



Clase 11. SQL

# ***SUBLENGUAJE DML 1***

***RECUERDA PONER A GRABAR LA  
CLASE***





## ***OBJETIVOS DE LA CLASE***

- Reconocer e implementar las sentencias del sublenguaje DDL
- Identificar en qué situación usar cada sentencia

# ***CRONOGRAMA DEL CURSO***

## Clase 10



### Workshop DDL



REPASO DE LO APRENDIDO  
HASTA AHORA



ENTREGA INTERMEDIA

## Clase 11



### Sublenguaje DML 1



CONCEPTOS GENERALES,  
SENTENCIAS Y  
SUBLENGUAJES.



INSERCIÓN Y ACTUALIZACIÓN  
DE TABLAS

## Clase 12



### Sublenguaje DML 2



SUBLENGUAJES CON DML  
CON SUBCONSULTAS

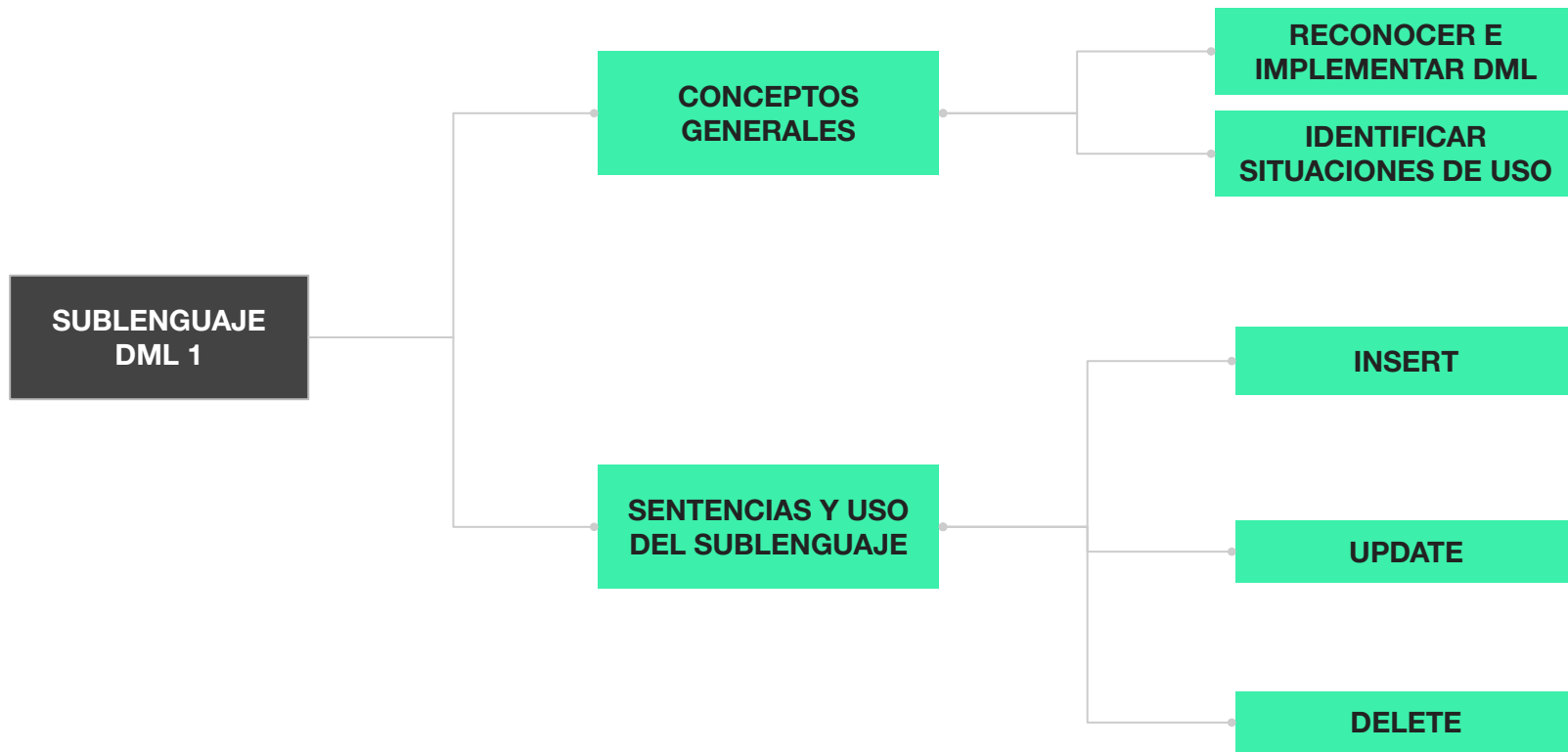


INSERCIÓN Y ACTUALIZACIÓN  
DE TABLAS II

# ***MAPA DE CONCEPTOS***

# MAPA DE CONCEPTOS CLASE 11

¡Para  
recordar!



# ***SENTENCIAS INSERT, UPDATE, DELETE***

# ***CONCEPTO GENERAL*** ***DML***



# ***SUBLENGUAJE DML 1***

Comenzamos a profundizar aún más el uso del DML, **Data Manipulation Language**.

En esta oportunidad conoceremos las otras sentencias SQL que nos permitirán **insertar, actualizar y eliminar** registros en las diferentes tablas de una base de datos.



# ***DATA MANIPULATION LANGUAGE***

Inicialmente aprendimos que **DML** posee una serie de sentencias que habilitan la manipulación de la información presente en las tablas de una DB, pudiendo **agregar, actualizar y/o eliminar** parte o toda esta información.

