



**Argentina
programa
4.0**



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

***primero
la gente***

Clase 26: Bases de datos Relacionales

Data Manipulation Language

Agenda de hoy

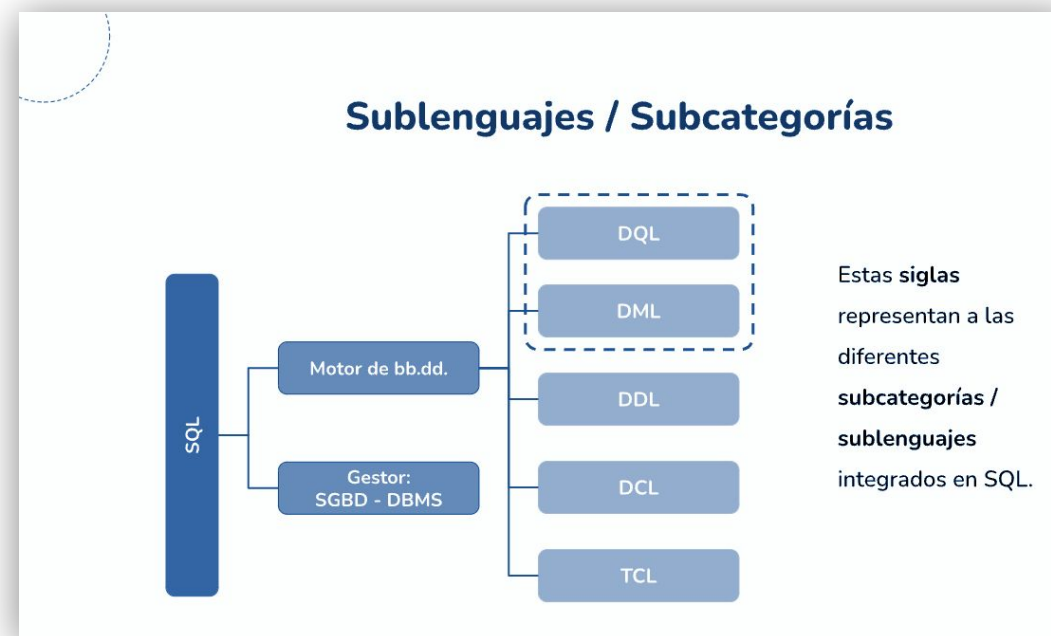
- A. Data Manipulation Language
- B. INSERT
 - a. individual
 - b. masivo
- C. UPDATE
 - a. individual
 - b. masivo
- D. DELETE
 - a. precauciones
- E. TRUNCATE



Data Manipulation Language

Data Manipulation Language

Seguimos recorriendo los diferentes sublenguajes/subcategorías de MySQL. En esta oportunidad analizaremos el poder de Data Manipulation Language, y todas las ventajas que nos da al momento de trabajar con modificaciones en los datos almacenados en una bb.dd.



Data Manipulation Language



DATA MANIPULATION LANGUAGE engloba todas las operaciones principales para manipular datos simples o cuantiosos de una bb.dd.

Data Manipulation Language

DQL

DML

DDL

DCL

TCL

DATA MANIPULATION LANGUAGE engloba el conjunto de comandos y operaciones que permiten manipular y modificar los datos almacenados en una base de datos.

En el contexto de MySQL, un sistema de gestión de bases de datos relacional, el DML se utiliza para realizar acciones como la inserción, actualización, eliminación y consulta de datos en las tablas.

Data Manipulation Language

DML en MySQL proporciona un conjunto de comandos, como:

- INSERT
- UPDATE
- DELETE
- TRUNCATE

Con ellas interactuamos con los datos de la base de datos de una manera controlada y estructurada.



Data Manipulation Language

Estos comandos permiten **agregar** nuevos datos, **actualizar** los existentes, **eliminar** registros de acuerdo con ciertos criterios.

