DML

- **Lenguaje de Manipulación de Datos** (**D**ata **M**anipulation **L**anguage, DML) es un lenguaje proporcionado por los sistemas gestores de bases de datos que permite a los usuarios de la misma llevar a cabo las tareas de consulta o modificación de los datos contenidos en las Bases de Datos del Sistema Gestor de Bases de Datos.

Comando	Descripción
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
DELETE	Utilizado para modificar los valores de los campos y registros especificados.
UPDATE	Utilizado para eliminar registros de una tabla de una base de datos.

INSERT

- Antes de poder consultar los datos de una base de datos, es preciso introducirlos con la sentencia INSER INTO VALUES, que tiene el formato:

```
INSERT INTO nombre_tabla [(columnas)]
{VALUES ({v1|DEFAULT|NULL}, ..., {vn/DEFAULT/NULL})|<consulta>};
```

```
INSERT INTO clientes
VALUES (10, 'ECIGSA', '37.248.573-C', 'ARAGON 242', 'Barcelona', DEFAULT);
```

o bien:

```
INSERT INTO clientes (nif, nombre_cli, codigo_cli, telefono, direccion,
ciudad)
VALUES ('37.248.573-C', 'ECIGSA', 10, DEFAULT, 'ARAGON 242', 'Barcelona');
```

SELECT

SELECT

- Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros que se pueden almacenar en un objeto recordset. Este conjunto de registros es modificable.

SELECT

SELECT Campos
FROM Tabla

SELECT Nombre, Teléfono
FROM Clientes

SELECT: ORDER BY

- Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la claúsula **ORDER BY** Lista de Campos. En donde Lista de campos representa los campos a ordenar. Ejemplo:

```
SELECT CodigoPostal, Nombre, Telefono
FROM Clientes
ORDER BY Nombre
```

```
SELECT CodigoPostal, Nombre, Telefono
FROM Clientes
ORDER BY CodigoPostal, Nombre
```

```
SELECT CodigoPostal, Nombre, Telefono
FROM Clientes
ORDER BY CodigoPostal DESC, Nombre ASC
```

SELECT: DISTINCT

- El predicado **DISTINCT** omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos.
- Por ejemplo, varios empleados listados en la tabla Empleados pueden tener el mismo apellido. Si dos registros contienen López en el campo Apellido, la siguiente instrucción SQL devuelve un único registro:

```
SELECT DISTINCT    Apellido
FROM               Empleados
```

Selección Condicional, Operadores de Comparación y Lógicos

Selección Condicional

- la cláusula **WHERE** se utiliza para determinar qué registros de las tablas enumeradas en la cláusula **FROM** aparecerán en los resultados de la instrucción **SELECT**. Depués de escribir esta cláusula se deben especificar las condiciones expuestas a continuación. Si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla. **WHERE** es opcional, pero cuando aparece debe ir a continuación de **FROM**.

Selección Condicional

=	Igual
<>	Distinto
<	Menor
>	Mayor
<=	Menor Igual
<=	Mayor Igual
between	Utilizado para especificar un intervalo de valores.
like	Utilizado en la comparación de un campo contra un patrón
in	Utilizado para verificar la existencia de un valor dentro de un conjunto de valores

El operador Between

- Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador Between cuya sintaxis es:

  campo [Not] Between valor1 And valor2

- En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo.

```
SELECT      *
FROM        Pedidos
WHERE       CodPostal
BETWEEN     28000 and 28999
```

El Operador Like

- Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

  expresión Like modelo

En donde expresión es un patrón contra el que se compara expresión. Se puede utilizar el operador **like** para encontrar valores en los campos que coincidan con el modelo especificado. Por modelo puede especificar un valor completo ("Ana María"), o se pueden utilizar caracteres comodín para encontrar un rango de valores (Like "An%").

- El operador **like** se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, si introduce Like "C%" en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra C. En una consulta con parámetros, puede hacer que el usuario escriba el modelo que se va a utilizar.

