

---

# UD6: MODELO FÍSICO DQL

## Parte 1. DQL Básico

---

### Tema Extendido

---

Bases de Datos (BD)  
CFGs DAM/DAW

Abelardo Martínez y Pau Miñana.  
Basado y modificado de Sergio Badal y Raquel Torres.  
Curso 2023-2024

---

# ÍNDICE

---

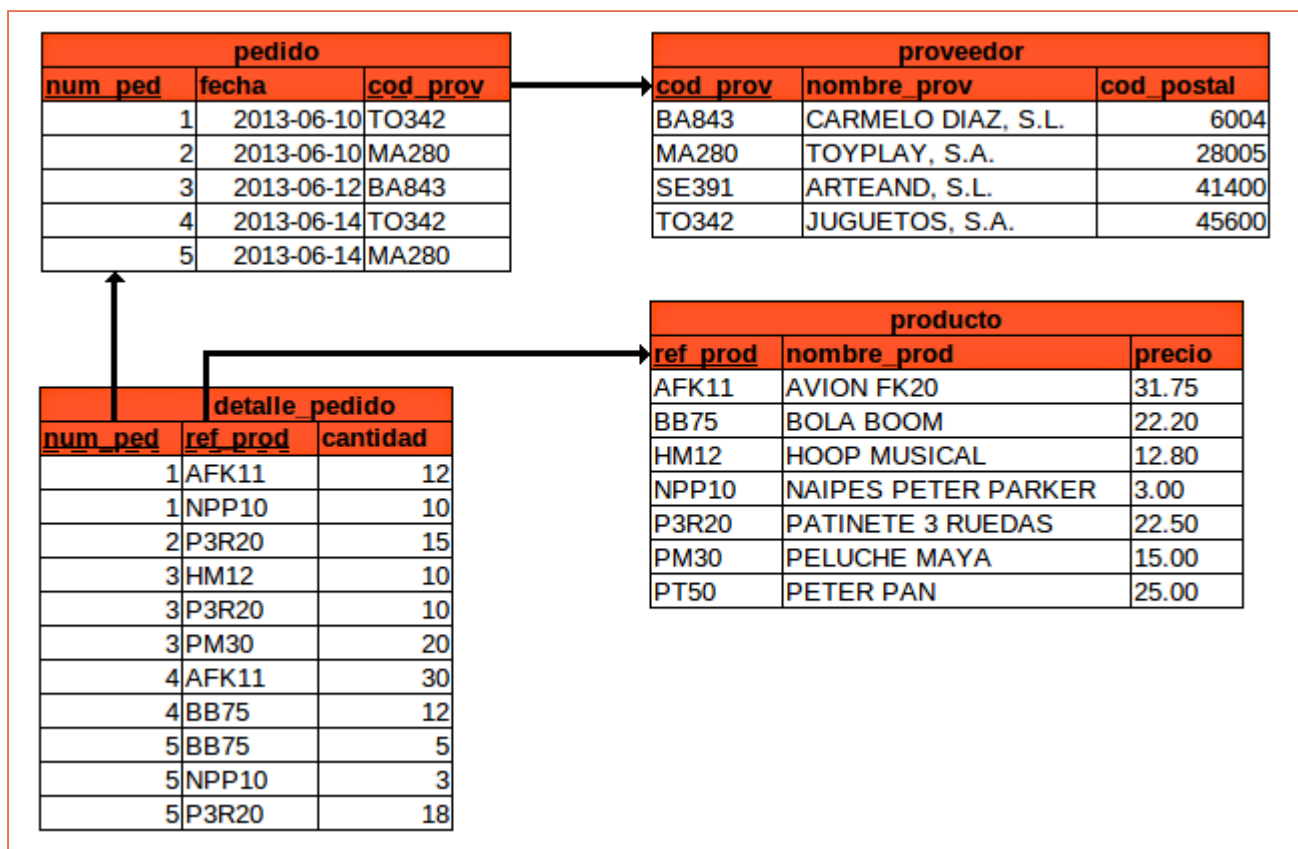
- [1. Introducción](#)
- [2. DQL](#)
  - [2.1. Diagrama físico](#)
- [3. SELECT](#)
- [4. FROM](#)
  - [4.1. Alias de tabla](#)
- [5. Proyección](#)
  - [5.1. NombreTabla.NombreColumna](#)
  - [5.2. Alias de columna](#)
  - [5.3. Expr\\_col: Cálculos y funciones](#)
  - [5.4. DISTINCT](#)
- [6. WHERE](#)
  - [6.1. Operadores relacionales](#)
  - [6.2. IS \[NOT\] NULL](#)
  - [6.3. Operadores lógicos](#)
  - [6.4. BETWEEN](#)
  - [6.5. \[NOT\] IN](#)
  - [6.6. \[NOT\] LIKE](#)
- [7. Ordenación de filas en la respuesta](#)
- [8. Índices, ordenación y eficiencia.](#)
- [9. Funciones agregadas](#)
- [10. Limitar las filas mostradas en la respuesta](#)
- [11. Bibliografía](#)

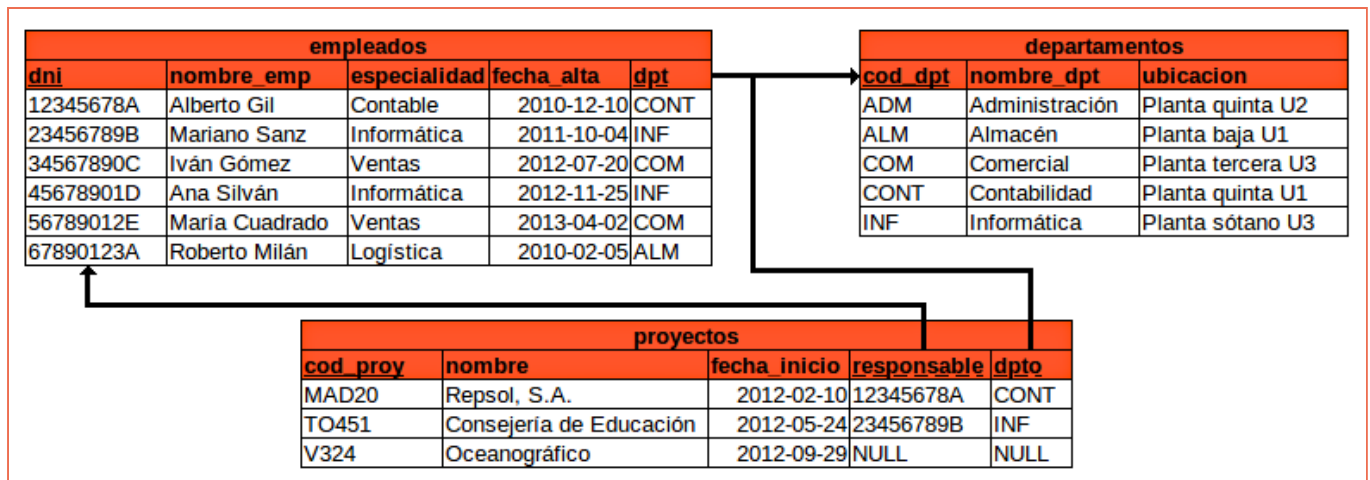
# 1. Introducción

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 (a menos que haya cambios en el contexto en el que fue creada) sin embargo las consultas se realizarán durante toda la vida de la base de datos. Por eso el aprender a realizar consultas y dominar el SQL en este aspecto es fundamental.

Puedes probar a realizar consultas en cualquiera de las bases de datos vistas hasta ahora, pero en el desarrollo de esta unidad se usan concretamente las bases de datos *db\_tienda\_friki* y *db\_empresa* para ilustrar los ejemplos pertinentes. En el aula virtual puedes encontrar los scripts de creación de las mismas y esta imagen para tener clara su estructura y datos:





Todos los ejemplos de esta unidad son válidos para SQL estándar salvo que se indique lo contrario.

## 2. DQL

📖 El **DQL** (Data Query Language) es la parte del lenguaje SQL dedicada a 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

⚠ Recuerda que aunque el lenguaje *SQL no es realmente sensible a mayúsculas/minúsculas*, en los nombres puede depender del SGBD o incluso del Sistema Operativo, como en MySQL. Por tanto, te recomendamos que seas fiel a la sintaxis usada al crear los metadatos (tablas, atributos, restricciones...). Además, se considera buena praxis:

- Usar mayúsculas para las palabras reservadas de SQL (SELECT, INSERT, FROM, WHERE...).
- Usar solo minúsculas o CamelCase en los nombres para los metadatos.
- Usar mayúsculas en los alias de las tablas para reconocerlos mejor en las consultas.

