagar el salario de sus empleados

```
mysql> SELECT dep_no "Departamento",
              SUM(salario) "Mayor Presupuesto"
    -> FROM empleados
    -> GROUP BY dep_no
    -> HAVING SUM(salario) = (SELECT SUM(salario)
    ->                           FROM empleados
    ->                           GROUP BY dep_no
    ->                           ORDER BY 1
    ->                           LIMIT 1);

+--------------+-------------------+
| Departamento | Mayor Presupuesto |
+--------------+-------------------+
|           20 |           4750.00 |
+--------------+-------------------+
1 row in set (0.00 sec)
```

# 5 - Subconsultas anidadas.

## 5.1 – Anidación de subconsultas

Cuando una subconsulta forma parte de una condición de selección en una cláusula `WHERE` o `HAVING` de otra subconsulta se dice que es una *subconsulta anidada.*

Las subconsultas se pueden anidar en varias cláusulas a la vez y en varios niveles. Esta posibilidad de anidamiento es lo que le da potencia a la instrucción select.

## 5.2 – Ejemplos de subconsultas anidadas

1- Visualizar el número y el nombre del departamento con más personal de oficio `VENDEDOR`.

```
mysql> SELECT dep_no "Nº Departamento", dnombre Departamento
    ->    FROM departamentos
    ->    WHERE dep_no=(SELECT dep_no
    ->                  FROM empleados
    ->                  WHERE oficio = 'VENDEDOR'
    ->                  GROUP BY dep_no
    ->                  HAVING COUNT(*)=(SELECT COUNT(*)
    ->                                   FROM empleados
    ->                                   WHERE oficio = 'VENDEDOR'
    ->                                   GROUP BY dep_no
    ->                                   ORDER BY 1 DESC
    ->                                   LIMIT 1));

+-----------------+--------------+
| Nº Departamento | Departamento |
+-----------------+--------------+
|              30 | VENTAS       |
+-----------------+--------------+
1 row in set (0.03 sec)
```

2 - Visualizar los datos, número, nombre y localidad, del departamento donde trabaja el empleado más antiguo con el mismo oficio que GIL

```
mysql> SELECT dep_no, dnombre, localidad
    -> FROM departamentos
    -> WHERE dep_no=(SELECT dep_no
    ->               FROM empleados
    ->               WHERE (oficio,fecha_alta)=
    ->                            (SELECT oficio, MIN(fecha_alta)
    ->                              FROM empleados
    ->                              WHERE oficio=(SELECT oficio
    ->                                       FROM empleados
    ->                                       WHERE apellido='GIL')
    ->                              GROUP BY oficio));

+--------+---------------+-----------+
| dep_no | dnombre       | localidad |
+--------+---------------+-----------+
|     20 | INVESTIGACION | VALENCIA  |
+--------+---------------+-----------+
1 row in set (0.00 sec)
```

# 6 – Subconsultas correlacionadas

En una subconsulta podemos hacer referencias a las columnas de la tabla de la consulta. Cuando los nombres de columnas que aparecen en una subconsulta son nombres de columnas de la consulta principal o de otra subconsulta más externa, caso de las anidadas, se dice que son *referencias externas* y la *subconsulta* que es *correlacionada.*

## 6.1 – Correlación entre consultas

Las subconsultas correlacionadas funcionan de la siguiente forma: cada vez que se procesa una nueva fila en la consulta principal para decir si esa fila se selecciona o no, se ejecuta la subconsulta. En esa subconsulta podemos hacer referencia a las columnas de la consulta y los valores serán los de la fila con la que estamos trabajando en ese momento

Por ejemplo:
Visualizar los empleados que ganan más salario que la media de la empresa

```
SELECT dep_no "Nº Departamento", oficio, salario
FROM empleados
WHERE salario>(SELECT AVG(salario)
               FROM empleados);
```

La tabla empleados se utiliza en la subconsulta para hallar el salario medio de la empresa y en la consulta para comprobar las filas que cumplen que el salario de ese empleado sea mayor que el salario medio calculado en la subconsulta.

Ahora supongamos que lo queremos modificar para que se visualicen los empleados que ganan más que la media de **su departamento**. En la subconsulta queremos hallar la media del departamento de cada empleado, es decir del departamento correspondiente al valor del campo dep_no en esa fila en la consulta. Por ejemplo si el primer empleado es del departamento 20 debemos calcular la media del del dep_no=20 para saber si el empleado gana mas que esa media, y si el siguiente empleado es del departamento 10 ahora deberemos calcular la media del dep_no=10.
Debemos referirnos a los valores de las columnas de la consulta dentro de la subconsulta y como son sobre la misma tabla tenemos que poner un alias para diferenciarlas.

En una subconsulta correlacionada si coincide el nombre de una columna de una referencia externa con el nombre de alguna columna de la tabla que está siendo seleccionada en la subconsulta, se deberá anteponer el nombre de la tabla externa para diferenciarlas. Si las tablas son la misma se deberá asignar un alias para diferenciarlas.

Hay dos posibilidades:

a) Poner un alias en ambas

```
SELECT e1.dep_no "Nº Departamento", e1.oficio, salario
FROM empleados e1
WHERE e1.salario>(SELECT AVG(e2.salario)
               FROM empleados e2
               WHERE e2.dep_no = e1.dep_no);
```

b) Poner un alias en la tabla de la consulta

```
SELECT dep_no "Nº Departamento", oficio, salario
FROM empleados e1
WHERE salario>(SELECT AVG(salario)
               FROM empleados
               WHERE dep_no = e1.dep_no);
```

Esta segunda opción es posible porque dentro de una subconsulta, si no se le indica nada, supone que los nombres de las columnas corresponden a esa subconsulta. Para referenciar nombre externos es necesario anteponer el nombre de la tabla de la consulta y si ambas, la consulta y la subconsulta son sobre la misma tabla, es imprescindible el alias.

## 6.2 – Ejemplos de subconsultas correlacionadas

1. Visualizar el número de departamento, el oficio y el salario de los empleados con mayor salario de cada departamento.

```
mysql> SELECT dep_no "Nº Departamento", oficio, salario
    ->  FROM empleados e1
    ->  WHERE salario = (SELECT MAX(salario)
    ->                     FROM empleados e2
    ->                     WHERE e1.dep_no=e2.dep_no);

+-----------------+------------+---------+
| Nº Departamento | oficio     | salario |
+-----------------+------------+---------+
|              30 | DIRECTOR   | 3850.12 |
|              10 | PRESIDENTE | 6000.00 |
|              20 | ANALISTA   | 3350.00 |
+-----------------+------------+---------+
3 rows in set (0.00 sec)
```

2 - Visualizar el empleado más antiguo de cada oficio

```
mysql> SELECT oficio, emp_no, apellido, fecha_alta
    ->  FROM empleados e1
    ->  WHERE fecha_alta = (SELECT MAX(fecha_alta)
    ->                       FROM empleados
    ->                       WHERE oficio=e1.oficio);

+------------+--------+----------+------------+
| oficio     | emp_no | apellido | fecha_alta |
+------------+--------+----------+------------+
| VENDEDOR   |   7654 | MARTIN   | 1981-09-28 |
| DIRECTOR   |   7782 | MARTINEZ | 1981-06-09 |
| PRESIDENTE |   7839 | REY      | 1981-11-17 |
| ANALISTA   |   7876 | GIL      | 1982-05-06 |
| EMPLEADO   |   7900 | JIMENEZ  | 1983-03-24 |
+------------+--------+----------+------------+
5 rows in set (0.02 sec)
```

3 - Visualizar los empleados que tienen el menor salario de cada departamento

```
mysql> SELECT oficio, emp_no, apellido, salario
    ->  FROM empleados e1
    ->  WHERE salario = (SELECT MIN(salario)
    ->                     FROM empleados
    ->                     WHERE dep_no=e1.dep_no);


+----------+--------+----------+---------+
| oficio   | emp_no | apellido | salario |
+----------+--------+----------+---------+
| VENDEDOR |   7499 | ALONSO   | 1400.00 |
| EMPLEADO |   7521 | LOPEZ    | 1350.50 |
| EMPLEADO |   7900 | JIMENEZ  | 1400.00 |
+----------+--------+----------+---------+
3 rows in set (0.00 sec)
```

# TEMA 8. CONSULTAS MULTITABLA

## 1 - Multiplicaciones de tablas.

Hasta ahora, las órdenes SQL que hemos utilizado están basadas en una única tabla, pero a menudo es necesario utilizar datos procedentes de dos o más tablas de la base de datos.

