***Fundamentos de Base de Datos.***

**Introducción.**

El almacenamiento en la nube tiene un gran pro comparada con los otros métodos de almacenamiento ya que es accesible desde cualquier parte del mundo. Además es centralizada y puede ser usada por varias personas al mismo tiempo.

Las bases de datos entran cuando hacemos la transición a medios digitales.

**Relacionales:** En la industria hay varias compañías dedicadas a ser manejadoras de bases de datos relacionales como **SQL Server, Oracle, MariaDB**, entre otras.  
**No relacionales:** Todavía están avanzando y existen ejemplos muy distintos como **cassandra, elasticsearch, neo4j, MongoDB**, entre otras.

**Servicios:**

* Auto administrados: Es la base de datos que instalas tú y te encargas de actualizaciones, mantenimiento, etc.
* Administrados: Servicios que ofrecen las nubes modernas como Azure y no debes preocuparte por mantenimiento o actualizaciones.

**Historia de las RDB(Relational Data Base)**

Las bases de datos surgen de la necesidad de conservar la información más allá de lo que existe en la memoria RAM.

Las bases de datos **basadas en archivos** eran datos guardados en texto plano, fáciles de guardar pero muy difíciles de consultar y por la necesidad de mejorar esto nacen las **bases de datos relacionales**. Su inventor **Edgar Codd** dejó ciertas reglas para asegurarse de que toda la filosofía de las bases de datos no se perdiera, estandarizando el proceso.

<https://medievalstrucos.wordpress.com/2013/07/18/12-reglas-de-codd-para-bases-de-datos-relacionadas/>

# Entidades y atributos

Una **entidad** es algo similar a un objeto (programación orientada a objetos) y representa algo en el mundo real, incluso algo abstracto. Tienen atributos que son las cosas que los hacen ser una entidad y por convención se ponen en plural.

Los **atributos compuestos** son aquellos que tienen atributos ellos mismos.

Los **atributos llave** son aquellos que identifican a la entidad y no pueden ser repetidos. Existen:

* Naturales: Son inherentes al objeto como el número de serie
* Clave artificial: No es inherente al objeto y se asigna de manera arbitraria.(**ID**)

**Entidades débiles:** No pueden existir sin una entidad fuerte y se representan con un cuadrado con doble línea.

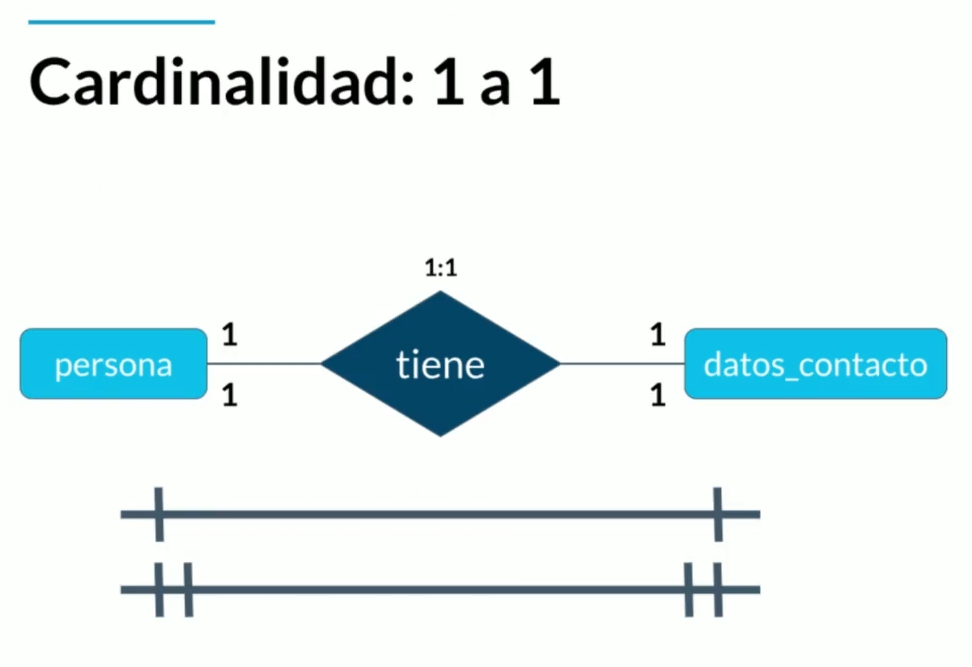
* Identidades débiles por identidad: No se diferencian entre sí más que por la clave de su identidad fuerte.
* Identidades débiles por existencia: Se les asigna una clave propia.

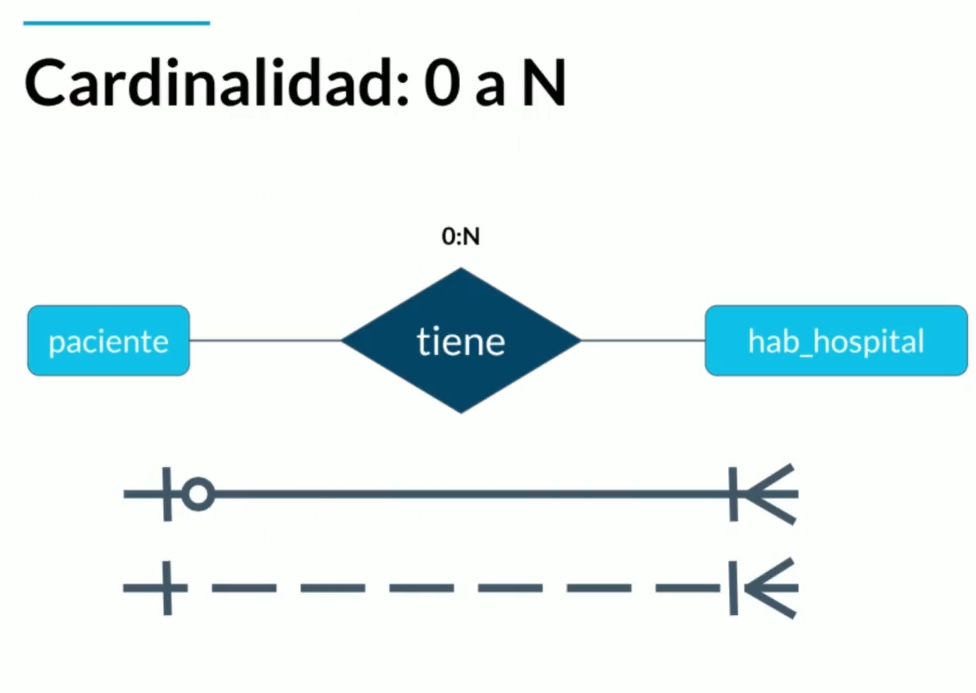
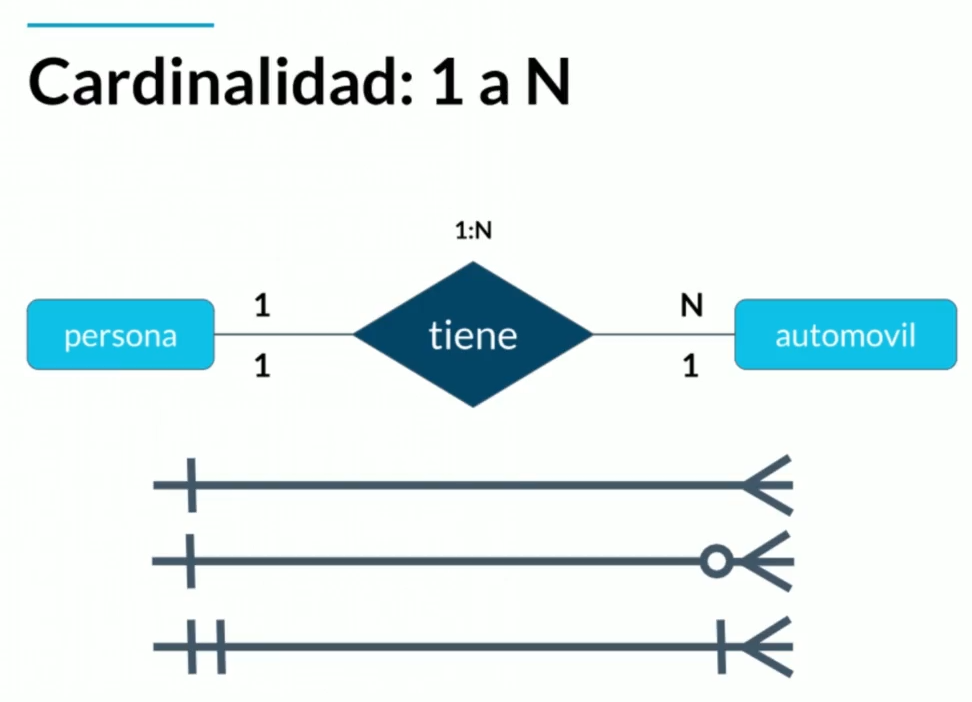
**Relaciones**

