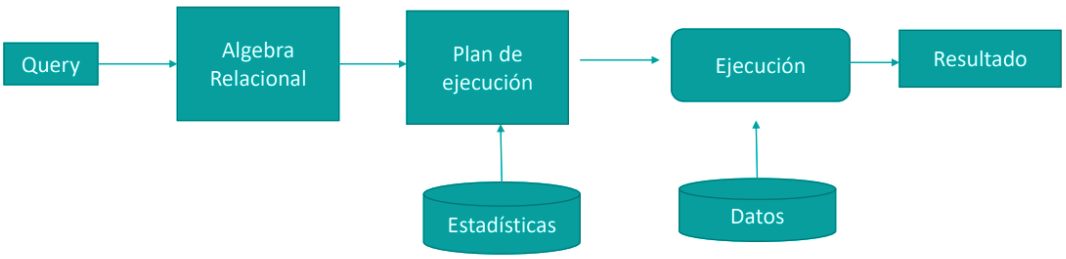
Guía 6: ✨Procesamiento y Optimización de Consultas✨



1. **Query (SQL)**
   La consulta que escribe el usuario.

2. **Álgebra Relacional (plan lógico)**
   El motor traduce la query a un plan lógico basado en operaciones de álgebra relacional:
   - selección (σ), proyección (π), join (⋈), unión, diferencia, etc.

3. **Optimizador y plan de ejecución (plan físico)**
   El optimizador toma el plan lógico y genera uno o varios **planes físicos**:
   - elegir **file scan** o **index scan**
   - elegir tipo de **join** (nested loop, hash join, merge join)
   - decidir el orden de joins, en qué momento aplicar filtros, etc.

4. **Estadísticas**
   El optimizador se apoya en estadísticas del **catálogo**:
   - cuántas filas tiene una tabla
   - cuántos bloques ocupa
   - distribución de valores, cardinalidad, etc.

5. **Ejecución**
   El motor de ejecución toma el plan físico y lo ejecuta:
   - accede a páginas en disco
   - recorre índices
   - aplica filtros
   - hace joins, agrupaciones, ordenamientos

6. **Datos y resultado**
    - La **ejecución** es la única etapa donde realmente se leen/escriben datos.
    - Finalmente se devuelven las filas pedidas al usuario.

Catálogo:

El catálogo es un conjunto de tablas internas del sistema que almacenan metadatos: nombre de las tablas, columnas, tipos, índices, claves primarias/foráneas, cantidad de filas, distribución de valores, estadísticas, permisos, vistas, etc.

- Siendo R una Relación (tabla)
- Siendo A un atributo (campo)
- n(R) cantidad de tuplas(filas)
- B(R) cantidad de bloques que ocupa la tabla en memoria
- V(A, R) cantidad de valores distintos que toma A en R.
- F(R) cantidad de tuplas de R que entran en un bloque.

$$F(R) = \frac{N(R)}{B(R)}$$

3. Tipos de Costos:

Cuando se estima el costo de un plan se tienen en cuenta, principalmente:

- **Acceso a disco** (lo más caro, lectura/escritura de bloques).
- **Red** (en sistemas distribuidos).
- **Memoria** (uso de buffers, operaciones en memoria).
- **Procesamiento (CPU)** (aplicar filtros, calcular expresiones, etc).

Regla mental: **I/O de disco suele dominar el costo** ↠ por eso se usan índices y se evita leer todo el archivo.

4. Índices

4.1 Idea básica de índice

Un **índice** es una estructura auxiliar que permite **encontrar filas rápidamente** sin leer toda la tabla.

Ejemplo conceptual:

- Tabla: Alumnos(padrón, dni, apellido, nombre, ...)
- Índice sobre dni:
    - Guarda pares (dni, posición física)
    - Permite saltar directamente a la fila en la tabla.

4.2 Tipos de índices por estructura

4.2.1 Índices asociativos (Hash)

- Usan una **función hash** sobre la clave para determinar en qué bucket va cada entrada.
- Alta performance para **búsquedas por igualdad**, por ejemplo:
  WHERE dni = 123

  WHERE email = 'x@y.com'

- Complejidad promedio de búsqueda ≈ O(1).

Limitaciones:

- No sirven bien para **rangos** (>, <, BETWEEN), porque el hash desordena los valores.
- No sirven para optimizar ORDER BY ni GROUP BY (los valores no están ordenados).
- Pueden tener **colisiones** (varias claves al mismo bucket ↠ listas/overflow).