Cada una de estas sentencias requiere de ciertos cuidados o parámetros al momento de utilizarlas. Los mismos serán mencionados a medida que analicemos cada una de ellas.

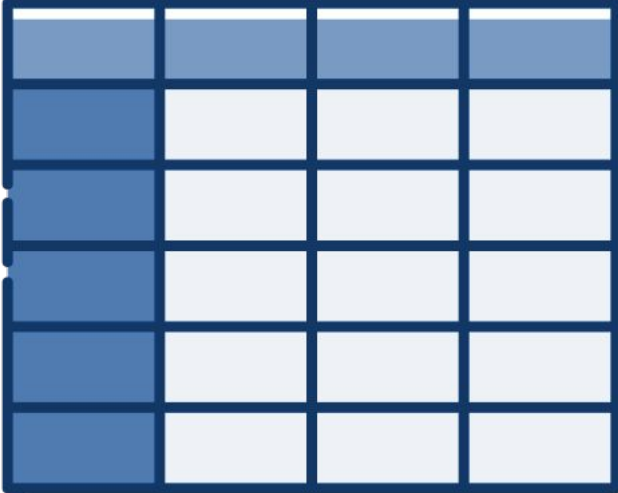


INSERT

INSERT

INSERT: Insertar datos en una tabla.

La información que debemos insertar puede ser realizada de forma individual (*un solo registro*), o de forma masiva (*varios registros a la vez*).



INSERT simple

INSERT

Estructura de la consulta **SQL INSERT**, para agregar un nuevo registro a una tabla determinada. Contiguo a la cláusula **VALUES** se abre paréntesis, y se agregan los datos de cada uno de los campos que tiene dicha tabla.

El orden es estricto de acuerdo a cómo se definieron los campos.

Cláusula INSERT

```
INSERT INTO contactos  
VALUES ('valor 1', 'valor 2', 'valor 3', 'valor 4')
```

INSERT

En este ejemplo, representamos el formato a definir para cada tipo de dato de acuerdo a como hayan sido creados los campos. Debemos tener presente siempre que, el formato fecha, se debe establecer con la **estructura ISO**.

Si el campo fecha es **datetime**, podemos obviar el ingreso de la hora. MySQL completará el dato agregando **00:00:00**.

```
Cláusula INSERT (tipos de datos)

INSERT INTO dbo.contactos
VALUES ('valor 1', '2022-05-27', 1145467899, TRUE)
--      texto      fecha-hora  numérico  boolean
```

INSERT

Cuando insertamos datos en una tabla que contiene un campo ID (*identificador*, *autonumérico*), debemos informar para este un dato del tipo **NULL**. MySQL resuelve internamente qué número le debe generar. Esta es otra diferencia que existe con SQL Server. En este último podemos obviar directamente, el pasarle un valor a los campos del tipo **id**.

Cláusula INSERT (campos autoincrementables)

```
INSERT INTO dbo.contactos
VALUES (NULL, 'valor 1', '2022-05-27', 1145467899, TRUE)
--      predefinir un valor NULL en los campos
--      autoincrementables
```

INSERT masivo (múltiple)

INSERT

Estructura de la consulta **SQL INSERT**, para agregar registros de forma masiva. Mantenemos la estructura de los campos a insertar, y separamos cada nuevo registro del anterior, con una coma.

Cláusula INSERT (registros masivos)

```
INSERT INTO dbo.contactos
VALUES ('valor a', 'valor b', 'valor c', 'valor d'),
      ('valor e', 'valor f', 'valor g', 'valor h'),
      ('valor i', 'valor j', 'valor k', 'valor l');
```

INSERT

Esta opción es muy útil para poder trabajar, desde una aplicación cliente, de forma desconectada de la base de datos, y solo establecer una ventana de conexión para agregar de una vez todos los registros que se necesiten.

```
Cláusula INSERT (registros masivos)

INSERT INTO dbo.contactos
VALUES ('valor a', 'valor b', 'valor c', 'valor d'),
      ('valor e', 'valor f', 'valor g', 'valor h'),
      ('valor i', 'valor j', 'valor k', 'valor l');
```

INSERT parcial

INSERT

Estructura de la consulta **SQL INSERT**, para agregar un nuevo registro, ingresando datos de forma parcial sobre determinados campos de la tabla.