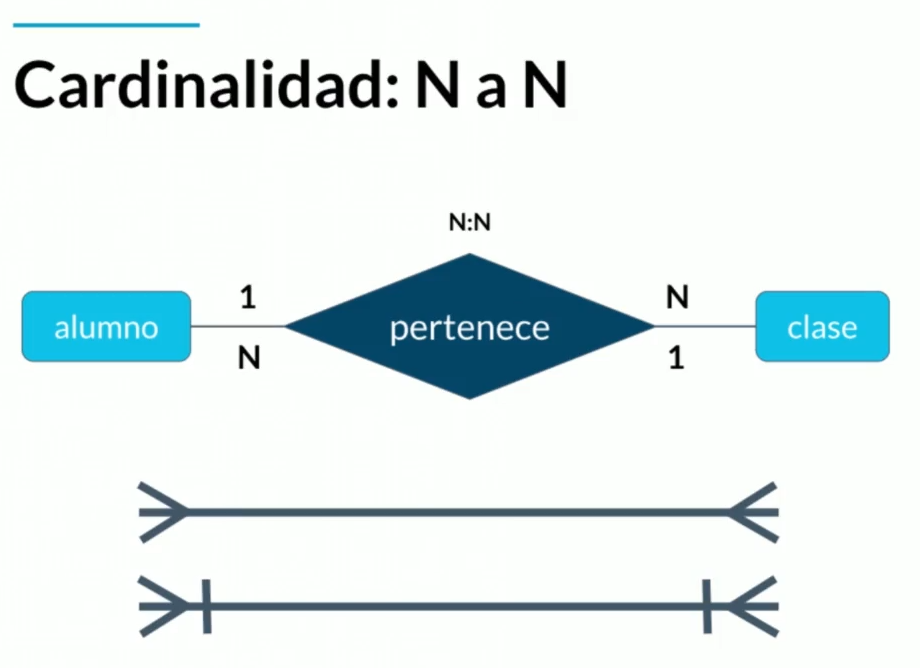
Las **relaciones** nos permiten ligar o unir nuestras diferentes entidades y se representan con rombos. Por convención se definen a través de verbos.

Las relaciones tienen una propiedad llamada **cardinalidad** y tiene que ver con números. Cuántos de un lado pertenecen a cuántos del otro lado:

* Cardinalidad: 1 a 1
* Cardinalidad: 0 a 1
* Cardinalidad: 1 a N
* Cardinalidad: 0 a N
* Cardinalidad: N a N



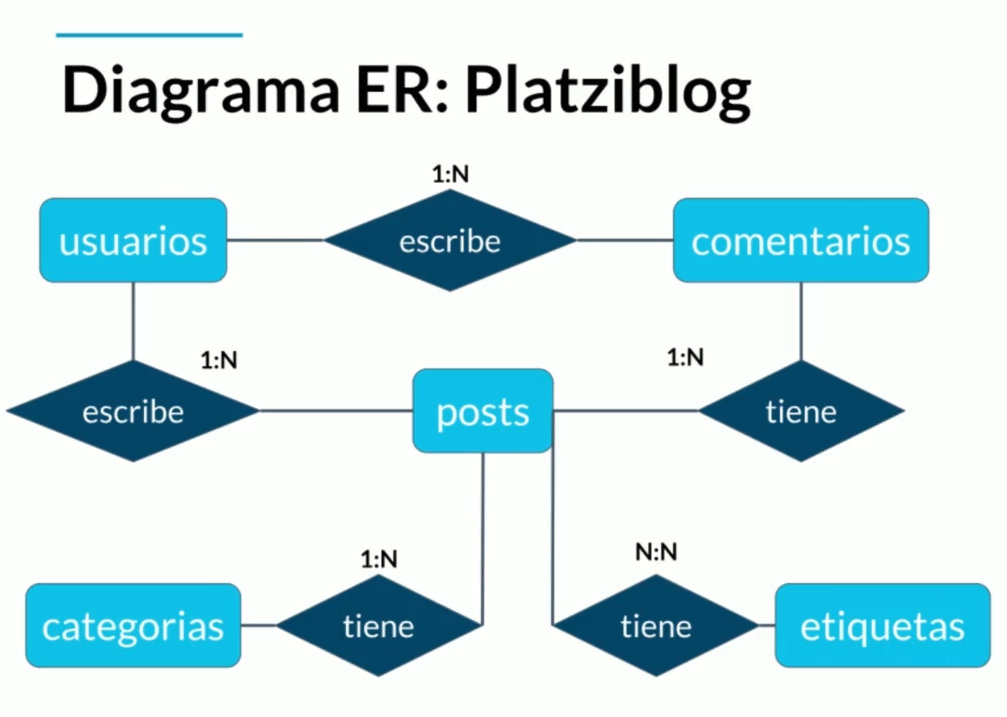
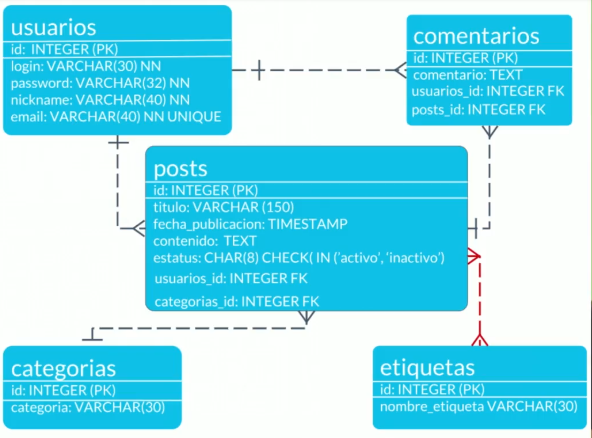




# Diagrama ER.

Un diagrama es como un mapa y nos ayuda a entender cuáles son las entidades con las que vamos a trabajar, cuáles son sus relaciones y qué papel van a jugar en las aplicaciones de la base de datos.

<https://app.diagrams.net/>

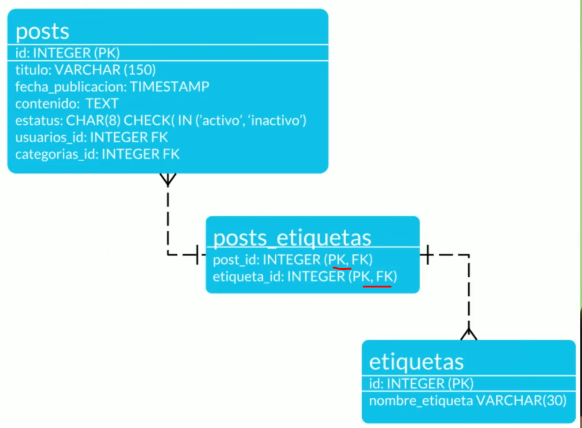
 

**Muchos a Muchos.**

En el ejemplo anterior tenemos un caso marcado con línea roja en el que nos encontramos con esta particularidad.

Como una opción debemos generar una nueva tabla donde pueda registrar la unión entre estas dos tablas generando adicionalmente una llave artificial agregada (**ID**) que es lo que se haría normalmente.

Sin embargo existe la creación de una Clave compuesta en la que tiene la clave de ambas entidades como se puede visualizar en la siguiente imagen.

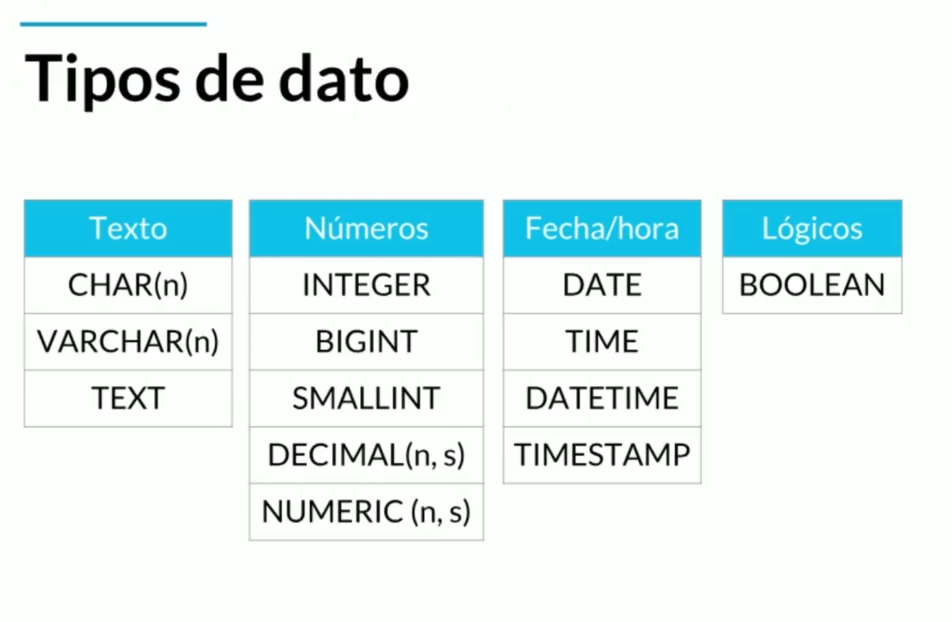


# Tipos de datos y Constraints.

Para llevar a la práctica un diagrama debemos ir más allá y darle detalle con parámetros como:

