



Argentina programa 4.0



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*

Clase 17: Bases de datos Relacionales

Introducción a SQL con el motor (MySQL)

Agenda de hoy

- A. Historia de las bb.dd.
- B. Fundamentos de las bb.dd relacionales
- C. Motores SQL
 - a. Oracle Database
 - b. SQL Server
 - c. MySQL
- D. Cómo funciona un motor SQL
- E. MySQL y MySQL Workbench
 - a. Objetos en una BB.DD.
 - b. Crear nuestra BB.DD.



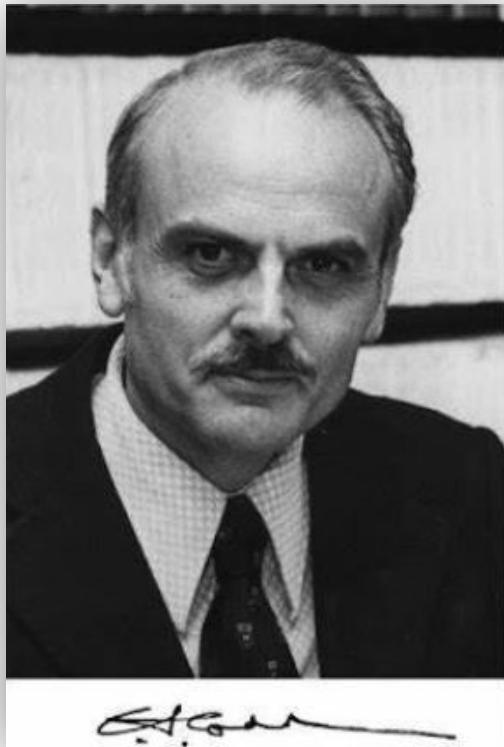
Bases de datos Relacionales

Llegó el momento de hacer una pausa en el mundo de la programación Backend, para concentrarnos en un terreno sumamente importante: **las bases de datos relacionales**.

Durante las próximas clases, nos centraremos en aprender esta tecnología enfocándonos completamente en el mundo de las bb.dd. SQL.



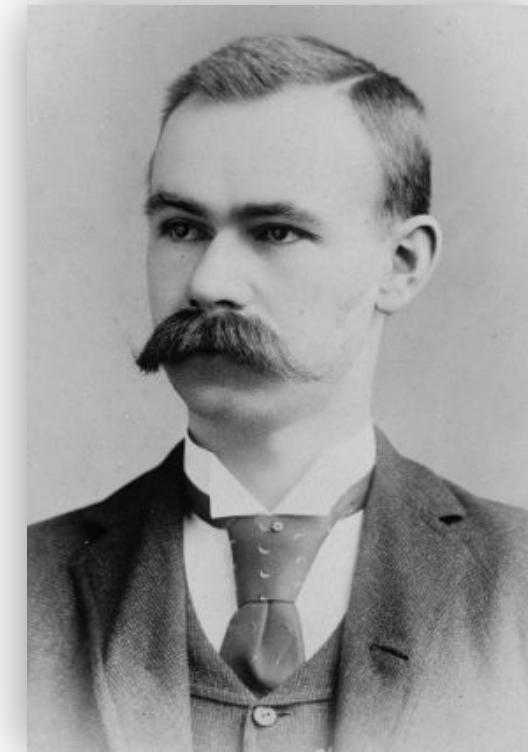
Historia de las bb.dd.



Edgard Codd, creador del modelo de bb.dd. relacionales (1963)

En nuestro aprendizaje de MongoDB, hicimos una introducción al mundo de las bb.dd. y cómo se inició esta necesidad de almacenar la información de forma efectiva.

Tanto *Edgard Codd* (izquierda), como *Herman Hollerith* (derecha), fueron los grandes responsables de concientizarnos y darnos herramientas efectivas, para poder desarrollar con el tiempo, el universo de las bases de datos. A pesar de la distancia en el tiempo, ambos fueron grandes influenciadores de cuál era la forma más efectiva de ordenar la información.



Herman Hollerith, creador de la máquina perforadora de tarjetas (1884)



Historia de las bb.dd.

Y más allá de que el modelo de bb.dd. relacionales prosperó efectivamente, el mismo sufrió muy pocos cambios a lo largo de su vida.

(1977 *Oracle Database* - Actualidad)

Al día de hoy, las bb.dd. relacionales siguen siendo el mecanismo de almacenamiento más efectivo y el más elegido para el modelo de negocios general que abordan la mayoría de las corporaciones en todo el mundo.



Historia de las bb.dd.

Como vimos oportunamente, SQL es un estándar en el mundo de las bb.dd., comandado por ANSI SQL y, a partir del mismo, se desprenden “*diferentes sabores*” de bases de datos, conocidos con el branding de la empresa de software que los representa.

Actualmente existen decenas de bb.dd. que utilizan SQL como lenguaje. De toda la variedad de sabores que existen, solo dos empresas concentran el mayor porcentaje de uso en el mercado corporativo.



Historia de las bb.dd.

Mercado de bb.dd. SQL



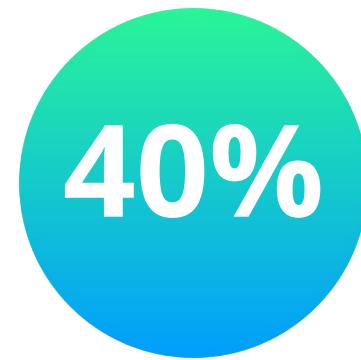
Oracle



MySQL



MS SQL Server



Microsoft Access

PostgreSQL

MongoDB

Firebase

SAP

Teradata

MariaDB

...

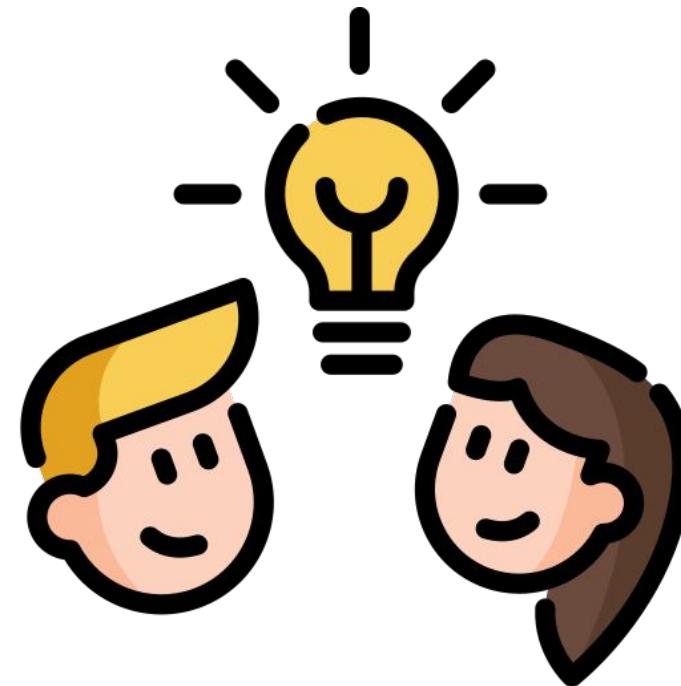


Historia de las bb.dd.

Oracle Corporation adquirió Sun Microsystems, la compañía que era propietaria de MySQL AB, el 27 de enero de 2010. Por lo tanto, se puede decir que Oracle compró MySQL en el año 2010.

Esto nos lleva a aunar a Oracle, en el gráfico anterior, como la empresa que maneja el 47% del mercado de bb.dd. SQL.

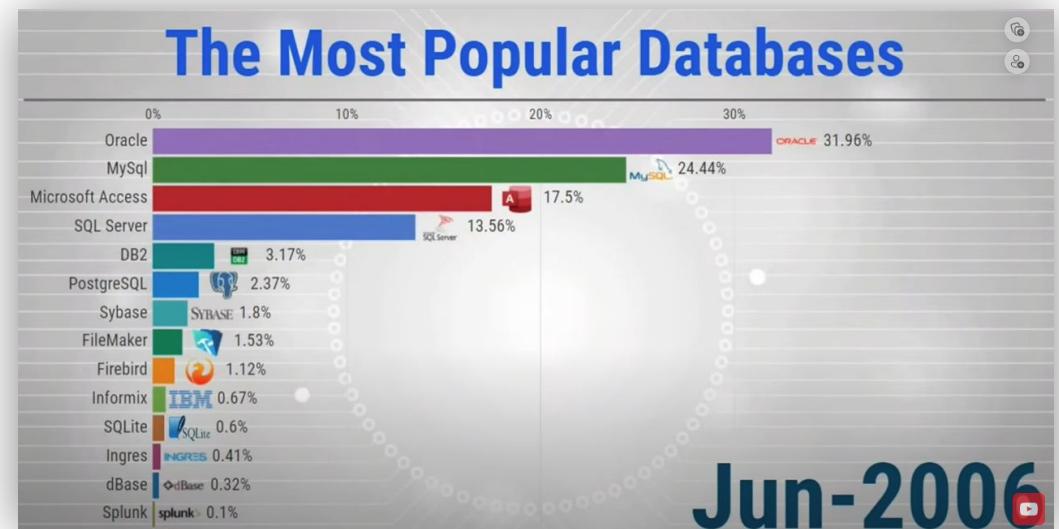
En el momento de la compra, MySQL era uno de los sistemas de gestión de bases de datos más populares del mundo.



Historia de las bb.dd.

En la imagen contigua agregamos un link para visualizar un video donde vemos cómo se movió el mercado de bb.dd entre Junio de 2006 y 2021.

Como podemos apreciar, los tres principales motores mencionados anteriormente, varían muy poco año tras año.

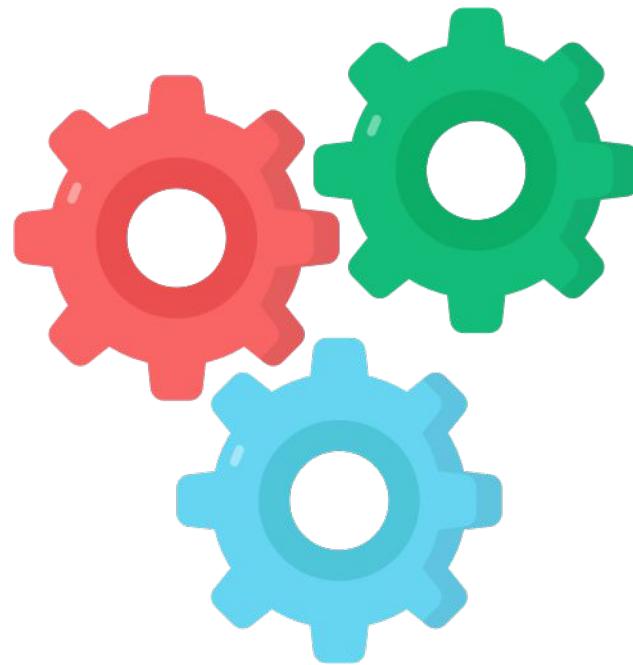


Fundamento de las bb.dd Relacionales

Fundamento de las bb.dd Relacionales

Las bases de datos relacionales se basan en el **Modelo Relacional usando N cantidad de tablas** para representar, tanto los datos, como las relaciones entre estos.

Es poco frecuente encontrar casos de una bb.dd. con una sola tabla pero, en el caso de que se dé esta situación, se le denomina como Base de datos Plana.



Fundamento de las bb.dd Relacionales

BASE DE DATOS PLANA

PROVEEDOR	FECHA	PEDIDO	IMPORTE PAGADO
MAYORISTA DE KIOSCOS S.A.	21/3/2021	1,234	1456,44
BEBIDAS DON PEDRO S.L.	30/3/2021	2,233	2451,87
MAYORISTA DE KIOSCOS S.A.	4/4/2021	1,299	2021,59
MAYORISTA DE KIOSCOS S.A.	10/4/2021	1,344	2163,71
BEBIDAS DON PEDRO S.L.	21/4/2021	2,398	3122,81
MAYORISTA DE KIOSCOS S.A.	22/4/2021	1,400	2022,77

Este es un ejemplo de cómo se suele estructurar la información en una base de datos plana.

BASE DE DATOS RELACIONAL

COMPRAS				
PROVEEDOR	FECHA	PEDIDO	IMPORTE PAGADO	
1	21/3/2021	1,234	1456,44	
2	30/3/2021	2,233	2451,87	
1	4/4/2021	1,299	2021,59	
1	10/4/2021	1,344	2163,71	
2	21/4/2021	2,398	3122,81	
1	22/4/2021	1,400	2022,77	
3	30/4/2021	8,422	3002,41	

PROVEEDORES	
CODIGO	NOMBRE
1	MAYORISTA DE KIOSCOS S.A.
2	BEBIDAS DON PEDRO S.L.
3	MINUTAS Y POSTRES S.A.
4	GASEOSAS Y AGUAS S.R.L.

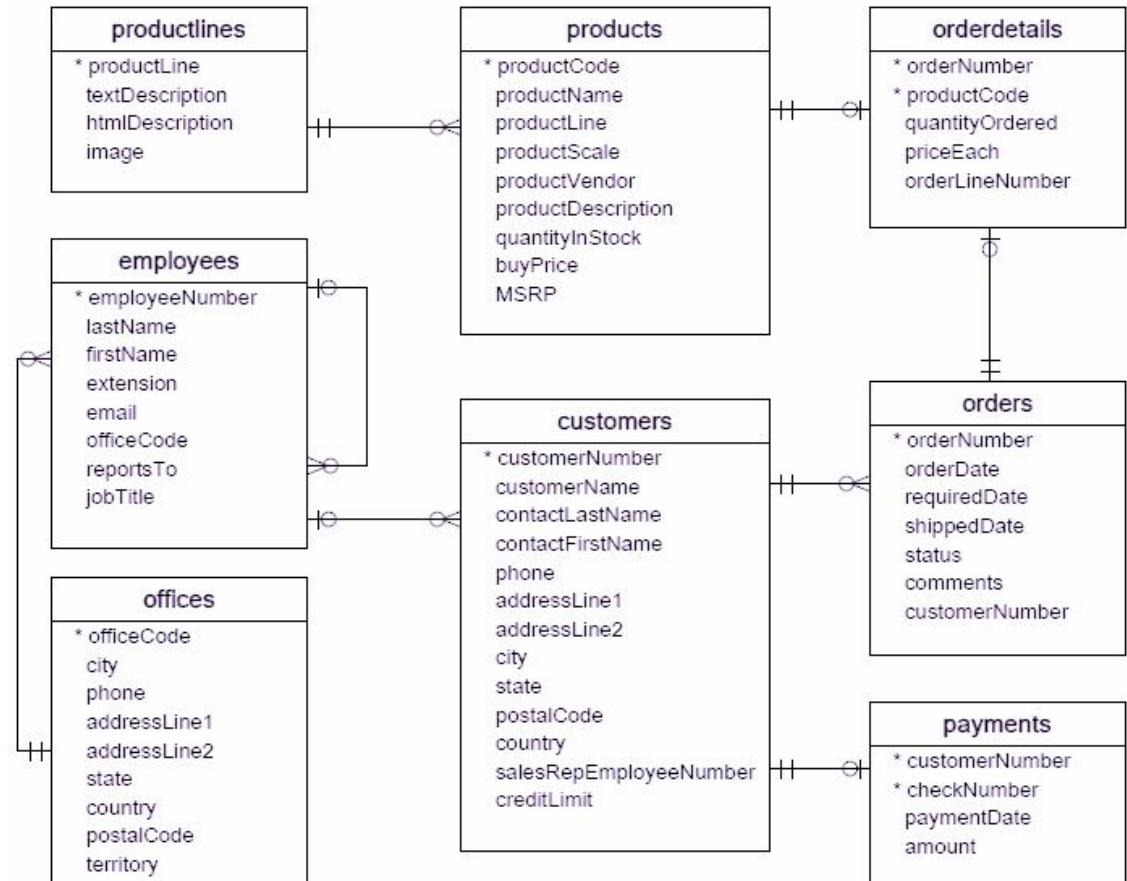
Para transformar la base de datos plana en una base de datos relacional, esta sería una forma mucho más correcta de “normalizar” la información que representamos.



Fundamento de las bb.dd Relacionales

Del modelo de bases de datos relacionales, se desprende también el **Esquema de una bb.dd.** (*Database Schema*)

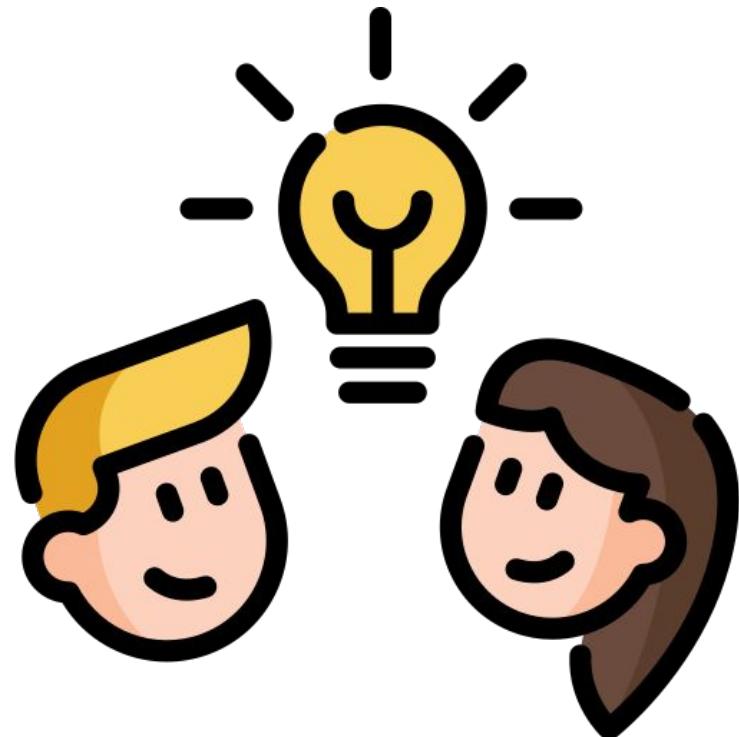
El mismo describe la estructura de una bb.dd. en un lenguaje formal soportado por un RDBMS. Este esquema define sus tablas, campos de c/tabla y relaciones entre cada campo y cada tabla. (fuente: Wikipedia)



Fundamento de las bb.dd Relacionales

Tal como nos muestra el ejemplo anterior, una **bb.dd. relacional** nos permitirá, entre otras cosas, **aumentar la eficiencia de la información** almacenada en ésta, como también **reducir su espacio** de manera considerable.

Esto último lo notaremos con el correr del tiempo, a medida que la bb.dd. crezca en información.



Motores SQL

Motores SQL

Oracle Database es un sistema de gestión de bases de datos relacional (RDBMS) desarrollado y comercializado por Oracle Corporation.

Oracle es uno de los sistemas de bases de datos más populares y ampliamente utilizados en el mundo empresarial, y se utiliza para gestionar datos críticos en una variedad de aplicaciones y entornos de negocio.



Motores SQL

Algunas de las características principales de Oracle incluyen:

- Arquitectura Client - Server
- Lenguaje SQL
- Escalabilidad
- Seguridad
- Alta disponibilidad
- Soporte de múltiples plataformas



Motores SQL

Microsoft SQL Server es otro sistema de gestión de bases de datos relacional (RDBMS) muy popular que se utiliza en una variedad de entornos empresariales.

Es una solución completa y robusta para la gestión de bases de datos empresariales, con una amplia variedad de características y opciones de configuración para satisfacer las necesidades de los usuarios.



Motores SQL

Algunas de sus características principales incluyen:

- Arquitectura Cliente - Servidor
- Lenguaje Transact SQL
- Escalabilidad
- Seguridad
- Alta disponibilidad
- Análisis de datos
- Integración con Productos MS



Motores SQL

MySQL es otro sistema de gestión de bases de datos relacional (RDBMS) muy popular que se utiliza en una variedad de entornos empresariales y web.

Si bien pertenece a Oracle Corp., aún sigue manteniendo su lógica de negocio por fuera de lo que es la marca comercial dueña.



Motores SQL

Algunas de sus características principales incluyen:

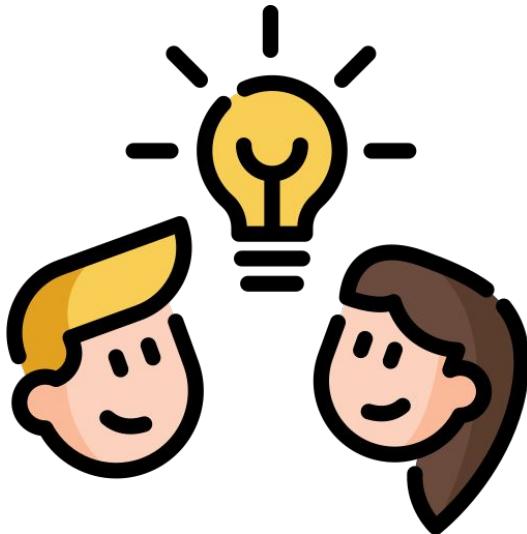
- Arquitectura Cliente - Servidor
- Lenguaje Transact SQL
- Escalabilidad
- Seguridad
- Alta disponibilidad
- Análisis de datos
- Integración con Productos MS



Motores SQL

Como podemos ver, los motores SQL más allá de su sabor, o branding, comparten prácticamente las mismas características.

Lo que más nos favorece a nosotras, como desarrolladoras de software es que, conociendo un motor SQL en profundidad, el salto hacia cualquier otro motor será con una curva de aprendizaje mínima, casi nula.



Cómo funciona un motor SQL

Cómo funciona un motor SQL

Cada base de datos SQL cuenta con un Motor.

Este se conforma por un componente de software que permite gestionar a la base de datos en sí.

Además, es el que se ocupa de las operaciones en general que se deben realizar sobre la BB.DD. para almacenar, consultar y/o eliminar información.



Cómo funciona un motor SQL

De igual forma que en lo que conocemos como “intérprete” en un lenguaje de programación, el motor de BB.DD. sería puntualmente este elemento.

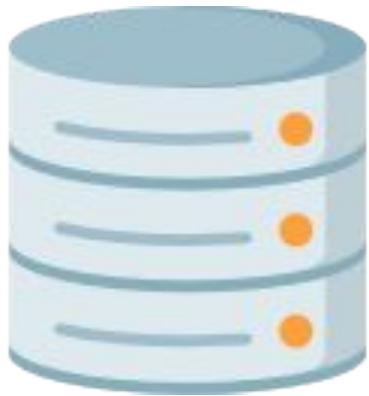
Entiende las instrucciones que ejecutamos sobre la bb.dd, la traduce a funcionalidades propias para gestionar de la forma más rápida posible la interacción con la información, y devuelve al gestor de la base de datos el resultado de la operación ejecutada.



Cómo funciona un motor SQL

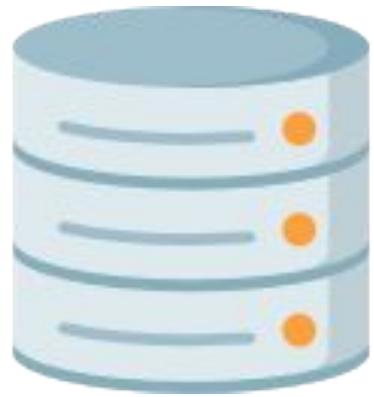
Los motores de las bases de datos cuentan con una optimización por demás destacable.

Son quienes manipulan la información a bajo nivel por lo tanto deben responder de la mejor forma posible a cada solicitud de operaciones.

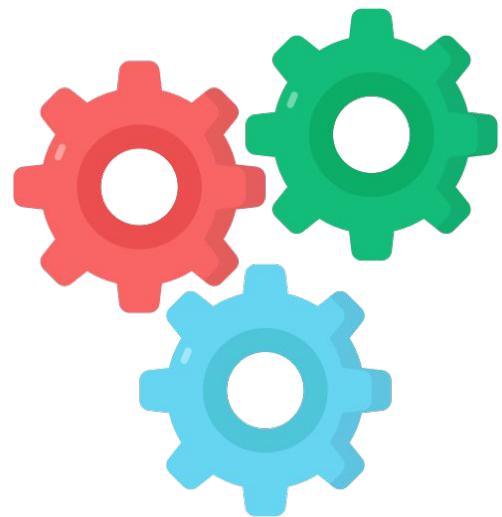


Cómo funciona un motor SQL

Composición de una bb.dd SQL



Motor SQL



Gestor de bb.dd

MySQL

MySQL

www.mysql.com

MySQL es la bb.dd. que utilizaremos en este curso. Respeta el estándar ANSI SQL, posee varias versiones, incluso gratuitas.

The screenshot shows the MySQL website homepage. The header includes the MySQL logo, a tagline "The world's most popular open source database", a search bar, and links for "Contact MySQL", "Login", and "Register". Below the header, there are navigation links for "MYSQL.COM", "DOWNLOADS", "DOCUMENTATION", and "DEVELOPER ZONE". A blue navigation bar below the header contains links for "Products", "Cloud", "Services", "Partners", "Customers", "Why MySQL?", "News & Events", and "How to Buy". The "Products" menu is expanded, showing options like "MySQL HeatWave", "MySQL Enterprise Edition", "MySQL Standard Edition", "MySQL Classic Edition" (which is highlighted with a blue background), "MySQL Cluster CGE", and "MySQL Embedded (OEM/ISV)". To the right, a section titled "MySQL Classic Edition" is displayed, featuring a brief description, a bulleted list of benefits (Lower TCO, Ease of Use, Low Administration, Supports over 20 platforms and operating systems), and a note about its availability to ISVs, OEMs, and VARs. A red "Get Started" button is visible at the bottom right.



MySQL

MySQL es una base de datos que permite el uso de diferentes motores de almacenamiento.

Esto ofrece mucha flexibilidad a las bases de datos para almacenamiento, pudiendo ajustar la performance del motor de acuerdo a la carga operativa que la bb.dd deba realizar.



MySQL

El motor MySQL posee internamente, soporte para diferentes tipos de motores. Entre ellos encontramos a:

- InnoDB
- MyISAM
- BerkeleyDB
- CSV
- NDB
- Federated Engine



MySQL

Más allá de toda esta variedad de motores, InnoDB y MyISAM son los más utilizados en sí.

Entre estos, la diferencia principal que podemos destacar es que MyISAM es más rápido a nivel lecturas e InnoDB es el más adecuado para aquellas bb.dd que deben estar todo el tiempo realizando transacciones.



RDBMS

MySQL

Como vimos anteriormente, los Sistemas de Administración de Bases de datos relacionales (*Relational Database Management System*) son aplicaciones de software con interfaz gráfica, que facilitan la gestión de un motor de base de datos relacional para operar sobre la información de ésta.



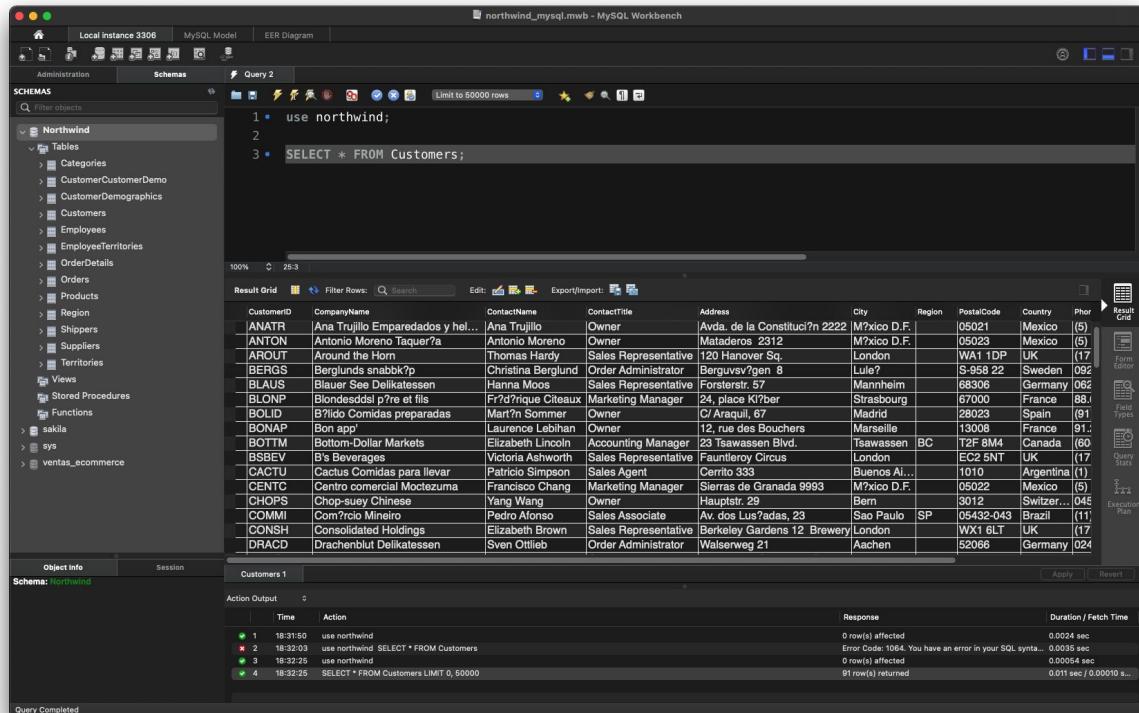
RDBMS - DBMS - SGBD - SGBDR

Estas últimas son las siglas con las que se suelen mencionar a estas aplicaciones de software.

MySQL

MySQL Workbench es el RDBMS que utilizaremos en este curso.

Cuenta con un montón de herramientas y funciona muy ágilmente en cualquier sistema operativo.



Sección práctica

Es momento de trabajar en nuestras computadoras, instalando MySQL (motor y gestor de bb.dd).

La profe te compartirá el link de descarga de MySQL desde la web oficial de esta base de datos.

- Debes elegir el instalador correspondiente a tu S.O.
- Descarga el mismo e inicia la instalación pertinente
- Consulta a la profe ante cualquier mensaje que te presente el proceso y no sepas cómo proceder



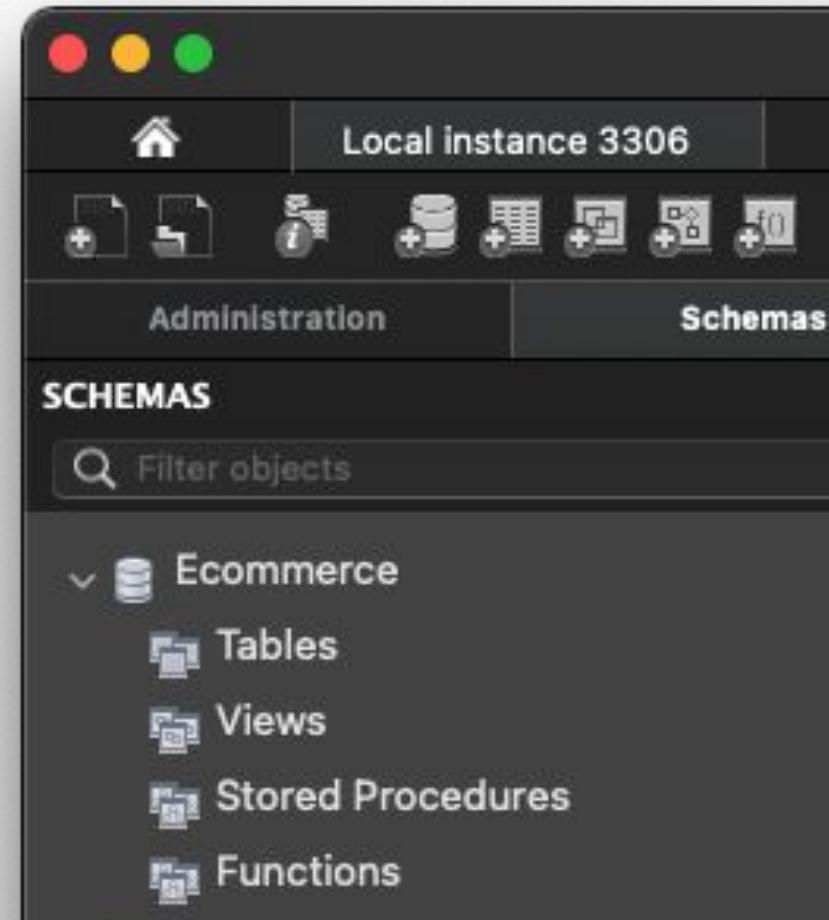
Tiempo estimado: **15 minutos**

Objetos en una bases de datos

Objetos en una bb.dd

Cada base de datos que creemos, cuenta con una serie de objetos principales, que permiten estructurar su contenido. Estos son:

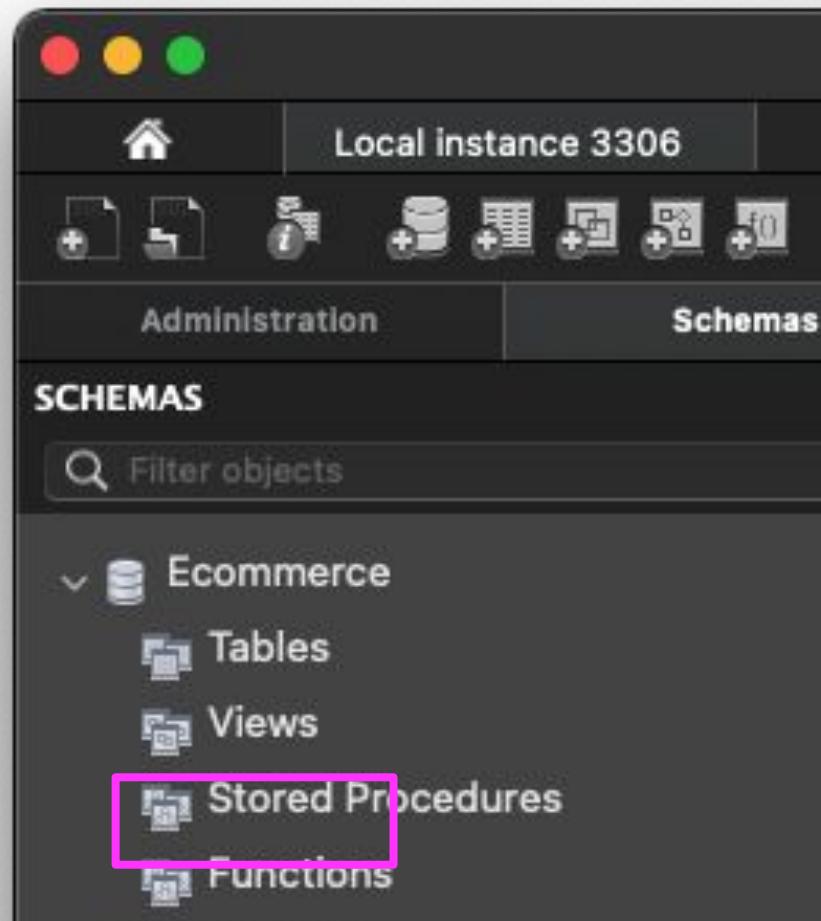
- Tablas
- Vistas
- Funciones
- Procedimientos Almacenados



Objetos en una bb.dd

Las tablas son el corazón de toda base de datos.

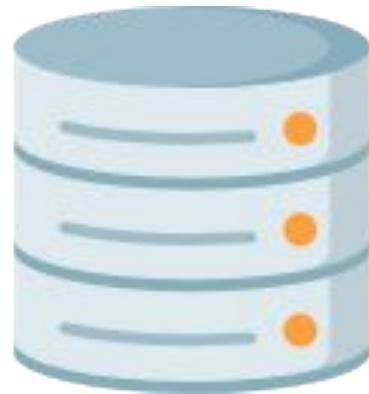
Allí se estructura y almacena la información general por lo tanto, cada una de las tablas, le darán vida al resto de los objetos que componen a la bb.dd.



Crear una base de datos

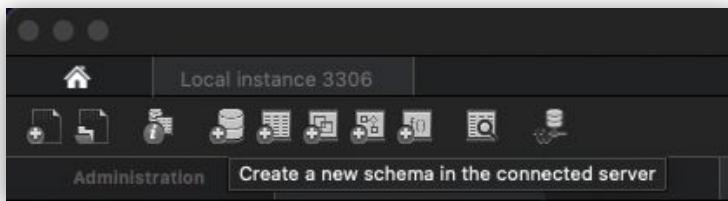
Crear una base de datos

Para entrar en contacto con el motor de bb.dd y el Administrador de base de datos relacionales MySQL Workbench, crearemos nuestra primera base de datos SQL.



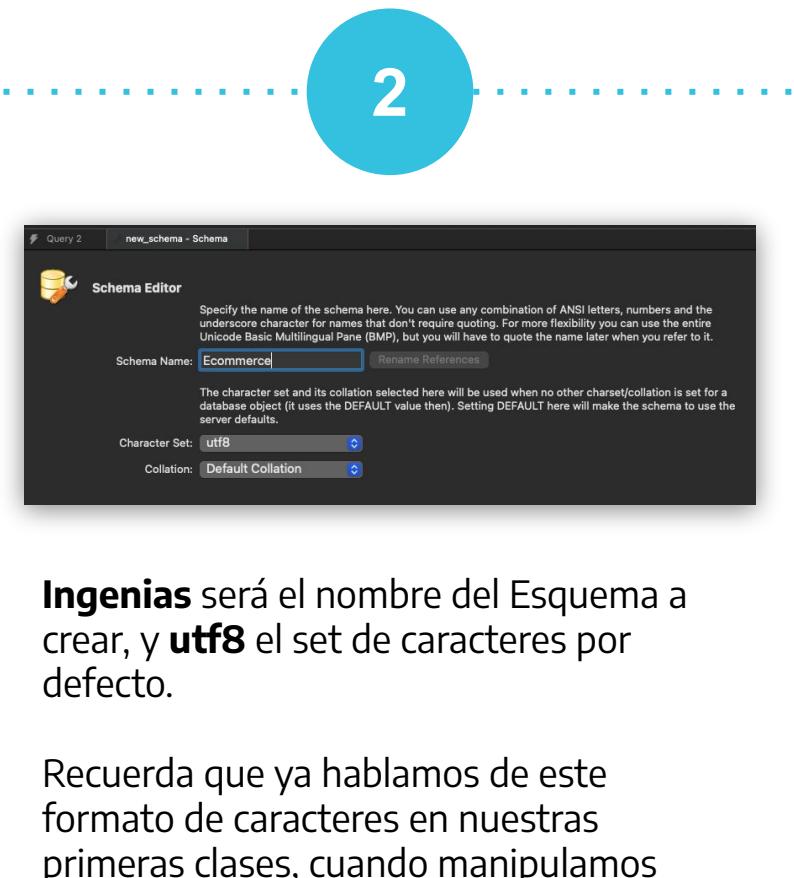
Utilizaremos a este último para realizar esta tarea de forma gráfica.

Crear una base de datos



Crearemos un nuevo **Esquema de BB.DD.** habiéndonos conectado previamente con **MySQL Workbench** al **Motor MySQL** instalado.

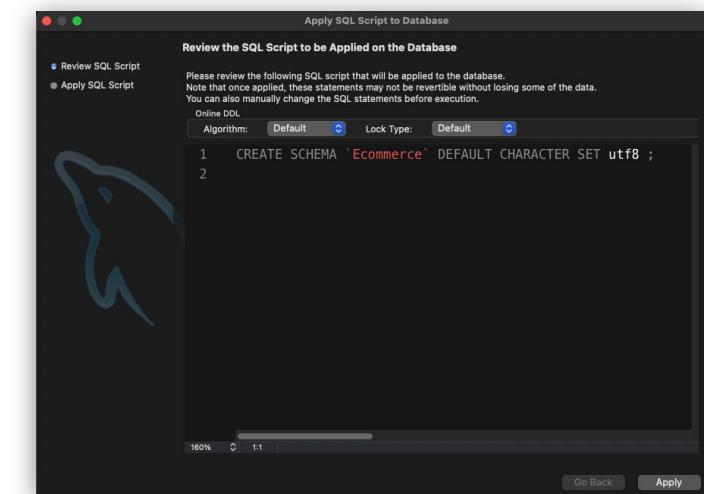
1



Ingenias será el nombre del Esquema a crear, y **utf8** el set de caracteres por defecto.

Recuerda que ya hablamos de este formato de caracteres en nuestras primeras clases, cuando manipulamos archivos mediante **el módulo 'fs'**.

Validamos el nombre de la base de datos y pulsamos el botón **Apply**.



3



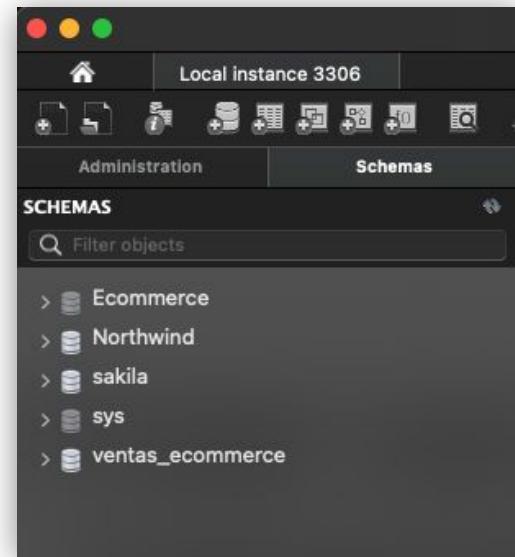
Crear una base de datos



Si todo fue bien, veremos el mensaje de confirmación por parte de MySQL.
Solo resta pulsar el botón Close.

1

2



En el apartado **SCHEMAS** veremos la nueva base de datos creada.



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



**Argentina
programa
4.0**



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

***primero
la gente***

Clase 18: Bases de datos Relacionales

Tablas y tipos de datos

Agenda de hoy

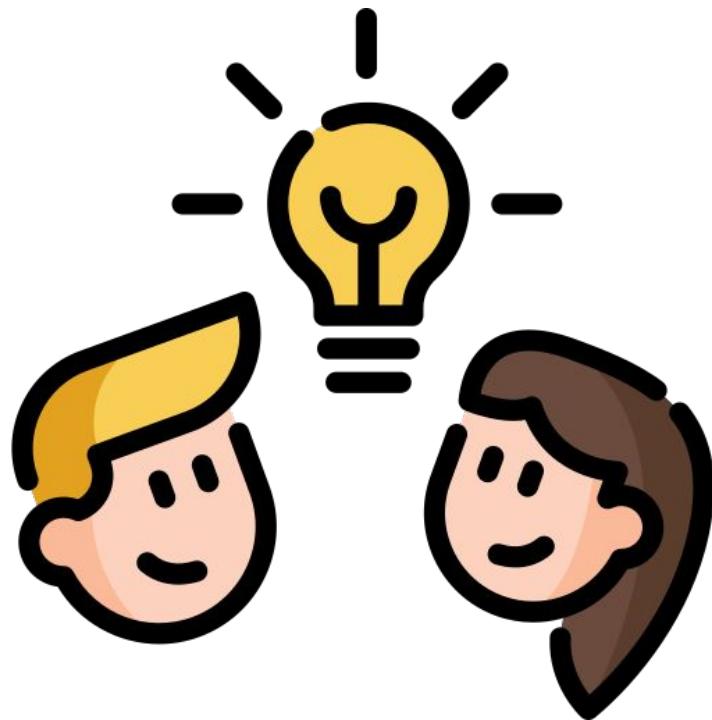
- A. Qué son las tablas
- B. Tipos de datos
- C. Estructurar la información
 - a. Columnas
 - b. Registros
- D. Crear una tabla
 - a. modificar una tabla creada
 - b. eliminar un campo
 - c. renombrar una tabla
 - d. Insertar registros
- E. La importancia de los índices



Qué son las tablas

Las tablas son objetos de la base de datos que contienen los datos organizados en filas y columnas.

Cada tabla tiene un nombre y está compuesta por campos que describen los datos que se almacenan en ella.



Qué son las tablas

Cada tabla almacena la información en forma de registros.

Para esto, respetan la estructura de cada dato que conforma un registro, el cual condice con la definición del campo que lo almacena.

7 -- TABLA PRODUCTOS

	id	nombre	existencia	precio	precio_compra
▶	1	TRACKPAD BLUETOOTH	1	22000	17600
▶	2	TRACKPAD INALÁMBRICO USB	1	22000	17600
▶	3	TECLADO INALÁMBRICO ES-BT	1	15000	10500
▶	4	TECLADO MECÁNICO CABLEADO ES	1	7500	5200
▶	5	TECLADO INALÁMBRICO ES-USB	0	12500	8700
▶	6	MOUSE INALÁMBRICO BT	1	6500	4500
▶	7	MOUSE CABLEADO	0	4100	2900
▶	8	MOUSE INALÁMBRICO USB	1	5500	3300
▶	16	UPS 10500 WATTS	1	15000	10500
▶	17	MOTHERBOARD AM4 - ASROCK A320M-HDV	0	6599	4619
▶	18	MOTHERBOARD AM4 - ASROCK B450M AC	0	9239	6467

Qué son las tablas



Qué son las tablas

Cada columna o campo, almacena un tipo de información específico. Este tipo de información se rige por lo que se denomina tipo de dato.

Como podemos apreciar en el gráfico, existen diferentes tipos de datos para identificar cada atributo que conforma finalmente un registro en la tabla.

id	nombre	existencia	precio	precio_compra
4	TECLADO MECÁNICO CABLEADO ES	1	7500	5200
5	TECLADO INALÁMBRICO ES-USB	0	12500	8700

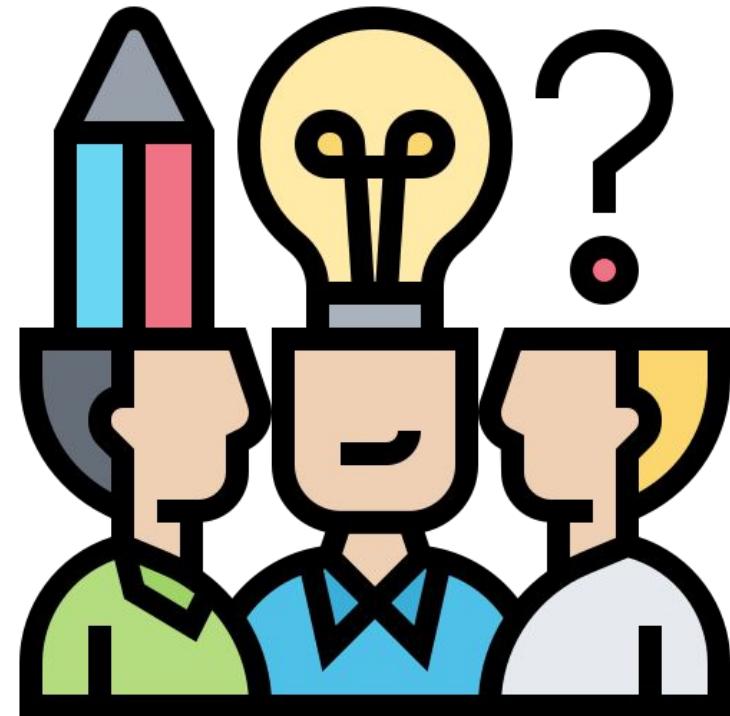
Diagrama que muestra una tabla con cinco columnas: id, nombre, existencia, precio y precio_compra. Se utilizan flechas para indicar el tipo de dato correspondiente a cada columna:

- La columna "id" apunta a un cuadro azul titulado "Numérico".
- La columna "nombre" apunta a un cuadro morado titulado "Texto".
- La columna "existencia" apunta a un cuadro gris titulado "Booleano".
- Las columnas "precio" y "precio_compra" apuntan a un cuadro rosado titulado "Monetario".

Qué son las tablas

En el ejemplo gráfico anterior apreciamos algunos de los diferentes tipos de datos que podemos encontrar en la información almacenada en una tabla.

Cada tipo de dato cumple un rol especial en sql, permitiendo no solo ahorrar espacio de almacenamiento, sino también etiquetar correctamente a la información a almacenar.



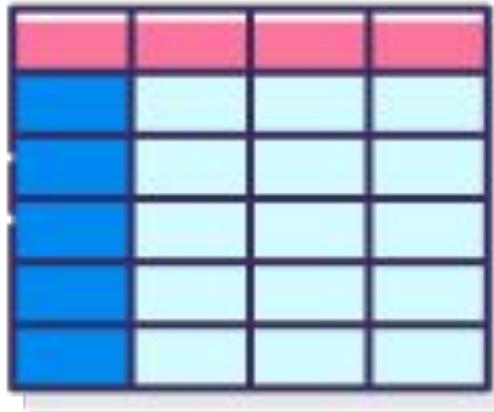
Tipos de datos

Tipos de datos

MySQL acepta varios tipos de datos SQL divididos en diferentes categorías. A continuación vemos las más comunes y utilizadas de forma frecuente:

- tipos numéricos
- fechas y horas
- tipos de cadenas de texto (caracteres y bytes)
- tipos de datos especiales

Entre otros.



Tipos de datos: cadenas de texto

- CHAR
- VARCHAR
- BINARY
- VARBINARY
- BLOB
- TEXT
- ENUM
- SET

REFERENCIA AL MANUAL OFICIAL:

<https://dev.mysql.com/doc/refman/8.0/en/string-types.html>



Tipos de datos: numéricos

- INT
- INTEGER
- SMALLINT
- TINYINT
- MEDIUMINT
- BIGINT
- DECIMAL
- NUMERIC
- FLOAT
- DOUBLE
- BIT

REFERENCIA AL MANUAL OFICIAL:

<https://dev.mysql.com/doc/refman/8.0/en/numeric-types.html>



Tipos de datos: Date/time

- DATE
- TIME
- DATETIME
- TIMESTAMP
- YEAR

REFERENCIA AL MANUAL OFICIAL:

<https://dev.mysql.com/doc/refman/8.0/en/date-and-time-types.html>



Tipos de datos: Spatial

Este tipo de información es más específica, para ser aplicada en ámbitos donde se deben guardar datos sobre objetos geométricos de dos y tres dimensiones.

Dentro de todos los tipos de datos que puede manejar Spatial, también soporta almacenar información del tipo geográfica, como ser **Latitud**, **Longitud**, **Altitud**; propios de las coordenadas de un GPS.

REFERENCIA DEL MANUAL:

<https://dev.mysql.com/doc/refman/8.0/en/spatial-types.html>

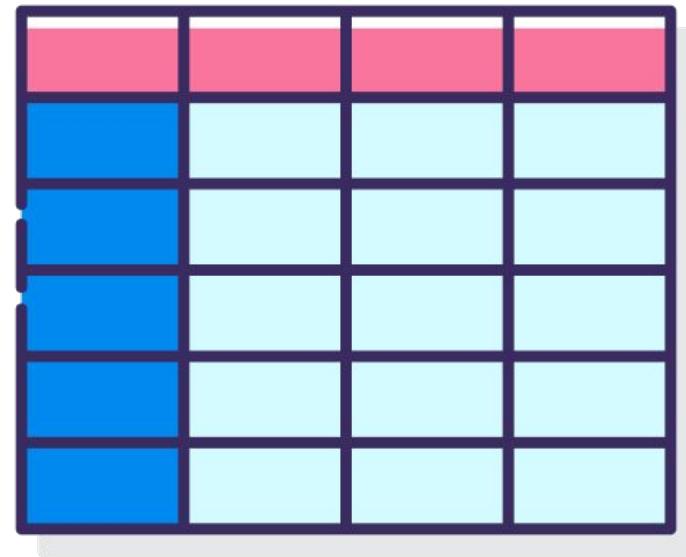


Crear tablas

Crear tablas

Al cierre de nuestro encuentro anterior creamos nuestra primera base de datos y, hasta ahora, conocimos los fundamentos de las tablas en una bb.dd, además de los tipos de datos que éstas pueden almacenar.

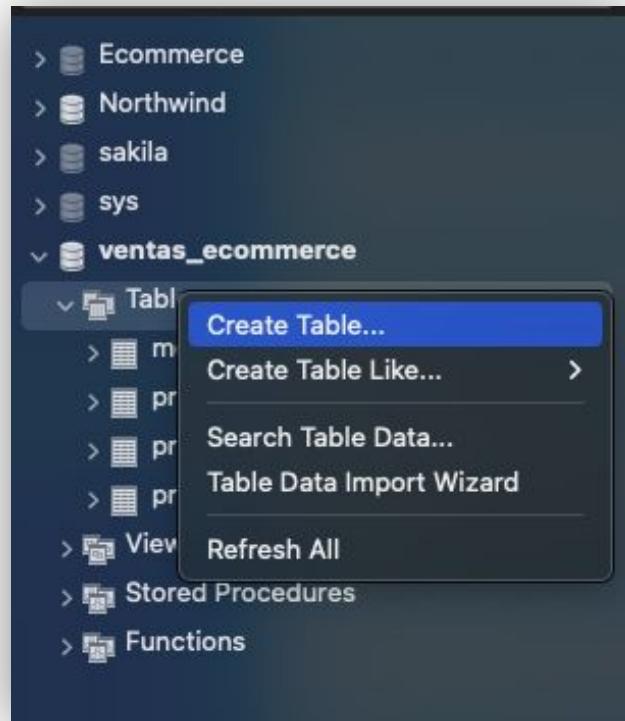
Veamos a continuación cómo crear una tabla aprovechando MySQL Workbench, y teniendo en cuenta el tipo de datos que deseamos guardar.



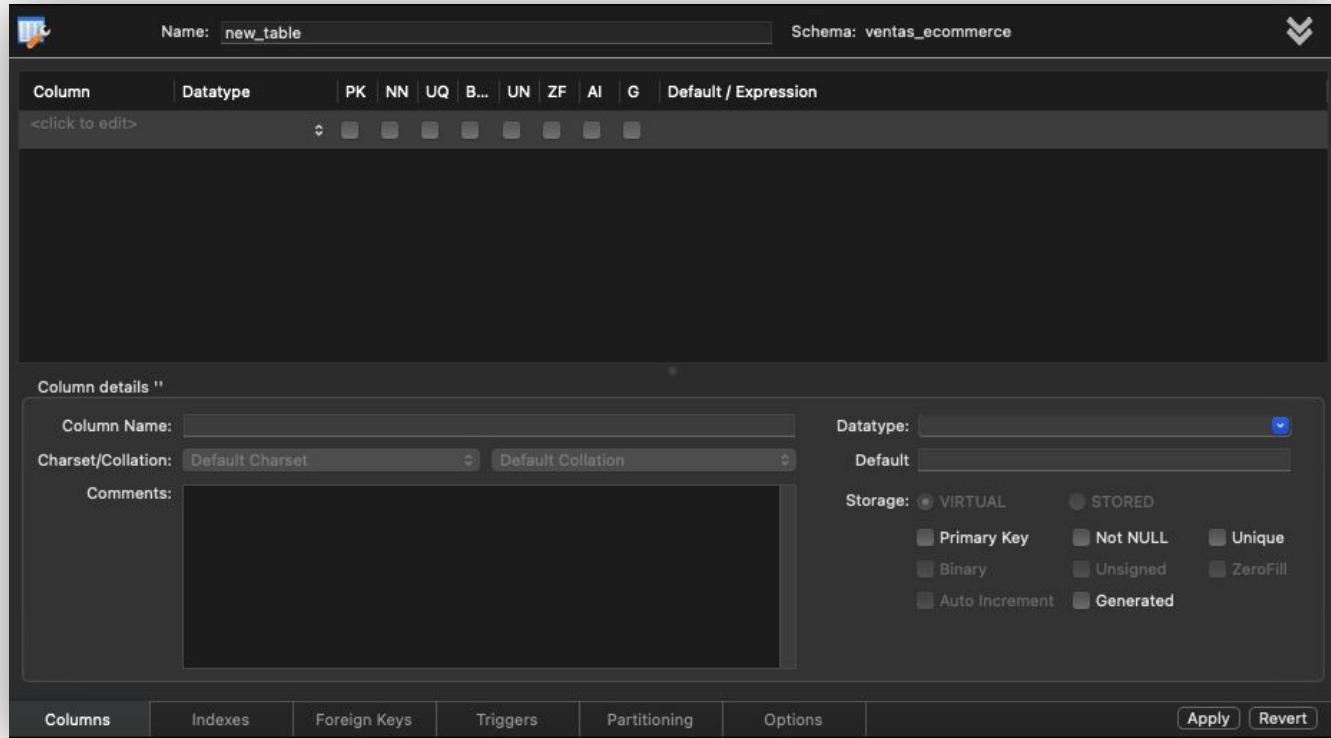
Crear tablas

Trabajaremos con esta base de datos, creada la clase anterior. Identifica la misma en el apartado **Schemas de MySQL Workbench** y despliega sus objetos.

Haz clic con el botón secundario del mouse. En el menú contextual que se despliega, selecciona el punto de menú **Create Table...**



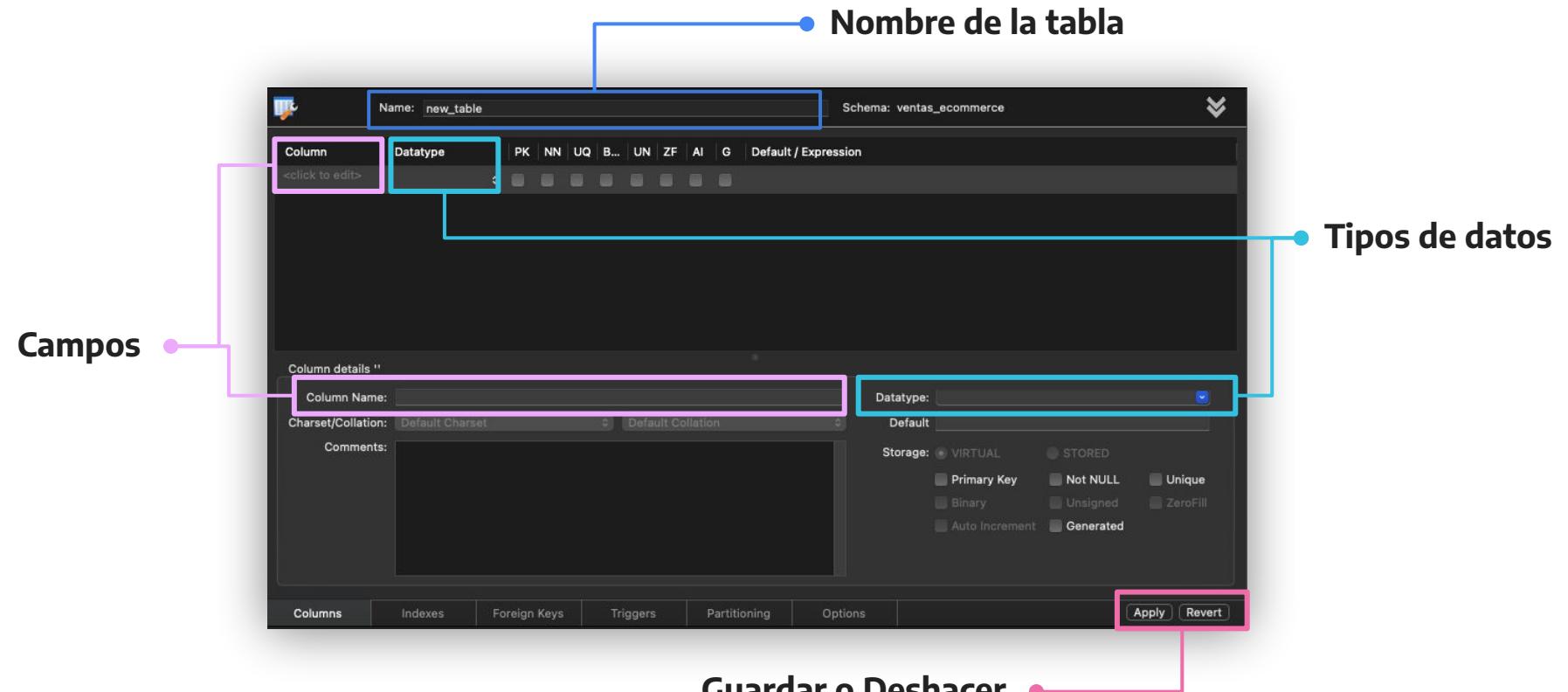
Crear tablas



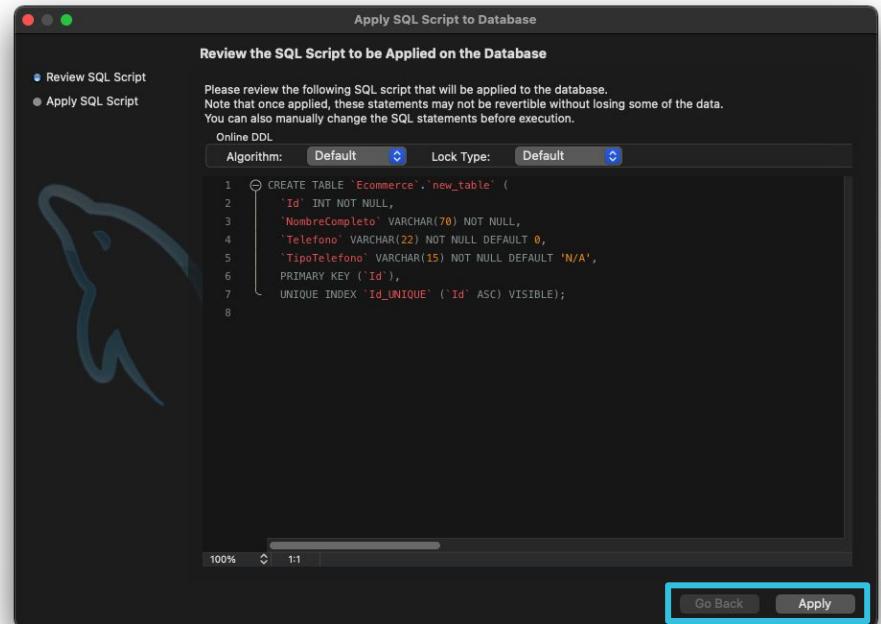
El editor de objetos integrado a MySQL Workbench se abrirá y ya está listo para que comencemos a agregar los campos (*o columnas*) a nuestra nueva tabla.

Crear tablas

Conozcamos el panel de creación de tablas de MySQL Workbench.



Crear tablas



La última pantalla nos permite confirmar la creación de la tabla, visualizando un resumen de esta en el lenguaje SQL.

Aplicar o Volver



Sección práctica

Trabajaremos sobre la base de datos que creamos en nuestro encuentro anterior. En ella crearemos nuestra primera tabla utilizando MySQL Workbench.

Ten presente que este ejercicio práctico nos permitirá seguir desarrollando un segundo ejercicio, por lo tanto, te recomendamos que lo hagas de cara a comenzar a adquirir experiencia en el terreno de MySQL.

Consulta con la profe todas aquellas dudas que te vayan surgiendo en el camino.

Prácticas

Creemos una tabla llamada **ContactosPersonales**, tomando como parámetros los campos a crear y sus atributos, declarados en la tabla.

Campo / Atributo	Tipo de dato	Características generales
Id	Entero	único, no nulo, autoincremental
NombreCompleto	Caracteres variables	50 caracteres, no nulo
Teléfono	Caracteres variables	12 caracteres, no nulo, 0 su valor predeterminado
TipoDeTelefono	Caracteres variables	10 caracteres, no nulo, 'N/A' su valor predeterminado

Tiempo estimado: **10 minutos**



Modificar una tabla creada

Modificar una tabla creada

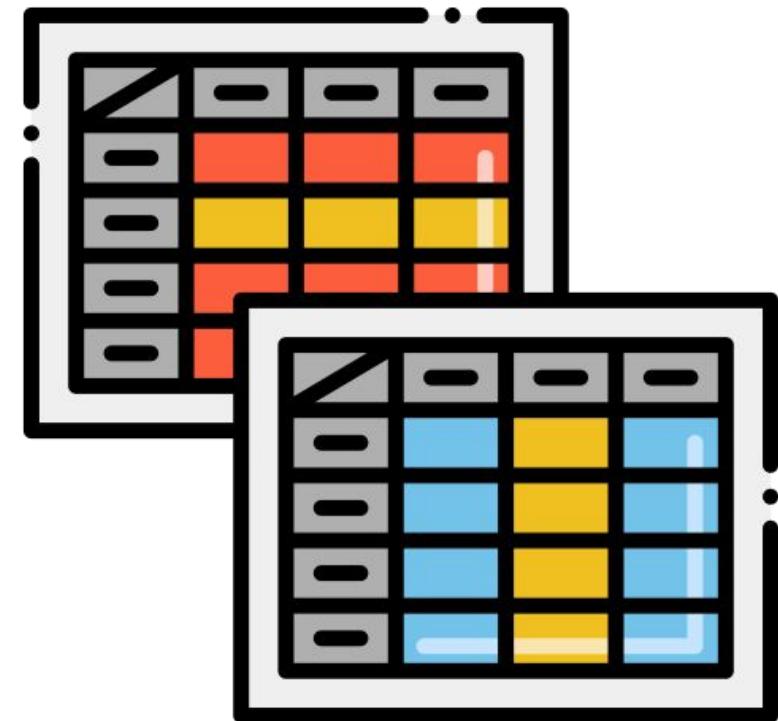
Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
Id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>						
NombreCom...	VARCHAR(70)		<input checked="" type="checkbox"/>							
Telefono	VARCHAR(22)		<input checked="" type="checkbox"/>							0
TipoTelefono	VARCHAR(15)		<input checked="" type="checkbox"/>							'N/A'
<click to edit>										

Si bien **MySQL Worbench** nos permite cambiar información de la estructura de un campo en una tabla, en este aspecto debemos tener presente (*siempre*), un montón de posibles variantes las cuales, a veces, no son tan perceptibles a simple vista.

Modificar una tabla creada

- No cambiar el tipo de datos de un campo existente
- Tipos de datos: podemos crecer pero nunca decrecer
- Verificar que los campos a modificar no sean Claves Primarias o Claves Foráneas
- Siempre tener presente el concepto de Integridad Referencial

Si bien, aún no vimos algunos conceptos de los mencionados aquí, la idea es que vayas captando las diferentes palabras técnicas para tenerlas en órbita de cara a nuestros próximos encuentros.



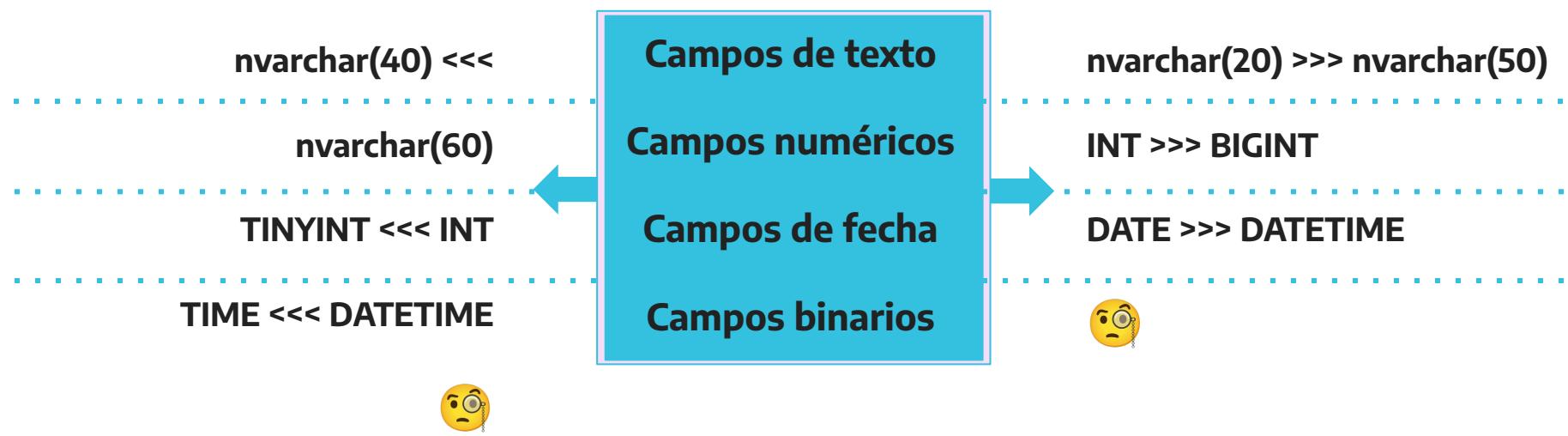
Modificar una tabla creada



Can't



Can



Prácticas

En la tabla creada, **modificaremos algunos datos sobre los campos creados**, de acuerdo a lo que se plantea en la tabla contigua.

Campo / Atributo	Tipo de dato	Características generales
Id	Entero	único, no nulo, autoincremental
NombreCompleto	Caracteres variables	70 caracteres , no nulo
Teléfono	Caracteres variables	22 caracteres , no nulo, 0 su valor predeterminado
TipoDeTelefono	Caracteres variables	15 caracteres , no nulo , 'N/A' su valor predeterminado

Tiempo estimado: **5 minutos**



Eliminar un campo



Eliminar un campo

- Nunca eliminar el campo de una tabla con datos

La única excepción que puede darse en este caso es cuando estamos diseñando una base de datos, y el Arquitecto encargado de dicho diseño, cambia la lógica de la misma.

Y, por supuesto, la base de datos no ha tenido siquiera un registro agregado.



Eliminar un campo

Los roles de Arquitectos de software, DBA's, Data Engineer, o Líderes técnicos, son generalmente quienes delinean la estructura principal de una base de datos.

Este proceso tiene un exhaustivo análisis y cuando llega el momento de crear tablas y campos, es porque dicho análisis será el definitivo.

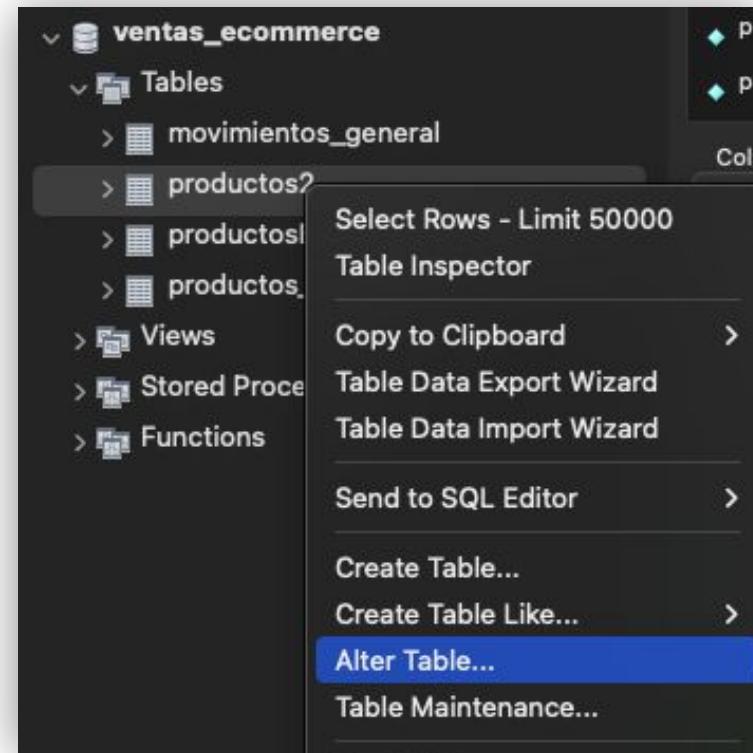
Muy rara vez puede darse la situación de que se creen campos y luego se busquen eliminar.

Renombrar una tabla

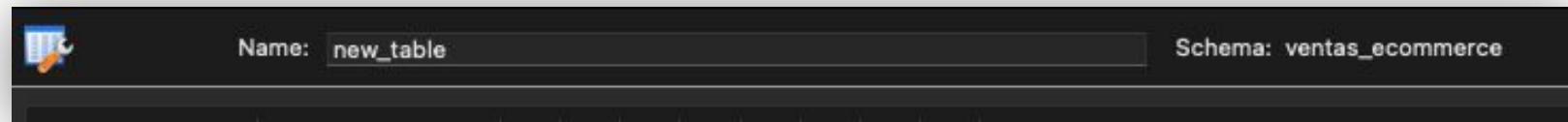
Renombrar una tabla

Muchas de las reglas mencionadas anteriormente también se aplican al cambio de nombre de una tabla.

En el caso de que no afecte en lo más mínimo a ningún contenido de la base de datos en cuestión, podemos cambiar su nombre accediendo a través del menú contextual, desde el botón secundario del mouse, al punto de menú **Alter Table...**



Renombrar una tabla



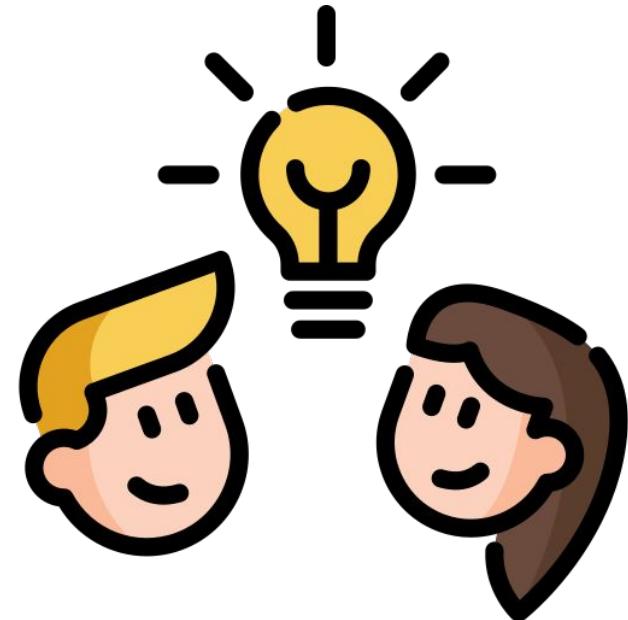
Si deseamos modificar el nombre de la tabla que creamos, a modo de práctica, podremos cambiar su nombre desde la ventana Asistente que se abre dentro de **MySQL Workbench**.

Realizado este proceso, pulsamos el botón **Apply**, para luego confirmar mediante la ventana emergente.

Eliminar un campo

**Existen restricciones al momento de renombrar una tabla,
de igual forma que con la necesidad de cambiar el tipo de
datos de un campo.**

Tengamos presente que, estas cosas sí se pueden realizar en MySQL Workbench y mientras estamos aprendiendo pero, **sobre bases de datos en producción, estas actividades No Se Deben Realizar, o se realizarán luego de un exhaustivo análisis sobre los posibles problemas que puedan surgir.**



Insertar registros

Insertar registros

Desde la representación de la tabla, tenemos la posibilidad de acceder no solo a visualizar su contenido, sino también a una barra de herramientas la cual nos permite realizar diferentes operaciones sobre la información de esta.

id	nombre	existencia	precio	precio_compra
1	TRACKPAD BLUETOOTH	1	22000	17600
2	TRACKPAD INALÁMBRICO USB	1	22000	17600
3	TECLADO INALÁMBRICO ES-BT	1	15000	10500
4	TECLADO MECÁNICO CABLEADO ES	1	7500	5200
5	TECLADO INALÁMBRICO ES-USB	0	12500	8700
6	MOUSE INALÁMBRICO BT	1	6500	4500
7	MOUSE CABLEADO	0	4100	2900
8	MOUSE INALÁMBRICO USB	1	5500	3800

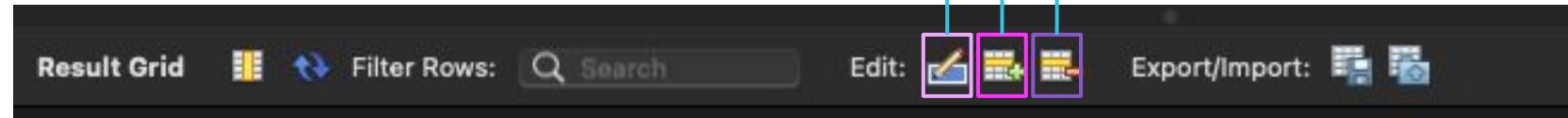


Insertar registros

Modificar una celda

Insertar un registro

Eliminar un registro



Confirmar / Deshacer

Apply

Revert

Prácticas

Insertaremos los siguientes registros en la tabla creada, para así comenzar a poblarla con datos efectivos.

NombreCompleto	Teléfono	Tipo de teléfono
Donna Clark	11-4455-0000	Móvil
Cameron Howe	11-4455-6688	Móvil
Gordon Clark	11-4455-0001	Móvil
Joe McMillian	11-4455-6677	(dejarlo vacío, sin datos)

Puedes dar de alta de a un registro e ir guardándolo, o dar de alta todos los registros juntos y aplicar los cambios de forma masiva.



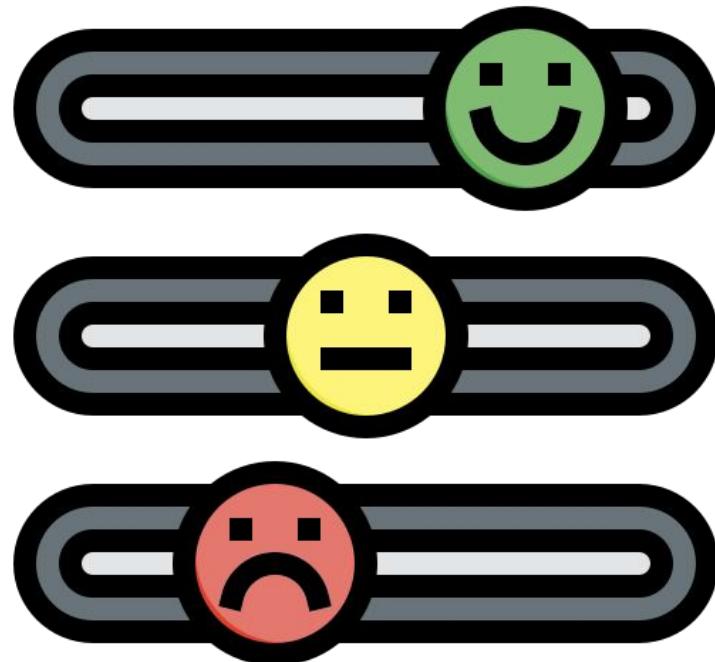
La importancia de los índices

La importancia de los índices

Un índice es una estructura de datos que se utiliza para mejorar el rendimiento de las consultas en una tabla de base de datos.

Un índice se suele crear en una o varias columnas de la tabla, y permite que la base de datos busque rápidamente los registros que coinciden con los valores de esas columnas.

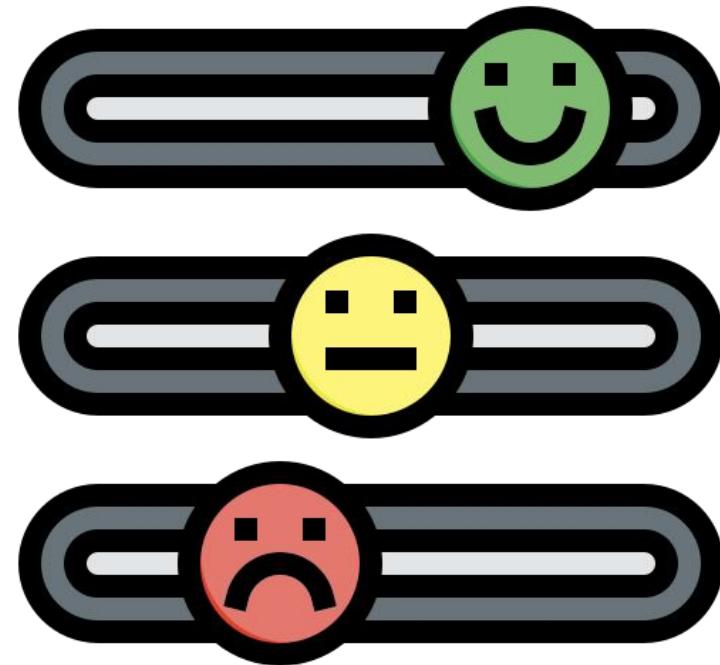
En nuestro ejemplo abordado, el índice de la tabla lo definimos mediante el campo **Id**.



La importancia de los índices

En lugar de tener que recorrer toda la tabla para encontrar los registros correspondientes, la base de datos puede utilizar el índice para buscar los registros de manera eficiente y reducir el tiempo de respuesta de las consultas.

El índice definido como **PRIMARY KEY** es el índice más importante de una tabla de base de datos. Garantiza la unicidad de los valores en una columna o en un conjunto de columnas en la tabla.

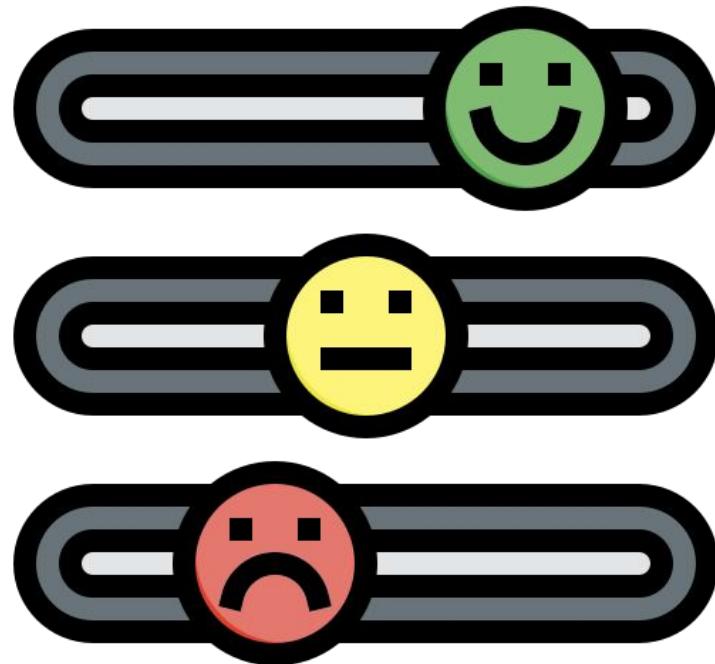


La importancia de los índices

Cuando se crea un índice PRIMARY KEY, la base de datos garantiza que no se puedan insertar registros duplicados en la tabla.

Además, un índice PRIMARY KEY se utiliza a menudo para definir las relaciones entre tablas en una base de datos relacional, ya que permite que otras tablas se relacionen con la tabla principal utilizando el valor de la clave principal.

