IIC2413

AYUDANTÍA I1

Contenidos

- ÁLGEBRA RELACIONAL
- MODELO ENTIDAD/RELACIÓN
- FORMAS NORMALES
- SQL
- LÓGICA EN LA BASE DE DATOS
- TRANSACCIONES Y LOCKS

ALGEBRA RELACIONAL

ÁLGEBRA RELACIONAL

- Actores(id, nombre, edad)
- Películas(id, título, director)
- ActúaEn(idActor, idPelícula)

Símbolo	Operación	Qué hace	Ejemplo
π	Proyección	Selecciona columnas (atributos).	$\pi_{nombre}(Actores)$
σ	Selección	Filtra filas (tuplas) por condición.	$\sigma_{edad<30}(Actores)$
$\cup$	Unión	Une dos relaciones con el mismo esquema.	$\pi_{nombre}(Actores) \cup \pi_{director}(Peliculas)$
$-$	Diferencia	Devuelve tuplas en una relación que no están en la otra.	$\pi_{nombre}(Actores) - \pi_{nombre}(Actores \bowtie ActuaEn)$
$\cap$	Intersección	Devuelve tuplas que están en ambas relaciones.	$\pi_{nombre}(Actores) \cap \pi_{director}(Peliculas)$
$\times$	Producto cruz	Combina todas las tuplas de una relación con todas las de otra.	$Actores \times Peliculas$
$\bowtie$	Join (con condición)	Une dos relaciones por un atributo clave.	$Actores \bowtie_{id=idActor} ActuaEn$
$\bowtie$ (natural)	Natural Join	Une por atributos con el mismo nombre (sin condición explícita).	$Actores \bowtie ActuaEn$
ρ	Renombrar	Cambia nombre de atributos o de la relación.	$\rho((nombre \rightarrow actorNombre), Actores)$

Notas: π proyecta atributos; σ aplica predicados; $\cup, \cap, -$ requieren esquemas compatibles; $\times$ es producto cartesiano; $\bowtie$ con subíndice indica condición ($\bowtie_\varphi$); el natural join usa coincidencia de nombres; ρ renombra relación y/o atributos.

Pregunta 1 - Álgebra Relacional

Sean las siguientes relaciones, que contienen información sobre **Jugadores**, **Rankings**, **Torneos** e **Inscripciones** de tenis. A continuación se muestran **ejemplos** de tuplas en las 4 tablas:

id_jugador	nombre	pais	edad
1	Novak Djokovic	Serbia	37
2	Carlos Alcaraz	España	21
3	Alejandro Tabilo	Chile	27
...	...	...	...
n	Luca Fernandini	Italia	23

Cuadro 1: Jugadores

id_jugador	ranking_singles	ranking_dobles
1	5	491
2	3	519
3	32	118
...	...	...
n	1	164

Cuadro 2: Rankings

id_torneo	nombre_torneo	ciudad	superficie	puntos
1	Wimbledon	Londres	pasto	2000
2	Roland Garros	Paris	arcilla	2000
3	Indian Wells	California	cemento	1000
...	...	...	...	...
m	Chile Open	Santiago	arcilla	250

Cuadro 3: Torneos

id_inscripcion	id_jugador	id_torneo	fecha_inscripcion
1	1	1	03-06-2025
2	3	1	12-06-2025
3	3	3	09-08-2025
...	...	...	...
k	n	m	31-01-2025

Cuadro 4: Inscripciones

Escriba las siguientes operaciones en álgebra relacional, utilizando la sintaxis vista en clases (puede encontrarla en el torpedo):

Hint: renombre las relaciones a J, R, T, I, respectivamente, con el operador correspondiente.

a) (3 pts) Liste el nombre y país de los jugadores mayores de 35 años con un ranking en dobles menor a 50.

b) (4 pts) Liste el nombre y fecha de inscripción de todos los jugadores que estén inscritos en torneos realizados en la ciudad de París.

a) (3 pts) Liste el nombre y país de los jugadores mayores de 35 años con un ranking en dobles menor a 50.

Solución:

Se renombran las relaciones por comodidad:

$$\rho(J,\ Jugadores), \quad \rho(R,\ Rankings), \quad \rho(T,\ Torneos), \quad \rho(I,\ Inscripciones)$$

$$\pi_{nombre_pais}\left(\sigma_{edad>35 \wedge ranking_dobles<50}\left(J \bowtie_{J.id_jugador=R.id_jugador} R\right)\right)$$

b) (4 pts) Liste el nombre y fecha de inscripción de todos los jugadores que estén inscritos en torneos realizados en la ciudad de París.

Solución:

$$\pi_{nombre, fecha_inscripcion}\left(J \bowtie_{J.id_jugador=I.id_jugador} I \bowtie_{I.id_torneo=T.id_torneo} \left(\sigma_{ciudad='Paris'} T\right)\right)$$

c) (4 pts) Liste la ciudad de los torneos en los que ningún jugador chileno se haya inscrito.

c) (4 pts) Liste la ciudad de los torneos en los que ningún jugador chileno se haya inscrito.

Solución:

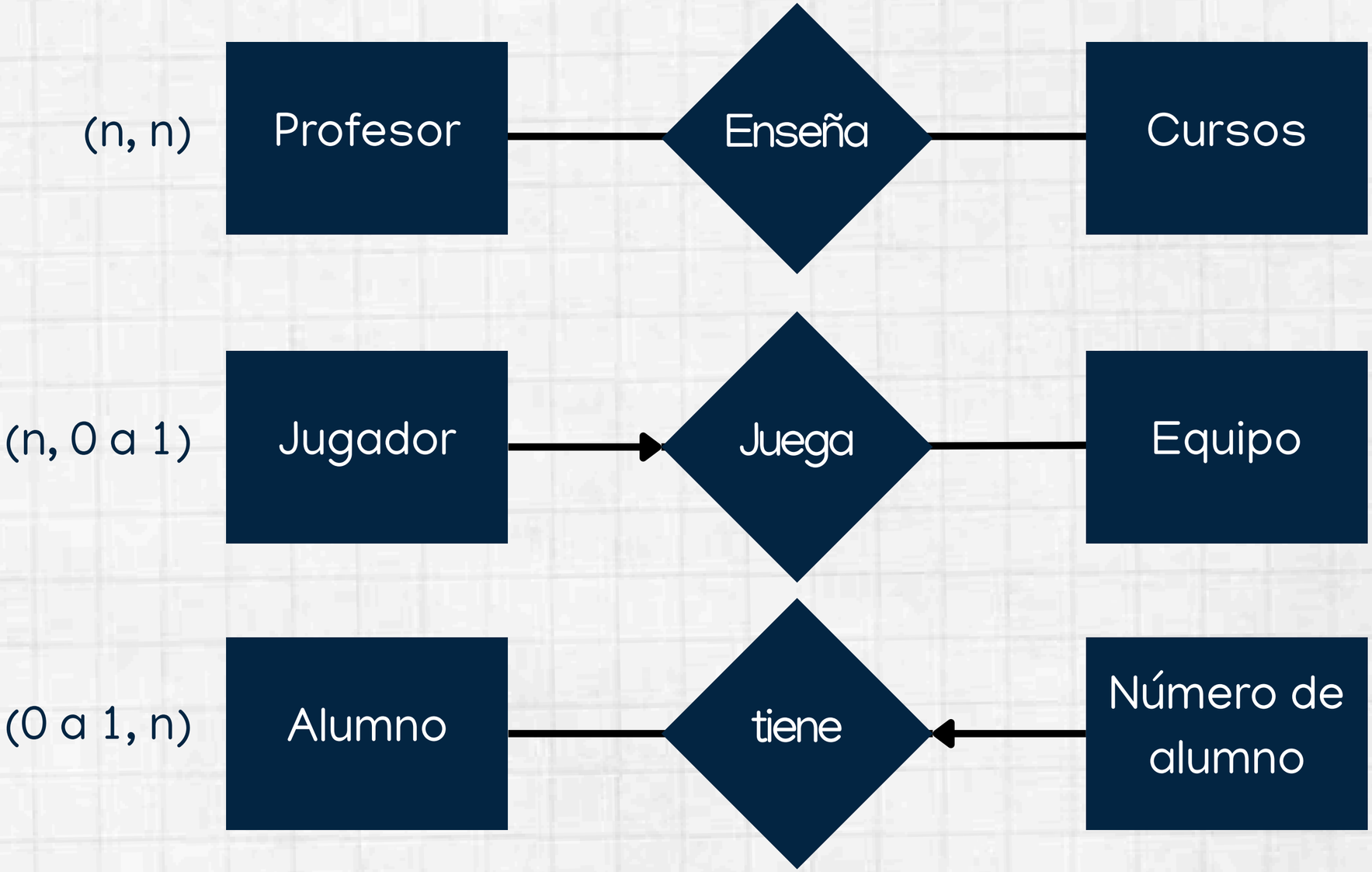
$$\rho(Chilenos,\ \sigma_{pais='Chile'}(J))$$

$$\rho(TorneosConChilenos,\ \pi_{id_torneo}\,(Chilenos \bowtie_{Chilenos.id_jugador=I.id_jugador} I))$$

$$\pi_{ciudad}(T) - \pi_{ciudad}(T \bowtie_{T.id_torneo=TorneosConChilenos.id_torneo} TorneosConChilenos)$$

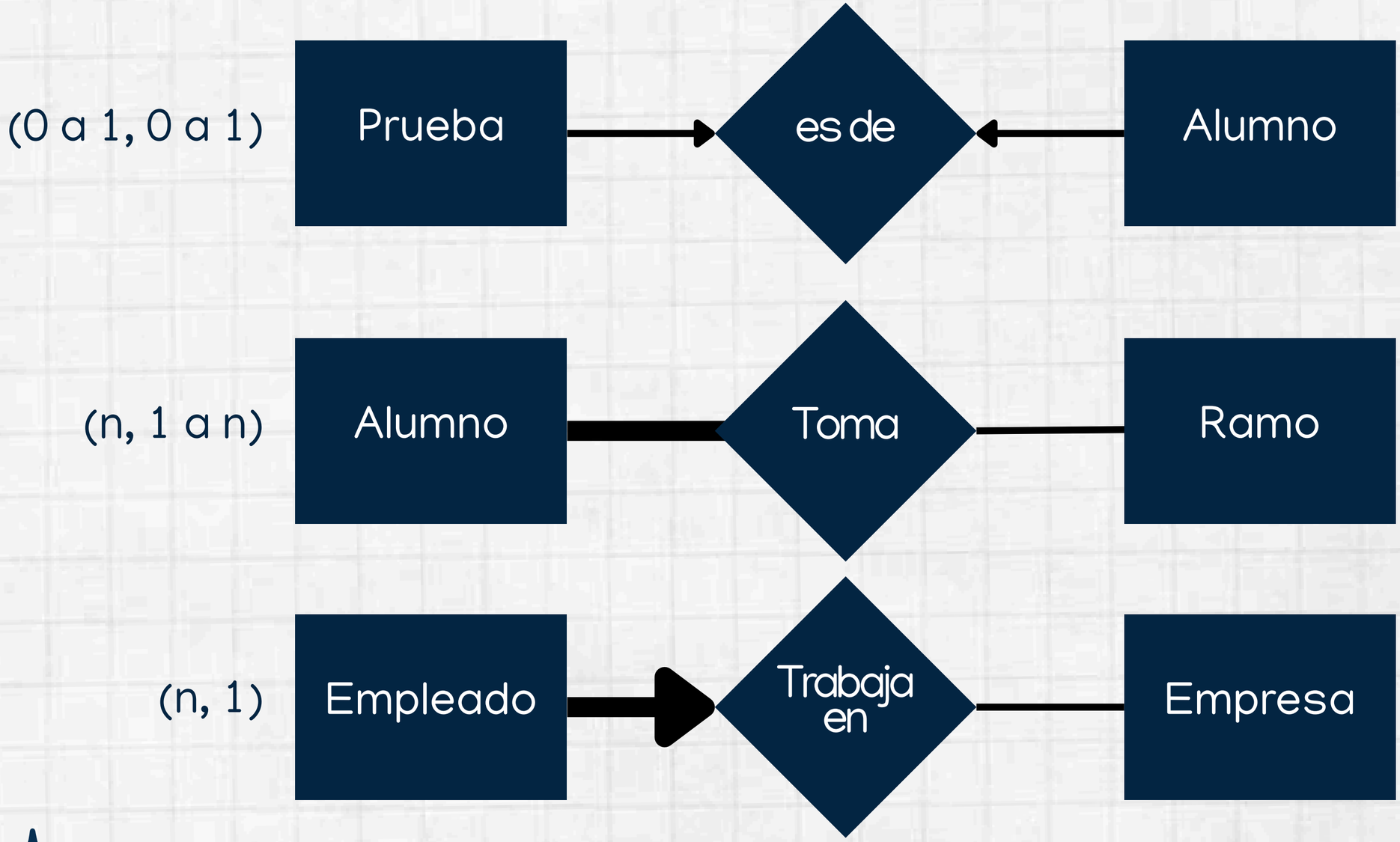
MODELO ENTIDAD/RELACIÓN

RELACIONES BINARIAS



- (n, n): N profesor enseña en N cursos. N curso es enseñado por N profesores.
- (n, 0 a 1): N jugadores juegan por 0 o 1 equipo. En 1 equipo juegan N jugadores
- (0 a 1, n): 0 a 1 alumno tiene N num de alumnos. 1 num de alumno es de 0 a 1 alumno.