Para poder acceder a dos o más tablas de una base de datos, SQL genera internamente una tabla en la que cada fila de una tabla se combina con todas y cada una de las filas de las demás tablas indicadas. Esta operación es el producto cartesiano de las tablas que se están accediendo y la tabla resultante contiene todas las columnas de todas las tablas que se han multiplicado.

Pueden unirse tantas tablas como se desee (aunque la experiencia aconseja que no sean muchas por optimización).

Comenzaremos con el formato más sencillo de multiplicación de tablas, reseñando solo las cláusulas significativas para realizar el producto.
Pueden ser utilizadas todas las cláusulas de selección vistas anteriormente y al final indicaremos el formato completo.

### 1.1 – Formato de la multiplicación de tablas

El formato más senillo para realizar un producto de tablas es:

SELECT [ALL/DISTINCT] *ExpresionColumna* [,*ExpresionColumna .....]*
**FROM *NombreTabla [AliasTabla] [ , NombreTabla [AliasTabla] ....]***

*Notación: el nombre de tabla puede aparecer una o varias veces, separados por comas.*

En la SELECT pueden seleccionarse columnas de ambas tablas. Si hay columnas con el mismo nombre en las distintas tablas de la FROM, deben identificarse como NombreTabla.NombreColumna o AliasTabla.NombreColumna. Para no tener que escribir siempre el nombre de la tabla, por comodidad, se suele utilizar un alias para cada tabla eligiendo un nombre corto (1 o 2 caracteres) que identifique a cada tabla. En algún caso, que veremos más adelante, este alias será imprescindible.

## 1.2 – Ejemplos de la multiplicación de tablas

Veremos la salida que produce la multiplicación, o producto cartesiano, de dos tablas con un ejemplo para obtener todos los empleados, indicando su número de empleado, su apellido, el nombre de su departamento y su localidad de este.

```
mysql> SELECT emp_no "Nº EMPLEADO", apellido "APELLIDO",
    ->      dnombre DEPARTAMENTO, localidad "LOCALIDAD"
    -> FROM empleados, departamentos;

+-------------+----------+---------------+-----------+
| Nº EMPLEADO | APELLIDO | DEPARTAMENTO  | LOCALIDAD |
+-------------+----------+---------------+-----------+
|        7499 | ALONSO   | CONTABILIDAD  | BARCELONA |
|        7499 | ALONSO   | INVESTIGACION | VALENCIA  |
|        7499 | ALONSO   | VENTAS        | MADRID    |
|        7499 | ALONSO   | PRODUCCION    | SEVILLA   |
|        7521 | LOPEZ    | CONTABILIDAD  | BARCELONA |
|        7521 | LOPEZ    | INVESTIGACION | VALENCIA  |
|        7521 | LOPEZ    | VENTAS        | MADRID    |
|        7521 | LOPEZ    | PRODUCCION    | SEVILLA   |
|        7654 | MARTIN   | CONTABILIDAD  | BARCELONA |
|        7654 | MARTIN   | INVESTIGACION | VALENCIA  |
|        7654 | MARTIN   | VENTAS        | MADRID    |
|        7654 | MARTIN   | PRODUCCION    | SEVILLA   |
|        7698 | GARRIDO  | CONTABILIDAD  | BARCELONA |
|        7698 | GARRIDO  | INVESTIGACION | VALENCIA  |
|        7698 | GARRIDO  | VENTAS        | MADRID    |
|        7698 | GARRIDO  | PRODUCCION    | SEVILLA   |
|        7782 | MARTINEZ | CONTABILIDAD  | BARCELONA |
|        7782 | MARTINEZ | INVESTIGACION | VALENCIA  |
|        7782 | MARTINEZ | VENTAS        | MADRID    |
|        7782 | MARTINEZ | PRODUCCION    | SEVILLA   |
|        7839 | REY      | CONTABILIDAD  | BARCELONA |
|        7839 | REY      | INVESTIGACION | VALENCIA  |
|        7839 | REY      | VENTAS        | MADRID    |
|        7839 | REY      | PRODUCCION    | SEVILLA   |
|        7844 | CALVO    | CONTABILIDAD  | BARCELONA |
|        7844 | CALVO    | INVESTIGACION | VALENCIA  |
|        7844 | CALVO    | VENTAS        | MADRID    |
|        7844 | CALVO    | PRODUCCION    | SEVILLA   |
|        7876 | GIL      | CONTABILIDAD  | BARCELONA |
|        7876 | GIL      | INVESTIGACION | VALENCIA  |
|        7876 | GIL      | VENTAS        | MADRID    |
|        7876 | GIL      | PRODUCCION    | SEVILLA   |
|        7900 | JIMENEZ  | CONTABILIDAD  | BARCELONA |
|        7900 | JIMENEZ  | INVESTIGACION | VALENCIA  |
|        7900 | JIMENEZ  | VENTAS        | MADRID    |
|        7900 | JIMENEZ  | PRODUCCION    | SEVILLA   |
+-------------+----------+---------------+-----------+
36 rows in set (0.00 sec)
```

Cada empleado de la tabla de `empleados` aparece tantas veces como departamentos hay en la tabla de `departamentos`, con los correspondientes valores de cada de las filas de la tabla `departamentos`.
Se obtienen, por tanto, 9 x 4= 36 filas.

# 2 - Composiciones o combinaciones simples (JOIN)

Acabamos de ver, en el ejemplo anterior, que la salida producida por la multiplicación de las tablas de `empleados` y de `departamentos` no tiene mucho sentido y poca aplicación.

La solución más adecuada del ejemplo sería enlazar la tabla `empleados` con la tabla `departamentos` de forma que para cada empleado de la tabla `empleados` solo tengamos de la tabla `departamentos` la fila correspondiente a su departamento.
Es decir:

```
mysql> SELECT emp_no "Nº EMPLEADO", apellido "APELLIDO",
              dnombre DEPARTAMENTO, localidad "LOCALIDAD"
    ->  FROM empleados, departamentos
    ->  WHERE empleados.dep_no = departamentos.dep_no;

+-------------+----------+---------------+-----------+
| Nº EMPLEADO | APELLIDO | DEPARTAMENTO  | LOCALIDAD |
+-------------+----------+---------------+-----------+
|        7521 | LOPEZ    | CONTABILIDAD  | BARCELONA |
|        7782 | MARTINEZ | CONTABILIDAD  | BARCELONA |
|        7839 | REY      | CONTABILIDAD  | BARCELONA |
|        7876 | GIL      | INVESTIGACION | VALENCIA  |
|        7900 | JIMENEZ  | INVESTIGACION | VALENCIA  |
|        7499 | ALONSO   | VENTAS        | MADRID    |
|        7654 | MARTIN   | VENTAS        | MADRID    |
|        7698 | GARRIDO  | VENTAS        | MADRID    |
|        7844 | CALVO    | VENTAS        | MADRID    |
+-------------+----------+---------------+-----------+
9 rows in set (0.00 sec)
```

Hemos visto que la salida que produce el producto de las tablas es cada fila de una tabla combinadas con todas las de otra tabla. Esta información no suele ser la deseada. Aparecen las ***composiciones o combinaciones de tablas*** que nos proporcionan la misma información pero filtrada. Una composición o combinación (join) consiste en aplicar una condición de selección a las filas obtenidas de la multiplicación de las tablas sobre las que se está realizando una consulta.

## 2.1 – Formato de la combinación de tablas

SELECT [ALL/DISTINCT] *ExpresionColumna* [,*ExpresionColumna .....]*

**FROM *NombreTabla [AliasTabla] [ , NombreTabla [AliasTabla].....]***

**[WHERE *CondicionComposicion* ]**

*Notación: la cláusula WHERE es opcional y por eso aparece entre corchetes.*

donde ***CondicionComposicion*** .....es una condición que selecciona las filas de la composición de las tablas.

La condición de selección o criterio de emparejamiento para las tablas también se denomina *condición o criterio de composición.*

Dependiendo de la condición de composición tendremos las combinaciones naturales, basadas en la igualdad, o las combinaciones basadas en la desigualdad.

Existen varios tipos según sea esta condición de composición

- ***Composición natural***: es la más sencilla y natural y es aquella en que la condición de selección se establece con el operador de igualdad entre las columnas que deban coincidir exactamente en tablas diferentes.
- ***Composición basada en desigualdad***: es menos utilizada y consiste en que la condición de selección no sea una igualdad.

# 3- Composición o combinación natural

Como hemos dicho la composición o combinación natural es la más sencilla y natural. Es aquella en que la condición de selección se establece con el operador de igualdad entre las columnas que deben coincidir exactamente en tablas diferentes.

Suele utilizarse para unir tablas en las que hay una relación a través de las claves ajenas, uniendo una tabla con su referenciada.