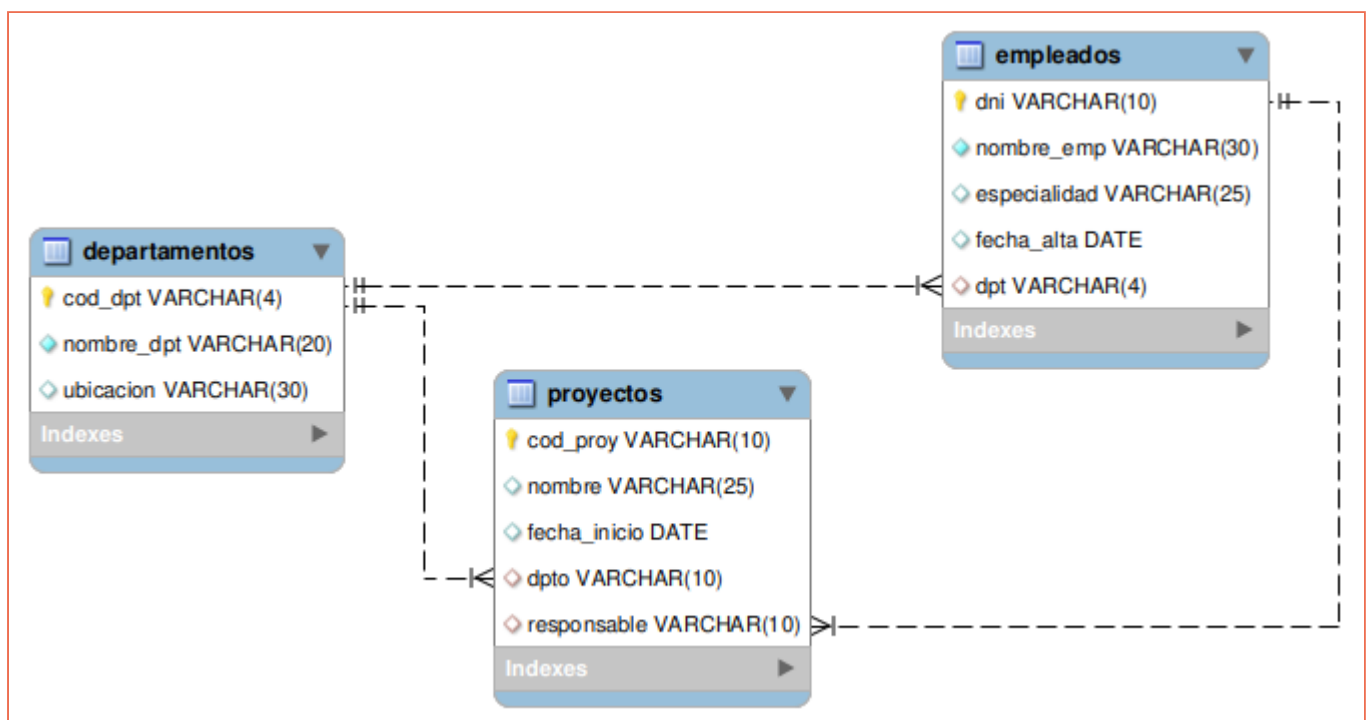
## 2.1. Diagrama físico

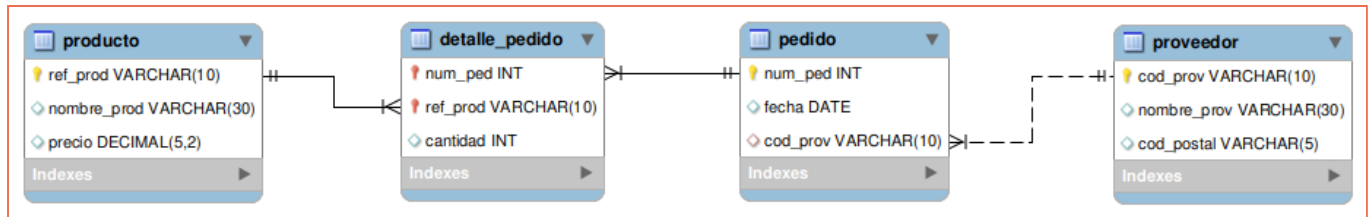
Los SGBD generalmente pueden dibujar un diagrama físico de sus bases de datos, estos representan las tablas y representan las "relaciones" entre ellas, de modo que resultan muy útiles para diseñar las consultas sin tener que interpretar un diagrama E-R ni comprobar la base de datos mediante comandos o consultando sus instrucciones de creación. Estos diagramas usan normalmente la notación de pata de gallo (crow's foot); puedes encontrar más información sobre la misma en el siguiente link:

<https://www.freecodecamp.org/news/crows-foot-notation-relationship-symbols-and-how-to-read-diagrams/>

⚠ Ten presente que muchas veces se llama a estos diagramas como diagramas de Entidad-Relación (los propios SGBD lo hacen), puesto que en cierto sentido expresan la misma información. De hecho, es cierto que se podría usar la notación para un modelo E-R pero en realidad estos diagramas ya tienen las relaciones "convertidas", aplicando claves foráneas y creando tablas para las relaciones pertinentes. Por tanto, el diagrama ya ha pasado por su interpretación a modelo relacional.

A continuación se muestran los diagramas físicos de las 2 bases de datos que se usarán en los ejemplos:





### 3. SELECT

La sintaxis más básica para una consulta es:

```
SELECT [DISTINCT] expr_col1 [AS nom_alias] [,expr_col2 [AS
nom_alias]]...
FROM nom_tabla1 [nom_alias]
[WHERE condiciones]
[ORDER BY expr_col1 [DESC] [, expr_col2 [DESC]]...]
```

Donde expr\_col (expresión\_columna) puede ser:

- Una columna.
- El comodín \* (todas las columnas) o alguna de sus variantes.
- Una constante.
- Una expresión aritmética.
- Una o varias funciones anidadas.

En las próximas semanas se ampliará esta sintaxis, pero de momento se limita a estas opciones. En los siguientes puntos se procede a analizarlas en detalle.

### 4. FROM

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

En este documento nos limitamos a consultar elementos de una sola tabla.

Supongamos que queremos ver los datos de los proyectos que lleva la empresa. Simplemente usaremos la palabra reservada SELECT seguida del comodín \* para mostrar todos los atributos de la tabla. Después colocamos la palabra reservada FROM seguida del nombre de la tabla a consultar. Recuerda que debes estar conectado a la base de datos a la que pertenece la tabla `use db_empresa`.

```
mysql> SELECT * FROM proyectos;
+-----+-----+-----+-----+-----+
| cod_proy | nombre                | fecha_inicio | dpto | responsable |
+-----+-----+-----+-----+-----+
| MAD20    | Repsol, S.A.          | 2012-02-10   | CONT | 12345678A   |
| T0451    | Consejería de Educación | 2012-05-24   | INF  | 23456789B   |
| V324     | Oceanográfico         | 2012-09-29   | NULL | NULL        |
+-----+-----+-----+-----+-----+
3 rows in set (0,00 sec)
```

Si se tienen los permisos suficientes, incluso se pueden consultar tablas de otras bases de datos, incluyendo el nombre de la base de datos antes del nombre de la tabla separados por un punto. Vamos a consultar de nuevo la tabla pero conectados desde la otra base de datos.

```
mysql> use db_tienda_friki
Database changed
mysql> SELECT * FROM proyectos;
ERROR 1146 (42S02): Table 'db_tienda_friki.proyectos' doesn't exist
mysql> SELECT * FROM db_empresa.proyectos;
+-----+-----+-----+-----+-----+
| cod_proy | nombre                | fecha_inicio | dpto | responsable |
+-----+-----+-----+-----+-----+
| MAD20    | Repsol, S.A.          | 2012-02-10   | CONT | 12345678A   |
| T0451    | Consejería de Educación | 2012-05-24   | INF  | 23456789B   |
| V324     | Oceanográfico         | 2012-09-29   | NULL | NULL        |
+-----+-----+-----+-----+-----+
3 rows in set (0,00 sec)
```

## 4.1. Alias de tabla

Otra opción que existe es el uso de **alias** para referenciar a la tabla. Estos alias no se quedan almacenados en ningún sitio, se usan simplemente en la propia consulta y se usan para acortar las llamadas a la tabla, como veremos en el próximo apartado.

📖 Para los **alias de las tablas** se suele usar el **mínimo número de letras MAYÚSCULAS** posible para distinguir esa tabla dentro de la consulta. Simplemente se añade el alias escogido detrás del nombre de la tabla, separado por un espacio.

Aunque por el momento no aporta nada, a modo de referencia la consulta anterior usando un alias quedaría del siguiente modo:

```
mysql> SELECT * FROM proyectos P;
```


Acuérdate de usar antes `use db_empresa` para conectarte de nuevo a la base de datos apropiada.

## 5. Proyección

En la proyección se especifica qué campos debe contener la respuesta de nuestra consulta. Aunque lo más habitual son atributos de la tabla, éstos también pueden incluir funciones y cálculos. Más adelante se amplía la información sobre su uso.