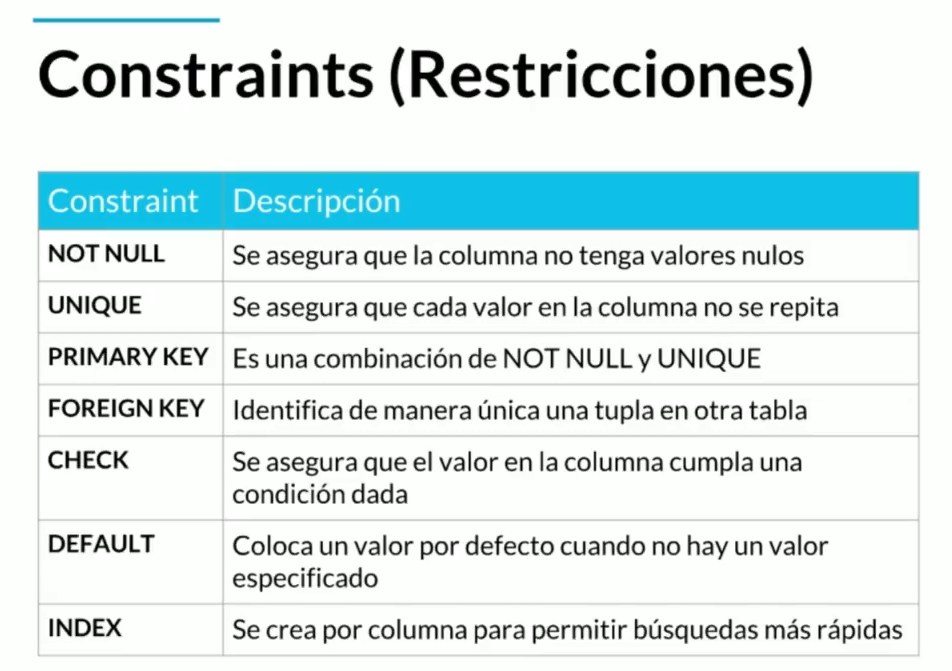
**Tipos de dato:**

* **Texto:** CHAR(n), VARCHAR(n), TEXT
* **Números:** INTEGER, BIGINT, SMALLINT, DECIMAL(n,s), NUMERIC(n,s)
* **Fecha/hora:** DATE, TIME, DATETIME, TIMESTAMP
* **Lógicos:** BOOLEAN

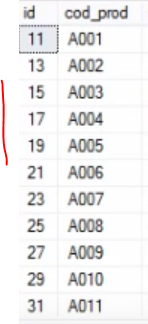
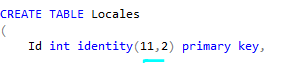


**Constraints (Restricciones)**

* **NOT NULL:** Se asegura que la columna no tenga valores nulos
* **UNIQUE:** Se asegura que cada valor en la columna no se repita
* **CHECK:** Se asegura que el valor en la columna cumpla una condición dada
* **DEFAULT:** Coloca un valor por defecto cuando no hay un valor especificado
* **INDEX:** Se crea por columna para permitir búsquedas más rápidas



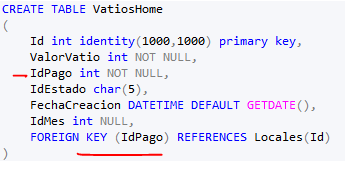
* **Identity(11,2):** Cuando asignamos el identity a una columna podemos determinar el numero con el que va a iniciar ‘**11**’, y el intervalo con el que va ir aumentando ‘2’

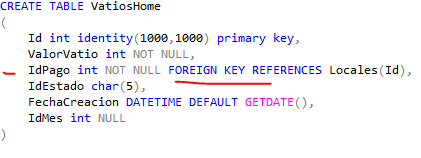


* **PRIMARY KEY:** Es una combinación de NOT NULL y UNIQUE. Automáticamente genera un INDEX en esa columna. Para consultar un INDEX:



* **FOREIGN KEY:** Identifica de manera única una tupla en otra tabla





**Tablas independientes**

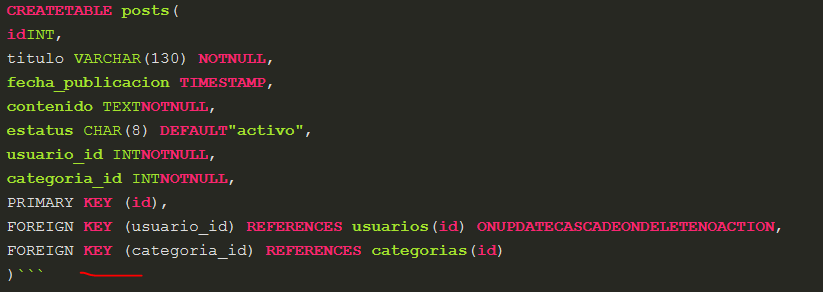
* Una buena práctica es comenzar creando las entidades que no tienen una llave foránea.
* Generalmente en los nombres de bases de datos se evita usar eñes o acentos para evitar problemas en los manejadores de las bases de datos.

**Tablas dependientes**

* El comando “cascade” sirve para que cada que se haga un update en la tabla principal, se refleje también en la tabla en la que estamos creando la relación.

Las Foreing Key options son las siguientes:

**On update:** Significa qué pasará con las relaciones cuando una de estas sea modificada en sus campos relacionados, Por ejemplo, pueden utilizarse los valores:  
cascade: Si el id de un usuario pasa de 11 a 12, entonces la relacion se actualizará y el post buscará el id nuevo en lugar de quedarse sin usuario.  
\_ restrict: \_Si el id de un usuario pasa de 11 a 12, no lo permitirá hasta que no sean actualizados antes todos los post relacionados.  
set null: Si el id de un usuario pasa de 11 a 12, entonces los post solo no estará relacionados con nada.no action: Si el id de un usuario pasa de 11 a 12, no se hará nada. Solo se romperá la relación.***On delete***\_ cascade: Si un usuario es eliminado entonces se borrarán todos los post relacionados.restrict: No se podrá eliminar un usuario hasta que sean eliminados todos su post relacionados.\_ set null: Si un usuario es eliminado, entonces los post solo no estará relacionados con nada.  
no action: Si un usuario es eliminado, no se hará nada. Solo se romperá la relación.



# Tablas transitivas

* Las tablas transitivas sirven como puente para unir dos tablas. No tienen contenido semántico.
* **Reverse Engineer** nos reproduce el esquema del cual nos basamos para crear nuestras tablas. Es útil cuando llegas a un nuevo trabajo y quieres entender cuál fue la mentalidad que tuvieron al momento de crear las bases de datos.

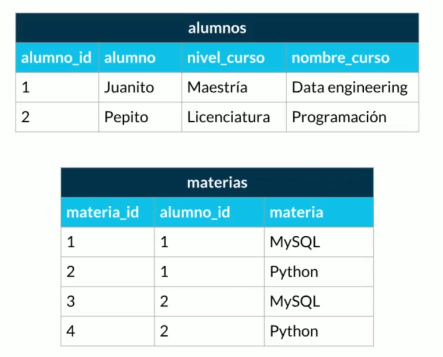
**Normalización.**

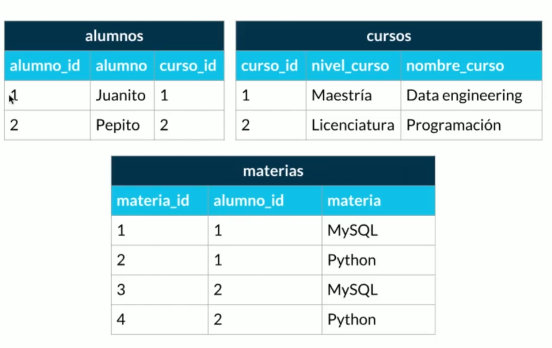
La normalización como su nombre lo indica nos ayuda a dejar todo de una forma normal. Esto obedece a las 12 reglas de **Codd** y nos permiten separar componentes en la base de datos:



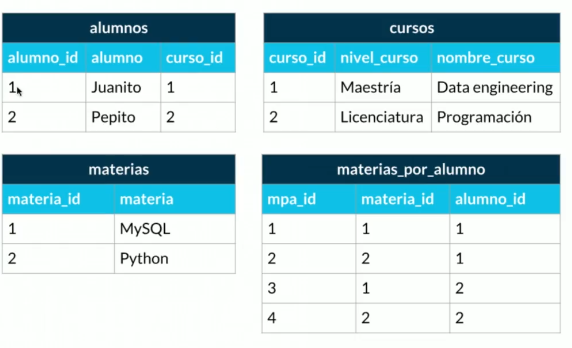
* **Primera forma normal (1FN):** Atributos atómicos (Sin campos repetidos)



* **Segunda forma normal (2FN):** Cumple 1FN y cada campo de la tabla debe depender de una clave única.
* 
* **Tercera forma normal (3FN):** Cumple 1FN y 2FN y los campos que NO son clave, NO deben tener dependencias.



* **Cuarta forma normal (4FN):** Cumple 1FN, 2FN, 3FN y los campos multivaluados se identifican por una clave única.

