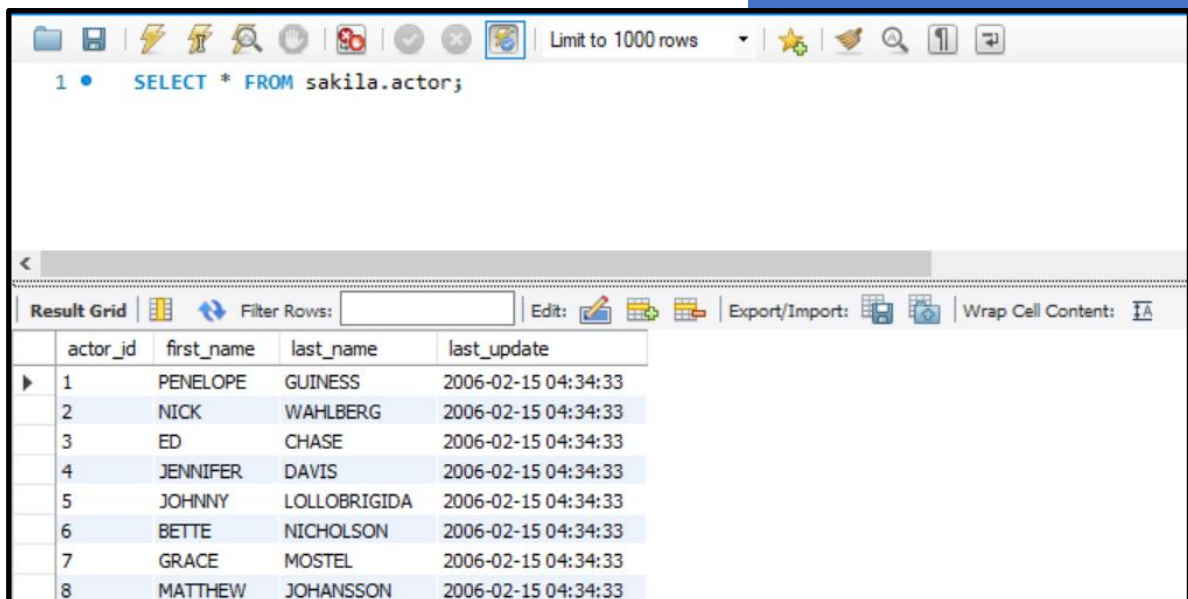


Tema 4

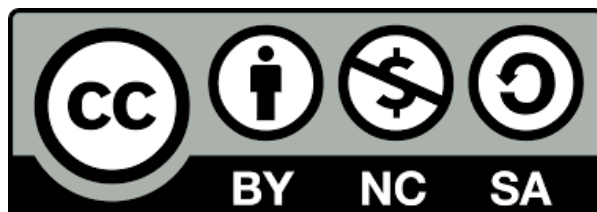
Consultas en bases de datos relacionales



The screenshot shows a database query tool interface. At the top, there is a toolbar with various icons for file operations, execution, and navigation. Below the toolbar, a SQL query is entered in a text area: `1 • SELECT * FROM sakila.actor;`. To the right of the query, there is a dropdown menu set to "Limit to 1000 rows". Below the query area, there is a "Result Grid" section. It includes a "Filter Rows:" input field, an "Edit:" button, and "Export/Import:" and "Wrap Cell Content:" options. The main part of the grid displays the results of the query as a table with four columns: `actor_id`, `first_name`, `last_name`, and `last_update`. The table contains eight rows of data, each representing an actor from the Sakila database.

	actor_id	first_name	last_name	last_update
▶	1	PENELOPE	GUINESS	2006-02-15 04:34:33
	2	NICK	WAHLBERG	2006-02-15 04:34:33
	3	ED	CHASE	2006-02-15 04:34:33
	4	JENNIFER	DAVIS	2006-02-15 04:34:33
	5	JOHNNY	LOLLOBRIGIDA	2006-02-15 04:34:33
	6	BETTE	NICHOLSON	2006-02-15 04:34:33
	7	GRACE	MOSTEL	2006-02-15 04:34:33
	8	MATTHEW	JOHANSSON	2006-02-15 04:34:33

Este documento ha sido elaborado para el alumnado del módulo “Bases de datos” de los CFGS DAM y DAW del IES Playamar.