Imaginemos que queremos solo el nombre de los proyectos y su fecha de inicio. En ese caso, en vez del comodín \*, usamos tras el SELECT el nombre de los atributos a mostrar separados por comas.

```
mysql> SELECT nombre, fecha_inicio FROM proyectos;
+-----+-----+
| nombre                | fecha_inicio |
+-----+-----+
| Repsol, S.A.          | 2012-02-10   |
| Consejería de Educación | 2012-05-24   |
| Oceanográfico         | 2012-09-29   |
+-----+-----+
3 rows in set (0,00 sec)
```

 Aunque es perfectamente correcto y muy usado en ámbitos académicos o pruebas, se considera una **mala praxis usar indiscriminadamente SELECT \*** en entornos de producción, por el consumo innecesario de recursos que suele producir, sobretudo en tablas grandes de miles de registros. Se recomienda limitar las consultas a las columnas estrictamente necesarias.

### 5.1. NombreTabla.NombreColumna

Opcionalmente se puede usar el nombre de la tabla antes del nombre de un atributo separados con un punto, pero esto **en ningún caso nos exime de poner la tabla en el FROM**. En este ejemplo no tiene sentido utilizar esta nomenclatura; sin embargo se vuelve obligatorio al usar varias tablas con atributos que compartan su nombre, ya que es el único modo de distinguirlos. Por ejemplo, si en nuestras tablas los campos nombre\_emp y nombre\_dpt no llevasen el sufijo, en una consulta que incluya las 3 tablas el atributo nombre quedaría indeterminado, al no saber a cual de las 3 se refiere. Y esto, por desgracia, suele ser habitual en muchas bases de datos que usan nombres demasiado genéricos.

```
mysql> SELECT proyectos.nombre, proyectos.fecha_inicio FROM proyectos;
+-----+-----+
| nombre                | fecha_inicio |
+-----+-----+
| Repsol, S.A.          | 2012-02-10   |
| Consejería de Educación | 2012-05-24   |
| Oceanográfico         | 2012-09-29   |
+-----+-----+
3 rows in set (0,00 sec)
```



El nombre de las tablas puede ser largo y tenerlo que escribir delante de todos los campos (o de una parte de ellos) es un poco tedioso, además de alargar el texto de la consulta y dificultar una rápida interpretación del mismo. Por ello, crear **alias para las tablas** en este tipo de consultas ayuda agilizar el proceso y facilitar la interpretación.

```
mysql> SELECT P.nombre, P.fecha_inicio FROM proyectos P;
+-----+-----+
| nombre          | fecha_inicio |
+-----+-----+
| Repsol, S.A.    | 2012-02-10  |
| Consejería de Educación | 2012-05-24  |
| Oceanográfico   | 2012-09-29  |
+-----+-----+
3 rows in set (0,00 sec)
```

En este caso concreto sigue siendo innecesario recurrir a esto, puesto que la primera opción sin usar el nombre de la tabla no admite confusión alguna, pero se puede ver el efecto para cuando usar esta nomenclatura resulte imprescindible.

## 5.2. Alias de columna

Por otro lado, las columnas de las consultas también admiten el uso de alias, que será simplemente el nombre que mostrará en la respuesta de la columna. Esto resulta especialmente útil cuando se usan funciones o cálculos (puesto que el nombre de la columna por defecto es la propia expresión) y cuando se muestran varias columnas con el mismo nombre.

📖 Para usar **alias en las columnas** se añade tras la columna la palabra **AS** seguida del **nombre** deseado. Si se quiere usar varias palabras en un alias se pueden poner entre comillas (simples o dobles) o, como es habitual, separar las palabras con un guion bajo.

```
mysql> SELECT nombre AS nombre_proyecto, fecha_inicio AS 'fecha inicio'
-> FROM proyectos;
+-----+-----+
| nombre_proyecto | fecha inicio |
+-----+-----+
| Repsol, S.A.    | 2012-02-10  |
| Consejería de Educación | 2012-05-24  |
| Oceanográfico   | 2012-09-29  |
+-----+-----+
3 rows in set (0,00 sec)
```

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

En realidad la versión actual de MySQL admite crear ambos alias tanto con el AS como sin el mismo, pero se recomienda seguir el estándar indicado arriba.

### 5.3. Expr\_col: Cálculos y funciones

Además de usar nombres de columnas existentes en las tablas, las columnas de las respuestas de una consulta pueden incluir **funciones y operaciones** entre columnas o entre éstas y números o cadenas de texto.

De hecho pueden incluir constantes e incluso operaciones que no impliquen a las columnas en absoluto, donde la respuesta será un resultado que se repetirá en todas las filas de la respuesta. Obviamente esta opción tiene aplicaciones muy limitadas y concretas, por lo que por el momento resulta irrelevante.

Las operaciones aritméticas  $+$  (suma),  $-$  (resta),  $*$  (multiplicación),  $/$  (división),  $\%$  (módulo) y  $^$  (potenciación) són las más típicas, y se pueden utilizar para hacer cálculos en las consultas. Cabe tener en cuenta que no modifican los datos originales en absoluto ni se almacenan en ningún sitio, simplemente se muestran en la respuesta de la consulta.

La prioridad de los 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.

Quando se usan expresiones en las columnas de la proyección, su nombre en la respuesta es la propia expresión, lo que muchas veces resulta poco práctico e incluso confuso. Por ello lo más habitual **es usar alias en las columnas con operaciones o funciones**.

Mostramos ahora todos los datos de los productos más el precio con IVA.

```
mysql> SELECT *, precio*1.21 AS precio_iva FROM producto;
+-----+-----+-----+-----+
| ref_prod | nombre_prod | precio | precio_iva |
+-----+-----+-----+-----+
| AFK11    | AVION FK20  | 31.75  | 38.4175    |
| BB75     | BOLA BOOM   | 22.20  | 26.8620    |
| HM12     | HOOP MUSICAL | 12.80  | 15.4880    |
| NPP10    | NAIPES PETER PARKER | 3.00  | 3.6300     |
| P3R20    | PATINETE 3 RUEDAS | 22.50  | 27.2250    |
| PM30     | PELUCHE MAYA | 15.00  | 18.1500    |
| PT50     | PETER PAN   | 25.00  | 30.2500    |
+-----+-----+-----+-----+
7 rows in set (0,00 sec)
```


```
mysql> SELECT *, precio*1.21 FROM producto;
+-----+-----+-----+-----+
| ref_prod | nombre_prod | precio | precio*1.21 |
+-----+-----+-----+-----+
| AFK11    | AVION FK20  | 31.75  | 38.4175     |
| BB75     | BOLA BOOM   | 22.20  | 26.8620     |
| HM12     | HOOP MUSICAL | 12.80  | 15.4880     |
| NPP10    | NAIPES PETER PARKER | 3.00  | 3.6300     |
| P3R20    | PATINETE 3 RUEDAS | 22.50  | 27.2250     |
| PM30     | PELUCHE MAYA | 15.00  | 18.1500     |
| PT50     | PETER PAN   | 25.00  | 30.2500     |
+-----+-----+-----+-----+
7 rows in set (0,00 sec)
```

Como se puede observar, si no se incluye un alias el nombre de la columna es literalmente el de la operación. Además se puede combinar el comodín para mostrar todas las columnas junto con las calculadas, o especificar cada columna individualmente.

Vamos a mostrar ahora el nombre del producto, el precio sin IVA, el IVA de cada producto y sumarlos para sacar el precio con IVA.

```
mysql> SELECT nombre_prod, precio, precio*0.21 AS IVA,
-> precio+precio*0.21 AS Pr_Total FROM producto;
+-----+-----+-----+-----+
| nombre_prod | precio | IVA | Pr_Total |
+-----+-----+-----+-----+
| AVION FK20  | 31.75 | 6.6675 | 38.4175 |
| BOLA BOOM   | 22.20 | 4.6620 | 26.8620 |
| HOOP MUSICAL | 12.80 | 2.6880 | 15.4880 |
| NAIPES PETER PARKER | 3.00 | 0.6300 | 3.6300 |
| PATINETE 3 RUEDAS | 22.50 | 4.7250 | 27.2250 |
| PELUCHE MAYA | 15.00 | 3.1500 | 18.1500 |
| PETER PAN   | 25.00 | 5.2500 | 30.2500 |
+-----+-----+-----+-----+
7 rows in set (0,00 sec)
```

```
mysql> SELECT nombre_prod, precio, precio*0.21 AS IVA,
-> precio+IVA AS Pr_Total FROM producto;
ERROR 1054 (42S22): Unknown column 'IVA' in 'field list'
```

 Se pueden usar varias columnas para hacer nuevos cálculos entre ellas pero no se pueden usar alias de columna dentro de los cálculos. Aún no se han creado y por tanto los SGBD no los reconocen.