# ¿DML o DQL?

Si bien mencionamos oportunamente la sentencia **SELECT** como parte de **DML**, en algunos casos podemos encontrar publicaciones que la agrupan bajo el paradigma **DQL** (*Data Query Language*).

```
SELECT *  
  
FROM system_user  
  
WHERE last_name like  
  
    'B%';
```

# ***¿DML o DQL?***

```
SELECT *  
  
FROM system_user  
  
WHERE last_name like  
  
    'B%';
```

Esto tiene lógica dado que, el término **DQL**, hace foco en “*queries*” o “*consultas*” en sí, que es lo que realmente hacemos utilizando la sentencia **SELECT** para obtener datos.

# ***Componentes de DML***

Por lo tanto, DML engloba bajo su paraguas las siguientes sentencias SQL:

- **INSERT**
- **UPDATE**
- **DELETE**



# ***Componentes de DML***

**INSERT:** se utiliza para insertar o agregar registros en una tabla.

**UPDATE:** se utiliza para actualizar registros existentes en una tabla.

**DELETE:** se utiliza para eliminar registros de una tabla.

Cada sentencia requiere de ciertos cuidados al momento de utilizarlas,  
que mencionaremos con el análisis detallado de cada una.

***INSERT***

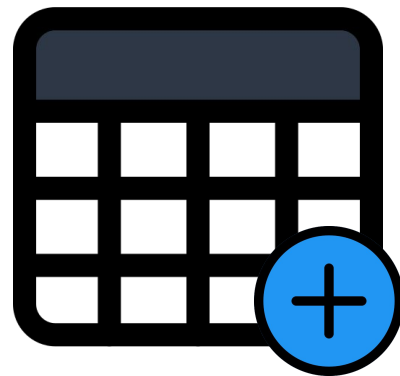
# ***INSERCIÓN DE REGISTROS***



# ¿Para qué se utiliza INSERT?

Como bien dijimos, **INSERT** se usa para **agregar o insertar datos** en una tabla. La información que agregamos puede ser :

- 👉 **De forma individual** (*1 registro*).
- 👉 **De forma plural** (*varios registros a la vez*).