Veamos con el ejemplo porqué es la forma más natural:

```
mysql> SELECT emp_no "Nº EMPLEADO", apellido "APELLIDO",
             dnombre DEPARTAMENTO, localidad "LOCALIDAD"
    ->  FROM empleados, departamentos
    ->  WHERE empleados.dep_no = departamentos.dep_no;

+-------------+----------+---------------+-----------+
| Nº EMPLEADO | APELLIDO | DEPARTAMENTO  | LOCALIDAD |
+-------------+----------+---------------+-----------+
|        7521 | LOPEZ    | CONTABILIDAD  | BARCELONA |
|        7782 | MARTINEZ | CONTABILIDAD  | BARCELONA |
|        7839 | REY      | CONTABILIDAD  | BARCELONA |
|        7876 | GIL      | INVESTIGACION | VALENCIA  |
|        7900 | JIMENEZ  | INVESTIGACION | VALENCIA  |
|        7499 | ALONSO   | VENTAS        | MADRID    |
|        7654 | MARTIN   | VENTAS        | MADRID    |
|        7698 | GARRIDO  | VENTAS        | MADRID    |
|        7844 | CALVO    | VENTAS        | MADRID    |
+-------------+----------+---------------+-----------+
9 rows in set (0.00 sec)
```

Si obtenemos el producto cartesiano de las tablas empleados por departamentos para cada empleado obtenemos la combinación con todas las filas de departamento. Pero la única que nos interesará será la del departamento al que pertenezca el empleado, es decir la que cumpla la condición: **departamento.dep_no = empleado.dep_no**

SQL resuelve el anterior ejercicio con el siguiente proceso:

- La cláusula FROM genera todas las combinaciones posibles de filas de la tabla de *empleados* (9 filas) por las de la tabla de *departamentos* (4 filas), resultando una tabla producto de 4x9=36 filas.

- La cláusula WHERE selecciona únicamente aquellas filas de la tabla producto donde coinciden los números de departamento, que necesitan el nombre de la tabla o el alias por tener el mismo nombre en ambas tablas. En total se han seleccionado 9 filas y las 27 restantes se eliminan.
- La sentencia SELECT visualiza las columnas especificadas de las tablas producto para las filas seleccionadas.

SQL no exige que las columnas de emparejamiento estén relacionadas como clave primaria y clave ajena, aunque suele ser lo habitual. Pueden servir cualquier par de columnas de dos tablas, siempre que tengan tipos de datos comparables.

## 3.1 – Formato de la combinación natural de tablas

Es el mismo de la combinación o composición pero donde la condición de composición es una igualdad entre campos de diferentes tablas:

**[NombreTabla1.] NombreColumna1 = [NombreTabla2.] NombreColumna2**

Los nombres de las columnas necesitarán anteponer el nombre de la tabla o el alias si el nombre es el mismo en ambas tablas.

## 3.2 – Ejemplos de combinación natural de tablas

1. Obtener los distintos departamentos existentes en la tabla de *empleados*.

```
mysql> SELECT DISTINCT d.dep_no "Nº Departamento",
                  d.dnombre Departamento
    -> FROM empleados e, departamentos d
    ->  WHERE e.dep_no = d.dep_no;

+-----------------+---------------+
| Nº Departamento | Departamento  |
+-----------------+---------------+
|              10 | CONTABILIDAD  |
|              20 | INVESTIGACION |
|              30 | VENTAS        |
+-----------------+---------------+
3 rows in set (0.02 sec)
```

2. Mostrar los siguientes datos relativos a empleados: número, apellido, nombre de departamento y localidad.

```
mysql> SELECT emp_no "Nº Empleado", apellido, dnombre, localidad
    ->          FROM empleados e, departamentos d
    ->          WHERE e.dep_no = d.dep_no;

+-------------+----------+---------------+-----------+
| Nº Empleado | apellido | dnombre       | localidad |
```

```
+-------------+----------+---------------+-----------+
|        7521 | LOPEZ    | CONTABILIDAD  | BARCELONA |
|        7782 | MARTINEZ | CONTABILIDAD  | BARCELONA |
|        7839 | REY      | CONTABILIDAD  | BARCELONA |
|        7876 | GIL      | INVESTIGACION | VALENCIA  |
|        7900 | JIMENEZ  | INVESTIGACION | VALENCIA  |
|        7499 | ALONSO   | VENTAS        | MADRID    |
|        7654 | MARTIN   | VENTAS        | MADRID    |
|        7698 | GARRIDO  | VENTAS        | MADRID    |
|        7844 | CALVO    | VENTAS        | MADRID    |
+-------------+----------+---------------+-----------+
9 rows in set (0.02 sec)
```

# 4 - Composiciones o combinaciones basadas en desigualdad

La condición de selección que establezcamos para componer o combinar tablas no tiene por qué ser siempre mediante el operador aritmético de igualdad, aunque su uso sea el más frecuente. SQL permite utilizar cualquier operador aritmético de comparación.
Estas composiciones son aquellas en las que el operador de la condición de selección NO es la igualdad.

## 4.1 – Formato de combinaciones basadas en desigualdad

Es el mismo de la combinación o composición pero la condición de composición es diferente de la igualdad entre campos de diferentes tablas.

## 4.2 – Ejemplos de combinaciones basadas en desigualdad

1. Listar los empleados de los departamentos diferentes al de VENTAS.

```
mysql> SELECT e.emp_no "NºEmpleado",e.apellido "Apellido"
    ->  FROM empleados e,departamentos d
->  WHERE d.dnombre = 'VENTAS' AND e.dep_no != d.dep_no;

+------------+----------+
| NºEmpleado | Apellido |
+------------+----------+
|       7521 | LOPEZ    |
|       7782 | MARTINEZ |
|       7839 | REY      |
|       7876 | GIL      |
|       7900 | JIMENEZ  |
+------------+----------+
5 rows in set (0.00 sec)
```

2. Listar los empleados de departamentos con códigos mayores que el código del departamento de contabilidad.

```
mysql> SELECT e.emp_no "NºEmpleado",e.apellido
    ->  FROM empleados e,departamentos d
```

```
->  WHERE d.dnombre = 'CONTABILIDAD' AND e.dep_no>d.dep_no;

+------------+----------+
| NºEmpleado | apellido |
+------------+----------+
|       7499 | ALONSO   |
|       7654 | MARTIN   |
|       7698 | GARRIDO  |
|       7844 | CALVO    |
|       7876 | GIL      |
|       7900 | JIMENEZ  |
+------------+----------+
6 rows in set (0.00 sec)
```

3. Listar los empleados de departamentos con códigos menores que el código del departamento de Barcelona

```
mysql> SELECT e.emp_no "NºEmpleado",e.apellido
    -> FROM empleados e,departamentos d
    -> WHERE d.localidad = 'BARCELONA'
             AND e.dep_no > d.dep_no;

+------------+----------+
| NºEmpleado | apellido |
+------------+----------+
|       7499 | ALONSO   |
|       7654 | MARTIN   |
|       7698 | GARRIDO  |
|       7844 | CALVO    |
|       7876 | GIL      |
|       7900 | JIMENEZ  |
+------------+----------+
6 rows in set (0.00 sec)
```

# 5 - Composiciones o combinaciones de una tabla consigo misma

Si desde una fila de una tabla podemos acceder a otra fila de la misma tabla, duplicando la tabla, estamos realizando una composición o combinación de una tabla consigo misma.

## 5.1 - Formato de una combinación de una tabla consigo misma.

Es el mismo de la combinación o composición pero las tablas del producto son las mismas tablas.

En este caso, como ambas tablas se llaman igual, es necesario obligatoriamente un alias para diferenciar las columnas de cada tabla.

## 5.2 - Ejemplos de una combinación de una tabla consigo misma

1. Obtener la lista de los empleados con los nombres de sus directores.

```
mysql>  SELECT e1.emp_no "NºEmpleado",
            e1.apellido "Nombre Empleado",
            e1.director "NºDirector",
            e2.apellido "Nombre Director"
    ->  FROM empleados e1,empleados e2
    ->  WHERE e1.director=e2.emp_no;

+------------+-----------------+------------+-----------------+
| NºEmpleado | Nombre Empleado | NºDirector | Nombre Director |
+------------+-----------------+------------+-----------------+
|       7499 | ALONSO          |       7698 | GARRIDO         |
|       7521 | LOPEZ           |       7782 | MARTINEZ        |
|       7654 | MARTIN          |       7698 | GARRIDO         |
|       7698 | GARRIDO         |       7839 | REY             |
|       7782 | MARTINEZ        |       7839 | REY             |
|       7844 | CALVO           |       7698 | GARRIDO         |
|       7876 | GIL             |       7782 | MARTINEZ        |
|       7900 | JIMENEZ         |       7782 | MARTINEZ        |
+------------+-----------------+------------+-----------------+
8 rows in set (0.00 sec)
```

Cada empleado de la tabla tiene una columna para su número de empleado *(emp_no)* y otra para el número de empleado de su director *(director).* A partir del dato de la columna *director* de un empleado se puede acceder a otro empleado que contenga el mismo dato en su columna *emp_no.* Las dos filas de la tabla se están relacionando a través de las columnas *director* y *emp_no.*
El uso de alias es obligado por tratarse de la misma tabla y coincidir los nombres de las columnas.

