

Ejercicios resueltos de Bases de Datos

Los 38 ejercicios de Bases de Datos de la web, tema a tema: cada uno con su enunciado, los datos que necesitas (código de partida, ejemplos de entrada y salida o la base de datos) y, al final, la solución explicada.

Índice

Sistemas de Almacenamiento

[1. Fichero secuencial frente a índice](#)

[3. Anonimizar datos personales: k-anonimato](#)

Modelo Entidad-Relación

[4. Comprobar cardinalidades con datos reales](#)

[6. Claves candidatas y forma normal](#)

Modelo Relacional

[7. ¿Qué columnas pueden ser clave?](#)

[9. Un intérprete de álgebra relacional](#)

Consultas SQL Avanzadas

[10. Encuentra el fallo: los clientes sin email](#)

[12. Encuentra el fallo: la media que sale entera](#)

[14. Encuentra el fallo: el LEFT JOIN que pierde clientes](#)

Modificación de Datos en SQL

[16. Encuentra el fallo: la actualización que no actualiza nada](#)

Programación en Bases de Datos

[18. Encuentra el fallo: el disparador que descuenta a todos](#)

Ejercicios de Bases de Datos

[19. Productos baratos](#)

[21. Búsqueda por patrón](#)

[23. Estadísticas del catálogo](#)

[25. Importe de cada pedido](#)

[27. Ciudades de clientes y proveedores](#)

[29. Crear una vista](#)

[31. Mejor cliente por gasto](#)

[33. Borrado con subconsulta](#)

Ejercicios largos

[35. La base de datos de una academia de idiomas](#)

[37. Liga de baloncesto: clasificación y funciones de ventana](#)

[2. Fragmentación de una base de datos distribuida](#)

[5. Del diagrama E/R a las tablas](#)

[8. Integridad referencial al borrar e insertar](#)

[11. Encuentra el fallo: las prioridades de AND y OR](#)

[13. Encuentra el fallo: el pedido fantasma](#)

[15. Encuentra el fallo: las bajas que lo borran todo](#)

[17. Encuentra el fallo: el stock que se vuelve NULL](#)

[20. Clientes sin email](#)

[22. Alta de un producto](#)

[24. Pedidos con su cliente](#)

[26. Clientes que nunca han comprado](#)

[28. Subida de precios](#)

[30. Productos más caros que la media de su categoría](#)

[32. Categorías con más de 100 € vendidos](#)

[34. Trigger de auditoría](#)

[36. Clínica: citas, especialidades y facturación](#)

[38. Banco: vistas, disparadores, UPSERT y transacciones](#)

Base de datos de la tienda. Los ejercicios de SQL que no traen su propia base de datos trabajan sobre esta: clientes, productos, pedidos y sus líneas.

Base de datos de la tienda (sql)

```
CREATE TABLE Clientes (  
  cliente_id INTEGER PRIMARY KEY,  
  nombre TEXT NOT NULL,  
  email TEXT UNIQUE,  
  ciudad TEXT,  
  fecha_registro DATE  
);  
  
CREATE TABLE Productos (  
  producto_id INTEGER PRIMARY KEY,  
  nombre TEXT NOT NULL,  
  categoria TEXT,  
  precio REAL CHECK (precio >= 0),  
  stock INTEGER DEFAULT 0,  
  proveedor TEXT,  
  fecha_alta DATE  
);  
  
CREATE TABLE Pedidos (  
  pedido_id INTEGER PRIMARY KEY,  
  cliente_id INTEGER REFERENCES Clientes(cliente_id),  
  fecha DATE,  
  estado TEXT DEFAULT 'pendiente'  
);  
  
CREATE TABLE DetallePedido (  
  pedido_id INTEGER REFERENCES Pedidos(pedido_id),  
  producto_id INTEGER REFERENCES Productos(producto_id),  
  cantidad INTEGER NOT NULL,  
  precio_unitario REAL NOT NULL,  
  PRIMARY KEY (pedido_id, producto_id)  
);  
  
INSERT INTO Clientes VALUES  
(1, 'Ana García', 'ana@email.com', 'Madrid', '2024-01-15'),  
(2, 'Carlos López', 'carlos@email.com', 'Barcelona', '2024-02-20'),  
(3, 'Lucía Martín', 'lucia@email.com', 'Madrid', '2024-03-05'),  
(4, 'Javier Ruiz', 'javier@email.com', 'Sevilla', '2024-04-10'),  
(5, 'Marta Díaz', NULL, 'Valencia', '2024-05-22');  
  
INSERT INTO Productos VALUES  
(1, 'Portátil HP', 'Informática', 899.99, 15, 'HP', '2023-09-01'),  
(2, 'Ratón Logitech', 'Periféricos', 24.99, 120, 'Logitech', '2023-09-01'),  
(3, 'Teclado mecánico', 'Periféricos', 89.90, 45, 'Logitech', '2023-11-12'),  
(4, 'Monitor 27"', 'Informática', 279.00, 20, 'Samsung', '2024-01-08'),  
(5, 'Auriculares BT', 'Audio', 59.95, 0, 'Sony', '2024-02-14'),  
(6, 'Webcam HD', 'Periféricos', 39.99, 60, 'Logitech', '2024-03-01'),  
(7, 'Altavoz portátil', 'Audio', 45.50, 30, 'JBL', '2024-04-18'),  
(8, 'Disco SSD 1TB', 'Informática', 99.00, 75, 'Samsung', '2024-05-02');  
  
INSERT INTO Pedidos VALUES  
(1, 1, '2024-06-01', 'entregado'),  
(2, 2, '2024-06-03', 'entregado'),  
(3, 1, '2024-06-10', 'enviado'),  
(4, 3, '2024-06-12', 'pendiente'),  
(5, 4, '2024-06-15', 'entregado');  
  
INSERT INTO DetallePedido VALUES  
(1, 1, 1, 899.99),  
(1, 2, 2, 24.99),  
(2, 4, 2, 279.00),  
(3, 3, 1, 89.90),  
(3, 6, 1, 39.99),  
(4, 8, 3, 99.00),  
(5, 5, 2, 59.95),  
(5, 7, 1, 45.50);
```

1. Fichero secuencial frente a índice

Medio · Sistemas de Almacenamiento · apuntesdam.com/subject/bases-datos/topic/almacenamiento

¿Por qué una base de datos con índice encuentra un registro entre millones casi al instante? La entrada tiene registros «id nombre» en el orden en que están en el fichero (desordenados) y búsquedas «buscar id». El índice ya está construido: una lista ordenada de (id, posición). Para cada búsqueda cuenta las lecturas de la búsqueda secuencial en el fichero (se leen registros desde el principio hasta encontrarlo; si no está, todos) y las comparaciones de una búsqueda binaria en el índice ($lo = 0$, $hi = n-1$; mientras $lo \leq hi$: $mid = (lo+hi)//2$, se cuenta una comparación y se mira si es el buscado, si es menor o si es mayor), más una lectura del registro si se encuentra. Muestra «buscar x: encontrado (nombre) · secuencial S lecturas · índice C comparaciones + 1 lectura» (o «no existe» y sin «+ 1 lectura»; en singular «1 lectura» y «1 comparación») y al final «Total secuencial: S lecturas · índice: I accesos · ahorro P %» con $P = 100 - 100 \cdot I/S$ redondeado.

Código de partida (python)

```
import sys

registros = []          # (id, nombre) en el orden del fichero (sin ordenar)
búsquedas = []
for linea in sys.stdin.read().splitlines():
    p = linea.split()
    if not p:
        continue
    if p[0] == "buscar":
        búsquedas.append(int(p[1]))
    else:
        registros.append((int(p[0]), p[1]))

indice = sorted((r[0], pos) for pos, r in enumerate(registros))  # (id, posición en el fichero)
# TODO: para cada búsqueda, contar lecturas secuenciales y comparaciones de la búsqueda binaria en el
# índice
```

Ejemplo: 40 registros

ENTRADA

```
431 Ana0
254 Luis1
504 Eva2
766 Juan3
149 Marta4
174 Pablo5
940 Lucía6
648 Hugo7
196 Sara8
474 Iker9
696 Ana10
159 Luis11
619 Eva12
319 Juan13
138 Marta14
188 Pablo15
544 Lucía16
528 Hugo17
171 Sara18
346 Iker19
192 Ana20
664 Luis21
534 Eva22
160 Juan23
946 Marta24
679 Pablo25
226 Lucía26
328 Hugo27
745 Sara28
742 Iker29
163 Ana30
690 Luis31
699 Eva32
506 Juan33
150 Marta34
326 Pablo35
147 Lucía36
670 Hugo37
979 Sara38
236 Iker39
buscar 431
buscar 679
buscar 236
buscar 5
buscar 474
```

SALIDA ESPERADA

```
buscar 431: encontrado (Ana0) · secuencial 1
lectura · índice 1 comparación + 1 lectura
buscar 679: encontrado (Pablo25) · secuencial 26
lecturas · índice 5 comparaciones + 1 lectura
buscar 236: encontrado (Iker39) · secuencial 40
lecturas · índice 6 comparaciones + 1 lectura
buscar 5: no existe · secuencial 40 lecturas ·
índice 5 comparaciones
buscar 474: encontrado (Iker9) · secuencial 10
lecturas · índice 5 comparaciones + 1 lectura
Total secuencial: 117 lecturas · índice: 26
accesos · ahorro 78 %
```

2. Fragmentación de una base de datos distribuida

Difícil · Sistemas de Almacenamiento · apuntesdam.com/subject/bases-datos/topic/almacenamiento

En una base de datos distribuida, una tabla se reparte en fragmentos guardados en distintas sedes. La entrada define la tabla («tabla nombre col1,col2...»), sus filas («fila v1,v2,...»), fragmentos horizontales («fragmento F sede=S campo=v1,v2»: las filas cuyo campo vale uno de esos valores) y consultas. Muestra «F (sede): n filas» por fragmento («1 fila» en singular, también en las consultas); «Compleitud: sí» o «Compleitud: NO, sin fragmento: id X (valor), ...» (las filas que no están en ningún fragmento se pierden; el valor es el del campo del primer fragmento); «Disjunción: sí» o «Disjunción: NO, id X en F1 y F4; ...» (filas en varios fragmentos). Para cada consulta («campo=valor», «campo>número», «campo<número» o «todos»), decide a qué fragmentos hay que preguntar: si la condición es una igualdad sobre el campo de fragmentación, solo a los que incluyen ese valor; si no, a todos. Muestra «consulta C → sedes S1, S2 · n filas: ids» (sedes sin repetir en el orden de los fragmentos, ids ordenados y con repeticiones si una fila está en dos fragmentos; «ninguna» y sin lista si no hay).

```
import sys

columnas, filas, fragmentos, consultas = [], [], [], []
for linea in sys.stdin.read().splitlines():
    p = linea.split(" ", 1)
    if p[0] == "tabla":
        columnas = p[1].split(" ")[1].split(",")
    elif p[0] == "fila":
        valores = p[1].split(",")
        filas.append(dict(zip(columnas, valores)))
    elif p[0] == "fragmento":
        # fragmento F1 sede=Madrid provincia=Madrid,Toledo
        nombre, sede, cond = p[1].split(" ")
        campo, valores = cond.split("=")
        fragmentos.append((nombre, sede.split("=")[1], campo, valores.split(",")
    elif p[0] == "consulta":
        consultas.append(p[1])

# TODO: filas de cada fragmento, completitud, disjunción y consultas
```

Ejemplo: Clientes por provincia

ENTRADA

```
tabla clientes id,nombre,provincia,saldo
fila 1,Ana,Madrid,1200
fila 2,Luis,Sevilla,300
fila 3,Eva,Cádiz,800
fila 4,Iker,Bizkaia,50
fila 5,Marta,Madrid,90
fila 6,Pablo,Lugo,700
fila 7,Sara,Huelva,610
fila 8,Hugo,Toledo,20
fragmento F1 sede=Madrid provincia=Madrid,Toledo
fragmento F2 sede=Sevilla
provincia=Sevilla,Cádiz,Huelva
fragmento F3 sede=Bilbao provincia=Bizkaia
fragmento F4 sede=Sevilla provincia=Toledo
consulta provincia=Cádiz
consulta saldo>500
consulta provincia=Lugo
consulta todos
```

SALIDA ESPERADA

```
F1 (Madrid): 3 filas
F2 (Sevilla): 3 filas
F3 (Bilbao): 1 fila
F4 (Sevilla): 1 fila
Completitud: NO, sin fragmento: id 6 (Lugo)
Disjunción: NO, id 8 en F1 y F4
consulta provincia=Cádiz → sedes Sevilla · 1
fila: 3
consulta saldo>500 → sedes Madrid, Sevilla,
Bilbao · 3 filas: 1, 3, 7
consulta provincia=Lugo → sedes ninguna · 0 filas
consulta todos → sedes Madrid, Sevilla, Bilbao ·
8 filas: 1, 2, 3, 4, 5, 7, 8, 8
```

3. Anonimizar datos personales: k-anonimato

Muy difícil · Sistemas de Almacenamiento · apuntesdam.com/subject/bases-datos/topic/almacenamiento

El RGPD permite publicar o compartir datos para estudios si están anonimizados, pero quitar el nombre no basta: la edad, el código postal y el sexo juntos identifican a mucha gente. La entrada empieza con «k K» y una tabla «nombre;dni;edad;cp;sexo;diagnostico». Anonimiza: elimina nombre y dni («Identificadores eliminados: nombre, dni»); generaliza la edad a su década («34» → «30-39») y el código postal a sus tres primeras cifras («28013» → «280**»). Los cuasi-identificadores son (edad, cp, sexo): las filas con los mismos valores forman un grupo, y k es el tamaño del grupo más pequeño («k antes de suprimir: n»). Suprime las filas de los grupos con menos de K filas; si no queda ninguna, «No queda ningún dato publicable» y termina. Si quedan, muestra la cabecera «edad;cp;sexo;diagnostico», las filas en su orden original y «Grupos: G · k = k (objetivo K) · filas suprimidas: s · l-diversidad: l», donde l es el menor número de diagnósticos distintos en un grupo; si l es 1, añade «Atención: en algún grupo todos tienen el mismo diagnóstico, que se puede deducir».

