CURSO DE PROGRAMACIÓN FULL STACK

BASES DE DATOS CON MYSQL

```
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (1,'Bruce Banner', 'Hulk', 160, '600 mil', 75,98,1962,'Fisico Nuclear', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (2,'Tony Stark', 'Iron man', 170, '200 mil', 70,123,1963,'Inventor Industrial',
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (3,'Thor Odinson ', 'Thor', 145, 'infinita', 100,235,1962,'Rey de Asgard', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (4,' Wanda Maximoff', 'Bruja Escarlata', 170, '100 mil', 90,345,1964,'Bruja',
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (5,' Carol Danvers', 'Capitana Marvel', 157, '250 mil', 85,128,1968,'Oficial d
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (6,'Thanos', 'Thanos', 170, 'infinita', 40,306,1973,'Adorador de la muerte', 1
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (7,'Peter Parker', 'Spiderman', 165, '25 mil', 80,74,1962,'Fotografo', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (8,'Steve Rogers', 'Capitan America', 145, '600', 45,60,1941,'Oficial Federa
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (9,'Bobby Drake', 'Iceman', 140, '2 mil', 64,122,1963,'Contador', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (10,'Barry Allen', 'Flash', 160, '10 mil', 129,168,1956,'Cientifico Forense', 2
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (11,'Bruce Wayne', 'Batman', 170, '500', 32,47,1939,'Hombre de Negocios
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (12,'Clarck Kent', 'Superman', 165, 'infinita', 120,182,1948,'Reportero', 2);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (13,'Diana Prince', 'Mujer Maravilla', 160, 'infinita', 95,127,1949,'Princesa C
```

MySQL®

EGG

Objetivos de la Guía

En esta guía aprenderás:

- Conocer el nuevo IDE.
- Comprender qué es una base de datos y cómo se conforma.
- Poder crear, eliminar y modificar una base de datos.
- Comprender el concepto de filas y columnas en las bases de datos.
- Poder realizar consultas a la base de datos y extraer información específica.
- Entender las relaciones entre tablas y los elementos que involucran.
- Realizar consultas multitablas.

GUÍA DE BASE DE DATOS

Hasta el momento hemos realizado algoritmos en Java que no almacenaban información. Es decir, cada vez que ejecutamos nuestros algoritmos, es como si los ejecutásemos por primera vez. El empleo de una base de datos nos permite almacenar la información de nuestros programas en cada ejecución. He aquí la importancia del uso de las bases de datos en la programación.

¿QUÉ ES UNA BASE DE DATOS?

Una base de datos es una colección organizada de datos típicamente almacenados electrónicamente en un sistema de computadora.

La información estructurada o datos se modelan típicamente en **filas** y **columnas** en una serie de **tablas.** Esto permite que el procesamiento y la consulta de datos sean eficientes. De este modo, se puede acceder, administrar, modificar, actualizar, controlar y organizar fácilmente los datos.

Las bases de datos requieren de un **lenguaje de consulta** para recibir instrucciones sobre cómo operar. Un lenguaje de consulta no es un lenguaje de programación. Es importante que distingamos las diferencias entre ambos, seguramente al avanzar sobre la guía lo comprendas mejor. La mayoría de las bases de datos utilizan lenguaje de consulta estructurado (SQL) para operar sobre bases de datos.

¿CUÁL ES LA IMPORTANCIA DEL USO DE UNA BASE DE DATOS?

Las bases de datos se consideran uno de los mayores aportes que ha dado la informática a las empresas. En una empresa, la administración de muchos datos e información es parte del día a día. Por lo que, lograr una gestión de bases de datos adecuada, resulta en una mayor eficacia y productividad.

Esto se debe a las principales utilidades que ofrece una base de datos a una empresa:

- Agrupar y almacenar todos los datos de la empresa en un único lugar.
- Facilitar que se compartan los datos entre los diferentes miembros de la empresa.
- Evitar la redundancia y mejorar la organización de la actividad.
- Visualizar los datos de un cliente o potencial: interacciones, ventas, datos de contacto.

Además, también, desde su creación, las bases de datos han significado un gran soporte para la organización en el desarrollo de software. La base de datos es la estructura fundamental para la administración eficiente de datos en el mundo de la informática.

La correcta gestión de estas bases de datos brinda a los sistemas una eficacia palpable. Es una mejoría que se traduce en rapidez de procesamiento, seguridad de datos y calidad de uso de los recursos.

Como método común de almacenamiento, es vital el manejo integral del área, pasando por la administración del sistema gestor, hasta llegar al desarrollo de softwares, los conceptos y tecnología asociados. Es por esto que existen ramas de la Programación especializadas en la ciencia de datos como, por ejemplo: Data Analytics y Data Science.

¿QUÉ ES UN SISTEMA DE GESTIÓN DE BASE DE DATOS (DBMS)?

Un sistema de gestión de base de datos (DMBS) es un software con el que se administra y gestiona la información que incluye una base de datos. Un DBMS permite administrar todo acceso a la base de datos, pues tienen el objetivo de servir de interfaz entre esta, el usuario y las aplicaciones.

Entre sus funciones se encuentra la de permitir almacenar la información y modificar datos. Asimismo, el gestor de base de datos también se ocupa de realizar consultas de una forma sistemática permitiendo a los usuarios recuperar, actualizar y administrar cómo se organiza y optimiza la información.

Algunos ejemplos de software de bases de datos o DBMS populares incluyen MySQL, Microsoft Access, Microsoft SQL Server, FileMaker Pro, Oracle Database y dBASE.

En este curso vamos a utilizar el software de bases de datos MySQL.

¿QUÉ ES UNA BASE DE DATOS DE MYSQL?

MySQL es un sistema de gestión de bases de datos relacionales de código abierto basado en **SQL**. Fue diseñado y optimizado para aplicaciones web y puede ejecutarse en cualquier plataforma. A medida que surgían nuevos y diferentes requisitos con Internet, MySQL se convirtió en la plataforma elegida por los desarrolladores web y las aplicaciones basadas en la web. Debido a que está diseñada para procesar millones de consultas y miles de transacciones. La flexibilidad bajo demanda es la característica principal de MySQL.

¿QUÉ ES UNA TABLA DE BASE DE DATOS?

Una tabla de base de datos es similar en apariencia a una hoja de cálculo en cuanto a que los datos se almacenan en filas y columnas.

Para aprovechar al máximo la flexibilidad de una base de datos, los datos deben organizarse en tablas para que no se produzcan redundancias. Por ejemplo, si se quiere almacenar información sobre los empleados, cada empleado debe especificarse solo una vez en la tabla que está configurada para los datos de los empleados. En tanto que, los datos sobre los productos se almacenarán en su propia tabla y, finalmente, los datos sobre las sucursales se almacenarán en otra tabla aparte. Este proceso se denomina normalización y lo abordaremos en más profundidad durante esta guía.

Componentes de una tabla

Cada fila de una tabla se denomina registro. En los registros se almacena información. Cada registro está formado por uno o varios campos. Los campos equivalen a las columnas de la tabla. Por ejemplo, se puede tener una tabla llamada "Empleados" donde cada registro (fila) contiene información sobre un empleado distinto y cada campo (columna) contiene otro tipo de información como nombre, apellido, dirección, etc. Los campos deben designarse como un determinado tipo de datos, ya sea texto, fecha u hora, número o algún otro tipo. Veamos una tabla de Empleado:

Empleado		
Nombre	Apellido	Edad
Agustín	Cocco	24
Martin	Bullón	21
Mariano	Fernández	18
Jerónimo	Gómez	23

Nombre de la tabla.

Nombre de las columnas: *Nombre, Apellido y Edad*

Fila / Registro:
Cada una de las
Filas tiene un empleado
distinto.

¿CUÁL ES LA DIFERENCIA ENTRE UNA BASE DE DATOS Y UNA HOJA DE CÁLCULO?

Las bases de datos y las hojas de cálculo (como Microsoft Excel) son dos formas convenientes de almacenar información. Las principales diferencias podemos encontrarlas en:

- **Cómo se almacenan y manipulan los datos**. Las bases de datos emplean una unidad lógica que relaciona los datos de manera que hace más eficiente su uso y almacenamiento.
- **Quién puede acceder a los datos**. Las bases de datos permiten a múltiples usuarios al mismo tiempo acceder y consultar los datos de forma rápida y segura utilizando una lógica y un lenguaje altamente complejos.
- **Cuántos datos se pueden almacenar.** Las hojas de cálculo se diseñaron originalmente para un usuario, y sus características lo reflejan. Son muy buenas para un solo usuario o un pequeño número de usuarios que no necesitan manipular una gran cantidad de datos complicados. Las bases de datos, por otro lado, están diseñadas para contener colecciones mucho más grandes de información organizada, cantidades masivas en ocasiones.

¿POR QUÉ INTERESA USAR UNA BASE DE DATOS EN LUGAR DE UNA HOJA DE CÁLCULO ?

- **Mayor independencia**. Los datos son independientes de las aplicaciones que los usan, así como de los usuarios.
- **Mayor disponibilidad**. Se facilita el acceso a los datos desde contextos, aplicaciones y medios distintos, haciéndolos útiles para un mayor número de usuarios.
- **Mayor seguridad** (protección de los datos). Por ejemplo, resulta más fácil replicar una base de datos para mantener una copia de seguridad que hacerlo con un conjunto de ficheros almacenados de forma no estructurada. Además, al estar centralizado el acceso a los datos, existe una verdadera sincronización de todo el trabajo que se haya podido hacer sobre estos (modificaciones), con lo que esa copia de seguridad servirá a todos los usuarios.
- **Menor redundancia**. Un mismo dato no se encuentra almacenado en múltiples archivos o con múltiples esquemas distintos, sino en una única instancia en la base de datos. Esto redunda en menor volumen de datos y mayor rapidez de acceso.
- **Mayor eficiencia en la captura, codificación y entrada de datos.**

CLASIFICACIÓN DE LAS BASES DE DATOS

Hay muchos tipos diferentes de bases de datos. La mejor base de datos para una organización específica depende de cómo la organización pretende utilizar los datos.

- **Bases de datos relacionales:** Los elementos de una base de datos relacional se organizan como un conjunto de tablas con columnas y filas. La tecnología de base de datos relacional proporciona la manera más eficiente y flexible de acceder a información estructurada. En la actualidad se usa de forma mayoritaria las bases de **datos relacionales.**
- **Nota:** este es el tipo de base de datos que vamos a trabajar y sobre el que vamos a profundizar.
- **Bases de datos orientadas a objetos:** La información en una base de datos orientada a objetos se representa en forma de objetos, como en la programación orientada a objetos. Además, se almacenan datos complejos y relaciones entre datos directamente, sin asignar filas, ni columnas. No es un tipo de bases de datos muy utilizado en la actualidad debido a que no hay criterios claros de estandarización y existe poca documentación sobre los proyectos en los que se han implementado.
- **Bases de datos NoSQL:** Una NoSQL (*Not Only SQL*), o una base de datos no relacional, permite que los datos no estructurados y semiestructurados se almacenen y manipulen, a diferencia de una base de datos relacional, que define cómo deben componerse todos los datos insertados en la base de datos. Las bases de datos NoSQL se hicieron populares a medida que las aplicaciones web se hacían más comunes y complejas.

BASE DE DATOS RELACIONALES

Las bases de datos relacionales se basan en el **modelo relacional**, una forma intuitiva y directa de representar datos en tablas. En una base de datos relacional, cada fila de la tabla es un registro único que posee un identificador específico llamado **clave primaria** o **llave primaria**.

Las columnas de una tabla se corresponden con los atributos de los datos, y cada registro o fila, generalmente, tiene un valor para cada atributo, lo que facilita el establecimiento de las relaciones entre los datos.

¿QUÉ ES UN MODELO DE BASE DE DATOS?

Un modelo de base de datos es la estructura lógica que adopta la base de base datos, incluyendo las relaciones y limitaciones que determinan cómo se almacenan y organizan y cómo se accede a los datos. Asimismo, un modelo de base de datos también define qué tipo de operaciones se pueden realizar con los datos, es decir, que también determina cómo se manipulan los mismos, proporcionando también la base sobre la que se diseña el lenguaje de consultas.

En general, prácticamente todos los modelos de base de datos pueden representarse a través de un diagrama de base de datos (en esta guía veremos algunos).

MANOS A LA OBRA!

Actividad 1

Hemos abordado temas centrales y pensamos que es oportuno analizar algunas definiciones. Por ello, te proponemos respondas las siguientes preguntas a fin de asentar mejor algunos conceptos clave:

1) ¿Qué es una base de datos?

   a. Un conjunto de datos organizados de manera aleatoria.
   b. Una colección organizada de datos no estructurados.
   c. Una tabla que posee filas y columnas que estructuran la información.
   d. Una colección organizada de información estructurada.

2) ¿Qué es una tabla?

   a. Las tablas constituyen la base de datos y organizan la información.
   b. Una tabla es una estructura organizada para almacenar información.
   c. Una tabla posee filas y columnas, en apariencia, es similar a una hoja de cálculo.
   d. Todas son correctas.

3) ¿Cuál de estos no es un tipo de base de datos?

   a. Base de datos no relacional
   b. Base de datos relacional
   c. Base de datos imperativa
   d. Base de datos orientada a objetos

4) ¿Cuál de estas SI es una ventaja al trabajar el almacenamiento de los datos con bases de datos en lugar de hojas de cálculo?

   a. Los datos están más protegidos.
   b. Hay mayor repetición de datos.
   c. Existe mayor eficiencia en la codificación, pero no en la captura.
   d. Los datos dependen de las aplicaciones que los usan.

5) En las bases de datos relacionales existe un elemento clave que identifica a cada registro o fila y la hace única, ¿cómo se llama ese elemento?

a. Clave única
b. Clave foránea
c. Llave primaria
d. Llave maestra

Revisemos lo aprendido hasta aquí

- Comprender qué es una base de datos, características y cómo se conforma.

MODELO RELACIONAL

El modelo relacional, para el modelado y la gestión de bases de datos, es un modelo de datos basado en la lógica de predicados y en la teoría de conjuntos. Este modelo de bases de datos fue creado por un científico informático inglés llamado Edgar Frank "Ted" Codd a finales de los 60'. Su idea fundamental es el uso de *relaciones*.

