# Pauta Examen IIC2413 – Bases de Datos

1er semestre 2026
Profesores: Eduardo Bustos – Christian Álvarez

8 de julio de 2026

## Pregunta 1 – Modelo ER + Diseño de BD + FN (15 pts)

### a) Modelo Entidad-Relación (10 pts)

**Entidades fuertes y atributos** (PK indicada):

- **DONANTE**: RUN (PK), nombres, apellidos, fecha_nacimiento, sexo, tipo_sangre, teléfono, dirección
- **PACIENTE**: RUN (PK), nombres, apellidos, fecha_nacimiento, tipo_sangre, diagnóstico, teléfono
- **PERSONAL_MEDICO**: RUN (PK), nombre, especialidad, turno
- **DONACION**: código (PK), fecha, hora, volumen_ml
- **BOLSA_SANGRE**: código (PK), fecha_vencimiento, tipo_sangre, componente, estado
- **SOLICITUD**: código (PK), tipo_sangre_requerido, componente, cantidad_bolsas, urgencia
- **TRANSFUSION**: código (PK), fecha, hora
- **EVENTO_ADVERSO**: código (PK, candidata a entidad débil), fecha, descripción, gravedad
- **EXAMEN_SEROLOGICO**: código (PK) – opcionalmente atributos de DONACION (relación 1:1 obligatoria); resultado (apta/no apta), motivo_rechazo

**Relaciones y cardinalidades** (ver diagrama gráfico a continuación, notación Chen):
**Entidad débil:** EVENTO_ADVERSO depende existencialmente de DONANTE o PACIENTE; su PK parcial sería un discriminante + la PK de la entidad fuerte.

**Herencia:** Se justifica **DONANTE / PACIENTE → PERSONA** (atributos comunes: RUN, nombres, apellidos, fecha_nacimiento, teléfono, tipo_sangre), disjunta y parcial, si se contempla que una persona sea donante y paciente a la vez. Aceptar también argumentación de herencia en PERSONAL_MEDICO si está bien justificada.

*Puntaje sugerido: 3 pts entidades+atributos, 3 pts relaciones+cardinalidades, 2 pts entidad débil, 2 pts justificación de herencia.*