Código de partida (python)

```
import sys
from collections import Counter, defaultdict

lineas = [l for l in sys.stdin.read().splitlines() if l.strip()]
k_objetivo = int(lineas[0].split()[1])
cabecera = lineas[1].split(";") # nombre;dni;edad;cp;sexo;diagnostico
filas = [dict(zip(cabecera, l.split(";"))) for l in lineas[2:]]

# TODO: eliminar identificadores, generalizar, calcular k y l, y suprimir los grupos pequeños
```

Ejemplo: Estudio médico

ENTRADA

```
k 3
nombre;dni;edad;cp;sexo;diagnostico
Ana López;12345678Z;34;28013;M;asma
Luis Gil;87654321X;37;28015;H;diabetes
Eva Ruiz;11111111H;31;28019;M;migraña
Marta Sanz;22222222J;38;28011;M;asma
Pablo Díaz;33333333P;52;41004;H;hipertensión
Iker Arana;44444444A;55;41007;H;hipertensión
Hugo Vidal;55555555K;58;41002;H;hipertensión
Sara Mora;66666666Q;23;08001;M;alergia
Lucía Paz;77777777B;36;28030;M;diabetes
```

SALIDA ESPERADA

```
Identificadores eliminados: nombre, dni
k antes de suprimir: 1
edad;cp;sexo;diagnostico
30-39;280*;M;asma
30-39;280*;M;migraña
30-39;280*;M;asma
50-59;410*;H;hipertensión
50-59;410*;H;hipertensión
50-59;410*;H;hipertensión
30-39;280*;M;diabetes
Grupos: 2 · k = 3 (objetivo 3) · filas
suprimidas: 2 · l-diversidad: 1
Atención: en algún grupo todos tienen el mismo
diagnóstico, que se puede deducir
```

Modelo Entidad-Relación

4. Comprobar cardinalidades con datos reales

Medio · Modelo Entidad-Relación · apuntesdam.com/subject/bases-datos/topic/modelo-er

En el modelo E/R, la cardinalidad (mínimo, máximo) de una entidad en una relación dice en cuántas ocurrencias de la relación participa cada una de sus instancias. La entrada define las instancias de dos entidades («entidad Nombre i1 i2 ...»), la relación («relacion nombre Entidad1(min,max) Entidad2(min,max)», donde max puede ser n) y las ocurrencias («nombre instancia1 instancia2»). Revisa las ocurrencias en orden: si alguna instancia no existe, «Ocurrencia desconocida: a b»; si el par ya apareció, «Par repetido: a b» (una relación no repite la misma pareja); ninguna de las dos cuenta. Muestra «Ocurrencias válidas de nombre: n» y, para cada instancia de la primera entidad y después de la segunda, «Entidad i participa k veces en nombre (mínimo m)» o «(máximo M)» si se sale de su cardinalidad. Termina con «Restricciones: correctas» o «Restricciones: N incumplimientos» («1 incumplimiento»; contando también desconocidas y repetidas).

Código de partida (python)

```
import sys

instancias = {}          # entidad → lista de ocurrencias
relacion = None          # (nombre, entidad1, (min1, max1), entidad2, (min2, max2))
pares = []
for linea in sys.stdin.read().splitlines():
    p = linea.split()
    if not p:
        continue
    if p[0] == "entidad":
        instancias[p[1]] = p[2:]
    elif p[0] == "relacion":          # relacion cursa Alumno(1,n) Modulo(0,2)
        lados = []
        for t in p[2:4]:
            nombre, card = t[:-1].split("(")
            mn, mx = card.split(",")
            lados.append((nombre, (int(mn), None if mx == "n" else int(mx))))
        relacion = (p[1], lados[0][0], lados[0][1], lados[1][0], lados[1][1])
    else:
        pares.append((p[1], p[2]))          # cursa A1 BD

# TODO: comprobar pares repetidos, instancias desconocidas y la participación mínima y máxima
```

Ejemplo: Alumnos y módulos

ENTRADA

```
entidad Alumno A1 A2 A3 A4
entidad Modulo BD PROG LM
relacion cursa Alumno(1,n) Modulo(0,2)
cursa A1 BD
cursa A2 BD
cursa A3 BD
cursa A1 PROG
cursa A1 BD
cursa A5 LM
cursa A2 LM
```

SALIDA ESPERADA

```
Par repetido: A1 BD
Ocurrencia desconocida: A5 LM
Ocurrencias válidas de cursa: 5
Alumno A4 participa 0 veces en cursa (mínimo 1)
Modulo BD participa 3 veces en cursa (máximo 2)
Restricciones: 4 incumplimientos
```

5. Del diagrama E/R a las tablas

Difícil · Modelo Entidad-Relación · apuntesdam.com/subject/bases-datos/topic/modelo-er

Aplica las reglas de transformación del modelo E/R al relacional. La entrada tiene entidades «entidad Nombre *clave atributo ...» (los atributos con * forman la clave primaria) y relaciones binarias «relacion nombre Entidad1 c1 Entidad2 c2 [atributos]» con c1 y c2 iguales a 1, N o M. Reglas: cada entidad es una tabla con sus atributos; en una 1:N (o N:1) la clave de la entidad del lado 1 pasa como clave ajena a la del lado N, junto con los atributos de la relación; en una 1:1 la clave de la primera pasa a la segunda como clave ajena única; una N:M crea una tabla con el nombre de la relación cuya clave primaria es la unión de las dos claves (ambas son claves ajenas), más sus atributos. Si la relación es reflexiva (la misma entidad en los dos lados), la clave ajena se llama clave_relacion; y si la tabla ya tiene una columna con el nombre de la clave ajena, se llama clave_entidad (en minúsculas). Muestra las tablas de las entidades en orden de definición (con las columnas añadidas por las relaciones en el orden de las relaciones) y después las nuevas, con el formato «Tabla(*clave, columna, fk→Tabla)» («*» delante de las columnas de la clave primaria y «fk→Tabla único» en las 1:1), y al final «Tablas: T (k por relaciones N:M)».

Código de partida (python)

```
import sys

entidades = {}          # nombre → (claves, atributos) en orden
relaciones = []         # (nombre, e1, card1, e2, card2, atributos)
for linea in sys.stdin.read().splitlines():
    p = linea.split()
    if not p:
        continue
    if p[0] == "entidad":          # entidad Cliente *dni nombre email
        claves = [a[1:] for a in p[2:] if a.startswith("*")]
        attrs = [a.lstrip("*") for a in p[2:]]
        entidades[p[1]] = (claves, attrs)
    elif p[0] == "relacion":       # relacion realiza Cliente 1 Pedido N [atributos...]
        relaciones.append((p[1], p[2], p[3], p[4], p[5], p[6:]))

# TODO: transformar a tablas y mostrar cada una como Tabla(*clave, columna, fk→Tabla)
```

Ejemplo: Tienda online

ENTRADA

```
entidad Cliente *dni nombre email
entidad Pedido *num fecha
entidad Producto *codigo nombre precio
entidad Tarjeta *numero caducidad
relacion realiza Cliente 1 Pedido N
relacion contiene Pedido N Producto M cantidad
precio_unidad
relacion tiene Cliente 1 Tarjeta 1
```

SALIDA ESPERADA

```
Cliente(*dni, nombre, email)
Pedido(*num, fecha, dni→Cliente)
Producto(*codigo, nombre, precio)
Tarjeta(*numero, caducidad, dni→Cliente único)
contiene(*num→Pedido, *codigo→Producto, cantidad,
precio_unidad)
Tablas: 5 (1 por relaciones N:M)
```

6. Claves candidatas y forma normal

Muy difícil · Modelo Entidad-Relación · apuntesdam.com/subject/bases-datos/topic/modelo-er

Normalizar empieza por saber qué claves tiene una relación y qué dependencias rompen cada forma normal. La primera línea es el esquema «R(A,B,C...)» y las siguientes, dependencias funcionales «X1,X2 → Y1,Y2». Implementa cierre(x) (todos los atributos que se deducen de x aplicando las dependencias hasta que no cambie nada) y después: «Claves candidatas: {...}, {...}» (los conjuntos mínimos cuyo cierre es todo el esquema, por tamaño y en el orden de combinations sobre el esquema) y «Atributos primos: ...» (los que están en alguna clave, en orden del esquema, o «ninguno»). Para cada dependencia, en orden: si la derecha está contenida en la izquierda, «X → Y: trivial»; si X es superclave, «X → Y: correcta (X es superclave)»; si todo lo que determina (sin contar X) es primo, «incumple FNBC (X no es superclave)»; si X es una parte propia de alguna clave, «incumple 2FN (dependencia parcial de la clave {...})» (la primera clave que la contenga); si no, «incumple 3FN (dependencia transitiva)». Termina con «Forma normal: F», la más alta que se cumple: FNBC si no hay incumplimientos; 3FN si solo se incumple FNBC; 2FN si hay alguna transitiva pero no parciales; 1FN si hay alguna parcial.

Código de partida (python)

```
import sys
from itertools import combinations

lineas = [l.strip() for l in sys.stdin.read().splitlines() if l.strip()]
atributos = lineas[0][lineas[0].index("(") + 1:-1].split(",") # R(A,B,C,D)
dependencias = []
for l in lineas[1:]:
    izq, der = l.split("->")
    dependencias.append([a.strip() for a in izq.split(",")], [a.strip() for a in der.split(",")]))

def ordenar(conjunto):
    return [a for a in atributos if a in conjunto]

def cierre(x):
    # TODO: atributos que se deducen a partir de x con las dependencias
    return set(x)

# TODO: claves candidatas, atributos primos, análisis de cada dependencia y forma normal
```

Ejemplo: Matrícula

ENTRADA

```
R(alumno,modulo,nota,nombre,profesor,despacho)
alumno,modulo -> nota
alumno -> nombre
modulo -> profesor
profesor -> despacho
```

SALIDA ESPERADA

```
Claves candidatas: {alumno,modulo}
Atributos primos: alumno, modulo
alumno,modulo -> nota: correcta (alumno,modulo es superclave)
alumno -> nombre: incumple 2FN (dependencia parcial de la clave {alumno,modulo})
modulo -> profesor: incumple 2FN (dependencia parcial de la clave {alumno,modulo})
profesor -> despacho: incumple 3FN (dependencia transitiva)
Forma normal: 1FN
```

Ejemplo: Transitiva

ENTRADA

```
R(dni,nombre,cp,ciudad)
dni -> nombre,cp
cp -> ciudad
```

SALIDA ESPERADA

```
Claves candidatas: {dni}
Atributos primos: dni
dni -> nombre,cp: correcta (dni es superclave)
cp -> ciudad: incumple 3FN (dependencia transitiva)
Forma normal: 2FN
```

Ejemplo: Casi FNBC

ENTRADA

```
R(A,B,C)
A,B -> C
C -> B
```

SALIDA ESPERADA

```
Claves candidatas: {A,B}, {A,C}
Atributos primos: A, B, C
A,B -> C: correcta (A,B es superclave)
C -> B: incumple FNBC (C no es superclave)
Forma normal: 3FN
```

Modelo Relacional

7. ¿Qué columnas pueden ser clave?

Medio · Modelo Relacional · apuntesdam.com/subject/bases-datos/topic/modelo-relacional

Antes de crear una tabla a partir de datos existentes (una hoja de cálculo, un CSV) hay que decidir la clave primaria. La entrada es un CSV sencillo: la cabecera y las filas, con los valores vacíos como nulos. Muestra para cada columna «col: k valores distintos de n» (sin contar nulos), «Columnas con nulos: ...» (o «ninguna»; una clave primaria no admite nulos) y «Claves candidatas según los datos: {a}, {b,c}»: las combinaciones de columnas sin nulos, de hasta 3 columnas, cuyos valores no se repiten en ninguna fila y que no contienen otra ya encontrada (mínimas), por tamaño y en el orden de las columnas; si no hay, «ninguna con hasta 3 columnas». Termina con el aviso «Aviso: los datos solo descartan claves; que hoy no haya repetidos no garantiza que no los haya mañana».

