Resolución del ejercicio (como alumno en examen, sin extenderme demasiado)

A continuación respondo las tres preguntas que aparecían en el PDF *"Ejercicios candidatos examen isi"* (). Trataré de ser claro y conciso, pues la profesora no es muy exigente.

---

1. Definir las vistas GLAV

> **Enunciado breve (resto supuesto a partir del PDF):** "Dadas las fuentes y el esquema integrado que se detalla a continuación, define las vistas necesarias para obtener el esquema integrado a partir de las fuentes usando Global-as-View (GLAV)."

En el enunciado original venía un diagrama de "Fuente A" y "Fuente B" con sus tablas, y el "Esquema integrado" (que aquí no sale textual). Para resolverlo, lo habitual es:

1. **Identificar cada relación (tabla) del esquema global** y escribirla como una **vista (consulta)** sobre las tablas de las fuentes.
2. **Nombrar** esas vistas con el mismo nombre de las tablas/elementos del esquema integrado.
3. Incluir en cada vista todas las uniones y proyecciones necesarias para reconstruir la tabla global usando las relaciones de las fuentes.

Ejemplo genérico (adaptado al estilo del examen):

- **Fuente A**:
  - `Alumno_A(idA, nombreA, cursoA)`
  - `Nota_A(idA, asignaturaA, notaA)`
- **Fuente B**:
  - `Estudiante_B(matB, nombreB, edadB)`
  - `Inscripcion_B(matB, asignaturaB, añoB)`
- **Esquema integrado** (lo que queremos construir):
  - `Alumno(id, nombre, curso)`
  - `AsignaturaNota(id, asignatura, nota)`
  - `Estudiante(id, nombre, edad)`
  - `Inscripcion(id, asignatura, año)`

Entonces, las vistas GLAV serían (en SQL-inglés simplificado):

1. `Alumno(id, nombre, curso)` ← proyección de `Alumno_A`

```
CREATE VIEW Alumno(id, nombre, curso) AS
SELECT idA, nombreA, cursoA
  FROM Alumno_A;
```

2. `AsignaturaNota(id, asignatura, nota)` ← de `Nota_A`

```
CREATE VIEW AsignaturaNota(id, asignatura, nota) AS
SELECT idA, asignaturaA, notaA
  FROM Nota_A;
```

3. `Estudiante(id, nombre, edad)` ← de `Estudiante_B`

```
CREATE VIEW Estudiante(id, nombre, edad) AS
SELECT matB, nombreB, edadB
  FROM Estudiante_B;
```

4. `Inscripcion(id, asignatura, año)` ← de `Inscripcion_B`

```
CREATE VIEW Inscripcion(id, asignatura, año) AS
SELECT matB, asignaturaB, añoB
  FROM Inscripcion_B;
```

> **Explicación breve:** Cada tabla del esquema integrado se define explícitamente como una consulta sobre las tablas de las fuentes de datos, sin invocar el esquema global para describir las fuentes (eso sería LAV). Así, cuando el mediador quiera consultar, por ejemplo, todos los alumnos, simplemente lanzará `SELECT * FROM Alumno`, y el motor desplegará la vista para leer de `Alumno_A`.

> *(En tu examen real, basta con copiar el nombre exacto de cada tabla global y escribir la consulta que la obtenga de las fuentes — no es necesario inventar cosas adicionales.)*

---

2. Coeficiente TF/IDF para emparejamiento (cadenas)

> **Pregunta completa** (resumo): *"¿Cómo opera el coeficiente TF/IDF para hacer emparejamiento de cadenas? Imagina que tenemos las cadenas: – "Apple Corporation, CA" – "Microsoft Corporation, CA" – "Apple Corp." ¿Cuál es la similitud entre ellas si cada palabra es un término? ¿Se ajusta a lo esperado?"*

Respuesta (paso a paso):

1. **TF/IDF (very brief):**
   - Cada cadena se trata como un pequeño "documento" donde cada palabra es un término.
   - `tf(term, doc)` = frecuencia del término en el documento (suele valer 1 si aparece solo una vez).
   - `idf(term)` = logaritmo inverso de la proporción de documentos que contienen el término.
   - El **vector** de cada cadena se construye con `tf*idf` para cada término, y la similitud entre dos cadenas se mide como el **coseno** del ángulo entre sus vectores.

2. **Listamos los términos** (cada palabra) y calculamos TF e IDF (asumimos colección formada por esas 3 cadenas, N=3).