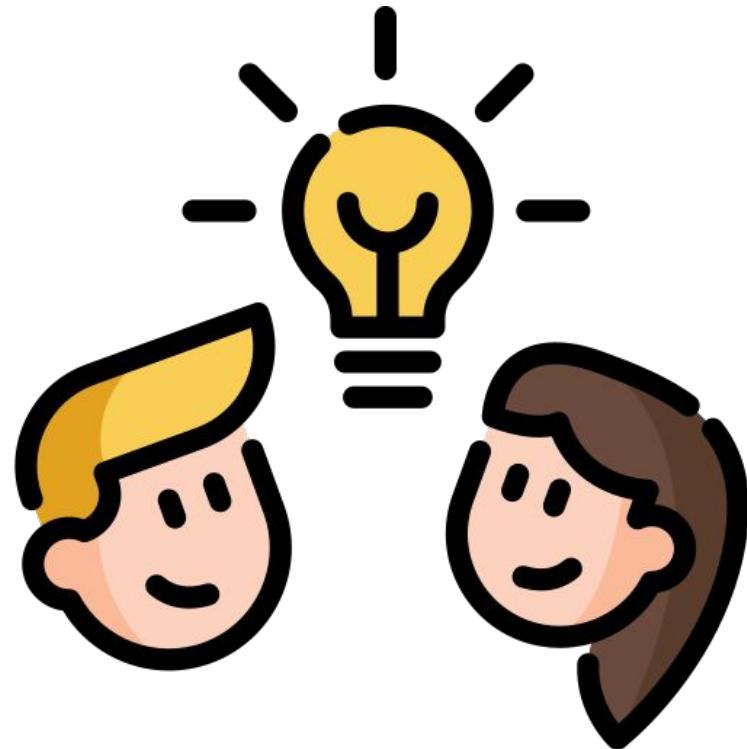
Lo veremos más en detalle cuando hablamos de relaciones entre tablas e integridad referencial.



La importancia de los índices

No es necesario que domines ya a la perfección el tema índices, pero sí que lo vayas teniendo presente.

En el correr de las unidades dedicadas a MySQL, iremos comprendiendo con mucho más detalle las diferentes aristas que componen el mundo de las bases de datos relacionales.



Lo que viene...

En nuestra próxima clase comenzaremos a trabajar con el lenguaje SQL puro, realizando consulta de datos insertados en la tabla, por lo tanto, te invitamos a que sigas agregando registros a la tabla que has creado.

Ganarás en práctica, conocerás mejor a MySQL Workbench, y tendrás muchas más información cargada para aprovechar en las próximas clases que tenemos por delante, donde manipularemos y filtraremos datos de tablas.

Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



Argentina programa 4.0



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*

Clase 19: Bases de datos Relacionales

El lenguaje SQL

Agenda de hoy

- A. EL LENGUAJE SQL
 - a. Subcategorías / Sublenguajes
 - i. DQL - DML
 - ii. DDL
 - iii. DCL
 - iv. TCL
- B. La cláusula SELECT
 - a. FROM
 - b. LIMIT
 - i. diferencias con SQL SERVER
 - c. ORDER
 - i. simple
 - ii. compuesto
- C. DISTINCT



El lenguaje SQL

Tal como venimos hablando desde el momento en el cual comenzamos a interactuar con bases de datos, el lenguaje SQL es un lenguaje de programación dedicado a trabajar con la manipulación de datos en general.

Este nació junto con la propuesta de Bases de datos Relacionales, en la década del 60', y se convirtió en un lenguaje regulado, regido por el Estándar ANSI SQL.



El lenguaje SQL

El proceso de establecer un Estándar para SQL surgió dado que entre los 70's y principio de los 80's comenzaron a abundar los motores de Bases de datos en el mercado de software, jactándose cada uno de ellos de utilizar el lenguaje SQL.

Dado que muchos lenguajes comenzaron a desviar la propuesta original del lenguaje en sí, SQL comenzó a ser regulado para que toda empresa de software que se jacte de dar soporte SQL en su base de datos, debiera cumplimentar con dicho Estándar.



El lenguaje SQL

El proceso de establecer un Estándar para SQL surgió dado que entre los 70's y principio de los 80's comenzaron a abundar los motores de Bases de datos en el mercado de software, jactándose cada uno de ellos de utilizar el lenguaje SQL.

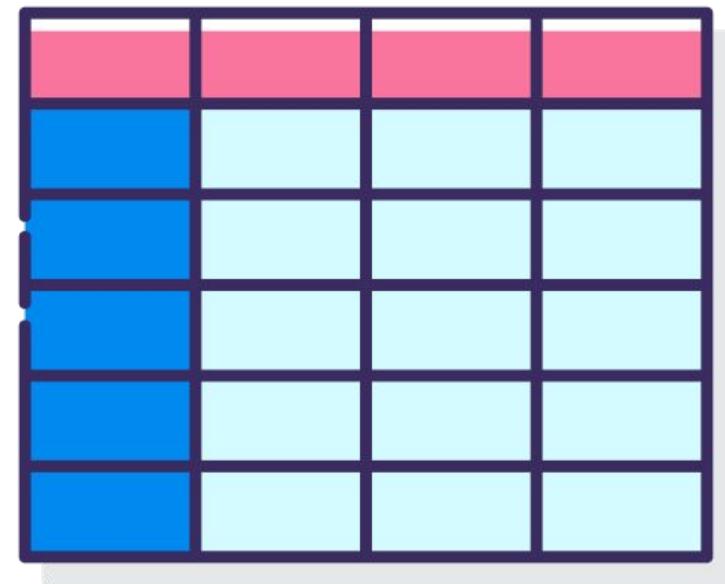
Dado que muchos lenguajes comenzaron a desviar la propuesta original del lenguaje en sí, SQL comenzó a ser regulado para que toda empresa de software que se jacte de dar soporte SQL en su base de datos, debiera cumplimentar con dicho Estándar.



El lenguaje SQL

Como vimos en nuestro encuentro anterior, SQL se apoya sobre el modelo de datos estructurados, almacenando el mismo en Tablas, y definiendo un tipo de datos “*tipado*”, para cada uno de los campos que conforman cada tabla.

Si bien SQL cuenta con otros tantos objetos dentro de una base de datos (*Vistas, Procedimientos Almacenados, Funciones Almacenadas*), las tablas son el punto de partida para que el resto de los objetos funcionen correctamente en torno a éstas.



El lenguaje SQL

Y toda la interacción, operaciones y manejo general de una base de datos, y de cualquiera de sus objetos, SQL lo logra utilizando el lenguaje SQL en sí.

Este es un lenguaje fácil y práctico de aprender. Su estructura está representada por una serie de instrucciones, o cláusulas, establecidas en formato imperativo, y definidas con un lenguaje muy cercano al idioma inglés.



El lenguaje SQL

Existen diferentes tipos de operaciones que este lenguaje nos permite realizar, podemos destacar a:

- Ejecutar consultas para recuperar datos
- agregar, modificar y/o eliminar registros
- crear nuevas bb.dd.
- crear nuevas tablas, vistas, funciones
- crear programas (*usando Stored Procedures*)
- modificar vistas de datos, funciones, tablas
- Definir permisos de uso de los objetos



El lenguaje SQL

Además, las diferentes operaciones que se pueden realizar en una base de datos SQL, están definidas bajo diferentes denominaciones regidas por siglas.

Estas siglas permiten identificar las diferentes funcionalidades que cada base de datos pone a nuestra disposición.

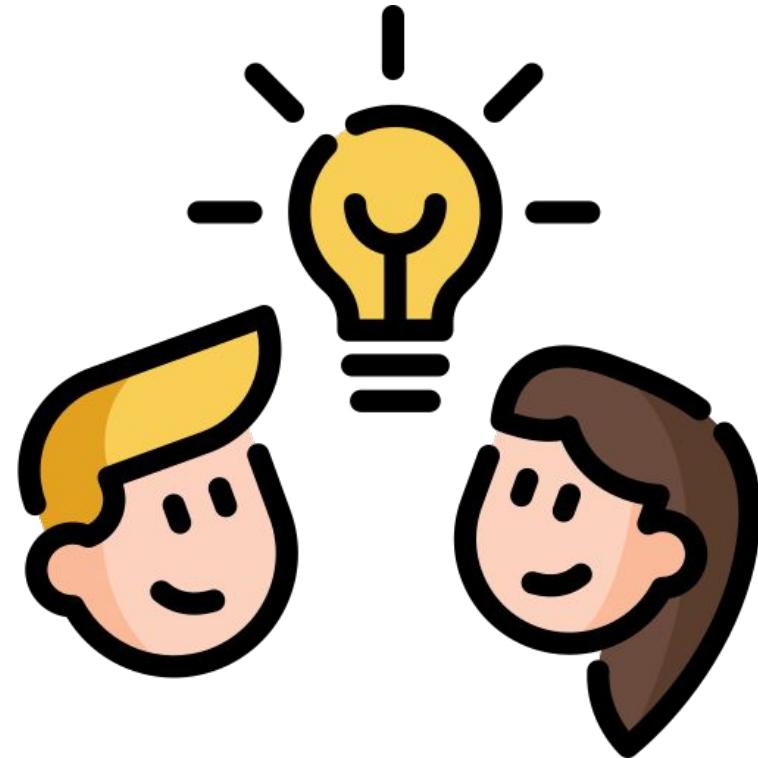
Las funcionalidades son denominadas Sublenguajes.



El lenguaje SQL

En resumen, todo lo que podemos hacer con mysql workbench también lo podemos hacer directamente con el lenguaje sql.

Existe una sentencia SQL para todo lo mencionado anteriormente, y más también. Si bien nos vamos a concentrar en la parte más importante, cubriremos gran parte del lenguaje base, que incluso se usará para operaciones o acciones más complejas.



Sublenguajes / Subcategorías

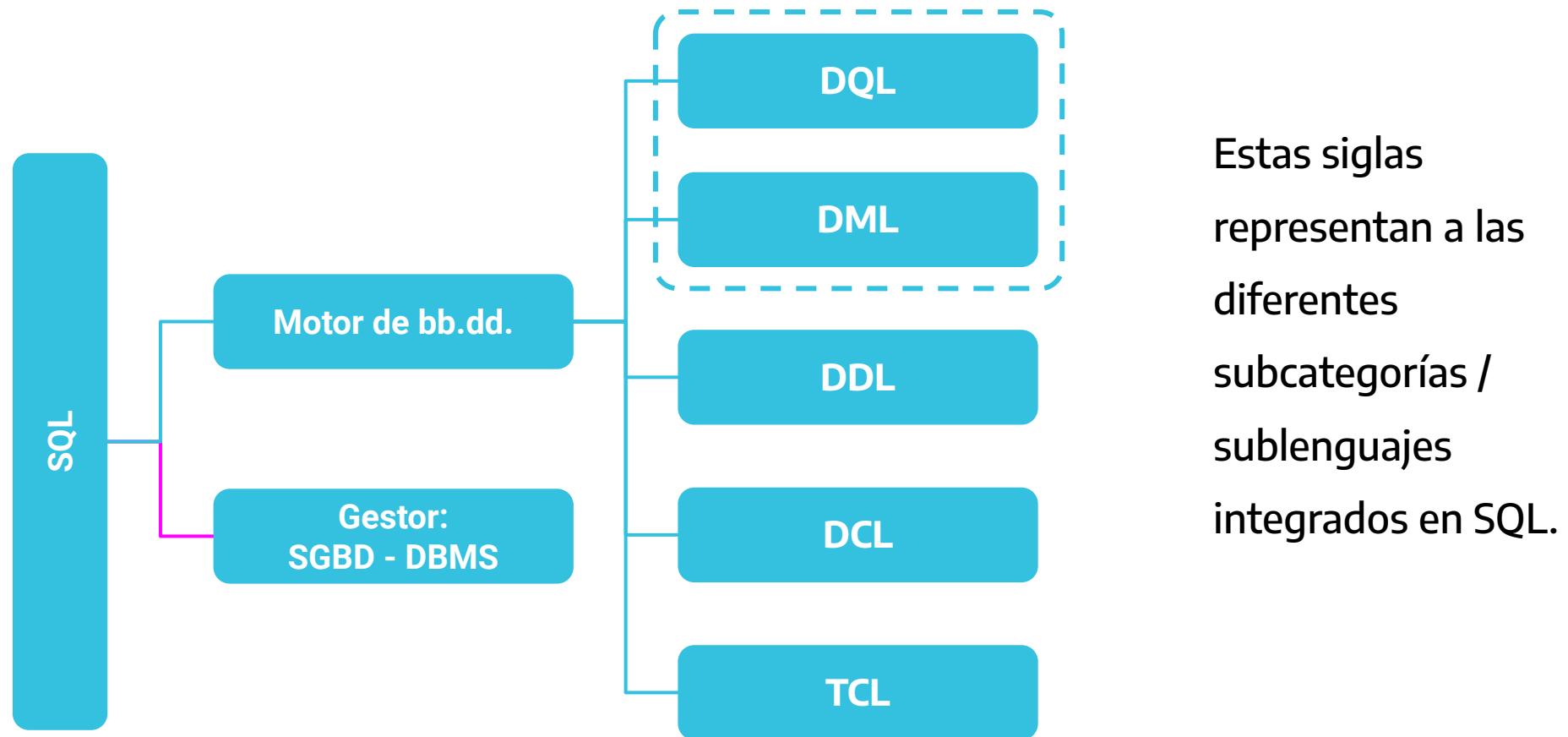
Sublenguajes / Subcategorías

Veamos a continuación un gráfico de una base de datos SQL, en donde representaremos todos los componentes que conforman a las mismas.

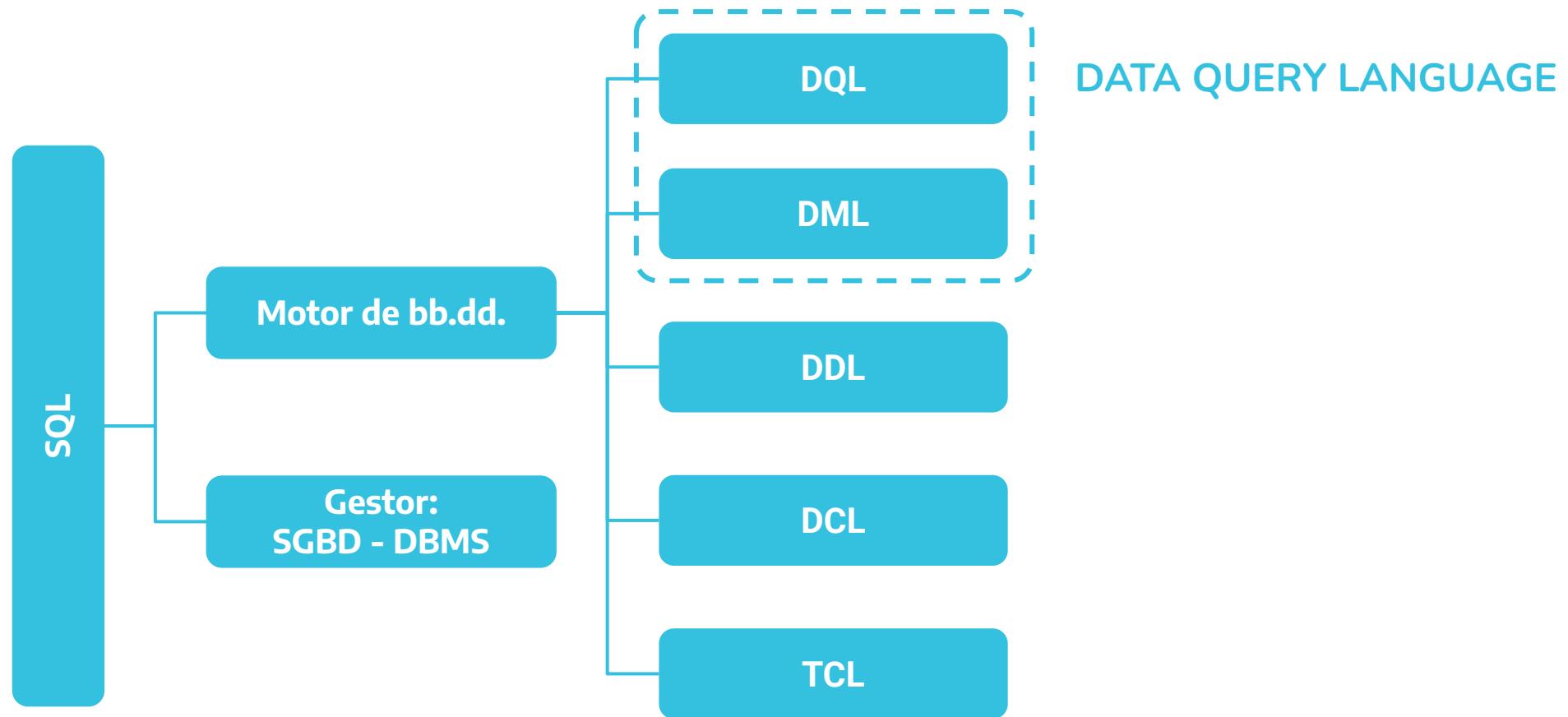
Cada sublenguaje, también denominados subcategorías, permiten ordenar rápidamente el tipo de operaciones que estas agrupan.



Sublenguajes / Subcategorías



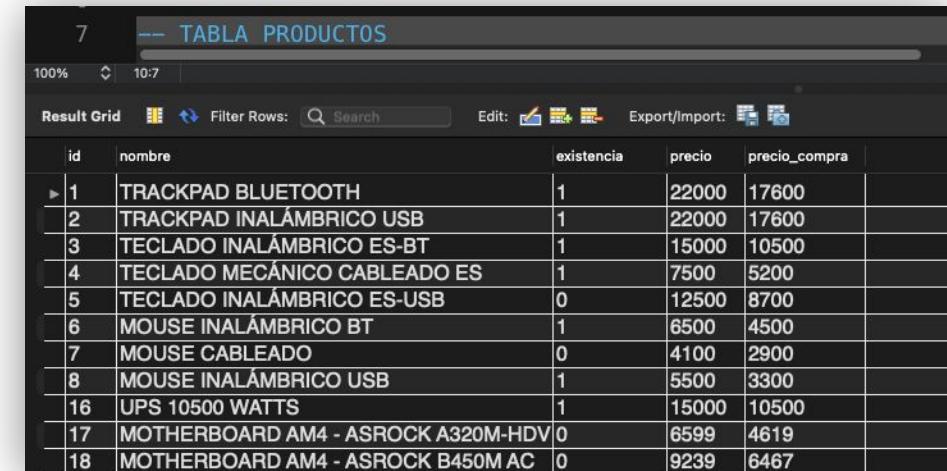
Sublenguajes / Subcategorías



Sublenguajes / Subcategorías

Hasta el momento, todo el manejo de información sobre una base de datos SQL lo realizamos a través de MySQL Workbench.

Seguiremos utilizando este gestor, tal como venimos haciendo hasta el momento, con la diferencia de que integraremos el sublenguaje DQL para solicitarle que realice operaciones mediante cláusulas SQL.



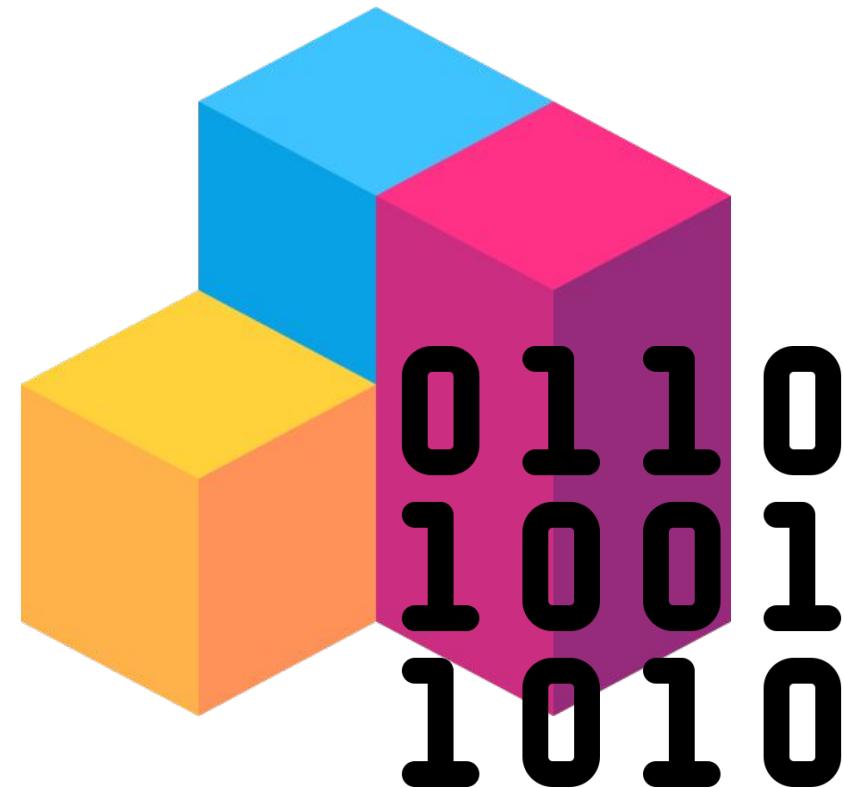
The screenshot shows a MySQL Workbench interface with a result grid titled "TABLA PRODUCTOS". The grid displays 18 rows of product data with columns: id, nombre, existencia, precio, and precio_compra. The data includes various computer peripherals like trackpads, keyboards, and mice, along with power supplies and motherboards.

	id	nombre	existencia	precio	precio_compra
▶	1	TRACKPAD BLUETOOTH	1	22000	17600
▶	2	TRACKPAD INALÁMBRICO USB	1	22000	17600
▶	3	TECLADO INALÁMBRICO ES-BT	1	15000	10500
▶	4	TECLADO MECÁNICO CABLEADO ES	1	7500	5200
▶	5	TECLADO INALÁMBRICO ES-USB	0	12500	8700
▶	6	MOUSE INALÁMBRICO BT	1	6500	4500
▶	7	MOUSE CABLEADO	0	4100	2900
▶	8	MOUSE INALÁMBRICO USB	1	5500	3300
	16	UPS 10500 WATTS	1	15000	10500
	17	MOTHERBOARD AM4 - ASROCK A320M-HDV	0	6599	4619
	18	MOTHERBOARD AM4 - ASROCK B450M AC	0	9239	6467

Sublenguajes / Subcategorías

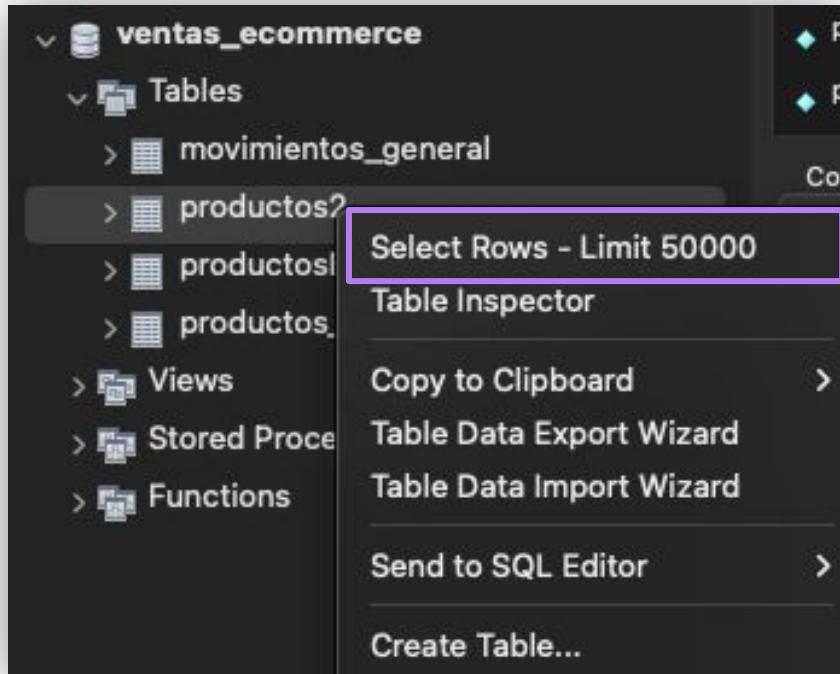
Una cláusula también se denomina “comando o sentencia”, son palabras reservadas del lenguaje, que permiten ejecutar acciones sobre la bb.dd.

Gracias a estas, nosotras como usuarias técnicas/ administradoras / programadoras, operaremos sobre la bb.dd. de acuerdo a la tarea que debamos realizar.



La cláusula SELECT

La cláusula SELECT



Cuando usamos MySQL Workbench para abrir una tabla y mirar su información, pulsamos sobre la opción **SELECT ROWS x000...**

Así se ejecuta la consulta sobre la tabla y visualizamos la misma en el entorno de trabajo de MySQL Workbench.

En la parte superior de la ventana de MySQL Workbench, veremos que se arma una consulta de selección, SQL.

La cláusula SELECT

Pongamos foco en la estructura; luego veremos cada palabra en detalle.

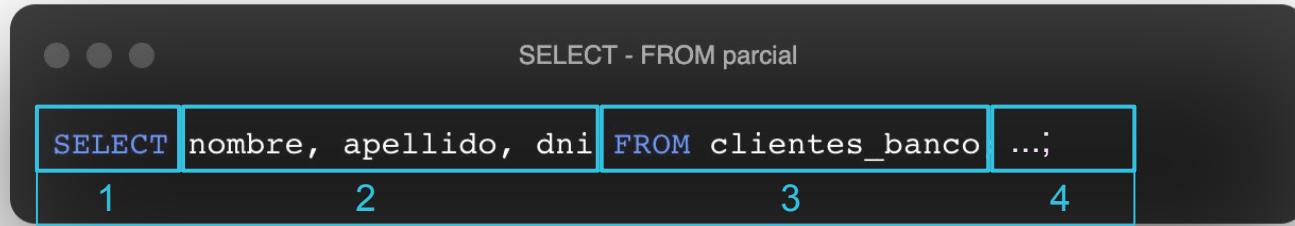
Para consultar determinados campos de una tabla:

```
...  
SELECT - FROM parcial  
  
SELECT nombre, apellido, dni FROM clientes_banco;
```

Para consultar todos los campos de una tabla:

```
...  
SELECT - FROM completo  
  
SELECT * FROM clientes_banco;
```

La cláusula SELECT



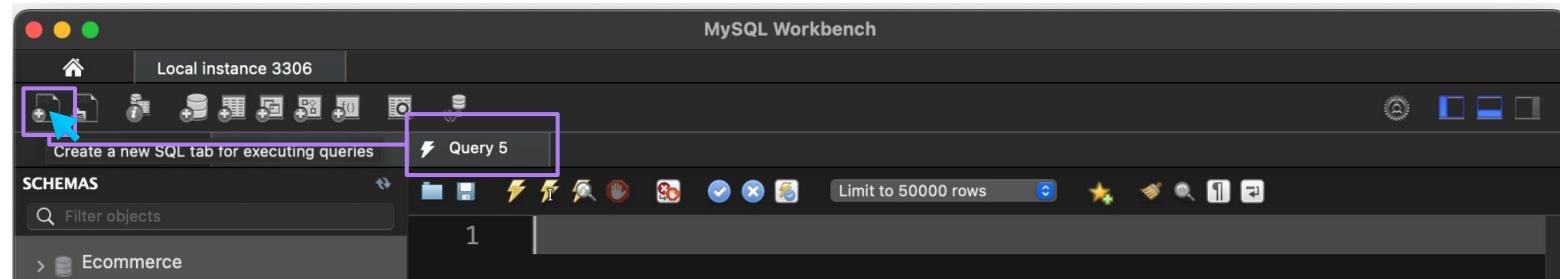
Las operaciones en SQL siguen una estructura que describe la operación que deseamos realizar. Si bien encontraremos consultas muy sencillas y otras más complejas, existen algunos elementos que aparecen con mayor frecuencia:

1. **Acción (keyword)**: crear, seleccionar, insertar, actualizar, eliminar, etc.
2. **Porción donde operaremos**: puede ser uno o más campos o un asterisco (*) para seleccionar todos los campos, o traer una o más tablas, etcétera
3. **Tabla a la cual queremos acceder**: la identificamos escribiendo el nombre
4. **Condiciones**: establecer criterios para operar sólo sobre los registros que los cumplan

La cláusula SELECT

- Las sentencias SQL no son sensibles a las mayúsculas y minúsculas.
No obstante, es importante respetarlas al colocar el nombre de un campo o tabla.
- Cada sistema de bases de datos tiene sus particularidades sintácticas. Sin embargo, si conocemos la base del SQL podremos adaptarnos sin dificultades.
- Cada consulta finaliza con punto y coma (;
- Previo a comenzar a escribir consultas en MySQL Workbench, debemos seleccionar la base de datos con la cual vamos a trabajar, a través de la cláusula **USE**.

La cláusula SELECT



En MySQL Workbench disponemos en la barra de herramientas superior, la opción de iniciar una nueva pestaña (*tab*) de SQL (*de script*) para ejecutar consultas.

Pulsemos el botón para crear una pestaña y comenzar a trabajar.

La cláusula **SELECT**



SELECT - FROM

```
SELECT * FROM tabla;
```

La sentencia SELECT, como lo indica su nombre, permite seleccionar información a extraer desde una tabla y visualizar la información resultante.

La cláusula **FROM** complementa a **SELECT**. Esta declara la tabla desde la cual se va a extraer la información.

La cláusula SELECT

```
...  
SELECT - FROM clientes
```

```
SELECT id, nombre FROM clientes;
```

Resultado de la consulta:

id	nombre
1	Marissa Mayer
2	Susan Wojcicki
3	Mira Murati
4	Melinda Gates

La cláusula SELECT



SELECT - FROM clientes campos según conveniencia

```
SELECT DNI, nombre, id FROM clientes;
```

Manejo de campos de una tabla:

El orden de los campos es irrelevante.

Podemos definir el que necesitemos en primer lugar, más allá de la posición donde éste haya sido definido cuando se construyó la tabla.

Al visualizar el resultado, el orden de los campos será tal como lo pedimos en la consulta.

ORDER BY

ORDER BY

Dentro de toda consulta SQL existe la posibilidad de aplicar un ordenamiento sobre la información a mostrar.

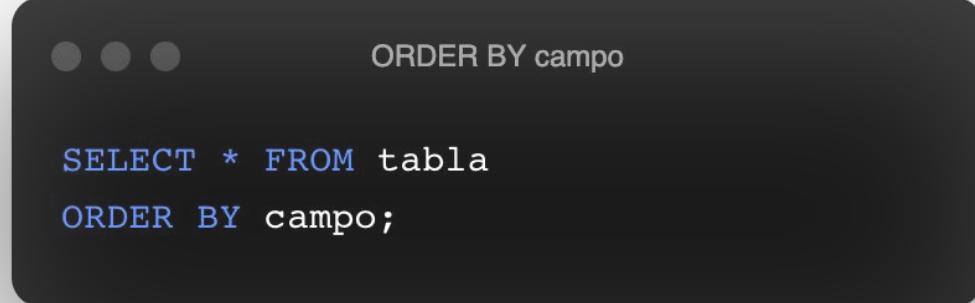
De acuerdo a nuestro propósito para con el tratamiento de la información, podemos listar la misma ordenándola por un campo específico.



ORDER BY

Para esto, debemos agregar a la consulta de selección, **la cláusula ORDER BY**.

La misma, se ubica al final de la consulta de selección, definiendo a continuación de esta cláusula, el campo en cuestión por el cual queremos ordenar los datos, previo a visualizarlos.



```
... ORDER BY campo  
SELECT * FROM tabla  
ORDER BY campo;
```

ORDER BY

En cuanto a ordenamiento refiere, es indistinto pedir uno, dos, o todos los campos en la consulta de selección y luego especificar en el ordenamiento, un campo que tal vez no necesitamos incluir en la consulta.

```
... ORDER BY nombre  
SELECT * FROM clientes  
ORDER BY nombre;
```

Tipo de ordenamiento

ORDER BY

También, dentro del ordenamiento de los datos, podemos especificar la forma en la cual queremos ordenarlos.

- Forma ascendente
- Forma descendente

Existen palabras reservadas para esta operación.

```
... ORDER BY campo ASC  
  
SELECT *  
FROM tabla  
ORDER BY campo ASC;
```

ORDER BY

Orden ascendente (opcional)

```
... ORDER BY nombre ASC
```

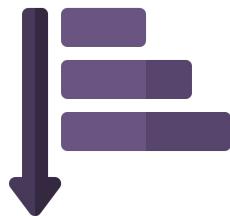
```
SELECT *
FROM clientes
ORDER BY nombre ASC;
```

Orden descendente (obligatorio)

```
... ORDER BY DNI DESC
```

```
SELECT *
FROM clientes
ORDER BY DNI DESC;
```

ORDER BY



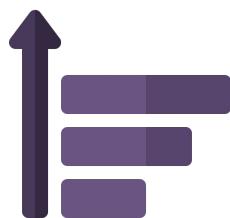
ASC ordena:

Textos varios: [A - Z]

Números: [0 - 9]

Fechas: [21-01-2022 - 25-05-2022]

ó [2022-01-21 - 2022-05-25]



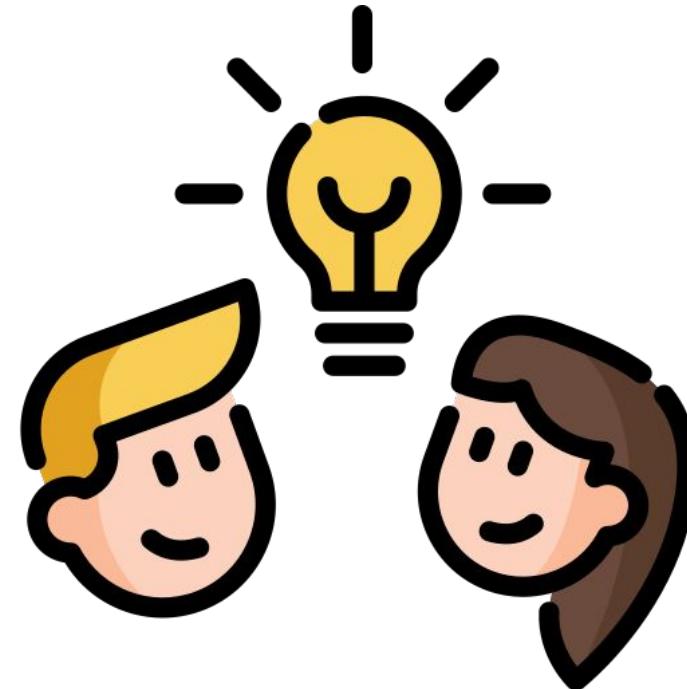
DESC ordena:

Textos varios: [Z - A]

Números: [9 - 0]

Fechas: [25-05-2022 - 21-01-2022]

ó [2022-05-25 - 2022-01-21]



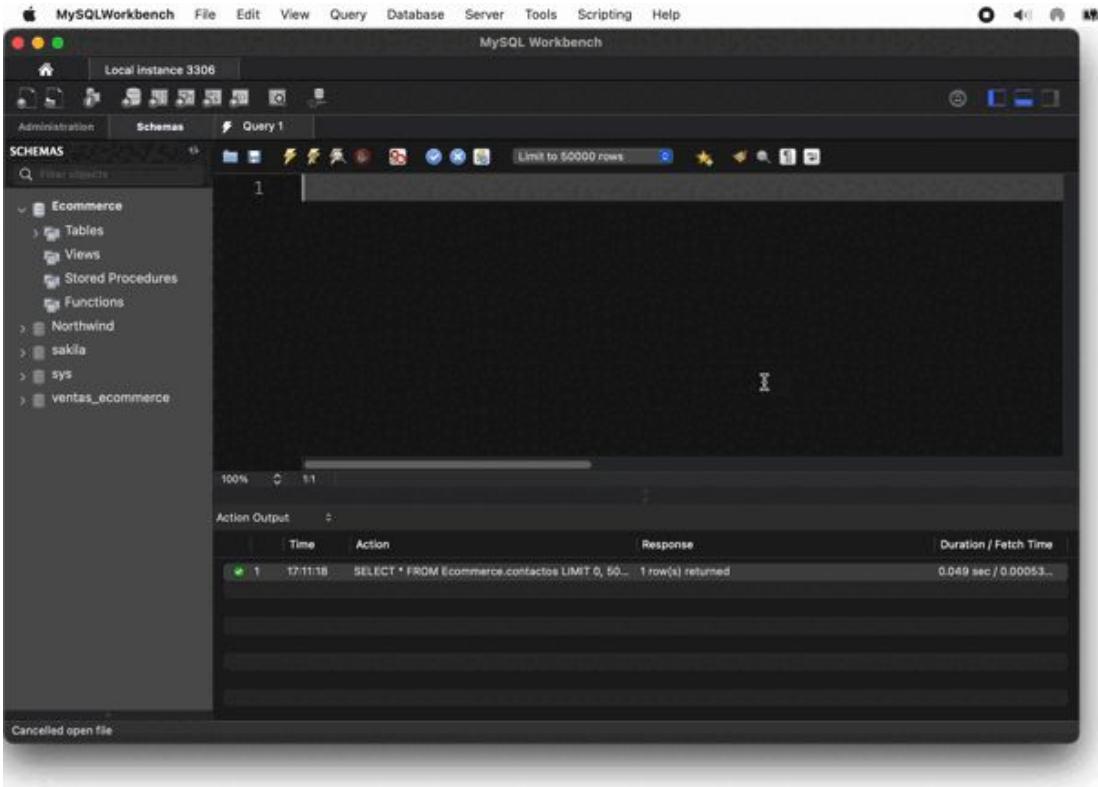
Sección práctica

Para poder abordar prácticas sobre datos contundentes, instalaremos una base de datos MySQL en nuestro entorno de trabajo denominada Northwind.

La misma ya posee una estructura de datos y una cantidad importante de registros, lo cual nos permitirá realizar diferentes tipos de consultas utilizando el lenguaje SQL.



Prácticas



- Instalaremos la **base de datos Northwind** en MySQL Workbench para comenzar a realizar prácticas.
- Descarga el **archivo .sql** compartido por la Profe
- Ábrelo con MySQL Workbench y pulsa el botón **Execute the Script**: (menú **Query > Execute**)

Tiempo estimado: **10 minutos**

Como referencia

Northwind es una base de datos de ejemplo que se utilizó en la documentación y los ejemplos de Microsoft Access desde las primeras versiones de esta aplicación de software.

Representa una empresa ficticia que vende productos a clientes y realiza transacciones comerciales, y contiene tablas de productos, categorías de productos, proveedores, clientes, pedidos, detalles de pedidos y empleados.

Northwind es una excelente herramienta para aprender y practicar la administración de bases de datos, la escritura de consultas y la creación de métricas. Debido a que ha sido utilizada en muchos ejemplos y tutoriales, es fácil encontrar recursos y guías online para trabajar con Northwind.



LIMIT

LIMIT

Corresponde a una cláusula SQL que se utiliza en las consultas de selección, para limitar el total de registros a visualizar.

La misma tiene algunas particularidades que debemos conocer para poder ejecutar consultas de selección efectivas.



Cláusula LIMIT

```
SELECT *
FROM tabla
ORDER BY campo DESC
LIMIT 10;
```

LIMIT

Puede recibir un parámetro, o dos.

Cuando definimos un solo parámetro en la consulta, se limitará a traer el total de registros correspondientes a dicho parámetro.



LIMIT CON 1 PARAMETRO

```
SELECT *
FROM clientes
ORDER BY DNI DESC
LIMIT 10;
```

LIMIT

Cuando definimos LIMIT con dos parámetros, le indicamos a través del primero de ellos, desde qué registro deseamos empezar a visualizar la información.

El segundo parámetro pasa a ser el que limitará el total de registros a mostrar.



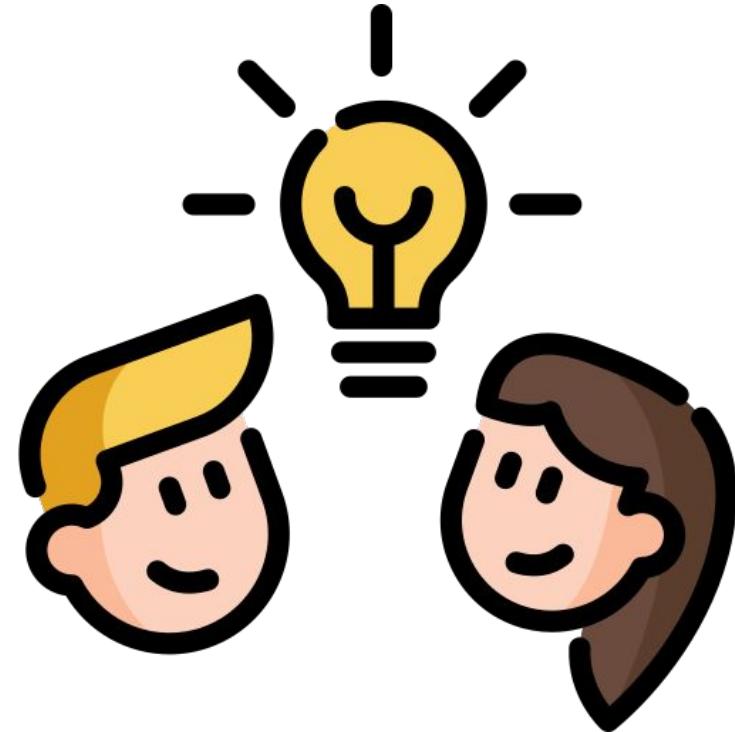
LIMIT CON 2 PARAMETROS

```
SELECT *
FROM clientes
ORDER BY DNI DESC
LIMIT 1, 10;
```

LIMIT

Las tablas en MySQL comienzan a contar los registros en forma de Array de datos, como en los lenguajes de programación. El primer registro de la tabla tendrá la posición número 0, el segundo registro tendrá la posición número 1, y así sucesivamente.

Debemos prestar atención con esto porque podemos ejecutar una consulta de selección importante, dejando de lado el primero de los registros de ésta si especificamos el número 1 como primer parámetro.

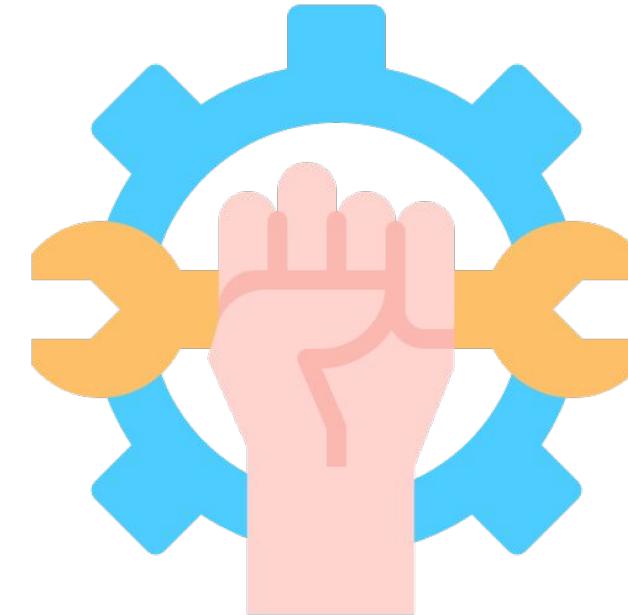


LIMIT

Aquí encontramos la primera diferencia entre MySQL con otro motor, como ser SQL Server, ya que en este último no existe la cláusula **LIMIT**.

En su defecto, incluye la cláusula **TOP n** para limitar el total de registros que deseamos obtener en la consulta de selección.

Y solo recibe un único parámetro; el total numérico.



LIMIT

Referencia comparativa entre MySQL y SQL Server.



MySQL versus SQL Server

-- MySQL

```
SELECT * FROM clientes ORDER BY DNI DESC LIMIT 0, 10;
```

-- SQL Server

```
SELECT TOP 10 * FROM clientes ORDER BY DNI DESC;
```

DISTINCT

DISTINCT

La cláusula **DISTINCT** se usa en combinación con **SELECT** y nos permite filtrar de una consulta de selección, aquellos registros que pueden estar repetidos en la tabla, en base a un dato en particular.



La cláusula DISTINCT

```
SELECT DISTINCT campo  
FROM tabla;
```

DISTINCT

Debemos sumar al menos el nombre de un campo. También podemos sumar más de un campo dentro de la cláusula **DISTINCT**.

A mayor cantidad de campos definidos, más laxo será el resultado de la consulta.



La cláusula DISTINCT con más campos

```
SELECT DISTINCT campo, otroCampo, otroMas  
FROM tabla;
```

DISTINCT

Analizando una tabla como la siguiente, donde parte de sus datos se repiten:

¿Se imaginan cuál es el resultado de la primera consulta? ➡

idCliente	nombre	apellido
1	Juana	Manso
2	Juana	Manso
3	Juana	Manso
4	Juana	Manso

¿Y el resultado de la segunda consulta? ➡

Consulta 1 DISTINCT

```
-- consulta 1  
SELECT DISTINCT idCliente, nombre, apellido  
FROM clientes;
```

Consulta 2 DISTINCT

```
-- consulta 2  
SELECT DISTINCT nombre, apellido  
FROM clientes;
```

DISTINCT

Resultados:

idCliente	nombre	apellido
1	Juana	Manso
2	Juana	Manso
3	Juana	Manso
4	Juana	Manso

nombre	apellido
Juana	Manso

Consulta 1 DISTINCT

```
-- consulta 1
SELECT DISTINCT idCliente, nombre, apellido
FROM clientes;
```

Consulta 2 DISTINCT

```
-- consulta 2
SELECT DISTINCT nombre, apellido
FROM clientes;
```

Sección práctica

Pongámonos a tono con la base de datos Northwind.

Para ello, desarrollarás las siguientes consultas de ejecución sobre una tabla específica, de acuerdo a las indicaciones que verás en el siguiente Slide.



Prácticas

1. Ejecuta una consulta de selección sobre todos los campos de la tabla **Customers**
2. Ejecuta una consulta de selección de los siguientes campos de la tabla **Customers**:
 - **CustomerID, CompanyName, ContactName, ContactTitle, City, Phone**
 - Ordena esta consulta por el campo **CompanyName**
3. Ejecuta una consulta de selección sobre los siguientes campos de la tabla **Customers**:
 - **CustomerID, CompanyName, ContactName, ContactTitle**
 - Ordena esta consulta por el campo **ContactName** de forma descendente
4. Ejecuta una consulta de selección sobre todos los campos de la tabla **Customers**:
 - Ordena esta consulta por el campo **CustomerID**
 - Limita el total de registros a visualizar a 20
5. Ejecuta una consulta de selección sobre todos los campos de la tabla **Customers**:
 - Ordena esta consulta por el campo **ContactName**
 - **Limita el total de registros a visualizar:** muestra solo 10 registros a partir del cliente número 10 de esta consulta resultante

Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



Argentina programa 4.0



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*

Clase 20: Bases de datos Relacionales

Filtros y Operadores SQL

Agenda de hoy

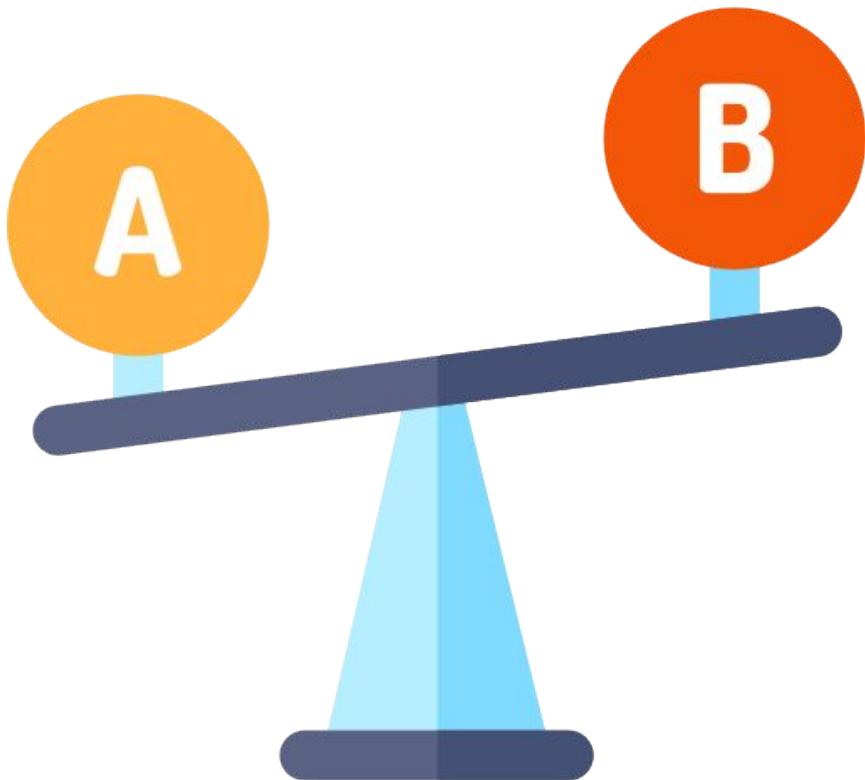
- A. Los operadores de comparación
- B. Operadores lógicos
- C. La cláusula WHERE
 - a. Filtrar información
 - i. filtro simple
 - ii. operador lógico AND
 - iii. operador lógico OR
 - iv. otros operadores
- D. El operador LIKE
 - a. Los caracteres comodín



Los operadores de comparación

Los **operadores de comparación** permiten evaluar una condición y determinar si el resultado de dicha condición es **verdadero o falso**.

Como ya podemos imaginar, estos operadores son comunes no solo a SQL, sino también a todos los lenguajes de programación más allá de alguna pequeña diferencia en la forma de escribirlos.



Los operadores de comparación

Los operadores de comparación:

=	<i>igual a</i>	IS [NOT] NULL	<i>no es nulo</i>	BETWEEN	<i>entre</i>
<	<i>menor a</i>	NOT	<i>NOT lógico</i>	[NOT] BETWEEN	<i>no esta entre</i>
>	<i>mayor a</i>	LIKE	<i>es como</i>	IN	<i>en (lista)</i>
<=	<i>menor o igual a</i>	[NOT] LIKE	<i>no es como</i>	[NOT] IN	<i>no esta en (lista)</i>
=>	<i>mayor o igual a</i>	IS [NOT] TRUE	<i>no es verdadero</i>	IS [NOT] FALSE	<i>no es falso</i>
!= ó <>	<i>distinto de</i>	AND	<i>AND lógico</i>	OR	<i>OR lógico</i>

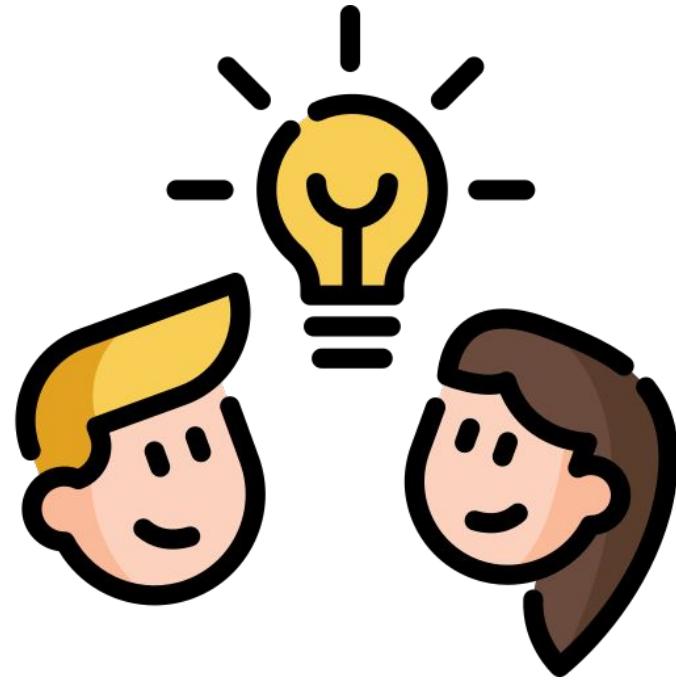
¿Los tienen identificados de algún otro lado? 😊



Los operadores de comparación

Como apreciamos en el slide anterior, esta lista de operadores es CROSS a cualquier lenguaje de programación, más allá de algunas pequeñas diferencias que puedan surgir entre formas de comparar.

El operador “distinto de”, originalmente se escribía de forma diferente entre los sabores más conocidos de SQL. En estos últimos años, se hicieron compatibles ambas opciones entre los principales motores SQL del mercado.



Los operadores lógicos

De esta lista, los operadores lógicos, identificados con el recuadro de color, son los que nos permiten, o combinar más de una posible opción en la cláusula de consulta, o nos permiten invertir el operador de comparación buscando, por ejemplo, aquellos que NO CUMPLEN una determinada condición.

=	<i>igual a</i>	IS [NOT] NULL	<i>no es nulo</i>	BETWEEN	<i>entre</i>
<	<i>menor a</i>	NOT	<i>NOT lógico</i>	[NOT] BETWEEN	<i>no está entre</i>
>	<i>mayor a</i>	LIKE	<i>es como</i>	IN	<i>en (lista)</i>
<=	<i>menor o igual a</i>	[NOT] LIKE	<i>no es como</i>	[NOT] IN	<i>no está en (lista)</i>
=>	<i>mayor o igual a</i>	IS [NOT] TRUE	<i>no es verdadero</i>	IS [NOT] FALSE	<i>no es falso</i>
!= ó <>	<i>distinto de</i>	AND	<i>AND lógico</i>	OR	<i>OR lógico</i>



La cláusula WHERE

La cláusula WHERE

Si bien en nuestro encuentro anterior, comenzamos a aplicar filtros de acuerdo a diferentes condiciones, hoy sumamos una de las cláusulas que mejor aplica la lógica condicional sobre un conjunto de registros resultantes.

Veamos de qué se trata y qué rol cumple dentro de las consultas de selección SQL.



La cláusula WHERE

Para poder obtener registros que cumplan una condición específica, debemos aplicar en la sentencia SQL, la cláusula **WHERE**.

Esta permite establecer condiciones para filtrar los resultados.



La cláusula WHERE

El uso de WHERE en cualquier tipo de consulta SQL permite aplicar diferentes filtros sobre la tabla o tablas que le indiquemos, consiguiendo así uno o más campos y especificando sobre éstos el valor a filtrar, integrando en la cláusula a los operadores de comparación.



La cláusula WHERE

El uso de WHERE en cualquier tipo de consulta SQL permite aplicar diferentes filtros sobre la tabla o tablas que le indiquemos, consiguiendo así uno o más campos y especificando sobre éstos el valor a filtrar, integrando en la cláusula a los operadores de comparación.



La cláusula WHERE

La cláusula WHERE

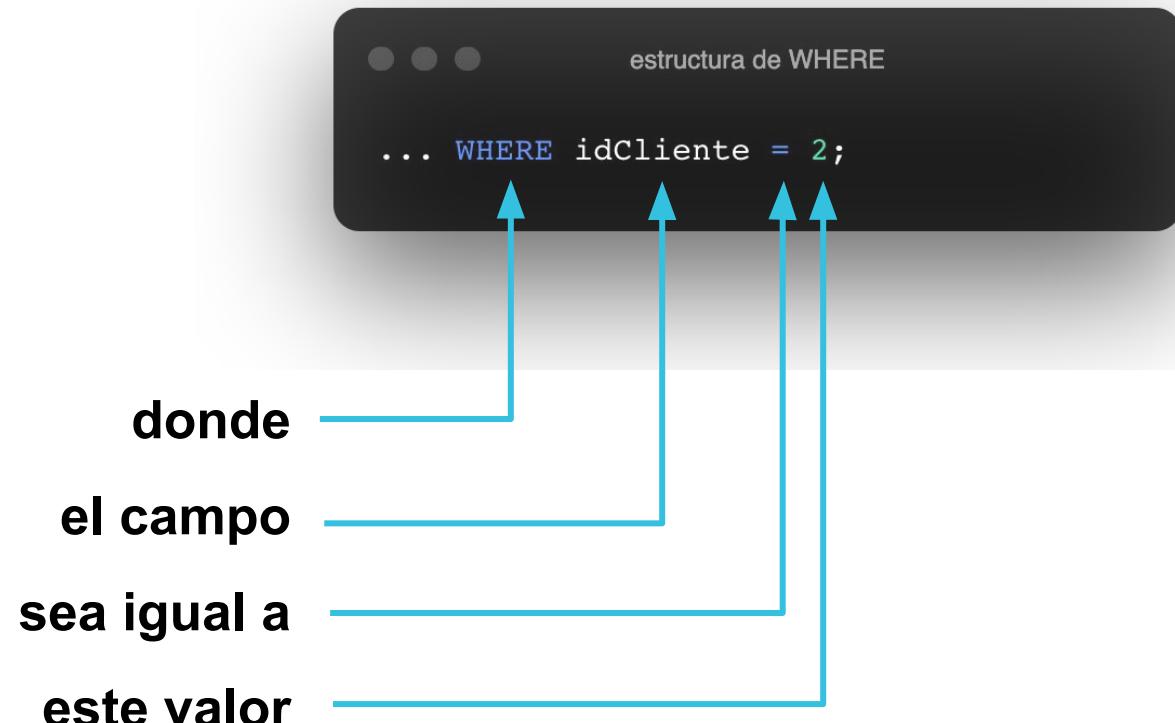
```
SELECT idCliente, nombre FROM clientes WHERE idCliente = 2;
```

idCliente	nombre
1	Marissa Mayer
2	Susan Wojcicki
3	Mira Murati
4	Melinda Gates

idCliente	nombre
2	Susan Wojcicki

Ejemplo de la cláusula **WHERE** aplicada junto al operador igual (=).

La cláusula WHERE



Ejemplo de la cláusula WHERE aplicada junto al operador igual (=).



Los operadores de comparación

El operador igual se reemplaza por cualquier otro operador de comparación de acuerdo a la tabla.

Los valores numéricos difieren de los valores del tipo cadenas de texto.

Los valores alfanuméricos deben ser encerrados entre comillas, mientras que en los valores numéricos no es necesario realizar este paso.

Consultas de filtro simple

WHERE y el operador de comparación =



WHERE numérico

```
SELECT idCliente, nombre FROM clientes WHERE idCliente = 2;
```



WHERE string

```
SELECT idCliente, nombre FROM clientes WHERE nombre = 'Julian';
```

Ejemplo de uso de la cláusula **WHERE** utilizando un campo del tipo numérico, versus la misma cláusula consultando un campo del tipo texto.

WHERE y el operador de comparación =



WHERE booleano

```
SELECT idCliente, nombre FROM clientes WHERE activo = TRUE;
```

Ante el caso de utilizar un campo de tabla booleano dentro de la cláusula WHERE, podremos definir tanto su valor booleano estándar (TRUE ó FALSE), como también su valor numérico (1 ó 0).

En ambos casos, los parámetros o valores los debemos definir sin comillas. SQL se ocupará de traducir el valor booleano al valor numérico, que es lo que este motor entiende.

Otros operadores de comparación

El operador de comparación (mayor a)

El **operador de comparación > (mayor a)**,
se puede aplicar en cualquier campo con
cualquier tipo de dato.

Comúnmente sobre tipos de datos
numéricos o fechas más que en campos
del tipo cadenas de texto.

```
... WHERE operador mayor a  
... FROM OrderDetails WHERE Quantity > 50;  
... FROM Orders WHERE OrderDate > '1997-01-01';
```

El operador de comparación (menor a)

El **operador de comparación < (menor a)**,
cumple con los mismos requisitos y
condiciones que el operador **(mayor a)**,
pero obviamente a la inversa.



WHERE booleano

```
... FROM OrderDetails WHERE Quantity < 20;  
... FROM Orders WHERE OrderDate < '1997-08-30';
```

El operador de comparación (mayor a)

El **operador de comparación \geq (mayor o igual a)**, de igual forma que el **operador mayor a**, con la diferencia de que incluye en el resultado cualquier valor que sea estrictamente igual al parámetro definido, además de todo aquel valor superior.

... FROM Orders WHERE OrderID \geq 11000;

El operador de comparación (mayor o igual a)

El **operador de comparación `>=` (mayor o igual a)**, de igual forma que el **operador mayor a**, con la diferencia de que incluye en el resultado cualquier valor que sea estrictamente igual al parámetro definido, además de todo aquel valor superior.

... FROM Orders WHERE OrderID `>=` 11000;

Sección práctica

Vayamos experimentando el uso de los operadores de comparación vistos hasta aquí.

Como ya tenemos práctica en el manejo de filtros de la mano de funciones de orden superior, no tendremos mayor problema en aplicar el manejo de los mismos pero dentro de diferentes cláusulas SQL.



Tiempo estimado: **15 minutos**

Prácticas

1. Ejecuta una consulta de selección sobre todos los campos de la tabla **Customers**
 - o donde la ciudad sea igual a ‘Buenos Aires’
2. Ejecuta una consulta de selección sobre los siguientes campos de la tabla **Customers**:
 - o **customerID, CompanyName, ContactName, ContactTitle**
 - o donde el campo **City** sea igual a ‘London’
3. Ejecuta una consulta de selección sobre todos los campos de la tabla **Employees**:
 - o donde el campo **Title** sea igual a ‘Sales Representative’
 - o Order por el campo **City** de forma descendente
4. Ejecuta una consulta de selección sobre los siguientes campos de la tabla **Employees**:
 - o **LastName, FirstName, Title, City**
 - o donde el campo **Country** sea igual a ‘USA’
 - o Ordena esta consulta por el campo **LastName**
5. Ejecuta una consulta de selección sobre los siguientes campos de la tabla **Suppliers**:
 - o **SupplierID, CompanyName, ContactName**
 - o donde el campo **ContactTitle** sea igual a ‘Accounting Manager’

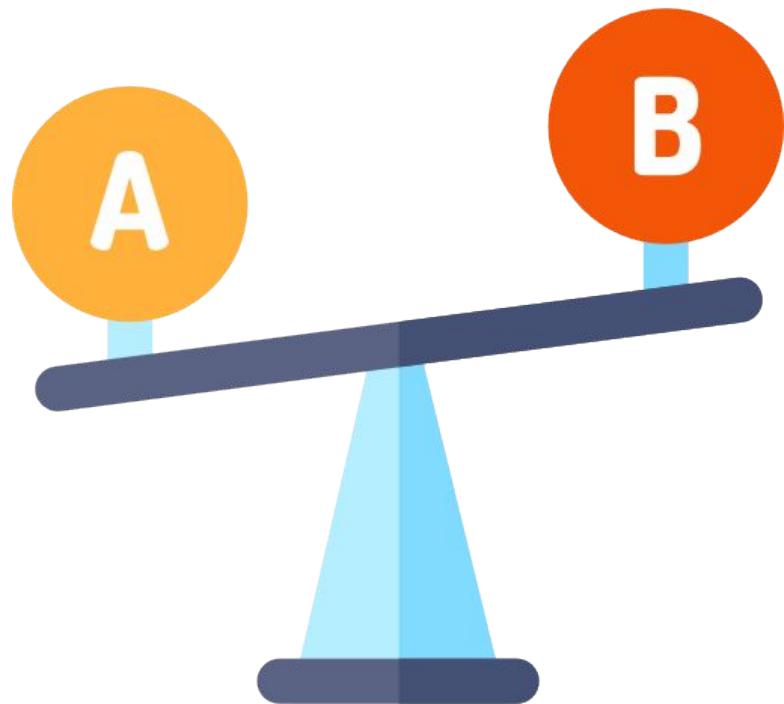


El operador lógico AND

El operador lógico AND

El **operador lógico AND** se utiliza para **combinar múltiples condiciones** en una cláusula WHERE.

Cuando se utiliza este operador, **todas las condiciones deben ser verdaderas** para que la fila o registro se incluya en el resultado de la consulta.



El operador lógico AND

Por ejemplo, ante una consulta de selección de datos de clientes donde queremos buscar un cliente con un determinado código, o Identificador, y que además esté con su estado **Activo**, debemos combinar allí ambas condiciones mediante el operador lógico AND.



WHERE operadores lógicos

```
... FROM clientes WHERE idCliente = 2 AND estado = 'Activo';
```

El operador lógico AND

Solo obtendremos como resultado aquel o aquellos registros que coincidan con ambos parámetros y valores especificados. Si ninguno cumple con los dos parámetros y valores, entonces la consulta no retornará ninguna fila como resultado.



WHERE operadores lógicos

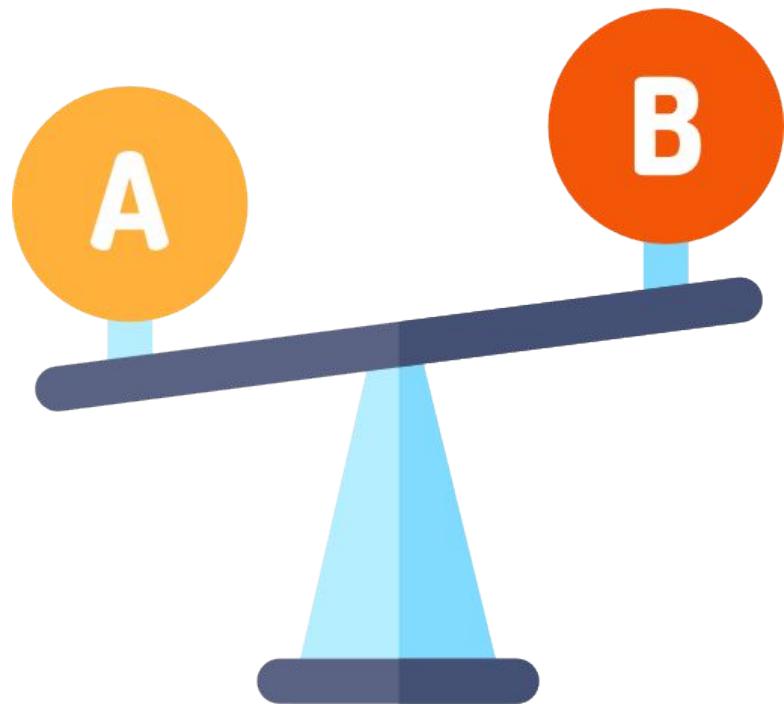
```
... FROM clientes WHERE idCliente = 2 AND estado = 'Activo';
```

El operador lógico OR

El operador lógico OR

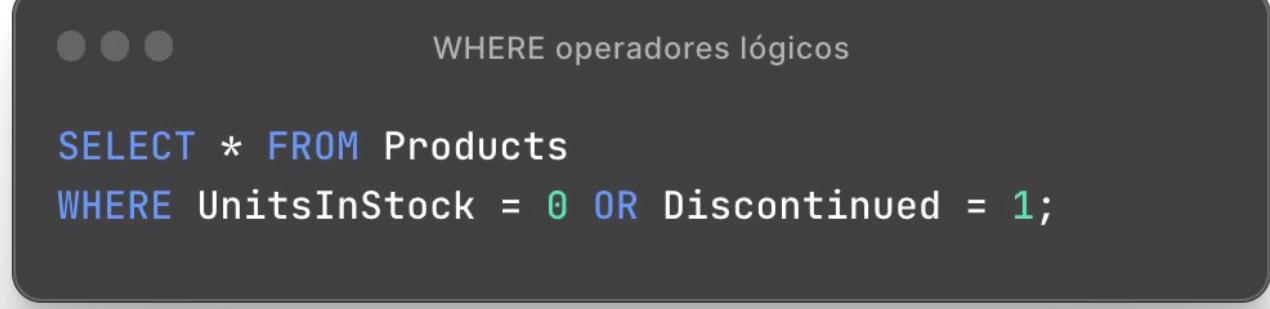
El **operador lógico OR** se utiliza para **combinar una condición con un valor específico** más una posible condición como valor de fallback.

Cuando se utiliza este operador, **si la primera condición no encuentra resultados en la tabla** se irá a buscar registros que coincidan con la segunda condición especificada.



El operador lógico OR

En este ejemplo, intentamos visualizar en la consulta resultante solo aquellos productos que tengan las unidades en stock en cero. Si no existe ningún producto con esta condición, entonces se intentará obtener aquellos productos cuyo campo Discontinued es igual a TRUE, o 1, que es el equivalente en valor booleano.



WHERE operadores lógicos

```
SELECT * FROM Products
WHERE UnitsInStock = 0 OR Discontinued = 1;
```



El operador lógico OR

En el caso que no se cumpla una condición o la otra, el resultado de esta consulta arrojará una tabla sin registros.

El operador lógico OR funciona como un operador de cortocircuito (operando 1 ú operando 2), de la misma forma que sucede en JavaScript.



WHERE operadores lógicos

```
SELECT * FROM Products  
WHERE UnitsInStock = 0 OR Discontinued = 1;
```

Otros operadores de comparación

El operador IN

El operador de comparación IN funciona como la intersección de Conjuntos en Matemáticas.

Debemos definir todos aquellos valores que deben estar incluidos en el campo que utilizamos para aplicar la condición.



WHERE otros operadores

```
SELECT * FROM Products  
WHERE categoryID IN (3, 4, 7);
```

El operador IN

Traerá como resultado solo aquellos productos cuyo campo categoryId tenga como valor el número 3, el número 4 o el número 7.



WHERE otros operadores

```
SELECT * FROM Products  
WHERE categoryId IN (3, 4, 7);
```

El operador “distinto de”

Este operador se ocupa de recuperar todos aquellos registros cuyo parámetro sea distinto del valor indicado.



WHERE otros operadores

```
SELECT * FROM Products  
WHERE categoryID <> 3;
```

El operador “distinto de”

Desde hace algunos años, MySQL soporta como operador “*distinto de*” el uso de los signos `<>` y también de los signos `!=`. Originalmente la primera de las opciones se utilizaba solamente en SQL Server, mientras que esta última opción se utilizaba dentro de todas las bases SQL que se ajustaban al estándar ANSI SQL.



WHERE otros operadores

```
SELECT * FROM Products  
WHERE categoryID != 3;
```

El operador Between

Este operador nos permite especificar un rango de valores para el parámetro por el cual vamos a realizar la consulta de selección.

En este caso, los valores del campo **UnitsInStock** deben encontrarse entre **10** y **40** unidades.



WHERE otros operadores

```
SELECT * FROM Products  
WHERE UnitsInStock BETWEEN 10 AND 40;
```

El operador Between

Sería el equivalente a utilizar en algún lenguaje de programación la conjugación de **Mayor o igual a** y **Menor o igual a**, solo que en SQL se utiliza la palabra reservada BETWEEN en conjunción con el operador lógico AND.



WHERE otros operadores

```
SELECT * FROM Products  
WHERE UnitsInStock BETWEEN 10 AND 40;
```

El operador LIKE

El operador LIKE

El **operador LIKE** se utiliza para buscar patrones en una columna o campo de una tabla.

A diferencia del resto de los operadores vistos hasta aquí, LIKE es un poco más laxo, dado que nos permite buscar un patrón aproximado al valor que le indicamos, y no exacto como es el caso de la mayoría de los otros operadores que trabajamos.



El operador LIKE

```
... operador LIKE  
  
SELECT idCliente, nombre  
FROM clientes  
WHERE nombre LIKE 'Ju%';
```

En este ejemplo, intentamos ubicar a uno o más clientes que posean como valor en su campo nombre, el valor 'Ju'. Si prestamos atención al parámetro indicado como valor, veremos que este posee además de las letras indicadas, el uso del carácter %. Este carácter oficia de comodín, representando al resto de los posibles caracteres que pueda llegar a tener el nombre en cuestión.



El operador LIKE

```
• • • operador LIKE  
  
SELECT idCliente, nombre  
FROM clientes  
WHERE nombre LIKE 'Ju%';
```

En este ejemplo, intentamos ubicar a uno o más clientes que posean como valor en su campo nombre, el valor 'Ju'. Si prestamos atención al parámetro indicado como valor, veremos que este posee además de las letras indicadas, el uso del carácter %. Este carácter oficia de comodín, representando al resto de los posibles caracteres que pueda llegar a tener el nombre en cuestión.



El operador LIKE



operador LIKE

```
SELECT idCliente, nombre  
FROM clientes  
WHERE nombre LIKE 'Ju%';
```

De la forma en la cual está planteado este parámetro como valor, podríamos obtener una lista de nombres con valores como los siguientes:
“Julia, Julio, Julián, Juliana, Julieta, Juan”, entre otros...

El operador LIKE



operador LIKE

```
SELECT idCliente, nombre  
FROM clientes  
WHERE nombre LIKE 'Ju%';
```

Aquí vemos la flexibilidad que nos da el uso de LIKE. Nos acerca un valor aproximado al indicado, pudiendo contener cero, uno o más caracteres aparte de los indicados como valor del parámetro.

Los caracteres comodín

Los caracteres comodín

Los caracteres comodín se utilizan junto a las cláusulas WHERE y LIKE, para especificar diferentes formas de obtener información específica.

El carácter **%** nos da una perspectiva indefinida además de los caracteres que le indiquemos junto a este carácter comodín.



Los caracteres comodín

Aquí tenemos un ejemplo amplio de cómo podemos aprovechar el uso del carácter comodín %, para poder buscar un término con diferentes posibles combinaciones dentro de una palabra.



WHERE otros operadores

```
...WHERE nombre LIKE 'Ju%';
-- retorna todos los nomrbes que comiencen con 'Ju'

...WHERE nombre LIKE '%ju';
-- retorna todos los nomrbes que finalicen con 'Ju'

...WHERE nombre LIKE '%ju%';
-- retorna todos los nomrbes que comiencen, finalicen,
o contengan en alguna parte del mismo el término 'ju'
```

Los caracteres comodín

Existe una diversidad de caracteres comodines y combinaciones, las cuales nos permiten alcanzar una amplia variedad de resultados, según la complejidad de búsqueda ante la cual nos encontramos.



Caracteres comodín y LIKE

```
SELECT * FROM Tabla WHERE Campo LIKE '_r%';  
  
SELECT * FROM Tabla WHERE Campo LIKE 'Au ____';  
  
SELECT * FROM Tabla WHERE Campo LIKE 'A_ %';  
  
SELECT * FROM Tabla WHERE Campo LIKE 'A____a';
```

Los caracteres comodín

El uso del carácter **_** (*underscore*) representa una sola letra en un parámetro.

Caracteres comodín y LIKE

```
SELECT * FROM Tabla WHERE Campo LIKE '_r%';
```

```
SELECT * FROM Tabla WHERE Campo LIKE 'Au ____';
```

```
SELECT * FROM Tabla WHERE Campo LIKE 'A_ %';
```

```
SELECT * FROM Tabla WHERE Campo LIKE 'A____a';
```

Los caracteres comodín



LIKE y caracteres comodín

```
...WHERE contry LIKE '___land';
```

En este ejemplo, si filtramos de una tabla todos los países del mundo, obtendremos como resultado los posibles valores '*Ireland*' y '*Iceland*', dentro de la consulta resultante.

Sección práctica

Ahora que ampliamos nuestro conocimiento con otros tantos operadores de comparación junto a la cláusula WHERE, pongamos a ejercitarnos realizando las siguientes consignas sobre la bb.dd Northwind.



Prácticas

1. Ejecuta una consulta de selección sobre todos los campos de la tabla **Products**
 - o ordena la consulta por el campo **productName**
2. Ejecuta una consulta de selección similar a la primera, aplicando la siguiente condición
 - o el campo **CategoryID** sea igual a 4 y 6
 - o mantén el ordenamiento indicado anteriormente
3. Ejecuta otra consulta de selección similar a la primera, aplicando la siguiente condición
 - o el campo **SupplierID** sea igual a 5 y el campo **CategoryID** sea igual a 4
4. Ejecuta otra consulta de selección similar a la primera, aplicando la siguiente condición
 - o el campo **UnitsInStock** tenga valores entre 25 y 40 unidades
5. Abre la tabla **Products** y modifica manualmente el campo **discontinued** = 1, en al menos 5 registros al azar. Recuerda aplicar / guardar los cambios efectuados
6. Ejecuta una consulta de selección similar a la primera, aplicando la siguiente condición
 - o el campo **UnitsInStock** sea mayor a 400 o el campo **discontinued** sea verdadero
 - o ordena la consulta por el campo **productName**

Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



**Argentina
programa
4.0**



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

***primero
la gente***

Clase 21: Bases de datos Relacionales

Funciones escalares

Agenda de hoy

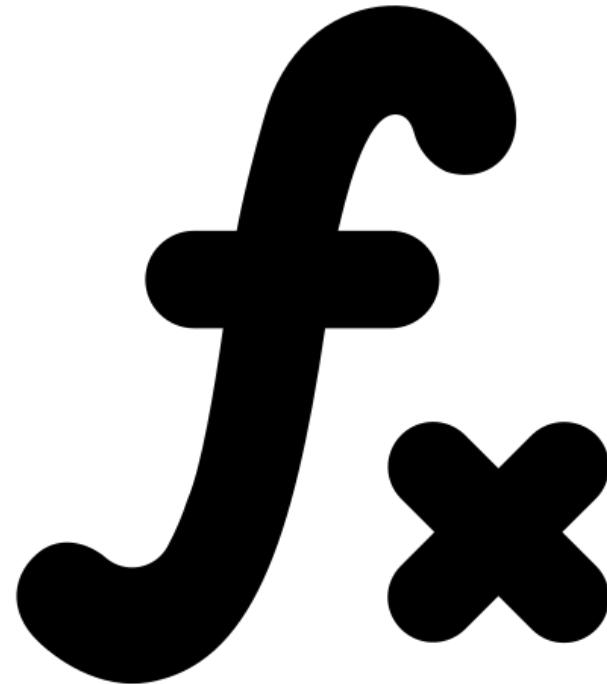
- A. Las funciones escalares
 - a. Qué es una función
- B. Tipos de funciones escalares
 - a. Funciones string
 - b. Funciones Date
 - c. Funciones Number
- C. Prácticas intercaladas con la implementación de diferentes funciones escalares



Qué es una función

De acuerdo a lo que aprendimos dentro del mundo de la programación JavaScript, una función es una pieza de código que realiza una operación determinada, por supuesto para lo que fue ideada.

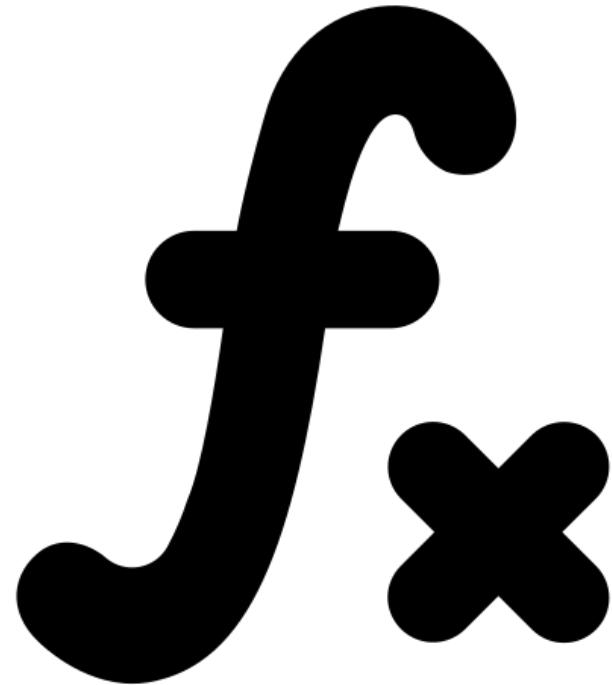
En algunos casos, la función espera uno o más parámetros a procesar, y en otros casos, simplemente se las ejecuta para que devuelvan un dato específico.



Qué es una función

Pensando como un lenguaje de programación, la función puede o no recibir parámetros de entrada, y siempre retornará un valor esperado por el usuario.

Sus nombres son definidos, acorde a lo que deben hacer, y se las utiliza para trabajar con los diferentes tipos de datos que pueden almacenarse en los registros de una tabla.

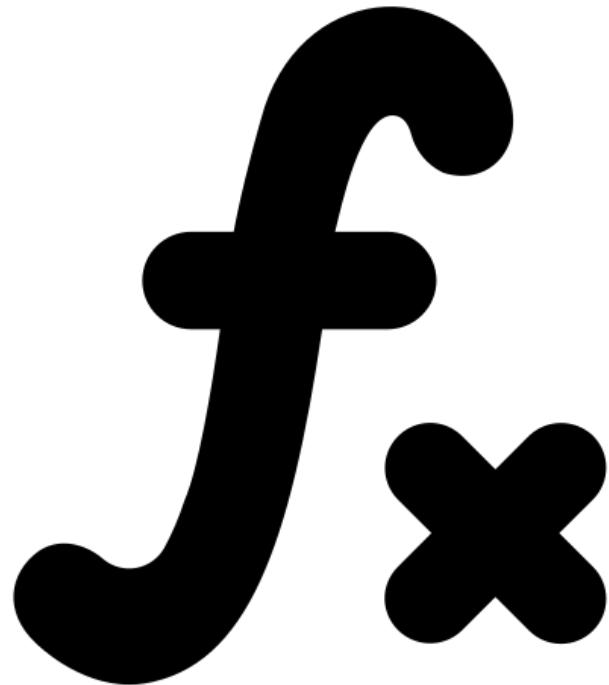


Las funciones escalares

Las funciones escalares

Al igual que los lenguajes de programación en general, SQL incluye una serie de funciones denominadas **Funciones Escalares**.

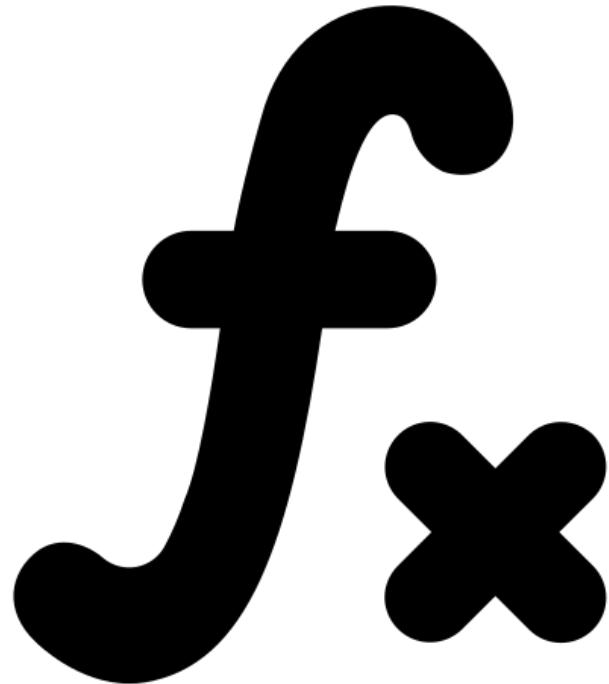
Permiten manipular datos cuando los recuperamos o antes de guardarlos, mediante operaciones predeterminadas, devolviendo un resultado específico, acorde a lo esperado.



Las funciones escalares

Algunas de las ventajas de implementar funciones escalares en nuestras operaciones con SQL, son:

- Reducir el re-trabajo de la lógica comercial
- Evitar la inconsistencia de datos que provenga de un software
- Ayudar a reducir el tráfico de red de aplicaciones cliente/servidor
- Mejorar en gran medida el rendimiento de los sistemas



Las funciones escalares

**Y cuando hablamos de funciones escalares en SQL,
encontramos que existen dos tipos posibles:**

- funciones integradas
- funciones almacenadas

En esta oportunidad trabajaremos con las funciones integradas en el lenguaje de base de datos.



Las funciones escalares

Las funciones almacenadas son habitualmente funciones escalares pero construidas por nosotras mismas.

En determinadas situaciones podemos encontrarnos que sería ideal contar con una función que realice una tarea específica, pero esta no existe.

