

UNIDAD 6

MODELO FÍSICO DQL

BASES DE DATOS 22/23
CFGS DAW

PARTE 1. NIVEL BÁSICO

Revisado por:

Sergio Badal, Abelardo Martínez y Pau Miñana

Autores:

Raquel Torres

Paco Aldarias

Fecha: 23/01/23

Licencia Creative Commons



Reconocimiento - NoComercial - CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

ÍNDICE DE CONTENIDO

1. DQL.....	4
1.1 ORGANIZACIÓN DE LA UNIDAD.....	5
2. SELECT.....	6
3. FROM.....	6
4. ALL / DISTINCT.....	8
5. AS.....	9
6. WHERE: CONSULTAS CON FILTROS.....	10
6.1 Operadores relacionales.....	10
6.1.1 Consulta 1.....	11
6.1.2 Consulta 2.....	11
6.1.3 Consulta 3.....	12
6.1.4 Consulta 4.....	12
6.1.5 Consulta 5.....	12
6.1.6 Consulta 6.....	13
6.1.7 Consulta 7.....	13
6.2 IS NULL / IS NOT NULL.....	14
6.2.1 Consulta 8.....	14
6.2.2 Consulta 9.....	14
6.3 Operadores lógicos.....	15
6.3.1 Consulta 10.....	16
6.3.2 Consulta 11.....	16
6.3.3 Consulta 12.....	17
6.3.4 Consulta 13.....	18
6.3.5 Consulta 14.....	18
6.3.6 Consulta 15.....	19
6.3.7 Consulta 16.....	19
6.4 BETWEEN.....	20
6.4.1 Consulta 17.....	20
6.4.2 Consulta 18.....	21
6.5 IN / NOT IN.....	21
6.5.1 Consulta 19.....	21
6.5.2 Consulta 20.....	22
6.6 LIKE / NOT LIKE.....	22
6.6.1 Consulta 21.....	23
6.6.2 Consulta 22.....	23
6.6.3 Consulta 23.....	23
6.6.4 Consulta 24.....	24
7. ORDENACIÓN DEL RESULTADO.....	24
7.1 Consulta 25.....	25
7.2 Consulta 26.....	25
7.3 Consulta 27.....	25
7.4 Consulta 28.....	26
7.5 Consulta 29.....	26
7.6 Consulta 30.....	26
7.7 Consulta 31.....	27
8. FUNCIONES AGREGADAS.....	27
8.1 Consulta 32.....	28
8.2 Consulta 33.....	28
8.3 Consulta 34.....	28
8.4 Consulta 35.....	28

8.5 Consulta 36.....	29
8.6 Consulta 37.....	29
8.7 Consulta 38.....	30
9. CÁLCULOS ARITMÉTICOS.....	30



CONSEJO

El lenguaje SQL **NO es sensible a mayúsculas/minúsculas** pero, como algunos sistemas operativos sí que lo son, te recomendamos que seas fiel a la sintaxis usada al crear los metadatos (tablas, atributos, restricciones...).

No obstante, se considera “buena praxis”:

=> Usar siempre minúsculas o UpperCamelCase en los metadatos.

=> Usar mayúsculas en los alias de las tablas si las tablas están en minúsculas y viceversa, ya que ayuda a entender mejor las consultas.

=> Los comandos SQL (SELECT, INSERT, FROM, WHERE...) también se recomiendan en mayúsculas, pero son igual de válidos en minúsculas.

UD6.1. MODELO FÍSICO DQL. NIVEL BÁSICO

1. DQL

Si has llegado hasta aquí quiere decir que ya sabes cómo crear, modificar y eliminar tablas y cómo insertar, modificar y eliminar registros.

A partir de ahora vamos a centrarnos en realizar consultas a nuestra base de datos para obtener la información que necesitamos. Realmente esta es la parte que más se utiliza, pues una vez que la base de datos está creada y funcionando, las tablas se suelen modificar raramente (si no hay cambios en el contexto en el que fue diseñada y creada) sin embargo las consultas se realizarán durante toda la vida de la base de datos y siempre surgirán nuevas cuestiones a las que habrá que dar solución con nuevas consultas.

Por eso el aprender a realizar consultas y dominar el SQL en este aspecto es fundamental.



Importante

El DQL (Data Query Language) lo forman las instrucciones capaces de consultar los datos de las tablas.

El único comando que pertenece a este lenguaje es el versátil comando SELECT. Este comando permite:

- Obtener datos de ciertas columnas de una tabla (proyección)
- Obtener registros (filas) de una tabla de acuerdo con ciertos criterios (selección)
- Mezclar datos de tablas diferentes (asociación, *join*)
- Realizar cálculos sobre los datos
- Agrupar datos

1.1 ORGANIZACIÓN DE LA UNIDAD

Esta es nuestra propuesta para estas cinco semanas de Unidad:

- SEMANA 1 (lunes 23 de enero)
 - **NIVEL BÁSICO**
 - SELECT, FROM, ALL/DISTINCT, AS, WHERE, ORDER, AGREGADOS
 - BOLETÍN – NIVEL BÁSICO
- SEMANA 2 (lunes 30 de enero)
 - **NIVEL MEDIO.**
 - AGRUPACIONES, MULTITABLA (JOIN)
 - BOLETÍN – NIVEL MEDIO
- SEMANA 3 (lunes 06 de febrero)
 - **NIVEL AVANZADO.**
 - REFLEXIVAS, SUBCONSULTAS, DERIVADOS, UNIONES, VISTAS, DQL EN DML
 - BOLETÍN – NIVEL AVANZADO
- SEMANA 4 (lunes 13 de febrero) y SEMANA 5 (lunes 20 de febrero)
 - **CONSULTAS SOBRE VARIAS BBDD COMPLEJAS**

ACTIVIDAD EVALUABLE 3

- **Fecha de presentación: lunes 27 de febrero**
- **Fecha de entrega: domingo 9 de abril (6 semanas)**

2. SELECT

La sintaxis básica para una consulta es:

```
SELECT [ALL | DISTINCT] [expre_col1, ... , expre_colN | *]  
  
FROM nom_tabla1 [,..., nom_tablaN]  
  
[WHERE condición]  
  
[ORDER BY expre_col1 [DESC | ASC] [, expre_col2 [DESC | ASC]]...]
```

Donde expre_col puede ser:

- Una columna.
- Una constante.
- Una expresión aritmética.
- Una o varias funciones anidadas.