Código de partida (python)

```
import sys
from itertools import combinations

lineas = [l for l in sys.stdin.read().splitlines() if l.strip()]
columnas = lineas[0].split(",")
filas = [[v if v != "" else None for v in l.split(",")] for l in lineas[1:]]

# TODO: valores distintos por columna, columnas con nulos y combinaciones únicas mínimas (hasta 3 columnas)
```

Ejemplo: Alumnos

ENTRADA

```
dni,nombre,email,telefono,ciudad,curso
12345678Z,Ana
García,ana@correo.es,600111222,Logroño,1DAM
87654321X,Luis Pérez,luis@correo.es,,Haro,1DAM
11111111H,Ana
García,ana.g@correo.es,600333444,Logroño,2DAM
22222222J,Marta
Gil,marta@correo.es,600555666,Arnedo,1DAM
33333333P,Luis
Pérez,lperez@correo.es,600777888,Logroño,2DAM
```

SALIDA ESPERADA

```
dni: 5 valores distintos de 5
nombre: 3 valores distintos de 5
email: 5 valores distintos de 5
telefono: 4 valores distintos de 5
ciudad: 3 valores distintos de 5
curso: 2 valores distintos de 5
Columnas con nulos: telefono
Claves candidatas según los datos: {dni},
{email}, {nombre,curso}
Aviso: los datos solo descartan claves; que hoy
no haya repetidos no garantiza que no los haya
mañana
```

8. Integridad referencial al borrar e insertar

Difícil · Modelo Relacional · apuntesdam.com/subject/bases-datos/topic/modelo-relacional

Una clave ajena impide que queden filas «huérfanas», y su regla ON DELETE decide qué pasa al borrar la fila referenciada: RESTRICT (se impide el borrado), CASCADE (se borran también las filas que la referencian) o SET NULL (su clave ajena pasa a NULL). La entrada define tablas («tabla nombre id col...»; la primera columna es siempre id), claves ajenas («fk tabla.columna -> otra.id ON DELETE REGLA»), filas («fila tabla valores...», NULL para nulo) y órdenes. «borrar tabla id» funciona como una transacción: o se hace con todos sus efectos o no se hace nada; muestra «borrar t id → borradas: t#id, hija#id, ...» (en el orden en que se borran, primero la fila pedida y después, recursivamente, sus dependientes) y, si hay, «; a NULL: tabla#id.columna, ...», o «borrar t id → ERROR: no se puede borrar t id: lo usan n filas de hija» (o «no existe t id»). «insertar tabla valores...» comprueba que el id no exista («ERROR: ya existe t id») y que cada clave ajena no nula apunte a una fila existente («ERROR: no existe otra v»); si no, «correcto». «mostrar» lista cada tabla «nombre: v1 v2 | v1 v2» o «(vacía)».

Código de partida (python)

```
import sys

tablas = {}          # nombre -> (columnas, {id: fila como dict})
claves_ajenas = []   # (tabla, columna, tabla_ref, regla)
ordenes = []
for linea in sys.stdin.read().splitlines():
    p = linea.split()
    if not p:
        continue
    if p[0] == "tabla":
        tablas[p[1]] = (p[2:], {})
    elif p[0] == "fk":          # fk pedido.cliente_id -> cliente.id ON DELETE CASCADE
        t, c = p[1].split(".")
        claves_ajenas.append((t, c, p[3].split(".")[0], " ".join(p[6:])))
    elif p[0] == "fila":
        cols, filas = tablas[p[1]]
        fila = dict(zip(cols, [None if v == "NULL" else v for v in p[2:])))
        filas[fila["id"]] = fila
    else:
        ordenes.append(p)

# TODO: procesar borrar, insertar y mostrar respetando las claves ajenas
```

Ejemplo: Tienda

ENTRADA

```
tabla cliente id nombre
tabla pedido id cliente_id fecha
tabla linea id pedido_id producto
tabla factura id pedido_id importe
fk pedido.cliente_id -> cliente.id ON DELETE
CASCADE
fk linea.pedido_id -> pedido.id ON DELETE CASCADE
fk factura.pedido_id -> pedido.id ON DELETE
RESTRICT
fila cliente 1 Ana
fila cliente 2 Luis
fila cliente 3 Eva
fila pedido 10 1 2025-09-01
fila pedido 11 2 2025-09-03
fila pedido 12 2 2025-09-05
fila linea 100 10 Teclado
fila linea 101 11 Ratón
fila linea 102 12 Monitor
fila linea 103 12 Cable
fila factura 500 10 45.90
borrar cliente 1
borrar cliente 2
insertar pedido 13 9 2025-10-01
insertar pedido 13 3 2025-10-01
insertar pedido 13 3 2025-10-02
borrar linea 999
mostrar
```

SALIDA ESPERADA

```
borrar cliente 1 -> ERROR: no se puede borrar
pedido 10: lo usan 1 fila de factura
borrar cliente 2 -> borradas: cliente#2,
pedido#11, linea#101, pedido#12, linea#102,
linea#103
insertar pedido 13 -> ERROR: no existe cliente 9
insertar pedido 13 -> correcto
insertar pedido 13 -> ERROR: ya existe pedido 13
borrar linea 999 -> ERROR: no existe linea 999
cliente: 1 Ana | 3 Eva
pedido: 10 1 2025-09-01 | 13 3 2025-10-01
linea: 100 10 Teclado
factura: 500 10 45.90
```

9. Un intérprete de álgebra relacional

Muy difícil · Modelo Relacional · apuntesdam.com/subject/bases-datos/topic/modelo-relacional

El álgebra relacional es la base teórica de SQL: cada consulta es una composición de operaciones sobre relaciones que devuelven relaciones. Programa un intérprete. La entrada define tablas («tabla nombre c1,c2,...», sus filas con valores separados por comas y «fin») y después órdenes «R = operación ...» y «mostrar R». Operaciones: «seleccion T col=valor» (también <, > y <; compara como número si los dos lo son); «proyeccion T c1,c2» (solo esas columnas y sin tuplas repetidas, conservando la primera aparición); «join T1 T2 a=b» (combina cada tupla de T1 con las de T2 que cumplan T1.a = T2.b; las columnas de T2 que ya existen en T1 se llaman T2.col); «union», «diferencia» e

«interseccion» de dos relaciones con las mismas columnas en el mismo orden (si no, «esquemas incompatibles (...) y (...)»), sin repetidos; y «renombrar T viejo=nuevo». Si falta una relación o una columna, muestra «ERROR en R: no existe X» o «ERROR en R: no existe la columna c en T» y R no se crea («mostrar» de algo inexistente: «ERROR: no existe X»). «mostrar R» imprime «R(c1, c2)», cada tupla como « v1 | v2 » y « n tuplas » («1 tupla»).

Código de partida (python)

```
import sys

relaciones = {}          # nombre → (columnas, lista de tuplas)
lineas = sys.stdin.read().splitlines()
i = 0
while i < len(lineas):
    l = lineas[i].strip()
    if l.startswith("tabla "):
        # tabla nombre col1,col2 ... filas ... fin
        nombre, cols = l.split()[1], l.split()[2].split(",")
        filas = []
        i += 1
        while lineas[i].strip() != "fin":
            filas.append(tuple(lineas[i].strip().split(",")))
            i += 1
        relaciones[nombre] = (cols, filas)
    elif l:
        pass # TODO: asignaciones «R = operación ...» y «mostrar R»
    i += 1
```

Ejemplo: Alumnos y matrículas

ENTRADA

```
tabla alumno id,nombre,curso
1,Ana,1DAM
2,Luis,2DAM
3,Eva,1DAM
4,Iker,1DAW
fin
tabla matricula alumno,module,nota
1,BD,7
1,PROG,4
2,BD,9
3,PROG,6
3,BD,5
fin
R1 = seleccion alumno curso=1DAM
R2 = join R1 matricula id=alumno
R3 = proyeccion R2 nombre,module
mostrar R3
R4 = seleccion R2 nota<5
R5 = proyeccion R4 nombre
mostrar R5
R6 = proyeccion alumno id
R7 = proyeccion matricula alumno
R8 = diferencia R6 R7
R9 = renombrar R7 alumno=id
R10 = diferencia R6 R9
mostrar R10
R11 = proyeccion R2 module
mostrar R11
R12 = seleccion alumno edad>18
R13 = union alumno matricula
mostrar R13
```

SALIDA ESPERADA

```
R3(nombre, module)
Ana | BD
Ana | PROG
Eva | PROG
Eva | BD
4 tuplas
R5(nombre)
Ana
1 tupla
ERROR en R8: esquemas incompatibles (id) y
(alumno)
R10(id)
4
1 tupla
R11(module)
BD
PROG
2 tuplas
ERROR en R12: no existe la columna edad en alumno
ERROR en R13: esquemas incompatibles
(id,nombre,curso) y (alumno,module,nota)
ERROR: no existe R13
```

Consultas SQL Avanzadas

10. Encuentra el fallo: los clientes sin email

Fácil · Consultas SQL Avanzadas · apuntesdam.com/subject/bases-datos/topic/consultas

La consulta debería mostrar el nombre de los clientes que no tienen email, pero no devuelve ninguna fila. Corrígela.

Usa la base de datos de la tienda del principio de la hoja.

Consulta de partida (sql)

```
SELECT nombre
FROM Clientes
WHERE email = NULL;
```

11. Encuentra el fallo: las prioridades de AND y OR

Medio · Consultas SQL Avanzadas · apuntesdam.com/subject/bases-datos/topic/consultas

Se quieren los productos de las categorías Periféricos o Audio que cuesten menos de 50 €, ordenados por nombre. La consulta devuelve también el teclado mecánico, que cuesta 89,90 €. Corrígela.

Usa la base de datos de la tienda del principio de la hoja.

Consulta de partida (sql)

```
SELECT nombre, precio
FROM Productos
WHERE categoria = 'Periféricos' OR categoria = 'Audio' AND precio < 50
ORDER BY nombre;
```

12. Encuentra el fallo: la media que sale entera

Medio · Consultas SQL Avanzadas · apuntesdam.com/subject/bases-datos/topic/consultas

Se quiere la media de unidades por pedido. La consulta da 2, pero entre los 5 pedidos suman 13 unidades. Corrígela para que dé el resultado exacto.

Usa la base de datos de la tienda del principio de la hoja.

Consulta de partida (sql)

```
SELECT SUM(cantidad) / COUNT(DISTINCT pedido_id) AS media
FROM DetallePedido;
```

13. Encuentra el fallo: el pedido fantasma

Medio · Consultas SQL Avanzadas · apuntesdam.com/subject/bases-datos/topic/consultas

La consulta cuenta los pedidos de cada cliente, incluidos los que no tienen ninguno. Pero Marta Díaz, que nunca ha comprado, aparece con 1 pedido. Corrígela.

Usa la base de datos de la tienda del principio de la hoja.

Consulta de partida (sql)

```
SELECT c.nombre, COUNT(*) AS pedidos
FROM Clientes c
LEFT JOIN Pedidos p ON p.cliente_id = c.cliente_id
GROUP BY c.cliente_id
ORDER BY pedidos DESC, c.nombre;
```

14. Encuentra el fallo: el LEFT JOIN que pierde clientes

Difícil · Consultas SQL Avanzadas · apuntesdam.com/subject/bases-datos/topic/consultas

Se quiere, para cada cliente, cuántos pedidos tiene entregados, con 0 para los que no tienen ninguno, ordenado por id de cliente. Lucía y Marta no aparecen. Corrígela sin quitar el LEFT JOIN.

Usa la base de datos de la tienda del principio de la hoja.

Consulta de partida (sql)

```
SELECT c.nombre, COUNT(p.pedido_id) AS entregados
FROM Clientes c
LEFT JOIN Pedidos p ON p.cliente_id = c.cliente_id
WHERE p.estado = 'entregado'
GROUP BY c.cliente_id
ORDER BY c.cliente_id;
```

15. Encuentra el fallo: las bajas que lo borran todo

Muy difícil · Consultas SQL Avanzadas · apuntesdam.com/subject/bases-datos/topic/consultas

La tabla Bajas guarda los emails de quienes se han dado de baja del boletín; uno se registró vacío (NULL). Se quiere el nombre de los clientes con email que siguen suscritos, por orden alfabético. La consulta no devuelve nada. Corrígela.

Base de datos (tablas y datos) (sql)

```
CREATE TABLE Clientes (  
  cliente_id INTEGER PRIMARY KEY,  
  nombre TEXT NOT NULL,  
  email TEXT UNIQUE,  
  ciudad TEXT,  
  fecha_registro DATE  
);  
  
CREATE TABLE Productos (  
  producto_id INTEGER PRIMARY KEY,  
  nombre TEXT NOT NULL,  
  categoria TEXT,  
  precio REAL CHECK (precio >= 0),  
  stock INTEGER DEFAULT 0,  
  proveedor TEXT,  
  fecha_alta DATE  
);  
  
CREATE TABLE Pedidos (  
  pedido_id INTEGER PRIMARY KEY,  
  cliente_id INTEGER REFERENCES Clientes(cliente_id),  
  fecha DATE,  
  estado TEXT DEFAULT 'pendiente'  
);  
  
CREATE TABLE DetallePedido (  
  pedido_id INTEGER REFERENCES Pedidos(pedido_id),  
  producto_id INTEGER REFERENCES Productos(producto_id),  
  cantidad INTEGER NOT NULL,  
  precio_unitario REAL NOT NULL,  
  PRIMARY KEY (pedido_id, producto_id)  
);  
  
INSERT INTO Clientes VALUES  
(1, 'Ana García', 'ana@email.com', 'Madrid', '2024-01-15'),  
(2, 'Carlos López', 'carlos@email.com', 'Barcelona', '2024-02-20'),  
(3, 'Lucía Martín', 'lucia@email.com', 'Madrid', '2024-03-05'),  
(4, 'Javier Ruiz', 'javier@email.com', 'Sevilla', '2024-04-10'),  
(5, 'Marta Díaz', NULL, 'Valencia', '2024-05-22');  
  
INSERT INTO Productos VALUES  
(1, 'Portátil HP', 'Informática', 899.99, 15, 'HP', '2023-09-01'),  
(2, 'Ratón Logitech', 'Periféricos', 24.99, 120, 'Logitech', '2023-09-01'),  
(3, 'Teclado mecánico', 'Periféricos', 89.90, 45, 'Logitech', '2023-11-12'),  
(4, 'Monitor 27"', 'Informática', 279.00, 20, 'Samsung', '2024-01-08'),  
(5, 'Auriculares BT', 'Audio', 59.95, 0, 'Sony', '2024-02-14'),  
(6, 'Webcam HD', 'Periféricos', 39.99, 60, 'Logitech', '2024-03-01'),  
(7, 'Altavoz portátil', 'Audio', 45.50, 30, 'JBL', '2024-04-18'),  
(8, 'Disco SSD 1TB', 'Informática', 99.00, 75, 'Samsung', '2024-05-02');  
  
INSERT INTO Pedidos VALUES  
(1, 1, '2024-06-01', 'entregado'),  
(2, 2, '2024-06-03', 'entregado'),  
(3, 1, '2024-06-10', 'enviado'),  
(4, 3, '2024-06-12', 'pendiente'),  
(5, 4, '2024-06-15', 'entregado');  
  
INSERT INTO DetallePedido VALUES  
(1, 1, 1, 899.99),  
(1, 2, 2, 24.99),  
(2, 4, 2, 279.00),  
(3, 3, 1, 89.90),  
(3, 6, 1, 39.99),  
(4, 8, 3, 99.00),  
(5, 5, 2, 59.95),  
(5, 7, 1, 45.50);  
  
CREATE TABLE Bajas (email TEXT);  
INSERT INTO Bajas VALUES ('carlos@email.com'), (NULL);
```

Consulta de partida (sql)

```
SELECT nombre
FROM Clientes
WHERE email NOT IN (SELECT email FROM Bajas)
ORDER BY nombre;
```

Modificación de Datos en SQL

16. Encuentra el fallo: la actualización que no actualiza nada

Fácil · Modificación de Datos en SQL · apuntesdam.com/subject/bases-datos/topic/modificacion

Hay que poner a 0 el stock de los productos de la categoría Audio, pero al ejecutar la sentencia no cambia ninguna fila. Corrígela.