Lo bueno de esto es que sabiendo programar, podremos construirla nosotras mismas para solventar dicha necesidad dentro de nuestros desarrollos.



Tipos de funciones escalares

Tipos de funciones escalares

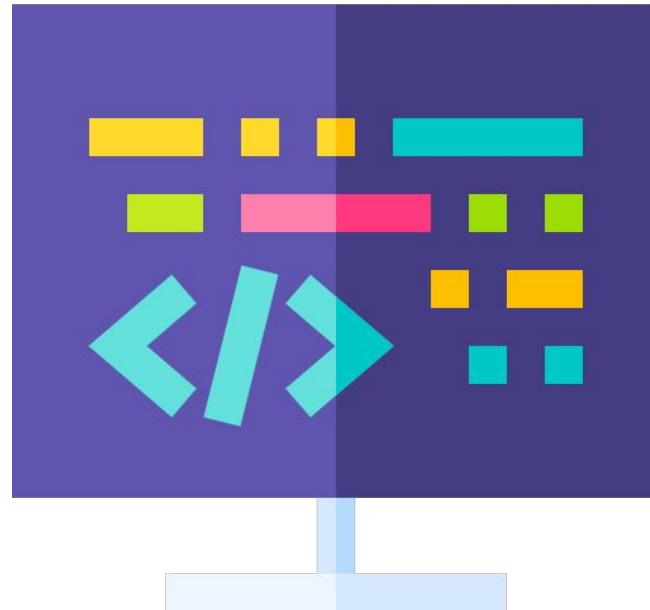
Las funciones escalares, integradas en SQL, se clasifican a su vez bajo las siguientes categorías:

- funciones de cadenas
- funciones numéricas
- funciones de fecha
- funciones de agregación



Tipos de funciones escalares: de cadenas

Las funciones escalares de cadena en SQL son funciones que operan en valores de tipo cadena (*por ejemplo, caracteres alfanuméricos o de texto*) y devuelven, o retornan, un resultado basado en esos valores.



Tipos de funciones escalares: fechas

Las funciones escalares numéricas en SQL son funciones que operan en valores de tipo numéricos.

Usualmente nos acercan una serie de mecanismos para realizar operaciones matemáticas y de cálculo en general, retornando valores de acuerdo a los parámetros indicados.

Tipos de funciones escalares: fechas

Las funciones escalares de fechas en SQL son funciones que operan en valores de tipo fecha y hora.

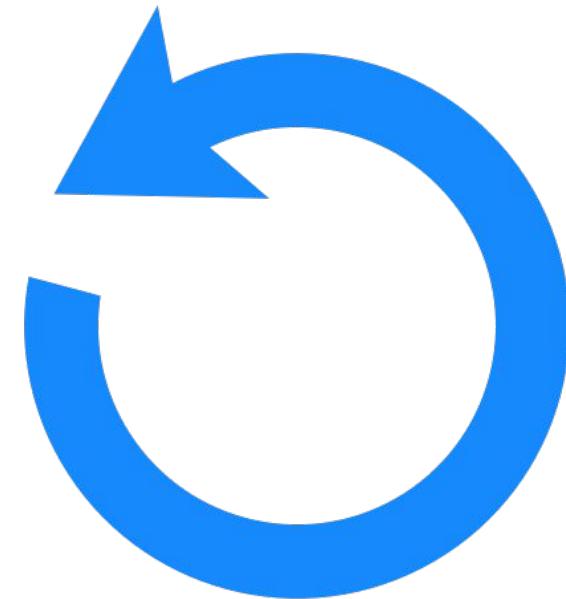
Nos acercan una serie de mecanismos para trabajar con operaciones entre fechas y/u horas retornando valores como resultado, de acuerdo a los parámetros indicados.



Tipos de funciones escalares

En todos los casos, las funciones retornan algún valor resultante generalmente identificado como operaciones de transformación o cálculo.

Como podemos apreciar, es el mismo principio que vimos oportunamente cuando aprendimos a utilizar y crear funciones en un lenguaje de programación como lo es JavaScript.



funciones escalares de cadenas (string)

Funciones escalares de cadenas (string)

Permiten operar con cualquier tipo de cadena de caracteres almacenada en una tabla (o por almacenarse). Podemos, entre otras cosas:

- convertir el texto a mayúsculas / minúsculas
- concatenar strings
- cortar una porción del texto
- eliminar espacios
- revertir el texto
- contar caracteres



Entre decenas de funciones más.

Funciones escalares de cadenas (string)

Fusiona cadenas de caracteres en un único bloque de datos.

Podemos, por ejemplo, unificar en un campo llamado **nombre_completo**, los campos **nombre** y **apellido** de una tabla.

• • • Función concat()

```
SELECT nombreCompleto,  
       concat(telefono, ' - ', tipodetelefono) AS Tel  
FROM Ecommerce.contacto;
```



Funciones escalares de cadenas (string)

Convierte un bloque de texto a mayúsculas.

Es ideal para cuando queremos normalizar la visualización de determinados datos, o incluso cuando queremos guardar datos normalizados.

Funciones escalares

```
SELECT UCASE>Title) FROM Employees;
```

Funciones escalares de cadenas (string)

Convierte un bloque de texto a minúsculas.

También cuenta con su nombre de función alternativo: **LOWER()**. Este último funciona de la misma forma que **LCASE()**.



Funciones escalares

```
SELECT LCASE(productName) FROM Products;
```

Funciones escalares de cadenas (string)

Invierte el orden de contenido de un campo específico.

Funciona tanto con textos como también en campos numéricos.



Funciones escalares

```
SELECT REVERSE(productName) AS nombreInvertido  
FROM Products;
```

Funciones escalares de cadenas (string)

Elimina espacios innecesarios al inicio o final
de un texto.

Es ideal para normalizar contenido que pueda venir con espacios por demás, por ejemplo, en campos de comentarios que deben tener una cantidad de caracteres mínima, y los usuarios suelen completarlos con espacios.

Función trim()

```
SELECT TRIM("Este es un texto más o menos extenso.") AS TextoCorrecto;
```



Funciones escalares de cadenas (string)

Existen también **LTRIM()** y **RTRIM()** para eliminar espacios en un texto, del lado izquierdo y derecho, respectivamente.



Funciones escalares

```
SELECT LTRIM('    Texto con espacio a la izquierda')  
AS TextoConEspacioIzquierdo;  
  
SELECT RTRIM('Texto con espacio a la derecha      ')  
AS TextoConEspacioDerecho;
```

Funciones escalares de cadenas (string)

Retorna una cantidad de espacios determinados, en base al número que recibe como parámetro.

Ideal por si debemos completar con espacios obligatorios un campo específico.



Función space()

```
SELECT SPACE(21)  
AS EspaciosEnTexto;  
  
-- retorna ' ' espacios
```

Funciones escalares de cadenas (string)

Permite reemplazar una porción de texto por otra, contenida en un bloque de texto determinado. Es ideal para cuando debemos reemplazar posibles caracteres extraños dentro de un bloque de texto, o normalizar quitando acentos, tildes, ñuflos, diéresis, o si debemos eliminar caracteres especiales como ser guiones o espacios en, por ejemplo, un número de teléfono.



Función replace()

```
SELECT REPLACE("SQL SERVER es el mejor", "SQL SERVER", "MySQL") AS ElPreferido;
```

Funciones escalares de cadenas (string)

Permite contar la cantidad de caracteres en un bloque de texto determinado.

Si debemos limitar la cantidad de caracteres a insertar en un determinado campo, podremos validar que esto se cumpla y no sobrepase los caracteres máximos esperados.



Función char_length()

```
SELECT CHAR_LENGTH( "Curso de MySQL" )  
      AS Curso;
```

Funciones escalares de cadenas (string)

Devuelve la posición de un texto o carácter determinado.

Es ideal para identificar si dentro de un bloque de texto hay o no alguna palabra esperada, y en qué posición se encuentra dicha palabra.

- Si no encuentra el término, devolverá 0 (*cero*).
- Si lo encuentra una o más veces, devuelve solo la primera posición encontrada para dicho bloque de texto.



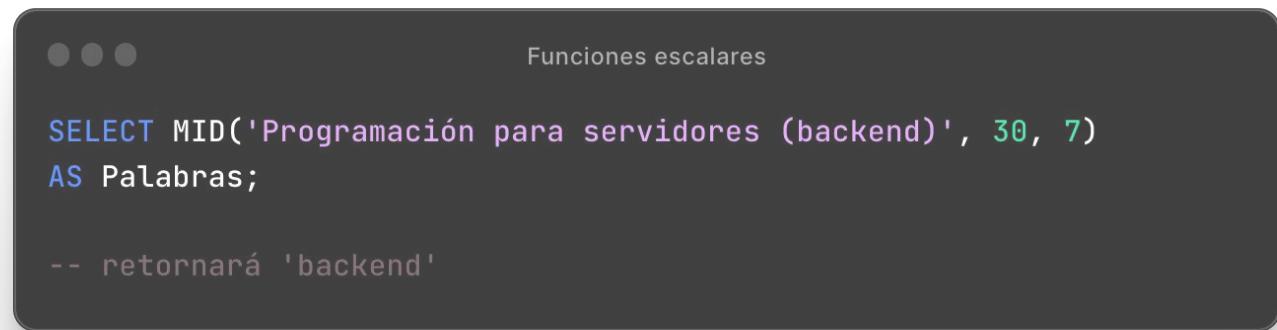
Funciones escalares

```
SELECT INSTR('Programación para servidores (backend)', 'backend')  
AS posicion;
```

Funciones escalares de cadenas (string)

Extrae una porción determinada de caracteres de un bloque de texto. Debemos especificar la posición inicial y el total de caracteres a extraer.

Es utilizada de forma frecuente para la limpieza de contenido en textos (*data analytics*), como también para reducir (*o truncar*) a un limitado número de caracteres un bloque de texto, previo a ser almacenado en un campo específico.



The screenshot shows a terminal window with a dark background. At the top right, there are three small gray dots. To the right of them, the text "Funciones escalares" is displayed. Below this, a SQL query is shown in white text:

```
SELECT MID('Programación para servidores (backend)', 30, 7)  
AS Palabras;  
  
-- retornará 'backend'
```

The number 30 and 7 in the query are highlighted in green, indicating they are constants being passed to the MID function.



Alias en funciones escalares

Alias en funciones escalares

Al ejecutar cualquier función escalar sobre un campo de tabla SQL, veremos que el nombre de la columna en el resultado de la consulta mostrará la función escalar encerrando el nombre del campo.

Para subsanar esto, debemos utilizar la cláusula Alias, la cual nos permite definir otro nombre para el campo. Podremos elegir el mismo o directamente otro diferente, y así conseguir una mejor visualización del resultado de la consulta.



Funciones escalares

```
SELECT MID('Programación para servidores (backend)', 30, 7)  
AS Palabras;  
  
-- retornará 'backend'
```

Alias en funciones escalares

Incluso el uso de Alias puede aplicarse en campos convencionales, para poder cambiar el nombre de ellos si no es muy claro.

Por ejemplo, podemos ejecutar una consulta de selección sobre una tabla SQL, y redefinir el nombre de sus campos con un alias, para pasar los mismos de inglés a español, tal como muestra el ejemplo contigo.

```
... Alias  
  
SELECT ProductID AS Código,  
       UCASE(ProductName) AS Descripción,  
       QuantityPerUnit AS Presentación,  
       UnitPrice AS PrecioUnitario  
  FROM products;
```

Sección práctica

Dado que son muchas las funciones escalares que debemos repasar, haremos una pausa aquí para implementar las funciones escalares del tipo **String** en algunas consultas de selección.

Veamos a continuación la consigna:



Prácticas

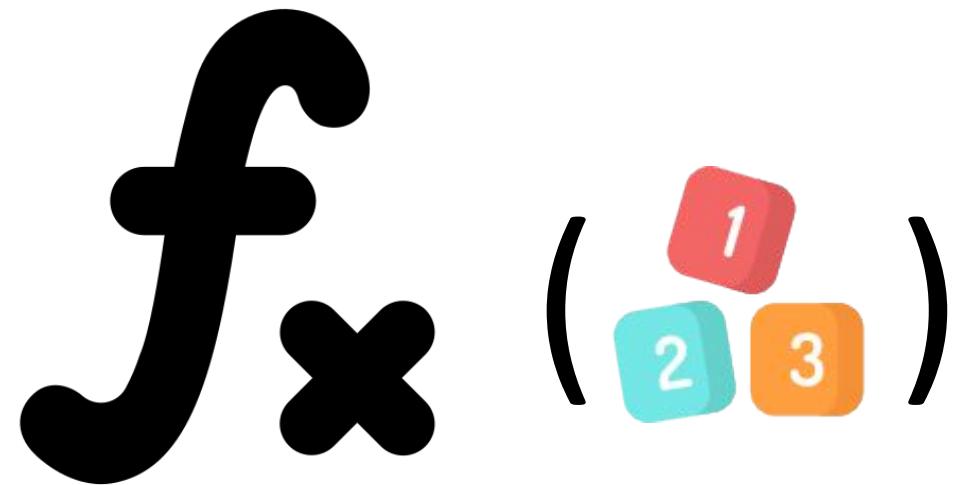
1. Ejecuta una consulta de selección sobre los siguientes campos de la tabla **Products**
 - **productID, productName, QuantityPerUnit, UnitPrice**
 - aplica un alias a cada uno de ellos (**Codigo, Descripcion, Presentacion, PrecioUnitario**)
2. Ejecuta una consulta de selección sobre los siguientes campos de la tabla **Products**
 - **ProductID, ProductName, QuantityPerUnit, UnitPrice**
 - Aplica sobre el campo **ProductName** la función escalar que transforma el texto a mayúsculas
3. Ejecuta una consulta de selección sobre los siguientes campos de la tabla **Products**
 - **ProductID, ProductName, QuantityPerUnit, UnitPrice**
 - aplica el mismo alias detallado en el punto uno (1)
 - Aplica sobre el campo **QuantityPerUnit** la función escalar de reemplazo de texto, buscando el texto '*boxes*' y reemplazando el mismo por '*cajas*'
 - la condición WHERE debe filtrar aquellos registros que posean en cualquier parte la palabra '*boxes*' en cualquier parte del campo **QuantityPerUnit**



funciones escalares numéricas

Funciones escalares numéricas

Permiten operar con cualquier tipo de datos numéricos, realizando operaciones matemáticas y aritméticas entre otras, y hasta transformando la forma de mostrar los números.



Funciones escalares numéricas

La función **abs()** recibe un número como parámetro y **retorna el absoluto.**



Función ABS()

```
SELECT ABS(-243.5) AS NroAbsoluto;
```

-- devuelve el nro. en positivo

Funciones escalares numéricas

La función **ceil()** recibe un número como parámetro y **retorna el número entero más próximo.**

• • • Función CEIL()

SELECT CEIL(21.375) AS NroEnteroProx;

-- devuelve el nro. entero más próximo



Funciones escalares numéricas

La función **floor()** es la función inversa de **ceil()**. Recibe un número como parámetro y **retorna el número entero anterior, más próximo.**



Función FLOOR()

```
SELECT FLOOR(21.375) AS NroEnteroAnt;
```

-- devuelve el nro. entero anterior más próximo

Funciones escalares numéricas

La función **greatest()** recibe un set de números como parámetro y **retorna el número mayor de dicha lista.**



Función GREATEST()

```
SELECT GREATEST(72, 1, 75, 3, 21, 96, 5, 30) AS Mayor;  
-- devuelve el nro. mayor de todo el grupo
```

Funciones escalares numéricas

La función **least()**, es la inversa de **greatest()**, dado que recibe un set de números como parámetro y **retorna el número menor de dicha lista.**



Función LEAST()

```
SELECT LEAST(72, 1, 75, 3, 21, 96, 5, 30) AS Menor;  
-- devuelve el nro. menor de todo el grupo
```

Funciones escalares numéricas

La función **mod()** recibe una división entre dos números, y retorna el módulo, o resto, de dicha operación aritmética.

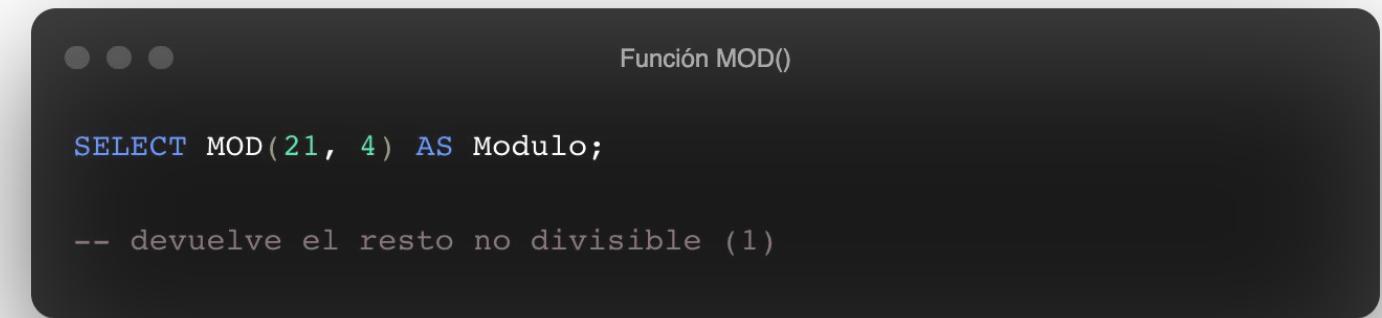
• • • Función MOD()

```
SELECT MOD(21, 4) AS Modulo;  
-- devuelve el resto no divisible (1)
```



Funciones escalares numéricas

La función **mod()** recibe una división entre dos números, y retorna el módulo, o resto, de dicha operación aritmética.



The screenshot shows a terminal window with a dark background. At the top, there are three small circular icons. To the right of them, the text "Función MOD()" is displayed. Below this, a SQL query is shown in white text:

```
SELECT MOD(21, 4) AS Modulo;  
  
-- devuelve el resto no divisible (1)
```

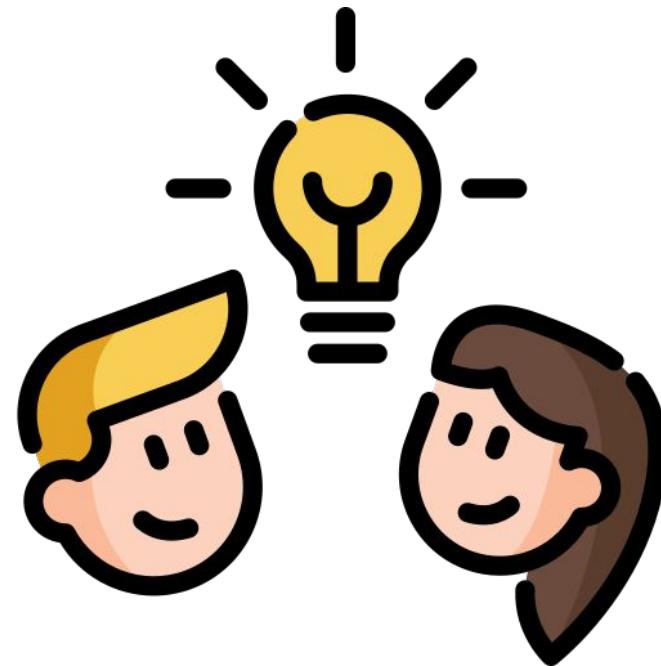


Funciones escalares numéricas

Como bien mencionamos en diferentes instancias, es imposible conocer y tener presente a todas las funciones escalares existentes.

Como siempre, los sistemas de ayuda oficial son los que nos guiarán en conocer la función apropiada y cómo se debe aplicar.

En su mayoría, las funciones escalares trabajan de igual forma, recibiendo uno o más parámetros y siempre retornando una transformación o cálculo como resultado.

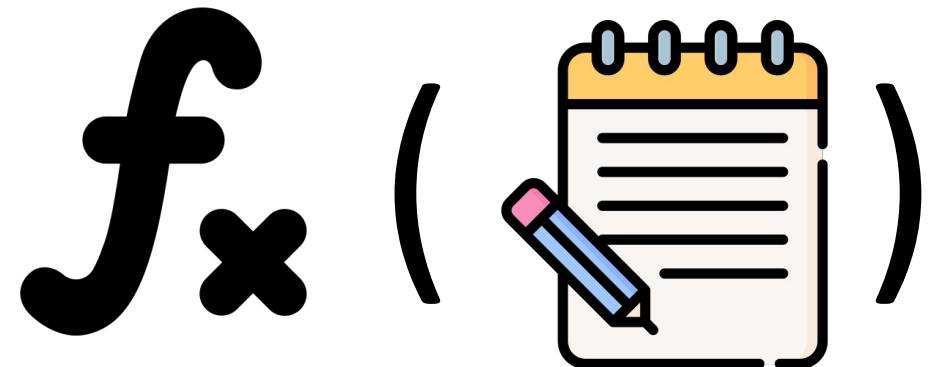


funciones escalares para fechas

Funciones escalares con fechas

Las funciones de fecha y hora, nos permiten trabajar con estas unidades de tiempo, de acuerdo a la necesidad.

En algunos casos, existe más de una función SQL Date, para realizar la misma tarea, tal como vimos oportunamente con las funciones escalares para el manejo de cadenas (string).



Funciones escalares con fechas

NOW() fecha y hora actual

CURDATE() ver el día actual

DAY() retorna el día para una fecha

MONTH() retorna el mes para una fecha

YEAR() retorna el año para una fecha

Funciones Date con fechas

```
SELECT NOW();  
  
SELECT CURDATE();  
  
SELECT DAY();  
  
SELECT MONTH();  
  
SELECT YEAR();
```



Funciones escalares con fechas

DATEDIFF() período en días que pasaron entre dos fechas

DAYOFYEAR() cuántos días transcurrieron en la fecha indicada

WEEKOFYEAR() cuántas semanas transcurrieron en la fecha indicada



Funciones Date con fechas parte 2

```
SELECT DATEDIFF(FECHA1, FECHA2);
```

```
SELECT DAYOFYEAR(FECHA);
```

```
SELECT WEEKOFYEAR(FECHA);
```

Funciones escalares con fechas

Devuelve el período trimestral de la fecha indicada

Enero a Marzo retorna 1

Abril a Junio retorna 2

Julio a Septiembre retorna 3

Octubre a Diciembre retorna 4



Período trimestral del año

```
SELECT QUARTER( "2022-08-03" );
```

Esto representa al término **QUARTER** o simplemente '**Q**' mencionado habitualmente en empresas, por las áreas administrativas y gerenciales.

Funciones escalares con fechas

Retorna el nombre del mes definido en una fecha (*en inglés*)



Nombre del mes

```
SELECT MONTHNAME("2022-03-21");
```

Funciones escalares con fechas

Retorna la diferencia entre un período:

- el período se define como año (*4 dígitos*), y el mes (*2 dígitos*)
- la fecha mayor va en primer lugar, la menor en el segundo
- devuelve el resultado en el número de meses transcurridos
- si el número de meses transcurridos ya pasó, lo muestra en positivo, caso contrario en negativo



Tiempo transcurrido en un período

```
SELECT PERIOD_DIFF(202203, 202109);
```

Funciones escalares con fechas

Permite agregar un intervalo de tiempo a una fecha específica.

El intervalo lo podemos definir en:

- YEAR
- MONTH
- DAY

... Intervalo de tiempo

```
SELECT DATE_ADD("2022-08-03", INTERVAL 4 YEAR)
```

Retorna la fecha resultante.



Funciones escalares con fechas

CURTIME() devuelve la hora actual

TIME_TO_SEC() cuántos segundos

transcurrieron en el tiempo indicado

TIMEDIFF() período de tiempo entre las

dos horas indicadas

Funciones con Horas

```
SELECT CURTIME();  
  
SELECT TIME_TO_SEC("19:30:10");  
  
SELECT TIMEDIFF(HORA1, HORA2);
```

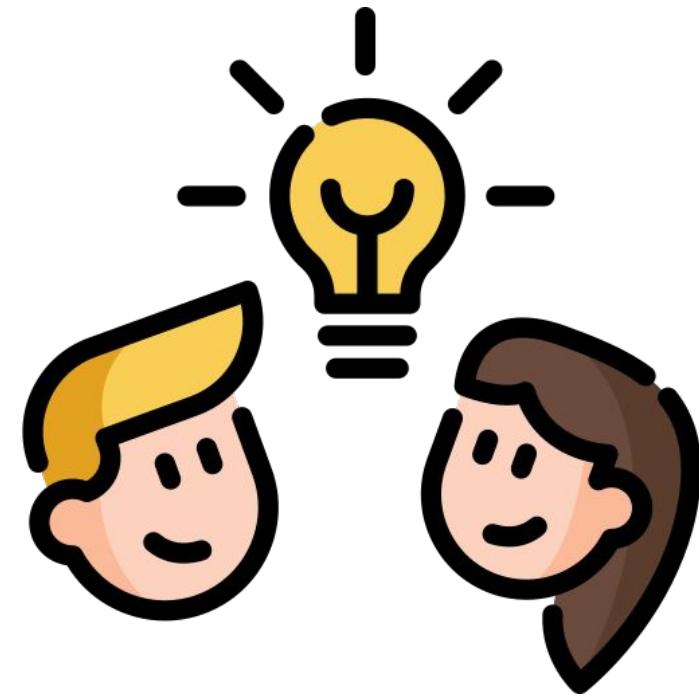


Funciones escalares con fechas

Muchas de estas funciones escalares que manejan fechas y horas pueden “causarnos ruido” a nivel de no comprender para qué se podrían utilizar.

Si aún no tenemos experiencia en empresas, esto es muy normal. Pero ya, cuando formas parte del mercado IT y comienzas a interactuar con otras áreas corporativas, muchos de estos términos se volverán moneda corriente.

Allí es donde verás, para casos puntuales como ser emitir informes gerenciales, muchas de estas funciones escalares nos ahorrarán tiempos notables en procesar cálculos complejos y agrupar resultados.



Sección práctica

Vayamos con un último ejercicio aplicando funciones escalares.

En esta oportunidad trabajaremos con la función escalar de concatenación de datos y con la función escalar de fecha, aplicándolas sobre la tabla *Employees*, la cual contiene información de los empleados de la bb.dd Northwind.



Prácticas

Necesitamos simplificar la visualización de datos de esta tabla, presentando en una consulta de selección, los siguientes campos:

- **EmployeeID, TitleOfCourtesy, LastName, FirstName, Title, BirthDate, HireDate**

Sobre esta consulta de selección base, realiza las siguientes consignas:

1. En una nueva consulta de selección con la base anterior, concatena los campos:
 - a. **(TitleOfCourtesy, LastName, FirstName)** con el alias **NombreCompleto**
 - b. respeta los espacios entre los diferentes campos mencionados
2. En una nueva consulta de selección con la base inicial:
 - a. elimina el formato fecha y hora sobre el campo **BirthDate**, utilizando la función **Date()**
 - b. aplica un alias a dicho campo para llamarlo **FechaNacimiento**
3. Copia la consulta resultante del punto dos, y modifícalo aplicando lo siguiente:
 - a. utiliza la función **YEAR** sobre campo **HireDate**, para mostrar sólo el año de contratación
 - b. aplica un alias a dicha campo, para llamarlo **AnioContratacion**



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



**Argentina
programa
4.0**



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

***primero
la gente***

Clase 22: Bases de datos Relacionales

El modelo relacional



Agenda de hoy

- A. Tipos de índices en SQL
 - a. Claves primarias, foráneas, concatenadas y candidatas
- B. Formas normales
 - a. cuántas son
 - b. primera forma normal
 - c. segunda forma normal
 - d. tercera forma normal
 - e. forma normal de (*Boyce-Codd*)
- C. Subconsultas SQL
- D. Select Case - When



El modelo relacional

Las tablas relacionales son un punto clave y crítico en el universo de las bases de datos SQL.

Es un tema que debemos aprender a dominar, porque el modelo relacional es el Core de este tipo de base de datos, y para lograr esto, debemos entender cuántos y qué tipos de índices podemos manejar con SQL.



El modelo relacional

A continuación haremos un repaso de los diferentes índices que pueden implementarse con el lenguaje SQL, para luego entrar en un espacio teórico donde hablaremos de las diferentes **Formas Normales**, para entender qué son y cómo sacar provecho de las mismas.

¡Vayamos entonces a la acción!



Tipos de índices

Tipos de índices

Los índices en bases de datos relacionales son estructuras de datos utilizadas para mejorar el rendimiento de las consultas.

Éstos funcionan como un catálogo que permite a la base de datos encontrar rápidamente las filas que satisfacen una determinada consulta.

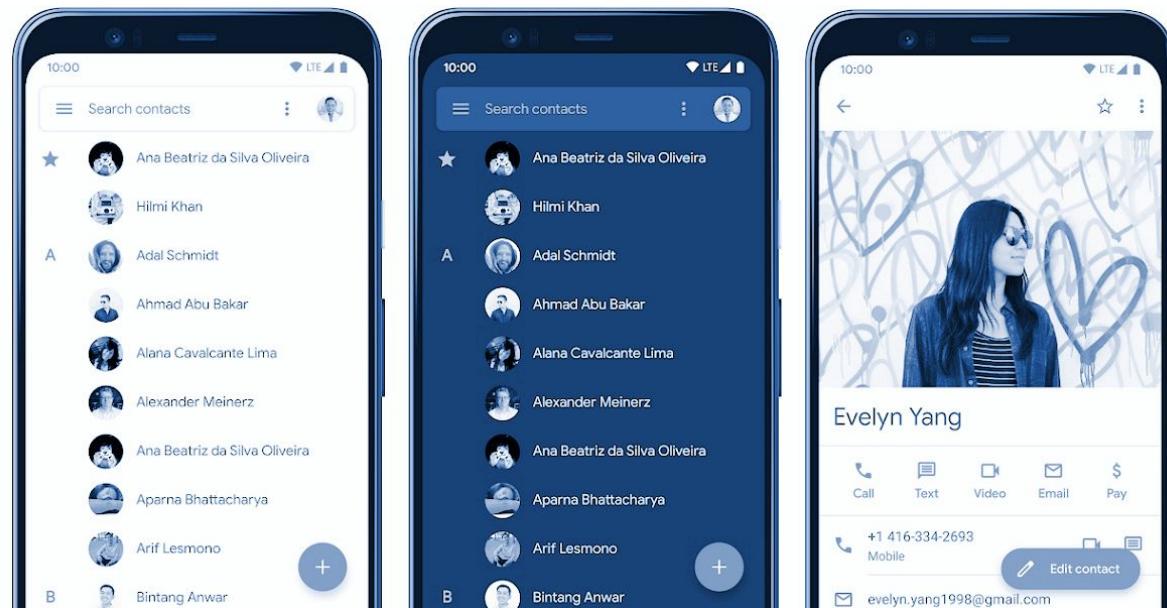


Tipos de índices

Para poder entender la finalidad de un índice, pensemos en la agenda de contactos en nuestro smartphone.

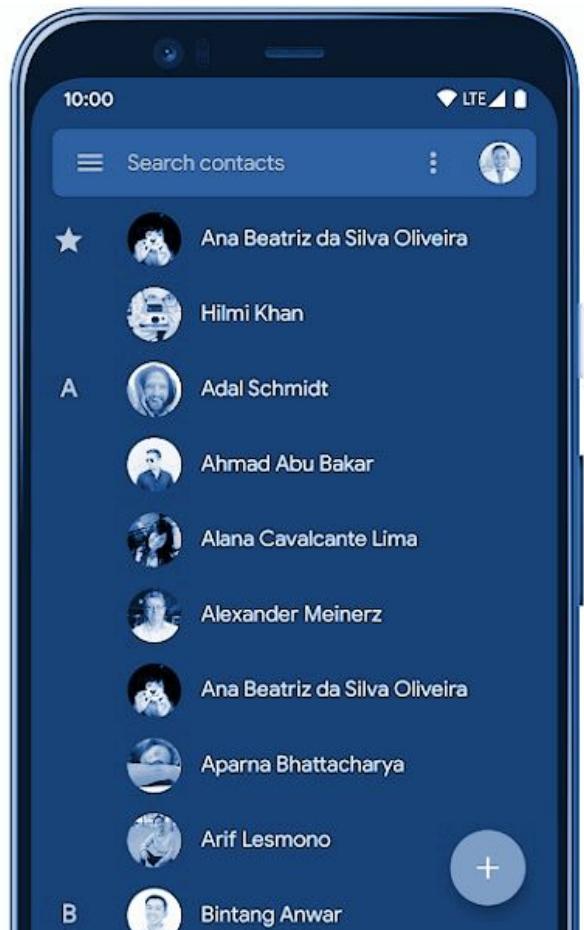
Esta agenda asigna un ordenamiento alfabético a los contactos almacenados en la misma.

Al igual que en una agenda, SQL recomienda siempre dar un índice, numérico en este caso, a cada uno de los registros que almacenemos en una tabla.



Tipos de índices

Y de la misma forma en la cual la agenda de contactos nos permite ubicar más rápido una persona para llamarla o enviarle un mensaje de texto, las bb.dd SQL toman esta filosofía para crear índices que permitan encontrar más rápidamente uno o más registros en una consulta de selección.

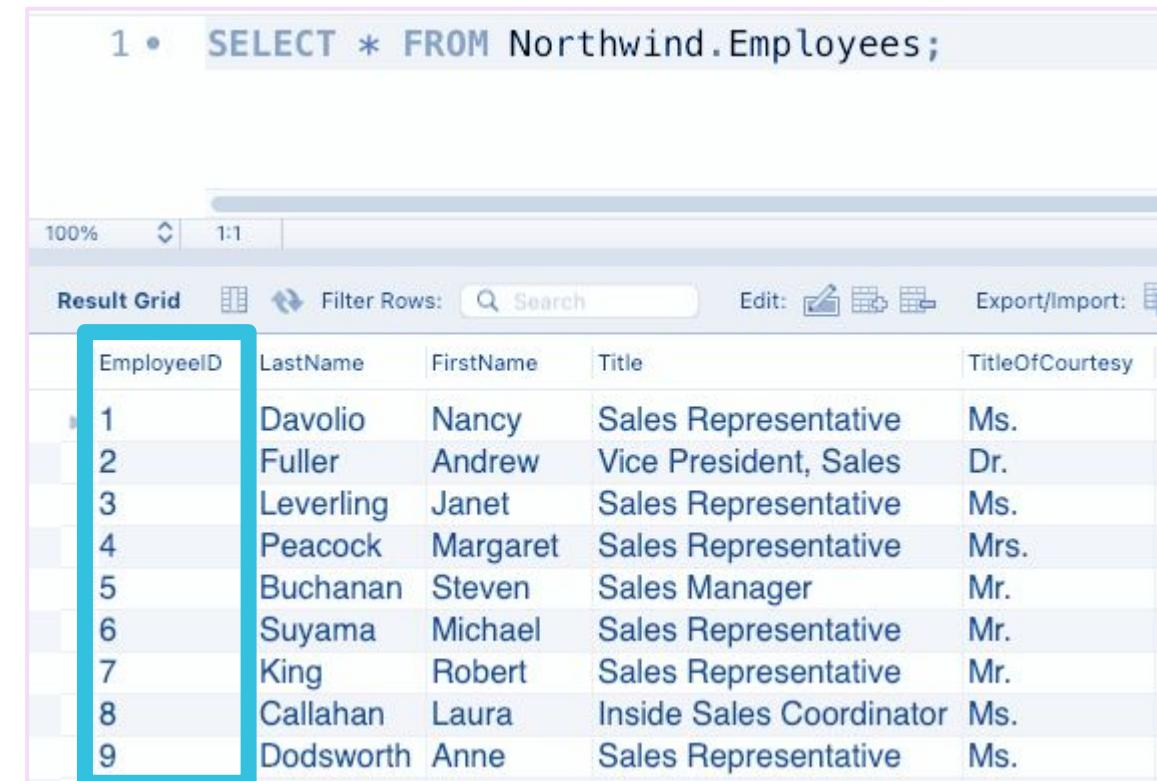


Tipos de índices

Por ello, cuando creamos una tabla, siempre es necesario crear un campo índice como primer campo de la tabla en cuestión.

Cuando se crea un índice en una tabla, la bb.dd crea una copia ordenada de las columnas del índice, lo que le permite buscar más rápidamente los valores de las filas que cumplen con los criterios de búsqueda, acelerando las consultas que buscan en grandes conjuntos de datos.

1 • `SELECT * FROM Northwind.Employees;`

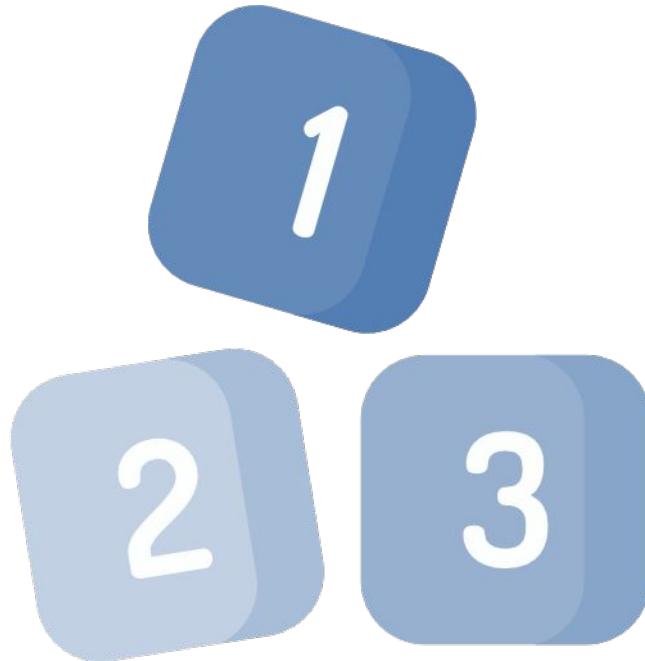


EmployeeID	LastName	FirstName	Title	TitleOfCourtesy
1	Davolio	Nancy	Sales Representative	Ms.
2	Fuller	Andrew	Vice President, Sales	Dr.
3	Leverling	Janet	Sales Representative	Ms.
4	Peacock	Margaret	Sales Representative	Mrs.
5	Buchanan	Steven	Sales Manager	Mr.
6	Suyama	Michael	Sales Representative	Mr.
7	King	Robert	Sales Representative	Mr.
8	Callahan	Laura	Inside Sales Coordinator	Ms.
9	Dodsworth	Anne	Sales Representative	Ms.

Tipos de índices

Los índices se pueden aplicar a una o varias columnas en una tabla, y hay varios tipos de índices que se pueden utilizar dependiendo del tipo de datos y las operaciones que se realizan en la tabla. Entre los diferentes índices que podemos definir en las tablas relacionales, encontramos los siguientes tipos:

- Primary Key
- Foreign Key
- Candidate Key
- Concatenated Key



Primary key



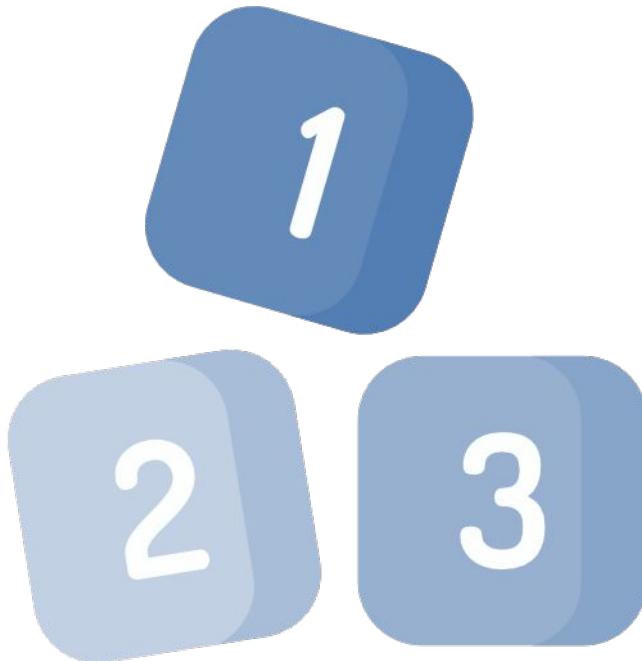
Tipos de índices

Primary Key

También llamado llave o clave primaria, hace que el registro sea único y, obligatoriamente, no nulo.

PK = primary key (*es la forma de abreviarla que se aplica en las definiciones de claves al momento de diseñar una tabla*).

Una clave primaria comprende de esta manera una columna o conjunto de columnas. No puede haber dos filas en una tabla que tengan la misma clave primaria.



Foreign key

Tipos de índices

Foreign Key

También llamada clave foránea, clave secundaria, o clave externa, puede ser -o no- una clave primaria dentro de una tabla.

Su característica convierte a este tipo de clave en el punto de enlace con otra tabla donde ésta (*foreign key*) tiene el rol de *primary key*.



FK = foreign key

Ejemplo de Primary & Foreign keys

Tipos de índices

	PK	Contactos						FK
	contactoID	nombre	apellido	domicilio	telefono	email	IDTipoTel	
1	1	Fernando	Moon	Tandil 1234	1122223333	fernando.luna@istea.com.ar	5	
2	2	Omar	Mercado	Tandil 1235	1122223444	omar.mercado@istea.com.ar	2	
3	3	Pepe	Argento	Bonifacio 1234	111122224444	pepe.argoento@quefamilia.com	NULL	
4	4	Moni	Argento	Bonifacio 1234	1122229999	moni.argoento@quefamilia.com	4	
5	5	Julian	Moon	Tandil 1234	1199997766	juli.moon@gmail.com	5	
6	6	Joe	McMillian	Texas 123	14455667788	jmm@cardiffcomputers.com	NULL	

Veamos un ejemplo de una tabla de contactos, similar a la que creamos en nuestras primeras clases de SQL. Aquí podemos apreciar que, las claves foráneas, a diferencia de las claves primarias, repetirán varias veces un valor en la tabla donde ofician como tales.

Solo hacen de nexo con otra tabla.

Tipos de índices

PK

	IDTipoTel	Descripcion
1	1	N/A
2	2	Particular
3	3	Trabajo
4	4	Móvil
5	5	Otro
6	6	Skype
7	7	Satélital

TiposDeTelefono

PK

Contactos

FK

	contactoID	nombre	apellido	domicilio	telefono	email	IDTipoTel
1	1	Fernando	Moon	Tandil 1234	1122223333	fernando.luna@istea.com.ar	5
2	2	Omar	Mercado	Tandil 1235	1122223444	omar.mercado@istea.com.ar	2
3	3	Pepe	Argento	Bonifacio 1234	111122224444	pepe.argoento@quefamilia.com	NULL
4	4	Moni	Argento	Bonifacio 1234	1122229999	moni.argoento@quefamilia.com	4
5	5	Julian	Moon	Tandil 1234	1199997766	juli.moon@gmail.com	5
6	6	Joe	McMillian	Texas 123	14455667788	jmm@cardiffcomputers.com	NULL

Las claves primarias (**PK**) de aquellas tablas que se interpretan como secundarias, son las claves foráneas (**FK**) de tablas con mayor protagonismo.

Concatenated key



Tipos de índices

Concatenated Key

Es un tipo de clave que ayuda a encontrar la singularidad en una tabla combinando dos o más campos.

Las claves concatenadas suelen darse en determinadas situaciones donde dos o más campos deben combinarse. Uno de los ejemplos que podemos tomar como referencia es una tabla que almacene información del encabezado de una factura de compra.



Tipos de índices

Concatenated Key

Aunque en muchos casos puede haber en una tabla una clave primaria que gestione esto, las claves concatenadas pueden llegar a ser una alternativa para prevenir posibles otras fallas en un diseño que terminen arrastrando problemas hacia otros procesos laborales.



Tipos de índices

PK	CK					encabezadoFactura						
	IDFac	TipoFactura	Letra	PuntoVenta	NumeroFactura	IDCliente	FechaFactura	FechaVencimiento	NotaCredito	NotaDebito	Cancelada	
21375	FC	C	2	1234	123	6/7/2022	6/9/2022	0	0	False		
21376	FC	C	2	1235	202	6/7/2022	6/9/2022	0	0	False		
21377	FC	C	2	1236	229	6/7/2022	6/9/2022	0	0	False		
21378	FC	C	2	1237	213	6/7/2022	6/9/2022	0	0	False		
21379	FC	C	2	1238	147	7/7/2022	7/9/2022	0	0	False		
21380	FC	C	2	1239	308	7/7/2022	7/9/2022	0	0	False		
21381	FC	C	2	1240	211	7/7/2022	7/9/2022	0	0	False		
21382	FC	C	2	1241	607	7/7/2022	7/9/2022	0	0	False		
21383	FC	C	2	1242	217	7/7/2022	7/9/2022	0	0	False		
21384	FC	C	2	1243	1	8/7/2022	8/7/2022	0	0	False		

Implementación de una clave concatenada sobre una tabla de facturación. La combinación de los cuatro campos definidos después del índice primario, impedirá la duplicación de una factura teniendo en cuenta la combinación de todos ellos.



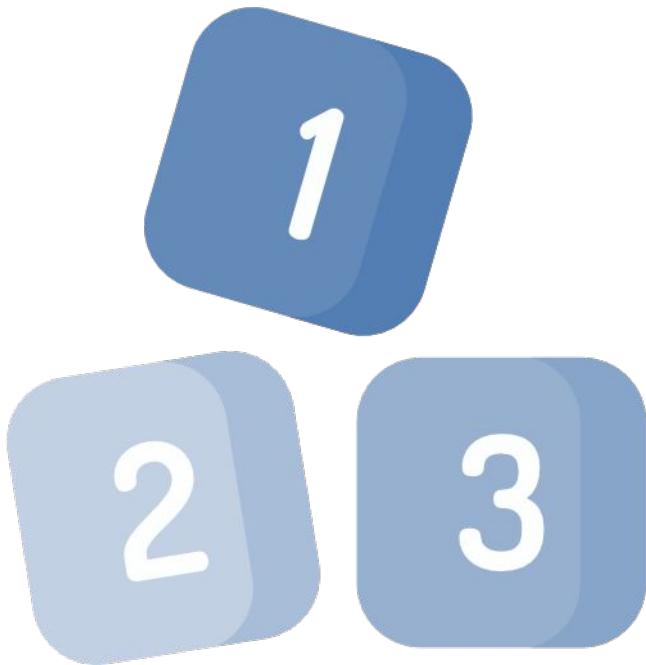
Candidate key



Tipos de índices

Candidate Key

Las claves candidatas suelen ser una clave primaria con una estructura numérica o alfanumérica, que usualmente ayudan a realizar búsquedas por algún valor más conocido que el que puede aportar una clave primaria convencional.

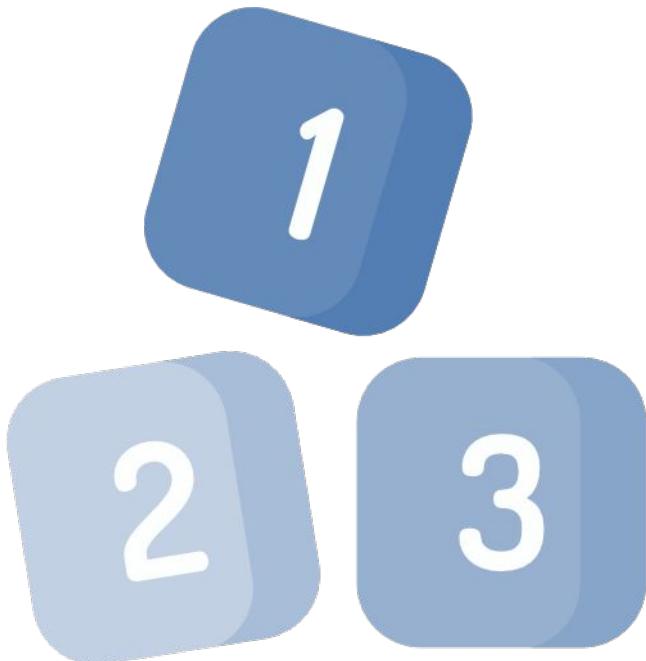


Tipos de índices

Candidate Key

Para entender esto, pensemos en una bb.dd. que gestiona los empleados de una empresa. En la tabla donde se almacenan estos empleados, seguramente existirá un campo llamado **empleadoID**, que oficiará de clave primaria.

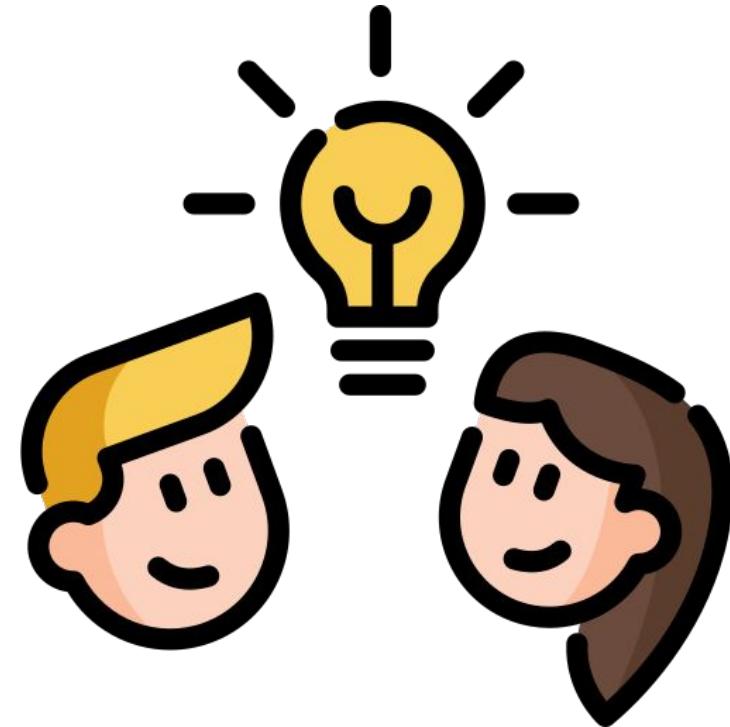
Pero, a su vez, contamos con otros campos, como ser, **legajo** y **documento** de identidad. Estos últimos, serán únicos también, y pueden ser tranquilamente denominados como campos de clave candidata.



Tipos de índices

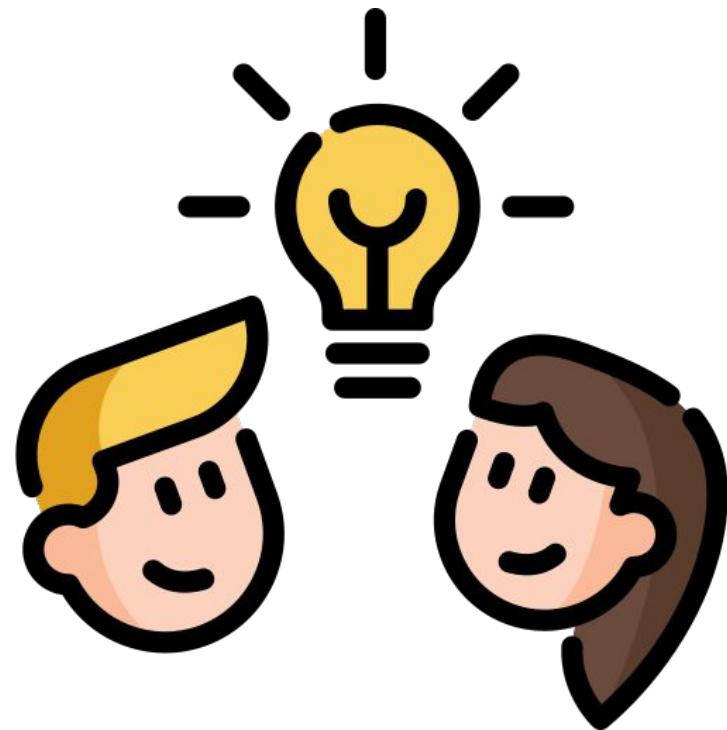
En resumen, los índices son una parte esencial del rendimiento de una base de datos relacional, ya que ayudan a mejorar la velocidad y eficiencia de las consultas.

Además de esto, los índices son el punto clave que le dieron origen al desarrollo efectivo del modelo relacional. Este desarrollo cuenta con una serie de Formas normales, las cuales permiten definir un fundamento teórico de porqué debemos separar la información en dos o más tablas y por qué, cuanta más información tenemos, más tablas deben ser las que se originen a partir de la misma.



Tipos de índices

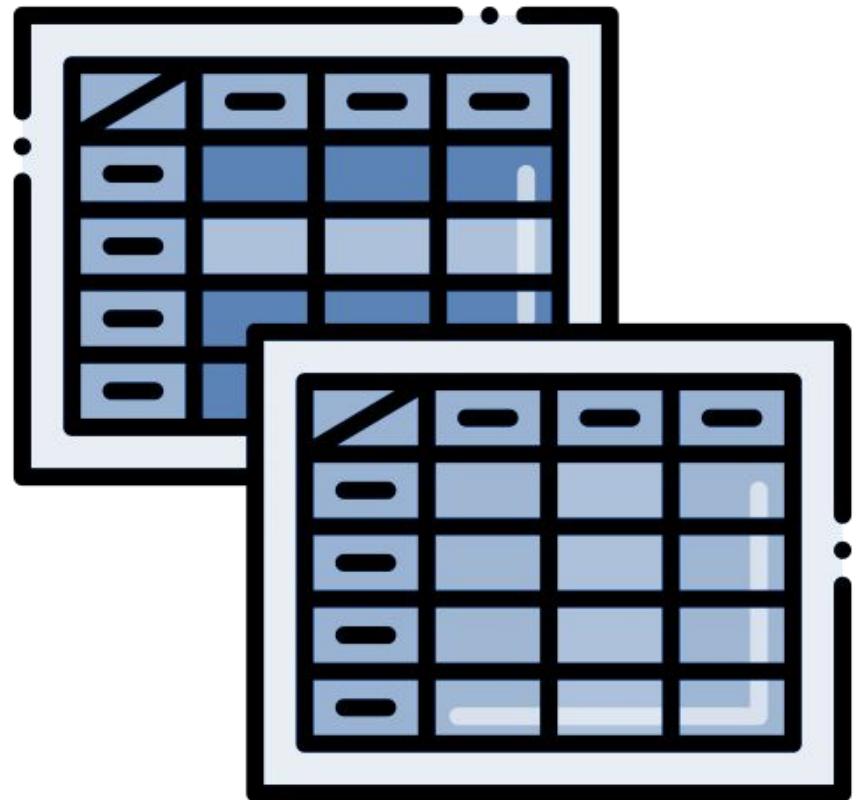
Si bien, cualquier persona tiende a pensar que, mientras más tablas tengamos en una bb.dd. esta será más grande y voluminosa, con el correr del tiempo y el incremento de la información histórica de cualquier sistema de software, el modelo relacional y el tipo de forma normal elegido hará que dicha base de datos sea más efectiva, rápida, y que su peso sea notablemente menor.



Formas normales

Formas Normales

En el contexto de bases de datos relacionales y SQL, las formas normales son una serie de reglas que se aplican para evitar redundancias y anomalías en los datos almacenados en las tablas de la base de datos.

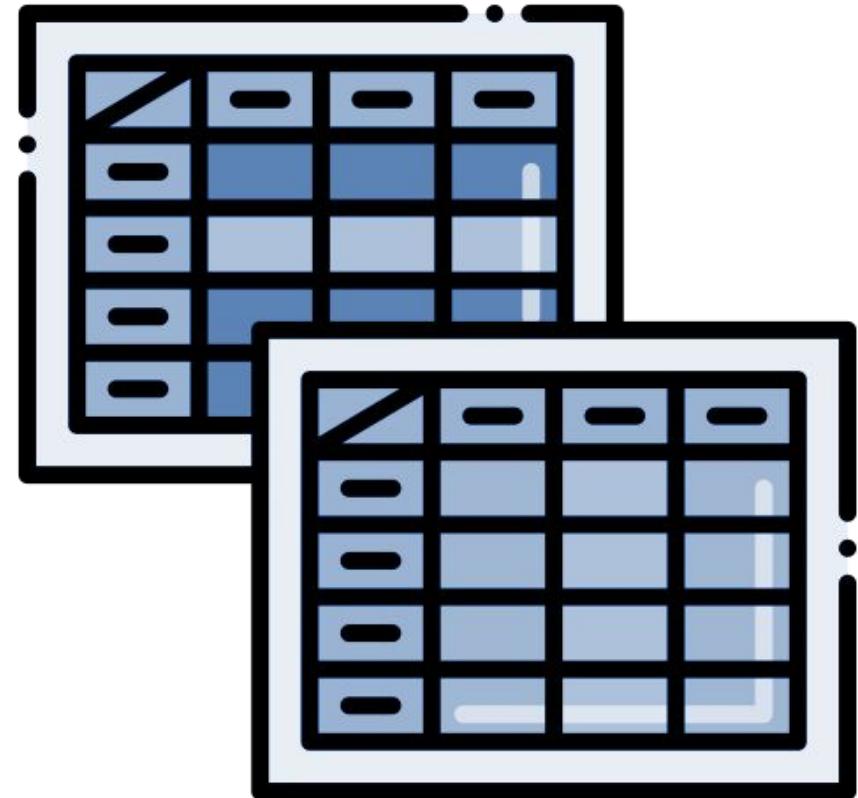


Formas Normales

Existen seis formas normales originales:

- 1^a forma normal
- 2^o forma normal
- 3^o forma normal
- Forma normal de *Boyce-Codd*
- 4^o forma normal
- 5^o forma normal

Pero en la práctica, las tres primeras formas normales son las más utilizadas.



Formas Normales

id	nombre_alumno	apellido_alumno	nombre_materia	nombre_profesor	apellido_profesor
1	Juan	Pérez	Matemáticas	Carlos	González
2	María	García	Matemáticas	Carlos	González
3	Pedro	Fernández	Matemáticas	Carlos	González
4	Laura	López	Matemáticas	Carlos	González
5	Carlos	Ramírez	Historia	Ana	Sánchez
6	Lucía	Martínez	Historia	Ana	Sánchez
7	Miguel	Hernández	Historia	Ana	Sánchez
8	Ana	Díaz	Física	Javier	Muñoz
9	Sofía	Alvarez	Física	Javier	Muñoz
10	Diego	Gómez	Física	Javier	Muñoz

Esta tabla estructura la información de alumnos, materias y profesores. Como podemos apreciar, cumple con el criterio de una tabla plana, tal como vimos en la clase inicial dedicada a entender las tablas SQL.

Si debemos aplicarle a esta tabla, la primera forma normal, deberíamos dividir estos datos en tablas más pequeñas, de modo que no existan campos repetidos.



Formas Normales

ALUMNOS

	id	nombre	apellido
▶	1	Juan	Pérez
	2	María	García
	3	Pedro	Fernández
	4	Laura	López
	5	Carlos	Ramírez
	6	Lucía	Martínez
	7	Miguel	Hernández
	8	Ana	Díaz
	9	Sofía	Alvarez
	10	Diego	Gómez

MATERIAS

	id	nombre
▶	1	Matemáticas
	2	Historia
	3	Física

PROFESORES

	id	nombre	apellido
▶	1	Carlos	González
	2	Ana	Sánchez
	3	Javier	Muñoz

Basándonos en la **1ª Forma Normal**, deberíamos crear tres tablas separadas: una para **Alumnos**, otra para **Materias** y otra para **Profesores**. Luego, crearemos una cuarta tabla la cual se relaciona a las tres anteriores, almacenando la información correspondiente a las relaciones.

Formas Normales

AlumnosMateriasProfesores

	id	id_alumno	id_materia	id_profesor
▶	1	1	1	1
	2	2	1	1
	3	3	1	1
	4	4	1	1
	5	5	2	2
	6	6	2	2
	7	7	2	2
	8	8	3	3
	9	9	3	3
	10	10	3	3

La cuarta tabla, llamada eventualmente **AlumnosMateriasProfesores**, será la más simple de todas, porque solo contendrá información de cada uno de los índices primarios de las tres tablas restantes.

Si bien aquí la información no es concisa para nuestra vista, esta forma es totalmente válida y hará que al crecer la bb.dd en contenido, éste comience a ser más eficiente con el correr del tiempo y del crecimiento en volumen de datos.

Segunda forma normal

Formas Normales (segunda forma normal)

Para aplicar la segunda forma normal a los datos que vimos anteriormente, es necesario que identifiquemos a las dependencias funcionales para luego separarlas en tablas independientes.

En este caso, la tabla **AlumnosMateriasProfesores** tiene una dependencia funcional parcial en el campo **nombre_materia** hacia el campo **nombre_profesor**, ya que los nombres de los profesores se repiten, (*o pueden repetirse en un futuro*), para diferentes materias.

MATERIAS

id	nombre
1	Matemáticas
2	Historia
3	Física

PROFESORES

id	nombre	apellido
1	Carlos	González
2	Ana	Sánchez
3	Javier	Muñoz

Formas Normales (segunda forma normal)

Por lo tanto, podemos separar esta dependencia funcional en una nueva tabla

ProfesoresMaterias, donde almacenaremos la relación entre los profesores y las materias que imparten.

De esta forma, ya sea para este modelo de datos o para un modelo de datos futuro, donde un profesor puede dictar más de una materia, la estructura de tablas de esta base de datos estará correctamente preparada.

MATERIAS

id	nombre
1	Matemáticas
2	Historia
3	Física

PROFESORES

id	nombre	apellido
1	Carlos	González
2	Ana	Sánchez
3	Javier	Muñoz

Formas Normales (segunda forma normal)

Por lo tanto, podemos separar esta dependencia funcional en una nueva tabla

ProfesoresMaterias, donde almacenaremos la relación entre los profesores y las materias que imparten.

De esta forma, ya sea para este modelo de datos o para un modelo de datos futuro, donde un profesor puede dictar más de una materia, la estructura de tablas de esta base de datos estará correctamente preparada.

MATERIAS

id	nombre
► 1	Matemáticas
2	Historia
3	Física

PROFESORES

id	nombre	apellido
► 1	Carlos	González
2	Ana	Sánchez
3	Javier	Muñoz

ProfesoresMaterias

id_profesor	id_materia
► 1	1
1	2
2	2
1	3
3	3



Otras formas normales

Otras Formas normales

Y para entender el resto de las formas normales, pensemos sobre lo trabajado hasta aquí. Solo con datos de alumnos, materias y profesores, terminamos creando 5 tablas.

Ahora, si además de registrar estos datos, debiéramos incluir el registro y control de **Aulas**, **Horarios**, **Matrículas**, y otros datos importantes de estudiantes y profesores, terminaremos sumergiéndonos seguramente en algún otro modelo relacional de la tercera o cuarta forma, o seguramente dentro de la forma *Boyce-Codd*.

MATERIAS

id	nombre
1	Matemáticas
2	Historia
3	Física

PROFESORES

id	nombre	apellido
1	Carlos	González
2	Ana	Sánchez
3	Javier	Muñoz

PROFESORESMATERIAS

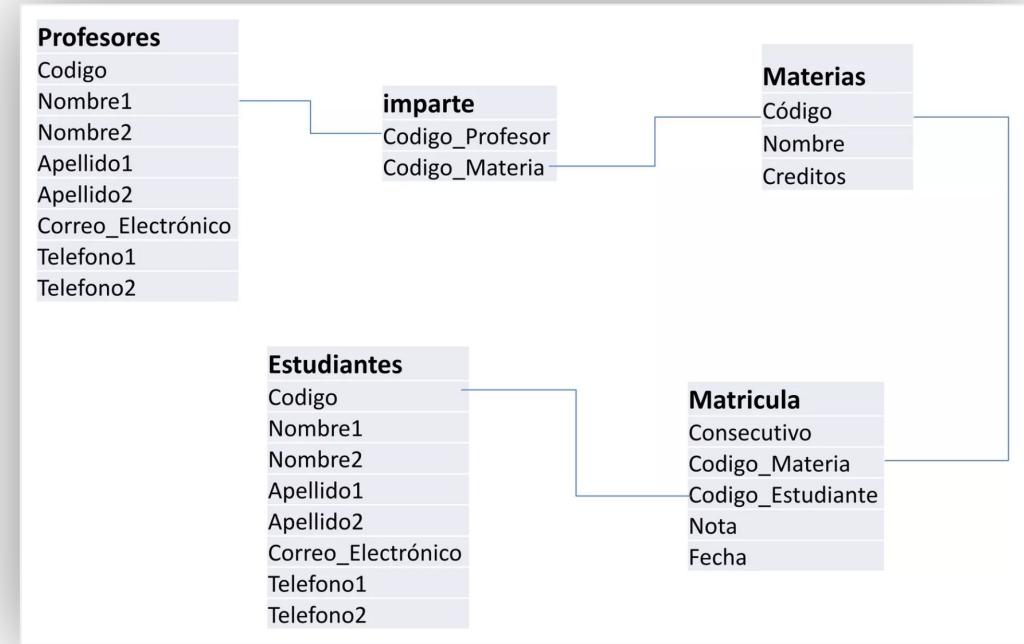
id_profesor	id_materia
1	1
1	2
2	2
1	3
3	3

ALUMNOS

id	nombre	apellido
1	Juan	Pérez
2	María	García
3	Pedro	Fernández
4	Laura	López
5	Carlos	Ramírez
6	Lucía	Martínez
7	Miguel	Hernández
8	Ana	Díaz
9	Sofía	Alvarez
10	Diego	Gómez

Otras Formas normales

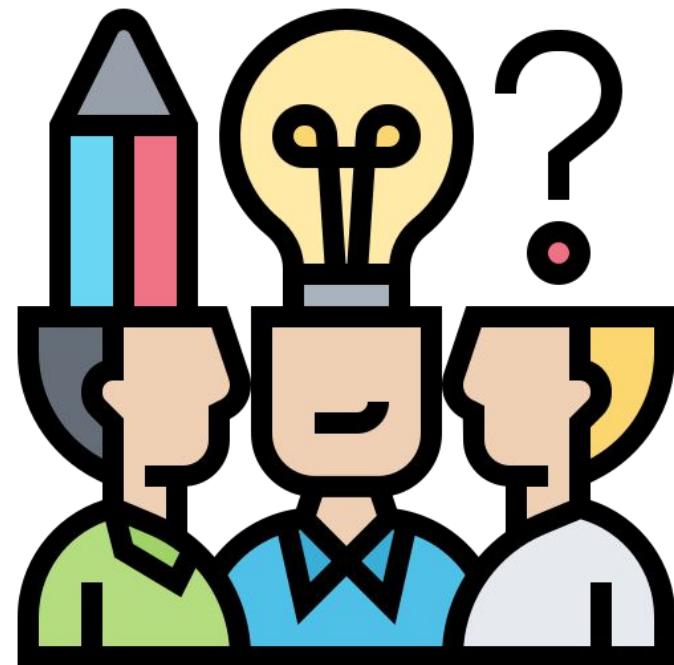
No es importante conocer en detalle todas estas formas normales, pero sí es clave que leamos sobre las mismas con tiempo, para así tenerlas presente y poder diseñar tablas en bases de datos, que sean eficientes y que podamos escalar en complejidad, mientras más datos simples necesitemos cruzar o almacenar en el modelo relacional que debamos construir.



Formas Normales

El uso de las formas normales es algo cotidiano, que si bien no debemos aprenderlo de memoria, sí es bueno ir teniendo el concepto general en la cabeza.

Te recomendamos repasar lo explicado hasta aquí, de cara a que puedas afianzar más la importancia de las formas normales, ya que en nuestro próximo encuentro volcaremos la teoría a el uso práctico de formas normales en tablas relacionales.

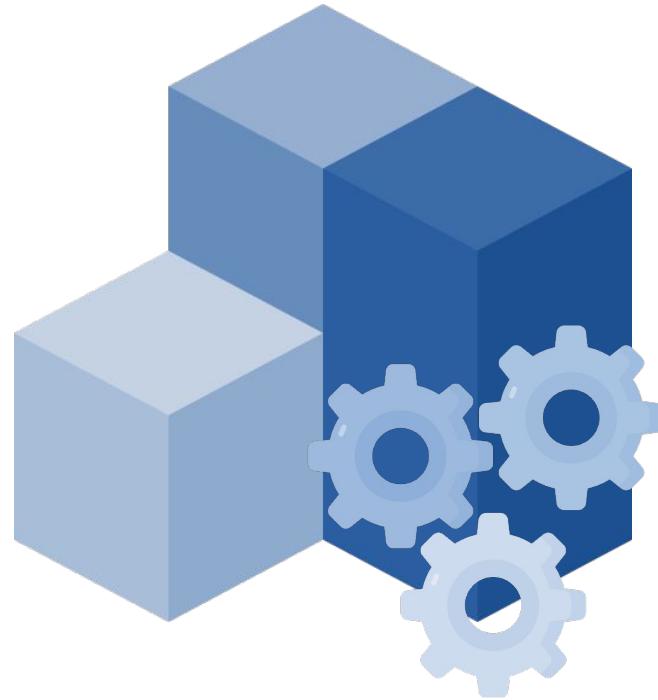


Subconsultas SQL

Subconsultas SQL

Las **Subconsultas** SQL son consultas insertadas dentro de otra consulta. Son ideales para cuando debemos filtrar datos por algún valor específico, donde sí conocemos su valor coloquial, pero no conocemos o recordamos su ID.

Siempre es más común tener presente una descripción o nombre asociado, que un ID. Con esta información podemos filtrar los datos, aplicando una subconsulta SQL.



Subconsultas SQL

Como muestra el ejemplo de código, una subconsulta es una nueva consulta SQL que puede ser ejecutada dentro de otra consulta SQL. Si te sientes como un personaje de la película **INCEPTION**, es normal... 😳 ... igual, entender esto, es más fácil de lo que parece a simple vista.



Subconsultas SQL

```
SELECT id, campo1, campo2
FROM tabla
WHERE campo2 = (SELECT id2
                  FROM otraTabla
                  WHERE campob = 'valor');;
```

Subconsultas SQL

Como muestra el ejemplo de código, una subconsulta es una nueva consulta SQL que puede ser ejecutada dentro de otra consulta SQL.

Si te sientes como un personaje de la película **INCEPTION**, es normal ... 😳 ... igual, entender esto, es más fácil de lo que parece a simple vista.



Subconsultas SQL

```
SELECT id, campo1, campo2  
FROM tabla  
WHERE campo2 = (SELECT id2  
                  FROM otraTabla  
                  WHERE campob = 'valor');
```

Subconsultas SQL

Supongamos que tenemos una consulta como la siguiente, donde **campo2** equivale a una clave foránea la cual es una clave primaria (*numérica*) en **otraTabla**.

Subconsultas SQL

```
SELECT id, campo1, campo2  
FROM tabla  
WHERE campo2 = ...;
```

Para ver el valor de dicha clave numérica en **otraTabla**, ejecutamos una consulta aislada, estricta (uso de =), la cual nos retornará un único registro y campo “**id2**”.

Subconsultas SQL

```
SELECT Id2 FROM otraTabla  
WHERE campob = 'valor';
```



Subconsultas SQL

Para resolver todo en un solo paso, incluimos la subconsulta SQL en búsqueda del ID correspondiente al valor coloquial que conocemos de memoria, encerrando la misma entre paréntesis.

Como esta subconsulta retorna un único registro y un único campo, ese valor será el que se aplique sobre el condicional establecido en la consulta principal.

Subconsultas SQL

```
SELECT id, campo1, campo2
FROM tabla
WHERE campo2 = (SELECT id2
                  FROM otraTabla
                  WHERE campob = 'valor');
```



Subconsultas SQL

Finalmente, será como haber escrito el número en cuestión dentro del condicional de la primera consulta, pero sin tener que saber el valor numérico en cuestión.

El uso de paréntesis en subconsultas hace que estas se comporten igual que en el mundo de las matemáticas que alguna vez aprendimos en la escuela: primero se resuelven los resultados entre paréntesis, para que luego se resuelva el resto de la ecuación.



Subconsultas SQL

```
SELECT id, campo1, campo2  
FROM tabla  
WHERE campo2 = 3;
```

Subconsultas SQL

Veamos un ejemplo no tan abstracto:

Aquí queremos obtener algunos campos de la tabla **Northwind.Products**, filtrando por un proveedor específico (**Suppliers**).

No recuerdo el código de **SupplierID**, pero sí recuerdo que la empresa se llama '*Exotic Liquids*'. De esta forma, armamos una subconsulta que nos retorne el **SupplierID** a partir del nombre del proveedor.

```
...  
Subconsultas SQL  
  
SELECT ProductID,  
       ProductName,  
       UnitPrice,  
       UnitsInStock  
FROM Northwind.Products  
WHERE SupplierID = (SELECT SupplierID  
                     FROM Suppliers  
                     WHERE CompanyName = 'Exotic Liquids');
```

Subconsultas SQL

También es totalmente válido utilizar en una subconsulta el condicional LIKE, aunque con este corremos riesgo de que nos retorne más de una coincidencia, lo cual puede no volver efectiva la consulta resultante.

Subconsultas SQL

```
...  
SELECT ProductID,  
       ProductName,  
       UnitPrice,  
       UnitsInStock  
FROM Northwind.Products  
WHERE SupplierID = (SELECT SupplierID  
                      FROM Suppliers  
                      WHERE CompanyName LIKE '%exotic%');
```

Case When

Case When

La cláusula **CASE** nos permite “*surfear*” entre diferentes condiciones y retornar un valor determinado cuando una de estas condiciones se cumple.

En el momento en el cual ocurre esto, la cláusula deja de comparar y devuelve el resultado indicado.



Case When

Para que la cláusula **CASE** funcione de manera óptima, debemos combinarla con la sentencia **WHEN**.

Si ninguna de las condiciones se cumple, tenemos la posibilidad de definir un **ELSE** para retornar un valor por defecto.



Case When

Aquí tenemos un ejemplo práctico,
combinando CASE - WHEN - ELSE.

Queremos visualizar información puntual
de la tabla Customers ordenando la
misma por País (Country).

Si este campo es nulo o está vacío,
entonces ordenamos por Ciudad.

```
... Fetch  
  
SELECT CompanyName, City, Country  
FROM Customers  
ORDER BY  
(CASE  
    WHEN Country IS NULL OR Country = "" THEN City  
    ELSE Country  
END);
```

Case When

El uso de **Case - When** puede aplicarse en diferentes lugares de una consulta de selección.

Por ejemplo, sobre tablas no normalizadas, podríamos reemplazar los campos con valores vacíos o nulos por información predeterminada.

En el siguiente ejemplo vemos cómo reemplazamos los datos vacíos en el campo **Region**, por un valor predeterminado: ‘N/A’.

```
... Fetch  
  
SELECT EmployeeID,  
       LastName,  
       FirstName,  
       Title,  
       CASE  
           WHEN Region = '' THEN 'N/A'  
           ELSE Region  
       END  
       As Region  
FROM Northwind.Employees;
```

Case When

El uso de **Case - When** puede aplicarse en diferentes lugares de una consulta de selección.

Por ejemplo, sobre tablas no normalizadas, podríamos reemplazar los campos con valores vacíos o nulos por información predeterminada.

En el siguiente ejemplo vemos cómo reemplazamos los datos vacíos en el campo **Region**, por un valor predeterminado: ‘N/A’.

```
... Fetch  
  
SELECT EmployeeID,  
       LastName,  
       FirstName,  
       Title,  
       CASE  
           WHEN Region = '' THEN 'N/A'  
           ELSE Region  
       END  
       As Region  
FROM Northwind.Employees;
```

Sección práctica

Nos piden generar una serie de reportes sobre la tabla Products de la base de datos Northwind.

Estos reportes son gerenciales, por lo tanto debemos destacar claramente la información que la gerencia desea conocer.

Veamos los requisitos a continuación:



Prácticas

Necesitamos simplificar la visualización de datos de la tabla **Products**, presentando en una consulta de selección, los siguientes campos:

- **ProductID, ProductName, UnitPrice, UnitsInStock, ReorderLevel**

Sobre esta consulta de selección base, realiza las siguientes consignas:

1. Ejecuta una consulta de selección de todos estos datos, ordenando los mismos por:
 - a. **CategoryID, ProductName**
2. En una nueva consulta de selección con la base inicial:
 - a. Muestra una leyenda en el campo **ReorderLevel**, que diga '*Reponer Stock*', en aquellos productos donde el campo **UnitsInStock** esté por debajo de **ReorderLevel**
 - b. Ordena los productos por **ProductName**
3. Ejecuta una consulta de selección igual al Punto 1, agregando la siguiente condición
 - a. **CategoryID = (el id de la categoría llamada 'Seafood')**
 - i. utiliza una subconsulta SQL en esta condición



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



Argentina programa 4.0



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*

Clase 23: Bases de datos Relacionales

Tablas combinadas en consultas SQL

Agenda de hoy

- A. La importancia de consultar datos combinando tablas
- B. Establecer relaciones entre tablas
 - a. INNER JOIN
 - b. LEFT JOIN
 - c. RIGHT JOIN
 - d. OUTER JOIN
 - e. UNION ALL
- C. Combinar JOIN y Subconsultas SQL



La importancia de consultar datos combinando tablas

Continuamos con el proceso de aprendizaje iniciado la clase anterior. Luego de una teoría bastante intensa, tomaremos al lenguaje SQL para que nos ayude a representar gráficamente los aspectos más importantes entre tablas relacionales y cómo aprovechar datos esparcidos en dos o más tablas, para armar una consulta con la información que más importancia tenga.



La importancia de consultar datos combinando tablas

Como hablamos al inicio de la cursada, mientras repasábamos los fundamentos de las bases de datos relacionales, es clave que tengamos estructurada la información en diferentes tablas.

A la larga, esta estructura será mucho más clara y limpia para nosotros, y gracias a SQL, podremos tomar aquello que necesitemos de cada tabla, de una forma prolíja y clara.



La importancia de consultar datos combinando tablas

En la estructura básica de creación de una base de datos relacional, las tablas y los índices primarios y foráneos son lo que mayormente utilizaremos cuando diseñamos una bb.dd de este tipo.

Las claves primarias son importantes para poder optimizar la búsqueda de registros resultantes de consultas donde aplicamos cláusulas WHERE simples o múltiples.



La importancia de consultar datos combinando tablas

Mientras que, los índices o claves foráneas, son importantes para cuando tenemos que traer múltiples datos de diferentes tablas, y visualizar una estructura de registros uniforme y coherente.

Allí, las claves foráneas son el nexo específico, no solo para establecer relación entre datos, sino también para evitar que la información almacenada en tablas, quede huérfana o inconsistente.



La importancia de consultar datos combinando tablas

Y, para comprender cómo podemos obtener múltiples datos de múltiples tablas, haremos un repaso sobre determinadas cláusulas importantes que incluye SQL.

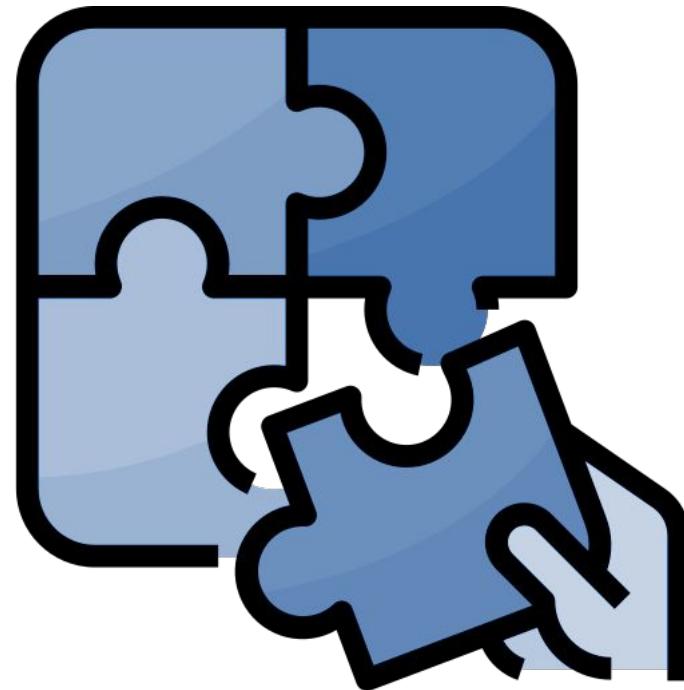
Estas cláusulas, se apoyan en las bases de la teoría de conjuntos de las matemáticas, para poder comprender más fácilmente qué información obtendremos, cuando tomamos información de múltiples tablas de datos.



JOIN

Establecer relaciones entre tablas

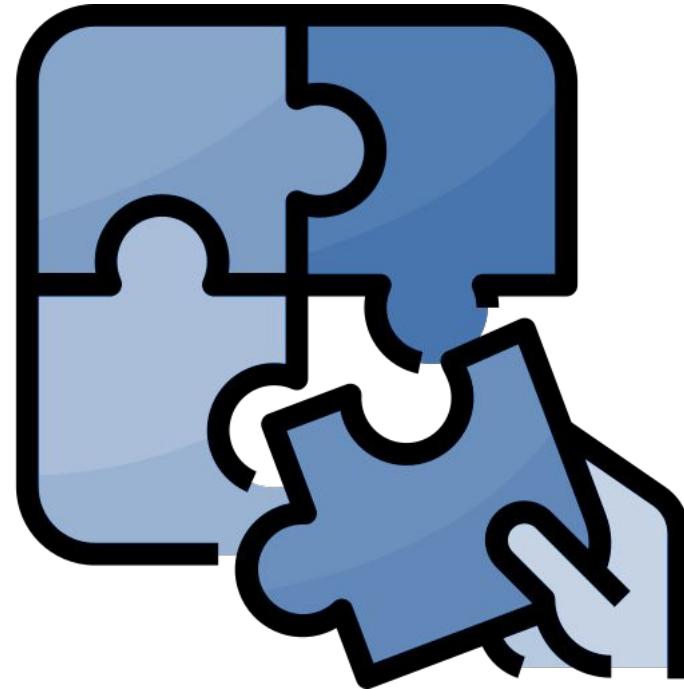
Como mencionamos recientemente, en el momento de trabajar bajo un modelo relacional es muy probable que, al ejecutar consultas de selección, debamos obtener datos que se complementan entre sí, desde dos o más tablas diferentes.



Establecer relaciones entre tablas

Para poder solucionar esto, contamos con la cláusula JOIN y sus diferentes variantes.