4.2.2 Índices ordenados (B+ Tree)

- Usan **árboles B / B + balanceados**.
- Los valores de la clave están **ordenados en las hojas**.
- Permiten:
    - búsqueda por igualdad (=)
    - búsqueda por rango
    - recorridos ordenados
    - optimizar ORDER BY y GROUP BY

Complejidad: O(log n) en búsqueda/inserción/borrado.

Son los índices más usados en motores modernos (Postgres, MySQL InnoDB, SQL Server, etc.).

4.3 Tipos de índices por relación con el almacenamiento

- **Índice primario**
  - Está definido sobre el atributo (o atributos) que determina el **orden físico** del archivo y suele coincidir con la **clave primaria**.

- **Índice de clustering**
  - Ordena físicamente la tabla por un atributo que **no es clave** (puede tener repetidos).
  - Beneficia lecturas por rango sobre ese atributo.

- **Índice secundario**
  - Índice sobre un atributo que **no define el orden físico**.
  - No reordena la tabla, solo sirve para acceso rápido.

4.4 Sintaxis de creación de índices

```
CREATE [UNIQUE] INDEX nombre_indice
ON nombre_tabla (col1 [, col2, ...]);
```

- `UNIQUE` fuerza que no existan filas con la misma combinación de columnas en el índice.

5. File Scan vs Index Scan

- **File Scan**: recorre **todas** las filas de la tabla.
  - Costo ≈ **B(R)** (leer todos los bloques de la relación).

- **Index Scan**: usa un índice para llegar solo a las filas que cumplen la condición.
  - Costo ≈ **altura del índice** + **bloques con filas resultado**.

---

6. Heurísticas de optimización (álgebra relacional)

Heurísticas típicas que usa el optimizador:

- **Selección primero**: aplicar σ lo antes posible → reduce cantidad de filas. Exceptuando en el caso de que haya atributos de gran tamaño como imágenes.
- **Proyección primero**: aplicar π temprano → reduce cantidad de columnas (y tamaño de fila).
- **Reemplazar producto cartesiano por join** cuando hay condiciones de igualdad (en lugar de hacer R×S y después filtrar). Evitarlo en lo posible.
- De haber dos join, se elije el que descarte más filas.

---

7. Costos aproximados (muy resumido)

7.1 Selección (sin condición)

- **Sin índice** → costo ≈ B(R)
- **Con índice** → costo ≈
    - altura del índice + número de bloques con resultado
    - si el índice es cluster, se aprovecha mejor.

7.2 Selección por igualdad (A = cte)

- **Índice cluster:**
  $costo \approx H + \frac{N(R)}{F(R) \cdot V(A,R)}$
- Índice **secundario**:
  $costo \approx H + \frac{N(R)}{V(A,R)}$
- Índice **primario**:
  $costo \approx H + 1$

7.3 Proyección con DISTINCT

- Sin DISTINCT (o si es sobre una clave) → leer la tabla una vez: costo ≈ B(R).
- Con DISTINCT → hay que **eliminar duplicados**, típicamente:

- ordenar (sort externo) + merge
- costo ≈ `2 · B(R) · log₂(B(R)) - B(R)` (aprox).

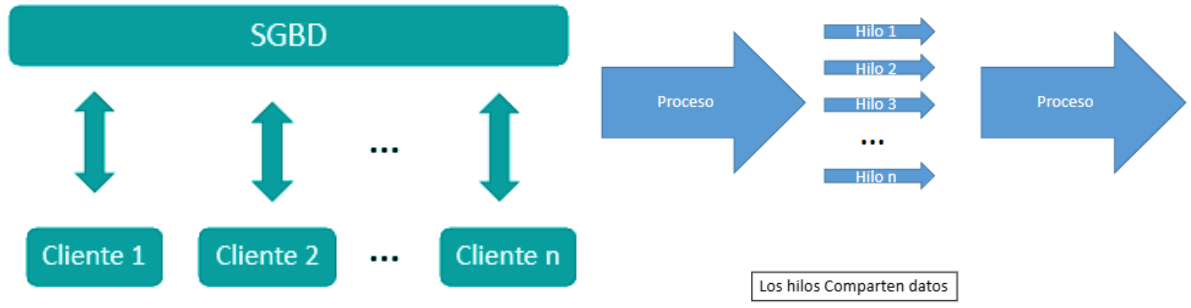
7.4 Join (nested loops y con índice)