# ***¿Cómo se utiliza INSERT?***

Su sintaxis se elabora mediante la cláusula

**INSERT INTO**, seguida del *nombre de la tabla*, la cláusula **VALUES**, y los datos que se insertarán en la misma.



```
INSERT INTO nombre_de_la_table (campo1, campo2, campo3,...)  
VALUES (dato1, dato2, dato3, ...);
```

# ¿Cómo se utiliza INSERT?

Cada “*dato*” hace referencia al dato individual que conformará el registro que deseamos insertar.



*Es conveniente detallar los nombres de los campos que componen la tabla, caso contrario tenemos que dar los valores a insertar en el orden que aparecen en la tabla (según orden de creación de la tabla).*

```
INSERT INTO nombre_de_la_table (campo1, campo2, campo3,...)  
VALUES (dato1, dato2, dato3, ...);
```

# ¿Cómo se utiliza INSERT?

MiTabla

campo1

campo2

campo3

campo4

Si mi tabla tiene 4 campos,  
entonces debería insertar 4  
datos, uno por cada campo.

VALUES

dato1

'dato2'

'dato3'

dato4

Si el tipo de datos de un campo es **number** o **boolean**, el dato a insertar  
no lleva comillas.

Si es **text** o **varchar**, sí lleva comillas.



## ***EJEMPLO EN VIVO***

*Insertión de datos.*

# Ejemplo práctico de INSERT

Ejemplo  
en vivo



```
INSERT INTO class (id_level,id_class,description) VALUES (1,999,'Spain comedy');
```

The screenshot shows a database management interface. On the left, a tree view displays the 'gammers\_model' database with various tables. The main area shows the execution of SQL queries. The first query is an INSERT statement, and the second is a SELECT statement to view the data. Below the queries, a 'Result Grid' displays the data inserted into the 'class' table. The row with id\_level 1, id\_class 999, and description 'Spain comedy' is highlighted with a green border.

SCHEMAS

Filter objects:

gammers\_model

- Tables
  - class
  - comment
  - commentary
  - game
  - level\_game
  - play
  - suggest
  - system\_user
  - user\_type
  - vote
- Views

Administration Schemas Information

Limit to 1000 rows

```
1 • INSERT INTO class (id_level, id_class, description) VALUES (1, 999, 'Spain comedy');
```

```
2
```

```
3 • SELECT * FROM class;
```

```
4
```

```
5
```

Result Grid

	id_level	id_class	description
11	294	Sport comedy	
12	297	Italian comedy	
11	300	British comedy	
1	999	Spain comedy	
*	NULL	NULL	NULL

Filter Rows:

Edit: Export/Import: Wrap Cell Content:

Result Grid

Form Editor

# ***Ejemplo práctico de INSERT***

Como observamos en la diapositiva anterior, **el total de datos insertados** en la tabla **corresponde con el total de campos** con la que fue declarada la misma.



```
INSERT INTO class (id_level, id_class, description)

VALUES (1, 999, 'Spain comedy');
```

***¿Y LOS CAMPOS 'AUTOINCREMENT'?***



# ***INSERT: Campo autoincrement***

En el caso de contar con campos **AUTOINCREMENT**, tenemos la posibilidad de permitir que el DBMS resuelva el valor correlativo para éste, pasando como parámetro el valor **NULL** o no colocandolo en la lista de campos. Tomemos como ejemplo la tabla PAY que creamos en la **clase 6**.



```
INSERT INTO pay VALUES (NULL, 250, 'U$S', '2021-07-22', 'Paypal', 850, 77);
```

# INSERT: Campo autoincrement

**SCHEMAS**

Filter objects

**gammers\_model**

- Tables
  - class
  - comment
  - commentary
  - game
  - level\_game
  - play
  - suggest
  - system\_user
  - user\_type
  - vote
- Views

Administration Schemas

Information

No object selected

Limit to 1000 rows

```
3      amount      REAL NOT NULL DEFAULT 0,
4      currency    VARCHAR(20) NOT NULL,
5      date_pay     DATE NOT NULL,
6      pay_type     VARCHAR(50),
7      id_system_user INT NOT NULL,
8      id_game      INT NOT NULL
9  );
10
11 • INSERT INTO pay VALUES (NULL, 250, 'U$S', '2021-07-22', 'Paypal', 850, 77);
12
13 • SELECT * FROM pay;
14
```