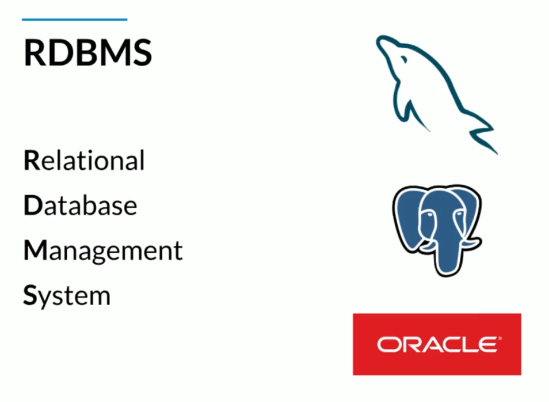


**Para información mas detallada de este tema revisar el archivo:**

*Formas normales en DB relacionales.docx*

**RBDMS.**

**RDBMS** significa **R**elational **D**atabase **M**anagement **S**ystem o sistema manejador de bases de datos relacionales. Es un programa que se encarga de seguir las reglas de **Codd** y se puede utilizar de manera programática.



**Instalación local de un RDBMS (Windows)**

Hay dos maneras de acceder a manejadores de bases de datos:

* Instalar en máquina local un administrador de bases relacional.
* Tener ambientes de desarrollo especiales o servicios cloud.

En este curso usaremos **MySQL** porque tiene un impacto histórico siendo muy utilizado y además es software libre y gratuito. La versión 5.6.43 es compatible con la mayoría de aplicaciones y frameworks.

* Root es el usuario principal que tendrá todos los permisos y por lo tanto en ambientes de producción hay que tener **mucho cuidado** al configurarlo.

<https://dev.mysql.com/downloads/mysql/5.7.html#downloads>

# Historia de SQL

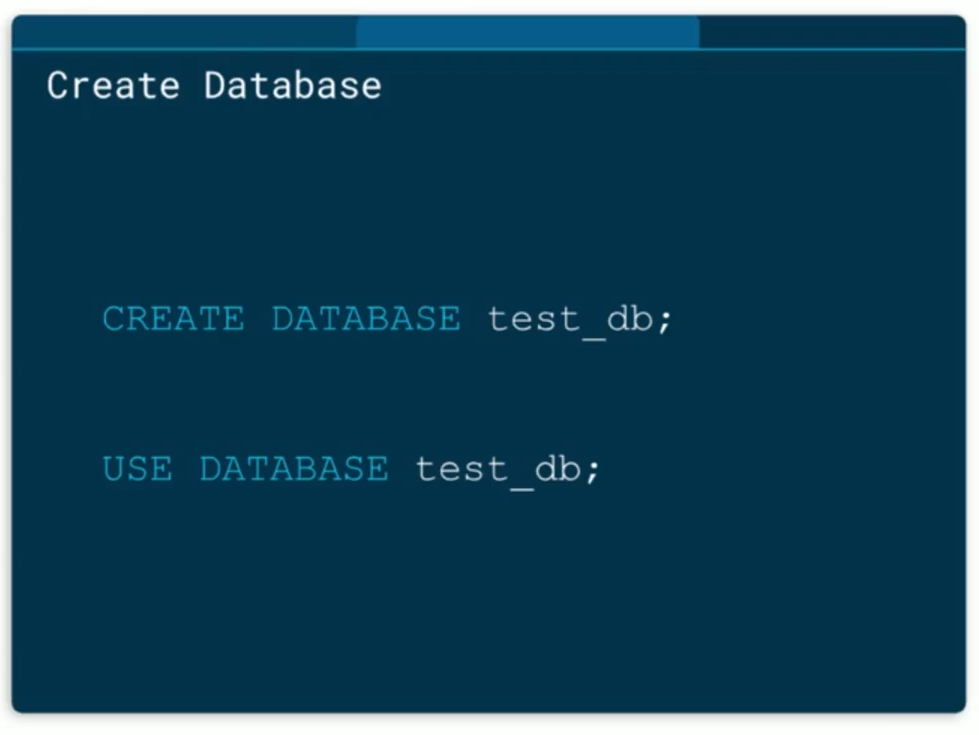
**SQL** significa **S**tructured **Q**uery **L**anguage y tiene una estructura clara y fija. Su objetivo es hacer un solo lenguaje para consultar cualquier manejador de bases de datos volviéndose un gran estándar.

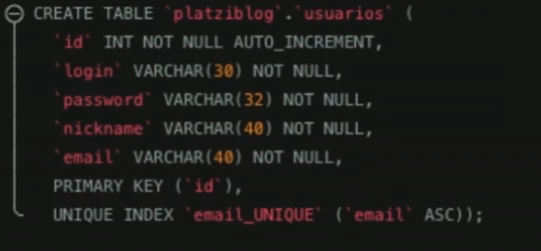
Ahora existe el **NOSQL** o **N**ot **O**nly **S**tructured **Q**uery **L**anguage que significa que no sólo se utiliza SQLen las bases de datos no relacionales.

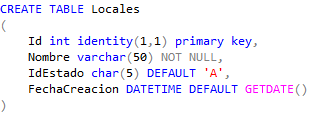
**DDL**

**SQL** tiene dos grandes sublenguajes:  
**DDL** o Data Definition Language que nos ayuda a crear la estructura de una base de datos. Existen 3 grandes comandos:

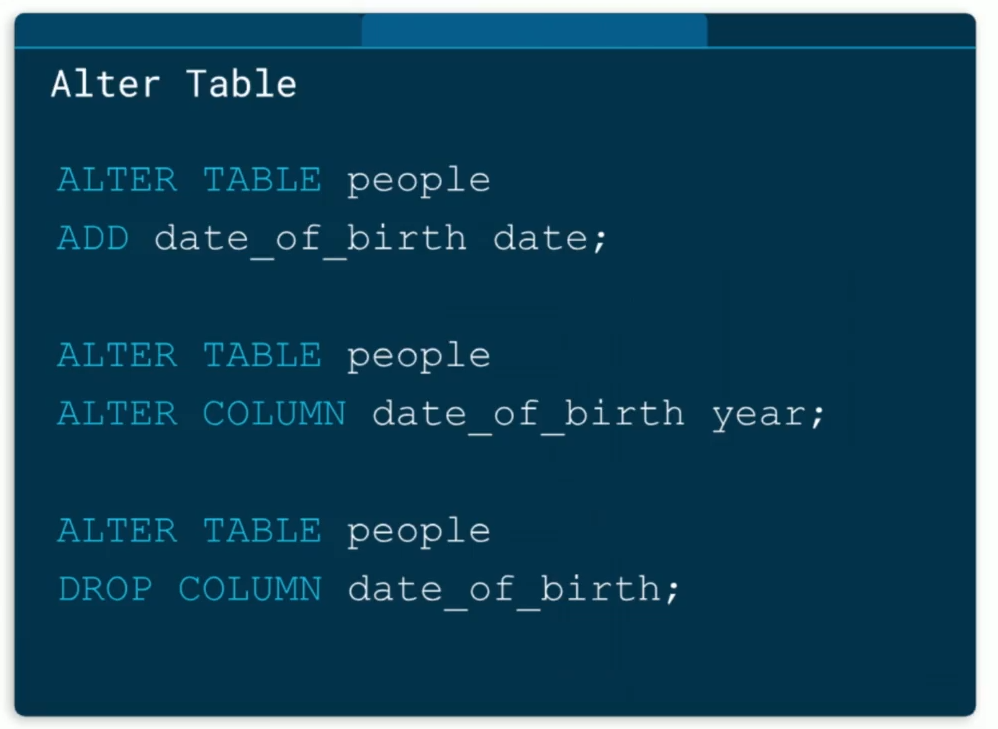
* Create: Nos ayuda a crear bases de datos, tablas, vistas, índices, etc.



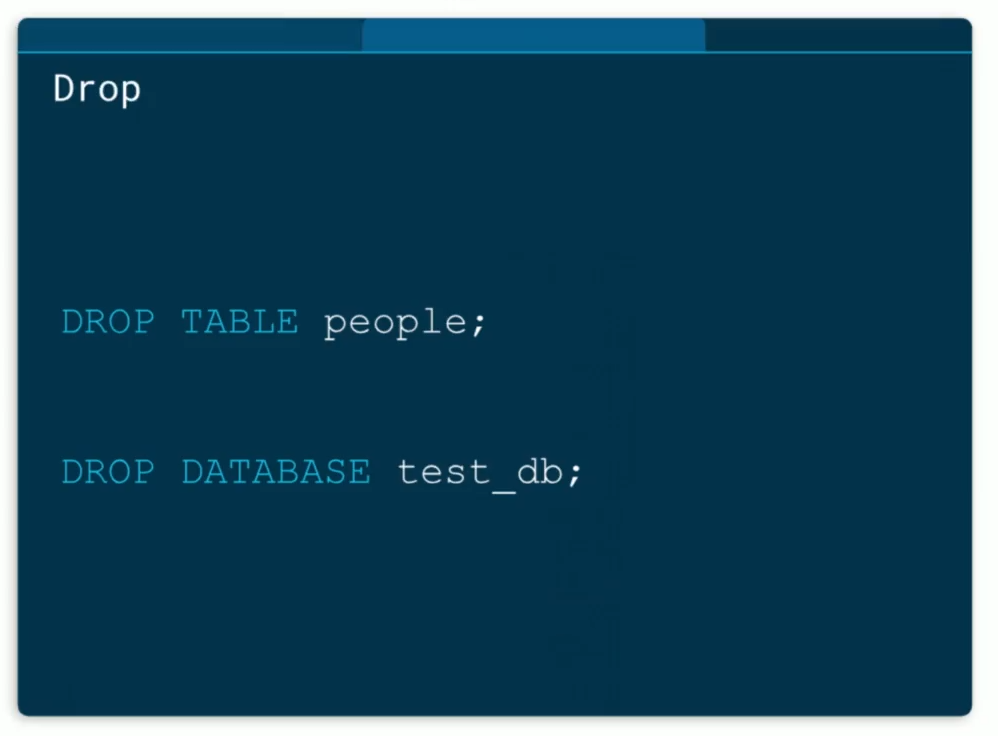




* Alter: Ayuda a alterar o modificar entidades.



