Links útiles merlino

https://docs.google.com/document/d/1R9oMFvSN-loHFqsN07HtWhusbGkYo2B1GZTHlMdxg1g/edit

Links para jugar sqlite-merlino:

https://library.lol/main/3BE3FB67C556A0E971054620F98BCBB4

https://link.springer.com/book/10.1007/978-1-4842-9493-2

https://github.com/Apress/getting-started-sql-databases#sample-database

Clase 1

SQL (Structured query language) Lenguaje de consulta más estandarizado)
SQL no es parte de la base de datos.
Es un **modulo** que se **adjunta a la base de datos** comodo y sirve. La base de datos tiene una estructura que no necesita trabajar con SQL, para consultar los datos.
El lenguaje SQL hace la consulta a la base de datos?? ->no! -> Parcial
No, **traduce la consulta** al **lenguaje nativo** que tiene **la base de datos.**
Consulta basada en lógica de primer orden o segundo orden.
Primer inconveniente.
Cada base de datos a la cual ustedes acceden tenían el problema de aprender el lenguaje de consulta nativo.
Con **SQL queda arreglado** el **tema de los consultas**.
99.9% veces trabajamos el lenguaje de consulta ANSI 92.
Analogía con excel:
Excel es una base de datos, cada planilla es una tabla.
Cuando hagan cualquier tipo de aplicación. Todo lo que se te ocurre con una base de datos. **PLSQL y TransactSQL** son casi lo mismo.
Las pocas diferencias que existen las busco en el momento.
Dentro de SQL:
Cálculo de costo de las consultas:

1) **Cálculo teórico:**
2) Cada base de datos tiene un análisis de la consulta.
3) **Full scan:** Recorres toda la tabla**->** Consulta de base de datos y no te diste cuenta los indice de la base de datos que necesitas . Se hace cuando no conoces los índices. Cuidado porque muchas consultas tardan mucho tiempo por no tener índices.

Había una **consulta que tardaba 48 horas** que unifica toda la información y lo veían los directores de esta empresa multinacional.
Merlino -> le dije que había un Full Scan al administrador y a regañadientes creó el índice. La ejecutamos a mitad de mes y tardó 12 minutos y medio.

1) **Experimentando en consultas**
2) **Saber de índices**
3) **Desarrolladores** -> **solo SQL. Sí quiero**
4) Cómo **acceder de manera programática a la base de datos** grabada. **Hacer todo el ida y vuelta.**

Cosa mas costosa de la **base de datos -> Proceso de conexión.** (Pull de dconexión). Cuando alguien se quiere conectar se empiezan a levantar conexiones.
Un pull de 10 conexiones puede dar abasto hasta 1000 usuarios. No tienen que estar el programa constantemente en la base de datos.
Pool de conexiones a la base de datos es costosa, debido a que cada conexión abierta a una base de datos **consume recursos del servidor de base de datos**, **requiere memoria y tiempo** del **procesador por cada nueva conexión**.

La forma de acceder programáticamente a la base de datos. **Se conecta, consulta y se va.**
La arquitectura más común de las aplicaciones pesadas.
Server de colas
Entre el **backend** y la **base de datos** no se hace una conexión directa. En el medio se pone una aplicación de **cola de mensajería**.
Cuando ustedes se conectan.
Te dan un nombre de la cola donde vamos a preguntar el resultado de la consulta.
La primera cola **recibe la consulta** y genera una nueva cola con el resultado de la consulta.
Backend se comunica de manera asincrónica con un servidor de colas.
Y Un servidor de colas se maneja de manera asincrónica con una base de datos.
Base de datos todo es de manera asíncrona.
Trabajar con una base de datos es una parte de la base de datos.
Formas asincrónicas
Consulta de datos a través de una cola de mensajería.

Adelante del frontend tienen un equipo BIP, tiene una IP .
Detrás del BIP tiene una granja de servidores.
Ustedes puedan tener una granja de servidores en cual pueden ir creciendo con sus clientes.

Toda la arquitectura se fue desarrollando en función de los limitantes de las primeras base de datos.

No hay nada desconectado. Todo está relacionado.
Base de datos relacionales: La palabra más apropiada porque tienen relaciones entre sus tablas (y cada una de estas con sus filas y columnas).
¿Forma en que se guardan los datos en una base de datos? -> **arbol B^+.**
Enfoquémonos en el índice primario.
Indice: Estructura de datos (Árbol B^+ o un tabla de hash) que agilizan la búsqueda de registros.
Índice primario: Físicamente están almacenados los datos en este orden. (El índice coincide con la primary key y además la tabla está ordenada por este índice).
Solo se puede tener un indice primario (o puede ser de clustering tambien).

Más eficiente **indice primario** , o los **índices secundarios** (Tabla en memoria donde esta el campo que representa el índice).
Diferencia entre Indice primario e Indice secundaria: es un acceso más.
Clave secundaria va ir pegando saltos hasta encontrar el índice.

id	nombre	apple	dire	cod.postal
1				
2				

Apellidos ORD ASC -> va del **id =2 al id= 100 y** luego al id = 300 y así.
Indice se usan para que sean eficientes las consultas.
Aplicación performance se hace con la **suma de pequeños pasos** que **van ordenando la aplicación** de **un extremo a otro.**
Con lo poco recursos que necesitan estos servidores para dar una respuesta eficiente a un cliente de homebanking.

Base de datos son archivos inmensos.
Como se desarrolla una aplicación:
1) Aplicación de **consumo masivo de base de datos** separemos **el diseño**.
2) Van a generar un mockup.

Consulta SQL guardada en un archivo para que?
Porque es un store procedure, esto es más rápido por los compiladores (se ahorra el análisis léxico, directamente lo transforma a código máquina.

Haces una consulta SQL, el administrador de base de datos lo pasa por validaciones y te va decir **vos** para hacer esa **consulta invoca el store procedure con estos parámetros**.
Stored Procedure -> técnica extremadamente usada en la vida real.
Es un programa (o **procedimiento**) **almacenado físicamente** en una **base de datos. La ventaja de un procedimiento almacenado** es que al ser ejecutado, en

respuesta a una petición de usuario, es ejecutado directamente en el motor de bases de datos, el cual usualmente corre en un servidor separado.

