

Fundamentos de bases de datos – Notas

1. 12 reglas de Codd para RDBMS

El Dr. Edgar Frank Codd (19 de agosto de 1923 - 18 de abril de 2003) fue un informático mientras trabajaba para IBM e inventó el modelo relacional para la gestión de bases de datos (base teórica para bases de datos relacionales). Codd propuso trece reglas (numeradas del cero al doce) y dijo que, si un Sistema de gestión de bases de datos cumple con estas reglas, puede llamarse como un Sistema de gestión de bases de datos relacionales. Estas reglas se denominan reglas de Codd12. Casi ningún producto comercial sigue a todos.

- **Regla cero:**

El sistema debe calificar como relacional, como base de datos y como sistema de gestión. Para que un sistema califique como un sistema de gestión de bases de datos relacionales (RDBMS), ese sistema debe usar sus recursos relacionales (exclusivamente) para gestionar la base de datos.

Las otras 12 reglas derivan de esta regla. Las reglas son las siguientes:

- I. **Regla 1** La regla de la información:

Toda la información en la base de datos debe representarse de una sola manera, es decir, mediante valores en posiciones de columna dentro de filas de tablas.

- II. **Regla 2:** La regla de acceso garantizado:

Todos los datos deben ser accesibles. Esta regla es esencialmente una reformulación del requisito fundamental para las claves primarias. Dice que cada valor escalar individual en la base de datos debe ser lógicamente direccionable especificando el nombre de la tabla que contiene, el nombre de la columna que contiene y el valor de la clave primaria de la fila que contiene.

- III. **Regla 3** Tratamiento sistemático de valores nulos:

El RDBMS debe permitir que cada campo permanezca nulo (o vacío). Específicamente, debe admitir una representación de "información faltante e información inaplicable" que es sistemática, distinta de todos los valores regulares (por ejemplo, "distinta de cero o cualquier otro número", en el caso de valores numéricos) e independiente de los datos tipo. También está implícito que tales representaciones deben ser manipuladas por el DBMS de manera sistemática.

- IV. **Regla 4** Catálogo en línea activo basado en el modelo relacional:
El sistema debe admitir un catálogo relacional en línea, en línea, al que puedan acceder los usuarios autorizados mediante su lenguaje de consulta habitual. Es decir, los usuarios deben poder acceder a la estructura de la base de datos (catálogo) utilizando el mismo lenguaje de consulta que utilizan para acceder a los datos de la base de datos.
- V. **Regla 5** La regla integral del sub-idioma de datos:

El sistema debe admitir al menos un lenguaje relacional que
1. Tiene una sintaxis lineal
 2. Se puede utilizar tanto de forma interactiva como dentro de los programas de aplicación,
 3. Admite operaciones de definición de datos (incluidas las definiciones de vista), operaciones de manipulación de datos (actualización y recuperación), restricciones de seguridad e integridad y operaciones de gestión de transacciones (inicio, confirmación y reversión).
- VI. **Regla 6** La regla de actualización de la vista:
Todas las vistas que pueden actualizarse teóricamente deben ser actualizadas por el sistema.
- VII. **Regla 7** inserción, actualización y eliminación de alto nivel:
El sistema debe admitir operadores de inserción, actualización y eliminación establecidos en un momento. Esto significa que los datos se pueden recuperar de una base de datos relacional en conjuntos construidos con datos de múltiples filas y / o tablas múltiples. Esta regla establece que las operaciones de inserción, actualización y eliminación deben ser compatibles con cualquier conjunto recuperable en lugar de solo para una sola fila en una sola tabla.
- VIII. **Regla 8** Independencia de los datos físicos:
Los cambios en el nivel físico (cómo se almacenan los datos, ya sea en matrices o listas vinculadas, etc.) no deben requerir un cambio en una aplicación basada en la estructura.
- IX. **Regla 9** Independencia de datos lógicos:
Los cambios en el nivel lógico (tablas, columnas, filas, etc.) no deben requerir un cambio en una aplicación basada en la estructura. La independencia de datos lógicos es más difícil de lograr que la independencia de datos físicos.
- X. **Regla 10** Independencia de integridad:

Las restricciones de integridad deben especificarse por separado de los programas de aplicación y almacenarse en el catálogo. Debe ser posible cambiar tales restricciones cuando sea apropiado sin afectar innecesariamente las aplicaciones existentes.

XI. **Regla 11** Independencia de distribución:

La distribución de porciones de la base de datos a varias ubicaciones debe ser invisible para los usuarios de la base de datos. Las aplicaciones existentes deberían continuar funcionando con éxito:

1. cuando se introduce por primera vez una versión distribuida del DBMS; y
2. cuando los datos distribuidos existentes se redistribuyen alrededor del sistema.

XII. **Regla 12:** La **regla de** no subversión:

Si el sistema proporciona una interfaz de bajo nivel (registro a la vez), entonces esa interfaz no puede usarse para subvertir el sistema, por ejemplo, omitiendo una restricción de seguridad o integridad relacional.

2. 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.

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.

*Entidades:

- Es algo similar a un objeto (programación orientada a objetos) y representa algo en el mundo real, incluso algo abstracto.
- Se representan con su nombre encerradas en un rectángulo
- Tienen atributos que son las cosas que los hacen ser una entidad
- Por convención las entidades se ponen en plural (porque una entidad representan un grupo de objetos)
- Tipos de entidades:
- + Fuertes - No dependen de nadie para existir

+ Débiles - Dependen de una entidad fuerte para existir, se representan con un rectángulo pero tienen doble línea

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

Por existencia: Se les asigna una clave propia.

- Atributos:
 - Los atributos pueden ser:
 - Atributos - son representados con óvalos
 - Atributos Multivaluado - Son aquellos que son dos o más del mismo, representados con DOS óvalos (son varios en uno, como llantas, ojos, piernas)
 - Atributos Compuestos - Son aquellos que están compuestos por más atributos
 - Atributos Derivados - Se obtienen a partir de otros y usan un ovalo con línea punteada
 - Atributos Llave - Son aquellos que identifican a la entidad y no pueden ser repetidos, pueden ser: <h1>Naturales - Inherentes al objeto como un número de serie</h1><h1>Clave artificial - No son inherentes al objeto y se asigna de manera arbitraria</h1>



3. Entidades de plaza blog. (PRIMEROS CONCEPTOS PARA EL PROYECTO).

Pasos del proyecto:

- I. Identificar las entidades.
- II. Identificar los atributos de la entidad .

4. 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

5. 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.

6. DIAGRAMA FÍSICO: Tipo de datos y constante.

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
- Numerous: INTEGER, BIGINT, SMALLINT, DECIMAL(n,s), NUMERIC(n,s)
- Fecha/hora: DATE, TIME, DATETIME, TIMESTAMP

Tipo de Columna	"Cero" Valor
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	0000000000000000
TIME	'00:00:00'
YEAR	0000

- 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
- 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.

En MySQL workbench, tenemos estas opciones cuando se está creando una tabla

- Primary Key
- Not Null
- Unique
- Binary
- Unsigned
- Zero Filled
- Autoincremental:
- Columna generada.

7. Tipos de datos en MYSQL

I. DATOS NUMÉRICOS

Listado de cada uno de los tipos de dato numéricos en MySQL, su ocupación en disco y valores.

- INT (INTEGER): Ocupación de 4 bytes con valores entre -2147483648 y 2147483647 o entre 0 y 4294967295.
- SMALLINT: Ocupación de 2 bytes con valores entre -32768 y 32767 o entre 0 y 65535.
- TINYINT: Ocupación de 1 bytes con valores entre -128 y 127 o entre 0 y 255.