2. Obtener los jefes de los empleados cuyo oficio sea el de VENDEDOR.

```
mysql> SELECT e1.emp_no "NºEmpleado", e1.apellido, e1.oficio,
    ->        e1.director "NºDirector", e2.apellido "Nombre Director"
    ->  FROM empleados e1, empleados e2
    ->  WHERE e1.director = e2.emp_no AND e1.oficio = 'VENDEDOR';

+------------+----------+----------+-------------+-----------------+
| NºEmpleado | apellido | oficio   |  NºDirector | Nombre Director |
+------------+----------+----------+-------------+-----------------+
|       7499 | ALONSO   | VENDEDOR |       7698  | GARRIDO         |
|       7654 | MARTIN   | VENDEDOR |       7698  | GARRIDO         |
|       7844 | CALVO    | VENDEDOR |       7698  | GARRIDO         |
+------------+----------+----------+-------------+-----------------+
3 rows in set (0.00 sec)
```

# 6 - Composiciones o combinaciones externas (OUTER JOIN)

Cuando se realiza una composición o combinación de tablas estableciendo una determinada relación entre sus columnas, puede ocurrir que no se emparejen todas las filas que debieran por faltar correspondencia entre algunas de ellas. Esto se debe a que existen filas en alguna tabla que no tienen correspondencia en la otra tabla y al aplicar la condición de selección de la composición no se seleccionarán.

Por ejemplo

```
mysql> SELECT dnombre Departamento, localidad, emp_no, apellido
    ->  FROM empleados e,departamentos d
```

```
    ->  WHERE e.dep_no=d.dep_no;
+---------------+-----------+--------+----------+
| Departamento  | localidad | emp_no | apellido |
+---------------+-----------+--------+----------+
| CONTABILIDAD  | BARCELONA |   7521 | LOPEZ    |
| CONTABILIDAD  | BARCELONA |   7782 | MARTINEZ |
| CONTABILIDAD  | BARCELONA |   7839 | REY      |
| INVESTIGACION | VALENCIA  |   7876 | GIL      |
| INVESTIGACION | VALENCIA  |   7900 | JIMENEZ  |
| VENTAS        | MADRID    |   7499 | ALONSO   |
| VENTAS        | MADRID    |   7654 | MARTIN   |
| VENTAS        | MADRID    |   7698 | GARRIDO  |
| VENTAS        | MADRID    |   7844 | CALVO    |
+---------------+-----------+--------+----------+
9 rows in set (0.00 sec)
```

Aquellos departamentos que no tengan empleados no aparecerán porque para esos departamentos no se cumplirá la igualdad empleados.dep_no = departamentos.dep_no

Es aconsejable que la salida, obtenida por una consulta en la que se pudiera presentar esta posibilidad, muestre todas las filas, aunque algunas con falta de información. Para conseguir este resultado se utiliza la composición o combinación externa (OUTER JOIN).

## 6.1 – Formato de combinaciones externas

SELECT [ALL/DISTINCT] ExpresionColumna [,ExpresionColumna .....]
**FROM NombreTabla [AliasTabla]**
**{LEFT|RIGHT [OUTER] JOIN NombreTabla [AliasTabla].....]**
**ON CondicionComposicion**

donde **LEFT|RIGHT [OUTER] JOIN…….....** indica que es un join externo y si la extensión del producto de las tablas se quiere realizar por la izquierda o por la derecha

**CondicionComposicion ……..** es la misma condición de composición anterior, pero escrita aquí en lugar de en la cláusula `WHERE`

El funcionamiento de un join externo es el siguiente:

- `LEFT JOIN`: join donde se obtienen todas las filas de la tabla de la izquierda, aunque no tenga correspondencia en la tabla de la derecha.
  Realiza el producto cartesiano de las tablas que se le indican, aplica la condición de composición (expresada en la cláusula `ON`) al resultado de este producto cartesiano y añade, por cada fila de la tabla de la izquierda que no tenga correspondencia en la tabla de la derecha, una fila con los valores de la tabla de la izquierda y en la tabla de la derecha valores `NULL` en todas las columnas.
- `RIGHT JOIN`: join donde se obtienen todas las filas de la tabla de la derecha, aunque no tengan correspondencia en la tabla de la izquierda.
  Realiza el producto cartesiano de las tablas que se le indican, aplica la condición de composición (expresada en la cláusula `ON`) al resultado de este producto cartesiano y añade, por cada fila de la tabla de la derecha que no tenga correspondencia en la tabla de la

izquierda, una fila con los valores de la tabla de la derecha y en la tabla de la izquierda valores NULL en todas las columnas.

Por ejemplo si queremos visualizar los datos de los departamentos y de sus empleados, visualizando también los departamentos que no tengan empleados.

```
mysql> SELECT dnombre Departamento, localidad,
        emp_no "Nº empleado", apellido
    ->  FROM departamentos d LEFT JOIN empleados e
    ->  ON d.dep_no=e.dep_no;

+---------------+-----------+------------+----------+
| Departamento  | localidad | Nºempleado | apellido |
+---------------+-----------+------------+----------+
| CONTABILIDAD  | BARCELONA |       7521 | LOPEZ    |
| CONTABILIDAD  | BARCELONA |       7782 | MARTINEZ |
| CONTABILIDAD  | BARCELONA |       7839 | REY      |
| INVESTIGACION | VALENCIA  |       7876 | GIL      |
| INVESTIGACION | VALENCIA  |       7900 | JIMENEZ  |
| VENTAS        | MADRID    |       7499 | ALONSO   |
| VENTAS        | MADRID    |       7654 | MARTIN   |
| VENTAS        | MADRID    |       7698 | GARRIDO  |
| VENTAS        | MADRID    |       7844 | CALVO    |
| PRODUCCION    | SEVILLA   |       NULL | NULL     |
+---------------+-----------+------------+----------+
10 rows in set (0.00 sec)
```

Aparecerán todas las filas de la tabla DEPARTAMENTOS, tanto si tienen correspondencia en la tabla EMPLEADOS como si no la tienen. Los departamentos que no tengan empleados también aparecerían. El departamento de PRODUCCIÓN no tiene ningún empleado asignado y se añade una fila en la tabla empleados con todos los campo con valor NULL en correspondencia.

Obtendremos el mismo resultado si cambiamos LEFT por RIGHT el orden de las tablas

```
mysql> SELECT dnombre Departamento, localidad,
    ->          emp_no "Nº empleado", apellido
    ->     FROM empleados e RIGHT JOIN departamentos d
    ->     ON d.dep_no=e.dep_no;

+---------------+-----------+-------------+----------+
| Departamento  | localidad | Nº empleado | apellido |
+---------------+-----------+-------------+----------+
| CONTABILIDAD  | BARCELONA |        7521 | LOPEZ    |
| CONTABILIDAD  | BARCELONA |        7782 | MARTINEZ |
| CONTABILIDAD  | BARCELONA |        7839 | REY      |
| INVESTIGACION | VALENCIA  |        7876 | GIL      |
| INVESTIGACION | VALENCIA  |        7900 | JIMENEZ  |
| VENTAS        | MADRID    |        7499 | ALONSO   |
| VENTAS        | MADRID    |        7654 | MARTIN   |
| VENTAS        | MADRID    |        7698 | GARRIDO  |
| VENTAS        | MADRID    |        7844 | CALVO    |
| PRODUCCION    | SEVILLA   |        NULL | NULL     |
+---------------+-----------+-------------+----------+
10 rows in set (0.00 sec)
```

Hay casos en los que hacer el OUTER JOIN obtendremos el mismo resultado con una combinación natural, ya que no hay filas sin correspondencia en la tabla de la correspondencia.

```
mysql> SELECT emp_no "Nºempleado", apellido,
            dnombre "Departamento", localidad
    ->  FROM departamentos d RIGHT JOIN empleados e
    ->  ON d.dep_no=e.dep_no;

+------------+----------+---------------+-----------+
| Nºempleado | apellido | Departamento  | localidad |
+------------+----------+---------------+-----------+
|       7499 | ALONSO   | VENTAS        | MADRID    |
|       7521 | LOPEZ    | CONTABILIDAD  | BARCELONA |
|       7654 | MARTIN   | VENTAS        | MADRID    |
|       7698 | GARRIDO  | VENTAS        | MADRID    |
|       7782 | MARTINEZ | CONTABILIDAD  | BARCELONA |
|       7839 | REY      | CONTABILIDAD  | BARCELONA |
|       7844 | CALVO    | VENTAS        | MADRID    |
|       7876 | GIL      | INVESTIGACION | VALENCIA  |
|       7900 | JIMENEZ  | INVESTIGACION | VALENCIA  |
+------------+----------+---------------+-----------+
9 rows in set (0.00 sec)
```

Aparecerán todas las filas de la tabla EMPLEADOS, tanto si tienen correspondencia en la tabla DEPARTAMENTOS como si no. Los empleados que no tuviesen departamento asignado también aparecerían. En este ejemplo como todos los empleados tienen departamento asignado el resultado es el mismo.