El Operador in

- Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los valores que se encuentran una lista explicitada. Su sintaxis es:

expresión [Not] In(valor1, valor2, . . .)

SELECT	*
FROM	Pedidos
WHERE	Provincia
IN	("Santa Fe", "Cordoba", "Buenos Aires")

Operadores Lógicos

- Hay tres operadores lógicos en SQL:

AND	Es el "y" lógico. Evalua dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdar si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

Agrupamiento de Registros

GROUP BY

- Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro se crea un valor sumario si se incluye una función SQL agregada, como por ejemplo SUM o COUNT, en la instrucción SELECT. Su sintaxis es:

SELECT campos FROM tabla WHERE criterio GROUP BY campos del grupo

GROUP BY

- GROUP BY es opcional. Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción SELECT. Los valores Null en los campos GROUP BY se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas.

- Se utiliza la cláusula WHERE para excluir aquellas filas que no desea agrupar, y la cláusula HAVING para filtrar los registros una vez agrupados.

- Todos los campos de la lista de campos de SELECT deben o bien incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada.

```
SELECT Id_Familia, Sum(Stock) FROM
Productos GROUP BY Id_Familia
```

- Una vez que **GROUP BY** ha combinado los registros, **HAVING** muestra cualquier registro agrupado por la cláusula GROUP BY que satisfaga las condiciones de la cláusula HAVING.
- **HAVING** es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando GROUP BY, HAVING determina cuales de ellos se van a mostrar.

SELECT	Carrera, Count(*)
FROM	Alumnos
GROUP BY	Carrera
HAVING	Count(*)>100

Esta consulta lista las carreras que tengan más de 100 alumnos

Función de Agregación: AVG

- Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta. Su sintaxis es la siguiente

Avg(expr)

En donde expr representa el campo que contiene los datos numéricos para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por **Avg** es la media aritmética (la suma de los valores dividido por el número de valores). La función **Avg** no incluye ningún campo Null en el cálculo.

```
SELECT    Avg(Gastos) as Promedio
FROM      Pedidos
WHERE     Gastos > 100
```

Esta consulta lista el promedio de gastos calculado sobre los pedidos en los que se gastó más de 100.

Función de Agregación: COUNT

- Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente:

Count(expr)

- Aunque expr puede realizar un cálculo sobre un campo, Count simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función Count no cuenta los registros que tienen campos null. Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos null. Count(*) es considerablemente más rápida que Count(Campo).

```
SELECT    Count(*) as Total
FROM      Pedidos
```

Funciones de Agregación: MAX y MIN

- Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo especifico de una consulta. Su sintaxis es:

Min(expr)
Max(expr)

En donde expr es el campo sobre el que se desea realizar el cálculo. Expr pueden incluir el nombre de un campo de una tabla, una constante.

```
SELECT    Max(Gastos) as ElMaximo
FROM      Pedidos
WHERE     Pais = "España"
```

```
SELECT    Min(Gastos) as ElMinimo
FROM      Pedidos
WHERE     Pais = "España"
```

Función de Agregación: SUM

- Devuelve la suma del conjunto de valores contenido en un campo especifico de una consulta. Su sintaxis es:

Sum(expr)

En donde expr respresenta el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos.

```
SELECT    Sum(PrecioUnidad * Cantidad) as Total
FROM      DetallePedido
```

Operadores Lógicos IN y EXISTS

IN

- Determina si un valor dado coincide con algún valor de una subconsulta o lista. Su sintaxix es:

WHERE {Campo | Expresión} [NOT] IN (Sub-Consulta | Constantes [, Constantes, ... n])

El resultado del operador lógico IN es un valor booleano, es decir Verdadero o Falso, para cada valor de Campo que se encuentre (o no) en la sub-consulta.

EXISTS

- Especifica una subconsulta para probar la existencia de filas. Su sintaxis es la siguiente:

WHERE [NOT] EXISTS (Sub - Consulta)

La subconsulta en realidad no produce ningún dato, al igual que **IN** devuelve el valor **Verdadero** o **Falso**.