- MEDIUMINT: Ocupación de 3 bytes con valores entre -8388608 y 8388607 o entre 0 y 16777215.

BIGINT: Ocupación de 8 bytes con valores entre -8388608 y 8388607 o entre 0 y 16777215.

- DECIMAL (NUMERIC): Almacena los números de coma flotante como cadenas o string.
- FLOAT (m,d): Almacena números de coma flotante, donde 'm' es el número de dígitos de la parte entera y 'd' el número de decimales.
- DOUBLE (REAL): Almacena número de coma flotante con precisión doble. Igual que FLOAT, la diferencia es el rango de valores posibles.
- BIT (BOOL, BOOLEAN): Número entero con valor 0 o 1.

II. FORMATO FECHA

- DATE: Válido para almacenar una fecha con año, mes y día, su rango oscila entre '1000-01-01' y '9999-12-31'.
- DATETIME: Almacena una fecha (año-mes-día) y una hora (horas-minutos-segundos), su rango oscila entre '1000-01-01 00:00:00' y '9999-12-31 23:59:59'.
- TIME: Válido para almacenar una hora (horas-minutos-segundos). Su rango de horas oscila entre -838-59-59 y 838-59-59. El formato almacenado es 'HH:MM:SS'.
- TIMESTAMP: Almacena una fecha y hora UTC. El rango de valores oscila entre '1970-01-01 00:00:01' y '2038-01-19 03:14:07'.
- YEAR: Almacena un año dado con 2 o 4 dígitos de longitud, por defecto son 4. El rango de valores oscila entre 1901 y 2155 con 4 dígitos. Mientras que con 2 dígitos el rango es desde 1970 a 2069 (70-69).

III. FORMATO STRING

- CHAR: Ocupación fija cuya longitud comprende de 1 a 255 caracteres.
- VARCHAR: Ocupación variable cuya longitud comprende de 1 a 255 caracteres.
- TINYBLOB: Una longitud máxima de 255 caracteres. Válido para objetos binarios como son un fichero de texto, imágenes, ficheros de audio o vídeo. No distingue entre minúsculas y mayúsculas.
- BLOB: Una longitud máxima de 65.535 caracteres. Válido para objetos binarios como son un fichero de texto, imágenes, ficheros de audio o vídeo. No distingue entre minúsculas y mayúsculas.

- MEDIUMBLOB: Una longitud máxima de 16.777.215 caracteres. Válido para objetos binarios como son un fichero de texto, imágenes, ficheros de audio o vídeo. No distingue entre minúsculas y mayúsculas.
- LONGBLOB: Una longitud máxima de 4.294.967.298 caracteres. Válido para objetos binarios como son un fichero de texto, imágenes, ficheros de audio o vídeo. No distingue entre minúsculas y mayúsculas.
- SET: Almacena 0, uno o varios valores una lista con un máximo de 64 posibles valores.
- ENUM: Igual que SET, pero solo puede almacenar un valor.
- TINYTEXT: Una longitud máxima de 255 caracteres. Sirve para almacenar texto plano sin formato. Distingue entre minúsculas y mayúsculas.
- TEXT: Una longitud máxima de 65.535 caracteres. Sirve para almacenar texto plano sin formato. Distingue entre minúsculas y mayúsculas.
- MEDIUMTEXT: Una longitud máxima de 16.777.215 caracteres. Sirve para almacenar texto plano sin formato. Distingue entre minúsculas y mayúsculas.
- LONGTEXT: Una longitud máxima de 4.294.967.298 caracteres. Sirve para almacenar texto plano sin formato. Distingue entre minúsculas y mayúsculas.

8. Diagrama físico: Normalización

Las bases de datos relacionales se normalizan para:

Evitar la redundancia de los datos.

Disminuir problemas de actualización de los datos en las tablas.

Proteger la integridad de los datos.

Facilitar el acceso e interpretación de los datos.

Reducir el tiempo y complejidad de revisión de las bases de datos.

Optimizar el espacio de almacenamiento.

Prevenir borrados indeseados de datos.

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:

- I. Primera forma normal (1FN): Atributos atómicos (Sin campos repetidos)
- II. Segunda forma normal (2FN): Cumple 1FN y cada campo de la tabla debe depender de una clave única.
- III. Tercera forma normal (3FN): Cumple 1FN y 2FN y los campos que NO son clave, NO deben tener dependencias.
- IV. Cuarta forma normal (4FN): Cumple 1FN, 2FN, 3FN y los campos multivaluados se identifican por una clave única.

EXPLICACIÓN DETALLADA:

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.

a. 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 de 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.

b. 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 si tenemos datos que pertenecen a diversas entidades, cada entidad debe tener un campo clave separado. Por ejemplo:

alumnos				
alumno_id	alumno	nivel_curso	nombre_curso	materia
1	Juanito	Maestría	Data engineering	MySQL
1	Juanito	Maestría	Data engineering	Python
2	Pepito	Licenciatura	Programación	MySQL
2	Pepito	Licenciatura	Programación	Python

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, ya que una materia. En el ejemplo anterior, quedaría de la siguiente manera:

alumnos			
alumno_id	alumno	nivel_curso	nombre_curso
1	Juanito	Maestría	Data engineering
2	Pepito	Licenciatura	Programación

materias		
materia_id	alumno_id	materia
1	1	MySQL
2	1	Python
3	2	MySQL
4	2	Python

c. 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.

alumnos		
alumno_id	alumno	curso_id
1	Juanito	1
2	Pepito	2

cursos		
curso_id	nivel_curso	nombre_curso
1	Maestría	Data engineering
2	Licenciatura	Programación

materias		
materia_id	alumno_id	materia
1	1	MySQL
2	1	Python
3	2	MySQL
4	2	Python

d. 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.

alumnos		
alumno_id	alumno	curso_id
1	Juanito	1
2	Pepito	2

cursos		
curso_id	nivel_curso	nombre_curso
1	Maestría	Data engineering
2	Licenciatura	Programación