RELACIONES BINARIAS

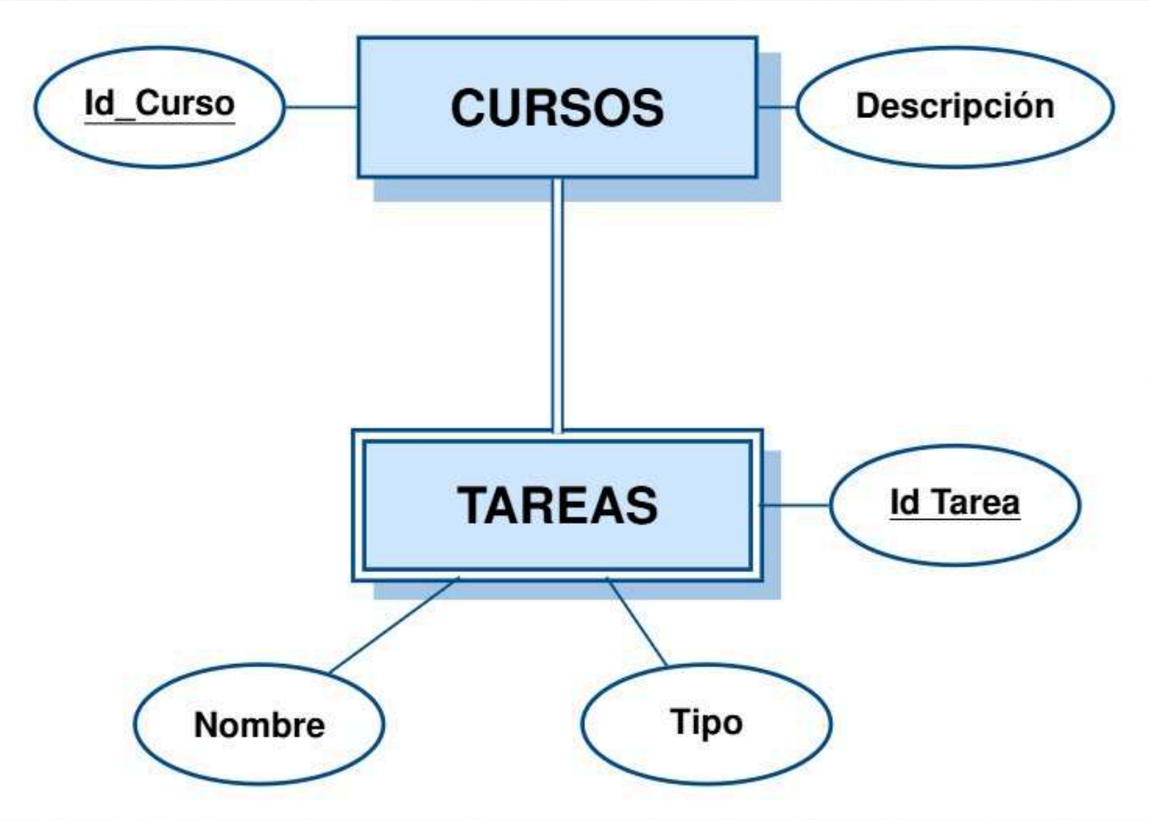


- (0 a 1, 0 a 1) 0 a 1 prueba es de 0 a 1 alumno, y viceversa
- (n, 1 a n): N alumnos toman 1 a N ramos (no pueden tomar 0 ramos)
- (n, 1): N empleados trabajan en 1 empresa

BIENVENIDOS AL

MODELO RELACIONAL

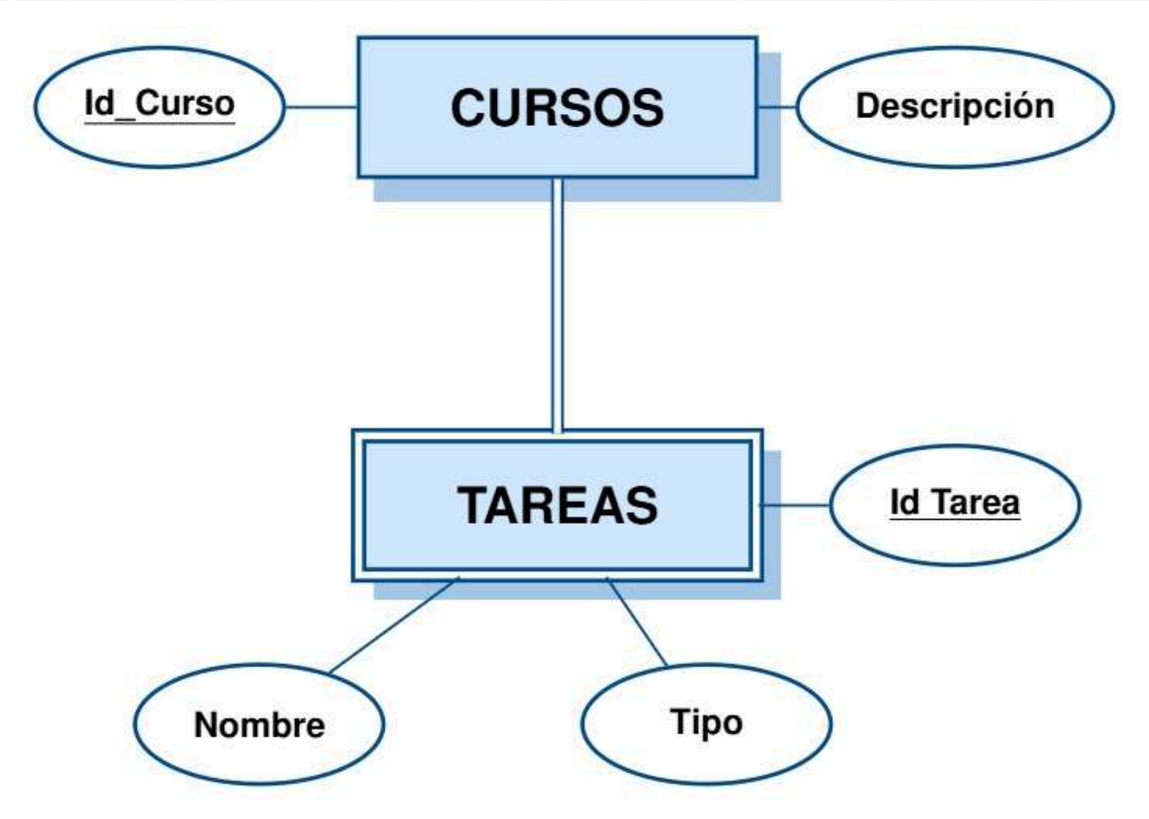
PASO A ESQUEMA



1. Entidades fuertes → tablas

2. Entidades débiles → tablas (incluye claves foráneas de las entidades padres)

PASO A ESQUEMA



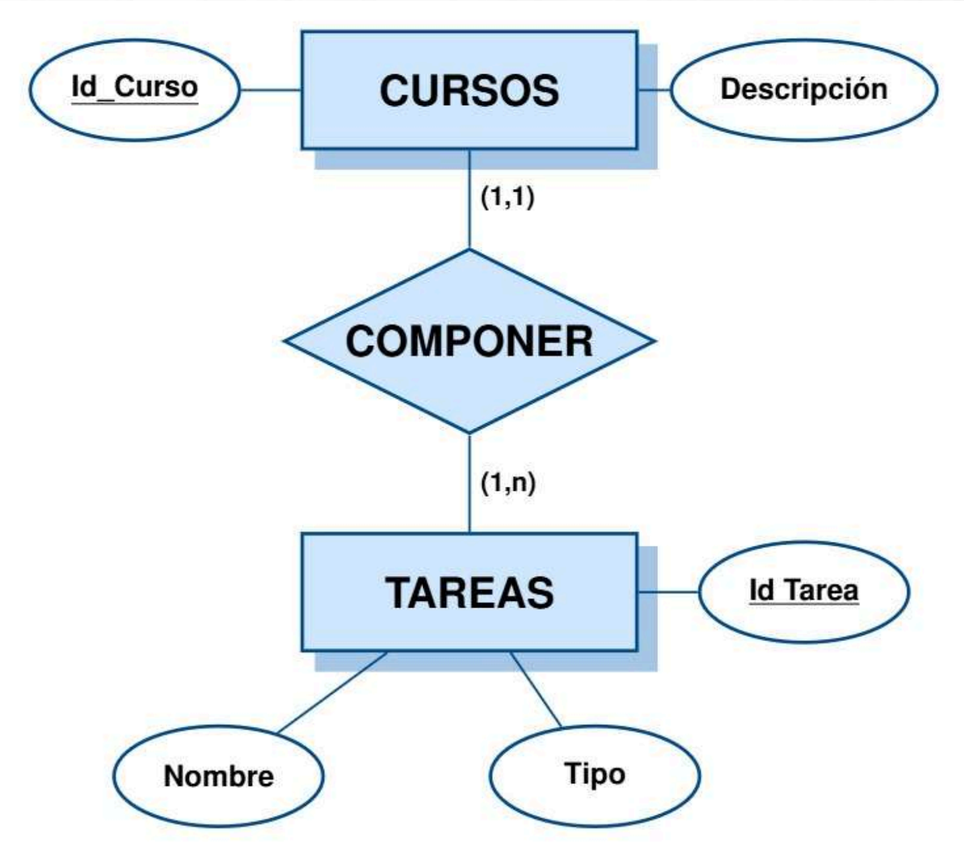
1. Entidades fuertes → tablas

2. Entidades débiles → tablas (incluye claves foráneas de las entidades padres)

Cursos (Id_Curso, Descripción)