Todos los ejemplos que vamos a ver en esta unidad son válidos para SQL Standard. Puedes probar a realizar estas consultas en las bases de datos vistas.

Analicemos la sintaxis en detalle a continuación.

3. FROM

El FROM es obligatorio porque especifica la tabla o tablas de las que se recuperarán los datos.

Imaginemos que queremos saber el código y el nombre de todos los departamentos de la empresa.

Fíjate que ahora no queremos todos los campos (recuerda que en anteriores consultas siempre hemos sacado todos los campos representándolos mediante un asterisco *), sino solamente un par de ellos. Para ello, seguiremos la estructura de la instrucción que vimos al comenzar este nuevo apartado.

La primera forma que vamos a utilizar para obtener el resultado es:

```
mysql> select CodDpto, Nombre from departamentos;  
+-----+-----+  
| CodDpto | Nombre |  
+-----+-----+  
| ADM     | Administración |  
| ALM     | Almacén        |  
| CONT    | Contabilidad   |  
| IT      | Informática     |  
+-----+-----+  
4 rows in set (0.00 sec)
```

Como puedes ver, se utiliza la palabra reservada **SELECT** seguida de los nombre de los campos a mostrar separados por comas. Después colocamos la palabra reservada **FROM** seguida del nombre de la tabla que queremos consultar.

Otra forma de hacerlo es colocando el **nombre de la tabla** delante de cada campo y unido a él mediante un punto. En este ejemplo no tiene mucho sentido utilizar esta nomenclatura, simplemente se comenta para mostraros otra forma de hacerlo.

Sin embargo, más adelante, se tendrá que utilizar obligatoriamente esta forma o un alias en algunas consultas, cuando ocurra lo siguiente: supongamos que tenemos las tablas departamentos y proyectos, ambas tienen un campo llamado Nombre. Si en la consulta participasen las dos tablas (aún no lo hemos visto, solo estamos adelantando algo que veremos posteriormente) y queremos mostrar el campo Nombre, si no colocamos delante el nombre de la tabla, la base de datos no sabría cuál mostrar, si el campo Nombre del departamento o el campo Nombre del proyecto. Al poner departamentos.Nombre o proyectos.Nombre ya le estaríamos indicando cuál queremos.

```
mysql> select departamentos.CodDpto, departamentos.Nombre from departamentos;
```

CodDpto	Nombre
ADM	Administración
ALM	Almacén
CONT	Contabilidad
IT	Informática

```
4 rows in set (0.00 sec)
```

Normalmente el nombre de las tablas es largo y tenerlo que escribir delante de todos los campos es un poco tedioso, por ello podemos crear un **alias** del nombre de la tabla (normalmente una sola letra) y colocar ese alias delante del campo unido a él por un punto.

El efecto es el mismo que en el ejemplo anterior, pero escribiendo menos texto en la instrucción.

En este caso hemos creado el alias **d** para departamentos y referenciamos a los campos como **d.CodDpto** y **d.Nombre**.

```
mysql> select d.CodDpto, d.Nombre from departamentos d;
```

CodDpto	Nombre
ADM	Administración
ALM	Almacén
CONT	Contabilidad
IT	Informática

```
4 rows in set (0.00 sec)
```



CONSEJO

Se considera una “mala praxis” (aunque es 100% correcto y muy usado en ámbitos académicos) usar **SELECT *** por el consumo innecesario de recursos que suele producir, **sobre todo en tablas grandes de miles de registros**.

```
SELECT * FROM PROFESORES;
```

4. ALL / DISTINCT

Con ALL, recuperamos todas las filas, aunque estén repetidas. Es la opción por defecto.

Por ejemplo, si ejecutamos:

```
SELECT ALL proveedor FROM Pedidos;
```

Equivale a:

```
SELECT proveedor FROM Pedidos;
```

En ambos casos obtendremos los proveedores de cada pedido, pudiendo existir filas repetidas.

```
mysql> SELECT ALL Proveedor FROM PEDIDOS;
+-----+
| Proveedor |
+-----+
| BA843     |
| MA280     |
| MA280     |
| TO342     |
| TO342     |
+-----+
5 rows in set (0.00 sec)
```

Sin embargo, DISTINCT solo recupera las filas que son distintas.

Es decir, si en el resultado de la consulta hay filas repetidas (el valor de las diferentes filas se repite), solo se muestra una de las filas y el resto de filas repetidas son omitidas.

En el ejemplo anterior obteníamos el proveedor al que pertenece cada pedido. Como hay dos pedidos que pertenecen al mismo proveedor, aunque se hayan realizado en fechas diferentes, la consulta anterior repita uno de los proveedores.

Si lo que nos interesa conocer son los distintos proveedores podríamos utilizar:

```
SELECT DISTINCT proveedor FROM Pedidos;
```



CONSEJO

Se considera una “mala praxis” (aunque es 100% correcto y muy usado en ámbitos académicos) usar **SELECT DISTINCT** cuando queremos eliminar los elementos repetidos de una consulta, ya que los elementos repetidos SUELEN ser consecuencia de una consulta mal diseñada como veremos más adelante.

5. AS

Con AS, podemos cambiar el nombre de las columnas mostrando el texto que deseemos.