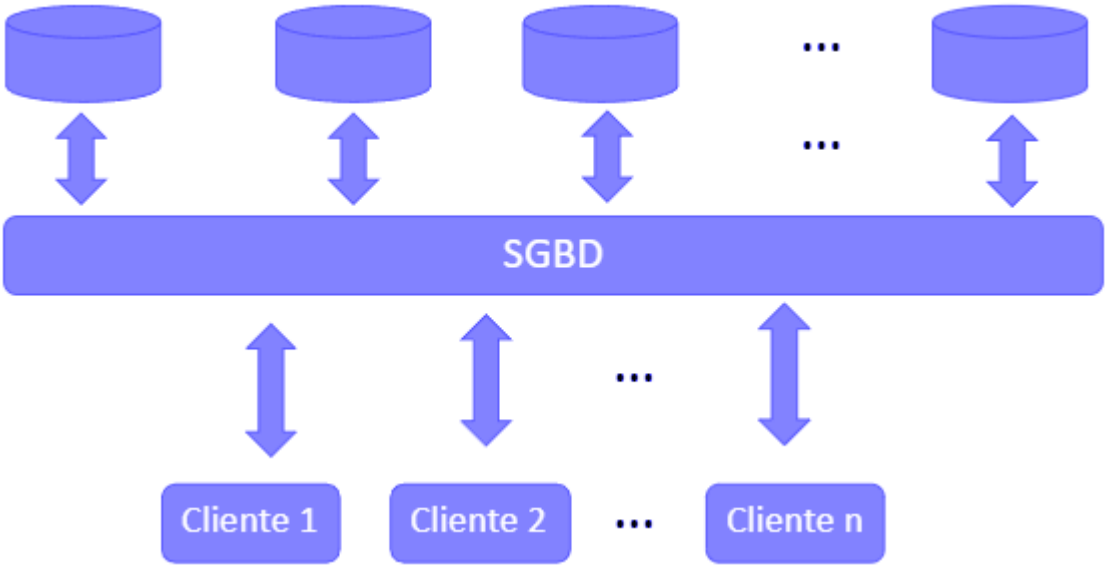
- **Nested loops por bloque entre dos tablas R y S:**
  Costo ≈ B(R)·B(S)+B(R)oB(S)·B(R)+B(S)
- **Join con un índice en la tabla interna:**
  costo ≈ `B(outer) + N(outer) · H(inner)` si se usa el índice para buscar la fila matching.

Ejercicio 1: Se tiene una tabla de clientes, una de Facturas y otra de ítems de Facturas. Es frecuente buscar a los clientes por Razón Social o Cuit para emitir facturas. También es común buscar facturas por fecha. La factura se debe poder imprimir. Definir las tablas con sus columnas y decir que índices se utilizarían, justificando. Escribir SQL- DDL