materias	
materia_id	materia
1	MySQL
2	Python

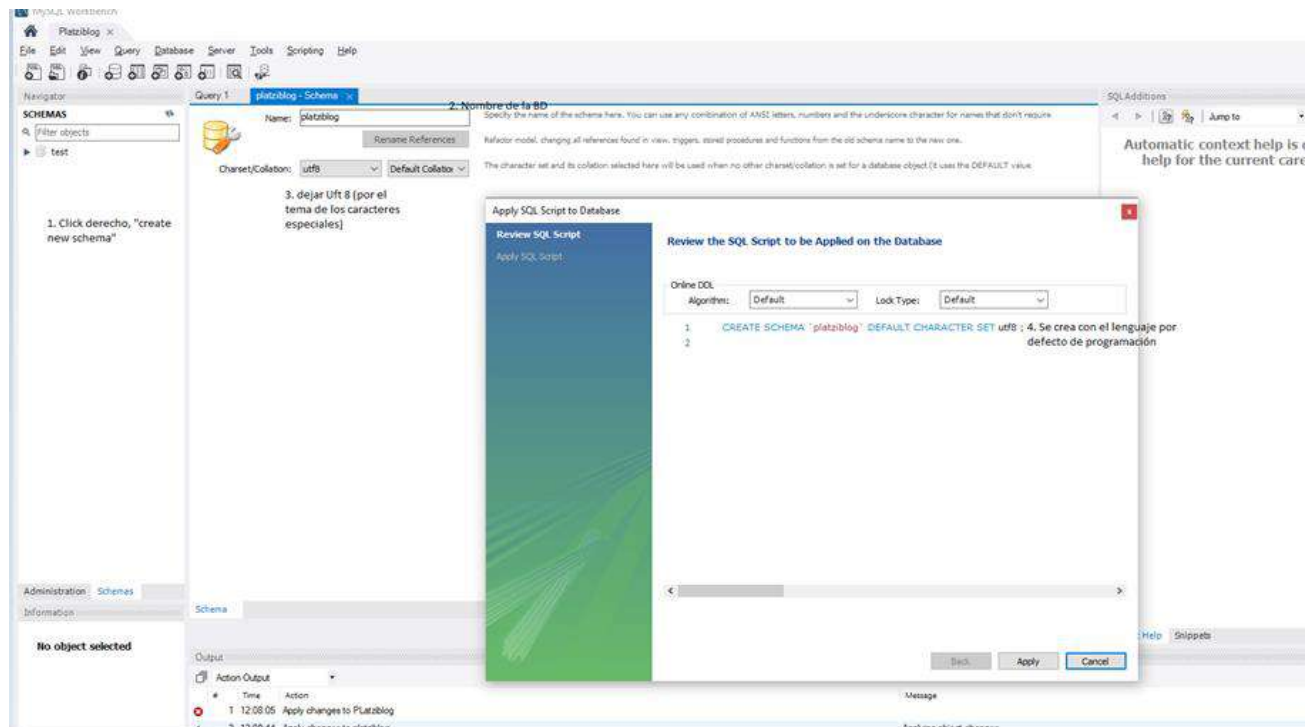
materias_por_alumno		
mpa_id	materia_id	alumno_id
1	1	1
2	2	1
3	1	2
4	2	2

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.

9. Clientes gráficos

Son aplicaciones que permiten conectarse al servidor y enviarle conjuntos de órdenes para manipular las bases de datos y los datos que contienen, a través de secuencias de órdenes en SQL. Los más clásicos son los que ofrecen una línea de órdenes («comandos» para los incultos), pero los más interesantes para usuarios de tipo novel y medio son aquellos que ofrecen una interfaz gráfica. Si además incluyen herramientas gráficas de diseño de modelos de datos E-R, pues mejor que mejor.



10. SQL – Tipos de lenguaje

SQL (structured query language) tiene los siguientes lenguajes:

- I. 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 (schemas)
- 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.

- II. DML - Data Manipulation Language o Lenguaje de Manipulación de Datos

Son un conjunto de instrucciones que apoyarán al proceso de construcción de la BD y afectan los registros en una tabla. Estas son operaciones básicas que realizamos sobre datos

tales como seleccionar algunos registros de una tabla, ar nuevos registros, eliminar registros innecesarios y actualizar / modificar registros existentes.

Opciones DML

- SELECT: para seleccionar registros de tablas
- INSERT: para insertar nuevos registros
- UPDATE: para actualizar y modificar registros
- DELETE: para eliminar registros existentes.

III. DCL - Lenguaje de Control de Datos

Las declaraciones DLC son las encargadas de controlar el acceso de los usuarios a las BD.

Opciones DCL

- GRANT:

Declaración que permite a los usuarios leer / escribir en objetos que digamos de la BD.

- REVOKE:

Es la que ofrece a los usuarios estar sin permiso de lectura / escritura en objetos de la BD.

IV. TLC - Lenguaje de Control de Transacciones

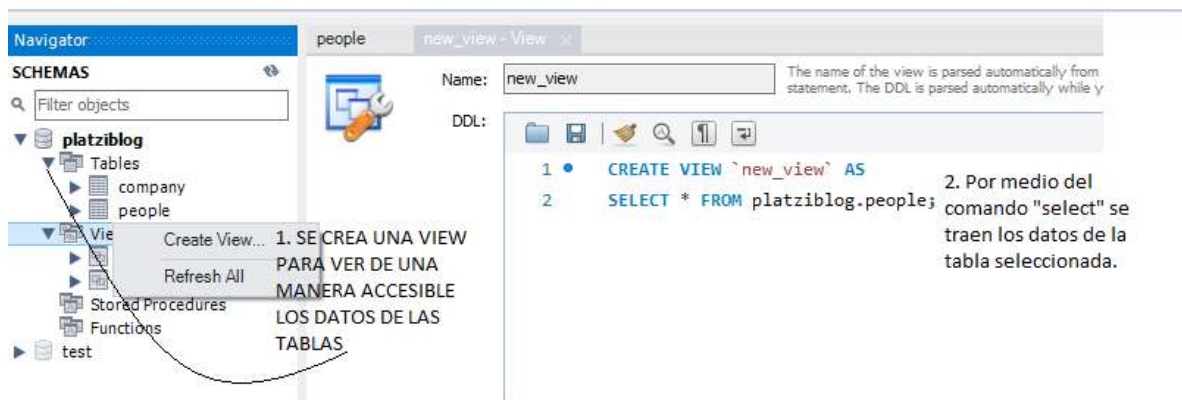
Instrucciones que permiten administrar transacciones y tener integridad de datos dentro de las declaraciones SQL. Se gestiona a través de las siguientes declaraciones

- BEGIN Transaction: Nos permite abrir una transacción
- COMMIT Transaction: Ofrece confirmar una transacción
- ROLLBACK Transaction: Devuelve una transacción en caso de error cometido.

11.Create view y DDL Alter

Una vez se tiene creada la tabla, en el caso del ejemplo es platziblog.people. Se procede a crea una “vista” o “view” con el fin de tomar los datos de una o distintas tablas y ponerlos de una manera presentable y usable. Esto quiere decir que puede tomar datos o columnas de diferentes tablas.

Al igual que en las tablas, en las vistas podemos alterar, y borrar por medio de los comandos de alter y drop. También se puede hacer mediante un click derecho en cada uno de los campos de tablas o vistas.



12.DML

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

- Insert into ("nombre de la tabla") ("Campos o columnas separadas por una coma)

Values ('Hernández', 'Laura', 'Calle 21', 'Bogotá')

NOTA: Los values claramente deben ser ingresados en orden.

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

- UPDATE people

SET last_name = 'Chávez', city= 'Mérida'

WHERE person_id = 1; = 1;

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

"NO OLVIDES PONER EL WHERE EN EL DELETE FROM" 😊

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

Select (las columnas que se quieren traer de la base de datos, separadas por coas) FROM la tabla de la que se quiere traer la información.

13. ¿Qué tan estándar es SQL?