Más información en <https://creativecommons.org/licenses/by-nc-sa/3.0/es/>

Contenido

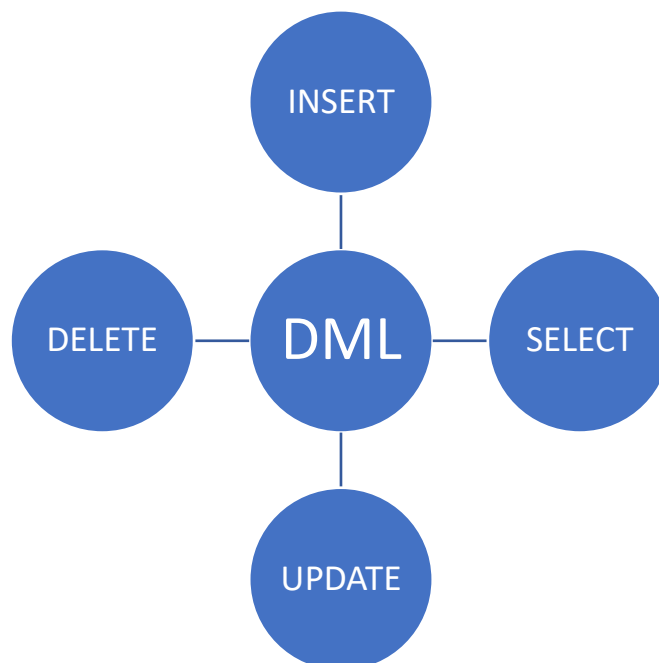
1. Introducción	1
2. INSERT	2
2.1. Inserción masiva de datos.....	4
3. SELECT	4
3.1. Límite de registros.....	6
3.2. Valores distintos	6
3.3. Filtros	6
3.4. Valores nulos (IS).....	8
3.5. Pertenencia a conjunto (IN)	8
3.6. Rango de valores (BETWEEN)	9
3.7. Patrón de valores (LIKE)	10
3.8. Ordenación.....	10
3.9. Consultas de resumen.....	11
3.10. Agrupaciones.....	12
4. Subconsultas.....	14
4.1. Comparación con subconsultas	14
4.2. Pertenencia a conjunto con subconsultas	14
4.3. Existencia de datos con subconsultas	14
4.4. Cuantificadores ALL y ANY	15
4.5. Subconsultas anidadas.....	15
5. Consultas multitable	16
5.1. Producto cartesiano	16
5.2. Combinación de tablas (versión 1).....	17
5.3. Combinación de tablas (versión 2).....	18
6. Combinación de consultas	21

6.1.	Unión de consultas (UNION)	22
6.2.	Intersección de consultas (INTERSECT)	23
6.3.	Diferencia de consultas (EXCEPT)	23
7.	Resumen	24

1. Introducción

Una vez creada una base de datos, bien con instrucciones DDL o con las herramientas del administrador del SGBD, el siguiente paso es guardar datos en ella para poder manipular la información almacenada.

Todas las operaciones de manipulación de datos se realizan con el sublenguaje DML de SQL. Es, sin ninguna duda, el conjunto de instrucciones de SQL que más se utilizan. Con DML es posible realizar la inserción (INSERT), consulta (SELECT), modificación (UPDATE) y borrado (DELETE) de los datos almacenados en una base de datos.



IMPORTANTE

En DDL y DML hay instrucciones para crear, modificar y eliminar, pero hacen referencia a conceptos distintos.

- Las estructuras se crean con CREATE, los datos con INSERT.
- Las estructuras se modifican con ALTER, los datos con UPDATE.
- Las estructuras se eliminan con DROP, los datos con DELETE.

En este tema se trata la inserción y la consulta, dejando la modificación y el borrado para el siguiente tema.

2. INSERT

La instrucción INSERT permite insertar uno o varios registros en una tabla de la base de datos. La sintaxis es muy sencilla (los corchetes indican expresiones opcionales):

```
INSERT [INTO] tabla [(columna1, columna2, ...)]  
VALUES (valor1, valor2, ...)  
[, (valor1, valor2, ...) ...]
```

tabla es el nombre de la tabla donde se quieren insertar los datos. Después del nombre de la tabla, de forma optativa, se pueden indicar las columnas donde se va a insertar la información. Si se especifican las columnas, la lista de valores (VALUES) a insertar se asociará correlativamente con los valores a las columnas indicadas. Si no se especifican las columnas, la lista de valores se escribirá conforme al orden de las columnas en la definición de la tabla. En cualquier caso, se deben indicar tantos valores como columnas se indiquen (si no se indican columnas, tantos valores como columnas tiene la tabla).

Las expresiones aceptadas como valor son:

- Si la columna es numérica entera, un literal numérico entero (por ejemplo: 25)
- Si la columna admite decimales, un literal decimal (por ejemplo: 3.14. Se utiliza el punto como separador de decimales).
- Si la columna es una cadena de texto, entre comillas simples (por ejemplo: 'Cliente').
- Si la columna es una fecha, entre comillas en formato AAAA-MM-DD (por ejemplo: '2023-09-15').
- En general, cualquier valor literal se puede indicar entre comillas simples en una inserción. Por tanto '25' y '3.14' son también valores válidos para campos numéricos enteros y decimales respectivamente.
- Si la columna define un valor por defecto, se puede indicar DEFAULT. Si se indica DEFAULT y no hay valor por defecto, la columna toma el valor NULL.
- Si la columna es opcional (es decir, admite valores nulos), se puede indicar NULL. Si no se le da valor a una columna, INSERT le asigna su valor por defecto.

CASO PRÁCTICO 1

En una tabla CLIENTES (id, nombre, apellidos, fecha_nacimiento, direccion), donde:

- id es auto incrementado.
- nombre, apellidos, fecha_nacimiento son obligatorios.
- direccion es opcional.

Estas son posibles instrucciones INSERT:

```
INSERT INTO CLIENTES VALUES (DEFAULT, 'Cliente1', 'Apellido1', '2000-08-31', NULL)
```

```
INSERT INTO CLIENTES (nombre, apellidos, fecha_nacimiento)  
VALUES ('Cliente2', 'Apellido2', '2000-06-11')
```

```
INSERT INTO CLIENTES (nombre, apellidos, fecha_nacimiento)  
VALUES ('Cliente4', 'Apellido4', '2000-01-01'),  
( 'Cliente5', 'Apellido5', '2000-02-01'),  
( 'Cliente6', 'Apellido6', '2000-03-01')
```

IMPORTANTE

Para insertar un registro no hace falta dar valor a todas las columnas de la tabla. Basta con indicar aquellos campos requeridos que no tengan un valor por defecto.