Result Grid

	id_pay	amount	currency	date_pay	pay_type	id_system_user	id_game
▶	1	250	U\$S	2021-07-22	Paypal	850	77
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

**CODER HOUSE**

# ***INSERTAR DATOS PARCIALES***

# ***Insertar datos parciales***

Como ya mencionamos, podemos insertar datos en determinados campos de una tabla. Para ello, debemos especificar en la sentencia **INSERT INTO**, cuáles son los campos donde insertamos información.

```
INSERT INTO pay
```

```
(id_pay, amount, currency,  
date_pay, id_system_user,  
id_game)
```

```
VALUES ( NULL, 300, 'U$S',  
'2021-07-22', 501, 13);
```

# ***Insertar datos parciales***

El total de datos a insertar debe condecir con el total de campos mencionados, además de tener que coincidir el tipo de dato.

Si la tabla posee un campo **AUTOINCREMENT**, debes referenciarlo, asignándole el valor **NULL**

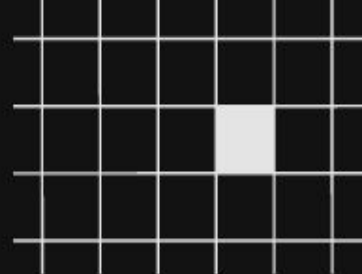
```
INSERT INTO pay  
(id_pay, amount, currency,  
date_pay, id_system_user,  
id_game)  
VALUES ( NULL, 300, 'U$S',  
'2021-07-22', 501, 13);
```

# ***Insertar datos parciales***

Si los campos que obviamos poseen un valor definido por defecto, estos aparecerán en el registro insertado.

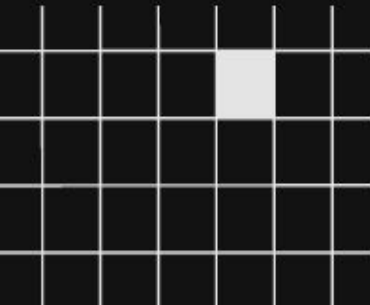
Caso contrario, el dato quedará como ***NULL***.

Column	Type	Default Value	Nullable	Character Set	Collation
◇ id_pay	int		NO		
◇ amount	double	0	NO		
◇ currency	varchar(20)		NO	utf8mb4	utf8mb4_0900_...
◇ date_pay	date		NO		
◇ pay_type	varchar(50)		YES	utf8mb4	utf8mb4_0900_...
◇ id_system_user	int		NO		
◇ id_game	int		NO		



## ***EJEMPLO EN VIVO***

*Insertión de datos parciales.*



# Insertar datos parciales

Ejemplo  
en vivo



```
INSERT INTO pay (id_pay, currency, date_pay, pay_type, id_system_user, id_game)  
VALUES ( NULL, 'U$S', '2021-07-22', '', 127, 91);
```

**SCHEMAS**

Filter objects

- gammers\_model
  - Tables
    - class
    - comment
    - commentary
    - game
    - level\_game
    - pay
    - play
    - suggest
    - system\_user
    - user\_type
    - vote

Administration Schemas

**Information**

**Table: pay**

**Columns:**

- id\_pay int AI PK
- amount double
- currency varchar(20)

**Result Grid**

	id_pay	amount	currency	date_pay	pay_type	id_system_user	id_game
▶	1	250	U\$S	2021-07-22	Pavpal	850	77
*	2	0	U\$S	2021-07-22		127	91