## 6.2 – Ejemplos de combinaciones externas

1. Obtener los departamentos con su nombre y localidad y el número de empleados trabajando en ellos, incluyendo los que no tienen empleados.

```
mysql> SELECT dnombre Departamento, localidad, COUNT(emp_no)
    ->  FROM departamentos d LEFT JOIN empleados e
    ->  ON d.dep_no=e.dep_no
    ->  GROUP BY dnombre, localidad;

+---------------+-----------+---------------+
| Departamento  | localidad | COUNT(emp_no) |
+---------------+-----------+---------------+
| CONTABILIDAD  | BARCELONA |             3 |
| INVESTIGACION | VALENCIA  |             2 |
| PRODUCCION    | SEVILLA   |             0 |
| VENTAS        | MADRID    |             4 |
+---------------+-----------+---------------+
4 rows in set (0.00 sec)
```

2. Obtener la lista de empleados con los nombres de sus directores, incluyendo al PRESIDENTE. (Ejemplo en autocomposiciones) .

```
mysql> SELECT e1.apellido "Nombre Empleado",
            e2.apellido "Nombre Director"
    -> FROM empleados e1 LEFT JOIN empleados e2
    -> ON e1.director = e2.emp_no;

+----------+-----------------+
| apellido | Nombre Director |
+----------+-----------------+
| ALONSO   | GARRIDO         |
| LOPEZ    | MARTINEZ        |
| MARTIN   | GARRIDO         |
| GARRIDO  | REY             |
| MARTINEZ | REY             |
| REY      | NULL            |
| CALVO    | GARRIDO         |
| GIL      | MARTINEZ        |
| JIMENEZ  | MARTINEZ        |
+----------+-----------------+
9 rows in set (0.00 sec)
```

# 7 - Composiciones y subconsultas

Hay ocasiones en que una consulta puede resolverse con una composición o combinación (join) de tablas o con una subconsulta.

Si puede solucionarse de ambas formas será preferible hacerlo con una subconsulta. El producto cartesiano es muy costoso pues hay que multiplicar todas las filas de una tabla por todas las de la otra tabla para después seleccionar solo algunas. Con tablas con miles o millones de registros esto es un trabajo muy costoso en tiempo y memoria, que puede resolver con una subconsulta.

En general, si no se necesita visualizar columnas de más de una tabla, se debe utilizar una subconsulta. Solamente si se necesita visualizar columnas de más de una tabla, se usará una composición o combinación.

Vamos a verlo con unos ejemplos:

**1- Con subconsulta:**
Obtener apellido y oficio de los empleados que tienen el mismo oficio y mismo número de departamento que el de INVESTIGACIÓN.

```
mysql> SELECT apellido, oficio
    ->  FROM empleados
    ->  WHERE oficio IN (SELECT oficio
    ->                FROM empleados
    ->                WHERE dep_no IN (SELECT dep_no
    ->                          FROM departamentos
    ->                          WHERE dnombre='INVESTIGACION'));

+----------+----------+
| apellido | oficio   |
+----------+----------+
| LOPEZ    | EMPLEADO |
| GIL      | ANALISTA |
```

```
| JIMENEZ  | EMPLEADO |
+----------+----------+
3 rows in set (0.00 sec)
```

Puede solucionarse con una subconsulta porque solo nos piden visualizar campos de la tabla empleados. Debe solucionarse, por tanto, utilizarse una subconsulta.

**2- Con composición de las tablas:**
Obtener apellido, el oficio y la localidad del departamento de los empleados que tienen el mismo oficio y mismo número de departamento que el de INVESTIGACIÓN.

```
mysql> SELECT apellido, oficio, localidad
    ->  FROM empleados e, departamentos d
    ->  WHERE oficio IN (SELECT oficio
    ->                   FROM empleados
    ->                   WHERE e.dep_no = d.dep_no
                               AND d.dnombre = 'INVESTIGACION');

+----------+----------+-----------+
| apellido | oficio   | localidad |
+----------+----------+-----------+
| GIL      | ANALISTA | VALENCIA  |
| JIMENEZ  | EMPLEADO | VALENCIA  |
+----------+----------+-----------+
2 rows in set (0.00 sec)
```

Es el mismo ejercicio pero, en este ejemplo, nos piden visualizar campos de la tabla empleados y de la tabla departamentos. No puede solucionarse con una subconsulta y debe, por tanto, solucionarse con un producto de ambas tablas.

# 8 - Formato completo de las consultas

Aunque hasta ahora no hemos utilizado todas las cláusulas estudiadas, en la composición de tablas está permitidas todas ellas. Podemos hacer selección de filas, agrupamientos, selección de grupo, ordenación y limitar el número de filas de igual forma que lo hacíamos para consultas con una sola tabla

**SELECT { * | [ALL/DISTINCT] *ExpresionColumna* [*AliasColumna*]**
**[, *ExpresionColumna* [*AliasColumna*]....] }**
**FROM *NombreTabla [AliasTabla] [,NombreTabla [AliasTabla]....]***
**[WHERE { *CondicionSeleccion* | *CondicionComposicion* }]**
**[GROUP BY *ExpresionColumna*|*Posicion [, ExpresionColumna*|*Posicion ...* ]**
**[HAVING *CondicionSelecionGrupos* ] ]**
**[ORDER BY *{ExpresionColumna*|*Posicion* } [ASC|DESC]**
**[,*{ExpresionColumna*|*Posicion* } [ASC|DESC].....] ]**
**[LIMIT [m, ] n ] ;**

# TEMA 9. CONSULTAS DENTRO DE OTRAS INSTRUCCIONES

## 1 - Creación de una tabla a partir de una selección de otra tabla.

En el momento de crear una tabla podemos utilizar la definición de otra ya creada, utilizando toda la definición o aquella parte que no interese.

### 1.1 – Formato para la creación de una tabla a partir de una selección de otra tabla.

```
CREATE TABLE [IF NOT EXISTS] NombreTabla
   [( DefinicionColumna [, DefinicionColumna ...] )]  [ IGNORE|REPLACE ]
 SentenciaSelect
```

*Notación: las definiciones de las columnas son opcionales, por lo que están entre corchetes.*

donde **NombreTabla .....** es el identificador de la nueva tabla que se crea
**SentenciaSelect .....** es cualquier sentencia select
**IGNORE|REPLACE .....** opciones de duplicados en campos únicos

Los nombres de las columnas de la nueva tabla son opcionales. En caso de que no se especifique tomarán los valores de la otra tabla o de los alias correspondientes. Pero debe tenerse cuidado con ello pues si hay expresiones o funciones en la lista del select y no tienen alias ni nuevo nombre, luego estas columnas no pueden referenciarse.

Las opciones de duplicados en campos únicos (claves primarias o unique) permiten indicar que hacer si hay un valor repetido en ese campo. Si no se indica nada se producirá un error. Para evitar esto errores podemos especificar una de las dos opciones `IGNORE` que ignora la fila y no la graba y `REPLACE` que reemplaza la fila por la anterior.

La nueva tabla creada no hereda las `CONSTRAINTS` que tenga asignada la tabla origen. Esto es para dar más flexibilidad al sistema. Si se desea que la nueva tabla tenga constraints deben indicarse en la creación o añadirse posteriormente con `ALTER TABLE`. Es importante recordarlo para que la nueva tabla quede con la definición completa.

Al crear la nueva tabla además se insertan las filas correspondientes de la tabla resultado de la sentencia select en la instrucción de creación.