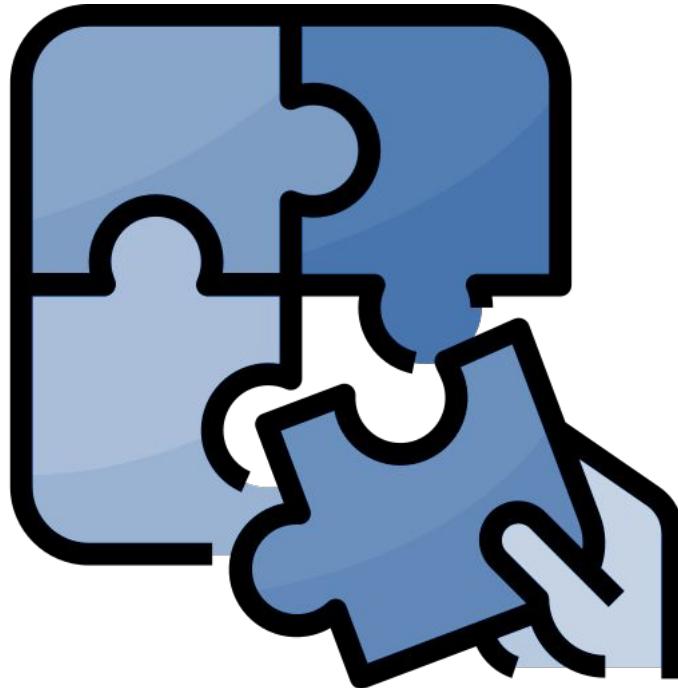
La misma forma parte del lenguaje SQL, y nos permite establecer el mecanismo de conexión entre dos o más tablas para ejecutar, en una consulta resultante, la conjunción de datos que éstas comparten entre sí.



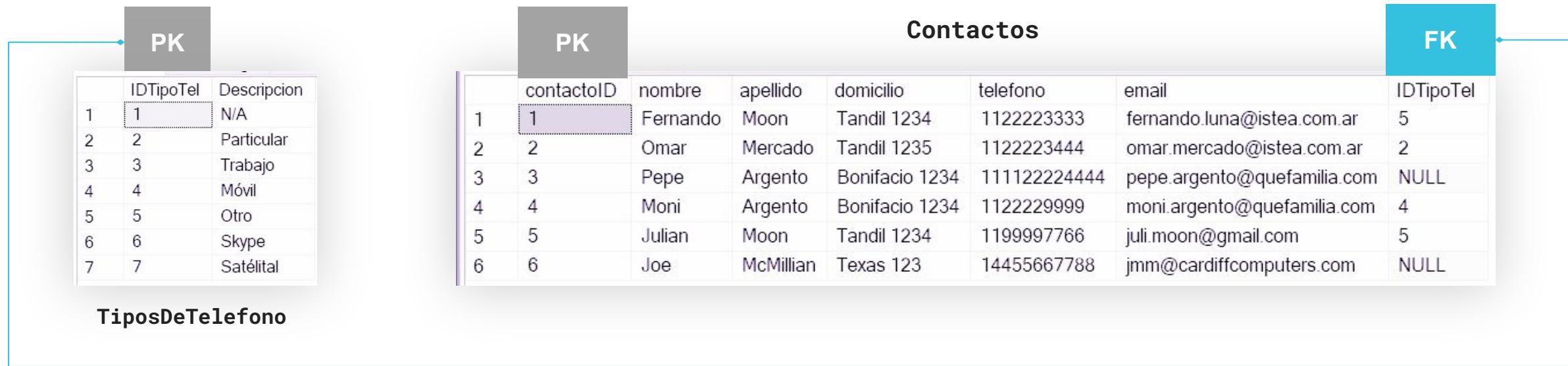
Establecer relaciones entre tablas

JOIN se ocupa de definir cuál es el medio de relación entre estas tablas, basándose en campos en común que estas tengan:

- claves primarias
- claves foráneas



Establecer relaciones entre tablas



TiposDeTelefono		Contactos						
IDTipoTel	Descripcion	contactoID	nombre	apellido	domicilio	telefono	email	IDTipoTel
1	N/A	1	Fernando	Moon	Tandil 1234	1122223333	fernando.luna@istea.com.ar	5
2	Particular	2	Omar	Mercado	Tandil 1235	1122223444	omar.mercado@istea.com.ar	2
3	Trabajo	3	Pepe	Argento	Bonifacio 1234	111122224444	pepe.argoento@quefamilia.com	NULL
4	Móvil	4	Moni	Argento	Bonifacio 1234	1122229999	moni.argoento@quefamilia.com	4
5	Otro	5	Julian	Moon	Tandil 1234	1199997766	juli.moon@gmail.com	5
6	Skype	6	Joe	McMillian	Texas 123	14455667788	jmm@cardiffcomputers.com	NULL
7	Satélital							

Retomamos el ejemplo de la tabla **Contactos** y **TiposDeTelefono** que utilizamos en nuestro encuentro anterior. Tomaremos registros específicos de cada una de ellas, utilizando JOIN.

Establecer relaciones entre tablas

Sobre el set de datos del ejemplo anterior, construimos la siguiente consulta de Selección, combinando ambas tablas. Aquí vemos en acción la cláusula **JOIN** junto con la palabra reservada **ON**.

Analicemos a continuación esta estructura de consulta SQL.



Subconsultas SQL

```
SELECT nombre, apellido, domicilio, email, teléfono, descripción
  FROM Contactos C
 JOIN TiposDeTeléfono T
    ON C.idTipoTel = T.idTipoTel;
```

Establecer relaciones entre tablas

```
Subconsultas SQL  
  
SELECT nombre, apellido, domicilio, email, teléfono, descripcion  
FROM Contactos C  
JOIN TiposDeTelefono T  
ON C.idTipoTel = T.idTipoTel;
```

JOIN se ocupa de unir la tabla **Contactos**, mencionada en la
línea de arriba, con la tabla **TiposDeTelefono**.



Establecer relaciones entre tablas

```
Subconsultas SQL  
  
SELECT nombre, apellido, domicilio, email, teléfono, descripcion  
      FROM Contactos C  
      JOIN TiposDeTelefono T  
        ON C.idTipoTel = T.idTipoTel;
```

La cláusula **ON** se utiliza para definir el punto en común entre estas dos tablas. En este caso, el campo **idTipoTel** es el nexo, ya que en la tabla **TiposDeTelefono** es la clave primaria, y en la tabla **Contactos** es la clave foránea.



Establecer relaciones entre tablas

```
Subconsultas SQL  
  
SELECT nombre, apellido, domicilio, email, teléfono, descripcion  
FROM Contactos C  
JOIN TiposDeTelefono T  
ON C.idTipoTel = T.idTipoTel;
```

Las letras que vemos por allí sueltas, son representaciones, o Alias, para cada una de las tablas. Cuando comenzamos a traer columnas de diferentes tablas, siempre conviene aplicar el uso de alias en tablas, para que a simple vista veamos de dónde proviene cada columna o campo.



Establecer relaciones entre tablas

Subconsultas SQL

```
SELECT C.nombre, C.apellido, C.domicilio, C.email, C.telefono, T.descripcion
  FROM Contactos C
  JOIN TiposDeTelefono T
    ON C.idTipoTel = T.idTipoTel;
```

Y cuando definimos alias en las tablas, es necesario aplicarlo también en la definición de las columnas que utilizaremos. En algunos motores SQL se resuelve automáticamente, pero en otros puede generar error, sobre todo cuando hay ambigüedad en el nombre de las columnas.

Esta sería la estructura correcta para la consulta de selección que utiliza JOIN.

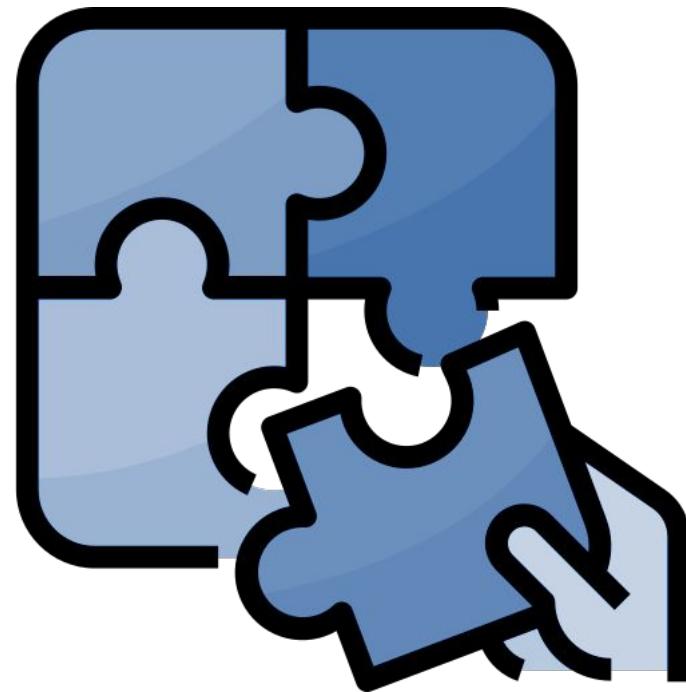


Tipos de JOIN

Tipos de JOIN

JOIN se puede utilizar de manera simple, tal como vimos en el ejemplo de consulta anterior, aunque en realidad dispone de varios tipos de JOIN.

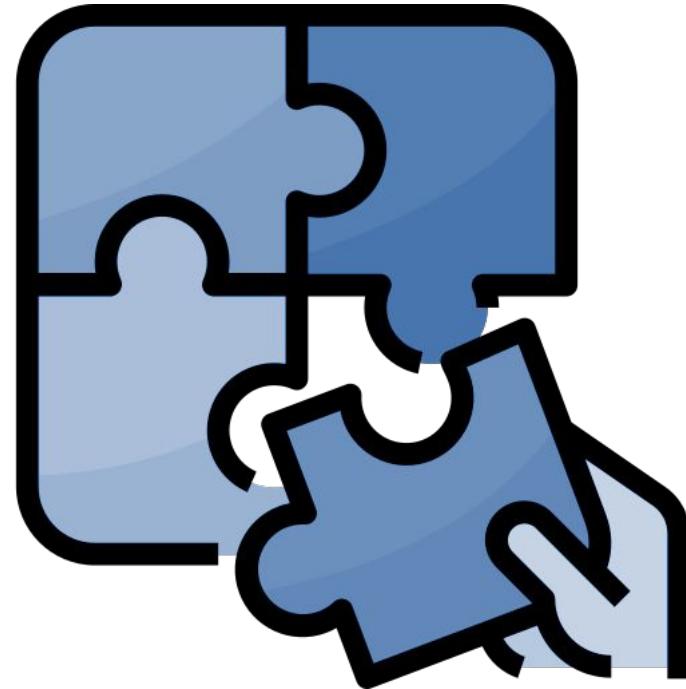
En la aplicación de sus diferentes tipos, entra en juego lo que se denomina **tabla de mayor peso** y **tabla de menor peso**. Esto es el condicional que nos permitirá ubicar a las tablas en la posición que les corresponde.



Tipos de JOIN

Y, para una mejor representación de JOIN, también se acopla el modelo de lo que alguna vez vimos en la escuela primaria/secundaria, relacionada a la **Teoría de Conjuntos**.

Este, es el mejor modelo representativo, para que apliquemos en el aprendizaje del uso de consultas SQL combinando dos o más tablas.



INNER JOIN

La profe te compartirá un archivo **.SQL** para que ejecutes su script sobre alguna base de datos nueva, existente, o de pruebas que tengas en MySQL.

Se crearán dos tablas de contactos y tipos de teléfono para abordar las diferentes **cláusulas SQL + JOIN**.

Tómate unos minutos para ejecutarlo y ver las tablas creadas. Recuerda agregar el comando USE con tu bb.dd. preferida.

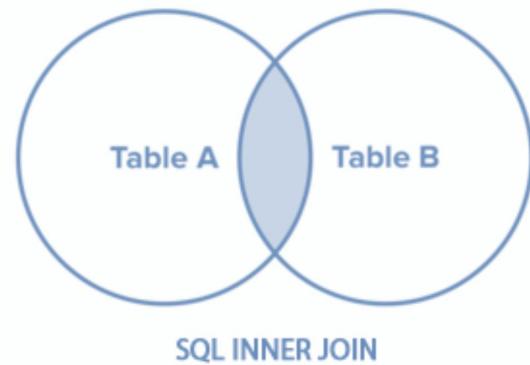


JOIN || INNER JOIN

INNER JOIN

Representación de **INNER JOIN**:

Trae todos los registros tomando la relación de ambas tablas y filtrando por aquellos registros donde sí hay un “*match*”.



INNER JOIN

El uso de JOIN o INNER JOIN es indistinto en SQL. Cualquiera de estas dos cláusulas nos devolverán el mismo resultado.

```
... JOIN  
  
SELECT nombreCompleto,  
       Email,  
       Telefono,  
       Descripcion  
  FROM Contactos C  
 INNER JOIN TiposDeTelefono T  
    ON C.IdTipoTel = T.IdTipoTel;
```



INNER JOIN

Si existen datos del tipo **null** en la tabla contactos, donde ésta enlaza con la tabla **TiposDeTelefono**, los mismos no serán visualizados en la consulta resultante.

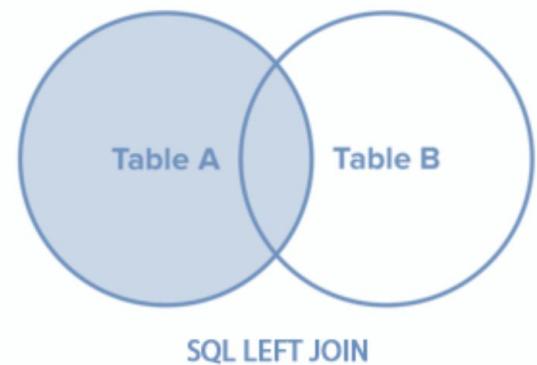
nombreCompleto	Email	Telefono	Descripcion
Cameron Howe	cameron@mutiny.com	11-4455-6699	Móvil
Gordon Clark	gordou@mutiny.com	11-4110-0909	Oficina
Joe McMillian	joe@mutiny.com	11-4844-1001	Satelital
Diane Gould	diane@mutiny.com	11-4844-1050	Skype
Malcolm Levitan	malc@mutiny.com	11-4110-5488	Oficina
Joanie Clark	joanie@mutiny.com	11-4810-1010	Personal
Haley Clark	hal@mutiny.com	11-4810-1010	Personal

LEFT JOIN

LEFT JOIN

Representación de **LEFT JOIN**:

Trae todos los registros de la tabla izquierda o de mayor peso, aplicando la relación de ambas tablas.



LEFT JOIN

Representación de **LEFT JOIN**:

Trae todos los registros de la tabla izquierda o de mayor peso, aplicando la relación de ambas tablas.

```
JOIN  
  
SELECT nombreCompleto,  
       Email,  
       Telefono,  
       Descripcion  
FROM Contactos C  
LEFT JOIN TiposDeTelefono T  
ON C.IdTipoTel = T.IdTipoTel;
```

LEFT JOIN

Si existen datos del tipo **null** en la tabla contactos, donde ésta enlaza con la tabla **TiposDeTelefono**, estos serán incluidos igual en la consulta resultante, completando con **null** el valor del campo **descripcion**.

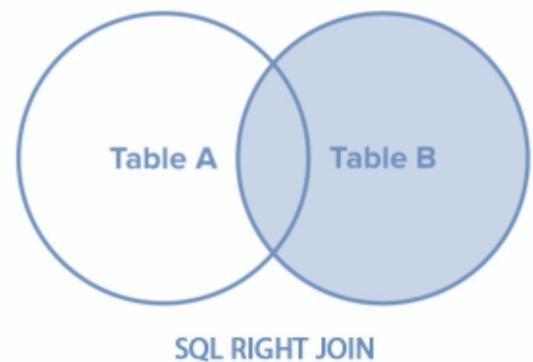
nombreCompleto	Email	Telefono	Descripcion
Cameron Howe	cameron@mutiny.com	11-4455-6699	Móvil
Donna Clark	donna@mutiny.com	11-4455-0000	NULL
Gordon Clark	gordou@mutiny.com	11-4110-0909	Oficina
Joe McMillian	joe@mutiny.com	11-4844-1001	Satelital
John Bosworth	bos@mutiny.com	11-4844-1000	NULL
Yo-Yo Engberk	yoyo@mutiny.com	11-4844-1091	NULL
Diane Gould	diane@mutiny.com	11-4844-1050	Skype
Malcolm Levitan	malc@mutiny.com	11-4110-5488	Oficina
Joanie Clark	joanie@mutiny.com	11-4810-1010	Personal
Haley Clark	hal@mutiny.com	11-4810-1010	Personal

RIGHT JOIN

RIGHT JOIN

Representación de **RIGHT JOIN**:

Trae todos los registros de la tabla derecha,
aplicando la relación de ambas tablas, y
haciendo el match con aquellos registros
relacionados desde la tabla izquierda.



RIGHT JOIN

Representación de **RIGHT JOIN**:

En este caso, se invierte la lógica respecto al uso de LEFT JOIN. Los registros que tendrán prioridad en la consulta resultante, serán los de la tabla **TiposDeTelefono**.

```
JOIN  
  
SELECT nombreCompleto,  
       Email,  
       Telefono,  
       Descripcion  
FROM Contactos C  
RIGHT JOIN TiposDeTelefono T  
ON C.IdTipoTel = T.IdTipoTel;
```

RIGHT JOIN

Todos aquellos datos provenientes de la tabla **Contactos** que no posean relación con los datos provenientes de la tabla **TiposDeTelefono** se completarán con valores **null**.

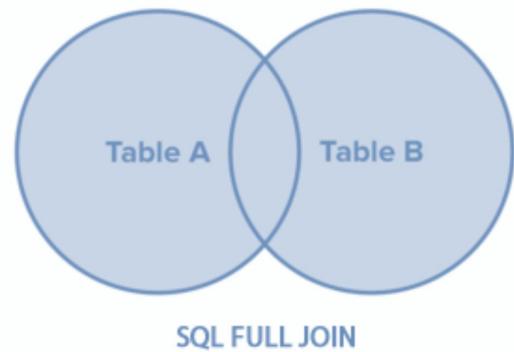
nombreCompleto	Email	Telefono	Descripcion
Haley Clark	hal@mutiny.com	11-4810-1010	Personal
Joanie Clark	joanie@mutiny.com	11-4810-1010	Personal
Malcolm Levitan	malc@mutiny.com	11-4110-5488	Oficina
Gordon Clark	gordou@mutiny.com	11-4110-0909	Oficina
Cameron Howe	cameron@mutiny.com	11-4455-6699	Móvil
Diane Gould	diane@mutiny.com	11-4844-1050	Skype
NULL	NULL	NULL	WhatsApp
Joe McMillian	joe@mutiny.com	11-4844-1001	Satelital
NULL	NULL	NULL	Otro
NULL	NULL	NULL	IP
NULL	NULL	NULL	Semipúblico
NULL	NULL	NULL	Recepción

OUTER JOIN || FULL OUTER JOIN

OUTER JOIN || FULL OUTER JOIN

Representación de **OUTER JOIN**:

Trae todos los registros de ambas tablas,
coincidiendo aquellos donde hay relación, y
completando con null aquellos donde no la haya.



OUTER JOIN || FULL OUTER JOIN

En el motor MySQL esta consulta no surte efecto alguno, porque **OUTER JOIN** no está disponible en el mismo.

Solo forma parte del motor de **Microsoft SQL SERVER**. En este último, trae todos los registros de ambas tablas combinando las coincidencias, y completando ambos extremos con valores **null** donde no haya coincidencias.



UNION ALL

UNION ALL

La cláusula **UNION ALL** se utiliza para combinar los resultados de dos o más consultas en una sola tabla resultante, devolviendo todas las filas, incluso si hay duplicados.



JOIN

```
SELECT NombreCompleto, Telefono  
FROM Contactos  
UNION ALL  
SELECT IdTipoTel, Descripcion  
FROM TiposDeTelefono;
```

UNION ALL

En este caso, para utilizar UNION ALL, debemos solicitar la misma cantidad de campos de una tabla y de la otra, más aún si no coinciden en datos. El resultado sobre nuestro modelo de datos, no es muy efectivo que digamos.

NombreCompleto	Telefono
Cameron Howe	11-4455-6699
Donna Clark	11-4455-0000
Gordon Clark	11-4110-0909
Joe McMillian	11-4844-1001
John Bosworth	11-4844-1000
Yo-Yo Engberk	11-4844-1091
Diane Gould	11-4844-1050
Malcolm Levitan	11-4110-5488
Joanie Clark	11-4810-1010
Haley Clark	11-4810-1010
1	Personal
2	Oficina
3	Móvil
4	Skype
5	WhatsApp
6	Satelital
7	Otro
8	IP
9	Semipúblico
10	Recepción

Combinar JOIN y consultas SQL

Combinar JOIN y consultas SQL

Trabajaremos sobre la base de datos **Northwind**, ejecutando consultas combinadas.

Trabajaremos con las tablas **Orders**, **OrderDetails**, **Products**, **Customers**, **Employees**, combinando los diferentes campos de estas para obtener consultas específicas.



Combinar JOIN y consultas SQL

La tabla **OrderDetails** y **Products** tienen en común el campo **ProductID**.

En nuestra tabla **OrderDetails** tenemos información muy resumida. Sobre los productos que se vendieron en dicha orden, sólo accedemos a ver su código. Veamos cómo realizar una consulta que nos muestre el código de producto, su nombre (*aquí la relación*), la cantidad vendida, y el precio unitario.

De paso, generamos un campo calculado llamado **Subtotal**, para ver cuál es la ganancia por cada producto vendido.

Result Grid	Filter Rows:	Search	Export:		
OrderID	ProductID	ProductName	Quantity	UnitPrice	Subtotal
▶ 10255	2	Chang	20	15.2	304
10255	16	Pavlova	35	13.9	486.5
10255	36	Inlagd Sill	25	15.2	380
10255	59	Raclette Courdavault	30	44	1320



Combinar JOIN y consultas SQL

Veamos ahora cómo combinar las tablas **Orders** y **Customers**. En esta oportunidad, visualizaremos los nombres de los clientes de tres órdenes de compra específicas.



Combinar JOIN y consultas SQL

Aprovechando que la tabla Orders no almacena información del cliente que solicitó la misma, más que su código de cliente (**CustomerID**), ejecutamos una consulta de selección que muestre la razón social (**companyName**) de los clientes para tres órdenes de compra específicas.

De paso, aprovechamos y sumamos una función escalar para eliminar la hora del campo Fecha de la orden (**Date**).

Result Grid	Filter Rows:		Search	Export:
OrderID	CustomerID	CompanyName	ContactName	Date
▶ 10249	TOMSP	Toms Spezialit?ten	Karin Josephs	1996-07-05
10252	SUPRD	Supr?mes d?lices	Pascale Cartrain	1996-07-09
10254	CHOPS	Chop-suey Chinese	Yang Wang	1996-07-11

Combinar JOIN y consultas SQL

Y, en este último ejemplo, incrementamos la complejidad de una consulta a múltiples tablas, trayendo información de las tablas **Orders**, **Customers** y **Employees**. Para ello, utilizaremos el uso de doble JOIN sobre la información pertinente.



Combinar JOIN y consultas SQL

La Tabla Orders almacena información del encabezado de una orden de compra, donde podemos visualizar tanto la información del cliente que la solicitó, y también del empleado o ejecutivo de ventas que la generó.

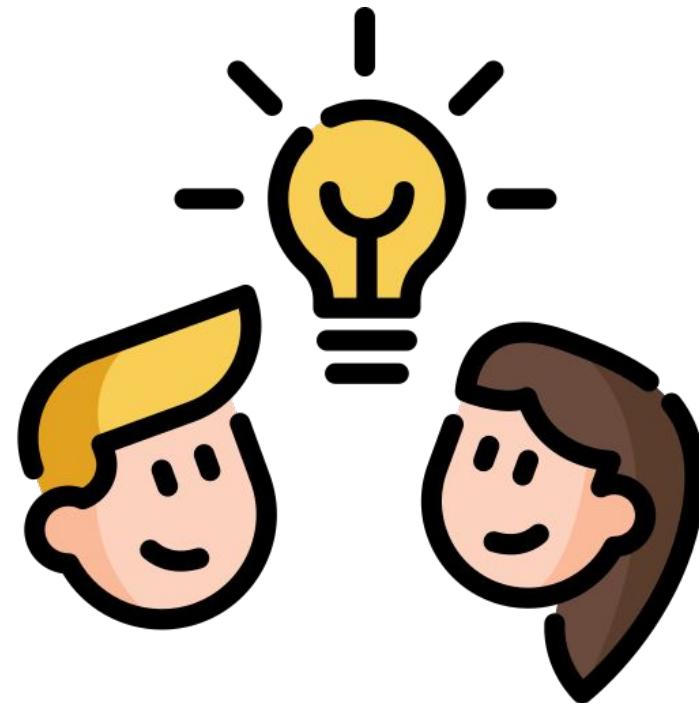
Aquí entra en juego el uso de **JOIN**, combinando las tablas **Employees** y **Customers**, con la tabla **Orders** en sí.

Result Grid	Filter Rows:	Search	Export:			
OrderID	CustomerID	CompanyName	EmployeeID	LastName	FirstName	
▶ 10260	OTTIK	Ottilies K?seladen	4	Peacock	Margaret	
10267	FRANK	Frankenversand	4	Peacock	Margaret	
10269	WHITC	White Clover Markets	5	Buchanan	Steven	

Combinar JOIN y consultas SQL

Como podemos apreciar, el uso de **JOIN** nos facilita mucho el poder acceder a datos más concisos, algo que las tablas en general, de forma aislada no suelen tener mucho sentido de consistencia.

Además, podemos apreciar cómo sumar el uso de campos calculados, concatenados, e integrar también fácilmente cualquier función escalar útil para nuestros propósitos.



Sección práctica

Es momento de poner en práctica el uso de tablas combinadas mediante JOIN.

Veamos a continuación tres ejercicios para resolver la visualización de datos provenientes de diferentes tablas.



Prácticas

Necesitamos simplificar la visualización de datos de la tabla **Products**, **Customers**, **Categories** y **Employees**, a través de diferentes consultas de selección:

Realiza para ello, las siguientes consignas:

1. Ejecuta una consulta de selección para obtener los campos **ProductID**, **ProductName**, **Quantity** y **UnitPrice**, combinando la tabla **Products** y la tabla **OrderDetails**.
 - a. Deberás visualizar los datos de la orden número: 10255

2. Ejecuta una consulta de selección para visualizar el campo **CustomerName**, de la tabla **Customers**, y los campos **FirstName** y **LastName** de la tabla **Employees**.
 - a. Concatena **FirstName** y **LastName** como un único campo llamado **EjecutivoDeCuentas**

1. Ejecuta una consulta de selección para visualizar los datos **ProductID**, **ProductName** de la tabla **Products** y los campos **CompanyName** y **ContactName** de la tabla **Suppliers**.
 - b. Visualizar la información solo de los productos correspondientes a la categoría 7



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



Argentina programa 4.0



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*

Clase 24: Bases de datos Relacionales

Funciones de agregación

Agenda de hoy

- A. Funciones de Agregación
- B. Qué es una función de agregación
 - a. count
 - b. sum
 - c. min - max
 - d. avg
- C. Consultas y Funciones de agregación simples y compuestas
- D. Agrupamiento
 - a. group by
 - b. having



Funciones de agregación

Algunos encuentros atrás nos sumergimos en el mundo de las **funciones escalares SQL**, las cuales nos permiten transformar específicamente datos de una bb.dd, previo a guardarlos o previo a mostrarlos como resultados.

Esta información nos ayuda a simplificar el tratamiento de la información, para mejorar o clarificar la misma dentro de procesos informáticos que involucren datos.



Funciones de agregación

Además de las funciones escalares, SQL posee las denominadas funciones de agregación, las cuales están preparadas para aportar un valor agregado significativo sobre la información que manejamos.

Más aún en aquellos casos donde debemos trabajar con un resumen de dichos datos, y no con volúmenes en crudo de estos.



Funciones de agregación

Por lo tanto, las **funciones de agregación SQL** se utilizan para resumir con un cálculo, diversas operaciones matemáticas básicas. Dentro de las operaciones más comunes, podemos encontrar:

- contabilizar registros de una tabla
- calcular mínimos, máximos y promedios
- representar operaciones aritméticas sobre dos o más datos numéricos



Entre otras tantas funcionalidades más.

La importancia de consultar datos combinando tablas

El objetivo de nuestra clase de hoy, será concentrarnos en el aprendizaje de estas funciones, que son muy pocas, y luego seguir con la aplicación de las mismas sobre tablas simples, y luego sobre tablas combinadas.

Allí entrará en juego el uso de JOIN en conjunto con estas funciones, donde veremos el máximo potencial de las consultas de selección SQL.



COUNT

Count

La función **count()** se ocupa de contabilizar el total de registros de una tabla, ya sea de forma directa (*sobre el total de registros*), como también aplicando filtros específicos mediante la cláusula **WHERE**.



Count

Se suele utilizar directamente el carácter asterisco (*) o nombre de un campo, dentro de los paréntesis de la función.

Esto es posible dado que count() siempre devolverá un mismo dato del tipo totalizar.



Count

Aquí tenemos un ejemplo de aplicación de la función **count()**. Debemos tener presente siempre que, tanto esta función como también el resto de las funciones de agregación, se ocupan de totalizar y retornar datos simples, o agrupados.



Agregación

```
SELECT count(*)  
FROM Northwind.Employees;
```

Count

En esta consulta, contabilizamos el total de registros de la tabla **Employees**. En este caso vemos que son 9 en total los empleados de **Northwind**.



The screenshot shows a MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is a query editor window containing the following SQL code:

```
1 • SELECT count(*)  
2 FROM Northwind.Employees;
```

Below the code, there are some status indicators: "100%" and "26.2". Underneath the editor is a results grid labeled "Result Grid". The grid has one column labeled "count(*)" and one row containing the value "9". There are also buttons for "Filter Rows", "Search", and "Export".



Count



Agregación

```
SELECT count(*)  
FROM Northwind.Employees  
WHERE Employees.Title = 'Sales Representative';
```

```
1 • SELECT count(*)  
2 FROM Northwind.Employees  
3 WHERE Title = 'Sales Representative';
```

100% 16:1

Result Grid Filter Rows: Search Export:

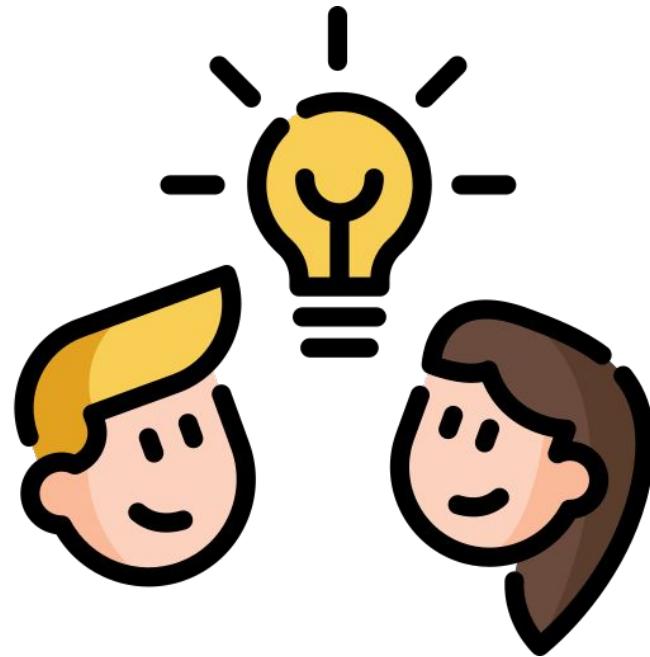
count(*)

▶ 6

De igual forma, podemos acceder a contabilizar los empleados de Northwind, que tengan un cargo específico dentro de la empresa. Por ejemplo: '*Representante de Ventas*'.

Uso de Alias

Al igual de lo que sucede cuando utilizamos funciones escalares, las funciones de agregación requieren también que, junto al procesamiento de datos que estas hagan, podamos definir un alias para el campo o columna resultante ya que, si no lo hacemos, veremos a la función de agregación como nombre de dicha columna.



Uso de Alias

Aquí vemos un ejemplo de aplicación de Alias para cambiar el nombre de la columna resultante.



Agregación

```
SELECT count(*) AS TotalEjecutivosDeVentas  
FROM Northwind.Employees  
WHERE Employees.Title = 'Sales Representative';
```

SUM

SUM

Realiza una sumatoria de números. Se debe especificar, por supuesto, un campo numérico para que pueda realizar el cálculo correcto.



SUM()

```
SELECT SUM(Price) FROM tabla;
```

SUM

En este otro ejemplo sumamos el total de dinero invertido en fletes sobre todas las órdenes de compra, a través del campo **Freight** de la tabla **Orders**.



Agregación

```
SELECT SUM(Freight) As TotalFletes  
FROM Northwind.Orders;
```

SUM

De igual forma, podemos aplicar una consulta similar, sumando el costo de fletes de una o más órdenes específicas incluyendo la cláusula WHERE.

Agregación

```
SELECT SUM(Freight) As TotalFletes  
FROM Northwind.Orders  
WHERE OrderID IN(10256, 10258, 10260);
```

The screenshot shows a MySQL query editor interface. At the top, there's a toolbar with various icons. Below it, the SQL query is written:

```
1 SELECT SUM(Freight) As TotalFletes  
2 FROM Northwind.Orders  
3 WHERE OrderID IN(10256, 10258, 10260);
```

Below the query, there's a progress bar indicating the query is running. At the bottom, there's a results grid labeled "Result Grid". The grid has one column titled "TotalFletes" and one row containing the value "209.57". There are also buttons for "Filter Rows", "Search", and "Export".



MIN y MAX

min y max

MIN() obtiene el número más bajo de un conjunto numérico.

MAX() obtiene el número más alto de un conjunto numérico.



MIN() y MAX()

```
SELECT MIN(edad) FROM tabla;
```

```
SELECT MAX(edad) FROM tabla;
```

min y max

Aquí utilizamos la función **min()** y **max()** para calcular los productos con el precio mínimo y con el precio máximo en la tabla homónima. Además, podemos apreciar cómo integramos dos funciones de agregación en una única consulta.



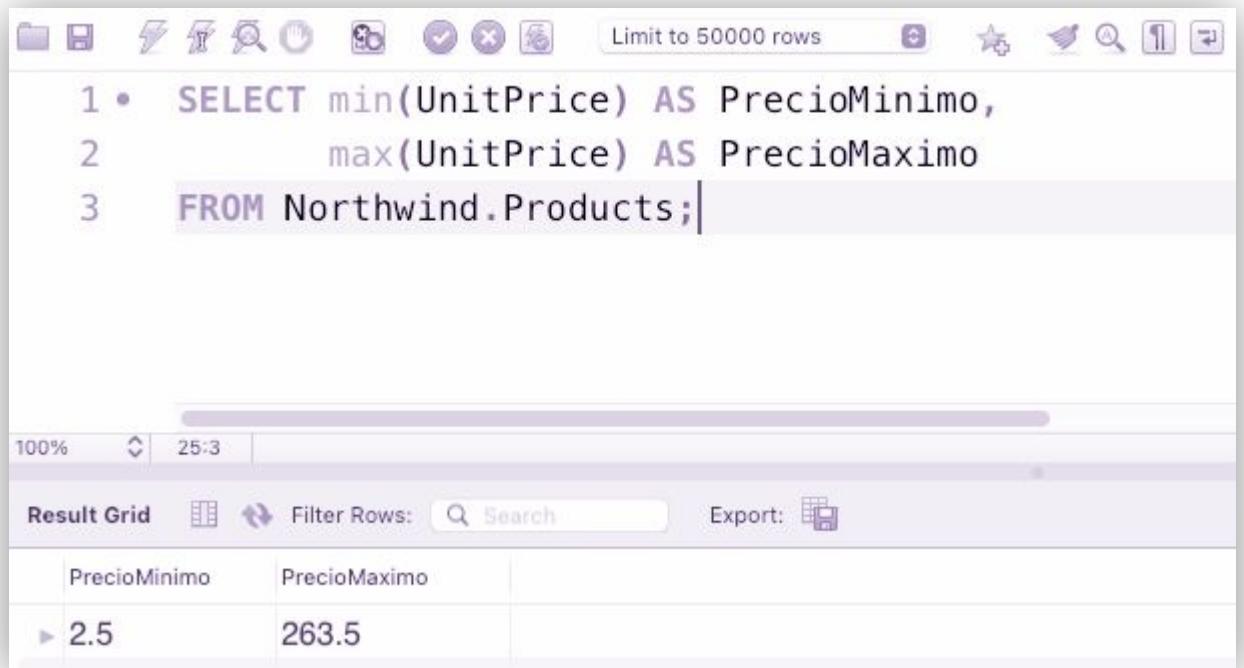
Agregación

```
SELECT min(UnitPrice) AS PrecioMinimo,  
       max(UnitPrice) AS PrecioMaximo  
  FROM Northwind.Products;
```

min y max

Siempre debemos contemplar que cada función de agregación nos retornará una única fila como resultado.

En este caso, en dos columnas por separado.



The screenshot shows a MySQL query editor interface. At the top, there are various icons for file operations like Open, Save, and Print. A toolbar includes a 'Limit to 50000 rows' button. Below the toolbar, the SQL query is displayed:

```
1 •  SELECT min(UnitPrice) AS PrecioMinimo,
2                 max(UnitPrice) AS PrecioMaximo
3  FROM Northwind.Products;
```

Below the query, the results are shown in a grid:

PrecioMinimo	PrecioMaximo
2.5	263.5



AVG

avg

AVG() obtiene el promedio en un campo numérico, en base a los valores almacenados dentro del conjunto de registros.

AVG()

```
SELECT AVG(edad) FROM tabla;
```

avg

Al cálculo de precio mínimo y precio máximo anterior, le agregamos una nueva columna para calcular el precio promedio, **avg()**, entre los casi 80 productos existentes en esta tabla.



Agregación

```
SELECT min(UnitPrice) AS PrecioMinimo,  
       max(UnitPrice) AS PrecioMaximo  
FROM Northwind.Products;
```

avg

Nuevamente conseguimos generar una tercera columna, para visualizar en una misma consulta de selección, el manejo de diferentes valores claves en la tabla **Products**.

The screenshot shows a MySQL query editor interface. At the top, there is a toolbar with various icons. Below it, the SQL query is displayed:

```
1 •  SELECT min(UnitPrice) AS PrecioMinimo,
2                 max(UnitPrice) AS PrecioMaximo,
3                 avg(UnitPrice) AS PrecioPromedio
4  FROM Northwind.Products;
```

Below the query, there is a progress bar indicating the query is running (00% to 25%). Underneath the progress bar, there is a "Result Grid" section with a table showing the results:

PrecioMinimo	PrecioMaximo	PrecioPromedio
2.5	263.5	28.866363636363637



Campos Calculados

Campos calculados

Aprovechando el conjunto de operaciones matemáticas básicas para cualquier lenguaje de programación podemos generar, en una consulta de selección, campos calculados.

Podemos realizar esto aplicando, sobre un campo numérico existente, una operación matemática creando así un nuevo campo a partir de un ALIAS.



Campos calculados

De esta forma podemos sumar, por ejemplo, dos campos numéricos, calcular subtotales, o proyectar incrementos o descuentos sobre algún valor puntual.

```
Calculated fields()

SELECT ProductID,
       Name,
       ListPrice,
       (ListPrice * 0.90) AS TenPercentOff
FROM Products;
```



Campos calculados

De esta forma podemos sumar, por ejemplo, dos campos numéricos, calcular subtotales, o proyectar incrementos o descuentos sobre algún valor puntual.



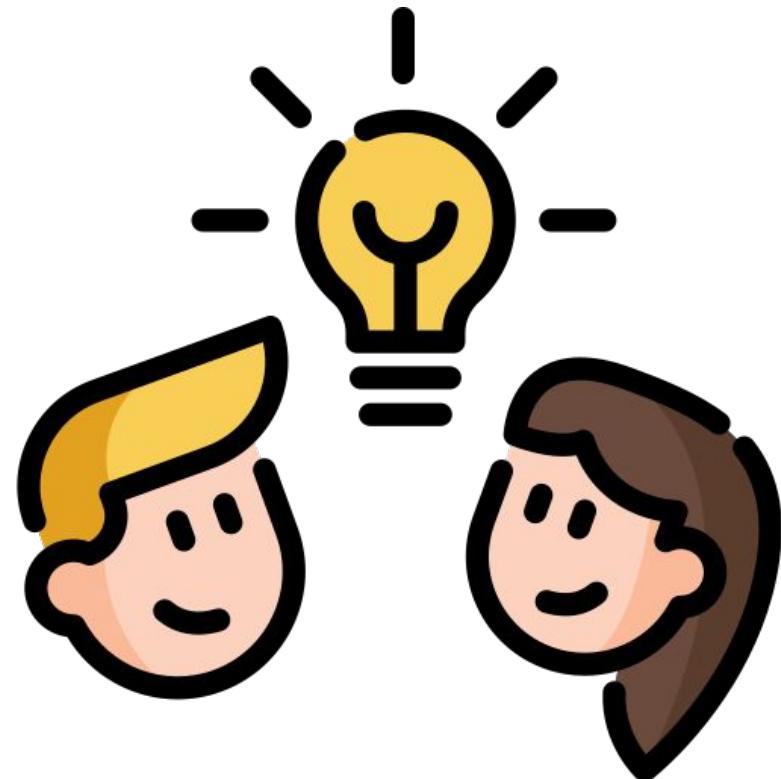
Agregación

```
SELECT ProductID,  
      ProductName,  
      CategoryID,  
     UnitPrice,  
    (UnitPrice * 1.20) AS VeintePorcientoOn,  
    (UnitPrice * 0.90) AS DiezPorcientoOff  
  FROM Products;
```

Campos calculados

Al igual que con los ejemplos anteriores, los campos calculados requieren el uso de Alias.

Si bien podemos obviar su uso, el nombre de la columna quedará poco legible respecto al nombre de las columnas estándar de la tabla.

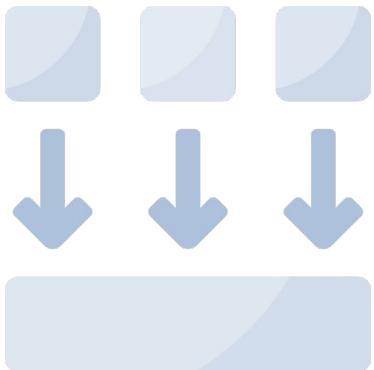


GROUP BY

GROUP BY

Junto al manejo de las funciones de agregación, podemos completar la lógica de uso utilizando mecanismos de agrupación.

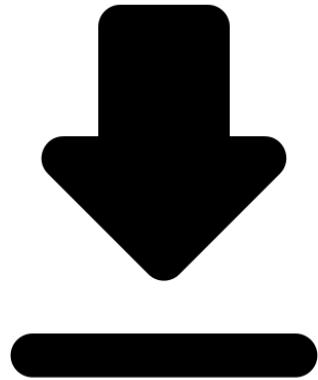
Estos son ideales para cuando tenemos que obtener datos agrupados, provenientes de dos tablas diferentes.



GROUP BY

GROUP BY es la cláusula fundamental para usarse junto a las funciones de agregación.

Nos facilita el trabajo cuando debemos obtener información que nace de la agrupación de registros.



GROUP BY

Aquí tenemos un ejemplo de uso de **GROUP BY**, calculando el precio promedio de nuestros productos agrupando los mismos por proveedor.

Además del precio promedio, la información del proveedor a quien le compramos los mismos es clave, por lo tanto, nos convendrá utilizar **JOIN** para obtener el nombre del proveedor en lugar del Código identificador del mismo.

The screenshot shows a MySQL query editor with the following code:

```
1 •  SELECT SupplierID,
2          AVG(UnitPrice) AS PrecioPromedio
3      FROM Products
4     GROUP BY SupplierID;
```

The results grid displays the following data:

SupplierID	PrecioPromedio
1	15.666666666666666
2	20.35
3	31.66666666666668
4	46
5	29.5
6	14.91666666666666
7	35.57
8	28.175
9	15
10	4.5
11	29.70999999999997
12	44.67800000000004
13	25.89
14	26.43333333333334
15	20
16	15.33333333333334

GROUP BY

Aquí vemos la función de agregación AVG combinando las tablas **Products** y **Suppliers**. En esta modificación sumamos el nombre de la empresa proveedora, y utilizamos el mismo para agrupar el resultado de la consulta.

De esta forma, la consulta de selección resultante nos aportará información más lógica.

```
... Agregación  
  
SELECT S.CompanyName,  
       AVG(P.UnitPrice) AS PrecioPromedio  
FROM Products P  
LEFT JOIN Suppliers S  
ON P.SupplierID = S.SupplierID  
GROUP BY CompanyName;
```

GROUP BY

¡Ahora sí! La información de promedio de precio de productos, agrupada por proveedor, es un poco más clara que antes.

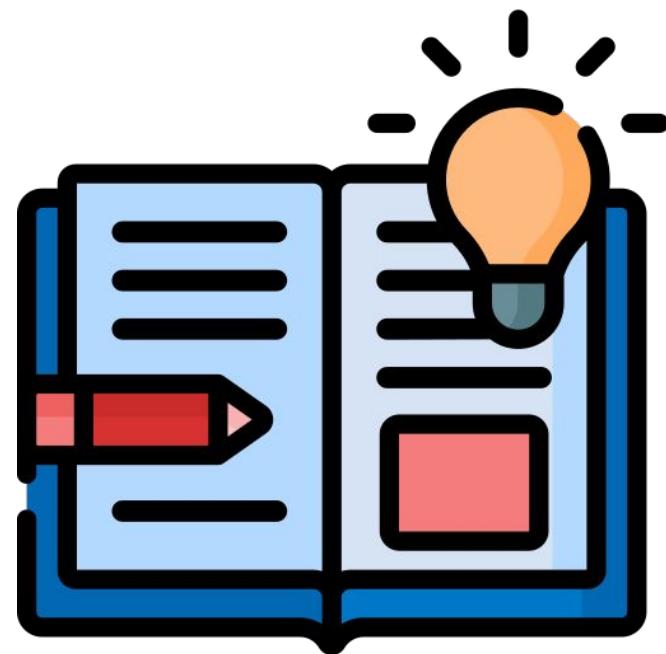
Solo nos falta sumar una función escalar, la cual se ocupe de redondear los decimales de cada precio promedio obtenido, para que la consulta resultante termine de ser clara en un informe.

CompanyName	PrecioPromedio
Exotic Liquids	15.666666666666666
New Orleans Cajun Delights	20.35
Grandma Kelly's Homestead	31.66666666666668
Tokyo Traders	46
Cooperativa de Quesos 'Las Cabras'	29.5
Mayumi's	14.91666666666666
Pavlova, Ltd.	35.57
Specialty Biscuits, Ltd.	28.175
PB Knöckebrot AB	15
Refrescos Americanas LTDA	4.5
Heli Superwaren GmbH & Co. KG	29.70999999999997
Plutzer Lebensmittelgrossmärkte AG	44.678000000000004
Nord-Ost-Fisch Handelsgesellschaft...	25.89

HAVING

HAVING

HAVING aporta una capa de filtro de la información, similar a la lógica que encontramos de la mano de las consultas que utilizan **WHERE**, sumando la posibilidad de incluir una función de agregación en su cláusula.



HAVING

Aquí aplicamos una contabilización de productos que le compramos a cada proveedor.

Agrupamos el resultado mediante el nombre del proveedor, y condicionamos el resultado final, utilizando **HAVING** para seleccionar solo aquellos proveedores que tienen más de 2 productos asociados.

• • • Agregación

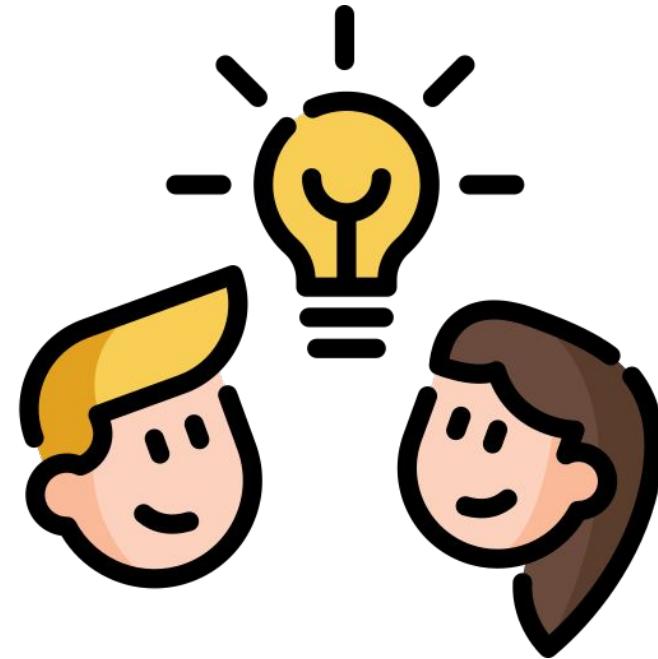
```
SELECT S.CompanyName,  
       COUNT(P.ProductID) AS TotalProducts  
  FROM Suppliers S  
 JOIN Products P  
    ON S.SupplierID = P.SupplierID  
 GROUP BY S.CompanyName  
 HAVING COUNT(P.ProductID) > 2;
```



Combinar JOIN y consultas SQL

HAVING funciona parecido a **WHERE**, estableciendo condiciones para filtrar los resultados. Para lograr esto, necesitamos generar campos con resultados filtrados, para luego recién aplicar **HAVING** dentro de la consulta **SELECT**.

Tengamos presente que esta sentencia, solo funciona con campos generados a partir de una función.



Sección práctica

Debemos comenzar a sacar partida de las funciones de agregación y de los campos calculados.

Desarrollemos para ello, los ejercicios que se plantean a continuación, así comenzamos a adaptar estas fabulosas funciones en nuestro día a día con SQL.



Prácticas

Trabajemos sobre la tabla Products, aplicando algunas funciones de agregación.

Realiza para ello, las siguientes consignas:

1. Ejecuta una consulta de selección para obtener los campos ProductID, UnitPrice
 - a. cuenta el total de Productos con el alias **TotalProductos**
 - b. contabiliza solo aquellos que tengan un precio superior a 30

1. Ejecuta una consulta de selección para visualizar el campo ProductID, y CategoryID
 - b. cuenta los productos de la tabla y muestra el resultado con el alias **TotalProductos**
 - c. agrupa por **CategoryID**

1. Replica la consulta anterior (punto 2), y agrega la siguiente condición:
 - c. muestra solo los resultados de aquellas categorías que tengan más de 7 productos
 - d. TIP: (*deberás utilizar HAVING en este último punto*)



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



Argentina programa 4.0



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*

Clase 25: Bases de datos Relacionales

Data Definition Language

Agenda de hoy

- A. Data Definition Language
- B. Crear tabla con el lenguaje SQL
 - a. CREATE TABLE
 - b. ALTER TABLE
 - c. Add
 - d. Modify
 - e. Drop
- C. Limitaciones al modificar una tabla
- D. Crear múltiples tablas utilizando DDL

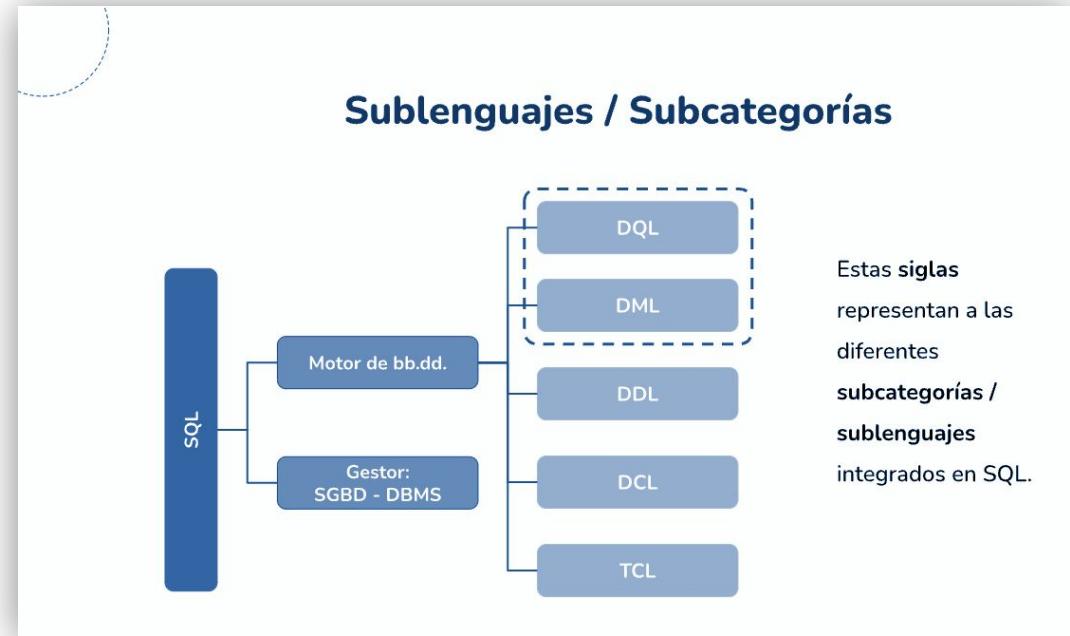


Data Definition Language



Data Definition Language

En nuestros primeros encuentros enfocados a bases de datos SQL, hablamos de los sublenguajes que este posee, que son como categorías que definen cómo hacer determinadas operaciones sobre esta base de datos.



Data Definition Language



Hoy profundizaremos sobre la categoría **DATA DEFINITION LANGUAGE**, para comenzar a sacar partido de todos sus beneficios.

Data Definition Language

DQL

DML

DDL

DCL

TCL

DATA DEFINITION LANGUAGE permite modificar la estructura de objetos de una base de datos SQL.

Está conformado por algunas sentencias que nos permiten crear, modificar, borrar o definir un cambio en la estructura de tablas, vistas y otros objetos que se almacenan en la base de datos.

Data Definition Language

Las sentencias disponibles a través de DDL, son:

- CREATE
- ALTER
- DROP
- TRUNCATE

Con ellas creamos, modificamos, y eliminamos objetos o información extendida contenida en estos.



CREATE

Create

La sentencia **CREATE** cumple la función de crear nuevos objetos en la base de datos.

Los tipos de objetos a crear pueden ser: tablas, índices, stored procedures y hasta nuevas bases de datos, además de usuarios específicos.



Create Database / Schema



Create Database / Schema

A continuación tenemos un ejemplo de cómo crear una base de datos utilizando la cláusula SQL CREATE.

CREATE SCHEMA o **CREATE DATABASE**, seguido del nombre de la base de datos, es la cláusula que necesitamos para crear una base de datos nueva dentro de nuestro motor MySQL. Además, para soportar todos los juegos de caracteres posibles, sumamos la definición del tipo de caracteres **UTF-8**



DDL

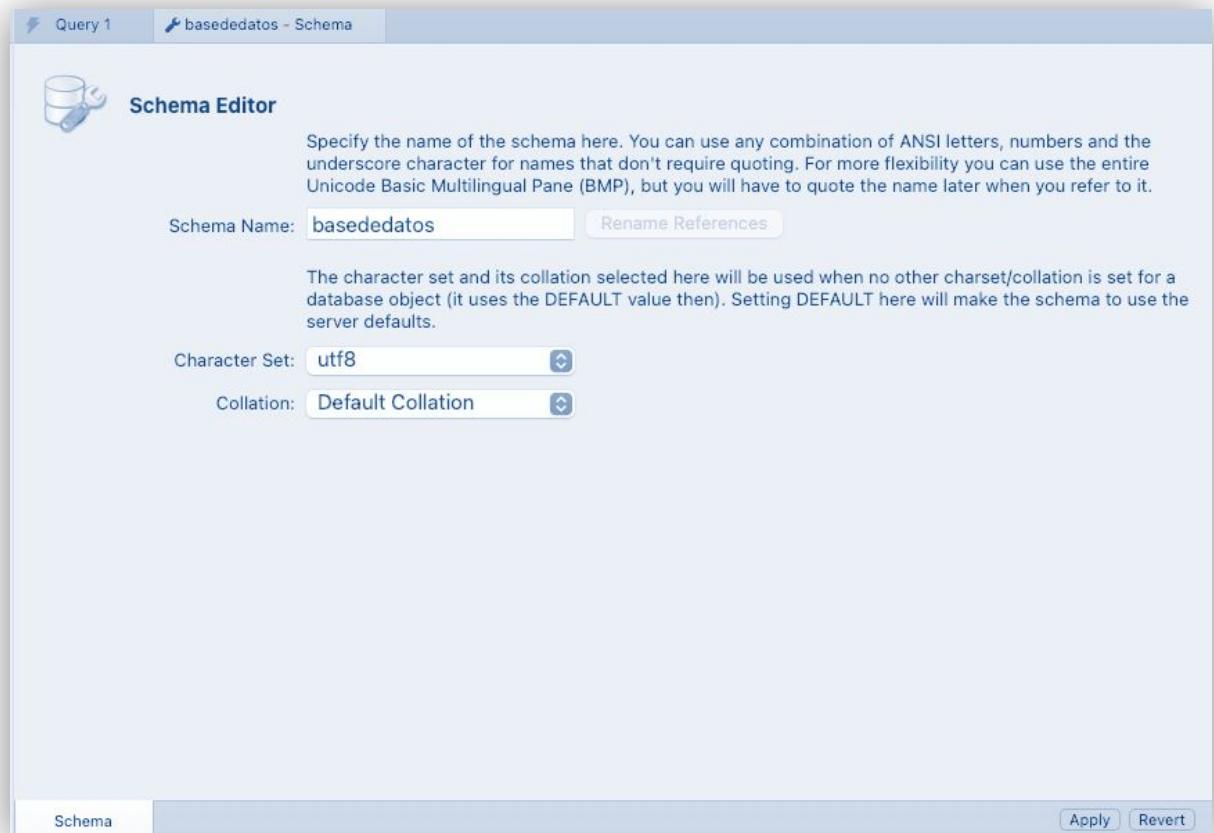
```
CREATE SCHEMA `basededatos`  
DEFAULT CHARACTER SET utf8;
```



Create Database / Schema

Es lo mismo que vimos oportunamente, cuando creamos una base de datos nueva utilizando MySQL Workbench.

Cualquiera de las dos vías, es totalmente válida, aunque en determinados escenarios de trabajo podemos no llegar a contar con un RDBMS y debamos recurrir a la Consola de Administración de MySQL.



Create Table

Create table

Para definir los campos o columnas al momento de crear una tabla, debemos indicar el nombre que queremos para ésta, el tipo de dato que contendrá (*con o sin límite en cantidad de caracteres*), y si acepta o no valores nulos.

De forma similar a cuando vimos la creación de una tabla de forma manual, pero integrando el proceso junto a la cláusula **CREATE**.

Create table

Aquí tenemos un ejemplo de la estructura de código que debemos aplicar utilizando la cláusula CREATE TABLE.

Analizando el mismo, vemos que respeta una estructura bastante similar a la herramienta gráfica de MySQL Workbench para crear tablas, definir campos, tipos, longitud, y parámetros adicionales de configuración.

```
SQL CREATE tabla

CREATE TABLE amigos (
    id INT NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(30) NOT NULL,
    apellido VARCHAR(30) NOT NULL,
    email VARCHAR(50) NOT NULL,
    ([parámetros de la tabla]));
```

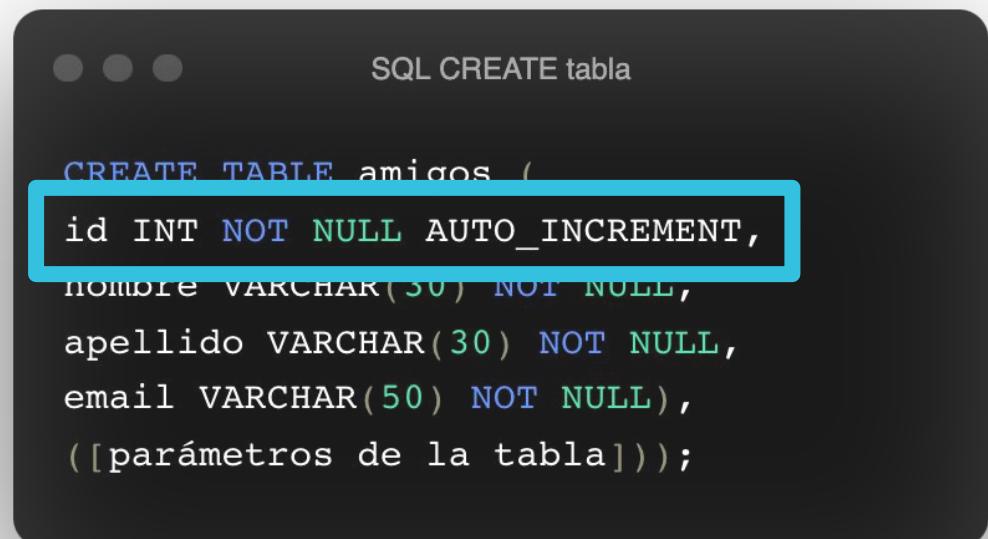


Create table

Tomemos como ejemplo el campo **id**:

Al definirlo, le indicamos una serie de parámetros que demarcan su tipo de dato además del comportamiento que se espera.

Por ejemplo, la información que contendrá es numérica, definida por el tipo de dato entero (**INT**), no acepta valores nulos o vacíos **NOT NULL**, y su valor será automática y autoincremental **AUTO_INCREMENT**.



The screenshot shows a mobile application interface with a dark background. At the top right, there are three small circular icons. To the right of them, the text "SQL CREATE tabla" is displayed. Below this, the SQL code for creating a table is shown in white text:

```
CREATE TABLE amigos (
    id INT NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    apellido VARCHAR(30) NOT NULL,
    email VARCHAR(50) NOT NULL,
    ([parámetros de la tabla]));

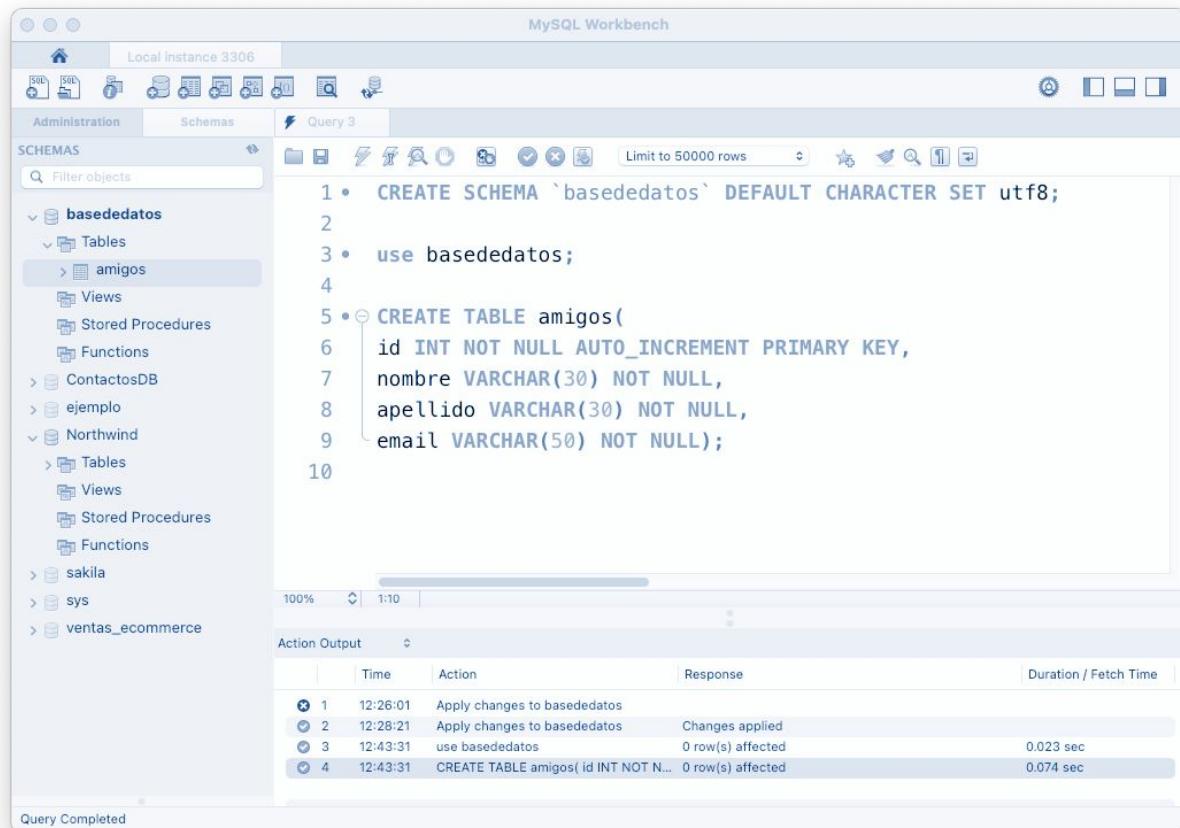
```

The first line of the code, which starts with "id INT NOT NULL AUTO_INCREMENT," is highlighted with a blue rectangular box.



Create table

Veamos un ejemplo de crear una **base de datos**, posicionarnos en ella, y crear a continuación una tabla denominada **amigos**.



The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' tree view displays several databases: 'basededatos' (selected), 'ContactosDB', 'ejemplo', 'Northwind', 'sakila', 'sys', and 'ventas_ecommerce'. Under 'basededatos', there are 'Tables' (with 'amigos' selected) and other objects like 'Views', 'Stored Procedures', and 'Functions'. The main pane shows a query editor with the following SQL code:

```
1 • CREATE SCHEMA `basededatos` DEFAULT CHARACTER SET utf8;
2
3 • use basededatos;
4
5 • CREATE TABLE amigos(
6     id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
7     nombre VARCHAR(30) NOT NULL,
8     apellido VARCHAR(30) NOT NULL,
9     email VARCHAR(50) NOT NULL);
10
```

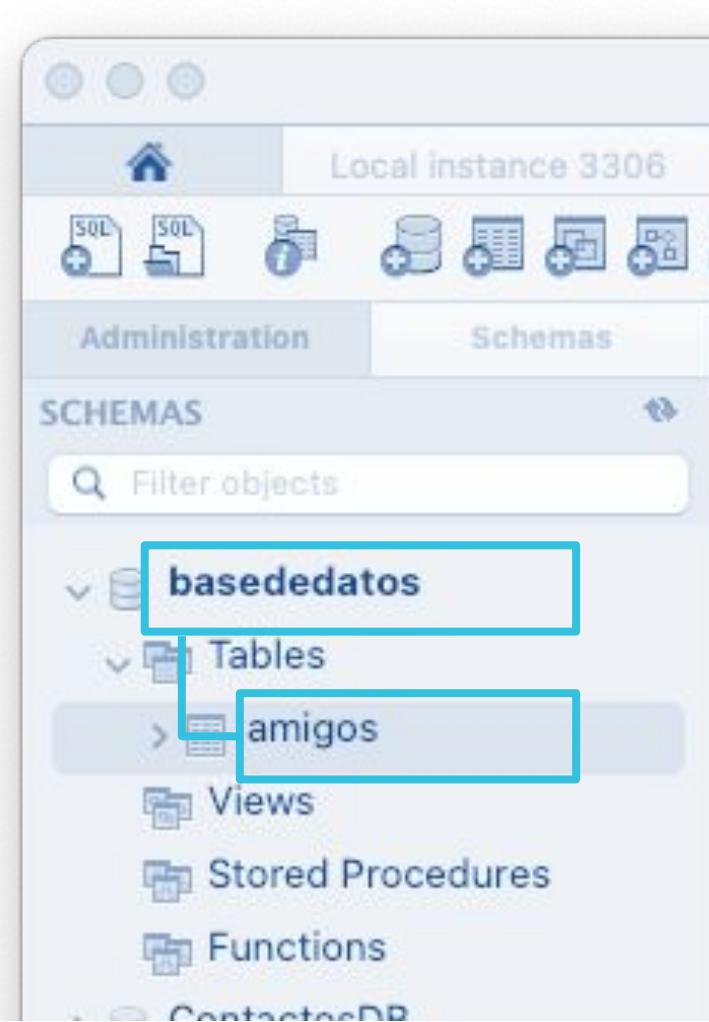
Below the code, the 'Action Output' table shows the results of the operations:

Action	Time	Response	Duration / Fetch Time
1	12:26:01	Apply changes to basededatos	
2	12:28:21	Apply changes to basededatos	Changes applied
3	12:43:31	use basededatos	0 row(s) affected
4	12:43:31	CREATE TABLE amigos(id INT NOT N...	0 row(s) affected

The status bar at the bottom indicates 'Query Completed'.

Create table

Si todo fue bien, debemos ver en el **panel Schemas** de MySQL Workbench, la base de datos (*homónima*) creada, y dentro del apartado **Tables**, la tabla **amigos**, también creada.



ALTER

Alter table

La sentencia **ALTER** , del inglés “*alterar*”, permite modificar la estructura de una tabla u objeto de base de datos.

Con ella podemos *agregar o quitar* campos, *modificar* el tipo de datos de un campo y *agregar o quitar* índices.



Alter table

Si debemos agregar un campo a una tabla, combinamos el uso de **ALTER TABLE** seguido de la cláusula **ADD**.



SQL ALTER

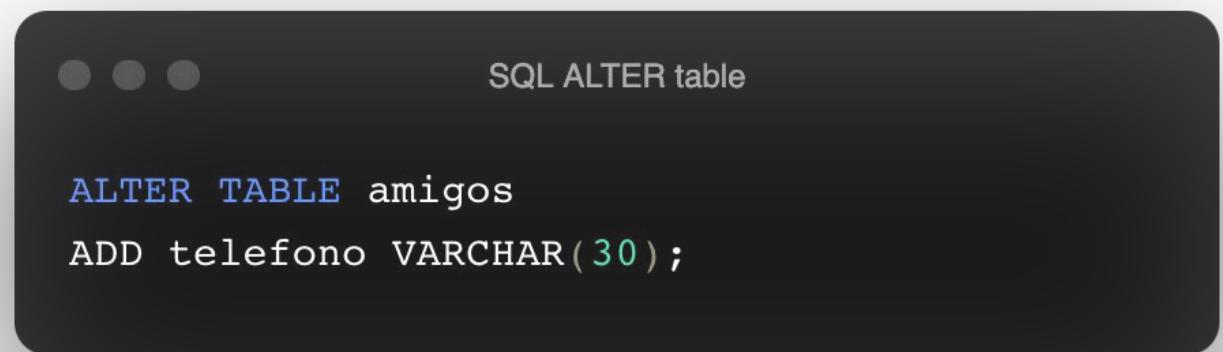
```
ALTER TABLE [nombre de la tabla]  
ADD [definiciones de columnas];
```

Alter table

[nombre de la tabla]: definimos el nombre distintivo de la tabla a modificar.

ADD: es la acción que realizaremos sobre la tabla; en este caso, agregar un **nuevo campo**.

[definición de columna]: Definimos la o las nuevas columnas, tal como hicimos con la sentencia **CREATE**.



The screenshot shows a dark-themed terminal window with three dots at the top left. The text "SQL ALTER table" is displayed in white. Below it, the SQL command "ALTER TABLE amigos ADD telefono VARCHAR(30);" is shown in blue and white text. The background of the terminal is dark gray.

```
SQL ALTER table
ALTER TABLE amigos
ADD telefono VARCHAR(30);
```

Alter table

En el caso de tener que agregar más de un campo en la misma operación, agregamos uno del otro separado por una coma. Cada uno debe llevar su definición correcta.



The image shows a dark-themed terminal window with three dots at the top left. The title bar reads "SQL ALTER table". The main area contains the following SQL code:

```
ALTER TABLE amigos
ADD telefono VARCHAR(30),
ADD comentarios VARCHAR(200);
```



modify

Alter table

Si deseamos modificar los valores para un campo existente, reemplazamos la instrucción **ADD** por **MODIFY**.

Seguido a ello, definimos el campo y el cambio que deseamos aplicar en su estructura.



Alter table

En el siguiente ejemplo vemos una mínima modificación en la longitud de el campo **telefono**, agregado anteriormente sobre la tabla **amigos**.

SQL ALTER modify

```
ALTER TABLE amigos
MODIFY telefono VARCHAR(50) NOT NULL;
```



Alter table

También, podemos hacer el mismo procedimiento que con la cláusula ADD, agregando una cláusula MODIFY por cada uno de los campos que deseemos modificar en un mismo proceso.



SQL ALTER table

```
ALTER TABLE amigos
MODIFY telefono VARCHAR(50) NOT NULL,
MODIFY comentarios VARCHAR(220);
```

Otras modificaciones

Otras modificaciones

CHANGE COLUMN: podemos cambiar el nombre de una columna previamente definida.

RENAME TO: podemos cambiar el nombre inicial de una tabla, por uno nuevo.

DROP COLUMN: podemos eliminar una columna o campo.



Change column name

Cambiamos el nombre de la columna
comentarios por el nombre **sugerencias**.



CHANGE field

```
ALTER TABLE amigos  
CHANGE comentarios sugerencias VARCHAR(500);
```

Change table name

Modificar el nombre de la tabla **amigos** por el nombre **contactos**.



RENAME table

```
ALTER TABLE amigos  
RENAME contactos;
```

Drop column

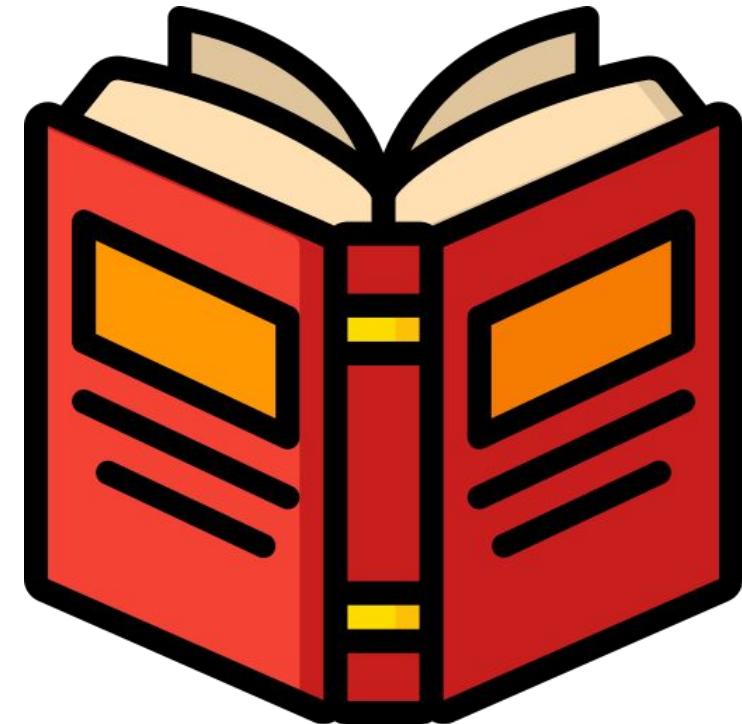
```
● ● ●          RENAME table  
  
ALTER TABLE contactos  
DROP sugerencias;
```

Limitaciones al modificar una tabla

Limitaciones al modificar una tabla

Cuando tuvimos nuestro primer contacto con MySQL Workbench para crear una tabla en una base de datos, verificamos diferentes condiciones que siempre se deben tener en cuenta una vez creada la tabla en cuestión.

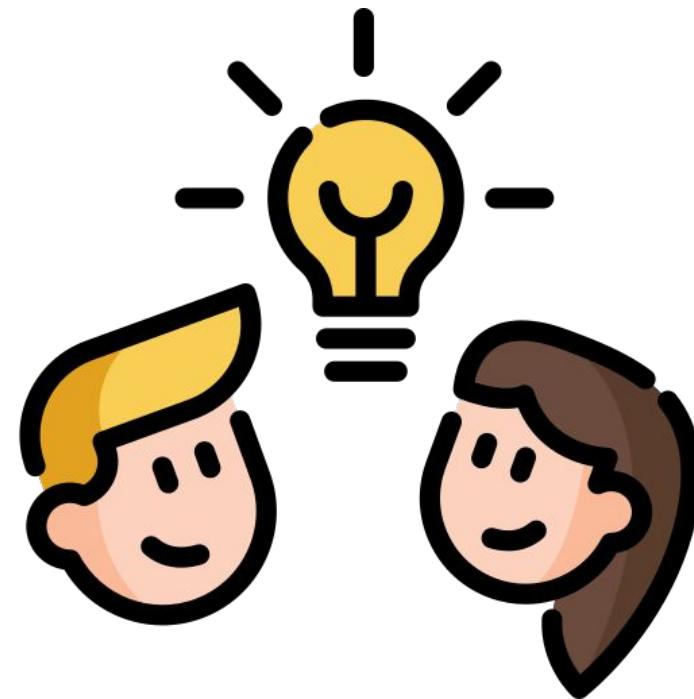
Veamos a continuación un refresh de dichas condiciones dado que estamos llevando a cabo el mismo procedimiento, pero esta vez mediante código SQL.



Limitaciones al modificar una tabla

Existen restricciones al momento de renombrar una tabla, de igual forma que con la necesidad de cambiar el tipo de datos de un campo.

Tengamos presente que, estas cosas sí se pueden realizar en MySQL Workbench y mientras estamos aprendiendo pero, **sobre bases de datos en producción, estas actividades No Se Deben Realizar, o se realizarán luego de un exhaustivo análisis sobre los posibles problemas que puedan surgir.**

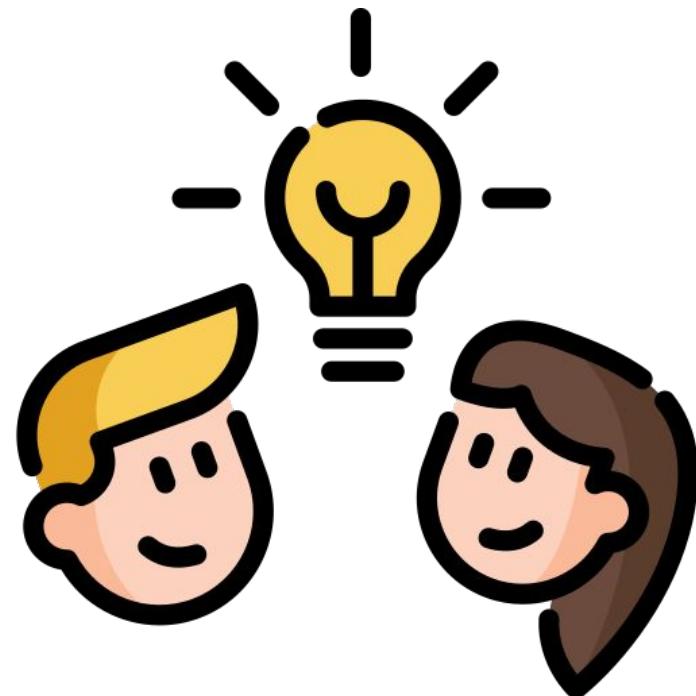


Limitaciones al modificar una tabla

Los roles de **Arquitectos de software, DBA's, Data Engineer, o Líderes técnicos**, son generalmente quienes delinean la estructura principal de una base de datos.

Este proceso tiene un exhaustivo análisis y cuando llega el momento de crear tablas y campos, es porque dicho análisis será el definitivo.

Muy rara vez puede darse la situación de que se creen campos y luego se busquen eliminar o modificar su estructura original.



**Crear múltiples
tablas mediante
DDL**

Crear múltiples tablas mediante DDL

Veamos a continuación un ejemplo de cómo podemos pensar la creación de múltiples tablas SQL a partir de la cláusula **CREATE TABLE**, teniendo en cuenta que estas tablas deben estar relacionadas entre sí.

Dicho ejemplo estará basado en un modelo de **Estudiantes, Profesores y Aulas**, relacionando estas tablas entre sí.



Crear múltiples tablas mediante DDL

La tabla estudiantes posee información simple relacionada a los estudiantes de una institución.

Su campo **idAula** será el nexo con la tabla **Aulas**, para saber en qué ubicación les toca cursar.

Estudiantes

idEstudiante
nombreCompleto
idAula
fechaNacimiento
domicilio

Crear múltiples tablas mediante DDL

La tabla **Profesores** tiene información de los profesores en cuestión, pero no posee relación con ninguna de las otras dos tablas que componen este modelo.

Profesores

idProfesor
nombreCompleto
materia
fechaContratacion
telefono



Crear múltiples tablas mediante DDL

La tabla **Aulas** se relaciona con la tabla profesores, teniendo un campo **idProfesor** que oficia de clave foránea.

A su vez, **idAula** será la clave foránea de la tabla **Estudiantes**.

Aulas

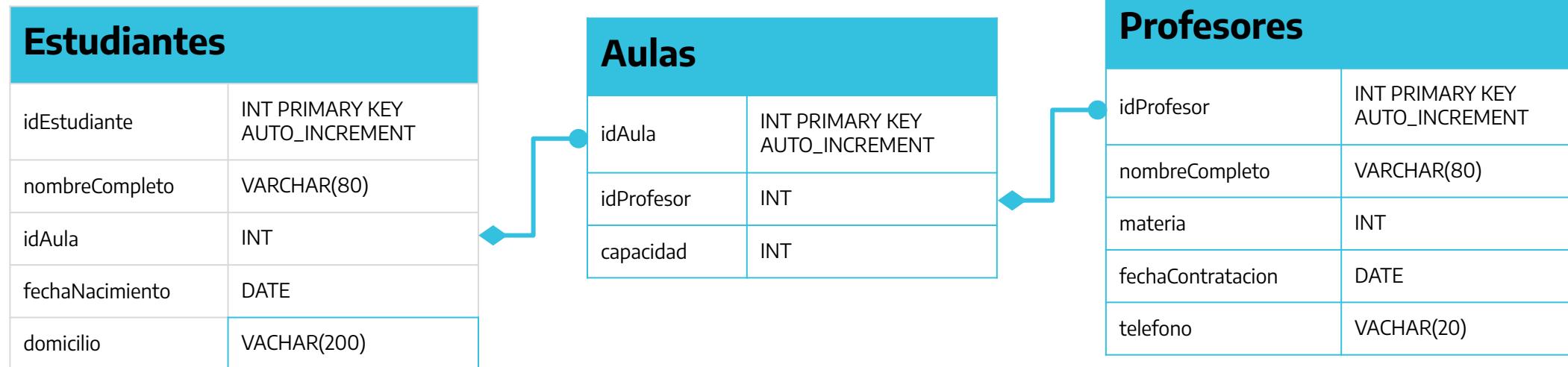
idAula

idProfesor

capacidad

Crear múltiples tablas mediante DDL

Veamos la estructura de datos de estas tres tablas, para realizar luego las cláusulas DDL que permitan crearlas, estableciendo desde DDL el tipo de relación entre ellas.