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City GO;
```

NoSQL: orientada a documentos, a grafos , wide-column.
Arranco el problema detectando la necesidad de acceso a esos datos.

Principal diferencia entre las bases de datos relacionales y el fierro que uno le ponga (el equipo sobre el cual uno lo ejecute).

DataWarehouse: Un dataWarehouse es un repositorio de datos donde se almacenan datos sumarizados, probablemente el origen sea una base relacional puede ser no relacional, se usa para el mercado o la industria.
Five tuning de base de datos.
Tuning de base datos:
Proceso que realizan los **administradores de bases de datos** para **optimizar el rendimiento** de una **base de datos**, incluye tanto **optimizar la base de datos** como el **hardware**.

Clase 2-Conceptos básicos

Usamos un Wrapper de un lenguaje de programación (sqlAlchemy, sql1, sql2, etc) para que nos conectemos a una base de datos.
ORM: Librería que permite manipular las tablas de una base de datos, como si fueran objetos.

Big data: aplica la **Arquitectura lambda**.
Arquitectura con Pipeline de ejecución lo que hace es insertar datos.

Programamos un pipeline para hacer un procesamiento de datos.
-- Crear y administrar un Datalane. Cualquier parte de la empresa tanto sea cientifico de datos.

Bases de datos transaccionales Son bases de datos que tiene como **fin el envío y recepción de datos a gran velocidad.**

Forma de trabajar con datos en transaccional.
Si un sistema transaccional tiene mucha lectura y poca escritura o uno que tiene poca escritura y lee mucho ?.
Genera arquitectura de base de datos totalmente distinta.

Cubos algo propio datawarehouse. ¿Qué es un dataWarehouse?
Es una estructura en general Toma datos de un sistema transaccional. Y se lo guarda en una estructura interna propia de motores (datawarehouse).
Cubos: manera **figurada** para decir cómo está **almacenada esa información** en el datawarehouse.

RPA: es una manera de automatizar tareas administrativas ejecutadas por humanos, por medio del uso de robots de software. Es usado en tareas como extraer datos, completar formularios, mover archivos, entre otros.

Tareas en la noche. **RPA (se toman los datos del transaccional)** y se inserta en los cubos. Los cubos pueden ser consultado en tiempo real que en general son aplicaciones de comando. UX: parte de analitica. Tablero de comando, toman datos de 3 áreas.
Pipeline: Secuencia de pasos pre-establecidos para generar una tarea de forma automática.
RPA: mejor RPA es programar en python. Debemos conocer las herramientas automáticas.
80% problemas con herramientas de drag and drop.

20% ponerse a trabajar de una manera mas precisa.
Web scrapping : proceso de extraer contenido y datos de un sitio web mediante software.

“Y conectarse a una información y extraer información de una NFTP.”
“Mover datos con un ambiente distribuido. Optimizar el procesamiento BATCH. “
Procesamiento BATCH: Proceso en el cual un ordenador realiza procesos, muchas veces de forma simultánea de forma continuada y secuencial.

Temas que entran en base de datos:

1)SQL
2)Formas normales
3)Acceso pragmático.
4)Tipos de BDD

El que no sabe estos 4 temas nunca aprobará.
Normalización y formas normal y Desnormalización.
“Siempre se desnormaliza algo”. -Merlino.

Mínimo apropiado para trabajar. la 3.5 (Boicot) sirve para evitarnos la búsqueda de duplicación.
Entender:

1. SQL: En cualquier examen (FULL stack -> no te escapas del examen SQL **más sepas mejor** es).
2. **Normalización/ Desnormalización.**
3. Cómo acceder pragmáticamente a una base de datos y cómo crear una BD.
4. Estructura, seguridad, recupero de datos.

Usamos las formas normales para:

1. **Minimizar redundancias.**
2. **Garantizar la integridad de datos.**
3. **Evitar anomalías de ABM.**
4. Interactuar con el administrador de la BD.

La integridad en el contexto de las bases de datos se refiere a la precisión, consistencia y exactitud de los datos almacenados.

Clase 3) Normalizacion de Base de datos

Motivos por el cual normalizamos:

1) **Minimizar la redundancia de los datos.**
    a) **2 Datos en un solo lugar**, **cuando hay datos repetidos** me tengo que recordar donde están.
2) **Evitar anomalias de ABM (Disminuir problemas al actualizar)**.
3) **Garantizar la integridad de los datos.**

"**Normalizamos para no tener quilombos con los datos**" -Merlino

Integridad de datos:

1) **Integridad Referencia.** Un valor de un campo FK de una tabla debe existir como valor en la PK de la tabla que hace referencia. **Definicion de FK.**
2) **Integridad de Entidad:** Asegura que cada fila se identifica de forma univoca. **Definicion de PK.**
3) **Integralidad de dominio**: Un campo debe tener un conjunto de valor homogeneos. (pertenecientes a un mismo dominio)
4) **Integridad de usuario:** Mantener la integridad de los datos de acuerdo con las reglas y especificaciones especificas definidas por el usuario o lógica de negocio.

(Web server = conjunto de páginas html).

En un primer momento la BD nos da toda la **concurrencia** hecha.
¿Es igual la normalización en todas las bases de datos ? -> **Ni**

Formas Normales:

No se puede llegar a la 3FN sin pasar por la 2FN ni por la 1FN.
En la industria piden Boyce-codd. (**3.5FN**).
4FN y 5FN.
Con 3 ok, todo nos plantamos en Boycee-codd (3.5N) mas no.

Comentario empresarial:

Algunas funciones de un administrador de base de datos :

1) Performance de la base de datos.
2) Licencia de la base de datos.
3) Encarga los permisos para acceder a la base de datos.
4) Ver que las tablas están normalizadas.

Primera forma Normal (1FN): RICARDO LUIZAGA.

1) **Todos los atributos son indivisibles**.

Calle

Av 25 de mayor 2100 CP: 1406

Direccion	**Altura**	**CP**
Av 25 de mayor	2100	1406

La direccion para que este bien debe tener calle, Nro y CP. La calle 25 de mayo, el 25 es altura o calle? Aplicación 1 a *.

Lo primero que debemos ver una aplicación?:

Ver la base de datos: las tablas y el diagrama de entidad-relacion (**DER**).

1) **DER**
2) **UX**

Si no está normalizado encomiendate a dios.

Clase 4 (Acceso programático a la DB):

sqlAlchemy: interfaz que se utiliza desde un lenguaje de programación para conectarse a una base de datos un ORM o mapeador.
Libreria de conexión para conectarse a cualquier base de datos. (SQL server, ORACLE, postGree,) y todas estas bases de datos es
Kaggle: Competencia de ciencia de datos.

Ofuscar datos: No podría poner los datos de cliente de un banco en un lugar como kaggle. Ofuscar datos es correr en un proceso y cuando alguien ve los datos, no tiene forma de saber a quién pertenece. **Lo que se mantiene son los patrone**s.
Procesos para encapsular o encriptar información sensible, manteniendo los patrones.

Av. Direccion	Saldo cuenta Banco	…
Alejandro knw mz 23 casa 200	140k	…

Av. Direccion	Saldo cuenta Banco	
av 123	100100	123

Leyes Habeas-Data:
Te protege si tus datos de identidad, de salud o de crédito son usados sin tu consentimiento.
Leyes Habeas-Data: dato del banco- AFIP tienen que estar suscrito por ley al concepto de avias data.

1) Creamos el engine (**el motor de base de datos**)<-/// -> nombre de la BD a conectar (en disco rigido).
    a) Si no pongo nada del lado derecho, estoy creando una base de datos en memoria
    b) **Base de datos en memoria**, es el modelo de base de datos **mas rápido en que podemos trabajar**, toma los datos y los muestra.
2) Dame la metadata
3) Decime las key de la tabla division: divisions.column.keys();

Wrappers de sqlAlquemy:
divisions.select()
Te escupe una *query* y prepara, ya sabe como escribir el select.
A esa query la ejecutó cuando le aplico un execute(). y me escupe un execute.
A execute le aplicó **fetchmany(5)** -> me obtiene los 5 primeros registros de mi tabla.

Tres pasos para laburar:

1) **Preparo la consulta (Select) ->** No consume tiempo (en el backend- el servidor lo único que hace es construir la ejecución)
2) **Ejecuto la consulta (execute)**
3) **Extraigo los resultados. (fetchamny)**

Si pedis asi el select te trae millones de registros, y el execute() te va demorar lo equivalente a 1 millón de registros.

Nunca traer todo de registros de una base de datos. No se puede hacer consultas sin sentido a una base de datos.

Consulta puntualmente siempre.

Tenemos un **web service, que tiene el negocio** (lo que hace el sistema), atras tenemos (la base de datos), como me conecto.

1) Usare una libreria wrapper de bd.
2) Generar consultas para traer la mínima cantidad de registros.
3) y luego **inmediatamente cerrar la conexión**.

menor tiempo de rspt de la consulta, menor uso de memoria,

La base de datos tiene pulls de conexión de base de datos. Abrir y cerrar la conexión.

Lo básico es una conexión de base de datos. Saber que nos conectamos, se puede saltar la estructura de tabla, conéctate usando el puntero a la conexión,

Dentro de una base de datos voy a poder obtener una o muchas tablas.

Puedo tener una base de datos con 0 tablas, aun no genero tablas.

Analogía: una base de datos tiene muchas tablas, un archivo de excel y muchas planillas.

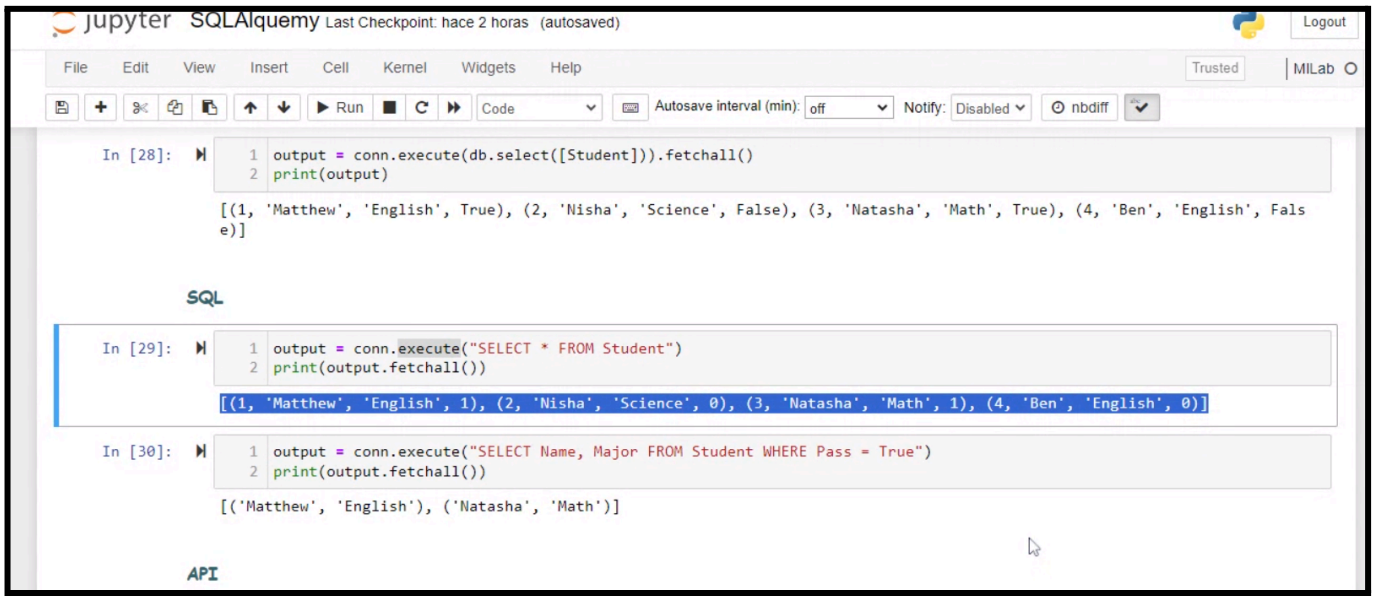
Query: puntero a la consulta.

exe: puntero a la ejecución.

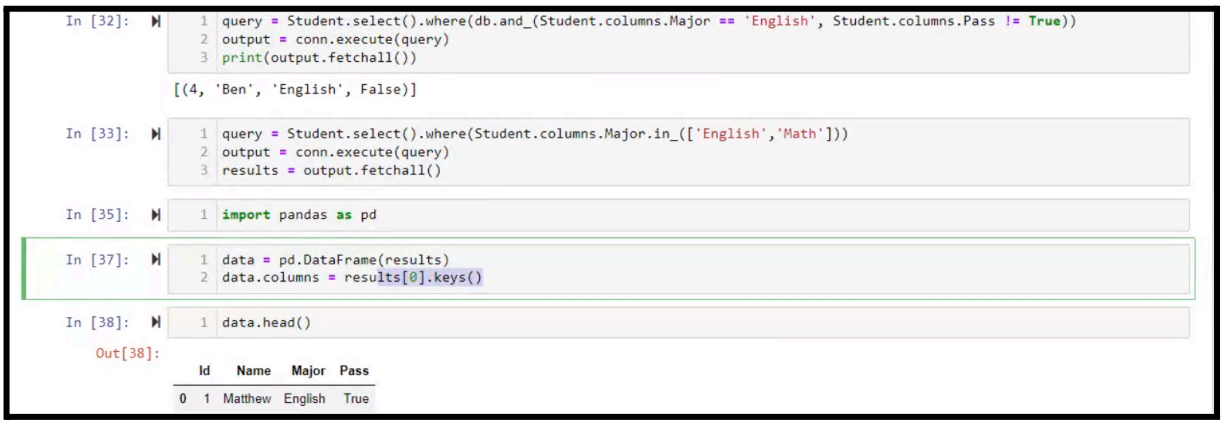
No importa si la tabla no existe. el **create_engine, conect y metaData** se hace igual.

db: puntero a la base de datos

metadata:



Where corta horizontalmente los registros. (**corte por filas**)



Dentro del lenguaje de programación le pegamos a la **Base de datos, SQL capa intermedia para simplificar la forma de hacer consultas** dentro de una base de datos. Tengo **que saber SQL, tengo que entender la normalización** para extraer **una consulta de SQL,** para juntar varias tablas.

Nocion de calculo de esfuerzo de una consulta saberlo. Entra en el parcial.

Si la aplicación es lenta. => culpamos primero a los programadores,el 80, 90% del tiempo promedio.

DER (Diagrama entidad-relacion, claves principales, claves primarias, y como se relacionan las tablas en definitiva).

Consume mucho tiempo => dejame ver el DER.

Mira el problema es que existe una tabla que no tiene una clave foránea de otra tabla. El administrador de base de datos va y crea el nuevo índice.

Clase 5) Lockeo de base de datos

Normalización: Aplicar una serie de reglas a nuestras tablas para:

1. **Eliminar redundancias:** Evitar repetir la misma información.
2. **Evitar Anomalías de ABM: actualizar** u**na fila no bast**a se **debe actualizar en todas las filas asociadas.**
3. **Garantizar la integridad de los datos**

Desnormalizar:

Se aplica para aplicaciones con un alta performance. (se hace una desnormalización de base de datos). Pongo en una tabla todos los datos que quiero grabar.

Lockeos

Hay 3 tipos de lockeo:

1) **Lockeo a nivel campo:** Primer nivel de lockeo me lockea a nivel camo
2) **Lockeo a nivel registro:** Segundo nivel de lockeo me lockea a nivel registro (toda una fila)
3) **Lockeo a nivel tabla:** Lockea toda la tabla. Lockea todos los campos

Base de datos: En el tiempo de modificar o grabar datos hace lockeos.

El motor de base de datos :

1. lockeo el registro.
2. modificó el dato.
3. deslockeo el registro.

Hace esos 3 pasos para modificar un registro

Cuando estaba cambiando el registro puede haber otra lectura y lee un dato invalido(null), por este motivo hacemos el lock, es la forma en que los motores de base de datos pueden resolver este problema.

El administrador de BD decide cómo se debe lockear los registros de BD.

Cuando la BD de datos detecta que estoy modificando varios registros en una tabla conviene lock por tabla.

Por cada operación de escritura y modificación, ¿cuantas operaciones hace la base de datos ? **3** :

-> Lockeo, modifico y deslockeo

Ejemplos:
Mínimo lockeo a nivel de campo, si tengo una tabla con:

idUsuario	nombre	apellido	telefono

y **si solo modifico el telefono** entonces puede ser que la **mejor opción** sea l**ockear el campo** telefono porque es el campo que voy a modificar.

Lock por registro: me lockea toda una fila el idUsuario, nombre, apellido y telefono, aunque cambie solo el telefono.

Si tengo que modificar 100k registros cuantas operaciones debo hacer ? En realidad hago 300k operaciones. (3 transacciones por cada modificacion). (usando lock por registro)
La base de datos piensa y deduce que es mejor usar lock a nivel de tabla.

1. Lock
2. 100k Modificaciones
3. Unlock.

Obtuvimos 100002 operaciones
El tiempo de operación es mucho menor y puedo seguir trabajando con base de datos.

Concepto de lockeo, para que sirven, que situaciones lo usa la bd
El codigo no es lento, lo que es lento es que el administrador de base de datos no definio bien el nivel de lockeo
Entender parcial -> lockeo de registro de base de datos.
1)A nivel de campo
2) A nivel de registro
3) A nivel de tabla.
El motor de base de datos decide cual es **mejor en base** a las modificaciones que se hacen en la tabla en tiempo real.

Ej1: Si queremos modificar la direccion y el telefono de un mismo registro usando **lockeando a nivel campo:**
Lockea el campo direccion
modifico direccion
unlock.
lockea el campo telefono
modifico telfono
unlock.

Tengo 6 transacciones por modificar dos campos, usando lock de campo

Si lockeo a nivel de registro: me ahorro dos transacciones
lockeo
modifico campo direccion
modifico campo telefono
unlock.
4 transacciones, me ahorre dos operaciones.
Esas 2 operaciones menos se justifican para hacer lockeo a nivel de registro y no de campos.

Lockeo ideal en una base de datos => **Lock a nivel de Campo**, solo retenemos ese campo.
Por lo general => Lock a nivel de registro
Actualizaciones masivas de datos => Lock a nivel de tabla.
Muchas veces las **aplicaciones** se vuelven **lentas** porque se esta **trabajando con un lockeo incorrecto.**

Desnormalización de base de datos:

Nuestro ideal es trabajar con la boicot 3.5.
Cierta parte puntual de la base de datos de algunas tablas se desnormaliza.
En lugar de acceder a dos tablas cuando hacemos modificaciones le pegamos y actualizamos ambas tablas.
Generamos una nueva tabla donde esta todo conjunto, accedemos solo una tabla, le pegamos a ella y la modificamos.
Desnormalizamos esta **mal si** , pero ganamos en tiempo de performance.

Concepto de equilibrar.

La forma normal se puede toca pero poniendo un equilibrio.
Si queremos que el **acceso a base de datos** sea **hiper performante**:

1. Esta **pequeño parte de la BD** la **desnormalizamos** a propósito para **acceder/ modificar datos** mucho más **rápido**.
2. Académicamente correcto es llevarlo a forma normal 3 y 3.5 (boicot).

La ingeniería de software hace que **seamos pragmáticos.**
Si tenemos una infraestructura no podemos comprar un nuevo equipo, con lo que tengo que dar respuesta a muchos usuarios, puede decidir que una parte de mi base de datos la desnormalizar, justificado por el peso de la cuestión que la aplicación debe ser performante.

Desnormalización de base de datos: es un tema que se aplica en la vida real cuando debo hacer aplicaciones muy performantes.

En la vida real tendremos que ver:

1) Grado de normalización, que tengo que desnormalizar.
2) Grado de lockeo que me generan las consultas, (consulta de lectura no bloquean)
3) Los update, los inserts que me hacen poca performante y ahí me van a pegar.

Tablas en memoria: Cuestiones de performance:

Para un motor de base de datos una tabla en memoria es igual a una tabla en disco
La tabla en memoria es muy eficiente, pero hay un limitante la cantidad de memoria.
Toda aplicación que alguna vez desarrollemos tiene una tabla en memoria:

Ejemplo:

1) Hacemos un aplicación web y tiene un logueo
2) Si queremos tener un grado mayor de detalle de la seccion de usuario (logeo) algo muy común en la web. Manejamos la seccion a nivel pragmático.
3) Una de las **opciones** que tiene el **framework** es poder manejar en memoria del backend (servidor del proceso) o hacer persistente esa seccion.

Persistente un dato, quiere decir que lo voy a guardar el dato en un base de datos.
Genero una tabla con los campos del framework con el que trabajo y lo guardo en mi base de datos
Le digo al admin BD Voy a tener en memoria esta tabla.
Es configurable las tablas en memoria se pueden cargar cuando:

El primer usuario accede		**Cundo se levanta el motor de la base de datos**	
Ventajas	Desventajas	Ventajas	Desventajas
Si entra un usuario 10 horas después de que levantamos el motor de BD no ocupamos espacio sin sentido	Primer usuario sufre una demora, porque la base de datos tiene que hacer dos operaciones levanta y actualiza	Primer usuario que acceda no tiene problemas, mismo tiempo que cualquier otro usuario	Los recursos del servidor no son usados de manera óptima, hay una tabla que solo sabe quién cuándo será usada.

Bases de datos en memoria:

Base de datos en memoria, levantó un dato y queda ese dato en memoria y cuando se baja el servidor se pierde la información. -> Esto se puede configurar.
Cuando se baja el motor de la base de datos se serializa (grabe) esa tabla, puedo configurar esa tabla se baje periódicamente (cada 3 o 4 horas- momento que no tiene mucho trabajo la BD).

Este es el **trabajo de un administrador de B.D**, para manejar y optimizar las B.D.

Un punto mas de performance:
Cuando tengo una aplicación que debe ser hiper performante y/o una aplicación que tiene clase de uso masivo de las bases de datos.
Se suele poner (**en una buena arquitectura de sistemas**) delante de la base de datos se pone un **queue Server (Servidor de colas de mensajeria).**

¿Que es un queue Server?

Es la mejor manera de hacer una **aplicacion sincronica en asincronica** sin que nadie se de cuenta.

Ejemplo de acceder al homebanking:
Actualiza el numero de telefono a la pagina de home baking.
Vamos al browser ponemos la url, la url fue buscada en los DNS encontro la ip la cambio y la fue a buscar, salto un montón de routers y llega al servidor.

Servidor reconoce que debe servir informacion y nos retorna la pagina web, como es el protocolo TCP/IP, parte en paquetes (512 bytes) y lo manda para su browser.

Browser arma la secuencia correcta, extraigo la información que tienen adentro los paquetes (que es html + js) se lo pasa al browser, lo grafica y nos muestra la informacion.

Pregunta de ingeniería de software: el procedimiento anterior es sincrónico o asincrónico?:
Es asincrónico, **Desarollo web es asincrónico.**
Canal seguro (tubito del browser al server del banco no es esto) es una encriptacion nada mas.
Aplicaciones web por definicion son asincronicas. (Nivel virtual- encriptacion- joda) ->
Cuando entramos a un sitio web y tarda mucho y vamos a otra URL y entramos rápido y la anterior sigue sin responder.
Las aplicaciones web por definición son asíncronas.
Si atras del browser se conecta con un servidor de aplicacion. (donde puse mi codigo se hace alta o baja, saldo de usuarios).

“”

> Que pasa si me conecto al banco, me logueo y espero el home-page (primera pagina despues del logueo).

Agarro mi codigo, leo el usuario y password, se donde esta la ip y me conecto a la base de datos le doy al usuario interno de la aplicación para que me conecte y me deje acceder a la BD)
Le pongo una consulta un select y en el where pongo el nombre de usuario y password la aplicó a la BD y me devuelve el resultado y yo voy para atrás. y respondo.

”””

¿Vieron alguna incompatibilidad en lo que acabo de decir:
La primera parte de la aplicación es asincrónica.
Luego cuando tome control y empeze con el application sever a conectarme a la base de datos es síncrono.

Sincrono y Asincrono.
Síncrono: Proceso se queda esperando a que le responda la BD, si BD esta muy ocupada, el servidor de aplicaciones va a quedar ocupando recurso bloqueado porque espera que le responda otro servidor.
A nivel de performance y a nivel de optimización de recursos **no es lo apropiado**, mezcló distinto tipos de procesos sincrono y asincrono.

Lo que tendría **que hacer** es que **todo** se **parezca a una arquitectura asíncrona.**

Ahí es donde aparecen las bases de datos y adelante se ponen los servidores de cola de mensajería (Queue server).
Porque la Queue server queue me convierte una aplicación síncrona en una aplicación asíncrona.
¿Porque la Queue Server me convierte una aplicación síncrona en una aplicación asíncrona?
Porque en lugar de hacer la consulta SQL con el usuario y password directamente a la BD y quedarme esperando a que me responda,
Yo hago la misma consulta pero se la pasó al **queue server** me va a dar el numero de cola (en el cual yo voy a preguntar periódicamente para decirme si ya respondió o no).

Cuando ponga los datos (consulta si juan con esta pass es usuario del banco) en el queue Server **(A)**, me libera los recursos y me devuelve el control. (Anda a buscar cada tanto tiempo en esta queue Server **(B)** si ya respondió la BD).

El queue Server **(A)** se da vuelta, se conecta con la base de datos le tira una transacción y la base de datos cuando responda retorna una queue Server de profundidad 1.

El cliente periódicamente va a preguntar:
¿Se creó la cola **B**?,
¿Se creo la cola **B**?
¿Se creo la cola **B**?
Si se creó la cola B, es porque está el dato, voy busco el dato y respondo.

Entonces con un Queue Server generamos un sistema mucho mas asíncrono.
No nos quedamos sin el control. (En el síncrono nos quedamos esperando tiempo en la consulta hasta que nos responda, perdemos el control).
Cuando **diseñamos un programa nunca** tenemos que **perder el contro**l, tenemos que ejecutar lineas (max que tarden ms) y que me **devuelva el control** para seguir **el flujo de la aplicación**.

Por eso se pone en el medio un queue Server.
queue Server: diseñado para **recibir** muchísimos **requerimientos**, poderle dar servicios a todos, se da vuelta y no saturar la base de datos, le va pasando las consultas a la base de datos y cuando va respondiendo el queue Server va colocando la respuesta en una nueva cola.
Para que cuando la persona que tomo el control cuando ejecuto la inserción en el queue Server (tarda ms), pregunte y si existe la cola ya extrae el dato.

Mas probable que pase en la vida profesional:
Pocas veces nos conectamos directamente con las base de datos y en la mayoría de las veces se tenga que conecta con un queue Server, le tiramos la consulta SQL y la queueServer nos retorna un nombre de queue Server que tenemos que ir a chequear. Lo hacemos tan burdo como **sleep(1)** y consultamos si existe la cola de mensajeria si existe tomamos el dato y volvemos.

En **aplicaciones web de alta performance:** siempre tendremos un queue Server (**nunca se le pega** directamente al **servidor de BD**).
El que ataja todas las consultas todos los requerimientos es un queue Server, generando asincronismo y encolando de manera apropiada los requerimientos al servidor de base de datos.

Cuestiones de performance que vimos en ls bases de datos:

1. **Normalización y Desnormalización, punto óptimo.**
2. **Base de datos en memoria.**
3. **Indices**: campos por el cual accedemos a los registros deberian tener indices para acceder rapidamente
4. **Tipos de Lockeo en registros**
5. **Usar un queue server adelante de la BD:**

Podria tardar mas por poner el queue server?
El ux es muy importante para el usuario, si quedamos lockeado y no podemos mandar información desde el servidor (pequeña anim de js en el cliente, no hace referencia a esto) es un problema. Pero como quedamos liberados en lugar del sleep(1) podemos mandar información al cliente para que el cliente vaya viendo todo el proceso **Mejora que tenemos.**

6. **Índices en el where:**

Cuando se accede a una tabla para hacer una consulta modificación o actualización no importa lo que hagamos el campo del WHERE **Deberia tener un INDICE,** quizas el mas obvio y el que muchas veces nos olvidamos. (sin tener en cuenta q esto es una base de datos y tiene índices, y deberíamos acceder por ellos)
Concepto de full scan y **queso gruyer.**

Clase 6 Repaso- Mas conceptos

SQL lenguaje de consulta.

Cualquier operatorio que un usuario podría hacer con una base de datos la podría hacer con SQL.

Se podría dividir en 2 el SQL:

1) **Lo propio de un usuario:**, trabajo de consultas a tablas, tener una forma ordenada de trabajar, unir datos de varias bases de datos, dar de alta, borrar, actualizar,
2) **Administrador de base de datos:** relacionado a la creación de una base de datos, tablas, de índices para las tablas, de mantenimiento de resguardo.

Normalización de las BD:

Podremos crear la base de datos como queramos juntar los datos, que esten repetidos, duplicados. Pero deberíamos normalizar los datos para mantener la integridad de los datos.

Integridad de los datos: Que no estén repetidos, que sean almacenados de manera eficiente.

Desnormalización de BD:

Una vez que normalizo me quedo plantado y eventualmente que se necesite generar una desnormalización (debemos ver cuestiones de performance, donde se necesita un tiempo de respuesta muy corto, muy eficiente).

Desnormalizados para que cuando grabemos/consultemos a una sola tabla y asi reducir el tiempo de respuesta.

Acceso programático a las bases de datos: Hay librerías que se conectan a múltiples bases de datos o librerías propias de la base de datos.

Uno tiene que hacer conexión a la base de datos con usuario y password, IP para encontrar dónde está la red.

El administrador de BD previamente me dio permisos a un conjunto de tablas o a una base de datos en particular. Estas tablas escribir/consultar/modificas, pero en estas otras tablas solo consultar, el grado de granularidad que tiene el admin BD para darme permisos de accesos a cada tabla es el máximo y el admin nos va recortando.

Todo el conjunto de operaciones de miles de usuario sobre una misma tabla, empezamos a entender el concepto de lockeo (lockeo de registros).

Cuando haces un update de un campo: la base de datos dice que este campo lo tengo bloqueado por un usuario.

Haciéndolo así impedimos que dos personas estén modificando un mismo campo de un mismo registro genera una inconsistencia en los datos. Se encolan todos los requerimientos.
Los lockeo el motor de BD hace automáticamente los lock, lo único que hace el programador es que:" modificarse esto".

Según el **nivel de modificación** que hago el **motor de base de datos puede ir** incrementando el nivel de **lockeo.**
Si Modificó muchos campos de un registro => Lock por Registro.
Si modifico muchos registros en un lockeo la BD puede decidir hacer un lockeo a nivel tabla, y mientras modificó la tabla nadie puede modificar la tabla.
Decidir que tipo de lock son operatorias que el motor de base de datos es el que hace automáticamente.

Indices: Estructura de datos (Árbol B^+, B, hash).
Permite acceder de manera muy rápida a los datos de esa base de datos.
Puede tener un Indices primarios y tener N índices secundarios:
Indice primario y secundario ambos son **estructuras en memoria que se guardan (se serializan** no se pierde) cuando haga **una consulta** y si consulto **por el campo** y justo ese **campo tiene índice**, va a ser mucho más **rápido** que si trato de acceder por un **campo** que **no tiene índice**.

Solución: Pongo todos los campos como indice y lo resuelvo.-> **No** necesariamente eso tiene un costo de espacio y de performance. -> **Busco equilibrio**

Diferencia entre índice primario y secundario:
El índice primario tanto como el índice secundario, pueden ser de un campo de la tabla o de n campos de la tabla (siendo n: cant de campos de la tabla).
Indice primario: es el **más rápido de todos los indice**, porque físicamente, la tabla (que tiene el indice primario) está almacenada en el **mismo orden** que **vemos en la tabla**.
¿Más rápido porque?
Porque si nos piden extraer todos los datos, leemos el primario y para el siguiente es el byte subsiguiente.

Índice secundario:
Fisicamente una tabla puede ser ordenada fisicamente de una forma, y por lo tanto tengo que ir saltando de parte a parte los registros donde estan fisicamente almacenados, porque no estan consecutivos.

Los índices se construyen con árboles B, tendrá un acceso mucho más rápido que si no tuviera un índice secundario.
Porque **cada valor en el árbol** se **guarda el registro** y también la **posición física del registro**.
(hablar de acceder es la parte del where q hacemos en el SQL).

Si accedemos por un **campo que no es índice vamos** a tener que **hacer** algo que se llama **Full scan** (**Recorrer todos los registros de la tabla**).
Ej:
Me piden todos los alumnos del depto de computación que están cursando base de datos:
*SELECT * FROM tabla alumnos a INNER JOIN tabla_base_datos t ON a.padron = t.padron WHERE t.cursaBaseDatos = true;*

Vamos a tener que recorrer de punta a punta toda la tabla y ver si el campo esta en True o False.
No debemos hacer nunca: “

> Una consulta en SQL donde accedemos por un campo en el cual **el motor de base de datos se vea obligado a hacer un full Scan** (**ese campo lo tengo que recorrer del primer registro al ultimo para dar la rspt**).

¿Como se resuelve esto?:
Hablar con el administrador de la BD, ver la posibilidad de que cree un indice secundario para ese campo , para no hacer un full scan.

Si lo hacemos de callados un full scan y el admin BD se dara cuenta, porque le salta la alerta de full Scan. (todos los admin de BD lo ponen: *Tira abajo en los tiempos de respuesta*). **Cuidado con esto.**

Usar SQL o la api para acceder a la base de datos:

	Usar SQL	Usar API
Ventaja	La consulta la podemos validar probar con cualquier interfaz de base de datos. Podemos generar consultas complejas, y además incluir sentencias propias de PSQL o SQL-Oracle	Estamos seguro que lo hizo en un dialecto de SQL que es SQL-ANSI-92 esto es un standard del mercado hecho en 1992 **para SQL**. Debo aprendo una forma nueva de consulta y me permite mayor portabilidad. Estamos seguro que lo hizo en un dialecto de SQL que es SQL-ANSI-92 esto es un standard del mercado hecho en 1992 **para SQL**. Si algun dia apunto a base de datos oracle y el administrador BD, decide migrar la base de datos a microsoft-SQL server. el admin BD paso todo. Si usamos el api SQL puede ser q haya usado una sentencia propia de PLSQL. Si lo hice usando API, lo unico que cambio es la IP (para el microsoft SQL)
Desventaja	-	Debo aprendo una forma nueva de consulta.

PLSQL- (Lenguaje programación SQL que implementa oracle). (incluye SQL-ANSI-92 + particularidades).
Microsoft hace lo mismo (Transact SQL -> SQL-ANSI-92 + particularidades).

SQL

