Base de datos

SQL

IPEAyT 186 - Capitán Luis Dario J. Castagnari (Las Higueras)

Profesor: Julián Conde

Lenguaje de consulta SQL

Structured Query Language
Lenguaje de Consulta Estructurado

SQL brinda diferentes sentencias para definir datos, realizar consultas y actualizaciones.

Lenguaje de definición de datos: DDL (Data Definition Language). **Permiten crear y definir nuevas bases de datos, campos e índices**

Lenguaje de manipulación de datos: DML (Data Manipulation Language). **Permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos, además incluye de sentencias para agregar modificar y eliminar datos**

Lenguaje de Control de Datos: DCL (Data Control Language): **permite manipular los permisos sobre las bases de datos y sus objetos**

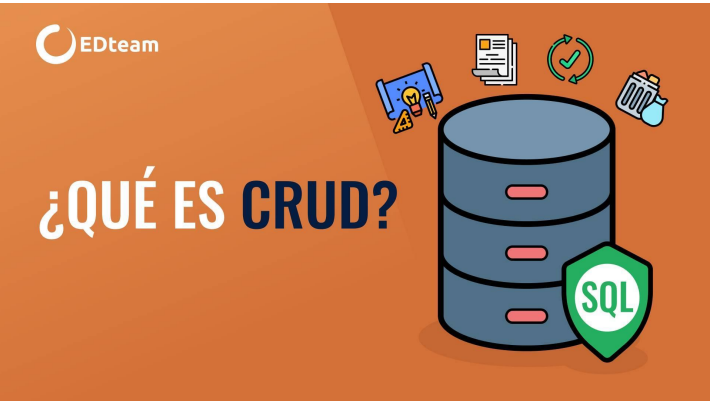
Tutorial útil:
www.w3schools.com/sql/default.asp

Lenguaje de consulta SQL

DML

- ✓ **SELECT**: Consulta registros de la base de datos que satisfagan un criterio determinado.
- ✓ **INSERT**: Carga lotes de datos en la base de datos en una única operación.
- ✓ **UPDATE**: Modifica los valores de los campos y registros especificados.
- ✓ **DELETE**: Elimina registros de una tabla de una base de datos.

SQL


EDteam
¿QUÉ ES CRUD?
SQL

¿QUÉ ES CRUD?
EDteam
Hace referencia a un acrónimo, en él unen las primeras letras de las cuatro operaciones principales de aplicaciones en sistemas de bases de datos:
SQL
CRUD
CREATE
Crea Registros
READ
Lee Registros
UPDATE
Actualiza Registros
DELETE
Borra Registros
Gracias a los CRUDs podrás usar estas operaciones para manejar los datos de tus aplicaciones.
Prof. Pablo España
Ahora sabes que cuando te digan "haremos el CRUD de esta entidad", significa las operaciones que aquí te explicamos.

Lenguaje de consulta SQL

SQL utiliza los términos tabla, fila y columna para los términos relación, tupla y atributo del modelo relacional formal, respectivamente.

SQL es un lenguaje de bases de datos global: cuenta con sentencias para definir datos, consultas y actualizaciones.

A tener en cuenta en su sintaxis que se utiliza para mostrar sus distintas sentencias:

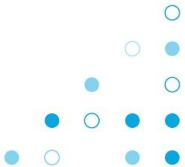
{**Alternativas**}, entre llaves se colocarán las palabras que tienen opciones o alternativas en la sentencia a la que pertenecen.

[**Opcional**], entre corchetes se colocarán las palabras que son opcionales en la sentencia, es decir que pueden colocarse o pueden obviarse.

Consultas SQL: SELECT

El comando más usado en SQL es la instrucción SELECT, se utiliza para recuperar datos de diferentes tablas de una base de datos.
Consta de seis cláusulas: las dos primeras (SELECT y FROM) obligatorias y las cuatro restantes opcionales.

SELECT <lista de atributos>
FROM <lista de tablas>
WHERE <condición>;



Consultas SQL: SELECT

SELECT [ALL|DISTINCT] { * | expr_columna_1 [AS c_alias_1]}

FROM nombre_tabla_1 [t_alias_1]

[**WHERE** condicionWhere]

[**GROUP BY** expr_columna_group1 [,expr_columna_group2] [,...]]

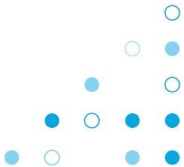
[**HAVING** condicionHaving]]

[**ORDER BY** nombre_campo_i1 [ASC|DESC] [, nombre_campo_j1 [ASC|DESC]][,...]]];

Consultas SQL: FROM

Cláusula FROM

La cláusula FROM lista las tablas que contienen los datos a recuperar por la consulta. La sintaxis de esta cláusula es:
FROM nombre_tabla_1 [t_alias_1][, nombre_tabla _2 [t_alias_2][,...]]