La instrucción INSERT también se puede escribir de otra forma (en este caso, sólo se puede insertar un registro por instrucción):

```
INSERT [INTO] tabla  
SET campo1 = valor1, campo2 = valor2...
```

CASO PRÁCTICO 2

Sobre la misma tabla anterior:

```
INSERT INTO CLIENTES SET id=DEFAULT, nombre='Cliente7', apellidos='Apellido7',  
fecha_nacimiento='2000-10-31'
```

2.1. Inserción masiva de datos

Los SGBD permiten realizar inserciones masivas de datos a partir de un fichero de datos. MySQL soporta diversos formatos: CSV, CSV separado por punto y coma, JSON, XML...

La opción más segura es utilizar CSV separado por punto y coma ya que este formato se puede manejar de forma sencilla en una hoja de cálculo. De este modo la importación masiva de datos en una tabla seguirá estos pasos:

1. Con una herramienta de hoja de cálculo, se crea un fichero CSV.
2. Se define una columna de la hoja de cálculo para cada columna de la tabla.
3. Se escribe en la hoja de cálculo los datos que se desean importar.
4. Se importa el fichero CSV en la base de datos.

3. SELECT

La instrucción SELECT permite consultar información de una o varias tablas. Es, con diferencia, la instrucción más potente de SQL y por tanto la más compleja. La sintaxis general es la siguiente:

```
SELECT [DISTINCT] columnas ...  
FROM tablas  
[WHERE filtros]  
[GROUP BY columnas ...]  
[HAVING criterios ...]  
[ORDER BY columnas ...]  
[LIMIT n]
```

Las únicas cláusulas obligatorias son SELECT y FROM, el resto son opcionales.

- SELECT indica las columnas que devuelve la consulta.
- FROM indica la tabla o tablas donde se encuentra la información que se desea consultar.
- WHERE indica los filtros que se aplican a la consulta. La consulta devuelve los datos que cumplen la condición indicada en esta cláusula.
- GROUP BY indica los campos que se utilizan para agrupar la información.

- HAVING indica los criterios para filtrar grupos. La consulta devuelve los grupos que cumplen las condiciones impuestas en esta cláusula.
- ORDER BY indica los criterios de ordenación del resultado. Por defecto, el resultado se muestra sin orden concreto (normalmente se aplica el orden de inserción de los datos en la base de datos).
- LIMIT indica el número máximo de registros que devuelve la consulta.

IMPORTANTE

Las cláusulas que aparezcan en la consulta deben escribirse en el orden indicado.

CASO PRÁCTICO 3

Para consultar la información de la tabla CLIENTES, la consulta más sencilla que se puede crear es:

```
SELECT * FROM clientes
```

La expresión * es un comodín que significa “todas las columnas de las tablas indicadas en la cláusula FROM”.

IMPORTANTE

Utilizar el comodín * como primera aproximación a una tabla para conocer su contenido es una opción muy sencilla, pero no se recomienda su uso en las consultas que necesites introducir en tus programas, ya que:

- La consulta devuelve todas las columnas, incluidas las que no necesitas.
- Si hay cambios en la base de datos (nuevas columnas en las tablas o columnas que desaparecen) se pueden producir efectos indeseados en el programa.

Para evitar problemas, es mejor indicar expresamente las columnas:

```
SELECT nombre, apellidos FROM clientes
```

Las columnas del resultado se identifican con los nombres de las columnas en la tabla, aunque se le puede indicar otro nombre (o alias) para identificarlo:

```
SELECT base, altura, base*altura AS area FROM rectángulo
```

El resultado se muestra en la forma: base, altura, area.

3.1. Límite de registros

Por defecto la consulta devuelve toda la información que cumpla las condiciones indicadas, pero es posible limitar el número máximo de registros que debe devolver con la cláusula LIMIT.

CASO PRÁCTICO 3

Para obtener los cien primeros clientes:

```
SELECT * FROM clientes LIMIT 100
```

La consulta devuelve los registros de la tabla CLIENTES y la limita a cien en caso de que haya más de cien registros.

3.2. Valores distintos

En una tabla puede haber valores repetidos. Por ejemplo, todos los clientes de una misma ciudad comparten la ciudad. Por defecto una consulta devuelve todos los valores, aunque haya repetidos. Si se desea que solo se devuelvan los valores únicos, se indica la cláusula DISTINCT.

CASO PRÁCTICO 4

Para obtener las distintas ciudades de los clientes:

```
SELECT DISTINCT ciudad FROM clientes
```

Esta consulta también se puede resolver con agrupamientos (ver más adelante).

3.3. Filtros

Los filtros son condiciones que se interpretan para seleccionar los registros que se devuelven. La consulta devuelve toda la información que satisfaga las condiciones indicadas en el filtro. Los filtros se construyen mediante expresiones. Una expresión, es una combinación de operadores, operandos y funciones que producen un resultado.

- **Operandos:** pueden ser constantes (por ejemplo: el número entero 3, el número real 3.14, el texto 'Informática' o la fecha '2023-09-15') o una columna de una tabla (por ejemplo: la columna nombre de la tabla CLIENTES). Por regla general, todos los

valores constantes van entre comillas simples a excepción de los valores numéricos.

Recuerda que el separador de decimales es el punto.