En la clase se hizo un mini repaso escribiendo las líneas en sql:

Create table

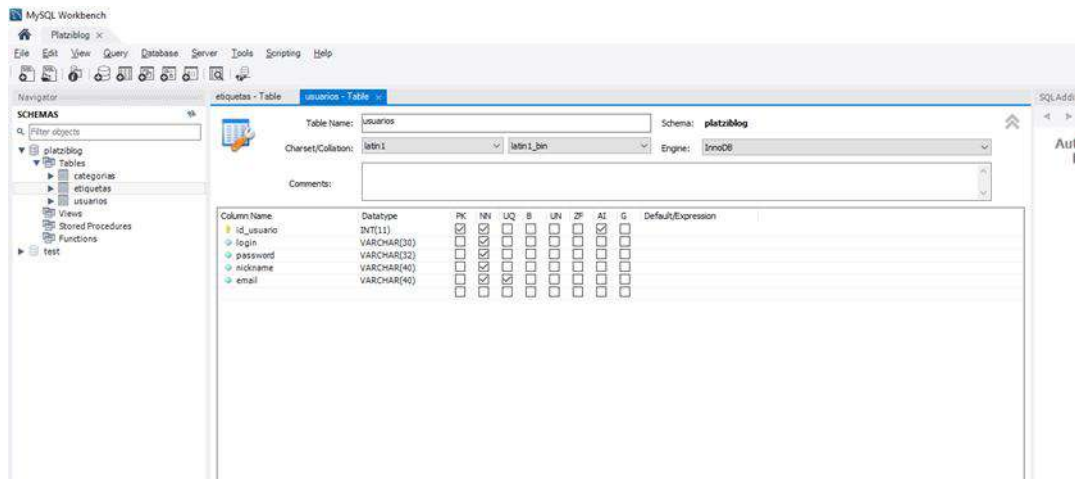
Insert into (definir las columnas) values (registrar los datos)

Drop table

Después de esto literalmente se copió y pegó las mismas líneas en Google cloud platform en un ambiente de PostgreSQL y en cuadro de comandos se definió que usa como base los comandos estudiados de SQL tanto DDL y DML y se llegó la conclusión en que la mayoría de RDBMS tiene con base SQL estudiado lo cual nos permitirá aplicarlo en diferentes ambientes.

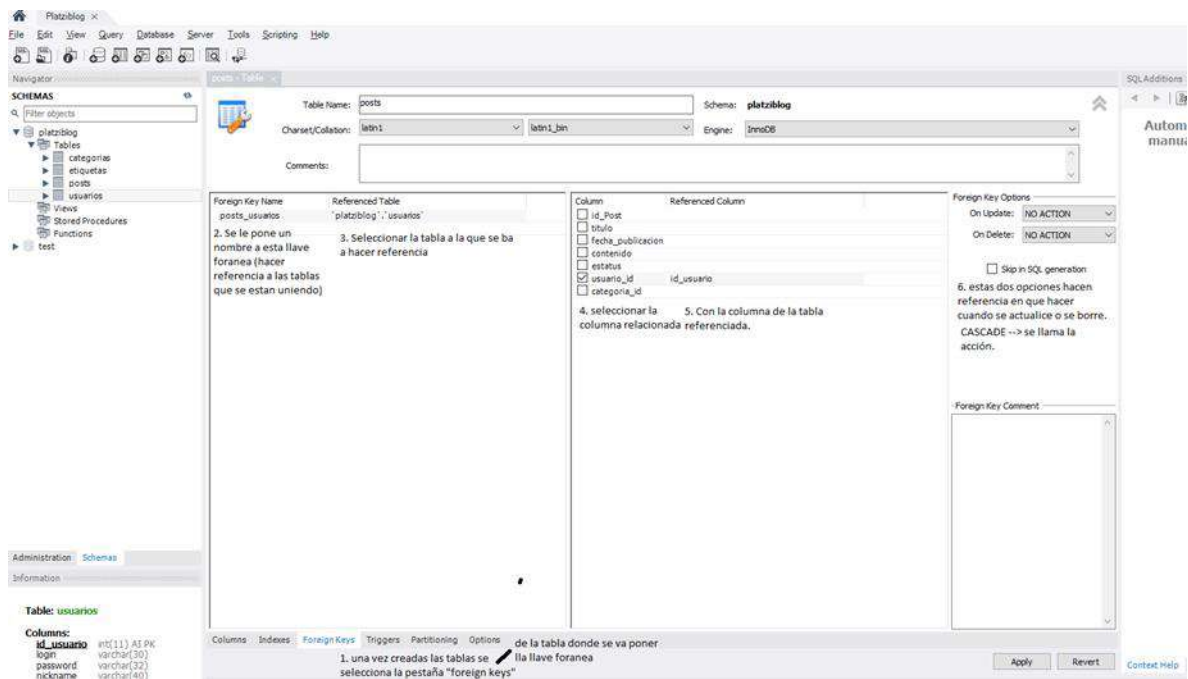
14. Creación de tablas Platzi blog

Teniendo en cuenta las entidades y atributos definidos en el mapa físico se procede a crear las tablas den la base de datos como se ha enseñado normalmente se crean primero las tablas que no tienen dependencia



15. Creación de platziblog: Tablas dependientes

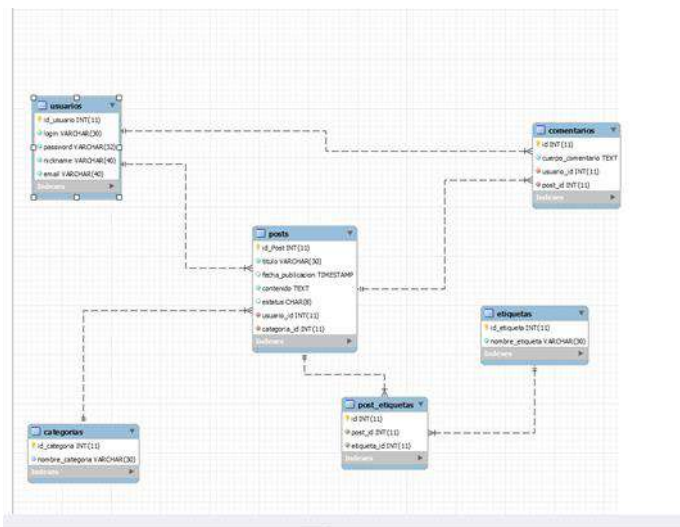
En esta clase para la entidad post creamos sus atributos y las llaves foránea proveniente de las entidades categorías y usuarios, como se muestra en la siguiente imagen:



16. Creación 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.



17. ¿Por qué son tan importantes las consultas?

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.

18. Estructura básica de un query

Una query es la forma en la que se estructuran las preguntas a una base de datos.

Las Queries pueden ser simples o muy complejas.

QUERIES:

- I. SELECT: Nos permite traer los datos que queremos mostrar. Ejemplo.

SELECT city, count (*) **AS** total

FROM people (→ de donde vamos a tomar los datos)

WHERE active = true (que es un booleano)*

GROUP BY city (nos permite agrupar por “criterio”)

ORDER BY total **DESC** (Ordena por total).

HAVING total >= 2; (Filtra del total cuando haya más de dos registros por criterio.).

SELECT * (EL ASTERISCO SOLO, NOS SELECCIONA TODO LO QUE EL QUERY ARROJARÁ)

FROM: señala de donde se tomará la información.

Where: permite filtrar los datos dependiendo lo que quedamos.

Notas:

- ✓ * Booleano: Un valor booleano representa un valor que puede ser verdadero (TRUE) o falso (FALSE). Una expresión o predicado booleano puede producir como resultado un valor desconocido, que se representa como el valor nulo.

19. SELECT

SELECT se encarga de proyectar o mostrar datos.

Estructura:

- i. Select (*) → trae toda la tabla
- ii. SELECT (nombre de un atributo) (se pueden traer varios atributos separados por coma).

SELECT portada, posts, comments.
- iii. FROM (Nombre de la tabla a seleccionar).

FROM posts;
- iv. FROM (NOMBRE DE LA BD/SCHEMA SI NO ESTA POR DEFAULT) (nombre de la tabla seleccionada)

FROM platziblog.posts;

- El nombre de las columnas o campos que estamos consultando puede ser cambiado por AS después del nombre del campo y poniendo el nuevo que queremos poner.

SELECT titulo AS encabezado

FROM posts;

- Existe una función de SELECT para poder contar la cantidad de registros. Esa información (número) será el resultado del Query.