Esto es ideal cuando la tabla permite nulos en los campos que no seleccionamos y/o, a su vez, esos campos tienen configurado un valor por defecto desde el diseño de la tabla.

```
Cláusula INSERT (Parcial)

INSERT INTO dbo.contactos
(id, campo2, campo3, campo5)
VALUES (NULL, 'valor 2', 'valor 3', 'valor 5')
```

INSERT

En este ejemplo, la tabla tiene configurado desde su diseño, la opción de aceptar un **dato nulo** en uno de sus campos. De esa forma, se registrará el término **NULL** en **campo4**, dado que no se le pasó un parámetro específico a dicho campo.

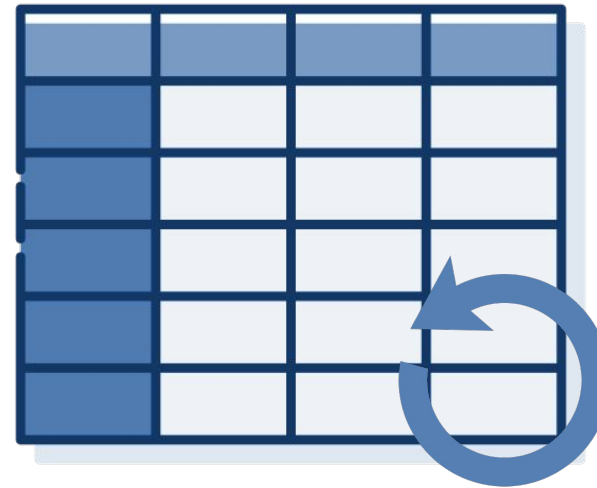
ID	campo2	campo3	campo4	campo5
10	valor 2	valor 3	<i>NULL</i>	valor 5

UPDATE

UPDATE

Actualiza datos existentes en una tabla.

UPDATE nos permite hacer una actualización masiva de datos, o solo en aquellos registros que cumplan determinada condición.



UPDATE

Estructura de la consulta **SQL UPDATE**, permite definir de forma parcial el campo en el cual se debe actualizar un valor almacenado.

Para ello, se antepone la palabra reservada **SET** antes del campo, para luego establecer el dato a actualizar. La instrucción **WHERE** permite definir en qué registro se debe aplicar dicha actualización.

Cláusula UPDATE

```
UPDATE dbo.contactos  
SET campo2 = 'valor 2'  
WHERE ID = 21
```


UPDATE

Si disponemos de varios campos a actualizar, y no uno solo, entonces debemos separar cada uno de ellos por una coma. Siempre utilizando la palabra reservada **SET**.

```
Cláusula UPDATE parcial

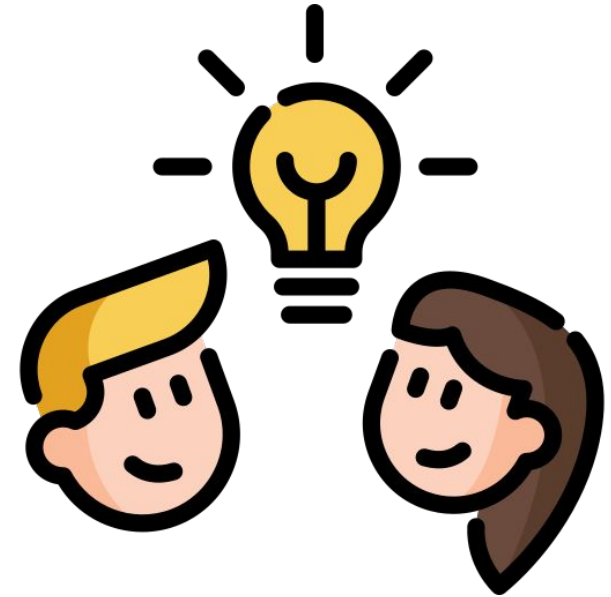
UPDATE dbo.contactos
SET campo2 = 'valor 2', campo3 = 'valor 3'
WHERE ID = 21;
```

UPDATE masivo (¡Atención!)