Para ello, después del nombre del campo escribiremos la palabra reservada AS seguida del nuevo nombre de la columna. Por ejemplo:

```
mysql> select CodDpto as MICOD, Nombre as MINOM from departamentos;
+-----+-----+
| MICOD | MINOM |
+-----+-----+
| ADM   | Administración |
| ALM   | Almacén        |
| CONT  | Contabilidad   |
| IT    | Informática    |
+-----+-----+
4 rows in set (0.00 sec)
```

Si el nombre de la columna va a estar formado por varias palabras será necesario colocarlas entre comillas dobles (en MySQL también sirven comillas simples).

```
mysql> select CodDpto as "MI COD", Nombre as "MI NOM" from departamentos;
+-----+-----+
| MI COD | MI NOM |
+-----+-----+
| ADM   | Administración |
| ALM   | Almacén        |
| CONT  | Contabilidad   |
| IT    | Informática    |
+-----+-----+
4 rows in set (0.00 sec)
```



MUY IMPORTANTE

En los alias de atributos **SIEMPRE** incluimos la palabra reservada AS.

En los alias de tablas **NUNCA** incluimos la palabra reservada AS.

```
SELECT P.NOMBRE AS NOMBREPROFE FROM PROFESORES P;
```

6. WHERE: CONSULTAS CON FILTROS

Con WHERE, seleccionamos los datos que cumplan ciertas condiciones.

La cláusula WHERE nos permitirá incluir las condiciones necesarias para filtrar la información. Esas condiciones que nos van a permitir filtrar la información es lo que denominamos filtros.

Comenzaremos con las consultas simples para después realizar algunas más complejas.

6.1 Operadores relacionales

Los operadores que nos permitirán comparar datos para establecer los filtros son:

Operador	Uso	Significado
=	A = B	Compara si A es igual a B
>	A > B	Compara si A es mayor que B
<	A < B	Compara si A es menor que B
<>	A <> B	Compara si A es distinto de B
>=	A >= B	Compara si A es mayor o igual que B
<=	A <= B	Compara si A es menor o igual que B

Debemos tener en cuenta que si la comparación de un campo se realiza con una cadena de caracteres, la cadena de caracteres debe estar entre comillas y que si se realiza con un campo numérico, el valor no debe ponerse entre comillas (aunque MySQL sí lo admite).

Realicemos algunos ejemplos.



CUIDADO

Los operadores == (doble igual) y != (distinto) **no son válidos en SQL**, en su lugar usamos un único igual y el símbolo combinado <>.

SELECT * FROM TABLA WHERE A != 'texto'

SELECT * FROM TABLA WHERE A == 'texto'

```
mysql> select * from departamentos;
+-----+-----+-----+
| CodDpto | Nombre          | Ubicacion          |
+-----+-----+-----+
| ADM     | Administración  | Planta quinta U2   |
| ALM     | Almacén         | Planta baja U1     |
| CONT    | Contabilidad    | Planta quinta U1   |
| IT      | Informática     | Planta sótano U3   |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> select * from empleados;
+-----+-----+-----+-----+-----+-----+
| dni      | nombre          | especialidad       | fechaalta | dpto | codp |
+-----+-----+-----+-----+-----+-----+
| 12345678A | Alberto Gil     | Contable           | 2010-12-10 | CONT | MAD20 |
| 23456789B | Mariano Sanz    | Informática        | 2011-10-04 | IT   | NULL  |
| 45678901D | Ana Silván      | Informática        | 2012-11-25 | IT   | MAD20 |
| 67890123F | Roberto Milán   | Logística          | 2010-05-02 | ALM  | NULL  |
| 78901234G | Rafael Colmenar | Informática        | 2013-06-10 | IT   | T0451 |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> select * from proyectos;
+-----+-----+-----+-----+-----+
| codproy | nombre          | fechainicio | dpto | responsable |
+-----+-----+-----+-----+-----+
| MAD20   | Repsol, S.A.    | 2012-02-10  | CONT | 12345678A   |
| T0451   | Consejería de Educación | 2012-05-24  | IT   | 23456789B   |
| U324    | Oceanográfico   | 2012-09-29  | NULL | NULL        |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

6.1.1 Consulta 1

Mostrar el nombre de los empleados del departamento de Informática (IT).

```
mysql> select nombre from empleados where dpto='IT';
+-----+
| nombre          |
+-----+
| Mariano Sanz    |
| Ana Silván      |
| Rafael Colmenar |
+-----+
3 rows in set (0.00 sec)
```

6.1.2 Consulta 2

Mostrar los empleados cuya especialidad sea la Logística.

```
mysql> select nombre from empleados where especialidad='Logística';
+-----+
| nombre          |
+-----+
| Roberto Milán   |
+-----+
1 row in set (0.00 sec)
```

6.1.3 Consulta 3

Muestra todos los datos del empleado que se llama Mariano Sanz.

```
mysql> select * from empleados where nombre = 'Mariano Sanz';
```

dni	nombre	especialidad	fechaalta	dpto	codp
23456789B	Mariano Sanz	Informática	2011-10-04	IT	NULL

```
1 row in set (0.00 sec)
```

6.1.4 Consulta 4

Mostrar el nombre y el precio de los productos cuyo precio es igual o mayor de 20 euros.

```
mysql> select nombreproducto, precio
-> from productosped
-> where precio >= 20;
```

nombreproducto	precio
AVION FK20	31.75
BOLA BOOM	22.2
PATINETE 3 RUEDAS	22.5

```
3 rows in set (0.00 sec)
```

6.1.5 Consulta 5

Mostrar la referencia y el precio de todos los productos cuyo precio es menor de 15 euros.

Fíjate que al poner solo menor, el producto PM30 (PELUCHE MAYA) que vale 15 euros no ha salido, pues solo hemos pedido los menores de esa cantidad. Para que se hubiese mostrado tendríamos que haber puesto menor o igual a 15 (precio <= 15).