Guía 7: ✨Concurrencia, Transacciones, Recuperación✨

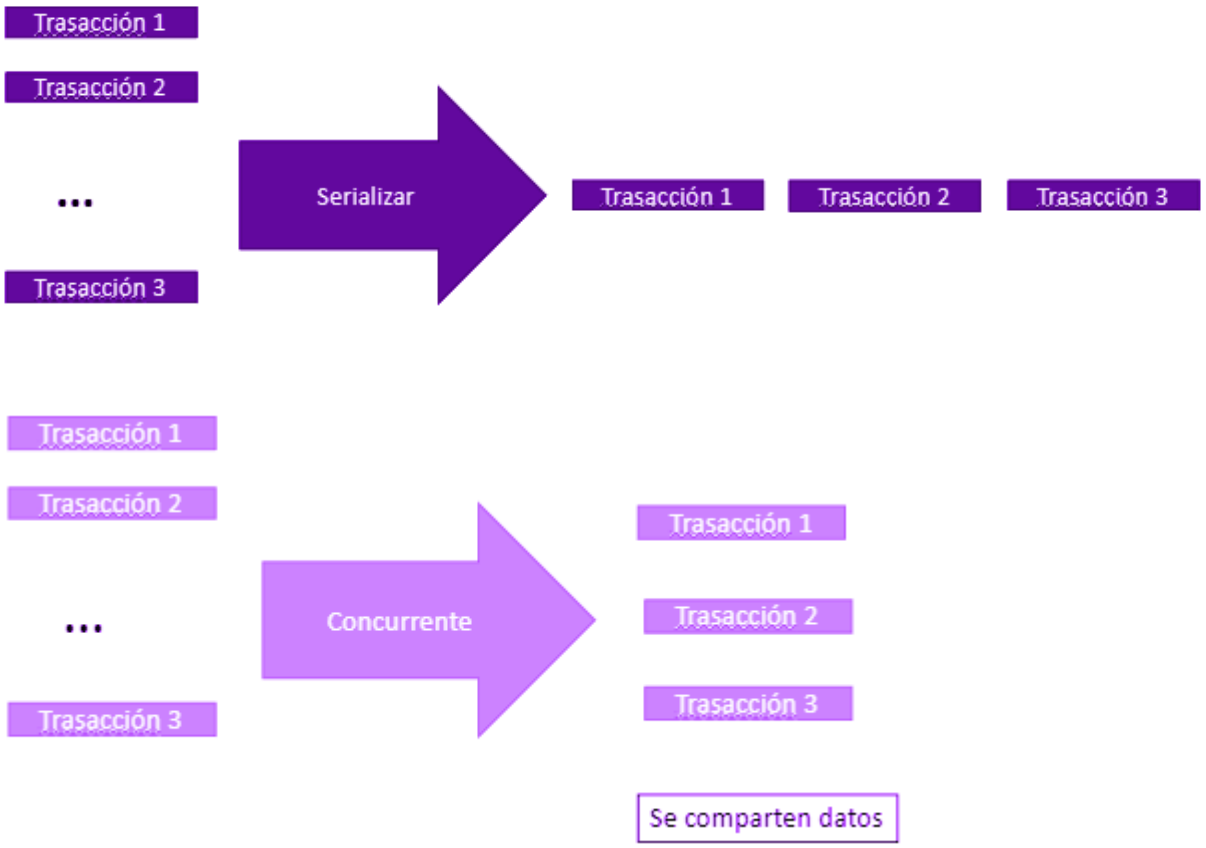
Concurrencia





Transacción: Secuencia ordenada de instrucciones atómicas.

Concurrencia: Posibilidad de ejecutar múltiples transacciones en forma simultánea (tareas).



Propiedades ACID
Atomicidad: Las transacciones deben ejecutarse de forma atómica, se ejecutan o no.
Consistencia: Cada ejecución debe preservar la consistencia de los datos.
Aislamiento (Isolation): La ejecución concurrente debe ser equivalente a una ejecución serial. Es decir, ejecutar las transacciones una detrás de otra o ejecutarlas de manera concurrente debería dar el mismo resultado.
Durabilidad: Una vez se completa una transacción, se debe garantizar la persistencia de dicha transacción aún si hay una falla.

Concurrencia solapada:
Se tiene un solo procesador.
El procesador ejecuta una sola instrucción (de una transacción) a la vez.
El scheduler puede suspender una transacción y continuar con otra.
Ninguna Transacción frena a otra.

Item:
Puede representar el valor de un atributo en una fila determinada de una tabla, una fila de una tabla, un bloque del disco, una tabla. El tamaño de este se conoce como granularidad, afecta sustancialmente al control de concurrencia.

Las propiedades se garantizan con mecanismos de recuperación. Buscan garantizar la visión de todo o nada de las transacciones. Se agregan algunas instrucciones especiales:
begin: se comenzó la transacción.
commit: se terminó con éxito.
abort: ocurrió un error y todos los efectos de la transacción deben ser deshechos. (rollback)

Anomalías:
Son situaciones que pueden violar **ACID**. Les decimos anomalía cuando ya no se puede arreglar, si se puede arreglar se llaman fenómenos.

> *Dirty Read:*
> T1 y T2 don transacciones
>
> T1 modifica X
>
> T2 lee X
>
> T1 Aborta
>
> El valor de X leído por T2 es falso

Lost Update:
T1 y T2 son transacciones

T1 lee X

T2 modifica X

T1 modifica X (se pierde la modificación de T2)

T1 vuelve a leer X (X leído es distinto del anterior)

Dirty Write:

T1 y T2 son transacciones

T1 lee X

T1 escribe X

T2 modifica X

T1 aborta

Phantom:

T1 y T2 son transacciones

T1 lee X (tabla)

T2 modifica X

T1 las condiciones pueden haber cambiado

Anomalías: Control de concurrencia
Criterio pesimista: busca garantizar que no se produzcan ciclos de conflictos.
Enfoque optimista: consiste en dejar hacer y deshacer (rollback) a las transacciones si en fase de validación se descubre un conflicto.

Seriabilidad:
Ejecución Serial: Las transacciones se ejecutan una detrás de otra, sin intercalado.

Solapado: Las transacciones se ejecutan al mismo tiempo, puede ocasionar problemas si no es controlado correctamente.