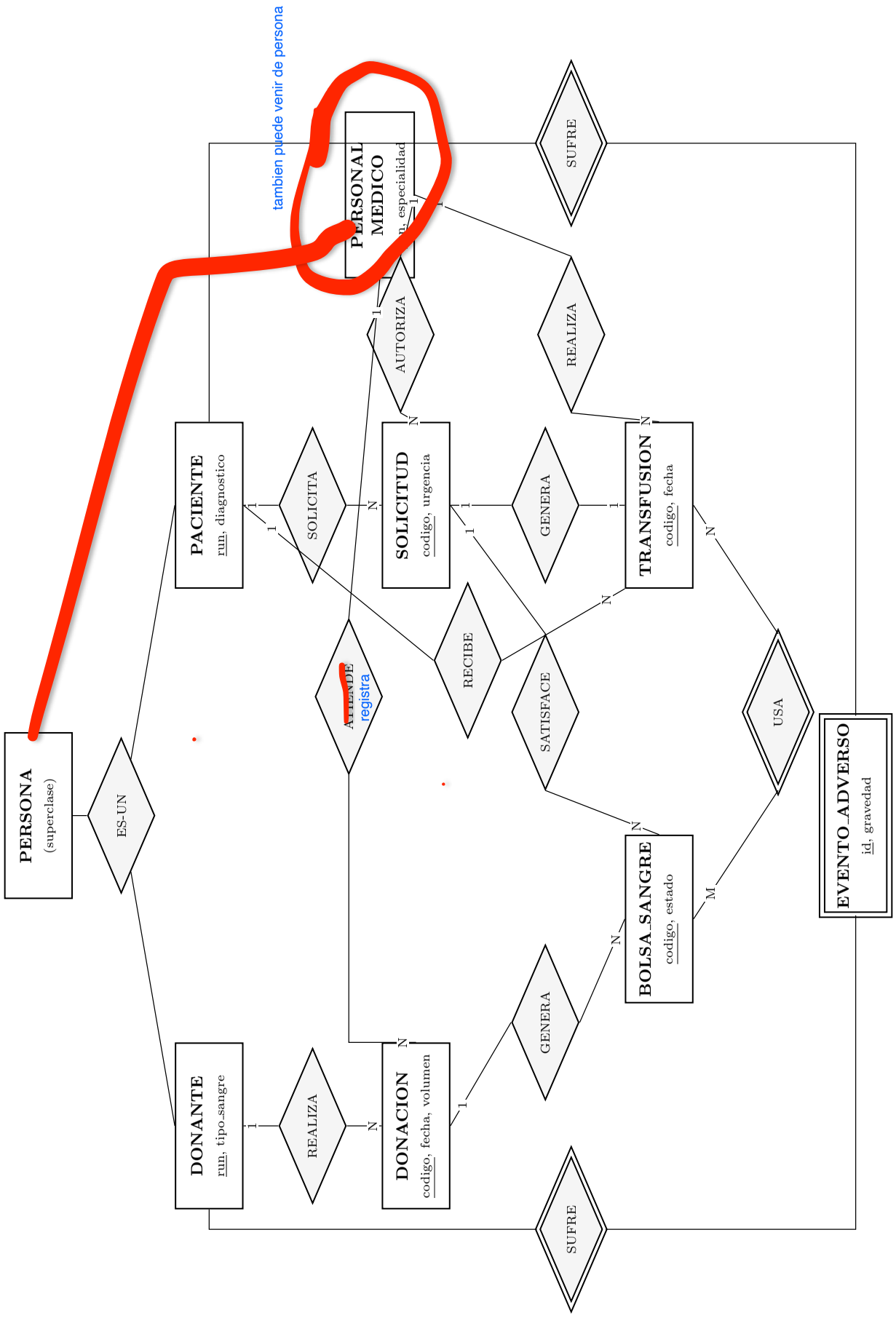



Figura 1: Modelo Entidad-Relación del banco de sangre (notación Chen). Doble línea = entidad/relación débil.

### b) Esquema Relacional (5 pts)

```
DONANTE(run PK, nombres, apellidos, fecha_nacimiento, sexo,
        tipo_sangre, telefono, direccion)

PACIENTE(run PK, nombres, apellidos, fecha_nacimiento,
         tipo_sangre, diagnostico, telefono)

PERSONAL_MEDICO(run PK, nombre, especialidad, turno)

DONACION(codigo PK, fecha, hora, volumen_ml,
         run_donante FK->DONANTE, run_personal FK->PERSONAL_MEDICO,
         resultado_examen CHECK IN ('apta','no apta'), motivo_rechazo)

BOLSA_SANGRE(codigo PK, fecha_vencimiento, tipo_sangre, componente,
             estado CHECK IN ('disponible','reservada','usada','vencida'),
             codigo_donacion FK->DONACION,
             codigo_solicitud FK->SOLICITUD NULL)

SOLICITUD(codigo PK, tipo_sangre_requerido, componente, cantidad_bolsas,
          urgencia CHECK IN('alta','media','baja'),
          run_paciente FK->PACIENTE, run_medico_autoriza FK->PERSONAL_MEDICO)

TRANSFUSION(codigo PK, fecha, hora, codigo_solicitud FK->SOLICITUD UNIQUE,
            run_paciente FK->PACIENTE, run_personal FK->PERSONAL_MEDICO)

TRANSFUSION_BOLSA(codigo_transfusion FK->TRANSFUSION,
                  codigo_bolsa FK->BOLSA_SANGRE,
                  PRIMARY KEY(codigo_transfusion, codigo_bolsa))

EVENTO_ADVERSO(id PK, fecha, descripcion, gravedad,
               run_donante FK->DONANTE NULL, run_paciente FK->PACIENTE NULL,
               CHECK (run_donante IS NOT NULL OR run_paciente IS NOT NULL))
```

**Tipos de datos:** RUN VARCHAR, fechas DATE/DATETIME, volumen_ml INT, tipo_sangre ENUM/CHECK (8 valores), cantidades INT, estados CHECK.

**Normalización:** El esquema está en **3NF/BCNF**: todos los atributos no-clave dependen completa y directamente de la PK de su tabla y no hay dependencias transitivas entre atributos no-clave (ej. `tipo_sangre_requerido` depende solo de `codigo` de SOLICITUD). TRANSFUSION_BOLSA no tiene atributos no-clave, por lo que cumple BCNF trivialmente.