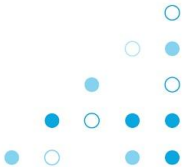
Consultas SQL: WHERE

Cláusula WHERE

En cláusula WHERE se define la condición que deben cumplir las filas o registros de datos que estarán en el resultado de la consulta.

Su sintaxis es: WHERE **condicionWhere**

CondicionWhere: Expresión que determina qué registros se incluirán en el resultado.



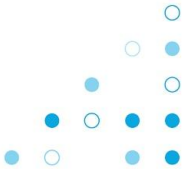
Consultas SQL: WHERE

CondicionWhere: Expresión que determina qué registros se incluirán en el resultado.

En una expresión se pueden comparar con operadores de comparación, columnas con otras columnas o con expresiones del tipo de la columna, las expresiones se pueden conectar con otras por medio de operadores lógicos.

<u>Operadores lógicos</u>: AND, OR, NOT, XOR

<u>Operadores de comparación</u>: < , >, <>, <=, >=, =



De pregunta a Query

Lo que quieres mostrar = SELECT

De donde voy a tomar los datos = FROM

Los filtros de los datos que quieres mostrar = WHERE

Los rubros por los que me interesa agrupar la información = GROUP BY

El orden en que quiero presentar mi información ORDER BY

Los filtros que quiero que mis datos agrupados tengan HAVING

SINTAXIS BÁSICA DE SQL

SQL es un lenguaje de consulta estructurada que permite **administrar y recuperar información** en la base de datos.

Indica las columnas que se quieren consultar.

Selecciona **las tablas.**

Define las condiciones para que se cumpla la consulta.

```
SELECT nombre, autor, genero
FROM libros
WHERE genero = 'Novela negra';
```

Sintaxis completa del SELECT

Muestra **cómo se agruparán los registros.** Se usa **Group by** cuando existen funciones como **COUNT().**

Indica **qué condición debe cumplirse como resultado.**
COUNT() cuenta los libros y la regla determina que sean mayores a 3.

Muestra **el orden de la consulta.**

```
SELECT autor, COUNT(libroID)
FROM libros
WHERE genero = 'Novela negra'
GROUP BY autor
HAVING COUNT(libroID) > 3
ORDER BY nombre ASC;
```

Ejemplos

De ahora en más los ejemplos de consultas SQL estarán hechos sobre el siguiente esquema de base de datos.

Articulos(Nart, desc, precio, cant, stock_min, stock_max)

Provee(Nprov,Nart, precio_venta)

Proveedores(Nprov, nombre, direccion)

Clientes(Ncli, nombre, direccion)

Compran(Ncli, Nart)

Con las siguientes instancias (datos):

Tablas con registros para los ejemplos

ARTICULOS					
Nart	descr	precio	cant	stock-min	stock-max
100	Café	5,2	8	10	50
101	Azucar	1,2	30	15	45
102	Harina	1,1	35	16	40

CLIENTES		
Ncli	nombre	dir
300	Pedro	Bs As 90
301	Juan	Jujuy 800
302	Maria	Cordoba 24

PROVEEDORES		
Nprov	nombre	dir
200	Grasano	Bs As 100
201	Atomo	Lima 85
202	VEA	Alvear 455

PROVEE		
Nprov	**Nart**	precio_Venta
200	100	4
200	101	1,1
201	101	1
201	102	0,5
202	102	1

COMPRAN	
Ncli	**Nart**
300	100
300	102
301	101
302	101

Ejemplos

1) Devolver un listado de artículos con el número, descripción y precio de cada artículo.

```
SELECT Nart, descr, precio
FROM articulos;
```

Nart	descr	precio
100	Café	5.2
101	Azucar	1.2
102	Harina	1.1

Ejemplos (sigue)

2)Devolver un listado de artículos con el número, descripción, precio de cada artículo y la ganancia (es el 20% del precio del articulo) .

```
SELECT Nart, Descr, precio, precio*0.2 AS ganancia
FROM articulos;
```

Nart	descr	precio	ganacia
100	Café	5.2	1.04
101	Azucar	1.2	0.24
102	Harina	1.1	0.22

Ejemplos (sigue)

3) `SELECT Nart FROM provee`

Nart
100
101
101
102
102

4) `SELECT DISTINCT Nart FROM provee`

Nart
100
101
102

Ejemplos (sigue)

5) Devolver un listado con donde se combinen todos los proveedores con todos los precios de ventas de los artículos.

```
SELECT p.Nprov, Nart, precio_venta, pro.Nprov, nombre, dir FROM provee p, proveedores pro
```

provee.Nprov	Nart	precio_Venta	proveedores.Nprov	nombre	dir
200	100	4	200	Grasano	Bs As 100
200	100	4	201	Atomo	Lima 85
200	100	4	202	VEA	Alvear 455
200	101	1,1	200	Grasano	Bs As 100
200	101	1,1	201	Atomo	Lima 85
200	101	1,1	202	VEA	Alvear 455
201	101	1	200	Grasano	Bs As 100
201	101	1	201	Atomo	Lima 85
201	101	1	202	VEA	Alvear 455
201	102	0,5	200	Grasano	Bs As 100
201	102	0,5	201	Atomo	Lima 85
201	102	0,5	202	VEA	Alvear 455
202	102	1	200	Grasano	Bs As 100
202	102	1	201	Atomo	Lima 85
202	102	1	202	VEA	Alvear 455