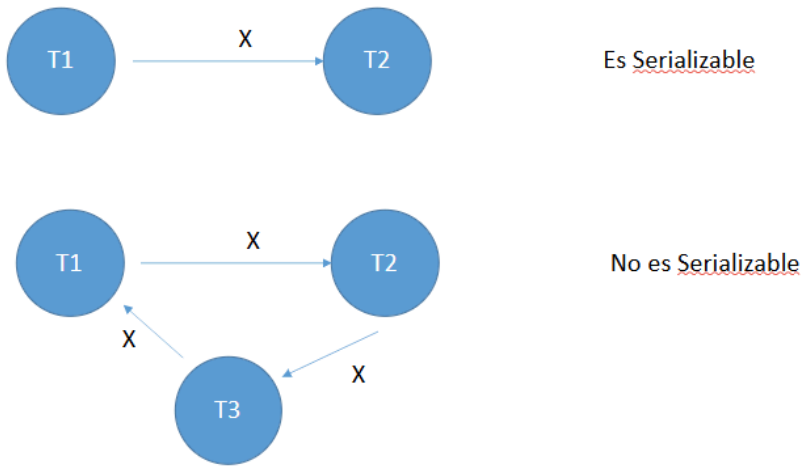
Diapositivas:
"Se respeta el orden de cada transacción": El intercalado solo puede mezclar bloques de cada transacción, pero no puede alterar el orden interno.

"Se separa en instrucciones de Read y Write": Se reducen las transacciones a Reads y Writes para facilitar el análisis de conflictos.

"Es serializable cuando es equivalente a al menos una de la serializaciones": Una ejecución solapada es serializable cuando produce el mismo resultado que una serial. De no ser serializable, hay riesgo de inconsistencias.

"Si son serialiazables ↠ están aisladas": Una transacción aislada se comporta como si fuera la única ejecutándose. Si las transacciones intercaladas producen el mismo resultado que una ejecución serial, entonces están aisladas entre sí correctamente. No se ven mutuamente ni se pisan datos de forma inconsistente.

Dado este gráfico donde cada vértice es una transacción y cada arista es un conflicto:



Notamos que si hay ciclos entre los vértices, NO es serializable.

Mecanismos de Control de Concurrencia:

Locks (enfoque pesimista)

El SGBD usa locks para bloquear el acceso a datos. La intención es evitar que dos transacciones accedan al mismo tiempo a unos mismos datos lo que puede llegar a generar inconsistencias.

Hay dos niveles de lockeo:

- Por fila: Bloquea una sola fila, mayor concurrencia, costoso si hay muchos locks para distintas filas.

- Por tabla: Bloquea toda la tabla, menor concurrencia, simple y rápido.

La sección crítica es la parte donde una transacción accede a datos compartidos. Los locks aseguran exclusión mútua, es decir, mientras una transacción tiene un lock sobre un dato, el resto de transacciones no pueden acceder a dicho dato.

Acquire (lockear): pedir un lock.

Release (unlock): liberar el lock al terminar.

Lectura:
Lock compartido
Muchas transacciones pueden leer el mismo dato a la vez.
No permite escribir.

Escritura:
Lock exclusivo
Solo 1 transacción puede tenerlo.
Nadie más puede leer ni escribir mientras esté tomado.

Los locks se desbloquean de manera inversa al orden en que se pidieron:
Lock 1
Lock 2
...
Lock n
...
Unlock n
..
Unlock 2
Unlock 1

Deadlock

Dos transacciones se quedan esperando eternamente algo que la otra no libera.

Ejemplo:

T1 tiene X y quiere Y
T2 tiene Y y quiere X
se bloquean mutuamente.

Opciones para evitar deadlocks:

1. **Prevenirlo**
    - Ordenar recursos (siempre lockear en el mismo orden).
    - Evitar ciclos.
2. **Detectarlo**
    - El SGBD mira si hay un ciclo de espera.
    - Si lo hay ↠ aborta una transacción.
3. **Timeout**
    - Si un lock tarda demasiado ↠ se asume deadlock y aborta una transacción.

Inanición (Starvation)

Una transacción espera demasiado tiempo, pero no por deadlock, sino porque otros siempre la pasan antes (nunca le toca).

Opciones para evitar inanición:

- Encolar pedidos de locks (FIFO).
- O dar prioridades.

Index Lock

Cuando se usa un índice (por ejemplo un árbol B+), también se debe bloquear:

- los nodos internos del índice
- las páginas hojas del índice
- además de la fila real

Esto evita inconsistencias mientras se navega el índice.

Lockeo de predicados (Predicate Locking)

Es un lock sobre una condición, no sobre una fila específica.

