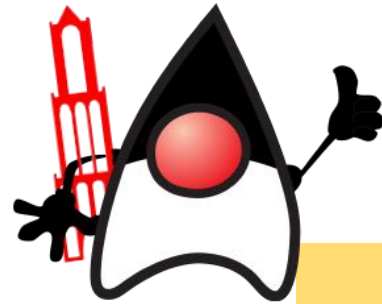
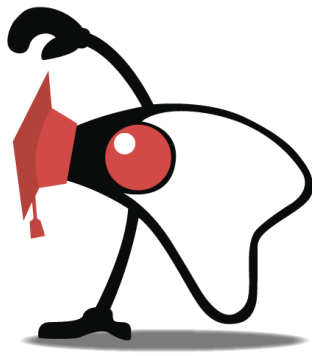




Curso FullStack Python

Codo a Codo 4.0



MySQL

Parte 3



Sentencias DML para modificar datos

- ❑ ¿Qué pasa cuando necesitamos ingresar una T-upla o más aún, modificar alguno de sus atributos?
- ❑ Existen **sentencias DML** especiales para realizar estas actividades dentro del lenguaje SQL:
 - ❑ Para agregar T-upla de datos (insertar registros): **INSERT**
 - ❑ Para modificar atributos de una o varias T-uplas (modificar registros): **UPDATE**
 - ❑ Para borrar T-uplas completas de una tabla (eliminar registros): **DELETE**

Sentencias SQL

DE ESCRITURA

- ❑ **INSERT INTO:** es utilizada para especificar en que tabla se pretende insertar un dato.
- ❑ **VALUES:** se utiliza en conjunto con **INSERT INTO** para especificar qué valores irán de la tabla.

En este caso, la lista de atributos será opcional, pero si no se define entonces el DBMS espera una lista de valores coherente con todos los atributos de la tabla.

Para hacer un INSERT vamos a la tabla donde queremos agregar los datos – Clic derecho – Send to SQL Editor – Insert Statement

```
1 • INSERT INTO `escuelas`.`alumnos`  
2   (`id`,  
3    `id_escuela`,  
4    `legajo`,  
5    `nombre`,  
6    `nota`,  
7    `grado`,  
8    `email`)  
9   VALUES  
10  (<{id: }>,  
11   <{id_escuela: }>,  
12   <{legajo: }>,  
13   <{nombre: }>,  
14   <{nota: }>,  
15   <{grado: }>,  
16   <{email: }>);  
17
```

Campos de los datos que le voy a pasar a la tabla

En el mismo orden

Valores que le voy a pasar a la tabla

Sentencias SQL



Ej 20

DE ESCRITURA: ejemplo

```
1 • INSERT INTO `escuelas`.`alumnos`  
2   (`id_escuela`, `legajo`, `nombre`, `nota`, `grado`)  
3   VALUES (4,1234,'Pedro Gómez',6,2);
```

ID	Escuela	Legajo	Nombre	Nota	Grado
9	4	1234	Pedro Gómez	6	2

Ejemplos sentencias SQL

❑ DE ESCRITURA

- ❑ Supongamos que tenemos una tabla de empleados y que queremos introducir un nuevo empleado, para ello realizamos la consulta:

```
INSERT INTO empleado  
VALUES (7, Carlos, Romero, M, 05-09-1985, 15500, abogado);
```

- ❑ El resultado de esta consulta sería la inserción de un nuevo registro en la tabla empleados.

Sentencias SQL

☐ DE MODIFICACIÓN

- ☐ **UPDATE:** es utilizada para especificar en que tabla se pretende modificar un dato.
- ☐ **SET :** Se utiliza en conjunto con **UPDATE** para especificar cuál será el nuevo valor/dato para el campo de ese/os registro/s en particular.

Para hacer un **UPDATE** vamos a la tabla donde queremos actualizar los datos – Clic derecho – Send to SQL Editor – Update Statement

¡CUIDADO! Si no hay cláusula WHERE lo que ocurrirá es que se actualizarán todas las T-uplas de la tabla.

Esto es un error muy común que cometen algunos usuarios de base de datos.