*Puntaje sugerido: 2 pts esquema con FKs, 1.5 pts tipos de datos/CHECKs, 1.5 pts justificación 3NF/BCNF.*

## Pregunta 2 – SQL + Lógica en la BD (15 pts)

### a) Consultas SQL (6 pts)

#### i) Usuarios con saldo insuficiente para tarifa "Normal" (2 pts)

```
SELECT u.nombres, u.correo
FROM USUARIO u
WHERE u.saldo < 2000
  AND u.saldo < (SELECT valor_desbloqueo FROM TARIFA WHERE nombre = 'Normal')
    ;
```

*El enunciado ya fija el umbral en $2.000; basta con* `WHERE saldo < 2000`*. Aceptar ambas versiones. Penalizar JOIN innecesario que duplique filas.*

**ii) Scooters disponibles por estación, mayor a menor (2 pts)**

```
SELECT e.cod_estacion, e.nombre, COUNT(s.cod_scooter) AS disponibles
FROM ESTACION e
LEFT JOIN SCOOTER s ON s.cod_estacion = e.cod_estacion
                    AND s.estado = 'disponible'
GROUP BY e.cod_estacion, e.nombre
ORDER BY disponibles DESC;
```

*Exigir LEFT JOIN para incluir estaciones con 0 disponibles.*

**iii) Ingreso total por tarifa (arriendos finalizados) (2 pts)**

```
SELECT t.cod_tarifa, t.nombre, SUM(a.costo_total) AS ingreso_total
FROM TARIFA t
JOIN ARRIENDO a ON a.cod_tarifa = t.cod_tarifa
WHERE a.fecha_fin IS NOT NULL
GROUP BY t.cod_tarifa, t.nombre;
```

## b) Procedimiento Almacenado `finalizar_arriendo` (4 pts)

```
CREATE OR REPLACE FUNCTION finalizar_arriendo(p_cod_arriendo INT,
                                              p_cod_estacion_fin INT)
RETURNS VOID AS $$
DECLARE
    v_fecha_inicio DATETIME;
    v_fecha_fin DATETIME;
    v_minutos INT;
    v_valor_desbloqueo DECIMAL(10,2);
    v_valor_minuto DECIMAL(10,2);
    v_costo DECIMAL(10,2);
    v_run_usuario VARCHAR(12);
    v_cod_scooter INT;
BEGIN
    SELECT fecha_inicio, run_usuario, cod_scooter
    INTO v_fecha_inicio, v_run_usuario, v_cod_scooter
    FROM ARRIENDO
    WHERE cod_arriendo = p_cod_arriendo AND fecha_fin IS NULL;

    IF v_fecha_inicio IS NULL THEN
        RAISE EXCEPTION 'Arriendo % no existe o ya esta finalizado',
              p_cod_arriendo;
    END IF;

    v_fecha_fin := NOW();
    v_minutos := EXTRACT(EPOCH FROM (v_fecha_fin - v_fecha_inicio)) / 60;

    SELECT ta.valor_desbloqueo, ta.valor_por_minuto
    INTO v_valor_desbloqueo, v_valor_minuto
    FROM ARRIENDO a JOIN TARIFA ta ON a.cod_tarifa = ta.cod_tarifa
    WHERE a.cod_arriendo = p_cod_arriendo;

    v_costo := v_valor_desbloqueo + v_valor_minuto * v_minutos;
```

optativo

```
    UPDATE ARRIENDO
    SET fecha_fin = v_fecha_fin,
        cod_estacion_fin = p_cod_estacion_fin,
        costo_total = v_costo
    WHERE cod_arriendo = p_cod_arriendo;

    UPDATE USUARIO SET saldo = saldo - v_costo WHERE run_usuario =
       v_run_usuario;

    UPDATE SCOOTER SET estado = 'disponible', cod_estacion =
       p_cod_estacion_fin
    WHERE cod_scooter = v_cod_scooter;
END;
$$ LANGUAGE plpgsql;
```

*Puntaje: 1 pt validación de existencia/en curso, 1.5 pts cálculo correcto de costo, 1 pt actualización de saldo y fecha_fin, 0.5 pt actualizar estado del scooter.*

### c) Trigger (3 pts)

```
CREATE OR REPLACE FUNCTION actualizar_estado_scooter()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE SCOOTER SET estado = 'arrendado' WHERE cod_scooter = NEW.
       cod_scooter;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_arriendo_insert
AFTER INSERT ON ARRIENDO
FOR EACH ROW
EXECUTE FUNCTION actualizar_estado_scooter();
```