* Drop: Nos ayuda a borrar. Hay que tener cuidado al utilizarlo.



**3 objetos que manipularemos con el lenguaje DDL:**

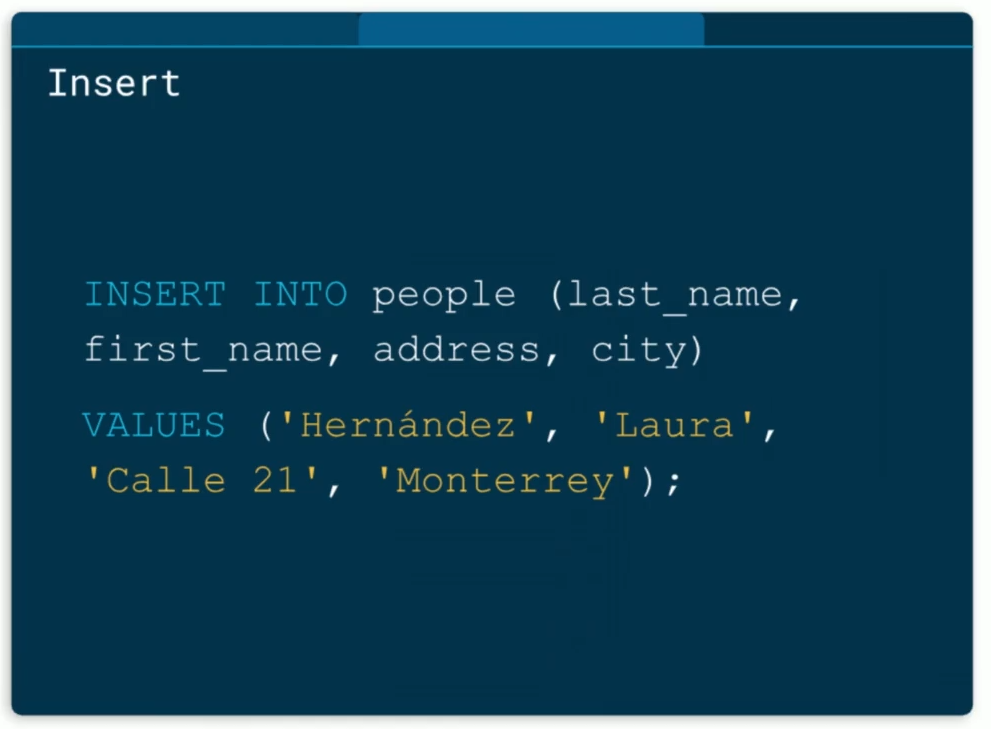
* Database o bases de datos
* Table o tablas. Son la traducción a SQL de las entidades
* View o vistas: Se ofrece la proyección de los datos de la base de datos de forma ente ndible.



**DML**

**DML** trata del contenido de la base de datos. Son las siglas de Data Manipulation Language y sus comandos son:

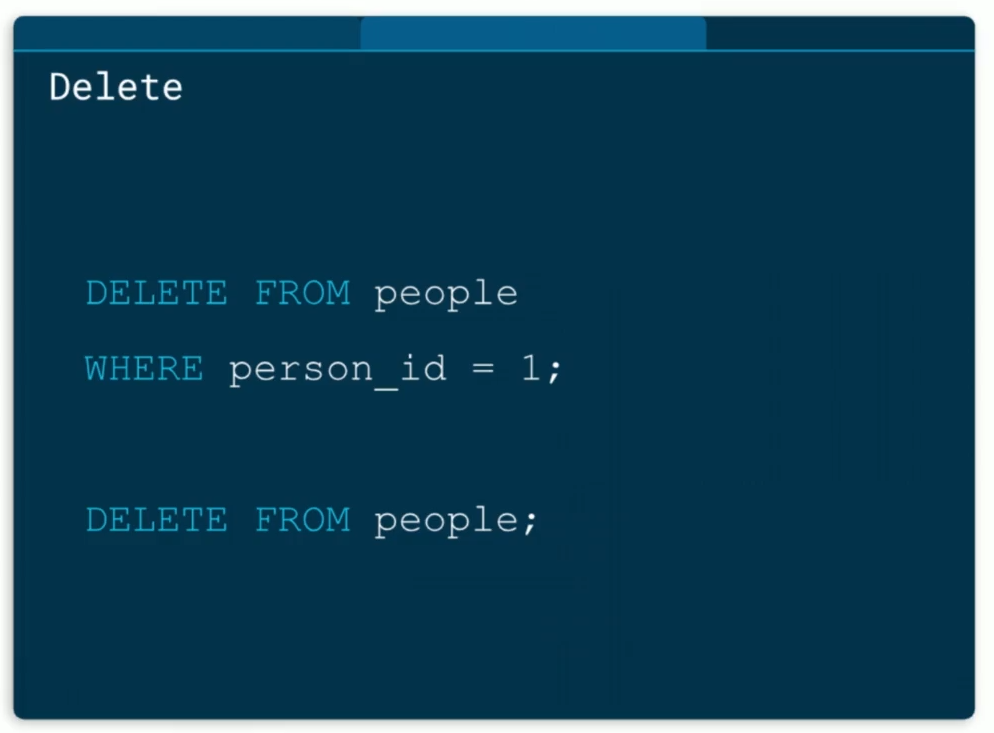
* **Insert:** Inserta o agrega nuevos registros a la tabla.



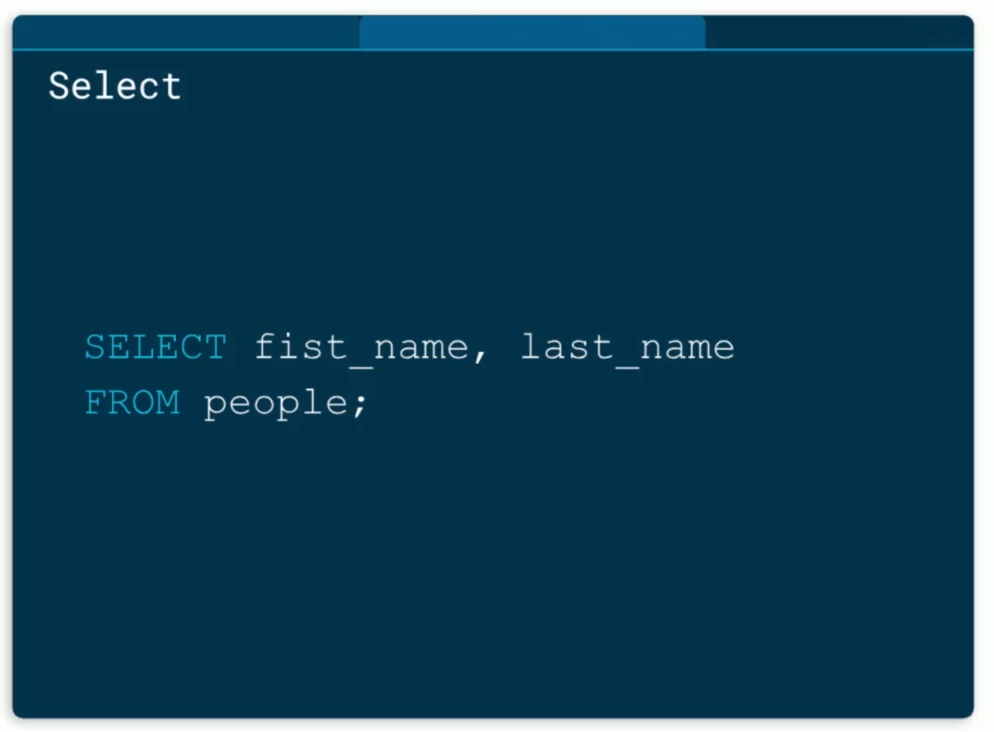
* **Update:** Actualiza o modifica los datos que ya existen.



* **Delete:** Esta sentencia es riesgosa porque puede borrar el contenido de una tabla.



* **Select:** Trae información de la base de datos.