```
mysql> select RefeProducto, Precio
-> from productosped
-> where precio < 15;
```

RefeProducto	Precio
HM12	12.8
NPP10	3

```
2 rows in set (0.00 sec)
```

6.1.6 Consulta 6

Muestra todos los datos de los proveedores cuyo código postal es diferente a 45600.

En algunos SGBD también se puede utilizar el símbolo != para indicar distinto, por ejemplo (Codpostal != '45600') pero cuidado, porque no es SQL estándar y no se admite en la mayoría de casos.

```
mysql> select * from proveedores
-> where CodPostal <> '45600';
```

CodProveedor	NombreProveedor	CodPostal
BA843	CARMELO DIAZ, S.L.	06004
MA280	TOYPLAY, S.A.	28005
SE391	ARTEAND, S.L.	41400

```
3 rows in set (0.00 sec)
```

6.1.7 Consulta 7

Mostrar todos los campos de los pedidos realizados antes del 12/06/2013.

Aquí debemos tener cuidado pues ya sabemos que las fechas son tratadas con formatos diferentes en MySQL y en Oracle. Aunque hay que tener en cuenta que para ambas bases de datos la fecha debe colocarse entre comillas en la comparación.

En MySQL.

```
mysql> select * from pedidos where fecha < '2013-06-12';
```

NumPedido	Fecha	Proveedor
1	2013-06-10	T0342
2	2013-06-10	MA280

```
2 rows in set (0.00 sec)
```

En Oracle.

```
SQL> select * from pedidos where fecha < '12/06/2013';
```

NUMPEDIDO	FECHA	PROVEEDOR
1	10/06/13	T0342
2	10/06/13	MA280

6.2 IS NULL / IS NOT NULL

Con **IS NULL** / **IS NOT NULL** / **NOT nom_campo IS NULL**, comprobamos si el un campo es nulo (no tiene datos).

Para ello disponemos del filtro **IS NULL**, o bien, si queremos ver lo contrario, es decir los que no son nulos, **IS NOT NULL**.



CUIDADO

Estas condiciones no son válidas:

SELECT * FROM TABLA WHERE A = NULL

SELECT * FROM TABLA WHERE A <> NULL

SELECT * FROM TABLA WHERE A NOT IS NULL

Sin embargo, como veremos más adelante, ésta sí que es válida:

SELECT * FROM TABLA WHERE NOT A IS NULL

6.2.1 Consulta 8

Mostrar los proyectos que no están asignados a ningún departamento.

```
mysql> select * from proyectos where dpto is null;
+-----+-----+-----+-----+-----+
| codproy | nombre           | fechainicio | dpto | responsable |
+-----+-----+-----+-----+-----+
| U324    | Oceanográfico   | 2012-09-29  | NULL | NULL        |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

6.2.2 Consulta 9

Mostrar los proyectos que están asignados a un departamento.

```
mysql> select * from proyectos where dpto is not null;
+-----+-----+-----+-----+-----+
| codproy | nombre           | fechainicio | dpto | responsable |
+-----+-----+-----+-----+-----+
| MAD20   | Repsol, S.A.     | 2012-02-10  | CONT | 12345678A   |
| T0451   | Consejería de Educación | 2012-05-24  | IT   | 23456789B   |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

6.3 Operadores lógicos

Ahora vamos a ver los operadores lógicos que nos permitirán crear filtros más elaborados que los vistos hasta este momento.

- **AND es un “y” lógico.** El resultado será verdadero si los dos elementos que une son verdaderos. Por ejemplo:

(condición_1) AND (condición_2)

Darán como resultado verdadero si la condición_1 y la condición_2 son verdaderas. La tabla de verdad del operador AND es la siguiente:

A	B	RESULTADO
V	V	V
V	F	F
F	V	F
F	F	F

- **OR es un “o” lógico.** El resultado será verdadero si uno de los dos elementos o los dos son verdaderos. Por ejemplo:

(condición_1) OR (condición_2)

El resultado será verdadero si la condición_1 es verdadera o si la condición_2 es verdadera o bien si ambas son verdaderas. La tabla de verdad del operador OR es la siguiente:

A	B	RESULTADO
V	V	V
V	F	V
F	V	V
F	F	F

- **NOT es la negación de una condición.** El resultado será lo contrario del resultado de la condición. Por ejemplo:

NOT (condición)

Si la condición_1 es verdadera el resultado de aplicar el NOT será falso y si el resultado de condición_1 es falso, el resultado de aplicar el NOT será verdadero. La tabla de verdad del operador NOT es la siguiente:

A	RESULTADO
V	F
F	V

Las condiciones se evalúan de izquierda a derecha. Además podemos emplear paréntesis para agrupar condiciones. Lo más aconsejable es que incluyáis paréntesis para aclararos cuando tengáis dudas de cómo van a ser evaluadas las condiciones. Si colocamos paréntesis de sobra no hay problema, el problema aparecerá si ponemos paréntesis de menos, pues el resultado de la expresión condicional puede no ser el que deseamos.

Veamos algunos ejemplos:

6.3.1 Consulta 10

Mostrar todos los datos de los empleados del departamento de informática (IT) o de Contabilidad.

```
mysql> select * from empleados where dpto='IT' or dpto='CONT';
```

dni	nombre	especialidad	fechaalta	dpto	codp
12345678A	Alberto Gil	Contable	2010-12-10	CONT	MAD20
23456789B	Mariano Sanz	Informática	2011-10-04	IT	NULL
45678901D	Ana Silván	Informática	2012-11-25	IT	MAD20
78901234G	Rafael Colmenar	Informática	2013-06-10	IT	T0451

```
4 rows in set (0.03 sec)
```

6.3.2 Consulta 11

Mostrar todos los datos de los empleados del departamento de informática (IT) que se han incorporado a la empresa antes del 1 de enero de 2013.

Al haber fechas, será diferente para MySQL y para Oracle (siempre que no hayáis cambiado el formato de fecha de MySQL).

MySQL.