SELECT COUNT (*)

FROM POSTS;

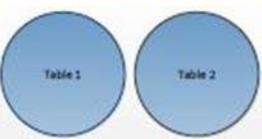
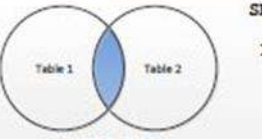

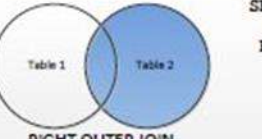




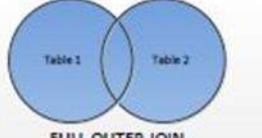
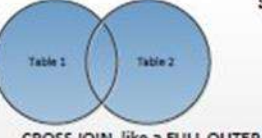
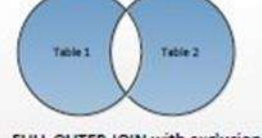
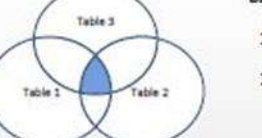
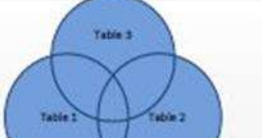
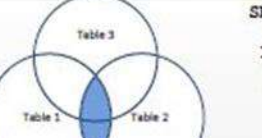
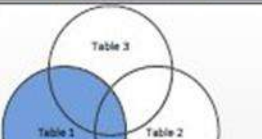
20.FROM (SENTENCIA JOIN)

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.

TSQL JOIN TYPES

Created by Steve Stedman

 <p>SELECT * FROM Table_1; SELECT * FROM Table_2;</p> <p>SELECT from two tables</p>	 <p>SELECT * FROM Table_1 t1 INNER JOIN Table_2 t2 ON t1.id = t2.fk;</p> <p>INNER JOIN</p>
 <p>SELECT * FROM Table_1 t1 LEFT JOIN Table_2 t2 ON t1.id = t2.fk;</p> <p>LEFT OUTER JOIN</p>	 <p>SELECT * FROM Table_1 t1 RIGHT JOIN Table_2 t2 ON t1.id = t2.fk;</p> <p>RIGHT OUTER JOIN</p>
 <p>SELECT * FROM Table_1 t1 WHERE EXISTS (SELECT 1 FROM Table_2 t2 WHERE t2.id = t1.fk);</p> <p>SEMI JOIN – Similar to INNER JOIN, with less duplication from Table 2.</p>	 <p>SELECT * FROM Table_1 t1 WHERE NOT EXISTS (SELECT 1 FROM Table_2 t2 WHERE t2.id = t1.fk);</p> <p>ANTI SEMI JOIN</p>
 <p>SELECT * FROM Table_1 t1 LEFT JOIN Table_2 t2 ON t1.id = t2.fk WHERE t2.fk is null;</p> <p>LEFT OUTER JOIN with exclusion – replacement for a NOT IN</p>	 <p>SELECT * FROM Table_1 t1 RIGHT JOIN Table_2 t2 ON t1.id = t2.fk WHERE t1.id is null;</p> <p>RIGHT OUTER JOIN with exclusion – replacement for a NOT IN</p>
 <p>SELECT * FROM Table_1 t1 FULL OUTER JOIN Table_2 t2 ON t1.id = t2.fk;</p> <p>FULL OUTER JOIN</p>	 <p>SELECT * FROM Table_1 t1 CROSS JOIN Table_2 t2;</p> <p>CROSS JOIN, like a FULL OUTER JOIN with out specifying JOIN condition.</p>
 <p>SELECT * FROM Table_1 t1 FULL OUTER JOIN Table_2 t2 ON t1.id = t2.fk WHERE t1.id is null OR t2.fk is null;</p> <p>FULL OUTER JOIN with exclusion – replacement for a double NOT IN</p>	 <p>SELECT * FROM Table_1 t1 INNER JOIN Table_2 t2 ON t1.id = t2.fk INNER JOIN Table_3 t3 ON t1.id = t3.fk;</p> <p>Two INNER JOINS</p>
 <p>SELECT * FROM Table_1 t1 FULL OUTER JOIN Table_2 t2 ON t1.id = t2.fk FULL OUTER JOIN Table_3 t3 ON t1.id = t3.fk;</p> <p>Two FULL OUTER JOINS</p>	 <p>SELECT * FROM Table_1 t1 INNER JOIN Table_2 t2 ON t1.id = t2.fk LEFT OUTER JOIN Table_3 t3 ON t1.id = t3.fk;</p> <p>INNER JOIN and a LEFT OUTER JOIN</p>
 <p>SELECT * FROM Table_1 t1 LEFT JOIN Table_2 t2 ON t1.id = t2.fk LEFT JOIN Table_3 t3 ON t1.id = t3.fk;</p> <p>Two LEFT OUTER JOINS</p>	

21. Utilizando la sentencia FROM

DIFERENCIA

I. LEFT OUTER JOIN.

En este join, se une los datos especificados de la tabla A (la primera que se pone, con lo que coincide de la tabla B (La Segunda que se pone). Esto quiere decir que se va a traer todo de la tabla A y lo que coincida con la tabla B.

Estas coincidencias se dan teniendo cuenta las relaciones de las tablas previamente establecidas. Es decir, teniendo en cuenta la Key de la tabla A y su respectiva Foreign Key en la Tabla B.

■ EJEMPLO:

```
SELECT *  
FROM usuarios ➡ Tabla A  
LEFT OUTER JOIN posts ON usuarios.id = posts.usuario_id ;
```

Tabla B **Llave Tabla A** **Llave foránea en la tabla B**

En este ejemplo está diciendo que tráiganme todos los campos de la tabla usuarios y todos los usuarios, tengan o no tengan posts.

II. RIGHT OUTER JOIN

Este join funciona igual que el left. Trae todos los datos de la tabla B y los que coinciden de la tabla A.

■ EJEMPLO:

```
SELECT *  
  
FROM usuarios  
  
RIGHT JOIN posts ON usuarios.id = post.usuario_id  
  
WHERE posts.usuario_id IS NULL;
```

(este es una condición para que nos traiga los “posts huérfanos” o datos de la tabla B que no tengan relación con la tabla A).

INTERSECCIÓN

III. INNER JOIN

Trae los datos que están relacionados de ambos lados. (entre tabla A y Tabla B)

■ EJEMPLO 3:

```
SELECT *
```

```
FROM usuarios
```

```
INNER JOIN posts ON usuarios.id = post.usuario_id
```

UNIÓN En algunas bases de datos, está la opción de usar “FULL OUTER JOIN” para traer todos los datos de ambas tablas. Pero como no es estándar en todas las BD se hace de la siguiente manera:

IV. UNIÓN SELECT COMPUESTO

■ EJEMPLO 4:

```
SELECT *
```

```
FROM usuarios
```

```
LEFT JOIN posts ON usuarios.id = posts.usuarios_id
```

```
UNION
```

```
SELECT *
```

```
FROM usuarios
```

```
RIGHT JOIN posts ON usuarios.id =posts.usuarios_id;
```

V. DIFERENCIA SIMETRICA CON WHERE

■ EJEMPLO 4:

```
SELECT *
```

```
FROM usuarios
```

```
LEFT JOIN posts ON usuarios.id = posts.usuarios_id
```

```
WHERE posts.usuario_id IS NULL
```

```
UNION
```

```
SELECT *
```

```
FROM usuarios
```

```
RIGHT JOIN posts ON usuarios.id =posts.usuarios_id
```

```
WHERE posts.usuario_id IS NULL;
```

22. Sentencia WHERE

Es la sentencia que ayuda filtrar registros (no columnas)

- ✓ Criterios, como fecha, cantidades, etc.