Siempre que definamos una consulta de actualización de datos en SQL, tengamos la precaución de escribir primero la condición WHERE.

Es un error muy común olvidar incluir la condición.

En caso de olvidarla, todos los registros serán actualizados en dicho campo, alterando su valor original.



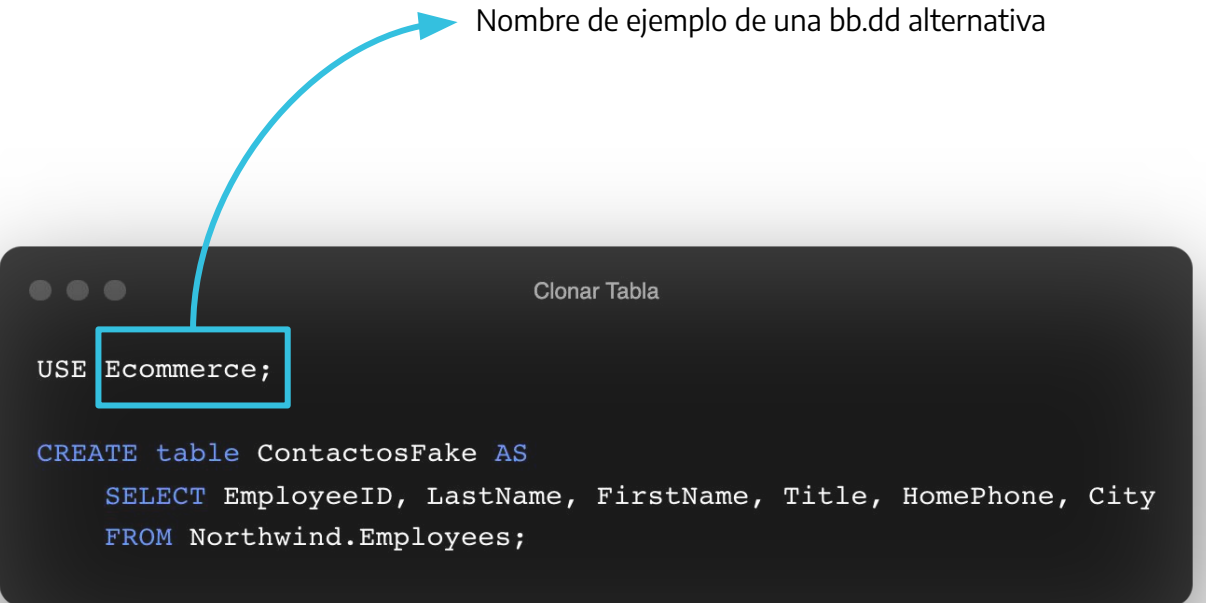
Clonar tablas a partir de una existente

Clonar tablas a partir de una existente

Realicemos entre todas un ejercicio de clonación de tablas. Para ello, utilicemos cualquier base de datos de pruebas que tengamos para clonar una tabla de **Northwind**, y trabajar luego con los ejercicios asociados:

Clonaremos entonces la tabla **Employees** de **Northwind** bajo el nombre **ContactosFake** en cualquier otra bb.dd.