```
mysql> select * from empleados where dpto='IT' and fechaalta < '2013-01-01';
```

dni	nombre	especialidad	fechaalta	dpto	codp
23456789B	Mariano Sanz	Informática	2011-10-04	IT	NULL
45678901D	Ana Silván	Informática	2012-11-25	IT	MAD20

```
2 rows in set (0.00 sec)
```


Oracle.

```
SQL> select * from empleados where dpto='IT' and fechaalta < '01/01/2013';
```

DNI	NOMBRE	ESPECIALIDAD	FECHAALT
DPTO	CODP		
23456789B IT	Mariano Sanz	Informática	04/10/11
45678901D IT	Ana Silván MAD20	Informática	25/11/12

6.3.3 Consulta 12

Mostrar todos los datos de los empleados que se han incorporado desde del 1 de enero de 2013 o bien trabajan en el departamento de Contabilidad (CONT) o en el Almacén (ALM).

MySQL.

```
mysql> select * from empleados
-> where fechaalta > '2013-01-01'
-> or Dpto = 'CONT'
-> or Dpto = 'ALM';
```

dni	nombre	especialidad	fechaalta	dpto	codp
12345678A	Alberto Gil	Contable	2010-12-10	CONT	MAD20
67890123F	Roberto Milán	Logística	2010-05-02	ALM	NULL
78901234G	Rafael Colmenar	Informática	2013-06-10	IT	T0451

3 rows in set (0.00 sec)

Oracle.

```
SQL> select * from empleados
2 where fechaalta > '01/01/2013'
3 or Dpto = 'CONT'
4 or Dpto = 'ALM';
```

DNI	NOMBRE	ESPECIALIDAD	FECHAALT
DPTO	CODP		
12345678A CONT	Alberto Gil MAD20	Contable	10/12/10
78901234G IT	Rafael Colmenar T0451	Informática	10/06/13
67890123F ALM	Roberto Milán	Logística	05/02/10

6.3.4 Consulta 13

Mostrar todos los datos de los proyectos que pertenecen al departamento de informática y se iniciaron antes del 1 de enero del 2013 o bien no tienen asignado un departamento aún.

MySQL.

```
mysql> select * from proyectos
-> where (fechainicio < '2013-01-01' and dpto = 'IT')
-> or dpto is null;
```

codproy	nombre	fechainicio	dpto	responsable
T0451	Consejería de Educación	2012-05-24	IT	23456789B
U324	Oceanográfico	2012-09-29	NULL	NULL

2 rows in set (0.05 sec)

Oracle.

```
SQL> select * from proyectos
2  where (fechainicio < '01/01/2013' and Dpto = 'IT')
3  or Dpto is null;
```

CODPROY	NOMBRE	FECHAINI	DPTO	RESPONSABL
T0451	Consejería de Educación	24/05/12	IT	23456789B
U324	Oceanográfico	29/09/12		

6.3.5 Consulta 14

Mostrar todos los datos de los empleados que no están trabajando en un proyecto o bien están trabajando en la empresa desde antes del 1 de enero de 2011 y pertenecen al departamento de contabilidad (CONT) o al almacén (ALM).

MySQL.

```
mysql> select * from empleados
-> where codp is null
-> or ( fechaalta < '2011-01-01' and
-> (dpto = 'CONT' or dpto = 'ALM'));
```

dni	nombre	especialidad	fechaalta	dpto	codp
12345678A	Alberto Gil	Contable	2010-12-10	CONT	MAD20
23456789B	Mariano Sanz	Informática	2011-10-04	IT	NULL
67890123F	Roberto Milán	Logística	2010-05-02	ALM	NULL

3 rows in set (0.00 sec)

Oracle.

```
SQL> select * from empleados
2  where codp is null
3  or <fechaalta < '01/01/2011'
4  and <dpto = 'CONT' or dpto = 'ALM'>;
```

DNI	NOMBRE	ESPECIALIDAD	FECHAALT
DPTO	CODP		
12345678A CONT	Alberto Gil MAD20	Contable	10/12/10
23456789B IT	Mariano Sanz	Informática	04/10/11
67890123F ALM	Roberto Milán	Logística	05/02/10

6.3.6 Consulta 15

Mostrar todos los datos de los empleados que se han dado de alta en la empresa desde el día 1 de junio de 2011 hasta el día 1 de Noviembre del mismo año.

MySQL.

```
mysql> select * from empleados
-> where fechaalta >= '2011-06-01' and fechaalta <= '2011-11-01';
+-----+-----+-----+-----+-----+-----+
| dni   | nombre   | especialidad | fechaalta | dpto | codp |
+-----+-----+-----+-----+-----+-----+
| 23456789B | Mariano Sanz | Informática | 2011-10-04 | IT   | NULL |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Oracle.

```
SQL> select * from empleados
2  where fechaalta >= '01/06/2011' and fechaalta <= '01/11/2011';
DNI      NOMBRE      ESPECIALIDAD      FECHAALT
-----
DPTO      CODP
-----
23456789B Mariano Sanz      Informática      04/10/11
IT
```

6.3.7 Consulta 16

Mostrar todos los datos de los productos cuyo precio está entre 10 y 20 euros ambos incluidos.

```
mysql> select * from productosped
-> where precio >= 10 and precio <= 20;
+-----+-----+-----+
| RefeProducto | NombreProducto | Precio |
+-----+-----+-----+
| HM12         | HOOP MUSICAL   | 12.8   |
| PM30         | PELUCHE MAYA   | 15     |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

6.4 BETWEEN

Con **BETWEEN**, seleccionamos solo los valores entre dos cotas.

En las dos últimas consultas, vemos que se están utilizando filtros para comparar un rango desde un valor a otro utilizando dos operadores de relación y el operador lógico AND.

Sin embargo, SQL nos proporciona el filtro BETWEEN (que significa entre dos valores en inglés) que nos facilita la forma de crear estos filtros.

Podemos utilizar este filtro de la siguiente forma:

```
nombre_campo BETWEEN valor_1 AND valor_2
```

El filtro será verdadero para todos los valores del campo que se encuentren entre valor_1 y valor_2 ambos incluidos.

Veamos los ejemplos anteriores utilizando esta nueva cláusula:

6.4.1 Consulta 17

Mostrar todos los datos de los empleados que se han dado de alta en la empresa desde el día 1 de junio de 2011 hasta el día 1 de Noviembre del mismo año (la misma que la consulta 15).

MySQL.