- **Operadores:** permiten combinar operandos para obtener resultados. Los operadores se resumen en la siguiente tabla:

TIPOS	OPERADORES
Aritméticos	+, -, *, /, %
Relacionales	>, >=, <, <=, =, <>
Lógicos	AND, OR, NOT
Otros	IS, IN, BETWEEN, LIKE

Los operadores tienen prioridades. Por ejemplo, la expresión $3*4 + 2$ evalúa primero la multiplicación y luego la suma para dar el resultado 14. Para combinar las expresiones y garantizar el orden en que se desean evaluar hay que utilizar paréntesis. Así, $3*(4 + 2)$ debe dar 18.

- **Funciones:** cada SGBD incorpora un conjunto de funciones para facilitar el tratamiento de datos. En especial para tratar cadenas de texto y fechas.

Funciones de tratamiento de textos en MySQL:

<https://dev.mysql.com/doc/refman/8.0/en/string-functions.html>

Funciones de tratamiento de fechas en MySQL:

<https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html>

CASO PRÁCTICO 5

Para obtener los clientes de Málaga:

```
SELECT nombre, apellidos FROM clientes WHERE ciudad = 'Málaga'
```

Para obtener los empleados con salario mayor a 20.000€:

```
SELECT nombre, apellidos FROM empleados WHERE salario > 20000
```

En la tabla PRODUCTOS se guarda el precio (SIN IVA) de cada producto. Imagina que a todos hay que aplicarle el 21% de IVA. Para obtener los productos a los que hay que añadirle 5€ o más de IVA:

```
SELECT nombre FROM productos WHERE 0.21*precio >= 5
```

Para obtener los clientes cuyo nombre o apellido comienza por MA:

```
SELECT nombre, apellidos FROM clientes  
WHERE (LEFT(nombre, 2) = 'MA') OR (LEFT(apellidos, 2) = 'MA')
```

Para obtener los clientes mayores de 25 años:

```
SELECT nombre, apellidos FROM clientes  
WHERE TIMESTAMPDIFF(YEAR, fecha_nacimiento, CURDATE()) > 25
```

3.4. Valores nulos (IS)

Un valor nulo representa una columna sin información. Cuando se necesita considerar valores nulos en los filtros de la consulta se necesita utilizar el operador IS.

CASO PRÁCTICO 6

Para obtener los clientes con la dirección vacía:

```
SELECT nombre, apellidos FROM clientes WHERE direccion IS NULL
```

Para obtener los clientes con la dirección informada:

```
SELECT nombre, apellidos FROM clientes WHERE direccion IS NOT NULL
```

IMPORTANTE

Es un error muy común tomar NULL como un valor y hacer filtros del siguiente tipo:

```
WHERE direccion = NULL
```

El valor NULL solo se puede manejar con el operador IS NULL o IS NOT NULL.

3.5. Pertenencia a conjunto (IN)

El operador IN permite comprobar si una columna tiene un valor igual que cualquier de los que están incluidos dentro de un conjunto de valores (una secuencia de valores separados por comas e indicado entre paréntesis). También se puede comprobar la no pertenencia a un conjunto con NOT IN.

CASO PRÁCTICO 7

Para obtener los clientes de Málaga, Cádiz y Granada:

```
SELECT nombre, apellidos FROM clientes
```

```
WHERE ciudad = 'Málaga' OR ciudad = 'Cádiz' OR ciudad = 'Granada'
```

Con el operador IN queda así:

```
SELECT nombre, apellidos FROM clientes
```

```
WHERE ciudad IN ('Málaga', 'Cádiz', 'Granada')
```

Los clientes que no son de Málaga ni Cádiz:

```
SELECT nombre, apellidos FROM clientes WHERE ciudad NOT IN ('Málaga', 'Cádiz')
```

3.6. Rango de valores (BETWEEN)

El operador de rango BETWEEN permite seleccionar los registros que estén incluidos en un rango, incluidos los extremos. También si están fuera de rango con NOT BETWEEN.

CASO PRÁCTICO 8

Para obtener los empleados con una categoría entre 1 y 3, se puede hacer lo siguiente:

```
SELECT nombre, apellidos FROM empleados WHERE categoria >= 1 AND categoria <= 3
```

Con el operador BETWEEN queda así:

```
SELECT nombre, apellidos FROM empleados WHERE categoría BETWEEN 1 AND 3
```

Empleados que no estén en la categoría 2, 3, 4 o 5:

```
SELECT nombre, apellidos FROM empleados WHERE categoria NOT BETWEEN 2 AND 5
```

IMPORTANTE

Primero se indica el extremo inferior del rango y luego el superior. Por ejemplo, para saber si un valor está entre 1 y 100:

✓ BETWEEN 1 AND 100

✗ BETWEEN 100 AND 1

3.7. Patrón de valores (LIKE)

El operador LIKE permite seleccionar las cadenas de texto que cumplan una determinada condición. Se pueden utilizar los comodines _ (para representar un carácter) o % (para representar cualquier número de caracteres, incluido cero caracteres). También sirve para comprobar que no cumplen un patrón con NOT LIKE.

CASO PRÁCTICO 9

Para obtener los empleados cuyo nombre comienza por M:

```
SELECT nombre, apellidos FROM empleados WHERE nombre LIKE 'M%'
```

Para obtener los empleados con apellido García (ya sea el primero o el segundo):

```
SELECT nombre, apellidos FROM empleados WHERE apellidos LIKE '%García%'
```

Para obtener los códigos de producto que tengan una X en la tercera posición:

```
SELECT codigo, nombre FROM productos WHERE codigo LIKE '__X%'
```

Para obtener las direcciones de empleados que no empiezan por 'CALLE':

```
SELECT direccion FROM empleados WHERE direccion NOT LIKE 'calle%'
```

3.8. Ordenación

Por defecto, el resultado de una consulta se muestra sin un orden concreto, normalmente se aplica el orden en el que se insertaron los datos en la base de datos.