Sentencias SQL

SQL

Actualizar este registro

DE MODIFICACIÓN: ejemplo

```
1 • UPDATE `escuelas`.`alumnos`  
2 SET  
3 `nombre` = 'Roberto Luis Sánchez',  
4 `email` = 'robertoluissanchez@gmail.com'  
5 WHERE `id` = 6;
```

Campos a
modificar y
nuevos valores

Condición (*fundamental*)

Original

6	1	190	Roberto Sanchez	8	3	
---	---	-----	-----------------	---	---	--

Actualizado

6	1	190	Roberto Luis Sánchez	8	3	robertoluissanchez@gmail.com
---	---	-----	----------------------	---	---	------------------------------

Ejemplos sentencias SQL

❑ DE MODIFICACIÓN

- ❑ Supongamos que tenemos la tabla empleados y queremos modificar la fecha de nacimiento de “Juan Perez” que tiene el id empleado 1.

```
UPDATE empleado  
SET fecha_nacimiento = '23/10/1962'  
WHERE id_empleado = 1;
```

- ❑ Si no colocamos la cláusula WHERE, se modificarían las fechas de nacimiento de TODOS los registros de la tabla.

Sentencias SQL

☐ DE BAJA

- ☐ **DELETE:** es utilizada para eliminar uno o varios registro/s de una tabla de forma permanente.

Para hacer un DELETE vamos a la tabla donde queremos eliminar el/los registros – Clic derecho – Send to SQL Editor – Delete Statement

Acá hacemos la misma salvedad que en el caso del UPDATE, **si no se usa la cláusula WHERE lo que ocurrirá es que se eliminarán TODAS las Tuplas de la tabla y la misma quedará vacía**, lo que no es la intención habitual.

Sentencias SQL



Eliminar este registro

DE BAJA: ejemplo

```
1 • DELETE FROM `escuelas`.`alumnos`  
2 WHERE `id` = 3; } Criterio para eliminar
```

2	2	1002	Tomás Smith	8	1	
4	1	101	Juan Perez	10	3	

El registro con id = 3 ha sido eliminado

Ejemplos sentencias SQL

❑ DE BAJA

- ❑ Supongamos que tenemos la tabla empleados y queremos eliminar al empleado con la id 3.

```
DELETE FROM empleado  
WHERE id_empleado = 3;
```

- ❑ Si no colocamos la cláusula WHERE, se eliminarían TODOS los registros de la tabla empleado.

Modificando la estructura de las tablas



Ej 19 y 21

- ☐ **CREATE STATEMENT**

- ☐ Permite crear una tabla.

- ☐ **ALTER TABLE**

- ☐ Permite realizar cambios en la tabla.

- ☐ **DROP TABLE**

- ☐ Permite eliminar una tabla.



Cláusula JOIN

- ❑ El **JOIN** se utiliza para indicar la manera en que se están relacionando las tablas, es decir, con qué atributos se está plasmando la relación entre ellas.

```
SELECT campo1, campo2, ...,campoN FROM tabla1  
      JOIN tabla2 ON tabla1.campo1 = tabla2.campo2  
      JOIN tabla3 ON tabla2.campo3 = tabla3.campo4
```

¿Por qué tenemos que relacionar? Para no duplicar datos. Si tenemos una tabla **escuelas** y un alumno que pertenece una escuela no debemos repetir los datos de la escuela en la tabla alumnos. Las tablas deben estar **normalizadas**, esto quiere decir que los datos deben estar almacenados en forma eficiente para poder hacer consultas más rápidas, para que la BD no pese tanto, para que las tablas no tengan tantos campos (esto es área del diseño de BD).

Para unir las tablas vamos a necesitar **un dato** que relacione a ambas tablas, que las una.

INNER JOIN

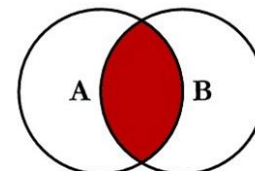


Ej 13 y 14

- ❑ **INNER JOIN** traerá solamente los registros que coincidan entre ambas tablas. Por ejemplo: si una escuela no tiene alumnos relacionados esa consulta no los traerá, del mismo modo si un alumno no tiene asignada una escuela tampoco lo mostrará.

```
1  -- Mostrar el legajo, el nombre
2  -- y el nombre de la escuela de todos los alumnos
3 • SELECT alu.legajo, alu.nombre, esc.nombre
4  FROM alumnos alu
5  INNER JOIN escuelas esc ON alu.id_escuela = esc.id;
```

Utilizo *alias de tabla* para identificarlas más fácilmente. Dentro del INNER JOIN establezco que para la tabla **escuelas** el campo **id_escuela** de la tabla **alumnos** está relacionado con el campo **id** de la tabla **escuela**.



```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```


LEFT JOIN



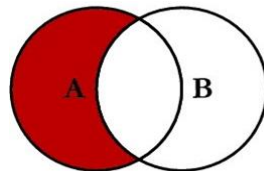
Ej 15

- ❑ **LEFT JOIN** tiene como condición que figure en, al menos una tabla. Left indica que va a tomar como tabla principal la de la izquierda. De esa tabla muestra todos los registros, sin importar si tiene registros asociados en la otra tabla.