Ejemplo Cláusula WHERE

1) Devolver un listado con todos los campos de los artículos a reponer.

```
SELECT * FROM articulos WHERE stock_min >cant
```

Nart	descr	precio	cant	stock-min	stock-max
100	Café	5.2	8	10	50

Ejemplos (sigue)

2) Dar un listado con los proveedores y los artículos que provee con su precio de venta

```
SELECT Nart, precio_venta,p.Nprov, nombre, dir
FROM provee p, proveedores pro WHERE p.Nprov = pro.Nprov
```

Nart	precio_Venta	Nprov	nombre	dir
100	4	200	Grasano	Bs As 100
101	1.1	200	Grasano	Bs As 100
101	1	201	Atomo	Lima 85
102	0.5	201	Atomo	Lima 85
102	1	202	VEA	Alvear 455

Expresiones

En una expresión se pueden cotejar con operadores de comparación, columnas con otras columnas o con expresiones del tipo de la columna, las expresiones se pueden conectar con otras por medio de operadores lógicos. Se pueden definir expresiones aritméticas, de cadenas, entre otras, dentro de las expresiones.

Operadores

Lógicos

- AND
- OR
- NOT
- XOR

Comparación

- < , >, <>, <=, >=, =
- **BETWEEN**: Utilizado para especificar un intervalo de valores. Utilizado en tipos String, numéricos y de fechas.
- **LIKE**: Utilizado en la comparación de un patrón. El carácter '%' se utiliza como comodín
- **IN**: Utilizado para saber si el valor de un campo se encuentra en una subconsulta de selección de una columna.
- **EXIST**: Devuelve verdadero si una subconsulta es vacía, si no devuelve falso.

Ejemplos

a) Listados de artículos cuya descripción empieza con la letra A

```
SELECT * FROM articulos WHERE desc like "A%"
```

b) Listados de artículos cuya descripción empieza con la letra A y termina con la letra K

```
SELECT * FROM articulos WHERE desc like "A%" AND desc like “%K"
```

c) Devolver los artículos que son provistos por algún proveedor.

```
SELECT * FROM articulos WHERE Nart IN (SELECT Nart FROM provee);
```

d) Devolver los artículos que vendiendo 2 unidades más estarían bajos en Stock

```
SELECT * FROM articulos WHERE stock_min >cant-2
```

Ejemplo

Devolver el nombre y dirección de todas las personas o empresas del negocio (clientes más proveedores)

SELECT nombre, dir **FROM** clientes

UNION

SELECT nombre, dir **FROM** proveedores

nombre	**dir**
Pedro	Bs As 90
Juan	Jujuy 800
Maria	Cordoba 24
Grasano	Bs As 100
Atomo	Lima 85
VEA	Alvear 455

Cláusula ORDER BY

La cláusula ORDER BY ordena los resultados de la consulta en base a los datos de una o más columnas. Si se omite, los resultados serán ordenados por el primer campo que sea clave en el índice que se haya utilizado.

Su sintaxis es:

ORDER BY *expresión_orden1* [ASC|DESC]

[, *expresión_orden2* [ASC|DESC]][,...];

Argumento	**Descripción**
Expresión_ordenN	Puede ser el nombre de un campo, expresión o el número de posición que ocupa la expresión de columna en la cláusula SELECT.
ASC	Produce el ordenamiento ascendente de los valores de la columna. Esta opción es la elegida por defecto.
DESC	Produce el ordenamiento descendente de los valores de la columna.

Ejemplo de Consulta ordenada

Nota: se puede observar la utilización de alias tanto en el nombre de tablas como en el de columnas.

1) Dar un listado con los artículos, que proveedores lo suministran y su respectivo precio de venta. El listado debe estar ordenado ascendentemente por el número de artículo.

```
SELECT DISTINCT Nart AS Numero_Art, Pv.Nprov AS Numero_prov, Nombre,Precio_Venta
FROM Proveedores Pd, Provee Pv
WHERE Pd.Nprov =Pv.Nprov
ORDER BY Nart
```

numero_Art	numero_prov	nombre	precio_venta
100	200	Grasano	4
101	200	Grasano	1,1
101	201	Atomo	1
102	201	Atomo	0,5
102	202	VEA	1

LIMIT y OFFSET en MySQL

En MySQL, las cláusulas **LIMIT** y **OFFSET** se utilizan para **restringir la cantidad de resultados** devueltos por una consulta y **saltar un número específico de filas**.

```
SELECT columnas
FROM tabla
LIMIT cantidad OFFSET inicio;
```

Devuelve solo las **primeras 5 filas** de la tabla Customers

```
SELECT * FROM Customers LIMIT 5;
```

Omite los **primeros 5 registros** y muestra los **siguientes 3**.

```
SELECT CustomerName, City
FROM Customers
LIMIT 3 OFFSET 5;
```

Esta técnica es muy usada en **paginación de resultados** (por ejemplo, mostrar resultados del 6 al 10 en una web).