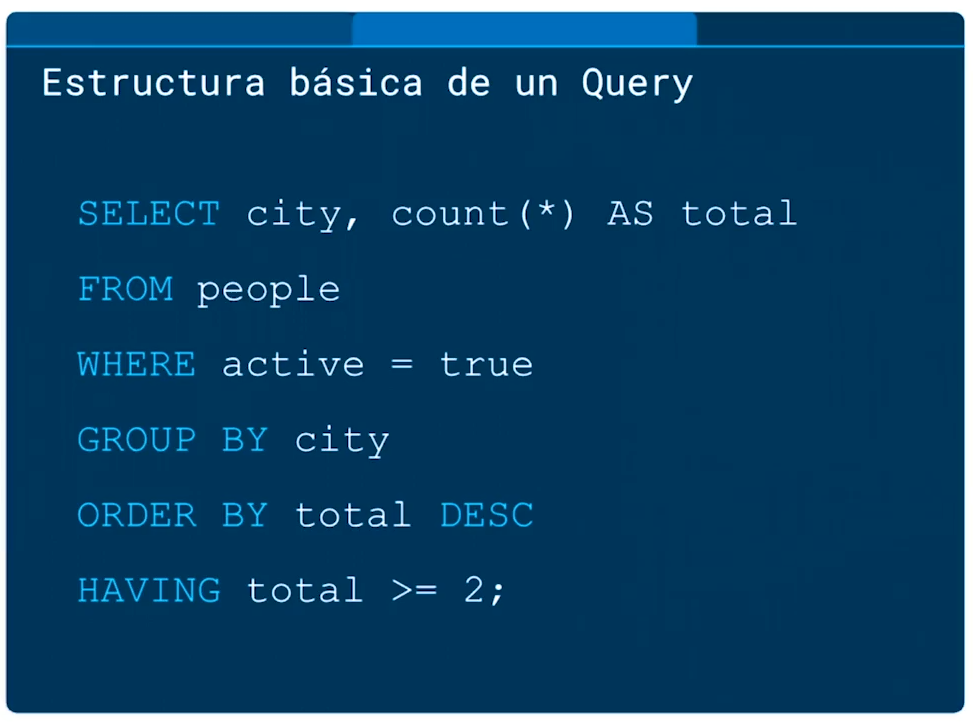


# Estructura básica de un Query

Los queries son la forma en la que estructuramos las preguntas que se harán a la base de datos. Transforma preguntas en sintaxis.

El query tiene básicamente 2 partes: **SELECT** y **FROM** y puede aparecer una tercera como **WHERE**.

* La estrellita o asterisco (\*) quiere decir que vamos a seleccionar todo sin filtrar campos.



# SELECT

**SELECT** se encarga de proyectar o mostrar datos.

* El nombre de las columnas o campos que estamos consultando puede ser cambiado utilizando **AS** después del nombre del campo y poniendo el nuevo que queremos tener:

SELECT titulo AS encabezado

FROM posts;

* Existe una función de SELECT para poder contar la cantidad de registros. Esa información (un número) será el resultado del query:

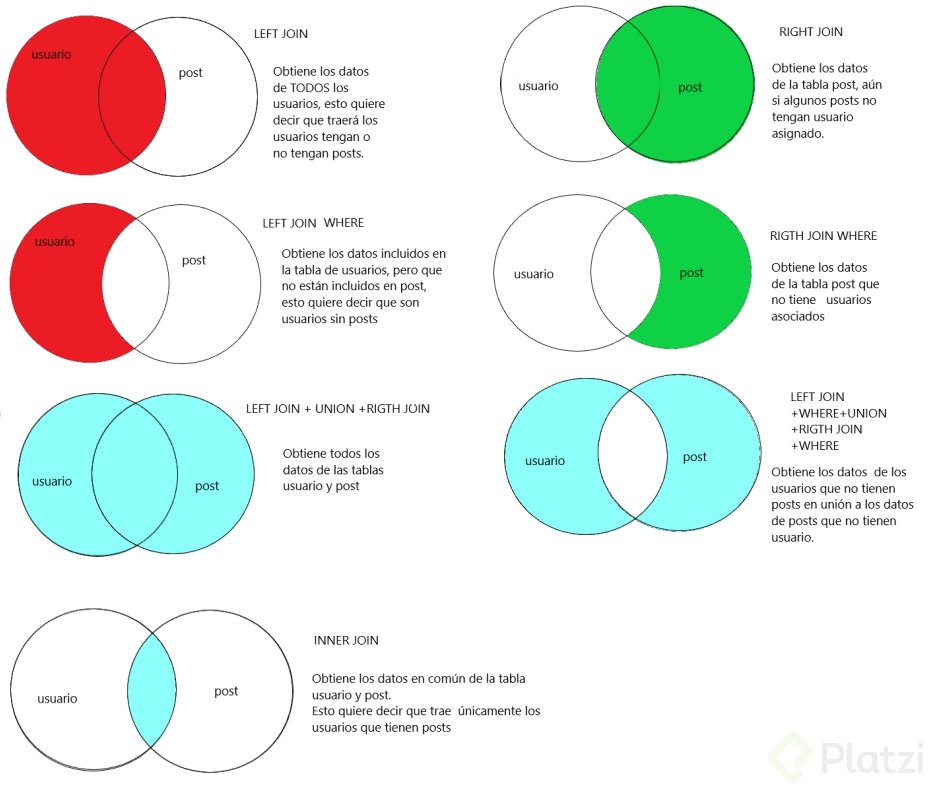
SELECT COUNT(\*)

FROM posts;

# FROM

**FROM** indica de dónde se deben traer los datos y puede ayudar a hacer sentencias y filtros complejos cuando se quieren unir tablas. La sentencia compañera que nos ayuda con este proceso es **JOIN**.

Los diagramas de Venn son círculos que se tocan en algún punto para ver dónde está la intersección de conjuntos. Ayudan mucho para poder formular la sentencia **JOIN** de la manera adecuada dependiendo del query que se quiere hacer.



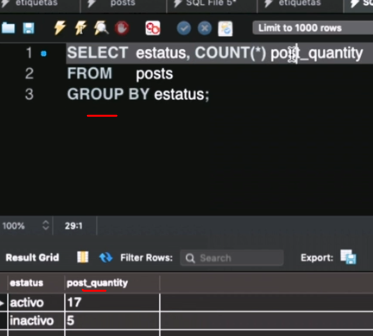
# WHERE

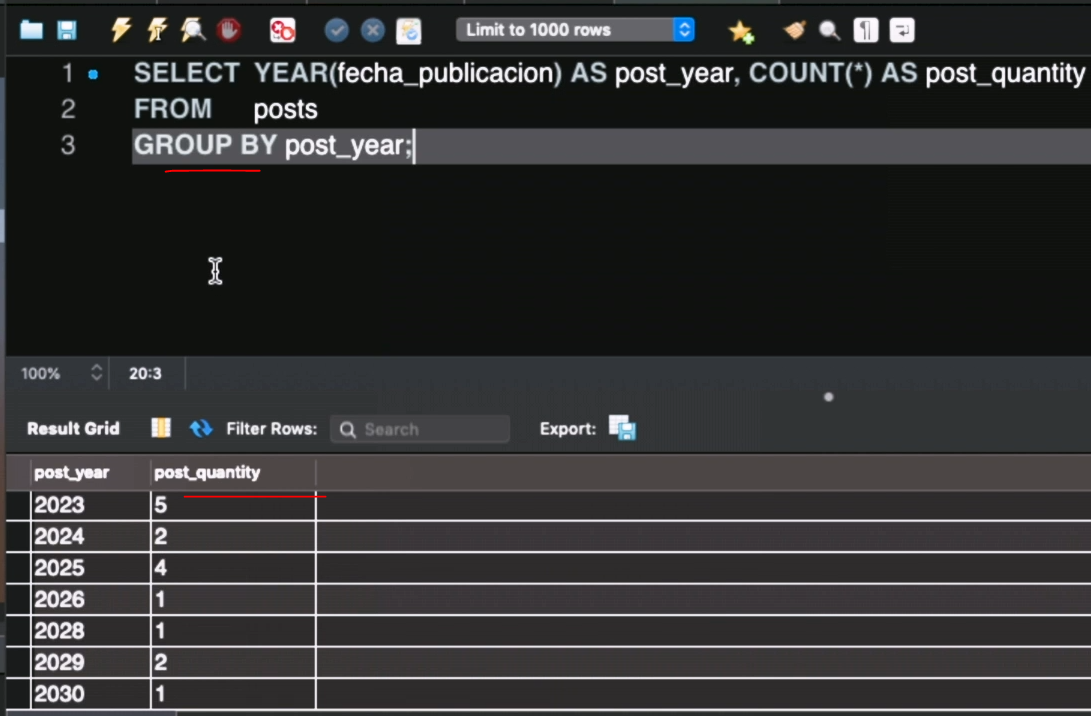
**WHERE** es la sentencia que nos ayuda a filtrar tuplas o registros dependiendo de las características que elegimos.