Vamos a ver ahora un ejemplo de función. La función `CONCAT()` permite concatenar cadenas de caracteres, así que la usaremos para mostrar en una única columna el nombre y referencia de los productos separados con una coma y espacios.

```
mysql> SELECT CONCAT(ref_prod,' ',nombre_prod) AS producto FROM producto;
+-----+
| producto |
+-----+
| AFK11 , AVION FK20 |
| BB75 , BOLA BOOM |
| HM12 , HOOP MUSICAL |
| NPP10 , NAIPES PETER PARKER |
| P3R20 , PATINETE 3 RUEDAS |
| PM30 , PELUCHE MAYA |
| PT50 , PETER PAN |
+-----+
7 rows in set (0,00 sec)
```

Existen multitud de funciones de distintos tipos que se pueden aplicar, y además muchas de ellas son propias de cada SGBD, por lo que aunque su estudio resulta de utilidad, excede el tiempo requerido para este curso. Puedes encontrar las funciones más típicas de MySQL en el siguiente link:

[https://www.w3schools.com/mysql/mysql\\_ref\\_functions.asp](https://www.w3schools.com/mysql/mysql_ref_functions.asp)

El último detalle a tener en cuenta es como se trata a los valores NULL en las operaciones y funciones.

 Cuando una operación aritmética se aplica sobre un valor NULL, el resultado siempre es el propio valor NULL. En la mayoría de funciones también.

Por ejemplo, si mostramos el código y el departamento concatenados con un espacio en un campo (o si se aplicasen operaciones a un campo numérico de valor NULL) de los proyectos de *db\_empresa*, se puede observar como al encontrar un valor NULL la respuesta completa de esa columna es NULL.

```
mysql> SELECT cod_proy,CONCAT(cod_proy,' ',dpto) AS proyecto FROM proyectos;
+-----+-----+
| cod_proy | proyecto |
+-----+-----+
| V324     | NULL     |
| MAD20    | MAD20 CONT |
| T0451    | T0451 INF |
+-----+-----+
3 rows in set (0,00 sec)
```

## 5.4. DISTINCT

📖 Escribir **DISTINCT** tras el **SELECT**, antes de la proyección, elimina las repeticiones en las respuestas de la consulta.

Esto es muy sencillo de ver con el ejemplo adecuado. Supongamos que queremos una lista de los proveedores a los que se ha hecho algún pedido (en *db\_tienda\_friki*). En ese caso solo necesitamos mostrar la columna *cod\_prov* en la tabla *pedido*:

```
mysql> SELECT cod_prov FROM pedido;
+-----+
| cod_prov |
+-----+
| BA843    |
| MA280    |
| MA280    |
| T0342    |
| T0342    |
+-----+
5 rows in set (0,00 sec)
```

Pero en esa lista aparece el proveedor una vez por cada pedido que ha realizado, y esto resulta innecesario. Añadiendo **DISTINCT** se evita esta repetición.

```
mysql> SELECT DISTINCT cod_prov FROM pedido;
+-----+
| cod_prov |
+-----+
| BA843    |
| MA280    |
| T0342    |
+-----+
3 rows in set (0,01 sec)
```

Conviene tener en cuenta que para que **DISTINCT** elimine una fila como repetida **todas las columnas de la respuesta deben coincidir con otra fila**. Por ejemplo, si se añade la *fecha* a la consulta anterior, el **distinct** ya no elimina ninguna fila, puesto que las fechas de los pedidos del mismo proveedor no coinciden.

La opción contraria a **DISTINCT** es **ALL**, que muestra la respuesta con repeticiones, pero es innecesario especificarlo ya que es la opción por defecto en SQL.

⚠ Usar **DISTINCT indiscriminadamente** se considera una **mala praxis**, así como usarlo para **eliminar repeticiones en una consulta mal diseñada**, que puede eliminarlas por el filtrado de resultados, puesto que es mucho menos eficiente.

## 6. WHERE

La cláusula **WHERE** filtra las filas o registros a mostrar en la respuesta, que deben cumplir con las condiciones especificadas.

Existen multitud de elementos a tener en cuenta para determinar las condiciones en esta cláusula.

### 6.1. Operadores relacionales

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

| Operador | Significado   |
|----------|---------------|
| >        | Mayor         |
| <        | Menor         |
| >=       | Mayor o igual |
| <=       | Menor o igual |
| =        | Igual         |
| <>       | Distinto      |

Recuerda que aunque **!=** es aceptado como sinónimo de distinto por muchos SGBD **no es un estándar** de SQL. Así mismo el comparador para la igualdad es un símbolo **=** simple y no doble como en otros lenguajes de programación, **no uses nunca ==**.

También se debe tener en cuenta que si la comparación de un campo se realiza con una **cadena de caracteres**, esta debe estar **entre comillas** simples o dobles. En cambio, si se realiza con un **campo numérico**, el valor **no** debe ponerse entre comillas.

Obviamente en las condiciones del WHERE también pueden aparecer columnas, funciones y cálculos, como en la proyección.

Veamos algunos ejemplos:

- Mostrar el nombre de los empleados del departamento de Informática (INF):

```
mysql> SELECT nombre_emp FROM empleados WHERE dpt='INF';
+-----+
| nombre_emp |
+-----+
| Mariano Sanz |
| Ana Silván |
+-----+
2 rows in set (0,00 sec)
```

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

```
mysql> select nombre_prod, precio FROM producto
-> WHERE precio>=20;
+-----+-----+
| nombre_prod | precio |
+-----+-----+
| AVION FK20 | 31.75 |
| BOLA BOOM | 22.20 |
| PATINETE 3 RUEDAS | 22.50 |
| PETER PAN | 25.00 |
+-----+-----+
4 rows in set (0,00 sec)
```

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

```
mysql> SELECT ref_prod AS referencia, nombre_prod FROM producto
-> WHERE precio<15;
+-----+-----+
| referencia | nombre_prod |
+-----+-----+
| HM12 | HOOP MUSICAL |
| NPP10 | NAIPES PETER PARKER |
+-----+-----+
2 rows in set (0,00 sec)
```

Algunos detalles en esta consulta:

- Se ha usado un alias para que se muestre *ref\_prod* como *referencia*.
- No es necesario añadir a la proyección los campos del filtrado.** Como puedes ver, no es imprescindible mostrar el precio, aunque sea el criterio usado en el filtro.



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


```
mysql> SELECT * FROM pedido WHERE fecha<'2013-06-12';
+-----+-----+-----+
| num_ped | fecha      | cod_prov |
+-----+-----+-----+
|      1 | 2013-06-10 | T0342    |
|      2 | 2013-06-10 | MA280    |
+-----+-----+-----+
2 rows in set (0,00 sec)
```

Aquí debemos tener cuidado, pues ya sabemos que las fechas son tratadas con formatos diferentes para su entrada como texto en los distintos SGBD. O usamos funciones de transformación del formato o ponemos el texto entre comillas en el formato correspondiente al SGBD usado; en MySQL AAAA-MM-DD.

- Mostrar todos los campos de las filas de detalle\_pedido donde el num\_ped es más grande que la cantidad + 1. Se usa esta consulta sin sentido lógico alguno con el fin académico de mostrar el uso de comparaciones entre distintas columnas en cada fila y con operaciones aritméticas.

```
mysql> SELECT * FROM detalle_pedido WHERE num_ped > cantidad + 1;
+-----+-----+-----+
| num_ped | ref_prod | cantidad |
+-----+-----+-----+
|      5 | NPP10    |      3   |
+-----+-----+-----+
1 row in set (0,00 sec)
```

## 6.2. IS [NOT] NULL

 Para comprobar valores nulos (no hay valor, no es lo mismo que 0) no se pueden usar los operadores relacionales.

Para devolver las filas donde un **parámetro es nulo se usa IS NULL** y para devolver las que el parámetro tenga cualquier valor, pero **no esté vacío, es IS NOT NULL**.

Mucho cuidado porque en MySQL usar = o <> junto con NULL siempre devuelve una respuesta vacía, pero no un error ni un warning. Recuerda que **NULL se escribe siempre sin comillas**, o se estaría comparando con la cadena de texto con esas letras.

