Índice

[Historia de la persistencia de la información 5](#_Toc16084130)

[¿Qué es una base de datos? 7](#_Toc16084131)

[**Tipos de bases de datos** 7](#_Toc16084132)

[ Relacionales 7](#_Toc16084133)

[ No relacionales 7](#_Toc16084134)

[**Servicios** 7](#_Toc16084135)

[ Auto administrados 7](#_Toc16084136)

[ Administrados 7](#_Toc16084137)

[Historia de las RDB 8](#_Toc16084138)

[Entidades y atributos 8](#_Toc16084139)

[ **Atributos compuestos** 8](#_Toc16084140)

[ **Atributos llave** 8](#_Toc16084141)

[ Naturales 8](#_Toc16084142)

[ Clave artificial 8](#_Toc16084143)

[**Entidades débiles** 9](#_Toc16084144)

[ **Identidades débiles por identidad** 9](#_Toc16084145)

[ **Identidades débiles por existencia** 9](#_Toc16084146)

[Relaciones 9](#_Toc16084147)

[ Cardinalidad: 1 a 1 10](#_Toc16084148)

[ Cardinalidad: 0 a 1 10](#_Toc16084149)

[ Cardinalidad: 1 a N 10](#_Toc16084150)

[ Cardinalidad: 0 a N 10](#_Toc16084151)

[ Cardinalidad: N a N 10](#_Toc16084152)

[Diagrama ER 11](#_Toc16084153)

[Diagrama Físico: tipos de datos y constraints 11](#_Toc16084154)

[**Tipos de datos** 11](#_Toc16084155)

[CHAR(n) 11](#_Toc16084156)

[VARCHAR(n) 11](#_Toc16084157)

[TEXT 11](#_Toc16084158)

[INTEGER 11](#_Toc16084159)

[BIGINT 11](#_Toc16084160)

[SMALLINT 11](#_Toc16084161)

[DECIMAL(n,s) 11](#_Toc16084162)

[NUMERIC(n,s) 11](#_Toc16084163)

[DATE 11](#_Toc16084164)

[TIME 11](#_Toc16084165)

[DATETIME 11](#_Toc16084166)

[TIMESTAMP 11](#_Toc16084167)

[BOOLEAN 12](#_Toc16084168)

[**Constraints** (Restricciones) 12](#_Toc16084169)

[ NOT NULL 12](#_Toc16084170)

[ UNIQUE 12](#_Toc16084171)

[ PRIMARY KEY 12](#_Toc16084172)

[ FOREIGN KEY 12](#_Toc16084173)

[ CHECK 12](#_Toc16084174)

[ DEFAULT 12](#_Toc16084175)

[ INDEX 12](#_Toc16084176)

[Normalización 12](#_Toc16084177)

[Formas normales en DB relacionales 14](#_Toc16084178)

[**Primera Forma Normal (1FN)** 15](#_Toc16084179)

[**Segunda Forma Normal (2FN)** 15](#_Toc16084180)

[**Tercera Forma Normal (3FN)** 16](#_Toc16084181)

[**Cuarta Forma Normal (4FN)** 17](#_Toc16084182)

[RDB ¿Qué? 20](#_Toc16084183)

[Instalación local de un RDBMS (Windows) 20](#_Toc16084184)

[Servicios administrados 21](#_Toc16084185)

[Historia de SQL 21](#_Toc16084186)

[DDL vs DML 21](#_Toc16084187)

[DDL create 21](#_Toc16084188)

[**Ventajas** 22](#_Toc16084189)

[DDL alter 23](#_Toc16084190)

[DDL drop 23](#_Toc16084191)

[DML 23](#_Toc16084192)

[ **Insert** 23](#_Toc16084193)

[ **Update** 23](#_Toc16084194)

[ **Delete** 23](#_Toc16084195)

[ **Select** 23](#_Toc16084196)

[¿Qué tan standard es SQL? 26](#_Toc16084197)

[Tablas independientes 26](#_Toc16084198)

[Tablas dependientes 26](#_Toc16084199)

[Tablas transitivas 26](#_Toc16084200)

[¿Por qué las consultas son tan importantes? 27](#_Toc16084201)

[Estructura básica de un Query 27](#_Toc16084202)

[**SELECT** 27](#_Toc16084203)

[**FROM** 27](#_Toc16084204)

[**WHERE** 27](#_Toc16084205)

[FROM 28](#_Toc16084206)

[ **LEFT JOIN** 28](#_Toc16084207)

[ **RIGHT JOIN** 28](#_Toc16084208)

[ **INNER JOIN** 29](#_Toc16084209)

[ **FULL OUTER JOIN** (unión) 29](#_Toc16084210)

[ **OUTER JOIN EXCLUDING INNER JOIN** (diferencia simétrica) 29](#_Toc16084211)

[Utilizando la sentencia FROM 30](#_Toc16084212)

[Where 32](#_Toc16084213)

[Utilizando la sentencia WHERE nulo y no nulo 33](#_Toc16084214)

[GROUP BY 33](#_Toc16084215)

[ORDER BY y HAVING 33](#_Toc16084216)

[ **ASC** 33](#_Toc16084217)

[ **DESC** 33](#_Toc16084218)

[ **LIMIT** 33](#_Toc16084219)

[El interminable agujero de conejo (Nested queries) 34](#_Toc16084220)

[¿Cómo convertir una pregunta en un query SQL? 34](#_Toc16084221)

[ **SELECT** 34](#_Toc16084222)

[ **FROM** 34](#_Toc16084223)

[ **WHERE** 34](#_Toc16084224)

[ **GROUP BY** 34](#_Toc16084225)

[ **ORDER BY** 34](#_Toc16084226)

[ **HAVING** 34](#_Toc16084227)

[ **GROUP\_CONCAT** 34](#_Toc16084228)

[¿Qué son y cuáles son los tipos de bases de datos no relacionales? 35](#_Toc16084229)