**CODER HOUSE**



# ***INSERTAR MÚLTIPLES DATOS***

# ***Insertar múltiples datos***

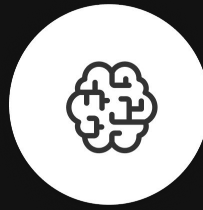
Finalmente, podemos aprovechar la sentencia **INSERT** para agregar múltiples registros en una misma ejecución. Cada nuevo registro debe encerrarse entre (...) y separarse con una coma.

```
INSERT INTO pay VALUES  
(NULL, 250, 'U$S', '2021-07-22', 'Paypal', 850, 77),  
(NULL, 3700, 'Pesos Arg', '2021-07-22', 'Visa', 38, 31),  
(NULL, 180, 'Libras', '2021-07-22', 'Transfer', 175, 16);
```

# ***Insertar múltiples datos***

De igual forma  
procederemos en el caso  
de tener que agregar  
múltiples registros sin  
especificar todos los  
campos de la tabla.

	id_pay	amount	currency	date_pay	pay_type	id_system_user	id_game
▶	1	250	U\$S	2021-07-22	Paypal	850	77
	2	0	U\$S	2021-07-22		127	91
	3	250	U\$S	2021-07-22	Paypal	850	77
	4	3700	Pesos Arg	2021-07-22	Visa	38	31
	5	180	Libras	2021-07-22	Transfer	175	16
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL



## ***¡PARA PENSAR!***

*¿Es más efectivo insertar múltiples registros en una tabla usando un INSERT por cada uno, o conviene usar una única sentencia INSERT para todos los registros?*

CONTESTA LA ENCUESTA DE ZOOM





***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**

**Si adquiriste un servicio de talento,  
recordá chequear tu correo de  
spam, no deseado, publicidad y/o  
social**

**En caso de no haberlo recibido, escribinos a [talento@coderhouse.com](mailto:talento@coderhouse.com)**

**¡5/10 MINUTOS Y VOLVEMOS!**

***CODER HOUSE***

***UPDATE***

***ACTUALIZAR REGISTROS***



# ***Acerca de SQL UPDATE***

La sentencia SQL UPDATE actualiza registros almacenados en uno o más campos de una tabla.



Su sintaxis es simple, y **permite hacer una actualización masiva** de datos, o **aquellos registros** que cumplan **con una determinada condición.**

# ***Sintaxis de UPDATE***

Su sintaxis se elabora mediante la cláusula **UPDATE tabla**, seguida la palabra **SET** y luego la(s) columna(s) o campo(s) con su(s) nuevo(s) valor(es).



```
UPDATE nombre_de_la_tabla SET campo2 = 'dato2';
```

# ***Sintaxis de UPDATE***

Para actualizar el valor de más de un campo, debemos

separar cada uno de éstos con una coma

Especificamos **campo = valor, otroCampo = otroValor**, y así con cada campo que deseamos actualizar



```
UPDATE nombre_de_la_tabla SET campo1 = 'dato1', campo2 = 'dato2', campo3 = 'dato3';
```

# ***Sintaxis de UPDATE***

Al igual que vimos con la sentencia **INSERT**;  
**UPDATE** sigue con la definición de delimitar los nuevos valores entre comillas cuando el campo es **TEXT** o **VARCHAR**, y sin comillas en aquellos casos que el campo es **NUMBER** o **BOOLEAN**.



```
UPDATE nombre_de_la_tabla SET categoria_id = 123, nombre = 'nuevo nombre';
```

***ACTUALIZAR DATOS QUE CUMPLAN  
DETERMINADA CONDICIÓN***

# ***UPDATE: Condicionar la actualización***

Podemos **limitar la actualización de datos** que cumplan una **determinada condición.**

Por ejemplo, **cambiar la fecha de pago** de aquellos realizados en el día. Para esto debemos **integrar la cláusula WHERE**, especificando la o las condiciones que deben cumplirse.