Para mostrar el resultado ordenado se utiliza la cláusula ORDER BY. Esta cláusula permite ordenar el conjunto de resultados de forma ascendente (ASC) o descendente (DESC) por una o varias columnas. El valor por defecto es ASC.

La columna por la que se quiere ordenar se puede expresar por el nombre de la columna, una expresión o bien la posición numérica del campo que se quiere ordenar. Las columnas utilizadas en la ordenación no tienen por qué incluirse en la propia consulta (es decir, no tienen por qué añadirse en la cláusula SELECT).

CASO PRÁCTICO 10

Para obtener los clientes ordenados por apellidos (si no se indica otra cosa, se supone ordenación ascendente):

```
SELECT nombre, apellidos FROM clientes ORDER BY apellidos
```

Los clientes ordenados por apellidos y nombre en orden descendente:

```
SELECT nombre, apellidos FROM clientes ORDER BY apellidos DESC, nombre DESC
```

Los clientes ordenados por el apellido (el segundo campo que se devuelve):

```
SELECT nombre, apellidos FROM clientes ORDER BY 2
```

3.9. Consultas de resumen

En SQL se pueden generar consultas más complejas que resumen cierta información, extrayendo información calculada de varios conjuntos de registros. Para poder generar información resumida hay que hacer uso de las funciones de resumen o de agregación. Estas funciones convierten un conjunto de registros en una información simple cuyo resultado es un cálculo. Las funciones de resumen más utilizadas se muestran en la siguiente tabla:

FUNCIÓN	DESCRIPCIÓN
SUM(columna)	Suma los valores de la columna.
AVG(columna)	Calcula el valor medio.
MIN(columna)	Calcula el mínimo.
MAX(columna)	Calcula el máximo.
COUNT(columna)	Cuenta el número de valores (sin contar valores nulos).
COUNT(*)	Cuenta el número de filas de una tabla.

Funciones de resumen que soporta MySQL:

<https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html>

IMPORTANTE

Las funciones SUM() y AVG() sólo tienen sentido sobre columnas de tipo numérico. Sobre el resto de tipos el resultado que devuelve siempre es cero.

CASO PRÁCTICO 11

Para obtener el número de clientes:

```
SELECT COUNT(*) FROM clientes
```

Para obtener el salario medio de los empleados:

```
SELECT AVG(salario) FROM empleados
```

Para obtener el número de direcciones informadas en la tabla de clientes:

```
SELECT COUNT(direccion) FROM clientes
```

Para obtener el primer y último apellido (por orden alfabético) de los clientes:

```
SELECT MIN(apellidos), MAX(apellidos) FROM clientes
```

Para obtener el importe de todos los productos del pedido P0000123:

```
SELECT SUM(precio * cantidad) FROM detalle_pedido WHERE pedido = 'P0000123'
```

3.10. Agrupaciones

Las funciones de resumen mostradas en el apartado anterior aplican a todos los registros de una tabla, pero se pueden realizar sobre agrupaciones de registros. Se denomina agrupación de registros a un conjunto de registros con una o varias columnas con el mismo valor. Por ejemplo:

- Todos los clientes de Málaga tienen en común la ciudad.
- Todos los empleados del departamento Ventas que trabajan en Málaga tienen en común el departamento y la ciudad donde trabajan.

CASO PRÁCTICO 12

Para obtener los empleados de cada departamento:

```
SELECT departamento, COUNT(*) FROM empleados GROUP BY departamento
```

Para obtener el salario medio de los empleados por departamento:

```
SELECT departamento, AVG(salario) FROM empleados GROUP BY departamento
```


Para obtener el salario más bajo de los empleados por departamento y ciudad:

```
SELECT departamento, ciudad, MIN(salario) FROM empleados  
GROUP BY departamento, ciudad
```

IMPORTANTE

En consultas agrupadas, en la cláusula SELECT sólo pueden aparecer columnas sobre las que se realiza el agrupamiento o funciones de resumen.

Si es necesario filtrar los grupos obtenidos en una consulta, se debe utilizar la cláusula HAVING (la cláusula WHERE es para filtrar registros y actúa antes de agrupar el resultado). En la cláusula HAVING sólo pueden añadirse las columnas o expresiones indicadas en la cláusula GROUP BY.

CASO PRÁCTICO 13

Para obtener los empleados de cada departamento con al menos 10 empleados:

```
SELECT departamento, COUNT(*) FROM empleados  
GROUP BY departamento HAVING COUNT(*) >= 10
```

Para obtener el salario medio de los empleados por departamento y ciudad distinta de Málaga, mostrando solo los departamentos cuyo salario medio sea inferior a 50.000€:

```
SELECT departamento, ciudad, AVG(salario) FROM empleados  
WHERE ciudad <> 'Málaga'  
GROUP BY departamento, ciudad HAVING AVG(salario) < 50000
```

Como la consulta agrupa por ciudad, también podría hacerse así:

```
SELECT departamento, ciudad, AVG(salario) FROM empleados  
GROUP BY departamento, ciudad  
HAVING AVG(salario) < 50000 and ciudad <> 'Málaga'
```

4. Subconsultas

Una subconsulta es una consulta dentro de otra, se utilizan para realizar filtrados con los datos de la subconsulta. Estos filtros pueden ser aplicados tanto en la cláusula WHERE (para filtrar registros) como en la cláusula HAVING (para filtrar grupos).