```
[ ]  output = conn.execute("SELECT * FROM Student")
     print(output.fetchall())
```

```
[(1, 'Matthew', 'English', 1), (2, 'Nisha', 'Science', 0), (3, 'Natasha', 'Math', 1), (4, 'Ben', 'English', 0)]
```

```
[ ]  output = conn.execute("SELECT Name, Major FROM Student WHERE Pass = True")
     print(output.fetchall())
```

```
[('Matthew', 'English'), ('Natasha', 'Math')]
```

API

```
[ ]  query = Student.select().where(Student.columns.Major == 'English')
     output = conn.execute(query)
     print(output.fetchall())
```

```
[(1, 'Matthew', 'English', True), (4, 'Ben', 'English', False)]
```

```
[ ]  query = Student.select().where(db.and_(Student.columns.Major == 'English', Student.columns.Pass != True))
     output = conn.execute(query)
     print(output.fetchall())
```

```
[(4, 'Ben', 'English', False)]
```

Cuando tenemos el arbol B (de los indices) puede ser que modifique uno de esos campos que es indice, que se borre un registro y demas cosas.
Queso Gruyer.
En ese árbol se empieza a *encontrar faltantes (hay agujeros).*
Esto NO quiere decir que no vaya a encontrar datos o me traen datos que no existen.
Lo que sucede es una **cuestión de performance**, porque habria que recrear todo el índice para que quede bien estructurado(ordenado), lo mas compacto posible, y reducir los tiempos de acceso.

Una tarea como administrador de BD, analizar con **el tracer** que tiene **cada base de datos** y meterse en cada tabla y ver el índice de modificaciones que van obteniendo. Puede ser que decida recrear el índice, (Borrarlo y crearlo de nuevo), esto tiene un costo en tiempos y cómputo pero el resultado de ese proceso es que el índice está optimizado y las personas que quieran utilizar este índice van a obtener un resultado más performante, que si no se hubiese hecho.

Porque se genera el queso gruyere?
porque se van haciendo modificaciones, inserciones, borrando datos. El indice tiene un montón de agujeros que no son utilizados.

Índices pueden tener características (tipos):

1) **Indice numerico:** El campo es numérico
2) **Para un campo o muchos campos.** (datos numéricos con texto,etc)
3) **Índice único:** Cada dato del campo no se puede repetir.
    a) **Ej: id_cliente:** Es un indice del **tipo numérico único**, (no hay cliente con el mismo id_cliente).

Si tengo un índice secundario que es {nombre, apellido} no voy a poner que es único porque me puede aparecer dos ‘Juan Perez’.

Un buen administrador BD **gestiona la creación de índices los fines de semana a la noche**. Con el tiempo se des optimiza el índice y tenemos que recrear el índice para que quede ordenado.

Base de datos relacionales, porque **relacionan filas con columnas.**

Parte 2 de la clase:
sqlite: Mismo concepto de oracle pero local, a nivel de diseño.
sqlite:

1. El concepto de guardar datos y poder consultarlos haciendo consultas SQL es práctico.
2. Por la cantidad de datos que guardamos no hago la parafernalia de acceder a un servidor, podemos usar un nodo local.
3. Puedo combinar ambos, por un lado tengo datos del cliente (A grabarse a la BD-oracle-mysql-posgtree) y los datos de sesión del usuario y **no lo quiero grabar** en la BD y **tampoco en un array**, **solución**:
    a. Me genero con sqlite, una pequeña tabla 'sesion' dentro de una BD, voy haciendo consulta, updateando, borrando, etc.
    b. Resolvimos un problema de diseño de sistemas de manera apropiada.

Pandas herramienta para la ciencia de datos, es como tener una base de datos en memoria.
Luego creamos una base de datos en duckdb.
connect (Si no lo encuentra genera una base de datos).
¿Que es un cursor? Importante tomar nota:
Es un puntero a un registro,muy importante en cualquier BD.
Ejemplo:

> *Estamos haciendo una aplicacion que muestre el patron de la ciudad de **Buenos Aires.** Debemos hacer que la persona consulte por DNI, o por apellido. Ponemos el DNI y obtenemos un único registro.*
> *Si ponemos apellido: ej 'Perez' me va a traer muchos resultados y me satura la cantidad de memoria que tengo que usar para mostrar los datos al cliente.*

Alternativas:

1) **Crear un cursor:** En lugar de **devolver los 3k perez el cursor** me devuelve **de a 10**, t**raeme los prox 10**, **traeme los prox 10**, y asi. (Cursor es como tener un puntero en esa tabla memoria resultado de la consulta). **En lugar de guardame toda la consulta me genero un cursor y voy mostrando los datos** a **medida que me piden.**
    a) El cursor me permite moverme dentro de la consulta que hice, hacer saltos de una manera **mas eficiente**.
    b) **Traer todos los registros** de **la consulta** y luego poner **10 registros en la primera pagina** y los **prox 10 en la sgt pagina** (Casi nadie llega a la 2 pagina) **ESTA MAL,** consumimos recursos innecesariamente.
2) Uso de LIMIT la consulta nos trae 10k registros pero con LIMIT 20, lo cortamos a los primeros 20.

3) Si es solo mostrar nos debemos traer solo la cantidad de registros que necesitamos mostrar.

Desventaja del cursor: El cursor mantiene recursos del motor de la base de datos ocupado.

Estrategia: Levantar pandas, aplicar las operaciones que queremos usando SQL y luego regresamos el dataset a pandas y trabajamos con los dos mundos.

Pandas desventaja: Cuando hacemos un **read_csv**, tendria que **poner que tipo de dato** es **cada** uno de estos **campos** para que lo reconozca. (por **default lo levanta** con el tipo de dato **mas amplio**).
En cambio puedo levantarlo primera vez, miro lo que es lo cambio los tipos de datos y luego lo paso a duckdb.
varChar: tipo de dato (para distinto tamaño) para bases de dato muy pesado.
Char variable con parte fija y variable.

Duckdb y sqlite compiten.
Cosas muy simple nada mejor que sqlite, **no es pesado, super Estandarizado**, estable. **Algo más sofisticado me conviene trabajar con duckDB**.

Lance y parquet.
Parquet: una base de datos que no llega a ser una base de datos, tampoco es un archivo común y esta a medio camino. Parquet Usado para big data, cuando los datos son muy grandes.

DuckDB- Posible TP que se haga en un futuro
Levantar un data set. Extraer los datos de manera gratutiita. LEvantar con pandas. Operaciones con pandas. Migrar a duckDB o Lance. Y Consultara una lista de acciones que le voy a pasar. Ese es el tp. en 2 semanas.
Clase 8 y clase 9. Parcial Es muy practico si van haciendo el TP.
Combinacion del trabajo practico y un par de preguntas en google form. las cuales, les dara simple preguntas, ahora tiene un tiempo perentorio si empiezan a googlear

Clase 7 (SQL- En 1 hora de cero a experto)

1) **Beeteewen:** Sentencia para tomar rango de numeros.
2) **Like** : para poder hacer matcheo de strings: %c%: empieza y termine con cualquier letra pero en el medio q exista c (incluye al inicio y tambien) con “a__%” Decimos que despues de la a tiene que haber **dos caracteres**.
   Wildcar (Comodín) : “ _ %”
   a) SELECT * FROM Products p
      WHERE p.ProductName LIKE "%c%"
3) **IN:** Para simplificar la búsqueda de varios elementos.
   a) Ej: SELECT * FROM Customers
      WHERE City IN (‘Paris’ , ‘London’, ‘Madrid’)
4) **AND, OR:** hacerlo con paréntesis y tener cuidado.
5) ORDER By (ASC es el default) : Podemos poner muchos Muy usado en las consultas, yo **descargo en la BD todo el esfuerzo del orden** y ya lo **recibo ordenado en** la **aplicacion**. (se debe hacer del lado del motor, porque tengo que aprovechar el recurso que tengo).
6) **TOP 4:** Para limtiar la cantidad de resultados.
7) **Min, Max**: son “***primitivas”*** que le aplicamos a un campo.
   a) SELECT * FROM Products p
      WHERE p.Price = (SELECT MIN(p2.Price) as minPrecio FROM Products p2)
8) **COUNT(), SUM():** mas primitivas (“**primitivas**”), cuenta cuantos valores sin repetir hay en un campo, sum suma todos datos de un campo.
9) **AS:** Alias para definir el nombre de una fila como quiero que me aparezca.
10) **Group by:** Me permite sumarizar agrupo por un campo.
11) **HAVING**: para filtrar agrupaciones.
    a) EJ:

```
SELECT COUNT(CustomerID), Country FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

WHERE se aplica sobre customerID y no sobre una operación por eso no podemos reemplazar el having por un where.
Lo que hace el SQL:
Va a ir a buscar la tabla customer y me piden dos cosas:

1. Que agrupe por pais (Por el group by)
2. Luego de que agrupo por el pais me dice que operación le hago al group by. Lo que hare es que a la column CustomerID le aplico el **COUNT,** y al resultado (que esta en una tabla en memoria temporal) y me dice que me quede con los count(customerID) >= 5.