Estas relaciones podrían considerarse en forma lógica como **conjuntos de datos.** Estos conjuntos de datos, se denominan **tuplas.** Pese a que esta es la teoría de las bases de datos relacionales creadas por Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar, pensando en cada relación como una **tabla** que está compuesta por registros (cada fila de la tabla sería un registro o *"tupla"*) y columnas (también llamadas *"campos"*).

Es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente.

Este modelo está basado en que todos los datos están almacenados en **tablas** (también llamadas entidades o relaciones) y cada una de estas es un conjunto de datos, por tanto una base de datos es un conjunto de relaciones.

Las tablas están formadas por **filas**, también llamadas **tuplas**, donde se describen los elementos que configuran la tabla (es decir, los elementos de la relación establecida por la tabla), **columnas** o **campos**, con los atributos y valores correspondientes, y el **dominio**, concepto que agrupa a todos los valores que pueden figurar en cada columna.

NORMALIZACIÓN EN EL MODELO RELACIONAL

La normalización de bases de datos es un proceso que consiste en designar y aplicar una serie de reglas a las relaciones obtenidas tras el paso del modelo entidad-relación al modelo relacional con objeto de minimizar la redundancia de datos, facilitando su gestión posterior.

Las bases de datos relacionales se normalizan para:

- Minimizar la redundancia de los datos.
- Disminuir problemas de actualización de los datos en las tablas.
- Proteger la integridad de datos.

CARACTERÍSTICAS DEL MODELO RELACIONAL

En el modelo relacional es frecuente llamar relación a una tabla; para que una tabla sea considerada como una relación tiene que cumplir con lo siguiente:

- Cada tabla debe tener un nombre único
- Los datos de cualquier columna corresponden a un solo tipo de dato (por ejemplo cadena, entero, doble).
- Las columnas de una relación se conocen como atributos.
- El orden de los atributos no importa: los atributos no están ordenados.
- Cada atributo de la tabla solo puede tener un valor en cada tupla
- Cada atributo tiene un nombre único en cada tabla (aunque pueden coincidir en tablas distintas)
- Cada atributo tiene un dominio.
- El orden de las filas no importa
- No se permiten filas repetidas mediante la clave primaría.
- No existen 2 filas en la tabla que sean idénticas.
- Los valores de los atributos son atómicos: en cada tupla (fila), cada atributo (columna) toma un solo valor. Esto aporta a la normalización de la base de datos.

ELEMENTOS DEL MODELO RELACIONAL

ATRIBUTOS

Un atributo en el Modelo Relacional representa una propiedad que posee esa tabla y equivale al atributo del Modelo E-R (Entidad-Relación). Es decir, un atributo de una relación (tabla) es análogo al atributo de una clase entidad.

También se corresponde con la idea de campo o columna.

Es necesario volver a mencionar que los atributos de una misma tabla no pueden nombrarse de igual manera. Cada atributo debe poseer un nombre único.

Por ejemplo, la información de los estudiantes de un curso se representa mediante la tabla Estudiantes, y posee cuatro atributos (representadas en columnas) **id_estudiante**, **nombre, apellido y edad.**

DOMINIO

El dominio dentro de la estructura del Modelo Relacional es el **conjunto de valores que puede tomar un atributo**.

Estudiantes			
id_estudiante	**nombre**	**apellido**	**edad**
Valor numérico	Cadena de texto	Cadena de texto	Valor numérico
Valor numérico	Cadena de texto	Cadena de texto	Valor numérico
Valor numérico	Cadena de texto	Cadena de texto	Valor numérico

- Un dominio contiene todos los posibles valores que puede tomar un determinado atributo. Dos atributos distintos pueden tener el mismo dominio.
- Un domino es un conjunto finito de valores del **mismo tipo**. Distintos tipos de dominios son: enteros, cadenas de texto, fecha, etc.

TUPLAS

Las tuplas son las filas de una tabla que contiene valores para cada uno de los atributos (equivale a los registros). Siguiendo la misma ejemplificación de la tabla Estudiantes, un ejemplo podría ser: 34563, José, Martínez, 19.

Una tupla representa un objeto único. Los datos que posee dicha tupla se encuentran estructurados en una tabla. Esto resulta en que, cada tupla (o registro), esté compuesta por *campos* que contienen los datos y representan a una entidad.

CLAVES

Las claves son campos cuyo valor es único para cada tupla o registro. Identificamos dos tipos: primaria y foránea

Clave primaria

Se denomina **clave primaria** o **identificador único** o **llave principal** al atributo que identifica a cada instancia de un registro.

Una base de datos relacional está diseñada para imponer la **exclusividad** de las claves primarias. Esto permite que haya **sólo una columna** con un valor de clave primaria en una tabla. Resultando en que no pueden existir dos instancias de un registro con el mismo valor de llave primaria.

Para mejorar el desempeño de la base de datos se recomienda utilizar claves primarias numéricas. Idealmente, se nombran “id” (abreviación de identificador) seguido por un guión bajo (_) y el nombre de la tabla. Por ejemplo: **id_estudiante.**

Idealmente las llaves primarias poseen un valor autogenerado y se sugiere no se elija llave primaria algún otro atributo de la tabla. Esto se debe a buenas prácticas de manipulación de los datos en bases de datos: la llave primaria debe ser un valor abstracto de la tabla en sí y su única función debe ser identificar una tupla, diferenciándola de otras.

estudiante			
id_estudiante	**nombre**	**apellido**	**edad**
12340321891	Agustín	Cocco	24
14334123546	Martin	Bullón	21
15445678145	Agustín	Cocco	18

En esta tabla “estudiante”, tenemos dos estudiantes con el mismo nombre y apellido, pero con distintos identificadores y distintas edades, por lo que no se consideran **datos duplicados**. Si nosotros nos basáramos en el nombre para evitar datos duplicados, no podríamos ingresar dos alumnos con el mismo nombre, esta es otra de las ventajas del identificador único o clave primaria.

CLAVE FORÁNEA

Una **clave foránea** o **llave foránea** es una columna de una tabla cuyos valores corresponden a los valores de la clave primaria de otra tabla. A veces, esto también se denomina clave de referencia.

Para asignarle un valor a un campo de llave foránea, se requiere de la existencia de una llave primaria con el mismo valor. Asimismo, la relación entre dos tablas coincide con la clave primaria en una de las tablas y con una clave foránea en la segunda tabla. Por ejemplo, si tenemos las tablas **profesor** y **curso**, para relacionarlas, tendríamos, en la tabla curso, una columna de clave foránea cuyo campo será el valor de la clave primaria de la tabla profesor. De esta manera se dice que un profesor específico pertenece a un curso específico.

Analicemos esto con ayuda del siguiente ejemplo:

Profesor			
id_profesor	Nombre	Apellido	Edad
1	Agustín	Cocco	24

Curso			
id_curso	Nombre	Costo	id_profesor
1	Curso de Programación	500	1

En este ejemplo la tabla Curso (tabla de la derecha), tiene una columna llamada *id_profesor*, esta columna es la que va a tener las **claves foráneas** y la que va mostrar que **hay una relación entre las dos tablas**. En este ejemplo, la columna *id_profesor* de la tabla curso, tiene el valor "1" que se corresponde con la llave primaria (*id_profesor*) de un registro de la tabla Profesor. Esto nos indicaría que Agustín es el Profesor del Curso de Programación.

MANOS A LA OBRA!

Actividad 2

Hemos abordado un montón de conceptos nuevos que te serán de mucha ayuda y utilidad para seguir avanzando en esta guía, ¿te parece si repasamos un poco con una actividad?

Para esta actividad deberás responder cada pregunta en base a las siguientes tablas creadas en Workbench:

Tabla casa

id_casa	calle	numeracion	es_departamento
163748	Avenida El Libertador	1465	1
NULL	NULL	NULL	NULL

Tabla usuario

id_usuario	nombre	apellido	id_casa
854267	Noelia	Devito	163748
NULL	NULL	NULL	NULL

1) Primero deberás localizar en ambas tablas los siguientes elementos:
   a. Llaves primarias
   b. Llave foránea
   c. Un campo que almacena datos de tipo carácter
   d. Los atributos de cada tabla
   e. Una tupla

2) Compara tus respuestas con las de tus compañeros: recuerda abordar cada duda que tengas, a partir de ellas es que fortalecemos el aprendizaje.

3) ¿Cuál es la función de una llave primaria? Selecciona la opción correcta:
   a. Una llave primaria identifica a cada registro imposibilitando la repetición de un mismo registro, y por defecto, la redundancia
   b. Una llave primaria, por buenas prácticas, debería ser un valor abstracto de la relación y su valor debería ser autogenerado
   c. Una llave primaria es un elemento clave en la relación entre tablas, ya que, para la existencia de una llave foránea, se requiere anteriormente, la de una llave primaria
   d. Todas las anteriores son correctas

Revisemos lo aprendido hasta aquí

- Comprender el concepto de filas y columnas en las bases de datos.
- Reconocer los elementos principales del modelo relacional.

RELACIONES

Uno de los aspectos fundamentales de las bases de datos relacionales son precisamente las relaciones. En pocas palabras, una “relación” es una asociación que se crea entre tablas, con el fin de vincularlas y garantizar la integridad referencial de sus datos.

Una relación es la abstracción de un conjunto de asociaciones que existen entre las tablas de dos tuplas, por ejemplo, existe una relación entre Película (tabla Película) y Director (tabla Director).

Para que una relación entre dos tablas exista, la tabla que deseas relacionar debe poseer una **clave primaria** o identificador único, mientras que la tabla donde estará el lado dependiente de la relación debe poseer una **clave foránea** o llave foránea de esa clave primaria.

- Las relaciones tienen sentido bidireccional.
- Las relaciones existen ya que las entidades representan aspectos del mundo real y en este mundo los componentes no están aislados, sino que se relacionan entre sí; es por esto es necesario que existan las relaciones entre las entidades.

TIPOS DE RELACIONES

Las bases de datos relacionales tienen diversos tipos de relaciones que podemos utilizar para vincular nuestras tablas.

Este *vínculo* va a depender de la cantidad de ocurrencias que tiene un registro o fila de una tabla dentro de otra tabla (esto se conoce como cardinalidad).

Veamos los tipos de vínculos o relaciones:

Relaciones uno a uno

Se presentan cuando *un registro* de una tabla **sólo** está relacionado con un registro de otra tabla, y viceversa.

Por ejemplo, supongamos que nuestros **empleados** deben almacenar su **información de contacto**. Para este caso, pudiésemos leer la relación de esta manera:

Un **empleado** tiene una sola **información de contacto**.

y

Una **información de contacto** pertenece a un solo **empleado**.

Dado que la **información de contacto** es la que depende principalmente del **empleado**, es en ella donde existirá la clave foránea para representar el vínculo.

Relaciones uno a muchos / muchos a uno

Esta relación es un poco más compleja que la anterior, así que vamos a usar las tablas **A** y **B** para explicarla.

Una relación de **uno a muchos** se presenta cuando *un registro* de la **tabla A** está relacionado con *ninguno o muchos registros* de **tabla B**, pero este *registro* en la **tabla B** solo está relacionado con *un registro* de la **tabla A**. Veamos un ejemplo de esto.

Supongamos que tenemos **ciudades** en las cuales viven nuestras **personas**, pero cada **persona** solo puede pertenecer a una **ciudad**. Para este caso, pudiésemos leer la relación de esta manera:

Una **ciudad** tiene muchas (o ningún) **personas.**

y

Una **persona** vive en una sola **ciudad.**

Dado que la **persona** es el que necesita de la **ciudad**, es en él donde existirá la clave foránea para representar el vínculo.

Relaciones muchos a muchos

Estas son las relaciones más complejas, se presentan cuando *muchos registros* de una tabla se relacionan con *muchos registros* de otra tabla. Vamos a verlo en un ejemplo.

Supongamos que nuestros **empleados** trabajan **turnos**. Por ejemplo, Juan trabaja en el turno mañana y de la noche, pero en el turno de la mañana trabajan Juan, Pedro y María.

Para este tipo de relación se crea una tabla intermedia conocida como **tabla asociativa**. Por convención, el nombre de esta tabla debe estar formado por el nombre de las tablas participantes (en singular y en orden alfabético) separados por un guion bajo (_). Esta tabla está compuesta por las claves primarias de las tablas que se relacionan con ella, así se logra que la relación sea de uno a muchos, en los extremos, de modo tal que la relación se lea:

Un **empleado** trabaja en muchos **turnos**

y

Un **turno** tiene muchos **empleados**.

La tabla recibirá las llaves primarias como llaves foráneas.

