

SELECT - Parte I

SELECT

Se utiliza para “selección” (y consulta) de filas y columnas

Para hacer consultas sobre los registros de la tabla y seleccionar datos, se utiliza la sentencia **SELECT**.

El resultado de la consulta es una nueva tabla que contiene la información solicitada.

Sintaxis:

```
SELECT column1, column2,..., columnN  
FROM table_name  
WHERE condition  
ORDER BY column1 ASC/DESC  
LIMIT cantidad_filas;
```

SELECT

Las instrucciones detalladas en azul son optativas.

WHERE: permite especificar alguna condición para filtrar datos.

ORDER BY: permite ordenar los resultados obtenidos en forma ascendente (**ASC**) o descendente (**DESC**)

LIMIT: permite limitar el resultado que se muestra, a la cantidad de filas indicadas (cantidad_filas)

Se usa el ***** para informar a la consulta que quiero todas las columnas. Es un “comodín”.

Sintaxis:

```
SELECT column1, column2,..., columnN  
FROM table_name  
WHERE condition  
ORDER BY column1 ASC/DESC  
LIMIT cantidad_filas;
```

Nos conectamos a Mysql, Base Northwind



1. Vamos trabajar con el schema de la base de datos Northwind.
2. En Playground pueden descargar el script **“Ejemplos Clase II SQL”**. En el mismo encontrarán todo los ejemplos que veremos en clase, para que puedan probarlos.

SELECT
WHERE

SELECT WHERE

Filtrar Filas (o registros)

En los casos en los que se quiere ejecutar una sentencia SELECT que retorne solo los registros que cumplen alguna condición específica se utiliza el operador **WHERE**. El uso de la cláusula **WHERE** es opcional y puede no incluirse en caso de no ser necesario.

```
SELECT *  
FROM products  
WHERE id = 3;
```

```
SELECT *  
FROM products  
WHERE list_price > 10;
```

En color **ROJO** se observan los operadores lógicos de comparación.

SELECT WHERE

Algunos Operadores Lógicos

Operator	Description	Example
=	Equal to	Author = 'Alcott'
<>	Not equal to (many DBMSs accept != in addition to <>)	Dept <> 'Sales'
>	Greater than	Hire_Date > '2012-01-31'
<	Less than	Bonus < 50000.00
>=	Greater than or equal	Dependents >= 2
<=	Less than or equal	Rate <= 0.05
BETWEEN	Between an inclusive range	Cost BETWEEN 100.00 AND 500.00
LIKE	Match a character pattern	First_Name LIKE 'Will%'
IN	Equal to one of multiple possible values	DeptCode IN (101, 103, 209)
IS or IS NOT	Compare to null (missing data)	Address IS NOT NULL
IS NOT DISTINCT FROM	Is equal to value or both are nulls (missing data)	Debt IS NOT DISTINCT FROM - Receivables
AS	Used to change a field name when viewing results	SELECT employee AS 'department1'

Select: Ejemplo #1

```
SELECT *  
FROM products  
WHERE id = 3;
```

Palabras reservadas de SQL

En las condiciones del WHERE, debo considerar el tipo de datos de la columna, en este ejemplo es un número entero.

Select: Ejemplo #2

```
SELECT *  
FROM products  
WHERE product_code = "NWTC0-3";
```

Palabras reservadas de SQL

En las condiciones del WHERE, debo considerar el tipo de datos de la columna, en este ejemplo es una cadena de caracteres, por lo que el valor constante debe ir entre comillas dobles.

Select: Ejemplo #3

```
SELECT *  
FROM orders  
WHERE order_date > '2006-05-01';
```

Palabras reservadas de SQL

En las condiciones del WHERE, debo considerar el tipo de datos de la columna, en este ejemplo es un date, por lo que el valor constante debe ir entre comillas simples.

Select: Ejemplo #4

```
SELECT id, product_code, product_name  
FROM products  
WHERE list_price > 10;
```

Palabras reservadas de SQL

Puedo seleccionar columnas, enumerando los nombres a continuación del **SELECT**.

Los **operadores lógicos** son muy importantes para definir condiciones de filtro.