Luego modificamos su clave primaria y definimos el campo id como autonumérico, dado que en una clonación, esta configuración se pierde.



```
USE Ecommerce;

CREATE table ContactosFake AS
SELECT EmployeeID, LastName, FirstName, Title, HomePhone, City
FROM Northwind.Employees;
```

Nombre de ejemplo de una bb.dd alternativa

Prácticas en clase

A partir de la tabla clonada anteriormente, realicemos el siguiente ejercicio insertando datos en la tabla recientemente clonada:

1. Actualiza en **ContactosFake** aquellos contactos que estén en la Ciudad de 'Seattle' por 'CABA'
2. Actualiza en **ContactosFake** el empleado cuyo ID es 5, su campo Título por el de 'Gerente de Ventas'
3. Actualiza en **ContactosFake** el campo Título por 'Analista de Ventas' para los empleados cuyo ID sean: 1, 3, 4, 6, 7, 9

1. Inserta los siguientes registros en la tabla **ContactosFake**:

LastName	FirstName	Title	HomePhone	City
Sandberg	Sheryl	COO @ Facebook	11-555-9999	Menlo Park
Wojcicki	Susan	CEO @ Youtube	11-555-2222	San Bruno
Rometty	Ginni	EX CEO @ IBM	11-555-5555	Armonk



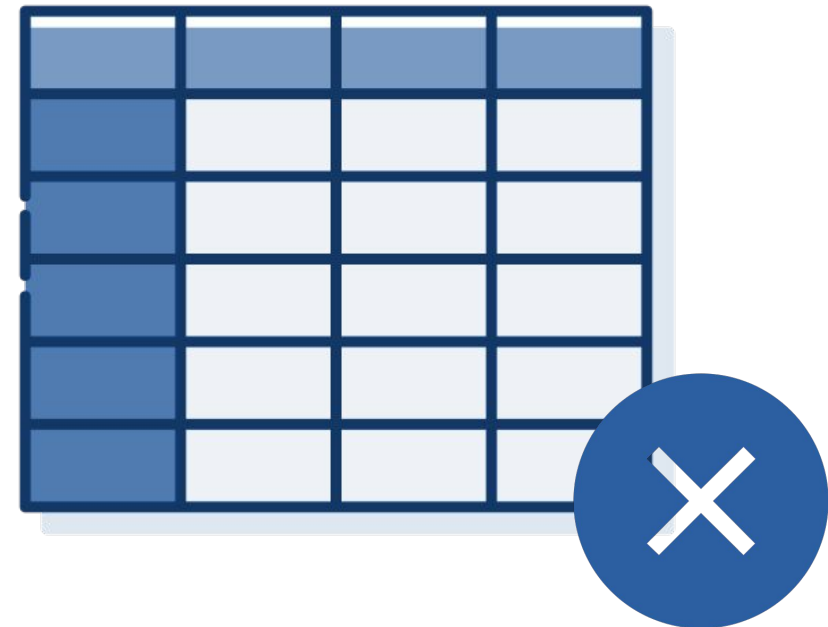
Puesta en común del ejercicio

DELETE

DELETE

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

Esta se encarga de eliminar registros de forma masiva, o sólo aquellos que coincidan con determinados parámetros que debemos indicar en la consulta de eliminación.



DELETE

Estructura de la consulta **SQL DELETE**, para eliminar un registro de la tabla. La instrucción **WHERE** determina cuál será el registro que deseamos eliminar.

Cláusula DELETE

```
DELETE FROM dbo.contactos  
WHERE ID = 21;
```

DELETE

Podemos integrar todas las combinaciones posibles utilizadas anteriormente en los filtros de las consultas de selección, para especificar una cantidad de registros a eliminar que cumplan con una condición específica o parcial.

Cláusula DELETE y condicional LIKE

```
DELETE FROM dbo.contactos  
WHERE Email LIKE '%@cardiffcomputers.com';
```

DELETE

Podemos integrar todas las combinaciones posibles utilizadas anteriormente en los filtros de las consultas de selección, para especificar una cantidad de registros a eliminar que cumplan con una condición específica o parcial.

Cláusula DELETE y condicional LIKE

```
DELETE FROM dbo.contactos  
WHERE Email LIKE '%@cardiffcomputers.com';
```

DELETE

Para una eliminación de selección variada, podemos aplicar el operador **IN**, seguido de las condiciones específicas (en este caso, los **ID**).