4.1. Comparación con subconsultas

Consiste en usar los operadores de comparación (=, <>, >, >=, <, <=) para comparar el valor de una columna o expresión con el resultado de una subconsulta. La subconsulta debe devolver un único valor y debe colocarse en la parte derecha de la comparación (es decir, a la derecha del operador utilizado).

CASO PRÁCTICO 14

Para obtener nombre y apellidos del empleado con el salario más alto:

```
SELECT nombre, apellidos FROM empleados  
WHERE salario = (SELECT MAX(salario) FROM empleados)
```

4.2. Pertenencia a conjunto con subconsultas

Consiste en usar el operador IN para filtrar los registros cuya expresión coincida con algún valor producido por la subconsulta.

CASO PRÁCTICO 15

Para obtener nombre y apellidos de los clientes de la provincia de Málaga:

```
SELECT nombre, apellidos FROM clientes  
WHERE ciudad IN (SELECT ciudad FROM ciudades WHERE provincia = 'Málaga')
```

4.3. Existencia de datos con subconsultas

Consiste en usar el operador EXISTS para filtrar los registros si existen resultados en la subconsulta. También se puede utilizar para el caso opuesto con NOT EXISTS.

CASO PRÁCTICO 16

Para obtener los pedidos que incluyen el producto 1031:

```
SELECT referencia FROM pedidos WHERE EXISTS  
(SELECT producto FROM detalle_pedido WHERE referencia = pedidos.referencia  
AND producto = 1031)
```

4.4. Cuantificadores ALL y ANY

Sirven para calcular la relación entre una expresión y todos los registros de la subconsulta (ALL) o algunos de los registros de la subconsulta (ANY).

CASO PRÁCTICO 17

Para obtener nombre y apellidos del empleado con el salario más alto (ver 4.1.):

```
SELECT nombre, apellidos FROM empleados  
WHERE salario >= ALL (SELECT salario FROM empleados)
```

Para obtener nombre y apellidos de los clientes de la provincia de Málaga (ver 4.2.):

```
SELECT nombre, apellidos FROM clientes  
WHERE ciudad = ANY (SELECT ciudad FROM ciudades WHERE provincia = 'Málaga')
```

4.5. Subconsultas anidadas

Es posible crear subconsultas anidadas. Es decir, que una subconsulta use a su vez una subconsulta para filtrar los resultados. No hay límite de anidamiento, por lo que se pueden anidar cuantas subconsultas sean necesarias.

CASO PRÁCTICO 18

Para obtener los clientes de Andalucía:

```
SELECT nombre, apellidos  
FROM clientes  
WHERE ciudad IN (SELECT ciudad FROM ciudades  
                  WHERE provincia IN (SELECT provincia  
                                      FROM provincias  
                                      WHERE ccaa = 'Andalucía'))
```

5. Consultas multitable

Hasta ahora se han mostrado ejemplos donde los resultados se almacenaban en una tabla. No obstante, es posible combinar más de una tabla en una consulta formando así una consulta multitable.

5.1. Producto cartesiano

La combinación de tablas es posible gracias al producto cartesiano. Consiste en combinar todos los registros de la primera tabla con todos los registros de la segunda tabla. Por ejemplo, si tenemos la tabla A(a, b) y B(c, d), con los siguientes valores:

Tabla A	
a	b
a1	b1
a2	b2
a3	b3

Tabla B	
c	d
c1	d1
c2	d2

El producto cartesiano resultante de combinar las tablas A y B consiste en la combinación de todas las filas y columnas de la tabla A con todas las filas y columnas de la tabla B:

A x B			
a	B	c	d
a1	b1	c1	d1
a1	b1	c2	d2
a2	b2	c1	d1
a2	b2	c2	d2
a3	b3	c1	d1
a3	b3	c2	d2

El producto cartesiano de las tablas A y B se representa en SQL así:

```
SELECT * FROM A, B
```

5.2. Combinación de tablas (versión 1)

El resultado del producto cartesiano sólo tiene sentido en aquellas filas donde hay relación de datos, es decir, donde la clave foránea de una tabla coincide con la clave primaria de la otra tabla. Por tanto, para realizar una combinación de tablas correcta (en inglés, *join*) el siguiente paso es filtrar el producto cartesiano.

CASO PRÁCTICO 19

Para obtener los nombre y apellidos de los empleados junto con el nombre del departamento en el que trabajan:

```
SELECT nombre, apellidos, departamento
FROM empleados, departamentos
WHERE departamento_id = id
```

IMPORTANTE

La consulta anterior funciona correctamente si los nombres de las columnas son únicos para cada tabla. En caso de colisión de nombres (es lo más común) hay que indicar junto a cada columna la tabla a la que pertenece. Para facilitar la consulta, es aconsejable indicar en la cláusula FROM un alias para cada tabla:

```
SELECT E.nombre, E.apellidos, D.departamento
FROM empleados E, departamentos D
WHERE E.departamento_id = D.id
```

En una consulta no hay límite de tablas y se pueden combinar tantas tablas como hagan falta.

CASO PRÁCTICO 20

Para obtener los nombre y apellidos de los empleados junto con el nombre del departamento en el que trabajan y la ciudad donde residen:

```
SELECT E.nombre, E.apellidos, D.departamento, C.nombre
FROM empleados E, departamentos D, ciudades C
WHERE (E.departamento_id = D.id) AND (E.ciudad_id = C.id)
```

5.3. Combinación de tablas (versión 2)

La combinación de tablas explicada en la versión anterior se incluyó en la primera versión de SQL (SQL1 o SQL-86) y sólo permite la combinación por equivalencia. Por ejemplo, imagina la siguiente consulta:

```
SELECT E.nombre, E.apellidos, D.departamento  
FROM empleados E, departamentos D  
WHERE E.departamento_id = D.id
```

El resultado sólo va a incluir a aquellos empleados que pertenezcan a un departamento. Es decir, aquellos empleados donde el departamento no esté informado (E.departamento_id IS NULL) no se incluyen en el resultado.