Por ejemplo, veamos los proyectos que no están asignados a ningún departamento y los que sí.



```
mysql> SELECT * FROM proyectos WHERE dpto IS NULL;
+-----+-----+-----+-----+-----+
| cod_proy | nombre          | fecha_inicio | dpto | responsable |
+-----+-----+-----+-----+-----+
| V324     | Oceanográfico  | 2012-09-29   | NULL | NULL         |
+-----+-----+-----+-----+-----+
1 row in set (0,00 sec)

mysql> SELECT * FROM proyectos WHERE dpto IS NOT NULL;
+-----+-----+-----+-----+-----+
| cod_proy | nombre          | fecha_inicio | dpto | responsable |
+-----+-----+-----+-----+-----+
| MAD20    | Repsol, S.A.    | 2012-02-10   | CONT | 12345678A   |
| T0451    | Consejería de Educación | 2012-05-24   | INF  | 23456789B   |
+-----+-----+-----+-----+-----+
2 rows in set (0,00 sec)

mysql> SELECT * FROM proyectos WHERE dpto = NULL;
Empty set (0,00 sec)

mysql> SELECT * FROM proyectos WHERE dpto <> NULL;
Empty set (0,00 sec)
```

## 6.3. Operadores lógicos

Los operadores lógicos que permitirán crear filtros más elaborados que los vistos hasta este momento. Con ellos se pueden comprobar varias condiciones a la vez o cambiar el sentido de una condición. Los más comunes son:

| Operador | Significado   |
|----------|---|
| AND      | Devuelve verdadero si las expresiones a su izquierda y derecha son ambas verdaderas         |
| OR       | Devuelve verdadero si cualquiera de las expresiones a izquierda o derecha son verdaderas    |
| NOT      | Invierte la lógica de la expresión que está a su derecha, si era verdadera pasa a ser falsa |

- **AND es un "Y" lógico.** El resultado será verdadero si los dos elementos que une son verdaderos. La tabla de verdad del operador AND es la siguiente:

condición\_1 AND condición\_2

| condición_1 | condición_2 | resultado |
|-------------|-------------|-----------|
| V           | V           | V         |
| V           | F           | F         |
| F           | V           | F         |
| F           | F           | F         |

- **OR es un "O" lógico.** El resultado será verdadero si cualquiera de los dos elementos es verdadero. La tabla de verdad del operador OR es la siguiente:

condición\_1 OR condición\_2

| condición_1 | condición_2 | 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. Si es verdadera devuelve falso y si es falsa devuelve verdadero.

**!** Las condiciones se evalúan de izquierda a derecha y podemos emplear paréntesis para agrupar condiciones. Lo más aconsejable es **incluir paréntesis siempre que se use más de un operador distinto** para evitar que las condiciones no funcionen como se pretende.

Además, se recomienda poner primero (más a la izquierda) las condiciones más fáciles de evaluar, para que se facilite el filtrado de datos y la consulta gaste menos recursos y se procese más rápido.

Veamos algunos ejemplos:

- Mostrar todos los datos de los empleados del departamento de informática o de contabilidad.

```
mysql> SELECT * FROM empleados WHERE dpt = 'INF' OR dpt = 'CONT';
+-----+-----+-----+-----+-----+
| dni      | nombre_emp | especialidad | fecha_alta | dpt |
+-----+-----+-----+-----+-----+
| 12345678A | Alberto Gil | Contable     | 2010-12-10 | CONT |
| 23456789B | Mariano Sanz | Informática  | 2011-10-04 | INF  |
| 45678901D | Ana Silván  | Informática  | 2012-11-25 | INF  |
+-----+-----+-----+-----+-----+
3 rows in set (0,00 sec)
```

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

```
mysql> SELECT * FROM empleados
-> WHERE dpt = 'INF' AND fecha_alta < '2013-01-01';
```

| dni       | nombre_emp   | especialidad | fecha_alta | dpt |
|-----------|--------------|--------------|------------|-----|
| 23456789B | Mariano Sanz | Informática  | 2011-10-04 | INF |
| 45678901D | Ana Silván   | Informática  | 2012-11-25 | INF |

2 rows in set (0,00 sec)

- Mostrar todos los datos de los empleados que se han incorporado a la empresa antes del 1 de enero de 2013 al departamento de informática o al de comercio.

```
mysql> SELECT * FROM empleados
-> WHERE dpt = 'INF' OR dpt = 'COM' AND fecha_alta < '2013-01-01';
```

| dni       | nombre_emp   | especialidad | fecha_alta | dpt |
|-----------|--------------|--------------|------------|-----|
| 34567890C | Iván Gómez   | Ventas       | 2012-07-20 | COM |
| 23456789B | Mariano Sanz | Informática  | 2011-10-04 | INF |
| 45678901D | Ana Silván   | Informática  | 2012-11-25 | INF |

3 rows in set (0,00 sec)

```
mysql> SELECT * FROM empleados
-> WHERE fecha_alta < '2013-01-01' AND dpt = 'INF' OR dpt = 'COM';
```

| dni       | nombre_emp     | especialidad | fecha_alta | dpt |
|-----------|----------------|--------------|------------|-----|
| 34567890C | Iván Gómez     | Ventas       | 2012-07-20 | COM |
| 56789012E | María Cuadrado | Ventas       | 2013-04-02 | COM |
| 23456789B | Mariano Sanz   | Informática  | 2011-10-04 | INF |
| 45678901D | Ana Silván     | Informática  | 2012-11-25 | INF |

4 rows in set (0,00 sec)

```
mysql> SELECT * FROM empleados
-> WHERE fecha_alta < '2013-01-01' AND (dpt = 'INF' OR dpt = 'COM');
```

| dni       | nombre_emp   | especialidad | fecha_alta | dpt |
|-----------|--------------|--------------|------------|-----|
| 34567890C | Iván Gómez   | Ventas       | 2012-07-20 | COM |
| 23456789B | Mariano Sanz | Informática  | 2011-10-04 | INF |
| 45678901D | Ana Silván   | Informática  | 2012-11-25 | INF |

3 rows in set (0,00 sec)

Aquí se puede observar la utilidad de los paréntesis. El resultado del centro incluye un trabajador posterior a enero del 2013, puesto que al evaluarse las condiciones de derecha a izquierda primero se evalúa el AND y después el OR, por lo que los empleados del departamento comercial ya no tienen un requisito de fecha\_entrada. Se puede evitar teniendo siempre en cuenta el orden de ejecución, como en la primera opción, o usando paréntesis como en la última.

- Mostrar todos los datos de proyectos asignados al departamento de informática antes del 1 de enero de 2013 o que aún no estén asignados a ningún departamento:

```
mysql> SELECT * FROM proyectos
-> WHERE (dpto = 'INF' AND fecha_inicio < '2013-01-01')
-> OR dpto IS NULL;
```

| cod_proy | nombre                  | fecha_inicio | dpto | responsable |
|----------|-------------------------|--------------|------|-------------|
| T0451    | Consejería de Educación | 2012-05-24   | INF  | 23456789B   |
| V324     | Oceanográfico           | 2012-09-29   | NULL | NULL        |

2 rows in set (0,01 sec)


Vamos a complicar (de forma innecesaria con fines meramente educativos) la solución de esta consulta para añadir un operador NOT. En este caso evaluaremos que *no se cumple* la condición de *que el departamento no es nulo*, lo que es lo mismo que en la primera opción sin la doble negación. El paréntesis está meramente para distinguir el operador lógico NOT de la condición, tampoco sería necesario.

```
mysql> SELECT * FROM proyectos
-> WHERE (dpto = 'INF' AND fecha_inicio < '2013-01-01')
-> OR NOT (dpto IS NOT NULL);
```

| cod_proy | nombre                  | fecha_inicio | dpto | responsable |
|----------|-------------------------|--------------|------|-------------|
| T0451    | Consejería de Educación | 2012-05-24   | INF  | 23456789B   |
| V324     | Oceanográfico           | 2012-09-29   | NULL | NULL        |

2 rows in set (0,00 sec)

## 6.4. BETWEEN

 **BETWEEN** selecciona los valores entre dos cotas. Su uso habitual es "BETWEEN x AND y", lo que sería equivalente a ">=x AND <=y"

Este operador se puede usar con cualquier tipo de dato, aunque lo más común es con fechas o tipos numéricos, ya que con cadenas de texto es una comparación sobre el orden alfabético. Por ejemplo:

- Mostrar todos los datos de los pedidos con número entre el 2 y el 4, ambos incluidos:

```
mysql> SELECT * FROM pedido WHERE num_ped BETWEEN 2 AND 4;
+-----+-----+-----+
| num_ped | fecha      | cod_prov |
+-----+-----+-----+
|      2 | 2013-06-10 | MA280    |
|      3 | 2013-06-12 | BA843    |
|      4 | 2013-06-14 | T0342    |
+-----+-----+-----+
3 rows in set (0,00 sec)
```

- Mostrar todos los datos de los pedidos de los días 12 y 13 de junio de 2013:

```
mysql> SELECT * FROM pedido WHERE fecha BETWEEN '2013-06-12' AND '2013-06-13';
+-----+-----+-----+
| num_ped | fecha      | cod_prov |
+-----+-----+-----+
|      3 | 2013-06-12 | BA843    |
+-----+-----+-----+
1 row in set (0,00 sec)
```

- Mostrar todos los datos de los pedidos a los proveedores con un código que empiece por una letra desde la B hasta la L:

```
mysql> SELECT * FROM pedido WHERE cod_prov BETWEEN 'B' AND 'M';
+-----+-----+-----+
| num_ped | fecha      | cod_prov |
+-----+-----+-----+
|      3 | 2013-06-12 | BA843    |
+-----+-----+-----+
1 row in set (0,00 sec)
```

En este caso se usa 'M' y no 'L' puesto que en la comparación alfabética cualquier código que empiece por L pero contenga cualquier otro carácter después ya es "mayor" que 'L'.