EXIST

- El ejemplo siguiente compara dos consultas que son semánticamente equivalentes. La primera consulta utiliza EXISTS y la segunda consulta utiliza IN. Ambas consultas devuelven la misma información.

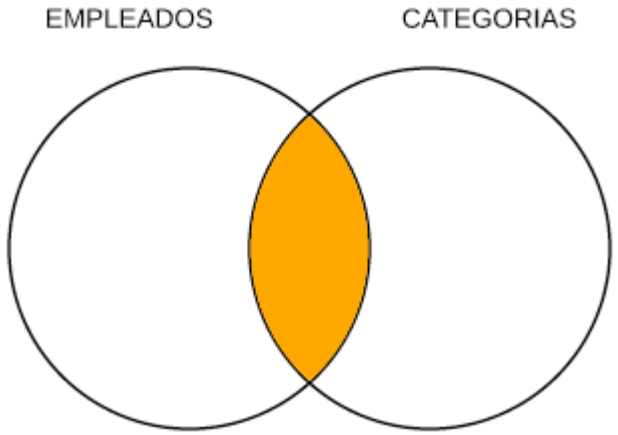
```
SELECT DISTINCT Nombre_Editor
FROM Editores E
WHERE EXISTS (
    SELECT *
    FROM Titulos T
    WHERE T.IdEditor = E.IdEditor
    AND T.Tipo="Negocios"
)
```

```
SELECT DISTINCT Nombre_Editor
FROM Editores
WHERE IdEditor IN (
    SELECT IdEditor
    FROM Titulos
    WHERE T.Tipo="Negocios"
)
```

Tipos de JOINS

INNER JOIN

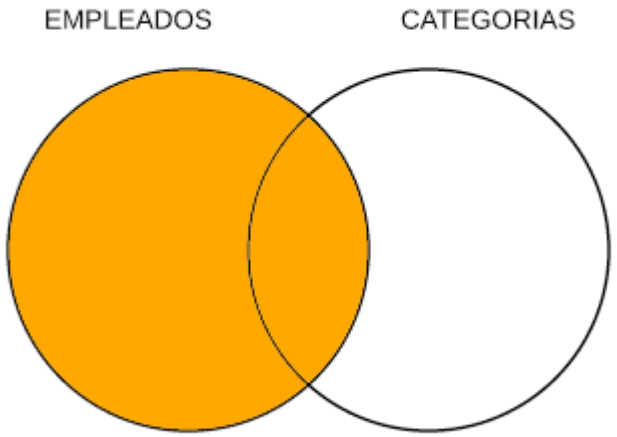
- La forma más utilizada de combinación se llama INNER JOIN, y el resultado es el cálculo cruzado de todos los registros. Se combinan los registros de la tabla Empleados con la tabla Categorías, pero sólo van a permanecer los registros que satisfagan la condición especificada.




```
SELECT e.Id, e.Apellido, c.Id, c.Nombre
FROM Empleados e
INNER JOIN Categorías c
    ON e.Categoria = c.Id
```

LEFT JOIN

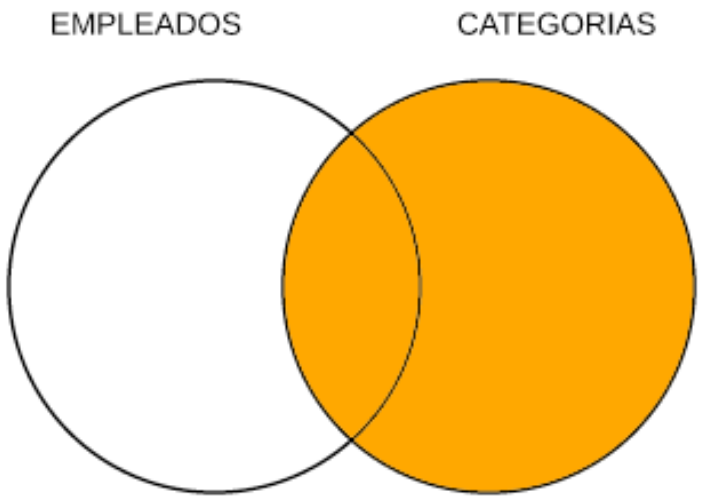
- Si lo que necesitamos es una lista de los empleados, sin importar si pertenecen o no a una categoría, entonces lo que tenemos que utilizar es LEFT JOIN. El resultado que produce este JOIN son todas las filas de la tabla que se encuentre a la izquierda, sin importar si coinciden con las filas de la derecha.




```
SELECT e.Id, e.Apellido, c.Id, c.Nombre
FROM Empleados e
LEFT JOIN Categorías c
    ON e.Categoria = c.Id
```