Crear múltiples tablas mediante DDL

La tabla **Profesores** será la primera que creamos. Al no tener una dependencia foránea en ninguno de sus campos, podremos crearla en primer lugar sin problema alguno.

```
CREATE TABLE Profesores  
    (idProfesor INT PRIMARY KEY AUTO_INCREMENT,  
     nombreCompleto VARCHAR(100),  
     materia VARCHAR(50),  
     fechaContratacion DATE,  
     telefono VARCHAR(20))
```

Profesores
idProfesor
nombreCompleto
materia
fechaContratacion
telefono



Crear múltiples tablas mediante DDL

Luego creamos la tabla **Aulas**. Esta tabla utilizará el campo Id de la tabla Profesores, como clave foránea.

```
CREATE TABLE Aulas (
    idAula INT PRIMARY KEY AUTO_INCREMENT,
    idProfesor INT,
    FOREIGN KEY (idProfesor) REFERENCES
    Profesores(idProfesor),
    capacidad INT
);
```

Aulas

idAula
idProfesor
capacidad



Crear múltiples tablas mediante DDL

Finalmente creamos la tabla **Estudiantes**. Esta tabla utiliza un campo con clave foránea a la columna **id** de la tabla **Aulas**.

```
CREATE TABLE Estudiantes (
    idEstudiante INT PRIMARY KEY AUTO_INCREMENT,
    nombreCompleto VARCHAR(100),
    idAula INT,
    fechaNacimiento DATE,
    direccion VARCHAR(200),
    FOREIGN KEY (idAula) REFERENCES Aulas(idAula));
```

Estudiantes

idEstudiante
nombreCompleto
idAula
fechaNacimiento
domicilio



Crear múltiples tablas mediante DDL

Aquí tenemos el resultado de la creación de las tres tablas en cuestión, más el agregado de algunos registros en su interior.

The screenshot shows a database interface with a sidebar on the left containing a tree view of the schema. The 'basededatos' node is expanded, showing its sub-components: 'Tables', 'Views', 'Stored Procedures', 'Functions', and two additional databases ('ContactosDB' and 'ejemplo'). Under 'Tables', there are four entries: 'amigos', 'Aulas', 'Estudiantes', and 'Profesores'. The 'Estudiantes' entry is highlighted with a blue rectangular border. To the right of the sidebar is a main panel titled 'Result Grid'. At the top of this panel is a toolbar with various icons for filtering, searching, and editing. Below the toolbar is a table with five columns: 'idEstudiante', 'nombreCompleto', 'idAula', 'fechaNacimiento', and 'direccion'. The table contains three data rows, each corresponding to one of the three students listed in the 'Estudiantes' table from the schema browser. The data is as follows:

idEstudiante	nombreCompleto	idAula	fechaNacimiento	direccion
1	Juan Pérez	1	1999-05-10	Calle Principal 123
2	María López	1	2000-02-15	Avenida Central 456
3	Carlos Ramírez	2	1998-09-20	Calle Secundaria 789

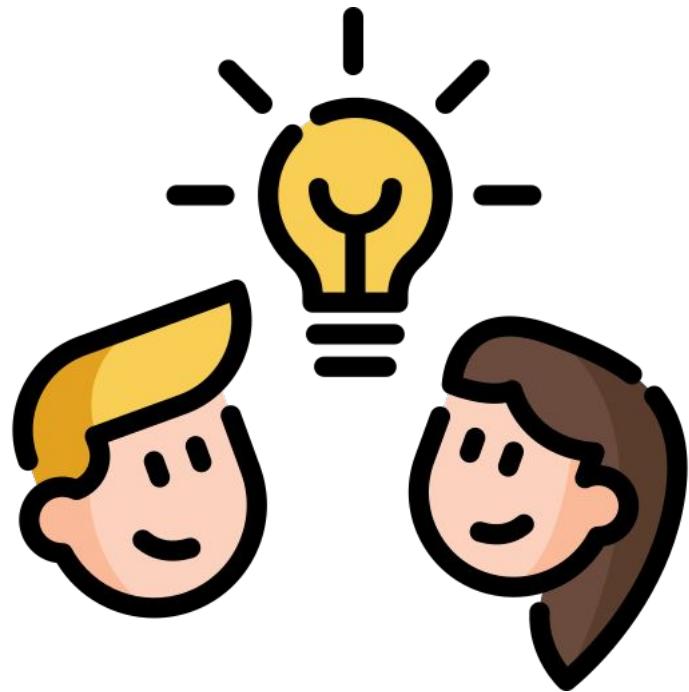


Crear múltiples tablas mediante DDL

Partiendo de este último ejemplo abordado en la creación de múltiples tablas relacionales, utilizando la referencia a campos foráneos.

¿Se animan a adivinar a qué Forma Normal pertenece este modelo de tablas creado?

Hagan memoria sobre nuestra clase titulada “*El modelo Relacional*” donde utilizamos un ejemplo similar al aquí abordado.



Sección práctica

Desarrollaremos a continuación un ejercicio de similares características a este último.

Veamos la consigna para luego comenzar a trabajar en SQL.



Prácticas

A partir del siguiente modelo de Tablas, deberás establecer cuál es la relación entre las mismas, y crear las cláusulas SQL DDL correspondientes para luego insertar datos.

Equipos
idEquipo
nombreEquipo
especialidad
idCliente

Clients
idCliente
nombreEmpresa
rubroEmpresa

Empleados
idEmpleado
nombreEmpleado
puestoEmpresa
idEquipo

Prácticas

Previo al proceso de crear las tablas, define:

1. El tipo de datos más apropiado para cada columna
1. El tipo de relación entre una tabla con otras tablas de este modelo
1. Crea a continuación las cláusulas SQL DDL en el orden que corresponde crearlas
 - a. Define en las cláusulas correspondientes, el uso de FOREIGN KEY
 - b. No olvides agregar PRIMARY KEY y AUTO_INCREMENT en los campos principales



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



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

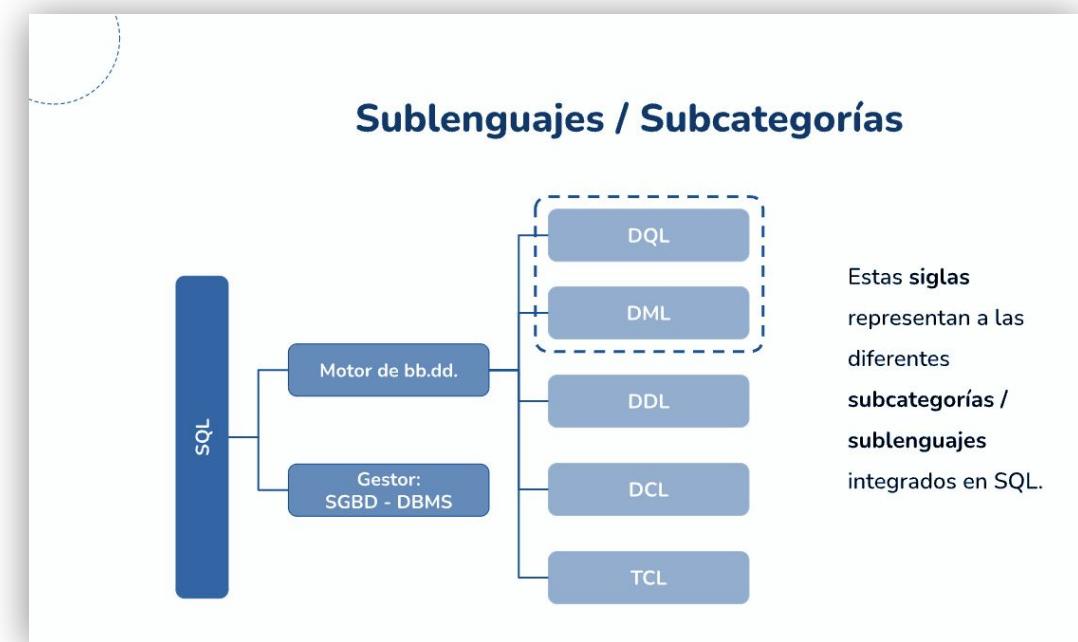
- A. Data Manipulation Language
- B. INSERT
 - a. individual
 - b. masivo
- C. UPDATE
 - a. individual
 - b. masivo
- D. DELETE
 - a. precauciones
- E. TRUNCATE



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

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.

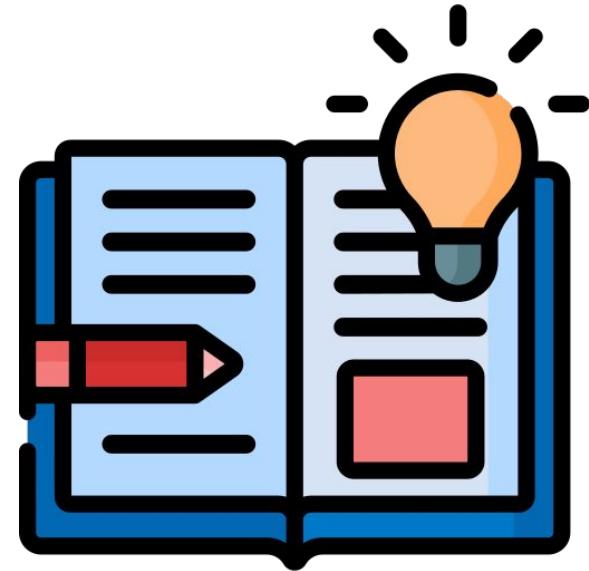


Data Manipulation Language

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 sepáramos cada nuevo registro del anterior, con una coma.

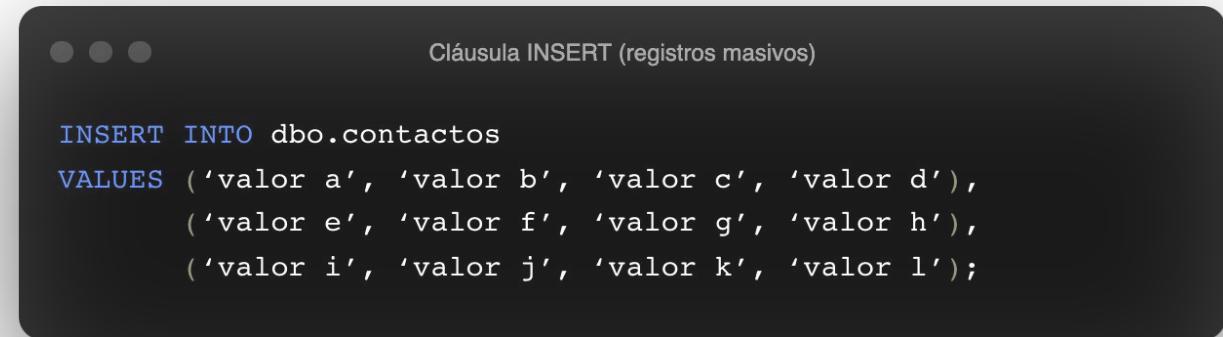


Cláusula INSERT (registros masivos)

```
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.

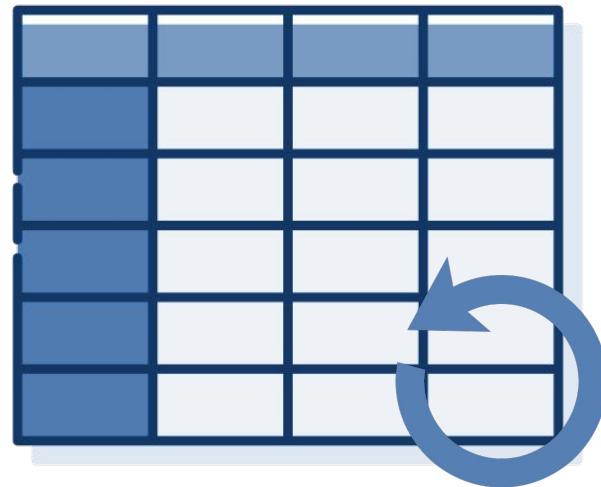
ID	campo2	campo3	campo4	campo5
10	valor 2	valor 3	<i>NULL</i>	valor 5

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;
```

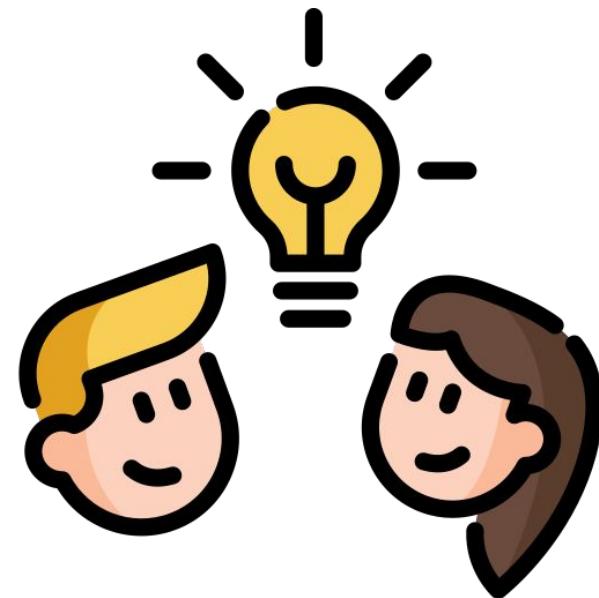


UPDATE masivo (¡Atención!)

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

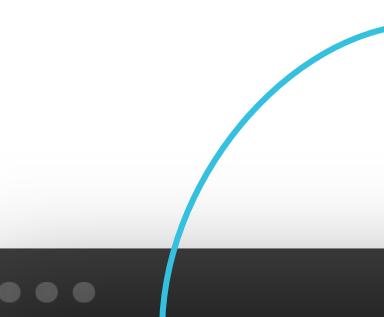


Clonar tablas a partir de una existente

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



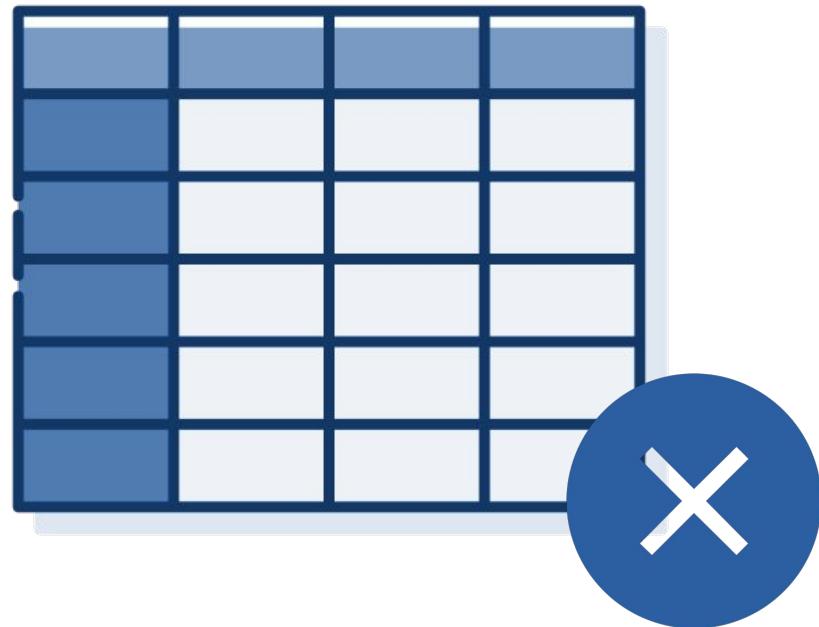
Puesta en común del ejercicio

DELETE

DELETE

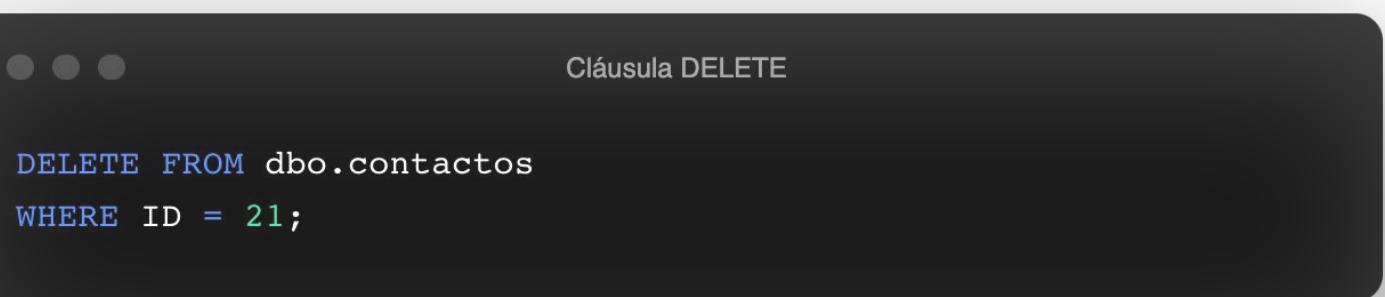
Finalmente, para eliminar registros de una tabla, utilizamos la sentencia **DELETE**.

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.



The screenshot shows a dark-themed terminal window with three dots at the top left. On the right, the text "Cláusula DELETE" is displayed. Below it, a SQL command is shown:

```
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);
```

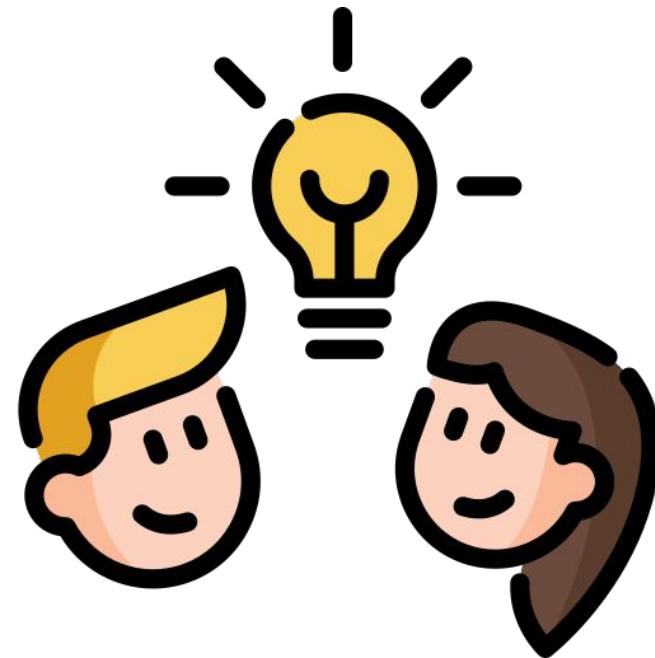
```
DELETE FROM dbo.Personas  
WHERE Edad >= 45;
```



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*



Argentina programa 4.0



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*

Clase 27: Bases de datos Relacionales

Desarrollar un modelo relacional



Agenda de hoy

- A. Migrar un modelo nosql hacia un modelo relacional
- B. Analizar el contenido a migrar
- C. Planificar la bb.dd.
- D. Crear las tablas, índices y relaciones
- E. Definir el Script de inserción de datos
 - a. Consignas para comenzar a trabajar en la pre-entrega 3

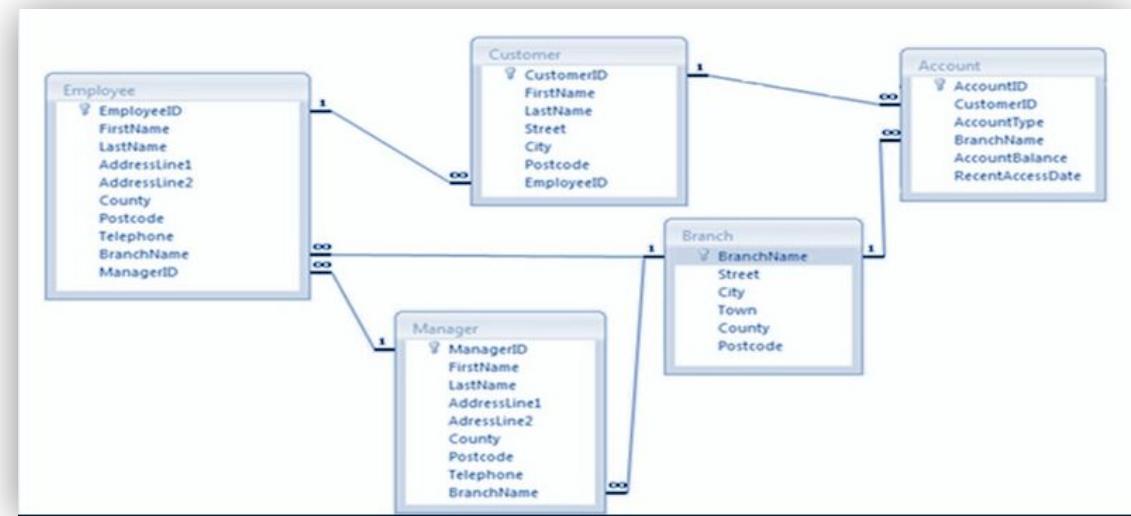


Desarrollar un modelo relacional

Desarrollar un modelo relacional

Como hemos visto hasta aquí, MySQL o SQL en general, se apoyan en el desarrollo y evolución de la información, basándose en un modelo de datos relacionales.

Este modelo es el único aprovechable para estructurar la información de la manera más efectiva posible.



Data Manipulation Language

Cuando aprendimos sobre el ecosistema de bb.dd NoSQL, vimos que estas bb.dd no utilizan un esquema fijo ni requieren que los datos se ajusten a un modelo predefinido, además permiten una mayor flexibilidad y escalabilidad horizontal, lo que las hace ideales para aplicaciones web y móviles de alta escala.



Data Manipulation Language

Pero cuando se trata de datos en cantidades notables donde estos crecerán de forma continua todo el tiempo, y que mucha de esta información se repetirá constantemente, allí el modelo de datos SQL es el modelo de información más asertivo para administrar todo este cúmulo de información de la manera más apropiada.



Data Manipulation Language

Y si nos posicionamos en algún ejemplo de datos específico, donde el volumen de la información seguramente se repita de forma frecuente, allí podremos entender que, para estos tipos de datos repetitivos, el modelo relacional será el que mejor se adapte para minimizar el crecimiento de esta información a futuro.

```
... JSON

{
  "id": 17,
  "poster": "./posters/17.jpg",
  "titulo": "Halt and Catch Fire",
  "categoria": "Serie",
  "genero": "Drama",
  "tags": "Ficción, Drama, Tecnología",
  "busqueda": "Halt and Catch Fire, Ficción, Drama, Tecnología, Lee Pace, Scoot McNairy, Mackenzie Davis, Kerry Bishé, Toby Huss, Alana Cavanaugh",
  "temporadas": "4",
  "reparto": "Lee Pace, Scoot McNairy, Mackenzie Davis, Kerry Bishé, Toby Huss, Alana Cavanaugh",
  "trailer": "https://www.youtube.com/embed/pWrioRji60A"
}
```

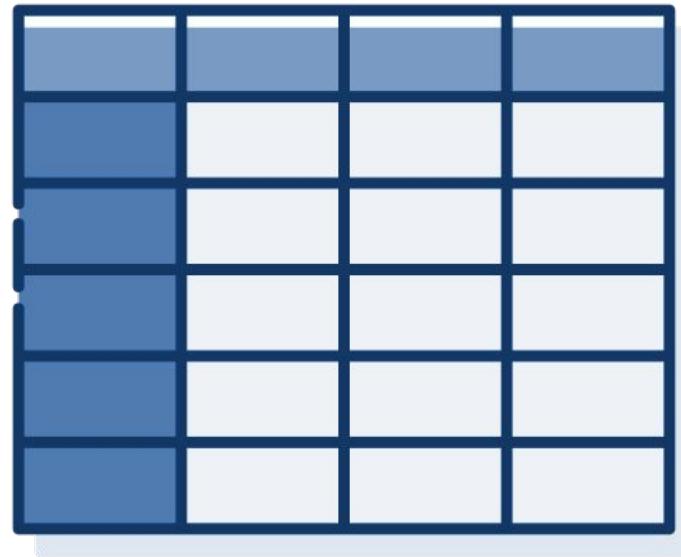


Migrar un modelo nosql hacia un modelo relacional

Migrar un modelo nosql hacia un modelo relacional

Como vimos con el ejemplo anterior, migrar un modelo de datos NoSQL hacia un modelo de datos relacional (SQL) puede ofrecer varias ventajas significativas.

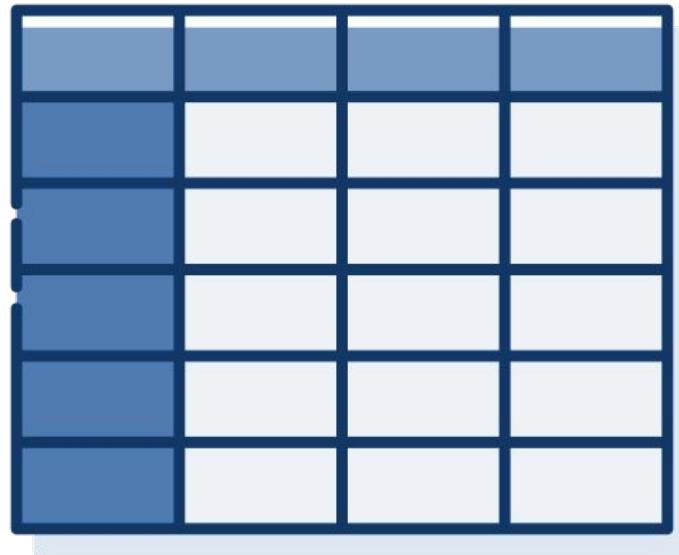
En un modelo relacional, sabemos que los datos se organizarán en tablas estructuradas con relaciones establecidas entre ellas, lo que permite un mejor control y coherencia de los datos.



Migrar un modelo nosql hacia un modelo relacional

Este mejor control y con una coherencia en los datos almacenados, nos facilita poder realizar consultas complejas y análisis más profundos (*métricas*).

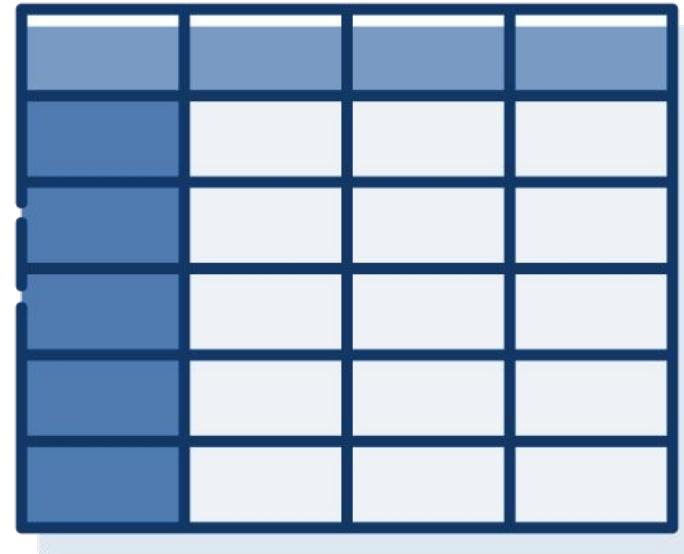
Además, los sistemas de gestión de bases de datos relacionales (RDBMS) ofrecen características como transacciones ACID, que aseguran la integridad y consistencia de los datos.



Migrar un modelo nosql hacia un modelo relacional

Esto es especialmente útil en aplicaciones empresariales donde la confiabilidad y la precisión son fundamentales. Otro beneficio clave es la capacidad de realizar consultas ad hoc y utilizar el lenguaje SQL estándar, lo que facilita el acceso y la manipulación de los datos.

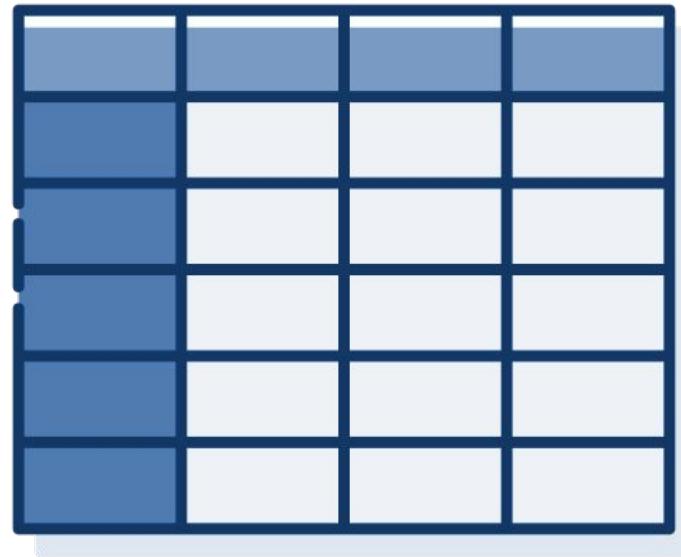
Además, los RDBMS tienen una amplia gama de herramientas y soporte, lo que simplifica el mantenimiento y la escalabilidad del sistema.



Migrar un modelo nosql hacia un modelo relacional

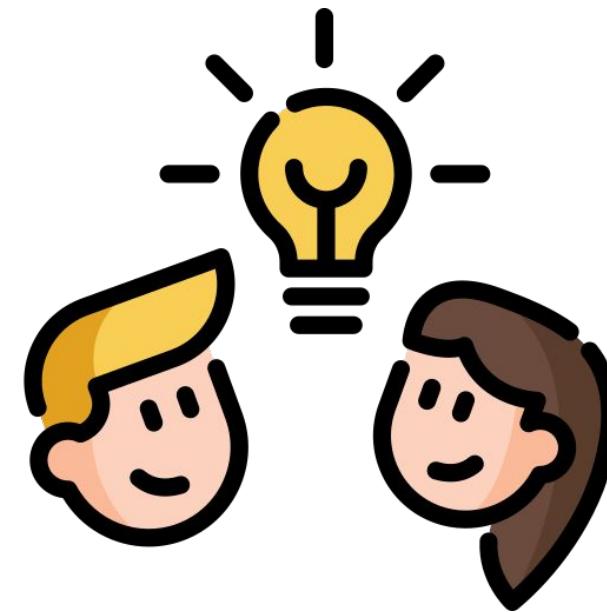
Esto es especialmente útil en aplicaciones empresariales donde la confiabilidad y la precisión son fundamentales. Otro beneficio clave es la capacidad de realizar consultas ad hoc y utilizar el lenguaje SQL estándar, lo que facilita el acceso y la manipulación de los datos.

Además, los RDBMS tienen una amplia gama de herramientas y soporte, lo que simplifica el mantenimiento y la escalabilidad del sistema.



Migrar un modelo nosql hacia un modelo relacional

En resumen, migrar a un modelo de datos relacional ofrece mayor control, consistencia, confiabilidad y facilidad de acceso a los datos, lo que puede ser beneficioso para aplicaciones empresariales y análisis de datos más complejos.



Analizar un contenido a migrar

Analizar un contenido a migrar

Miremos este otro ejemplo de estructura de datos, donde tenemos un modelo de datos basado en un producto electrónico.

¿Podemos dilucidar entre todas cuántos datos podrían desprenderse o convertirse en diferentes tablas SQL?

```
JSON

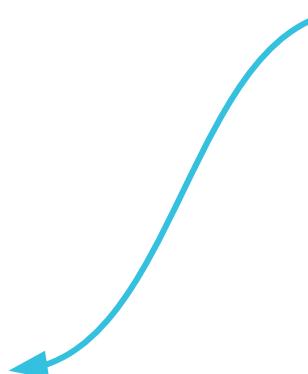
{
  "codigo": 1234,
  "nombre": "Smartphone XYZ",
  "categoria": "Electrónica",
  "precio": 599.99,
  "características": {
    "pantalla": "6.5 pulgadas",
    "resolucionPantalla": "1080 x 2400",
    "procesador": "Snapdragon 855",
    "memoriaRAM": "8 GB",
    "almacenamiento": "128 GB",
    "camaraTrasera": "12 MP",
    "camaraFrontal": "8 MP",
    "bateria": "4000 mAh",
    "sistemaOperativo": "Android 14"
  },
  "stock": 25,
  "fotos": [
    ".productos/imagenes/fotos/1234-1.jpg",
    ".productos/imagenes/fotos/1234-2.jpg",
    ".productos/imagenes/fotos/1234-3.jpg"
  ],
  "videoDemo": "https://www.youtube.com/embed/abcd1234"
}
```



Analizar un contenido a migrar

Respecto a nuestra comparación con el catálogo de películas o series, vemos que aquí tenemos algunos puntos que se repiten, como por ejemplo **Categorías, Nombre, Código.**

El modelo relacional, más allá del segmento donde se vaya a aplicar el mismo, tendrá muchos puntos en común, repetitivos.



```
JSON

{
  "codigo": 1234,
  "nombre": "Smartphone XYZ",
  "categoria": "Electrónica",
  "precio": 599.99,
  "caracteristicas": {
    "pantalla": "6.5 pulgadas",
    "resolucionPantalla": "1080 x 2400",
    "procesador": "Snapdragon 855",
    "memoriaRAM": "8 GB",
    "almacenamiento": "128 GB",
    "camaraTrasera": "12 MP",
    "camaraFrontal": "8 MP",
    "bateria": "4000 mAh",
    "sistemaOperativo": "Android 14"
  },
  "stock": 25,
  "fotos": [
    ".productos/imagenes/fotos/1234-1.jpg",
    ".productos/imagenes/fotos/1234-2.jpg",
    ".productos/imagenes/fotos/1234-3.jpg"
  ],
  "videoDemo": "https://www.youtube.com/embed/abcd1234"
}
```



Analizar un contenido a migrar

Características es un punto a detallar en escenarios de productos de diferentes segmentos, como ser: (*computación, electrodomésticos, línea blanca, IoT, electrónica de entretenimiento, etcétera*).

Dada la variedad de segmentos y de posibles características de cada producto, este es un escenario que debemos pensar bien cómo introducirlo dentro de un modelo relacional.



```
JSON
{
  "codigo": 1234,
  "nombre": "Smartphone XYZ",
  "categoria": "Electrónica",
  "precio": 599.99,
  "caracteristicas": {
    "pantalla": "6.5 pulgadas",
    "resolucionPantalla": "1080 x 2400",
    "procesador": "Snapdragon 855",
    "memoriaRAM": "8 GB",
    "almacenamiento": "128 GB",
    "camaraTrasera": "12 MP",
    "camaraFrontal": "8 MP",
    "bateria": "4000 mAh",
    "sistemaOperativo": "Android 14"
  },
  "stock": 25,
  "fotos": [
    ".productos/imagenes/fotos/1234-1.jpg",
    ".productos/imagenes/fotos/1234-2.jpg",
    ".productos/imagenes/fotos/1234-3.jpg"
  ],
  "videoDemo": "https://www.youtube.com/embed/abcd1234"
}
```



Analizar un contenido a migrar

ProductosCaracteristicas												
IDCARACTERISTICA	IDPRODUCTO	PANTALLA	RESOLUCION	PROCESADOR	MEMORIA	ALMACENAMIENTO	CAMARA	BATERIA	SO	SLOTS	PUERTOSHDMI	
122	1234	6.5 pulgadas	1080x2400	snapdragon 855	8 gb	128 gb	12 mp	4500 mAh	Android 14			

En el caso de definir diferentes características para un producto, de acuerdo al tipo de negocio en cuestión, puede darse la definición de una estructura única que conlleve columnas con los nombres de todas las características posibles, y que solo se completen aquellas que aplican al producto en cuestión.

Este modelo horizontal es un ejemplo posible para llevar adelante, pero luego requerirá un trabajo de lógica de código condicional que solo “represente en pantalla” aquellas características que están completas con algún valor. No es malo, pero es poco útil con el evolucionar del tiempo.



Analizar un contenido a migrar

Características	
IDCARACTERISTICA	DESCRIPCION
1	PANTALLA
2	RESOLUCION
3	PROCESADOR
4	PROCESADOR
5	MEMORIA
6	ALMACENAMIENTO
7	CAMARA
8	BATERIA
9	SO
10	SLOTS
11	PUERTOSHDMI
12	...

El modelo vertical para la definición de una estructura de características, es el más apropiado para estas situaciones.

Aquí se creará una tabla intermedia, que pueda officiar de nexo entre un producto específico, y aquellas características que éste utilice. De esta forma, se crearán una sola vez características genéricas, las cuales podrán aplicarse a N cantidad de productos a lo largo del tiempo.



Analizar un contenido a migrar

Producto	
IDPRODUCTO	DESCRIPCION
1234	SMARTPHONE XYZ
1235	TABLET PAD 9.8
1236	SMARTWATCH 1.7
1237	LIGHTBOOK 14
1238	...

ProductosCaracteristicas			
IDPRODCARAC	IDPRODUCTO	IDCARACTERISTICA	VALOR
1	1234	1	6.5 pulgadas
2	1234	2	1080x2400
3	1234	3	snapdragon 855
4	1234	4	8 gb
5	1234	5	128 gb
6	1234	6	12 mpx
7	1234	7	4500 mAh
8	1234	8	Android 14

Caracteristicas	
IDCARACTERISTICA	DESCRIPCION
1	PANTALLA
2	RESOLUCION
3	PROCESADOR
4	PROCESADOR
5	MEMORIA
6	ALMACENAMIENTO
7	CAMARA
8	BATERIA
9	SO
10	SLOTS
11	PUERTOSHDMI
12	...

Este es un modelo de tres tablas, donde vemos una tabla dedicada a **Productos**, otra tabla dedicada a **Características** en general “sólo el nombre”, y una tercera tabla **ProductosCaracteristicas**, la cual oficia de nexo entre un producto, y sus características específicas. Este modelo es conocido como **Tablas Intermedias**.



Analizar contenido a migrar

Este tipo de análisis es la forma principal que se utiliza para poder diseñar una base de datos.

En la clase denominada “*El modelo Relacional*”, aplicamos la revisión de un trabajo como éste, entendiendo la importancia del modelo relacional y de cómo planificar un diseño efectivo.



Planificar la bb.dd

Planificar la bb.dd

Planificación de una base de datos relacional

Entonces, de acuerdo al repaso realizado hasta aquí, la planificación de una base de datos relacional es esencial para organizar y estructurar la información de manera eficiente.

Nuestro trabajo implica identificar las entidades y relaciones entre ellas, definir los atributos y las restricciones de los datos. Una buena planificación nos ayuda a garantizar la integridad, la consistencia, y la flexibilidad de nuestra base de datos.



Planificar la bb.dd

Pasos clave en la planificación de una base de datos relational

Pasemos en limpio los puntos que debemos cubrir:

1. Identificar las entidades
2. Definir atributos
3. Establecer relaciones
4. Normalizar los datos
5. Diseño del esquema
6. Optimización



Planificar la bb.dd

Herramientas adicionales

Y, para el diseño de la base de datos, podemos apoyarnos en herramientas de diseño externas.

Si bien podemos realizar esta tarea directamente sobre el motor de bb.dd, el diseño de una base de datos relacional requiere de herramientas adecuadas que nos permitan visualizar y organizar la estructura de manera efectiva.



Planificar la bb.dd

Herramientas adicionales

Existen diversas opciones disponibles para simplificar este proceso, brindándonos la capacidad de crear modelos de tablas y establecer relaciones entre ellas de forma intuitiva.

Y dado los tiempos actuales, muchas de estas herramientas pueden ejecutarse directamente en un navegador web y almacenar nuestros trabajos en una plataforma Cloud como OneDrive o Google Drive.



Planificar la bb.dd

Veamos a continuación, algunos ejemplos de herramientas para el diseño de bases de datos relacionales:

1. DB Designer
2. Draw.io
3. Lucidchart

Todas ellas son accesibles por internet, sin que debamos instalar nada adicional en nuestra computadora.

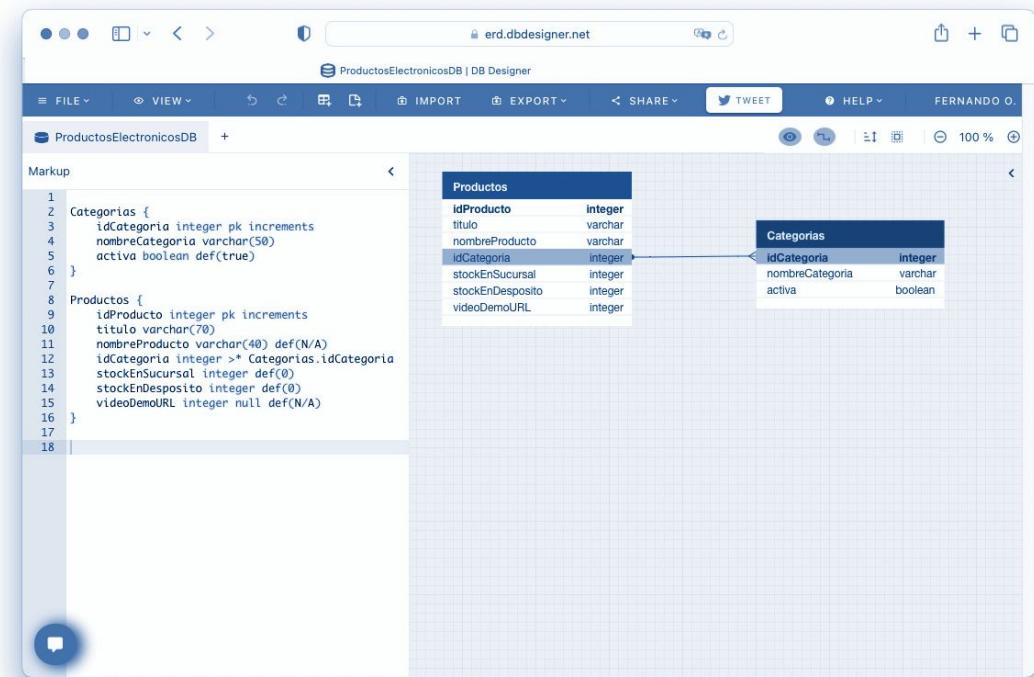


Planificar la bb.dd

DB Designer

DB Designer es una herramienta que nos permite crear modelos de tablas relacionales de manera visual y sencilla. Ofrece una interfaz intuitiva y opciones para definir tablas, atributos y relaciones.

<https://www.dbdesigner.net/>

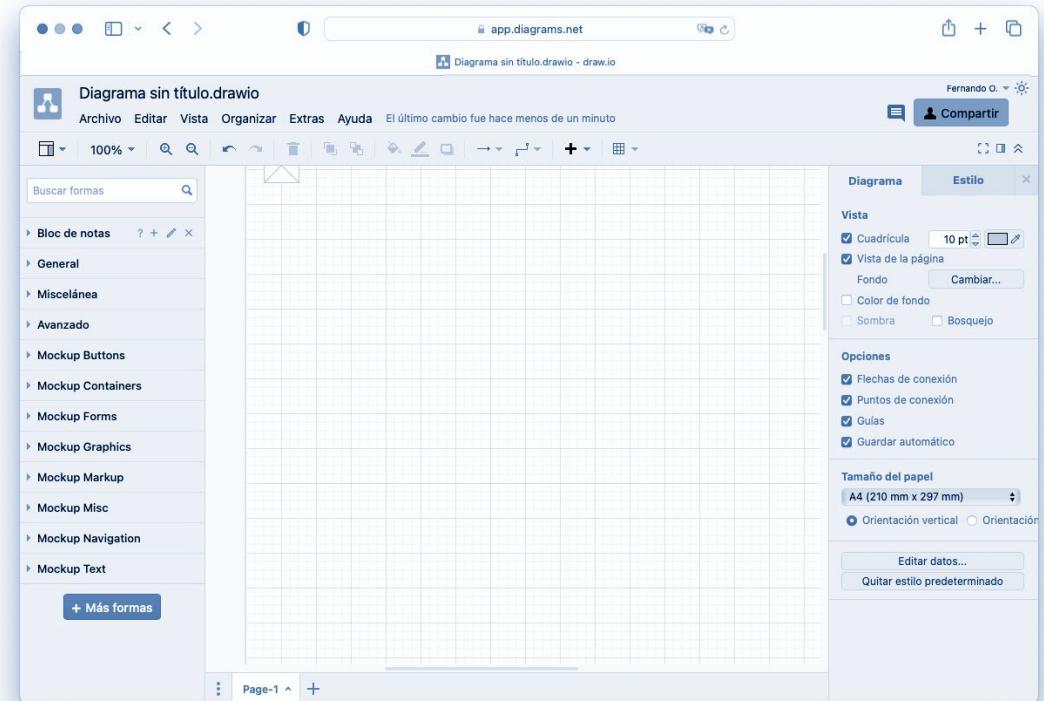


Planificar la bb.dd

Lucidchart

Lucidchart es una herramienta completa, que permite crear diagramas y flujos de trabajo, además de herramientas para el diseño de bases de datos relacionales. Permite crear modelos de tablas y definir relaciones de manera eficiente.

<https://www.lucidchart.com/>

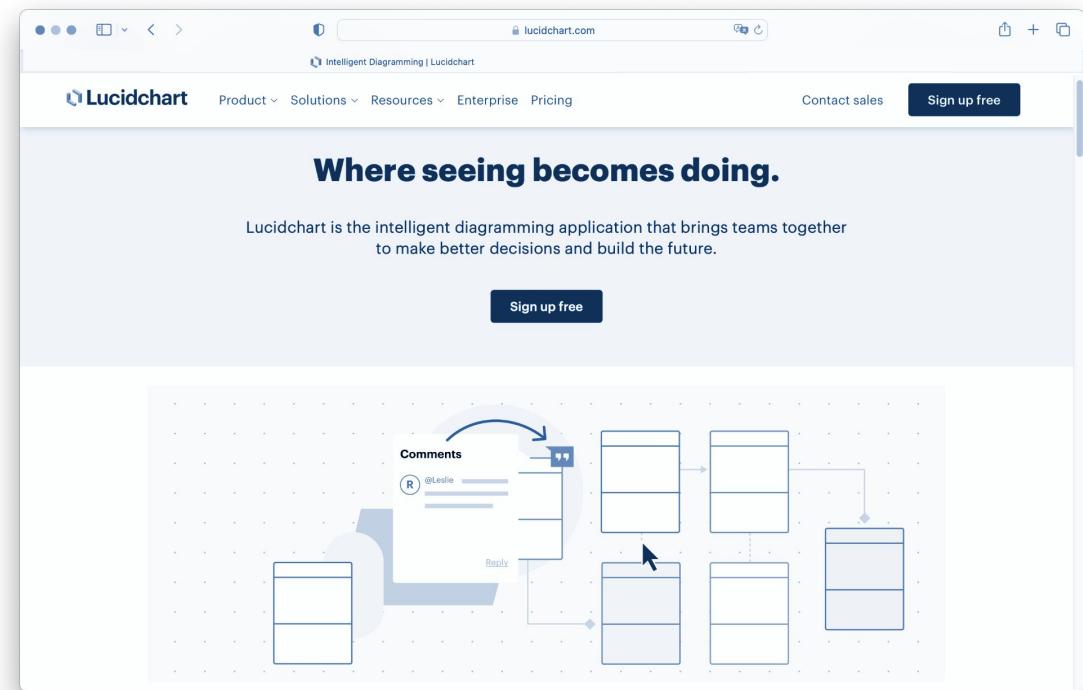


Planificar la bb.dd

Draw.io

Aunque es principalmente utilizado para diagramas, Draw.io también puede ser aplicado en el diseño/diagramado de bases de datos relacionales. Permite crear tablas y establecer relaciones entre ellas de forma visual y flexible.

<https://app.diagrams.net/>



Planificar la bb.dd

Cualquiera de estas herramientas simplifican el proceso de diseño de bases de datos relacionales, brindándonos una interfaz amigable y opciones flexibles para construir estructuras de datos eficientes y bien organizadas.

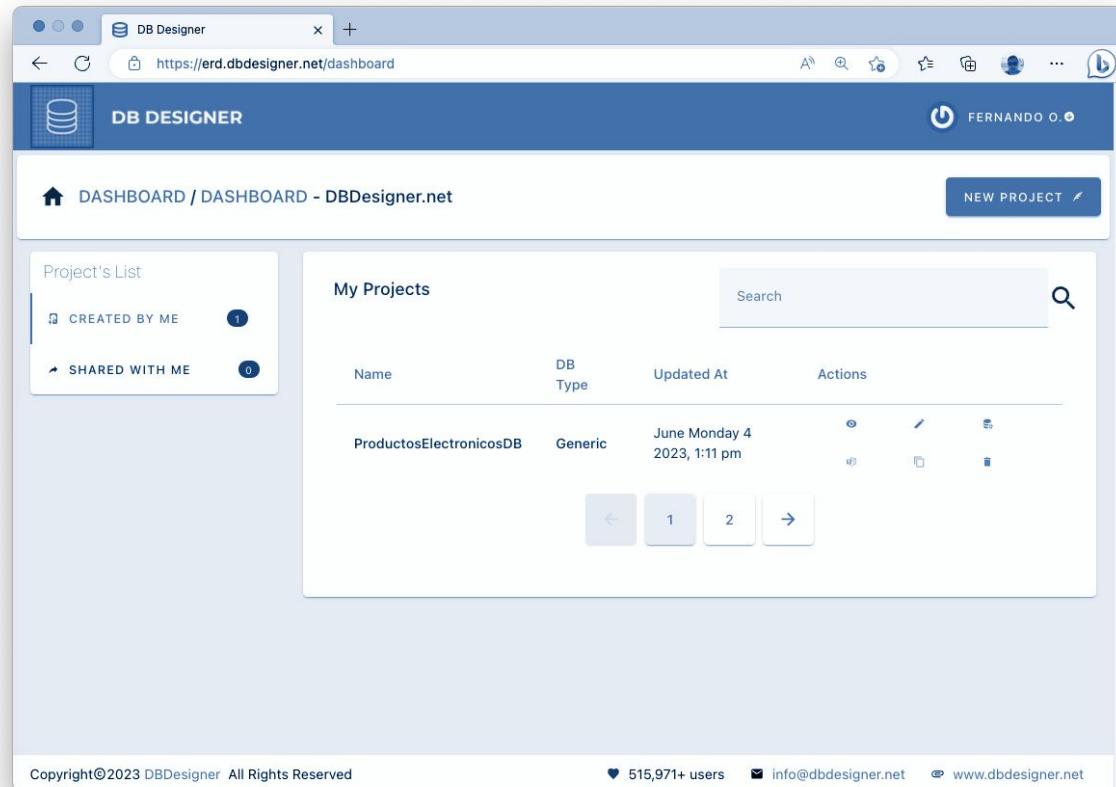
Por una característica destacable aplicada al diseño de esquemas externos, elegiremos a DB Designer como nuestra aplicación amiga para llevar adelante este proceso.



DB Designer



DBDESIGNER.NET

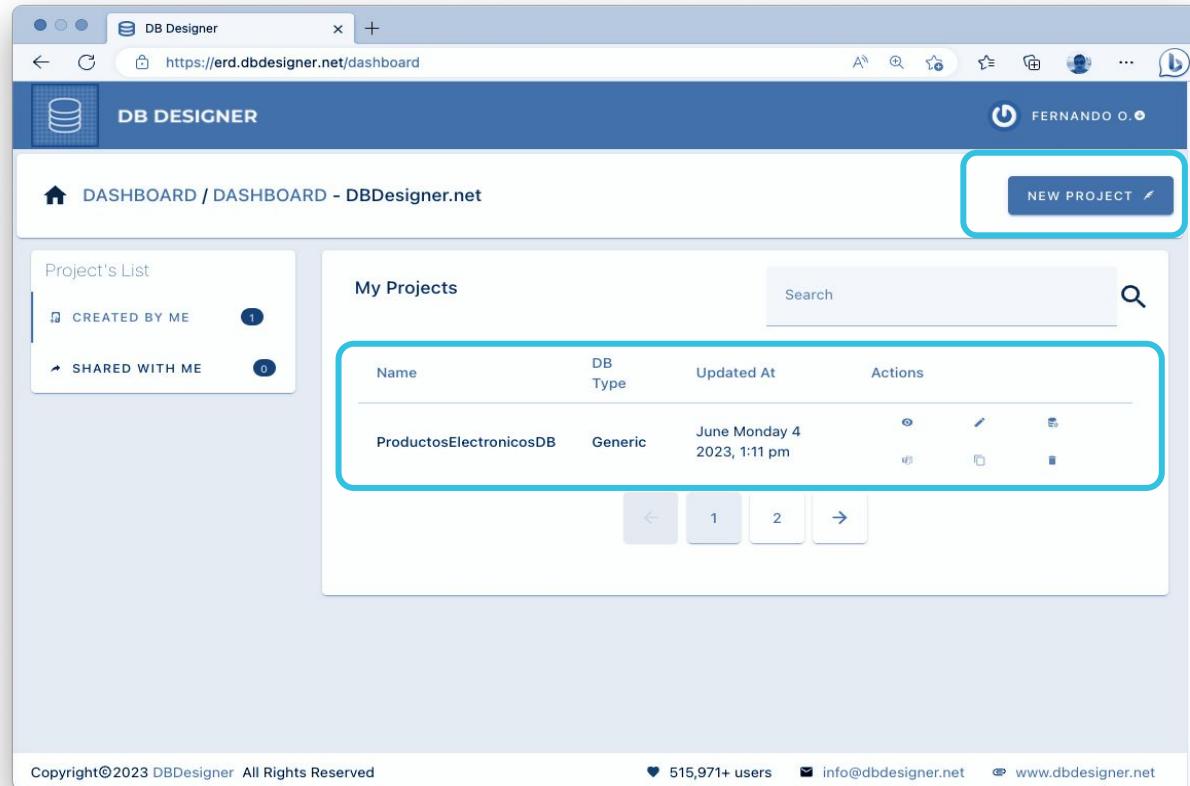


DB Designer nos ofrece una herramienta de fácil uso y completa, para poder elaborar un diseño de esquema de bb.dd de forma fácil y rápida.

Para utilizar esta herramienta, desde su sitio web principal podemos identificarnos como usuaria rápidamente, utilizando una cuenta de Github o Google.



DBDESIGNER.NET

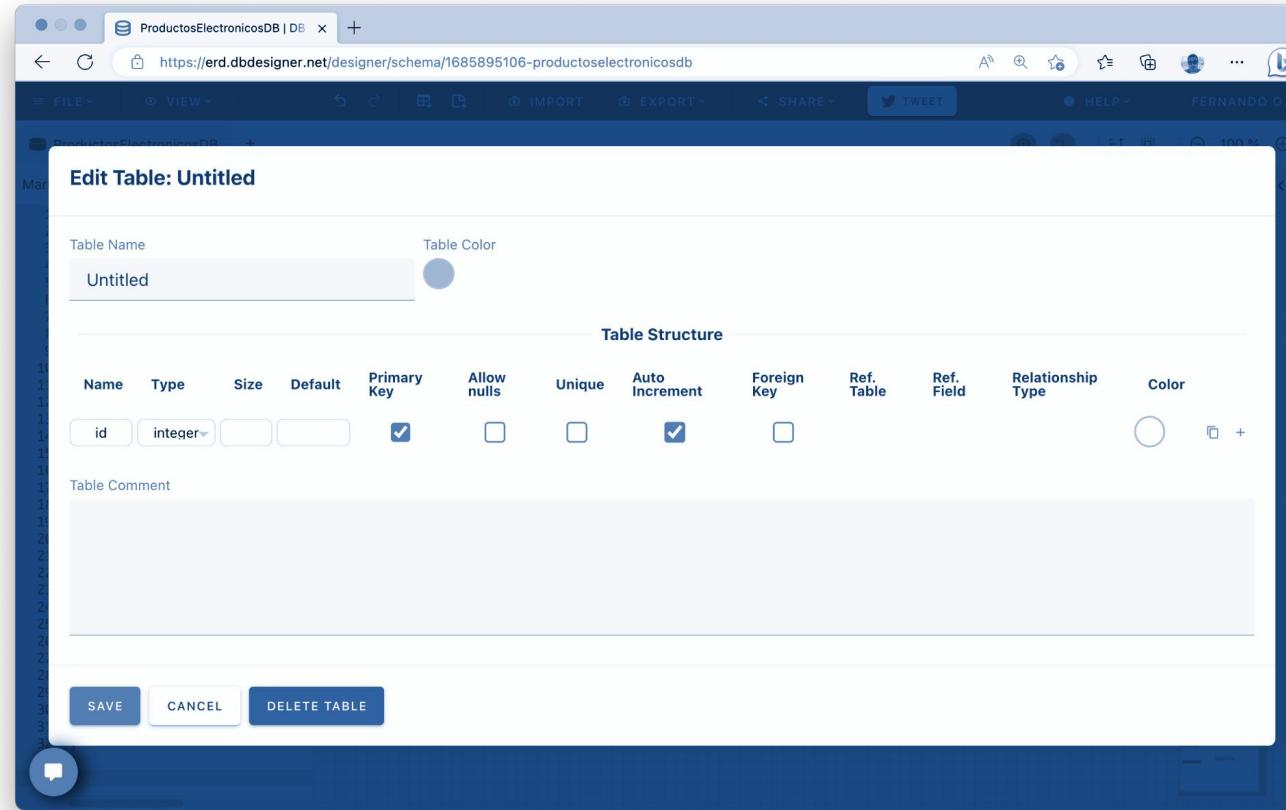


Una vez que ingresamos a **DB Designer**, en el apartado **Dashboard** podremos **crear un nuevo proyecto** pulsando en el botón homónimo.

También podemos **visualizar los diseños que hayamos creado** previamente y abrir cualquiera de ellos para seguir trabajando.

Crear tablas, índices y relaciones

Crear tablas, índices y relaciones

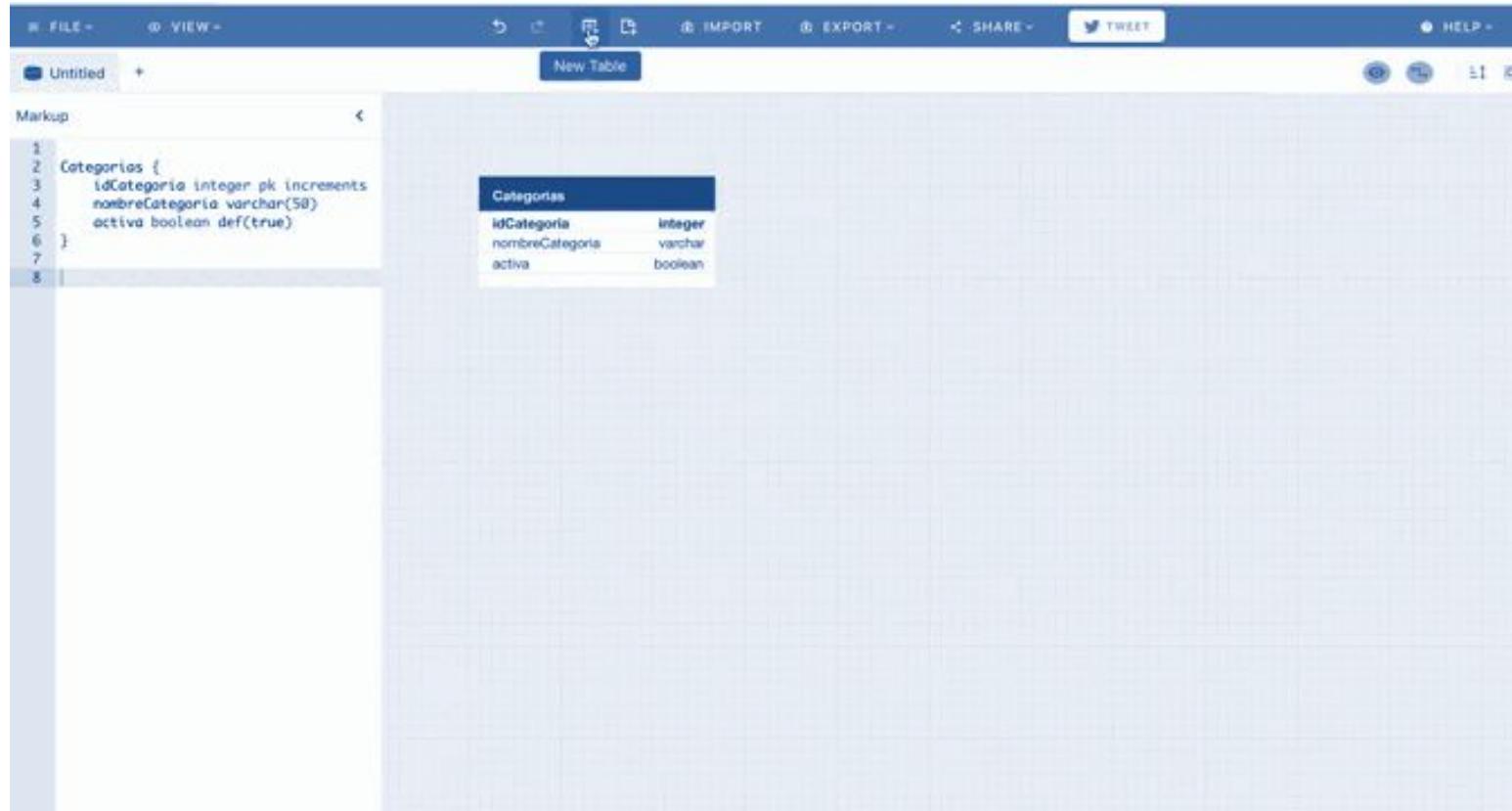


Una vez iniciado un nuevo proyecto, podemos comenzar a crear las tablas de este desde la pantalla principal.

Su uso es muy intuitivo y fácil de comprender. Se asimila mucho a la forma en la cual MySQL nos permite crear nuevas tablas de manera gráfica.



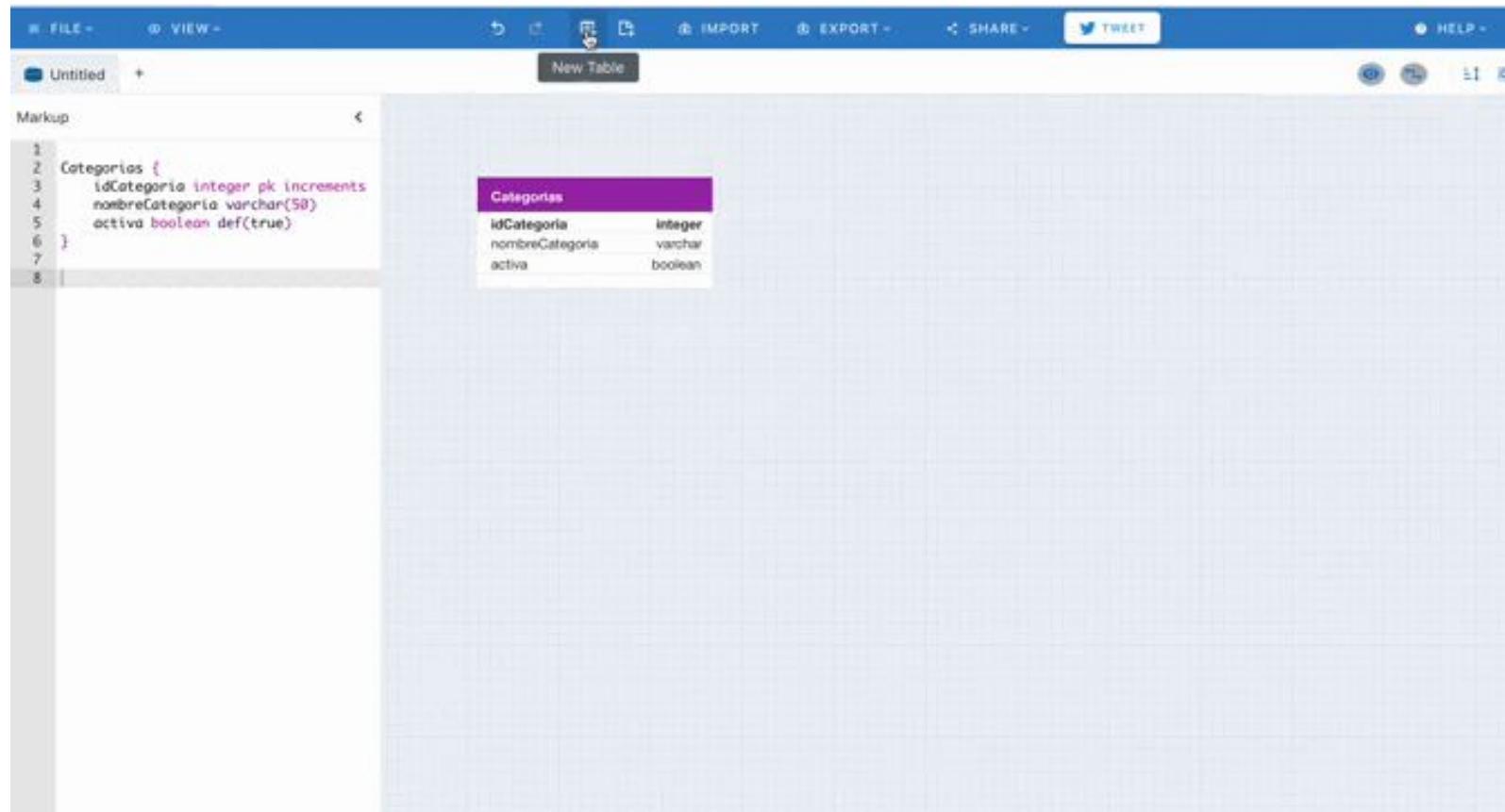
Crear tablas, índices y relaciones



Comenzamos a definir entonces la primera tabla, indicándole su nombre en el apartado superior de la ventana emergente.

Luego vamos definiendo sus campos, tipos de datos y eventualmente las relaciones entre claves primarias y foráneas.

Crear tablas, índices y relaciones



The screenshot shows a software interface for designing database schemas. On the left, there is a code editor window titled "Untitled" containing the following SQL-like code:

```
1 Categories {  
2     idCategoria integer pk increments  
3     nombreCategoria varchar(50)  
4     activa boolean def(true)  
5 }  
6  
7  
8
```

On the right, there is a table definition window titled "Categories". It shows the following schema:

Categorías	
idCategoria	integer
nombreCategoria	varchar
activa	boolean

Tengamos presente en este proceso, comenzar a diseñar todas las tablas de menor peso o “*entidad débil*”. Por ejemplo: **Categorías**, **Características**, etcétera.

Esto nos permitirá luego armar las tablas definidas como “*entidad fuerte*”, y establecer las claves foráneas con las claves primarias correspondientes.



Crear tablas, índices y relaciones

ProductosCaracteristicas

Table Color: Yellow

Table Structure:

Type	Size	Default	Primary Key	Allow nulls	Unique	Auto Increment	Foreign Key	Ref. Table	Ref. Field	Relationship Type	Color
integer			<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Productos	idProducto	Many to Many	
integer			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Caracteristicas	idCaracteristica	Many to Many	
varchar	60	N/A	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

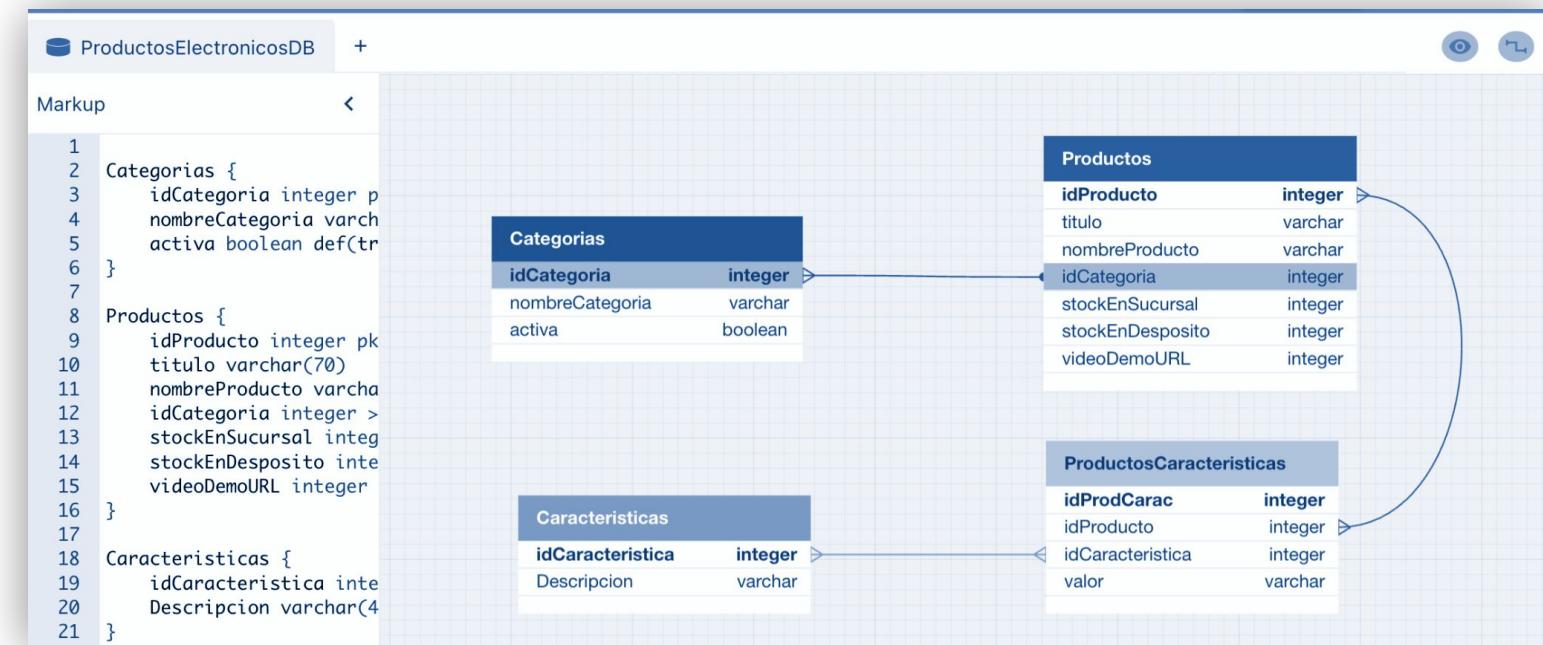
Todas las relaciones del tipo clave foránea pueden aplicarse durante el diseño de la tabla. Es por ello que, para agilizar tiempos, debemos crear primero las tablas de menor peso. De esta forma estarán disponibles cuando debamos definir estas claves.



Crear tablas, índices y relaciones

A medida que vamos creando más tablas y sus correspondientes relaciones, veremos el diseño de la bb.dd en cuestión dentro del panel canvas de esta aplicación web.

Si marcamos con diferentes colores las tablas más importantes, tendremos una vista destacada del diseño en cuestión, y será más fácil identificar las tablas principales cuando este modelo haya crecido significativamente.

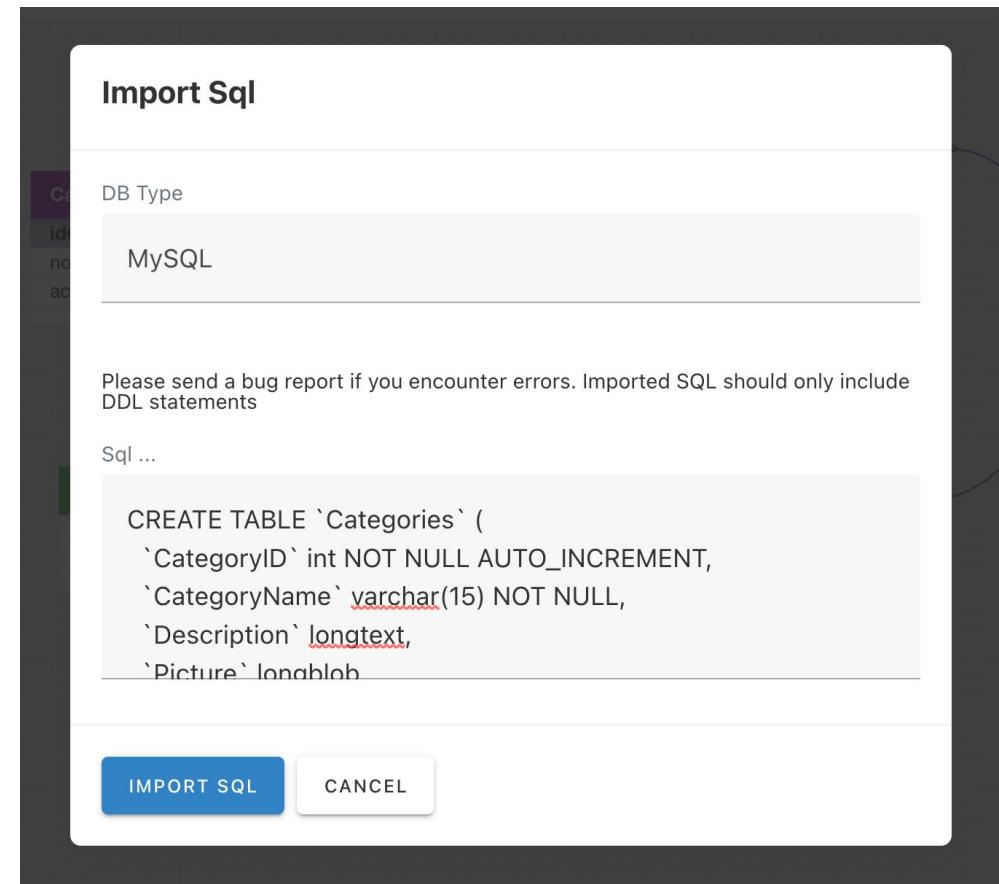


Exportar nuestro diseño

Exportar nuestro diseño

En el menú superior encontraremos dos puntos de menú importantes: **IMPORT** y **EXPORT**.

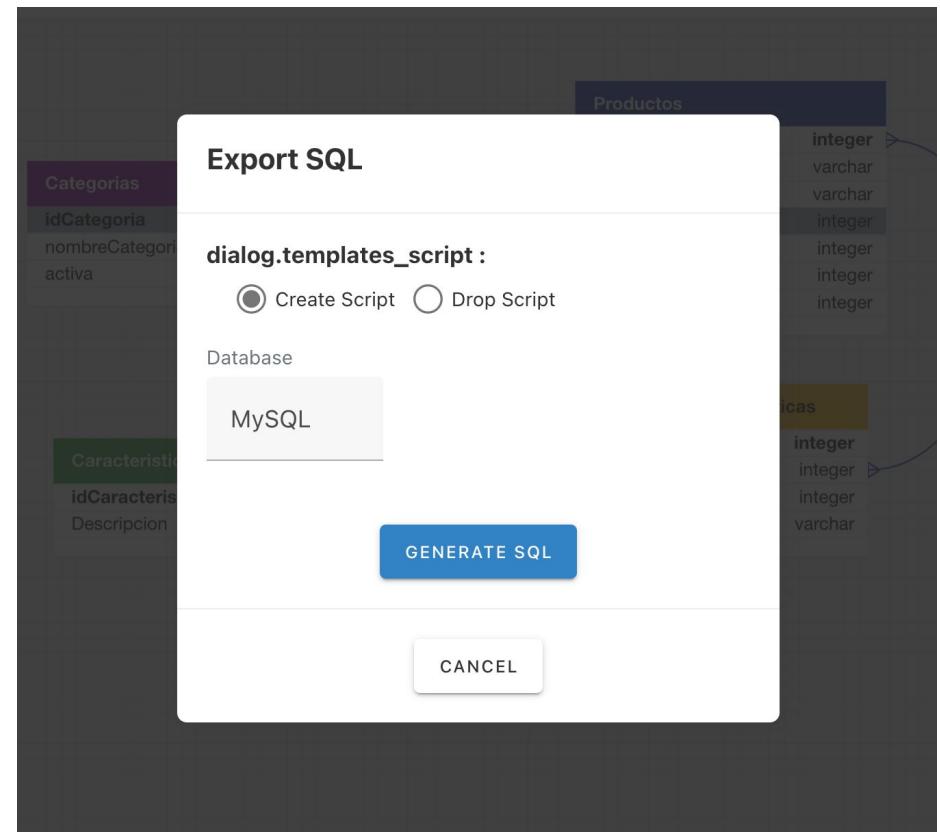
El primero de ellos, nos permite pegar un script de creación de tabla si es que ya tenemos un modelo de tabla similar diseñado en otra base de datos. Podemos pegar el script de creación para generar la misma dentro de nuestro diseño.



Exportar nuestro diseño

Mientras que, el punto de menú **EXPORT**, nos permite exportar el diseño que tengamos definido. Lo ideal siempre es exportar el diseño completo.

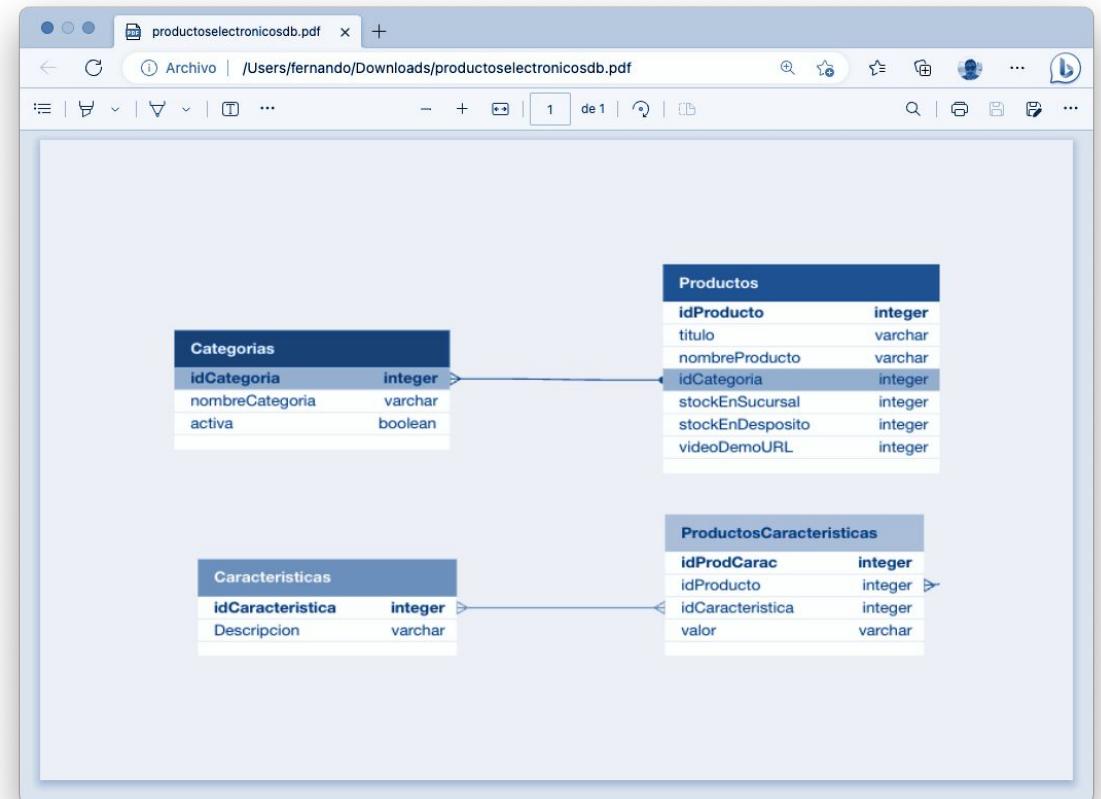
Dentro de las opciones a Exportar, encontramos a MySQL, SQL Server, SQLite, Oracle, Postgresql y Snowflake. Como podemos ver, la oferta de **DB Designer** es muy amplia y efectiva.



Exportar nuestro diseño

También contamos con la posibilidad de exportar el diseño del esquema de base de datos al formato PDF, por si debemos anexar el mismo a la documentación oficial del proyecto.

Y, por supuesto, también podemos generar un archivo en formato PNG, con este mismo diseño.

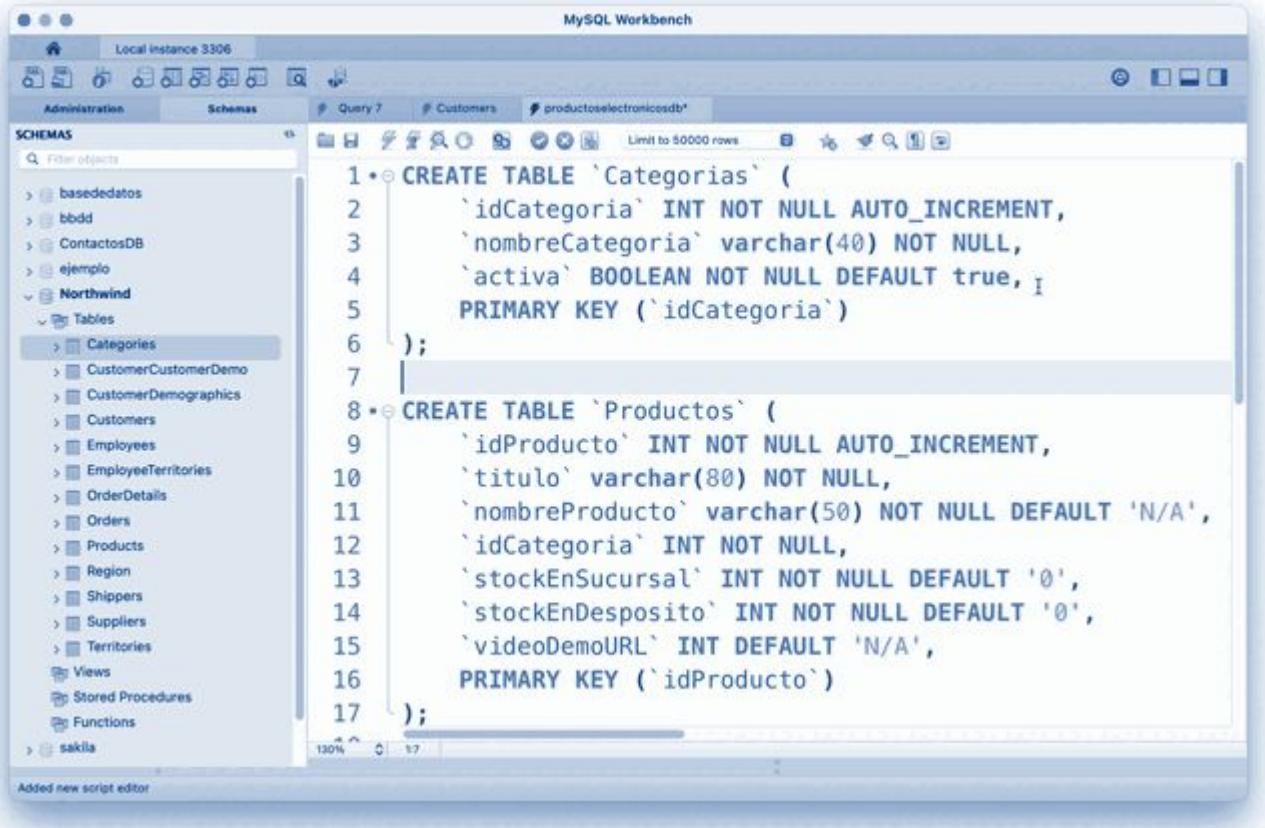


Crear la base de datos

Crear la base de datos

Con el archivo exportado al formato MySQL, ya podremos abrir el mismo con MySQL Workbench y ver su contenido.