## 6.5. [NOT] IN

Esta opción, que ya se ha visto en unidades anteriores como cláusula a usar en el CHECK, permite comparar con un listado de valores. Su uso es el siguiente:

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

Permite comprobar si la expresión/columna especificada está o no está entre un listado de valores. Sería equivalente a encadenar comparaciones de 'Valor\_X' con operadores OR, abreviando mucho el texto de la consulta. Más adelante, en la parte correspondiente a sub-consultas, se podrá ver que esta cláusula resulta muy útil también para seleccionar valores que pertenezcan al resultado de otra consulta.



- Mostrar los datos de los empleados de los departamentos de informática, comercial y contabilidad.

```
mysql> SELECT * FROM empleados
-> WHERE dpt IN ('INF','COM','CONT');
```

| dni       | nombre_emp     | especialidad | fecha_alta | dpt  |
|-----------|----------------|--------------|------------|------|
| 34567890C | Iván Gómez     | Ventas       | 2012-07-20 | COM  |
| 56789012E | María Cuadrado | Ventas       | 2013-04-02 | COM  |
| 12345678A | Alberto Gil    | Contable     | 2010-12-10 | CONT |
| 23456789B | Mariano Sanz   | Informática  | 2011-10-04 | INF  |
| 45678901D | Ana Silván     | Informática  | 2012-11-25 | INF  |

```
5 rows in set (0,01 sec)
```

```
mysql> SELECT * FROM empleados
-> WHERE dpt = 'INF' OR dpt = 'COM' OR dpt = 'CONT';
```

| dni       | nombre_emp     | especialidad | fecha_alta | dpt  |
|-----------|----------------|--------------|------------|------|
| 34567890C | Iván Gómez     | Ventas       | 2012-07-20 | COM  |
| 56789012E | María Cuadrado | Ventas       | 2013-04-02 | COM  |
| 12345678A | Alberto Gil    | Contable     | 2010-12-10 | CONT |
| 23456789B | Mariano Sanz   | Informática  | 2011-10-04 | INF  |
| 45678901D | Ana Silván     | Informática  | 2012-11-25 | INF  |

```
5 rows in set (0,01 sec)
```

- Mostrar los datos de los empleados que no pertenezcan a los departamentos de informática, comercial ni contabilidad.

```
mysql> SELECT * FROM empleados
-> WHERE dpt NOT IN ('INF','COM','CONT');
```

| dni       | nombre_emp    | especialidad | fecha_alta | dpt |
|-----------|---------------|--------------|------------|-----|
| 67890123A | Roberto Milán | Logística    | 2010-02-05 | ALM |


```
1 row in set (0,00 sec)
```

```
mysql> SELECT * FROM empleados
-> WHERE dpt <> 'INF' AND dpt <> 'COM' AND dpt <> 'CONT';
```

| dni       | nombre_emp    | especialidad | fecha_alta | dpt |
|-----------|---------------|--------------|------------|-----|
| 67890123A | Roberto Milán | Logística    | 2010-02-05 | ALM |

```
1 row in set (0,00 sec)
```

## 6.6. [NOT] LIKE

 **LIKE** permite filtrar campos de tipo texto usando patrones en vez de valores concretos. Se usa del siguiente modo:  
 expr\_col [NOT] LIKE 'Patrón'

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:

- **\_** Representa cualquier carácter
- **%** Representa cualquier conjunto de caracteres.

Por ejemplo, **LIKE 'C\_M'** busca todas las palabras de 3 letras que empiecen por C y acaben en M, mientras que **LIKE 'C%M'** busca todas las palabras que empiecen por C y acaben en M. Se pueden usar y combinar tantos comodines como se desee.

- Mostrar todos los empleados cuyo nombre empiece por 'A'.

```
mysql> SELECT * FROM empleados WHERE nombre_emp LIKE 'A%';
+-----+-----+-----+-----+-----+
| dni      | nombre_emp | especialidad | fecha_alta | dpt |
+-----+-----+-----+-----+-----+
| 12345678A | Alberto Gil | Contable     | 2010-12-10 | CONT |
| 45678901D | Ana Silván  | Informática  | 2012-11-25 | INF  |
+-----+-----+-----+-----+-----+
2 rows in set (0,00 sec)
```

- Mostrar todos los empleados cuyo nombre (completo, por tanto el apellido) acabe en 'n'.

```
mysql> SELECT * FROM empleados WHERE nombre_emp LIKE '%n';
+-----+-----+-----+-----+-----+
| dni      | nombre_emp | especialidad | fecha_alta | dpt |
+-----+-----+-----+-----+-----+
| 45678901D | Ana Silván  | Informática  | 2012-11-25 | INF  |
| 67890123A | Roberto Milán | Logística    | 2010-02-05 | ALM  |
+-----+-----+-----+-----+-----+
2 rows in set (0,00 sec)
```

- Mostrar todos los empleados cuyo nombre (completo) contenga la cadena 'il'.

```
mysql> SELECT * FROM empleados WHERE nombre_emp LIKE '%il%';
+-----+-----+-----+-----+-----+
| dni      | nombre_emp | especialidad | fecha_alta | dpt |
+-----+-----+-----+-----+-----+
| 12345678A | Alberto Gil | Contable     | 2010-12-10 | CONT |
| 45678901D | Ana Silván  | Informática  | 2012-11-25 | INF  |
| 67890123A | Roberto Milán | Logística    | 2010-02-05 | ALM  |
+-----+-----+-----+-----+-----+
3 rows in set (0,00 sec)
```

- Mostrar todos los departamentos cuyo código tenga 4 letras

```
mysql> SELECT * FROM departamentos WHERE cod_dpt LIKE '____';
+-----+-----+-----+
| cod_dpt | nombre_dpt | ubicacion |
+-----+-----+-----+
| CONT    | Contabilidad | Planta quinta U1 |
+-----+-----+-----+
1 row in set (0,00 sec)
```

- Mostrar todos los empleados cuyo nombre (sólo el nombre esta vez) tenga exactamente 7 letras y acabe en 'to'.

```
mysql> SELECT * FROM empleados WHERE nombre_emp LIKE '____to%';
+-----+-----+-----+-----+-----+
| dni      | nombre_emp | especialidad | fecha_alta | dpt |
+-----+-----+-----+-----+-----+
| 12345678A | Alberto Gil | Contable     | 2010-12-10 | CONT |
| 67890123A | Roberto Milán | Logística    | 2010-02-05 | ALM  |
+-----+-----+-----+-----+-----+
2 rows in set (0,00 sec)
```

## 7. Ordenación de filas en la respuesta

SQL nos proporciona la posibilidad de ordenar el conjunto de filas que obtenemos como resultado de una consulta.

📖 Para ordenar la respuesta se usa la cláusula **ORDER BY** al final de nuestra instrucción SQL.

**ORDER BY {expr\_col | posición} [DESC] , ...**

Para indicar el tipo de ordenación que deseamos realizar podemos indicar el nombre de la columna a utilizar, una expresión (que puede contener funciones) o bien directamente un número que representa la posición de la columna en la proyección. Por otro lado, ten presente que se puede incluso ordenar la respuesta por atributos de la tabla que no se muestren en la proyección. Se pueden aplicar sucesivamente tantos criterios de ordenación como se desee, separados por comas, que se aplican de izquierda a derecha.

La ordenación se realiza según el tipo de campo y por defecto es de forma ascendente (de la A a la Z, de menor a mayor, etc). Esto se puede indicar explícitamente con ASC en vez de DESC, pero no es necesario. Si se desea ordenar de forma descendente se añade la palabra DESC al final del criterio; cada criterio puede ser ascendente o descendente de forma independiente.

La opción de usar el número de la posición en la proyección resulta especialmente útil para abreviar la cláusula cuando se quiere ordenar por un campo de la respuesta formado por funciones, expresiones complejas o sub-consultas; sin



embargo, conviene tener muy presente que si se modifica la proyección se debe tener en cuenta porque puede afectar al criterio.