Usa la base de datos de la tienda del principio de la hoja.

Consulta de partida (sql)

```
UPDATE Productos
SET stock = 0
WHERE categoria = 'audio';
```

17. Encuentra el fallo: el stock que se vuelve NULL

Difícil · Modificación de Datos en SQL · apuntesdam.com/subject/bases-datos/topic/modificacion

Al enviar el pedido 3 hay que descontar del stock las unidades de cada uno de sus productos. Tras ejecutar la sentencia, el stock de los productos que no están en ese pedido queda vacío (NULL). Corrígela para que solo cambien los productos del pedido.

Usa la base de datos de la tienda del principio de la hoja.

Consulta de partida (sql)

```
UPDATE Productos
SET stock = stock - (
    SELECT d.cantidad FROM DetallePedido d
    WHERE d.pedido_id = 3 AND d.producto_id = Productos.producto_id
);
```

Programación en Bases de Datos

18. Encuentra el fallo: el disparador que descuenta a todos

Difícil · Programación en Bases de Datos · apuntesdam.com/subject/bases-datos/topic/programacion-bdd

El disparador debería restar del stock del producto vendido las unidades de cada nueva línea de pedido. Al insertar una línea con 5 ratones, baja el stock de todos los productos. Corrígelo (el INSERT de prueba se queda igual).

Usa la base de datos de la tienda del principio de la hoja.

Consulta de partida (sql)

```
CREATE TRIGGER descontar_stock
AFTER INSERT ON DetallePedido
BEGIN
    UPDATE Productos
    SET stock = stock - NEW.cantidad
    WHERE producto_id = producto_id;
END;

INSERT INTO DetallePedido VALUES (4, 2, 5, 24.99);
```

Ejercicios de Bases de Datos

19. Productos baratos

Fácil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Muestra el nombre y el precio de los productos que cuestan menos de 50 €, ordenados de más barato a más caro.

Usa la base de datos de la tienda del principio de la hoja.

20. Clientes sin email

Fácil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Muestra el nombre y la ciudad de los clientes que no tienen email registrado.

Usa la base de datos de la tienda del principio de la hoja.

21. Búsqueda por patrón

Fácil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Muestra los productos cuyo nombre contiene la palabra «portátil» o empieza por «Disco», sin distinguir mayúsculas. Devuelve solo el nombre.

Usa la base de datos de la tienda del principio de la hoja.

22. Alta de un producto

Fácil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Da de alta el producto 9: «Alfombrilla XL», categoría «Periféricos», 14.95 €, 80 unidades, proveedor «Logitech», fecha de alta 2024-07-01.

Usa la base de datos de la tienda del principio de la hoja.

23. Estadísticas del catálogo

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Para cada categoría muestra: número de productos, precio mínimo, precio máximo y stock total, ordenado por categoría.

Usa la base de datos de la tienda del principio de la hoja.

24. Pedidos con su cliente

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Lista cada pedido con su número, la fecha, el nombre del cliente y la ciudad, ordenado por fecha.

Usa la base de datos de la tienda del principio de la hoja.

25. Importe de cada pedido

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Calcula el importe total de cada pedido (suma de cantidad × precio unitario de sus líneas), mostrando el número de pedido y el importe redondeado a 2 decimales, del mayor al menor.

Usa la base de datos de la tienda del principio de la hoja.

26. Clientes que nunca han comprado

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Muestra el nombre de los clientes que no tienen ningún pedido. Resuélvelo con LEFT JOIN o con NOT EXISTS.

Usa la base de datos de la tienda del principio de la hoja.

27. Ciudades de clientes y proveedores

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Obtén una lista única de «nombres» que aparecen como ciudad de algún cliente o como proveedor de algún producto, ordenada alfabéticamente.

Usa la base de datos de la tienda del principio de la hoja.

28. Subida de precios

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Sube un 5 % el precio de los productos del proveedor «Samsung», redondeando a 2 decimales.

Usa la base de datos de la tienda del principio de la hoja.

29. Crear una vista

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Crea la vista StockBajo con el nombre, la categoría y el stock de los productos con menos de 25 unidades.

Usa la base de datos de la tienda del principio de la hoja.

30. Productos más caros que la media de su categoría

Difícil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Muestra el nombre, la categoría y el precio de los productos cuyo precio supera la media de su propia categoría.

Usa la base de datos de la tienda del principio de la hoja.

31. Mejor cliente por gasto

Difícil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Muestra el nombre del cliente que más ha gastado en total y su gasto (2 decimales). Si hubiera empate basta con uno.

Usa la base de datos de la tienda del principio de la hoja.

32. Categorías con más de 100 € vendidos

Difícil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Muestra las categorías cuyo importe vendido supera los 100 €, con el importe (2 decimales), de mayor a menor.

Usa la base de datos de la tienda del principio de la hoja.

33. Borrado con subconsulta

Difícil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Elimina los pedidos en estado «pendiente» de clientes de Madrid. Recuerda borrar antes sus líneas de DetallePedido para no dejar huérfanos.

Usa la base de datos de la tienda del principio de la hoja.

34. Trigger de auditoría

Difícil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Crea la tabla HistorialPrecios(producto_id, precio_anterior, precio_nuevo) y un trigger que inserte una fila en ella cada vez que cambie el precio de un producto. Pruébalo subiendo 10 € el precio del producto 3.

Usa la base de datos de la tienda del principio de la hoja.

Ejercicios largos

35. La base de datos de una academia de idiomas

Difícil · SQL · 75 minutos · apuntesdam.com/ejercicios/sql/academia-de-idiomias

Una academia de idiomas de Madrid acaba de pasar sus listas de Excel a una base de datos y te pide los informes que hasta ahora hacían a mano. Tiene cinco profesores, siete cursos de inglés, francés, alemán e italiano, y diez alumnos que se han matriculado en septiembre.

Cada curso tiene un profesor, un número de plazas y un precio mensual. Un alumno puede estar en varios cursos; la nota de cada matrícula es `NULL` mientras el alumno no ha hecho el primer examen. En la tabla de pagos está lo que cada alumno ha pagado en octubre, que debería coincidir con la suma de los precios de sus cursos, pero no siempre es así.

Las tablas tienen exactamente los datos de abajo: antes de escribir cada consulta, calcula a mano qué filas deberían salir. Es la mejor forma de detectar un JOIN que duplica filas o un `NULL` que se cuela.

Requisitos

- Escribe una consulta para cada apartado. Se corrigen por separado comparando tu resultado con el esperado: importan las filas, el número de columnas y, cuando el apartado pide un orden, el orden; los nombres de las columnas no.
- Los importes y porcentajes que pidan decimales se redondean con `ROUND`. Ojo con la división entera: en SQLite, `3 / 12` es 0.
- Los apartados van de menos a más difíciles. Si uno se te atasca, usa sus pistas antes de mirar la solución completa del final.

Base de datos (tablas y datos) (sql)

```
CREATE TABLE profesores (  
  profesor_id INTEGER PRIMARY KEY,  
  nombre TEXT NOT NULL,  
  idioma TEXT NOT NULL,  
  contratado DATE NOT NULL  
);  
CREATE TABLE cursos (  
  curso_id INTEGER PRIMARY KEY,  
  idioma TEXT NOT NULL,  
  nivel TEXT NOT NULL,  
  profesor_id INTEGER REFERENCES profesores(profesor_id),  
  plazas INTEGER NOT NULL,  
  precio_mes REAL NOT NULL  
);  
CREATE TABLE alumnos (  
  alumno_id INTEGER PRIMARY KEY,  
  nombre TEXT NOT NULL,  
  ciudad TEXT,  
  fecha_nacimiento DATE,  
  email TEXT  
);  
CREATE TABLE matriculas (  
  alumno_id INTEGER REFERENCES alumnos(alumno_id),  
  curso_id INTEGER REFERENCES cursos(curso_id),  
  fecha DATE NOT NULL,  
  nota REAL,  
  PRIMARY KEY (alumno_id, curso_id)  
);  
CREATE TABLE pagos (  
  pago_id INTEGER PRIMARY KEY,  
  alumno_id INTEGER REFERENCES alumnos(alumno_id),  
  fecha DATE NOT NULL,  
  importe REAL NOT NULL,  
  metodo TEXT NOT NULL  
);  
  
INSERT INTO profesores (profesor_id, nombre, idioma, contratado) VALUES  
  (1, 'Laura Gómez', 'inglés', '2019-09-01'),  
  (2, 'Pierre Martin', 'francés', '2021-01-15'),  
  (3, 'Anna Schmidt', 'alemán', '2022-09-01'),  
  (4, 'David Ruiz', 'inglés', '2023-02-01'),  
  (5, 'Chiara Rossi', 'italiano', '2024-09-01');  
  
INSERT INTO cursos (curso_id, idioma, nivel, profesor_id, plazas, precio_mes) VALUES  
  (1, 'inglés', 'A2', 1, 12, 55),  
  (2, 'inglés', 'B1', 1, 12, 60),  
  (3, 'inglés', 'B2', 4, 10, 65),  
  (4, 'francés', 'A1', 2, 10, 50),  
  (5, 'francés', 'B1', 2, 8, 58),  
  (6, 'alemán', 'A1', 3, 10, 52),  
  (7, 'inglés', 'C1', 4, 8, 75);  
  
INSERT INTO alumnos (alumno_id, nombre, ciudad, fecha_nacimiento, email) VALUES  
  (1, 'Marta López', 'Madrid', '2001-03-14', 'marta@correo.es'),  
  (2, 'Javier Pérez', 'Toledo', '1998-11-02', 'javier@correo.es'),  
  (3, 'Sofía Navarro', 'Madrid', '2004-07-21', NULL),  
  (4, 'Hugo Martín', 'Getafe', '1995-01-30', 'hugo@correo.es'),  
  (5, 'Lucía Ortega', 'Madrid', '2003-05-09', 'lucia@correo.es'),  
  (6, 'Daniel Serrano', 'Toledo', '2000-12-12', NULL),  
  (7, 'Elena Castro', 'Alcalá', '1999-08-25', 'elena@correo.es'),  
  (8, 'Pablo Vidal', 'Getafe', '2002-02-17', 'pablo@correo.es'),  
  (9, 'Irene Molina', 'Madrid', '1997-10-05', 'irene@correo.es'),  
  (10, 'Álvaro Ramos', 'Alcalá', '2005-04-28', 'alvaro@correo.es');  
  
INSERT INTO matriculas (alumno_id, curso_id, fecha, nota) VALUES  
  (1, 2, '2026-09-15', 8.5),  
  (1, 4, '2026-09-16', 7),  
  (2, 1, '2026-09-15', 5.5),  
  (3, 2, '2026-09-17', 9),  
  (3, 6, '2026-09-18', NULL),
```

```
(4, 3, '2026-09-15', 6),
(5, 2, '2026-09-20', 4),
(5, 5, '2026-09-20', 8),
(6, 1, '2026-09-22', NULL),
(7, 3, '2026-09-15', 7.5),
(7, 6, '2026-09-16', 6.5),
(8, 1, '2026-09-25', 3.5),
(9, 4, '2026-09-15', 9.5),
(9, 5, '2026-09-16', NULL),
(9, 2, '2026-09-17', 6);
```

```
INSERT INTO pagos (pago_id, alumno_id, fecha, importe, metodo) VALUES
(1, 1, '2026-10-01', 110, 'tarjeta'),
(2, 2, '2026-10-02', 55, 'bizum'),
(3, 3, '2026-10-01', 112, 'tarjeta'),
(4, 5, '2026-10-03', 60, 'efectivo'),
(5, 6, '2026-10-05', 55, 'bizum'),
(6, 7, '2026-10-01', 117, 'tarjeta'),
(7, 9, '2026-10-02', 168, 'tarjeta');
```

Apartados

- Nombre y email de los alumnos de Madrid que tienen email, por orden alfabético del nombre.
- Todos los cursos con el nombre de su profesor: idioma, nivel, nombre del profesor y precio al mes, ordenados por idioma y, dentro de cada idioma, por nivel.
- Cuántos alumnos hay en cada curso, incluidos los cursos que no tienen ninguno: idioma, nivel y número de matriculados, ordenados por idioma y nivel.
- Ocupación de cada curso: idioma, nivel, matriculados, plazas y porcentaje de ocupación redondeado a un decimal (3 de 12 plazas es 25.0). De mayor a menor ocupación y, a igualdad, por idioma y nivel.
- Nota media por idioma, contando solo las matrículas que ya tienen nota: idioma, número de alumnos evaluados y nota media redondeada a dos decimales, de mayor a menor nota media.
- Lo que debe cada alumno de octubre: nombre, total de sus cursos, lo que ha pagado y la deuda (total menos pagado). Solo los que deben algo, de mayor a menor deuda. Quien no tiene ningún pago ha pagado 0.
- Para cada profesor, incluidos los que no dan clase, cuántos alumnos distintos tiene en total entre todos sus cursos: nombre, idioma y número de alumnos, de más a menos alumnos y, a igualdad, por nombre.
- El alumno con la nota más alta de cada idioma: idioma, nombre del alumno y nota, ordenado por idioma.
- La academia sube un 5 % el precio de los cursos de inglés que tienen más de 2 alumnos matriculados. Escribe el UPDATE (se comprobarán los precios de todos los cursos después de ejecutarlo).