[Tipos de bases de datos no relacionales 35](#_Toc16084230)

[ **Clave - valor** 35](#_Toc16084231)

[ **Basadas en documentos** 35](#_Toc16084232)

[ **Basadas en grafos** 36](#_Toc16084233)

[ **En memoria** 36](#_Toc16084234)

[ **Optimizadas para búsquedas** 36](#_Toc16084235)

[Servicios administrados y jerarquía de datos 37](#_Toc16084236)

[Jerarquía de datos 37](#_Toc16084237)

[ Base de datos 37](#_Toc16084238)

[ Colección 37](#_Toc16084239)

[ Documento 37](#_Toc16084240)

[Top level collection con firebase 37](#_Toc16084241)

[**Tipos de datos en Firebase** 37](#_Toc16084242)

[ String 37](#_Toc16084243)

[ Number 38](#_Toc16084244)

[ Boolean 38](#_Toc16084245)

[ Map 38](#_Toc16084246)

[ Array 38](#_Toc16084247)

[ Null 38](#_Toc16084248)

[ Timestamp 38](#_Toc16084249)

[ Geopoint 38](#_Toc16084250)

[ Reference 38](#_Toc16084251)

[Colecciones vs subcolecciones 38](#_Toc16084252)

[Transformando tu proyecto en una db no relacional 38](#_Toc16084253)

[Big Data 40](#_Toc16084254)

[Data warehouse 40](#_Toc16084255)

[**Diferencias con Big Data** 41](#_Toc16084256)

[**Características principales de un Warehouse** 41](#_Toc16084257)

[Data mining 41](#_Toc16084258)

[**¿Por qué surge?** 41](#_Toc16084259)

[ETL 42](#_Toc16084260)

[Business intelligence 43](#_Toc16084261)

[Machine Learning 43](#_Toc16084262)

[Tiene dos casos de uso particulares 44](#_Toc16084263)

[ Clasificación 44](#_Toc16084264)

[ Predicción 44](#_Toc16084265)

[**Diferencias con business intelligence** 44](#_Toc16084266)

[Data Science 44](#_Toc16084267)

# Historia de la persistencia de la información

Históricamente la información se pasaba de boca en boca, desde la antigüedad se empezaron a dar cuenta que era necesario de guardar la información, no cambiara y que persistiera más allá del tiempo de vida de un ser humano. Es por esto es que se inventaron los sistemas de escrituras primitivos, unos de los primeros que encontramos arqueológicamente es la escritura cuneiforme de los sumerios que, aunque es primitiva es compleja en su estructura y ésta la guardaban básicamente en piedras talladas y en tablas de arcillas.

Este tipo de escritura tenía varias desventajas, entre ellas.

1. No eran muy transportables.
2. Las tablillas de arcillas se rompían fácilmente.
3. Entre otras.

Entonces se dio el salto a otro tipo de información como fue el papiro y el pergamino, estos eran muchos más portátiles y ligeros, el único problema que tenían que eran vasados en materia vegetal o animal por lo tanto se descomponían con facilidad o era fácil que fueran atacados por hongos, por este motivo acababan destruyendo el documento. Fue entonces que los chinos llegaron a una gran revolución que fue el papel, el papel cambio todo eso porque tenía todas las ventajas de portabilidad, pero no tenía la desventaja de ser tan fácilmente destruible como era el papiro. Después de esto pasaron muchos siglos y el siguiente salto que tenemos en el almacenamiento de datos fue hacia el siglo XX con lo que se conoce como el microfilm (es un sistema de archivo, gestión y difusión documental), esté fue una tecnología, aunque ahora nos suene un poquito obsoleta, tiene una gran ventaja:

* Puede almacenar datos de manera virtualmente infinita.
* Pueden durar miles de años sin ningún problema.

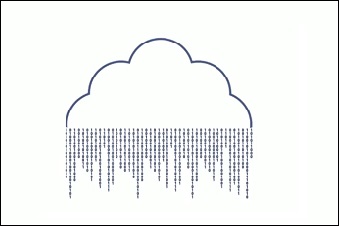
**¿Cuál era la desventaja de este sistema?**

Para poder guardar, modificar y leer información requería maquinas muy especializadas que no son fáciles de conseguir y el proceso no era tan fácil, pero si quieres almacenar información por un periodo muy largo de tiempo el microfilm es una gran solución.

Luego de esto saltamos a las grandes revoluciones que fueron los medios digitales. Estos medios incluían, por ej:

1. Discos duros.
2. Discos en estado sólidos.
3. Incluso los CDs, muchos ya no los utilizan actualmente, pero son parte de este periodo histórico.

Fueron medios de almacenamiento digital en que ya se guardaba la información no escritas, ni como fotografías, sino en formatos de 1 y 0 (ceros), bits y bytes que es lo que estamos acostumbrados a esta parte de la historia.

Después de eso hubo un periodo de recesión en el que no hubo grandes cambios y el siguiente gran cambio que se dio fue la nube, las ventajas que trae este medio son:

* Es accesible desde cualquier parte del mundo.
* Si tu almacenas algo ahí puedes extraerlo desde cualquier lugar y con diferentes dispositivos con acceso a internet.
* Es un medio centralizado.
* Puede ser utilizado por varias personas al mismo tiempo.

# ¿Qué es una base de datos?

Es un almacén de información. Las BBDD entran en el periodo donde entramos en los medios digitales como los discos duros y ahora la nube. Nos servían para complementar la arquitectura de Von Neumann (matemático húngaro-estadounidense) que es la arquitectura de computación clásica que incluye CPU, Memoria y elementos de entradas y salida.