Ejemplo 5:

a. **SELECT ***
from posts
where id <= 50;

NOTA: POR ESO ES IMPORTANTE QUE LOS DATOS DE ID SEAN NÚMERICOS Y NO CADENA DE TEXTO.

b. **SELECT***
FROM posts
WHERE titulo **LIKE** '%escandalo%';

%escandalo → trae solo lo que termina en la palabra

escandalo% → trae todo lo que empieza con la palabra

Nota 2: %% → este signo trae lo que se escriba dentro. No olvidar poner en comillas ya que es un string.

c. **SELECT***
FROM posts
WHERE fecha_publicacion > '2025-01-01'

Nota 3: con la fecha funciona igual, se puede filtrar entre fechas.

d. **SELECT***
FROM posts
WHERE fecha_publicacion **BETWEEN** '2023-01-01' **AND** '2025-12-31'

e. **SELECT***
FROM posts
WHERE **YEAR**(fecha_publicacion) **BETWEEN** '2023' **AND** '2024';

f. **SELECT***
FROM posts
WHERE MONTH(fecha_publicacion) = 04;

Utilizando la sentencia **WHERE** nulo y no nulo

a. **SELECT***
FROM posts
WHERE usuario_id IS NOT NULL;

b. **SELECT***
FROM posts
WHERE usuario_id IS NULL;

c. **SELECT***
FROM posts
WHERE usuario_id IS NOT NULL
AND estatus ='activo'
AND id < 50
AND categoria_id = 2
AND YEAR(fecha_publicacion) = '2025'
;

NOTA: AND nos permite unir consultas en where

23. GROUP BY

SELECT estatus, **COUNT (*)** post_quantity
FROM posts
GROUP BY estatus;

Lo que se nos muestra es diferenciado por el estatus cuantos posts hay →

ESTATUS	POST_QUANTITY
Activo	17
Inactivo	5

Es útil cuando se necesita para agrupar por alguna categoría o por algún parámetro y que te arroje una cantidad por grupos.

Es interesante para datos agregados para informes por ejemplo cantidades de posts en un tiempo, un gasto por parámetro, etc.

```
SELECT YEAR(fecha_publicación) AS post_year, COUNT(*) AS post_quantity  
FROM Posts  
GROUP BY post_year;
```

Esto lo que nos está diciendo es: seleccionar solo el año del campo (fecha_publicacion) nombrarlo como "post_year", luego contar todos los registros y nombrar la columna como "post_quantity) De Posts, agrupado por "post_year". Y nos lo traerá de la siguiente manera:

Post_year | Post_quantity

2022	5
2021	2
2020	3 etc

```
Select MONTHNAME(fecha_publicación) AS post_MONTHr, COUNT(*) AS  
post_quantity  
FROM Posts  
GROUP BY post_year;
```

LOS MISMO, PERO NOS TRAE EL NOMBRE DEL MES.

```
SELECT estatus, YEAR(fecha_publicación) AS post_year, COUNT(*) AS  
post_quantity  
FROM Posts  
GROUP BY estatus, post_year; (agrupar por varios criterios)
```

```
SELECT order_purchase_timestamp, substring(order_purchase_timestamp,1,11)  
AS 'date_purchase'  
FROM ORDERS_STATUS;
```

24. ORDER BY – HAVING

ORDER BY → ordena los datos por un criterio, campo, etc. (Fecha, Cadena → título)

Hay dos maneras de ordenarlo, de forma ascendente (ASC = ascending) o descendente (DESC = DESCENDING).

EJEMPLO 1:

```
SELECT *  
FROM posts  
ORDER BY fecha_publicacion ASC;
```

```
SELECT *  
FROM posts  
ORDER BY fecha_publicacion DESC;
```

```
SELECT *  
FROM posts  
ORDER BY título ASC/DESC;
```

```
SELECT *  
FROM posts  
ORDER BY usuario_id ASC/DESC;
```

Con la sentencia **LIMIT** es un complemento al order by, nos ayuda a limitar la cantidad de datos que queremos traer.

EJEMPLO 2:

```
SELECT *  
FROM posts  
ORDER BY fecha_publicación DESC  
LIMIT 5;
```

Con esta opción Podemos traer información definida previamente, como un top 5, 10, etc, de algo.

HAVING → tiene una similitud con where. La diferencia está en que cuando quieras hacer filtros de duplas o filas agrupadas (usando group by, agrupando por fecha y mostrando la cantidad de datos en esa fecha) se debe usar “having” ya que where no reconoce estas agrupaciones de filas y ni cambios de nombres. Campos dinámicos

EJEMPLO 3:

SELECT MONTHNAME(fecha_publicación) **AS** post_month, estatus, **COUNT(*) AS**
post_quantity

FROM posts

WHERE post_quantity > 1 ---→ **ERROR.** ya que post_quantity no ocurre hasta la
sentencia "group by" y no reconoce también el nombre.

GROUP BY estatus, post_month

ORDER BY post_month

(forma correcta)

SELECT MONTHNAME(fecha_publicación) **AS** post_month, estatus, **COUNT(*) AS**
post_quantity

FROM posts

GROUP BY estatus, post_month

HAVING post_quantity > 1 -→ después del group by debido que es aquí donde se
agrupan los posts, dando el número y es llamado de esa manera.

ORDER BY post_month

The screenshot shows a MySQL IDE window with a SQL query and its results. The query is:

```
1 SELECT MONTHNAME(fecha_publicacion) AS post_month, estatus, COUNT(*) AS post_quantity
2 FROM posts
3 GROUP BY estatus, post_month
4 HAVING post_quantity > 1
5 ORDER BY post_month
6 ;
```

The results are displayed in a table with the following columns: post_month, estatus, post_quantity. The data rows are:

post_month	estatus	post_quantity
December	activo	3
January	activo	2
May	activo	2
October	activo	2

The IDE also shows the Action Output panel with the following response:

```
Response
Q, 1000 22 row(s) returned
LIMIT 5 5 row(s) returned
h, estatus,... 14 row(s) returned
h, estatus,... Error Code: 1054. Unknown column 'post_quantity' in 'where clause'
h, estatus,... Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'HAVING post_qu...
```

25. NESTED QUERIES

Se usa cuando no se puede solucionar simplemente consultando la tabla propia sino que se necesita agregar u otra operación de otra tabla o de la misma tabla pero que cuando se agrupa no se puede hacer un query de un where o con otra condición, entonces en las condiciones que están separadas se hace es hacer primero un query en un tabla o valor dependiendo se necesitó, luego eso se mete como entrada por from o where dentro del segundo query, es decir que alimenta al segundo query.

Los nested query hay que tener cuidado porque se puede convertir en un producto cartesiano es decir se multiplican los datos de las tablas y al crecer la cantidad de datas entonces se vuelve pesado el query. Se vuelve query ya que cada vez que aumente la cantidad de operación en el query padre se hace lo mismo en el segundo entonces se hace muy pesado.

Es mejor hacerlo en consultas que sabemos que no van a ser escalables.

- Las consultas anidadas son la mejor opción cuando los valores dependen de otras tablas, y estas no se encuentran relacionadas entre sí.
- Las consultas anidadas son la mejor opción para casos de INSERT, DELETE, UPDATE, cuya condición dependa del escenario explicado en el punto anterior
- Los JOINS son la mejor opción para casos de SELECT

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.