Revisemos siempre que se haya exportado correctamente, y que no aparezca ningún error indicado por MySQL Workbench cuando editamos el archivo **.sql** en cuestión.



```
CREATE TABLE `Categorias` (
  `idCategoria` INT NOT NULL AUTO_INCREMENT,
  `nombreCategoria` varchar(40) NOT NULL,
  `activa` BOOLEAN NOT NULL DEFAULT true,
  PRIMARY KEY (`idCategoria`)
);

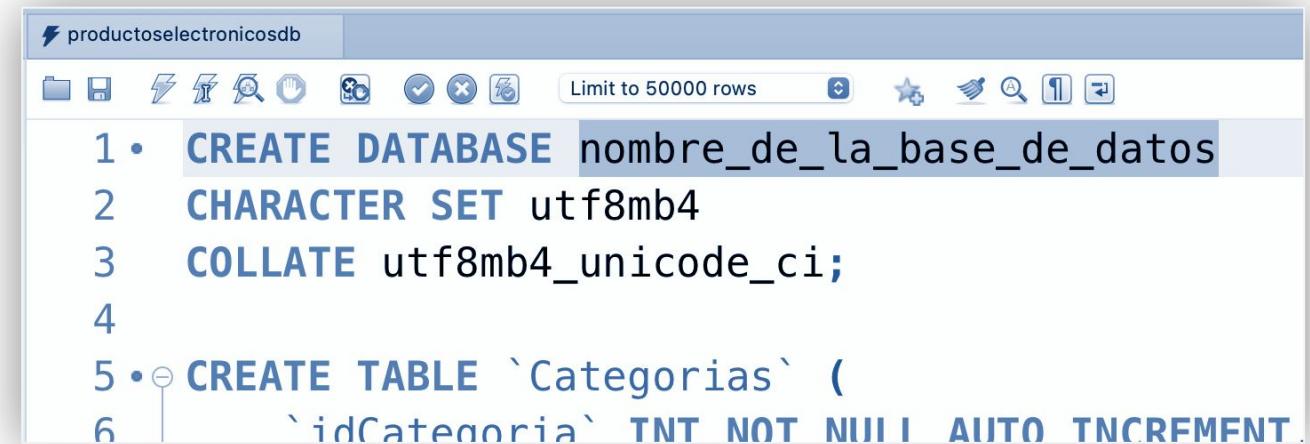
CREATE TABLE `Productos` (
  `idProducto` INT NOT NULL AUTO_INCREMENT,
  `titulo` varchar(80) NOT NULL,
  `nombreProducto` varchar(50) NOT NULL DEFAULT 'N/A',
  `idCategoria` INT NOT NULL,
  `stockEnSucursal` INT NOT NULL DEFAULT '0',
  `stockEnDeposito` INT NOT NULL DEFAULT '0',
  `videoDemouiRL` INT DEFAULT 'N/A',
  PRIMARY KEY (`idProducto`)
);
```



Crear la base de datos

También queda de nuestro lado agregar al inicio del script el nombre de la base de datos que crearemos.

Y, por supuesto, el juego de caracteres que utilizaremos. Si todo irá en nuestra lengua madre, entonces **UTF-8** es el **character set** a elegir.



The screenshot shows a MySQL Workbench interface with a database named 'productoselectronicosdb'. The main window displays a SQL script for creating a database and a table:

```
1 • CREATE DATABASE nombre_de_la_base_de_datos
2 CHARACTER SET utf8mb4
3 COLLATE utf8mb4_unicode_ci;
4
5 • CREATE TABLE `Categorias` (
    `idCategoria` INT NOT NULL AUTO_INCREMENT,
```

The first line of the script, 'CREATE DATABASE nombre_de_la_base_de_datos', has its first word 'CREATE' highlighted in blue, indicating it is currently selected or being edited.



Definir un script de inserción de datos

Definir un script de inserción de datos

Con casi toda la bb.dd diseñada, queda el proceso más importante pero a su vez más pesado de realizar: *migrar los datos*.

MySQL tiene muchas formas de importar los datos desde diferentes formatos, entre ellos JSON, a una bb.dd relacional.

Pero aquí comienza una ardua tarea, en especial por el tipo de datos JSON, que requiere mucho trabajo previo para adaptarlo al formato estructurado de MySQL.

```
... JSON

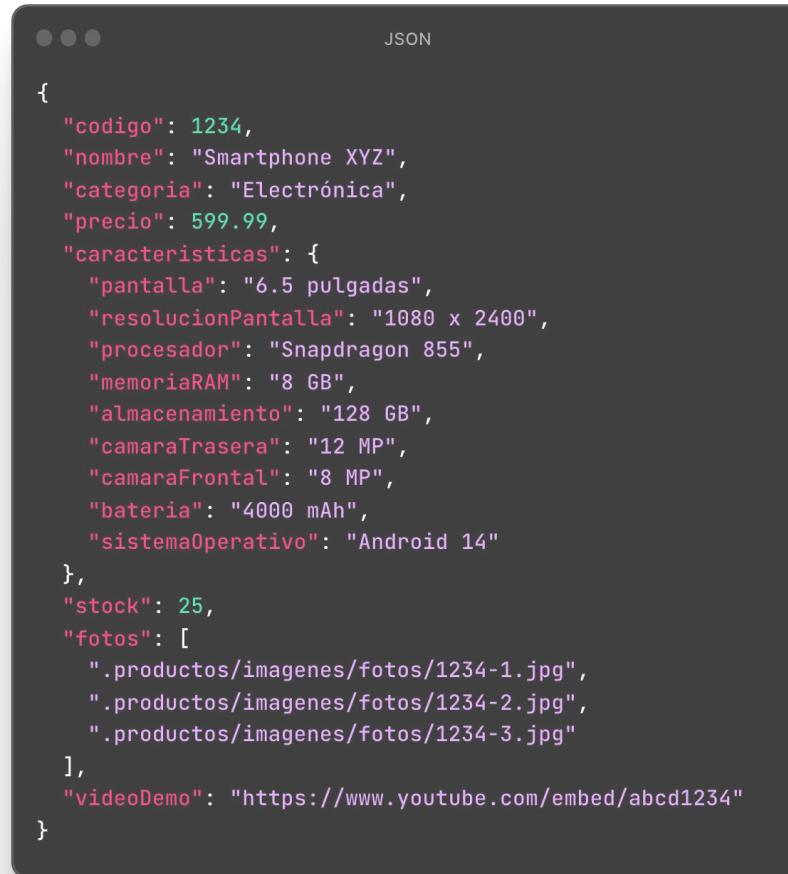
{
  "codigo": 1234,
  "nombre": "Smartphone XYZ",
  "categoria": "Electrónica",
  "precio": 599.99,
  "características": {
    "pantalla": "6.5 pulgadas",
    "resolucionPantalla": "1080 x 2400",
    "procesador": "Snapdragon 855",
    "memoriaRAM": "8 GB",
    "almacenamiento": "128 GB",
    "camaraTrasera": "12 MP",
    "camaraFrontal": "8 MP",
    "bateria": "4000 mAh",
    "sistemaOperativo": "Android 14"
  },
  "stock": 25,
  "fotos": [
    ".productos/imagenes/fotos/1234-1.jpg",
    ".productos/imagenes/fotos/1234-2.jpg",
    ".productos/imagenes/fotos/1234-3.jpg"
  ],
  "videoDemo": "https://www.youtube.com/embed/abcd1234"
}
```



Definir un script de inserción de datos

Tenemos varias formas de trabajar esta información, previo a importarla a la bb.dd SQL.

Una de ellas sería copiar el archivo JSON y comenzar a extraer los datos a un nuevo archivo, aplicando los filtros correspondientes de propiedades: valores, para dejar solo aquello que necesitemos.



A screenshot of a Mac OS X terminal window titled "JSON". The window contains a single JSON object representing a smartphone product. The object has properties like "codigo", "nombre", "categoria", "precio", "caracteristicas", "stock", and "fotos". The "caracteristicas" property is a nested object containing details about the screen, processor, memory, storage, cameras, battery, and operating system. The "fotos" property is an array of three image URLs.

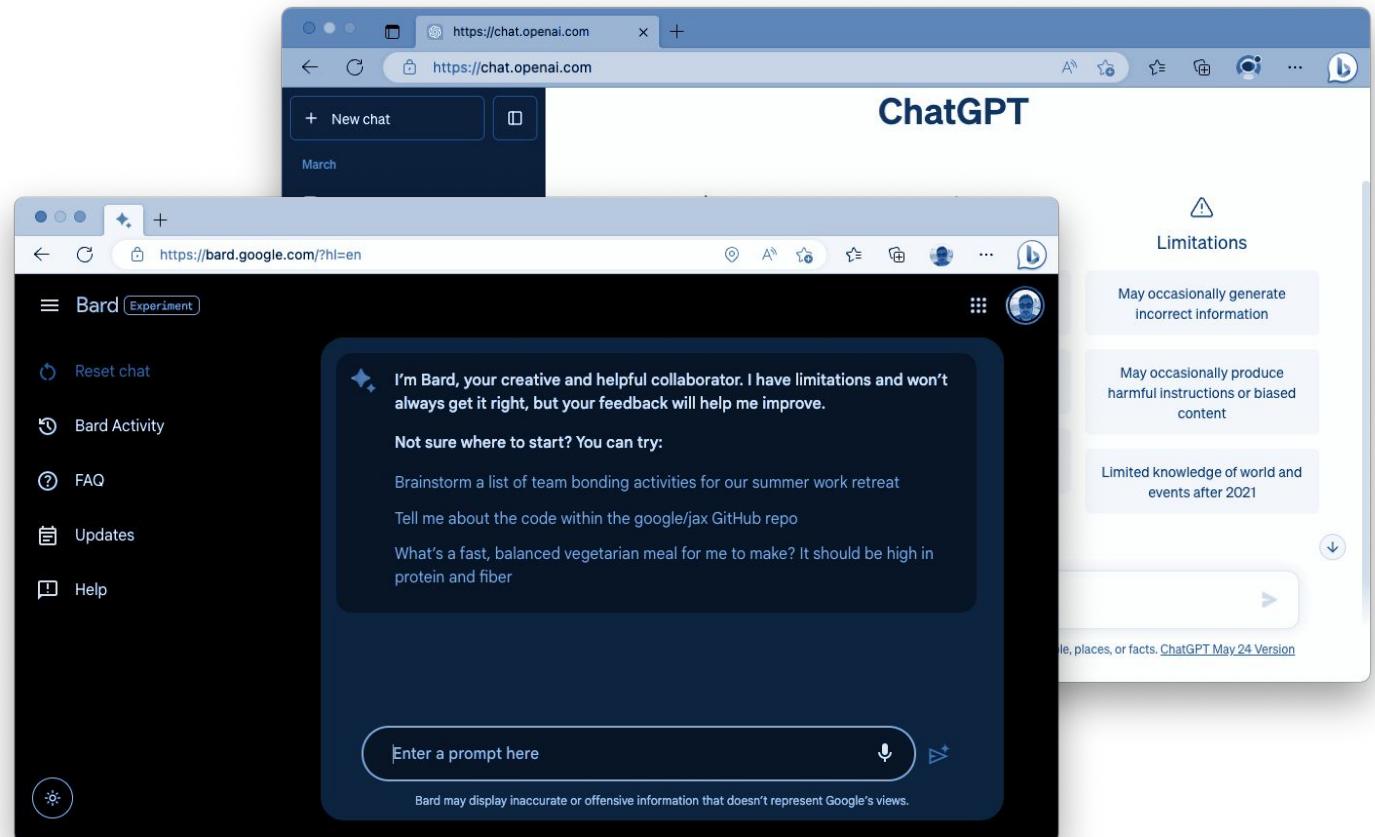
```
{
  "codigo": 1234,
  "nombre": "Smartphone XYZ",
  "categoria": "Electrónica",
  "precio": 599.99,
  "caracteristicas": {
    "pantalla": "6.5 pulgadas",
    "resolucionPantalla": "1080 x 2400",
    "procesador": "Snapdragon 855",
    "memoriaRAM": "8 GB",
    "almacenamiento": "128 GB",
    "camaraTrasera": "12 MP",
    "camaraFrontal": "8 MP",
    "bateria": "4000 mAh",
    "sistemaOperativo": "Android 14"
  },
  "stock": 25,
  "fotos": [
    ".productos/imagenes/fotos/1234-1.jpg",
    ".productos/imagenes/fotos/1234-2.jpg",
    ".productos/imagenes/fotos/1234-3.jpg"
  ],
  "videoDemo": "https://www.youtube.com/embed/abcd1234"
}
```



Definir un script de inserción de datos

Otra opción muy efectiva en la actualidad, es la de utilizar las inteligencias artificiales actuales, para que nos ayuden a adaptar rápidamente la estructura para que esta quede lista para ser insertada en la bb.dd SQL.

[ChatGPT](#) es una de las más vigentes, y [Google Bard](#) puede ser otra alternativa.



Definir un script de inserción de datos

FE

```
{  
  "codigo": 1234,  
  "nombre": "Smartphone XYZ",  
  "categoria": "Electrónica",  
  "precio": 599.99,  
  "características": {  
    "pantalla": "6.5 pulgadas",  
    "resolucionPantalla": "1080 × 2400",  
    "procesador": "Snapdragon 855",  
    "memoriaRAM": "8 GB",  
    "almacenamiento": "128 GB",  
    "camaraTrasera": "12 MP",  
    "camaraFrontal": "8 MP",  
    "bateria": "4000 mAh",  
    "sistemaOperativo": "Android 14"  
  },  
  "stock": 25,  
  "fotos": [  
    ".productos/imagenes/fotos/1234-1.jpg",  
    ".productos/imagenes/fotos/1234-2.jpg",  

```

con este objeto JSON modelo que te compartí, crea una consulta de inserción SQL sobre una tabla productos, que posee los siguientes campos:

idProducto, titulo, nombreProducto, idCategoria, stockEnSucursal, videoDemoURL.

Utiliza para cada campo mencionado, la información de las siguientes propiedades JSON: código, nombre, 'repite nombre', categoria (pon 1 como valor), stock, videoDemo.

Utiliza para cada campo mencionado, la información de las siguientes propiedades JSON: código, nombre, 'repite nombre', categoria (pon 1 como valor), stock, videoDemo.



Definir un script de inserción de datos

 La consulta de inserción SQL para la tabla "productos" utilizando los valores del objeto JSON proporcionado sería la siguiente:

```
sql Copy code
INSERT INTO productos (idProducto, titulo, nombreProducto, idCategoria, stockEnSucursal, videoDemoURL)
VALUES (1234, 'Smartphone XYZ', 'Smartphone XYZ', 1, 25, 'https://www.youtube.com/watch?v=...')
```

En esta consulta, se insertan los valores correspondientes en los campos de la tabla "productos" de acuerdo a la estructura del objeto JSON. El campo "idProducto" se obtiene del valor de "codigo", "titulo" y "nombreProducto" se obtienen del valor de "nombre", "idCategoria" se establece como 1, "stockEnSucursal" se obtiene del valor de "stock" y "videoDemoURL" se obtiene del valor de "videoDemo".

Una vez que obtenemos la respuesta, podemos analizar si efectivamente es lo que esperábamos.

En el caso que falte información, o debamos cambiar algún dato erróneo, podemos seguir interactuando con esta IA para poder corregir estos datos y que regenere las consultas que necesitamos.



Definir un script de inserción de datos

Imagina todos los escenarios donde la I.A. nos puede dar una gran mano para acelerar los procesos de migración y adaptación:

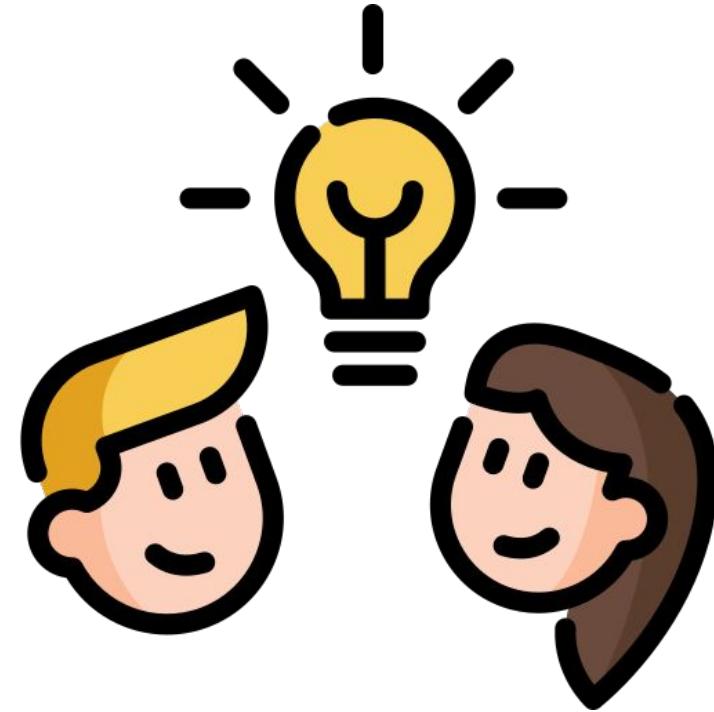
- cientos o miles de datos repetidos que pueden analizarse y convertirse en un set de datos uniforme
- generar automáticamente consultas de inserción de esos cientos de datos uniformes en una bb.dd SQL
- resumir decenas de horas de pensamiento lógico y de código para limpiar y procesar datos, a tan solo algunas horas de dedicación
- entre otros tantos escenarios...



Definir un script de inserción de datos

Para nuestro actual escenario, el de aprendizaje, podemos implementar sin problemas las ayudas basadas en I.A., pero, en el ámbito laboral general, debemos tener cuidado y/o consultar si podemos apoyarnos en herramientas como Chat GPT, Google Bard, o Copilot, entre otras, dado que existen muchas cláusulas de confidencialidad.

Esto último hace que, compartir datos sensibles con una herramienta externa a la empresa, no siempre sea bien visto.

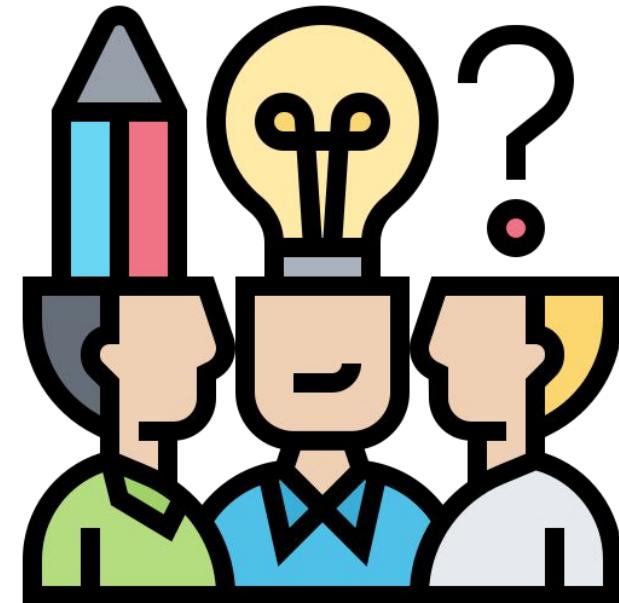


Pre-Entrega 3

Pre-entrega 3

La pre-entrega 3 requerirá que elabores un proyecto Node.js + MySQL + Documentación acorde.

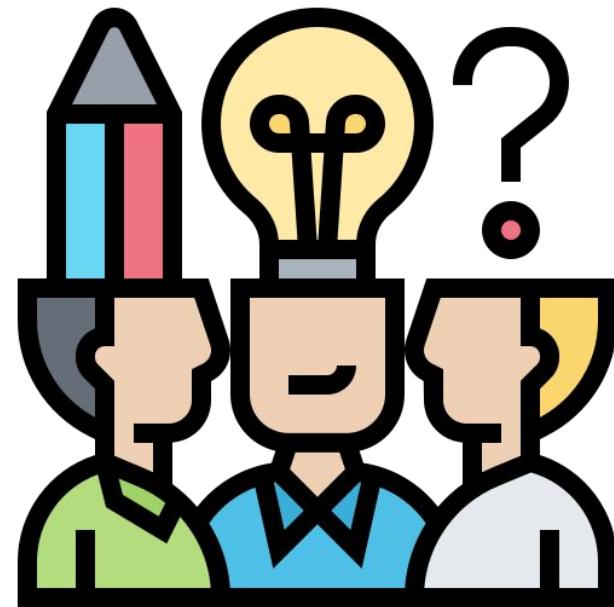
Aprovecha el contenido visto en la clase de hoy, para comenzar a diseñar parte del trabajo que deberás entregar al finalizar el curso.



Pre-entrega 3

Basándote en el modelo del archivo **trailerflix.json**, comienza a diseñar el modelo de datos correspondiente a una plataforma de streaming, utilizando la información del archivo JSON que aprovechaste en las primeras clases.

Analiza todas sus propiedades y diseña de este modelo, al menos 4 a 6 tablas relacionales, donde luego deberás migrar los datos de este JSON.



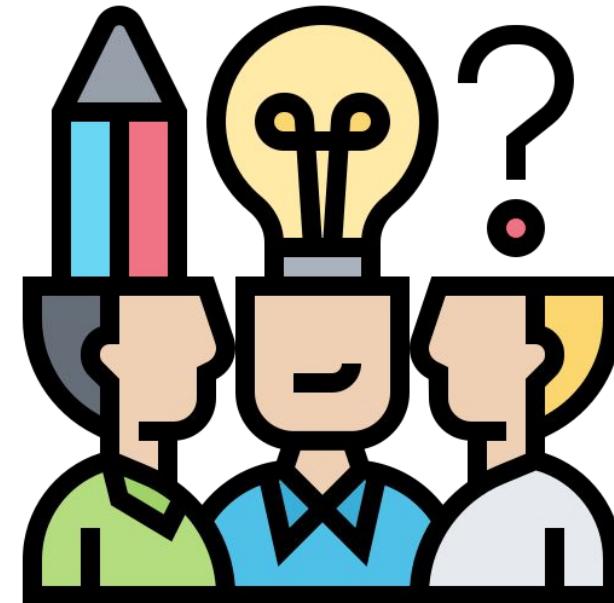
Pre-entrega 3

Aprovecha la plataforma DB Designer que te sugerimos en este encuentro, para realizar un diseño efectivo de la bb.dd y sus tablas.

La bb.dd se llamará trailerflix y deberá tener al menos 6 tablas, como ser:

- contenido
- categorías
- géneros
- actores

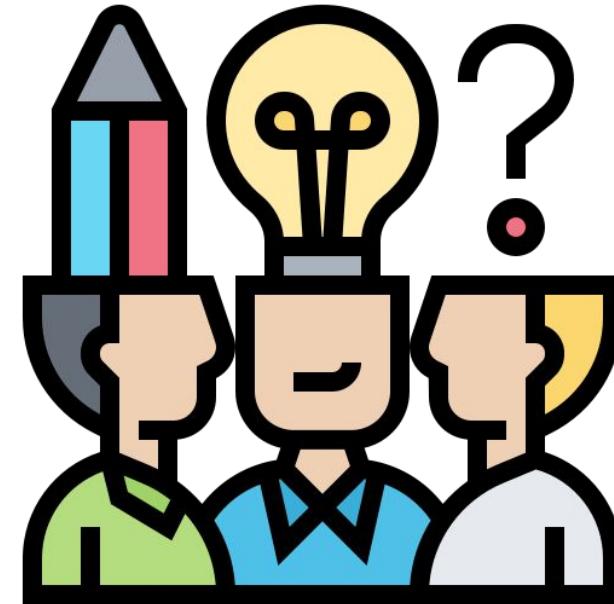
Más las tabla intermedias que conectan algunas de estas tablas con las otras



Pre-entrega 3

Una vez que tengas el diseño bocetado en una hoja, vuélcalo en DB Designer y genera el archivo de exportación para generar las tablas y sus relaciones en MySQL.

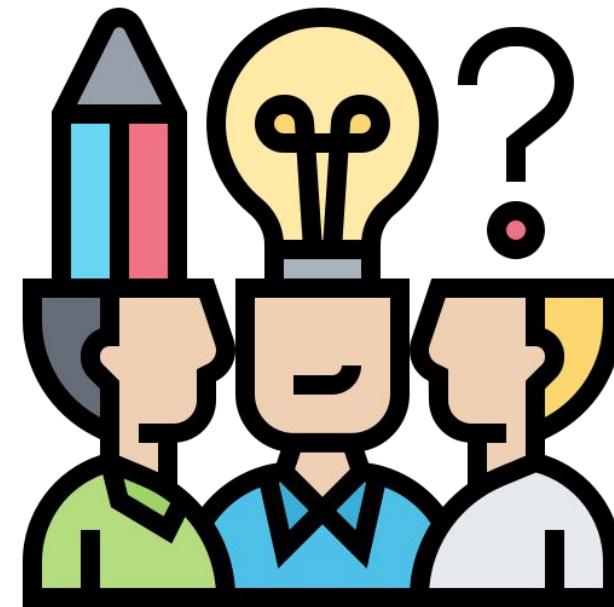
Aprovecha luego alguna I.A. de las sugeridas. Compártele un objeto JSON del archivo traileflix, agrega la información de las tablas SQL que tienes, y que te sugiera cómo separar el contenido de este archivo para insertar el mismo más rápidamente en MySQL.



Pre-entrega 3

Aprovecha a adelantar estos pasos con tiempo, porque debes tener esto concluido para luego continuar con el resto del trabajo final:

- crear los endpoints necesarios para consultar esta información
- realizar la documentación para saber qué cómo utilizar los endpoint



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



Argentina programa 4.0



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*

Clase 28: Sequelize y Node JS

Utilizar bases de datos MySQL

Agenda de hoy

- A. Sequelize
 - a. Fundamentos de Sequelize
 - b. Ventajas de uso
- B. Instalar Sequelize
- C. Configurar Sequelize en Node.js
- D. Los métodos más importantes
- E. Definir un CRUD con MySQL
- F. Crear los endpoint

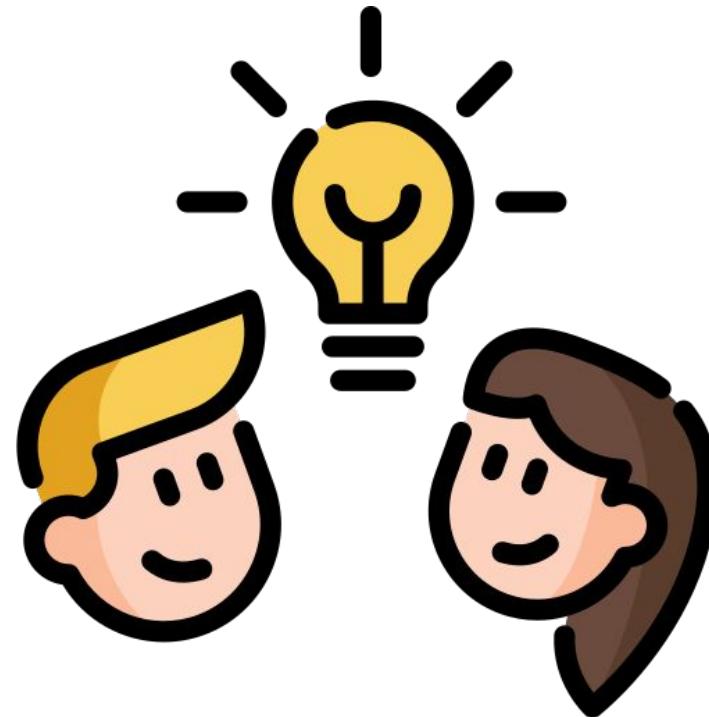


Welcome back!

Nos alejaremos por una clase de MySQL, para volver a trabajar con Node.js.

La idea es poder conocer herramientas Node que nos permitan integrar bases de datos SQL al entorno de desarrollo de aplicaciones backend.

Hoy veremos qué es Sequelize y cómo podemos interactuar desde este con MySQL, de cara a la segunda etapa de elaboración de nuestra pre-entrega 3.



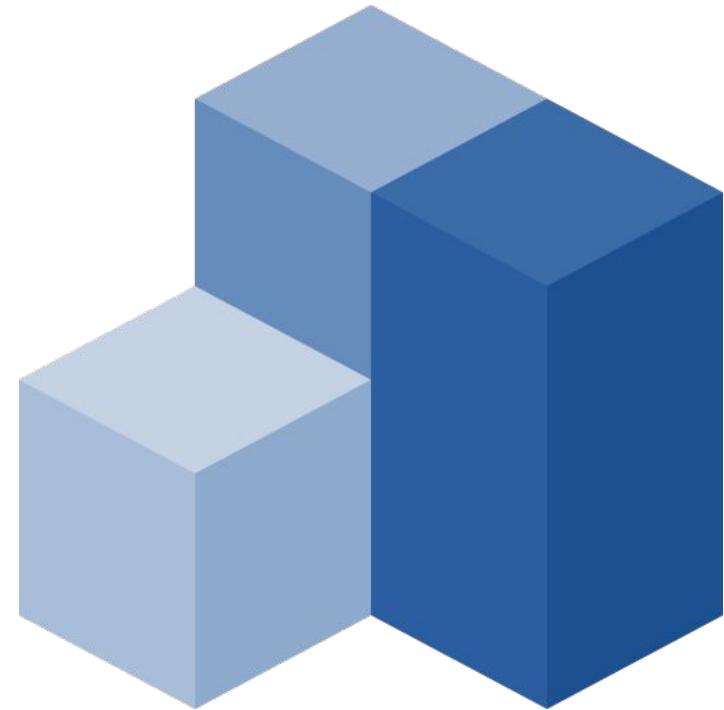
Sequelize



Sequelize

Sequelize es una librería JS que actúa como una herramienta de abstracción de base de datos. Simplifica y agiliza la forma en que interactuamos con bases de datos en nuestras aplicaciones.

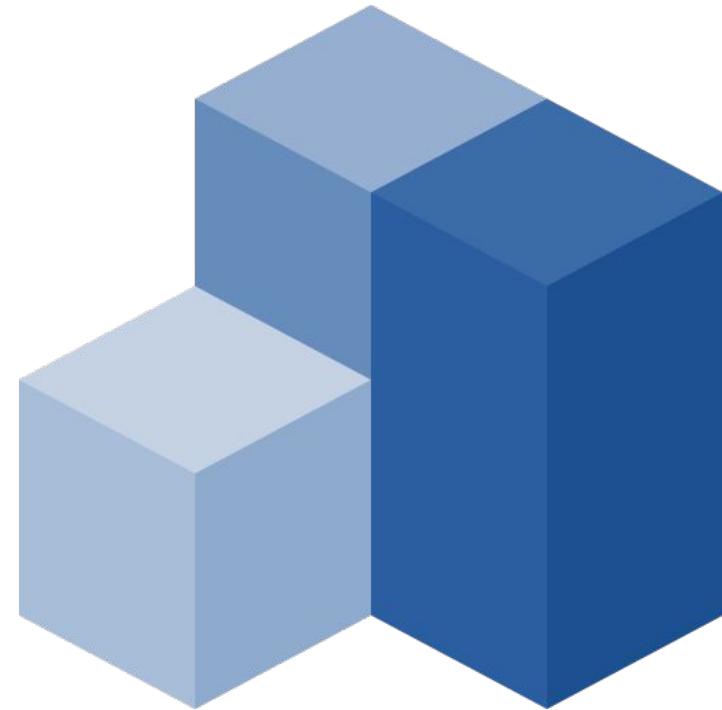
Para evitar escribir consultas SQL complejas y manejar manualmente la conexión e intercambio de datos con la bb.dd, Sequelize proporciona una capa de abstracción que nos permite realizar estas tareas más sencillamente.



Sequelize

Con Sequelize, podemos interactuar con la base de datos utilizando un lenguaje de programación familiar como JavaScript, lo que facilita el proceso de desarrollo.

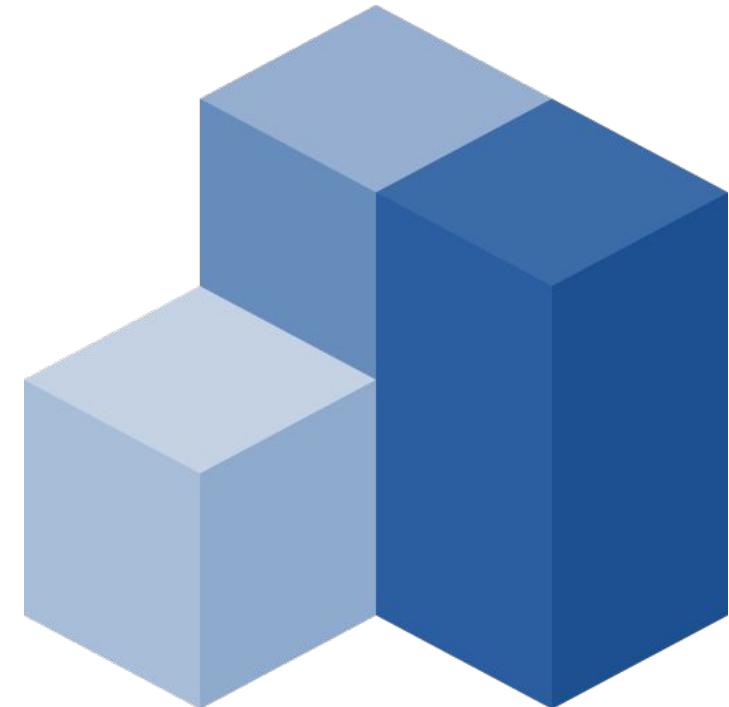
En lugar de tener que aprender y escribir consultas en SQL, podemos utilizar métodos y funciones de Sequelize para realizar operaciones de creación, lectura, actualización y eliminación de datos en la base de datos.



Sequelize

Además, proporciona un conjunto de herramientas que nos permiten modelar los datos de nuestra aplicación de una manera más intuitiva. Podemos definir modelos que representan las tablas de la base de datos, y luego trabajar con los datos de forma similar a cómo lo haríamos con objetos en JavaScript.

Esto simplifica aún más la interacción con la base de datos y nos ayuda a organizar y estructurar la información de manera más eficiente.

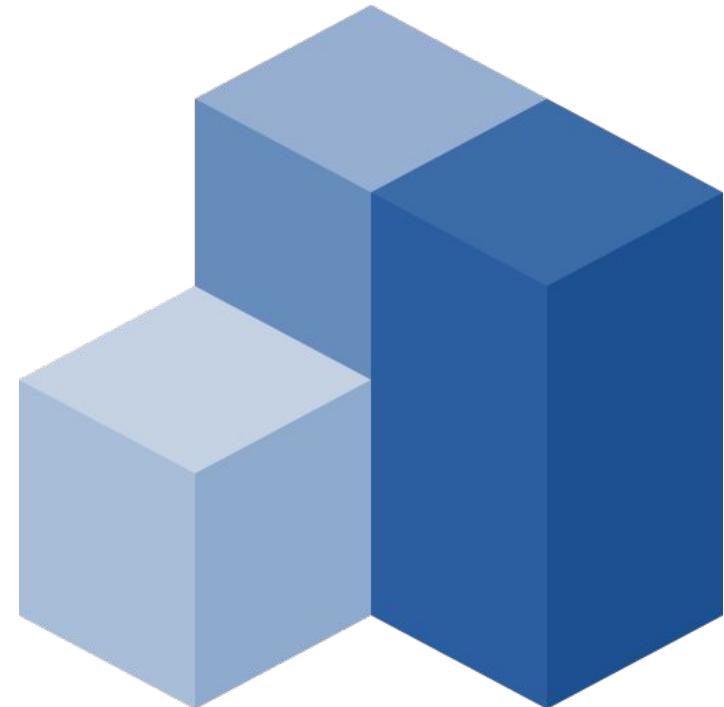


Sequelize

Migraciones y control de versiones

Dentro de los beneficios de utilizar Sequelize, encontramos el poder realizar migraciones, o cambios estructurales en la bb.dd, de forma controlada y organizada.

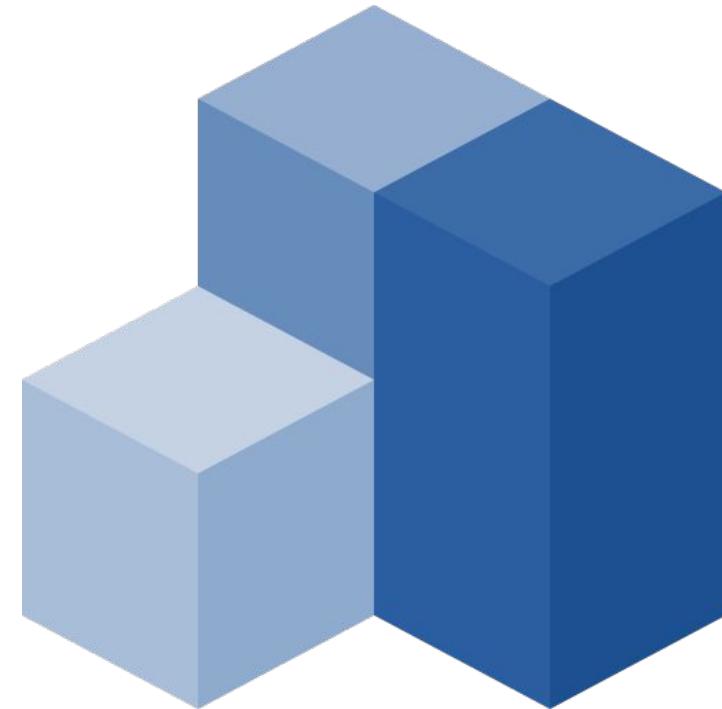
Podremos crear y aplicar las migraciones de manera incremental, para facilitar el control de versiones y el trabajo con otras compañeras del equipo de desarrollo.



Sequelize

Ventajas de implementar Sequelize

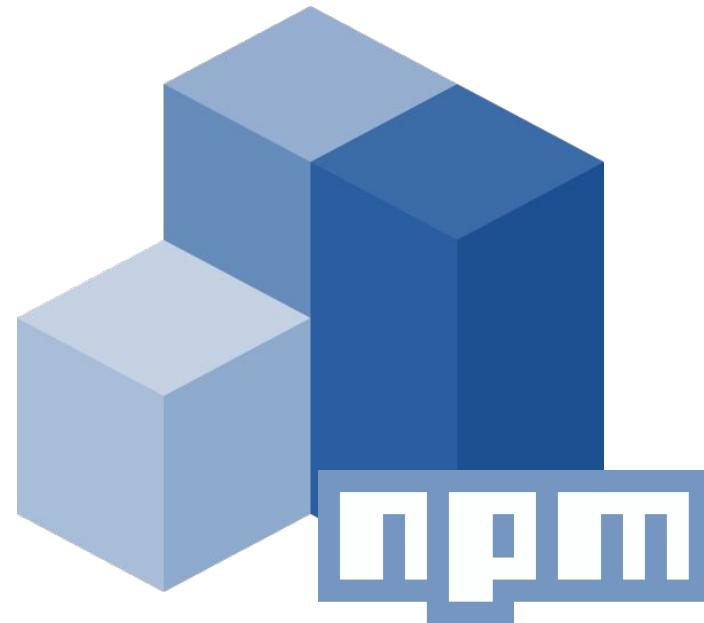
- Facilita la interacción con la bb.dd, simplificando las consultas y el modelado de datos
- Proporciona un control de versiones para los cambios en la estructura de la base de datos
- Es compatible con MySQL, MariaDB, PostgreSQL SQLite, SQL Server, lo que nos brinda flexibilidad en nuestras aplicaciones



Instalar Sequelize

Instalar Sequelize

En la web oficial: www.sequelize.org encontraremos la documentación oficial de esta librería y la información referente a cómo instalar la misma dentro de un proyecto de Node.js.



Instalar Sequelize

El comando para su instalación, se cumplimenta tal cual venimos trabajando hasta ahora dentro de los proyectos Node.js. NPM será nuestra vía de acceso a descargar Sequelize.

```
...          npm  
npm install --save sequelize
```

Instalar Sequelize

Como Sequelize permite trabajar con múltiples bases de datos del tipo SQL, necesitaremos instalar aparte el soporte correspondiente de acuerdo al “sabor” de SQL que necesitamos trabajar.



Instalar Sequelize

Sequelize llama a esto “*dialectos*”, tal como sucede con el idioma chino en sus diferentes regiones, el ruso, o en la mismísima Italia.

Los dialectos son los que nos permitirán comunicarnos de una forma efectiva con SQL Server, o MySQL, o Postgresql, etc.



Instalar Sequelize

Una vez instalado Sequelize, debemos instalar manualmente el driver correspondiente a la bb.dd con la que trabajaremos. Este driver cuenta con el “dialecto” necesario para que Sequelize se comunique de manera efectiva con el motor de MySQL o cualquier otra bb.dd.

```
● ● ● npm  
# One of the following:  
$ npm install --save pg pg-hstore # Postgres  
$ npm install --save mysql2  
$ npm install --save mariadb  
$ npm install --save sqlite3  
$ npm install --save tedious # Microsoft SQL Server  
$ npm install --save oracledb # Oracle Database
```



Instalar Sequelize

Aquí tenemos representado el comando correspondiente según la bb.dd que utilizemos.

En nuestro caso, instalaremos el que nos permite trabajar con MySQL.

```
● ● ●          npm  
  
# One of the following:  
$ npm install --save pg pg-hstore # Postgres  
$ npm install --save mysql2  
$ npm install --save mariadb  
$ npm install --save sqlite3  
$ npm install --save tedious # Microsoft SQL Server  
$ npm install --save oracledb # Oracle Database
```



Configurar Sequelize en Node.js



Configurar Sequelize en Node.js

Ya teniendo instalada la librería Sequelize y el mecanismo de dialecto correspondiente a nuestra base de datos, ya podemos integrar Sequelize a Node.js.

Para ello, importamos en principio la librería, tal como venimos trabajando con otras librerías anteriormente.



Configurar Sequelize en Node.js

Definimos entonces una constante que nos permite comenzar a manipular Sequelize en nuestros proyectos Node.js.

Luego de referenciarla, debemos pensar en establecer el mecanismo que nos conecte a la bb.dd.

```
● ● ● Node.js  
const Sequelize = require('sequelize');
```



Configurar Sequelize en Node.js

La conexión a la base de datos debemos pensar que será tratada tal como lo hicimos anteriormente con MongoDB.

- Nos conectamos
- Operamos
- Nos desconectamos

Esto nos ayudará a no sobrecargar con transacciones y múltiples conexiones al motor de MySQL.



Configurar Sequelize en Node.js

Otro de los puntos que debemos tener en cuenta para conectarnos a la bb.dd, es volcar la información de la conexión en un archivo **.env**.

De esta forma agregamos una capa más de seguridad, y podemos manejar por separado lo que será la configuración de nuestro ambiente de desarrollo y el futuro ambiente de testing, el de producción, etcétera.



Configurar Sequelize en Node.js

Debemos instanciar la librería **Sequelize**, informando en esta el nombre de la bb.dd, el nombre de usuario, y la contraseña, a través de parámetros separados.

Luego, en un objeto literal que también viaja como parámetro, definimos la URL del servidor de bb.dd, su dialecto, y cualquier otro parámetro adicional requerido, como ser la encriptación si es que manejamos.

```
Node.js  
  
const sequelize = new Sequelize('database', 'username', 'password', {  
  host: 'localhost',  
  dialect: 'mssql',  
  dialectOptions: { options: { encrypt: true } },  
  define: { timestamps: false }  
});
```



Configurar Sequelize en Node.js

También debemos crear una función para autenticar la conexión a la base de datos. Para ello utilizamos el método **sequelize.authenticate()**.

Si ocurre algún error durante la autenticación, se captura el mismo mediante el bloque catch.

Este proceso se debe ejecutar previo a realizar operaciones CRUD.

```
...  
Node.js  
  
async function authenticate() {  
  try {  
    await sequelize.authenticate();  
    console.log('Conexión a la base de datos establecida  
correctamente.');//  
  } catch (error) {  
    console.error('Error al conectar a la base de datos:', error);  
  }  
}
```

Configurar Sequelize en Node.js

También debemos crear una función para autenticar la conexión a la base de datos.

Para ello utilizamos el método **sequelize.authenticate()**.

Si ocurre algún error durante la autenticación, se captura el mismo mediante el bloque **catch**.

Este proceso se debe ejecutar previo a realizar operaciones CRUD.

```
... Node.js

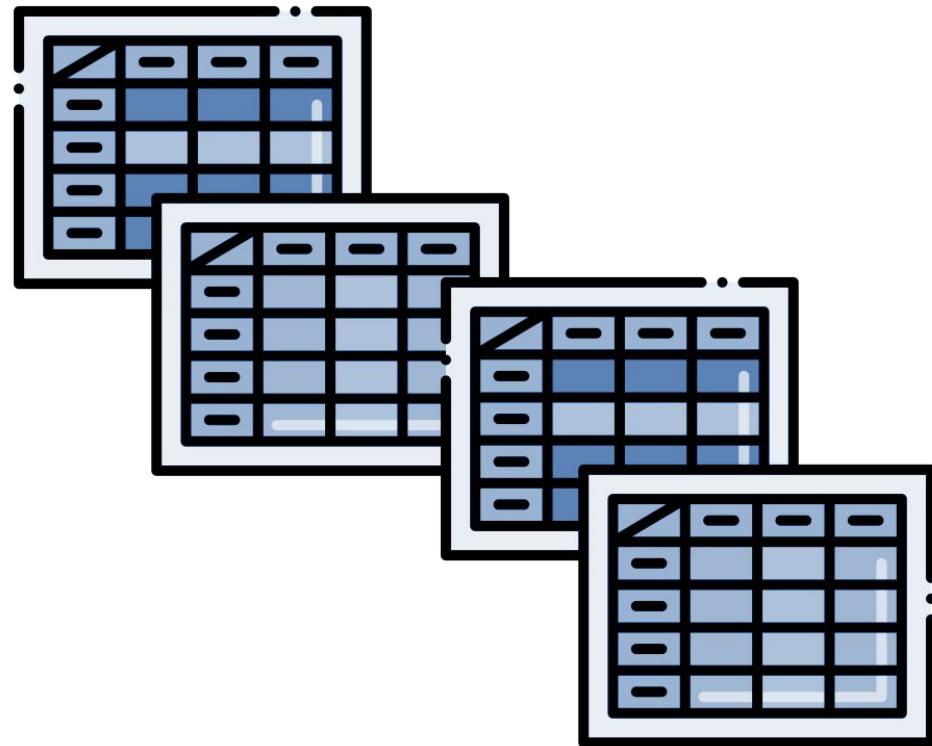
async function closeConnection() {
  try {
    await sequelize.close();
    console.log('Conexión cerrada correctamente.');
  } catch (error) {
    console.error('Error al cerrar la conexión:', error);
  }
}
```

Definir los modelos para nuestras tablas



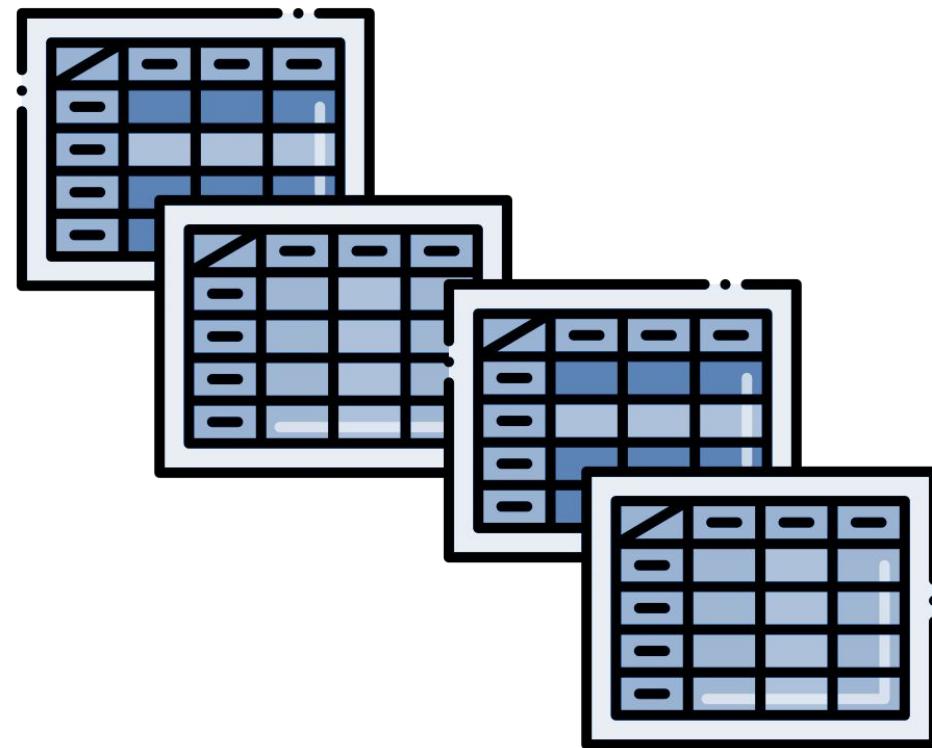
Definir los modelos para nuestras tablas

Una vez conectados a MySQL desde Node.js y Sequelize, para comenzar a interactuar con las tablas de la bb.dd, debemos definir un modelo de datos que nos permita generar esta interacción desde Node.js.



Definir los modelos para nuestras tablas

Este modelo de datos a generar será la estructura intermedia que nos permitirá trabajar con los registros de una Tabla SQL, desde Node.js, tal como si tuviésemos los registros de la tabla en un array de objetos JS.



Definir los modelos para nuestras tablas

Mediante el método **define()**, sequelize nos permite definir el nombre de la estructura de datos que nos conectará con, por ejemplo, una tabla SQL llamada **contactos**. Esta tiene un **ID**, un campo **nombreCompleto**, un campo **Email**, otro campo **Telefono**, y tal vez algunos otros más.

Además de definir el modelo de datos, indicamos cuál será el tipo de dato que almacenará cada uno, entre otros datos técnicos, propios de una bb.dd relacional.

```
Node.js

const Contactos = sequelize.define('Contactos', {
  id: {
    type: Sequelize.INTEGER,
    autoIncrement: true,
    primaryKey: true,
  },
  nombreCompleto: {
    type: Sequelize.STRING,
  },
  Email: {
    type: Sequelize.STRING,
    unique: false,
  },
  Telefono: {
    type: Sequelize.STRING,
    unique: false,
  },
})
```



Definir los modelos para nuestras tablas

Volvemos a MySQL Workbench y editamos la tabla **Products**, de nuestra base de datos de ejemplo llamada **Northwind**.

Aquí tenemos la estructura de datos de la tabla **Products** con sus columnas, los tipos de datos definidos, y algunas propiedades adicionales más.

Veamos entonces cómo podemos crear un modelo con Sequelize que sea aplicable a esta tabla.

Name:	Products									
Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
ProductID	INT	✓	✓	✓	■	■	■	■	✓	■
ProductName	VARCHAR(40)	□	✓	□	□	□	□	□	□	□
SupplierID	INT	□	□	□	□	□	□	□	□	NULL
CategoryID	INT	□	□	□	□	□	□	□	□	NULL
QuantityPerU...	VARCHAR(20)	□	□	□	□	□	□	□	□	NULL
UnitPrice	DOUBLE	□	□	□	□	□	□	□	□	'0'
UnitsInStock	SMALLINT	□	□	□	□	□	□	□	□	'0'
UnitsOnOrder	SMALLINT	□	□	□	□	□	□	□	□	'0'
ReorderLevel	SMALLINT	□	□	□	□	□	□	□	□	'0'
Discontinued	TINYINT(1)	□	□	✓	□	□	□	□	□	'0'
<click to edit>										



Definir los modelos para nuestras tablas

Aquí tenemos una representación de la tabla **Products** convertida a un modelo de datos de Sequelize. Cada campo del modelo se define como una propiedad en el objeto pasado a **define()**, donde la clave representa el nombre del campo y el valor representa el tipo de datos y las opciones adicionales.

Por ejemplo, **ProductID** se define como una **clave primaria** (**primaryKey: true**), **autoincrementable** (**autoIncrement: true**), y **no nula** (**allowNull: false**).



The screenshot shows a code editor window with a dark theme. The title bar says "Sequelize Model". The code defines a "Product" model with the following properties:

```
const Product = sequelize.define('Product', {
  ProductID: {
    type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true,
    allowNull: false
  },
  ProductName: {
    type: DataTypes.STRING(40), allowNull: false
  },
  SupplierID: {
    type: DataTypes.INTEGER, defaultValue: null
  },
  CategoryID: {
    type: DataTypes.INTEGER, defaultValue: null
  },
  QuantityPerUnit: {
    type: DataTypes.STRING(20), defaultValue: null
  },
  UnitPrice: {
    type: DataTypes.DOUBLE, defaultValue: 0
  },
  UnitsInStock: {
    type: DataTypes.SMALLINT, defaultValue: 0
  },
  UnitsOnOrder: {
    type: DataTypes.SMALLINT, defaultValue: 0
  },
  ReorderLevel: {
    type: DataTypes.SMALLINT, defaultValue: 0
  },
  Discontinued: {
    type: DataTypes.BOOLEAN, allowNull: false, defaultValue: false
  }
}, {
  tableName: 'Products',
  timestamps: false,
})

module.exports = Product
```

The `ProductID` field is highlighted with a purple rounded rectangle.



Definir los modelos para nuestras tablas

La opción **tableName** se establece en **Products** para indicar que el modelo está asociado a la tabla "*Products*" en la base de datos.

La opción **timestamps** se establece en **false** para desactivar la creación automática de las columnas **createdAt** y **updatedAt** en el modelo.

Por último, se exporta el modelo **Product** para que pueda ser utilizado en otras partes de la aplicación.



```
const Product = sequelize.define('Product', {
  ProductID: {
    type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true,
    allowNull: false
  },
  ProductName: {
    type: DataTypes.STRING(40), allowNull: false
  },
  SupplierID: {
    type: DataTypes.INTEGER, defaultValue: null
  },
  CategoryID: {
    type: DataTypes.INTEGER, defaultValue: null
  },
  QuantityPerUnit: {
    type: DataTypes.STRING(20), defaultValue: null
  },
  UnitPrice: {
    type: DataTypes.DOUBLE, defaultValue: 0
  },
  UnitsInStock: {
    type: DataTypes.SMALLINT, defaultValue: 0
  },
  UnitsOnOrder: {
    type: DataTypes.SMALLINT, defaultValue: 0
  },
  ReorderLevel: {
    type: DataTypes.SMALLINT, defaultValue: 0
  },
  Discontinued: {
    type: DataTypes.BOOLEAN, allowNull: false, defaultValue: false
  },
}, {
  tableName: 'Products',
  timestamps: false,
})
module.exports = Product
```

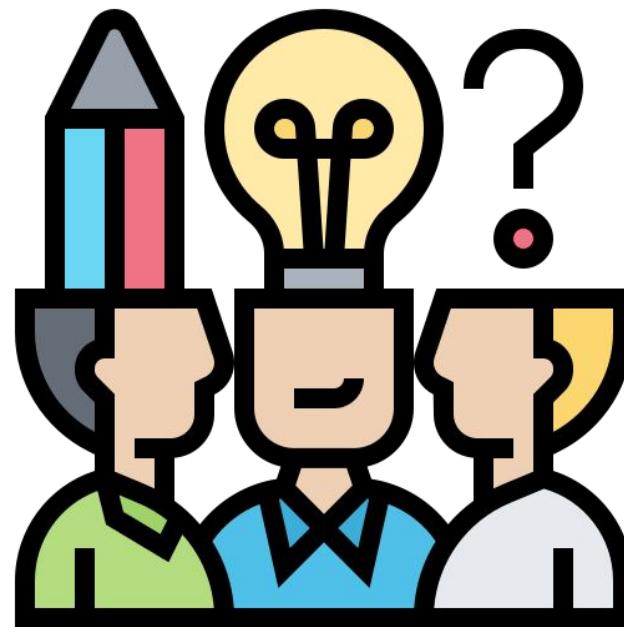


Los métodos más importantes

Los métodos más importantes

Sequelize ofrece varios métodos para realizar operaciones en MySQL desde Node.js.

Veamos a continuación, una tabla con la referencia a los métodos más comunes:



Los métodos más importantes

Métodos Sequelize	
Método	Descripción
create	Crea un nuevo registro en la tabla.
findAll	Busca todos los registros que cumplan con uno o más criterios
findOne	Busca un registro específico en la tabla que cumpla con ciertos criterios
update	Actualiza uno o varios registros en la tabla
destroy	Elimina uno o varios registros en la tabla

Aquí tenemos las operaciones básicas que podemos realizar con Sequelize utilizando Node.js y trabajando de forma conectada a MySQL.

Los métodos más importantes

Métodos Sequelize	
Método	Descripción
count	Cuenta el número de registros en una tabla.
sum	Calcula la suma de un campo específico en los registros de la tabla.
max	Obtiene el valor máximo de un campo específico en los registros de una tabla.
min	Obtiene el valor mínimo de un campo específico en los registros de una tabla.

En este caso, podemos ver que estamos frente a métodos que responden a las funciones de agregación que vimos oportunamente en MySQL.

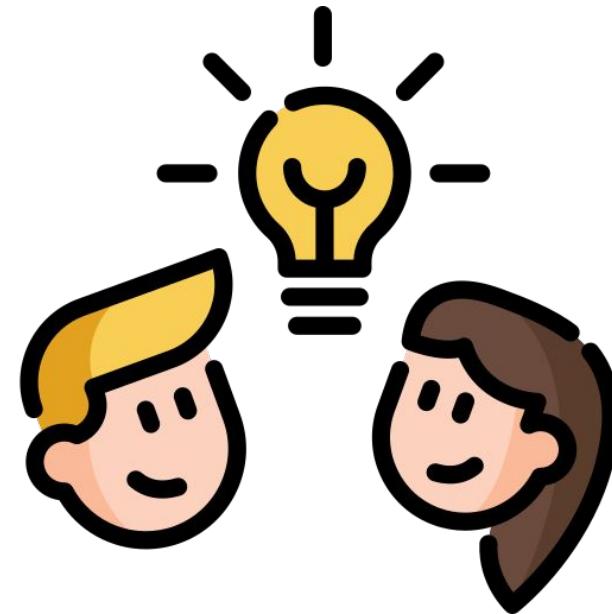


Los métodos más importantes

Estos son solo algunos de los métodos disponibles en Sequelize para realizar operaciones en MySQL.

Cada uno de ellos tiene su propia función y acepta diferentes parámetros para personalizar la consulta y los resultados obtenidos.

Te recomendaría consultar la documentación oficial de Sequelize para obtener más información sobre cada método y cómo utilizarlos de manera efectiva.



Los métodos más importantes

Aquí tenemos un ejemplo de cómo crear un nuevo registro en una hipotética tabla **Users**. Informamos el **nombre** y **Email** en un objeto, y enviamos dicho objeto como parámetro del método **create()**.

Luego, controlamos el proceso de alta con una promesa, y de igual manera cualquier error que surja.

```
Sequelize INSERT

User.create({ nombre: 'Donna Clark', email: 'donna.clark@mutiny.com' })
  .then((usuario) => {
    console.log('Usuario creado:', usuario);
  })
  .catch((error) => {
    console.error('Error al crear el usuario:', error);
  })
```

Los métodos más importantes

En este otro ejemplo de código, utilizamos el método **findOne()** para buscar un registro específico en la tabla **Users**, a partir de su nombre.

Notemos aquí que, el objeto que enviamos como parámetro a **findOne**, se encierra en una propiedad **where**.

```
Sequelize findOne

User.findOne({ where: { nombre: 'Cameron Howe' } })
  .then((usuario) => {
    if (usuario) {
      console.log('Usuario encontrado:', usuario);
    } else {
      console.log('Usuario no encontrado');
    }
  })
  .catch((error) => {
    console.error('Error al buscar el usuario:', error);
  })
```

Los métodos más importantes

En este otro ejemplo de código utilizamos el método **findAll()** para obtener todos los registros de la tabla **Users**.

Notemos aquí que el método no recibe ningún parámetro. Por ello, retornará un array con todos los registros encontrados.

```
Sequelize findAll

User.findAll()
  .then((usuarios) => {
    console.table(usuarios);
  })
  .catch((error) => {
    console.error('Error al buscar los usuarios:', error);
  })
```

Los métodos más importantes

En este otro ejemplo de código utilizamos el método **findAll()** y pasamos un objeto con una propiedad **where** la cual contiene la condición de búsqueda.

La condición es definida utilizando el operador **Op.like** para realizar una comparación con el filtro **LIKE** ...

```
Sequelize findAll

User.findAll({
  where: { email: { [Op.like]: '%@mutiny.com' } }
})
  .then((usuarios) => {
    console.table(usuarios)
  })
  .catch((error) => {
    console.error('Error al buscar los usuarios:', error)
  })
```

Los métodos más importantes

... la expresión **%@mutiny.com** especifica que estamos buscando registros en la columna **email** que contengan la cadena "**@mutiny.com**" en cualquier posición.

`findAll` devuelve un array de objetos que representan los usuarios encontrados que cumplan con la condición de búsqueda.

```
Sequelize findAll

User.findAll({
  where: { email: { [Op.like]: '%@mutiny.com' } }
})
  .then((usuarios) => {
    console.table(usuarios)
  })
  .catch((error) => {
    console.error('Error al buscar los usuarios:', error)
  })
}
```

Definir un CRUD con MySQL

Definir un CRUD con MySQL

Tomemos el modelo de ejemplo de la tabla **Northwind.Products**, y veamos cómo debemos realizar un CRUD sobre esta tabla, combinando **Express** y **Sequelize**.

En principio, tenemos un proyecto con toda la base necesaria de Express y la conexión a MySQL resuelta.

```
Sequelize INSERT

User.create({ nombre: 'Donna Clark', email: 'donna.clark@mutiny.com' })
  .then((usuario) => {
    console.log('Usuario creado:', usuario);
  })
  .catch((error) => {
    console.error('Error al crear el usuario:', error);
  })
```

Definir un CRUD con MySQL

Tanto el modelo de datos realizado para la tabla **Products**, como el resto de los modelos de datos, deben ser almacenados en una subcarpeta del proyecto dedicada llamada **/models**.

Luego, debemos importar el archivo del modelo a nuestra aplicación Node.js para poder utilizarlo.



Node.js

```
// Importar el modelo de Product
const Product = require('./models/Product');
```

CRUD



Definir un CRUD con MySQL

Create

Definimos el endpoint correspondiente junto al método **post()** el cual nos permite realizar las operaciones de creación de un nuevo recurso. Este método debe ser asincrónico, y deberá contener el manejo de la operación bajo la estructura **try - catch**, para que nos garantice poder controlar cualquier tipo de error que ocurra durante el proceso de alta.

```
Node.js

app.post('/products', async (req, res) => {
  try {

    } catch (error) {
      console.error('Error al crear el producto:', error);
      res.status(500).json({ error: 'Error al crear el producto' });
    }
})
```



Definir un CRUD con MySQL

Create

Dentro del bloque **try {}**, desestructuramos el parámetro recibido del cliente a través de **req.body**. Luego, invocamos el método **create()**, enviando a este un objeto completo con los valores de la desestructuración realizada por cada uno de los campos.

Por último, informamos la operación exitosa mediante el **código de estado 201**.

```
Node.js

const { ProductName, SupplierID, CategoryID,
        QuantityPerUnit, UnitPrice, UnitsInStock,
        UnitsOnOrder, ReorderLevel, Discontinued } = req.body;

const nuevoProducto = await Product.create({
    ProductName, SupplierID, CategoryID, QuantityPerUnit,
    UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel,
    Discontinued });

res.status(201).json(nuevoProducto);
```



Definir un CRUD con MySQL

Read

El método más simple de todos: **GET**.

En este ejemplo retornamos todos los registros de la tabla **Products**. Si ocurre algún error en el proceso, respondemos con un **código de estado 500**.

Al igual que el resto de los endpoints, este también debe ser asincrónico.

```
...  
Node.js  
  
try {  
  const productos = await Product.findAll();  
  res.json(productos);  
} catch (error) {  
  console.error('Error al consultar los productos:', error);  
  res.status(500).json({ error: 'Error al consultar los productos' });  
}
```

Definir un CRUD con MySQL

Read One

También podemos utilizar **GET** para ubicar un solo registro. Para ello, obtenemos el ID del producto desde **req.params.productId**. Utilizamos **Product.findByPk()** para buscar el producto por su **ID**, y si lo encuentra lo retornamos, sino, retornamos el error con el **código de estado 404**.

Al igual que el resto de los endpoints, este también debe ser asincrónico.

```
Node.js

try {
  const productId = req.params.productId;
  const producto = await Product.findByPk(productId);

  !producto ? res.status(404).json({ error: 'Producto no encontrado' })
            : res.json(producto);

} catch (error) {
  console.error('Error al buscar el producto:', error);
  res.status(500).json({ error: 'Error al buscar el producto' });
}
```

Definir un CRUD con MySQL

Update

En este ejemplo modificamos un registro existente. Para ello, obtenemos los datos actualizados desde **req.body**. Usamos el método **Product.findByPk()**, para buscar el producto por su **ID**.

Si no existe, enviamos un **error 404** como respuesta. Si existe, usamos el método **Product.update()** junto a **updatedData** para que actualice el registro.

```
Node.js

try {
    const productId = req.params.productId;
    const updatedData = req.body;
    const producto = await Product.findByPK(productId);
    !producto ? res.status(404).json({ error: 'Producto no encontrado' })
        : await producto.update(updatedData);

    res.json(producto);
} catch (error) {
    console.error('Error al actualizar el producto:', error);
    res.status(500).json({ error: 'Error al actualizar el producto' });
}
```

Ten presente que en este ejemplo, el endpoint también debe ser asincrónico y utilizar el método **PUT**.



Definir un CRUD con MySQL

Delete

En este ejemplo eliminamos un registro existente obteniendo los datos del mismo desde **req.params.productId**. Usamos el método **Product.findByPk()**, para buscar el producto por su **ID**.

Si no existe, enviamos un **error 404** como respuesta. Si existe, eliminamos el mismo mediante el método **Product.destroy()**.

```
...  
Node.js  
  
try {  
  const productId = req.params.productId;  
  const producto = await Product.findByPk(productId);  
  
  !producto ? res.status(404).json({ error: 'Producto no encontrado' }) ;  
    : await producto.destroy();  
  
  res.json({ message: 'Producto eliminado correctamente' });  
} catch (error) {  
  console.error('Error al eliminar el producto:', error);  
  res.status(500).json({ error: 'Error al eliminar el producto' });  
}
```

Ten presente que en este ejemplo, el endpoint también debe ser asincrónico y utilizar el método **DELETE**.



Crear los endpoint

Crear los endpoint

Para este modelo de ejemplo debemos crear los siguientes endpoint funcionales:

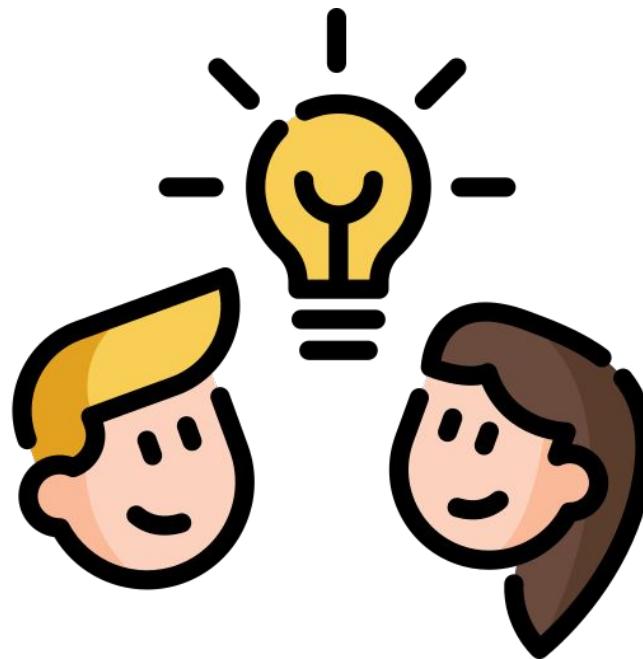
Endpoint	Método	Descripción
/products	GET	Obtienes el listado total de los registros, utilizando modelo.findAll() .
/products:id	GET	Obtienes un registro específico, utilizando modelo.findAll(id) .
/products	POST	Creas un nuevo registro en la tabla, utilizando modelo.create(param1, param2, etc) .
/products	PUT	Creas un nuevo registro en la tabla, utilizando modelo.update(updatedData) .
/products/:id	DELETE	Eliminas un registro de la tabla, utilizando modelo.destroy() .



Crear los endpoint

Ya lo analizamos en los ejemplos de código, pero igual lo reforzamos:

Todos los métodos deberán ser asincrónicos, y las operaciones (CRUD) a realizar, deben estar controladas por un bloque try - catch.



Pre-Entrega 3

Pre-Entrega 3

Aprovecha el contenido visto en la clase de hoy, para seguir diseñando otra parte del tramo del trabajo final. En esta oportunidad, define la estructura base del proyecto creando el código base de Node.js, con Express.

Integra la librería Sequelize, genera la conexión, y crea los modelos de datos para las tablas del proyecto Traileflix.



Pre-entrega 3

Al crear la conexión desde Node.js a MySQL,
recuerda **definir el host, el usuario** y la
contraseña, dentro de un archivo **.env**.

Prueba la conexión a MySQL, enviando un
mensaje a la consola (Terminal) de Node.js.

Crea la carpeta **/models**, donde almacenarás
los modelos de las tablas SQL que hayan
surgido de nuestro encuentro anterior.



Pre-entrega 3

Define el modelo que tendrán todos los endpoints de tu proyecto, a través del código JS.

Ten presente que, el endpoint principal de trailerflix, deberá retornar el set de datos tal como tenemos en el modelo de archivo JSON.

Para resolver esto, debes crear una vista SQL respetando la estructura del archivo JSON.



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



Argentina programa 4.0



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*

Clase 29: Bases de datos Relacionales

Backup y Restauración desde Node.js

Agenda de hoy

- A. Administración de bb.dd. - Backup Y Restore
- B. Backup de bases de datos
 - a. backup mediante Dump Project Folder
 - b. backup mediante Self contained File
- C. Restauración de base de datos
 - a. Mecanismos de restauración
 - b. Dump Project Folder
 - c. Self Contained File
- D. Prácticas



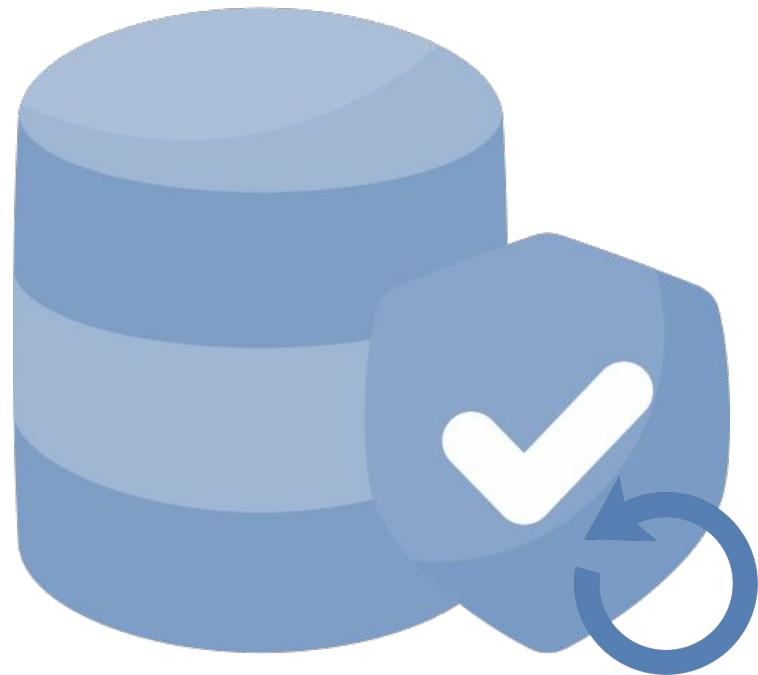
Administración de bases de datos



Administración de bases de datos

Ya hemos hablado anteriormente de la importancia de los datos hoy en día. En principio, la información se almacenaba solo por obligaciones impositivas de mantener un histórico de operaciones.

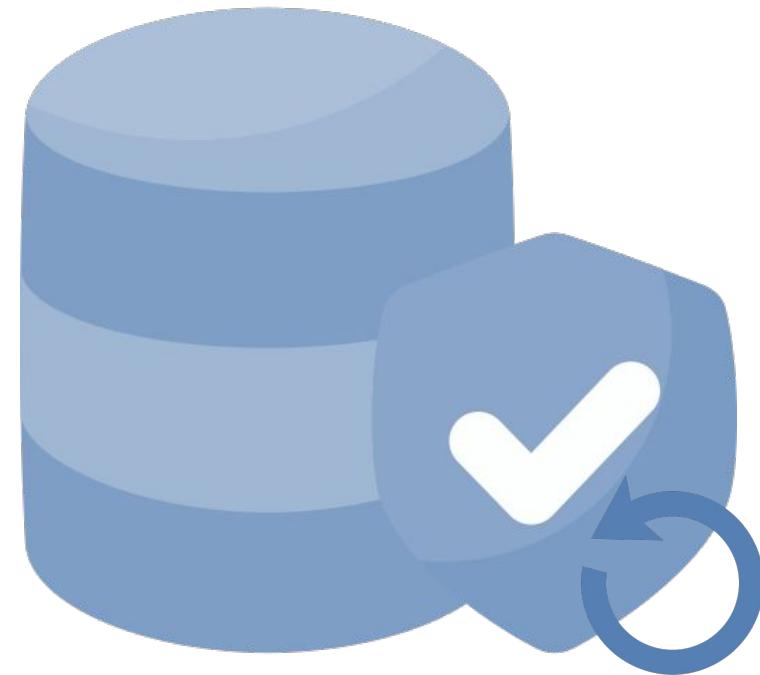
Desde hace algunos años, se descubrió que los datos “*pueden hablar*” y/o representar la información como métricas. Y estas métricas pueden decirnos, anticiparnos, o ayudarnos a tomar decisiones de una forma más asertiva.



Administración de bases de datos

Y más allá de que nuestro rol principal se enfoque en el desarrollo de software, en nuestro día a día vamos a tener la necesidad de interactuar con datos todo el tiempo.

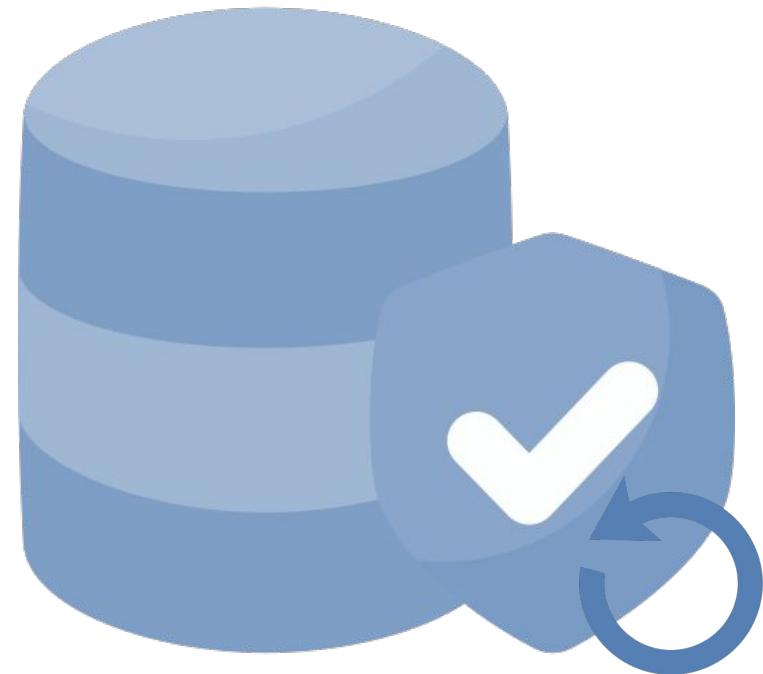
Y como parte de esta tarea, tener en cuenta la forma más apropiada de realizar backups y restauración de la información, es un punto que no podemos pasar por alto.



Administración de bases de datos

Además, la importancia de resguardar los datos también aplica porque toda información digital es uno de los activos más valiosos.

Por ello, hablaremos en este encuentro sobre la importancia y fundamentos de backup y restauración de bases de datos, y así garantizar la seguridad y disponibilidad continua de la información generada.

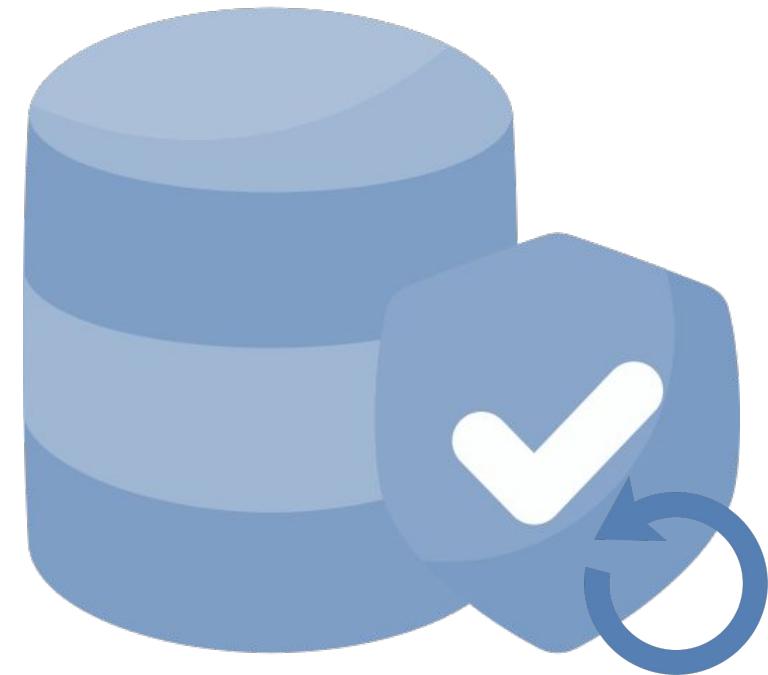


Backup y Restauración

Administración de bases de datos

La importancia del **backup y restauración** radica en su capacidad para **prevenir la pérdida de datos** debido a **errores humanos, fallas de hardware, ataques ciberneticos o desastres naturales.**

Contar con una estrategia sólida de backup y restauración nos **ayuda a minimizar el impacto de estos imprevistos.**

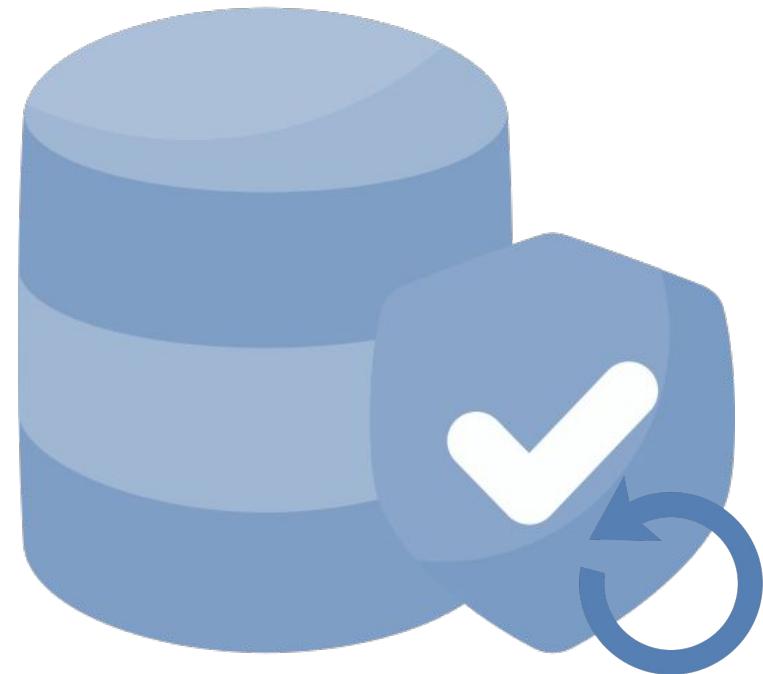


Administración de bases de datos

Dentro de las ventajas destacables en el terreno de backup y restauración de la información, podemos destacar:

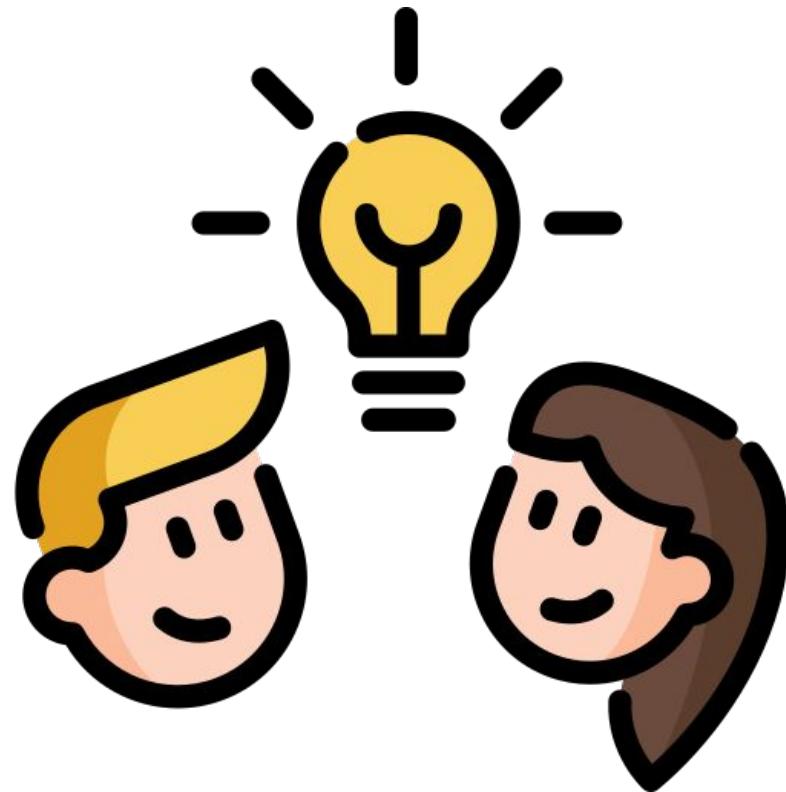
- Disponibilidad y continuidad del negocio
- Facilitar recuperación rápida y eficiente
- Brindar tranquilidad con datos protegidos

Entre otros tantos beneficios.



Administración de bases de datos

Además, en la actualidad, las nubes de datos ayudan mucho en este terreno, facilitando copias continuas de la información generada en estos datacenters, y almacenando la misma de forma espejada (*varías copias en diferentes lugares*).