## **Tipos de bases de datos**

* Relacionales: Es un tipo de base de datos que cumple con el modelo relacional (es un modelo de datos basado en la lógica de predicados y en la teoría de conjuntos creadas por Codd, que fue un científico informático inglés). En la industria hay varias compañías dedicadas a ser manejadoras de bases de datos relacionales como:



* + **Privativas:** SQL Server, Oracle.
  + **Open Source**:MySQL, MariaDB, PostgreSQL entre otras.
* No relacionales: Son aquellas que no usan SQL (por sus siglas en inglés **Structured Query Language**) como lenguaje principal de consultas. 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: Son servicios que ofrecen las nubes modernas como **Google**, **Azure**, **AWS** y no debes preocuparte por mantenimiento o actualizaciones.

# Historia de las RDB

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

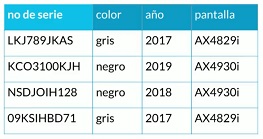
Las bases de datos basadas en archivos eran datos guardados en texto plano, fáciles de guardar, pero muy difíciles de consultar y por la necesidad de mejorar esto nacen las bases de datos relacionales. Su inventor Edgar Codd dejó ciertas reglas para asegurarse de que toda la filosofía de las bases de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

Codd también invento el álgebra relacional, se trata de como tenemos datos que se puedan empezar a mezclar y a unir a través de diferentes propiedades y características.

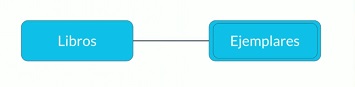
# Entidades y atributos

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. Existen dos tipos de atributos.

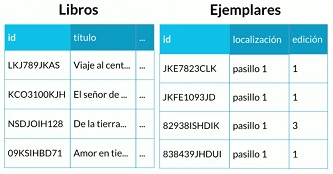
* **Atributos compuestos** son aquellos que tienen atributos ellos mismos.
* **Atributos llave** son aquellos que identifican a la entidad y no pueden ser repetidos. Existen los:



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

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

* ******Identidades débiles por identidad**: No se diferencian entre sí más que por la clave de su identidad fuerte.

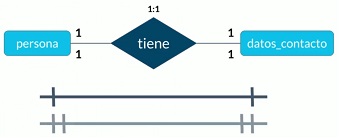


* **Identidades débiles por existencia**: Se les asigna una clave propia. Por ej: Aunque se agregue un id que es diferente de libros y es propio del ejemplar, no se puede tener un ejemplar sin un libro. Aunque no depende por el id o por la identidad, de la entidad fuerte, no se puede tener un ejemplar sin un libro primero.

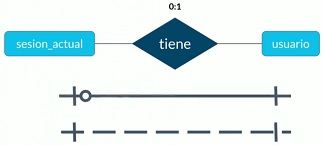
# Relaciones

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

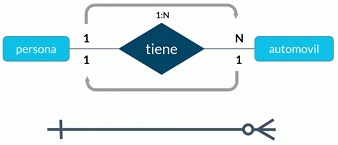
Las relaciones tienen una propiedad llamada cardinalidad y tiene que ver con números. Cuántos de un lado pertenecen a cuántos del otro lado:



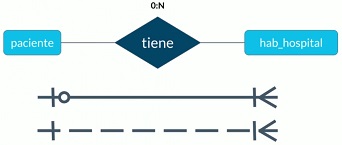
## Cardinalidad: 1 a 1



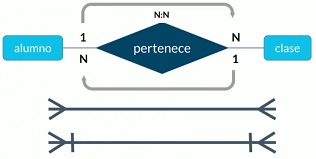
## Cardinalidad: 0 a 1



## Cardinalidad: 1 a N



## Cardinalidad: 0 a N



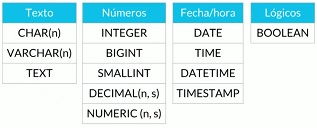
## Cardinalidad: N a N

# 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.

# Diagrama Físico: tipos de datos y constraints

## **Tipos de datos**

CHAR(n): Almacena cadenas.

VARCHAR(n): Almacena cadenas, pero la diferencia con el CHAR es que el anterior reserva todo el pedacito de memoria que indiques, el VARCHAR lo hace de manera dinámica, es decir, se le pasa un máximo de caracteres que se puedan almacenar y él se encarga de reservar el pedacito de memoria que ocupe ese campo solamente. Por ej: Si tenemos un VARCHAR(50) y en el campo sólo ocupa 40 caracteres, en memoria solo ocupará 40 y no 50.

TEXT: Para guardar muy grandes cantidades de caracteres.

INTEGER: Guarda sólo números enteros.

BIGINT: Para guardar números enteros muy grandes.

SMALLINT: Para guardar números pequeños.

DECIMAL(n,s): Para guardar números decimales. **n** representa la parte entera, **s** representa la parte decimal de un número.

NUMERIC(n,s): Guarda números al igual que DECIMAL. **n** representa la parte entera, **s** representa la parte decimal de un número.

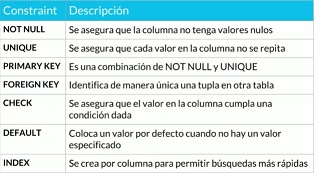
DATE: Guarda la fecha corta en formato dd/MM/yyyy

TIME: Guarda la hora en formato hh:mm:ss:ml

DATETIME: Guarda la fecha y hora en formato dd/MM/yyyy hh:mm:ss:ml

TIMESTAMP**:** Guarda la fecha y hora al igual que DATETIME.

BOOLEAN: Guarda datos de tipo lógicos, es decir, True y False. Sólo puede tener esos dos estados.



## **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.
* PRIMARY KEY: Es una combinación de NOT NULL y UNIQUE.
* FOREIGN KEY: Identifica de manera única una tupla en otra tabla.
* 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.

# Normalización

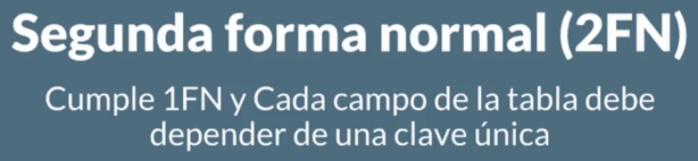
La normalización como su nombre lo indica nos ayuda a dejar todo de una forma normal. Esto obedece a las 12 reglas de Codd y nos permiten separar componentes en la base de datos:

* **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.