No funciona el where CustomerID > 5 , porque ahora la tabla en memoria que genero las primeras 2 lineas no existe una columna con nombre CustomerID, existe una columna con **COUNT(CustomerID).**
Como es una tabla en memoria lo que tenemos que hacer es **HAVING y ponerle la condición.** No es un campo con el que puedo consultar con where.

```
SELECT SupplierName FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE
Products.SupplierID = Suppliers.supplierID AND Price < 20);
```

Se pudo haber hecho de esta otra manera buscando usando el **IN.**

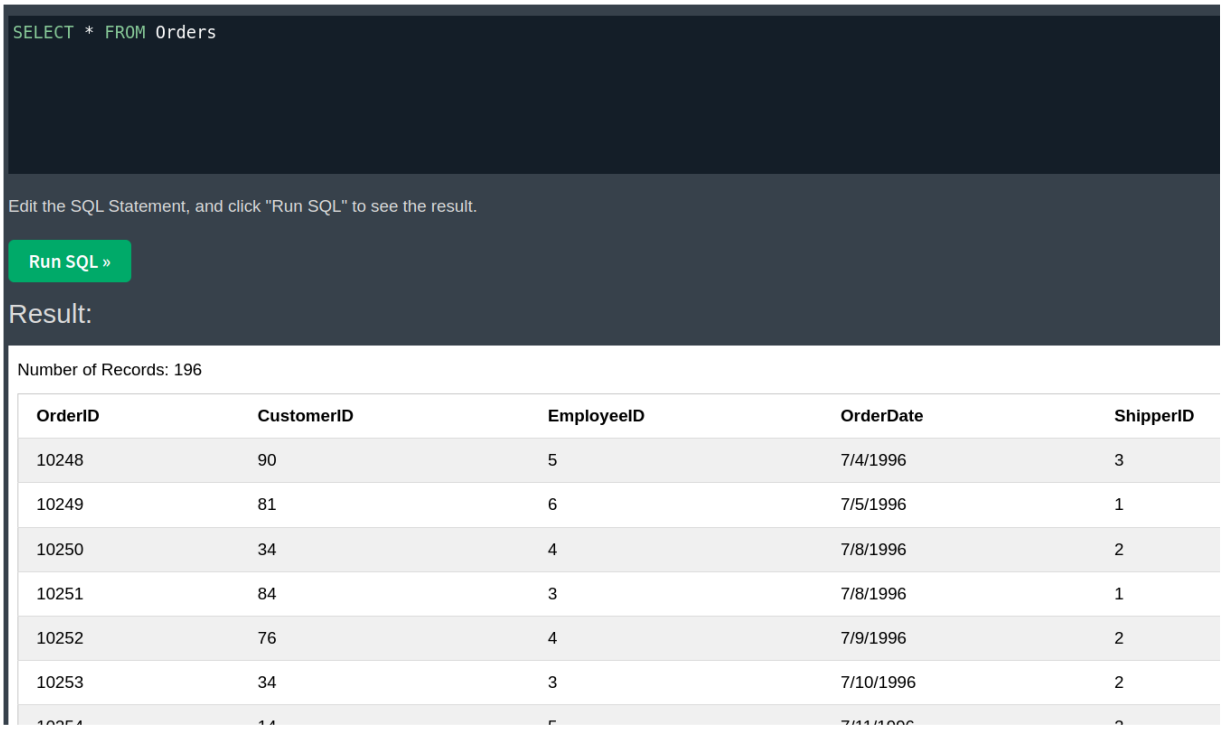
```
SELECT SupplierName FROM Suppliers s
WHERE s.supplierID IN (SELECT p.SupplierID FROM Products p WHERE Price <
20);
```

La subconsulta en EXITS crea una tabla temporal en memoria.
Es la forma que debemos pensar cuando hacemos consultas SQL.
¿Porque mataría al motor de base de datos?
Porque muchas veces voy a generar tablas temporales (**resultado de una subconsulta**) en memoria que son teras de información **va a explotar.**
Motores de base de datos relacionales se manejan con tablas en memoria que crean y destruyen.
En performance Exists vs Join:
No son iguales en performance, **si el join esta bien hecho es lo mejor que podemos hacer.** Con Join podemos configurarlo mucho mas. Con Exits le damos al motor que lo resuelva todo.
Si soy bueno con el join uso join.

Muchas veces podemos obtener el mismo resultado de varias formas distintas, lo que ahí entra es: **¿Cuál es la menos costosa? acá entramos a un 2do paso.**
Entender que es más costoso para la base de datos ir por A o por B.
Grano fino lo veremos cuando trabajemos.
SQL -> miércoles prox no hay clases merlino ta en salta.

Clase 8) Joins

Implica unir 1, 2 o más tablas.
Cuanto habla de normalización se refiere en el ejemplo a **CustomerID**.
Si hago un **cambio en customer esta** y quedo limpio.



OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	7/4/1996	3
10249	81	6	7/5/1996	1
10250	34	4	7/8/1996	2
10251	84	3	7/8/1996	1
10252	76	4	7/9/1996	2
10253	34	3	7/10/1996	2

EJ:
Al momento de hacer un **insert en esta tabla Orders**, tengo que hacer una consulta mas ir a customer y extraer el CustomerID de juan perez **saber que es el 90,** hacer un insert en la segunda transacción para insertarla en nuestro tabla. (Esto es una contra de normalizar).

Recordar que es normalización?
Ver el CustomerID tiene un numero y todos los datos de ese customerID está en otra tabla.
Otro ej:
Insertar una n**ueva orden para Juan perez**, busco a **juan Perez** obtener su CustomerID y creó un OrderID con el CustomerID de juan Perez.
No pueden tener millones de customers en memoria. Podemos tener una vista.

Una Vista es una tabla virtual cuyo contenido esta definido por una consulta Mas info en clase 11., tabla que realmente no existe.
Vista: Simular una tabla pero con menos campos o combinación de ellos y se pueden hacer consultas de sql como si fueran tablas.
La vista esta en memoria, la tabla customer accedemos mucho de lectura y poco de escritura, me ayuda a resolver este tipo de problemas.

Las bases de datos nos dan un monton de funcionalidad, no quiere decir que siempre lo tenemos que utilizar, siempre todo depende de la ingenieria de software, como diseñamos el sistema.
Error común de diseño: Generar mas de lo que realmente necesitan.
Trabajo: Te toman en el trabajo consultas con joins, entender los joins, normalización y de índices.
Índices es **importante entender** por que campos hacer esto:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

Tablas cazabobos donde tienen **mas de un campo** por el cual se podria unir. **Otra:** union entre la Tabla **A** y la Tabla **C.** por un campo, pero en la tabla C ese campo **no es un campo indice**. (**Otro error)**. -> Trampa

Campo A es el resultado de la búsqueda pero tienen que extraer de la tabla C, el **campo donde haríamos el join no es un campo indice**, y por lo tanto **obligamos al motor BD a hacer un gran esfuerzo.**

Muchas veces tenemos una **tabla B**, donde tienen un **campo indic**e que se puede r**elacionar con la tabla A** y un **campo índice** que se pueda relacionar con **la tabla C**, obtenemos el **mismo resultado sin hacer full scan.**

Lo más costoso al trabajar con SQL, es hacer un where por un **campo que no está indexado**.
Where **consulta unica** que no se hace mucho -> **bueno hacemos el full scan.**

Full scan: Recorre toda la tabla para **ver si la condición se cumple** en algún registro, se recorre de punta a punta la tabla.

Cuando tiene indice: (estructura árbol B) el costo de recorrer ese campo **es mínimo** sabemos todos los registros a ir a buscar, porque en cada nodo tiene una referencia a la **posición física del registro**).

Examen de trabajo: te ponen consulta y no tienen que ejecutar el concepto de full scan. Si no tenemos indice **comentar** que se hara un full scan y no tenemos el indice.

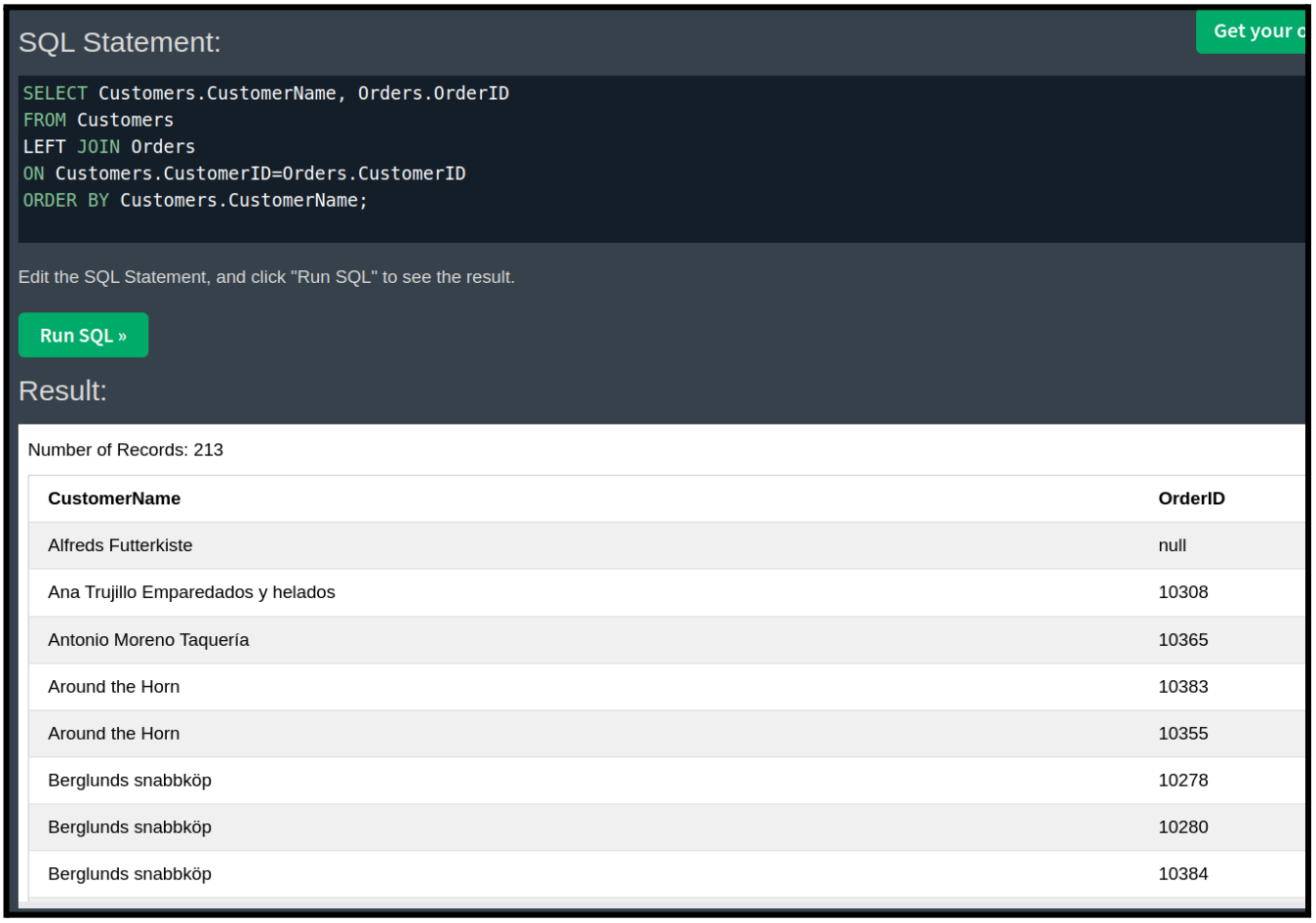
Entonces regresando:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

Aca al menos el **campo de destino** `Customers.CustomerID` tendría que tener un índice. Si **ambos lo tienen mucho mejor.**

Ver los tipos de join:
Left Join: todo de la tabla izquierda y lo que comparte con la tabla derecha (intersección). Tengo que tener cuidado con:

1) **Tengo todo de la parte izquierda** y trato de unirlo con la parte derecha, si no tengo parte derecha de un registro **pongo null.** (Alfedro tiene orderID null porque nunca hizo un order)
2) Puedo t**ener 2 órdenes para un customer** y me lo repite entonces. (Around, Around)



Si no entiendo bien el left Join, vamos a multiplicar tablas en memoria gigantes.
Como minimo la consulta nos dara la cantidad de registros q tenga la tabla Customers.
Y el maximo es la cantidad max de order que tenga todos los customers.

Evaluar el costo de una consulta, cuantos registros vamos a obtener.

Entender bien si debo usar un RIGHT O LEFT Join.
Tener mucho cuidado y hacer pruebas antes de ejecutar una consulta que dejaremos en una aplicación, más pruebas ejecuten mejor es.

Inner, LEFT y RIGHT tenemos que tener muy en cuenta.

Self join:
Cuando unimos por la misma tabla.
Nunca lo uso merlino en su vida, lo que si usa es **RIGHT, LEFT , INNER join.**

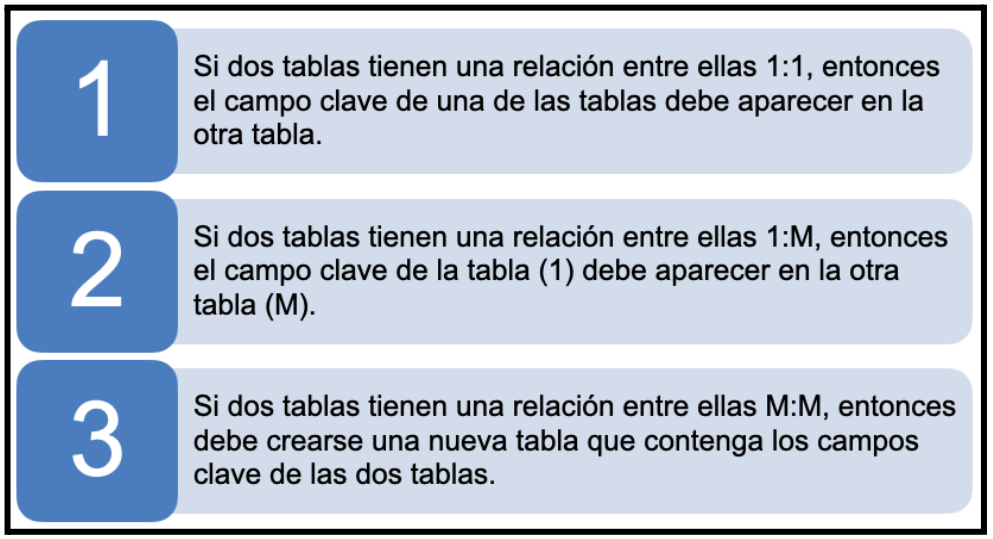
Clase 9) Teorica Libro Patricio

Tengo una índice Primario (coincide con la clave primaria).
Va a ser la de más rápido acceso, ordenado físicamente donde cada campo es el siguiente.
Supongamos que tenemos un indice secundario tambien por apellido.
Entonces cuando diga gomez al lado tendrá un id fisico al registro.

Indice primario y secundario diferencias:

1. Desde el lado de usuario no hay diferencia entre indice primario y secundario.
2. **Índice primario**: es el índice por el cual fisicamente el archivo está ordenado.
3. **Indice secundario:** Se levanta en memoria (pero luego se serializa), es un estructura de datos arbol-B, y cuando llegamos al nodo hoja donde encontre 'Gomez', al lado tenemos una direccion hexadecimal que apunta a la direccion fisica del registro

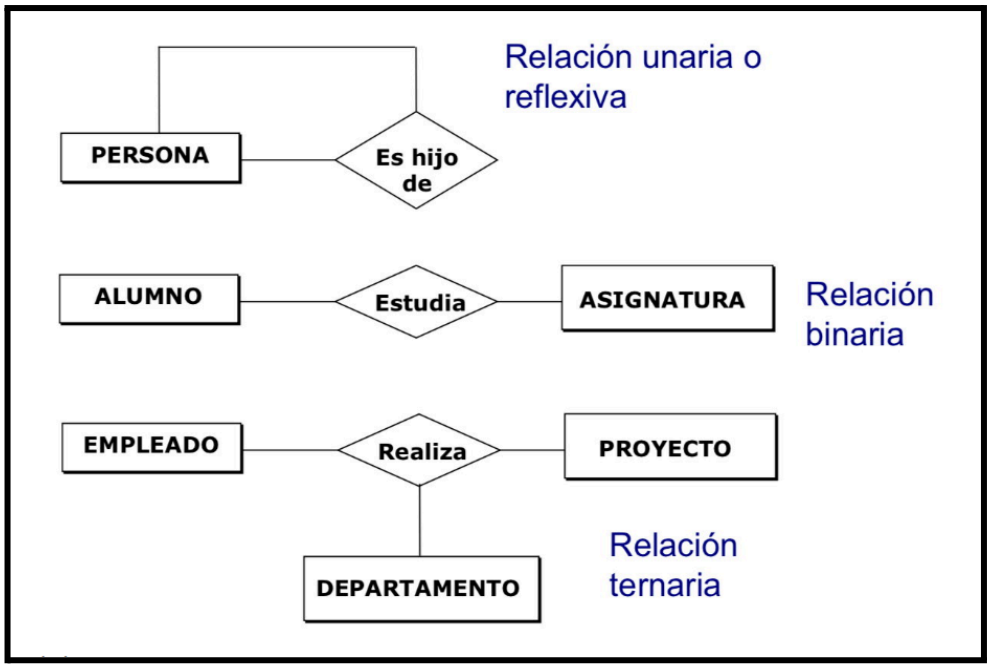
Reglas de integridad:



1) Un registro de una tabla tiene una relacion univoca con el registro de otra tabla. Ej: persona y telefono. (seamos laxo)