```
mysql> select * from empleados
-> where fechaalta BETWEEN '2011-06-01' AND '2011-11-01';
+-----+-----+-----+-----+-----+-----+
| dni      | nombre      | especialidad | fechaalta | dpto | codp |
+-----+-----+-----+-----+-----+-----+
| 23456789B | Mariano Sanz | Informática  | 2011-10-04 | IT   | NULL |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Oracle.

```
SQL> select * from empleados
2 where fechaalta BETWEEN '01/06/2011' AND '01/11/2011';
```

DNI	NOMBRE	ESPECIALIDAD	FECHAALTI
DPTO	CODP		
23456789B	Mariano Sanz	Informática	04/10/11
IT			

6.4.2 Consulta 18

Mostrar todos los datos de los productos cuyo precio está entre 10 y 20 euros ambos incluidos (la misma que la consulta 16).

```
mysql> select * from productosped
-> where precio BETWEEN 10 AND 20;
```

RefeProducto	NombreProducto	Precio
HM12	HOOP MUSICAL	12.8
PM30	PELUCHE MAYA	15

2 rows in set (0.05 sec)

6.5 IN / NOT IN

Con **IN / NOT IN**, podemos filtrar valores dentro / fuera de un conjunto de datos.

Se utilizaría de la siguiente forma:

```
nombre_campo [NOT] IN (Valor_1, Valor2, Valor_3, ....)
```

6.5.1 Consulta 19

Mostrar todos los datos de los empleados de los departamentos de Contabilidad (CONT), Informática (IT) y Almacén (ALM).

Podemos realizar esta consulta sin emplear el nuevo operador IN de la siguiente forma:

```
mysql> select * from empleados
-> where dpto = 'CONT'
-> OR dpto = 'IT'
-> OR dpto = 'ALM';
```

dni	nombre	especialidad	fechaalta	dpto	codp
12345678A	Alberto Gil	Contable	2010-12-10	CONT	MAD20
23456789B	Mariano Sanz	Informática	2011-10-04	IT	NULL
45678901D	Ana Silván	Informática	2012-11-25	IT	MAD20
67890123F	Roberto Milán	Logística	2010-05-02	ALM	NULL
78901234G	Rafael Colmenar	Informática	2013-06-10	IT	T0451

```
5 rows in set (0.00 sec)
```

Pero utilizando el operador IN que acabamos de ver puede resultar más cómodo y sencillo.

```
mysql> select * from empleados
-> where dpto IN ('CONT','IT','ALM');
```

dni	nombre	especialidad	fechaalta	dpto	codp
12345678A	Alberto Gil	Contable	2010-12-10	CONT	MAD20
23456789B	Mariano Sanz	Informática	2011-10-04	IT	NULL
45678901D	Ana Silván	Informática	2012-11-25	IT	MAD20
67890123F	Roberto Milán	Logística	2010-05-02	ALM	NULL
78901234G	Rafael Colmenar	Informática	2013-06-10	IT	T0451

```
5 rows in set (0.03 sec)
```

6.5.2 Consulta 20

Utilizando el operador IN mostrar todos los proveedores cuyo código postal no es 41400 ni 28005 ni 45600.

```
mysql> select * from proveedores
-> where CodPostal NOT IN ('41400','28005','45600');
```

CodProveedor	NombreProveedor	CodPostal
BA843	CARMELO DIAZ, S.L.	06004

```
1 row in set (0.00 sec)
```

6.6 LIKE / NOT LIKE

Con LIKE, filtramos información en campos de tipo alfanumérico (cadenas de caracteres) que coincidan con un patrón de búsqueda especificado.

Para utilizarlo emplearemos el filtro de la siguiente forma:

Campo [NOT] LIKE 'Patrón_de_búsqueda'

El patrón de búsqueda puede contener cualquier combinación de caracteres y comodines entre comillas simples (en MySQL también pueden ser comillas dobles). Los comodines son dos: el subrayado "_" y el porcentaje "%". El primero permite substituir a un carácter cualquiera y el porcentaje representa a cualquier conjunto de caracteres.

6.6.1 Consulta 21

Mostrar todos los datos de los empleados que trabajan en un proyecto cuyo código comienza por MA.

```
mysql> select * from empleados
-> where codp like 'MA%';
```

dni	nombre	especialidad	fechaalta	dpto	codp
12345678A	Alberto Gil	Contable	2010-12-10	CONT	MAD20
45678901D	Ana Silván	Informática	2012-11-25	IT	MAD20

2 rows in set (0.00 sec)

6.6.2 Consulta 22

Mostrar todos los datos de los empleados que pertenecen a un departamento con un código de dos caracteres.

```
mysql> select * from empleados
-> where dpto like '__';
```

dni	nombre	especialidad	fechaalta	dpto	codp
23456789B	Mariano Sanz	Informática	2011-10-04	IT	NULL
45678901D	Ana Silván	Informática	2012-11-25	IT	MAD20
78901234G	Rafael Colmenar	Informática	2013-06-10	IT	T0451

3 rows in set (0.00 sec)

6.6.3 Consulta 23

Mostrar todos los datos de los empleados cuyo DNI incluye la secuencia de números 567

```
mysql> select * from empleados
-> where dni like "%567%";
```

dni	nombre	especialidad	fechaalta	dpto	codp
12345678A	Alberto Gil	Contable	2010-12-10	CONT	MAD20
23456789B	Mariano Sanz	Informática	2011-10-04	IT	NULL
45678901D	Ana Silván	Informática	2012-11-25	IT	MAD20

3 rows in set (0.00 sec)

6.6.4 Consulta 24

Mostrar todos los datos de los proveedores de productos cuyo nombre termina en S.A.

```
mysql> select * from proveedores
-> where nombreproveedor like '%S.A.';
+-----+-----+-----+
| CodProveedor | NombreProveedor | CodPostal |
+-----+-----+-----+
| MA280       | TOYPLAY, S.A.   | 28005     |
| T0342       | JUGUETOS, S.A.  | 45600     |
+-----+-----+-----+
2 rows in set (0.05 sec)
```

7. ORDENACIÓN DEL RESULTADO

SQL nos proporciona la posibilidad de ordenar el conjunto de filas que obtenemos como resultado de una consulta. Para realizar la ordenación emplearemos la cláusula ORDER BY al final de nuestra instrucción SQL.