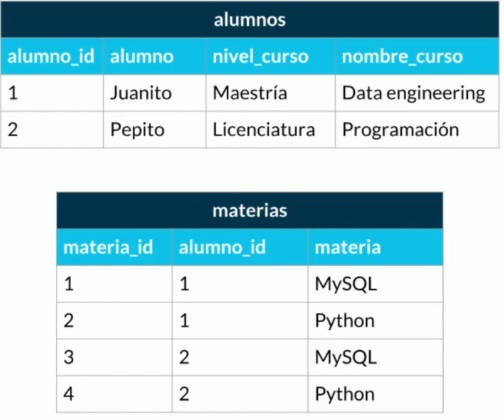


La tabla anterior tenia los campos **materia\_1** y **materia\_2**, aquí lo unificamos en un sólo campo, de esta forma los datos quedarían atómicos quedando así.





En la segunda forma separamos el campo **materia** en una entidady ahora cada registro está identificado con una clave única.





En la tercera forma separamos el campo **nombre\_curso** en otra entidad.





En la cuarta forma tenemos que en la tabla materia tenemos repetido registros, por esto los separamos en otra tabla lo cual quedaría.



# Formas normales en DB relacionales

La normalización en las bases de datos relacionales es uno de esos temas que, por un lado, es sumamente importante y por el otro suena algo esotérico. Vamos a tratar de entender las formas normales (FN) de una manera simple para que puedas aplicarlas en tus proyectos profesionales.

## **Primera Forma Normal (1FN)**

Esta FN nos ayuda a eliminar los valores repetidos y no atómicos dentro de una base de datos.

Formalmente, una tabla está en primera forma normal si:

* Todos los atributos son atómicos. Un atributo es atómico si los elementos del dominio son simples e indivisibles.
* No debe existir variación en el número de columnas.
* Los campos no clave deben identificarse por la clave (dependencia funcional).
* Debe existir una independencia del orden tanto de las filas como de las columnas; es decir, si los datos cambian de orden no deben cambiar sus significados.

Se traduce básicamente a que si tenemos campos compuestos como por ejemplo “nombre\_completo” que en realidad contiene varios datos distintos, en este caso podría ser “nombre”, “apellido\_paterno”, “apellido\_materno”, etc.

También debemos asegurarnos que las columnas son las mismas para todos los registros, que no haya registros con columnas de más o de menos.

Todos los campos que no se consideran clave deben depender de manera única por el o los campos que si son clave.

Los campos deben ser tales que, si reordenamos los registros o reordenamos las columnas, cada dato no pierda el significado.

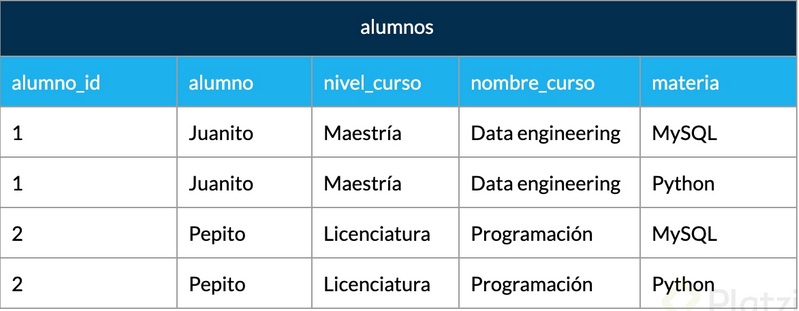
## **Segunda Forma Normal (2FN)**

Esta FN nos ayuda a diferenciar los datos en diversas entidades.

Formalmente, una tabla está en segunda forma normal si:

* Está en 1FN
* Sí los atributos que no forman parte de ninguna clave dependen de forma completa de la clave principal. Es decir, que no existen dependencias parciales.
* Todos los atributos que no son clave principal deben depender únicamente de la clave principal.

Lo anterior quiere decir que sí tenemos datos que pertenecen a diversas entidades, cada entidad debe tener un campo clave separado. Por ejemplo:



En la tabla anterior tenemos por lo menos dos entidades que debemos separar para que cada uno dependa de manera única de su campo llave o ID. En este caso las entidades son alumnos por un lado y materias por el otro. En el ejemplo anterior, quedaría de la siguiente manera:



## **Tercera Forma Normal (3FN)**

Esta FN nos ayuda a separar conceptualmente las entidades que no son dependientes.

Formalmente, una tabla está en tercera forma normal si:

* Se encuentra en 2FN
* No existe ninguna dependencia funcional transitiva en los atributos que no son clave
* Esta FN se traduce en que aquellos datos que no pertenecen a la entidad deben tener una independencia de las demás y debe tener un campo clave propio. Continuando con el ejemplo anterior, al aplicar la 3FN separamos la tabla alumnos ya que contiene datos de los cursos en ella quedando de la siguiente manera.



## **Cuarta Forma Normal (4FN)**

Esta FN nos trata de atomizar los datos multivaluados de manera que no tengamos datos repetidos entre rows.

Formalmente, una tabla está en cuarta forma normal si:

* Se encuentra en 3FN
* Los campos multivaluados se identifican por una clave única

Esta FN trata de eliminar registros duplicados en una entidad, es decir, que cada registro tenga un contenido único y de necesitar repetir la data en los resultados se realiza a través de claves foráneas.

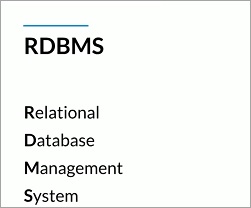
Aplicado al ejemplo anterior la tabla materia se independiza y se relaciona con el alumno a través de una tabla transitiva o pivote, de tal manera que si cambiamos el nombre de la materia solamente hay que cambiarla una vez y se propagara a cualquier referencia que haya de ella.