### d) Vista VW_SCOOTERS_ESTADO (2 pts)

```
CREATE VIEW VW_SCOOTERS_ESTADO AS
SELECT s.cod_scooter, s.modelo, s.estado, e.nombre AS nombre_estacion,
       e.comuna,
       CASE WHEN s.nivel_bateria < 20
                 OR s.estado IN ('mantencion','fuera_servicio')
            THEN 'SI' ELSE 'NO' END AS requiere_atencion
FROM SCOOTER s
JOIN ESTACION e ON s.cod_estacion = e.cod_estacion;
```

## Pregunta 3 – Transacciones, Conflict Serializability y Locks (15 pts)

Schedule (orden temporal según la tabla del enunciado):

$$R_1(A),\ R_2(B),\ W_1(A),\ R_3(B),\ W_2(B),\ W_3(A),\ R_1(C),\ W_1(C)$$

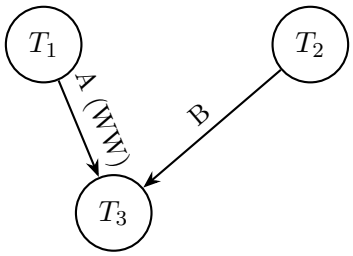
### a) Identificación de conflictos (4 pts)

| Par | Elem. | Tipo | Anomalía |
|---|---|---|---|
| $W_1(A)$ – $W_3(A)$ | A | Escritura-Escritura (WW) | Actualización perdida: si $T_3$ sobrescribe A sin considerar el cambio de $T_1$, el efecto de $T_1$ se pierde |
| $R_3(B)$ – $W_2(B)$ | B | Lectura-Escritura / Escritura-Lectura (según orden) | Lectura no repetible / inconsistencia: el valor leído por $T_3$ puede quedar obsoleto respecto de lo que escribe $T_2$ |

*Nota para el corrector: verificar el orden exacto de $R_3(B)$ y $W_2(B)$ según la tabla recibida, ya que determina si el conflicto es RW o WR. Dar puntaje completo si identifican correctamente el par WW sobre A y el conflicto sobre B entre $T_2$ y $T_3$, con anomalía coherente con el orden usado.*

### b) Conflict Serializable (8 pts)

#### i) Grafo de precedencia (4 pts)




Aristas: $T_1 \to T_3$ (conflicto WW sobre A: $W_1(A)$ antes de $W_3(A)$); $T_2 \to T_3$ (conflicto sobre B). Aceptar el grafo consistente con el orden exacto entregado a los alumnos, exigiendo que cada arista tenga su par conflictivo explícito.

**ii) ¿Es conflict serializable? (2 pts)**

Si el grafo no tiene ciclos ($T_1 \to T_3$, $T_2 \to T_3$), **es conflict serializable**, con orden serial equivalente $T_1, T_2, T_3$ (o $T_2, T_1, T_3$). Si el alumno encuentra una arista adicional que cierre un ciclo (ej. $T_3 \to T_2$), debe señalar explícitamente el ciclo y concluir que NO es serializable.

**iii) Verificación por permutaciones (2 pts)**

Se espera mostrar al menos 2 intercambios de operaciones no conflictivas (ej. $R_2(B)$ con $R_1(A)$, o mover $R_1(C)/W_1(C)$ que no interfieren con $T_2$ ni $T_3$) para llegar a un orden serial $T_1; T_2; T_3$, justificando cada intercambio por actuar sobre elementos distintos o no ser ambas escrituras/lecturas conflictivas.

### c) Two-Phase Locking – 2PL (3 pts)

#### a) Paso a paso (2 pts):

1. $T_1$ solicita $S(A) \to$ **otorgado** (nadie tiene lock sobre A).
2. $T_1$ solicita $X(B) \to$ **otorgado** (nadie tiene lock sobre B).
3. $T_2$ solicita $X(B) \to$ **en espera** ($B$ tiene X-lock de $T_1$).
4. $T_2$ solicita $X(A) \to$ **en espera** (además $T_1$ mantiene $S(A)$ bajo Strict 2PL).