Supongamos que tenemos la siguiente relación. Tenemos dos tablas: ticket y producto. En un registro Ticket se verá reflejado el ticket de las compras del supermercado donde un cliente puede comprar varios productos. Los productos cargados en el ticket podrán ser visualizados a través de la tabla intermedia. Donde podemos ver que tenemos dos productos asociados al id_ticket de valor ***915***. Por lo tanto, el ticket 915 tiene registro de compra de **Pan (6714)** y **Agua (3452).** Sin embargo, al mismo tiempo un producto puede ser comprado por varios clientes. Esta es una relación muchos a muchos. Las tablas se verían así:

Producto	
ID	Nombre
3452	Agua
7811	Azúcar
6714	Pan

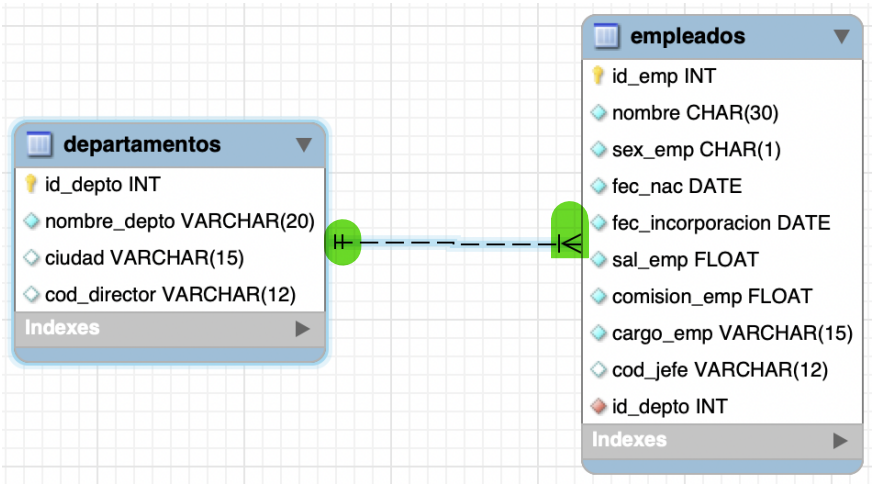
Ticket	
ID	PrecioTotal
915	2000
624	5500

Tabla intermedia

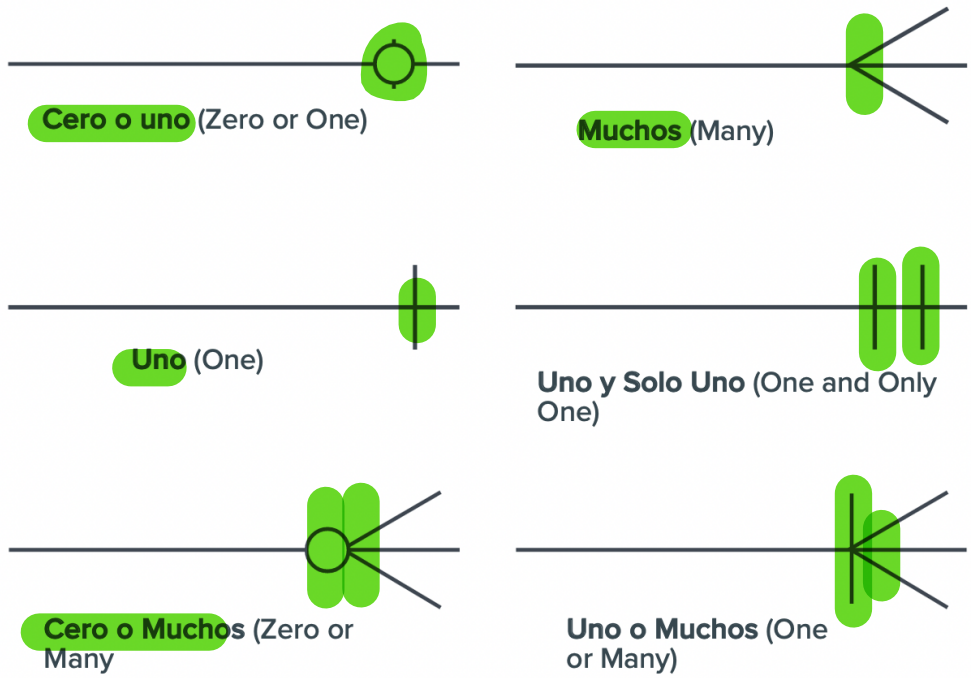
Producto_Ticket	
ID_Producto	ID_Ticket
3452	915
6714	915
7811	624

DIAGRAMAS ERD

Los **diagramas de relación de entidades** (ERD) son representaciones visuales de bases de datos que muestran cómo los elementos dentro de una base de datos están relacionados entre sí. Un ERD se compone de dos tipos de objetos: **entidades** y **relaciones**. Las entidades van a ser lo que nosotros conocemos como tablas, y las relaciones tienen finales de línea especiales llamados cardinalidades que describen cómo dos elementos de la base de datos interactúan entre sí.



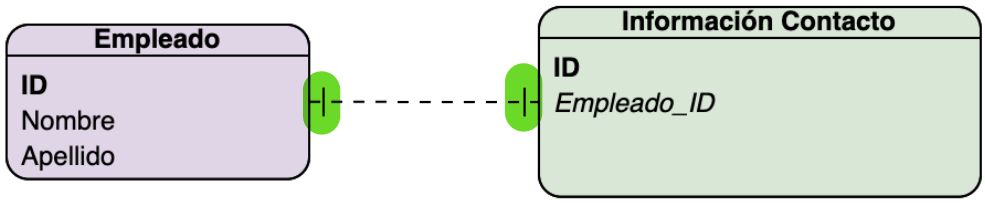
Estos diagramas nos van a servir para representar las relaciones previamente mencionadas, para ello existen las siguientes líneas que unen las tablas entre sí:



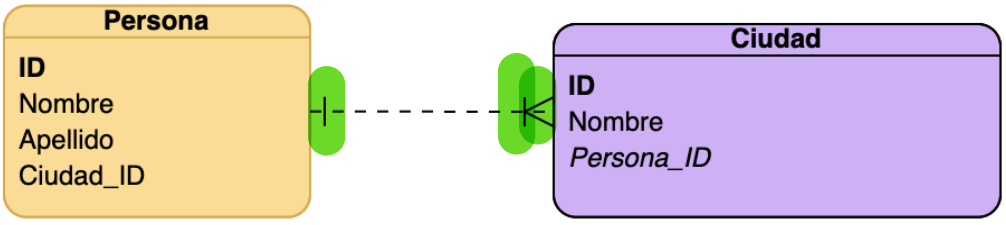
¿NECESITAS UN **EJEMPLO**?

Teniendo en cuenta las líneas que se usan para cada tipo de relación vamos a ver cómo serían cada una de las relaciones:

Relaciones uno a uno



Relaciones uno a muchos / muchos a uno



Relaciones muchos a muchos

MANOS A LA OBRA!

Actividad 3

Marca verdadero o falso. Justifica tus respuestas. Luego, compara tus respuestas con tus compañeros de mesa.

- Para que exista una relación se requiere de una llave foránea. ✓
- Pueden existir relaciones entre tablas de muchos a muchos. ✓
- En un diagrama ERD con una llave dorada se identifica la llave foránea. ✗
- Existen cuatro tipos de relaciones entre tablas. ✗

Revisemos lo aprendido hasta aquí

- Relaciones entre tablas y los elementos que hacen posible este vínculo.

¿QUÉ ES EL SOFTWARE DE BASE DE DATOS?

Para poder trabajar con base de datos, tablas, sus columnas, filas, relaciones, con el modelo relacional, etc. Tenemos que utilizar un **software de base de datos**.

El software de base de datos se utiliza para crear, editar y mantener archivos y registros de bases de datos, lo que facilita la creación de archivos y registros, la entrada de datos, la edición de datos, las actualizaciones y los informes. El software también se encarga del almacenamiento de datos, las copias de seguridad y los informes, el control de acceso múltiple y la seguridad. La sólida seguridad de las bases de datos es especialmente importante hoy en día, ya que el robo de información se vuelve más frecuente. En ocasiones, el software de base de datos también se denomina "sistema de administración de bases de datos" (DBMS).

El software de base de datos simplifica la gestión de la información al permitirles a los usuarios almacenar datos en una forma estructurada y luego, acceder a ellos. Por lo general, tiene una interfaz gráfica para ayudar a crear y administrar los datos y, en algunos casos, los usuarios pueden crear sus propias bases de datos mediante el software de base de datos.

LENGUAJE DE CONSULTA ESTRUCTURADO SQL

SQL es un acrónimo en inglés para **Structured Query Language,** un **Lenguaje de Consulta Estructurado**. Un tipo de **lenguaje de programación** que te permite acceder, manipular y descargar datos de una base de datos mediante comandos, mejor conocido como consultas (Querys).

Tiene capacidad de hacer cálculos avanzados y álgebra. Es utilizado en la mayoría de empresas que almacenan datos en una base de datos. Ha sido y sigue siendo el lenguaje de programación más usado para bases de datos relacionales.

El lenguaje SQL también se usa para controlar el acceso a datos y para la creación y modificación de esquemas de Base de datos. SQL utiliza los términos tabla, fila y columna para los términos relación, tupla y atributo del modelo relacional formal, respectivamente. Por lo tanto, es posible utilizar todos estos términos indistintamente.

Existen dos tipos de comandos SQL:

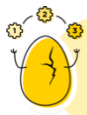
1. **Lenguaje de Definición de Datos (DDL)**: permite crear y definir nuevas bases de datos, campos e índices.
    - **CREATE:** Crea nuevas tablas, campos e índices.
    - **DROP:** Elimina tablas e índices.
    - **ALTER:** Modifica las tablas agregando campos o cambiando la definición de los campos.
2. **Lenguaje de Manipulación de Datos (DML):** permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.
    - **SELECT:** Consulta registros de la base de datos que satisfagan un criterio determinado.
    - **INSERT:** Carga lotes de datos en la base de datos en una única operación.
    - **UPDATE:** Modifica los valores de los campos y registros especificados.
    - **DELETE:** Elimina registros de una tabla de una base de datos.

SCRIPTS

Los scripts son fragmentos de código. Podremos encontrar scripts de distintos lenguajes y con distintos objetivos. Cuando hablamos de scripts de bases de datos, éstos contienen consultas específicas a la base de datos. Se utilizan para compartir consultas con otras personas. En general encontraremos dos funciones principales:

Scripts para compartir bases de datos. Al ejecutar este tipo de scripts podremos obtener una misma base de datos que otra persona.

Scripts para compartir consultas a una base de datos específica. Al ejecutar este tipo de scripts podremos obtener consultas. Pueden ser utilizadas para guardar en nuestros ordenadores consultas complejas que se utilizan a diario.



En este módulo utilizaremos mucho los scripts para realizar ejercitación. Encontrarás más material al respecto conforme avanzamos en la guía.

CONSULTAS DE DEFINICIÓN DE DATOS (DDL):

Estas son las consultas que vamos a escribir en nuestro software de base de datos para crear, actualizar, borrar, acceder y manipular información de nuestra base de datos.

CONSULTAS PARA CREAR

1. CREATE DATABASE

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nombre_base_datos
```

- Esta sentencia sirve para crear una base de datos con un nombre específico.
- Para poder crear una base de datos, el usuario que la crea debe tener privilegios de creación asignados.
- IF NOT EXISTS significa: SI NO EXISTE, por lo tanto, esto es útil para validar que la base de datos sea creada en caso de que no exista, si la base de datos existe y se ejecuta esta sentencia, se genera un error.

2. CREATE TABLE

CREATE SCHEMA o CREATE DATABASE son sinónimos.

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nombre_de_tabla (
campo1 tipoDato [NULL/NOT NULL] | CHECK (expresiónLógica) | [
DEFAULT expresiónConstante],
campo2 tipoDato [NULL/NOT NULL] | CHECK (expresiónLógica) | [
DEFAULT expresiónConstante ],
campo-N,
PRIMARY KEY(campo_llave),
FOREIGN KEY (campo_llave) REFERENCES tabla2 (campo_llave-tabla2))
```

Ligaduras

Tipo: Integridad de Dominio o Columna

Especifica un conjunto de valores que son válidos a ingresar sobre una columna específica para una tabla de la base de datos. Esta integridad se verifica a través de una la validación de los valores de datos que se ingresan y el tipo de los datos a introducir (numérico, alfanumérico, alfabético, etc.).

- *DEFAULT:* Esta restricción asigna un valor específico a una columna cuando el valor para ello no haya sido explícitamente proporcionado para tal columna en una sentencia "INSERT" o de adición de un nuevo registro en la tabla.
- *CHECK:* Específica los valores de datos que el DBMS acepta le sean ingresados para una columna.
- *REFERENCES:* Específica los valores de datos que el DBMS acepta le sean ingresados para una columna.

Tipo: Integridad de Entidad o Tabla

Específica que, en una tabla o entidad, todas sus filas tengan un identificador único que diferencie a una fila de otra y también que se establezcan columnas cuyo contenido es un valor único que las hace llaves candidatas para un futuro como, por ejemplo: número de cédula, número de seguro social o cuenta de email.

- *PRIMARY KEY:* Este tipo de restricción se aplica a todas las filas permitiendo que exista un identificador, que se conoce como llave primaria y que se asegura que los usuarios no introduzcan valores duplicados. Además, asegura que se cree un índice para mejorar el desempeño. Los valores nulos no están permitidos para este tipo de restricción.
- *UNIQUE:* Con esta restricción se previene la duplicación de valores en columnas que tienen valor único y que no son llave primaria pero que pueden ser una llave alternativa o candidata para el futuro. Asegura que se cree (Por parte del DBMS) un índice para mejorar el desempeño. Y al igual que las llaves primarias, no se le está permitido que se introduzcan valores nulos.

Tipo: Integridad Referencial

La Integridad Referencial asegura que las relaciones que existe entre llave primaria (en la tabla referenciada) y la llave foránea (en las tablas referenciantes) serán siempre mantenidas. Una fila o registro en la tabla referenciada (tabla donde reside la llave primaria) no puede ser borrada o su llave primaria cambiada si existe una fila o registro con una llave foránea (en la tabla referenciante) que se refiere a esa llave primaria.

- *FOREIGN KEY:* En esta restricción se define una llave foránea, una columna o combinación de columnas en las cuales su valor debe corresponder al valor de la llave primaria en la misma u en otra tabla.

CONSULTAS PARA ACTUALIZAR Y BORRAR

3. DROP TABLE

La sentencia para eliminar una tabla y por ende todos los objetos asociados con esa tabla es DROP TABLE, donde r es el nombre de una tabla existente.

```
DROP TABLE r
```

4. ALTER TABLE

Después que una tabla ha sido utilizada durante algún tiempo, los usuarios suelen descubrir que desean almacenar información adicional con respecto a las tablas. La sentencia ALTER TABLE se utiliza sobre tablas que ya poseen desde cientos a miles de filas por ser tablas de un sistema de software.

```
ALTER TABLE nombre_tabla acción
```

Siendo acción una de las siguientes:

- RENAME TO nuevo_nombre
- ADD [COLUMN] nombre_atributo definición_atributo
- DROP [COLUMN] nombre_atributo
- MODIFY nombre_atributo definición_atributo
- CHANGE nombre_atributo nuevo_nombre nueva_definición
- ALTER COLUMN nombre_atributo nuevo_nombre nueva_definición

Los cambios que se pueden realizar con la sentencia SQL ALTER TABLE son:

- Añadir una definición de columna a una tabla. Puede crearse con valores nulos o con valores.
- Eliminar una columna de la tabla. Pero antes de su eliminación deben ser eliminados por ALTER TABLE todas las restricciones que estén definidas sobre esta columna.
- Eliminar la definición de: llave primaria, foránea o restricciones de ligaduras de integridad (check), existentes para una tabla. Esta acción no elimina a la columna con sus valores, ella permanece tal cual como está, solo se elimina su definición.
- Definir una llave primaria para una tabla. La columna(s) a la cual se le dará esta responsabilidad debe contener previamente valores únicos por fila.
- Definir una nueva llave foránea para una tabla. La columna a definir como llave foránea debe contener previamente valores que corresponden a la llave primaria de otra tabla.