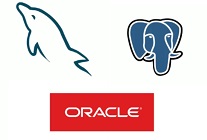


De esta manera, aunque parezca que la información se multiplicó, en realidad la descompusimos o normalizamos de manera que a un sistema le sea fácil de reconocer y mantener la consistencia de los datos.

Algunos autores precisan una 5FN que hace referencia a que después de realizar esta normalización a través de uniones (JOIN) permita regresar a la data original de la cual partió.

# RDB ¿Qué?

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.

En el mercado existen distintos tipos de RDBMS, algunos de ellos son como por ej: MySQL, PostgreSQL, ORACLE.

# 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.

# Servicios administrados

Hoy en día muchas empresas ya no tienen instalados en sus servidores los RDBMS, sino que los contratan a otras personas. Estos servicios administrados cloud te permiten concentrarte en la base de datos y no en su administración y actualización.

# 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 en 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 SQL. Las bases de datos no relacionales.

# DDL vs DML

**DDL** = **D**ata **D**efinition **L**anguages (CREATE, DROP, ALTER)

* Para crear la estructura de la Base de datos

**DML** = **D**ata **M**anipulation **L**anguage (INSERT, UPDATE, DELETE)

* Para manipular la data.

# DDL create

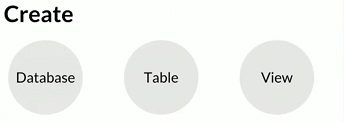
SQL tiene dos grandes sublenguajes:

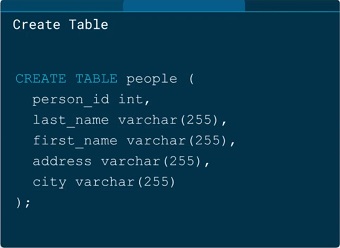
**DDL** o **Data Definition Language** que nos ayuda a crear la estructura de una base de datos y **DML** o **Data Manipulation Language** que nos ayuda a manipular la data.

Existen 3 grandes comandos de DDL:

* 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:

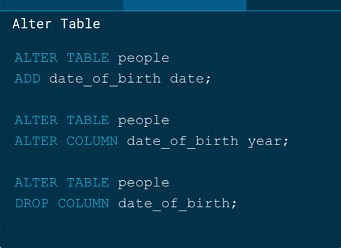
* Database o bases de datos.
* Table o tablas: Son la traducción a SQL de las entidades.
* View o vistas: Se ofrece la proyección de los datos de la base de datos de forma entendible.



Los Views toman datos de la base de datos, los hacen presentables y los convierten en algo que podamos consultar de manera recurrente.

**Ventajas**: Se mantienen en memoria sin la necesidad de estar haciendo consultas cada vez que se necesiten mostrar los datos.

# DDL alter

Este comando nos permite modificar los datos de una tabla.

# DDL drop

Está puede ser la sentencia ¡**más peligrosa**!, sobre todo cuando somos principiantes. Básicamente borra o desaparece de nuestra base de datos algún elemento.

# DML

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

## **Insert**

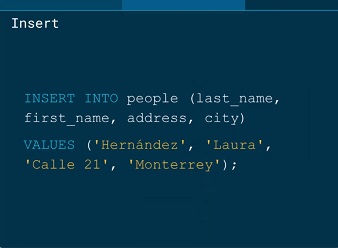
## **Update**

## **Delete**

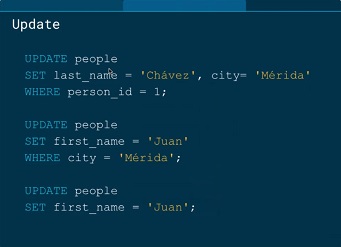
* **Select**

Insert

Inserta o agrega nuevos registros o tupla a la tabla de nuestra bbdd.

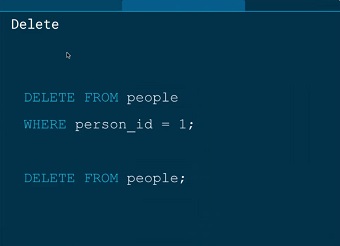


Update

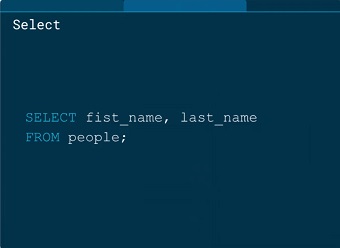
Actualiza o modifica los datos que ya existen.

Delete

Está sentencia es riesgosa porque puede borrar el contenido completo de una tabla si no se utiliza con la sentencia **WHERE**.



Select

Trae información de la base de datos.

# ¿Qué tan standard es SQL?

La utilidad más grande de SQL fue unificar la forma en la que pensamos y hacemos preguntas a un repositorio de datos. Ahora que nacen nuevas bases de datos igualmente siguen tomando elementos de SQL.

# Tablas independientes

Son aquellas tablas que no dependen de la existencia de otras, es decir, que no poseen una FK (foreign key.)

* 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

Son aquellas que dependen de la existencia de otras, es decir, poseen una FK (foreign key.)

* 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.

# Tablas transitivas

Las tablas transitivas o pivote 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.

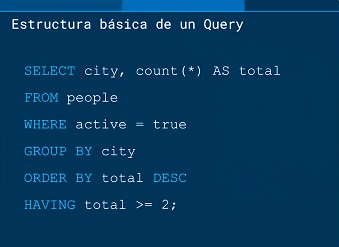
# ¿Por qué las consultas son tan importantes?

Las consultas o queries a una base de datos son una parte fundamental ya que esto podría salvar un negocio o empresa.

Alrededor de las consultas a las bases de datos se han creado varias especialidades como ETL o transformación de datos, business intelligence e incluso machine learning.