* La propiedad **LIKE** nos ayuda a traer registros de los cuales conocemos sólo una parte de la información.
* La propiedad **BETWEEN** nos sirve para arrojar registros que estén en el medio de dos. Por ejemplo los registros con id entre 20 y 30.
* El valor nulo en una tabla generalmente es su valor por defecto cuando nadie le asignó algo diferente. La sintaxis para hacer búsquedas de datos nulos es **IS NULL**. La sintaxis para buscar datos que no son nulos es **IS NOT NULL**

# GROUP BY

**GROUP BY** tiene que ver con agrupación. Indica a la base de datos qué criterios debe tener en cuenta para agrupar.



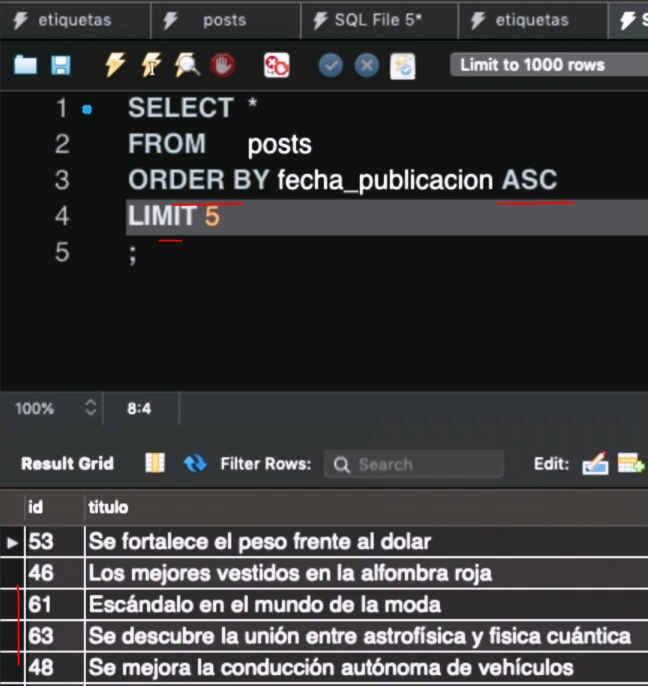


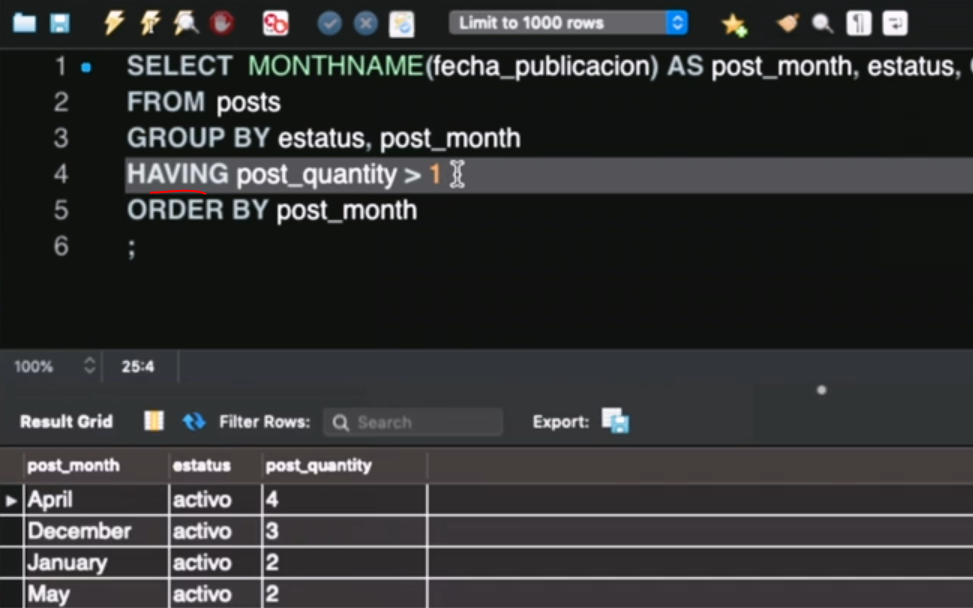
# ORDER BY y HAVING

La sentencia **ORDER BY** tiene que ver con el ordenamiento de los datos dependiendo de los criterios que quieras usar.

* **ASC** sirve para ordenar de forma ascendente.
* **DESC** sirve para ordenar de forma descendente.
* **LIMIT** se usa para limitar la cantidad de resultados que arroja el query.

**HAVING** tiene una similitud muy grande con **WHERE**, sin embargo el uso de ellos depende del orden. Cuando se quiere seleccionar tuplas agrupadas únicamente se puede hacer con **HAVING**.

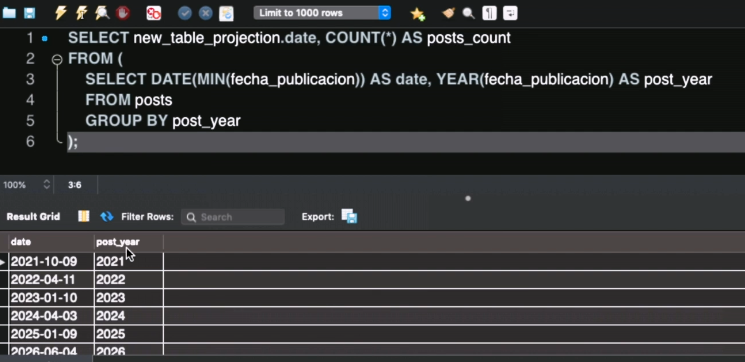




# Nested queries.

Los **Nested queries** significan que dentro de un query podemos hacer otro query. Esto sirve para hacer join de tablas, estando una en memoria. También teniendo un query como condicional del otro.

Este proceso puede ser tan profundo como quieras, teniendo infinitos queries anidados.  
Se le conoce como un producto cartesiano ya que se multiplican todos los registros de una tabla con todos los del nuevo query. Esto provoca que el query sea difícil de procesar por lo pesado que puede resultar.



# ¿Cómo convertir una pregunta en un query SQL?

**De pregunta a Query**

* **SELECT:** Lo que quieres mostrar
* **FROM:** De dónde voy a tomar los datos
* **WHERE:** Los filtros de los datos que quieres mostrar
* **GROUP BY:** Los rubros por los que me interesa agrupar la información
* **ORDER BY:** El orden en que quiero presentar mi información
* **HAVING:** Los filtros que quiero que mis datos agrupados tengan

# Bases de Datos no Relacionales.

Dentro de las bases de datos relacionales tenemos diferentes niveles de datos. En primer lugar tenemos las Bases de Datos o Esquemas como repositorios donde vivirán los datos que nos interesa guardar. Dentro del esquema existen las Tablas que provienen del concepto de entidades; y a su vez dentro de las tablas tenemos las tuplas o renglones.

Cuando trabajamos con bases de datos basadas en documentos como Firestore, aún existe la figura de la base de datos, sin embargo cambiaremos las tablas en favor de las colecciones y las tuplas en lugar de los documentos.

Recuerda:

Tabla -> Colección

Tupla -> Documento

Dentro de las Colecciones existen 2 grandes tipos. Las Top level collection o colecciones de nivel superior y las subcollections o subcolecciones. Estas últimas viven únicamente dentro de un documento padre.

**¿Cómo saber cuál escoger?**

Para determinar si tu colección debe ser top level o subcolección no hay una regla escrita en piedra y más bien tiene que ver con el caso de uso en particular y con la experiencia que hayas ganado como desarrollador.

Lo cierto es que no hay una sola forma de estructurar nuestra DB basada en documentos, y por tanto no existe una respuesta correcta, sin embargo a continuación te ofrezco un par de reglas guía que puedes utilizar para transformar tu proyecto que ya trabajaste en bases de datos relacionales en un proyecto no relacional.

**Regla 1. Piensa en la vista de tu aplicación**

La primera pista que te puedo dar es que pienses en un inicio en la manera en que los datos serán extraídos. En el caso de una aplicación, la mejor forma de pensarlo es en términos de las vistas que vas a mostrar a un momento determinado en la aplicación.

Es decir, al armar la estructura en la base de datos que sea un espejo o que al menos contenga todos los datos necesarios para llenar las necesidades que tiene nuestra parte visual en la aplicación.

En el caso de Platziblog por ejemplo si tienes una vista de un blog post individual, generalmente conviene mostrar además de los datos inherentes al post como el contenido, datos adicionales como las etiquetas que tiene o por ejemplo el autor (o autores si es colaborativo), en este caso tal vez convenga guardar estas dos “entidades” (autores y etiquetas) como subcolecciones de cada documento blog post.

**Regla 2. La colección tiene vida propia**

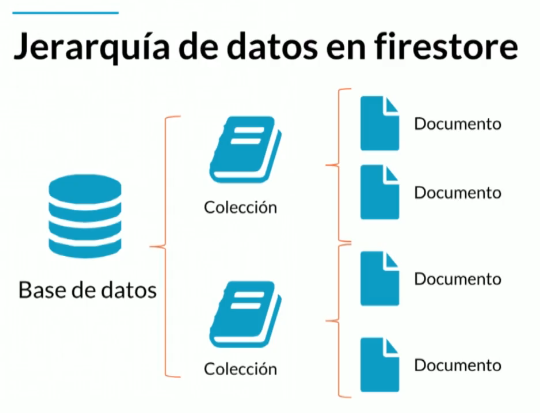
Esta regla se refiere a que la excepción a la regla 1 es cuando tenemos un caso en que la “entidad” que tiene necesidad de vivir y modificarse constantemente de manera independiente a las otras colecciones. Por ejemplo en Platziblog podemos en el ejemplo anterior hacer una excepción a autores porque nos conviene tenerlas como top level collection en el sentido que se añadan, borren, cambien o listen los usuarios sin depender del blog post.