SELECT WHERE

Condiciones Complejas usando conectores lógicos:

En caso de querer aplicar más de una condición dentro de la cláusula **WHERE**, se utilizaran los conectores lógicos **AND** y **OR**

```
SELECT *  
FROM products  
WHERE id < 8  
and list_price > 10;
```

```
SELECT *  
FROM products  
WHERE list_price < 10  
or list_price > 50;
```

Si la cláusula **WHERE** se extiende demasiado y se vuelve realmente compleja incluyendo múltiples operadores **OR** y **AND** es conveniente utilizar paréntesis para agrupar operaciones e indicar precedencia (como en la calculadora)

Select: Ejemplo #5

La siguiente consulta incluye una cláusula **WHERE** con tres grupos de condiciones unidas por operadores **OR**.

```
SELECT id, product_code, product_name, category, list_price
FROM products
WHERE (list_price > 10 AND category Like 'B%') OR
(list_price < 5 AND category Like '_e%') OR
(list_price BETWEEN 2 AND 20 AND category = 'Grains');
```

Puedo seleccionar columnas, enumerando los nombres a continuación del **SELECT**.

Los **operadores y conectores lógicos** son muy importantes para definir condiciones de filtro. **Between**, **Like** (con **%** como comodín) son muy utilizados.

Conectores Lógicos

Name	Description
<u>AND</u> , &&	Logical AND
<u>NOT</u> , !	Negates value
<u>OR</u> ,	Logical OR
<u>XOR</u>	Logical XOR

SELECT
ORDER BY

ORDER BY

Ordenar Resultados

En caso de querer ordenar los resultados de una consulta se utilizará el operador **ORDER BY**.
Si se quiere ordenar en manera Ascendente, se utiliza la opción ASC
Si se quiere ordenar en manera Descendente se utiliza la opción DESC

```
SELECT id, product_code,  
product_name, category,  
list_price  
FROM products  
ORDER BY product_name ASC;
```

```
SELECT id, product_code,  
product_name, category,  
list_price  
FROM products  
ORDER BY category ASC;
```

```
SELECT id, product_code,  
product_name, category,  
list_price  
FROM products  
ORDER BY discontinued ASC;
```

ORDER BY es un operador opcional. **ASC**, es la forma por defecto. Puedo ordenar por cualquier columna, aunque no esté en la lista de columnas del **SELECT**.

Ejemplo SELECT ORDER BY #7

```
SELECT id, product_code, product_name, category, list_price  
FROM products  
ORDER BY category ASC, list_price DESC, product_name ASC;
```

```
SELECT id, product_code, product_name, category, list_price  
FROM products  
ORDER BY 4 ASC, 5 DESC, 3 ASC;
```

Se puede ordenar por múltiples columnas

En manera Ascendente o Descendente, se puede indicar a continuación.
Se puede indicar el número de orden de la columna en el lugar de su nombre.

SELECT
Limit

LIMIT

Limitar la cantidad de filas en los Resultados de la consulta

Con **LIMIT** puedo limitar la cantidad de filas que me muestre como resultado de la consulta. Funciona como un filtro adicional que se aplica a los resultados, pero siempre después de todos los filtros que se aplican en la consulta (**WHERE** y otros que veremos más adelante).

```
SELECT id, product_code,  
product_name, category,  
list_price  
FROM products  
LIMIT 3;
```

```
SELECT id, product_code,  
product_name, category,  
list_price  
FROM products  
ORDER BY id DESC  
LIMIT 3;
```

```
SELECT id, product_code,  
product_name, category,  
list_price  
FROM products  
ORDER BY id DESC  
LIMIT 5
```

LIMIT es un operador opcional. En general se utiliza combinado con **ORDER BY**, por ejemplo para hacer consultas de por ejemplo “los 5 mejoras productos” o los “10 peores clientes”.

Ejercicio Individual

Ejercicio 1: Obtener un listado de todos los clientes que son "Owner".

Ejercicio 2: Obtener un listado de todos los clientes que pertenecen a la compañía "A".

Ejercicio 3: Obtener un listado con los datos: City, first_name, last_name, job_title de todos los clientes que sean de la ciudad "Boston".

Ejercicio 4: Obtener un listado de todas las órdenes con forma de pago de tarjeta de crédito o efectivo.

Ejercicio 5: ¿Cuáles son los distintos métodos de pago?