Funciones Agregadas

SQL proporciona operadores agregados que toman el nombre de una columna como argumento. El valor del operador agregado se calcula sobre todos los valores de la columna especificada en la tabla completa. Estas funciones se pueden combinar con la cláusula **GROUP BY** que se verá más adelante.

Si no se quieren eliminar los repetidos para los cálculos utilizar la cláusula distinct.

Las funciones agregadas son:

- AVG (Calcula el promedio)
- COUNT (Cuenta la cantidad de registros)
- SUM (Suma los valores de una columna)
- MIN (Calcula el mínimo valor de una columna)
- MAX (Calcula el máximo valor de una columna)

Funciones Agregadas: AVG (promedio)

Devuelve el promedio de los valores de una expresión de columna.

Ejemplo:

SELECT AVG(Precio) as promedio **FROM** articulos

Devolverá el promedio de precios de todos los artículos.

promedio
2,5

Funciones Agregadas: SUM (suma)

Devuelve la suma total de los valores de una expresión de columna o campo numérico.

Ejemplo:

SELECT SUM(cant) as cantidad **FROM** articulos

Devolverá la cantidad total de artículos.

cantidad
73

Funciones Agregadas: COUNT (contador)

Simplemente cuenta el número de registros. No importan los valores que están almacenados en los registros.

Ejemplo 1:

```
SELECT COUNT(Nart) as cant_articulos FROM Provee
```

cant_articulos
5

Devolverá la cantidad de artículos diferentes.

Ejemplo 2, si no se quieren contar repetidos:

```
SELECT COUNT( distinct Nart) as cant_articulos FROM Provee
```

Devolverá la cantidad de artículos diferentes.

cant_articulos
3

Funciones Agregadas: MAX (máximo)

Devuelve el valor más alto de los contenidos en una expresión de columna.

Ejemplo:

```
SELECT MAX(Precio) as precio_maximo FROM articulos
```

Devolverá el precio del artículo más caro.

precio_maximo
5,2

Funciones Agregadas: MIN (mínimo)

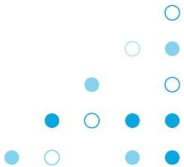
Devuelve el valor más bajo de los contenidos en una expresión de columna.

Ejemplo:

```
SELECT MIN(Precio) as precio_minimo FROM articulos
```

Devolverá el precio del artículo más barato.

precio_minimo
1,1



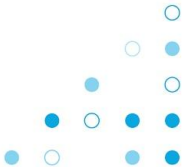
Cláusula GROUP BY

La cláusula GROUP BY especifica una **consulta sumaria**. En vez de producir una fila de resultados por cada fila de datos de la base de datos, una consulta sumaria agrupa todas las filas similares y luego produce una fila sumaria de resultados para cada grupo.

[GROUP BY *expr_columna_group1* [,*expr_columna_group2*] [,...]]

Donde:

expr_columna_groupN deben coincidir con las expresión de columna utilizada en la cláusula SELECT.



Ejemplo de Group by

1) Retornar un listado con los números de artículos y el precio promedio de venta de los proveedores para cada artículo.

```
SELECT Nart, AVG(precio_venta) as promedio_p_v
FROM Provee
GROUP BY Nart;
```

Nart	promedio_p_v
100	4
101	1.05
102	0.75

Cláusula Having

La cláusula HAVING trabaja muy similarmente a la cláusula WHERE, y se utiliza para considerar sólo aquellos grupos que satisfagan la condición dada en la cláusula HAVING.

Su sintaxis es: **HAVING *condicionHaving***

Donde *condicionHaving* es la condición que deben cumplir los grupos. Las expresiones permitidas en la cláusula HAVING deben involucrar funciones agregadas. Cada expresión que utilice sólo atributos planos deberá ubicarse en la cláusula WHERE. Por otro lado, toda expresión que involucre funciones agregadas debe aparecer en la cláusula HAVING.

Ejemplo

Retornar un listado con los números de artículos y el precio promedio de venta de los artículos cuyo precio promedio de venta de los proveedores es menor a $2.