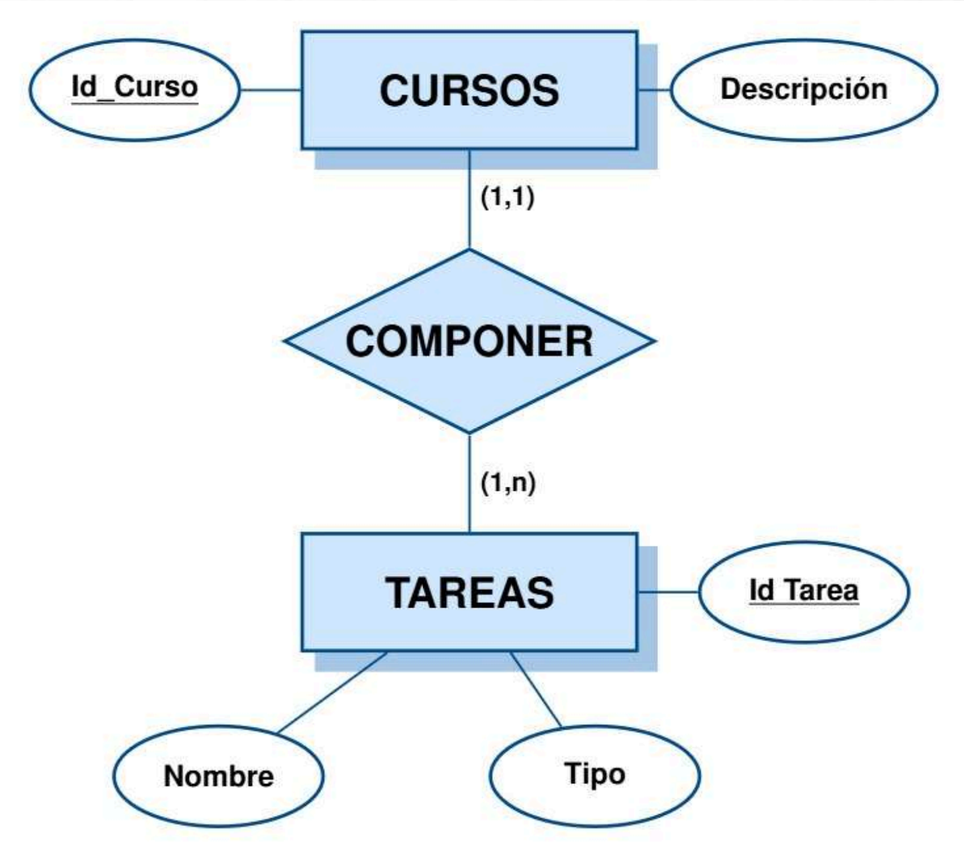
Tareas (Id_Tarea, Id_Curso , Nombre, Tipo)

PASO A ESQUEMA



3. Relaciones 1:N sin atributos→ clave foránea en la tabla "muchos"

PASO A ESQUEMA

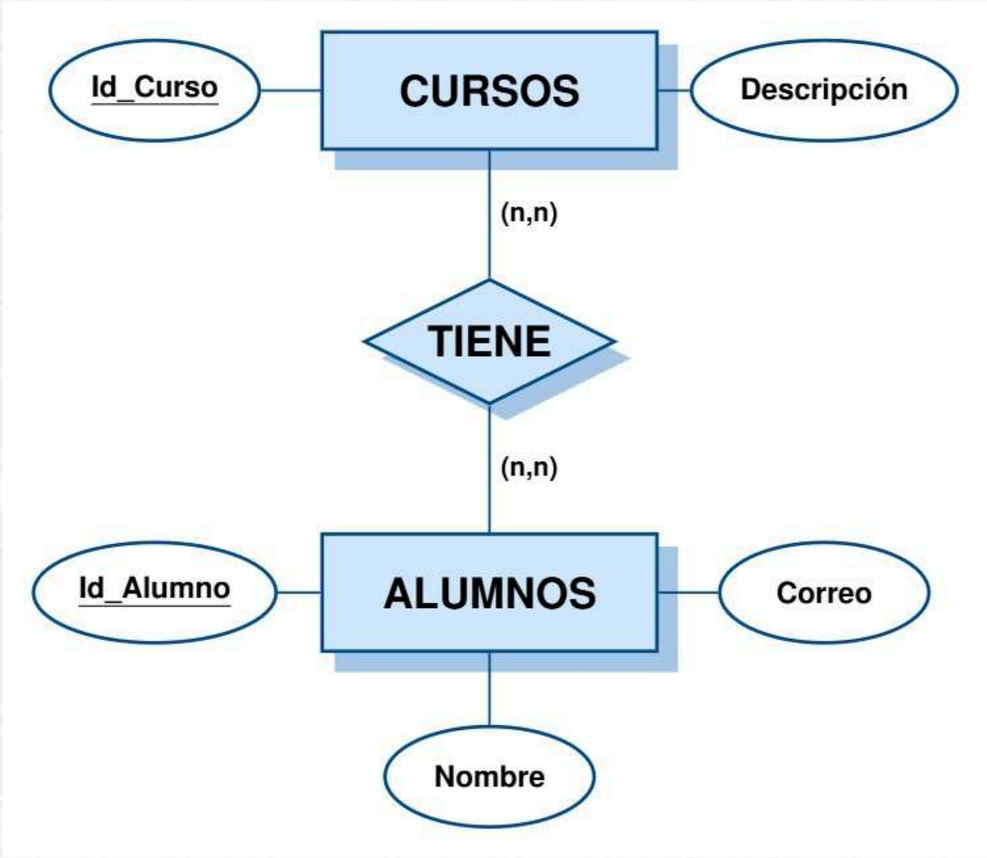


3. Relaciones 1:N sin atributos→ clave foránea en la tabla "muchos"

Cursos(Id_Curso, Descripción)

Tareas(Id_Tarea , Nombre, Tipo, Id_Curso FK)

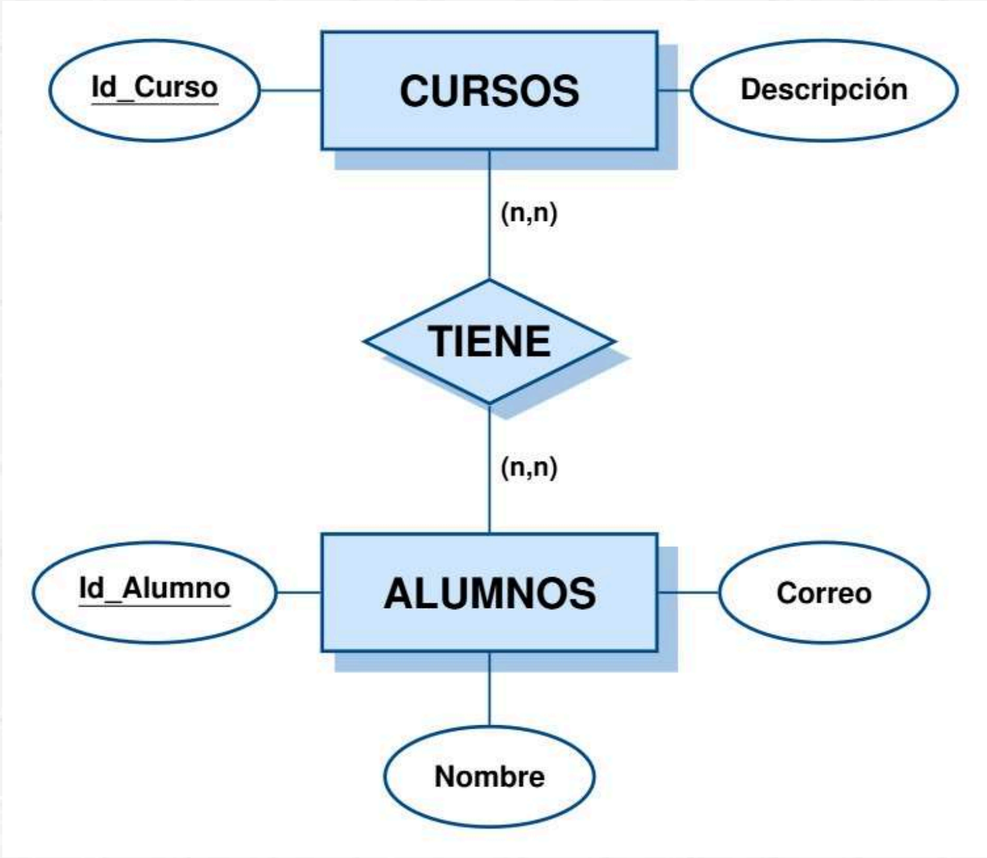
PASO A ESQUEMA



4. Relaciones N:M con o sin atributos →

tabla intermedia que incluye llave foránea de las entidades relacionadas

PASO A ESQUEMA



4. Relaciones N:M con o sin atributos →

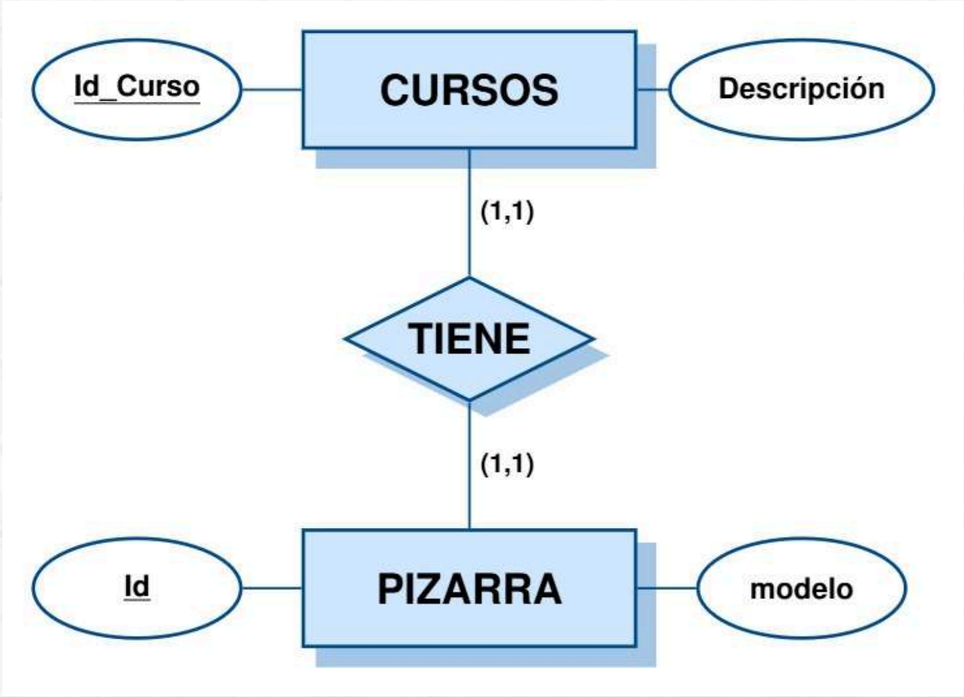
tabla intermedia que incluye llave foránea de las entidades relacionadas

Cursos(Id_Curso, Descripción)

Alumnos(Id_Alumno, Nombre,)

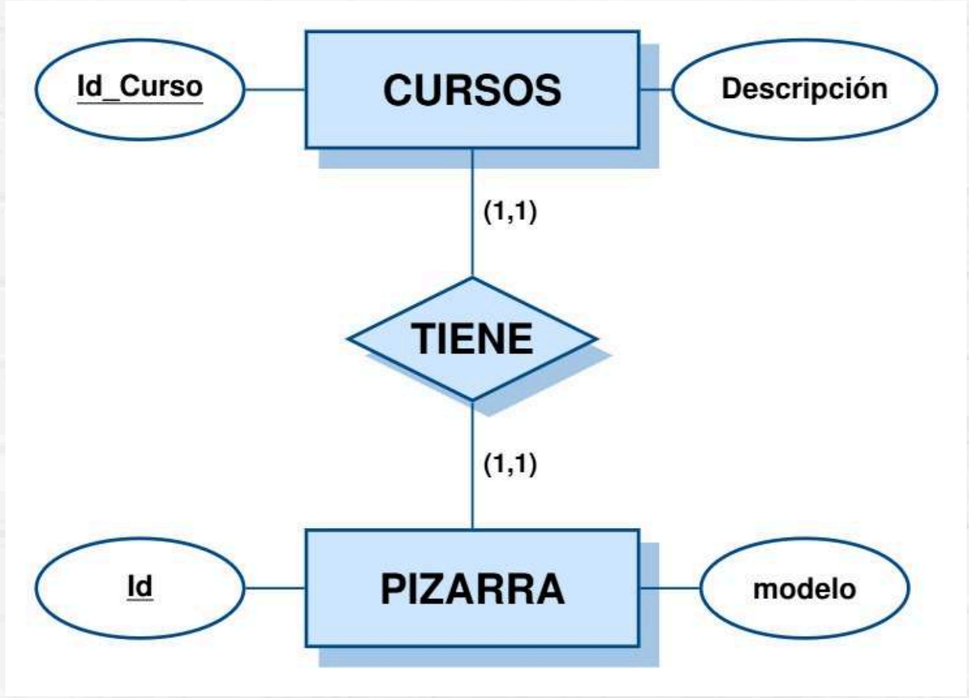
Tiene (Id_Curso, Id_Alumno)