# ***UPDATE:***

## ***Condicionar la actualización***

Volviendo a nuestro ejemplo de pagos; a través de la sentencia **UPDATE** representada aquí, actualizaremos la fecha de pago de aquellos pagos del día.

```
UPDATE pay  
  
SET date_pay =  
  
CURRENT_DATE - 1  
  
WHERE date_pay =  
  
CURRENT_DATE;
```



## ***EJEMPLO EN VIVO***

*Actualización de datos masiva.*



# Actualización masiva de datos

Ejemplo  
en vivo

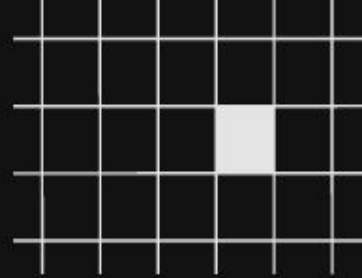
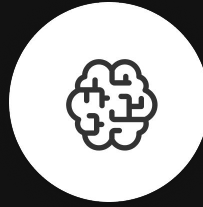


	id_pay	amount	currency	date_pay	pay_type	id_system_user	id_game
▶	1	250	U\$S	2021-07-22	Paypal	850	77
	2	0	U\$S	2021-07-22		127	91
	3	250	U\$S	2021-07-22	Paypal	850	77
	4	3700	U\$S	2021-07-22	Visa	38	31
	5	180	Libras	2021-07-22	Transfer	175	16
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
UPDATE pay
```

```
SET currency = 'U$S'
```

```
WHERE id_pay = 4;
```

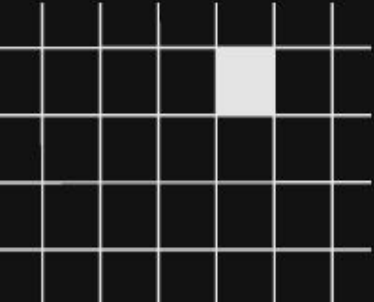


## ***¡PARA PENSAR!***

*Si deseamos actualizar los niveles de las clases de juegos, pasar a nivel 8 todas las clases que están entre la 1 y la 20 inclusive y cuyos niveles actuales están por debajo del 13  
¿Cuántos registros se actualizarían yCuál sería la cláusula  
**UPDATE?***



CONTESTA EN EL CHAT



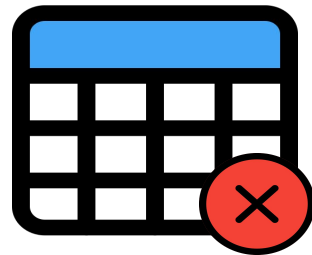
***DELETE***

***ELIMINAR REGISTROS***

# ***¿Para qué se utiliza DELETE?***

Finalmente, para eliminar registros de una tabla, debemos utilizar la sentencia **DELETE**.

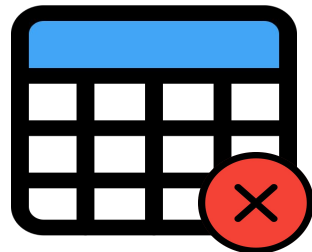
Se ocupa de eliminar todos los registros, o sólo aquellos que coincidan con determinados parámetros que le indiquemos en la condición del **WHERE**.



# ***Sintaxis de DELETE***

Su sintaxis se elabora mediante la cláusula **DELETE FROM** seguido del **nombre de la tabla**.

Además debemos agregar la cláusula **WHERE**, para **indicarle cuál o cuáles registros se deben eliminar**.



```
DELETE FROM nombre_de_la_tabla WHERE (campo = 'dato');
```



## ***¡ATENCIÓN CON ESTA CLÁUSULA!***

Seamos cuidadosos al utilizar **DELETE**. Siempre comencemos escribiendo **WHERE**, junto a la o las condiciones que deben cumplir los registros a eliminar.

Un error involuntario, como el olvidar el condicional, hará que **perdamos toda la información** de la tabla

# ***Eliminar el producto con DELETE***

Si deseamos eliminar la clase cuyo **id\_level** y **id\_class** sean (1, 999) debemos elaborar la siguiente sentencia **DELETE**.

En éste caso el borrado cumple con más de una condición.