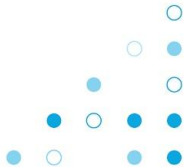
```
SELECT Nart, AVG(precio_venta)
FROM Provee
GROUP BY Nart
Having AVG(precio_venta) < 2;
```

Nart	avg
101	1.05
102	0.75

Consulta SQL: Subconsultas

Una subconsulta es una **consulta anidada** en una instrucción SELECT, INSERT, UPDATE o DELETE, o bien en otra subconsulta.

```
SELECT * FROM clientes
WHERE id IN
   (SELECT cliente_id FROM facturas
    WHERE fecha BETWEEN 2016-10-10 AND 2017-01-01);
```



Consulta SQL: Subconsultas

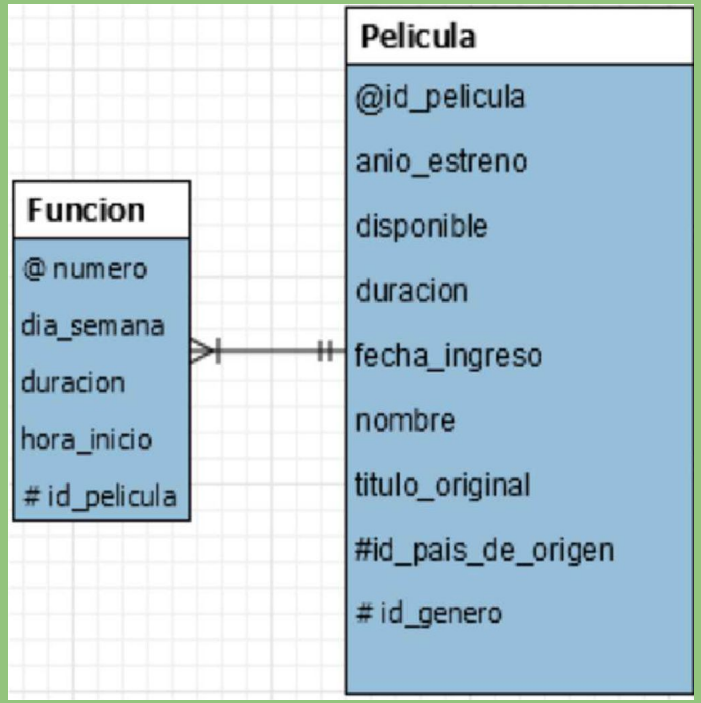
Operadores para Subconsultas

```
SELECT * FROM tabla WHERE columna ANY ( [ SUBCONSULTA ] )

SELECT * FROM tabla WHERE columna ALL ( [ SUBCONSULTA ] )

SELECT * FROM tabla WHERE columna IN ( [ SUBCONSULTA ] )

SELECT * FROM tabla WHERE EXISTS ( [ SUBCONSULTA ] )
```

JOINS

Consulta multi-tablas.

```
SELECT Película.nombre, Función.dia_semana,Función.hora_inicio
FROM Pelicula INNER JOIN Funcion
                    ON Funcion.id_pelicula = Pelicula.id_pelicula
```



Consultas SQL: Multi-tablas (Joins)

Abreviación del nombre de la tabla.

```
SELECT pel.nombre, fun.hora_inicio, fun.dia_semana AS dia,
FROM Pelicula pel INNER JOIN Funcion fun
ON fun.id_pelicula = pel.id_pelicula
```

Renombrado de columna

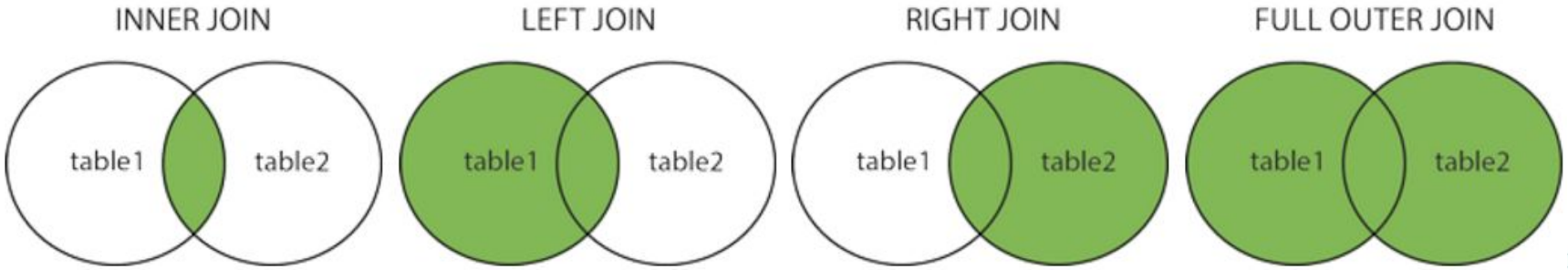
LEFT JOIN != RIGHT JOIN != INNER JOIN

Consultas SQL: Multi-tablas (Joins)

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table



Tablas para Ejemplos

Consideremos las siguientes relaciones:

ingenieros

#Emp	Nombre	#Dep
201	Juan	1001
202	Manuel	1002
203	Carlos	1001
204	Ariel	1004

departamentos

#Dep	Descripcion
1001	Electricidad
1002	Mecanica
1003	Mantenimiento

Tipo de JOIN, INNER

INNER: Devuelve sólo las filas de las tablas que están relacionadas entre si.

Ejemplo: Listar los ingenieros con sus departamentos (nombre incluido), sólo deberá mostrar los ingenieros que tengan departamento y sólo los departamentos que tengan algún Ingeniero.