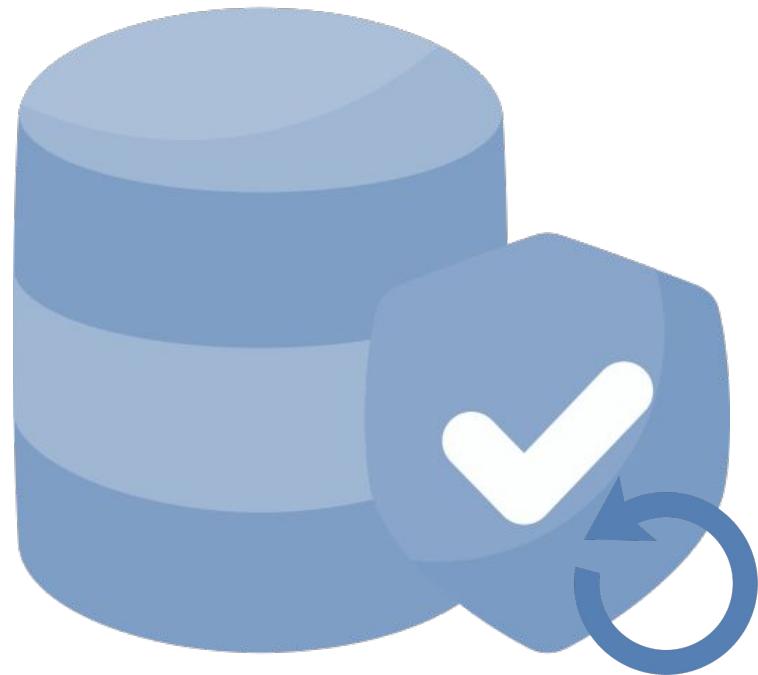


Análisis previo

Administración de bases de datos

Para realizar una copia de seguridad de la información que utilizamos de manera frecuente, lo primero que debemos considerar son los siguientes factores:

- La frecuencia y programación de los backups
- El resguardo físico de las copias de seguridad
- Verificar regularmente la integridad de los backups
- Documentar claramente los procedimientos de copia de seguridad y restauración



Administración de bases de datos

Resumiendo lo visto hasta aquí, podemos decir que el backup y restauración de bases de datos MySQL es una práctica esencial para garantizar la protección y disponibilidad de nuestros valiosos datos. Contar con un plan sólido y seguir buenas prácticas, nos ayuda a minimizar riesgos y asegurar la continuidad de nuestras operaciones.

Exploraremos a continuación las estrategias de backup y restauración, herramientas, y mejores prácticas recomendadas para lograr una gestión efectiva de las bases de datos MySQL.



Migrar un modelo nosql hacia un modelo relacional

Data Manipulation Language

Pero cuando se trata de datos en cantidades notables donde estos crecerán de forma continua todo el tiempo, y que mucha de esta información se repetirá constantemente, allí el modelo de datos SQL es el modelo de información más asertivo para administrar todo este cúmulo de información de la manera más apropiada.



Data Manipulation Language

Y si nos posicionamos en algún ejemplo de datos específico, donde el volumen de la información seguramente se repita de forma frecuente, allí podremos entender que, para estos tipos de datos repetitivos, el modelo relacional será el que mejor se adapte para minimizar el crecimiento de esta información a futuro.

```
... JSON

{
  "id": 17,
  "poster": "./posters/17.jpg",
  "titulo": "Halt and Catch Fire",
  "categoria": "Serie",
  "genero": "Drama",
  "tags": "Ficción, Drama, Tecnología",
  "busqueda": "Halt and Catch Fire, Ficción, Drama, Tecnología, Lee Pace, Scoot McNairy, Mackenzie Davis, Kerry Bishé, Toby Huss, Alana Cavanaugh",
  "temporadas": "4",
  "reparto": "Lee Pace, Scoot McNairy, Mackenzie Davis, Kerry Bishé, Toby Huss, Alana Cavanaugh",
  "trailer": "https://www.youtube.com/embed/pWrioRji60A"
}
```



Migrar un modelo nosql hacia un modelo relacional

Como vimos con el ejemplo anterior, migrar un modelo de datos NoSQL hacia un modelo de datos relacional (SQL) puede ofrecer varias ventajas significativas.

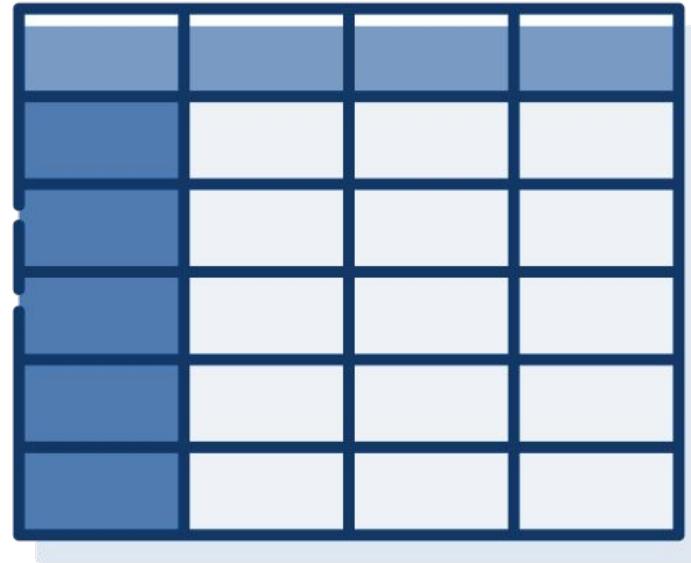
En un modelo relacional, sabemos que los datos se organizarán en tablas estructuradas con relaciones establecidas entre ellas, lo que permite un mejor control y coherencia de los datos.



Migrar un modelo nosql hacia un modelo relacional

Este mejor control y con una coherencia en los datos almacenados, nos facilita poder realizar consultas complejas y análisis más profundos (*métricas*).

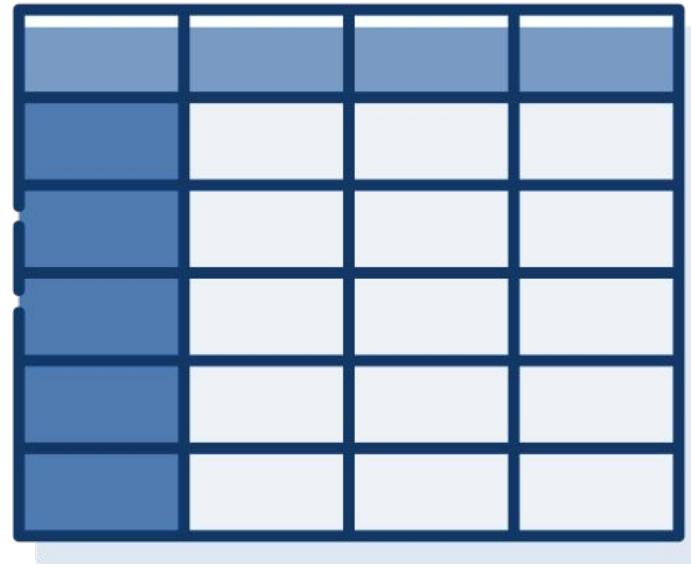
Además, los sistemas de gestión de bases de datos relacionales (RDBMS) ofrecen características como transacciones ACID, que aseguran la integridad y consistencia de los datos.



Migrar un modelo nosql hacia un modelo relacional

Esto es especialmente útil en aplicaciones empresariales donde la confiabilidad y la precisión son fundamentales. Otro beneficio clave es la capacidad de realizar consultas ad hoc y utilizar el lenguaje SQL estándar, lo que facilita el acceso y la manipulación de los datos.

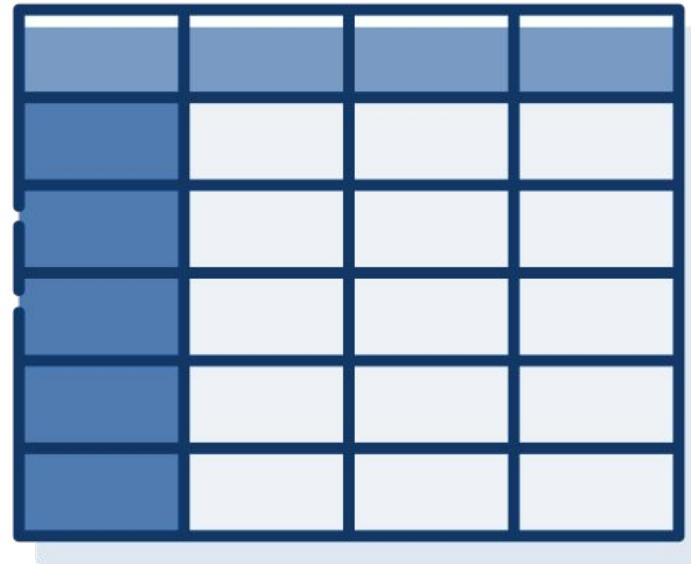
Además, los RDBMS tienen una amplia gama de herramientas y soporte, lo que simplifica el mantenimiento y la escalabilidad del sistema.



Migrar un modelo nosql hacia un modelo relacional

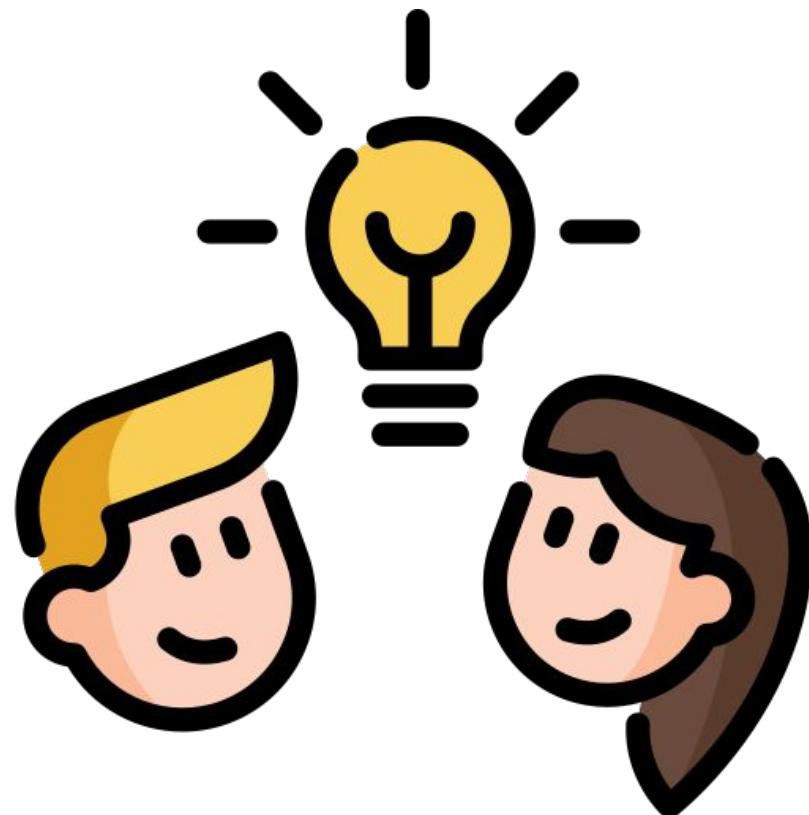
Esto es especialmente útil en aplicaciones empresariales donde la confiabilidad y la precisión son fundamentales. Otro beneficio clave es la capacidad de realizar consultas ad hoc y utilizar el lenguaje SQL estándar, lo que facilita el acceso y la manipulación de los datos.

Además, los RDBMS tienen una amplia gama de herramientas y soporte, lo que simplifica el mantenimiento y la escalabilidad del sistema.



Migrar un modelo nosql hacia un modelo relacional

En resumen, migrar a un modelo de datos relacional ofrece mayor control, consistencia, confiabilidad y facilidad de acceso a los datos, lo que puede ser beneficioso para aplicaciones empresariales y análisis de datos más complejos.



Backup de bases de datos

Backup de bases de datos

Al inicio del trabajo con bb.dd hablamos oportunamente de que MySQL cuenta con diversas **Herramientas de Consola**, o ventana **Terminal**, que nos permiten realizar diferentes tareas; entre ellas, administrar políticas de backup.

```
$/> mysqldump
```

mysqldump es el nombre de esta herramienta, y forma parte del motor de base de datos para gestionar copias de seguridad.



Backup de bases de datos

Debemos definir el comando **mysqldump**, seguido de los parámetros **-u (usuario)** **-p (contraseña)** y seguido a ello el nombre de la bb.dd o esquema que deseamos respaldar.

Por último, definimos la ruta de destino donde realizaremos el backup en cuestión.

```
...  
Mysql Dump  
mysqldump -u [usuario] -p [contraseña] [nombre_de_la_base_de_datos] > [ruta_del_archivo_de_backup.sql]
```



Backup de bases de datos

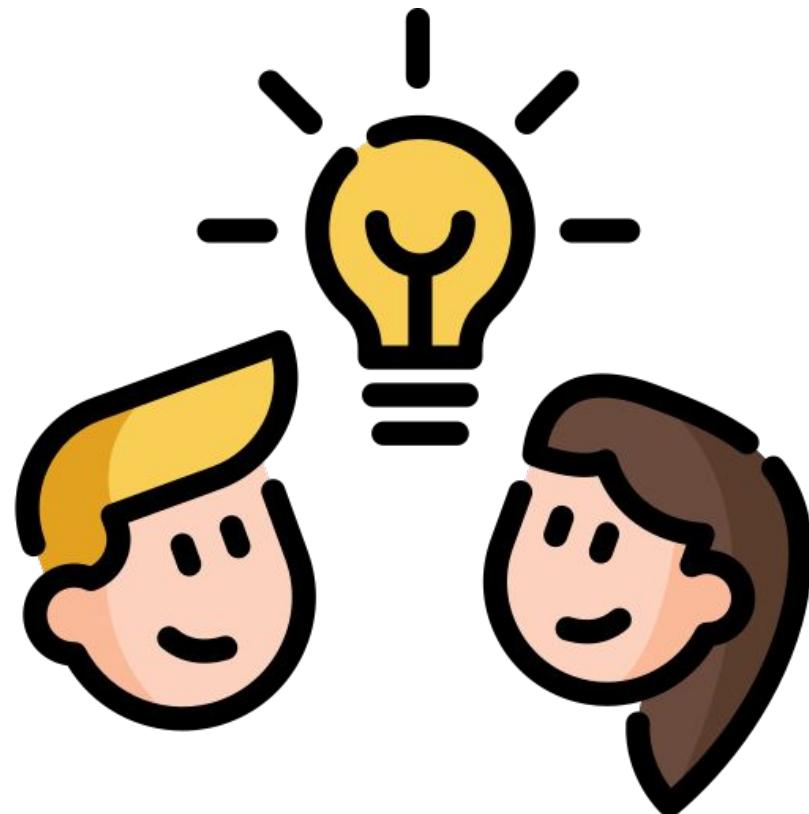
Todo backup que realicemos de una bb.dd siempre se realiza sobre un archivo con extensión **.sql**.

El proceso habitual de backup es básicamente generar la estructura de la bb.dd con sus relaciones y valores por defecto y, a continuación, la referencia de inserción de registros de acuerdo al total de datos almacenados.

```
...  
Mysql Dump  
mysqldump -u root -p mydatabase > /ruta/al/archivo/backup.sql
```

Backup de bases de datos

Si bien, hoy tenemos varios mecanismos modernos y gráficos para realizar estos procedimientos, no siempre tendremos entornos de bases de datos que nos permitan acceder a su información a través de aplicaciones RDBMS.



Backup y MySQL Workbench

Si tenemos acceso a las bases de datos que debemos resguardar utilizando MySQL Workbench, este gestor nos brinda herramientas efectivas para poder llevar a cabo estos procesos más cómodamente.



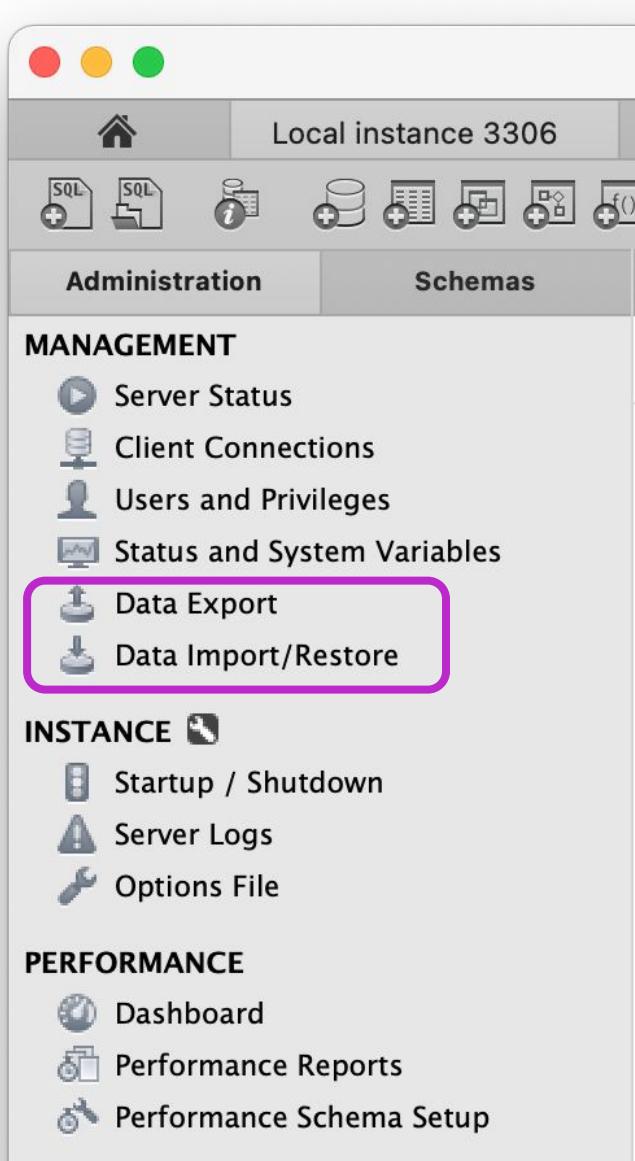
Backup desde MySQL Workbench



Backup y MySQL Workbench

En el apartado **Administration**, encontraremos un set de herramientas para acceder a los procesos de backup y restauración.

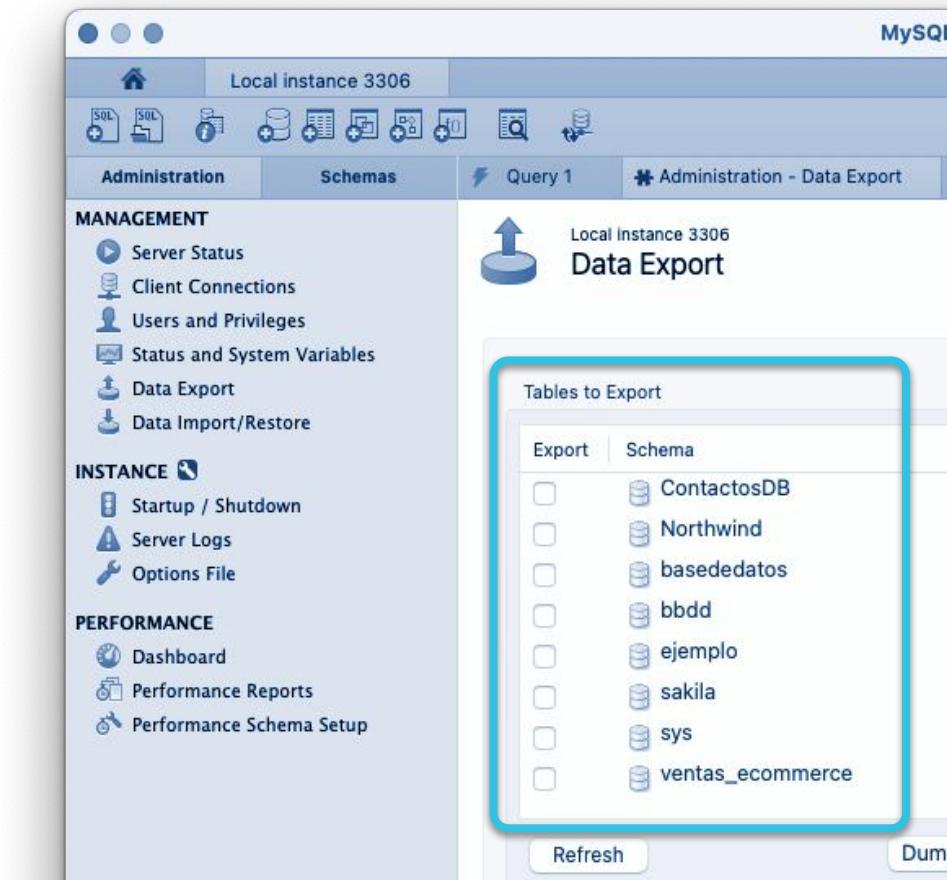
Los mismos se encuentran en las secciones **Data Export** y **Data Import/Restore**.



Backup y MySQL Workbench

Cuando ingresamos a la sección **Data Export**, la misma despliega un panel donde podremos ver todas las bb.dd (*esquemas*) disponibles para realizar copias de seguridad.

El proceso en sí puede aplicarse solo a una bb.dd o a varias, en simultáneo.



Backup y MySQL Workbench

The screenshot shows the 'Tables to Export' interface in MySQL Workbench. On the left, under 'Tables to Export', there are two tabs: 'Export' and 'Schema'. The 'Schema' tab is selected, showing a list of databases: ContactosDB, Northwind, basededatos, bbdd, ejemplo, sakila, sys, and ventas_ecommerce. The database 'sakila' is selected, indicated by a checked checkbox and highlighted with a blue background. On the right, under 'Export' and 'Schema Objects', there is a list of schema objects: actor, actor_info, address, category, city, country, customer, customer_list, and film. All objects are selected, indicated by checked checkboxes.

Al seleccionar una base de datos, podremos visualizar en el panel derecho el listado de **Tablas** y **Vistas** de la misma. A su vez, podemos seleccionar todas las tablas y/o vistas que consideremos necesario incluir en el backup, tildando y/o destildando cada casillero lateral.

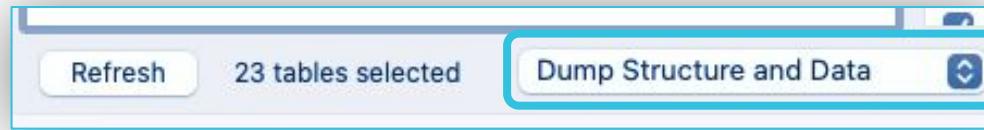


Backup y MySQL Workbench



Los botones inferiores a los paneles nos permiten seleccionar las **vistas**, **tablas**, **deseleccionar todo**, **refrescar** su contenido (*por si algo cambió por afuera*), y seleccionar qué información vamos a respaldar.

Backup y MySQL Workbench



Tenemos varias formas de definir el tipo de copia de respaldo que deseamos hacer.

Dump Structure and Data: realiza una copia de seguridad de los datos, y de la estructura de la base de datos.

Dump Data Only: realiza una copia de seguridad de los datos de la base de datos.

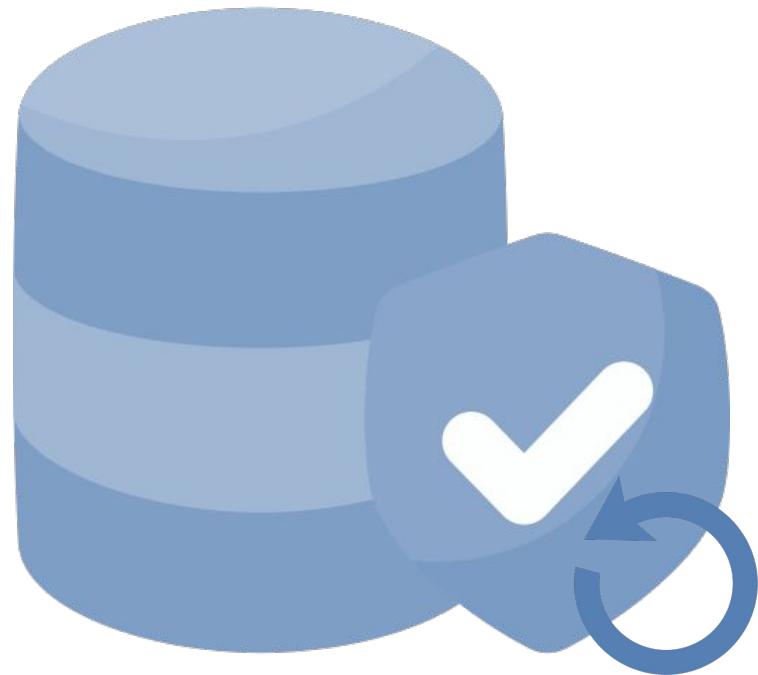
Dump Structure Only: realiza una copia de seguridad de la estructura de la base de datos.

¿Cuándo utilizar las diferentes opciones de backup?

Dump Data Only

Si la base de datos es de uso constante y su mantenimiento es esporádico, podemos elegir esta opción la cual resguarda solo los datos almacenados en ésta.

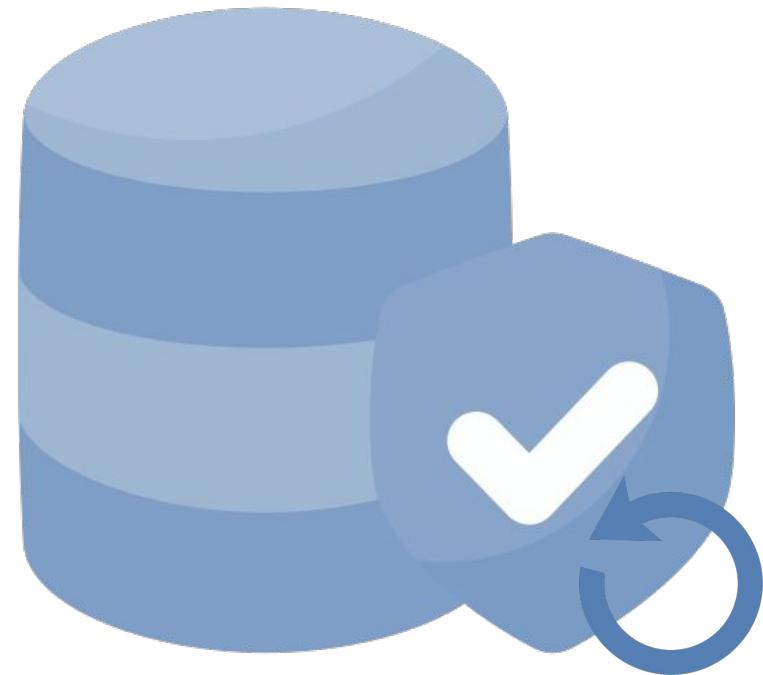
Si es una base de datos de mucha concurrencia, la opción más apropiada de backup de datos será todos los días.



Dump Structure Only

Este opción se utiliza cuando implementamos nuevas **tablas, vistas, funciones y procedimientos almacenados**, o cuando cambiamos la estructura de algunos de los objetos de la bb.dd.

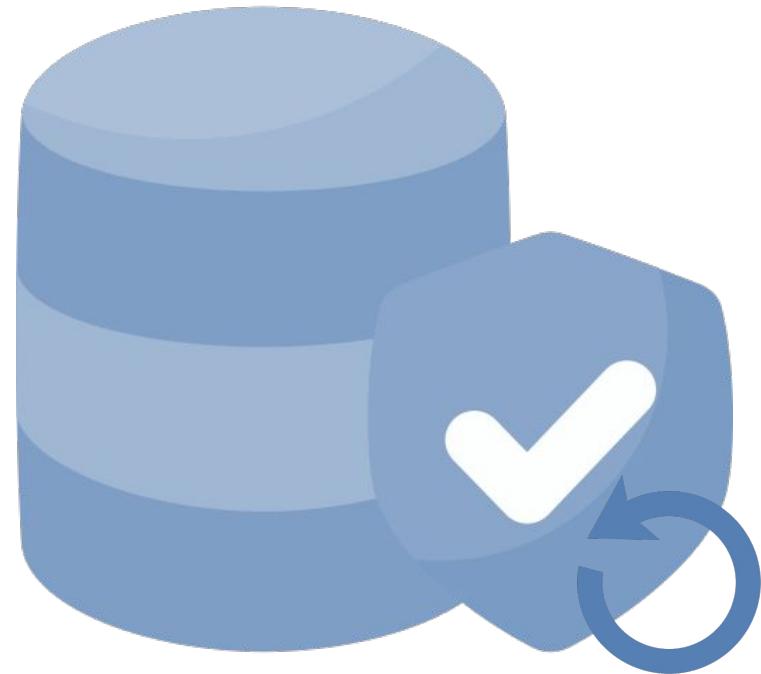
Su uso es esporádico; solo cuando ocurre algún evento de los aquí mencionados.



Dump Data And Structure

Si la base de datos está en un ambiente que no es el de **Producción**, o una base de datos personal donde realizamos poca interacción con la información y/o modificación de su estructura, podemos pensar en utilizar este mecanismo de backup.

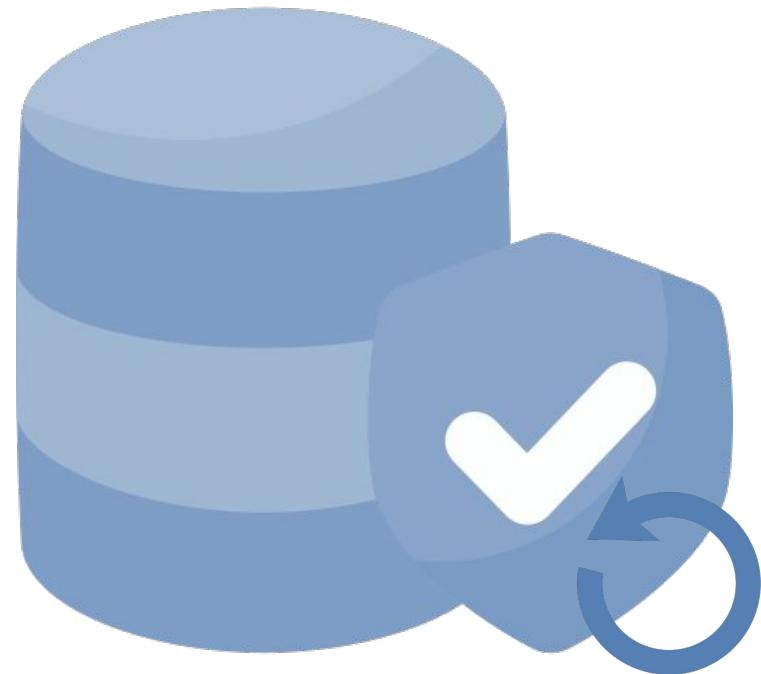
Su uso también puede ser esporádico, ya que es el más lento de todos, porque resguarda toda la información.



Analizar un contenido a migrar

Si la base de datos es de uso constante y su mantenimiento es esporádico, podemos elegir esta opción la cual resguarda solo los datos almacenados en ésta.

Si es una base de datos de mucha concurrencia, la opción más apropiada de backup de datos será todos los días.



Seleccionar los Objetos a exportar

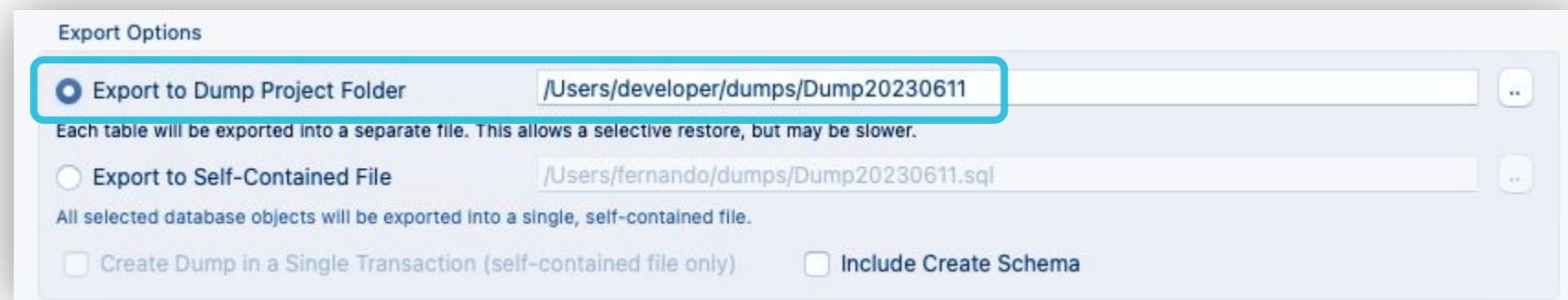


Seleccionar los Objetos a exportar



Los botones inferiores a los paneles nos permiten seleccionar las **vistas, tablas, deseleccionar todo, refrescar** su contenido (*por si algo cambió por afuera*), y seleccionar qué información vamos a respaldar.

Backup y MySQL Workbench



Finalmente queda elegir el tipo de backup a realizar. Entre las opciones tenemos:

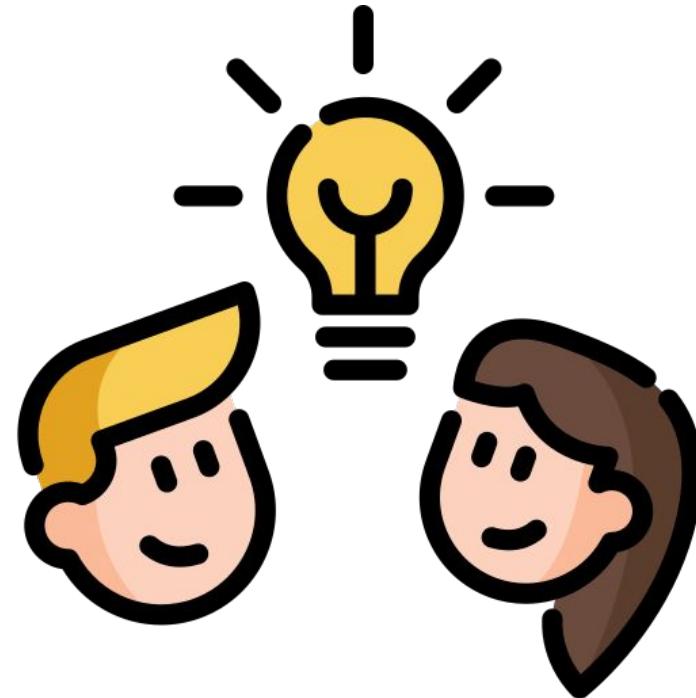
- **Export to Dump Project Folder**, vuelca a una carpeta de proyecto la(s) base(s) de datos
- **Export to Self-Contained File**, genera un archivo **.sql** con la información seleccionada

Esta opción permite seleccionar la ruta hacia la carpeta donde queremos guardar el backup. Cada objeto Tabla se exportará en un archivo separado. Esto nos garantiza que, de tener que restaurar, podremos seleccionar cuál(es) tabla(s) recuperar.

Backup y MySQL Workbench

Si la base de datos se encuentra en un servidor, recomendamos realizar la copia de datos local y luego, dicha carpeta, sea trasladada a un disco de red, disco externo, Cloud Storage, o cualquier otro medio de almacenamiento seguro.

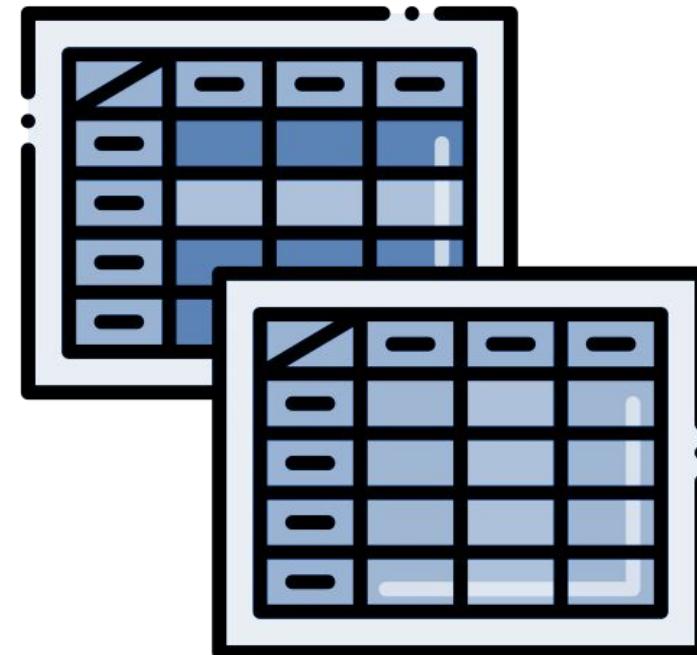
El backup local permitirá reducir los tiempos de este proceso.



Backup y MySQL Workbench

Y, cada carpeta donde realicemos un nuevo backup, debemos nombrarla con la fecha del día, y hasta incluir la hora de realización del backup.

Esto facilitará ubicar rápidamente un backup antiguo en un disco que almacena el historial de backups.



El proceso de Backup

El proceso de Backup

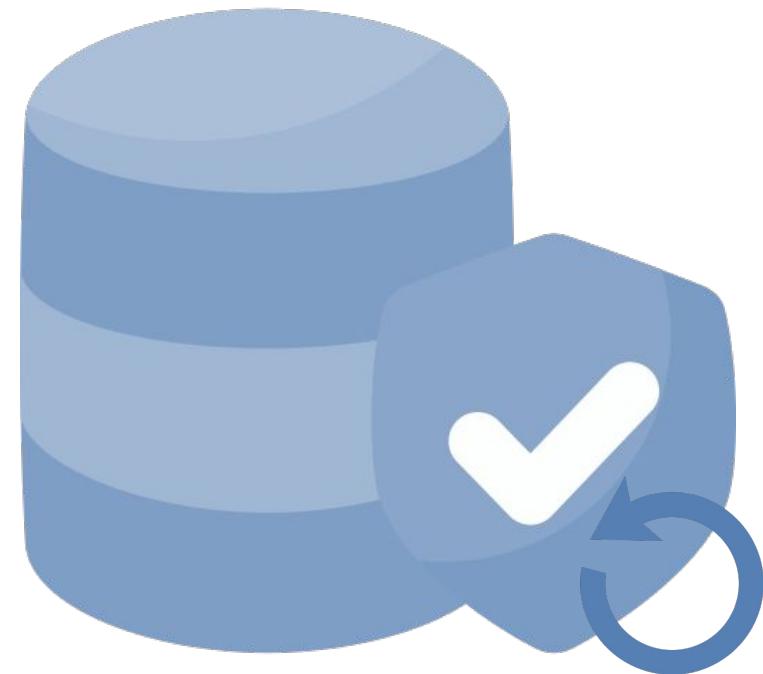


Cuando pulsamos el botón **Iniciar Proceso**, este comenzará a realizar el backup de la o las bases de datos y objetos seleccionados. Mediante la Barra de progreso y la ventana de LOG, podemos seguir el avance. Su tiempo de demora será acorde al tamaño de la información seleccionada para respaldar.

El proceso de Backup

```
21:55:52 Export of /Users/developer/dumps/Dump20230611.sql has finished
```

Dentro del LOG encontraremos un mensaje similar a este, cuando el backup ha llegado a su fin. Solo nos queda consultar la carpeta seleccionada para verificar que el o los archivo(s) haya(n) sido creado(s).



Restauración de base de datos

Restauración de base de datos

El proceso de restauración de bases de datos de la mano de MySQL Workbench es similar al que realizamos recientemente.

Las interfaces gráficas nos ayudan a trabajar más rápido y mejor en cuanto a selección de los recursos a restaurar. Veamos entonces, qué nos ofrece MySQL Workbench a través de la sección **Data Import/Restore**.

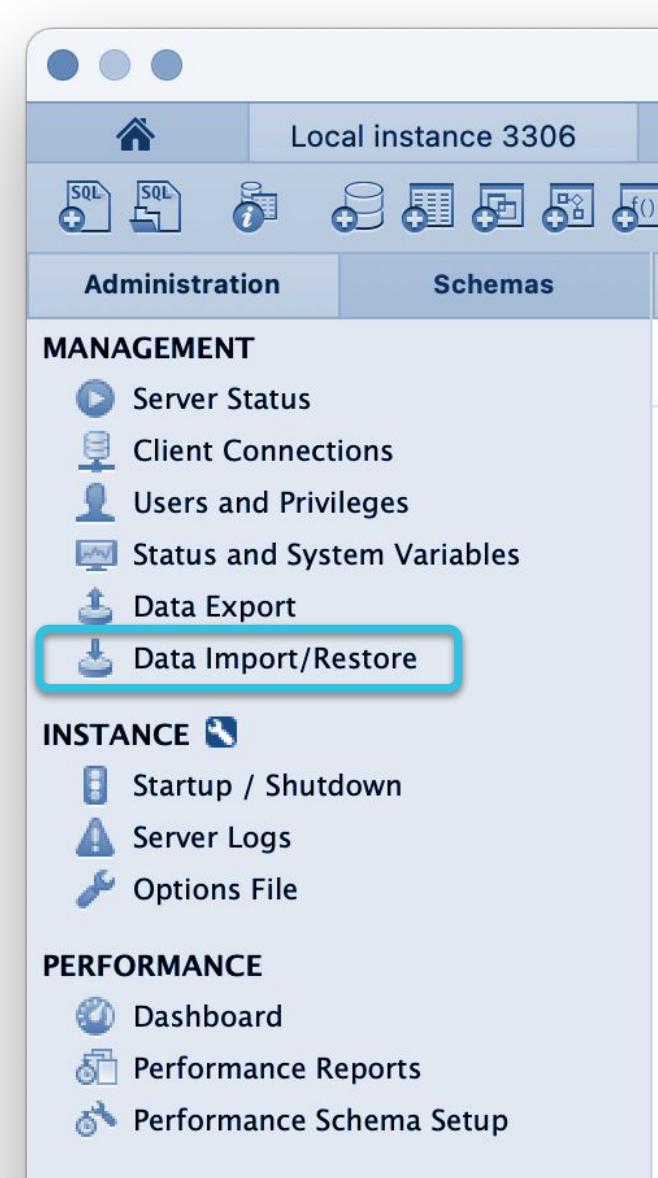


Restauración de base de datos

Ingresamos al apartado **Administration > Data**

Import/Restore, donde encontraremos un panel similar a **Data Export**.

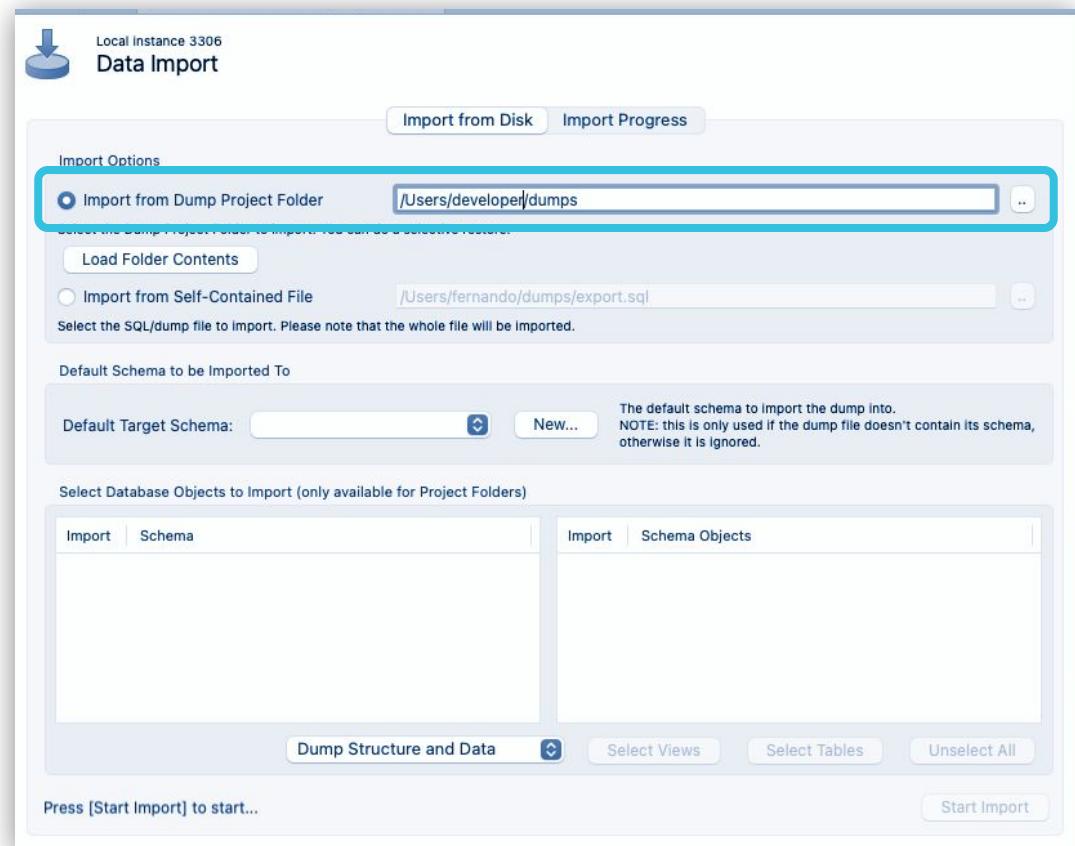
El proceso que debemos realizar a continuación, es justamente el inverso al realizado anteriormente, de cara a poder recuperar los datos u objetos de forma total o parcial en alguna base de datos existente.



Restauración de base de datos

El panel de restauración tiene una interfaz similar al anterior. Aquí se concentran dos puntos específicos, desde donde debemos seleccionar qué tipo de backup deseamos recuperar.

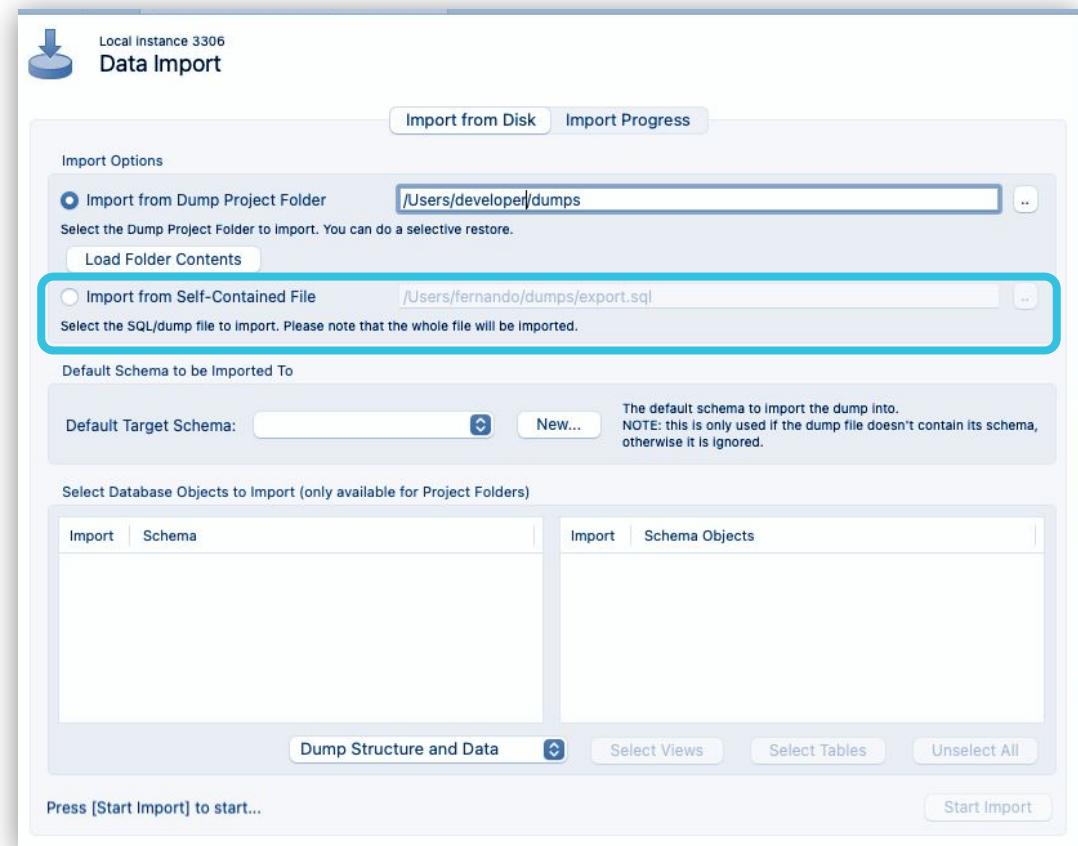
Si realizamos un backup a través del método **Dump Project Folder**, debemos tildar entonces la misma opción y elegir la carpeta de origen desde donde restauraremos la copia de seguridad.



Restauración de base de datos

En cambio, si buscamos recuperar un backup desde el método **Self-Contained File**, debemos seleccionar esta otra opción y elegir a continuación el archivo **.sql** que contiene el backup de la base de datos.

En ambos casos, disponemos de los botones laterales que nos permiten navegar dentro del sistema de archivos de la unidad donde estos se encuentren alojados.



Restauración de base de datos

El paso siguiente será validar de todas las posibles bases de datos elegidas anteriormente para formar parte del backup, cuál de ellas necesitamos para restaurar un backup específico.

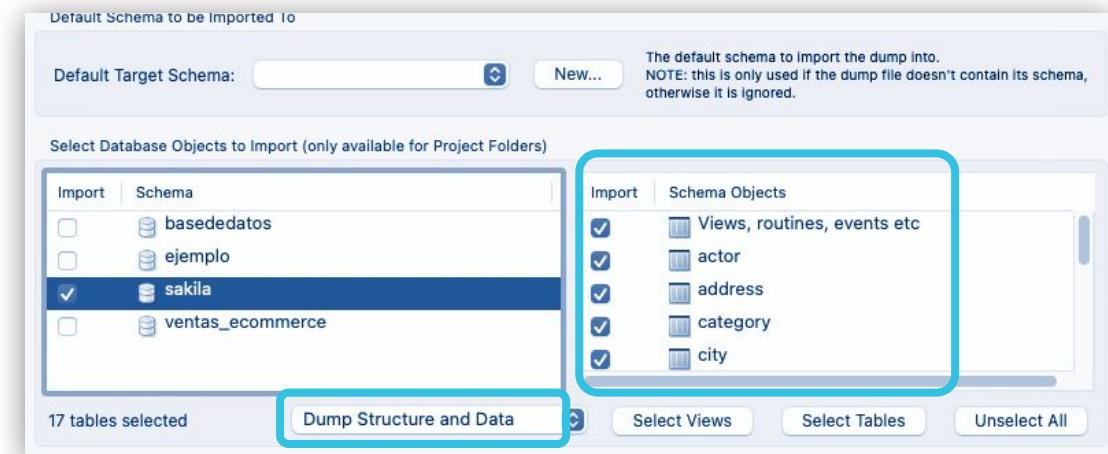
Podemos dejar a todas las opciones tildadas o solo seleccionar aquella o aquellas base(s) de datos desde donde deseamos recuperar la información.



Restauración de base de datos

Al elegir una base de datos, veremos todos los objetos de esta y podremos seleccionar de forma parcial o total cuáles de ellos queremos restaurar.

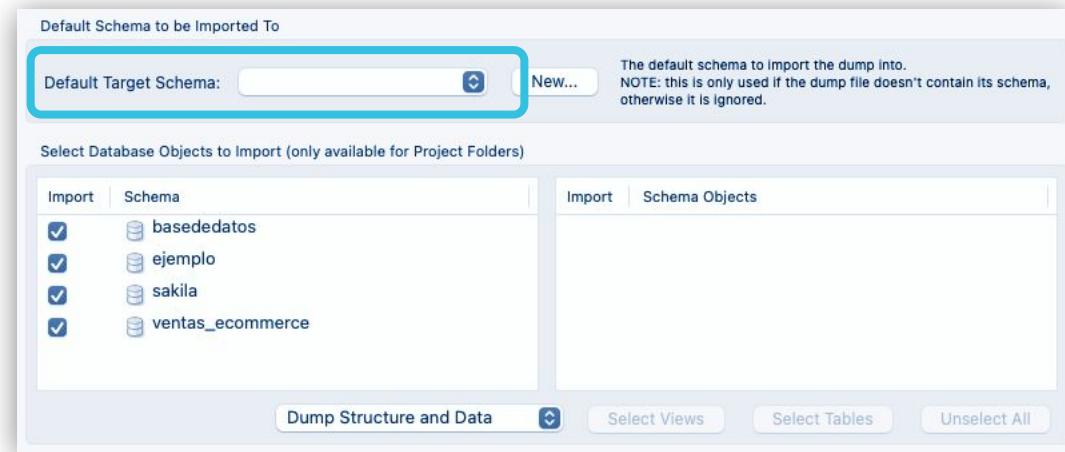
Debajo también podemos elegir cuál es el proceso de restauración (*solo su estructura, solo sus datos, o estructura y datos*).



Restauración de base de datos

Si vamos a trabajar con un solo esquema, debemos elegir el esquema de destino desde el apartado superior, para saber dónde se restaurará la información en cuestión.

Más allá de que debamos restaurar información de más de una bb.dd, recomendamos siempre realizar el proceso de a una base de datos por vez, para así tener un mejor control de los resultados de restauración de datos.



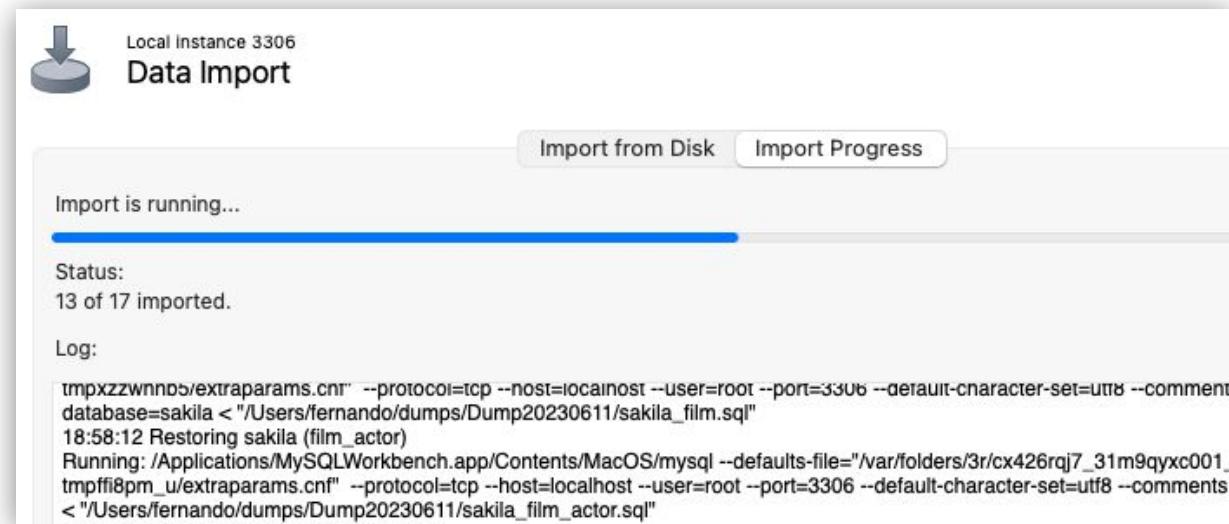
Iniciar Restauración



Iniciar Restauración

Finalmente, ajustados todos los parámetros y seleccionados todos los objetos involucrados, pulsamos el botón **Start Import** para que dé inicio al proceso de restauración.

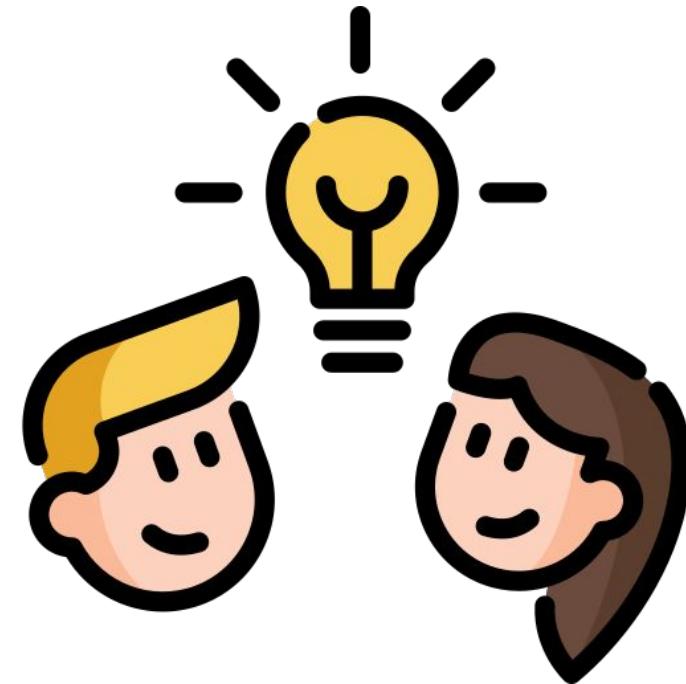
Veremos el progreso del mismo en el mismo panel donde estamos trabajando.



Iniciar Restauración

Los tiempos de recuperación pueden ser muy diferentes en cada caso, dependiendo del procesador, RAM, velocidad del disco de almacenamiento, tamaño del archivo de base de datos, etcétera.

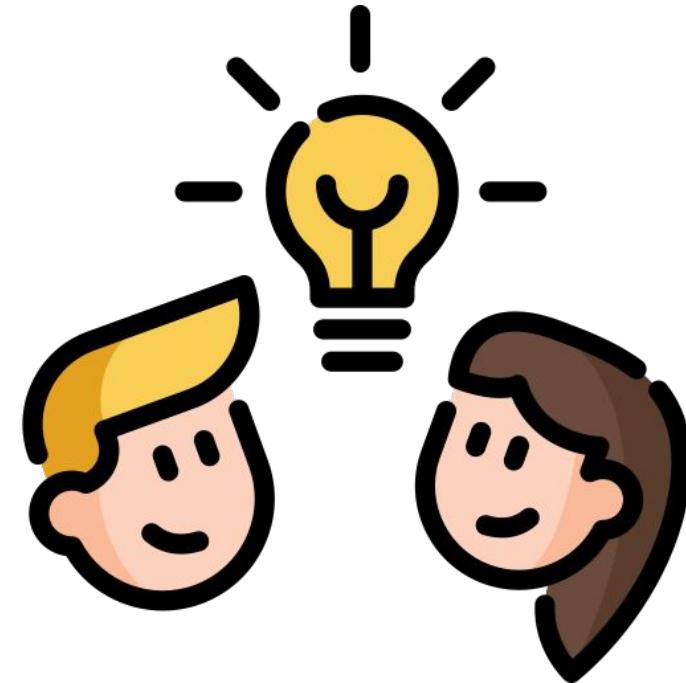
En este último caso, a mayor tamaño de la base de datos, mayor será el tiempo estimado en recuperar la información respaldada.



Iniciar Restauración

Y como escenario ideal para recuperar datos, recomendamos que el motor de base de datos esté dedicado a este proceso. Evita que la base de datos esté brindando acceso a otros usuarios y/o aplicaciones.

El escenario ideal es poner el servidor en mantenimiento, así podrás acortar los tiempos de recuperación de la información.



Sección práctica

Tomaremos la base de datos Northwind para implementar en ella un proceso de Backup.

Realizaremos ambos pasos (backup - restauración), sobre su estructura y datos.



Prácticas

Practica con la bb.dd. Northwind instalada en Mysql Workbench. Realiza las siguiente tareas:

- Prepara el ambiente para realizar una copia de seguridad
- Elige la opción **Dump to Self-Contained File**
- Inicia el proceso y espera a que finalice el mismo
- Ya finalizado, Realiza la restauración del mismo con la herramienta **Data Import/Restore**



Prácticas

Ya finalizado, Realiza la restauración del mismo con la herramienta **Data Import/Restore**.

- Elimina primero la bb.dd Northwind instalada en tu motor de base de datos
- Crea un nuevo Esquema con el mismo nombre, y agrega en éste la sigla “v2.0”
Ejemplo: **NorthwindV2.0**
- Finalmente restaura el contenido de tablas, vistas y datos.



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



**Argentina
programa
4.0**



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

***primero
la gente***

Clase 30: Bases de datos Relacionales

Transaction Control Language

Agenda de hoy

- A. TRANSACTION CONTROL LANGUAGE
 - a. START TRANSACTION / BEGIN
 - b. COMMIT
 - c. ROLLBACK
 - d. Savepoint
- B. VISTAS SQL
 - a. Crear una Vista SQL
 - b. Modificar una Vista SQL
 - c. Eliminar una Vista SQL
- C. CONSULTAS SOBRE EL PROYECTO FINAL



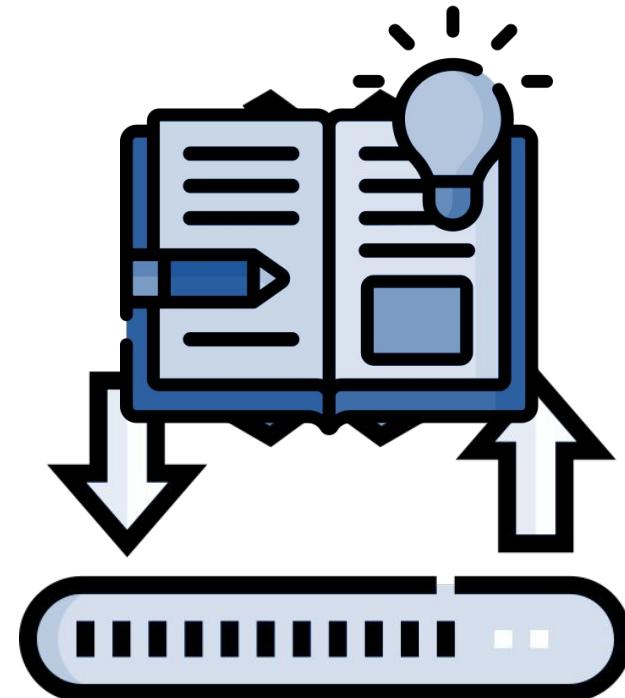
Transaction Control Language

Transaction Control Language

Hasta el momento, todas las operaciones DML que realizamos sobre SQL manejan una estructura transaccional directa.

Esto significa que, cuando ejecutamos cualquier operación de inserción, modificación o eliminación de datos, estas impactan directamente en la o las tablas vinculadas.

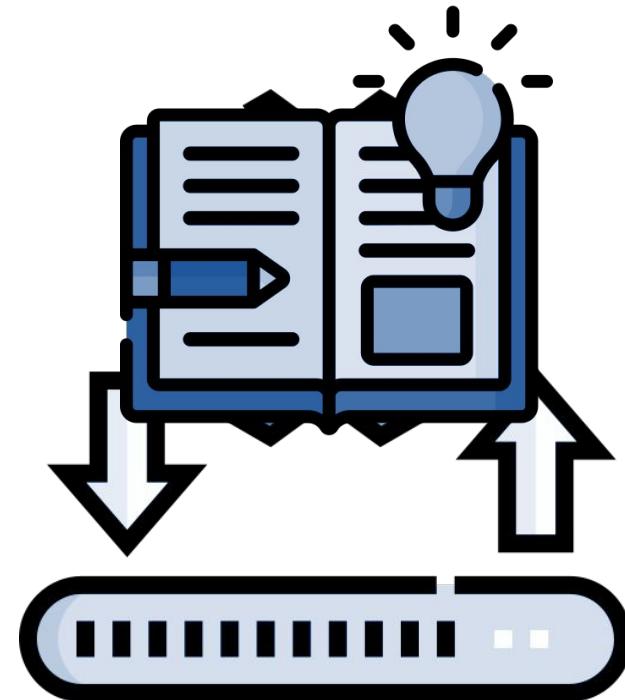
Pero si analizamos este proceso, ante el más mínimo error que cometamos en la instrucción, esta afectará los datos de manera directa.



Transaction Control Language

Pero, entre todas las opciones que maneja el lenguaje SQL, para los diferentes escenarios de operaciones que se pueden realizar, encontramos uno que nos ayudará a prevenir cualquier posible error: este es el sublenguaje **TCL**.

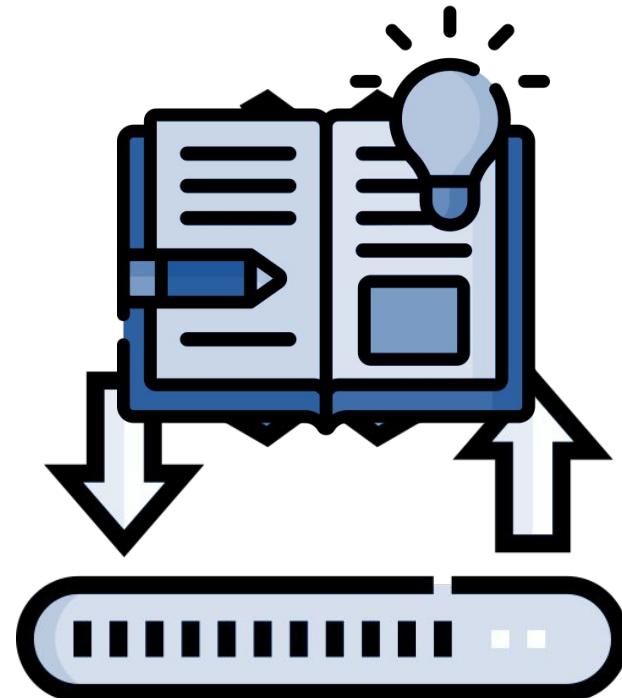
Su sigla proviene de **Transaction Control Language**, y es el sublenguaje encargado de administrar las transacciones en una base de datos. TCL es utilizado para administrar cada operación que realicemos mediante cláusulas DML.



Transaction Control Language

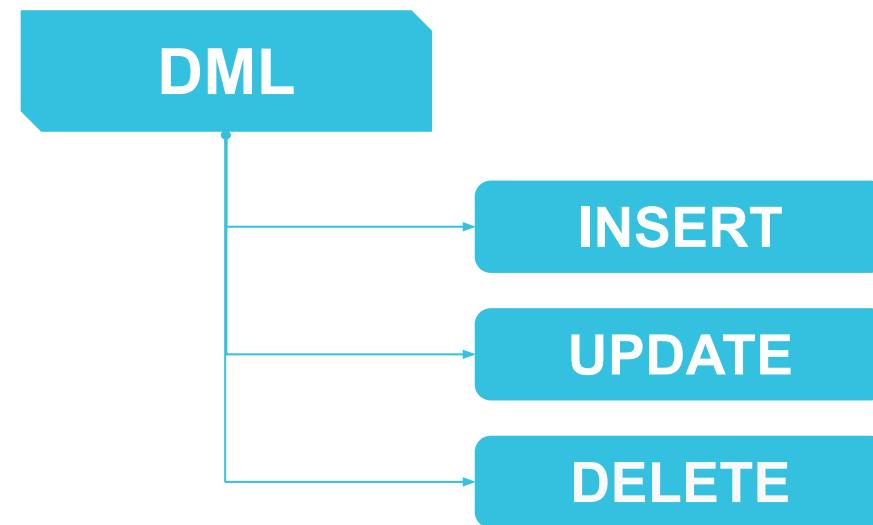
El rol de TCL es fundamental ya que, a través del mismo, obtenemos el control de las cláusulas operativas DML, agrupando las mismas de manera tal para que se establezca una lógica transaccional cuando realizamos múltiples operaciones afectando a los datos de una o más tablas.

TCL es la herramienta efectiva, la cual nos ayuda a mantener la integridad de los datos manipulados.



Transaction Control Language

Para refrescar conocimientos, las operaciones DML son aquellas que nos permiten trabajar con los tres tipos de operaciones más importantes sobre una bb.dd:

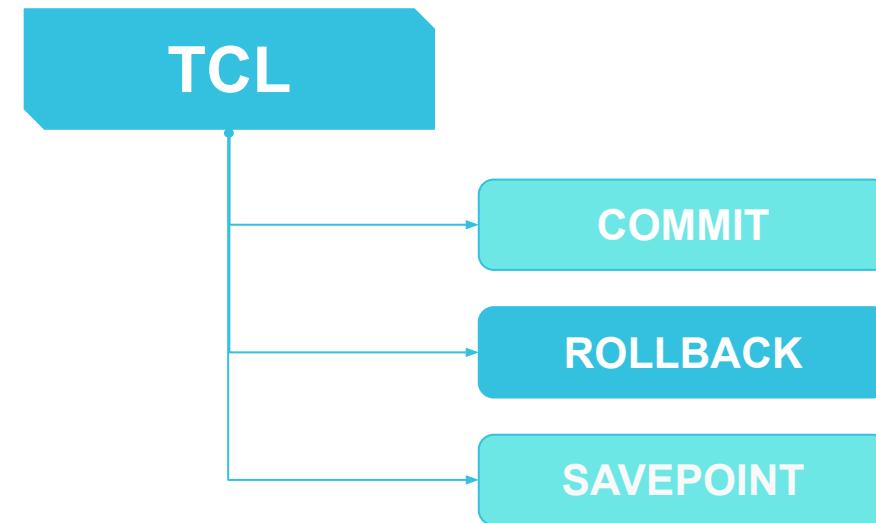


COMANDOS TCL PARA CONTROLAR TRANSACCIONES

Comandos TCL para controlar transacciones

MySQL incluye tres comandos integrados en el lenguaje SQL, los cuales se integran a las cláusulas homónimas, para controlar las operaciones DML durante el proceso de ejecución de las mismas.

Estos son:



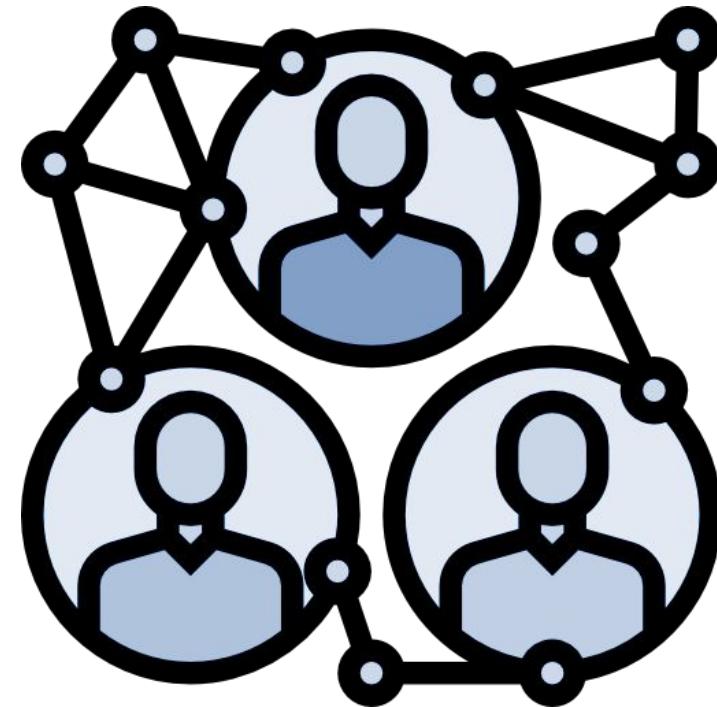
COMMIT



Comandos TCL para controlar transacciones

COMMIT es un comando el cual permite “*confirmar*” la transacción o transacciones realizadas sobre una o más tablas.

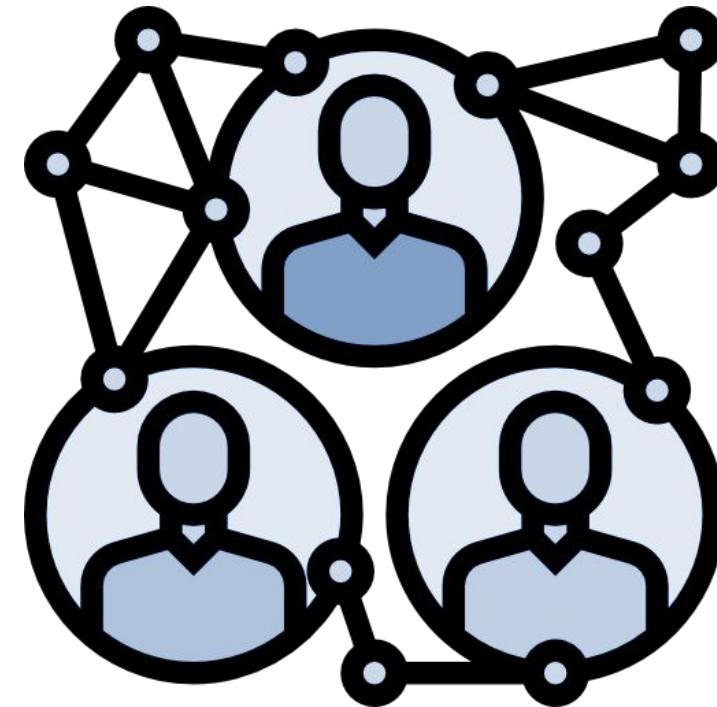
Cuando éste es ejecutado, se ocupa de guardar los cambios realizados en la o las tabla(s) a través del proceso de confirmación (commit), haciendo a estos permanentes.



Comandos TCL para controlar transacciones

Cuando ejecutamos alguna de las cláusulas DML, la(s) modificación(es) realizada(s) se guardan de forma “*temporal*” en la memoria de la computadora que las ejecuta.

Al invocar el comando COMMIT, dichas modificaciones terminan impactando de forma definitiva en el Servidor de base de datos, reflejando así los cambios en la tabla o las tablas.

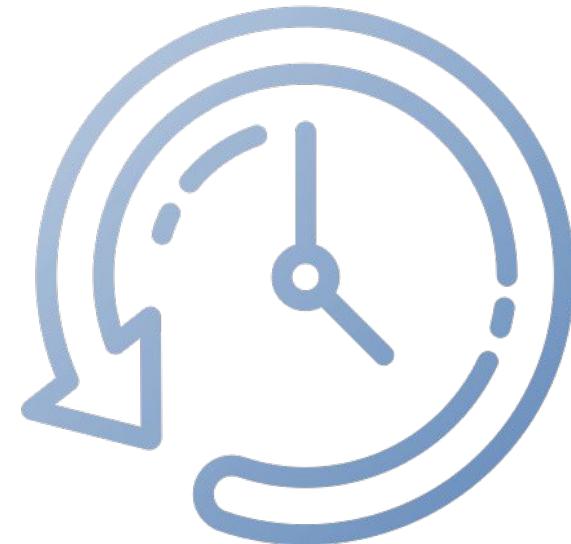


ROLLBACK

Comandos TCL para controlar transacciones

ROLLBACK “deshace” la operación DML realizada previamente. Su rol es, básicamente, volver al estado anterior todos los cambios aplicados sobre todas las tablas en cuestión.

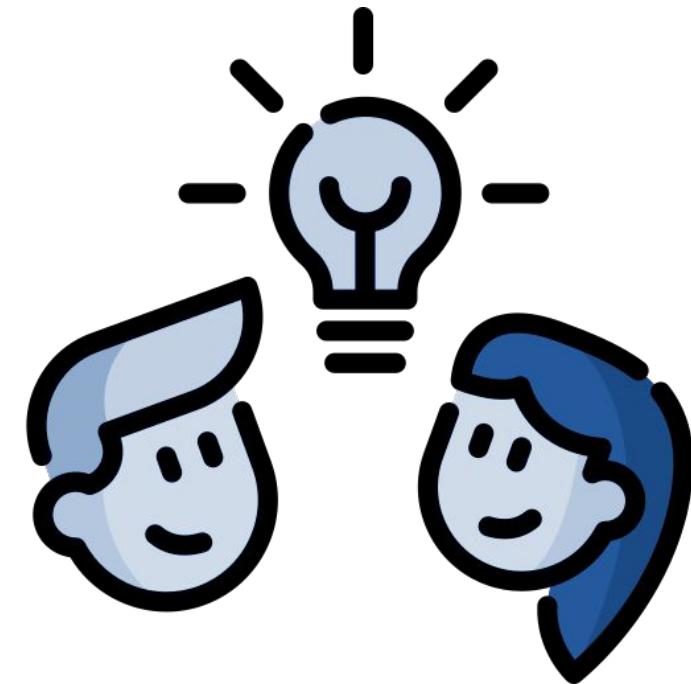
Su comportamiento es similar al uso del comando **Undo** (*deshacer*), o **Ctrl + Z**, que utilizamos de forma frecuente en cualquier aplicación de software en una computadora.



Comandos TCL para controlar transacciones

El comando ROLLBACK solo funciona para revertir modificaciones, en escenarios donde no se ha ejecutado previamente el comando COMMIT.

En el caso de realizar una operación DML y ejecutar el comando COMMIT inmediatamente, la operación habrá impactado en el motor de base de datos y ROLLBACK no surtirá efecto alguno.

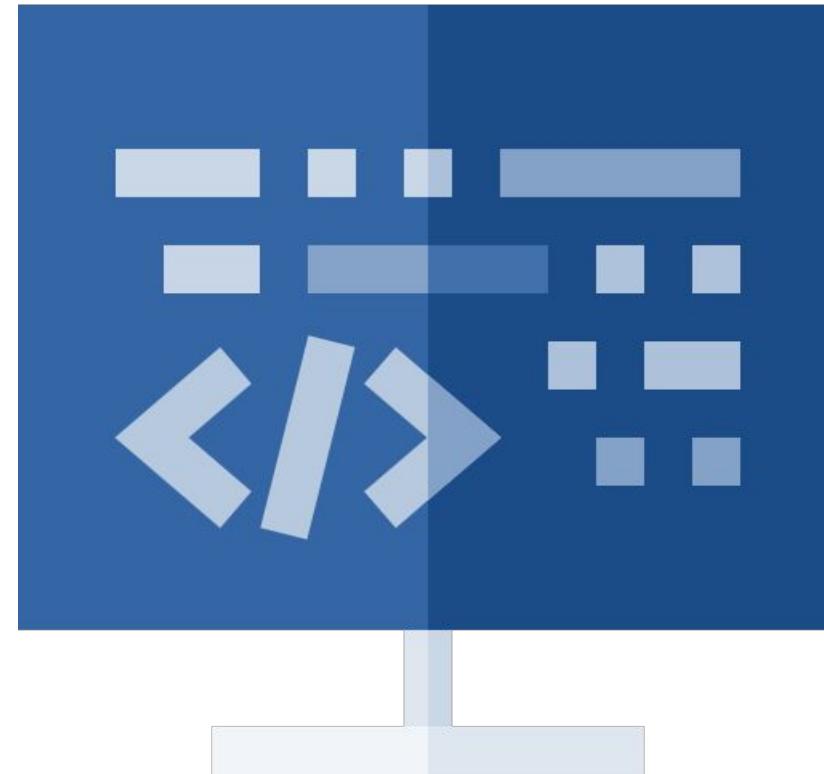


Savepoint

Comandos TCL para controlar transacciones

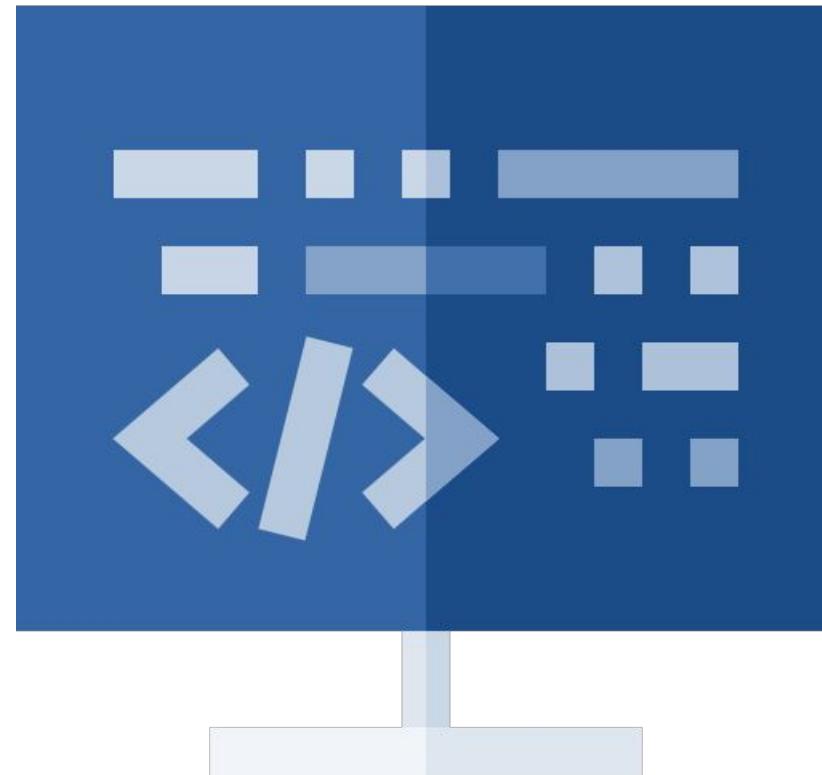
SAVEPOINT funciona en combinación con **Rollback** como una especie de Bookmark para establecer un punto de retroceso al momento de ejecutar el comando **ROLLBACK**.

Es ideal para implementarlo en modificaciones masivas de registros, estableciendo una marca específica cada cierto bloque de registros modificados.



Comandos TCL para controlar transacciones

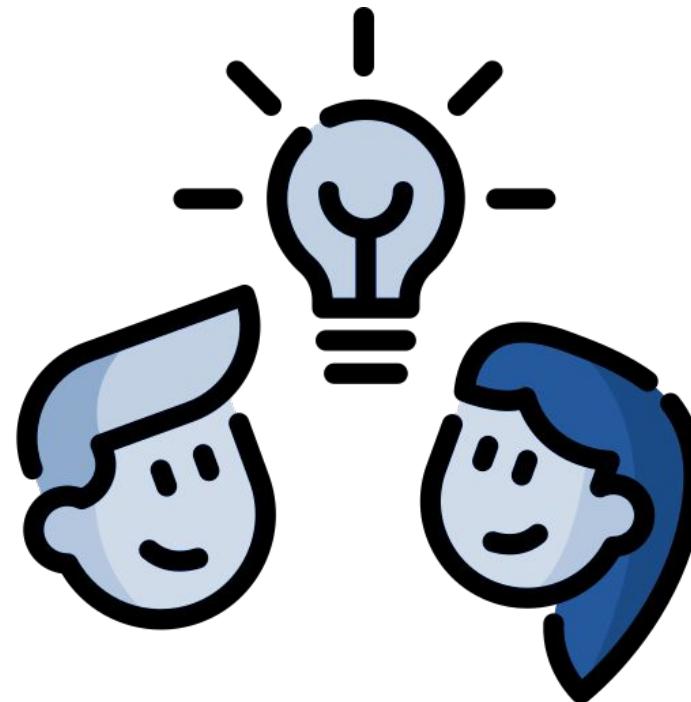
De esta forma, si en algún punto de la modificación masiva debemos ejecutar ROLLBACK, podemos hacerlo definiendo alguno de los savepoint definidos, para no tener que perder todo el bloque de registros modificados.



Comandos TCL para controlar transacciones

Al igual que lo visto anteriormente con el uso de Rollback, una vez ejecutado el comando COMMIT sobre alguna transacción en cuestión, todo tipo de SAVEPOINT que hayamos establecido previamente, se perderá.

Esto sucede porque, SAVEPOINT, funciona en combinación con ROLLBACK.