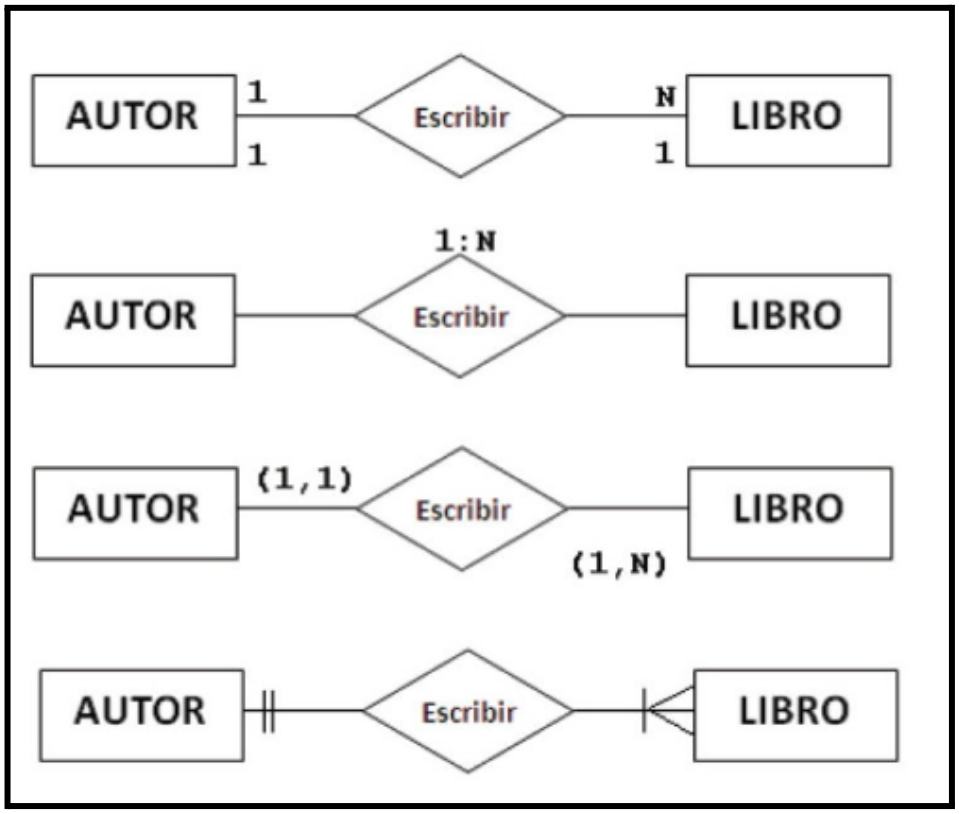
2) 1 a muchos, teniendo la relacion Padre e hijos, puedo tener una relacion para un conjunto de padres puede tener 0, 1 o muchos hijos. (se pone el maximo).
3) Relacion M: M. Ejemplo de facturas y articulos. Muchas facturas apuntan a muchos articulos, y muchos articulos apuntan a muchas facturas.

Modelado de datos:
Tiene 3 esquemas (diseños) **conceptual**, **lógico y físico.**
Conceptual: Básicamente aplicamos el modelo entidad-relacional (MER).
Empezamos a **trabajar con la BD usando el MER**. Del análisis que hacemos surgen conceptos, lo hacemos para ver qué tipos de tabla tendremos y como irán relacionadas.

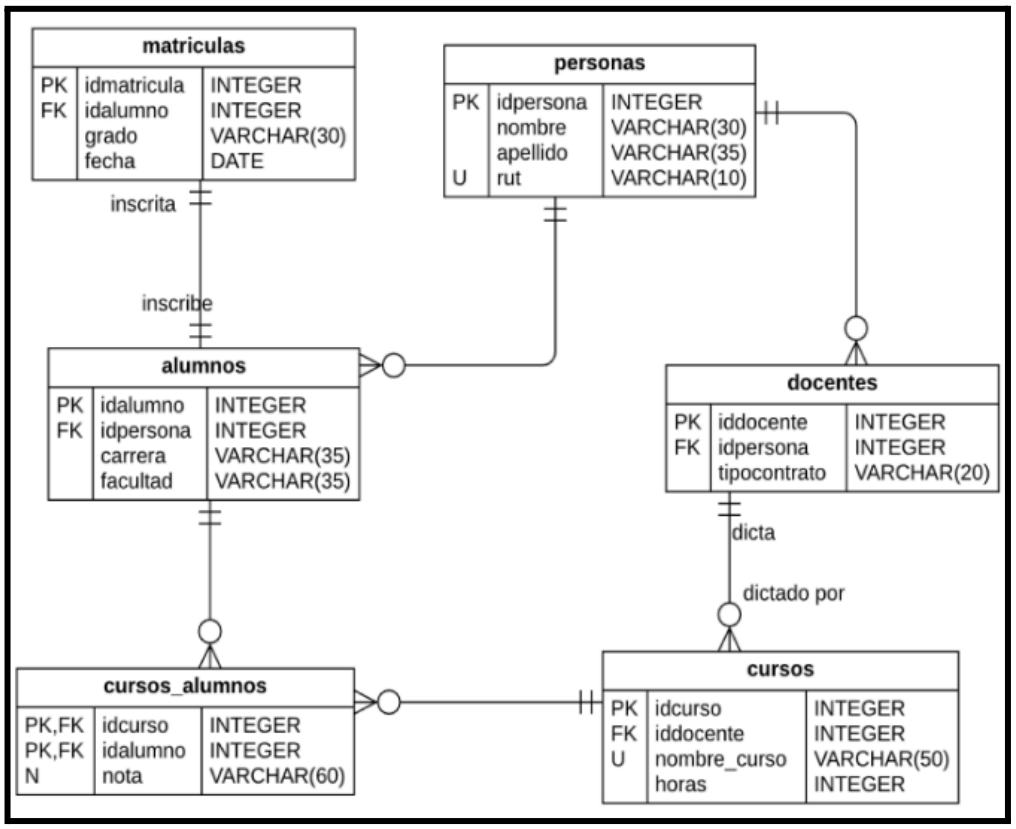


Modelamiento conceptual

Muchas veces se hace de forma intuitiva y nos mandamos a crear tablas.
este es el problema.
Primero debemos definir las relaciones, luego hacer el modelo lógico (empieza a tener la cardinalidad).
Las 2 lineas || -> es 1 y si tenemos un tridente es muchos.



Modelamiento Físico.



Matricula tiene una relacion 1 a 1 con alumnos.
Persona tiene una relacion muchos a muchos con alumnos y docentes.
Cada rectángulo que vemos es una tabla y el conjunto es una base de datos.
(**Tenemos el backend de la BD armada.)**

El modelo conceptual sale del analisis que hacen los analistas funcionales, cuando llego al equipo de diseñadores, los arquitectos (deben entender por ej : debe ser persistente -> base de datos).

Todo los campos y los tipos de relacion salieron del análisis (me lo pasaron los analistas funcionales).
Modelado físico representa cómo se construirá el modelo en la base de datos. Un modelo de base de datos física es lo que se vuelve a las bases de datos relacionales.

Modelado teorico de datos termina con el modelado fisico.
Bases de datos NoSQL, orientadas a grafos, documentos, etc. (Que no son tuplas-como el modelo relacional).
Como minimo se acepta la F.N 3 lo ideal es 3.5 (boicot).
Modelado fisico lo **pasamos por el tamiz de la normalizacion** (3FN , -FB)
Con esto vamos a la BD-Relacional elegimos SQlite, Mysql, postgree,etc. Conecto y creo las tablas.
Base de datos, ciclo de vida
¿Que es el diagrama entidad relacion?
Es un diagrama que nos permite diseñar inicialmente nuestra base de datos definiendo las entidades y las relaciones entre estas, usualmente se usa **la notacion de Chen**.

¿Que es el modelado físico ?
Es la representación de como se construirá el modelo de datos en la base de datos, es lo que se volcara en las bases de datos relacionales.

Modelo no relaciones (NoSQL):
Bases de datos clave-valor, orientada a documentos (puedo poner cualquier cosa adentro q no tenga el formato de una tupla), base de datos orientado a grafos.

Eso internamente tiene una base de datos orientada a grafos, tiene nodos y arcos que estan relacionados.
Optimización de base de datos: (3 grandes componentes).

1) **Formas normales:** para evitar redundancia de datos, que no esten duplicados los datos, integridad de los datos.
    a) ¿Que es la integridad de los datos?
2) **Índices**: índice, primario, secundario, clustering. los costos de tener cada una de estas tablas.

Data modeling: (DER) diagrama entidad relacion, en d
KDnuggets: leerlo todo obligatorio.

Muchas veces se utiliza una combinacion de bases de datos.

1) **¿Cuando trabajamos con bases de datos documentales?**
   si tenemos que guardar documentos PDF,etc.
2) **Quiero tener en mi aplicación la lista de usuario que accedieron y su actividad en los últimos 20 minutos, para luego sacarlo:**
   Puedo usar una base de datos key-value.
3) **Tengo un proceso de actualización masivo me llegan muchos datos:**
   Lo pongo en una base de datos que tome el registro como viene, muchas columnas y luego con un proceso lo dejo en mi base de datos relacional con la FN que corresponde.

Manejo de grandes volúmenes de información.
Si guardo documentos pdf o excel, montó una base de datos noSQL. para cuando venga lo tiro en la base de datos noSQL. y le pongo un rótulo y este rótulo estará la otra base de datos relacional para que cuando consulte a un usuario pueda ir a la base de datos nosql y extraer y ver cual era el documento pdf que me había dejado.

Clase 10 Merlino- Parcial algunos tips:

Para saber si lo aprendiste explicalo de manera simple, si cualquier persona te entiende, lo entendiste vos.
Indices entenderlo, cuestion de los join, base de datos que existen,
cual es el uso apropiado de cada uno.
Una aplicación combina distinto base de datos (una SQL y una NoSQL).

ODBC: quasi estándar de la **forma de conectarte a través de una librería** a una **base de datos**.

¿Como generar una arquitectura que soporte cientos de miles de usuarios a la vez?

Como diseñar una base de datos, grado de normalización que debemos dar a la aplicación?

mail: hmerlino@fi.uba.ar
Encabezado: Evaluación Base de datos 01

Se debe responder en forma sucinta las siguientes preguntas
Tiempo para enviar la respuesta por mail 40 minutos

1. Defina en forma empírica qué es SQL

2. Que son las formas normales y cual a la que se recomienda llegar

3. Lea la siguiente situación y de cuál sería la solución que aplicaría para que la aplicación sea performante.
    a. Se debe hacer una consulta desde una app a una Base de Datos que contienen unos cientos de registros. Que solución se debería aplicar.

4. Para solucionar qué tipo de problemas de acceso de Base de Datos se utiliza un servidor de colas.

5. Detalle los distintos tipos de joins que se pueden aplicar a una consulta

Clase 11 SQLite :

Las tablas en una base de datos están almacenadas **físicamente**.
Las vistas para el usuario final son tablas, consulta como una tabla.
Una Vista es una tabla virtual cuyo contenido esta definido por una consulta, tabla que realmente no existe.
Se usa por 2 razones:

1) **Cuestión de seguridad:** le pongo un grado más de seguridad a la base de datos, los usuarios nunca acceden a una tabla, acceden a una vista y de la vista se accede a la tabla física.
2) **Cuestión de normalización** puedo tener 2 o 3 tablas de la información, por eso tenemos **published** y **authorId** tenemos numeros en ambos campos.

a)

	id	authorid	title	published	price	genres
	Filter	Filter	Filter	Filter	Filter	Filter
1	4	396	Belinda	1801	10.0	NULL
2	6	191	De Profundis	1905	17.5	NULL
3	7	838	Lettres de mon moulin	1869	18.5	NULL
4	9	296	Eugénie Grandet	1833	10.5	NULL
5	12	665	Notes from the Underground	1864	10.0	NULL
6	13	607	Rimas y leyendas	1871	18.0	NULL

b) Como trabajamos mucho con esto el admin BD, **podemos genera**r una **vista** que ya esté toda la información, En lugar de **published** se vea **editorial atlantica**, y en **authorid** que **se vea campos de nombre** ej: juan perez y le sacamos el genero (mucho null), le podemos cambiar de nombre incluso a los campos. (published_name , authorname). Para el usuario final le parecera una consulta en una tabla, no se da cuenta que es una vista.
c) Hay una **pérdida mínima en el performance,** sin embargo es recomendable trabajar con vistas.

Triggers: Un trigger es una acción que se ejecuta automáticamente cuando suceda algo, el suceder algo puede ser tan simple como acceder a una base de datos o hacer una consulta (un insert en otra base de datos, etc).
El trigger lo podemos configurar para que se ejecute antes de la consulta que está realizando en la BD, o después que se haya ejecutado.

Si hay una solicitud de hacer un update/delete de un registro de una base de datos, se pone un trigger (sin q se de cuenta la persona q hace el update o delete) toma los datos de ese registro y lo inserta en otra tabla de log (para ver que se modificó).
Y entonces el administrador BD puede seguir la trazabilidad de todos los cambios que se dieron sobre un registro.
Porque cada vez que hubo una solicitud de cambio o que se borre un registro, lo que se hizo fue grabar en otra tabla. **Concepto mas difundido de trigger.**
Trigger: Cuando se produce una acción se ejecuta algo.
Al ejecutar algo alternativas:

1) Yo puedo poner la sentencia de SQL cruda (donde se dispara el trigger)
2) (**Mas eficiente**) Meter lo que quiero hacer en un store procedure.

¿Que es un store procedure ('Procedimiento almacenado')?
SQL no es el lenguaje de consulta de la base de datos, el acceso a datos se hace con el algoritmos del B-Tree.
Toma el SQL (se hizo para simplificar el acceso a las bd).
Antes acceder a una base de datos eran combinaciones de las tablas AND y OR de todos los tipos que se le puede ocurrir.
La **base de datos necesita hacer ese traspaso**, pero **como le pusieron el SQL** necesita un paso mas, como si fuera un compilador.
Cuando compilamos un programa perímetro se hace un análisis-lexicográfico lo que hace es validar que la sentencia sea correcta y compatible con el lenguaje de programación.
El **intérprete SQL válida que la sentencia esté bien escrita** y después lo pasa para que eso sea ejecutado.
A alguien se le ocurrió decir, que siempre tiene que hacer una consulta y lo único que **cambia son los parámetros**, EJ: **consulto las facturas de francisco mañana las de yusself pasado las de juan.** Hagamos algo para que esto sea más rápido:

Store procedure: Toma la consulta y pone el nombre que le pasan, **es lo mismo que una función** te d**evuelve algo que es el resultado de la consulta**, le podemos pasar variables, nos ahorramos el **Analisis lexicografo**, se **hace una sola vez** y cuando lo **grabamos en la base de datos** (si no se modifica) el sql sabe que esa sentencia es **lexicograficamente correcta**, entonces toma el valor de afuera y se lo escupe directamente a la bd para que haga la consulta.
Mejora en performance que tiene la consulta sql, hoy ese tiempo es ínfimo.

Rocio es la que mejor escribe sentencias de SQL, muy performante, merlino: perro que hace consultas de sql.
Diseñadores de software definen las consultas y las pasan a Rocio. Rocio desarolla las consultas y las guarda dentro de un store procedure cada uno de ella, las testea, las prueba, trazabilidad, y las deja guardada en la BD.
En el **repo pone**:

1) **Nombre del store-proceddure**, **parametros** y listo.

merlino perro en lugar de escribir la consulta cruda, voy a invocar la store proceddure. A **los admin BD les encanta**, no permiten que se ejecuten consultas que no sean a través de store procedure, así lo tiene controlado.
El único punto de contacto con la BD seria través de store procedure.

Store procedure: Es básicamente una consulta de sql guardada en la base de datos, la consulta puede tener parámetros (para hacer genérico la consulta) que al momento de ser invocada deben ser colocados.
La ventaja que tiene es que es más performante que hacer una consulta cruda, porque nos ahorramos de hacer el análisis **lexicográfico** (el análisis lexicográfico se hace al guardar la consulta en la BD).
Además es preferida por los administradores de base de datos porque podemos hacer la consulta muy performante y luego ser usada por el desarrollador.