MANOS A LA OBRA!

Actividad 4

Para la realización de los ejercicios que se describen a continuación, es necesario descargar el archivo **scriptsBD.zip** que contiene algunos scripts con las bases de datos sobre las cuales se va a trabajar. En cada ejercicio se indica el nombre del script que se debe utilizar. Para abrir y ejecutar los scripts van a encontrar un pdf de cómo hacerlo en Moodle, con el nombre de **Tutorial Scripts SQL.**

1. ¡Ahora ejecutaremos un script y crearemos una base de datos por primera vez! Primero, deberemos abrir el script llamado “superhéroes.sql” y ejecutarlo de modo tal que se cree la base de datos y todas sus tablas.

2. Analiza el script detenidamente, línea por línea, ayudándote con la guía. Intenta comprender qué pasa en cada sentencia del script. Luego contesta:

2.1. ¿Cuántas columnas posee la tabla “creador”?

a. 0
b. 1
c. 2
d. 3

2.2. ¿Cuántas columnas numéricas posee la tabla “personaje”?

a. 11
b. 9
c. 10
d. 12

2.3. ¿Cuántas columnas de la tabla “personaje” pueden ser nulas?

a. Todas las columnas
b. 12
c. 10
d. Ninguna columna

2.4. ¿Qué significa la primera línea del script?

a. Selecciona la base de datos sobre la cual se aplicarán las siguientes sentencias.
b. Elimina la base de datos al finalizar de ejecutar el script.
c. Elimina la base de datos en caso de ya estar creada.
d. Ninguna de las anteriores

3. Elimina la base de datos “superhéroes.sql”.

Revisemos lo aprendido hasta aquí

- Uso y ejecución de scripts.
- Reconocimiento e implementación de consultas de Definición de Datos (DDL)

CONSULTAS DE MANIPULACIÓN DE DATOS (DML)

1. INSERT INTO

En su formato más sencillo, **INSERT** se utiliza para añadir una sola fila a una tabla. Debemos especificar el nombre de la tabla y una lista de valores para la fila. Los valores deben suministrarse en el mismo orden en el que se especificaron los atributos correspondientes en el comando CREATE TABLE.

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

2. UPDATE

El comando **UPDATE** se utiliza para modificar los valores de atributo de una o más filas seleccionadas. Una cláusula WHERE en el comando UPDATE selecciona las filas de una tabla que se van a modificar. La sentencia UPDATE tiene la siguiente forma:

```
UPDATE nombre_tabla
SET nombre_columna1 = valor1,
nombre_columna2 = valor2,
[ORDER BY ...] [WHERE condicion]
```

3. DELETE

El comando **DELETE** elimina filas de una tabla. Incluye una cláusula WHERE, para seleccionar las filas que se van a eliminar.

Las filas se eliminan explícitamente sólo de una tabla a la vez. Sin embargo, la eliminación se puede propagar a filas de otras tablas si se han especificado opciones de acciones referenciales en las restricciones de integridad referencial del DDL.

En función del número de filas seleccionadas por la condición de la cláusula WHERE, ninguna, una o varias filas pueden ser eliminadas por un solo comando DELETE. La ausencia de una cláusula WHERE significa que se borrarán todas las filas de la relación; sin embargo, la tabla permanece en la base de datos, pero vacía. Debemos utilizar el comando DROP TABLE para eliminar la definición de la tabla.

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

PASOS A REALIZAR PARA IMPLEMENTAR UNA BASE DE DATOS:

PASO	Descripción
1	Definir en el disco duro, el área física que contendrá las tablas de la base de datos. Sentencia SQL → CREATE DATABASE
2	Crear las diferentes tablas de la base de datos. Sentencia SQL → CREATE TABLE
3	Insertar en las filas los datos a las diferentes tablas, sin violar la integridad de los datos. Sentencia SQL → INSERT INTO
4	Actualizar los datos que cambien con el tiempo en las diferentes tablas. Sentencia SQL → UPDATE

5	Eliminar las filas que ya no se requieran en las diferentes tablas. Sentencia SQL → DELETE
6	Realizar las consultas deseadas a las tablas de la base de datos a través de la poderosa sentencia de consultas del SQL, llamada SELECT
7	Dar nombre a las consultas, elaboradas en el paso No.6 cuando se requiera ocultar el diseño y columnas de las tablas a través de la creación de vistas lógicas. Sentencia SQL → CREATE VIEW

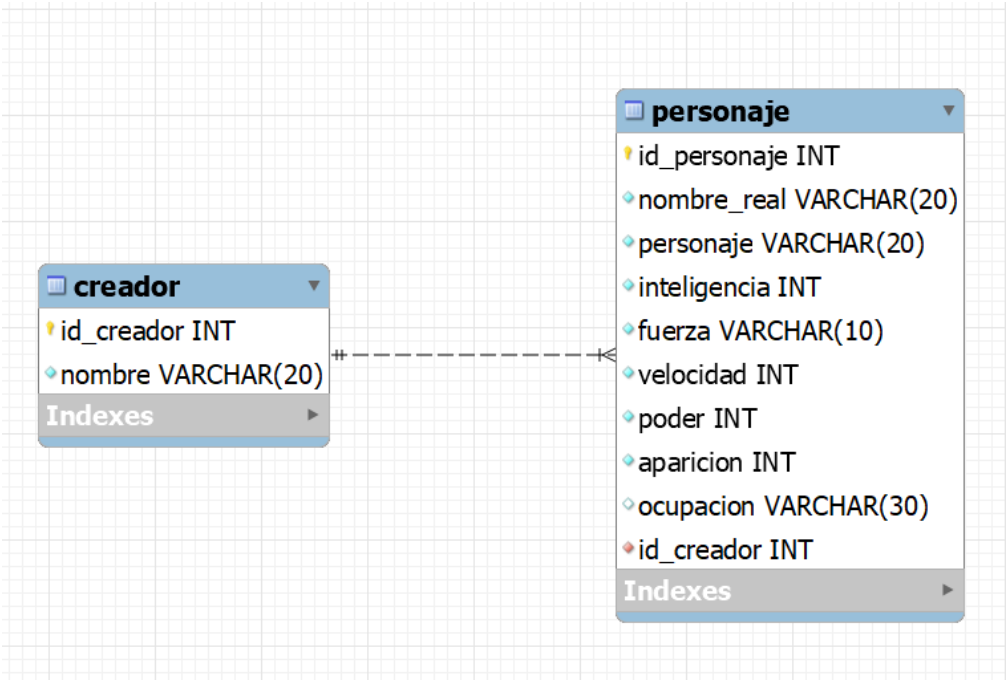
MANOS A LA OBRA!

Actividad 5

Volveremos a abrir y ejecutar el script “superhéroe”.

Hasta el momento, nuestra base de datos “superhéroe” se encuentra vacía. Sólo tenemos la estructura (tablas y columnas). Por lo que ahora es el turno de insertar registros en cada tabla.

A continuación, te mostramos el modelo Entidad-Relación de la tabla superhéroe:




```
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (1,'Bruce Banner', 'Hulk', 160, '600 mil', 75,98,1962,'Fisico Nuclear', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (2,'Tony Stark', 'Iron man', 170, '200 mil', 70,123,1963,'Inventor Industrial', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (3,'Thor Odinson ', 'Thor', 145, 'infinita', 100,235,1962,'Rey de Asgard', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (4,' Wanda Maximoff', 'Bruja Escarlata', 170, '100 mil', 90,345,1964,'Bruja', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (5,' Carol Danvers', 'Capitana Marvel', 157, '250 mil', 85,128,1968,'Oficial de Inteligencia', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (6,'Thanos', 'Thanos', 170, 'infinita', 40,306,1973,'Adorador de la muerte', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (7,'Peter Parker', 'Spiderman', 165, '25 mil', 80,74,1962,'Fotografo', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (8,'Steve Rogers', 'Capitan America', 145, '500', 45,80,1941,'Oficial Federal', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (9,'Bobby Drake', 'Iceman', 140, '2 mil', 64,122,1963,'Contador', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (10,'Barry Allen', 'Flash', 160, '10 mil', 120,168,1956,'Cientifico Forense', 2);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (11,'Bruce Wayne', 'Batman', 170, '500', 32,47,1939,'Hombre de Negocios', 2);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (12,'Clarck Kent', 'Superman', 165, 'infinita', 120,182,1948,'Reportero', 2);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (13,'Diana Prince', 'Mujer Maravilla', 160, 'infinita', 95,127,1949,'Princesa Guerrera', 2);
```

```
INSERT INTO creador (id_creador,nombre) values ('1','MARVEL');
INSERT INTO creador (id_creador,nombre) values ('2','DC Comics');
```

a) Primero insertar en la tabla “creador” los siguientes datos:

Tabla creador

id_creador	creador
1	Marvel
2	DC Comics

b) Ahora, continuaremos insertando registros en la tabla personaje

Tabla personaje

id_personaje	nombre_real	personaje	Inteligencia	fuerza	velocidad	poder	aparicion	ocupación	id_creador
1	Bruce Banner	Hulk	160	600 mil	75	98	1962	Físico Nuclear	1
2	Tony Stark	Iron Man	170	200 mil	70	123	1963	Inventor Industrial	1
3	Thor Odinson	Thor	145	infinita	100	235	1962	Rey de Asgard	1
4	Wanda Maximoff	Bruja Escarlata	170	100 mil	90	345	1964	Bruja	1
5	Carol Danvers	Capitana Marvel	157	250 mil	85	128	1968	Oficial de inteligencia	1
6	Thanos	Thanos	170	infinita	40	306	1973	Adorador de la muerte	1
7	Peter Parker	Spiderman	165	25 mil	80	74	1962	Fotógrafo	1
8	Steve Rogers	Capitan America	145	600	45	60	1941	Oficial Federal	1
9	Bobby Drake	Ice Man	140	2 mil	64	122	1963	Contador	1
10	Barry Allen	Flash	160	10 mil	120	168	1956	Científico forense	2
11	Bruce Wayne	Batman	170	500	32	47	1939	Hombre de negocios	2
12	Clarck Kent	Superman	165	infinita	120	182	1948	Reportero	2
13	Diana Prince	Mujer Maravilla	160	infinita	95	127	1949	Princesa guerrera	2

¡Felicitaciones! Creaste y completaste tu primera base de datos. A continuación, te pediremos realices algunos cambios en ella así seguimos practicando lo aprendido

a) Cambiar en la tabla personajes el año de aparición a 1938 del personaje Superman.

```
update personajes set aparicion = 1938 where personaje = 'Superman' ;
```

b) Eliminar el registro que contiene al personaje Flash.

```
DELETE FROM personajes WHERE personaje = 'Flash';
```

Actividad extra

Vamos a aprovechar que ya estamos trabajando con scripts y vamos a abrir el script llamado **“nba”** y ejecutarlo de modo tal que se cree la base de datos y todas sus tablas, Este proceso puede demorar, es normal y es super importante que lo hagas: ¡ten paciencia!

Esto lo hacemos porque más adelante la vamos a necesitar en el módulo de MySQL, entonces mejor adelantarnos a ese momento y no perder tiempo cuando tengamos que trabajar sobre dicha base de datos.

Revisemos lo aprendido hasta aquí

- Uso y ejecución de scripts.
- Realización de consultas de Manipulación de Datos (DML)

CONSULTAS PARA ACCEDER Y MANIPULAR INFORMACIÓN

1. SELECT

La sentencia SELECT es muy poderosa y ampliamente rica en sus cláusulas y variantes permitiendo la capacidad de atender en poco tiempo a consultas complejas sobre la base de datos. Está en el especialista desarrollador de aplicaciones conocerlo a profundidad para explotar las bondades y virtudes.

Se usa para listar las columnas de las tablas que se desean ver en el resultado de una consulta. Además de las columnas se pueden listar columnas a calcular por el SQL cuando actué la sentencia. Esta cláusula no puede omitirse.

La sentencia SELECT, obtiene y nos permite mostrar filas de la base de datos, también permite realizar la selección de una o varias filas o columnas de una o varias tablas. Para seleccionar la tabla de la que queremos obtener dichas filas vamos a utilizar la sentencia FROM.

La sentencia FROM lista las tablas de donde se listarán las columnas enunciadas en el SELECT. Esta cláusula no puede omitirse.

```
SELECT nombres de las columnas FROM tablaOrigen;
```

```
SELECT nombre, apellido FROM Alumnos;
```

Teniendo la siguiente tabla de alumnos:

Alumnos			
Id	**Nombre**	**Apellido**	**Edad**
1	Agustín	Cocco	24
2	Martin	Bullón	21

El resultado que mostraría sería:

Alumnos	
Nombre	**Apellido**
Agustín	Cocco
Martin	Bullón

Para mostrar todos los datos de una tabla usamos el símbolo (*). Esto nos mostraría la primera tabla.

```
SELECT * FROM Alumnos;
```

También en las consultas SELECT podemos hacer operaciones matemáticas entre los datos numéricos de las tablas que elijamos. Usualmente ponemos estas operaciones entre paréntesis para separar la operación del resto de la consulta.

```
SELECT nombre,(salario+comision) FROM Empleados;
```

Teniendo la siguiente tabla de Empleados:

Empleados			
Id	**Nombre**	**Salario**	**Comisión**
1	Agustín	5000	300
2	Martin	2000	250

El resultado que mostraría sería:

Empleados	
Nombre	**(salario+comision)**
Agustín	5300
Martin	2250

A la consulta SELECT le podemos sumar cláusulas que van a alterar el resultado de filas que obtenga el SELECT, esto nos puede servir para traer ciertas filas y evitar algunas que no queremos mostrar.