Ejercicio 6: ¿Existen órdenes que la ciudad de envío sea las vegas y el tipo de pago sea Tarjeta de Crédito?

Ejercicio 7: ¿Cuáles son los empleados con puesto "Sales Representative".

Ejercicio 8: ¿Cuál es el id de empleado de "Sergienko"?

Ejercicio 9: ¿Existen órdenes con employee_id del empleado del ejercicio anterior?

Funciones de Agregación

Agregar o agrupar valores de un conjunto de filas en un total o en subconjuntos agrupados por otras columnas

Las **funciones de agregación** operan en conjuntos de valores que corresponde a una o varias filas.

Se utilizan para realizar cálculos u operaciones sobre ese conjunto de valores almacenados en varias filas.

A menos que se indique lo contrario, las funciones de agregación ignoran los valores **NULL**.

Funciones de Agregación

A menudo estas funciones se utilizan con la cláusula **GROUP BY**, para agrupar valores en subconjuntos o “grupos” en base a otras columnas.

El uso de una función de agregación en una consulta, que no contiene ninguna cláusula **GROUP BY**, es equivalente a agrupar en todas las filas. Para obtener más información, consulte el [Manual de REF. MySQL - Sección 12.20.3, “Manejo MySQL de GROUP BY”](#).

Funciones de Agregación

Funciones habituales

Nombre	Descripción
<u>AVG()</u>	Devuelve el valor promedio del argumento
<u>COUNT()</u>	Devuelve un recuento del número de filas devueltas
<u>COUNT(DISTINCT)</u>	Devuelve el recuento de varios valores diferentes
<u>GROUP_CONCAT()</u>	Devuelve una cadena concatenada
<u>MAX()</u>	Devuelve el valor máximo
<u>MIN()</u>	Devuelve el valor mínimo
<u>SUM()</u>	Devuelve la suma

Funciones de Agregación

Funciones Estadísticas

Nombre	Descripción
<u>STD()</u>	Devuelve la desviación estándar de la población
<u>STDDEV()</u>	Devuelve la desviación estándar de la población
<u>STDDEV_POP()</u>	Devuelve la desviación estándar de la población
<u>STDDEV_SAMP()</u>	Devuelve la desviación estándar de la muestra
<u>VAR_POP()</u>	Devuelve la varianza estándar de la población
<u>VAR_SAMP()</u>	Devuelve la varianza de la muestra
<u>VARIANCE()</u>	Devuelve la varianza estándar de la población

Funciones de Agregación

Función **COUNT()** devuelve la cantidad de filas, resultantes de la consultas.
Combinado con **Distinct** la cantidad de filas distintas

```
SELECT COUNT(*)  
FROM products;
```

```
SELECT  
COUNT(Distinct category)  
FROM products;
```

```
SELECT COUNT(*)  
FROM customers  
WHERE city = 'Las Vegas';
```

```
SELECT  
COUNT(Distinct city)  
FROM customers;
```

Funciones de Agregación

Funciones con Operaciones Numéricas.

```
SELECT SUM(quantity)
FROM order_details;
```

```
SELECT AVG(unit_price)
FROM order_details;
```

```
SELECT
SUM(quantity * unit_price)
FROM order_details;
```

```
SELECT SUM(quantity*unit_price)
/ SUM(quantity)
FROM order_details;
```

Funciones de Agregación

Funciones MIN y MAX

```
SELECT MIN(unit_price)  
FROM order_details;
```

```
SELECT MAX(unit_price)  
FROM order_details;
```

```
SELECT MAX(unit_price) -  
MIN(unit_price)  
FROM order_details;
```

```
SELECT SUM(quantity*unit_price)  
/ SUM(quantity)  
FROM order_details;
```


GROUP BY

El **GROUP BY** agrupa el resultado de la consulta en diferentes subgrupos y a cada subgrupo, se le aplica una o varias funciones de agregación que indicamos en el **SELECT**. Las columnas indicadas en el **GROUP BY** se denominan columnas de agrupación.