## ⚠ ! Nunca uses alias de columna en el ORDER BY.

Aunque algunos SGBD lo admiten (como MySQL), la mayoría no aplican los alias hasta después y devolverán un error. Veamos ahora algunos ejemplos de ordenación:

- Mostrar todos los datos de los empleados ordenados por nombre

```
mysql> SELECT * FROM empleados ORDER BY nombre_emp;
+-----+-----+-----+-----+-----+
| dni      | nombre_emp | especialidad | fecha_alta | dpt |
+-----+-----+-----+-----+-----+
| 12345678A | Alberto Gil | Contable     | 2010-12-10 | CONT |
| 45678901D | Ana Silván  | Informática  | 2012-11-25 | INF  |
| 34567890C | Iván Gómez  | Ventas       | 2012-07-20 | COM  |
| 56789012E | María Cuadrado | Ventas       | 2013-04-02 | COM  |
| 23456789B | Mariano Sanz | Informática  | 2011-10-04 | INF  |
| 67890123A | Roberto Milán | Logística    | 2010-02-05 | ALM  |
+-----+-----+-----+-----+-----+
6 rows in set (0,00 sec)
```

- Mostrar el nombre y fecha de alta de los empleados ordenados por fecha de contratación, del más reciente al más antiguo.

```
mysql> SELECT nombre_emp, fecha_alta FROM empleados ORDER BY fecha_alta DESC;
+-----+-----+
| nombre_emp | fecha_alta |
+-----+-----+
| María Cuadrado | 2013-04-02 |
| Ana Silván    | 2012-11-25 |
| Iván Gómez    | 2012-07-20 |
| Mariano Sanz  | 2011-10-04 |
| Alberto Gil   | 2010-12-10 |
| Roberto Milán | 2010-02-05 |
+-----+-----+
6 rows in set (0,01 sec)
```

- Repetir la consulta anterior mostrando solo los nombres.

```
mysql> SELECT nombre_emp FROM empleados ORDER BY fecha_alta DESC;
+-----+
| nombre_emp |
+-----+
| María Cuadrado |
| Ana Silván    |
| Iván Gómez    |
| Mariano Sanz  |
| Alberto Gil   |
| Roberto Milán |
+-----+
6 rows in set (0,00 sec)
```

- Mostrar todos los datos de los empleados ordenados por departamento y dentro del departamento, de más reciente a más antiguo.

```
mysql> SELECT * FROM empleados ORDER BY dpt,fecha_alta DESC;
+-----+-----+-----+-----+-----+
| dni      | nombre_emp | especialidad | fecha_alta | dpt |
+-----+-----+-----+-----+-----+
| 67890123A | Roberto Milán | Logística | 2010-02-05 | ALM |
| 56789012E | María Cuadrado | Ventas | 2013-04-02 | COM |
| 34567890C | Iván Gómez | Ventas | 2012-07-20 | COM |
| 12345678A | Alberto Gil | Contable | 2010-12-10 | CONT |
| 45678901D | Ana Silván | Informática | 2012-11-25 | INF |
| 23456789B | Mariano Sanz | Informática | 2011-10-04 | INF |
+-----+-----+-----+-----+-----+
6 rows in set (0,00 sec)

mysql> SELECT * FROM empleados ORDER BY dpt,4 DESC;
+-----+-----+-----+-----+-----+
| dni      | nombre_emp | especialidad | fecha_alta | dpt |
+-----+-----+-----+-----+-----+
| 67890123A | Roberto Milán | Logística | 2010-02-05 | ALM |
| 56789012E | María Cuadrado | Ventas | 2013-04-02 | COM |
| 34567890C | Iván Gómez | Ventas | 2012-07-20 | COM |
| 12345678A | Alberto Gil | Contable | 2010-12-10 | CONT |
| 45678901D | Ana Silván | Informática | 2012-11-25 | INF |
| 23456789B | Mariano Sanz | Informática | 2011-10-04 | INF |
+-----+-----+-----+-----+-----+
6 rows in set (0,00 sec)

mysql> SELECT dpt, fecha_alta, dni, nombre_emp, especialidad
-> FROM empleados ORDER BY 1, 2 DESC;
+-----+-----+-----+-----+-----+
| dpt | fecha_alta | dni      | nombre_emp | especialidad |
+-----+-----+-----+-----+-----+
| ALM | 2010-02-05 | 67890123A | Roberto Milán | Logística |
| COM | 2013-04-02 | 56789012E | María Cuadrado | Ventas |
| COM | 2012-07-20 | 34567890C | Iván Gómez | Ventas |
| CONT | 2010-12-10 | 12345678A | Alberto Gil | Contable |
| INF | 2012-11-25 | 45678901D | Ana Silván | Informática |
| INF | 2011-10-04 | 23456789B | Mariano Sanz | Informática |
+-----+-----+-----+-----+-----+
6 rows in set (0,01 sec)
```

Se muestran 3 opciones; en la primera se usan los nombres de las columnas para la ordenación, en la segunda se usa la posición de la fecha\_alta (notar que un cambio en la estructura de la tabla podría hacer que la fecha no fuese 4 y "romper" esta consulta), en la última se usa la posición para los 2 campos de ordenación, pero se especifican los campos de la proyección sin usar comodines para evitar problemas si cambia la estructura de la tabla (se han puesto los campos de la ordenación en las 2 primeras posiciones para ver claramente la ordenación, no hay ninguna obligación de hacerlo así).

## 8. Índices, ordenación y eficiencia.

Como se comentó en el tema anterior **los índices pueden ayudar a mejorar la eficiencia de las consultas**. El inconveniente es que es **a costa de ralentizar las operaciones DML** (INSERT, UPDATE, DELETE), puesto que cada vez que se modifica la tabla se deben actualizar los índices. Por ello, se deben usar con cuidado y solo cuando se considere que la mejora en la eficiencia de las consultas compensa el coste de mantener los índices.

En las bases de datos de ejemplo que manejamos en el curso la cantidad de datos es irrisoria para que se note la diferencia, pero en una base de datos real con millones de registros la diferencia puede ser abismal. Es fácil comprender como **actualizar índices de millones de registros puede ser un proceso muy costoso**, en cambio, comprender como ayudan a aumentar la eficiencia de las consultas requiere una explicación más detallada.


Veamos un ejemplo con la tabla de empleados de *db\_empresa*, aprovechando que las FOREIGN KEY ya crean índices automáticamente.

```
mysql> SHOW INDEX FROM empleados;
```

| Table     | Non_unique | Key_name   | Seq_in_index | Column_name | Collation | Cardinality |
|-----------|------------|------------|--------------|-------------|-----------|-------------|
| empleados | 0          | PRIMARY    | 1            | dni         | A         | 6           |
| empleados | 1          | emp_dpt_fk | 1            | dpt         | A         | 4           |

```
2 rows in set (0,01 sec)
```

Nos fijamos en la cardinalidad de cada índice. Esta es la cantidad de valores distintos que hay en la columna. En la PRIMARY KEY obviamente es igual al número total de filas de la tabla, 6 en este caso, mientras que en la FK tenemos una cardinalidad de 4, puesto que solo hay 4 departamentos distintos. Esto significa que el índice ya tiene localizadas todas las filas que pertenecen a cada departamento, por lo que si se filtra una consulta por departamento la búsqueda será mucho más rápida que si no hubiese índice. Es como si se filtrase entre 4 filas en vez de tener que revisar las 6. Extrapolando esta situación a una empresa con 1000 empleados y 10 departamentos, se puede entender que el filtrado por departamentos es 100 veces más rápido con el índice que sin él.

 **La cardinalidad de un índice es el número de valores distintos que hay en el mismo.** Cuanto más baja sea la cardinalidad respecto al total de registros, más eficiente será el índice.

Para el caso de la ordenación de los resultados de una consulta sucede lo mismo. Si se ordena por un campo que tiene un índice, la ordenación será mucho más

rápida que si no lo tiene, puesto que el índice ya está ordenado, por tanto la consulta no necesita ordenar los datos, solo mostrarlos en el orden del índice.

Una vez comprendido el funcionamiento de los índices, recomendamos volver a la unidad anterior y revisar los consejos en la creación de índices, puesto que ahora se entiende mejor el porqué de los mismos.

Recuerda que los índices se actualizan automáticamente cada vez que se realiza una operación DML, pero a día de hoy MySQL inexplicablemente no actualiza la información mostrada en el comando `SHOW INDEX` (los índices sí) en una sesión, lo que puede llevar a errores respecto a la interpretación de la eficiencia de un índice, al no tener la cardinalidad actualizada. Para forzar a que actualice esta información se debe ejecutar el comando `ANALYZE TABLE nombre_tabla;`. Vamos a insertar un empleado del departamento de administración para comprobarlo.

```
mysql> INSERT INTO empleados (dni,nombre_emp,dpt) VALUES ('000000000A','Borrar','ADM');
Query OK, 1 row affected (0,01 sec)
```

```
mysql> SHOW INDEX FROM empleados;
```

| Table     | Non_unique | Key_name   | Seq_in_index | Column_name | Collation | Cardinality |
|-----------|------------|------------|--------------|-------------|-----------|-------------|
| empleados | 0          | PRIMARY    | 1            | dni         | A         | 6           |
| empleados | 1          | emp_dpt_fk | 1            | dpt         | A         | 4           |

```
2 rows in set (0,00 sec)
```

```
mysql> ANALYZE TABLE empleados;
```

| Table                | Op      | Msg_type | Msg_text |
|----------------------|---------|----------|----------|
| db_empresa.empleados | analyze | status   | OK       |

```
1 row in set (0,02 sec)
```

```
mysql> SHOW INDEX FROM empleados;
```

| Table     | Non_unique | Key_name   | Seq_in_index | Column_name | Collation | Cardinality |
|-----------|------------|------------|--------------|-------------|-----------|-------------|
| empleados | 0          | PRIMARY    | 1            | dni         | A         | 7           |
| empleados | 1          | emp_dpt_fk | 1            | dpt         | A         | 5           |

```
2 rows in set (0,00 sec)
```

```
mysql> DELETE FROM empleados WHERE dni = '000000000A';
Query OK, 1 row affected (0,01 sec)
```

No olvides borrar el empleado insertado para que no se quede en la tabla.