PASO A ESQUEMA



5. Relaciones 1:1 sin atributos → Cualquiera de las dos entidades absorbe llave foránea de la otra entidad

PASO A ESQUEMA



5. Relaciones 1:1 sin atributos → Cualquiera de las dos entidades absorbe llave foránea de la otra entidad

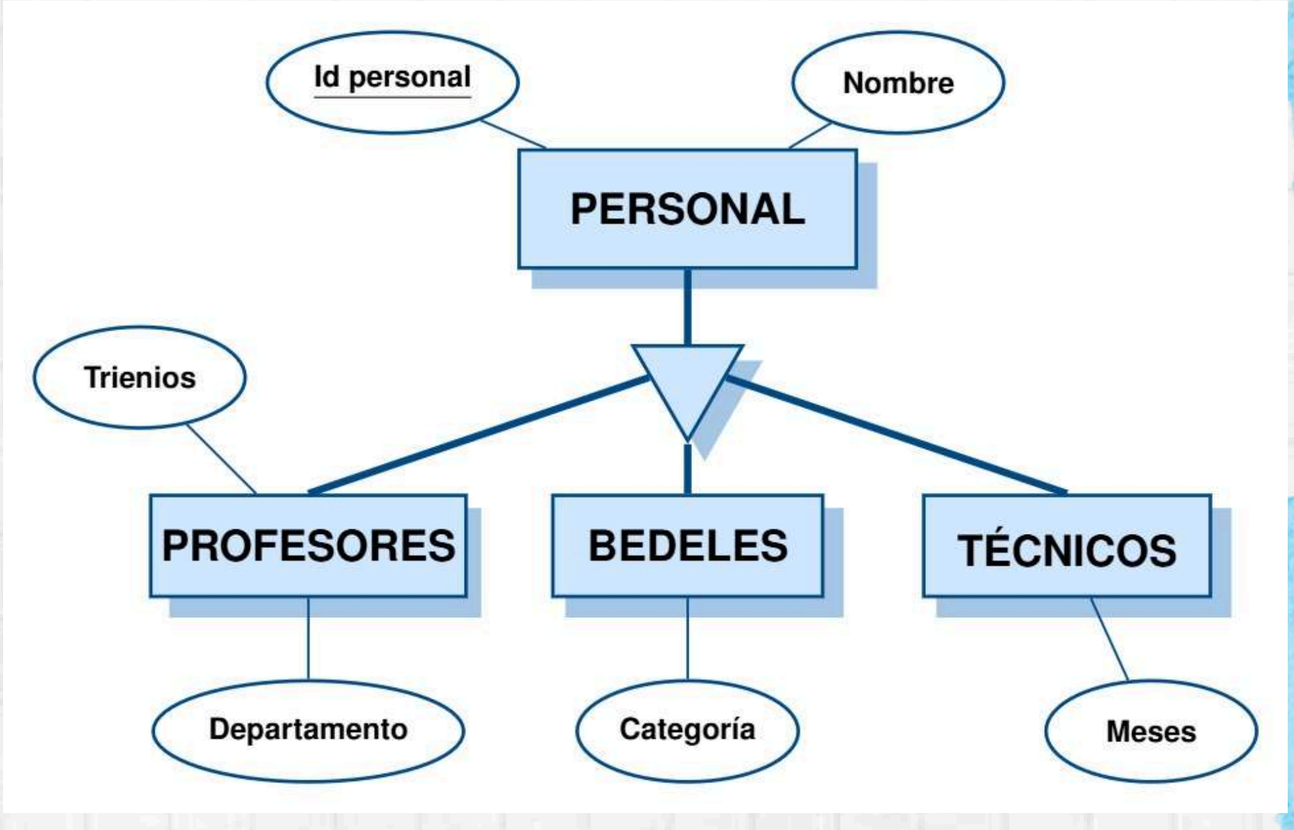
Cursos(Id_Curso, Descripción)

Pizarra(Id , modelo, Id_Curso FK)

Cursos(Id_Curso, Descripción, Id_Pizarra FK)

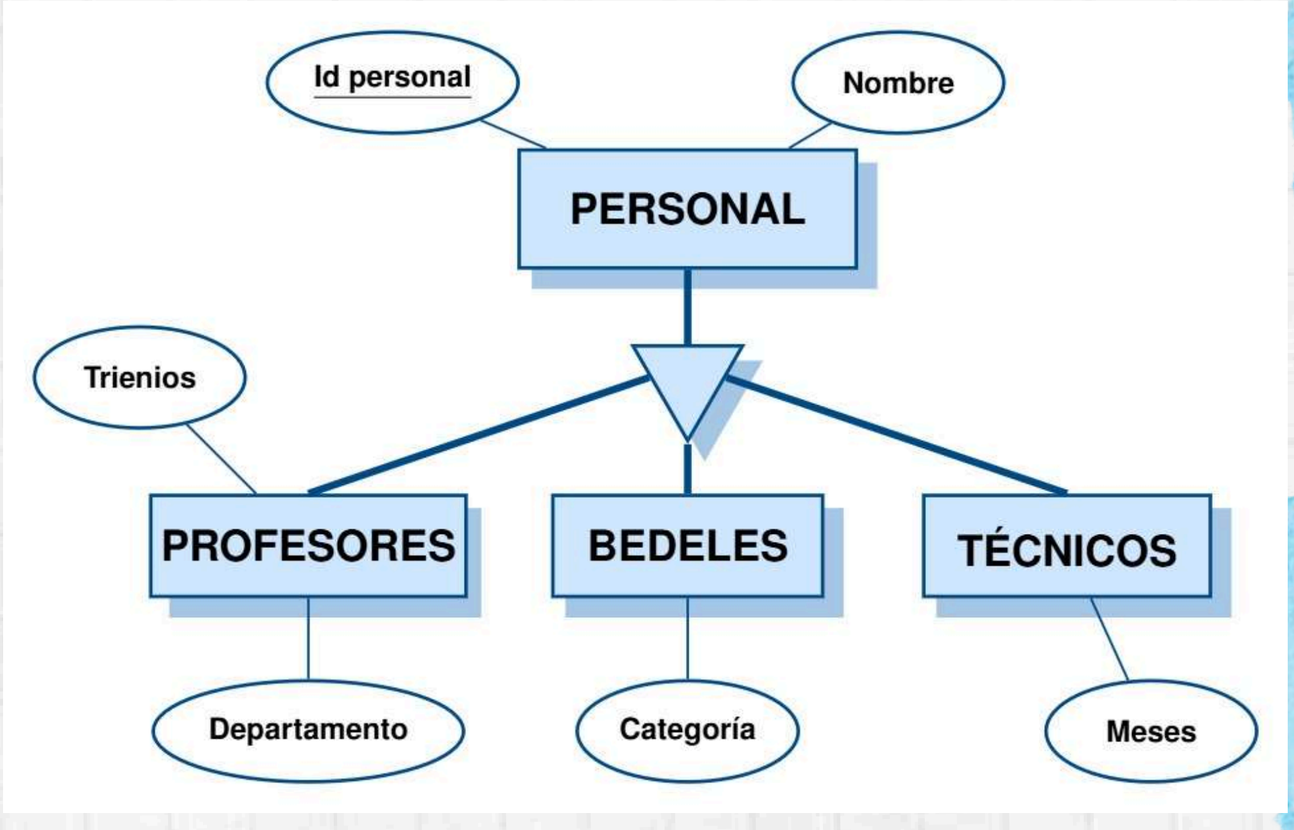
Pizarra(Id , modelo)

PASO A ESQUEMA



6. Relaciones “IsA” → Heredar id del padre.

PASO A ESQUEMA



6. Relaciones “IsA” → Heredar id del padre.

Personal (Id_personal, Nombre)

Profesores (Id_personal, Trienios, Departamento)

Bedeles (Id_personal, Categoría)

Técnicos (Id_personal, meses)

RESUMEN PASO A TABLAS

1. Entidades fuertes → tablas
2. Entidades débiles → tablas (incluyendo claves foráneas de las entidades fuertes)
3. Relaciones 1:N sin atributos→ clave foránea en la tabla "muchos"
4. Relaciones N:M con o sin atributos→ tabla intermedia que incluye llave foránea de las entidades relacionadas.
5. Relaciones 1:1 sin atributos → cualquiera de las dos entidades absorbe llave foránea de la otra entidad
6. Relaciones “IsA” → Heredar id del padre.

Pregunta 2 - Diagrama E/R y Esquema Relacional

Considere la red social de música DCCanciones (inspirada en Spotify), en la que los usuarios pueden seguir a otros usuarios y también interactuar con contenido musical.

Cada usuario se identifica por un `id_usuario` y tiene atributos nombre, usuario, correo electrónico, canción favorita, y tipo. Los tipos de usuario pueden ser básico o premium. Los usuarios pueden seguir a otros usuarios dentro de la plataforma. Para esto, deben enviar una solicitud de seguimiento a otro usuario, la cual puede ser aceptada o rechazada posteriormente. Esta se registra con una fecha en la que se realizó la solicitud, y su estado actual.

Los usuarios pueden crear listas de reproducción (playlists), las que contienen canciones. Cada lista de reproducción tiene un nombre, una serie de canciones, una fecha de creación y está asociada a un único creador (usuario).

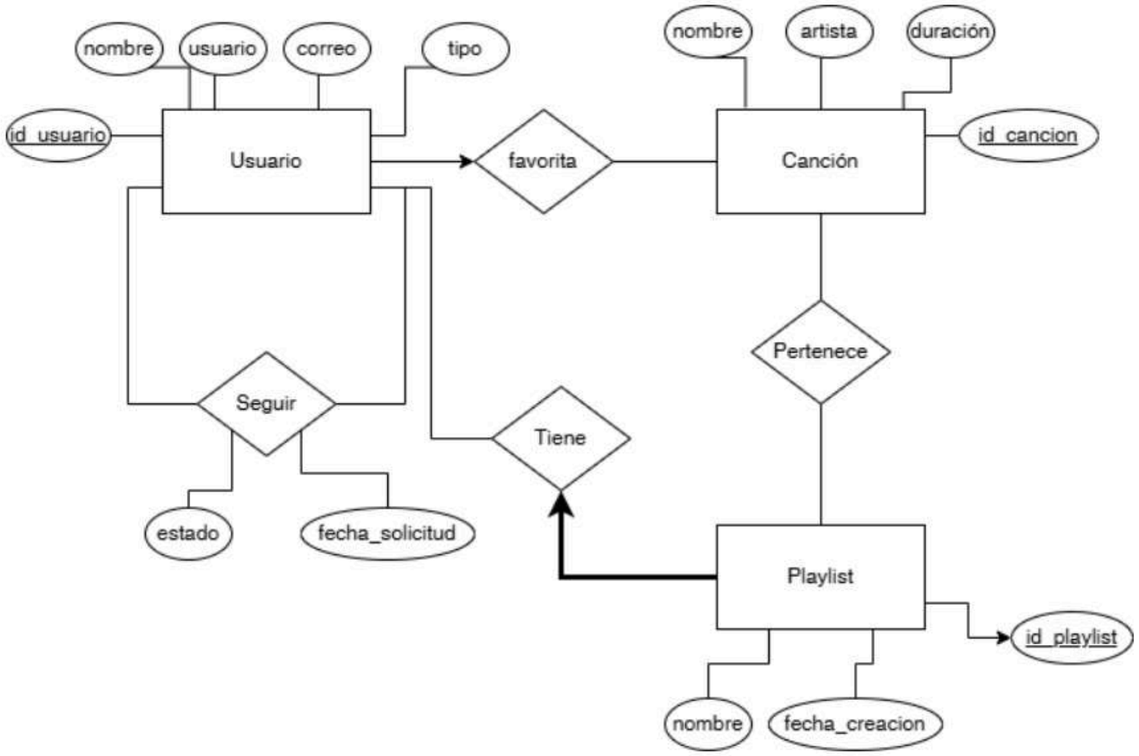
Las canciones están identificadas por un `id_cancion` y tienen atributos nombre, artista y duración. Una misma canción puede estar incluida en múltiples playlists.

a) (10 pts) Diseñe el Diagrama Entidad/Relación de DCCanciones. Identifique claramente las entidades, entidades débiles, relaciones, cardinalidades, atributos, y llaves primarias y parciales.

b) (5 pts) A partir del Diagrama E/R propuesto, genere el Esquema Relacional asociado.

a) (10 pts) Diseñe el Diagrama Entidad/Relación de DCCanciones. Identifique claramente las entidades, entidades débiles, relaciones, cardinalidades, atributos, y llaves primarias y parciales.