36. Clínica: citas, especialidades y facturación

Difícil · SQL · 80 minutos · apuntesdam.com/ejercicios/sql/clinica-citas-y-facturacion

Una clínica privada con cinco especialidades guarda en su base de datos los médicos, los pacientes y las citas. Cada cita tiene un estado: `realizada` (y entonces tiene importe), `cancelada` o `pendiente` (sin importe todavía). Los pacientes sin aseguradora (`NULL`) son privados y pagan ellos.

La dirección te pide una serie de informes para la reunión de octubre de 2026. Antes de escribir cada consulta, busca en las tablas qué filas deberían salir: varios apartados tienen trampas pensadas (un médico que factura exactamente 100 €, una paciente que cumple años al día siguiente, una especialidad sin citas).

Requisitos

- Escribe una consulta (o una sentencia, en el último apartado) para cada apartado. La fecha de referencia de los informes es el 1 de octubre de 2026.
- Cada apartado se corrige por separado sobre una copia nueva de la base de datos, con exactamente los datos de las tablas de arriba: lo que hagas en uno no afecta a los demás.
- Importan las filas, el número y el orden de las columnas y, cuando el apartado pide un orden, el orden de las filas; los nombres de las columnas no importan. Redondea con `ROUND` cuando se pidan decimales.

Base de datos (tablas y datos) (sql)

```
CREATE TABLE especialidades (  
    especialidad_id INTEGER PRIMARY KEY,  
    nombre TEXT NOT NULL,  
    precio_consulta REAL NOT NULL  
);  
CREATE TABLE medicos (  
    medico_id INTEGER PRIMARY KEY,  
    nombre TEXT NOT NULL,  
    especialidad_id INTEGER NOT NULL REFERENCES especialidades(especialidad_id),  
    anio_alta INTEGER NOT NULL  
);  
CREATE TABLE pacientes (  
    paciente_id INTEGER PRIMARY KEY,  
    nombre TEXT NOT NULL,  
    nacimiento DATE NOT NULL,  
    ciudad TEXT NOT NULL,  
    aseguradora TEXT  
);  
CREATE TABLE citas (  
    cita_id INTEGER PRIMARY KEY,  
    paciente_id INTEGER NOT NULL REFERENCES pacientes(paciente_id),  
    medico_id INTEGER NOT NULL REFERENCES medicos(medico_id),  
    fecha DATE NOT NULL,  
    hora TEXT NOT NULL,  
    estado TEXT NOT NULL CHECK (estado IN ('realizada', 'cancelada', 'pendiente', 'no presentado')),  
    importe REAL  
);  
  
INSERT INTO especialidades (especialidad_id, nombre, precio_consulta) VALUES  
    (1, 'Medicina general', 40),  
    (2, 'Pediatría', 50),  
    (3, 'Dermatología', 65),  
    (4, 'Traumatología', 70),  
    (5, 'Cardiología', 85);  
  
INSERT INTO medicos (medico_id, nombre, especialidad_id, anio_alta) VALUES  
    (1, 'Ana Ruiz', 1, 2015),  
    (2, 'Luis Gómez', 1, 2019),  
    (3, 'Marta Peña', 2, 2012),  
    (4, 'Jorge Sanz', 3, 2020),  
    (5, 'Elena Ríos', 4, 2017),  
    (6, 'Pablo Mena', 5, 2024);  
  
INSERT INTO pacientes (paciente_id, nombre, nacimiento, ciudad, aseguradora) VALUES  
    (1, 'Carmen López', '1958-03-12', 'Madrid', 'Sanitas'),  
    (2, 'Diego Martín', '2018-11-30', 'Madrid', NULL),  
    (3, 'Lucía Fernández', '1990-10-01', 'Getafe', 'Adeslas'),  
    (4, 'Hugo Navarro', '2020-06-15', 'Madrid', 'Sanitas'),  
    (5, 'Rosa Jiménez', '1975-10-02', 'Madrid', NULL),  
    (6, 'Iván Torres', '1988-01-20', 'Alcorcón', NULL),  
    (7, 'Sara Molina', '2001-12-24', 'Madrid', 'Asisa'),  
    (8, 'Tomás Ortega', '1949-07-07', 'Getafe', NULL);  
  
INSERT INTO citas (cita_id, paciente_id, medico_id, fecha, hora, estado, importe) VALUES  
    (1, 1, 1, '2026-09-02', '09:00', 'realizada', 40),  
    (2, 2, 3, '2026-09-03', '10:30', 'realizada', 50),  
    (3, 3, 4, '2026-09-08', '12:00', 'realizada', 65),  
    (4, 1, 5, '2026-09-10', '16:00', 'realizada', 70),  
    (5, 5, 1, '2026-09-15', '09:30', 'cancelada', NULL),  
    (6, 6, 2, '2026-09-16', '11:00', 'realizada', 40),  
    (7, 4, 3, '2026-09-18', '10:00', 'realizada', 50),  
    (8, 7, 4, '2026-09-22', '13:00', 'realizada', 65),  
    (9, 8, 5, '2026-09-25', '17:30', 'cancelada', NULL),  
    (10, 1, 1, '2026-09-29', '09:00', 'realizada', 40),  
    (11, 3, 2, '2026-09-30', '12:30', 'pendiente', NULL),  
    (12, 5, 4, '2026-10-05', '10:00', 'pendiente', NULL),  
    (13, 2, 3, '2026-10-05', '09:30', 'pendiente', NULL),  
    (14, 8, 1, '2026-10-05', '12:00', 'pendiente', NULL),  
    (15, 6, 5, '2026-10-06', '16:30', 'pendiente', NULL),  
    (16, 7, 2, '2026-09-12', '11:30', 'cancelada', NULL),
```

Apartados

- Pacientes privados (sin aseguradora) que viven en Madrid, con su edad cumplida el 1 de octubre de 2026: nombre y edad, de mayor a menor edad.
- La agenda del 5 de octubre de 2026: hora, nombre del paciente, nombre del médico y especialidad, por orden de hora.
- Número de citas realizadas de cada especialidad, incluidas las que no tienen ninguna: especialidad y número de citas, de más a menos citas y, a igualdad, por nombre de la especialidad.
- Facturación de septiembre de 2026 por médico, contando solo las citas realizadas: médico, número de citas e importe total, solo de los médicos que facturan más de 100 €, de mayor a menor importe.
- Pacientes que tienen alguna cita y no han cancelado nunca ninguna: nombre, por orden alfabético.
- Porcentaje de citas canceladas de cada médico que tenga al menos 3 citas (en cualquier estado): médico, citas, canceladas y porcentaje con un decimal, de mayor a menor porcentaje y, a igualdad, por nombre.
- Pacientes que han tenido citas realizadas con médicos de más de una especialidad: nombre y número de especialidades distintas, por orden alfabético.
- La última cita realizada de cada paciente que tenga alguna: nombre del paciente, fecha y nombre del médico, por orden alfabético del paciente.
- Las citas pendientes con fecha anterior al 1 de octubre de 2026 pasan al estado 'no presentado'. Escribe el UPDATE (se comprueban el id y el estado de todas las citas).

37. Liga de baloncesto: clasificación y funciones de ventana

Muy difícil · SQL · 90 minutos · apuntesdam.com/ejercicios/sql/liga-de-baloncesto-funciones-de-ventana

Una liga de baloncesto amateur de cuatro equipos lleva tres jornadas. Cada partido enfrenta a un equipo local y uno visitante, y la tabla `estadisticas` guarda los puntos, rebotes y asistencias de cada jugador en cada partido (cada equipo tiene tres jugadores en esta versión reducida: los puntos de un equipo en un partido son la suma de los de sus jugadores).

Las preguntas de una liga (quién es el máximo anotador de cada partido, cómo va el acumulado de cada jugador, quién está por encima de la media de su posición) son el terreno de las funciones de ventana: calculan algo sobre un grupo de filas sin agruparlas, así que cada fila conserva sus datos y además recibe el ranking, el acumulado o la media de su grupo.

Requisitos

- Escribe una consulta para cada apartado.
- Cada apartado se corrige por separado sobre una copia nueva de la base de datos, con exactamente los datos de las tablas de arriba: lo que hagas en uno no afecta a los demás.
- Importan las filas, el número y el orden de las columnas y, cuando el apartado pide un orden, el orden de las filas; los nombres de las columnas no importan. Redondea con `ROUND` cuando se pidan decimales.

Base de datos (tablas y datos) (sql)

```
CREATE TABLE equipos (  
    equipo_id INTEGER PRIMARY KEY,  
    nombre TEXT NOT NULL,  
    ciudad TEXT NOT NULL  
);  
CREATE TABLE jugadores (  
    jugador_id INTEGER PRIMARY KEY,  
    nombre TEXT NOT NULL,  
    equipo_id INTEGER NOT NULL REFERENCES equipos(equipo_id),  
    posicion TEXT NOT NULL  
);  
CREATE TABLE partidos (  
    partido_id INTEGER PRIMARY KEY,  
    jornada INTEGER NOT NULL,  
    fecha DATE NOT NULL,  
    local_id INTEGER NOT NULL REFERENCES equipos(equipo_id),  
    visitante_id INTEGER NOT NULL REFERENCES equipos(equipo_id),  
    puntos_local INTEGER NOT NULL,  
    puntos_visitante INTEGER NOT NULL  
);  
CREATE TABLE estadisticas (  
    partido_id INTEGER NOT NULL REFERENCES partidos(partido_id),  
    jugador_id INTEGER NOT NULL REFERENCES jugadores(jugador_id),  
    puntos INTEGER NOT NULL,  
    rebotes INTEGER NOT NULL,  
    asistencias INTEGER NOT NULL,  
    PRIMARY KEY (partido_id, jugador_id)  
);  
  
INSERT INTO equipos (equipo_id, nombre, ciudad) VALUES  
    (1, 'Leones de Madrid', 'Madrid'),  
    (2, 'Tiburones de Valencia', 'Valencia'),  
    (3, 'Halcones de Sevilla', 'Sevilla'),  
    (4, 'Osos de Bilbao', 'Bilbao');  
  
INSERT INTO jugadores (jugador_id, nombre, equipo_id, posicion) VALUES  
    (1, 'Álex Moreno', 1, 'base'),  
    (2, 'Iker Gil', 1, 'alero'),  
    (3, 'Dani Prieto', 1, 'pívot'),  
    (4, 'Raúl Vidal', 2, 'base'),  
    (5, 'Marc Soler', 2, 'alero'),  
    (6, 'Joan Ferrer', 2, 'pívot'),  
    (7, 'Pablo Romero', 3, 'base'),  
    (8, 'Luis Cano', 3, 'alero'),  
    (9, 'Sergio Lara', 3, 'pívot'),  
    (10, 'Unai Etxeberria', 4, 'base'),  
    (11, 'Jon Ibarra', 4, 'alero'),  
    (12, 'Mikel Arana', 4, 'pívot');  
  
INSERT INTO partidos (partido_id, jornada, fecha, local_id, visitante_id, puntos_local, puntos_visitante)  
VALUES  
    (1, 1, '2026-10-03', 1, 2, 62, 58),  
    (2, 1, '2026-10-03', 3, 4, 56, 61),  
    (3, 2, '2026-10-10', 2, 3, 64, 63),  
    (4, 2, '2026-10-10', 4, 1, 56, 60),  
    (5, 3, '2026-10-17', 1, 3, 59, 66),  
    (6, 3, '2026-10-17', 2, 4, 61, 66);  
  
INSERT INTO estadisticas (partido_id, jugador_id, puntos, rebotes, asistencias) VALUES  
    (1, 1, 24, 3, 7),  
    (1, 2, 18, 5, 2),  
    (1, 3, 20, 11, 1),  
    (1, 4, 21, 4, 8),  
    (1, 5, 25, 6, 3),  
    (1, 6, 12, 9, 1),  
    (2, 7, 15, 2, 9),  
    (2, 8, 22, 7, 2),  
    (2, 9, 19, 12, 0),  
    (2, 10, 27, 4, 6),  
    (2, 11, 14, 6, 3),
```

```
(2, 12, 20, 10, 2),
(3, 4, 18, 3, 10),
(3, 5, 30, 5, 2),
(3, 6, 16, 8, 1),
(3, 7, 20, 3, 7),
(3, 8, 22, 6, 1),
(3, 9, 21, 13, 2),
(4, 10, 19, 5, 9),
(4, 11, 22, 4, 2),
(4, 12, 15, 9, 1),
(4, 1, 28, 4, 6),
(4, 2, 18, 6, 3),
(4, 3, 14, 12, 2),
(5, 1, 21, 2, 8),
(5, 2, 21, 7, 1),
(5, 3, 17, 10, 3),
(5, 7, 25, 4, 6),
(5, 8, 18, 5, 2),
(5, 9, 23, 11, 1),
(6, 4, 22, 2, 7),
(6, 5, 19, 6, 4),
(6, 6, 20, 9, 0),
(6, 10, 24, 3, 5),
(6, 11, 24, 7, 2),
(6, 12, 18, 8, 1);
```

Apartados

- La clasificación: equipo, partidos jugados, victorias, derrotas, puntos a favor, puntos en contra y diferencia, ordenada por victorias (de más a menos), después por diferencia (de más a menos) y por último por nombre.
- El máximo anotador de cada partido: número de partido, jugador y puntos, por número de partido. Si dos jugadores empatan, el primero por nombre.
- El ranking de anotadores de la liga: puesto, jugador, equipo y puntos totales. Los empatados comparten puesto y el siguiente se salta (como en `RANK`). Ordenado por puesto y, a igualdad, por nombre del jugador.
- Los puntos de cada jugador de los Leones de Madrid en cada jornada y su acumulado hasta esa jornada: jugador, jornada, puntos y acumulado, por jugador y jornada.
- La evolución de los jugadores de los Halcones de Sevilla: jugador, jornada, puntos y diferencia con sus puntos de la jornada anterior (NULL en la primera jornada), por jugador y jornada.
- Qué parte de los puntos de su equipo ha anotado cada jugador en toda la liga: equipo, jugador, puntos y porcentaje sobre el total del equipo con un decimal, por equipo y de mayor a menor porcentaje.
- Los jugadores que promedian más puntos por partido que la media de su posición: jugador, posición, su media y la media de la posición (las dos con un decimal), por posición y de mayor a menor media del jugador.