ORDER BY {nombre_columna | expr | posición} [ASC | DESC] , ...

Para indicar el tipo de ordenación que deseamos realizar podemos indicar el nombre de la columna a utilizar o bien una expresión o la posición de la columna. Además, para cada una de ellas, lo podemos hacer de forma ascendente (para campos alfanuméricos de la A a la Z) o descendente (para campos alfanuméricos de Z a la A).

Por ejemplo:

- **ORDER BY nombre** : ordenaría las filas por el campo nombre en forma ascendente (es el tipo de ordenación por defecto).
- **ORDER BY nombre DESC** : ordenaría las filas por el campo nombre en forma descendente.
- **ORDER BY precio** : ordenaría las filas por el campo precio de menor a mayor
- **ORDER BY precio DESC** : ordenaría las filas por el campo precio de mayor a menor
- **ORDER BY fecha DESC, NumPedido, NumLinea** : ordenaría las filas por el campo fecha de forma descendente (primero las de mayor fecha), después las que tengan la misma fecha aparecerían ordenadas por el número del pedido de menor a mayor y por último para los que tengan la misma fecha y el mismo número de pedido se ordenarían por el número de línea de menor a mayor.

También se puede poner el número del campo en lugar de su nombre; sin embargo, es aconsejable utilizar siempre el nombre del campo, pues si se modifica la estructura de la tabla y se cambia el orden de los campos que la forman, la consulta que habíamos creado puede dejar de funcionar como esperábamos.

7.1 Consulta 25

Mostrar todos los datos de los departamentos ordenados por el código de departamento.

```
mysql> select * from departamentos order by CodDpto;
+-----+-----+-----+
| CodDpto | Nombre          | Ubicacion          |
+-----+-----+-----+
| ADM     | Administración  | Planta quinta U2   |
| ALM     | Almacén         | Planta baja U1     |
| CONT    | Contabilidad    | Planta quinta U1   |
| IT      | Informática     | Planta sótano U3   |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

7.2 Consulta 26

Mostrar el nombre y el departamento de los empleados ordenado por el nombre del empleado.

```
mysql> select Nombre, Dpto from empleados
-> order by Nombre;
+-----+-----+
| Nombre          | Dpto |
+-----+-----+
| Alberto Gil     | CONT |
| Ana Silván     | IT   |
| Mariano Sanz    | IT   |
| Rafael Colmenar | IT   |
| Roberto Milán   | ALM  |
+-----+-----+
5 rows in set (0.00 sec)
```

7.3 Consulta 27

Mostrar el departamento y el nombre del empleado ordenado por departamento y dentro del mismo departamento ordenado por el nombre del empleado.

```
mysql> select Dpto, Nombre from empleados
-> Order By Dpto, Nombre;
+-----+-----+
| Dpto | Nombre          |
+-----+-----+
| ALM  | Roberto Milán  |
| CONT | Alberto Gil     |
| IT   | Ana Silván     |
| IT   | Mariano Sanz    |
| IT   | Rafael Colmenar |
+-----+-----+
5 rows in set (0.00 sec)
```

7.4 Consulta 28

Mostrar el departamento, el nombre del empleado y el proyecto en el que trabaja ordenado por departamento y dentro del mismo departamento ordenado por empleado en forma descendente.

```
mysql> select Dpto, Nombre, Codp from empleados
-> Order by Dpto, Nombre DESC;
+-----+-----+-----+
| Dpto | Nombre      | Codp |
+-----+-----+-----+
| ALM  | Roberto Milán | NULL |
| CONT | Alberto Gil  | MAD20 |
| IT   | Rafael Colmenar | T0451 |
| IT   | Mariano Sanz  | NULL |
| IT   | Ana Silván   | MAD20 |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

7.5 Consulta 29

Mostrar los empleados que no están asignados a un proyecto ordenados por el nombre del empleado.

```
mysql> Select * from empleados
-> where codp is null
-> order by nombre;
+-----+-----+-----+-----+-----+-----+
| dni      | nombre      | especialidad | fechaalta | dpto | codp |
+-----+-----+-----+-----+-----+-----+
| 23456789B | Mariano Sanz | Informática  | 2011-10-04 | IT   | NULL |
| 67890123F | Roberto Milán | Logística    | 2010-05-02 | ALM  | NULL |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

7.6 Consulta 30

Mostrar todos los datos de los pedidos de los proveedores cuyo código comienza por TO mostrando primero los más recientes.