SELECT * **FROM** ingenieros **NATURAL INNER JOIN** departamentos

#Emp	Nombre	#Dep	Descripcion
201	Juan	1001	Electricidad
202	Manuel	1002	Mecanica
203	Carlos	1001	Electricidad

Ejemplo (sigue)

La consulta anterior también se puede escribir de las siguientes forma:

SELECT *
FROM ingenieros i, departamentos d
WHERE (i.#dep = d.#dep)

SELECT * **FROM** ingenieros i **NATURAL INNER JOIN** departamentos d

SELECT * **FROM** ingenieros i **INNER JOIN** departamentos d **ON** (i.#dep = d.#dep)

SELECT * **FROM** ingenieros i **INNER JOIN** departamentos d **USING** (#dep)

LEFT OUTER

Devuelve todas las filas de la relación de la izquierda aunque no estén relacionadas con una fila de la tabla de la derecha. Las filas del JOIN que no están relacionadas, los campos de la tabla de la derecha se completa con NULL.

Ejemplo: Devolver todos los ingenieros con sus departamentos, si hay ingenieros cuyo departamento no exista, los debe listar también.

SELECT * **FROM** ingenieros **NATURAL LEFT OUTER JOIN** departamentos

#Emp	Nombre	#Dep	Descripcion
201	Juan	1001	Electricidad
202	Manuel	1002	Mecanica
203	Carlos	1001	Electricidad
204	Ariel	1004	null

RIGHT OUTER

Devuelve todas las filas de la tabla de la derecha aunque no estén relacionadas con una fila de la tabla de la izquierda. Las filas del JOIN que no están relacionadas, los campos de la tabla de la izquierda se completa con NULL.

Ejemplo: Listar todos los ingenieros con sus departamentos, sólo deberá mostrar los ingenieros que tengan departamento, si los departamentos no tienen ingenieros los deberá mostrar.

SELECT * **FROM** ingenieros **NATURAL RIGHT OUTER JOIN** departamentos

#Emp	Nombre	#Dep	Descripcion
201	Juan	1001	Electricidad
202	Manuel	1002	Mecanica
203	Carlos	1001	Electricidad
null	null	1003	Mantenimiento

FULL OUTER

Es la unión del LEFT y RIGHT OUTER JOIN.

Ejemplo: Listar todos los ingenieros con sus departamentos, si hay ingenieros cuyo departamento no exista, los debe listar también, igualmente si hay departamentos que no tengan ingenieros también deben listarse.

SELECT * **FROM** ingenieros **NATURAL FULL OUTER JOIN** departamentos

#Emp	Nombre	#Dep	Descripcion
201	Juan	1001	Electricidad
202	Manuel	1002	Mecanica
203	Carlos	1001	Electricidad
204	Ariel	1004	null
null	null	1003	Mantenimiento

JOINS. Ejemplo:
base nacidosia y sus tablas Persona y Ciudad

Name: Persona Schema: nacidosia

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G
id	INT	☑	☑	☐	☐	☐	☐	☑	☐
nombre	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	☐
apellido	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	☐
dni	VARCHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐
email	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	☐
fecha_nacimiento	DATE	☐	☐	☐	☐	☐	☐	☐	☐
telefono	VARCHAR(15)	☐	☐	☐	☐	☐	☐	☐	☐
ciudad_id	INT	☐	☐	☐	☐	☐	☐	☐	☐
		☐	☐	☐	☐	☐	☐	☐	☐

Name: Ciudad

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G
id	INT	☑	☑	☐	☐	☐	☐	☑	☐
nombre	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	☐
provincia	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	☐
codigo_postal	VARCHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐
		☐	☐	☐	☐	☐	☐	☐	☐

Name: Persona Schema: nacidosia

Foreign Key Name	Referenced Table	Foreign Key Columns	
		Column	Referenced Column
Persona_ibfk_1	`nacidosia`.`Ciudad`		
		☐ id	
		☐ nombre	
		☐ apellido	
		☐ dni	
		☐ email	
		☐ fecha_nacimiento	
		☐ telefono	
		☑ ciudad_id	id

JOINS. Ejemplo:
base nacidosia y sus tablas Persona y Ciudad

```
1 •  SELECT * FROM nacidosia.Persona;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell

#	id	nombre	apellido	dni	email	fecha_nacimient	telefono	ciudad_id
1	1	Juan	Perez	12345678A	juan@example.com	1990-05-15	555-123-456	1
2	2	Maria	Lopez	98765432B	maria@example.com	1985-09-20	555-789-123	2
3	3	Julian	Conde	30303030	jconde@ispc.com	1983-04-10	22222222	1
4	4	Laura	Trifi	2344567	lau@vedia.com	1982-10-06	3333333	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
1 •  SELECT * FROM nacidosia.Ciudad;
```

Result Grid | Filter Rows: | Edit:

#	id	nombre	provincia	codigo
1	1	Vedia	BsAs	6030
2	2	Rio4	Córdoba	5800
3	3	Las Higueras	Córdoba	5805
*	NULL	NULL	NULL	NULL