**b) ¿Deadlock? (1 pt)**

$T_2$ queda bloqueada esperando indefinidamente a $T_1$ por $B$, y además necesita $A$ (retenido por $T_1$ bajo Strict 2PL hasta el commit/abort). Si en algún punto $T_1$ solicitara un lock ya tomado por $T_2$, se cerraría el ciclo $T_1 \to T_2 \to T_1$ y habría deadlock formal. Con la secuencia dada, $T_2$ espera a $T_1$ pero $T_1$ no llega a solicitar nada que $T_2$ posea, por lo que estrictamente **no se cierra un ciclo formal de espera**; se debe evaluar con flexibilidad según el razonamiento del alumno sobre la espera de $T_2$.

## Pregunta 4 – EEDD, Almacenamiento e Índices (15 pts)

Tabla $R(w, x, y, z)$ con 2.400.000 tuplas, $B$ tuplas por página, $P$ punteros por página en el índice B+, $M$ páginas de memoria.

### a) Índice Hash Unclustered (5 pts)

```
SELECT x, y FROM R WHERE w = 358
```

Costo = acceso al índice hash + acceso a página de datos por cada tupla (unclustered, 4 tuplas en 4 páginas distintas).

- Acceso al índice hash: $\approx 1$ I/O (hash directo al bucket). ACCESO AL HASH = 0 si está en memoria
- Como es **unclustered** y hay 4 tuplas con $w = 358$ en 4 páginas distintas: 4 I/Os adicionales.

$$\text{Costo total} \approx 1 + 4 = 5 \text{ I/Os}$$

(aceptar $2 + 4 = 6$ si consideran 2 niveles de hash). Aceptar 4 si I/O de hash =0

### b) Índice B+ Clustered (5 pts)

```
SELECT w, z FROM R WHERE x < 200
```

Con índice **clustered**, las tuplas que satisfacen $x < 200$ están almacenadas contiguamente.

$$\text{Costo total} \approx h + \frac{\frac{1}{8} \cdot 2.400.000}{B} = h + \frac{300.000}{B} \text{ I/Os}$$

donde $h = \log_P(\text{número de hojas})$ es la altura del árbol B+ hasta la primera hoja relevante. Al ser clustered, no se paga acceso adicional por tupla individual, solo el recorrido secuencial de páginas de datos.

*Dejar la fórmula en términos de $B$ y $P$ ya que el enunciado no da valores numéricos; exigir que el razonamiento distinga claramente clustered de unclustered.*

### c) V o F (5 pts, 1 pt c/u)

i) **Verdadero.** Unclustered: cada tupla que cumple la condición puede estar en una página distinta, requiriendo un I/O por tupla en el peor caso, además de los accesos al propio índice.

ii) **Verdadero.** B+ clustered es eficiente tanto para igualdad (búsqueda directa) como para rangos (datos contiguos, recorrido secuencial). si dicen que es FALSO porque hash es mas eficiente, tambien vale

iii) **Falso.** Esa es la definición de índice *clustered*; en un índice unclustered el orden físico NO coincide con el orden del índice.

iv) **Verdadero.** La altura depende de $P$ (fan-out) y del número de hojas necesarias (que depende del número de tuplas y $B$).

v) **Verdadero.** Solo puede existir un índice clustered por tabla (determina el orden físico), pero pueden existir múltiples índices unclustered.

## Bonus – Recuperación de Fallas (5 pts, 1 pt c/u)

a) **Falso.** RAID protege contra fallas de disco/hardware a nivel de almacenamiento, no es un mecanismo genérico para fallas de Sistemas Distribuidos.

b) **Falso.** Esa es la definición de **Redo Logging** (registra el valor nuevo). Undo Logging registra el valor **antiguo** de cada escritura.

c) **Verdadero.** Nonquiescent checkpoints permiten que las transacciones sigan ejecutándose durante el checkpoint, sin detener el sistema.

d) **Verdadero.** Redo Logging invierte la lógica de Undo Logging: reglas de escritura a disco opuestas (Undo: log antes del dato en disco, commit después; Redo: log completo antes del commit, datos pueden escribirse después).

e) **Verdadero.** Las copias distribuidas (réplicas geográficas) permiten recuperarse de fallas catastróficas que afectan a un sitio completo, a diferencia del log local.

*Pauta sujeta a criterio del corrector para variantes de respuesta correctamente justificadas, especialmente en P1 (herencia) y P3 (orden exacto de operaciones y análisis de deadlock).*