## 1.2 – Ejemplos de la creación de una tabla a partir de una selección de otra tabla.

**1.** Crear una tabla de vendedores seleccionando éstos de la tabla de empleados.

```
mysql> CREATE TABLE vendedores
    -> SELECT *
    -> FROM empleados
    -> WHERE oficio = 'VENDEDOR';
Query OK, 3 rows affected (0.13 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM vendedores;
+--------+----------+---------+---------+------------+---------+---------+--------+
| EMP_NO | APELLIDO | OFICIO  |DIRECTOR | FECHA_ALTA | SALARIO | COMISION | DEP_NO|
+--------+----------+---------+---------+------------+---------+----------+-------+
|   7499 | ALONSO   | VENDEDOR|    7698 | 1981-02-23 | 1400.00 |  650.00 |  30    |
|   7654 | MARTIN   | VENDEDOR|    7698 | 1981-09-28 | 1500.00 | 1850.00 |  30    |
|   7844 | CALVO    | VENDEDOR|    7698 | 1981-09-08 | 1800.00 |  250.00 |  30    |
+--------+----------+---------+---------+------------+---------+---------+--------+
3 rows in set (0.00 sec)
```

**2.** Crear una nueva tabla sólo con los nombres y números de los departamentos a partir de la tabla ya creada con los mismos.

```
mysql> CREATE TABLE nombres_dep (depart_no INT(4),
                  nombre VArchar(10))
    ->   SELECT dep_no, dnombre
    ->   FROM departamentos;
Query OK, 4 rows affected (0.23 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM nombres_dep;
+--------+---------------+
| dep_no | dnombre       |
+--------+---------------+
|     10 | CONTABILIDAD  |
|     20 | INVESTIGACION |
|     30 | VENTAS        |
|     40 | PRODUCCION    |
+--------+---------------+
4 rows in set (0.00 sec)
```

# 2 - Actualización de una tabla a partir de una subconsulta

Todas las instrucciones de actualización de datos que hemos visto pueden ampliarse con la utilización de subconsultas dentro de ellas.

## 2.1- Inserciones con subconsultas

Podemos insertar en una tabla el resultado de una consulta sobre otra tabla. En este caso normalmente se insertarán varias filas con una sola sentencia. Utilizaremos el siguiente formato:

**INSERT INTO NombreTabla [( NombreColumna [,NombreColumna...] ) ]**
**SELECT FormatoSelect**

*Notación: la lista de columnas en las que insertamos va es opcional, por lo que va entre corchetes.*

En el formato anterior podemos destacar:

- La **lista de columnas es opcional** pero deberá especificarse cuando las columnas que devuelve la consulta no coinciden en número o en orden con las columnas de la tabla destino.
- La **consulta puede ser cualquier comando de selección** válido siempre que exista una correspondencia entre las columnas devueltas y las columnas de la tabla destino, o la lista de columnas.

En este caso existe correspondencia en número y en orden entre las columnas de la tabla destino y las columnas de la selección; por tanto, no hace falta especificar la lista de columnas y el comando requerido será:

```
mysql> INSERT INTO departamentos2
    -> SELECT * FROM departamentos
    -> WHERE LENGTH(dnombre)> 8;
Query OK, 3 rows affected (0.05 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT *
    -> FROM departamentos2;
+--------+---------------+-----------+
| DEP_NO | DNOMBRE       | LOCALIDAD |
+--------+---------------+-----------+
|     10 | CONTABILIDAD  | BARCELONA |
|     20 | INVESTIGACION | VALENCIA  |
|     40 | PRODUCCION    | SEVILLA   |
+--------+---------------+-----------+
3 rows in set (0.00 sec)
```

Si la tabla destino tuviese una estructura diferente deberemos forzar la correspondencia, bien al especificar la lista de selección, bien especificando la lista de columnas, o bien utilizando ambos recursos.

Por ejemplo, supongamos que la tabla destino es *n2dep*

```
mysql> CREATE TABLE n2dept
    -> ( DEP_NO                INT(2),
    ->   NOMBRE               VARCHAR(14),
    ->   CONSTRAINT PK_DEPARTAMENTOS_DEP_NO PRIMARY KEY (DEP_NO)
    -> );
Query OK, 0 rows affected (0.52 sec)
```

En este caso procederemos:

```
mysql> INSERT INTO n2dept
```

```
    -> SELECT dep_no, dnombre FROM departamentos
    -> WHERE LENGTH(dnombre)> 8;
Query OK, 3 rows affected (0.40 sec)
Records: 3  Duplicates: 0  Warnings: 0


mysql> SELECT *
    -> FROM n2dept;

+--------+---------------+
| DEP_NO | NOMBRE        |
+--------+---------------+
|     10 | CONTABILIDAD  |
|     20 | INVESTIGACION |
|     40 | PRODUCCION    |
+--------+---------------+
3 rows in set (0.00 sec)
```

## 2.2 – Modificaciones con subconsultas

En ocasiones la condición que deben cumplir las filas que deseamos modificar implica realizar una subconsulta a otras tablas. En estos casos se incluirá la subconsulta en la condición y los operadores necesarios tal como estudiamos en el apartado correspondiente del tema SUBCONSULTAS. La subconsulta estará en la cláusula `WHERE`.
Esta subconsulta tiene las siguientes limitaciones:

- La tabla destino no puede aparecer en la consulta.
- No se puede incluir una cláusula ORDER BY en la consulta.

Por ejemplo, supongamos que se desea elevar en 500 Euros. el salario de todos los empleados cuyo departamento no esté en MADRID

```
mysql> UPDATE empleados
    ->  SET salario = salario + 500
    ->  WHERE dep_no NOT IN (SELECT dep_no
    ->                       FROM departamentos
    ->                       WHERE localidad <> 'MADRID');

Query OK, 4 rows affected (0.05 sec)
Rows matched: 4  Changed: 4  Warnings: 0
```

## 2.3 – Eliminaciones con subconsultas

En ocasiones la condición que deben cumplir las filas que deseamos eliminar implica realizar una subconsulta a otras tablas. En estos casos se incluirá la subconsulta en la condición y los operadores necesarios tal como estudiamos en el apartado correspondiente del tema subconsultas. Esta aparecerá en la cláusula WHERE con las mismas restricciones que en las modificaciones:

- La tabla destino no puede aparecer en la consulta.
- No se puede incluir una cláusula ORDER BY en la consulta.

Supongamos que queremos eliminar de la tabla `departamentos` aquellos que no tienen ningún empleado.

```
mysql> DELETE FROM departamentos
    -> WHERE dep_no NOT IN
```

```
                    (SELECT DISTINCT dep_no FROM empleados);
Query OK, 1 row affected (0.02 sec)

mysql> SELECT *
    -> FROM departamentos;
+--------+--------------+-----------+
| DEP_NO | DNOMBRE      | LOCALIDAD |
+--------+--------------+-----------+
|     10 | CONTABILIDAD | BARCELONA |
|     20 | INVESTIGACION | VALENCIA  |
|     30 | VENTAS       | MADRID    |
+--------+--------------+-----------+
3 rows in set (0.00 sec)
```

El siguiente ejemplo eliminará los departamentos que tienen menos de tres empleados.

*Nota: esta orden la ejecutamos con las tablas departamentos2 y empleados2 que tiene borrado en cascada (con empleados y departamentos no sería posible por la restricción de integridad)*

```
mysql> DELETE FROM departamentos2
    -> WHERE dep_no IN (SELECT dep_no
    ->             FROM empleados2
    ->             GROUP BY dep_no
    ->             HAVING count(*) < 3);
Query OK, 1 row affected (0.39 sec)

mysql> SELECT *
    -> FROM departamentos2;
+--------+--------------+-----------+
| DEP_NO | DNOMBRE      | LOCALIDAD |
+--------+--------------+-----------+
|     10 | CONTABILIDAD | BARCELONA |
|     30 | VENTAS       | MADRID    |
|     40 | PRODUCCION   | SEVILLA   |
+--------+--------------+-----------+
3 rows in set (0.00 sec)
```

El ejemplo anterior borrará los departamentos que tengan menos de tres empleados pero, para que entre en la lista de selección, el departamento deberá tener al menos un empleado. Por tanto, los empleados que no tengan ningún empleado no se borrarán. Para evitar esto podemos cambiar la condición indicando que se borren aquellos departamentos que no están en la lista de departamentos con tres o más empleados.

```
mysql> DELETE FROM departamentos2
    -> WHERE dep_no NOT IN (SELECT dep_no
    -> FROM empleados2
    -> GROUP BY dep_no
    -> HAVING count(*) >= 3);
Query OK, 2 rows affected (0.12 sec)

mysql> SELECT *
    -> FROM departamentos2;
+--------+--------------+-----------+
| DEP_NO | DNOMBRE      | LOCALIDAD |
+--------+--------------+-----------+
```

```
|     10 | CONTABILIDAD | BARCELONA |
|     30 | VENTAS       | MADRID    |
+--------+--------------+-----------+
2 rows in set (0.00 sec)
```

Esta última orden borrará todos los departamentos que tienen: ninguno, uno o dos empleados.