Solución:



b) (5 pts) A partir del Diagrama E/R propuesto, genere el Esquema Relacional asociado.

```
CREATE TABLE Usuario (
    id_usuario INT PRIMARY KEY,
    nombre VARCHAR(50),
    usuario VARCHAR(30),
    correo VARCHAR(100),
    cancion_favorita INT,
    tipo VARCHAR(10),
    FOREIGN KEY (cancion_favorita) REFERENCES Cancion(id_cancion)
);

CREATE TABLE Cancion (
    id_cancion INT PRIMARY KEY,
    nombre VARCHAR(100),
    artista VARCHAR(100),
    duracion INT -- duración en segundos
);

CREATE TABLE Playlist (
    id_playlist INT PRIMARY KEY,
    nombre VARCHAR(100),
    fecha_creacion DATE,
    creador INT,
    FOREIGN KEY (creador) REFERENCES Usuario(id_usuario)
);
```

```
CREATE TABLE Seguir (
    id_origen INT,
    id_destino INT,
    estado VARCHAR(10), -- pendiente, aceptada, rechazada
    fecha_solicitud DATE,
    PRIMARY KEY (id_origen, id_destino),
    FOREIGN KEY (id_origen) REFERENCES Usuario(id_usuario),
    FOREIGN KEY (id_destino) REFERENCES Usuario(id_usuario)
);

CREATE TABLE Pertenece (
    id_playlist INT,
    id_cancion INT,
    PRIMARY KEY (id_playlist, id_cancion),
    FOREIGN KEY (id_playlist) REFERENCES Playlist(id_playlist),
    FOREIGN KEY (id_cancion) REFERENCES Cancion(id_cancion)
);
```

BIENVENIDOS A

FORMAS NORMALES

1NF

- Cada columna debe tener solo un valor, no una lista

Ejemplo tabla no normalizada:

id_alumno	nombre	curso
1	Juan	Matemáticas, Física
2	María	Química

1NF

- Cada columna debe tener solo un valor, no una lista

Ejemplo tabla no normalizada:

id_alumno	nombre	curso
1	Juan	Matemáticas, Física
2	María	Química

Ejemplo tabla normalizada (1NF):

id_alumno	nombre	curso
1	Juan	Matemáticas
1	Juan	Física
2	María	Química

2NF

- Debe estar en 1NF
- Cada columna debe depender de la clave primaria completa, no solo de una parte

Ejemplo tabla no normalizada (2NF):

ID_Alumno	ID_Curso	Nombre_ Alumno	Nombre_ Curso	Profesor
1	101	Juan	Matemáticas	Prof. Pérez
1	102	Juan	Física	Prof. López
2	103	María	Química	Prof. Díaz

2NF

- Debe estar en 1NF
- Cada columna debe depender de la clave primaria completa, no solo de una parte

Ejemplo tabla no normalizada (2NF):

ID_Alumno	ID_Curso	Nombre_ Alumno	Nombre_ Curso	Profesor
1	101	Juan	Matemáticas	Prof. Pérez
1	102	Juan	Física	Prof. López
2	103	María	Química	Prof. Díaz

"Nombre_Alumno" solo depende de "ID alumno", no de toda la clave primaria (ID_Alumno, ID_Curso)

ID_Alumno → Nombre_Alumno

La llave completa es (ID_Alumno, ID_Curso)

3NF

- Debe estar en 2NF
- Se deben eliminar las dependencias transitivas

Ejemplo tabla no normalizada (3NF):

ID_Alumno	Nombre	ID_Ciudad	Ciudad	País
1	Juan	10	Santiago	Chile
2	María	11	Lima	Perú

3NF

- Debe estar en 2NF
- Se deben eliminar las dependencias transitivas

Ejemplo tabla no normalizada (3NF):

ID_Alumno	Nombre	ID_Ciudad	Ciudad	País
1	Juan	10	Santiago	Chile
2	María	11	Lima	Perú

"Ciudad" y "País" dependen de "ID_Ciudad", la cual depende de "ID_Alumno", generando dependencia transitiva
ID_Alumno → Nombre, ID_Ciudad
ID_Ciudad → Ciudad, País

BCNF

- Debe estar en 3NF y cada dependencia funcional A → B, A debe ser superllave

Curso_ID → Profesor, Departamento

Profesor → Departamento

Ejemplo tabla no normalizada (BCNF)

Curso_ID	Profesor	Departamento
C1	Ana	Matemáticas
C2	Ana	Matemáticas
C3	Juan	Física

BCNF

- Debe estar en 3NF y cada dependencia funcional A → B, A debe ser superllave

Curso_ID → Profesor, Departamento

Profesor → Departamento

Ejemplo tabla no normalizada (BCNF)

Curso_ID	Profesor	Departamento
C1	Ana	Matemáticas
C2	Ana	Matemáticas
C3	Juan	Física

El determinante Profesor no es superllave, porque no determina toda la tupla (por ejemplo, no determina el Curso_ID).
Por lo que Profesor no es superllave

Usted ha sido contratado para analizar y rediseñar una base de datos utilizada por una empresa de transporte. Actualmente, la información sobre los envíos realizados por clientes se encuentra almacenada en una única tabla, como se muestra a continuación:

id_envío	cliente	rut_cliente	peso_kg	fecha	ciudad_entrega
1001	Sofía	15.815.461-7	10	2025-10-25	Santiago, Rancagua
1002	Paula	17.531.050-9	25	2025-11-06	Santiago, Talca, Curicó
1003	Ignacio	12.899.241-3	7	2025-03-17	Antofagasta, Calama
1004	Olivia	14.923.887-5	3	2025-01-21	Santiago, Temuco, Valdivia
1005	Consuelo	20.362.056-K	18	2025-09-08	Iquique, Arica
1006	Agustín	4.816.960-0	2	2025-08-30	Puerto Montt

Cuadro 5: Tabla `Envíos`

La columna `ciudad_entrega` contiene una lista con los nombres de las ciudades que forman parte del trayecto de entrega de cada envío. Un mismo envío puede pasar por una o más ciudades dependiendo de su destino, y el orden no importa.

A partir de la tabla anterior, responda lo siguiente:

a) (5 pts) Identifique todas las dependencias funcionales presentes en la tabla.

b) (5 pts) Indique si la tabla se encuentra en 1NF, 2NF o 3NF. Justifique su respuesta.

c) (5 pts) Proponga una descomposición del esquema que cumpla con BCNF, mostrando el nuevo conjunto de tablas y sus llaves primarias. Si no es posible, justifique por qué no es posible.

a) (5 pts) Identifique todas las dependencias funcionales presentes en la tabla.

Solución:

- `id_envío` → `cliente, rut_cliente, peso_kg, fecha, ciudad_entrega`
- `rut_cliente` → `cliente`

b) (5 pts) Indique si la tabla se encuentra en 1NF, 2NF o 3NF. Justifique su respuesta.

Solución:

La tabla no se encuentra en 1NF, ya que el atributo `ciudad_entrega` contiene listas de ciudades, lo que viola el principio de atomicidad de la 1FN. Por lo tanto, tampoco cumple con 2NF ni 3NF.

c) (5 pts) Proponga una descomposición del esquema que cumpla con BCNF, mostrando el nuevo conjunto de tablas y sus llaves primarias. Si no es posible, justifique por qué no es posible.

Solución:

Se propone la siguiente descomposición en BCNF:

- `Cliente(rut_cliente: string, cliente: string)`
  Donde `rut_cliente` → `cliente`
- `Envío(id_envío: int, rut_cliente: string, peso_kg: int, fecha: date)`
  Donde `id_envío` → `rut_cliente`, `peso_kg`, `fecha`
- `EnvíoCiudad(id_envío: int, nombre_ciudad: string)`
  No hay dependencias funcionales no triviales fuera de la llave primaria.

En la tabla `Cliente`, el determinante de la única dependencia funcional es `rut_cliente`, el cual es llave primaria. En la tabla `Envío` ocurre lo mismo con la llave `id_envío`. En `EnvíoCiudad`, todos los atributos dependen únicamente de la llave compuesta (`id_envío`, `nombre_ciudad`). Por ende, la descomposición propuesta garantiza que toda dependencia funcional esté determinada por una llave candidata, lo que cumple con BCNF.

Nota de corrección: Se asume que el nombre de una ciudad es único. En caso de asumir lo contrario, basta con definir una nueva tabla que relacione el nombre de una ciudad con un id serial, e incluir el id en la tabla `EnvíoCiudad` en lugar del nombre de la ciudad.

BIENVENIDOS A

SQL
(STRUCTURED QUERY LANGUAGE)

- **SELECT ... FROM**

  Extrae columnas específicas (SELECT) de una tabla (FROM)

- **WHERE**

  filtra filas que cumplen una condición

- **DISTINCT**

  Elimina valores duplicados de los resultados

- **LIKE**

  Busca un patrón específico dentro de un texto (ej: LIKE 'text')

- **ORDER BY**

  Ordena los resultados de forma ascendente (ASC) o descendente (DESC)

```
SELECT DISCTINCT nombre, categoria
FROM Peliculas
WHERE (categoría = 'SciFi' OR categoria = '
Adventure') AND nombre LIKE '%o%' ORDER
BY nombre ASC;
```

- **SELECT ... FROM**

  Extrae columnas específicas (SELECT) de una tabla (FROM)

- **WHERE**

  filtra filas que cumplen una condición

- **DISTINCT**

  Elimina valores duplicados de los resultados

- **LIKE**

  Busca un patrón específico dentro de un texto (ej: LIKE 'text')

- **ORDER BY**

  Ordena los resultados de forma ascendente (ASC) o descendente (DESC)

- **INSERT INTO**

  Agrega una nueva fila de datos a la tabla

- **UPDATE ... SET**

  Modifica datos de filas existentes

```
SELECT DISCTINCT nombre, categoria
FROM Peliculas
WHERE (categoría = 'SciFi' OR categoria = '
Adventure') AND nombre LIKE '%o%' ORDER
BY nombre ASC;
```

```
INSERT INTO Peliculas(id, nombre, categoria)
VALUES (6, 'Dune', 'SciFi');
```

```
UPDATE Peliculas SET calificacion = 8.0
WHERE id = 6;
```

- **DELETE FROM**

  Elimina filas de una tabla.

```
DELETE FROM Peliculas WHERE id = 6;
```

- **DELETE FROM**

  Elimina filas de una tabla.

```
DELETE FROM Peliculas WHERE id = 6;
```

- **GROUP BY**

Agrupa filas con el mismo valor para aplicarles una función de agregación.