Ejemplos:

```
SELECT new_table_projection.date, COUNT(*) AS post_count
FROM (
  SELECT DATE(MIN(fecha_publicacion)) AS date, YEAR(fecha_publicacion) AS
  post_year
  FROM posts
  GROUP BY post_year
) AS new_table_projection
GROUP BY new_table_projection.date
ORDER BY new_table_projection.date;
```

```
.  
SELECT *  
FROM posts  
WHERE fecha_publicacion = (  
SELECT MAX(fecha_publicacion)  
FROM posts  
);
```

26. ¿Como convertir una pregunta en un query en 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

27. PREGUNTANDOLE A LA BASE DE DATOS

- I. **Group_concat:** sirve para crear una columna separada con los datos separados por una coma

■ Ejercicio práctico 1:

```
SELECT customer_city, group_concat(review_score)
```

```
from customers
```

```
left join orders_status ON customers.id = orders_status.customer_id
```

```
left join order_reviews ON orders_status.order_id = order_reviews.order_id
```

```
group by customer_city;
```

Query 1

```

1 SELECT customer_city, group_concat(review_score)
2 FROM customers
3   LEFT JOIN orders_status ON customers.id = orders_status.customer_id
4   LEFT JOIN order_reviews ON orders_status.order_id = order_reviews.order_id
5 GROUP BY customer_city

```

Limit to 1000 rows

Result Grid

customer_city	group_concat(review_score)
boninal	3
botucatu	5
braganca paulista	2023
brasil	5,5,5,4,5,5,5,4,3,1,4
caçador	5
cachoeiro de itapem...	4
camanducaia	5
camaragibe	1
campinas	3,5,4,4
campo grande	4,5
campos dos goytaca...	5
capelinha	1
caracaruí	4,4,4,5

28. Atajo en SQL

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

Ejemplo:

```
- FROM categorias AS c
```

29. ¿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.

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.

30. 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

Colección

Documento

31. top level collection (FIREBASE/FIRESTORE)

Las top level collections o colecciones de nivel más alto son las colecciones a las que vamos a tener de inmediato en nuestro proyecto

Las top level collections son las colecciones que se tienen de inmediato o entrada en el proyecto.

Las bases de datos no relacionales, no están optimizadas para hacer “queries”, sino mantener el estado de la aplicación por ende si se desean luego hacer reportes en la aplicación de diferentes datos, no es recomendable usar BD NoSQL

no todas las bases de datos no relaciones no están optimizadas para hacer “queries”, hay algunas que sí, como “Big Query”, sino que las basadas por documento, se enfocan más en mantener los datos vivos (Son muy rapidas, activas) , entonces para estas si aplica eso. Y recuerda que las NoRDB no son un conjunto unico como generalmente las RDB. Las No Relational Data Bases son de diferente tipo.

32. contruyendo platziblo en firestore

Las top level collections son las colecciones más altas que dentro de ellas contienen otras colecciones menores. Esto se define en la planeación de la base de datos y de como vamos a usarla o consultarla.

Por ejemplo en platziblog las etiquetas no será algo que se querrá consultar aparte o ajeno a los posts por lo cual no son top level colecciones sino que irán dentro de la toplevel collection de posts.

Es importante aclarar que en este caso las BD no relacionales en general no están hechas para hacer Queries. Sino para diferentes tipos de usos. En el caso de Firebase, está orientada a aplicaciones o servicios web para que se consulte el paquete de datos.

33. construyendo platziblog en firebase/firestore

Resumen

Emplear colecciones de nivel cero para entidades fuertes, que mas adelante se empleen para realizar búsquedas exhaustivas.

Emplear subcolecciones para entidades débiles, que solo permitan plasmar información relacionada, pero que dicha data no sea una fuente de consulta primaria para nuestra aplicación.

34. Proyecto final: 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 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.

Juan José Calderón Estudiante · hace 25 días

Les comparto mis apuntes de la parte de Bases de datos No Relacionales. Espero les sirva 😊❤️

Basadas en Documentos(Firestore, MongoDB)

Colecciones y documentos

Donde

Colecciones -> Tablas

Documentos -> Registros

- **Top level collections:** Que están en la raíz de la base de datos
- **Subcollections:** Que están y tiene como padre un documento

The screenshot illustrates the Firebase console interface. On the left sidebar, the 'posts' collection is highlighted with a red arrow and labeled 'TOP LEVEL COLLECTION'. The main panel shows the 'posts' collection with a document selected. This document has a 'SUBCOLLECTION' named 'etiquetas', indicated by a red arrow. The document's fields include 'autor', 'categoria', and 'contenido'.

35. BASES DE DATOS EN LA VIDA REAL

Hoy en día, se utilizan diversos tipos de bases de datos según 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 número de datos a manejar, se volvieron ineficientes en muchos casos.

Firestore o MongoDB nos permiten obtener los datos actuales de la aplicación de manera simple. Sin embargo, no nos permite hacer, por ejemplo, queries muy complejos.

En una misma disciplina, es probable que haya que utilizar más de un tipo de bases de datos.

36. 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.

En otras palabras es guardar muchos datos por segundo o por milisegundo para ser consultados de entrada o en un instante (ejemplo una aplicación)

LINK

o <https://noticias.universia.es/ciencia-tecnologia/noticia/2017/09/12/1155659/machine-learning-como-usa-big-data.html>

otros apuntes

Grandes cantidades de datos, el reto que empezó a cerrar las puertas fueron que se estuvieron manejando cada vez más grandes, grandes volúmenes de datos. Se refiere a que en este momento en milisegundos vamos a guardar grandes cantidades de datos. Es un gran movimiento que surgió con Youtube y Facebook ya que necesitaban guardar muchos datos rápido.

37. DATA WAREHOUSE

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.

A diferencia de Big Data, que es guardar muchos datos por segundo o por milisegundo para ser consultados de entrada o en un instante (ejemplo una aplicación), en Data Warehouse se guardan aún una cantidad mayor de manera histórica.

Se manejan usualmente diferentes bases de datos en la ejecución de una aplicación, por ejemplo durante la ejecución se manejan bases de datos orientado a documentos como firebase para tener esa agilidad y accesibilidad a los datos demandados, sin embargo, luego de que muchos de esos datos vayan dejando de ser relevantes se van almacenando en un data warehouse para posterior manejo. Ya que en los data warehouse tienen la capacidad de guardar inmensas cantidades de datos ☞ Big table.

Los principales beneficios de un data warehouse son:

- I. Permite guardar cantidades inmensas de datos históricos.

II. Permite consultar esos datos para varias finalidades que pueda presentar una compañía.

Soluciones para data warehouse

- o BIG QUERY.

LINKS

- o <https://www.datamation.com/cloud-computing/top-cloud-data-warehouses.html>

38. DATA MINING

Es una disciplina que se encarga a minar datos, esto quiere decir que cuando existen grandes cantidades de datos en diferentes lugares puede ser (data warehouse) es extraer los datos de diferentes bases de datos, tratarlos, y darles sentido (puede ser guardarlos nuevamente para darle sentido). En otras palabras, ordenar los datos.

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.