38. Banco: vistas, disparadores, UPSERT y transacciones

Muy difícil · SQL · 100 minutos · apuntesdam.com/ejercicios/sql/banco-vistas-disparadores-y-transacciones

Un banco pequeño guarda sus clientes, sus cuentas (corrientes y de ahorro) y los movimientos de cada cuenta: ingresos en positivo y cargos en negativo. La tabla `resumen` guarda los ingresos y gastos de cada cuenta por mes, y se rellena con un proceso nocturno.

Todos los apartados de este ejercicio cambian algo: crean objetos (vistas y disparadores) o modifican datos. Como en una base de datos real no se puede ver el resultado de un `UPDATE` directamente, cada apartado dice qué consulta se usará para comprobar el estado final de la base de datos. Si tu sentencia da un error, el apartado no se corrige.

En SQLite los disparadores no pueden usar variables ni bloques como en MySQL o PL/SQL, pero tienen lo esencial: `NEW` y `OLD` para la fila afectada, `WHEN` para la condición y `RAISE` para detener o ignorar la operación.

Requisitos

- Escribe la sentencia o sentencias de cada apartado (puedes poner varias separadas por `;`). Cada apartado indica la consulta con la que se comprueba.

- Cada apartado se corrige por separado sobre una copia nueva de la base de datos, con exactamente los datos de las tablas de arriba: lo que hagas en uno no afecta a los demás.
- Importan las filas, el número y el orden de las columnas y, cuando el apartado pide un orden, el orden de las filas; los nombres de las columnas no importan. Redondea con `ROUND` cuando se pidan decimales.

Base de datos (tablas y datos) (sql)

```
CREATE TABLE clientes (
  cliente_id INTEGER PRIMARY KEY,
  nombre TEXT NOT NULL,
  dni TEXT NOT NULL UNIQUE
);
CREATE TABLE cuentas (
  cuenta_id INTEGER PRIMARY KEY,
  cliente_id INTEGER NOT NULL REFERENCES clientes(cliente_id),
  numero TEXT NOT NULL UNIQUE,
  tipo TEXT NOT NULL CHECK (tipo IN ('corriente', 'ahorro')),
  saldo REAL NOT NULL,
  abierta DATE NOT NULL,
  cerrada DATE
);
CREATE TABLE movimientos (
  movimiento_id INTEGER PRIMARY KEY,
  cuenta_id INTEGER NOT NULL REFERENCES cuentas(cuenta_id),
  fecha DATE NOT NULL,
  concepto TEXT NOT NULL,
  importe REAL NOT NULL
);
CREATE TABLE resumen (
  cuenta_id INTEGER NOT NULL REFERENCES cuentas(cuenta_id),
  mes TEXT NOT NULL,
  ingresos REAL NOT NULL,
  gastos REAL NOT NULL,
  PRIMARY KEY (cuenta_id, mes)
);

INSERT INTO clientes (cliente_id, nombre, dni) VALUES
(1, 'Ana Gil', '12345678Z'),
(2, 'Luis Mora', '23456789D'),
(3, 'Eva Ruiz', '34567890V'),
(4, 'Pedro Sanz', '45678901G');

INSERT INTO cuentas (cuenta_id, cliente_id, numero, tipo, saldo, abierta, cerrada) VALUES
(1, 1, 'C-001', 'corriente', 1250.4, '2019-05-10', NULL),
(2, 1, 'A-001', 'ahorro', 8300, '2020-01-15', NULL),
(3, 2, 'C-002', 'corriente', 340.75, '2021-03-01', NULL),
(4, 3, 'C-003', 'corriente', 2210, '2018-11-20', NULL),
(5, 3, 'A-002', 'ahorro', 950, '2022-06-30', NULL),
(6, 4, 'C-004', 'corriente', 0, '2015-02-01', '2020-12-31');

INSERT INTO movimientos (movimiento_id, cuenta_id, fecha, concepto, importe) VALUES
(1, 1, '2026-09-01', 'Nómina', 1800),
(2, 1, '2026-09-03', 'Alquiler', -750),
(3, 1, '2026-09-15', 'Supermercado', -120.35),
(4, 3, '2026-09-01', 'Nómina', 1200),
(5, 3, '2026-09-05', 'Hipoteca', -610),
(6, 3, '2026-09-20', 'Luz', -48.2),
(7, 4, '2026-09-02', 'Transferencia recibida', 300),
(8, 2, '2026-09-30', 'Intereses', 10.37),
(9, 6, '2019-06-01', 'Recibo', -25),
(10, 6, '2020-11-15', 'Recibo', -30),
(11, 1, '2026-10-01', 'Nómina', 1800),
(12, 3, '2026-10-02', 'Supermercado', -95.6);

INSERT INTO resumen (cuenta_id, mes, ingresos, gastos) VALUES
(1, '2026-08', 1800, 900),
(1, '2026-09', 0, 0);
```

Apartados

- Crea la vista `v_salDOS` con el nombre del cliente, el número de cuenta, el tipo y el saldo de las cuentas abiertas (las que no tienen fecha de cierre). Se comprueba con `SELECT * FROM v_salDOS ORDER BY 2`.
- Crea el disparador `trg_salDO` que, después de insertar un movimiento, sume su importe al saldo de su cuenta (redondeando el saldo a dos decimales). Después inserta estos dos movimientos: cuenta 3, 2026-10-03, 'Gimnasio', -39.90; y cuenta 5, 2026-10-03, 'Ingreso', 200. Se comprueba con `SELECT cuenta_id, saldo FROM cuentas ORDER BY cuenta_id`.
- Las cuentas de ahorro no pueden quedarse en negativo. Crea el disparador `trg_ahorro` que, antes de insertar un movimiento en una cuenta de ahorro, ignore la inserción (sin dar error) si el saldo actual más el importe sería menor que 0. Después inserta: cuenta 2, 2026-10-04, 'Retirada', -9000; cuenta 5, 2026-10-04, 'Retirada', -500; y cuenta 1, 2026-10-04, 'Recibo', -2000. Se comprueba con `SELECT cuenta_id, concepto, importe FROM movimientos WHERE fecha = '2026-10-04' ORDER BY cuenta_id`.
- Aplica los intereses: las cuentas de ahorro abiertas con un saldo de más de 1.000 € ganan un 1,5 %, con el saldo redondeado al céntimo. Se comprueba con `SELECT cuenta_id, saldo FROM cuentas ORDER BY cuenta_id`.
- Cobra la comisión de mantenimiento: inserta un movimiento de -2 € con fecha 2026-10-31 y concepto 'Comisión de mantenimiento' en cada cuenta corriente abierta con un saldo de menos de 1.500 € (no hace falta actualizar el saldo). Se comprueba con `SELECT cuenta_id, fecha, concepto, importe FROM movimientos WHERE fecha = '2026-10-31' ORDER BY cuenta_id`.
- Borra los movimientos de las cuentas cerradas antes del 1 de octubre de 2021. Se comprueba con `SELECT movimiento_id FROM movimientos ORDER BY movimiento_id`.
- Calcula el resumen de septiembre de 2026 de cada cuenta con movimientos en ese mes (ingresos: suma de los importes positivos; gastos: suma de los negativos, en positivo; ambos redondeados a dos decimales) y guárdalo en `resumen`: inserta las cuentas que no tienen fila de ese mes y actualiza la que ya la tiene. Se comprueba con `SELECT * FROM resumen ORDER BY cuenta_id, mes`.
- Traspasa 150 € de la cuenta C-001 a la A-002 dentro de una transacción: resta el importe de una, súmalo a la otra y registra los dos movimientos con fecha 2026-10-05 y los conceptos 'Traspaso enviado' (-150) y 'Traspaso recibido' (150). Se comprueba con el saldo y el número de movimientos de ese día de las dos cuentas.

Soluciones

1. Fichero secuencial frente a índice

Medio · Sistemas de Almacenamiento · apuntesdam.com/subject/bases-datos/topic/almacenamiento

Solución (python)

```
import sys

registros = []          # (id, nombre) en el orden del fichero (sin ordenar)
busquedas = []
for linea in sys.stdin.read().splitlines():
    p = linea.split()
    if not p:
        continue
    if p[0] == "buscar":
        busquedas.append(int(p[1]))
    else:
        registros.append((int(p[0]), p[1]))

indice = sorted((r[0], pos) for pos, r in enumerate(registros))  # (id, posición en el fichero)

def plural(n, palabra, varias):
    return f"{n} {palabra if n == 1 else varias}"

total_sec = total_ind = 0
for x in busquedas:
    # Secuencial: se leen registros desde el principio hasta encontrarlo
    pos = next((i for i, r in enumerate(registros) if r[0] == x), None)
    lecturas = pos + 1 if pos is not None else len(registros)
    # Índice: búsqueda binaria sobre la lista ordenada
    lo, hi, comparaciones, encontrado = 0, len(indice) - 1, 0, None
    while lo <= hi:
        mid = (lo + hi) // 2
        comparaciones += 1
        if indice[mid][0] == x:
            encontrado = indice[mid][1]
            break
        if indice[mid][0] < x:
            lo = mid + 1
        else:
            hi = mid - 1
    total_sec += lecturas
    total_ind += comparaciones + (1 if encontrado is not None else 0)
    sec = f"secuencial {plural(lecturas, 'lectura', 'lecturas')}"
    ind = f"índice {plural(comparaciones, 'comparación', 'comparaciones')}"
    if encontrado is not None:
        print(f"buscar {x}: encontrado ({registros[encontrado][1]}) · {sec} · {ind} + 1 lectura")
    else:
        print(f"buscar {x}: no existe · {sec} · {ind}")
print(f"Total secuencial: {total_sec} lecturas · índice: {total_ind} accesos · ahorro {round(100 - 100 * total_ind / total_sec)} %")
```

La búsqueda secuencial lee el fichero desde el principio y, en el peor caso (el registro no está o es el último), lo lee entero: con un millón de registros, un millón de lecturas. El índice es una lista ordenada de (clave, posición) en la que se puede hacer una búsqueda binaria: cada comparación descarta la mitad de lo que queda, y la posición encontrada lleva al registro con una sola lectura.

Con 40 registros la binaria nunca necesita más de 6 comparaciones ($\log_2 40 \approx 5,3$), y con un millón bastarían 20. Esa es la idea de los índices B-tree de cualquier base de datos, que además organizan el índice en bloques de disco.