Funciones de agregación

- **COUNT ( )**

  Cuenta el número de filas.

- **SUM ( )**

  Suma los valores de una columnaa númerica.

- **AVG ( )**

  Calcula el promedio de una columna numérica.

- **MAX ( ) / MIN ( )**

  Obtiene el valor máximo o mínimo de una columna.

```
SELECT
    director,
    COUNT(id) AS total_peliculas,
    AVG(calificacion) AS promedio_cal
FROM Peliculas
GROUP BY director
HAVING COUNT(id) >1;
```

- **HAVING**

  Filtra los grupos basados en una condición de agregación (se usa después de GROUP BY).

- **JOIN**

  Combina filas de dos tablas cuando los valores de las columnas realcionadas coinciden. Solo muestra resultados que existen en ambas tablas.

- **OUTER JOIN**

  Combina filas e incluye las que no tienen coincidencia en una de las tablas, rellenando los datos faltantes con NULL.

- **ANIDACIÓN**

  Es una consulta SELECT que se ejecuta dentro de otra consulta principal. Se usa como filtro o como una tabla temporal.

```
SELECT
    a.nombre,
    agg.total_peliculas AS
peliculas_actuadas
FROM Actores AS a
LEFT JOIN (
    SELECT id_actor, COUNT(id_pelicula)
AS total_peliculas
    FROM Actuo_en
    GROUP BY id_actor
) AS agg ON a.id = agg.id_actor;
```

Considere el siguiente esquema que tiene tablas que almacenan información sobre jugadores y equipos de fútbol:

- `Jugador(nombre: string, equipo: string, origen: string, fecha_contratacion: date, calificacion: int, sueldo: bigint, posicion: ENUM(delantero, mediocampo, defensa, arquero))`
- `Partido(fecha: int, equipo_local: string, equipo_visita: string, golesLocal: smallint, golesvisita: smallint)`
- `Equipo(nombre: string, nombre_liga: string, pais_liga: string, division: int, continente: string)`

En base a este esquema, construya las siguientes consultas utilizando SQL:

(3 pts) El jugador que lleva más tiempo contratado entre todos los equipos. consulta simple

(3 pts) Todos los jugadores de equipos europeos cuyo origen es africano.

(3 pts) El jugador que lleva más tiempo contratado entre todos los equipos. consulta simple

Solución:

```
SELECT nombre
FROM Jugador
WHERE fecha_contratacion = (SELECT MIN(fecha_contratacion) FROM Jugador);

-- Una manera alternativa válida:
SELECT nombre
FROM Jugador
ORDER BY fecha_contratacion ASC
LIMIT 1;
```

(3 pts) Todos los jugadores de equipos europeos cuyo origen es africano.

Solución:

```
SELECT Jugador.nombre, equipo.nombre    -- equipo.nombre es opcional
FROM Jugador, Equipo
WHERE Jugador.equipo = Equipo.nombre
AND Equipo.continente = 'Europa'
AND Jugador.origen = 'África';

-- Una manera alternativa válida:
SELECT Jugador.nombre
FROM Jugador
JOIN Equipo ON Jugador.equipo = Equipo.nombre
WHERE Equipo.continente = 'Europa' AND Jugador.origen = 'África';
```

BIENVENIDOS A

LÓGICA EN LA BASE DE DATOS

LÓGICA EN BDD

Lógica en la base de datos es implementar reglas y procesos directamente dentro del motor de BD (PostgreSQL, MySQL), en vez de hacerlo en el código de la aplicación.

- Vista → es guardarle un nombre a un SELECT para reutilizarlo. No guarda datos, solo la consulta. Cada vez que la usas, ejecuta el SELECT original sobre las tablas base.

- Vista Materializada → igual que una vista, pero sí guarda los resultados en disco. Es más rápida para leer, pero si los datos cambian hay que refrescarla manualmente con REFRESH MATERIALIZED VIEW.

- Trigger → un procedimiento que la BD ejecuta solo cuando ocurre algo (INSERT, UPDATE, DELETE). Siempre son dos pasos: defines una función y luego un trigger que la dispara ante el evento.

- Stored Procedure → una función con lógica compleja guardada en la BD. Puede tener variables, IF, FOR, múltiples queries. Se escribe en PL/pgSQL y se llama con SELECT.

VISTA

Crear

```
CREATE VIEW NombreVista AS
  SELECT col1, col2, AVG(eval) AS promedio
  FROM   Tabla
  WHERE  condicion
  GROUP BY col1, col2;
```

Usar

```
-- como cualquier tabla
SELECT * FROM NombreVista;

-- con filtros y joins
SELECT * FROM NombreVista
WHERE  col1 = 'valor';
```

Ejemplo Real

```
CREATE VIEW PelEval AS
  SELECT pelicula, director,
         AVG(eval) AS promedio
  FROM   Evaluaciones
  GROUP BY pelicula, director;

-- Consultar la vista:
SELECT * FROM PelEval
WHERE  pelicula = 'The Prestige';
```

VISTA

Crear

```sql
CREATE VIEW NombreVista AS
  SELECT col1, col2, AVG(eval) AS promedio
  FROM   Tabla
  WHERE  condicion
  GROUP BY col1, col2;
```

Usar

```sql
-- como cualquier tabla
SELECT * FROM NombreVista;

-- con filtros y joins
SELECT * FROM NombreVista
WHERE  col1 = 'valor';
```

Ejemplo Real

```sql
CREATE VIEW PelEval AS
  SELECT pelicula, director,
         AVG(eval) AS promedio
  FROM   Evaluaciones
  GROUP BY pelicula, director;

-- Consultar la vista:
SELECT * FROM PelEval
WHERE  pelicula = 'The Prestige';
```

Una vista es una tabla VIRTUAL definida por un SELECT.
La base de datos guarda la consulta, no los datos.
Cada consulta a la vista re-ejecuta el SELECT original.

VISTA MATERIALIZADA

Una vista materializada guarda los RESULTADOS en disco. Leer es rapidísimo, pero si los datos base cambian, los resultados NO se actualizan solos.

Crear

```sql
CREATE MATERIALIZED VIEW NombreVista AS
  SELECT col1, AVG(col2) AS promedio
  FROM   Tabla
  GROUP BY col1;
```

Usar

```sql
-- como cualquier tabla
SELECT * FROM NombreVista;

-- con filtros y joins
SELECT * FROM NombreVista
WHERE  col1 = 'valor';
```

Refrescar (actualizar datos)

```sql
-- Ejecutar manualmente cuando los datos
cambian
REFRESH MATERIALIZED VIEW NombreVista;

-- Sin bloquear lecturas concurrentes
(PostgreSQL 9.4+)
REFRESH MATERIALIZED VIEW CONCURRENTLY
NombreVista;
```

TRIGGER

Un trigger se ejecuta AUTOMÁTICAMENTE cuando ocurre un evento en la BD. Siempre se crea en DOS pasos: función + trigger que la llama.

Paso 1: función del trigger

```
CREATE OR REPLACE FUNCTION nombre_funcion()
RETURNS TRIGGER AS $$
BEGIN
  -- Lógica: usa NEW (fila nueva) y OLD (fila anterior)
  UPDATE OtraTabla SET col = NEW.col
  WHERE  id = NEW.id_fk;

  RETURN NEW;   -- FOR EACH ROW requiere RETURN NEW/OLD/NULL
END;
$$ LANGUAGE plpgsql;
```

Paso 2: crear el trigger

```
CREATE TRIGGER nombre_trigger

  -- momento:
  BEFORE | AFTER | INSTEAD OF

  -- evento (se pueden combinar con OR):
  INSERT | UPDATE | DELETE

  ON nombre_tabla

  FOR EACH [ROW|STATEMENT]

  EXECUTE FUNCTION nombre_funcion();
```

TRIGGER: EJEMPLO

Ejemplo: Bajar Stock al crear una venta

Paso 1: función de l trigger

```sql
CREATE OR REPLACE FUNCTION fn_bajar_stock()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE Productos
  SET    stock = stock - NEW.cantidad
  WHERE  id = NEW.id_producto;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Paso 2: crear el trigger

```sql
CREATE TRIGGER bajar_stock
  AFTER INSERT ON Ventas
  FOR EACH ROW
  EXECUTE FUNCTION
    fn_bajar_stock();
```

Resultado

```sql
-- Esto activa el trigger automáticamente:
INSERT INTO Ventas (id_producto, cantidad)
VALUES (1, 3);
-- Stock del producto 1 baja en 3 sin código
extra
```

SP (STORES PROCEDURES)

Un Stored Procedure es una FUNCIÓN almacenada en la base de datos. Puede ejecutar múltiples queries, tener variables, usar IF y FOR, y devolver resultados. Se escribe en PL/pgSQL, el lenguaje de PostgreSQL.

Esqueleto básico

```
CREATE OR REPLACE FUNCTION
  nombre_funcion(param1 TIPO, param2 TIPO)
RETURNS TIPO_RETORNO AS $$

DECLARE
  mi_variable   INTEGER;   -- variables locales
  mi_texto      VARCHAR;
  mi_fila       RECORD;   -- cualquier fila de
tabla

BEGIN
  -- sentencias SQL aquí
  RETURN mi_variable;
END;
$$ LANGUAGE plpgsql;
```

Variantes de returns

```
-- No retorna nada
RETURNS VOID
-- ... no necesitas RETURN al final

-- Retorna un valor simple
RETURNS INTEGER | VARCHAR | BOOLEAN
  RETURN mi_variable;

-- Retorna una tabla completa
RETURNS TABLE(col1 TIPO, col2 TIPO)
  RETURN QUERY SELECT ...;

-- Llamar la función:
SELECT nombre_funcion('arg1', 42);
```

ALGUNOS FLUJOS

IF / ELSIF / ELSE

```
-- Asignar resultado de query a variable
SELECT COUNT(*) INTO v_total
FROM Peliculas WHERE director = p_dir;

IF v_total >= 10 THEN
  v_clase := 'Prolífico';
ELSIF v_total >= 5 THEN
  v_clase := 'Activo';
ELSE
  v_clase := 'Emergente';
END IF;
```

FOR LOOP sobre query

```
-- Por cada producto, mostrar su nombre en
mayúsculas
FOR v_prod IN
   SELECT * FROM Productos
LOOP
   -- acceder a columnas con v_prod.columna
   v_nombre := UPPER(v_prod.nombre);
   INSERT INTO Resumen
   VALUES (v_prod.id, v_nombre);
END LOOP;
```

RAISE, mensajes y errores

```
RAISE NOTICE 'Info: valor = %', p_valor;   -- informativo
RAISE WARNING 'Advertencia: %', p_valor;
RAISE EXCEPTION 'Error: valor negativo %', p_valor; -- detiene + rollback
```

SP (STORES PROCEDURES): EJEMPLO