# 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**, **FROM** y puede aparecer una tercera como **WHERE**.

Ej:

**SELECT** Columna/s

**FROM** Tabla/s

**WHERE** (Condicion1 and Condicion2 and CondicionN)

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

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:



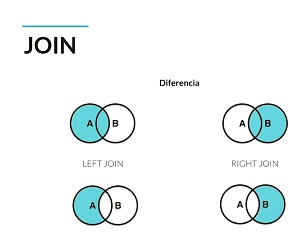
* Existe una función de SELECT para poder contar la cantidad de registros. Esa información (un número) será el resultado del query:

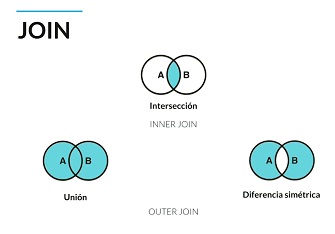


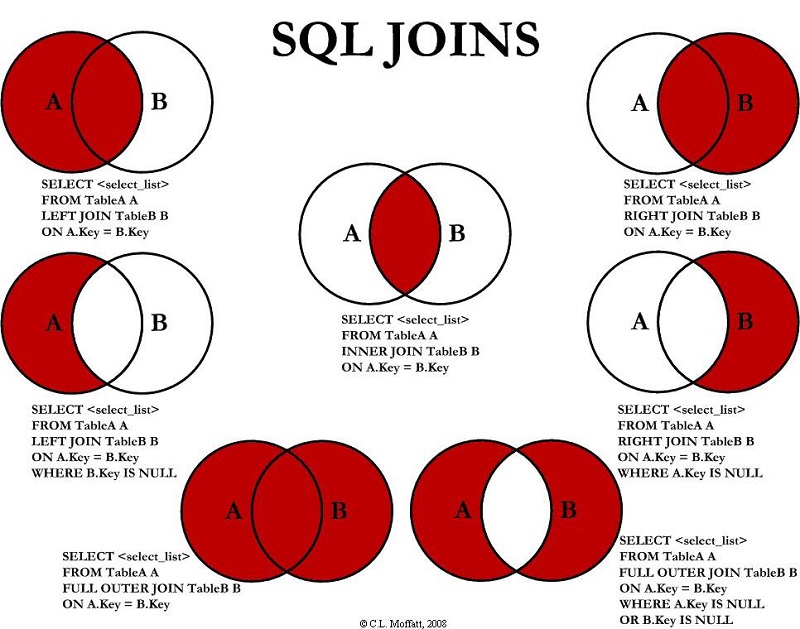
# FROM

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**.

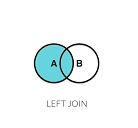
Los diagramas de Venn son círculos que se tocan en algún punto para ver dónde está la intersección de conjuntos. Ayudan mucho para poder formular la sentencia JOIN de la manera adecuada dependiendo del query que se quiere hacer.

Existen diferentes tipos de JOIN:

* **LEFT JOIN**: En el primer JOIN normal trae todos los datos estén o no estén en la tabla B. En el segundo sólo traerá los datos que solamente estén en la tabla A.
* **RIGHT JOIN**: En el primer JOIN traerá todo de la tabla B sin importar si está o no en la tabla A. En el segundo JOIN traerá sólo los datos asociado a la tabla B.
* ******INNER JOIN**: Traerá sólo la intersección de ambas tablas (A y B)
* **FULL OUTER JOIN** (unión): Traerá todo de ambas tablas.
* **OUTER JOIN EXCLUDING INNER JOIN** (diferencia simétrica): Traerá todo menos la intersección de ambas tablas.



# Utilizando la sentencia FROM

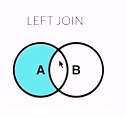
SELECT \*

FROM usuario LEFT JOIN post

ON usuario.id = post.usuario\_id

El resultado de este query será que traerá todos los **usuarios** sin importar que no tengan un **post** y del otro lado sólo traerá los post que si están asociado a un usuario.

SELECT \*

FROM usuario LEFT JOIN post

ON usuario.id = post.usuario\_id

WHERE post.usuario\_id IS NULL

El resultado de este query será que traerá todos los **usuarios** que no tengan **post**.

SELECT \*

FROM usuario RIGHT JOIN post

ON usuario.id = post.usuario\_id

El resultado de este query será que traerá todos los **post** sin importar que tengan o no **usuarios** y del otro lado sólo traerá los usuarios que **si** están asociado a un post.

SELECT \*

FROM usuario RIGHT JOIN post

ON usuario.id = post.usuario\_id

WHERE post.usuario\_id IS NULL

El resultado de este query será que traerá todos los post que **no tenga**n asociado un usuario.

SELECT \*

FROM usuario INNER JOIN post

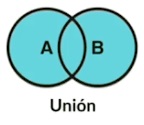
ON usuario.id = post.usuario\_id

El resultado de este query será que traerá todos los datos que tengan relación, es decir, los datos que estén en ambas tablas y no traerá aquellos datos que sea **null** o **huérfanos**.

En algunos administradores de bbdd no utilizan el **FULL OUTER JOIN**, en su lugar podríamos simular uno de la siguiente manera.

SELECT \*

FROM usuario LEFT JOIN post

ON usuario.id = post.usuario\_id

**UNION**

SELECT \*

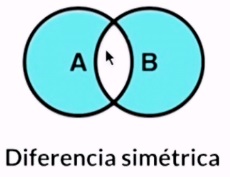
FROM usuario RIGHT JOIN post

ON usuario.id = post.usuario\_id

En esta query juntaremos los registros de la tabla de la izquierda con los registros de la tabla de la derecha. Está forma simulamos un **FULL OUTER JOIN** en cualquier bbdd.

SELECT \*

FROM usuario LEFT JOIN post

ON usuario.id = post.usuario\_id

WHERE post.usuario\_id IS NULL

UNION

SELECT \*

FROM usuario RIGHT JOIN post

ON usuario.id = post.usuario\_id

WHERE post.usuario\_id IS NULL

En esta query nos traerá solamente los registros que no tengan relación, es decir, que sean **null** o **huérfanos** de ambas tablas.