Se pueden utilizar subconsultas anidadas a varios niveles, pero respetando la siguiente **restricción: la tabla destino no puede aparecer en la cláusula FROM de ninguna de las subconsultas que intervienen en la selección. Si se permiten referencias externas,** como en el siguiente ejemplo:

```
mysql> DELETE FROM departamentos
    -> WHERE NOT EXISTS (SELECT *
    ->                   FROM empleados
    ->                   WHERE empleados.dep_no=departamentos.dep_no);
Query OK, 0 rows affected (0.00 sec)
```

En estos casos la subconsulta con la referencia externa realiza la selección sobre la tabla destino antes de que se elimine ninguna fila.

*Nota: debe tenerse en cuenta que algunos productos comerciales permiten saltar esta restricción pero otros no.*

# 3 - Vistas

Podemos definir una vista como una consulta almacenada en la base de datos que se utiliza como una tabla virtual. Se define asociadas a una o varias tablas y no almacena los datos sino que trabaja sobre los datos de las tablas sobre las que está definida, estando así en todo momento actualizada.

## 3.1 - ¿Qué son las vistas y para qué sirven?.

Se trata de una perspectiva de la base de datos o ventana que permite a uno o varios usuarios ver solamente las filas y columnas necesarias para su trabajo.

Entre las ventajas que ofrece la utilización de vistas cabe destacar:

- Seguridad y confidencialidad: ya que la vista ocultará los datos confidenciales o aquellos para los que el usuario no tenga permiso.
- Comodidad: ya que solamente muestra los datos relevantes, permitiendo, incluso trabajar con agrupaciones de filas como si se tratase de una única fila o con composiciones de varias tablas como si se tratase de una única tabla.
- Independencia respecto a posibles cambios en los nombres de las columnas, de las tablas, etcétera.

Por ejemplo, la siguiente consulta permite al departamento de VENTAS realizar la gestión de sus empleados ocultando la información relativa a los empleados de otros departamentos.

```
SELECT *
FROM EMPLEADOS
WHERE dep_no=30;
```

La siguiente consulta permite a cualquier empleado de la empresa obtener información no confidencial de cualquier otro empleado ocultando las columnas SALARIO y COMISION:

```
SELECT emp_no, apellido, oficio, director, fecha_alta, dep_no FROM
```

```
empleados;
```

Para ello crearemos vistas y permitiremos a los usuarios tener acceso a las vistas sin tenerlo de la tabla completa.

## 3.2 - Creación y utilización de vistas

Como hemos dicho son tablas virtuales resultado de una consulta realizadas sobre tablas ya existentes. Las vistas no ocupan espacio en la base de datos ya que lo único que se almacena es la definición de la vista. El gestor de la base de datos se encargará de comprobar los comandos SQL que hagan referencia a la vista, transformándolos en los comandos correspondientes referidos a las tablas originales, todo ello de forma transparente para el usuario.

### 3.2.1 – Formato de la creación de vistas.

Para crear una vista se utiliza el comando CREATE VIEW según el siguiente formato genérico:

```
CREATE VIEW NombreVista
            [(DefiniciónColumna [,DefiniciónColumna....] )]
AS Consulta;
```

*Notación: la lista de columnas que define las columnas de la vista es opcional.*

donde **NombreVista……** es el nombre que tendrá la vista que se va a crear.
**DefinicionColumnas……..** permite especificar un nombre y su correspondiente tipo para cada columna de la vista.
Si no se especifica, cada columna quedará con el nombre o el alias correspondiente y el tipo asignado por la consulta.
**Consulta……..** en una consulta cuyo resultado formará la vista.

El siguiente ejemplo crea la vista emple_dep30 para la gestión de los empleados del departamento 30 mencionada en el apartado anterior.

```
mysql> CREATE VIEW emple_dep30 AS
    ->  SELECT * FROM EMPLEADOS
    ->  WHERE DEP_NO = 30;
Query OK, 0 rows affected (0.55 sec)

CREATE VIEW emple_dep30 AS
SELECT * FROM EMPLEADOS
WHERE DEP_NO = 30;
```

A continuación se muestra la sentencia que crea la vista `datos_emple` que contiene información de todos los empleados ocultando la información confidencial.

```
mysql> CREATE VIEW datos_emple AS
    -> SELECT emp_no, apellido, oficio, director,
                fecha_alta, dep_no
    -> FROM empleados;
```

```
Query OK, 0 rows affected (0.00 sec)
```

Las vistas pueden a su vez definirse sobre otras vistas. Si ya tenemos creada las vista `datos_emple`, podríamos crear otra vista sobre ella:

```
mysql> CREATE VIEW datos_emple_10 AS
    ->  SELECT *
    ->  FROM datos_emple
    ->  WHERE dep_no=10;
Query OK, 0 rows affected (0.00 sec)
```

### 3.2.2 – Utilización de vistas

Una vez definida puede ser utilizada para consultas de igual forma que una tabla.
Con algunas restricciones, todos los formatos de selección vistos para las tablas son aplicables para la selección de filas en las vistas.

Por ejemplo:

```
mysql> SELECT * FROM datos_emple;

+--------+----------+-----------+----------+------------+--------+
| emp_no | apellido | oficio    | director | fecha_alta | dep_no |
+--------+----------+-----------+----------+------------+--------+
|   7499 | ALONSO   | VENDEDOR  |     7698 | 1981-02-23 |     30 |
|   7521 | LOPEZ    | EMPLEADO  |     7782 | 1981-05-08 |     10 |
|   7654 | MARTIN   | VENDEDOR  |     7698 | 1981-09-28 |     30 |
|   7698 | GARRIDO  | DIRECTOR  |     7839 | 1981-05-01 |     30 |
|   7782 | MARTINEZ | DIRECTOR  |     7839 | 1981-06-09 |     10 |
|   7839 | REY      | PRESIDENTE|     NULL | 1981-11-17 |     10 |
|   7844 | CALVO    | VENDEDOR  |     7698 | 1981-09-08 |     30 |
|   7876 | GIL      | ANALISTA  |     7782 | 1982-05-06 |     20 |
|   7900 | JIMENEZ  | EMPLEADO  |     7782 | 1983-03-24 |     20 |
+--------+----------+-----------+----------+------------+--------+
9 rows in set (0.01 sec)
```

También podemos seleccionar solo algunas columnas y poner una condición

```
mysql> SELECT apellido, director
    ->  FROM datos_emple
    ->  WHERE oficio = 'VENDEDOR';

+----------+----------+
| apellido | director |
+----------+----------+
| ALONSO   |     7698 |
| MARTIN   |     7698 |
| CALVO    |     7698 |
+----------+----------+
3 rows in set (0.00 sec)
```

Pero debe tenerse en cuenta que si al definir la vista hemos indicado nuevos nombres para columnas y expresiones si podremos hacer referencia a ellos en las sentencias de selección, pero si hemos omitido la definición de las columnas y en la sentencia de creación hemos realizado la selección de todas las columnas (creada con select *) solo puede hacerse una selección de todas las columnas de

la vista (con *)

La vista `emple_dep30` la creamos sin especificar nuevo nombre para las columnas de la vista y con una sentencia select *. Podemos obtener los datos de la vista si escribimos:

```
mysql> SELECT * FROM emple_dep30;

+-------+----------+---------+---------+------------+---------+--------+--------+
|EMP_NO | APELLIDO | OFICIO  |DIRECTOR | FECHA_ALTA | SALARIO |COMISION| DEP_NO |
+-------+----------+---------+---------+------------+---------+--------+--------+
|  7499 | ALONSO   | VENDEDOR|    7698 | 1981-02-23 | 1400.00 | 400.00 |     30 |
|  7654 | MARTIN   | VENDEDOR|    7698 | 1981-09-28 | 1500.00 |1600.00 |     30 |
|  7698 | GARRIDO  | DIRECTOR|    7839 | 1981-05-01 | 3850.12 |   NULL |     30 |
|  7844 | CALVO    | VENDEDOR|    7698 | 1981-09-08 | 1800.00 |   0.00 |     30 |
+-------+----------+---------+---------+------------+---------+--------+--------+
4 rows in set (0.06 sec)
```

Pero no podemos referenciar las columnas en la selección al recuperar datos de la vista:

```
mysql> SELECT apellido FROM emple_dep30;
ERROR 1054 (42S22): Unknown column 'apellido' in 'field list'
```

### 3.2.3 - Restricciones para la creación y utilización de vistas

No se puede usar la cláusula `ORDER BY` en la creación de una vista ya que las filas de una tabla no están ordenadas (la vista es una tabla virtual). No obstante, si se puede utilizar dicha cláusula a la hora de recuperar datos de la vista.