CLÁUSULAS

1. SELECT DISTINCT

El SELECT DISTINCT se utiliza cuando queremos traer solo registros diferentes. En las tablas a veces puede haber valores repetidos, para evitarlos usamos esta sentencia.

```
SELECT DISTINCT nombres de las columnas FROM tablaOrigen;
```

```
SELECT DISTINCT nombre, apellido FROM Alumnos;
```

Teniendo la siguiente tabla de alumnos:

Alumnos			
Id	**Nombre**	**Apellido**	**Edad**
1	Agustín	Cocco	24
2	Martin	Bullón	21
3	Agustín	Cocco	1

El resultado que mostraría sería:

Alumnos	
Nombre	**Apellido**
Agustín	Cocco
Martin	Bullón

2. WHERE

Establece criterios de selección de ciertas filas en el resultado de la consulta gracias a las condiciones de búsqueda. Si no se requiere condiciones de búsqueda puede omitirse y el resultado de la consulta serán todas las filas de las tablas enunciadas en el FROM.

```
SELECT nombres de las columnas FROM tablaOrigen WHERE condición de Búsqueda;
```

```
SELECT nombre, apellido FROM Alumnos WHERE nombre = “Agustin”;
```

En este ejemplo traerá todos los alumnos con nombre Agustín. Nótese que el nombre está en comillas dobles, esto es porque si vamos a poner una cadena en la condición debe estar entre comillas dobles, si fuese un número no sería necesario.

Teniendo la siguiente tabla alumnos:

Alumnos			
Id	**Nombre**	**Apellido**	**Edad**
1	Agustín	Cocco	24
2	Martin	Bullón	21

El resultado que mostraría sería:

Alumnos	
Nombre	**Apellido**
Agustín	Cocco

En las condiciones **WHERE** podemos utilizar operadores lógicos, operadores relaciones operadores propios de SQL.

Operadores Relacionales

Operador	Significado	Ejemplo
>	Mayor que	`SELECT * FROM Alumnos WHERE edad > 21;`
<	Menor que	`SELECT * FROM Alumnos WHERE edad < 18;`
=	Igual que	`SELECT * FROM Alumnos WHERE edad = 20;`
>=	Mayor o igual que	`SELECT * FROM Alumnos WHERE edad >= 30;`
<=	Menor o igual que	`SELECT * FROM Alumnos WHERE edad <= 10;`
<> o !=	Distinto que	`SELECT * FROM Alumnos WHERE edad <> 5;`

En todos estos ejemplos estamos buscando las filas donde la edad de un alumno sea mayor, menor, etc, a x edad. Usamos edad, pero puede ser cualquier valor numérico o valor de tipo cadena.

Operadores Lógicos

Operador	Significado	Ejemplo
AND	El operador AND muestra un registro si todas las condiciones separadas por AND son verdaderas	`SELECT * FROM Alumnos WHERE edad = 18 AND edad = 21;`
OR	El operador OR muestra un registro si algunas de las condiciones separadas por OR es verdadera.	`SELECT * FROM Alumnos WHERE edad = 15 OR edad = 20;`
NOT	El operador NOT muestra un registro si la/s condición/es no es verdadera.	`SELECT * FROM Alumnos WHERE NOT edad = 20;`

Los operadores lógicos sirven para filtrar resultados basados en más de una condición.

Operadores propios de SQL

a) BETWEEN

El operador **BETWEEN** selecciona valores dentro de un rango determinado. Los valores pueden ser números, texto o fechas.

```
SELECT nombre/s de la/s columna/s FROM tablaOrigen WHERE condición de
Búsqueda BETWEEN valor1 AND valor2;

SELECT * FROM Alumnos WHERE edad BETWEEN 21 AND 40;
```

Usamos edad, pero puede ser cualquier valor numérico o valor de tipo cadena.

Teniendo la siguiente tabla:

Id	Nombre	Apellido	Edad
1	Agustín	Cocco	24
2	Martin	Bullon	39
3	Mariela	Lima	60
4	Juliana	Martínez	30
5	Gastón	Vidal	26

El resultado sería:

Id	Nombre	Apellido	Edad
1	Agustín	Cocco	24
2	Martin	Bullon	39
4	Juliana	Martínez	30
5	Gastón	Vidal	26

b) IN

El operador **IN** te permite especificar varios valores para una condición de una cláusula WHERE. Es un atajo para no escribir varias condiciones OR.

```
SELECT nombre/s de la/s columna/s FROM tablaOrigen WHERE condición de
Búsqueda IN (valor1, valor2, valor3, …);

SELECT * FROM Alumnos WHERE nombre IN (“Agustín, “Mariela”, “Juliana”);
```

Usamos nombre, pero puede ser cualquier valor numérico o valor de tipo cadena.

Teniendo la siguiente tabla de alumnos:

Id	Nombre	Apellido	Edad
1	Agustín	Cocco	24
2	Martin	Bullon	15
3	Mariela	Lima	20

El resultado que mostraría sería:

Id	Nombre	Apellido	Edad
1	Agustín	Cocco	24
3	Mariela	Lima	20

c) LIKE

El operador **LIKE** se usa en una cláusula WHERE para buscar un patrón específico en una columna. También se usa cuando queremos utilizar una cadena en una comparación WHERE

Hay dos símbolos que se utilizan a menudo junto con el operador LIKE:

El signo de porcentaje (%) representa cero, uno o varios caracteres y el guión bajo (_) para representar un carácter. Estos signos se pueden utilizar por separado o juntos.

```
SELECT nombre/s de la/s columna/s FROM tablaOrigen WHERE condición de Búsqueda LIKE patrón de valores o cadena;
```

```
SELECT nombre, apellido FROM Alumnos WHERE nombre LIKE ‘s%’;
```

Esa consulta, lo que hace es traer todos los alumnos donde el nombre empiece con el carácter s.

Teniendo la siguiente tabla de alumnos:

Id	Nombre	Apellido	Edad
1	Sebastián	Gómez	24
2	Sabrina	Martínez	15
3	Mariela	Lima	20

El resultado que mostraría sería:

Id	Nombre	Apellido	Edad
1	Sebastián	Gómez	24
2	Sabrina	Martínez	15

Operador LIKE	Significado
WHERE nombre LIKE ‘a%’	Encuentra cualquier nombre que empiece con “a”.
WHERE nombre LIKE ‘%a’	Encuentra cualquier nombre que termine con “a”.

WHERE nombre LIKE ‘%ar%’	Encuentra cualquier nombre que tenga “ar” en cualquier posición.
WHERE nombre LIKE ‘_e%’	Encuentra cualquier nombre que tenga “e” en la segunda posición.
WHERE nombre LIKE ‘e_%’	Encuentra cualquier nombre que empiece con “e” y sea por lo menos de 2 de largo.
WHERE nombre LIKE ‘e__%’	Encuentra cualquier nombre que empiece con “e” y sea por lo menos de 3 de largo.
WHERE nombre LIKE ‘a%n%’	Encuentra cualquier nombre que empiece con “a” y termine con “n”.

MANOS A LA OBRA!

Actividad 6

a) Ahora veamos cómo ha quedado la tabla “superheroes” que creaste anteriormente. Para ello necesitarás una consulta de tipo SELECT.

```
SELECT * FROM personajes ;
```

b) Realiza una consulta que devuelva todos los valores de la columna “nombre_real” de la tabla superhéroe.

```
select nombre_real FROM Personajes ;
```

c) Realiza una consulta que devuelva todos los nombres reales de los personajes cuyo nombre empieza con “B”.

```
select nombre_real FROM Personajes where nombre_real like 'B%';
```

Revisemos lo aprendido hasta aquí

- Poder realizar consultas a la base de datos y extraer información específica.
- Reconocer y utilizar consultas de Definición de Datos (DDL)

3. ORDER BY

La cláusula ORDER BY permitirá establecer la columna o columnas sobre las cuales las filas que se mostrarán de la consulta deberán ser ordenadas. Este orden puede ser ascendiente se agrega la palabra ASC y descendiente si se agrega la palabra DESC al final.

Esta cláusula puede omitirse.

```
SELECT nombre/s de la/s columna/s FROM tablaOrigen ORDER BY columna
ASC|DESC;
```

```
SELECT nombre, apellido FROM Alumno ORDER BY nombre ASC;
```

En este caso mostraría los resultados ordenados de manera ascendiente, según el nombre de los alumnos.

Teniendo la siguiente tabla de alumnos:

Alumnos			
Id	**Nombre**	**Apellido**	**Edad**
1	Jerónimo	Wiunkhaus	24
2	Ana	Gadea	15
3	Mariela	Lima	20

El resultado que mostraría sería:

Alumnos			
Id	**Nombre**	**Apellido**	**Edad**
2	Ana	Gadea	15
1	Jerónimo	Wiunkhaus	24
3	Mariela	Lima	20

MANOS A LA OBRA!

Actividad 7

Pongamos a prueba esta nueva cláusula: order by. Seguiremos trabajando con la tabla “superhéroe”. Realiza una consulta que devuelva todos los registros ordenados por “inteligencia”.

```
SELECT * FROM PERSONAJES order by INTELIGENCIA;
```

4. GROUP BY

Especifica una consulta sumaria. En vez de producir una fila de resultados por cada fila de datos de la base de datos, una consulta sumaria agrupa todas las filas similares y luego produce una fila sumaria de resultados para cada grupo de los nombres de columnas enunciados en esta cláusula.

En otras palabras, esta cláusula permitirá agrupar un conjunto de columnas con valores repetidos y utilizar las funciones de agregación sobre las columnas con valores no repetidos. Esta cláusula puede omitirse.

SELECT count(PERSONAJE) ,INTELIGENCIA FROM PERSONAJES GROUP by INTELIGENCIA;

```
SELECT nombre/s de la/s columna/s FROM tablaOrigen GROUP BY nombres de
columna/s por la cual Agrupar;
```

¿Qué son las funciones de agregación?

En la gestión de bases de datos, una función de agregación es una función en la que los valores de varias filas se agrupan bajo un criterito para formar un valor único más significativo. Estas funciones se ponen el SELECT.

Existen 5 tipos de funciones de agregación, **MAX(), MIN(), COUNT(), SUM(), AVG().**

a) MAX

Esta función retorna el valor más grande de una columna.

```
SELECT MAX(nombre de la columna) FROM tablaOrigen;
```

```
SELECT MAX(salario) FROM Empleados;
```

Teniendo la siguiente tabla de empleados:

Id	Nombre	Apellido	Salario
1	Franco	Medina	1000
2	Agustina	Koch	2000
3	Ignacio	Pérez	1500

El resultado que mostraría sería:

MAX(Salario)
2000

b) MIN

Esta función retorna el valor más chico de una columna.

```
SELECT MIN (nombre de la columna) FROM tablaOrigen;
```

```
SELECT MIN(salario) FROM Empleados;
```

Teniendo la siguiente tabla de empleados:

Id	Nombre	Apellido	Salario
1	Franco	Medina	1000
2	Agustina	Koch	2000
3	Ignacio	Pérez	1500
4	Martin	Bruno	3000

El resultado que mostraría sería:

MIN(Salario)
1000

c) AVG

Esta función retorna el promedio de una columna.

```
SELECT AVG(nombre de la columna) FROM tablaOrigen;
```

```
SELECT AVG(salario) FROM Empleados;
```

Teniendo la siguiente tabla de empleados:

Id	Nombre	Apellido	Salario
1	Franco	Medina	1000
2	Agustina	Koch	2000
3	Ignacio	Pérez	1600
4	Valentín	Mazuran	700

El resultado que mostraría sería:

AVG(Salario)
1325

d) COUNT

Esta función retorna el número de filas de una columna.

```
SELECT COUNT(nombre de la columna) FROM tablaOrigen;
```

```
SELECT COUNT(Id) FROM Empleados;
```

Teniendo la siguiente tabla de empleados:

Id	Nombre	Apellido	Salario
1	Franco	Medina	1000
2	Agustina	Koch	2000
3	Ignacio	Pérez	1600

El resultado que mostraría sería:

COUNT(Id)
3

En este caso ponemos el id, para saber cuántos empleados tenemos en la tabla empleados.

También podemos usar el COUNT(*), este no requiere que le pasamos una columna concreta y cuenta todas filas de una tabla, mostrando tanto los valores repetidos como los valores en null.

Id	Nombre	Apellido	Salario
1	Franco	Medina	1000
2	Agustina	Koch	2000
3	Franco	Medina	1600
4	Martin	Santiago	null

El resultado que mostraría sería:

COUNT(Id)
4

Entonces, volviendo al Group By, vamos a utilizar esta sentencia junto con las funciones de agregación para agrupar los valores que devuelva dicha función. Existen dos tipos de GROUP BY.

```
SELECT nombre, SUM(salario) FROM Empleados GROUP BY nombre;
```

Teniendo la siguiente tabla de empleados:

Empleados			
Id	**Nombre**	**Salario**	**Comisión**
1	Franco	1000	500
2	Mariela	2000	200
3	Franco	1000	800
4	Mariela	2000	350

El resultado que mostraría sería:

Empleados	
Nombre	**SUM(Salario)**
Franco	2000
Mariela	4000

El resultado de la consulta muestra que agrupa todos los nombres repetidos bajo un solo nombre y el salario es la suma de los salarios de las filas que fueron agrupadas.

Otro ejemplo de un group by sería:

```
SELECT COUNT(ID), pais FROM Personas GROUP BY pais;
```

Teniendo la siguiente tabla de personas:

Personas		
Id	**Nombre**	**País**
1	Franco	Argentina
2	Juliana	Alemania
3	Agustín	Argentina

El resultado que mostraría sería:

Personas	
COUNT(Id)	**País**
2	Argentina
1	Alemania

En la consulta hacemos un count del id de personas para saber cuántos hay, pero al agrupar el resultado por países, nos muestra cuantas personas hay en cada país.

5. HAVING

Esta cláusula le dice al SQL que incluya sólo ciertos grupos producidos por la cláusula GROUP BY en los resultados de la consulta. Al igual que la cláusula WHERE, utiliza una condición de búsqueda para especificar los grupos deseados. La cláusula HAVING es la encargada de condicionar la selección de los grupos en base a los valores resultantes en las funciones agregadas utilizadas debidas que la cláusula WHERE condiciona solo para la selección de filas individuales. Esta cláusula puede omitirse.

```
SELECT nombre/s de la/s columna/s FROM tablaOrigen GROUP BY nombres de
columnas por la cual Agrupar HAVING condiciónBúsqueda para Group By;

SELECT COUNT(ID), pais FROM Personas GROUP BY pais HAVING COUNT(ID) > 1;
```

Teniendo la siguiente tabla de personas:

Personas		
Id	**Nombre**	**País**
1	Franco	Argentina
2	Juliana	Alemania
3	Agustín	Argentina
4	Gastón	Alemania
5	Mariela	Uruguay

El resultado que mostraría sería:

Personas	
COUNT(Id)	**País**
2	Argentina
2	Alemania

En la consulta hacemos un **COUNT del ID** de personas para saber cuántos hay, las agrupamos por países para que nos muestre cuantas personas hay en cada país. Pero, con el HAVING le decimos que nos muestre solo los resultados donde el COUNT sea mayor a 1, o en otras palabras, mostramos los países que tienen más de una persona.