2. Fragmentación de una base de datos distribuida

Difícil · Sistemas de Almacenamiento · apuntesdam.com/subject/bases-datos/topic/almacenamiento

Solución (python)

```
import sys

columnas, filas, fragmentos, consultas = [], [], [], []
for linea in sys.stdin.read().splitlines():
    p = linea.split(" ", 1)
    if p[0] == "tabla":
        columnas = p[1].split(" ")[1].split(",")
    elif p[0] == "fila":
        valores = p[1].split(",")
        filas.append(dict(zip(columnas, valores)))
    elif p[0] == "fragmento":
        # fragmento F1 sede=Madrid provincia=Madrid,Toledo
        nombre, sede, cond = p[1].split(" ")
        campo, valores = cond.split("=")
        fragmentos.append((nombre, sede.split("=")[1], campo, valores.split(",")))
    elif p[0] == "consulta":
        consultas.append(p[1])

def filas_texto(n):
    return f"{n} fila" if n == 1 else f"{n} filas"

contenido = {f[0]: [r for r in filas if r[f[2]] in f[3]] for f in fragmentos}
for nombre, sede, _, _ in fragmentos:
    print(f"{nombre} ({sede}): {filas_texto(len(contenido[nombre]))}")

# Completitud: cada fila debe estar en algún fragmento
campo0 = fragmentos[0][2]
perdidas = [r for r in filas if not any(r in contenido[f[0]] for f in fragmentos)]
if perdidas:
    print("Completitud: NO, sin fragmento: " + ", ".join(f"id {r['id']} ({r[campo0]})" for r in
    perdidas))
else:
    print("Completitud: sí")

# Disjunción: ninguna fila en dos fragmentos
repetidas = []
for r in filas:
    donde = [f[0] for f in fragmentos if r in contenido[f[0]]]
    if len(donde) > 1:
        repetidas.append(f"id {r['id']} en " + ", ".join(donde[:-1]) + " y " + donde[-1])
print("Disjunción: " + ("NO, " + "; ".join(repetidas) if repetidas else "sí"))

def cumple(r, campo, op, valor):
    if op == "=":
        return r[campo] == valor
    return float(r[campo]) > float(valor) if op == ">" else float(r[campo]) < float(valor)

for c in consultas:
    if c == "todos":
        campo = op = valor = None
    else:
        op = next(o for o in "<>" if o in c)
        campo, valor = c.split(op)
    elegidos = [f for f in fragmentos
        if not (op == "=" and campo == f[2]) or valor in f[3]] # poda solo en igualdades del
        campo
    sedes = list(dict.fromkeys(f[1] for f in elegidos)) # sin repetir, en orden
    ids = sorted((r["id"] for f in elegidos for r in contenido[f[0]] if campo is None or cumple(r, campo,
        op, valor)), key=int)
    texto = f"consulta {c} → sedes {'', '.join(sedes) or 'ninguna'} · {filas_texto(len(ids))}"
    print(texto + (": " + ", ".join(ids) if ids else ""))
```

Cada fragmento horizontal es una selección de la tabla: las filas cuyo campo toma alguno de sus valores. Con el contenido de cada fragmento calculado, las dos propiedades de una buena fragmentación se comprueban directamente: completitud (toda fila está en algún fragmento; si no, se pierde) y disjunción (ninguna está en dos; si no,

hay que mantener dos copias coherentes).

En las consultas se ve la ventaja de fragmentar: una igualdad sobre el campo de fragmentación solo necesita preguntar a las sedes cuyo fragmento incluye ese valor, y si ninguno lo incluye, a ninguna. Cualquier otra condición obliga a preguntar a todas. `dict.fromkeys` quita las sedes repetidas conservando el orden.

3. Anonimizar datos personales: k-anonimato

Muy difícil · Sistemas de Almacenamiento · apuntesdam.com/subject/bases-datos/topic/almacenamiento

Solución (python)

```
import sys
from collections import Counter, defaultdict

lineas = [l for l in sys.stdin.read().splitlines() if l.strip()]
k_objetivo = int(lineas[0].split()[1])
cabecera = lineas[1].split(";") # nombre;dni;edad;cp;sexo;diagnostico
filas = [dict(zip(cabecera, l.split(";"))) for l in lineas[2:]]

print("Identificadores eliminados: nombre, dni")
anonimas = []
for f in filas:
    decada = int(f["edad"]) // 10 * 10
    anonimas.append({"edad": f"{decada}-{decada + 9}", "cp": f["cp"][:3] + "***",
                    "sexo": f["sexo"], "diagnostico": f["diagnostico"]})

clave = lambda f: (f["edad"], f["cp"], f["sexo"]) # cuasi-identificadores
tamanos = Counter(clave(f) for f in anonimas)
print(f"k antes de suprimir: {min(tamanos.values())}")

publicables = [f for f in anonimas if tamanos[clave(f)] >= k_objetivo]
if not publicables:
    print("No queda ningún dato publicable")
    sys.exit()

print("edad;cp;sexo;diagnostico")
for f in publicables:
    print(f"{f['edad']}:{f['cp']}:{f['sexo']}:{f['diagnostico']}")
diagnosticos = defaultdict(set)
for f in publicables:
    diagnosticos[clave(f)].add(f["diagnostico"])
k = min(tamanos[g] for g in diagnosticos)
l_div = min(len(d) for d in diagnosticos.values())
print(f"Grupos: {len(diagnosticos)} · k = {k} (objetivo {k_objetivo}) · filas suprimidas: {len(anonimas) - len(publicables)} · l-diversidad: {l_div}")
if l_div == 1:
    print("Atención: en algún grupo todos tienen el mismo diagnóstico, que se puede deducir")
```

Quitar el nombre y el DNI no basta: la combinación de edad, código postal y sexo identifica a mucha gente. Por eso se generalizan (década y tres cifras del código postal) hasta que cada combinación la comparten varias personas;

`Counter` cuenta el tamaño de cada grupo, y el más pequeño es la k del conjunto. Los grupos con menos de K personas se suprimen, porque en ellos alguien sería reconocible.

La l-diversidad muestra el límite del k-anonimato: si todos los de un grupo tienen el mismo diagnóstico, saber que alguien está en ese grupo basta para conocer su enfermedad, aunque no se sepa cuál de las filas es la suya. Un

`defaultdict(set)` reúne los diagnósticos distintos de cada grupo.

4. Comprobar cardinalidades con datos reales

Medio · Modelo Entidad-Relación · apuntesdam.com/subject/bases-datos/topic/modelo-er

Solución (python)

```
import sys

instancias = {}          # entidad → lista de ocurrencias
relacion = None          # (nombre, entidad1, (min1, max1), entidad2, (min2, max2))
pares = []
for linea in sys.stdin.read().splitlines():
    p = linea.split()
    if not p:
        continue
    if p[0] == "entidad":
        instancias[p[1]] = p[2:]
    elif p[0] == "relacion":
        # relacion cursa Alumno(1,n) Modulo(0,2)
        lados = []
        for t in p[2:4]:
            nombre, card = t[:-1].split("(")
            mn, mx = card.split(",")
            lados.append((nombre, (int(mn), None if mx == "n" else int(mx))))
        relacion = (p[1], lados[0][0], lados[0][1], lados[1][0], lados[1][1])
    else:
        pares.append((p[1], p[2]))          # cursa A1 BD

nombre, e1, card1, e2, card2 = relacion
incumplimientos = 0
validos = []
for a, b in pares:
    if a not in instancias[e1] or b not in instancias[e2]:
        print(f"Ocurrencia desconocida: {a} {b}")
        incumplimientos += 1
    elif (a, b) in validos:
        print(f"Par repetido: {a} {b}")
        incumplimientos += 1
    else:
        validos.append((a, b))
print(f"Ocurrencias válidas de {nombre}: {len(validos)}")

for entidad, (minimo, maximo), lado in ((e1, card1, 0), (e2, card2, 1)):
    for i in instancias[entidad]:
        k = sum(1 for par in validos if par[lado] == i)          # en cuántas ocurrencias participa
        if k < minimo:
            print(f"{entidad} {i} participa {k} veces en {nombre} (mínimo {minimo})")
            incumplimientos += 1
        elif maximo is not None and k > maximo:
            print(f"{entidad} {i} participa {k} veces en {nombre} (máximo {maximo})")
            incumplimientos += 1

if incumplimientos:
    print(f"Restricciones: {incumplimientos} {'incumplimiento' if incumplimientos == 1 else 'incumplimientos'}")
else:
    print("Restricciones: correctas")
```

La cardinalidad (mínimo, máximo) de una entidad dice en cuántas ocurrencias de la relación puede aparecer cada una de sus instancias. Comprobarla con datos es contar: para cada instancia, cuántos pares válidos la contienen en su lado, y compararlo con el mínimo (participación obligatoria si es 1) y el máximo (n significa sin límite, que se guarda como `None`).

Los pares desconocidos o repetidos no cuentan como ocurrencias, igual que en una base de datos una clave ajena inexistente o una fila duplicada en una tabla con clave primaria serían rechazadas. Recorrer las dos entidades con el mismo bucle, indicando qué lado del par mirar, evita escribir dos veces el mismo código.

5. Del diagrama E/R a las tablas

Difícil · Modelo Entidad-Relación · apuntesdam.com/subject/bases-datos/topic/modelo-er

Solución (python)

```
import sys

entidades = {}          # nombre → (claves, atributos) en orden
relaciones = []         # (nombre, e1, card1, e2, card2, atributos)
for linea in sys.stdin.read().splitlines():
    p = linea.split()
    if not p:
        continue
    if p[0] == "entidad":          # entidad Cliente *dni nombre email
        claves = [a[1:] for a in p[2:] if a.startswith("*")]
        attrs = [a.lstrip("*") for a in p[2:]]
        entidades[p[1]] = (claves, attrs)
    elif p[0] == "relacion":       # relacion realiza Cliente 1 Pedido N [atributos...]
        relaciones.append((p[1], p[2], p[3], p[4], p[5], p[6]))

# Cada tabla: nombre → lista de columnas (nombre, es_clave, tabla_referenciada, única)
tablas = {e: [(a, a in claves, None, False) for a in attrs] for e, (claves, attrs) in entidades.items()}
nm = 0

def clave_ajena(tabla, clave, relacion, origen, destino):
    """Nombre de la columna que recibe la clave de origen en la tabla destino."""
    if origen == destino:
        return f"{clave}_{relacion}"          # relación reflexiva
    if any(col[0] == clave for col in tablas[tabla]):
        return f"{clave}_{origen.lower()}"     # el nombre ya está ocupado
    return clave

for nombre, e1, c1, e2, c2, attrs in relaciones:
    if c1 == "1" and c2 == "1":               # 1:1 → clave ajena única en la segunda
        for k in entidades[e1][0]:
            tablas[e2].append((clave_ajena(e2, k, nombre, e1, e2), False, e1, True))
        tablas[e2] += [(a, False, None, False) for a in attrs]
    elif c1 == "1" or c2 == "1":              # 1:N → la clave del lado 1 pasa al lado N
        uno, muchos = (e1, e2) if c1 == "1" else (e2, e1)
        for k in entidades[uno][0]:
            tablas[muchos].append((clave_ajena(muchos, k, nombre, uno, muchos), False, uno, False))
        tablas[muchos] += [(a, False, None, False) for a in attrs]
    else:                                     # N:M → tabla nueva
        nm += 1
        tablas[nombre] = []
        for e in (e1, e2):
            for k in entidades[e][0]:
                col = f"{k}_{nombre}" if e1 == e2 and e is e2 else clave_ajena(nombre, k, nombre, e,
                    nombre)
                tablas[nombre].append((col, True, e, False))
        tablas[nombre] += [(a, False, None, False) for a in attrs]

for t, cols in tablas.items():
    textos = [(f"*" if pk else "") + col + (f"→{ref}" if ref else "") + (" único" if unico else "")
              for col, pk, ref, unico in cols]
    print(f"{t}({', '.join(textos)})")
print(f"Tablas: {len(tablas)} ({nm} por relaciones N:M)")
```

Cada tabla se representa como una lista de columnas con lo necesario para escribirla después: nombre, si es clave primaria, a qué tabla referencia y si es única. Las entidades dan las tablas iniciales y cada relación, según su tipo, añade columnas a una tabla existente o crea una nueva: en 1:N la clave del lado 1 viaja al lado N (cada pedido guarda su cliente); en 1:1, igual pero única; y en N:M hace falta una tabla intermedia cuya clave es la unión de las dos claves.

La función `clave_ajena` resuelve los conflictos de nombres: en una relación reflexiva (un empleado dirige a empleados) la columna se llama por la relación, `id_dirige`; y si el nombre ya existe en la tabla, se le añade la entidad de origen, como `nombre_departamento`. Los diccionarios de Python conservan el orden de inserción, así que las tablas de las relaciones N:M salen detrás de las de las entidades.

6. Claves candidatas y forma normal

Muy difícil · Modelo Entidad-Relación · apuntesdam.com/subject/bases-datos/topic/modelo-er

Solución (python)

```
import sys
from itertools import combinations

lineas = [l.strip() for l in sys.stdin.read().splitlines() if l.strip()]
atributos = lineas[0][lineas[0].index("(") + 1:-1].split(",") # R(A,B,C,D)
dependencias = []
for l in lineas[1:]:
    izq, der = l.split("->")
    dependencias.append([a.strip() for a in izq.split(",")], [a.strip() for a in der.split(",")])

def ordenar(conjunto):
    return [a for a in atributos if a in conjunto]

def cierre(x):
    resultado = set(x)
    cambia = True
    while cambia:
        cambia = False
        for izq, der in dependencias:
            if set(izq) <= resultado and not set(der) <= resultado:
                resultado |= set(der)
                cambia = True
    return resultado

todos = set(atributos)
claves = []
for tam in range(1, len(atributos) + 1):
    for combo in combinations(atributos, tam):
        if cierre(combo) == todos and not any(set(k) <= set(combo) for k in claves): # mínimas
            claves.append(combo)
primos = ordenar(set().union(*map(set, claves)))
print("Claves candidatas: " + ", ".join("{ " + ", ".join(k) + "}" for k in claves))
print("Atributos primos: " + (", ".join(primos) or "ninguno"))

fallos = set()
for izq, der in dependencias:
    texto = f"{' '.join(izq)} -> {' '.join(der)}"
    x = ", ".join(izq)
    if set(der) <= set(izq):
        print(f"{texto}: trivial")
    elif cierre(izq) == todos:
        print(f"{texto}: correcta ({x} es superclave)")
    elif set(der) - set(izq) <= set(primos):
        print(f"{texto}: incumple FNBC ({x} no es superclave)")
        fallos.add("FNBC")
    else:
        parcial = next((k for k in claves if set(izq) < set(k)), None)
        if parcial:
            print(f"{texto}: incumple 2FN (dependencia parcial de la clave {{{ ' '.join(parcial)}}})")
            fallos.add("2FN")
        else:
            print(f"{texto}: incumple 3FN (dependencia transitiva)")
            fallos.add("3FN")

if "2FN" in fallos:
    forma = "1FN"
elif "3FN" in fallos:
    forma = "2FN"
elif "FNBC" in fallos:
    forma = "3FN"
else:
    forma = "FNBC"
print(f"Forma normal: {forma}")
```

El cierre de un conjunto de atributos es todo lo que se puede deducir de él aplicando las dependencias una y otra vez hasta que no se añada nada. Con él se responde a todo lo demás: un conjunto es superclave si su cierre es el esquema entero, y las claves candidatas son las superclaves mínimas, que se buscan probando combinaciones de menor a mayor tamaño y descartando las que contienen una clave ya encontrada.