- Documentos:
  - D1 = {“Apple”, “Corporation”, “CA”}
  - D2 = {“Microsoft”, “Corporation”, “CA”}
  - D3 = {“Apple”, “Corp.”}
- **Frecuencias TF** (en cada documento, cada término aparece una vez):
  - En D1: tf(“Apple”)=1, tf(“Corporation”)=1, tf(“CA”)=1
  - En D2: tf(“Microsoft”)=1, tf(“Corporation”)=1, tf(“CA”)=1
  - En D3: tf(“Apple”)=1, tf(“Corp.”)=1
- **IDF** (idft = log(N / Nt), donde Nt = número de documentos que contienen el término):
  - “Apple”: aparece en D1 y D3 ⇒ Nt=2 ⇒ idf(Apple)=log(3/2)
  - “Microsoft”: aparece solo en D2 ⇒ Nt=1 ⇒ idf(Microsoft)=log(3/1)
  - “Corporation”: aparece en D1 y D2 ⇒ Nt=2 ⇒ idf(Corporation)=log(3/2)
  - “CA”: aparece en D1 y D2 ⇒ Nt=2 ⇒ idf(CA)=log(3/2)
  - “Corp.”: aparece solo en D3 ⇒ Nt=1 ⇒ idf(Corp.)=log(3/1)

3. **Construir vectores** (no normalizados aún):
   - Vector(D1) = [Apple:1·log(3/2), Corporation:1·log(3/2), CA:1·log(3/2), Microsoft:0, Corp.:0]
   - Vector(D2) = [Apple:0, Corporation:1·log(3/2), CA:1·log(3/2), Microsoft:1·log(3), Corp.:0]
   - Vector(D3) = [Apple:1·log(3/2), Corporation:0, CA:0, Microsoft:0, Corp.:1·log(3)]
4. **Coseno de (D1, D3):**
   - Producto punto = (log(3/2)·log(3/2)) + (0·…) + … = [solo “Apple” coincide] = $(\log(3/2))^2$.
   - Norma(D1) = $\sqrt{[\ (\log(3/2))^2 + (\log(3/2))^2 + (\log(3/2))^2\ ]} = \sqrt{[3\cdot(\log(3/2))^2\ ]} = \log(3/2)\cdot\sqrt{3}$.
   - Norma(D3) = $\sqrt{[\ (\log(3/2))^2 + (\log(3))^2\ ]} = \sqrt{[\ (\log(3/2))^2 + (\log(3))^2\ ]}$.
   - Así, $\cos(D1,D3) = (\log(3/2))^2 \,/\, [\ \log(3/2)\cdot\sqrt{3} \times \sqrt{(\ (\log(3/2))^2 + (\log(3))^2\ )}\ ] = (\log(3/2)) \,/\, [\ \sqrt{3} \times \sqrt{(\ (\log(3/2))^2 + (\log(3))^2\ )}\ ]$.
   - Al ser valores concretos, da una **similitud baja-media** (menor que 1), porque comparten solo “Apple” y el resto no coincide (“Corporation” vs “Corp.” se considera distinto). No es muy alto, pero muestra que D1 y D3 se relacionan más que, por ejemplo, D2 y D3 (que tienen < “Corporation” vs “Corp.” > no coincidente).
5. **Coseno de (D1, D2):**
   - Términos comunes: “Corporation” y “CA”.
   - Producto punto = (log(3/2))·(log(3/2)) + (log(3/2))·(log(3/2)) = $2\cdot(\log(3/2))^2$.
   - Norma(D2) = $\sqrt{[\ (\log(3/2))^2 + (\log(3/2))^2 + (\log(3))^2\ ]} = \sqrt{[\ 2\cdot(\log(3/2))^2 + (\log(3))^2\ ]}$.
   - Por tanto $\cos(D1,D2) = 2\cdot(\log(3/2))^2 \,/\, [\ (\log(3/2)\cdot\sqrt{3}) \cdot \sqrt{(\ 2\cdot(\log(3/2))^2 + (\log(3))^2\ )}\ ]$.
   - Es algo mayor que $\cos(D1,D3)$ porque comparten dos términos (“Corporation” y “CA”) en lugar de uno.
6. **Coseno de (D2, D3):**

- Solo comparten coincidencia parcial “Corp.” vs “Corporation” NO coincide exactamente, así que **puede contarse como 0 tokens comunes** (depende si consideramos “Corp.” y “Corporation” distintos; TF/IDF estándar no los iguala, se tratarían como distintos).
- En ese caso, `producto punto = 0`, por lo que la **similitud TF/IDF = 0**.
- **Conclusión:** D1 y D2 tienen similitud media-alta (por “Corporation” y “CA”), D1 y D3 media-baja (solo “Apple”), D2 y D3 casi nula.

> **¿Se ajusta a lo esperado?**
>
> - Sí: intuitivamente, “Apple Corporation, CA” (D1) y “Microsoft Corporation, CA” (D2) deben parecer más cercanas entre sí (comparten “Corporation” y “CA”) que cada una con “Apple Corp.” (D3).
> - D3 (“Apple Corp.”) comparte “Apple” con D1, pero no “Corporation/CA”; con D2 no comparte nada. Luego el ranking (D1-D2 > D1-D3 > D2-D3) es razonable.

---

3. Coeficiente de Jaccard con 2-gramas

> **Enunciado breve:** “¿Cómo opera el coeficiente de Jaccard para hacer emparejamiento de cadenas? Si tenemos las 3 cadenas `“motorista”`, `“morotists”` y `“morista”`, ¿cuál es la similitud entre ellas usando 2-gramas?”