Clase 12 Repaso parte 1

Repaso:

1) **5 formas normales**, la forma normal 3, y la 3.5 (toma cosas de la 3 , algunas cosas mínimas de la 4, hace un merge).
2) **Índices primarias y secundarias,** relaciones 1 a muchos, muchos a muchos, uno a uno.
3) Tablas de base de datos podemos agregarle una o varias índices de búsqueda.
4) **Árbol b-tree:** la misma estructura que esta codeado los datos en la base de datos. Con un indice puedo encontrar los datos de forma rápida.
5) Si no tengo un índice estoy forzado a recorrer todo el árbol. El nombre técnico de reconocer la tabla es el FULL SCAN. (Nunca usarlo), carga demasiado la base de datos
6) Claves principales (solo existe una) es la clave que coincide con el almacenamiento físico de los registros. Por esa clave el motor de base de datos no tiene que ir saltando
7) Todas las claves que voy a
8) Un índice puede ser una **combinación de uno o varios campos**.
    a) **Identificador** : **índice primario** (Una clave principal puede estar compuesta de varios campos). Internamente la base de datos ordena en función de la clave principal
    b) **Índice secundario**: **que tenga direccion y numero**.
    c) **Clave foránea (Foreign key): surgen del proceso de normalización**, Ej yo saque de la facturas la descripción del producto y solo deje el product_id y eso apunta a otra tabla ‘producto’ donde el id dice sabana de 2 plazas y su precio.
    d) **Única diferencia** entre indice primario y secundario **en programación es indistinto.** Internamente a nivel físico la BD está ordenada por la indice primario, los campos que lo definen lo decidimos nosotros.
    e) En la tabla **facturacion** voy a tener **factura_id** y **id_cliente**, debo tener tambien un campo item, cantidad y valor.
    En la tabla facturacion no me hace falta el valor del producto, porque lo puedo sacar por calculo, **Bueno NO** uno tiene que hacer desnormalización, (pasar de la 3.5 a la 3) le inserto el valor de lo que costo el producto ya luego con la inflacion lo actualizo luego.
    f) **Desnormalización de la base de datos**: **¿Cuando aplicas la desnormalización? Situaciones:**
        i) Cuando en cuestiones de sistema es apropiado sacar algo de esa perfección que nos da la normalización.
        ii) Genero una tabla con muchos datos que me viene crudos, y no voy a buscar los datos (de juanperez,etc). Grabo en la tabla a lo bruto (actualización masiva), no tengo tiempo para generar todo el

proceso, envio la info a la BD y con un proceso batch lo pasare a otra tabla normalizado.Ambos están relacionados en cuestiones del diseño de sistemas, en cuestiones de performance hay que desnormalizamos.

   g) **Normalización/Desnormalización**: ambos conceptos muy relacionado con el concepto de diseño de sistemas.
      i) Es bueno normalizar en medida de que el sistema no se vea perjudicado.
      ii) Cuando el sistema empieza a tener problemas de performance o algún otro tipo de problema lo tengo que dejar de algo.

9) **Laburamos en BD relacionales** estamos **trabajados con tuplas**.
10) Hay **3 niveles** que tenemos que **entender** muy bien:
   a) El **motor de base de datos** es el programa que administra un conjunto de bases de datos, el **motor Administra de 1 a N bases de datos**
   b) Dentro de **cada base de datos**, **tiene 1 a N tablas**.
   c) Dentro de cada tabla tiene una **primary key**, **indices secundarios 0 a N**, y 0 a N **foreing keys**.
11) Oracle, mariaDB, mysql, postgree, SQLAlchemy no son base de datos, son motores de base de datos, un programador puede crear una base de datos ahi.
12) **¿Por qué existen las bases de datos?**
   Porque es una forma de manejar volúmenes de información de manera óptima, antes no existian y administramos los archivos indexados de manera de que hoy lo hace la base de datos.
13) **Perfil: Administrador de base de datos**: nunca les deja hacer un **fullScan**. El **admin de BD** se encarga de los **resguardos de BD** mas usado es el **incremental**, la **BD tiene marcas** de las modificaciones que se va haciendo, Forma mas perfomante e hacer un resguardo de datos es:
   a) Un backup total, (como copiar **todas las tablas**).
   b) Genera un chrome equivalente al S.O, todos los dias a las 3 am hace un backup incremental.
   c) Backup incremental: **miro el ultimo backup** que se hizo y empieza a buscar apartir de esa fecha **si hubo cambios**, solo genera un **resguardo de datos** de los **cambios generados** de **esa fecha en adelante.**
14) **Caso mala suerte** se rompe la BD a las 16 pm, el **admin BD levanta el motor de base de datos**, crea una base de datos genérica y recupera el backup que tenía. **El motor de base de datos mira los backup hasta** encontrar **el backup total** y de **ahí hacia adelante va a bajar los backup incremental** y así hasta el último que tenga.
15) Periódicamente el admin BD hace un backup total (sabados a la madrugada 23 pm a 6 am domingo), durante la semana del domingo a la noche del sabado

va haciendo backup incrementales. Esto es **Política de resguardo o política de backup**

16) **Índices si tienen muchas modificaciones (inserts, deletes) Queso gruyere: índices quedan rotos**, (no es que quedan los valores incorrectos) los valores estan correctos pero fue pensado fue pensado **para otro estructura de datos** (**queda desbalanceado**).
    a) Otra tarea del admin BD agarrar y generar una re-indexación.
    b) re-indexación del dato: borrar el indice y lo crea de nuevo. (se hace ej 3:am de un domingo).
17) **chronometer del sistema operativo: Todos los S.O tienen un reloj,** marca una hora, podemos asignar tareas a ese chronometer para que se automatice.
    a) A las 3:00 am que haga un backup total.
    b) Podemos hacer un script que cuando termine del backup total, hacemos la re-indexación para mantener la performance de la BD.
    c) hacerlo con cmd, antes se llama AT y ahora scheduler TAS.
    d) Admin BD sabe muy bien como trabajar con scripts para hacer tareas de mantenimiento de la base de datos.
18) **BCP**: **bulk copy program**. **(programa de copia masiva)**
    a) **Cuando** se **pone un sistema nuevo en producción** y se tiene que actualizar tablas
    b) Cuando hacemos algun **cocinero (**cuando bajo los datos, bajamos un dump (un exe con permisos) donde hacemos un dump de la base de datos, en general es para hacer un dump completo de la base de datos. (obtenemos un csv con todos los registros). Esto se llama un **BCP**.
    c) **Todas las BD maneja un nombre de programa distinto**, pero **todas tienen** un programa que es un **Bulk copy program:** Sirve **para**:
        i) **bajar los datos de forma masiva** sin hacer una interfaz o una consulta de SQL.
        ii) Cargar los datos en una base de datos.
    d) Típica pregunta para entrar en el área de la base de datos:
        i) Tengo una tabla en una base de datos y tengo que subir datos por atrás con este programa BCP, esta tabla tiene un **índice primario y secundario y está en blanco**.
        ii) Por atras usando bcp levantamos el archivo y hacemos un insert masivo por atras de datos.
        iii) La pregunta es: ¿**Dejo los índices o los borro**?
            (1) Si son pocos datos (miles de registros) no los borren, es la misma operación que la base de datos haga todo.
            (2) Si son muchos datos (teras de información), la recomendación es borrar todos los índices, dejar tabla sin

índices, para que la BD sube los datos directamente con el programa BCP y luego nosotros creamos a mano los índices.

19) **Detalle de color para ser admin BD**, algoritmo de indexacion el sort, **todos los sorts quedaron en el pasado.**Todos esos sort existian cuando no teniamos la base de datos, teniamos que ordenar nosotros mismos los archivos de texto plano con algun sort.
    a) Tengo un archivo que debo cargar a la base de datos, si soy zorro y un buen admin BD:
    b) Borro todos los indices,
    c) Agarro el archivo plano y me hago un programa en C, y codeo el sort tal que me ordene por **esta clave** que **sera la pk**. Me deja el archivo ordenado de la manera que lo necesitare para la primary key.
    d) Subo el archivo que esta ordenado tiene la misma forma que la clave principal, y cuando esta arriba de la base de datos en la tabla que no tiene indice a este campo que ya esta ordenado le digo esta es mi indice primario, va a ser el mismo orden que se guardo, (por que yo ya le di ordenado). Por lo tanto la creacion del indice solo tarda lo que tarda en memoria la BD en armar el arbol B. (**Solucion de viejo zorro**)
20) Cuando no existían las BD ordenamos los archivos de texto planos se ordenaban, y carga a la base de datos. Creación del índice tarda lo que tarde la BD de armar el árbol B. Solución de viejo zorro para optimizar el manejo de base de datos.
21) Si no se hubiera hecho lo anterior (20), la base de datos tiene que mover internamente los registros para dejar la clave principal en el mismo orden que tengo los datos físicamente
22) Los sorts tiene una **razón** de ser porque dependen de cómo estar desordenado el archivo, si el **indice que le pasan a un archivo** esta **muy desordenado** en términos generales el más rápido es el bubble sort.
    a) Si el archivo no está muy desordenado, puede ser que el postman Sort o el quicksort sea el más óptimo.
    b) Si el archivo es grande y no tienes idea que tan desordenado **está el mergesort** es el indicado.
23) Vemos que los sorts que aprendimos nos pueden servir para hacer un bcp de manera más óptima y también entender arboles B.

2DA parte clase 12

Base de datos relacional se puede poner un **imagen o un pdf**, en términos generales tiene un tipo de dato **'blob**'.
Es muy doloroso para el motor de base de datos trabajar con esos tipos de datos.
Si tenemos que trabajar con mails, archivos planos, imágenes, json , xmls, binarios de cualquier tipo, es por esto que surgen las bases de datos noSQL.
Bases de datos NoSQL: es un **enfoque de diseño de bases de datos** que permite **almacenar y consultar datos** como mails, archivos planos, imágenes, json, xmls, binarios,etc.
Bases de datos no relacionales mayoría están **orientadas a documentos**,
Bases de datos NoSQL tiene un motor de base de datos (iguales a las BD relacionales), nada más que fueron pensando para un tipo de archivo particular.

MongoDB: de uso general y podemos guardar cualquier tipo de registro. (archivos, imágenes, gráficos, emails, xml, binarios). Le ponemos un rótulo y luego el archivo. Buscamos por el rótulo y no por el archivo (es muy costoso no se hace nunca).

Diferencia **BD Relaciones** vs **BD noSQL**
BD noSQL: **pensadas para un tipo de archivo particular**, fueron optimizadas para eso.Todas vienen con un símil-sql para extraer información.
Sistema moderno tiene un **uso compartido de BD relaciones y no SQL**.
Tiene un montón de tipos:

1) **Bases de datos orientadas a documentos:** MongoDB.
2) **Bases de datos de tuplas en memoria:** Redis, como tener un diccionario en memoria, supereficiente cuando tengo indice y valor.
3) **Bases de datos orientadas a columnas:**
4) **Bases de datos orientadas a grafos:** Por ejemplo cuando estamos accediendo a google maps.
    a) Hacerlo con una BD relacional es muy costoso.
    b) En cambio los grafos con nodos y arcos es una manera muy rápida de encontrar los dos desde donde quiero estar a llegar

Ahora existen json, sistema posicionamiento, xml, toda esa info tiene que tener conocimiento para mandar esta parte datos a una base de datos NoSQL orientada a documentos y esta parte de los datos va a una BD en memoria.
Existen decenas de bases de datos NoSQL tienen que leer para que fueron creadas.
Tomen 3 bases de datos:

1) **mysql o mariaDB** (BD relacional) practicar a acceder tablas, insertar, etc.
2) **mongoDB (**noSQL): La mas utilizada hoy por hoy, creen hagan consulta.
3) **base de datos en memoria** (**Redis**):

Si hacen un pequeño proyecto con las 3, intercambian datos entre las 3, tienen todo el conocimiento necesario para **entender las BD.**

Las **bases de datos NoSQL son distribuidas**, es un requerimiento que le pusieron y nacieron distribuidas (**nada especial**).
Bases de datos distribuidas: BD que pueden tener varios nodos.
Hoy por hoy todas las **bases de datos relaciones pueden manejar varios nodos**
Punto importante al manejar bases de datos distribuidas : Mecanismo de actualización:
Buscar en la **bibliografía de la base de datos** con la que trabajamos en caso de **tener mas de un nodo**, **cual es el mecanismo de actualización**.
Tengo el nodo A, B y C, un usuario accede al nodo A y modifica un registro del nodo A. (Por una cuestión de proximidad eso lo **resuelve la base de datos**). Hay un momento entre que se modifica el nodo A y se actualiza en el nodo B y C, en que queda "**desequilibrado**" (desactualizado) la **BD**.
Cada motor de base de datos implementa una **sincronización de bases de datos distinta**. Hay una parte de la configuración que podemos tocar.

Cuando se **modifica un registro en uno de los nodos**:

1) **NO** se manda el dato modificado a los otros nodos
2) Se **manda un broadcast** a **todos los nodos**, que dice "este registro está siendo modificado en el nodo "A".
    a) Si en ese momento una persona está consultando en el **nodos B (o C)** ese registro modificado en el nodo A, hace el costo y va hasta el nodo A para traer la última versión de ese registro, para cualquier otro registro le entrega la **copia que tiene en el nodo B.**

Cada **tanto tiempo va actualizando en todos los nodos la información.**
Sincronización de nodos distribuidos en base de datos (una materia en si)
Hay diversos tipos de sincronización.
Tenemos varios nodos para tener cuestión de rapidez y le pegamos a distintos nodos y le hacemos modificaciones en los distintos nodos.
Tiene **que haber un mecanismo** para **ir sincronizado todos los nodos**.
Bases de datos relacionales en general hay siempre un solo nodo.
BD relacionales mas sofisticada pueden permitir n cantidad de nodos.
Los distintos modelos de sincronización de las bases de datos relacionales también son usados por las bases de datos noSQL.

Hay una forma que los administradores hacen el cálculo del costo de una consulta en función:

1) Índices
2) Los registros que se quieren recuperar, a través de una consulta de SQL.

Administrador de BD hace un cálculo y me da el costo de la consulta, y puede decir '**hace otra consulta esta es muy costosa.'**

Hoy en día el administrador tiene herramientas gráficas que le muestra el costo de la consulta (update, consulta, delete, etc).
Con esa evaluación de costo el administrador te dice: 'Cambiame esa consulta de SQL tiene mucho costo'.

Clase 13 Repaso (chistes - historia merlino)

Resiliencia: importante para la vida adulta.
Estudiar por cuenta propia, trabajar a destajo, solucionar problemas (trabajar y sin saber el final del tunel).
Aula de las heras curso de 400 personas para un aula de 50.

Entra en el examen de merlino:

1) Formas normales (1, 2, 3, 3.5, 4 y 5) ¿Que es la tercera forma normal?
2) 1 Cuestiones conceptual de SQL. ¿que es un join, para que sirve?
3) 1 Ejercicio de SQL (o conceptual). / Tipos de base de datos
4) Arquitectura, que tipos de datos utilizarían para esto documentos.
5) ¿Para que sirve un server queue?, ¿Ventajas de usar un store procedure?, cosas que vimos a lo largo de la cursada.

Clase 14 :

Resumen de bases de datos NoSQL

¿Que son las Bases de datos NoSQL? en general explique las distintas bases de datos.

Son bases de datos no relacionales, no usan el esquema tabular de filas y columnas. Con estas se busca aumentar la cantidad de consultas, aumentar el volumen de datos y poder guardar información (audio, video, imagenes, jsons,etc) , aprovechando las ventajas de los sistemas distribuidos. Para ello usan el ***sistema gestión de base datos distribuidos,*** el **SGBD distribuido** este nos permite un sistema distribuido en distintas computadoras sobre una red y nos da la **abstracción** de ser un único sistema coherente.

Bases de datos orientado a documentos:

La unidad estructural de información son **documentos.** Un documento básicamente es un conjunto de pares claves:valor, además un valor también puede ser un documento. No se puede hacer un join de forma eficiente, por eso en los documentos tratamos de usar agregados. (son entidades distintas que se relacionan y los agrupamos para ser almacenados juntos ej: **post de facebook y comentario**.)

Ejemplo: **MongoDB**, usa JSON para la estructura de documentos. El análogo a una tabla es una ***colección,*** y a una fila seria un documento y las columnas serian

‘campos’. Claves en JSON siempre son “strings”, y los valores pueden ser int, strings, objetos json valores pueden ser multivaluados.

Bases de datos clave-valor:
Son bases de datos que **almacena información en forma de diccionarios** (par clave-valor), las claves son únicas y el único requisito sobre su dominio es que sean comparables por **=.** Tiene 4 operaciones: **insertar, modificar, borrar, obtener.**
No hay **funciones de agregacio**n y **debemos implementarlo a mano haciendo fullScan** o usando herramientas como **spark o amazon aws.**
Ejemplos:
En Dynamo: por ej gestor de base de datos de amazon, tiene claves del estilo: “**PrecioProductos:ChocolateBlancoMilky**:**Precio**”:5000.0.

Ejemplos de base de datos netamente clave-valor:
dynamoDB: que es usado por amazon,
redis: también usa base de datos clave-valor (se caracterizan por su alto nivel de rendimiento) gran operaicon de lectura/escritura en periodo de tiempos. (**altísima velocidad,** desventaja mayor volatilidad en los datos)

cassandra es un híbrido entre bases de datos clave-valor y base de datos wideColum. usada por facebook y twitter, netflix

Bases de datos wideColum:
Básicamente son **una evolución** de las **bases de datos clave-valor.**
Estas bases permiten **agrupar columnas en forma dinámica a cualquier fila**, donde ahora la **estructura de la información** son **wideRow**. (**filas anchas**).
Optimizado para bases con alta escritura.
Ejemplos:
Google BigTable
Cassandra: Base de datos utilizada por facebook, twitter, netflix.