EJEMPLOS:

- o KNIME

LINK

- o <https://financesonline.com/top-15-data-mining-software-systems/>

39. ETL

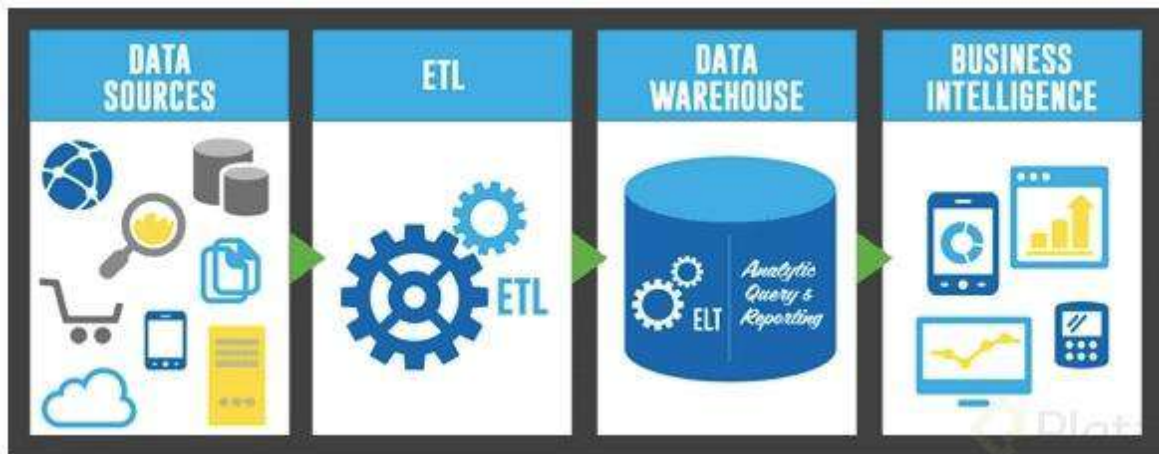
ETL son las siglas de Extract, Transform, Load (extraer, transformar y cargar). Es una técnica basada en una idea que se trata de tomar datos de archivos muertos y convertirlos en algo que sea de utilidad para el negocio.

También ayuda a tomar los datos vivos de la aplicación, transformarlos y guardarlos en un data warehouse periódicamente.

ETL systems extract data from one system, transform the data and load the data into a database or data warehouse.

LINKS

- o <https://www.alooma.com/blog/what-is-a-data-pipeline>

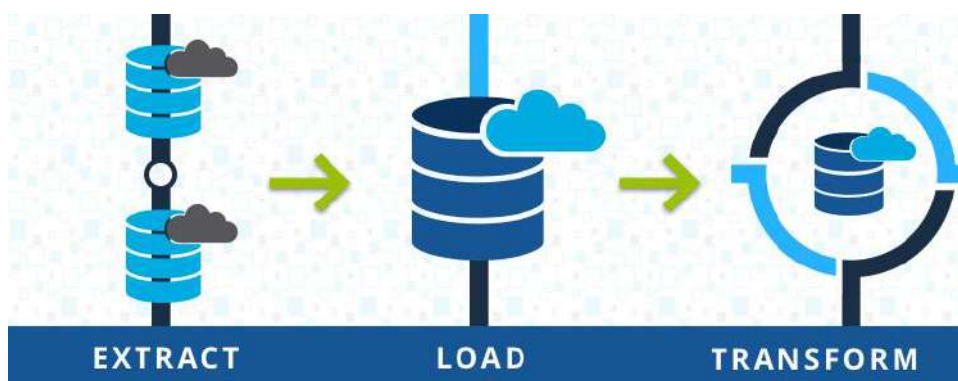


Las ETL se usan en el ambito del DataWarehouse, sin embargo con el desarrollo del cloud nace el DataLake que usa las ELT, entonces, la diferencia entre ETL y ELT depende de dónde se transforman los datos y cuántos datos se retienen en almacenes de datos operativos.

- ETL



- ELT



40. 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.

Business Intelligence es la habilidad para transformar los datos en información, y la información en conocimiento, de forma que se pueda optimizar el proceso de toma de decisiones en los negocios.

LINKS

- https://www.sinnexus.com/business_intelligence/
- Datastudio

41. Machine Learning

Machine learning son una serie de técnicas que involucran la inteligencia artificial y la identificaciones de patrones en diferentes ámbitos.

Es un paso más allá al business intelligence.

Machine learning en el campo de los datos se diferencia de business intelligence debido a que en vez de tener a alguien buscándole sentidos a los datos, se hacen modelos tratando de ir por un patrón. Intenta encontrar correlaciones que normalmente una persona no se espera.

EN RESUMEN:

IA: abarca a machine learning y deep learning y busca eliminar al humano en la creación de algoritmos y que todo ese proceso lo haga una computadora

Machine learning: como dijo Israel, crea modelo y al señalarle te encontrada modelos fortuitos, cuando le des una mar de datos él te va encontrar patrones

Deep learning: le busca sentido a los datos.

Una IA es un sistema que es inteligente, usando machine learning puedes buscar en una BBDD cuando tus clientes te compran más y te das cuenta que es en finales de años y usando deep learning la computadora es capaz de encontrar el porqué te compran más.

Apuntes extra

Son una serie de tecnicas que involucran la inteligencia artificial y deteccion de patrones. A diferencia de Business Intelligence que queremos buscar un patron en especifico con la informacion que ya tenemos, mientras que con Machine Learning dado un conjunto de datos buscamos encontrar patrones que no esperas o no eran obvios para un ser humano, ademas que busca saber en un futuro como se comportarian nuestros usuarios. Una bd muy buena para esto es BigQuery que nos sirve como warehouse y ademas nos permite hacer queries que nos podrian ayudar con BusinessIntelligence como son Data Studio. Entre los dos casos de uso principales de ML son: clasificacion y prediccion.

Con la clasificacion nos referimos a que si tenemos varios datos historicos y queremos conocer por ejemplo cuales temas fueron mas interesantes durante un periodo en especifico, no serviria con buscar palabras claves, para este tipo de casos se utiliza la tecnica de ML denominada Procesamiento de Lenguaje Natural lo que hace es tomar un texto o reconocimiento de voz que utiliza el lenguaje natural humano nada estructurado, lo empieza a procesar y nos devuelve los patrones que encontro. Por ejemplo con el Platziblog: Si tenemos un modelo muy bien entrenado, es decir, que le hayamos pasado una buena cantidad de articulos que si sabemos que son de politica y reconoce estos patrones. Cuando le pasamos toda la informacion de Platziblog nos devuelve articulos que no especificaban por ninguna parte que se trataban de politica pero gracias al modelo pudimos identificarlos.

Con la prediccion nos ayuda a la toma de decisiones. Por ejemplo le damos las ventas de unos años pasados, procesa las ventas y genera un modelo, que cuando le pasemos las ventas actuales nos ayuda a identificar las relaciones y patrones que nos ayuda a visualizar a donde nos dirigimos, es decir, si le pasamos ventas proyectadas a futuro este nos dice que va a pasar, si se va a comportar igual, etc.

42. DATA SCIENCE

BigQuery es una buena herramienta debido a que con un lenguaje similar a SQL pero con mas funcionalidades nos permite ademas de hacer consultas, almacenarla y alimentarla a un algoritmo de modelos de ML que con el mismo lenguaje SQL al ejecutarlo estamos creando modelos que nos ayudaran a predecir valores futuros.

Data Science: Es aplicar todo lo que hemos visto, técnicas de ETL, Data Mining, Business Intelligence. Aunque esta mas dirigida a personas con background de estadísticas, hoy en día también participan personas con el perfil de Data Engineering. Al hacer Data Science estamos aplicando todo lo visto en los temas anteriores, no solo a nivel técnico sino que desarrollamos la experiencia y conocimientos sobre las distintas tecnologías, en que parte del desarrollo de nuestro proyecto tenemos que utilizarlas, etc. Con lo cual el Data Science juega el papel de manager dentro de un equipo de profesionales de datos.