# 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.
  + El operador **‘%**cadena\_a\_buscar**%’** en la sentencia LIKE nos permite buscar lo que sea que tenga antes de la **cadena\_a\_buscar** y lo que sea después de la misma.
  + Si utilizamos **‘cadena\_a\_buscar%’** nos buscará todo lo que comience con la cadena especificada en adelante, por ej: Si buscamos la cadena ‘escándalo%’

Resultado: **escándalo** con el boxeador del momento.

* + Si utilizamos **‘%cadena\_a\_buscar’** buscará todo lo que termine con la cadena especificada. Por ej: **‘%roja’**

Resultado: los mejores vestidos en la alfombra **roja**.

* La propiedad **BETWEEN** nos sirve para arrojar registros que estén entre medio de dos valores. Por ejemplo, los registros con **idcliente** entre 20 y 30.



Ambos operadores **LIKE** y **BETWEEN** pueden ser negados con el operador **NOT**, por ej:

* NOT LIKE.
* NOT BETWEEN.

# Utilizando la sentencia WHERE nulo y no nulo

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

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**.

# El interminable agujero de conejo (Nested queries)

Los Nested queries (consultas anidadas) significa 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.
* **GROUP\_CONCAT:** Toma el resultado del query y lo pone como campo separado por comas. Se utiliza en la sentencia **SELECT**.

Puedes usar una abreviación para evitar escribir lo mismo cada vez.

Ejemplo:



# ¿Qué son y cuáles son los tipos de bases de datos no relacionales?

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.
  + Ventajas: Podemos extraer de manera muy rápida los datos si conocemos la clave.
  + Desventajas: Si se realizan consultas complejas puede que nos resulte difícil acceder a los valores si no conocemos la clave.

Se basan en algoritmo de **hash** o **hashmap**, es decir, son diccionarios de datos que están compuestos por una clave y un valor.

* ******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.
  + Ventajas: Nos ayudan a guardar el estado actual de nuestra aplicación, por ej: Un juego con varias personas, si se quiere guardar el estado actual del juego de un usuario, se puede almacenar en documentos sumamente fácil.
  + Desventajas: No son tan buenas para realizar query complejas de búsquedas como se haría con el lenguaje SQL en una bbdd relacional.
* **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.
  + Ventajas: Consultas más amplias y no demarcadas por tablas, no hay que definir un número determinado de atributos, Ej: Se utilizan mucho en el mundo de **AI** para formar redes neuronales.
  + Desventajas: El modelo de datos no está estandarizado (dificultad de cambio de gestor), recopilación de grafos completos y no subgrafos.
* ******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.
  + Ventajas: Son sumamente rápidas.
  + Desventajas: Cuentan ciertos límites, tienen que estar guardando datos en disco y son volátiles, es decir, si el servidor se reinicia probablemente se deba volver a indexar o volver a traer los datos del disco para ser llevados a la memoria nuevamente. También es un gran trabajo estar manteniendo en sincronía guardando datos en un lugar con más durabilidad como por ej: Un disco duro.
* ******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.
  + Ventajas: Se pueden hacer query muy complejas en cuestiones muy rapidas de tiempo, se pueden generar modelos de Machine Learning con un lenguaje similar al SQL sumamente fácil.

Este tipo de bases de datos surgieron por la necesidad de resolver el problema que se da cuando una base de datos relacional comienza a tener muchos registros empiezan a ralentizarse y el tipo de relaciones comienzan a ser más complejas.

# Servicios administrados y jerarquía de datos

**Firebase** es un servicio de Google donde puedes tercerizar muchos elementos en la nube.

Jerarquía de datos:

* Base de datos: Contiene toda la información que queremos guardar.
* Colección: Son similares a las tablas en las bbdd relacionales, donde se almacenan los documentos. Una colección no puede existir hasta que un documento sea creado.
* Documento: Es la información que queremos guardar, se guardan en un formato muy similar a JSON.

# Top level collection con firebase

El modelo de bases de datos no relacionales es un poco más cercano al mundo real en su comportamiento.

* Las **top level collections** son las colecciones que se tienen de inmediato o entrada en el proyecto.
* **Firebase** es un servicio que tiene múltiples opciones y está pensado principalmente para aplicaciones móviles y web.

**Tipos de datos en Firebase**.

* String: Cadenas de caracteres.
* Number: Valores numéricos.
* Boolean: Valores lógicos, True o False
* Map: Permite meter documentos dentro de otro documento u objetos JSON.
* Array: Permite almacenar un conjunto de valores de distintos tipos.
* Null: Permite definir si el campo es null.
* Timestamp: Permite almacenar fecha en formato dd/MM/yyyy hh:mm:ss:ml
* Geopoint: Permite guardar una localización geográfica.
* Reference: Permite referenciar a un documento dentro de la colección actual o en otra 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 (consultas recurrentes) es recomendado usar un top level collection. Por el otro lado si se necesita hacer algo intrínseco al documento (que no tendrá muchas consultas recurrentes) es aconsejable usar subcolecciones.

# Transformando tu proyecto en una db no relacional

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 los documentos en lugar de las tuplas.

Recuerda:

Dentro de las Colecciones existen 2 grandes tipos.

* Las Top level collection o colecciones de nivel superior.
* 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.

**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.

# 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.

Muchos lo confunden con **business intelligence** (inteligencia empresarial o BI), pero no es lo mismo, BI está enfocado a la administración y creación de conocimiento sobre el medio, a través del análisis de los datos existentes en una organización o empresa.

Big Data es un gran movimiento de diferentes bases de datos que ayudan a meter y sacar datos sumamente rápido.