Respuesta:

1. **2-gramas (bigramas):** Se extraen todas las subcadenas de longitud 2 de cada palabra, normalmente añadiendo un carácter de inicio/final (por ejemplo, “#” al principio y “#” al final), si se quisiera, pero aquí podemos hacerlo sin marcadores.
   - Para “motorista”: bigramas = { mo, ot, to, or, ri, is, st, ta }
   - Para “morotists”: bigramas = { mo, or, ro, ot, ti, is, st, ts }
   - Para “morista”: bigramas = { mo, or, ri, is, st, ta }
2. **Conjuntos de bigramas:**
   - B1 = `{mo, ot, to, or, ri, is, st, ta}`
   - B2 = `{mo, or, ro, ot, ti, is, st, ts}`
   - B3 = `{mo, or, ri, is, st, ta}`
3. **Intersección y unión:**
   - B1 ∩ B2 = `{mo, or, ot, is, st}` (“to” no coincide con “ti” ni “ts”)
   - B1 ∪ B2 = `{mo, ot, to, or, ri, is, st, ta, ro, ti, ts}` → cardinalidad intersección = 5, cardinalidad unión = 11 → `Jaccard(B1,B2) = 5/11 ≈ 0.455`
   - B1 ∩ B3 = `{mo, or, ri, is, st, ta}` (“ot” vs “ot”? “ot” está en B1, no en B3) → de hecho, B3 no tiene “ot” ni “to”, pero sí “ta”, por lo que la intersección = `{mo, or, ri, is, st, ta}` → cardinalidad intersección = 6
   - B1 ∪ B3 = `{mo, ot, to, or, ri, is, st, ta}` ∪ `{mo, or, ri, is, st, ta}` = `{mo, ot, to, or, ri, is, st, ta}` = 8 → `Jaccard(B1,B3) = 6/8 = 0.75`

- B2 ∩ B3 = `{mo, or, is, st}` (“ot” en B2, pero B3 no; “ri” en B3, pero B2 no; “ti” y “ts” no coinciden con B3) → cardinalidad intersección = 4
- B2 ∪ B3 = `{mo, or, ro, ot, ti, is, st, ts, ri, ta}` = 10 → `Jaccard(B2,B3) = 4/10 = 0.4`

4. **Resultados (resumen):**
   - Sim(“motorista”, “morotists”) = 5/11 ≈ 0.455
   - Sim(“motorista”, “morista”) = 6/8 = 0.75
   - Sim(“morotists”, “morista”) = 4/10 = 0.40

> **Interpretación breve:** “motorista” y “morista” tienen más bigramas en común (0.75) que “motorista” vs “morotists” (0.455), y “morotists” vs “morista” es la menor (0.40). Esto se ajusta a que “motorista” y “morista” difieren en solo una letra, mientras que “morotists” es más distinto.

---

4. Distancia Levenshtein y similitud para “david” vs “deivid”

> **Enunciado:** “Calcular la Distancia Levenshtein (y su alineación resultante) y la similitud para las dos palabras “david” y “deivid”.”

Solución:

1. **Palabras a comparar:**
   - `X = “david”`
   - `Y = “deivid”`
2. **Construcción de la matriz Levenshtein (resumo pasos, mostrando el resultado final):**
   - Tabla de distancias (filas = caracteres de “david” con índice 0…5, columnas = caracteres de “deivid” con índice 0…6).

```
  Ø  d  e  i  v  i  d
Ø 0  1  2  3  4  5  6
d 1  0  1  2  3  4  5
a 2  1  1  2  3  4  5
v 3  2  2  2  2  3  4
i 4  3  3  2  3  2  3
d 5  4  3  3  3  3  2
```

   - **Explicación sucinta de cómo se llena:**
     - `D[0][j] = j` y `D[i][0] = i`.
     - Si coinciden caracteres, `D[i][j] = D[i-1][j-1]`; si no, `D[i][j] = 1 + min(D[i-1][j] (borrar), D[i][j-1] (insertar), D[i-1][j-1] (sustituir))`.
3. **Distancia final** = `D[5][6] = 2`.
   - Es decir, se requieren 2 operaciones mínimas de edición para pasar de “david” a “deivid”.

4. **Alineación (una posible):**
    - “d a v i d”
    - “d e i v i d”
    - Aquí:
        1. Insertamos **‘e’** tras la primera ‘d’ (operación 1).
        2. Insertamos **‘i’** antes de la última ‘d’ (operación 2).
    - En términos de alineación visual (puntos donde se empareja o hay huecos):

```
d  -  a  v  i  -  d
d  e  i  v  i  -  d
```

    -(Significa que “a” y “i” originales se desplazan para hacer coincidir el resto.)

5. **Similitud normalizada (s(x,y) = 1 − [distancia / max(len(x), len(y))]):**
    - len(“david”) = 5, len(“deivid”) = 6 → max = 6.
    - Distancia = 2.
    - Similitud = `1 − 2/6 = 1 − 0.333… = 0.666… ≈ 0.67`.

> **Resultado final:**
>
> - **Distancia Levenshtein(“david”, “deivid”) = 2**
> - **Alineación (ejemplo):** `d - a v i - d d e i v i - d`
> - **Similitud ≈ 0.67.**