En cassandra:
Una **tabla sería** un **ColumnFamily**.
Una fila sería estaría compuesta por:
1) Una clave primaria (que se divide en 2 partes)
2) Un conjunto de pares **clave-valor** (columna) asociado a una fila.

Diseño de cassandra está orientado a consulta, es decir primero debemos pensar que consultas hacer **antes de armar el modelo lógico**.Con el objetivo de solucionar una consulta usando una unica column family.
No existe el conjunto de join (junta) entonces los datos necesarios para la consulta debemos guardarlo en una **tabla desnormalizada desde el comienzo**.

Base de datos orientada a grafos:

Los elementos principales son nodos y arcos, estas bases resulta útil para modelar relaciones complejas entre entidades. En cada nodo se mantiene una referencia directa a sus nodos adyacentes, buena idea utilizarla cuando los tipos de entidades mantiene una relacion con otras entidades del mismo tipo. por ej **personas e hijo**.

Ventaja de resolver problemas clásicos de grafos:

1) Encontrar **patrones** de nodos
2) Encontrar **caminos** de nodos
3) Encontrar la **ruta mas corta,**

Un gran ejemplo es la base de datos **Neo4J** cuyo lenguaje de consulta es cypher inspirado en SQL. cypher se propone como estándar para bases de datos orientados a grafos GQL.

Primer fecha de final)

Tema 1:

1) **Explicar la tercera forma normal**
2) **Detalle los distintos tipos de joins que se pueden aplicar a una consulta**
3) **Dar un ejemplo de consulta sql que no devuelva todos los registros.**
4) **Explicar las BDD Nosql en general.**
5) **Se quiere diseñar un sistema de home-banking con cierto tiempo de respuesta y atender muchas consultas ¿Que pondrias entre el backend y la BDD?**

Tema 2:

1) **Explicar la forma normal de boyce codd.**
2) **Diferencia de una base de datos NoSQL y una BD orientada a grafos.**
3) **Dado un monton de imagenes, pdf, xml, mails que base de datos usarias para almacenar estos datos?**
4) **Explicar las BDD No SQL.**
5) **Se quiere diseñar un sistema de home-banking con cierto tiempo de respuesta y atender muchas consultas ¿Que pondrias entre el backend y la BDD?**

Segunda fecha de final)

Tema 2:

1) **¿Para que sirve la materia base de datos?**
2) **¿Que tipos de índices conoces?**
3) **¿Que es un store procedure? ¿y para que sirve?**
4) **Dada una base de datos con mas de 10k registros y tenes que almacenar xml, documento, json, y archivos planos y ademas un acceso rapido:**
    a) **¿Que tipos de base de datos usarías?**
    b) **¿Como mejoramos la capacidad que muchos usuarios se conecten?**
5) **Explicar la tercera forma normal**

Resuelto finales de merlino:
Tema 1)

1) **Explicar la tercera forma normal**

Una tabla está en 3FN cuando está en 2FN y no existen dependencias transitivas de atributos no primo con la clave primaria*.(*clave candidata: en caso de que haya más de una posible clave primaria).
Ejemplos tenemos la tabla facturas con los siguientes campos(columnas):
Facturas(nro_factura, nro_item, cod_producto, nombre_producto, cantidad)
siendo primary key = { nro_factura, nro_item}
vemos que nro_factura, nro_item implica el cod_producto y cod_producto implica al nombre_producto: de forma grafica:
nro_factura, nro_item -> cod_producto -> nombre_producto

Tenemos una dependencia transitiva de un atributo no primo (**nombre_producto**) Viola la 3FN. Lo podemos arreglar asi:
Facturas(nro_factura, nro_item, cod_producto, cantidad)
Productos(cod_producto, nombre_producto)

2) **Detalle los distintos tipos de joins que se pueden aplicar a una consulta**

Un join básicamente es una operación de junta que se realiza entre dos tablas, se realiza un producto cartesiano de cada fila. obteniendo una nueva tabla (mas ancha y mas larga). Generalmente se hace join de dos tablas A,B igualando la pk de A con la foreign key de la tabla B (que referencia a la pk de A).
Distintos tipos de join son:

1) Full Join: Básicamente hacemos el join sin ninguna condición de junta, cuidado si tenemos una tabla de 3x3 y otra de 3x3 tendremos una tabla con 6 columnas y 9 filas .
2) Self Join: hacemos el join usando una misma tabla dos veces.
3) Right Join: nos aseguramos de quedamos con todos los registros de la tabla a la derecha.
4) Left Join: nos aseguramos de quedarnos con todos los registros de la tabla al izquierda.
5) inner JOIN:

3) **Dar un ejemplo de consulta sql que no devuelva todos los registros.**

Con muchas clausula podemos evitar que se resuelvan todos los registros:

1) Where:
   SELECT nota FROM notas WHERE nota >= 7;
2) Con LIMIT:
   SELECT nota FROM notas LIMIT 10;

3) GROUP BY and HAVING:
   SELECT nota FROM notas GROUP BY nota HAVING count(nota) >= 10
4) **Explicar las BDD Nosql en general.**

Las bases de datos NoSQL, son bases de datos **no relacionales (es decir no usan el esquema tabular de filas y columnas**) surgieron para:

1) **Aumentar la cantidad de consultas.**
2) **Aumentar el volumen de datos.**
3) **Poder trabajar con otros datos (pdfs, imágenes, xml, json).**

Las BD noSQL además aprovechan las ventajas de los sistemas distribuidos implementando un **sistema gestor de bases de datos distribuidos** para darnos la **abstracción de ser un único sistema coherente** (cuando **nuestros datos están fragmentados en varios nodos**).
La desventaja es que perdemos **redundancia** a cambio de escalabilidad.
Los **joins no existen en algunas BBDD NoSQL en otros son costosos**, asi que se suelen **agrupar** lo**s datos relacionados** para ser **almacenados juntos y no se distribuyan.**
4 tipos de bases de datos:

1) **Bases de datos clave-valor:**

Se estructura la información en forma de par “clave-valor”, las claves deben ser strings, únicas y comparables por **=.** Las operaciones que se pueden hacer son insertar, eliminar, actualizar, buscar. Tampoco existen las funciones de agregacion las tenemos que hacer a mano haciendo un full scan o usando spark.
Ejemplo:

1) **dynamo** usado por Amazon, guarda **claves del estilo**: “**PrecioProductos**:**ChocolateMilky**:Precio”: 1500.50.
2) **redis**: base de datos en memoria (muy veloz, guarda en ram los datos, pero hay mayor volatilidad en los datos).

2) **Bases de datos wide Column:**

Son la evolución de las bases de datos clave-valor, básicamente a cada fila podemos agregarle columnas de forma dinámica. ahora la estructura de información son **wideRow.** Optimizado para bases con alta tasa de escritura.
Ejemplo:
Cassandra: que es usado por **facebook, twitter y netflix.**

1) Es una base de **datos híbrida entre un wideColum y una clave-valor**.
2) El concepto de **tabla** seria un **columnFamily** y el concepto de fila sigue siendo el mismo, la **columna** ahora es un par un clave-valor asociado a una fila.
3) No existe la junta: (si queremos hacer junta guardemos los datos en una tabla desnormalizada desde el comienzo).
4) Diseño de **cassandra es orientado a consultas,** debemos saber apriori las consultas que queremos hacer antes de hacer el modelo logico, esto para

poder diseñar las column Family y que cada consulta se resuelva con una unica column Family.

3) **Bases de datos orientado a documentos:**

E**structura la información usando documentos**. un documento **es un par clave-valo**r, y que a la vez e**l valor puede ser otro documento**. No se pueden **hacer joins eficientemente** por eso agrupamos entidades que se relacionan para almacenarlos juntos.

Ejemplo:

MongoDB usa json para la estructura de los documentos. El análogo a una tabla seria una colección, a una fila sería un documento, una columna sería un campo, podemos tener atributo multivaluados en los campos.

4) **Base de datos orientado a grafos:**

Los elementos principales son los nodos y aristas, resultan bastante útiles para:

1) Representar relaciones complejas entre las entidades
2) Modelar una entidad que se relaciona con otra entidad y ambas son del mismo tipo.

Además nos permite resolver los problemas clásicos de grafos:

1) Encontrar el camino mínimo entre dos nodos
2) Encontrar patrones de nodos
3) Encontrar camino entre nodos

Ejemplos:

Neo4J que usa el lenguaje de consultas cypher inspirado en SQL.

5) **Se quiere diseñar un sistema de home-banking con cierto tiempo de respuesta y atender muchas consultas ¿Que pondrias entre el backend y la BDD?**

Para poder atender muchas consultas y ganar performance se usa un ***server Queue.***

Un server queue (diseñado para recibir muchos requerimientos y evitar sobrecargar la base de datos) va recibir toda las consultas de SQL **provenientes del backend** con el siguiente proceso:

1) El **backend envia la consulta sql al server queue**.
2) El **server queue toma la consulta** y le dice al **backend que consulte cada tanto** tiempo en esta **cola de nombre por ej 'B'**.
3) El **server queue manda la consulta** a la **base de datos**,una vez **resuelta** el **server queue crea la cola 'B'** y **pushea ahí la respuesta a la consulta**.
4) El **backend por mientras va preguntando** cada cierto tiempo **si existe la cola 'B'**, y **cuando exista obtiene el dato.**

Con el **server queue** logramos que nuestra aplicación se vuelva asincronica, no perdemos el control de la aplicación quedándonos bloqueado esperando una consulta, seguimos con el flujo del programa.

Final 1 Tema 2)

1) **Explicar la forma normal de boyce codd.**

Para que una tabla este en forma normal boycecood (3.5) necesita:

1) Estar en 3FN
2) No deben existir dependencia transitivas de clave candidatas, es decir se prohíben las dependencia transitivas de atributos primos en la clave candidata.

Ejemplo tenemos una tabla:
Cursada(alumno, materia, profesor)
Identifico PK = {alumno, profesor} y hay otra posible clave candidata:
Ck1 = {alumno, materia}
Pero además tenemos que profesor implica materia. y materia es un atributo primo
Viola Boycecood viendolo de forma grafica:

alumno, profesor -> profesor -> materia

Lo arreglamos creando dos tablas:
Materias(Profesor,materia) Pk = {profesor}
Cursada(alumno, profesor) Pk = {Alumno, Profesor}

2) **Diferencia de una base de datos NoSQL y una BD orientada a grafos.**

Base de datos orientada a grafos tiene como elementos principales nodos y aristas una base de datos NoSQL en general usa un formato de clave-valor. Si necesitamos modelar entidades que se relacionan con sí misma la mejor opción es una BD orientada a grafos.

3) **Dado un montón de imágenes, pdf, xml, mails que base de datos usarias para almacenar estos datos?**

Usaría una base de datos mongodb porque con la estructura de documentos json podría guardarme las rutas y toda la información relacionada a esos datos.

4) **Explicar las BBDD Nosql.**
   respondido
5) **Se quiere diseñar un sistema de home-banking con cierto tiempo de respuesta y atender muchas consultas ¿Que pondrías entre el backend y la BDD?**
   respondido

Final 2)

1) **¿Para qué sirve la materia base de datos?**

La materia base de datos nos sirvió para poder conocer las herramientas que tenemos para poder guardar conjuntos de datos. Conocimos las bases de datos relacional y no relacionales y el software que se encarga de administrar/controlar/hacer consultas a la base de datos que es el gestor de base de datos. Conocimos el lenguaje para realizar consultas a las bases de datos SQL. (que actúa como un compilador porque traduce la consulta al lenguaje nativo que tiene la base de datos). Conoces las reglas para tener un buen diseño en las bases de datos relacional que son las formas normales. También aprendimos que técnicas podemos usar para hacer más performante nuestras aplicaciones:

1) Realizar consultas en el where **usando índices**
2) Desnormalización.
3) Usar una server queue entre el backend y la base de datos.
4) Usando locks aveces conviene hacer un lock por registro/tabla asi ahorrando la cantidad de transacciones.

También vimos bases de datos no relacionales: documentos, clave-valor, wide-Column,

2) **¿Qué tipos de índices conoces?**

3 Tipos de índices:
Un índice básicamente es un estructura Árbol B+,*,etc. Se usa para agilizar la búsqueda de registros a partir de un atributo o un conjunto de atributos.
Existen 3 tipos de índices:

1) **Índice primario:** Cuando el **índice se construye sobre la primary key,** y además la tabla **está ordenada físicamente por esta.**
2) **Índice de clustering**: Es un í**ndice construido en un campo que no es clave** y además que no es clave, la tabla **está ordenada físicamente por ese campo**.(como no es clave puede esta repetido algunos valores).
3) **Índice secundario:** Es un índice que se construye sobre los atributos, pero que no determinan el ordenamiento físico de la tabla.

Índice primario y de clustering solo pueden existir uno a la vez en una tabla, la tabla está ordenado únicamente por un campo. **Indice primario es útil cuando hago una consulta por rango.**

3) **¿Qué es un store procedure? ¿y para que sirve?**

Un store procedure básicamente es un archivo guardado en la base de datos que contiene consultas sql, estas consultas además pueden recibir parámetros (asi como una función) para que al momento de invocar una consulta le pasemos los argumentos.Sirve para poder realizar consultas sql que sabemos a priori que cumple con el “analisis lexicografico del lenguaje” (El analisis -lexicografico se hace 1 sola vez cuando se guarda el store proccedure). y que muy posiblemente lo usamos en nuestro dia a dia.

Los administradores de base de datos les encantan los store procedure, ya que con estos no permiten que se ejecuten consultas que no sean atraves del store procedure asi lo tenemos controlado. los admin BD pueden hacer consultas muy eficiente (que no hacen full scan) y guardarlas para que los desarolladores la usen.

4) Dada una base de datos con más de 10k registros y tienes que almacenar xml, documento, json, y archivos planos y además un acceso rápido:

a) ¿Qué tipos de base de datos usarías?

Podríamos usar base de datos orientada a documentos, mongoDB, y guardar en estructura json todos los documentos xml, json, archivos planos de forma muy simple.

b) ¿Cómo mejoramos la capacidad para que muchos usuarios se conecten?

Si queremos mejorar la capacidad de que haya muchos usuarios, entonces también aumentaría la cantidad de consultas que se realizan a la base de datos, para poder hacer más performante este diseño. Usaremos una server queue que será colocada entre el backend y la base de datos.

El objetivo es poder hacer asincronica la aplicación, de esta manera no perdemos el control del programa al esperar que responda una consulta la base de datos sino que preguntamos periódicamente si se creo una queue y obtenemos el dato cuando exista esa queue.

5) Explicar la tercera forma normal

Una tabla está en la tercera forma normal cuando se cumplen las 2 condiciones siguientes:

1) Está en 2FN
2) No existen dependencias transitivas de atributos **no primos** con la clave primaria*. (Clave candidata en caso de haber mas de una posible clave primaria).

Ejemplo tenemos una tabla facturas.

Facturas(nro_factura, nro_item, cod_producfo, nombre_producto, cantidad)

Pk = {nro_factura, nro_item }

vemos que:

nro_factura, nro_item implica cod_producto y que este implica nombre_producto, de forma grafica:

nro_factura, nro_item -> cod_producto-> nombre_producto

Entonces observamos una dependencia transitiva del atributo no primo con la clave primaria. VIOLA La **3FN.**

Creamos una nueva tabla **producto** que contenga tanto **cod_producto** y **nombre_producto** y eliminamos nombre_producto de la tabla Facturas.

Producto(**cod_producto**, nombre producto)

Facturas(nro_factura, nro_item, cod_producfo, cantidad)

Preguntas de final:

1) **¿Qué es SQL y para qué se utiliza en el contexto de bases de datos?**

SQL (Struct Query Language) Básicamente es el lenguaje estándar para poder realizar consultas a una base de datos relacional, el gestor de base de datos toma la consulta de SQL y luego la traduce al lenguaje nativo de la base de datos.

2) **Explica la diferencia entre las cláusulas WHERE y HAVING en una consulta SQL. ¿Cuándo usarías una u otra?**

La diferencia entre where y having es que where filtra los registros de una tabla mientras que having se usa luego luego de haber agrupado campos usando la cláusula **GROUP BY** esto generar una tabla temporal en memoria donde la manera de filtrar registros de esta nueva tabla en memoria es usando la clausula **HAVING**.