RIGHT JOIN

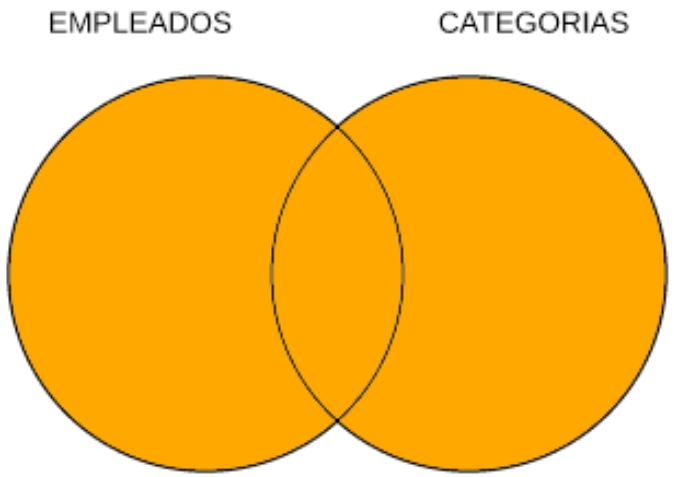
- De la misma forma que podemos obtener todas las filas de la tabla de la izquierda con LEFT JOIN, con RIGHT JOIN obtenemos todas las filas de la tabla derecha, sin importar si coinciden con las de la tabla de la izquierda.




```
SELECT e.Id, e.Apellido, c.Id, c.Nombre
FROM Empleados e
RIGHT JOIN Categorías c
    ON e.Categoria = c.Id
```

FULL JOIN

- El FULL JOIN se utiliza cuando queremos obtener todas las filas de las dos tablas, sin importarnos que tengan coincidencias. MySQL no soporta este tipo de JOIN, por lo tanto para lograr el mismo resultado, hay que hacer LEFT JOIN + RIGHT JOIN.




```sql
SELECT e.Id, e.Apellido, c.Id, c.Nombre
FROM Empleados e
FULL JOIN Categorías c
    ON e.Categoria = c.Id
```

DELETE

- Para borrar valores de algunas filas de una tabla podemos utilizar la sentencia DELETE FROM WHERE. Su formato es el siguiente:

```
DELETE FROM nombre_tabla
[WHERE condiciones];
```

- En cambio, si lo que quisiéramos conseguir es borrar todas las filas de una tabla, entonces sólo tendríamos que poner la sentencia DELETE FROM, sin WHERE.

Podemos dejar la tabla `proyectos` sin ninguna fila:

```
DELETE FROM proyectos;
```

En nuestra base de datos, borrar los proyectos del cliente 2 se haría de la forma que mostramos a continuación:

```
DELETE FROM proyectos
WHERE codigo_cliente = 2;
```

UPDATE

- Si quisiéramos modificar los valores de algunas filas de una tabla, tendríamos que utilizar la sentencia UPDATE SET WHERE. A continuación presentamos su formato:

```
UPDATE nombre_tabla
SET columna = {expresión|DEFAULT|NULL}
    [, columna = {expr|DEFAULT|NULL} ...]
WHERE condiciones;
```

Supongamos que queremos incrementar el sueldo de todos los empleados del proyecto 2 en 1.000 euros. La modificación a ejecutar sería:

```
UPDATE empleados
SET sueldo = sueldo + 1000
WHERE num_proyec = 2;
```