Con SQL2 (SQL-92) se introduce una nueva sintaxis (aunque la anterior también está soportada) y se amplía el concepto de composición de tablas distinguiéndose los siguientes tipos:

1. Producto cartesiano (CROSS JOIN).
2. Composición interna: composición de equivalencia (INNER JOIN) y composición natural (NATURAL JOIN).
3. Composición externa: de tabla izquierda (LEFT OUTER JOIN), de tabla derecha (RIGHT OUTER JOIN) y completa (FULL OUTER JOIN).

PRODUCTO CARTESIANO (CROSS JOIN)

Es equivalente al producto cartesiano de SQL1.

CASO PRÁCTICO 21

El producto cartesiano con sintaxis SQL1:

```
SELECT * FROM A, B
```

En SQL2 se representa así:

```
SELECT * FROM A CROSS JOIN B
```

COMPOSICIÓN DE EQUIVALENCIA (INNER JOIN)

Es equivalente a la composición interna de SQL1.

CASO PRÁCTICO 22

La composición de equivalencia con sintaxis SQL1:

```
SELECT E.nombre, E.apellidos, D.departamento, C.nombre  
FROM empleados E, departamentos D, ciudades C  
WHERE (E.departamento_id = D.id) AND (E.ciudad_id = C.id)
```

En SQL2 se representa así:

```
SELECT E.nombre, E.apellidos, D.departamento, C.nombre FROM empleados E  
INNER JOIN departamentos D ON E.departamento_id = D.id  
INNER JOIN ciudades C ON E.ciudad_id = C.id
```

Al igual que en la consulta en SQL1, no se tienen en cuenta los valores nulos. Por tanto, el resultado de ambas consultas contiene únicamente los empleados donde la ciudad y el departamento están informados.

COMPOSICIÓN NATURAL (NATURAL JOIN)

Es una especialización de INNER JOIN. En este caso se comparan todas las columnas con el mismo nombre en ambas tablas. El resultado contiene una única columna por cada par de columnas con el mismo nombre en ambas tablas, por lo que para esas columnas no es necesario poner como prefijo el nombre de la tabla.

CASO PRÁCTICO 23

La consulta con INNER JOIN:

```
SELECT E.nombre, E.apellidos, D.dpto_id, D.departamento  
FROM empleados E INNER JOIN departamentos D ON E.dpto_id = D.dpto_id
```

Se puede representar así con NATURAL JOIN:

```
SELECT E.nombre, E.apellidos, dpto_id, D.departamento  
FROM empleados E NATURAL JOIN departamentos D
```

IMPORTANTE

Ambas consultas son equivalentes siempre y cuando en las tablas no hay más campos con el mismo nombre (sólo el campo dpto_id que se usa para combinar).

COMPOSICIÓN DE TABLA IZQUIERDA (LEFT OUTER JOIN)

Las composiciones externas se incluyen en SQL2 para dar tratamiento a los valores nulos. En estos casos no se requiere que haya una equivalencia y un registro de una tabla puede ser mostrado en el resultado de la consulta, aunque no haya una correspondencia en la otra tabla.

Si se desea añadir todos los registros de la tabla de la izquierda, aunque no haya equivalencia con la tabla de la derecha, se realiza un LEFT OUTER JOIN o LEFT JOIN (la palabra OUTER es opcional).

CASO PRÁCTICO 24

Listado de todos los empleados asociados a departamento, no mostrando los que están sin departamento:

```
SELECT E.nombre, E.apellidos, D.dpto_id, D.departamento  
FROM empleados E INNER JOIN departamentos D ON E.dpto_id = D.dpto_id
```

Si además se quieren mostrar los empleados sin departamento, es necesario realizar un LEFT JOIN:

```
SELECT E.nombre, E.apellidos, D.dpto_id, D.departamento  
FROM empleados E LEFT JOIN departamentos D ON E.dpto_id = D.dpto_id
```

Se utiliza LEFT JOIN porque la tabla de empleados está a la izquierda de la instrucción LEFT JOIN. En esta consulta se incluyen todos los empleados y, si tienen asociado departamento, el nombre del departamento. Si no lo tiene, se muestra NULL.

COMPOSICIÓN DE TABLA DERECHA (RIGHT OUTER JOIN)

Esta composición funciona exactamente del mismo modo que el caso anterior simplemente se cambia el sentido.

Si se desea añadir todos los registros de la tabla de la derecha, aunque no haya equivalencia con la tabla de la izquierda, se realiza un RIGHT OUTER JOIN o RIGHT JOIN (la palabra OUTER es opcional).

CASO PRÁCTICO 25

Si se desea mostrar los empleados con el departamento al que pertenecen e incluir los departamentos sin empleados:

```
SELECT E.nombre, E.apellidos, D.dpto_id, D.departamento  
FROM empleados E RIGHT JOIN departamentos D ON E.dpto_id = D.dpto_id
```

Al indicar RIGHT JOIN, se incluyen todos los departamentos (la tabla de la derecha) aunque no tengan asociado empleados.