Ejemplo:
SELECT * FROM empleados WHERE sueldo > 1000

Un predicate lock bloquea todas las filas que cumplan esa condición, incluyendo filas:

- que ya existen
- y filas que podrían insertarse después y también cumplirla

Se usa para garantizar serializabilidad cuando los índices no alcanzan (muy importante para evitar phantoms).

TimeStamps (enfoque optimista)

- Número de secuencia o la fecha actual del reloj
- Deben ser únicos y determinarán el orden serial respecto al cual el solapamiento deberá ser equivalente.
- Permite que ocurran conflictos pero siempre que las transacciones de cada conflicto aparezcan de acuerdo al orden serial equivalente.

SnapShot Isolation

Lo usa postgresql. Cada transacción ve una “foto” consistente de la bdd. No ve cambios de otras transacciones activas.

Al comenzar T, se crea un snapshot con todas las versiones válidas en ese momento. T trabaja sobre esas versiones como si estuviera sola. Cuando quiere subir los cambios, el sistema verifica si hubo conflicto.

Ventajas:

- Mayor solapamiento

Desventajas:

- Requiere mayor espacio en disco o memoria
- Cuando ocurren conflictos de tipo Write - Write requiere deshacer una de las transacciones.

Anomalía Fantasma

Una anomalía fantasma ocurre cuando una transacción vuelve a ejecutar una consulta y aparecen nuevas filas que coinciden con la condición, producidas por otra transacción concurrente.

Ejemplo:

T1: SELECT * FROM empleados WHERE sueldo > 1000
T2: INSERT INTO empleados VALUES (... sueldo = 1500 ...)
T1 vuelve a ejecutar la consulta ↣ aparece un "fantasma" nuevo.

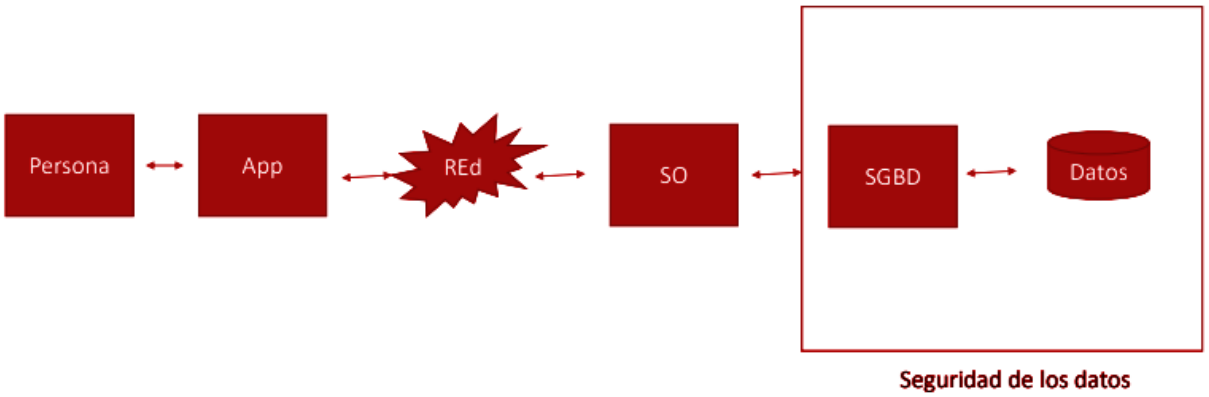
Ocurre porque no se lockean las filas que todavía no existen, solo las que coinciden con la condición en ese momento.

Por eso sirve mucho el lock de predicados (explicado un cachito más arriba) y el lock de tablas, si se bloquea la condición "sueldo > 1000" no habrá más fantasmas, tampoco los habrá si se bloquea toda la tabla.

✨Seguridad en Base de Datos✨

Seguridad de la información: Es el conjunto de procedimientos y medidas para proteger a los componentes de los sistemas de información