```
mysql> select * from pedidos
-> where Proveedor like 'TO%'
-> Order by Fecha DESC;
+-----+-----+-----+
| NumPedido | Fecha      | Proveedor |
+-----+-----+-----+
| 4         | 2013-06-14 | T0342     |
| 1         | 2013-06-10 | T0342     |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

7.7 Consulta 31

Mostrar el número de pedido y la cantidad pedida de los pedidos realizados del producto P3R20 ordenados por la cantidad de mayor a menor.

```
mysql> select Numpedido, Cantidad from productospedido
-> where RefeProducto = 'P3R20'
-> Order by cantidad DESC;
+-----+-----+
| Numpedido | Cantidad |
+-----+-----+
|         5 |       18 |
|         2 |       15 |
|         3 |       10 |
+-----+-----+
3 rows in set (0.00 sec)
```

8. FUNCIONES AGREGADAS

SQL también nos permite realizar algunos cálculos con los datos contenidos en las columnas. Las funciones más utilizadas, sobre todo con las columnas numéricas, son:

- **SUM** calcula la suma de los valores de la columna.
- **AVG** calcula la media aritmética de los valores de la columna.
- **COUNT** devuelve el número de elementos que tiene la columna.
- **MAX** devuelve el valor máximo de la columna.
- **MIN** devuelve el valor mínimo de la columna.

El uso de count nos permite contar cuántas filas tiene un resultado. Por ejemplo, para saber cuántos empleados tenemos, se utilizaría count(*), o bien podemos contar el número de elementos de un campo con count(nombre_campo) teniendo en cuenta que no contará los valores nulos.

Además con count también se suele emplear la palabra reservada DISTINCT para que solamente cuente los que son valores distintos. Para ello utilizaríamos count(DISTINCT nombre_campo).



Importante

Es recomendable poner un alias para los agregados, ya que clarifica lo que estamos obteniendo (por ejemplo, avg(salario) se entendería mejor con avg(salario) AS 'Salario Medio'. Ahora bien, **en los agregados NO debemos ordenar por alias**, ya que suele fallar. En su lugar, usaremos el mismo agregado o un índice.

Veamos cómo emplear estas funciones de resumen:

8.1 Consulta 32

Mostrar la suma de los precios de los productos disponibles;

```
mysql> select sum<precio> from productosped;
+-----+
| sum<precio> |
+-----+
| 107.25000095367 |
+-----+
1 row in set (0.05 sec)
```

8.2 Consulta 33

Mostrar el precio medio de los productos disponibles;

```
mysql> select avg<precio> from productosped;
+-----+
| avg<precio> |
+-----+
| 17.875000158946 |
+-----+
1 row in set (0.00 sec)
```

8.3 Consulta 34

Mostrar el precio máximo y el precio mínimo de los productos disponibles;

```
mysql> select max<precio>,min<precio> from productosped;
+-----+-----+
| max<precio> | min<precio> |
+-----+-----+
| 31.75 | 3 |
+-----+-----+
1 row in set (0.00 sec)
```

8.4 Consulta 35

Mostrar el número de proveedores que hay.

```
mysql> select count(*) from proveedores;
+-----+
| count(*) |
+-----+
|         4 |
+-----+
1 row in set (0.00 sec)
```

8.5 Consulta 36

Mostrar el número de proyectos que hay.

```
mysql> select count(*) from proyectos;
+-----+
| count(*) |
+-----+
|         3 |
+-----+
1 row in set (0.00 sec)
```

8.6 Consulta 37

Mostrar el número de proyectos que hay asignados a un departamento.

```
mysql> select count(Dpto) from proyectos;
+-----+
| count(Dpto) |
+-----+
|           2 |
+-----+
1 row in set (0.00 sec)
```

Otra forma algo más compleja sería:

```
mysql> select count(*) from proyectos
-> where Dpto is not null;
+-----+
| count(*) |
+-----+
|         2 |
+-----+
1 row in set (0.00 sec)
```

8.7 Consulta 38

Cuántas especialidades distintas tienen los empleados de la empresa.

```
mysql> select count(distinct especialidad) from empleados;
+-----+
| count(distinct especialidad) |
+-----+
| 3 |
+-----+
1 row in set (0.00 sec)
```

9. CÁLCULOS ARITMÉTICOS

Los operadores + (suma), - (resta), * (multiplicación) y / (división), se pueden utilizar para hacer cálculos en las consultas. Cuando se utilizan como expresión en una consulta SELECT, no modifican los datos originales sino que como resultado de la vista generada por SELECT, aparece una nueva columna.

Ejemplo. Imaginemos que queremos calcular el precio de los productos con IVA:

```
SELECT nombreProducto, Precio, Precio*1.21 FROM productosPed;
```

```
mysql> SELECT nombreProducto, Precio, Precio*1.21 FROM productosPed;
+-----+-----+-----+
| nombreProducto | Precio | Precio*1.21 |
+-----+-----+-----+
| AVION FK20     | 31.75  | 38.4175     |
| BOLA BOOM      | 22.2   | 26.86200092315674 |
| HOOP MUSICAL   | 12.8   | 15.488000230789185 |
| NAIPES PETER PARKER | 3      | 3.63        |
| PATINETE 3 RUEDAS | 22.5   | 27.224999999999998 |
| PELUCHE MAYA   | 15     | 18.15       |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

Esa consulta obtiene tres columnas. La tercera tendrá como nombre la expresión utilizada, para poner un alias basta utilizar dicho alias tras la expresión:

```
SELECT nombreProducto, Precio, Precio*1.21 AS Precio_IVA FROM productosPed;
```

```
mysql> SELECT nombreProducto, Precio, Precio*1.21 AS Precio_IVA FROM productosPed;
```

nombreProducto	Precio	Precio_IVA
AVION FK20	31.75	38.4175
BOLA BOOM	22.2	26.862000092315674
HOOP MUSICAL	12.8	15.4880000230789185
NAIPES PETER PARKER	3	3.63
PATINETE 3 RUEDAS	22.5	27.224999999999998
PELUCHE MAYA	15	18.15

```
6 rows in set (0.00 sec)
```

La prioridad de esos operadores es la habitual en matemáticas: tienen más prioridad la multiplicación y división, después la suma y la resta. En caso de igualdad de prioridad, se realiza primero la operación que esté más a la izquierda. Como es lógico se puede evitar cumplir esa prioridad usando paréntesis; el interior de los paréntesis es lo que se ejecuta primero.

Cuando una expresión aritmética se calcula sobre valores NULL, el resultado es el propio valor NULL.