Sintaxis:

```
SELECT columna1, columna2,  
FuncionAGG(columna3)  
FROM NombreTabla  
GROUP BY columna1, columna2;
```


GROUP BY - Ejemplo # 9

```
SELECT city, COUNT(*)  
FROM customers  
GROUP BY city
```

```
SELECT category, AVG(list_price)  
FROM products  
GROUP BY category
```

```
SELECT product_id,  
SUM(quantity)  
FROM order_details  
GROUP BY product_id
```

```
SELECT city, COUNT(*)  
FROM customers  
WHERE country_region = 'USA'  
GROUP BY city
```

GROUP BY

¿Cómo funciona el GROUP BY?

Si estamos agrupando por DNI, se crean grupos diferentes por cada DNI que exista en la tabla. En este ejemplo, existen dos grupos.

SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
2	35.186.928	02/01/2017	60
3	33.241.677	03/01/2017	70
4	35.186.928	04/01/2017	40

GROUP BY

Agrupación

SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
2	35.186.928	02/01/2017	60
3	33.241.677	03/01/2017	70
4	35.186.928	04/01/2017	40

SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
3	33.241.677	03/01/2017	70

SKU	DNI	Fecha	Precio
2	35.186.928	02/01/2017	60
4	35.186.928	04/01/2017	40

GROUP BY

Aplicación de la función de agregación

Sobre cada **grupo**, se aplica la función de agregación que se indicó en el SELECT.
En este caso, se aplica la función suma sobre la columna precio

SUM

SKU	DNI	Fecha	Precio
2	35.186.928	02/01/2017	60
4	35.186.928	04/01/2017	40

SUM

SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
3	33.241.677	03/01/2017	70

GROUP BY

Resultado

El resultado de la consulta, es una tabla que contiene el resultado de cada grupo.

DNI	SUM(PRECIO)
35.186.928	100
33.241.677	120

HAVING

HAVING

Agrupación de registros con condiciones

HAVING permite eliminar los grupos que no cumplan con la condición indicada. La condición será una función de agregación, ya que se aplica al grupo entero. Es similar al **WHERE**, pero se aplica al grupo entero y no a cada fila.

```
SELECT Funcion(ColumnaX), ColumnaY  
FROM NombreDeLaTabla  
GROUP BY ColumnaY  
HAVING condicion
```

HAVING

Agrupación de registros

Ejemplo:

```
SELECT DNI, SUM(Precio)
FROM Compras
GROUP BY DNI
HAVING SUM(Precio) > 110
```

Compras
SKU (PK) (FK)
DNI (FK)
Fecha
Precio

HAVING

¿Cómo funciona el **GROUP BY** con **HAVING**?

Como estamos agrupando por DNI, se crean grupos diferentes por cada DNI que exista en la tabla. En este ejemplo, existen dos grupos.

SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
2	35.186.928	02/01/2017	60
3	33.241.677	03/01/2017	70
4	35.186.928	04/01/2017	40

HAVING

Agrupación

↓	SKU	DNI	Fecha	Precio	↓
	1	33.241.677	01/01/2017	50	
	2	35.186.928	02/01/2017	60	
	3	33.241.677	03/01/2017	70	
	4	35.186.928	04/01/2017	40	

SKU	DNI	Fecha	Precio
2	35.186.928	02/01/2017	60
4	35.186.928	04/01/2017	40

SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
3	33.241.677	03/01/2017	70

HAVING

Aplicación de la función de Agregación

Sobre cada **grupo**, se aplica la función de agregación que se indicó en el SELECT.
En este caso, se aplica la función suma sobre la columna precio.

SUM			
SKU	DNI	Fecha	Precio
2	35.186.928	02/01/2017	60
4	35.186.928	04/01/2017	40

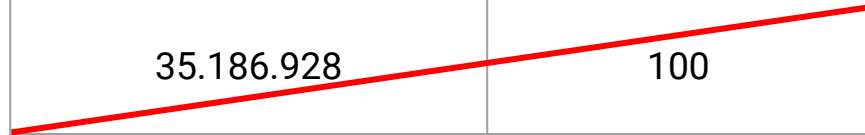
SUM			
SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
3	33.241.677	03/01/2017	70

HAVING

Aplicación de la condición del HAVING

Se eliminan los grupos que no cumplan la condición indicada. En este ejemplo, se eliminan todos los grupos cuya suma es menor a 110.

DNI	SUM(PRECIO)
35.186.928	100
33.241.677	120



HAVING

Resultado

El resultado de la consulta, es una tabla que contiene el resultado de cada grupo que cumple la condición.