6. AS

La sentencia AS, le da un alias a una o la columna de una tabla, un nombre temporal. El alias existe solo por la duración de la consulta.

El alias se usa para darle a una columna un nombre más legible

```
SELECT nombre/s de la/s columna/s AS alias FROM tablaOrigen;
```

```
SELECT nombre AS Nombre Alumno, apellido As Apellido Alumno FROM
Alumnos;
```

Teniendo la siguiente tabla de alumnos:

Alumnos			
Id	**Nombre**	**Apellido**	**Edad**
1	Xavier	Collado	24
2	Ana	Gadea	15
3	Mariela	Lima	20

El resultado que mostraría sería:

Alumnos	
Nombre Alumno	**Apellido Alumno**
Ana	Gadea
Xavier	Collado
Mariela	Lima

Todas estas cláusulas / sentencias pueden ser usadas juntas, no es necesario que las usen separadas.

```
SELECT nombres de las columnas AS Alias FROM tablaOrigen
WHERE condición de Búsqueda
GROUP BY nombres de columnas por la cual Agrupar
HAVING condiciónBúsqueda para Group By
ORDER BY nombre de columnas [ASC | DESC]
```

7. ROUND

La sentencia round sirve para redondear los decimales de un número que se pida en un select.

```
SELECT AVG(salario) FROM Empleados;
```

Empleados
AVG(Salario)
1325,55

```
SELECT ROUND(AVG(salario)) FROM Empleados;
```

Empleados
AVG(Salario)
1326

8. LIMIT

La cláusula LIMIT se utiliza para establecer un límite al número de resultados devueltos por SQL.

```
SELECT nombres de las columnas FROM tablaOrigen LIMIT numero x;
```

```
SELECT nombre, apellido FROM Alumnos LIMIT 1;
```

Teniendo la siguiente tabla de alumnos:

Alumnos			
Id	Nombre	Apellido	Edad
1	Jerónimo	Wiunkhaus	24
2	Ana	Gadea	15
3	Mariela	Lima	20

El resultado que mostraría sería:

Alumnos	
Nombre	Apellido
Jerónimo	Wiunkhaus

CONSULTAS MULTITABLAS

Como habíamos dicho previamente en la teoría, estamos trabajando con base de datos relacionales, esto significa que tenemos tablas relacionadas entre sí. Y dentro de esas tablas, tenemos fila relacionadas con filas de otras tablas.

Pero ¿cómo hacemos para traer la información de una tabla y la información de la tabla con la que está relacionada? Una sentencia muy útil para unificar información de tablas relacionales es el **JOIN**.

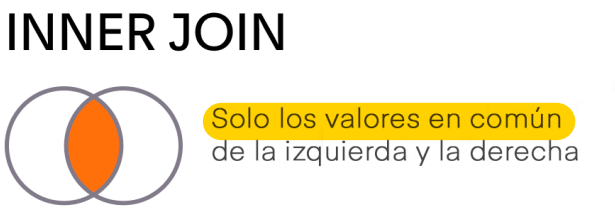
SQL JOIN

La sentencia **JOIN**, se usa para combinar data o filas de dos o más tablas que tengan un campo en común entre ellas. Usualmente es la llave foránea.

Los diferentes tipos de JOIN son:

1. **INNER JOIN**

   El **INNER JOIN** selecciona todas las filas que tengan un valor en común con la/s tabla/s. Si hay una fila, que no tiene un valor en común con la otra tabla no la trae.




```sql
SELECT *
FROM TABLE_1
INNER JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
```

```sql
SELECT nombre/s de la/s columna/s FROM tabla1 INNER JOIN tabla2 ON
tabla1.nombre_columna = tabla2.nombre_columna;
```

```sql
SELECT Nombre, Nombre_curso FROM Profesores INNER JOIN Cursos ON
Profesores.Id = Cursos.Id_profesor;
```

Teniendo la siguiente tabla de profesores:

Profesores			
Id	**Nombre**	**Apellido**	**Edad**
1	Agustín	Oviedo	24
2	Ana	Gadea	15
3	Mariela	Lima	20
4	Francisco	Chirino	30

Teniendo la siguiente tabla de Cursos:

Cursos			
Id	**Nombre_curso**	**Costo**	**Id_profesor**
1	Curso de Programación	1000	1
2	Curso de Mecánica	2000	2
3	Curso de Cocina	500	3

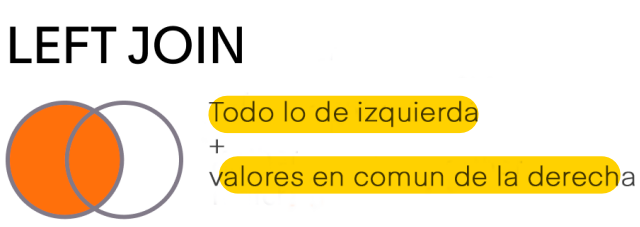
El resultado que mostraría sería:

Resultado INNER JOIN	
Nombre	**Nombre_curso**
Agustín	Curso de Programación
Ana	Curso de Mecánica
Mariela	Curso de Cocina

Gracias al INNER JOIN, podemos mostrar todos los nombres de los profesores, junto al curso que dan, nosotros decimos que son el profesor de ese curso, porque tienen asignado su id en la tabla curso. Y como usamos el INNER JOIN, solo mostramos los profesores que tenían su id en la tabla curso. Esto se por la condición que pusimos arriba en el ON, donde decíamos que el valor a chequear por posible coincidencia era el id en la tabla profesor y el id_profesor en la tabla curso.

2. LEFT JOIN

La sentencia **LEFT JOIN** retorna todos los registros de la tabla de la izquierda (tabla1) y todos los registros con coincidencia de la tabla de la derecha (tabla2). Si no existe ninguna coincidencia para alguna de las filas de la tabla de la izquierda, de igual forma todos los resultados de la primera tabla se muestran.




```sql
SELECT *
FROM TABLE_1
LEFT JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
```

```sql
SELECT nombre/s de la/s columna/s FROM tabla1 LEFT JOIN tabla2 ON
tabla1.nombre_columna = tabla2.nombre_columna;
```

```sql
SELECT Nombre, Nombre_curso FROM Profesores LEFT JOIN Cursos ON
Profesores.Id = Cursos.Id_profesor;
```

Teniendo la siguiente tabla de profesores:

Profesores			
Id	**Nombre**	**Apellido**	**Edad**
1	Agustín	Oviedo	24
2	Ana	Gadea	15
3	Mariela	Lima	20
4	Francisco	Chirino	30

Teniendo la siguiente tabla de Cursos:

Cursos			
Id	**Nombre_curso**	**Costo**	**Id_profesor**
1	Curso de Programación	1000	1
2	Curso de Mecánica	2000	2

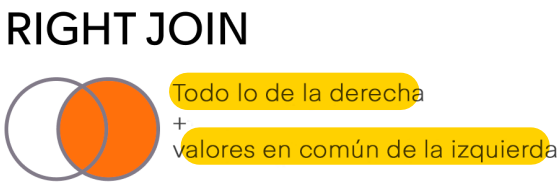
El resultado que mostraría sería:

Resultado LEFT JOIN	
Nombre	**Nombre_curso**
Agustín	Curso de Programación
Ana	Curso de Mecánica
Mariela	NULL
Francisco Chirino	NULL

Si nos fijamos en el resultado de la consulta, podemos ver que trajo todas las filas de la tabla de la izquierda, in importar si las filas tenían coincidencia o no.

3. RIGHT JOIN

Esta sentencia es parecida a la anterior, pero le da prioridad a la tabla de la derecha.




```
SELECT *
FROM TABLE_1
RIGHT JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
```

```
SELECT nombre/s de la/s columna/s FROM tabla1 RIGHT JOIN tabla2 ON
tabla1.nombre_columna = tabla2.nombre_columna;
```

```
SELECT Nombre, Nombre_curso FROM Profesores RIGHT JOIN Cursos ON
Profesores.Id = Cursos.Id_profesor;
```

Teniendo la siguiente tabla de profesores:

Profesores			
Id	**Nombre**	**Apellido**	**Edad**
1	Agustín	Oviedo	24
2	Ana	Gadea	15
3	Mariela	Lima	20

Teniendo la siguiente tabla de Cursos:

Cursos			
Id	**Nombre_curso**	**Costo**	**Id_profesor**
1	Curso de Programación	1000	1
2	Curso de Mecánica	2000	2
3	Curso de Natación	600	NULL

El resultado que mostraría sería:

Resultado RIGHT JOIN	
Nombre	Nombre_curso
Agustín	Curso de Programación
Ana	Curso de Mecánica
NULL	Curso de Natación

Si nos fijamos en el resultado de la consulta, podemos ver que trajo todas las filas de la tabla de la derecha, sin importar si las filas tenían coincidencia o no.

SUBCONSULTAS

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

Anteriormente hemos utilizado la cláusula WHERE para seleccionar los datos que deseábamos comparando un valor de una columna con una constante, o un grupo de ellas. Si los valores de dichas constantes son desconocidos, normalmente por proceder de la aplicación de funciones a determinadas columnas de la tabla, tendremos que utilizar subconsultas. Por ejemplo, queremos saber la lista de empleados cuyo salario supere el salario medio.

En primer lugar, tendríamos que averiguar el importe del salario medio:

```
SELECT AVG(salario) "Salario Medio" FROM Empleados;
```

Empleados
Salario Medio
256666,67

Ahora que sabemos el dato podríamos usarlo para la consulta:

```
SELECT nombre, salario FROM Empleados WHERE > 256666,67;
```

Empleados	
Nombre	**Salario**
Agustín	385000
Ana	608000

Esto estaría bien, pero, es porque primero buscamos el dato en una consulta y una vez que conseguimos el dato, ahí hicimos la consulta. Pero, lo mejor sería que en vez de hacer dos consultas usemos una subconsulta, para que al mismo tiempo que averiguamos el salario medio, se calcule cuáles son los empleados que tienen un sueldo mayor a ese salario medio.

```
SELECT nombre, salario FROM Empleados WHERE > (SELECT AVG(salario) FROM Empleados);
```

Empleados	
Nombre	**Salario**
Agustín	385000
Ana	608000

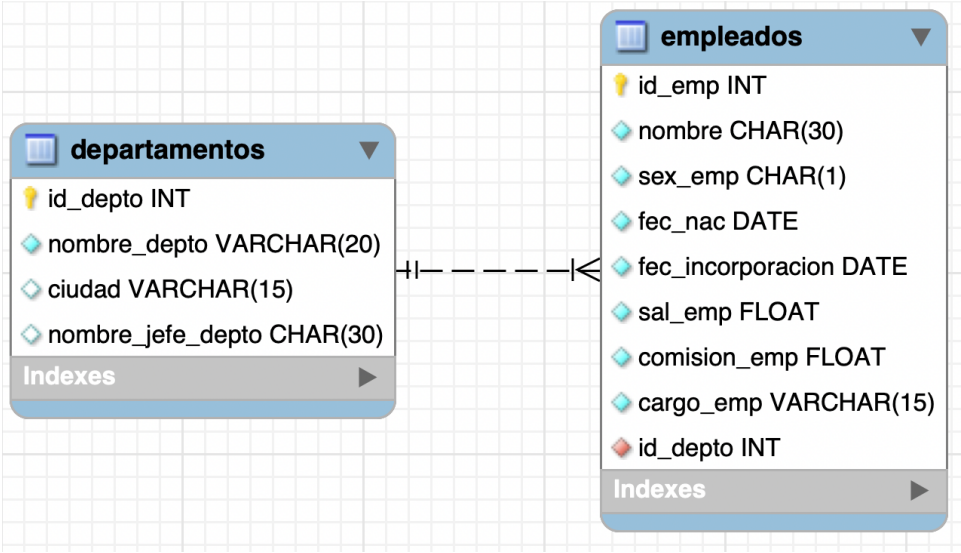
Esto nos daría el mismo resultado, pero sin la necesidad de hacer dos consultas para saber el dato. Estos son los casos donde usaríamos una subconsulta, donde no sabíamos el salario medio antes de hacer la consulta.

EJERCICIOS DE APRENDIZAJE

Para la realización de los ejercicios que se describen a continuación, es necesario descargar el archivo scriptsBD.zip que contiene algunos scripts con las bases de datos sobre las cuales se va a trabajar. En cada ejercicio se indica el nombre del script que se debe utilizar. Para abrir y ejecutar los scripts van a encontrar un pdf de cómo hacerlo en Moodle, con el nombre de Tutorial Scripts SQL.

VIDEOS: Te sugerimos ver los videos relacionados con este tema, antes de empezar los ejercicios, los podrás encontrar en tu aula virtual o en nuestro canal de YouTube.

1. Abrir el script llamado “personal” y ejecutarlo de modo tal que se cree la base de datos “personal”, se creen las tablas y se inserten todos los datos en las tablas para que quede de la siguiente manera:



a) A continuación, realizar las siguientes consultas sobre la base de datos personal:

1. Obtener los datos completos de los empleados.
2. Obtener los datos completos de los departamentos.
3. Listar el nombre de los departamentos.
4. Obtener el nombre y salario de todos los empleados.
5. Listar todas las comisiones.
6. Obtener los datos de los empleados cuyo cargo sea ‘Secretaria’.
7. Obtener los datos de los empleados vendedores, ordenados por nombre alfabéticamente.
8. Obtener el nombre y cargo de todos los empleados, ordenados por salario de menor a mayor.
9. Obtener el nombre de o de los jefes que tengan su departamento situado en la ciudad de “Ciudad Real”

10. Elabore un listado donde para cada fila, figure el alias ‘Nombre’ y ‘Cargo’ para las respectivas tablas de empleados.

11. Listar los salarios y comisiones de los empleados del departamento 2000, ordenado por comisión de menor a mayor.

12. Obtener el valor total a pagar a cada empleado del departamento 3000, que resulta de: sumar el salario y la comisión, más una bonificación de 500. Mostrar el nombre del empleado y el total a pagar, en orden alfabético.