Escenario transaccional actual

Escenario transaccional actual

Antes de ingresar en las prácticas con código SQL, veamos cómo Mysql Workbench ayuda a entender las transacciones, a través de sus herramientas gráficas para manipular operaciones DML.

The screenshot shows the MySQL Workbench interface with a query results window. The query executed is:

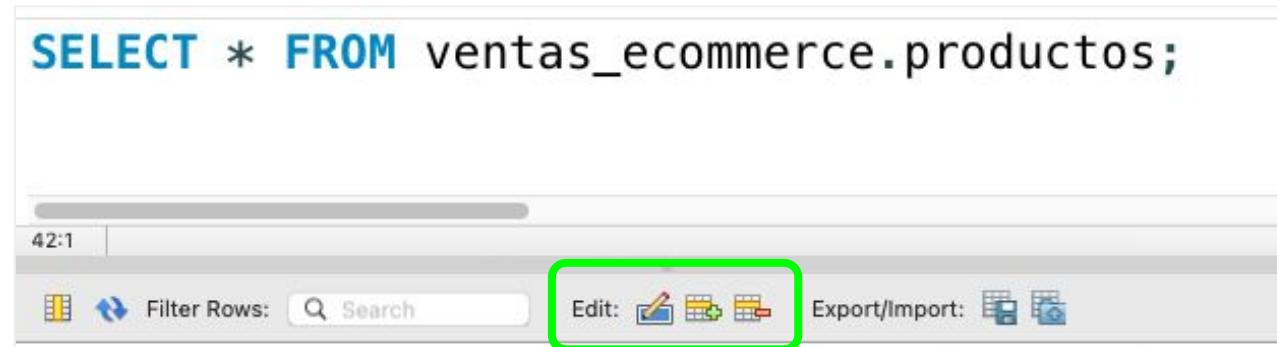
```
1 • SELECT * FROM ventas_ecommerce.productos;
```

The results grid displays data from the 'productos' table in the 'ventas_ecommerce' schema. The columns are: id, nombre, existencia, precio, and precio_compra. The data includes various computer peripherals and components. A vertical scroll bar is visible on the right side of the grid.

id	nombre	existencia	precio	precio_compra
1	TRACKPAD BLUETOOTH	1	22000	17600
2	TRACKPAD INALÁMBRICO USB	1	22000	17600
3	TECLADO INALÁMBRICO ES-BT	1	15000	10500
4	TECLADO MECÁNICO CABLEADO ES	1	7500	5200
5	TECLADO INALÁMBRICO ES-USB	0	12500	8700
6	MOUSE INALÁMBRICO BT	1	6500	4500
8	MOUSE INALÁMBRICO USB	1	5500	3300
17	MOTHERBOARD AM4 - ASROCK A320...	0	6599	4619
18	MOTHERBOARD AM4 - ASROCK B450...	0	9239	6467
19	MOTHERBOARD AM4 - ASROCK B550...	0	8299	5809
20	MOTHERBOARD AM4 - GIGABYTE GA...	0	7499	5249
21	MOTHERBOARD AM4 - GIGABYTE GA...	0	8289	5802
22	PLACA DE VIDEO GEFORCE G 210 1G...	0	5599	3919
23	PLACA DE VIDEO GEFORCE G 210 1G...	0	5899	4129
24	PLACA DE VIDEO GEFORCE GT 710 1...	0	6689	4682



Escenario transaccional actual



```
SELECT * FROM ventas_ecommerce.productos;
```

42:1

Edit: Export/Import:

Ejecuta una consulta sobre alguna tabla. Verás que, al cargar la misma, existe un apartado para **agregar, editar, o eliminar registros**.

Pulsa, ahora, alguno de estos botones y realiza en la tabla, la operación con el registro que has elegido.

Escenario transaccional actual

id	DETALLE DEL PRODUCTO	CANTIDAD	PRECIO UNITARIO	PRECIO TOTAL	ESTADO
23	PLACA DE VIDEO GEFORCE G 210 1G...	0	5899	4129	
24	PLACA DE VIDEO GEFORCE GT 710 1...	0	6689	4682	

productos 1

Apply Revert

Los **botones de edición** funcionan como inicio de una transacción:

Puedes eliminar uno o más registros, agregar, o modificar uno existente pero, si no pulsas el botón **Apply**, los cambios no se harán efectivos.

Por lo tanto, el botón **Apply** toma el rol de la cláusula **COMMIT**, y el botón **Revert** el rol de la cláusula **ROLLBACK**.

Begin Transaction

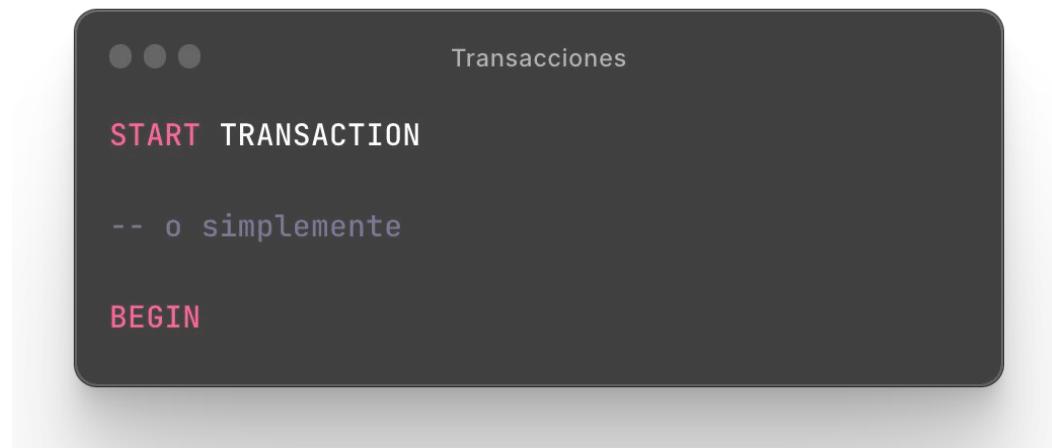
Si deseamos aplicar los comandos mencionados anteriormente, podemos hacerlo ante cualquier escenario u operación DML, escribiendo simplemente la cláusula **START TRANSACTION**.



Este será quien defina el ámbito de espacio temporal para cualquier tipo de modificación que realices, invocando una o más cláusulas DML.

Begin Transaction

Alternativamente, y a modo de poder ser compatible con otros motores SQL existentes en el mercado, MySQL también le da soporte a la cláusula **BEGIN**, propia de otros motores.



Autocommit



Begin Transaction

Mysql cuenta con una variable de entorno llamada **autocommit**, donde ajusta su valor a **1** para que cada operación DML impacte automáticamente en la tabla (*sin requerir confirmar una transacción*).

Para comenzar a trabajar con transacciones debemos desactivar previamente esta variable.

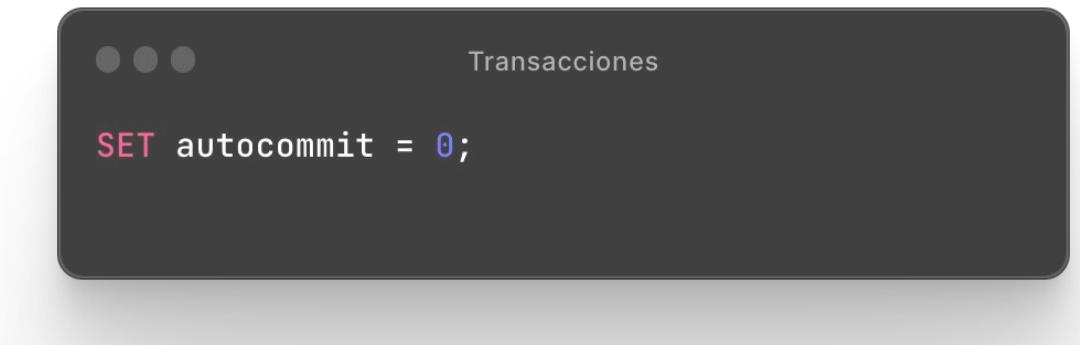
Escribamos en una ventana de script, lo siguiente:

```
••• Transacciones  
SELECT @@autocommit;
```



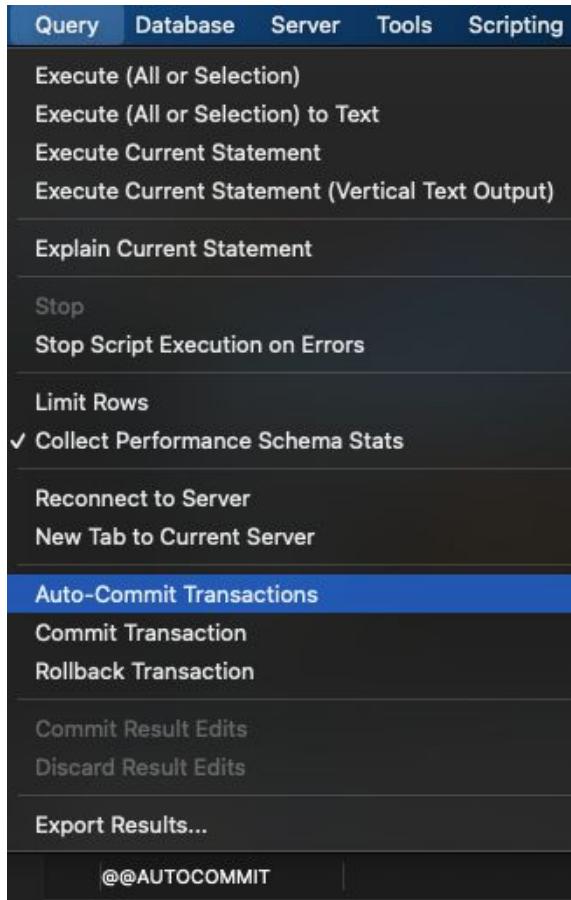
Begin Transaction

Si su valor es **1** debemos pasarlo a **0**, ejecutando el siguiente comando en la pestaña de script:



```
SET autocommit = 0;
```

También podemos verificar en el menú **Query > Auto-Commit Transactions** que no tenga el check. De tenerlo, haz clic sobre el punto de menú para desactivarlo.



Iniciar una Transacción

Iniciar una transacción

Llevemos estos ejemplos transaccionales a cada comando que vimos en la primera parte de esta clase. Comenzamos por el principal:

START TRANSACTION.

Veamos cómo se comporta el mismo al momento de ejecutar operaciones DML en una tabla de datos.



Iniciar una transacción

Ejecutando una consulta DML del tipo **UPDATE**, iniciaremos previamente la sentencia **START TRANSACTION**.

Esto nos permitirá ver que, el registro afectado, se modificará sin problema alguno.

Elige para probar esta cláusula, cualquier tabla de cualquier bb.dd de tu motor MySQL.

```
••• Transacciones

START TRANSACTION;

UPDATE ventas_ecommerce.productos
SET
    nombre = 'TRACKPAD BT'
WHERE
    id = 1;
SELECT * FROM ventas_ecommerce.productos;
```

Iniciar una transacción

Luego de ejecutar esta cláusula, salimos de Mysql Workbench y volvemos a ingresar.

Volvemos a ejecutar una consulta de selección sobre esta tabla, y podremos ver que el cambio solicitado al registro de la tabla en cuestión, no se confirmó. Esto sucede porque no finalizamos la transacción, ejecutando **COMMIT** para aplicar los cambios.

id	nombre	existencia
1	TRACKPAD BLUETOOTH	1
2	TRACKPAD INALÁMBRICO USB	1
3	TECLADO INALÁMBRICO ES-BT	1
4	TECLADO MECÁNICO CABLEADO ES	1
5	TECLADO INALÁMBRICO ES-USB	0
6	MOUSE INALÁMBRICO BT	1
7	MOUSE CABLEADO	0
8	MOUSE INALÁMBRICO USB	1
16	UPS 10500 WATTS	1
17	MOTHERBOARD AM4 - ASROCK A320M-HDV	0
18	MOTHERBOARD AM4 - ASROCK B450M AC	0
19	MOTHERBOARD AM4 - ASROCK B550M HDV	0
20	MOTHERBOARD AM4 - GIGABYTE GA-A320M-H	0
21	MOTHERBOARD AM4 - GIGABYTE GA-A520M-H	0
22	PLACA DE VIDEO GEFORCE G 210 1GB MSI	0

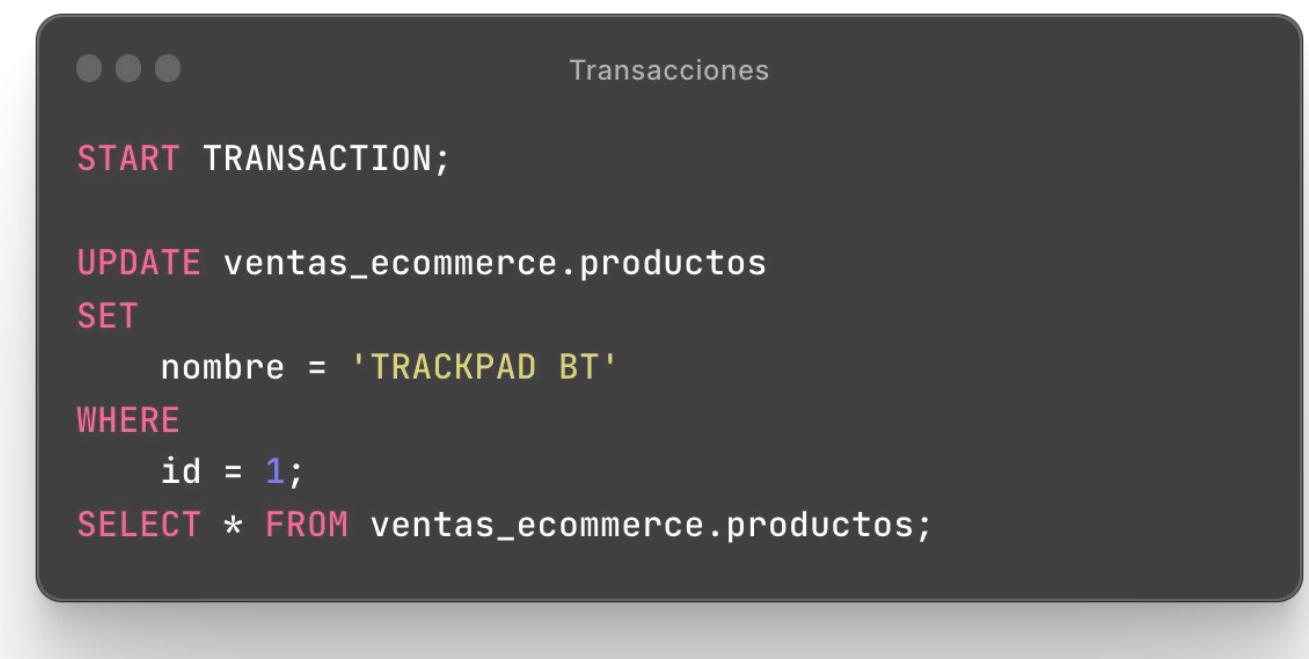


Confirmar una Transacción

Confirmar una transacción

Repitamos la ejecución de la consulta de modificación usando la cláusula **UPDATE**, e iniciando previamente la sentencia **START TRANSACTION**.

Apliquemos el cambio reutilizando la cláusula anterior, si no borramos la sentencia en cuestión.



The screenshot shows a mobile application interface with a dark background. At the top, there are three small circular icons followed by the text "Transacciones". Below this, a block of SQL code is displayed in white text:

```
START TRANSACTION;  
  
UPDATE ventas_ecommerce.productos  
SET  
    nombre = 'TRACKPAD BT'  
WHERE  
    id = 1;  
SELECT * FROM ventas_ecommerce.productos;
```



Confirmar una transacción

Al finalizar la ejecución de la cláusula **UPDATE**, agreguemos el comando **COMMIT**, el cual nos permite validar la transacción previamente ejecutada.

Luego de ello, refrescamos la visualización de la tabla mediante una consulta **SELECT**.

```
● ● ● Transacciones

START TRANSACTION;

UPDATE ventas_ecommerce.productos
SET
    nombre = 'TRACKPAD BT'
WHERE
    id = 1;
SELECT * FROM ventas_ecommerce.productos;

COMMIT;
```



Iniciar una transacción

Finalmente, veremos que el cambio ejecutado con la cláusula **UPDATE**, ha impactado de forma efectiva sobre el o los registro(s) indicados.

De esta manera, podremos tener el control necesarios sobre las operaciones riesgosas, pudiendo aplicar cláusulas DML y luego de validar las mismas, confirmar las operaciones.

9 * SELECT * FROM ventas_ecommerce.productos;		
10		
% 42:9		
id	nombre	existencia
1	TRACKPAD BT	1
2	TRACKPAD INALÁMBRICO USB	1
3	TECLADO INALÁMBRICO ES-BT	1
4	TECLADO MECÁNICO CABLEADO ES	1
5	TECLADO INALÁMBRICO ES-USB	0
6	MOUSE INALÁMBRICO BT	1
7	MOUSE CABLEADO	0
8	MOUSE INALÁMBRICO USB	1
16	UPS 10500 WATTS	1
17	MOTHERBOARD AM4 - ASROCK A320M-HDV	0
18	MOTHERBOARD AM4 - ASROCK B450M AC	0
19	MOTHERBOARD AM4 - ASROCK B550M HDV	0
20	MOTHERBOARD AM4 - GIGABYTE GA-A320M-H	0
21	MOTHERBOARD AM4 - GIGABYTE GA-A520M-H	0
22	PLACA DE VIDEO GEFORCE G 210 1GB MSI	0



Deshacer una Transacción

Confirmar una transacción

Ejecutemos una consulta de eliminación masiva de registros, sobre alguna tabla que permita aplicar una condición que afecte a múltiples filas de la tabla.

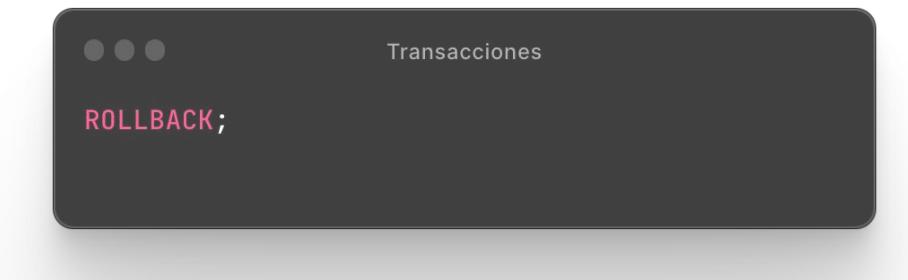
Iniciemos, como siempre, una transacción previo a ejecutar esta operación **DML** de eliminación.

```
• • • Transacciones  
START TRANSACTION;  
DELETE FROM  
    ventas_ecommerce.productos  
WHERE  
    nombre = 'MOTHERBOARD%';  
SELECT * FROM ventas_ecommerce.productos;
```

Confirmar una transacción

La cláusula SQL SELECT, nos mostrará que los registros en cuestión, han sido eliminados correctamente de la tabla.

Ahora, ejecutemos la sentencia rollback:



7 • SELECT * FROM ventas_ecommerce.productos;				
00% 10:8				
Result Grid Edit: Filter Rows: Search Export/Import:				
id	nombre	existencia	precio	precio_compra
1	TRACKPAD BLUETOOTH	1	22000	17600
2	TRACKPAD INALÁMBRICO USB	1	22000	17600
3	TECLADO INALÁMBRICO ES-BT	1	15000	10500
4	TECLADO MECÁNICO CABLEADO ES	1	7500	5200
5	TECLADO INALÁMBRICO ES-USB	0	12500	8700
6	MOUSE INALÁMBRICO BT	1	6500	4500
7	MOUSE CABLEADO	0	4100	2900
8	MOUSE INALÁMBRICO USB	1	5500	3300
16	UPS 10500 WATTS	1	15000	10500
22	PLACA DE VIDEO GEFORCE G 210 1GB MSI	0	5599	3919
23	PLACA DE VIDEO GEFORCE G 210 1GB EVGA	0	5899	4129
24	PLACA DE VIDEO GEFORCE GT 710 1GB MSI	0	6689	4682
25	MICROBOARD RPI 4TX	0	0	0
27	TRACKBALL VINTAGE GENIUS	0	950	1240
HULL	HULL	HULL	HULL	HULL



Confirmar una transacción

Con una nueva consulta de selección, posterior a **ROLLBACK**, veremos que el set de registros eliminados, volverá a su estado original.



The screenshot shows a MySQL command-line interface. At the top, there are two statements: 'ROLLBACK;' and 'SELECT * FROM ventas_ecommerce.productos;'. Below the statements is a progress bar at 00% and a timestamp of 1:19. The interface includes tabs for 'Result Grid' and 'Edit', and buttons for 'Filter Rows', 'Search', 'Export/Import', and other database management functions. The main area displays a grid of product data:

id	nombre	existencia	precio	precio_compra
1	TRACKPAD BLUETOOTH	1	22000	17600
2	TRACKPAD INALÁMBRICO USB	1	22000	17600
3	TECLADO INALÁMBRICO ES-BT	1	15000	10500
4	TECLADO MECÁNICO CABLEADO ES	1	7500	5200
5	TECLADO INALÁMBRICO ES-USB	0	12500	8700
6	MOUSE INALÁMBRICO BT	1	6500	4500
7	MOUSE CABLEADO	0	4100	2900
8	MOUSE INALÁMBRICO USB	1	5500	3300
16	UPS 10500 WATTS	1	15000	10500
17	MOTHERBOARD AM4 - ASROCK A320M-HDV	0	6599	4619
18	MOTHERBOARD AM4 - ASROCK B450M AC	0	9239	6467
19	MOTHERBOARD AM4 - ASROCK B550M HDV	0	8299	5809
20	MOTHERBOARD AM4 - GIGABYTE GA-A320M-H	0	7499	5249
21	MOTHERBOARD AM4 - GIGABYTE GA-A520M-H	0	8289	5802
22	PLACA DE VIDEO GEFORCE G 210 1GB MSI	0	5599	3919



Savepoint

Savepoint

Por último, nos queda ver cómo **SAVEPOINT** nos ayudará a controlar una modificación masiva de registros, pudiendo confirmar o deshacer por lotes, según consideremos, acorde a la lógica operativa.

Debemos tener presente que, toda instrucción asociada a este comando, sólo será ejecutable en bases de datos **innoDB**.



Savepoint

El comando **SAVEPOINT** requiere establecer un identificador cada cierto punto para definir la posición del lote de registros de modificación masiva.

El criterio del nombre de cada es definido por nosotras, de acuerdo a algún parámetro válido o simplemente a discreción.

```
••• Transacciones  
  
USE ventas_ecommerce;  
  
START TRANSACTION;  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #1', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #2', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #3', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #4', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #5', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #6', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #7', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #8', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #9', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #10', 0, 0, 0);  
SAVEPOINT lote_1_10;  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #11', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #12', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #13', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #14', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #15', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #16', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #17', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #18', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #19', 0, 0, 0);  
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #20', 0, 0, 0);  
SAVEPOINT lote_11_20;  
...
```



Savepoint

9	•	SELECT * FROM ventas_ecommerce.productos;		
10				
100% ◇ 42:9				
Result Grid				
Filter Rows:		Search		
Edit:				
Export/Import:				
id	nombre	existencia	precio	precio_compra
29	REGISTRO GENÉRICO #1	0	0	0
30	REGISTRO GENÉRICO #2	0	0	0
31	REGISTRO GENÉRICO #3	0	0	0
32	REGISTRO GENÉRICO #4	0	0	0
33	REGISTRO GENÉRICO #5	0	0	0
34	REGISTRO GENÉRICO #6	0	0	0
35	REGISTRO GENÉRICO #7	0	0	0
36	REGISTRO GENÉRICO #8	0	0	0
37	REGISTRO GENÉRICO #9	0	0	0
38	REGISTRO GENÉRICO #10	0	0	0
39	REGISTRO GENÉRICO #11	0	0	0
40	REGISTRO GENÉRICO #12	0	0	0
41	REGISTRO GENÉRICO #13	0	0	0
42	REGISTRO GENÉRICO #14	0	0	0
43	REGISTRO GENÉRICO #15	0	0	0
44	REGISTRO GENÉRICO #16	0	0	0
45	REGISTRO GENÉRICO #17	0	0	0
46	REGISTRO GENÉRICO #18	0	0	0
47	REGISTRO GENÉRICO #19	0	0	0
48	REGISTRO GENÉRICO #20	0	0	0

Podemos ver como **SAVEPOINT** impactó correctamente cada uno de los registros insertados en esta operación **DML** masiva.

Action Output	Time	Action
✓ 22	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #10', 0, 0, 0)
✓ 23	11:09:05	SAVEPOINT lote_1_10
✓ 24	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #11', 0, 0, 0)
✓ 25	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #12', 0, 0, 0)
✓ 26	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #13', 0, 0, 0)
✓ 27	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #14', 0, 0, 0)
✓ 28	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #15', 0, 0, 0)
✓ 29	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #16', 0, 0, 0)
✓ 30	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #17', 0, 0, 0)
✓ 31	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #18', 0, 0, 0)
✓ 32	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #19', 0, 0, 0)
✓ 33	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #20', 0, 0, 0)
✓ 34	11:09:05	SAVEPOINT lote_11_20
✓ 35	11:09:14	SELECT * FROM ventas_ecommerce.productos LIMIT 0, 50000

Y, en la pestaña del log denominada **Action Output**, encontraremos los dos bookmarks generados mediante **SAVEPOINT**.

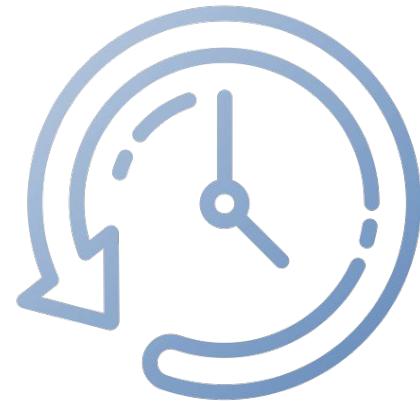


Savepoint

A través del comando **ROLLBACK TO SAVEPOINT**, podemos retroceder o “deshacer” el lote de comandos ejecutados hasta ese momento, de forma rápida y práctica.

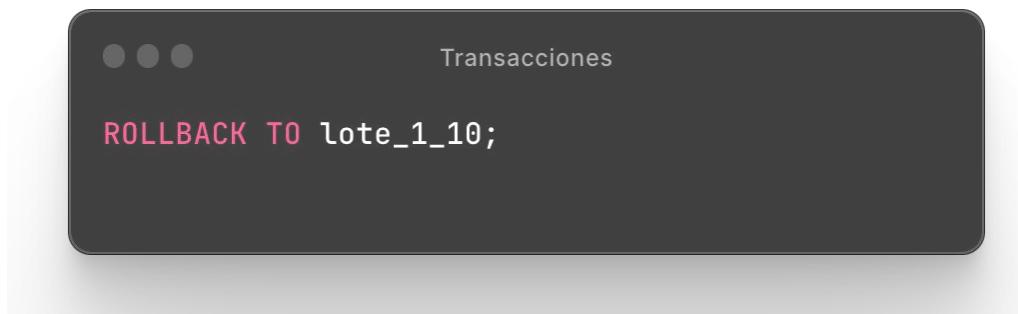
Su sentencia es:

```
...  
Transacciones  
  
ROLLBACK TO <savepoint>;
```



Savepoint

Si ejecutamos la cláusula:



desharemos los registros 11 al 20 insertados de forma masiva.

10 • ROLLBACK TO lote_1_10;
11 • SELECT * FROM ventas_ecommerce.productos;

id	nombre	existencia	precio	precio_compra
7	MOUSE CABLEADO	0	4100	2900
8	MOUSE INALÁMBRICO USB	1	5500	3300
16	UPS 10500 WATTS	1	15000	10500
17	MOTHERBOARD AM4 - ASROCK A320M-HDV	0	6599	4619
18	MOTHERBOARD AM4 - ASROCK B450M AC	0	9239	6467
19	MOTHERBOARD AM4 - ASROCK B550M HDV	0	8299	5809
20	MOTHERBOARD AM4 - GIGABYTE GA-A320M-H	0	7499	5249
21	MOTHERBOARD AM4 - GIGABYTE GA-A520M-H	0	8289	5802
22	PLACA DE VIDEO GEFORCE G 210 1GB MSI	0	5599	3919
23	PLACA DE VIDEO GEFORCE G 210 1GB EVGA	0	5899	4129
24	PLACA DE VIDEO GEFORCE GT 710 1GB MSI	0	6689	4682
25	MICROBOARD RPI 4TX	0	0	0
27	TRACKBALL VINTAGE GENIUS	0	950	1240
28	TRACKBALL VINTAGE MICROSOFT	0	0	0
29	REGISTRO GENÉRICO #1	0	0	0
30	REGISTRO GENÉRICO #2	0	0	0
31	REGISTRO GENÉRICO #3	0	0	0
32	REGISTRO GENÉRICO #4	0	0	0
33	REGISTRO GENÉRICO #5	0	0	0
34	REGISTRO GENÉRICO #6	0	0	0
35	REGISTRO GENÉRICO #7	0	0	0
36	REGISTRO GENÉRICO #8	0	0	0
37	REGISTRO GENÉRICO #9	0	0	0
38	REGISTRO GENÉRICO #10	0	0	0
HULL	HULL	HULL	HULL	HULL



Savepoint

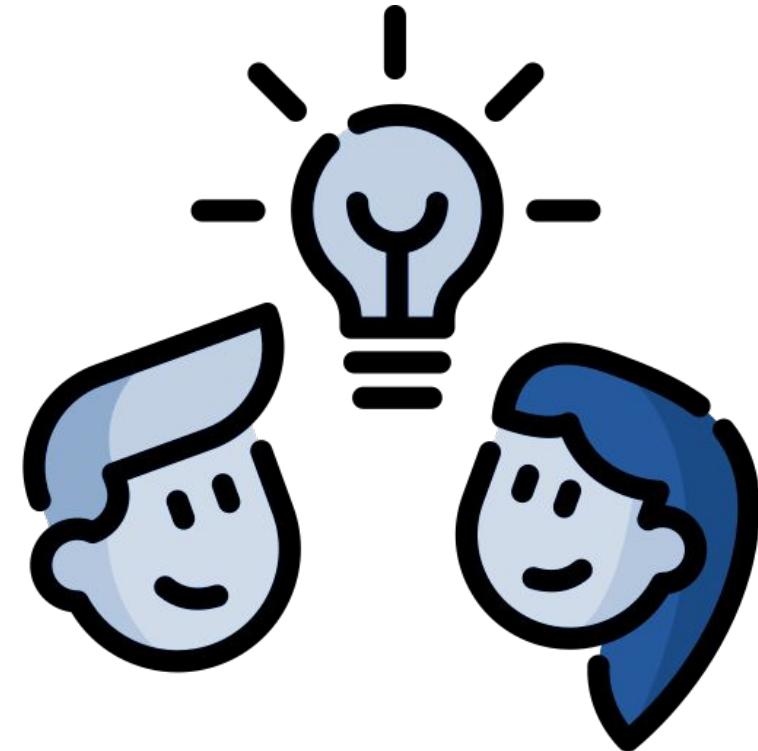
También, cuando el caso que lo amerite, podemos eliminar un **SAVEPOINT** ejecutando la cláusula:

```
● ● ●          Transacciones  
  
RELEASE <savepoint>;  
  
-- por ejemplo: RELEASE lote_1_10;
```

Savepoint

La implementación de un control de transacciones, tanto para confirmar como para deshacer las mismas, es utilizado principalmente dentro de los Stored Procedures.

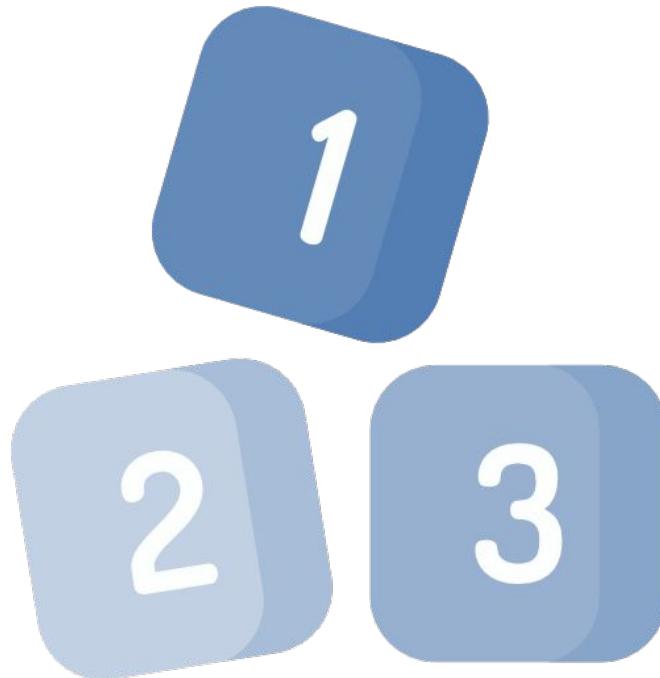
Combinando el mismo con el uso de variables, y la ejecución de cláusulas DML que dependan unas de otras, el control de transacciones ayudará a mantener consistente las tablas de la base de datos.



Vistas SQL

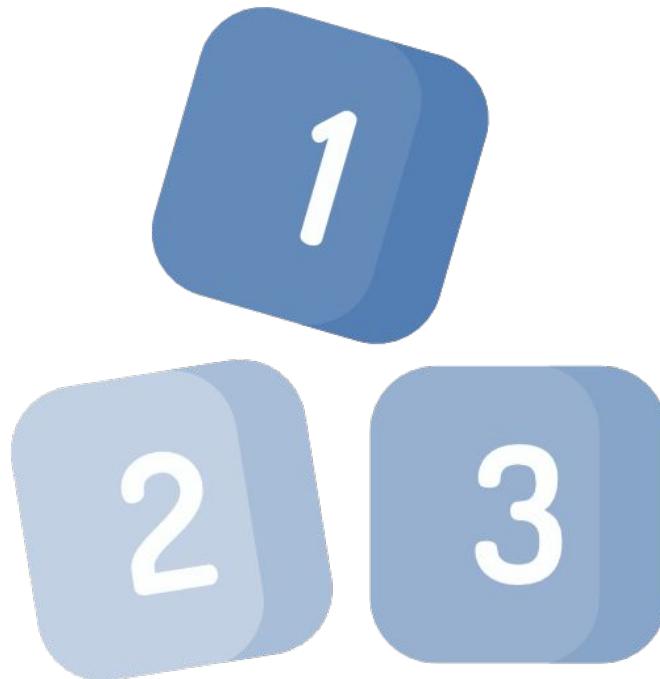
Vistas SQL

Desde el momento en el cual tengamos que repetir una consulta de selección de manera frecuente, la forma más fácil de simplificar esta tarea es creando una Vista SQL.



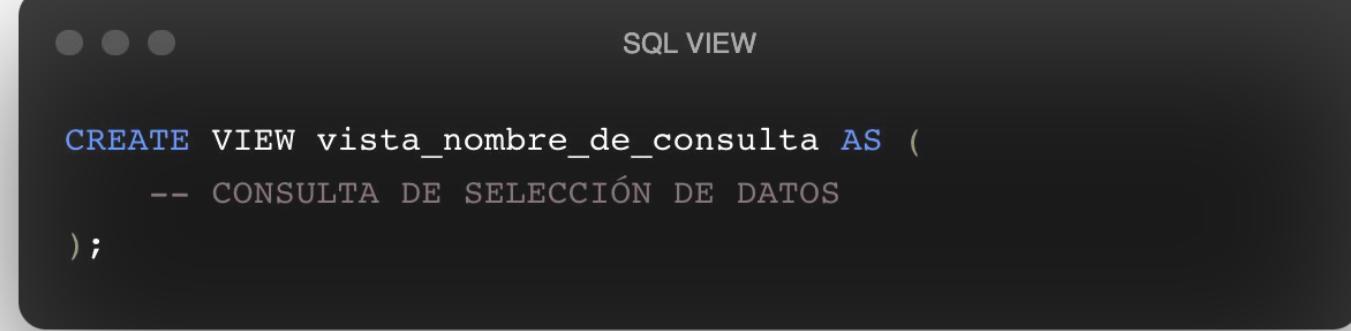
Vistas SQL

Las vistas almacenan la estructura de la consulta dentro del objeto **VIEWS** de nuestra base de datos, y desde allí podremos comenzar a ejecutarla de forma frecuente, sin tener que elaborar la consulta en cuestión cada vez que necesitemos acceder a la misma información.



Vistas SQL

Su estructura basada en **DDL** es muy simple.
Cuando definimos el nombre de la Vista, es
este mismo con el cual se guardará la misma
en el apartado **VIEWS** de la base de datos en
uso.



The screenshot shows a terminal window with a dark background and light-colored text. At the top right, it says "SQL VIEW". Below that, there is a code snippet for creating a view:

```
CREATE VIEW vista_nombre_de_consulta AS (
    -- CONSULTA DE SELECCIÓN DE DATOS
);
```



Vistas SQL

Veamos a continuación un ejemplo de implementación de Vista SQL, sobre la tabla **Orders** combinando en la misma la cláusula **WHEN THEN ELSE**, de acuerdo a diferentes estados de los registros aquí almacenados.

```
Vistas SQL

CREATE VIEW `new_view` AS
SELECT
    orders.OrderID AS OrderID,
    CAST(orders.OrderDate AS DATE) AS orderDate,
    CAST(orders.ShippedDate AS DATE) AS shippeDate,
    (CASE
        WHEN (orders.ShippedDate IS NULL) THEN 'Pendiente'
        WHEN (orders.ShippedDate > orders.OrderDate) THEN 'Entregado'
        ELSE 'En tránsito'
    END) AS OrderStatus
FROM
    orders;
```



Vistas SQL

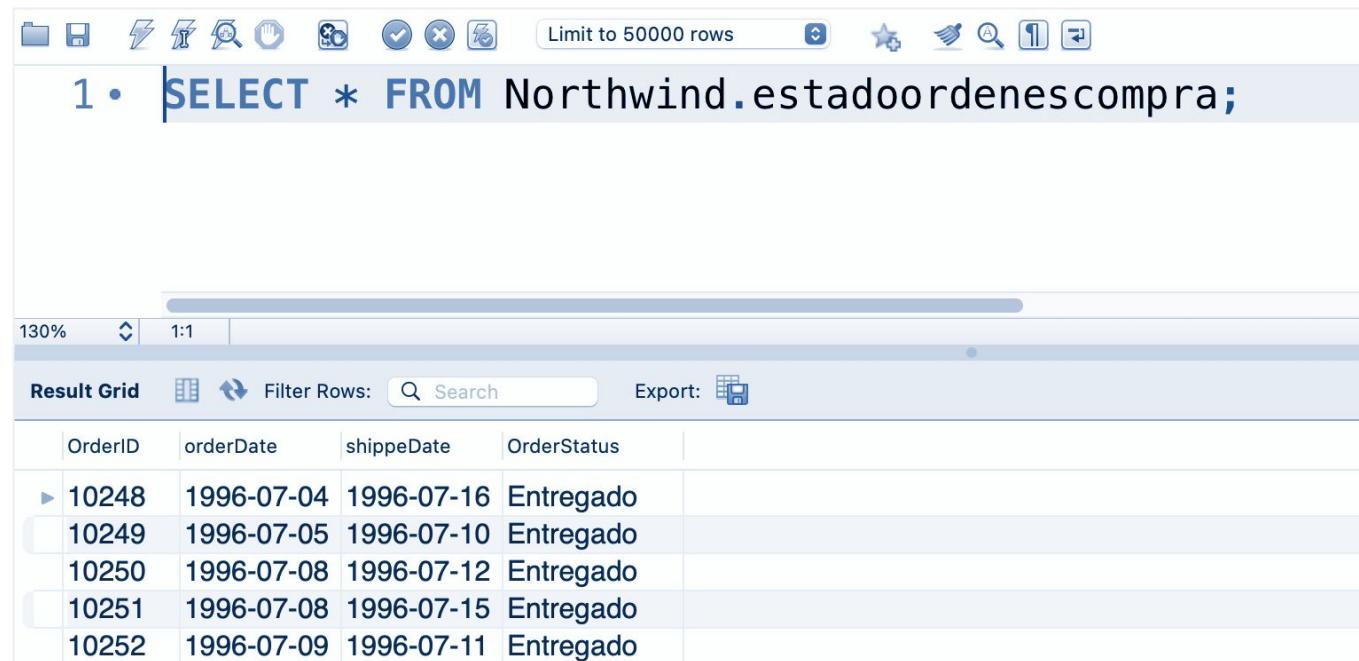
La cláusula **CREATE VIEW** nos permite definir a continuación, el nombre que queremos darle a la vista. Luego, el comando **AS** nos permite referenciar la consulta de selección correspondiente para que ya quede almacenada como vista SQL.

```
Vistas SQL

CREATE VIEW `estadooderdenesdecompra` AS
SELECT
    orders.OrderID AS OrderID,
    CAST(orders.OrderDate AS DATE) AS orderDate,
    CAST(orders.ShippedDate AS DATE) AS shippeDate,
    (CASE
        WHEN (orders.ShippedDate IS NULL) THEN 'Pendiente'
        WHEN (orders.ShippedDate > orders.OrderDate) THEN 'Entregado'
        ELSE 'En tránsito'
    END) AS OrderStatus
FROM
    orders;
```

Vistas SQL

Una vez creada la vista, podremos acceder a la misma tal como si fuese una tabla más en la bb.dd, con la diferencia que su información será de sólo lectura.



The screenshot shows a MySQL Workbench interface. At the top, there is a toolbar with various icons for file operations, search, and navigation. Below the toolbar, a query editor window displays the following SQL statement:

```
1 • | SELECT * FROM Northwind.estadoordenescCompra;
```

Below the query editor is a result grid. The grid has a header row with columns labeled: OrderID, orderDate, shippeDate, and OrderStatus. The data rows show the following information:

OrderID	orderDate	shippeDate	OrderStatus
10248	1996-07-04	1996-07-16	Entregado
10249	1996-07-05	1996-07-10	Entregado
10250	1996-07-08	1996-07-12	Entregado
10251	1996-07-08	1996-07-15	Entregado
10252	1996-07-09	1996-07-11	Entregado



Vistas SQL

La misma vista, nos permitirá establecer filtros sobre los datos representados, ya que oficia de intermediaria sobre la o las tablas vinculadas dentro de ésta.



The screenshot shows a MySQL Workbench interface. At the top, there is a toolbar with various icons. Below the toolbar, a SQL editor window displays the following code:

```
2 FROM Northwind.estadoordenesc compra
3 WHERE OrderID BETWEEN 10252 AND 10255
4 ORDER BY orderDate DESC;
```

Below the SQL editor is a progress bar indicating the execution status. The main area shows a "Result Grid" with the following data:

OrderID	orderDate	shippeDate	OrderStatus
10255	1996-07-12	1996-07-15	Entregado
10254	1996-07-11	1996-07-23	Entregado
10253	1996-07-10	1996-07-16	Entregado
10252	1996-07-09	1996-07-11	Entregado



Vistas SQL

A través del menú contextual del mouse, podremos acceder a la opción **ALTER VIEW**, para poder editar la consulta asignada a esta vista, y aplicarle cualquier cambio.

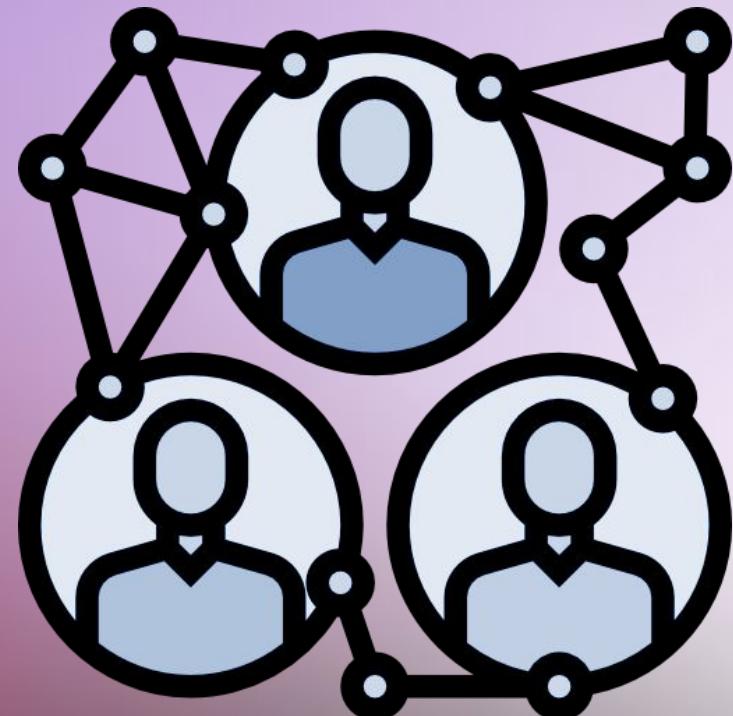


```
Name: estadodenordenescompra

1 • CREATE
2     ALGORITHM = UNDEFINED
3     DEFINER = `root`@`localhost`
4     SQL SECURITY DEFINER
5     VIEW `estadodenordenescompra` AS
6     SELECT
7         `orders`.`OrderID` AS `OrderID`,
8         CAST(`orders`.`OrderDate` AS DATE) AS `orderDate`,
9         CAST(`orders`.`ShippedDate` AS DATE) AS `shippeDate`,
10        (CASE
11            WHEN (`orders`.`ShippedDate` IS NULL) THEN 'Pendiente'
12            WHEN (`orders`.`ShippedDate` > `orders`.`OrderDate`) THEN 'Entregado'
13            ELSE 'En tránsito'
14        END) AS OrderStatus
15     FROM
16         orders
```

Espacio de Consultas

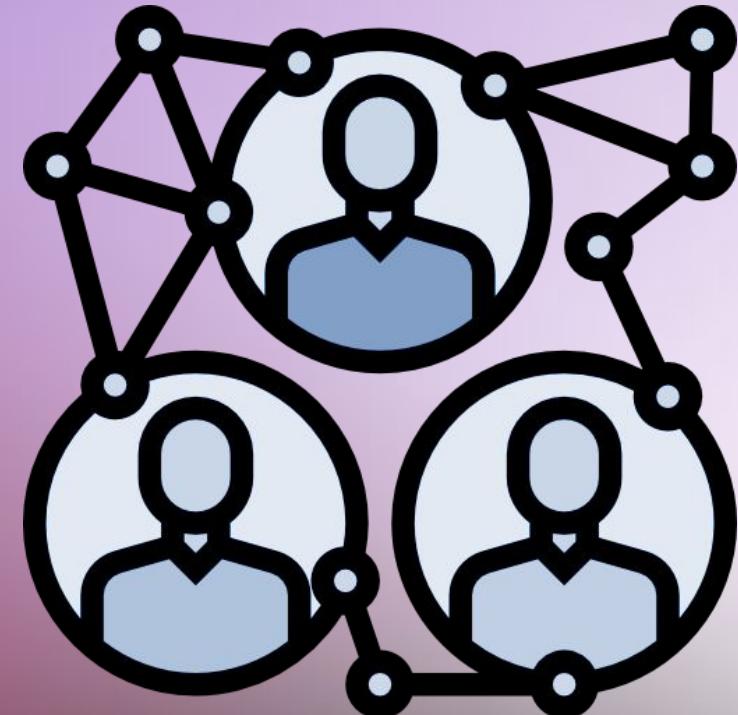
Abrimos este espacio de cara a realizar consultas, comentarios, o repasar algún tema específico, relacionado a finalizar de forma efectiva el último proyecto integrador.



Espacio de consultas

Recordemos qué debemos realizar en este:

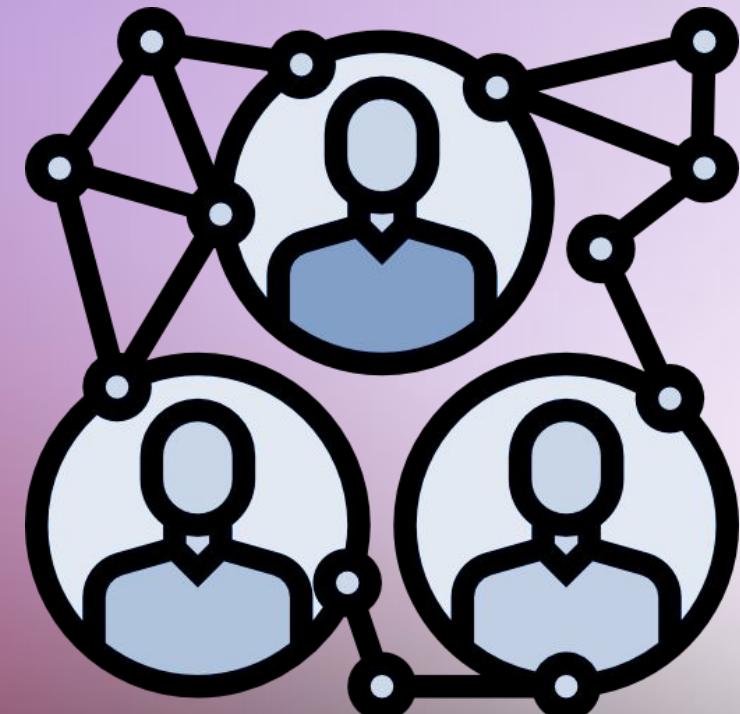
- diseña un modelo relacional de bb.dd utilizando la información del archivo **trailerflix.json**
 - la bb.dd debe contar con un estimado de 6+ tablas relacionales, y los datos JSON migrados
- crea los endpoint necesarios para ver:
 - información de las películas y series
 - actrices/actores y sus trabajos filmicos
 - filtrar por una película o serie específica
 - ver solo películas
 - ver solo series
 - y otros endpoint que veas viable incluir
- debes realizar la documentación acorde que explique cómo utilizar los endpoint existentes



Espacio de consultas

Finalmente, con el trabajo realizado, deberás crear un repositorio en Github, y publicar:

- la bb.dd trailerflix con sus tablas y datos cargados
- el modelo diseñado para su creación
- el código del proyecto Node
- la documentación en formato Markdown



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*



Argentina programa 4.0



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*

Clase 31: Node.js y Express

Publicar un proyecto en la nube - Por dónde seguir

Agenda de hoy

- A. Publicar un proyecto backend en la nube
- B. Opciones para publicar nuestro proyecto backend
 - a. Limitaciones de las nubes
 - b. Consignas del proyecto final
 - c. Por dónde seguir (Node JS)
- C. Otros conceptos importantes en backend
 - a. Manejo de Sockets
 - b. Consumir endpoint remotos
 - c. Manejo de Sesiones y Cookies
 - d. Por dónde seguir (Bases de Datos)



Publicar un proyecto backend en la nube

Publicar un proyecto backend en la nube

Hoy todo es una nube. Desde las más populares, como lo son [AWS](#), [Azure](#), o [GCP](#), hasta las menos resonantes pero con gran aceptación, todas se han vuelto una herramienta fundamental para alojar y desplegar aplicaciones.

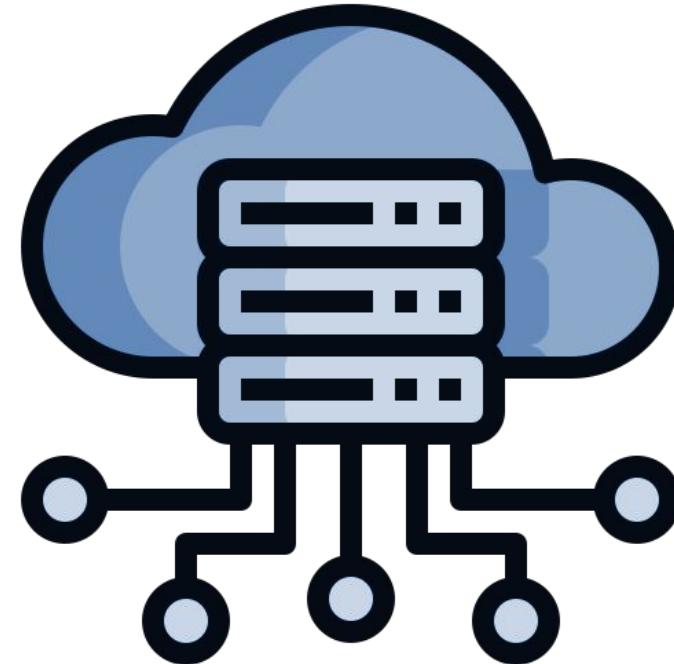
En esta última entrega del curso, veremos cómo podemos publicar nuestros proyectos backend de forma gratuita en alguna de las nubes existentes.



Publicar un proyecto backend en la nube

Cuando tenemos que publicar algún proyecto cloud en la nube, para una instancia como la nuestra que es utilizarlo como Portfolio y realizar experiencia en este terreno, debemos tener en cuenta algunos factores específicos:

- Ventajas de las nubes
- Plataformas gratuitas
- Pasos para publicar
- Conclusiones

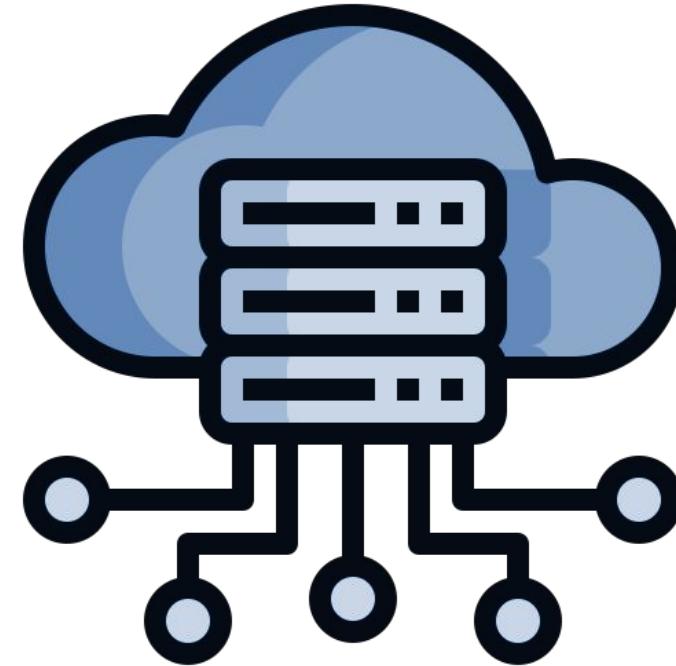


Publicar un proyecto backend en la nube

Ventajas de la nube

Al elegir la nube para alojar nuestro proyecto backend, disfrutamos de numerosas ventajas.

En primer lugar, no tenemos que preocuparnos por la infraestructura física, ya que la nube nos brinda una plataforma virtualizada y escalable que se adapta a nuestras necesidades.



Publicar un proyecto backend en la nube

Ventajas de la nube

Además, la nube nos ofrece un alto nivel de disponibilidad y redundancia, lo que garantiza que nuestra aplicación esté siempre en línea y accesible para los usuarios.



Publicar un proyecto backend en la nube

Plataformas gratuitas

Existen varias plataformas en la nube que ofrecen planes gratuitos para alojar proyectos backend desarrollados en **Node.js**.

Estas plataformas nos permiten desplegar nuestras aplicaciones de manera sencilla y sin incurrir en costos.



Publicar un proyecto backend en la nube

Plataformas gratuitas

Algunas de las opciones más populares incluyen a:

- Netlify
- Alibaba Cloud
- Render
- Fly.io
- Railway
- Glitch



Además de opciones varias que podemos encontrar en Google Cloud Platform, Microsoft Azure o Amazon Web Services.

Publicar un proyecto backend en la nube

Plataformas gratuitas

Estas plataformas proporcionan herramientas y servicios adicionales que facilitan la gestión y el monitoreo de nuestras aplicaciones en la nube.

Pero **un punto importante que debemos tener en cuenta** es que, **toda plataforma Cloud cambia constantemente sus *Términos y Condiciones***, por lo cual, lo que hoy puede ser gratuito en alguna plataforma, en algún tiempo indeterminado puede dejar de serlo.



Publicar un proyecto backend en la nube

Plataformas gratuitas

Estas plataformas proporcionan herramientas y servicios adicionales que facilitan la gestión y el monitoreo de nuestras aplicaciones en la nube.

Pero **un punto importante que debemos tener en cuenta** es que, **toda plataforma Cloud cambia constantemente sus *Términos y Condiciones***, por lo cual, lo que hoy puede ser gratuito en alguna plataforma, en algún tiempo indeterminado puede dejar de serlo.



Publicar un proyecto backend en la nube

Pasos para publicar un proyecto backend en la nube

Para publicar nuestro proyecto backend en la nube, seguimos algunos pasos clave. En primer lugar, creamos una cuenta en la plataforma de nuestra elección y configuramos nuestra aplicación.

Luego, utilizamos herramientas como la línea de comandos o la integración con sistemas de control de versiones para enviar nuestro código a la nube.



Publicar un proyecto backend en la nube

Pasos para publicar un proyecto backend en la nube

A continuación, configuramos variables de entorno y otros parámetros necesarios para que nuestra aplicación funcione correctamente en el entorno de la nube.

Por último, desplegamos nuestra aplicación y la ponemos en funcionamiento en la nube.

Casi todas las nubes cuentan hoy con herramientas vía **Terminal (npm)**, para realizar estos procesos.



Limitaciones de las nubes

Limitaciones de las nubes

Como bien venimos hablando, cada nube tiene su limitación en cuanto a qué servicios podemos utilizar gratuitamente, y cuáles requieren un pago.

Incluso en algunos casos, nos dan una capa gratuita bastante interesante, donde podremos publicar proyectos de media o alta complejidad, con múltiples bases de datos y mecanismos de seguridad. Pero esta publicación tendrá un límite de tiempo de gratuidad. Luego de ese límite, seguramente debamos comenzar a pagar un abono por los servicios utilizados.



Limitaciones de las nubes

Es clave poder analizar todo esto con el mayor detenimiento posible, y hasta incluso prestar atención a posibles límites internos, como ser una cuota de uso (xxx MB / mes ó xx GB / mes).

Todo tiene su costo y, a diferencia de las redes sociales, el mantenimiento operativo de estos servicios específicos, no pueden vivir de cuotas publicitarias como sí lo pueden hacer Facebook, Twitter o Linkedin.



Limitaciones de las nubes

En otros escenarios, podemos tal vez montar una aplicación Node.js en nube de forma gratuita, pero no podemos utilizar una bb.dd SQL o MongoDB local.

Algunas pueden permitirnos algunas bb.dd locales con un tiempo limitado de activación diario, otras pueden permitir una conexión a una bb.dd remota como ser los clúster de MongoDB, y otras plataformas tal vez sólo permitan un archivo JSON que simule un set de datos a servir vía API RESTFUL.



Publicando un proyecto Cloud

Publicando un proyecto Cloud

Veamos a continuación la sugerencia de la profe, para ver una plataforma Cloud donde podemos publicar algún proyecto basado en Node.js, de todos los que realizamos a lo largo de la cursada.



Espacio dedicado al proyecto final

**Abrimos nuevamente un espacio para que
sigas consultándonos sobre el proyecto
final, nos cuentes cómo vas trabajando en
éste, y/o te animes a mostrar tus avances
compartiendo pantalla.**



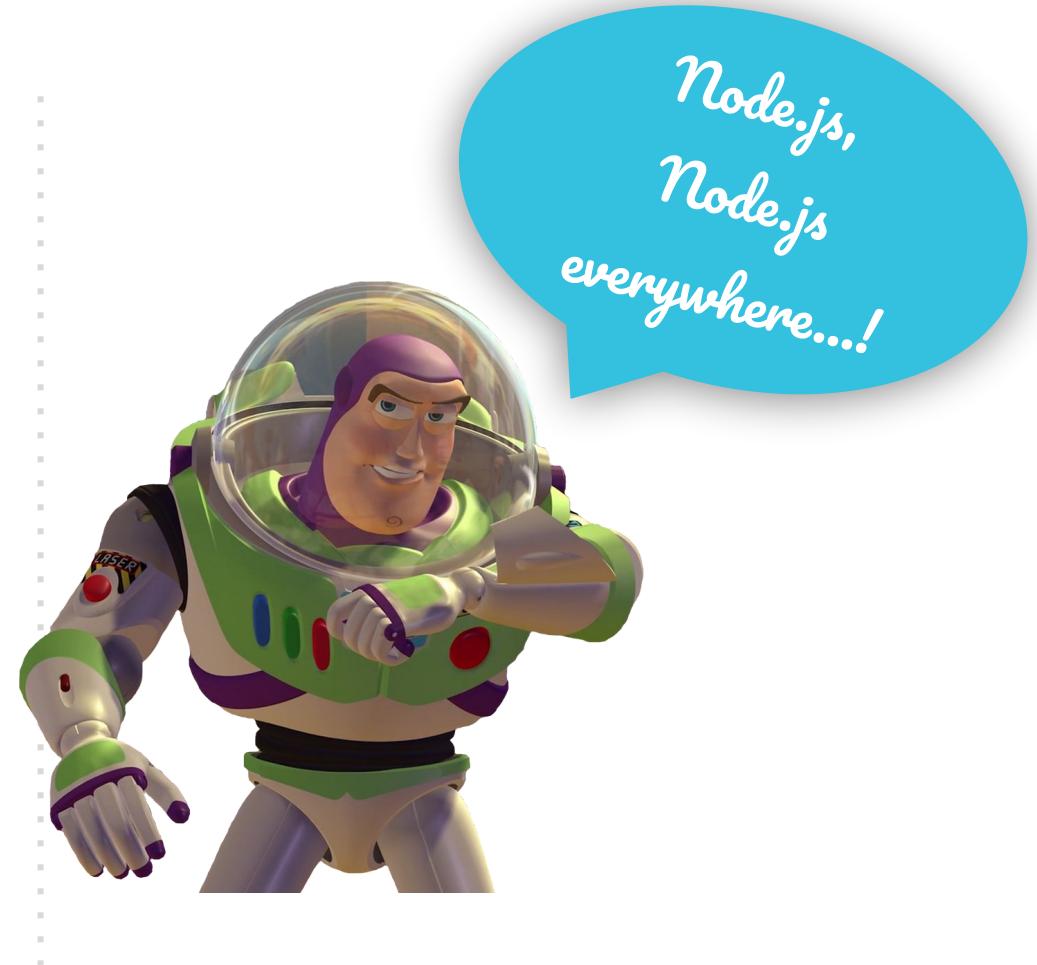
Por dónde seguir (Node JS)

Por dónde seguir (Node JS)

Estamos llegando al final de la cursada. Pero esto no es todo, aún nos queda mucho camino por recorrer.

Las bases de este curso te dejan los conocimientos más importantes de Node.js del lado del backend (los pilares).

Y contando con los pilares de esta fabulosa herramienta, puedes continuar el perfeccionamiento de la misma enfocándote en aprender el resto de las tecnologías satélite que la complementan.



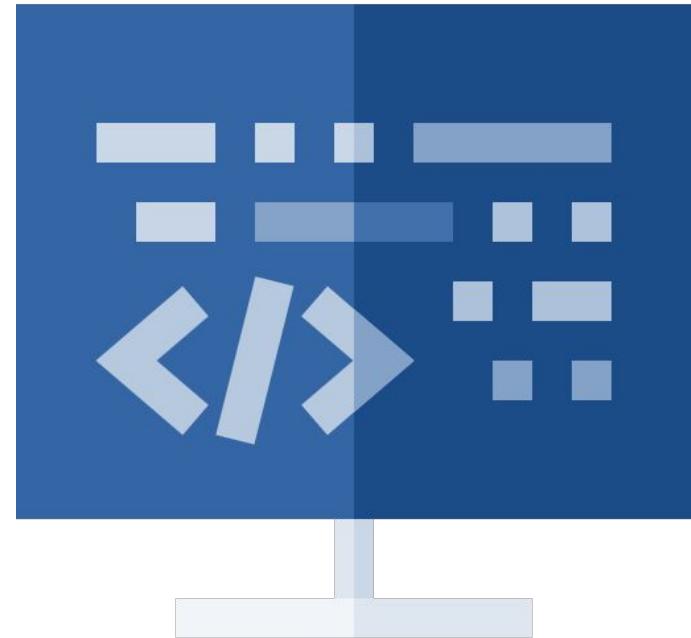
Más backend con Node.js

Más backend con Node.js

Bases de datos

Hay mucho más de backend en Node.js. En este curso nos enfocamos en explorar solo dos bases de datos específicas y de las más demandadas a nivel laboral: (MongoDB y MySQL)

Con las bases de MySQL aprendidas, puedes aprovechar y utilizar, por ejemplo, SQL Server, para entender cómo se integra este otro motor de bb.dd a Node.JS, utilizando el mismo Driver.



Más backend con Node.js

Bases de datos

De la mano de MongoDB, otra base de datos similar a MongoDB es Apache Cassandra. Esta es una bb.dd NoSQL, la cual almacena datos en formato JSON-like, y está diseñada para manejar grandes volúmenes de datos.



Puedes integrar Cassandra con Node.js utilizando el conector **Cassandra-Driver**, para realizar las operaciones CRUD.

Más backend con Node.js

Bases de datos

De la mano de MongoDB, otra base de datos similar a MongoDB es Apache Cassandra. Esta es una bb.dd NoSQL, la cual almacena datos en formato JSON-like, y está diseñada para manejar grandes volúmenes de datos.



Puedes integrar Cassandra con Node.js utilizando el conector **Cassandra-Driver**, para realizar las operaciones CRUD.

Seguridad

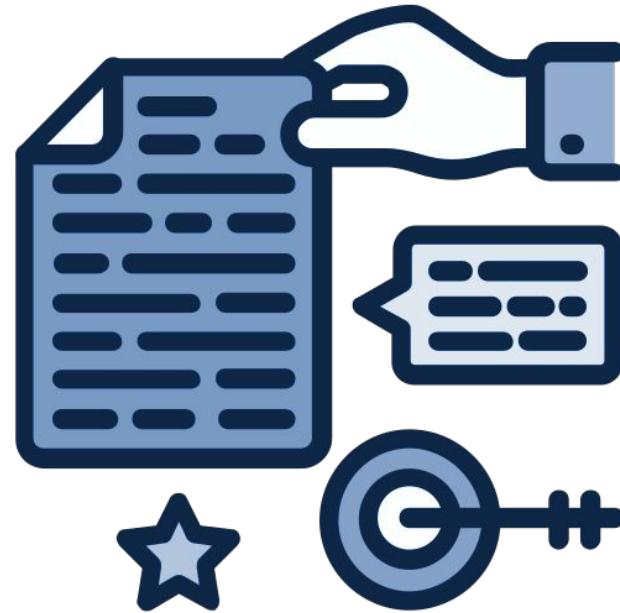


Más backend con Node.js

Seguridad

Hemos aprendido a manejar los niveles de seguridad, integrando JWT como herramienta generadora de Token.

Además de esta, en la actualidad se utiliza **Passport.js** para el manejo de Login y Registro de usuarios. Es ideal que adquieras experiencia en el manejo de la misma.

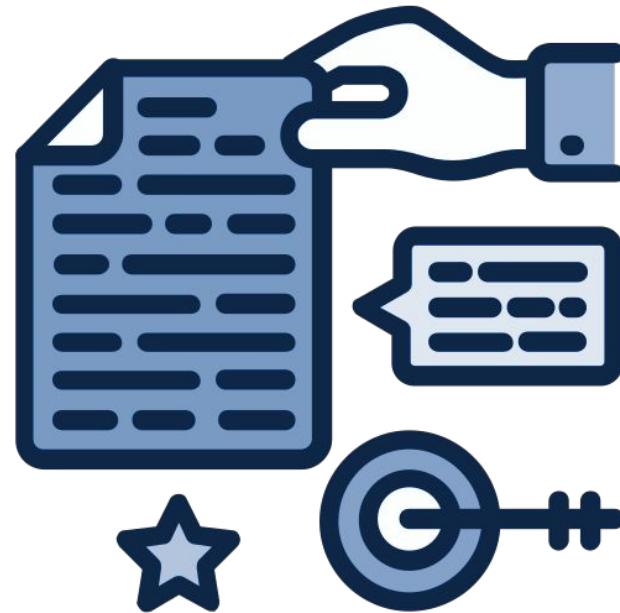


Más backend con Node.js

Seguridad

Passport.js nos permite crear y validar usuarios a través de un registro previo y el almacenamiento de contraseñas encriptadas en bb.dd dedicadas.

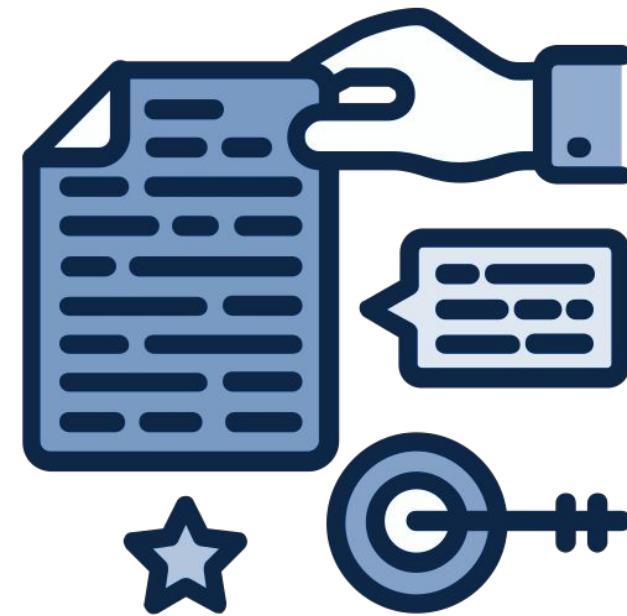
También nos da soporte para integrar validaciones de usuarios a través de cuentas de correo de: Facebook, Github, Google o Microsoft, entre otras opciones.



Más backend con Node.js

Seguridad

Y dentro de esta gama de validación y registro de usuarios, también tenemos a **OAuth 2.0**; otra opción completamente integrable a Node.js, la cual nos permite validar mediante el servicio de esta plataforma, el registro de usuarios genéricos o a través de cuentas existentes de las redes sociales más populares.

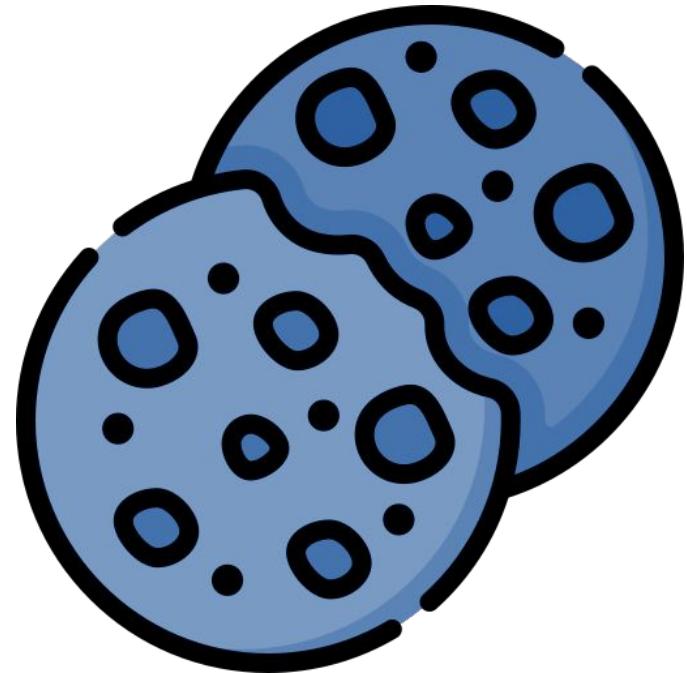


Más backend con Node.js

Seguridad

Incluso, todos los servicios anteriores relacionados a la seguridad, se pueden complementar con el manejo de sesiones y cookies.

Estas son opciones disponibles para trabajar desde Node.js, de una forma fácil y rápida de implementar.



Mensajería



Más backend con Node.js

Mensajería

Existen también varias opciones para integrar mecanismos de mensajería en aplicaciones Node.js. Por ejemplo, algunos servicios válidos y muy requeridos hoy, son aquellos que permiten el envío de SMS.

Entre los más populares encontramos a **Twilio**, **Nexmo** y **Plivo**. Pero son servicios de terceros que brindan sus propias APIs que debemos consumir.



Más backend con Node.js

Mensajería

En estos casos, tal vez sea más difícil probar algunos servicios, pero si tal vez conviene leer la documentación de ellos para entender a nivel teórico, cómo se deben implementar, saber qué opciones existen, y cómo cobran el uso de los servicios tercerizados.



A mayor conocimiento, más chances tendremos en el mercado laboral, y menos deberemos aprender al insertarnos en éste.

Más backend con Node.js

Mensajería

Otras plataformas que debemos conocer también, son las que nos permiten enviar mensajes de correo electrónico. Aquí encontramos al módulo **Nodemailer**, como el candidato ideal para estos procesos.

Este nos permite conectarnos a proveedores de servicios de correo electrónico vía (*SMTP, SendGrid, Gmail, entre otros tantos*), de forma sencilla y rápida.



Más backend con Node.js

Mensajería

Con un código simple y conciso, podremos implementar soluciones de envío de mensajes de validación de usuarios a un correo electrónico, para recuperar una contraseña, o para registrar por primera vez un usuario en una plataforma.

Nodemailer es muy fácil de implementar y usar, y se complementa muy bien con Passport.js u OAuth 2.0.

```
... Nodemailer

const nodemailer = require('nodemailer');

const transporter = nodemailer.createTransport({
  service: 'Gmail',
  auth: {
    user: 'tucorreo@gmail.com',
    pass: 'tupassword',
  },
});

const mailOptions = {
  from: 'tucorreo@gmail.com',
  to: 'destinatario@example.com',
  subject: '¡Hola!',
  text: 'Este es un mensaje de prueba.',
};

transporter.sendMail(mailOptions, (error, info) => {
  if (error) {
    console.error('Error al enviar el correo:', error);
  } else {
    console.log('Correo enviado:', info.response);
  }
});
```



Servicios

Más backend con Node.js

Servicios

Puedes también generar experiencia creando algunos servicios rápidos para los usuarios.

Entre ellos podemos mencionar a algunos populares, como ser, un acortador de URL's, un generador de códigos QR, o un generador de Token numérico como los de las aplicaciones de home banking.



Más backend con Node.js

Acortador de URLs

Un acortador de URL's es un servicio fácil y práctico de realizar, recibiendo una URL extensa como parámetro, almacenando la misma en una bb.dd, y retornando caracteres alfanuméricos creados con **Crypto.randomUUID()**.

```
... URL Shortener ...  
  
const URLShortener = Crypt.randomUUID();  
  
const shortenerFirstPart = URLShortener.split("-");  
  
console.log(shortenerFirstPart[0]);
```

Puedes tomar la primera parte del UUID generado, y entregarlo como el acortador de URL. Luego generas un segundo Endpoint que reciba el token UUID y retorne la URL vinculada a este.



Más backend con Node.js

Generador de códigos QR

El módulo **qrcode** facilita la generación de códigos QR utilizando Node.js. Al instalarlo, solo debemos enviarle un texto específico, el cual puede provenir de un String simple, una URL, una estructura de datos JSON, entre otras.

A través del texto recibido, generamos una URL que retorne una imagen QR la cual tendrá embebida la información en cuestión.



The screenshot shows a terminal window with a dark background. At the top right, it says "QR Code". Below that is a snippet of Node.js code:

```
const qr = require('qrcode');

// Generar un código QR a partir de un texto
qr.toDataURL('Hola, mundo!', (err, url) => {
  if (err) {
    console.error('Error al generar el código QR:', err);
    return;
  }

  console.log('URL del código QR:', url);
});
```



Más backend con Node.js

Generador de códigos numéricos

El generador de códigos numéricos es ideal para demostrar la misma interacción que utiliza una aplicación móvil actual la cual complementa a los home banking.

El código numérico se podrá mostrar en una aplicación frontend, y a través de un segundo endpoint, validar su efectividad. Incluso puedes complementarlo con Nodemailer para que envíe el código a un correo electrónico específico.

```
...  
Token Number  
  
const tokenNumber = parseInt(Math.random() * 1000000);  
  
console.log(tokenNumber);
```

Manejo de socket

Más backend con Node.js

Sockets

El manejo de sockets con Node.js es especialmente útil en aplicaciones en tiempo real que requieren comunicación bidireccional entre el servidor y el cliente. Socket.io es el módulo más popular en Node.js, y podremos crear:

- aplicaciones colaborativas
- Videojuegos multiplayer
- Aplicaciones de seguimiento en tiempo real
- Aplicaciones de streaming en tiempo real



Más backend con Node.js

Sockets

Dentro de los ejemplos que más se utiliza Socket.io, es para la creación de un chat en tiempo real.

Esta tecnología es aplicable en terrenos actuales como los de ChatGPT, WhatsApp, el seguimiento de vehículos en tiempo real, entre otros.

No debemos dominarla a la perfección, pero sí entender cómo funciona y experimentarla con un ejemplo simple.



Contenedores



Más backend con Node.js

Contenedores

Los contenedores son una forma de encapsular y distribuir aplicaciones junto con todas sus dependencias en un entorno aislado y portátil.

A diferencia de las máquinas virtuales, los contenedores no requieren un sistema operativo completo, lo que los hace más livianos y rápidos de implementar.



Más backend con Node.js

Contenedores

Kubernetes es una plataforma de orquestación de contenedores de código abierto ampliamente utilizada para automatizar y gestionar la implementación, escalado y administración de aplicaciones en contenedores. Esta plataforma distribuye y equilibra automáticamente las cargas de trabajo en función de las necesidades de la aplicación, lo que permite una mayor disponibilidad, escalabilidad y tolerancia a fallos.



Más backend con Node.js

Contenedores

Docker es una plataforma de virtualización de contenedores que simplifica el desarrollo y despliegue de aplicaciones.

Empaque las dependencias de una aplicación en un contenedor independiente y liviano, garantizando la portabilidad y reproducibilidad del entorno.

Así podremos ejecutar aplicaciones de manera consistente en diferentes entornos, desde el desarrollo hasta la producción, logrando una mayor eficiencia en el uso de recursos.



Por dónde seguir (Bases de datos)

Más backend con Bases de Datos

De la mano de MySQL hemos recorrido un gran camino con este, el cual no solo nos deja un poder de conocimiento importante, sino que lo podemos aprovechar para utilizar cualquier otro motor alternativo que maneje también el lenguaje SQL, y así conseguir experiencia con otros motores.



Más backend con Bases de Datos

Para terminar de complementar todo lo aprendido de la mano de MySQL, te recomendamos agregar a tus conocimientos:

- Funciones almacenadas
- Procedimientos almacenados
- Usuarios y Roles



Más backend con Bases de Datos

Funciones almacenadas

Recordemos que, una función almacenada, es un bloque de código reutilizable que realiza una tarea específica y puede ser invocado desde una consulta para obtener un resultado.

Aprenderlas nos ayudará a conocer el lenguaje SQL enfocado en el ámbito de la programación.



Más backend con Bases de Datos

Usuarios y Roles

La administración de usuarios y roles en MySQL es el proceso de asignar y gestionar los permisos y privilegios de los usuarios, permitiendo un control preciso sobre el acceso y las acciones que pueden realizar en la base de datos.

Es totalmente complementable con Node.js y con los diferentes perfiles de usuarios pensados a nivel seguridad.



Plataformas Cloud

Plataformas Cloud

PaaS: Plataforma como servicio (PaaS) es un modelo en la nube que ofrece a los desarrolladores un entorno completo para desarrollar, probar y desplegar aplicaciones sin tener que preocuparse por la infraestructura subyacente. (*el lugar donde publicamos nuestra aplicación backend*)

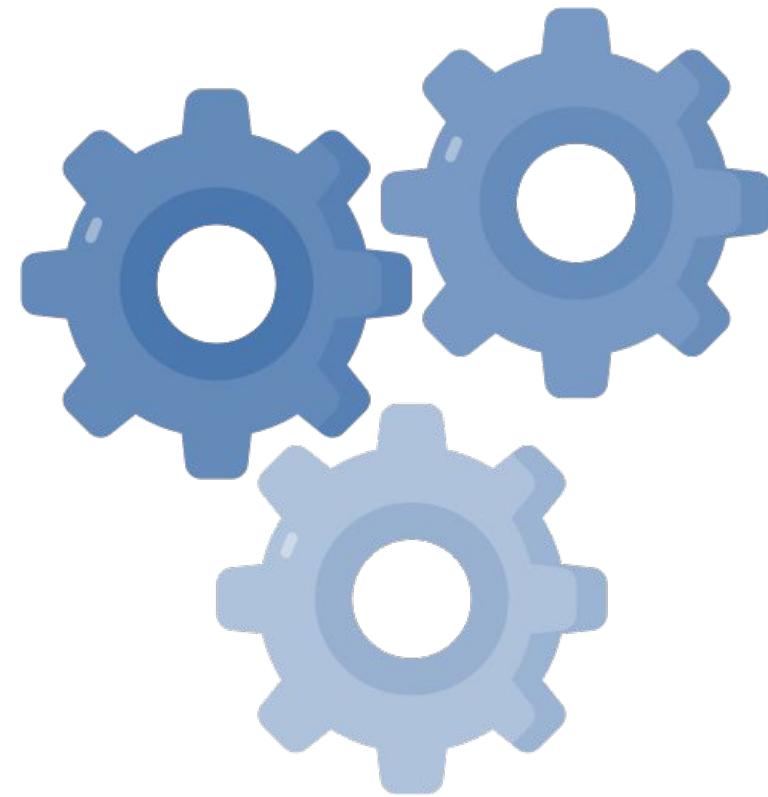
SaaS: Software como servicio (SaaS) es un modelo de entrega de software en la nube en el que las aplicaciones están alojadas y gestionadas por un proveedor de servicios, y los usuarios acceden a ellas a través de la web. (*Google Drive, OneDrive, Dropbox, entre otras plataformas de uso frecuente*)

IaaS: Infraestructura como servicio (IaaS) es un modelo en la nube que proporciona a los usuarios una infraestructura informática completa, incluyendo servidores virtuales, almacenamiento y redes, permitiendo a los usuarios desplegar y gestionar sus propias aplicaciones y sistemas operativos.

Plataformas Cloud

No podremos conocer en detalle todos los servicios PaaS, IaaS y SaaS del mercado, pero sí es clave poder diferenciar a cada uno de ellos fácilmente, entendiendo qué son y qué servicios podemos utilizar o crear para estos.

Es muy amplio el mercado de servicios de backend actual, y es muy importante nunca dejarnos estar, en relación a seguir adquiriendo conocimientos y seguir practicando, para estar listas a la primera oportunidad de inserción laboral que nos surja.



Gracias por elegirnos para tu formación profesional

"El aprendizaje constante y la pasión por la programación backend te llevarán a alcanzar nuevos horizontes en el emocionante mundo de la tecnología."



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*