Es obligatorio especificar la lista de nombres de columnas de la vista o un alias cuando la consulta devuelve funciones de agrupamiento como `SUM`, `COUNT`, etcétera y posteriormente quiere hacerse referencia a ellas.

Pueden utilizarse funciones de agrupación sobre columnas de vistas que se basan a su vez en funciones de agrupación lo que permite resolver los casos en los que un doble agrupamiento que no está permitido por el estándar. Así creamos una vista con una primera función de agrupación y sobre ella aplicamos la segunda función de agrupación, obteniendo el resultado deseado.

### 3.2.4 - Ejemplos creación y utilización de vistas

Como hemos dicho una vez creada la vista se puede utilizar como si se tratase de una tabla (observando las restricciones anteriores). Veamos lo que podemos hacer con las vistas con los ejemplos.

1- El siguiente ejemplo crea la vista *datos_vendedores* que muestra solamente las columnas *emp_no, apellido, director, fecha_alta, dep_no,* de aquellos empleados cuyo oficio es VENDEDOR.

```
mysql> CREATE VIEW datos_vendedores
    ->  (num_vendedor, apellido, director, fecha_alta, dep_no) AS
    ->  SELECT emp_no, apellido, director, fecha_alta, dep_no
    ->  FROM empleados
    ->  WHERE oficio = 'VENDEDOR';
Query OK, 0 rows affected (0.00 sec)
```

Los datos accesibles mediante la vista creada serán:

```
mysql> SELECT  *
    ->  FROM datos_vendedores;
+--------------+----------+----------+------------+--------+
| num_vendedor | apellido | director | fecha_alta | dep_no |
+--------------+----------+----------+------------+--------+
|         7499 | ALONSO   |     7698 | 1981-02-23 |     30 |
|         7654 | MARTIN   |     7698 | 1981-09-28 |     30 |
|         7844 | CALVO    |     7698 | 1981-09-08 |     30 |
+--------------+----------+----------+------------+--------+
3 rows in set (0.02 sec)
```

2- También se pueden crear vistas a partir de consultas que incluyen agrupaciones, como en el siguiente ejemplo:

```
mysql> CREATE VIEW resumen_dep1
    ->  (dep_no, num_empleados, suma_salario, suma_comision) AS
    ->  SELECT dep_no, COUNT(emp_no), SUM(salario),
              SUM(IFNULL(comision,0))
    ->  FROM empleados
    ->  GROUP BY dep_no;
Query OK, 0 rows affected (0.01 sec)
```

En estos casos, cada fila de la vista corresponderá a varias filas en la tabla original tal como se puede comprobar en la siguiente consulta:

```
mysql> SELECT *
    ->  FROM resumen_dep1;
+--------+---------------+--------------+---------------+
| dep_no | num_empleados | suma_salario | suma_comision |
+--------+---------------+--------------+---------------+
|     10 |             3 |      9800.50 |          0.00 |
|     20 |             2 |      4750.00 |          0.00 |
|     30 |             4 |      8550.12 |       2000.00 |
+--------+---------------+--------------+---------------+
3 rows in set (0.02 sec)
```

Normalmente la mayoría de las columnas de este tipo de vistas corresponden a funciones de columna tales como SUM, AVERAGE, MAX, MIN, etcétera. Por ello el estándar SQL establece en estos casos la obligatoriedad de especificar la lista de columnas o de alias si posteriormente quiere hacerse referencia a ellas. Aunque algunos gestores de bases de datos permiten saltar esta restricción. No es aconsejable ya que las columnas correspondientes de la vista quedarán con nombres como COUNT(EMP_NO), *SUM(SALARIO), SUM(COMISION)* lo cual no resulta operativo para su posterior utilización.

```
mysql> SELECT dep_no, num_empleados, suma_salario
    -> FROM resumen_dep1;
+--------+---------------+--------------+
| dep_no | num_empleados | suma_salario |
+--------+---------------+--------------+
|     10 |             3 |      9800.50 |
|     20 |             2 |      4750.00 |
|     30 |             4 |      8550.12 |
+--------+---------------+--------------+
3 rows in set (0.14 sec)
```

3- Sobre esta vista, con una función de agrupación, podemos hacer otra función de agrupación, por ejemplo obtener el departamento con mayor suma de salarios para sus empleados:

Creamos la vista `resumen_dep2` con dos totales resultado e aplicar funciones de grupo

```
mysql> CREATE VIEW resumen_dep2
    -> AS SELECT dep_no, COUNT(emp_no) "num_empleados",
          SUM(salario) "suma_salario",
    -> FROM empleados
    -> GROUP BY dep_no;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT dep_no, num_empleados, suma_salario
    -> FROM resumen_dep2;
+--------+---------------+--------------+
| dep_no | num_empleados | suma_salario |
+--------+---------------+--------------+
|     10 |             3 |      9800.50 |
|     20 |             2 |      4750.00 |
|     30 |             4 |      8550.12 |
+--------+---------------+--------------+
3 rows in set (0.00 sec)
```

Y sobre ella volvemos a aplicar una función de grupo para hallar el máximo de las sumas de los salarios

```
mysql> SELECT MAX(suma_salario)
    -> FROM resumen_dep2;

+-------------------+
| MAX(suma_salario) |
+-------------------+
|           9800.50 |
+-------------------+
1 row in set (0.00 sec)
```

Y también podemos obtener el departamento el que pertenece este salario máximo

```
mysql> SELECT dep_no, dnombre
    -> FROM departamentos
    -> WHERE dep_no = (SELECT dep_no
    ->                 FROM resumen_dep2
    ->                 WHERE suma_salario=(SELECT MAX(suma_salario)
    ->                                     FROM resumen_dep2));

+--------+--------------+
| dep_no | dnombre      |
+--------+--------------+
|     10 | CONTABILIDAD |
+--------+--------------+
1 row in set (0.00 sec)
```

4 –Así mismo, se pueden crear vistas que incluyan todas o varias de las posibilidades estudiadas. Por ejemplo la siguiente vista permite trabajar con datos de dos tablas, agrupados y seleccionando las filas que interesan (en este caso todos los departamentos que tengan más de dos empleados):

```
mysql> CREATE VIEW resumen_emp_dep
    ->  (departamento,  num_empleados, suma_salario) AS
    ->   SELECT dnombre, COUNT(emp_no), SUM(salario)
    ->   FROM empleados, departamentos
    ->   WHERE empleados.dep_no = departamentos.dep_no
    ->   GROUP BY empleados.dep_no,dnombre
    ->   HAVING COUNT(*) > 2;
Query OK, 0 rows affected (0.00 sec)


mysql> SELECT  departamento, num_empleados
    -> FROM resumen_emp_dep;
+--------------+---------------+
| departamento | num_empleados |
+--------------+---------------+
| CONTABILIDAD |             3 |
| VENTAS       |             4 |
+--------------+---------------+
2 rows in set (0.02 sec)
```

## 3.3 - Eliminación de vistas

### 3.3.1 - Formato de borrado de vistas

La sentencia DROP VIEW permite eliminar la definición de una vista.

**DROP VIEW [IF EXISTS] NombreVista**
**[RESTRICT | CASCADE ]**

La cláusula IF EXISTS previene los errores que puedan producirse si no existe la tabla que queremos borrar
La cláusula `CASCADE Y RESTRICT` están permitidas pero no implementada en la versión 5.

### 3.3.2 - Ejemplos de borrado de vistas

```
mysql> DROP VIEW IF EXISTS resumen_emp_dep;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

### 3.3.3 - Borrado de las tablas o vistas asociadas a una vista

Si se borran las tablas a las vistas sobre las que están definidas las vistas la vista se queda invalida (no se borra su definición pero no está utilizable).

```
mysql> SELECT *
```

```
    -> FROM datos_emple_10;

+--------+----------+------------+----------+------------+--------+
| emp_no | apellido | oficio     | director | fecha_alta | dep_no |
+--------+----------+------------+----------+------------+--------+
|   7521 | LOPEZ    | EMPLEADO   |     7782 | 1981-05-08 |     10 |
|   7782 | MARTINEZ | DIRECTOR   |     7839 | 1981-06-09 |     10 |
|   7839 | REY      | PRESIDENTE |     NULL | 1981-11-17 |     10 |
+--------+----------+------------+----------+------------+--------+
3 rows in set (0.05 sec)
```

Si borramos `datos_emple` sobre la que está definida `datos_emple_10` esta pasará a estar inválida.

```
mysql> DROP VIEW IF EXISTS datos_emple;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT *
    -> FROM datos_emple_10;
ERROR 1356 (HY000): View 'test.datos_emple_10' references invalid
table(s) or column(s)
```