Experimenta aplicando estas dos reglas a un proyecto que ya conozcas en una base de datos relacional y trata de convertirla en un proyecto de Firestore y comentanos los retos a los que te enfrentaste.

Respecto a las bases de datos no relacionales, no existe un solo tipo aunque se engloben en una sola categoría.

**Tipos de bases de datos no relacionales:**

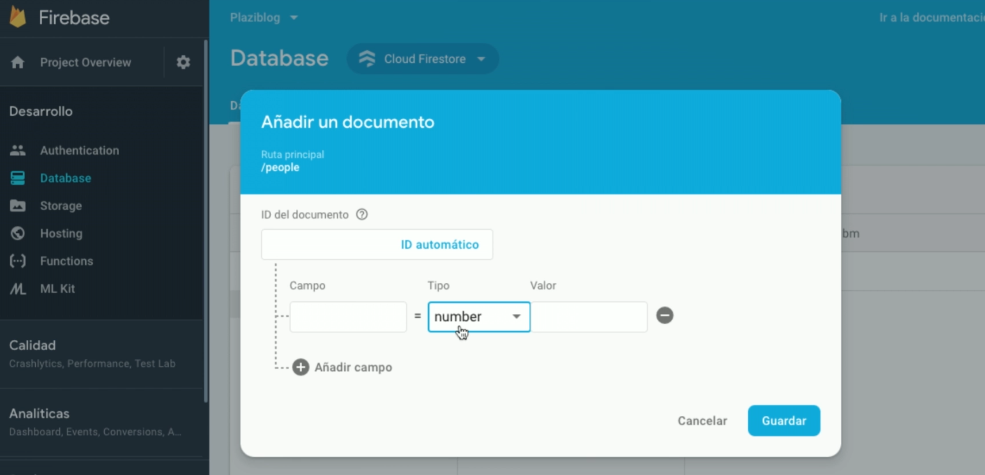
* **Clave - valor:** Son ideales para almacenar y extraer datos con una clave única. Manejan los diccionarios de manera excepcional. Ejemplos: **DynamoDB**, **Cassandra**.
* **Basadas en documentos:** Son una implementación de clave valor que varía en la forma semiestructurada en que se trata la información. Ideal para almacenar datos JSON y XML. Ejemplos: **MongoDB, Firestore**.
* **Basadas en grafos:** Basadas en teoría de grafos, sirven para entidades que se encuentran interconectadas por múltiples relaciones. Ideales para almacenar relaciones complejas. Ejemplos: **neo4j, TITAN**.
* **En memoria:** Pueden ser de estructura variada, pero su ventaja radica en la velocidad, ya que al vivir en memoria la extracción de datos es casi inmediata. Ejemplos: **Memcached, Redis**.
* **Optimizadas para búsquedas:** Pueden ser de diversas estructuras, su ventaja radica en que se pueden hacer queries y búsquedas complejas de manera sencilla. Ejemplos: **BigQuery, Elasticsearch**.



# Tipos de datos en Firestore.

*Tipos de datos en Firestore*:

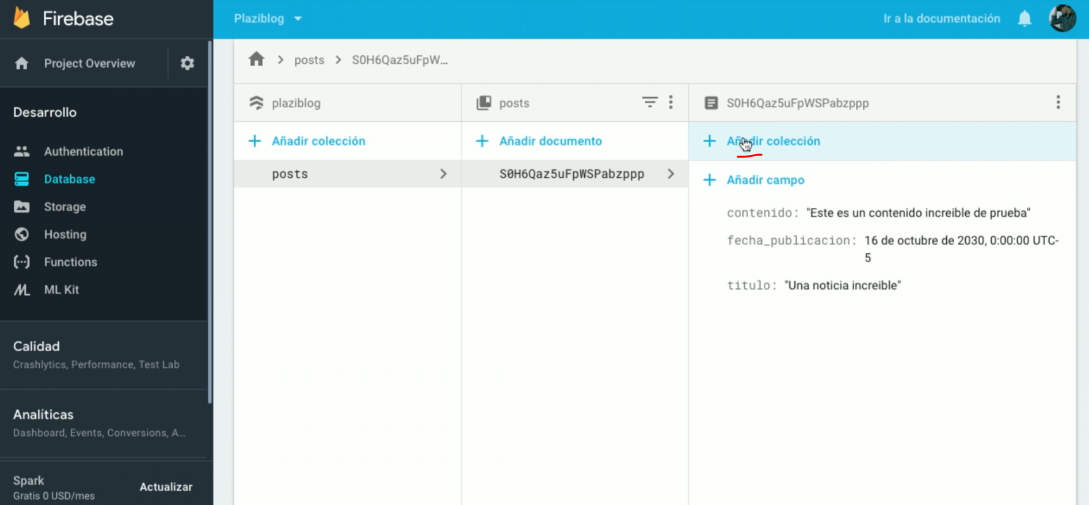
1. **String**: Cualquier tipo de valor alfanumérico
2. **Number**: Soporta enteros y flotantes.
3. **Boolenan**: Los clásicos valores True y False
4. **Map**: Permite agregar un documento dentro de otro.
5. **Array**: Permite agregar un conjunto de datos (soporte multi type) sin nombre e identificador.
6. **Null**: Indica que no se ha definido un valor.
7. **Timestamp**: Permite almacenar fechas (guarda el año, mes, día y hora).
8. **Geopoint**: Guarda una localización geográfica (coordenadas latitud-longitud).
9. **Reference**: Permite referencia un documento (relaciona dos documentos, no importa su colección).



# Colecciones vs subcolecciones

La particularidad de las top level collections es que existen en el primer nivel de manera intrínseca. Las subcolecciones ya no vivirán al inicio de la base de datos.

Si tienes una entidad separada que vas a referenciar desde muchos lugares es recomendado usar un top level collection. Por el otro lado si se necesita hacer algo intrínseco al documento es aconsejable usar subcolecciones.



# Bases de datos en la vida real.

Hoy en dia, se utilizan diversos tipos de bases de datos **segun el problema que se quiera resolver**:

* Las **bases de datos relacionales**, durante mucho tiempo, fueron utilizadas para resolver todo tipo de situaciones, pero al aumentar enormemente el numero de datos a manejar, se volveron **ineficientes** en muchos casos.
* **Firestore o MongoDB nos permiten obtener los datos actuales** de la aplicacion de manera simple. Sin embargo, **no nos permite hacer**, por ejemplo, ***queries* muy complejos**.
* En una misma disciplina, es probable que haya que **utilizar mas de un tipo de bases de datos.**

# Big Data

**Big Data** es un concepto que nace de la necesidad de manejar grandes cantidades de datos. La tendencia comenzó con compañías como YouTube al tener la necesidad de guardar y consultar mucha información de manera rápida.  
Es un gran movimiento que consiste en el uso de diferentes tipos de bases de datos.

# Data warehouse

**Data Warehouse (almacén)** trata de guardar cantidades masivas de datos para la posteridad. Allí se guarda todo lo que no está viviendo en la aplicación pero es necesario tenerlo.  
Debe servir para guardar datos por un largo periodo de tiempo y estos datos se deben poder usar para poder encontrar cuestiones interesantes para el negocio.

# Data mining

El **Data Mining** se dedica a minar datos, a extraerlos de donde sea que estén (archivos muertos, base de datos actual, etc…) y hacer sentido de ellos para darles un uso.

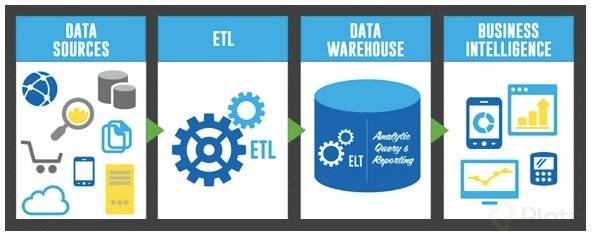


# ETL

**ETL** son las siglas de Extract, Transform, Load (extraer, transformar y cargar). Se trata de tomar datos de archivos muertos y convertirlos en algo que sea de utilidad para el negocio.  
También ayuda a tomar los datos vivos de la aplicación, transformarlos y guardarlos en un data warehouse periódicamente.

# Business intelligence

**Business Intelligence** es una parte muy importante de las carreras de datos ya que es el punto final del manejo de estos. Su razón de ser es tener la información lista, clara y que tenga todos los elementos para tomar decisiones en una empresa.  
Es necesario tener una buena sensibilidad por entender el negocio, sus necesidades y la información que puede llevar a tomar decisiones en el momento adecuado al momento de realizar business intelligence.



# Machine Learning

**Machine Learning** tiene significados que varían. Es una serie de técnicas que involucran la inteligencia artificial y la detección de patrones.  
Machine learning para datos tiene un gran campo de acción y es un paso más allá del business intelligence.  
Nos ayuda a hacer modelos que encuentran patrones fortuitos encontrando correlaciones inesperadas.

**Tiene dos casos de uso particulares:**

* Clasificación
* Predicción

# Data Science

**Data Science** es aplicar todas las técnicas de procesamiento de datos. En su manera más pura tiene que ver con gente con un background de estadísticas y ciencias duras.