De igual forma, si queremos eliminar un rango mayor o igual a un valor condicional específico, y hasta los operadores lógicos (**AND - OR - NOT**) pueden ser incluídos.

Cláusula DELETE y otros condicionales

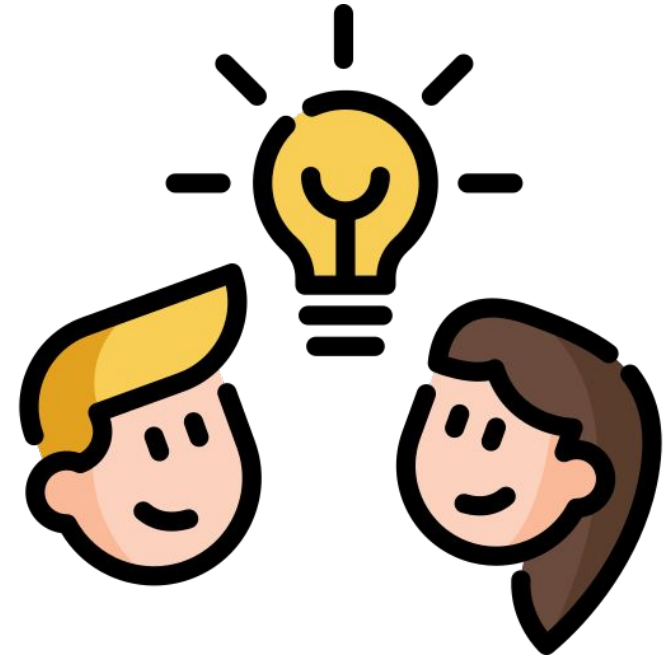
```
DELETE FROM dbo.contactos  
WHERE ID IN (1, 8, 9, 13);
```

```
DELETE FROM dbo.Personas  
WHERE Edad >= 45;
```

DELETE MASIVO (¡Atención!)

De igual forma a lo recomendado con las consultas de actualización, en el uso de consultas **DELETE**, siempre se recomienda definir el **WHERE** en primer lugar.

De obviarlo, y ejecutar esta consulta de eliminación accidentalmente, eliminaremos todos los registros de la tabla en cuestión.



DELETE

Dada la importancia de afianzar los conocimientos para el uso de la cláusula **DELETE** durante el manejo de datos en una base de datos relacional, se creó una canción bastante pegadiza que cuenta con humor, las posibles implicancias del mal uso de DELETE.

Aprendamos la letra, lo cual nos ayudará a entender mejor estos escenario críticos en el mundo de las bases de datos: 😂



Limitaciones en el uso de cláusulas DML

Limitaciones en el uso de cláusulas DML

Existe una serie de limitaciones a tener en cuenta cuando realizamos algún tipo de operación DML sobre las tablas. Estas se dan en determinadas situaciones, como ser:

- campos obligatorios (*NOT NULL*)
- campos con claves foráneas (*Foreign Key*)
- inexistencia de valores por defecto (*Default value or binding*)

TRUNCATE

TRUNCATE

En el caso de tener que eliminar todos los registros de una tabla, debemos recurrir a la sentencia **TRUNCATE** en lugar de utilizar **DELETE**.

Esta sentencia es mucho más performante por su forma de trabajar, que la que ofrecida por la sentencia **DELETE**, la cual está más enfocada a eliminar registros en pequeños grupos y no de forma total.



TRUNCATE

Esta sentencia cuenta con algunas particularidades:

- No puede eliminar registros con **constraint**
- Resetea el **campo autoincremental** al eliminar registros
- No registra en el archivo **LOG**, la eliminación de datos



TRUNCATE

Su sentencia es la más simple de todas.

Debemos estar totalmente seguros de la eliminación de datos, porque no hay vuelta atrás una vez ejecutada esta sentencia.

Cláusula TRUNCATE TABLE

```
TRUNCATE dbo.contactos;
```

Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*