COMPOSICIÓN COMPLETA (FULL OUTER JOIN)

Si lo que se desea es añadir registros sin correspondencia tanto a izquierda como a derecha, se puede utilizar un FULL OUTER JOIN o FULL JOIN (la palabra OUTER es opcional).

CASO PRÁCTICO 26

Para mostrar los empleados sin departamento y los departamentos sin empleados:

```
SELECT E.nombre, E.apellidos, D.dpto_id, D.departamento  
FROM empleados E FULL JOIN departamentos D ON E.dpto_id = D.dpto_id
```

IMPORTANTE

MySQL aun no soporta la composición FULL JOIN, pero es posible simularlo con la instrucción UNION (se ve a continuación).

6. Combinación de consultas

En ocasiones puede ser necesario combinar los resultados de consultas. Hay que tener en cuenta que las tablas son conjuntos de datos, SQL soporta las siguientes operaciones sobre conjuntos:

- Unión. Devuelve todos los resultados de las consultas, comunes y no comunes.
- Intersección. Devuelve los resultados comunes de las consultas.
- Diferencia. Para consultas A y B, devuelve los resultados de A que no están en B.

Se pueden combinar dos o más consultas con las siguientes restricciones:

- Cada consulta debe devolver el mismo número de columnas.
- El tipo de datos de cada columna debe coincidir en todas las consultas. Por ejemplo, si la primera columna de la primera consulta es un número, el resto de consultas deben devolver un número en la primera columna.
- Sólo se puede añadir una cláusula ORDER BY al final de la combinación, que aplica al resultado de la combinación de las consultas. Además, no se pueden utilizar los nombres de las columnas sino sus posiciones. Por ejemplo, para ordenar por el primer campo en sentido descendente debe indicarse ORDER BY 1 DESC.

6.1. Unión de consultas (UNION)

La unión de consultas combina todos los resultados de las consultas. Los registros repetidos se filtran por defecto, a menos que expresamente se indique lo contrario (UNION ALL).

CASO PRÁCTICO 27

Como ya se comentó antes, la composición externa FULL JOIN no está soportada por MySQL, pero se puede simular con una unión entre LEFT JOIN y RIGHT JOIN.

Para obtener los empleados con su departamento (incluyendo los empleados sin departamento y los departamentos sin empleados), ordenados por departamento, apellidos y nombre:

```
SELECT E.nombre, E.apellidos, D.dpto_id, D.departamento
FROM empleados E LEFT JOIN departamentos D ON E.dpto_id = D.dpto_id
UNION
SELECT E.nombre, E.apellidos, D.dpto_id, D.departamento
FROM empleados E RIGHT JOIN departamentos D ON E.dpto_id = D.dpto_id
ORDER BY 3, 2, 1
```

6.2. Intersección de consultas (INTERSECT)

La intersección de consultas combina todos los resultados comunes de las consultas.

CASO PRÁCTICO 28

Supongamos que un mismo empleado puede trabajar para más de un departamento. Para conocer los empleados que trabajan en Ventas y Dirección:

```
SELECT E.nombre, E.apellidos FROM empleados E
INNER JOIN departamentos D ON E.dpto_id = D.dpto_id AND D.nombre = 'Ventas'
INTERSECT
SELECT E.nombre, E.apellidos FROM empleados E
INNER JOIN departamentos D ON E.dpto_id = D.dpto_id AND D.nombre = 'Dirección'
```

NOTA: si un empleado puede estar en más de un departamento la tabla Empleados no estaría normalizada (el campo E.dpto es multivaluado), se deja así para mostrar de manera sencilla la intersección de consultas.

6.3. Diferencia de consultas (EXCEPT)

La diferencia de consultas combina todos los resultados de la primera tabla que no están en la segunda.

IMPORTANTE

La unión y la intersección de conjuntos son operaciones asociativas. Es decir:

$$(A \cup B) \cup C = A \cup (B \cup C)$$

Sin embargo, la diferencia no es asociativa. Por tanto, cuando sea necesaria la diferencia de más de dos consultas, es conveniente indicar con paréntesis el orden de las operaciones.

CASO PRÁCTICO 29

Supongamos que un mismo empleado puede trabajar para más de un departamento. Para conocer los empleados que trabajan en Ventas y no en Dirección:

```
SELECT E.nombre, E.apellidos FROM empleados E  
INNER JOIN departamentos D ON E.dpto_id = D.dpto_id AND D.nombre = 'Ventas'  
EXCEPT  
SELECT E.nombre, E.apellidos FROM empleados E  
INNER JOIN departamentos D ON E.dpto_id = D.dpto_id AND D.nombre = 'Dirección'
```

7. Resumen

El DML es la parte del lenguaje SQL que se encarga de la gestión de la información contenida en una base de datos. La instrucción INSERT permite añadir (insertar) información y la instrucción SELECT permite recuperar (consulta, lectura) información.

SELECT es la instrucción SQL más utilizada. Su sintaxis varía entre la sencillez de las consultas a una tabla y la complejidad de las composiciones internas y externas (consultas a varias tablas, en el segundo de los casos incluyendo los registros que no guardan correspondencia con registros en otras tablas). SELECT también permite incluir consultas (subconsultas) dentro de consultas y combinar consultas mediante los operadores UNION, EXCEPT e INTERSECT.

La información que se obtiene se depura aún más gracias al uso de funciones de agregación usadas en solitario o con las cláusulas GROUP BY y HAVING. Finalmente, con la cláusula ORDER BY se puede ordenar la información obtenida en la consulta por el criterio más conveniente en cada caso.