13. Muestra los empleados cuyo nombre empiece con la letra J.

14. Listar el salario, la comisión, el salario total (salario + comisión) y nombre, de aquellos empleados que tienen comisión superior a 1000.

15. Obtener un listado similar al anterior, pero de aquellos empleados que NO tienen comisión.

16. Obtener la lista de los empleados que ganan una comisión superior a su sueldo.

17. Listar los empleados cuya comisión es menor o igual que el 30% de su sueldo.

18. Hallar los empleados cuyo nombre no contiene la cadena “MA”

19. Obtener los nombres de los departamentos que sean “Ventas”, “Investigación” o ‘Mantenimiento.

20. Ahora obtener el contrario, los nombres de los departamentos que **no** sean “Ventas” ni “Investigación” ni ‘Mantenimiento.

21. Mostrar el salario más alto de la empresa.

22. Mostrar el nombre del último empleado de la lista por orden alfabético.

23. Hallar el salario más alto, el más bajo y la diferencia entre ellos.

24. Hallar el salario promedio por departamento.

Consultas con Having

25. Hallar los departamentos que tienen más de tres empleados. Mostrar el número de empleados de esos departamentos.

26. Hallar los departamentos que no tienen empleados

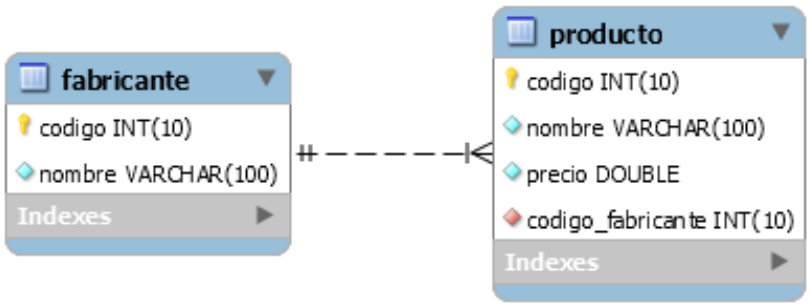
Consulta Multitabla (Uso de la sentencia JOIN/LEFT JOIN/RIGHT JOIN)

27. Mostrar la lista de empleados, con su respectivo departamento y el jefe de cada departamento.

Consulta con Subconsulta

28. Mostrar la lista de los empleados cuyo salario es mayor o igual que el promedio de la empresa. Ordenarlo por departamento.

2. Abrir el script de la base de datos llamada “tienda.sql” y ejecutarlo para crear sus tablas e insertar datos en las mismas. A continuación, generar el modelo de entidad relación. Deberá obtener un diagrama de entidad relación igual al que se muestra a continuación:



A continuación, se deben realizar las siguientes consultas sobre la base de datos:

1. Lista el nombre de todos los productos que hay en la tabla producto.
2. Lista los nombres y los precios de todos los productos de la tabla producto.
3. Lista todas las columnas de la tabla producto.
4. Lista los nombres y los precios de todos los productos de la tabla producto, redondeando el valor del precio.
5. Lista el código de los fabricantes que tienen productos en la tabla producto.
6. Lista el código de los fabricantes que tienen productos en la tabla producto, sin mostrar los repetidos.
7. Lista los nombres de los fabricantes ordenados de forma ascendente.
8. Lista los nombres de los productos ordenados en primer lugar por el nombre de forma ascendente y en segundo lugar por el precio de forma descendente.
9. Devuelve una lista con las 5 primeras filas de la tabla fabricante.
10. Lista el nombre y el precio del producto más barato. (Utilice solamente las cláusulas ORDER BY y LIMIT)
11. Lista el nombre y el precio del producto más caro. (Utilice solamente las cláusulas ORDER BY y LIMIT)
12. Lista el nombre de los productos que tienen un precio menor o igual a $120.
13. Lista todos los productos que tengan un precio entre $60 y $200. Utilizando el operador BETWEEN.
14. Lista todos los productos donde el código de fabricante sea 1, 3 o 5. Utilizando el operador IN.
15. Devuelve una lista con el nombre de todos los productos que contienen la cadena Portátil en el nombre.

Consultas Multitabla

1. Devuelve una lista con el código del producto, nombre del producto, código del fabricante y nombre del fabricante, de todos los productos de la base de datos.
2. Devuelve una lista con el nombre del producto, precio y nombre de fabricante de todos los productos de la base de datos. Ordene el resultado por el nombre del fabricante, por orden alfabético.
3. Devuelve el nombre del producto, su precio y el nombre de su fabricante, del producto más barato.
4. Devuelve una lista de todos los productos del fabricante Lenovo.
5. Devuelve una lista de todos los productos del fabricante Crucial que tengan un precio mayor que $200.
6. Devuelve un listado con todos los productos de los fabricantes Asus, Hewlett-Packard. Utilizando el operador IN.
7. Devuelve un listado con el nombre de producto, precio y nombre de fabricante, de todos los productos que tengan un precio mayor o igual a $180. Ordene el resultado en primer lugar por el precio (en orden descendente) y en segundo lugar por el nombre (en orden ascendente)

Consultas Multitabla

Resuelva todas las consultas utilizando las cláusulas LEFT JOIN y RIGHT JOIN.

1. Devuelve un listado de todos los fabricantes que existen en la base de datos, junto con los productos que tiene cada uno de ellos. El listado deberá mostrar también aquellos fabricantes que no tienen productos asociados.
2. Devuelve un listado donde sólo aparezcan aquellos fabricantes que no tienen ningún producto asociado.

Subconsultas (En la cláusula WHERE)

Con operadores básicos de comparación

1. Devuelve todos los productos del fabricante Lenovo. (Sin utilizar INNER JOIN).
2. Devuelve todos los datos de los productos que tienen el mismo precio que el producto más caro del fabricante Lenovo. (Sin utilizar INNER JOIN).
3. Lista el nombre del producto más caro del fabricante Lenovo.
4. Lista todos los productos del fabricante Asus que tienen un precio superior al precio medio de todos sus productos.

Subconsultas con IN y NOT IN

1. Devuelve los nombres de los fabricantes que tienen productos asociados. (Utilizando IN o NOT IN).
2. Devuelve los nombres de los fabricantes que no tienen productos asociados. (Utilizando IN o NOT IN).

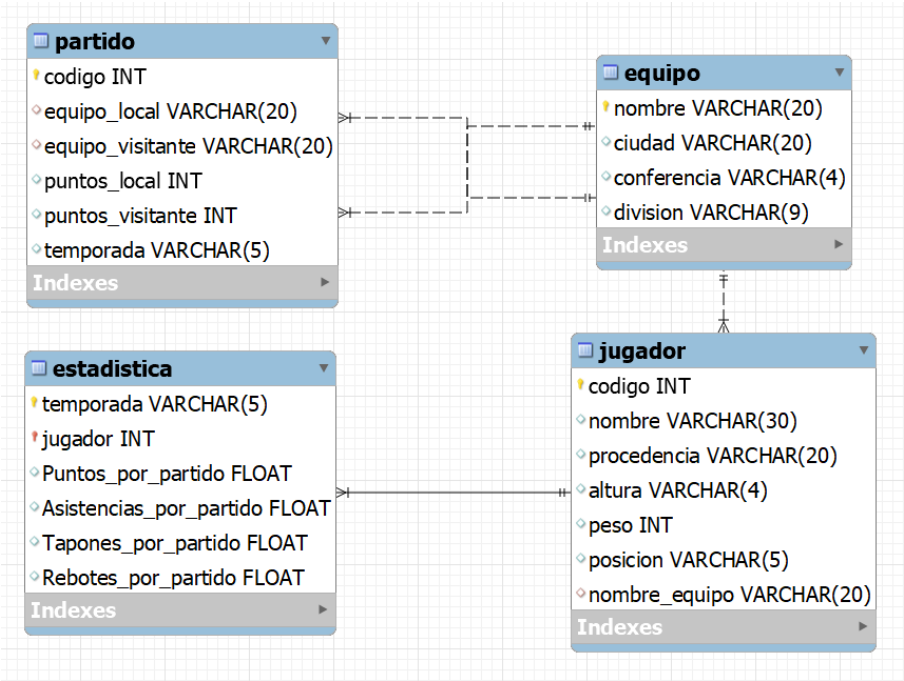
Subconsultas (En la cláusula HAVING)

1. Devuelve un listado con todos los nombres de los fabricantes que tienen el mismo número de productos que el fabricante Lenovo.

EJERCICIOS DE APRENDIZAJE EXTRA

Estos van a ser ejercicios para reforzar los conocimientos previamente vistos. Estos pueden realizarse cuando hayas terminado la guía y tengas una buena base sobre lo que venimos trabajando. Además, si ya terminaste la guía y te queda tiempo libre en las mesas, podes continuar con estos ejercicios extra, recordando siempre que no es necesario que los termines para continuar con el tema siguiente. Por último, recordá que la prioridad es ayudar a los compañeros de la mesa y que cuando tengas que ayudar, lo más valioso es que puedas explicar el ejercicio con la intención de que tu compañero lo comprenda, y no sólo mostrarlo. ¡Muchas gracias!

1. Abrir el script de la base de datos llamada “nba.sql” y ejecutarlo para crear todas las tablas e insertar datos en las mismas. A continuación, generar el modelo de entidad relación. Deberá obtener un diagrama de entidad relación igual al que se muestra a continuación:

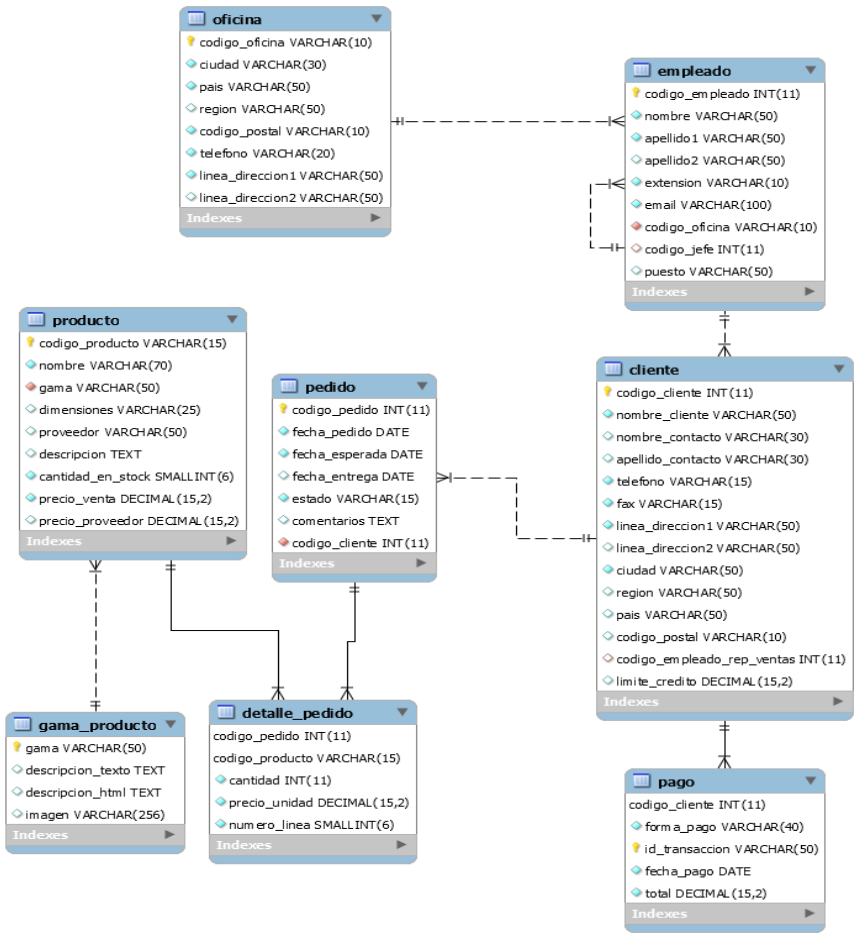


A continuación, se deben realizar las siguientes consultas sobre la base de datos:

1. Mostrar el nombre de todos los jugadores ordenados alfabéticamente.
2. Mostrar el nombre de los jugadores que sean pivots (‘C’) y que pesen más de 200 libras, ordenados por nombre alfabéticamente.
3. Mostrar el nombre de todos los equipos ordenados alfabéticamente.
4. Mostrar el nombre de los equipos del este (East).
5. Mostrar los equipos donde su ciudad empieza con la letra ‘c’, ordenados por nombre.
6. Mostrar todos los jugadores y su equipo ordenados por nombre del equipo.
7. Mostrar todos los jugadores del equipo “Raptors” ordenados por nombre.
8. Mostrar los puntos por partido del jugador ‘Pau Gasol’.
9. Mostrar los puntos por partido del jugador ‘Pau Gasol’ en la temporada ’04/05’.
10. Mostrar el número de puntos de cada jugador en toda su carrera.
11. Mostrar el número de jugadores de cada equipo.

12. Mostrar el jugador que más puntos ha realizado en toda su carrera.
13. Mostrar el nombre del equipo, conferencia y división del jugador más alto de la NBA.
14. Mostrar la media de puntos en partidos de los equipos de la división Pacific.
15. Mostrar el partido o partidos (equipo_local, equipo_visitante y diferencia) con mayor diferencia de puntos.
16. Mostrar la media de puntos en partidos de los equipos de la división Pacific.
17. Mostrar los puntos de cada equipo en los partidos, tanto de local como de visitante.
18. Mostrar quien gana en cada partido (codigo, equipo_local, equipo_visitante, equipo_ganador), en caso de empate sera null.

2. Abrir el script de la base de datos llamada “jardineria.sql” y ejecutarlo para crear todas las tablas e insertar datos en las mismas. Deberá obtener un diagrama de entidad relación igual al que se muestra a continuación:



A continuación, se deben realizar las siguientes consultas sobre la base de datos:

Consultas sobre una tabla

1. Devuelve un listado con el código de oficina y la ciudad donde hay oficinas.
2. Devuelve un listado con la ciudad y el teléfono de las oficinas de España.
3. Devuelve un listado con el nombre, apellidos y email de los empleados cuyo jefe tiene un código de jefe igual a 7.