```sql
-- Aplica un descuento a un producto según su precio
CREATE OR REPLACE FUNCTION
   aplicar_descuento(p_id INTEGER)
RETURNS VARCHAR AS $$
DECLARE
   v_precio NUMERIC;
   v_mensaje VARCHAR;
BEGIN
   -- guardar el precio del producto en una variable
   SELECT precio INTO v_precio
   FROM Productos WHERE id = p_id;
   -- decidir el descuento según el precio
   IF v_precio > 100 THEN
      UPDATE Productos SET precio = v_precio * 0.8
      WHERE id = p_id;
      v_mensaje := 'Descuento 20% aplicado';
   ELSIF v_precio > 50 THEN
      UPDATE Productos SET precio = v_precio * 0.9
      WHERE id = p_id;
      v_mensaje := 'Descuento 10% aplicado';
   ELSE
      v_mensaje := 'Precio muy bajo, sin descuento';
   END IF;
   RETURN v_mensaje;
END;
$$ LANGUAGE plpgsql;
-- Uso:
SELECT aplicar_descuento(3);
-- → 'Descuento 20% aplicado'
```

Una empresa necesita gestionar el inventario de sus activos (equipos de oficina, computadores, vehículos, etc.). Para ello, se cuenta con el siguiente modelo de datos simplificado:

- **Activo**
  - `id_activo` (PK): Identificador único del activo
  - `nombre`: Nombre del activo
  - `categoria`: Tipo de activo, puede tomar uno de los siguientes valores:
    - `'EQUIPO'` (computadores, impresoras, etc.)
    - `'MUEBLE'` (escritorios, sillas, etc.)
    - `'VEHICULO'` (autos, camionetas, etc.)
  - `valor`: Costo del activo en moneda local (debe ser mayor a 0)
  - `estado`: Situación actual del activo, por ejemplo:
    - `'EN USO'`
    - `'FUERA DE USO'`
    - `'EN MANTENCIÓN'`
- **Ubicacion**
  - `id_ubicacion` (PK): Identificador único de la ubicación
  - `nombre`: Nombre del lugar (ej. "Oficina Central")
  - `descripcion`: Detalle adicional de la ubicación
- **Movimiento**
  - `id_movimiento` (PK): Identificador único del movimiento
  - `id_activo` (FK): Activo que se mueve
  - `id_ubicacion` (FK): Ubicación del movimiento
  - `fecha_movimiento`: Fecha en que se realizó el movimiento
  - `tipo_movimiento`: Puede ser:
    - `'ENTRADA'`: El activo ingresa a una ubicación.
    - `'SALIDA'`: El activo sale de una ubicación.

a) Cree una vista llamada `vw_activos_detalle` que muestre: nombre del activo, categoría, valor, estado, última ubicación y fecha del último movimiento. (5 pts)

c) Cree un trigger llamado `trg_valor_activo` que: (5 pts)

- Se dispare **antes de insertar** en la tabla **Activo**.
- Valide que el valor del activo sea mayor a 0.
- Si no se cumple, debe impedir la inserción y mostrar un mensaje de error.

a) Cree una vista llamada `vw_activos_detalle` que muestre: nombre del activo, categoría, valor, estado, última ubicación y fecha del último movimiento. (5 pts)

a) Vista `vw_activos_detalle`

```
CREATE VIEW vw_activos_detalle AS
SELECT a.nombre, a.categoria, a.valor, a.estado,
       u.nombre AS ultima_ubicacion,
       m.fecha_movimiento AS fecha_ultimo_movimiento
FROM Activo a
JOIN Movimiento m ON a.id_activo = m.id_activo
JOIN Ubicacion u ON m.id_ubicacion = u.id_ubicacion
WHERE m.fecha_movimiento = (
    SELECT MAX(fecha_movimiento)
    FROM Movimiento
    WHERE id_activo = a.id_activo
);
```

c) Cree un trigger llamado `trg_valor_activo` que: (5 pts)

- Se dispare **antes de insertar** en la tabla **Activo**.
- Valide que el valor del activo sea mayor a 0.
- Si no se cumple, debe impedir la inserción y mostrar un mensaje de error.