# Data warehouse

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, por ej: Los post de Facebook, los post van entrando a cada segundo, pero no se van guardando en la bbdd principal, cuando se dejan de utilizar de manera recurrente, es decir, solicitar ver el mismo post una y otra vez desde la app, éste tipo de dato se guarda en un lugar histórico para su posterior consulta cuando se solicite ver nuevamente en un futuro. Para esto sirven las bbdd de Data Warehouse.

## **Diferencias con Big Data**

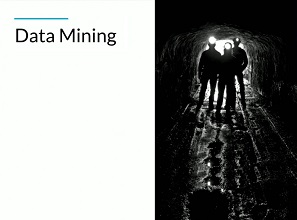
**Big data** almacena mucha cantidad de datos por segundos o por milisegundos, en **Data Warehouse** se guardan muchos más datos todavía de los que se guardan en **BigData** de manera histórica, es decir, archivos muertos, datos que ya no se utilizan, pero que es necesario tenerlos para su posterioridad uso en un futuro.

**Características principales de un Warehouse.**

* Guardar datos de manera histórica, eterna prácticamente.
* Extraer esos datos para poder analizarlos en favor del beneficio del negocio.

Una de las bases de datos utilizada en Warehouse es BigQuery que almacena masivas cantidades de datos en su interior (miles de millones o billones de datos.)

# 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. Por ej: Existen empresas que tienen una gran cantidad de datos históricos, muchas veces ni siquiera en un Data Warehouse, sino que lo tienen en la bbdd de producción conectada a una app o tienen una especie de Warehouse donde han ido guardado todo sin orden, sin coherencia.

Data Mining trata de extraer datos, de tratarlos, darles un sentido e incluso de volver a guardarlos, pero de una manera de que sea aprovechable en el futuro.

## **¿Por qué surge?**

Data Mining surge de la necesidad de solucionar el problema del vertedero de datos que se producían en los Data Warehouse, se almacenaban masivas cantidades de datos en un sólo lugar sin un orden, sin una coherencia, difícil de aprovechar en un futuro, entonces llega Data Mining para solucionarlo. Donde probablemente existan datos valiosísimos dentro de esa mina de datos que se ha construido con los años, aplicando por ej, business intelligence, análisis de datos, etc.

# ETL

Son las siglas de **E**xtract, **T**ransform, **L**oad (extraer, transformar y cargar). Se trata de convertir 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.

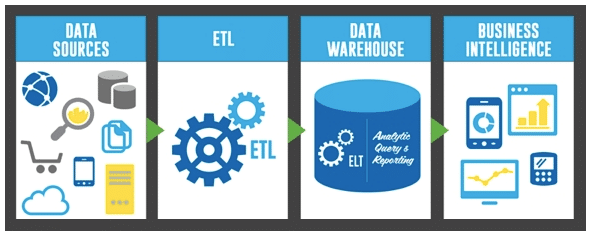
El ETL sirve para dos casos de usos:

* El Data Mining donde ya tienes una gran fuente de datos en el cual no tiene orden de los mismos, para finalmente cargarlo en una nueva bbdd o probablemente en un Data Warehouse ya ordenado para su aprovechamiento.
* En tiempo real o casi en tiempo real, por ej: Utilizamos una bbdd basada en documentos para llevar el estado actual de una app, esos son los datos que están presentes en tiempo real dentro de la app, sin embargo, también se puede utilizar por el otro lado una bbdd que no sea muy útil en el caso de Warehouse como por ej, Big Query.

**¿Dónde es que se conectan estos dos mandos?**

Es ahí donde entra la parte del ETL, es la técnica que nos va a ayudar a tomar los datos en tiempo real de la app, aplicarles trasformaciones nuevamente y guardarlos en un Data Warehouse de forma que estén listos para ser explotados posteriormente en análisis de datos, analítica, business intelligence e incluso Machine Learning.

ETL trata de trasformar lo que tenemos como carbón (una bbdd sin forma o nuestra app en tiempo real) en un diamante, algo que realmente nos ofrezca valor y que nos permita analizar un poco más a detalle lo que el negocio está haciendo, logrando así poder tomar decisiones de manera inteligente basándonos en estos datos.



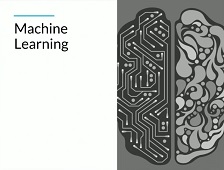
# Business intelligence

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.

Business intelligence se vale de Data Mining, ETL para poder realizar decisiones, graficas, etc, dentro de una empresa.

# 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, por ejemplo: tenemos un post con una base de datos muy grande a la cual queremos saber los temas de política, para esto se entrena previamente al algoritmo de Machine Learning con una serie de datos para identificar que es y como luce un tema de política, una vez entrenado, se le proporciona la bbdd completa de los post, los resultados que arrojará serán patrones donde encuentre temas de política u otros que no tengan que ver con el contenido para lo cual fue entrenado pero, hay ciertas similitudes en esa data como, dentro del post se menciona a un político, una acción política, una causa, etc.
* Predicción, por ejemplo: Tenemos una base de datos de ventas y quieres analizar la jerarquía de ventas, donde se compra más, etc. Para entrenar al algoritmo de Machine Learning se le pasa las ventas de años anteriores, se entrena y una vez entrenado el algoritmo te entregará donde hubo mayores ventas, las menores, si hay una relación geográfica, una relación de género, entre otras. Esto permitirá ver lo que ha sucedido históricamente y preguntar al algoritmo que sucederá en el futuro si hacemos determinados cambios en las ventas.

**Diferencias con business intelligence.**

Business Intelligence: Alguien está buscando hacer sentido a los datos que ya posees.

Machine Learning: Nos ayuda hacer modelos los cuales no van a tratar de ir por un patrón, sino que van a encontrar patrones fortuitos, es decir, agarra la gran cantidad de datos ya almacenada e intenta encontrar correlaciones que no estaban estipuladas para el negocio. Analiza patrones que un ser humano no encontraría o se tardaría mucho tiempo en hacerlo.

# 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.