```
1 -- Mostrar TODOS los alumnos con los datos de la escuela (opcional)
2 • SELECT alu.legajo, alu.nombre, esc.nombre
3 FROM alumnos alu
4 LEFT JOIN escuelas esc ON alu.id_escuela = esc.id;
```

	legajo	nombre	nombre
	101	Juan Perez	Normal 1
	105	Pedro Go...	Normal 1
	106	Martín Bo...	Normal 1
	1000	Ramón M...	Gral. San ...
	1002	Tomás Smith	Gral. San ...
	100	Ramiro Es...	EET Nro 2
	1234	Pedro Gó...	EET Nro 2
	190	Roberto L...	NULL

El último alumno no tiene escuela asociada o tiene una escuela asociada que no existe.



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

RIGHT JOIN



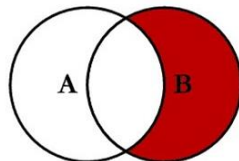
Ej 16

- ❑ **RIGHT JOIN** tiene como condición que figure en, al menos una tabla. Right indica que va a tomar como tabla principal la de la derecha. De esa tabla muestra todos los registros, sin importar si tiene registros asociados en la otra tabla.

```
1 -- Mostrar TODAS las escuelas con el nombre de cada alumno
2 • SELECT esc.*, alu.nombre
3 FROM escuelas esc
4 RIGHT JOIN alumnos alu ON esc.id = alu.id_escuela;
```

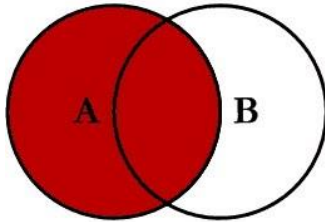
	id	nombre	localidad	provincia	capacidad	nombre
	1	Normal 1	Quilmes	Buenos Aires	250	Juan Perez
	1	Normal 1	Quilmes	Buenos Aires	250	Pedro Go...
	2	Gral. San ...	San Salvador	Jujuy	100	Ramón M...
	2	Gral. San ...	San Salvador	Jujuy	100	Tomás Smith
	4	EET Nro 2	Avellaneda	Buenos Aires	500	Ramiro Es...
	4	EET Nro 2	Avellaneda	Buenos Aires	500	Pedro Gó...
	NULL	NULL	NULL	NULL	NULL	Roberto L...
	NULL	NULL	NULL	NULL	NULL	Martín Bo...

Los últimos dos alumnos no tienen escuela asignada o tienen una escuela asignada que no existe.

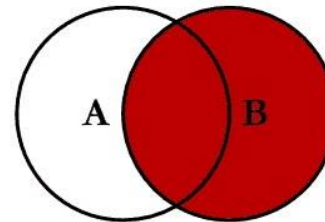


```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

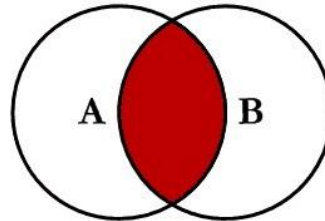
SQL JOINS



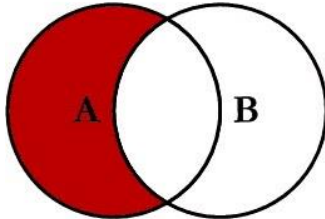
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



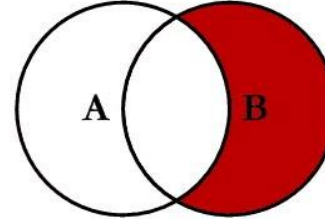
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



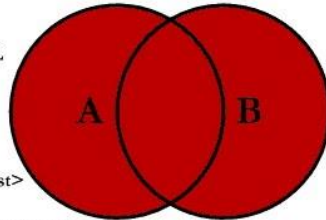
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



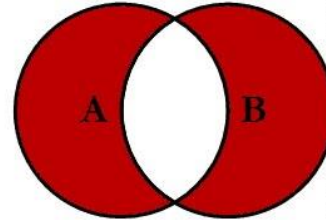
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```



USO DE FUNCIONES AGREGADAS



Ej 23 a 30

- ❑ Las funciones de agregación más comunes disponibles en el lenguaje son:

- ❑ SUM
- ❑ AVG
- ❑ MAX
- ❑ MIN
- ❑ COUNT

- ❑ La sintaxis del uso de las funciones agregadas es como sigue:

SELECT <lista de campos>, función agregada

FROM <tabla1 JOIN tabla2 ON....>

GROUP BY <lista de campos>

[HAVING función agregada <condición>]

USO DE FUNCIONES AGREGADAS

- ❑ Se utilizan conjuntamente con el **SELECT** ya que siempre van asociadas a una consulta.
- ❑ Además conceptualmente lo que hacen es reunir un conjunto de T-uplas de manera de juntar los datos para poder llevar a cabo la operación en cuestión (suma, promedio, cuenta, etc), por lo tanto van a agrupar T-uplas, de ahí la necesidad de la cláusula **GROUP BY**.
- ❑ Debés tener en cuenta que la <lista de campos> en el GROUP BY y en el SELECT es la misma.
- ❑ Si no hay una lista de campos, quiere decir que vamos a obtener una suma total, por lo tanto la cláusula GROUP BY tampoco es necesaria.

USO DE FUNCIONES AGREGADAS

- ❑ Veamos ahora un ejemplo para entender cómo funciona la suma y luego lo extenderemos al resto de las funciones agregadas citadas.
- ❑ La sintaxis del uso de las funciones agregadas es como sigue:

```
SELECT SUM(cant*pr.precio) FROM pedidos_productos pp  
      JOIN productos pr ON pp.codproducto=pr.codigo  
      JOIN pedidos p ON p.nro=pp.codpedido  
      WHERE month(p.fecha)=1 and year(p.fecha)=2017
```

- ❑ Observá que tenemos que usar la tabla pedidos para poder usar la condición del mes= enero y el año=2017 con la fecha, la tabla productos porque necesitamos los precios y la tabla pedidos_productos porque necesitamos la cantidad comprada de cada producto por cada pedido.

USO DE FUNCIONES AGREGADAS

- ❑ Si quisiéramos saber cuántas unidades se vendieron de cada producto por mes para el 2017, podríamos formularlo de la siguiente manera:

```
SELECT month(p.fecha) as Mes, SUM(cant) as Cantidad
FROM pedidos_productos pp
JOIN productos pr ON pp.codproducto=pr.codigo
JOIN pedidos p ON p.nro=pp.codpedido
WHERE year(p.fecha)=2017
GROUP BY month(p.fecha)
```

Mes	Cantidad
1	148
3	174
4	130

USO DE FUNCIONES AGREGADAS

- ❑ Si hubiéramos querido saber los pedidos cuyo total fuera superior a \$ 1000 hubiéramos tenido que hacer lo siguiente:

```
SELECT p.Nro, SUM(cant*precio) as total FROM pedidos_productos pp
      JOIN productos pr ON pp.codproducto=pr.codigo
      JOIN pedidos p ON p.nro=pp.codpedido
      GROUP BY p.nro
      HAVING SUM(cant*precio)>1000
```

ENTENDIENDO LAS CLÁUSULAS HAVING Y WHERE



Ej 31

- ❑ **WHERE** opera sobre registros individuales, mientras que **HAVING** lo hace sobre un grupo de registros.
- ❑ La anterior es la diferencia principal entre estas dos cláusulas. Con **WHERE podemos establecer una condición usando registros individuales**, aquellos que cumplan con esta condición serán seleccionados (eliminados o actualizados); ahora bien, con **HAVING podemos establecer una condición sobre un grupo de registros**, algo muy importante es que HAVING acostumbra ir acompañado de la cláusula GROUP BY. Esto último es así dado que HAVING opera sobre los grupos que nos “retorna” GROUP BY.
- ❑ Entonces cuándo usar HAVING o WHERE?, deberíamos usar HAVING solo cuando se vea implicado el uso de funciones de grupo (AVG, SUM, COUNT, MAX, MIN), debido a que con WHERE no podemos realizar condiciones que impliquen estas funciones.

SUBCONSULTAS



Ej 32

- ❑ Una subconsulta en SQL consiste en utilizar los resultados de una consulta dentro de otra, que se considera la principal. Esta posibilidad fue la razón original para la palabra “estructurada” en el nombre Lenguaje de Consultas Estructuradas (*Structured Query Language*, SQL).

```
SELECT numemp, nombre, (SELECT MIN(fechapedido) FROM pedidos WHERE rep = numemp)
FROM empleados;
```

En este ejemplo la consulta principal es **SELECT... FROM empleados**.

La subconsulta es **(SELECT MIN (fechapedido) FROM pedidos WHERE rep = numemp)**.