4. Devuelve el nombre del puesto, nombre, apellidos y email del jefe de la empresa.
5. Devuelve un listado con el nombre, apellidos y puesto de aquellos empleados que no sean representantes de ventas.
6. Devuelve un listado con el nombre de los todos los clientes españoles.
7. Devuelve un listado con los distintos estados por los que puede pasar un pedido.
8. Devuelve un listado con el código de cliente de aquellos clientes que realizaron algún pago en 2008. Tenga en cuenta que deberá eliminar aquellos códigos de cliente que aparezcan repetidos. Resuelva la consulta:
    - Utilizando la función YEAR de MySQL.
    - Utilizando la función DATE_FORMAT de MySQL.
    - Sin utilizar ninguna de las funciones anteriores.
9. Devuelve un listado con el código de pedido, código de cliente, fecha esperada y fecha de entrega de los pedidos que no han sido entregados a tiempo.
10. Devuelve un listado con el código de pedido, código de cliente, fecha esperada y fecha de entrega de los pedidos cuya fecha de entrega ha sido al menos dos días antes de la fecha esperada.
    - Utilizando la función ADDDATE de MySQL.
    - Utilizando la función DATEDIFF de MySQL.
11. Devuelve un listado de todos los pedidos que fueron rechazados en 2009.
12. Devuelve un listado de todos los pedidos que han sido entregados en el mes de enero de cualquier año.
13. Devuelve un listado con todos los pagos que se realizaron en el año 2008 mediante Paypal. Ordene el resultado de mayor a menor.
14. Devuelve un listado con todas las formas de pago que aparecen en la tabla pago. Tenga en cuenta que no deben aparecer formas de pago repetidas.
15. Devuelve un listado con todos los productos que pertenecen a la gama Ornamentales y que tienen más de 100 unidades en stock. El listado deberá estar ordenado por su precio de venta, mostrando en primer lugar los de mayor precio.
16. Devuelve un listado con todos los clientes que sean de la ciudad de Madrid y cuyo representante de ventas tenga el código de empleado 11 o 30.

Consultas multitabla (Composición interna)

Las consultas se deben resolver con INNER JOIN.

1. Obtén un listado con el nombre de cada cliente y el nombre y apellido de su representante de ventas.
2. Muestra el nombre de los clientes que hayan realizado pagos junto con el nombre de sus representantes de ventas.
3. Muestra el nombre de los clientes que no hayan realizado pagos junto con el nombre de sus representantes de ventas.
4. Devuelve el nombre de los clientes que han hecho pagos y el nombre de sus representantes junto con la ciudad de la oficina a la que pertenece el representante.
5. Devuelve el nombre de los clientes que no hayan hecho pagos y el nombre de sus representantes junto con la ciudad de la oficina a la que pertenece el representante.

6. Lista la dirección de las oficinas que tengan clientes en Fuenlabrada.
7. Devuelve el nombre de los clientes y el nombre de sus representantes junto con la ciudad de la oficina a la que pertenece el representante.
8. Devuelve un listado con el nombre de los empleados junto con el nombre de sus jefes.
9. Devuelve el nombre de los clientes a los que no se les ha entregado a tiempo un pedido.
10. Devuelve un listado de las diferentes gamas de producto que ha comprado cada cliente.

Consultas multitabla (Composición externa)

Resuelva todas las consultas utilizando las cláusulas LEFT JOIN, RIGHT JOIN, JOIN.

1. Devuelve un listado que muestre solamente los clientes que no han realizado ningún pago.
2. Devuelve un listado que muestre solamente los clientes que no han realizado ningún pedido.
3. Devuelve un listado que muestre los clientes que no han realizado ningún pago y los que no han realizado ningún pedido.
4. Devuelve un listado que muestre solamente los empleados que no tienen una oficina asociada.
5. Devuelve un listado que muestre solamente los empleados que no tienen un cliente asociado.
6. Devuelve un listado que muestre los empleados que no tienen una oficina asociada y los que no tienen un cliente asociado.
7. Devuelve un listado de los productos que nunca han aparecido en un pedido.
8. Devuelve las oficinas donde no trabajan ninguno de los empleados que hayan sido los representantes de ventas de algún cliente que haya realizado la compra de algún producto de la gama Frutales.
9. Devuelve un listado con los clientes que han realizado algún pedido, pero no han realizado ningún pago.
10. Devuelve un listado con los datos de los empleados que no tienen clientes asociados y el nombre de su jefe asociado.

Consultas resumen

1. ¿Cuántos empleados hay en la compañía?
2. ¿Cuántos clientes tiene cada país?
3. ¿Cuál fue el pago medio en 2009?
4. ¿Cuántos pedidos hay en cada estado? Ordena el resultado de forma descendente por el número de pedidos.
5. Calcula el precio de venta del producto más caro y más barato en una misma consulta.
6. Calcula el número de clientes que tiene la empresa.
7. ¿Cuántos clientes tiene la ciudad de Madrid?
8. ¿Calcula cuántos clientes tiene cada una de las ciudades que empiezan por M?
9. Devuelve el nombre de los representantes de ventas y el número de clientes al que atiende cada uno.

10. Calcula el número de clientes que no tiene asignado representante de ventas.

11. Calcula la fecha del primer y último pago realizado por cada uno de los clientes. El listado deberá mostrar el nombre y los apellidos de cada cliente.

12. Calcula el número de productos diferentes que hay en cada uno de los pedidos.

13. Calcula la suma de la cantidad total de todos los productos que aparecen en cada uno de los pedidos.

14. Devuelve un listado de los 20 productos más vendidos y el número total de unidades que se han vendido de cada uno. El listado deberá estar ordenado por el número total de unidades vendidas.

15. La facturación que ha tenido la empresa en toda la historia, indicando la base imponible, el IVA y el total facturado. La base imponible se calcula sumando el coste del producto por el número de unidades vendidas de la tabla detalle_pedido. El IVA es el 21 % de la base imponible, y el total la suma de los dos campos anteriores.

16. La misma información que en la pregunta anterior, pero agrupada por código de producto.

17. La misma información que en la pregunta anterior, pero agrupada por código de producto filtrada por los códigos que empiecen por OR.

18. Lista las ventas totales de los productos que hayan facturado más de 3000 euros. Se mostrará el nombre, unidades vendidas, total facturado y total facturado con impuestos (21% IVA)

Subconsultas con operadores básicos de comparación

1. Devuelve el nombre del cliente con mayor límite de crédito.

2. Devuelve el nombre del producto que tenga el precio de venta más caro.

3. Devuelve el nombre del producto del que se han vendido más unidades. (Tenga en cuenta que tendrá que calcular cuál es el número total de unidades que se han vendido de cada producto a partir de los datos de la tabla detalle_pedido. Una vez que sepa cuál es el código del producto, puede obtener su nombre fácilmente.)

4. Los clientes cuyo límite de crédito sea mayor que los pagos que haya realizado. (Sin utilizar INNER JOIN).

5. Devuelve el producto que más unidades tiene en stock.

6. Devuelve el producto que menos unidades tiene en stock.

7. Devuelve el nombre, los apellidos y el email de los empleados que están a cargo de Alberto Soria.

Subconsultas con ALL y ANY

1. Devuelve el nombre del cliente con mayor límite de crédito.

2. Devuelve el nombre del producto que tenga el precio de venta más caro.

3. Devuelve el producto que menos unidades tiene en stock.

Subconsultas con IN y NOT IN

1. Devuelve el nombre, apellido1 y cargo de los empleados que no representen a ningún cliente.

2. Devuelve un listado que muestre solamente los clientes que no han realizado ningún pago.

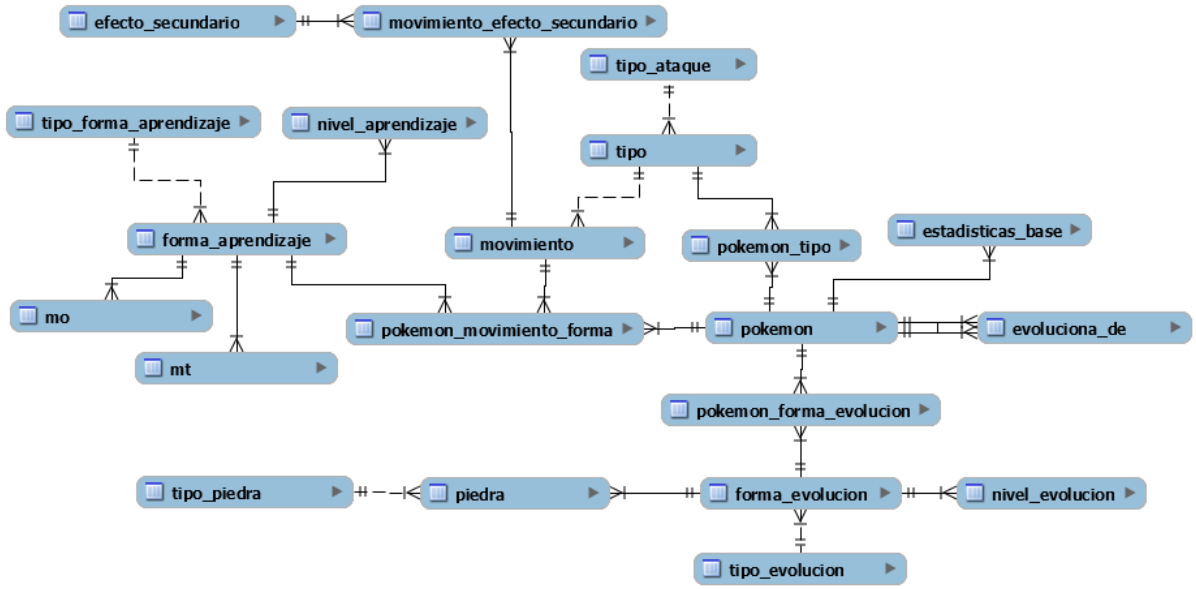
3. Devuelve un listado que muestre solamente los clientes que sí han realizado ningún pago.

4. Devuelve un listado de los productos que nunca han aparecido en un pedido.
5. Devuelve el nombre, apellidos, puesto y teléfono de la oficina de aquellos empleados que no sean representante de ventas de ningún cliente.

Subconsultas con EXISTS y NOT EXISTS

1. Devuelve un listado que muestre solamente los clientes que no han realizado ningún pago.
2. Devuelve un listado que muestre solamente los clientes que sí han realizado ningún pago.
3. Devuelve un listado de los productos que nunca han aparecido en un pedido.
4. Devuelve un listado de los productos que han aparecido en un pedido alguna vez.

✓ 3. Importar el script de la base de datos llamada “pokemondb.sql” y ejecutarlo para crear todas las tablas e insertar los registros en las mismas. A continuación, generar el modelo de entidad relación y reorganizar las tablas para mayor claridad de sus relaciones. Deberá obtener un diagrama de entidad de relación similar al que se muestra a continuación:



A continuación, se deben realizar las siguientes consultas:

1. Mostrar el nombre de todos los pokemon.
2. Mostrar los pokemon que pesen menos de 10k.
3. Mostrar los pokemon de tipo agua.
4. Mostrar los pokemon de tipo agua, fuego o tierra ordenados por tipo.
5. Mostrar los pokemon que son de tipo fuego y volador.
6. Mostrar los pokemon con una estadística base de ps mayor que 200.
7. Mostrar los datos (nombre, peso, altura) de la prevolución de Arbok.
8. Mostrar aquellos pokemon que evolucionan por intercambio.
9. Mostrar el nombre del movimiento con más prioridad.

10. Mostrar el pokemon más pesado.
11. Mostrar el nombre y tipo del ataque con más potencia.
12. Mostrar el número de movimientos de cada tipo.
13. Mostrar todos los movimientos que puedan envenenar.
14. Mostrar todos los movimientos que causan daño, ordenados alfabéticamente por nombre.
15. Mostrar todos los movimientos que aprende pikachu.
16. Mostrar todos los movimientos que aprende pikachu por MT (tipo de aprendizaje).
17. Mostrar todos los movimientos de tipo normal que aprende pikachu por nivel.
18. Mostrar todos los movimientos de efecto secundario cuya probabilidad sea mayor al 30%.
19. Mostrar todos los pokemon que evolucionan por piedra.
20. Mostrar todos los pokemon que no pueden evolucionar.
21. Mostrar la cantidad de los pokemon de cada tipo.

BIBLIOGRAFÍA

Información sacada de las páginas:

- https://www.oracle.com/ar/database/what-is-a-relational-database/

- https://www.geeksforgeeks.org/sql-tutorial/

- https://count.co/blog/posts/take-your-sql-from-good-to-great-part-3

- https://jorgesanchez.net/

- https://bookdown.org/paranedagarcia/database/modelo-relacional.html

- https://www.datacentric.es/

INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (1,'Bruce Banner', 'Hulk', 160, '600 mil', 75,98,1962,'Fisico Nuclear', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (2,'Tony Stark', 'Iron man', 170, '200 mil', 70,123,1963,'Inventor Industrial',
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (3,'Thor Odinson ', 'Thor', 145, 'infinita', 100,235,1962,'Rey de Asgard', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (4,' Wanda Maximoff', 'Bruja Escarlata', 170, '100 mil', 90,345,1964,'Bruja',
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (5,' Carol Danvers', 'Capitana Marvel', 157, '250 mil', 85,128,1968,'Oficial d
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (6,'Thanos', 'Thanos', 170, 'infinita', 40,306,1973,'Adorador de la muerte', 1
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (7,'Peter Parker', 'Spiderman', 165, '25 mil', 80,74,1962,'Fotografo', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (8,'Steve Rogers', 'Capitan America', 145, '600', 45,60,1941,'Oficial Federa
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (9,'Bobby Drake', 'Iceman', 140, '2 mil', 64,122,1963,'Contador', 1);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (10,'Barry Allen', 'Flash', 160, '10 mil', 120,168,1956,'Cientifico Forense', 2
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (11,'Bruce Wayne', 'Batman', 170, '500', 32,47,1939,'Hombre de Negocios
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (12,'Clarck Kent', 'Superman', 165, 'infinita', 120,182,1948,'Reportero', 2);
INSERT INTO personajes (id_personaje, nombre_real, personaje, inteligencia, fuerza, velocidad, poder, aparicion, ocupacion,id_creador) values (13,'Diana Prince', 'Mujer Maravilla', 160, 'infinita', 95,127,1949,'Princesa C