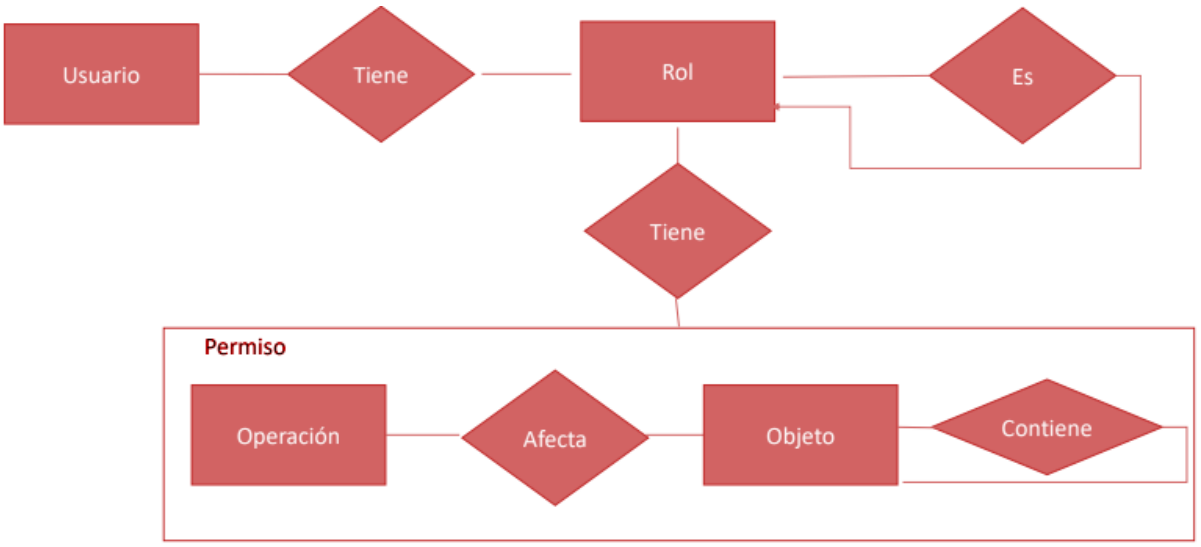
Factores:
Confidencialidad: La información no es ofrecida a individuos que no están autorizados a verla.
Integridad: Asegura la correctitud durante el ciclo de vida de la información.
Disponibilidad: Asegura que la información esté disponible cuando las personas autorizadas la requieran.
No repudio: Quien accedió a la información no puede negar haberlo hecho. Queda un registro del acceso.



Política de seguridad de datos (data security policy)

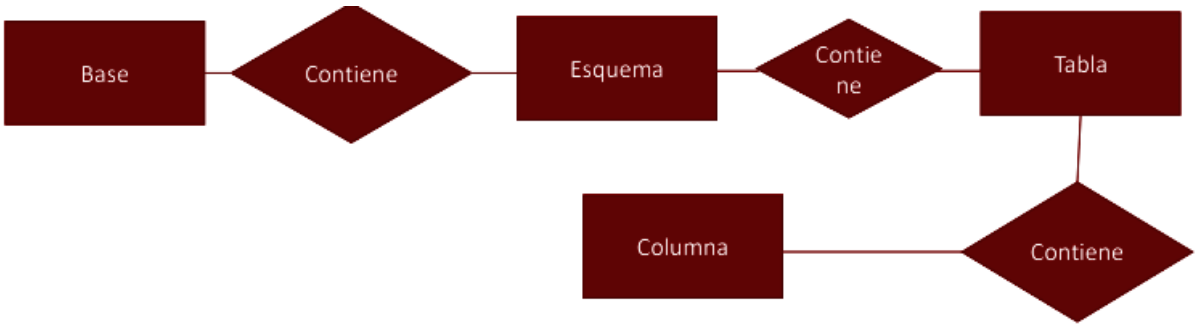
Control de acceso basado en roles (RBAC)
Define roles para las distintas actividades y funciones desarrolladas por los miembros de una organización, con el objetivo de regular el acceso de los usuarios a los recursos disponibles.

Los elementos/entidades son:
Usuarios: las personas.
Roles: conjuntos de funciones y responsabilidades.
Objetos: aquellos a proteger..
Operaciones: acciones que pueden realizarse sobre objetos.
Permisos: acciones concedidas o revocadas a un usuario o rol sobre un objeto determinado.



Soporta tres principios:

Criterio del menor privilegio posible: si un usuario no va a realizar una operación, no debería de tener los permisos para realizarla.
División de responsabilidades: nadie debe de tener suficientes privilegios para usar el sistema en beneficio propio. Necesita de roles excluyentes.
Abstracción de datos: los permisos son abstractos, dependen del objeto en cuestión.



FALTA SQL GRANTS, SQL INJECTION, ETC AUNQUE NO CREO QUE SEA IMPORTANTE :V

✨Data Warehousing✨

1. ¿Qué es un Data Warehouse?

Un Data Warehouse (DW) es una base de datos orientada a consultas analíticas, no a transacciones.
Se usa para toma de decisiones, análisis de tendencias, reportes, KPI, minería de datos, etc.

Es distinto a un sistema transaccional (OLTP).