Tipo de JOIN, INNER

INNER: Devuelve sólo las filas de las tablas que están relacionadas entre si.

```
1 •  SELECT p.id, p.nombre, p.apellido, c.nombre, c.id
2    FROM nacidosia.Persona as p INNER JOIN nacidosia.Ciudad as c
3    ON c.id = p.ciudad_id;
```

sult Grid Filter Rows: Export: Wrap Cell Content:

id	nombre	apellido	nombre	id
1	Juan	Perez	Vedia	1
2	Maria	Lopez	Rio4	2
3	Julian	Conde	Vedia	1

LEFT OUTER

Devuelve todas las filas de la relación de la izquierda aunque no estén relacionadas con una fila de la tabla de la derecha. Las filas del JOIN que no están relacionadas, los campos de la tabla de la derecha se completa con NULL.

```
1 •  SELECT p.id, p.nombre, p.apellido, c.nombre, c.id
2    FROM nacidosia.Persona as p LEFT OUTER JOIN nacidosia.Ciudad as c
3    ON c.id = p.ciudad_id;
```

Result Grid Filter Rows: Export: Wrap Cell Content:

#	id	nombre	apellido	nombre	id
1	1	Juan	Perez	Vedia	1
2	2	Maria	Lopez	Rio4	2
3	3	Julian	Conde	Vedia	1
4	4	Laura	Trifi	NULL	NULL

RIGHT OUTER

Devuelve todas las filas de la tabla de la derecha aunque no estén relacionadas con una fila de la tabla de la izquierda. Las filas del JOIN que no están relacionadas, los campos de la tabla de la izquierda se completa con NULL.

```
1 •  SELECT p.id, p.nombre, p.apellido, c.nombre, c.id
2    FROM nacidosia.Persona as p RIGHT OUTER JOIN nacidosia.Ciudad as c
3    ON c.id = p.ciudad_id;
```

Result Grid Filter Rows: Export: Wrap Cell Content:

#	id	nombre	apellido	nombre	id
1	1	Juan	Perez	Vedia	1
2	3	Julian	Conde	Vedia	1
3	2	Maria	Lopez	Rio4	2
4	NULL	NULL	NULL	Las Higueras	3

FULL OUTER

Es la unión del LEFT y RIGHT OUTER JOIN.

```sql
SELECT p.id, p.nombre, p.apellido, c.nombre, c.id
FROM nacidosia.Persona as p LEFT OUTER JOIN nacidosia.Ciudad as c ON c.id = p.ciudad_id

UNION -- Como MySQL no implementa FULL JOIN se emula haciendo la union de left y right

SELECT p.id, p.nombre, p.apellido, c.nombre, c.id
FROM nacidosia.Persona as p RIGHT OUTER JOIN nacidosia.Ciudad as c ON c.id = p.ciudad_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	id	nombre	apellido	nombre	id
1	1	Juan	Perez	Vedia	1
2	2	Maria	Lopez	Rio4	2
3	3	Julian	Conde	Vedia	1
4	4	Laura	Trifi	NULL	NULL
5	NULL	NULL	NULL	Las Higueras	3

Tres del CRUD: UPDATE, INSERT y DELETE

INSERTAR DATOS (INSERT)

ACTUALIZAR DATOS (UPDATE)

ELIMINAR DATOS (DELETE)

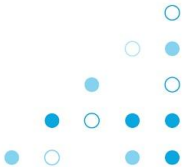
Insertar datos (La C del CRUD)

```
INSERT INTO nombre_tabla (columna1, columna2, columna3,...)
VALUES (valor, valor2, valor3,...);
```

```
INSERT INTO pelicula

    VALUES (2, 2014, true, 122, 01/08/2014, "Relatos Salvajes", "Relatos
Salvajes",1)
```

https://www.w3schools.com/sql/sql_insert.asp

https://www.w3schools.com/sql/sql_insert_into_select.asp



Insertar datos. Múltiples filas (La C del CRUD)

Función a utilizar al volcar datos: INSERT

Sintáxis a utilizar al insertar datos:

- ○ incluir los nombres de las columnas en cada INSERT declaración
  Ejemplo: `INSERT INTO tbl_name (col_A,col_B,col_C,col_C) VALUES (1,2,3)`
- ○ incluir múltiples filas en cada sentencia INSERT
  Ejemplo: `INSERT INTO tbl_name VALUES (1,2,3), (4,5,6), (7,8,9)`
- ◉ ambos de los anteriores
  Ejemplo: `INSERT INTO tbl_name (col_A,col_B,col_C) VALUES (1,2,3), (4,5,6), (7,8,9)`
- ○ ninguno de los anteriores
  Ejemplo: `INSERT INTO nombre_tabla VALUES (1,2,3)`

Insertar datos. Múltiples filas (La C del CRUD)

El comando SQL INSERT también puede utilizarse para insertar múltiples filas en una sola consulta. Por ejemplo:

```
INSERT INTO clientes (nombre, correo)
VALUES ('Cliente1', 'cliente1@email.com'),
       ('Cliente2', 'cliente2@email.com'),
       ('Cliente3', 'cliente3@email.com');
```

Este comando añadirá tres nuevos clientes a la tabla «clientes».

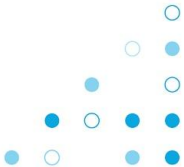
Eliminar datos (La D del CRUD)

```
DELETE FROM nombre_tabla
[WHERE condicion]
[ORDER BY ...]
[LIMIT cantidad_filas]
```

```
DELETE FROM Pelicula WHERE anio_estreno <1990
```

```
DELETE FROM Pelicula
```

https://www.w3schools.com/sql/sql_delete.asp



Actualizar datos (La U del CRUD)

```
UPDATE Pelicula
SET disponible=1, duracion=160
WHERE id_pelicula=4
```

```
UPDATE nombre_tabla
   SET col_nombre_1={valor1|DEFAULT} ,
       col_nombre_2={valor2|DEFAULT}
[WHERE condicion]
[ORDER BY ...]
[LIMIT cantidad_filas]
```

https://www.w3schools.com/sql/sql_update.asp

Actualizar datos (La U del CRUD). MySQL UPDATE

Si prueban actualizar y sale un error, puede ser que en MySQL Workbench tengan que sacarle el modo de seguridad.

UPDATE cliente **SET** NOMBRE = "EZE", DIRECCION = "San Luis 80"
WHERE NOMBRE = "MATIAS"

UPDATE cliente SET NOMBRE = "EZE", DIRECCION = "San Luis 80" WHERE NOMBRE = "MATIAS" Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column To disable safe mode, toggle the option in Preferences -> SQL Editor and reconnect.

SET SQL_SAFE_UPDATES = 0

Ejemplos

De ahora en más los ejemplos de consultas SQL estarán hechos sobre el siguiente esquema de base de datos.

articulos(Nart, desc, precio, cant, stock_min, stock_max)

Provee(Nprov,Nart, precio_venta)

Proveedores(Nprov, nombre, direccion)

Clientes(Ncli, nombre, direccion)

Compran(Ncli, Nart)

Con las siguientes instancias (datos):

Tablas con registros para los ejemplos

articulos

Nart	descr	precio	cant	stock-min	stock-max
100	Café	5.2	8	10	50
101	Azucar	1.2	30	15	45
102	Harina	1.1	35	16	40

clientes

Ncli	nombre	dir
300	Pedro	Bs As 90
301	Juan	Jujuy 800
302	Maria	Cordoba 24

proveedores

Nprov	nombre	dir
200	Grasano	Bs As 100
201	Atomo	Lima 85
202	VEA	Alvear 455

provee

Nprov	Nart	precio_Venta
200	100	4
200	101	1.1
201	101	1
201	102	0.5
202	102	1

compran

Ncli	Nart
300	100
300	102
301	101
302	101

Tablas con registros para los ejemplos

ARTICULOS					
Nart	descr	precio	cant	stock-min	stock-max
100	Café	5,2	8	10	50
101	Azucar	1,2	30	15	45
102	Harina	1,1	35	16	40

CLIENTES		
Ncli	nombre	dir
300	Pedro	Bs As 90
301	Juan	Jujuy 800
302	Maria	Cordoba 24

PROVEEDORES		
Nprov	nombre	dir
200	Grasano	Bs As 100
201	Atomo	Lima 85
202	VEA	Alvear 455

PROVEE		
Nprov	**Nart**	precio_Venta
200	100	4
200	101	1,1
201	101	1
201	102	0,5
202	102	1

COMPRAN	
Ncli	**Nart**
300	100
300	102
301	101
302	101

Ejemplo INSERTAR

1)
```
INSERT INTO articulos
(Nart,Descripcion,precio,Stock_max,Stock_min)
VALUES (5, 'Vino x1L', 1.5,90,20);
```

En este ejemplo se agrega un nuevo registro a la tabla artículos, pero en el campo cantidad no se pone ningún valor (ver definición tabla artículos)

2)
```
INSERT INTO proveedores  SELECT * FROM clientes
```

En este ejemplo se agrega a la tabla proveedores todos los clientes

Ejemplos ACTUALIZAR

1)

```
UPDATE articulos SET precio = precio * 1.2;
```

Esta instrucción incrementa los precios de todos los artículos en un 20 por ciento.

2)

```
UPDATE articulos SET precio = precio * 1.2
WHERE precio <2;
```

La ejecución de este comando incrementa los precios de los artículos cuyo precio es menor a $2, en un 20 por ciento.

Ejemplos ELIMINAR

1)

```
DELETE FROM proveedores;
```

La ejecución de este comando producirá que la tabla proveedores quede **vacía**.

2)

```
DELETE FROM proveedores
WHERE Nprov=200;
```

Este comando elimina el registro correspondiente al proveedor número 200.