En esta subconsulta tenemos una referencia externa (*numemp*) es un campo de la tabla empleados (origen de la consulta principal).

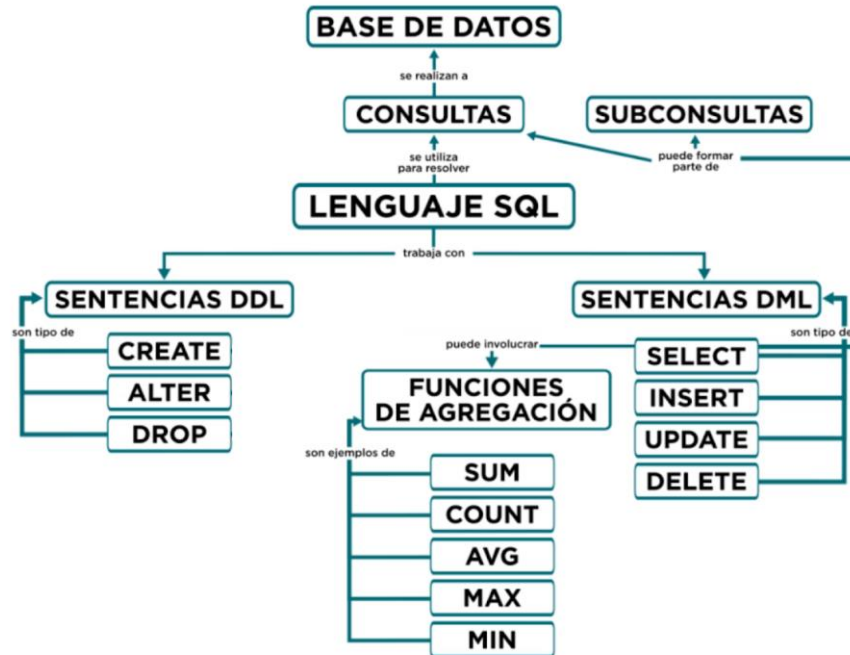
¿Qué pasa cuando se ejecuta la consulta principal?

- se toma el primer empleado y se calcula la subconsulta sustituyendo numemp por el valor que tiene en el primer empleado. La subconsulta obtiene la fecha más antigua en los pedidos del rep = 101,
- se toma el segundo empleado y se calcula la subconsulta con numemp = 102 (numemp del segundo empleado)... y así sucesivamente hasta llegar al último empleado.

Al final obtenemos una lista con el número, nombre y fecha del primer pedido de cada empleado.

SÍNTESIS

- Se presentaron todos los conceptos básicos que usted debe conocer para poder comenzar a trabajar con consultas en lenguaje SQL y para poder obtener información a partir de sus bases de datos.





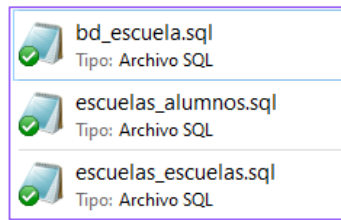
EXPORTAR UNA BD (BACKUP)



Ejercitar con BD Escuelas

Podemos exportar una Base de datos desde Workbench con el objetivo de hacer un backup:

1. Ir a **Server – Data export**
2. Seleccionar la base de datos (*schema*) que se desea exportar del cuadro de la izquierda dentro de Object Selection.
3. Seleccionar del cuadro de la derecha aquellas tablas que se desean exportar.
4. Determinar a qué carpeta se exportará la base de datos y cómo se exportarán los datos:
 - Si elegimos **Export to Dump Project Folder** se exportarán las tablas por separado.
 - Con **Export to Self-Contained File** podremos darle un nombre al archivo, pero con todas las tablas juntas.
5. Hacer clic en Start Export y colocar la contraseña del host.



MATERIAL COMPLEMENTARIO



- ❑ **Resumen SQL.pdf:** resumen con las sentencias SQL básicas más utilizadas.
- ❑ **Guía práctica de SQL:** guía de ejercicios con los que podrá poner en práctica los conocimientos de esta Unidad.

Nota: la guía **NO ES OBLIGATORIA** pero les dará la práctica necesaria para poder trabajar sin problemas con las bases de datos.

- ❑ **world.sql:** script para generar la base de datos que deberá utilizar para resolver la guía práctica.
- ❑ **der-bd-world.jpg:** DER de la base de datos anteriormente mencionada.
- ❑ **W3SCHOOLS – SQL Tutorial:** <https://www.w3schools.com/sql/>
- ❑ **Página Oficial MYSQL:** <https://dev.mysql.com/>