DNI	SUM(PRECIO)
33.241.677	120

HAVING

Agrupación de registros

Ejemplo Avanzado:

```
SELECT DNI, SUM(Precio)
FROM Compras
WHERE Fecha > "02/01/2017"
GROUP BY DNI
HAVING SUM(Precio) >= 70
```

Compras
SKU (PK) (FK)
DNI (FK)
Fecha
Precio

HAVING

¿Cómo funciona el **GROUP BY** con **HAVING** y con **WHERE**?

Lo primero que se ejecuta en esta consulta es la cláusula **WHERE**. Se eliminan todas las filas que no cumplan la condición.

SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
2	35.186.928	02/01/2017	60
3	33.241.677	03/01/2017	70
4	35.186.928	04/01/2017	40

HAVING

Agrupación

Luego, se continúa con la agrupación. Como estamos agrupando por DNI, se crean grupos diferentes por cada DNI que exista en la tabla. En este ejemplo, existen dos grupos.

SKU	DNI	Fecha	Precio
3	33.241.677	03/01/2017	70
4	35.186.928	04/01/2017	40

SKU	DNI	Fecha	Precio
4	35.186.928	04/01/2017	40

SKU	DNI	Fecha	Precio
3	33.241.677	03/01/2017	70

HAVING

Aplicación de la función de Agregación

Sobre cada **grupo**, se aplica la función de agregación que se indicó en el SELECT.
En este caso, se aplica la función suma sobre la columna precio

SUM			
SKU	DNI	Fecha	Precio
4	35.186.928	04/01/2017	40

SUM			
SKU	DNI	Fecha	Precio
3	33.241.677	03/01/2017	70

HAVING

Aplicación de la condición del HAVING

Se eliminan los grupos que no cumplan la condición indicada. En este ejemplo, se eliminan todos los grupos cuya suma es menor a 70.

DNI	SUM(PRECIO)
35.186.928	40
33.241.677	70

HAVING

Resultado

El resultado de la consulta, es una tabla que contiene el resultado de cada grupo que cumple la condición.

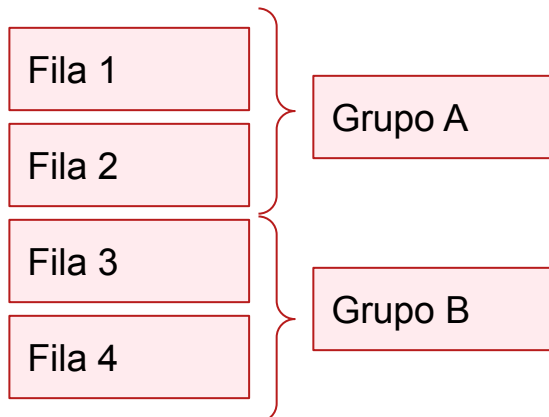
DNI	SUM(PRECIO)
33.241.677	70

HAVING vs WHERE

WHERE permite filtrar a nivel fila.

HAVING permite filtrar a nivel GRUPO (según lo definido en el GROUP BY)

1. Primero se aplican las condiciones definidas en **WHERE** para filtrar a nivel fila.



2. Luego se agrupan las filas por la condición del **GROUP BY**

3. Por último se aplican las condiciones definidas en **HAVING** para filtrar a nivel grupo.



Ejercicio Individual

Ejercicio 1: ¿Cuántos clientes existen?

Ejercicio 2: ¿Cuántos clientes hay por ciudad?

Ejercicio 3: ¿Cuántas órdenes existen por método de pago? ¿Cuál es el más elegido?

Ejercicio 4: ¿Cuántas órdenes existen por método de pago y con status igual a 3?

Ejercicio 5: Tomando la tabla order_detail, ¿cual es la cantidad promedio de productos vendidos?

Ejercicio 6: ¿Cuál es la máxima cantidad vendida?

Ejercicio 7: ¿Cuál es el product_id más pedido?

Ejercicio 8: ¿A qué producto corresponde ese id?

Ejercicio 9: ¿Cuánto es la facturación total? Considerando que la facturación es igual al precio por cantidad.

Ejercicio 10: Tomando la tabla orders, ¿qué id de empleado tuvo más órdenes? ¿A qué empleado corresponde ese id?