```
CREATE OR REPLACE FUNCTION fn_validar_valor() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.valor <= 0 THEN
        RAISE EXCEPTION 'El valor del activo debe ser mayor a 0';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trg_valor_activo
BEFORE INSERT ON Activo
FOR EACH ROW EXECUTE FUNCTION fn_validar_valor();
```

BIENVENIDOS A

TRANSACCIONES Y LOCKS

TRANSACCIONES

Una transacción es una **secuencia de 1 o más operaciones** que modifican o consultan la **base de datos,** donde su principal propósito es garantizar la integridad de los datos. Ejemplos de uso:

- Transferencias de dinero entre cuentas
- Compra por internet
- Inscribir un curso

Acciones:

- **R(X):** Lectura (Read) del dato X.
- **W(X):** Escritura (Write) o modificación del dato X.
- **C:** Commit (Confirmación de que la transacción terminó y sus cambios son definitivos).
- **A:** Abort o Rollback (Anulación de la transacción, sus cambios se deshacen).

ACID

Las transacciones tiene que asegurar la presencia de **ACID:**

- Atomicitiy → (Atomicidad)
- Consistency → (Consistencia)
- Isolation → (Aislamiento)
- Durability → (Durabilidad)



Transaction Manager se encarga de asegurar **Isolation** y **Consistency.**

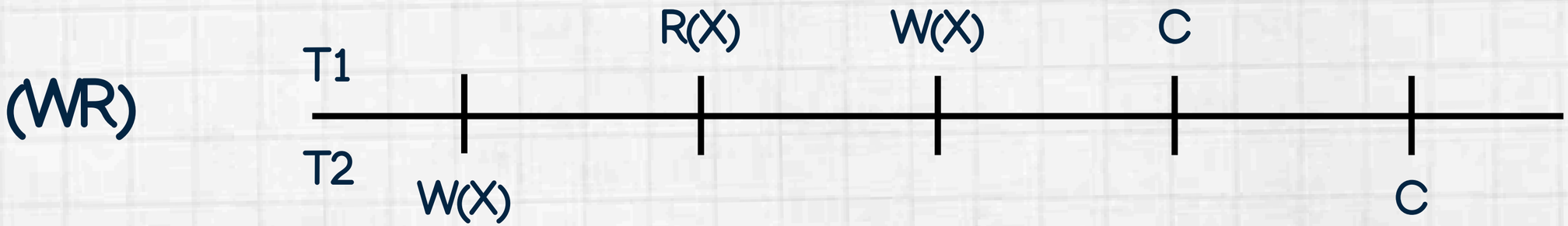
Log y Recovery Manager se encargan de asegurar **Atomicity** y **Durability**

ACID

- **Atomicitiy (Atomicidad):** O se ejecutan todas las operaciones de la transacción, o no se ejecuta ninguna.
- **Consistency (Consistencia):** Cada transacción preserva la consistencia de la BD (restricciones de integridad, etc.).
- **Isolation (Aislamiento):** Cada transacción debe ejecutarse como si se estuviese ejecutando sola, de forma aislada.
- **Durability (Durabilidad):** Los cambios que hace cada transacción son permanentes en el tiempo, independiente de cualquier tipo de falla.

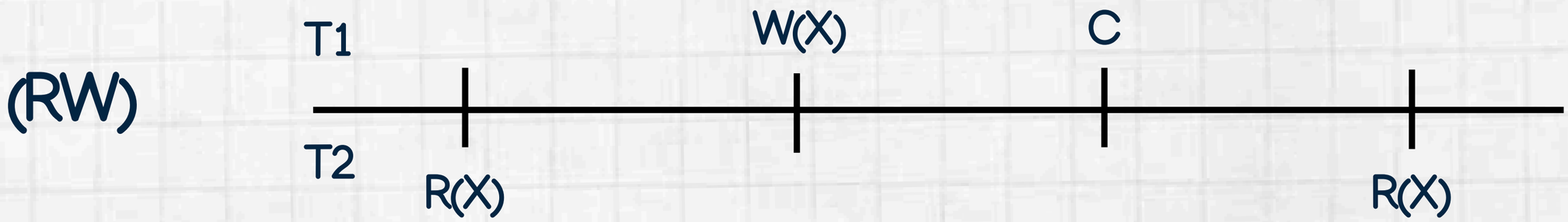
CONFLICTOS CON TRANSACCIONES

- **Lecturas sucias (Write – Read):** Ocurren cuando una transacción **accede a datos que han sido modificados** por otra transacción que aún no se ha terminado.



CONFLICTOS CON TRANSACCIONES

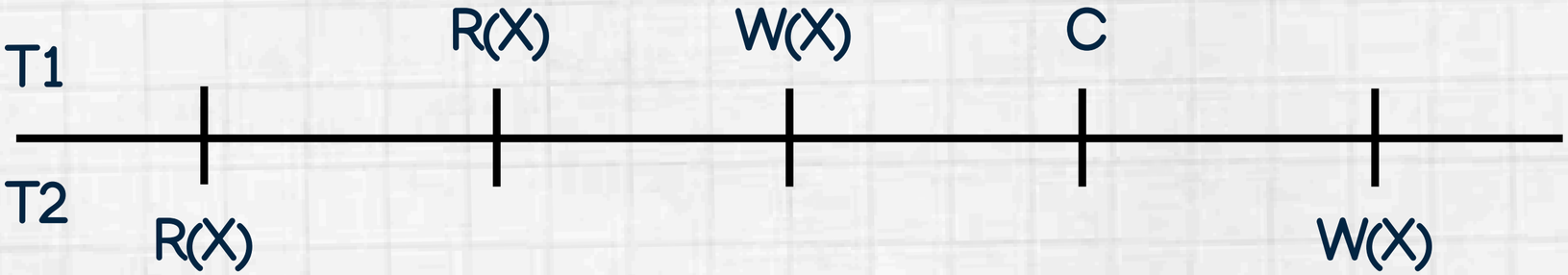
- **Lecturas no repetibles (Read – Write):** Ocurren cuando una fila es leída dos veces y el valor cambia entre ambas lecturas. Esto se debe a que otra transacción actualiza o modfica la fila entre las dos lecturas individuales.



CONFLICTOS CON TRANSACCIONES

- **Actualización perdida o reescritura de datos temporales (Write – Write):** Ocurren cuando dos transacciones que intentan actualizar la misma fila son procesadas en un tiempo que permite que una de las actualizaciones sobreescriba a la otra.

(WW)



SCHEDULE

Un schedule S es una secuencia de operaciones primitivas de una o más transacciones, tal que para toda transacción, las acciones de ella aparecen en el mismo orden que en su definición

Transacciones

T1
READ(A,x)
x:= x + 100
WRITE(A,x)
READ(B,x)
x:= x + 200
WRITE(B,x)

T2
READ(A,y)
y:= y * 2
WRITE(A,y)
READ(B,y)
y:= y * 3
WRITE(B,y)

Schedule

T1	T2
READ(A,x)	
x:= x + 100	
WRITE(A,x)	
READ(B,x)	
x:= x + 200	
WRITE(B,x)	
	READ(A,y)
	y:= y * 2
	WRITE(A,y)
	READ(B,y)
	y:= y * 3
	WRITE(B,y)

SCHEDULE SERIAL

Un schedule S es serial si no hay intercalación entre las acciones

T1	T2
READ(A,x)	
x:= x + 100	
WRITE(A,x)	
READ(B,x)	
x:= x + 200	
WRITE(B,x)	
	READ(A,y)
	y:= y * 2
	WRITE(A,y)
	READ(B,y)
	y:= y * 3
	WRITE(B,y)

SCHEDULE SERIALIZABLE

Un **schedule** S es **serializable** si existe algún **schedule S'** serial con las mismas transacciones, tal que el resultado de S y S' es el mismo para todo estado inicial de la BD

Las operaciones para A en x e y no afectan al orden de x e y de B, entonces S es serializable

S

T1	T2
READ(A,x)	
x:= x + 100	
WRITE(A,x)	
	READ(A,y)
	y:= y * 2
	WRITE(A,y)
READ(B,x)	
x:= x + 200	
WRITE(B,x)	
	READ(B,y)
	y:= y * 3
	WRITE(B,y)

S'

T1	T2
READ(A,x)	
x:= x + 100	
WRITE(A,x)	
READ(B,x)	
x:= x + 200	
WRITE(B,x)	
	READ(A,y)
	y:= y * 2
	WRITE(A,y)
	READ(B,y)
	y:= y * 3
	WRITE(B,y)

SCHEDULE NO SERIALIZABLE

T1	T2
READ(A,x)	
x:= x + 100	
WRITE(A,x)	
	READ(A,y)
	y:= y * 2
	WRITE(A,y)
	READ(B,y)
	y:= y * 3
	WRITE(B,y)
READ(B,x)	
x:= x + 200	
WRITE(B,x)	

En T2, la transacción de READ(B, y) realiza cambios que afectan a READ(B,x), por lo mismo el schedule No es serializable

CONFLICT SERIALIZABLE

Un schedule es **conflict serializable** si puedo transformalor a uno serial usando permutaciones.

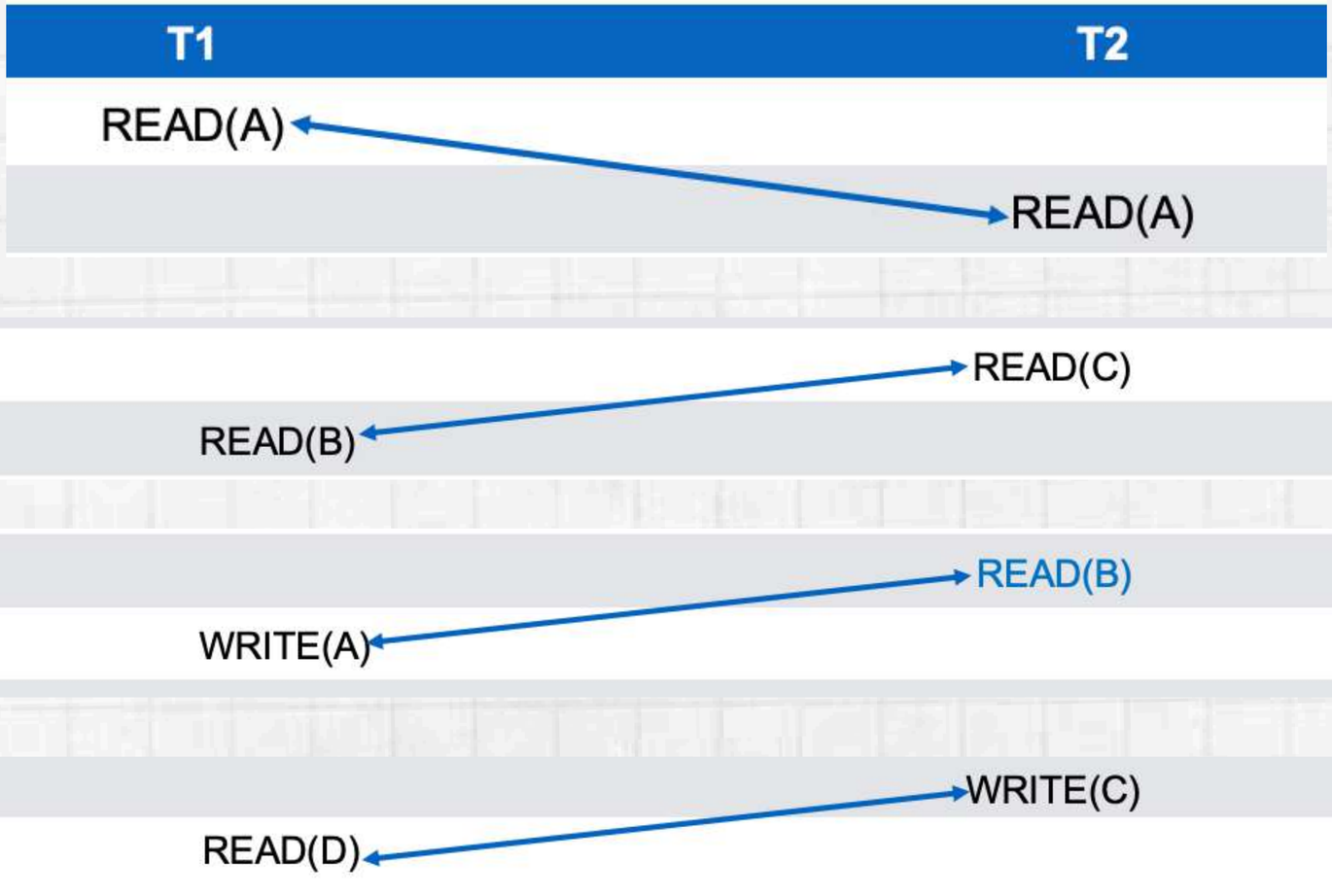
Si un **schedule es conflict serializable implica que también es serializable,** pero hay schedules serializables que no son conflict serializable.

Con permutaciones se puede:

- Transformar el **schedule en uno serial**
- Preservar el orden de **todos** los conflictos

SCHEDULE - PERMUTACIONES

Permitidas



R(x) ⟷ R(x)

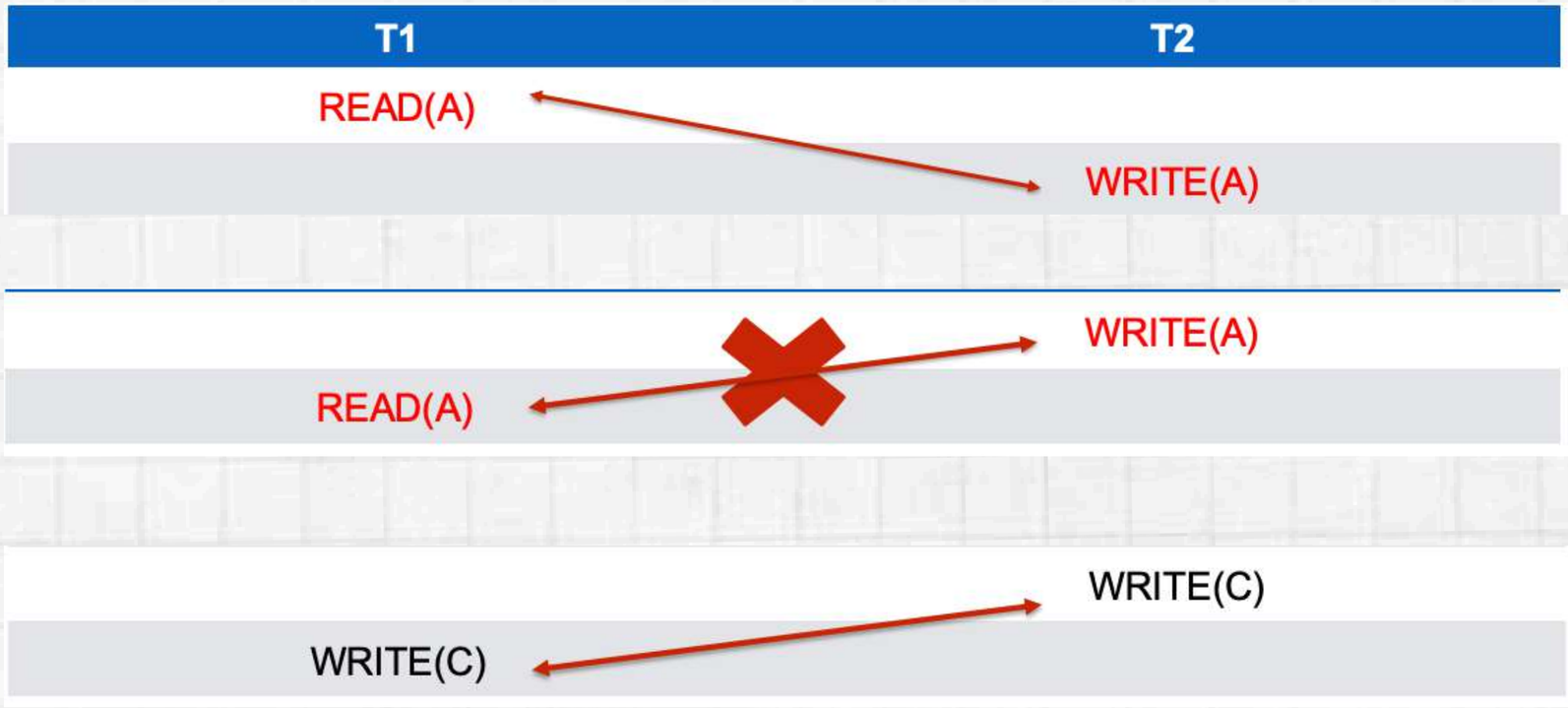
R(x) ⟷ R(y)

W(x) ⟷ R(y)

R(x) ⟷ W(y)

SCHEDULE - PERMUTACIONES

NO Permitidas



R(x) ⟷ W(x)

W(x) ⟷ R(x)

W(x) ⟷ W(x)

a) Defina los siguientes conceptos:

i. Schedule Serializable. (2 pts)

ii. Schedule Serial. (2 pts)

iii. Schedule Conflict Serializable. (2 pts)

a) Defina los siguientes conceptos:

I. Schedule Serializable. (2 pts)

Solución: Es un orden de ejecución de transacciones concurrentes que produce el mismo resultado que la ejecución de un schedule serial.

II. Schedule Serial. (2 pts)

Solución: Es aquel en que las transacciones se ejecutan una después de otra, sin intercalación entre ellas.

III. Schedule Conflict Serializable. (2 pts)

Solución: Un schedule conflict serializable es un tipo de schedule serializable donde es posible reordenar las operaciones (sin cambiar el orden relativo de operaciones conflictivas) para obtener un schedule serial equivalente.

b) Para cada una de las siguientes ejecuciones, especifique si corresponde a **Lecturas Sucias**, **Lecturas No Repetibles** o **Actualización Perdida**. Justifique su respuesta. (2 pts c/u)

T1	T2	T3
R(a)		
	W(a)	
R(b)		
	W(c)	
W(b)		
		R(a)
R(a)		
		R(c)

Schedule 1

T1	T2	T3
R(a)		
	R(a)	
R(b)		
		W(c)
W(b)		
	W(a)	
R(c)		
		Abort

Schedule 2

T1	T2	T3
R(a)		
	R(a)	
W(b)		
		W(c)
W(a)		
	W(a)	
R(c)		
	W(c)	

Schedule 3

b) Para cada una de las siguientes ejecuciones, especifique si corresponde a **Lecturas Sucias**, **Lecturas No Repetibles** o **Actualización Perdida**. Justifique su respuesta. (2 pts c/u)

T1	T2	T3
R(a)		
	W(a)	
R(b)		
	W(c)	
W(b)		
		R(a)
R(a)		
		R(c)

Schedule 1

T1	T2	T3
R(a)		
	R(a)	
R(b)		
		W(c)
W(b)		
	W(a)	
R(c)		
		Abort

Schedule 2

T1	T2	T3
R(a)		
	R(a)	
W(b)		
		W(c)
W(a)		
	W(a)	
R(c)		
	W(c)	

Schedule 3

Solución: En Schedule 1, es Lectura no repetible. T1 lee **a**, después T2 escribe en **a** y cuando T1 nuevamente lee **a**, el valor fue actualizado por T2.

En Schedule 2, es Lectura Sucia. T3 escribe en **c**, T1 lee **c** y después T3 aborta. T1 leyó un valor de **c** que no se confirmó.

En Schedule 3, es Actualización Perdida. T1 lee **a**, después lee T2, posteriormente T1 escribe en **a** y finalmente T2 lo sobreescribe. Por tanto, se pierde la escritura de T1.

c) Aplicando la técnica de permutación, demuestre si la siguiente ejecución es conflict serializable. (3 pts)

T1	**T2**	**T3**
R(b)		
	R(a)	
W(b)		
		W(c)
W(b)		
		W(a)
R(c)		
	W(c)	

Cuadro 1: Conflict Serializable

c) Aplicando la técnica de permutación, demuestre si la siguiente ejecución es conflict serializable. (3 pts)

T1	T2	T3
R(b)		
	R(a)	
W(b)		
		W(c)
W(b)		
		W(a)
R(c)		
	W(c)	

Cuadro 1: Conflict Serializable

Solución: Hay dependencia entre la T2 y la T3 por las operaciones sobre **a**, por lo tanto no es posible permutar. También hay conflicto entre T3 y T2 por las operaciones sobre **c**, por lo tanto tampoco se pueden permutar. Esto produce que no es posible dejar T2 antes que T3 o T3 antes que T2.

¡GRACIAS!