3) **Describe el propósito de la cláusula JOIN en una consulta SQL. Proporciona ejemplos de diferentes tipos de JOIN.**

La clausula Join tiene el proposito de realizar una junta entre dos tablas, usualmente la junta se hace con dos tablas (A, B) usando algun atributo en comun que tnenga ambas usualmente la Pk de A con la foreinkey de B que hace referencia la Pk de A (o viceversa). Hay diferentes tipos de join:

1) Self Join: Se realiza la junta usando la misma tabla. por algún campo
2) Left Join: Se realiza la junta asegurándose de obtener todas los registros de la tabla izquierda aunque no exista un valor asociado en la tabla derecha (se reemplaza con null).
3) **Right Join**: Idem a left. pero se asegura de obtener todos los registros de la tabla derecha.
4) **Inner join:** Se realiza la junta entre dos tablas quedándonos con los registros que coinciden por el atributo de junta.
5) **Full Join:** Se realiza un producto cartesiano entre las dos tablas.

4) **¿Qué es la normalización de bases de datos y por qué es importante? Proporciona ejemplos de al menos dos formas normales.**

La normalización de la base de datos son una serie de reglas para que nuestro modelado de base de datos tenga un diseño que cumpla lo siguiente:

1) **Minimizar redundancia**: minimizar la cantidad de datos que se duplican en muchos lugares.
2) **Integridad de los datos:** No olvidar la pk y las fk de las tablas.
3) **Anomalías de ABM:** Cuando actualizo en una fila no me basta tengo que actualizar en todas las filas.

2FN: Básicamente para que una tabla este en 2FN debe estar en 1FN y ademas todos los atributos no primos deben tener una dependencia funcional completa de la clave primaria*. (clave candidata* si hay mas de una posible clave primaria).

3FN: Para que una tabla en 3FN debe estar en 2FN y además no deben existir dependencias transitivas de atributos no primos con la clave primaria*. (Clave candidata en caso de que haya más de una posible clave primaria).

5) **Explique la diferencia entre una clave primaria y una clave foránea en una tabla. ¿Por qué son importantes en el diseño de bases de datos relacionales?**

Ambas claves (primaria y foreing) de una tabla se define al momento de definir la tabla.
La Primary key es importante porque permite identificar de forma unívoca cada fila (registro de una tabla).
La foreign key (clave foránea) en una tabla nos permite **juntar** esa tabla con la tabla que tiene como pk nuestra foreing key. Gracias a los foreign key es lo que nos permite poder juntar distintas tablas, establece una relacion entre dos tablas

6) **En una consulta SQL, ¿qué función cumple la cláusula GROUP BY? Proporciona un ejemplo práctico.**

GROUP BY es una cláusula de SQL que nos permite agrupar distintos datos que pertenecen a un mismo campo.
EJ:
SELECT n.nota, COUNT(1) FROM notas n where n.nota >= 6 GROUP BY n.nota
Aca agrupara solo las notas mayores a 6 y contara cuantas notas de cada tipo de nota (>=6) hay.

7) **Explica la diferencia entre las funciones agregadas COUNT, SUM, AVG y MAX en SQL. ¿En qué situaciones utilizarías cada una de ellas?**

Las funciones de agregacion. COUNT, SUM, AVG, MAX tienen una cosa en común es que todas ellas colapsan un montón de filas pertenecientes a un campo en una sola fila. COUNT cuenta la cantida de registros que hay, SUM(A), suma todos los datos asociado al campo A, AVG(A), saca el promedio de esos datos, y MAX(A) obtiene el máximo valor de esos datos.

8) **¿Qué es una subconsulta y cuáles son sus ventajas en comparación con las consultas simples? Proporciona un ejemplo de uso.**

Una subconsulta es una consulta anidada dentro de otra consulta, la ventaja es que nos permite hacer consultas más complejas.

```
SELECT * FROM Notas n
WHERE n.nota >= (SELECT MAX(n2.nota) FROM notas n2)
```

9) **Explica el concepto de transacciones en SQL. ¿Por qué son importantes y cuál es su propósito principal?**

Una transacción en SQL es una secuencia ordenada de instrucciones atomicas, se ejecutan como una unidad indivisible. Su propósito es garantizar la integridad y la consistencia de los datos. Por ejemplo podemos hacer una transacción donde

insertamos 1k registros pero en el registro 500 se encuentra un error, asi que se deshace todo lo que hizo, como si no se hubiera hecho ningún insert.

10) Describe el uso de las cláusulas ORDER BY y LIMIT en una consulta SQL. ¿Cómo afectan los resultados de la consulta?

La clausula **ORDER BY** ordenada los registros por defecto es ASC (de menor a mayor) tambien podes ordenarlo descendentemente (**DESC**). Luego con LIMIT podemos limitar la cantidad de registros que nos devuelva la consulta perfecto para evitar que nos muestra millones de registros.

11) ¿Que puedo hacer si quiero obtener mayor performance en mi base de datos?

Si queremos mayor performance podemos hacer lo siguiente:

1) **Desnormalizar tablas** para que en lugar de acceder a dos tablas o más tengamos todo en una única tabla (sacrificamos redundancia por perfomance)
2) **Uso de índices:** En lugar de estar buscando por un campo que no tiene índice en el where, agregamos un índice y el tiempo de consulta mejorará mucho, ya que evitamos hacer un **full scan.**
3) Puedes hacer uso de **tablas en memoria** para poder guardar algunos datos que duran un determinado tiempo.
4) En general podemos hacer uso de un queue server en mi aplicación.
5) **Recrear indices**: quizas nuestros indices quedaron desactualizados (queso gruyere) con muchos agujeros y los tiempos de acceso aumentaron, debemos volverlos a crear para que queden lo mas estructurado, ordenado y compacto posible.

12) ¿Que es un full scan y por que debe evitarse?

Un full scan ocurre cuando una consulta por un campo obliga al motor de la base de datos a recorrer todos los registros (filas) de una tabla (de punta a punta) para ver si se cumple una condición en algún registro .

Debe evitarse porque si tenemos una tabla con millones de datos entonces esta consulta va a tardar mucho tiempo. Además esta consumiendo mas recursos del sistema como el CPU y la memoria.

Ademas un full scan podria lockear una tabla por mucho tiempo, haciendo que otros procesos no accedan a esa tabla.

13) Explique el concepto de queso Gruyere

El concepto de queso gruyere hace referencia a que cuando nosotros por ejemplo creamos un indice para un campo en un tabla A. y luego al pasar el tiempo hacemos inserciones, deletes, modificamos la tabla A, el índice también se modifica pero este quedaria tambien con **‘agujeros’** debido a que se borraron nodos, ser insertaron nuevos, se modificaron otros,etc. Esos agujeros son espacio que no estan siendo utilizados en el indice. Por eso debemos “Recrear el índice” para que el árbol B

quede lo más estructurado, ordenado y compacto posible, además estamos también reduciendo los tiempos de acceso al recrearlo.

14) Cual es la diferencia entre "File Scan" y "Index Scan"

FileScan ocurre cuando realizamos una consulta por un campo y este obliga al motor de base de datos a recorrer todos los registros (filas) de una tabla para ver si cumple una condición en algún registro, esto se debe a que no estamos realizando una consulta por un campo indexado.

Un **Index scan**: ocurre cuando realizamos una consulta por un campo indexado (que tiene indice) por lo tanto accederemos, por lo tanto como el índice agiliza la búsqueda, obtendremos directamente los bosques donde esta la información buscada.

15) ¿Qué es un índice?

Un índice básicamente es una estructura de de datos Árbol B,B*, B^+ usada para poder agilizar la búsqueda de registros a partir de un campo o un conjunto de campos. Tenemos tres tipos índices:

1) **Índice Primario:** Básicamente el índice está construido sobre la clave primaria de la tabla, además la tabla esa ordenada físicamente por la primary key.
2) **Indice Clustering**: Básicamente el indice esta construido sobre un campo que no es la primary key, además está ordenado físicamente por ese campo. No impone una restricción de unicidad.
3) **Indice Secundario:** basicamente el indice esta construido sobre un campo que no es la primary key, y tampoco esta la tabla esta ordenado físicamente por ese campo.

Solo es posible Tener o un indice primario o de clustering, (evitar conflictos.)

16) Cuál sería una mejora eficaz para optimizar la comunicación y el rendimiento entre la aplicación y el motor de la base de datos?

Usando una server Queue mejoraría la comunicación y performance entre la aplicación y la base de datos:

1) colocando una server Queue entre el backend y la base de datos permite que todas las consultas sean atajadas por el server Queue, además esta esta diseñada para recibir muchos requerimientos y no saturar la base de datos.
2) El **server queue** a medida que recibe consulta se las va pasando a la base de datos y le dice al backend que revise periódicamente por ej la cola “B” que ahí está la respuesta de la consulta.
3) Mientras que el server queue espera, el backend continua con el flujo de programa sin bloquearse, cuando el server queue tenga la respuesta crea la cola ‘B’ y pushea ahí el resultado. El backend va a la queue y obtiene el resultado.

4) De esta manera el backend no pierde el control del programa, y la aplicación se transforma de sincrona a asíncrona.

17)¿Que es una stored procedure?

Básicamente un stored procedure es un archivo que contiene consultas SQL y que además estás pueden tener recibir argumentos (como si fueran funciones) estan almacenadas en la base de datos (asi que el analisis lexicográfico se hace solo 1 vez) listas para ser invocadas.

Las stored procedure sirven les encantan a las admón de BD porque de esta manera el desarollar no hace una consulta de sql cruda (posiblemente haga un full scan) sino que usa las stored procedure (posiblemente testeada- eficientes-probadas) de esta manera se mantiene controlado las consultas a la base de datos, el único punto de contacto serían las stored procedure.

18) ¿Qué tipos de claves puede tener una tabla y cuál es su propósito?

Podemos tener:

Primary keys: Básicamente son las claves primaria que tiene cada tabla su propósito es identificar de forma unívoca cada fila de la tabla.

Foreign key: Basicamente son claves foráneas que estan en un tabla que hace referencia a la pk que esta en otra tabla, esto nos permite establecer una relacion entre dos tablas.

Claves candidatas: Cuando en un tabla haya mas de una posible clave primaria entonces tenemos claves candidatas.

super Clave: Básicamente es una clave candidata que ademas de tener una clave candidata puede tener un campo mas y por lo tanto ahi no seria minimal.

19) ¿Qué es una transacción?

Una transacción básicamente es un conjunto de instrucciones atomicas que se ejecutan como una unidad lógica de trabajo.

Es decir si por ejemplo tenemos una transacción que consiste **en 1000 insert** si en el inser 500 fallamos vamos deshacer todos los insert que hicimos y regresar al **estado inicial** cuando no teniamos ningun insert, la transacción realiza todo o no se realiza nada.

Existen 4 propiedades deseables en las transacciones conocidas como **ACID:**

Atomicidad: Las transacciones **deben ejecutarse de manera atómica**, es decir o bien se realiza completo la transacción o no se realiza.

Consistencia: La ejecución de la transacción debe ser consistente debe preservar las reglas de negocios en todo momento.

Isolation: La ejecución de las transacciones debe ser el mismo como si se ejecutaran de forma aislada una tras otra.

Durability: Una vez que el gestor termina la transacción debe garantizar la persistencia de la misma.

20) ¿Que es un lock?
Lock es básicamente un mecanismo que se utiliza para controlar el acceso simultáneo a los recursos compartidos (un campos, filas o tablas) por parte de múltiples transacciones.
Tenemos tres tipos de locks:

1) Lock por campo: Bloqueamos unicamente un campo
2) Lock por registro:Bloqueamos toda una fila de un tabla.
3) Lock por tabla: Bloqueamos todo una tabla.

21) ¿Qué tipo de bases de datos existen?
Bases de datos relacionales: Las bases de datos relacionales usan el esquema tabular de filas y columnas para organizar los datos.

Bases de datos NoSQL: Bases de datos que no usan el esquema tabular de filas y columnas. Se creará con el objetivo de:

1) Aumentar la cantidad de consultas a la base de datos
2) Aumentar el volumen de datos
3) Poder almacenar tipos de JSON, xml, pdf, imágenes, videos.

Además las bases de datos Nosql buscan aprovechar las ventajas de los sistemas distribuidos usando un gestor de bases de datos distribuidos para que al momento de fragmentar la información en muchos nodos, este gestor nos dea la abstracción de un sistema único.

Bases de dato clave-valor:
Usa como estructura de la información un par “clave-valor”, donde la clave debe ser **única, string y comparable** (con =) y el valor puede ser de cualquier tipo. Tiene 4 operaciones básicas insertar, eliminar, modificar, buscar. No existen funciones de agregacion asi que hay que implementarlos a mano haciendo full scan o usando spark.
Ejemplos
dynamoDB (Amazon).
“PrecioProductos:milkyChocolateBlanco:Precio”:245.50
luego tenemos redis:
base de datos en memoria alta velocidad pero con una volatilidad mayor (si se apaga o se reinicia se pierden los datos)

Bases de datos wide column:
Es una evolución de las bases de datos clave-valor.
Ahora se puede agregar de forma dinámica un par clave-valor a una fila. Ahora la estructura de información sería una “fila ancha”. (wide row).
Están optimizados para la escritura.

Ejemplos:
Cassandra: gestor de base de datos de datos que usa facebook, twitter y netflix.
es un híbrido entre las bases de datos wide-column y clave-valor.
El análogo a una tabla seria un columnFamily.
Cada fila tiene dos partes:

1) la clave primaria (tiene 2 partes)
2) un conjunto de pares clave-valor (columna) que se asocia a cada fila.

No existe el concepto de join, debemos guardar los datos en una tabla desnormalizada desde un inicio.
El diseño de **cassandra esta orientado a las consultas**, primero debemos saber que consultas vamos a hacer y luego realizar el modelo lógico.

Base de datos orientada a documentos:
La estructura de información que usa documentos, un documento es par clave-valor, donde un valor puede ser otro documento.
No se pueden hacer join de manera eficiente, por lo tanto las entidades que se relacionan se agrupan para almacenarlas juntas.
Ejemplos:
MongoDB: Usa JSON para la estructura de documentos. Donde el análogo a una tabla sería una colección, una fila seria un documento, la columna sería un campo, claves son strings y usa una estructura de json. Las claves en json son siempre strings y los valores puede ser multivaluados y tipos como int, strings, objetos json, array json,etc.

Base de datos orientada a grafos
Los elementos principales son nodos y arcos. Cada nodo tiene una referencia a los nodos adyacentes estas bases de datos se usan para:

1) Modelar relaciones complejas entre entidades.
2) Relaciones entre entidades del mismo tipo.

Nos da la ventaja de resolver problemas clásicos de grafos:

1) encontrar patrones de grafos
2) encontrar un camino entre dos nodos
3) encontrar camino mínimo entre dos nodos.

Ejemplos tenemos Neo4J que usa el lenguaje cypher.

¿Ventajas de usar una base de datos distribuida? ¿conoces alguna?
Una base de datos distribuida usa gestor de bases de datos distribuidos que pueden fragmentar la información de su base de datos en distintos nodos. Las bases de datos no sql como documentos, clave-valor, wide-column, grafos nacieron distribuidas.

La ventaja que tiene es que si en caso de que se caiga un nodo, tenemos otro nodo donde podemos hacer la consulta.Además algunos nodos se pueden usar como backup. Nos permite también realizar un balanceo de carga, para que no trabaje un único nodo y muchas otras ventajas mas que nos ofrece los sistemas distribuidos.

¿Una base de datos NoSQL es una base de datos distribuida?
Si las bases de datos NoSQL son un tipo de base de datos distribuidas, están diseñadas para operar en un entorno distribuido.

¿Qué es una base de datos? ¿Qué es un gestor de base de datos?
Una base de datos es un conjunto de datos que se relacionan entre sí, los datos tienen vínculos entre ellos por lo que pertenecen a un mismo **universo-conceptual.**

Un gestor de base de datos es un software o sistema que gestiona y controla la creación, manipulación y el acceso a la base de datos.
Funciones de un gestor de base de datos:

1) Almacenamiento y consulta
2) Seguridad
3) Concurrencia
4) Recuperación
5) Administrar los usuarios del gestor de base de datos.

¿Cuándo sería apropiado utilizar una base de datos en memoria?
Son bases de datos cuyos datos están almacenados en la memoria principal, para tener tiempo de respuestas más rápido. Tienen un gran nivel de rendimiento (cantidad de operaciones de escritura/lectura) que se realizan en un periodo determinado, debido a que no vamos a disco a recuperarlo.

Se usan en caso de que queramg os reducir la latencia de nuestra aplicacion, (reducir el tiempo de consulta. También es ideal para aplicaciones que procesan muchos datos.

¿Cuál es la principal diferencia entre bases de datos relacionales y bases de datos NoSQL?
Diferencias:

	SQL	NoSQL
esquema	Tabular con filas y columnas.	No tabular, se usan par clave-valor, row colum, documentos, nodos y aristas (grafos).

Consistencia	Alta	mas relajada, baja
lenguaje de consulta	SQL	cassandra query language, mongodb, neo4j.
ACID	suele seguir el modelo ACID que garantiza la consistencia y la integridad de los datos.	Relajan las propiedades ACID a cambio de una mayor escalabilidad y flexibilidad.