2. OLTP Y OLAP

- **OLTP (On-line transaction processing)**
    - Datos que se generan dinámicamente
    - Capacidad transaccional (capacidad para procesar el volumen de datos)
      Arquitectura de 3 capas (imágen de la ppt):
        - *Presentación:* la interfaz en la que el usuario carga sus datos.
        - *Lógica:* es la capa intermedia, hace de servidor. Recibe la consulta y la ejecuta.
        - *Capa de datos:* los nodos de almacenamiento.

- **OLAP (On-line analytical processing)**
    - **Consultas lentas y pesadas (JOINs, agregaciones).**
    - Se tiene registro de los datos y la capacidad de procesarlos, surgió la idea de aprovecharlos para tomar decisiones.
    - Hace falta reducir la cantidad de datos y poder expresar consultas más complejas.

3. Arquitectura típica (3 capas)

Según las diapositivas:

1. **Presentación**: donde el usuario consume reportes / dashboards.
2. **Lógica de negocio**: reglas de transformación, consultas OLAP.
3. **Datos**: DW, Data Marts, staging area.

Incluye:

- Cliente
- Servidor
- Base de Datos

4. ETL (Extract - Transform - Load)

Proceso fundamental del Data Warehousing.

E: Extraer

- Toma datos desde múltiples fuentes (OLTP, archivos, APIs).

T: Transformar

- Limpieza: eliminar duplicados, corregir errores.

- Homogeneización: unificar formatos.
- Calcular campos derivados.
- Validación.

L: Load

- Insertar datos limpios al DW (generalmente en tablas de hechos y dimensiones).

5. Modelo Multidimensional

Es la **forma natural de organizar un DW**.
Incluye:x|

- **Tabla de Hechos**
- **Tablas de Dimensiones**
- **Modelo Estrella (Star Schema)**

6. Modelo Estrella (Star Schema)

- Una **tabla de hechos** central: contiene métricas cuantitativas (“ventas”, “importe”, “unidades”).
- Varias **dimensiones**: describen los hechos (tiempo, cliente, rubro, producto).

Ejemplo dado: “cantidad de ventas por cliente, mes y rubro”.

La tabla de hechos almacena **valores agregables** (SUM, COUNT), y las dimensiones categorizan esos valores.

7. Jerarquías en Dimensiones

Las dimensiones pueden tener **niveles jerárquicos**:

Ejemplo dado:

- Provincia: *Mendoza*
- Ciudades: *San Rafael*, *Luján de Cuyo*

Esto permite:

- Agregar datos en niveles superiores.
- Bajar al detalle cuando sea necesario.

8. Operaciones OLAP

1. Roll-Up (Subir jerarquía). Ej: de ciudad -> provincia -> país
2. Drill-Down(Bajar jerarquía). Ej: de año -> trimestre -> mes -> día
3. Pivoteo (cambiar punto de vista o reorientar las dimensiones) Reordena el cubo para observar otra perspectiva.
4. Slicing (Fijar una dimensión en un valor)
5. Dicing (Fija n dimensiones a la vez)

9. Mantenimiento de Cubos (Recalculo)

- Lazy -> El cubo se recalcula solo cuando alguien lo consulta
- Periódico -> se recalcula cada cierto tiempo
- Forzado por cantidad de cambios -> Si el DW se modificó mucho desde el último recálculo se vuelve a refrescar

PARCIAL

¿CÓMO AFECTAN A LAS OPERACIONES?

✔ CONSULTA (SELECT)

- **No modifica nada**, solo está limitada por valores válidos del dominio (si consultás por algo inexistente, no trae filas).

✔ INSERTAR (INSERT)

- **Dominio:** no te deja insertar valores inválidos.
- **Unicidad:** no te deja repetir claves únicas.
- **Entidad:** la PK no puede ser NULL.
- **Referencial:** no te deja insertar una FK que no exista en la tabla padre.

✔ MODIFICAR (UPDATE)

- **Dominio:** no podés cambiar un valor a uno inválido.
- **Unicidad:** no podés actualizar a un valor ya existente si es unique.
- **Entidad:** no podés poner PK en NULL.
- **Referencial:** si cambiás una FK, debe apuntar a una fila válida.

✔ ELIMINAR (DELETE)

- **Dominio** / **Unicidad** / **Entidad:** no afectan.
- **Referencial:** si BORRÁS un padre que tiene hijos, depende de la acción definida:

Acciones sobre integridad referencial:

- **RECHAZAR (RESTRICT / NO ACTION):**
  No permite borrar el padre si tiene hijos.
- **PROPAGAR (CASCADE):**
  Borra automáticamente los hijos.
- **NULLEAR (SET NULL):**
  Pone en NULL las FKs de los hijos.