```
DELETE FROM class
```

```
WHERE id_level = 1 and id_class = 999;
```



# ***POSIBLES ERRORES AL INTENTAR ELIMINAR REGISTROS***

# ***DELETE: Posibles errores***

Si intentamos eliminar registros de una tabla cuya **PRIMARY KEY** es **FOREIGN KEY** en otra u otras tablas, SQL no realizará la operación y nos advertirá dicho impedimento a través de la consola.

Por ejemplo **delete from level\_game where id\_level = 5;**



Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails (`gamers`.`level\_game`, CONSTRAINT `fk\_class\_level` FOREIGN KEY (`id\_level`) REFERENCES `le` (`id`))

# ***DELETE: Posibles errores***

Action Output				
		Time	Action	Response
✓	1	19:51:12	USE ventas_ecommerce	0 row(s) affected
✗	2	19:51:12	DELETE FROM productos WHERE...	Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails (`ventas_ecommerce`.`stock`, CONSTRAINT...

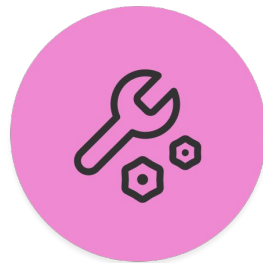
Ante esta situación, debemos eliminar primero el registro asociado mediante la **FOREIGN KEY** en la otra tabla, para luego proceder con la eliminación de este registro.

# ***ELIMINACIÓN TOTAL DE REGISTROS DE UNA TABLA***

# ***TRUNCATE TABLE***

Recordemos que para eliminar todos los registros de una tabla podemos utilizar **TRUNCATE** en lugar de **DELETE sin where**. Internamente, **TRUNCATE** borra todos los registros mientras que **DELETE**, recorre uno a uno y los va eliminando.

```
TRUNCATE nombre_de_la_tabla;
```



# ***INSERCIÓN Y ACTUALIZACIÓN DE TABLAS***

Realizaremos operaciones de inserción y actualización de registros.

Tiempo estimado: 10 minutos

# INSERCIÓN Y ACTUALIZACIÓN DE TABLAS

Desafío  
generico



Basado en las tablas creadas con el **Diagrama E-R** de la **clase 7**, insertar al menos 2 registros en cada tabla.

Luego de realizado, elige una tabla y modifica al menos el dato de 1 registro insertado.

Tiempo estimado: 10 minutos

## DISEÑO DE UN DIAGRAMA E-R

Desafío  
generico

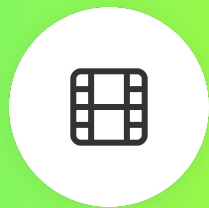


Recuperaremos el **Diagrama E-R** que realizamos en la **Clase 1**. Con los conocimientos adquiridos hasta ahora, crearemos una base de datos nueva, integrando a la misma las tablas realizadas en este diagrama E-R.

Tengan presente que cada una de las tablas deberá tener, como mínimo, unos 3 campos. Si tienen o desean agregarles más, mejor aún.

**CODER HOUSE**

**CODER HOUSE**



***¿QUIERES SABER MÁS? TE DEJAMOS  
MATERIAL AMPLIADO DE LA CLASE***





- [Insertar datos en una tabla](#) | **Anner Barrena**
- [Actualizar datos de una tabla](#) | **Anner Barrena**
- [Eliminar filas de una tabla](#) | **Anner Barrena**



Finalmente, le ponemos un poco de humor... basado en hechos reales

**No te olvides de poner el WHERE** - [Youtube](#)

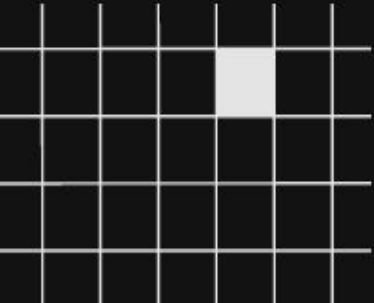
***¿PREGUNTAS?***





# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Sentencias insert, update, delete.
  - Implementación de cada sentencia.
- 



***OPINA Y VALORA ESTA CLASE***

***#DEMOCRATIZANDO LA EDUCACIÓN***

***CODER HOUSE***