Finalmente, ten presente que en esta explicación se han usado índices simples para facilitar la comprensión, pero los índices pueden estar compuestos por varios campos e incluso expresiones.

## 9. Funciones agregadas

Las funciones agregadas, también llamadas de agregación o de resumen, solo se pueden usar en la proyección. Estas reciben como parámetro una `expr_col` y realizan cálculos con esos valores trabajando sobre grupos de filas para devolver un único resultado para el grupo, en vez de uno por cada fila.

Con ellas se pueden calcular estadísticas o resumir información a partir de las tablas. Las más usadas son:

| Función | Cálculo   |
|---------|---|
| SUM()   | Suma los valores de la <code>expr_col</code>                |
| AVG()   | Media aritmética de los valores de la <code>expr_col</code> |
| COUNT() | Cuenta el numero de filas                                   |
| MAX()   | Calcula el valor máximo de la <code>expr_col</code>         |
| MIN()   | Calcula el valor mínimo de la <code>expr_col</code>         |

⚠ La respuesta de las funciones agregadas devuelve un cálculo **agrupando las filas de la respuesta**. Aunque ese valor sea igual al de una fila de la tabla, como en el caso de MAX y MIN, **una función agregada no selecciona filas**, por lo que no se pueden mostrar directamente otras columnas, ya que la respuesta no se corresponde a ninguna fila.

Lo que sí que se permite hacer es usar varias columnas con funciones agregadas en una misma consulta, ya que todas están haciendo cálculos sobre los grupos de filas.

**COUNT()** permite contar las filas del resultado de la consulta, por lo que generalmente recibe una columna o el comodín \*. Recibir una expresión con cálculos resulta innecesario para realizar una cuenta de respuestas.

- **COUNT(\*)** devuelve directamente el número total de filas de una tabla (o respuesta).
- **COUNT(nombre\_columna)** devuelve el número de filas en las que ese campo *no es nulo*.
- **COUNT(DISTINCT nombre\_columna)** permite contar cuantos valores distintos hay en una columna, ignorando los nulos.

💡 Es altamente recomendable usar un alias cuando se añaden funciones agregadas, ya que clarifica lo que estamos obteniendo. Además en caso contrario el nombre de la columna en la respuesta será literalmente la expresión usada, lo que puede quedar muy largo. Por ejemplo, COUNT(DISTINCT dpt) se entiende mucho mejor y queda mas abreviado como COUNT(DISTINCT dpt) AS 'total\_dep'.

Recuerda que a pesar de esto, si se ordena por esta columna la consulta, no se debe usar el alias. En su lugar es mejor usar la posición si se quiere evitar escribir de nuevo la expresión completa.

Veamos algunos ejemplos de como usar estas funciones de resumen:

- Mostrar cuánto cuesta en total comprar una unidad de cada producto disponible:

```
mysql> SELECT SUM(precio) FROM producto;
+-----+
| SUM(precio) |
+-----+
|      132.25 |
+-----+
1 row in set (0,00 sec)

mysql> SELECT SUM(precio) AS Total FROM producto;
+-----+
| Total |
+-----+
| 132.25 |
+-----+
1 row in set (0,00 sec)
```

- Mostrar el precio medio de los productos:

```
mysql> SELECT AVG(precio) AS "Precio Medio" FROM producto;
+-----+
| Precio Medio |
+-----+
|    18.892857 |
+-----+
1 row in set (0,00 sec)
```

- Mostrar el precio más caro y el más barato de la lista de productos:

```
mysql> SELECT MAX(precio) AS maximo, MIN(precio) AS minimo FROM producto;
+-----+-----+
| maximo | minimo |
+-----+-----+
|    31.75 |     3.00 |
+-----+-----+
1 row in set (0,00 sec)
```



- Mostrar cuál es el producto más caro de la lista de productos:

No podemos hacerlo con nuestras herramientas actuales, se verá una alternativa propia de MySQL en el próximo apartado, pero no es estándar y tiene sus limitaciones. Más adelante veremos cómo solventarlas. Se muestran los típicos errores cometidos al intentar esta consulta.

```
mysql> SELECT nombre_prod, MAX(precio) FROM producto;
ERROR 1140 (42000): In aggregated query without GROUP BY, expression #1 of SELECT list contains nonaggregated column 'db_tienda_friki.producto.nombre_prod'; this is incompatible with sql_mode=only_full_group_by
mysql> SELECT nombre_prod FROM producto WHERE precio=MAX(precio);
ERROR 1111 (HY000): Invalid use of group function
```

No se pueden usar otras columnas junto a una de agregado, pues las filas se han agrupado en una. Tampoco se pueden usar funciones de agregación directamente en el WHERE, solo en la proyección.

- Mostrar el número de proyectos en la empresa:

```
mysql> SELECT COUNT(*) AS num_proyectos FROM proyectos;
+-----+
| num_proyectos |
+-----+
|             3 |
+-----+
1 row in set (0,00 sec)
```

- Mostrar el número de proyectos asignados a algún departamento:

```
mysql> SELECT COUNT(dpto) AS num_proyectos FROM proyectos;
+-----+
| num_proyectos |
+-----+
|             2 |
+-----+
1 row in set (0,00 sec)
```

Al usar el campo dpto, en vez del comodín \*, no cuenta los NULL


- Mostrar cuántas especialidades distintas tienen entre todos los empleados de la empresa.

```
mysql> SELECT COUNT(DISTINCT especialidad) AS total_especialidades
-> FROM empleados;
+-----+
| total_especialidades |
+-----+
|             4 |
+-----+
1 row in set (0,01 sec)
```

## 10. Limitar las filas mostradas en la respuesta

En consultas que devuelven muchas filas, puede ser interesante limitar el número de filas que se muestran en la respuesta. Aunque de nuevo esto es completamente innecesario en los ejemplos usados en esta unidad.

Por desgracia no existe un estándar para hacer esto, por lo que cada SGBD tiene su propia sintaxis. En MySQL se usa LIMIT, que se coloca al final de la consulta y recibe 2 parámetros: el número de filas a mostrar y el número de fila desde la que empezar a mostrar. Si solo se especifica un número, se entiende que es el número de filas a mostrar y se empieza desde la primera.

 **LIMIT**, usado al final de la consulta, permite limitar el número de filas que se muestran en la respuesta de una consulta en MySQL. Esto no es SQL estándar.

Esto resulta especialmente útil para buscar valores máximos o mínimos cuando el campo está indexado, puesto que se puede ordenar la respuesta de forma instantánea y limitar a 1 el número de filas a mostrar. Por ejemplo, para mostrar el precio del producto más caro de la lista de productos:

```
mysql> SELECT precio FROM producto ORDER BY precio DESC LIMIT 1;
+-----+
| precio |
+-----+
|  31.75 |
+-----+
1 row in set (0,00 sec)
```

Se puede pensar que esta estructura incluso permite mostrar los otros datos de la fila añadiéndolos a la proyección, sin los límites de las funciones de agregado. De hecho, en internet lo encontraréis en multitud de ocasiones, **pero eso es una mala praxis que se debe evitar por completo.**

Esto es debido a que la consulta no devuelve una información verídica. Supongamos que hay más de un producto con el precio más caro; LIMIT 1 en ese caso mostrará uno de ellos, pero no se sabe cual, y puesto que desconocemos el número de filas "empatadas" en el primer puesto, no podemos usar LIMIT X para mostrar todas ellas sin una consulta previa, lo que, además de resultar muy poco eficiente, requiere cambiar el número en el LIMIT cada vez, por lo que no se puede guardar esa consulta para un uso automático de la misma.

 **Nunca se debe usar LIMIT para mostrar los datos de la fila máxima/mínima, solo el valor de esa columna.**



Existen algunos SGBD que tienen funcionalidades propias para mostrar todos los valores "empatados", pero no es el caso de MySQL. En la parte de subconsultas se estudiará cómo solventar realmente esta problemática.

## 11. Bibliografía

---

- MySQL 8.0 Reference Manual.  
<https://dev.mysql.com/doc/refman/8.0/en/>
- Oracle Database Documentation  
<https://docs.oracle.com/en/database/oracle/oracle-database/index.html>
- W3Schools. MySQL Tutorial.  
<https://www.w3schools.com/mysql/>
- GURU99. Tutorial de MySQL para principiantes Aprende en 7 días.  
<https://guru99.es/sql/>
- SQL Tutorial - Learn SQL.  
<https://www.sqltutorial.net/>