Cada dependencia no trivial cuya parte izquierda no es superclave rompe alguna forma normal, y la clasificación sigue la teoría: si solo determina atributos primos, rompe únicamente la FNBC; si su izquierda es parte de una clave, es una dependencia parcial (rompe la 2FN); y si no, es transitiva (rompe la 3FN). La forma normal de la relación es la más alta que no se incumple.

En la f-string, `{{ y }}` escriben llaves literales, necesarias para mostrar una clave como `{alumno,modulo}`.

7. ¿Qué columnas pueden ser clave?

Medio · Modelo Relacional · apuntesdam.com/subject/bases-datos/topic/modelo-relacional

Solución (python)

```
import sys
from itertools import combinations

lineas = [l for l in sys.stdin.read().splitlines() if l.strip()]
columnas = lineas[0].split(",")
filas = [[v if v != "" else None for v in l.split(",")] for l in lineas[1:]]

n = len(filas)
for i, col in enumerate(columnas):
    distintos = {f[i] for f in filas if f[i] is not None}
    print(f"{col}: {len(distintos)} valores distintos de {n}")

con_nulos = [col for i, col in enumerate(columnas) if any(f[i] is None for f in filas)]
print("Columnas con nulos: " + (", ".join(con_nulos) or "ninguna"))

candidatas = [i for i, col in enumerate(columnas) if col not in con_nulos] # una clave no admite nulos
claves = []
for tam in range(1, 4):
    for combo in combinations(candidatas, tam):
        if any(set(k) <= set(combo) for k in claves):
            continue # no es mínima
        valores = {tuple(f[i] for i in combo) for f in filas}
        if len(valores) == n: # ninguna combinación de valores se repite
            claves.append(combo)
if claves:
    print("Claves candidatas según los datos: " + ", ".join("{ " + ", ".join(columnas[i] for i in k) + "}"
    for k in claves))
else:
    print("Claves candidatas según los datos: ninguna con hasta 3 columnas")
print("Aviso: los datos solo descartan claves; que hoy no haya repetidos no garantiza que no los haya mañana")
```

Una combinación de columnas puede ser clave si no tiene nulos y sus valores no se repiten en ninguna fila: basta con meter las tuplas de valores en un conjunto y comprobar que tiene tantos elementos como filas. Se prueban combinaciones de 1, 2 y 3 columnas en orden, descartando las que contienen una clave ya encontrada, para quedarse solo con las mínimas.

El aviso final es la lección importante: los datos pueden demostrar que algo no es clave (hay un repetido), pero nunca que lo sea. Que hoy no haya dos alumnos con el mismo nombre y curso no significa que no los vaya a haber; la clave se decide por el significado de los datos, y los datos solo sirven para descartar.

8. Integridad referencial al borrar e insertar

Difícil · Modelo Relacional · apuntesdam.com/subject/bases-datos/topic/modelo-relacional

Solución (python)

```
import sys
import copy

tablas = {}          # nombre -> (columnas, {id: fila como dict})
claves_ajenas = []   # (tabla, columna, tabla_ref, regla)
ordenes = []
for linea in sys.stdin.read().splitlines():
    p = linea.split()
    if not p:
        continue
    if p[0] == "tabla":
        tablas[p[1]] = (p[2:], {})
    elif p[0] == "fk":                                     # fk pedido.cliente_id -> cliente.id ON DELETE CASCADE
        t, c = p[1].split(".")
        claves_ajenas.append((t, c, p[3].split(".")[0], " ".join(p[6:])))
    elif p[0] == "fila":
        cols, filas = tablas[p[1]]
        fila = dict(zip(cols, [None if v == "NULL" else v for v in p[2:]]))
        filas[fila["id"]] = fila
    else:
        ordenes.append(p)

class Rechazo(Exception):
    pass

def borrar(datos, tabla, ident, borradas, anuladas):
    """Borra la fila y aplica las reglas ON DELETE de quienes la referencian (recursivamente)."""
    del datos[tbl][1][ident]
    borradas.append(f"{tbl}#{ident}")
    for hija, columna, ref, regla in claves_ajenas:
        if ref != tabla:
            continue
        usan = [f for f in datos[hija][1].values() if f[columna] == ident]
        if not usan:
            continue
        if regla == "RESTRICT":
            n = len(usan)
            raise Rechazo(f"no se puede borrar {tbl} {ident}: lo usan {n} {'fila' if n == 1 else 'filas'} de {hija}")
        for f in usan:
            if regla == "CASCADE":
                if f["id"] in datos[hija][1]:
                    borrar(datos, hija, f["id"], borradas, anuladas)
            else:
                f[columna] = None
                anuladas.append(f"{hija}#{f['id']}.{columna}")

for orden in ordenes:
    if orden[0] == "borrar":
        tabla, ident = orden[1], orden[2]
        prefijo = f"borrar {tbl} {ident} ->"
        if ident not in tablas[tbl][1]:
            print(f"{prefijo} ERROR: no existe {tbl} {ident}")
            continue
        copia = copy.deepcopy(tablas)
        borradas, anuladas = [], []
        try:
            borrar(copia, tabla, ident, borradas, anuladas)
        except Rechazo as e:
            print(f"{prefijo} ERROR: {e}")
            # rollback: la copia se descarta
            continue
        tablas = copia
        # commit
        print(f"{prefijo} borradas: {' '.join(borradas)}" + (f"; a NULL: {' '.join(anuladas)}" if anuladas else ""))
    elif orden[0] == "insertar":
        tabla = orden[1]
```

```

cols, filas = tablas[tbl]
fila = dict(zip(cols, [None if v == "NULL" else v for v in orden[2:]]))
prefijo = f"insertar {tbl} {fila['id']} →"
if fila["id"] in filas:
    print(f"{prefijo} ERROR: ya existe {tbl} {fila['id']}")
    continue
falta = next(((ref, fila[col]) for t, col, ref, _ in claves_ajenas
              if t == tbl and fila[col] is not None and fila[col] not in tablas[ref][1]), None)
if falta:
    print(f"{prefijo} ERROR: no existe {falta[0]} {falta[1]}")
    continue
filas[fila["id"]] = fila
print(f"{prefijo} correcto")
elif orden[0] == "mostrar":
    for nombre, (cols, filas) in tablas.items():
        texto = " | ".join(" ".join("NULL" if f[c] is None else f[c] for c in cols) for f in
                           filas.values())
        print(f"{nombre}: {texto or '(vacía)'}")

```

El borrado es recursivo: se borra la fila y, para cada clave ajena que apunta a su tabla, se aplica su regla a las filas que la usan; con CASCADE se las borra a ellas, con lo que la regla se aplica también a sus dependientes. Así se recorre en profundidad todo lo que cuelga de la fila, en el orden que muestra el resultado.

Para que el borrado sea «todo o nada» se trabaja sobre una copia profunda de las tablas: si en cualquier punto aparece un RESTRICT, se lanza una excepción, la copia se descarta y nada ha cambiado (rollback); si todo va bien, la copia sustituye a las tablas (commit). Es lo que hace un SGBD con una transacción.

Insertar comprueba lo contrario: que el id no exista y que cada clave ajena no nula apunte a una fila existente, porque una fila hija no puede quedar huérfana.

9. Un intérprete de álgebra relacional

Muy difícil · Modelo Relacional · apuntesdam.com/subject/bases-datos/topic/modelo-relacional

Solución (python)

```
import sys

relaciones = {}          # nombre → (columnas, lista de tuplas)

class ErrorAlgebra(Exception):
    pass

def obtener(nombre):
    if nombre not in relaciones:
        raise ErrorAlgebra(f"no existe {nombre}")
    return relaciones[nombre]

def columna(nombre, cols, c):
    if c not in cols:
        raise ErrorAlgebra(f"no existe la columna {c} en {nombre}")
    return cols.index(c)

def sin_repetidos(tuplas):
    return list(dict.fromkeys(tuplas))          # conserva la primera aparición

def compara(a, op, b):
    try:
        a, b = float(a), float(b)              # como números si los dos lo son
    except ValueError:
        pass
    return {"=": a == b, "<": a < b, ">": a > b, "<=": a <= b, ">=": a >= b}[op]

def operar(op, args):
    if op == "seleccion":
        cols, filas = obtener(args[0])
        for o in ("<>", "=", ">", "<"):
            if o in args[1]:
                c, valor = args[1].split(o, 1)
                break
        i = columna(args[0], cols, c)
        return cols, [t for t in filas if compara(t[i], o, valor)]
    if op == "proyeccion":
        cols, filas = obtener(args[0])
        indices = [columna(args[0], cols, c) for c in args[1].split(",")]
        return [cols[i] for i in indices], sin_repetidos(tuple(t[i] for i in indices) for t in filas)
    if op == "join":
        c1, f1 = obtener(args[0])
        c2, f2 = obtener(args[1])
        a, b = args[2].split("=")
        i, j = columna(args[0], c1, a), columna(args[1], c2, b)
        cols = c1 + [c if c not in c1 else f"{args[1]}.{c}" for c in c2]
        return cols, [t1 + t2 for t1 in f1 for t2 in f2 if t1[i] == t2[j]]
    if op in ("union", "diferencia", "interseccion"):
        c1, f1 = obtener(args[0])
        c2, f2 = obtener(args[1])
        if c1 != c2:
            raise ErrorAlgebra(f"esquemas incompatibles ({','.join(c1)}) y ({','.join(c2)})")
        if op == "union":
            return c1, sin_repetidos(f1 + f2)
        if op == "diferencia":
            return c1, sin_repetidos(t for t in f1 if t not in f2)
        return c1, sin_repetidos(t for t in f1 if t in f2)
    if op == "renombrar":
        cols, filas = obtener(args[0])
        viejo, nuevo = args[1].split("=")
        i = columna(args[0], cols, viejo)
        return cols[:i] + [nuevo] + cols[i + 1:], list(filas)
    raise ErrorAlgebra(f"operación desconocida {op}")

lineas = sys.stdin.read().splitlines()
i = 0
```

```

while i < len(lineas):
    l = lineas[i].strip()
    if l.startswith("tabla "):
        # tabla nombre col1,col2 ... filas ... fin
        nombre, cols = l.split()[1], l.split()[2].split(",")
        filas = []
        i += 1
        while lineas[i].strip() != "fin":
            filas.append(tuple(lineas[i].strip().split(",")))
            i += 1
        relaciones[nombre] = (cols, filas)
    elif l.startswith("mostrar "):
        nombre = l.split()[1]
        if nombre not in relaciones:
            print(f"ERROR: no existe {nombre}")
        else:
            cols, filas = relaciones[nombre]
            print(f"{nombre}({', '.join(cols)})")
            for t in filas:
                print(" " + " | ".join(t))
            print(f" {len(filas)} {'tupla' if len(filas) == 1 else 'tuplas'}")
    elif l:
        destino, expresion = [x.strip() for x in l.split("=", 1)]
        op, *args = expresion.split()
        try:
            relaciones[destino] = operar(op, args)
        except ErrorAlgebra as e:
            print(f"ERROR en {destino}: {e}")
    i += 1

```

Cada operación del álgebra relacional recibe relaciones (columnas y tuplas) y devuelve otra relación, así que se pueden encadenar guardando cada resultado con un nombre. La función `operar` implementa cada una en pocas líneas: la selección filtra tuplas, la proyección se queda con columnas, el join combina las tuplas que cumplen la condición y las operaciones de conjuntos exigen el mismo esquema.

Una relación es un conjunto: no tiene tuplas repetidas, y por eso la proyección y las operaciones de conjuntos las eliminan con `dict.fromkeys`, que además conserva el orden de la primera aparición. Los errores se lanzan como excepción desde donde se detectan y se capturan en un solo sitio; así el resultado no se guarda y un `mostrar` posterior dice que no existe.

Esta es la teoría que hay debajo de SQL: `SELECT nombre FROM alumno JOIN matricula ... WHERE curso = '1DAM'` es una selección, un join y una proyección.

10. Encuentra el fallo: los clientes sin email

Fácil · Consultas SQL Avanzadas · apuntesdam.com/subject/bases-datos/topic/consultas

Solución (sql)

```

SELECT nombre
FROM Clientes
WHERE email IS NULL;

```

Cómo se resuelve

- ¿Qué resultado da comparar cualquier valor con NULL usando `=`?
- NULL significa «desconocido»: `email = NULL` no es verdadero ni falso, es desconocido, y WHERE solo deja pasar lo verdadero.
- Para preguntar por NULL se usa `IS NULL` (o `IS NOT NULL`).

11. Encuentra el fallo: las prioridades de AND y OR

Medio · Consultas SQL Avanzadas · apuntesdam.com/subject/bases-datos/topic/consultas

Solución (sql)

```
SELECT nombre, precio
FROM Productos
WHERE (categoria = 'Periféricos' OR categoria = 'Audio') AND precio < 50
ORDER BY nombre;
```

Cómo se resuelve

- Evalúa la condición a mano para el teclado mecánico: ¿qué parte la hace verdadera?
- AND se evalúa antes que OR, igual que la multiplicación antes que la suma: la condición es «Periféricos» OR («Audio» AND barato).
- Agrupa las dos categorías entre paréntesis, o usa categoria IN ('Periféricos', 'Audio').

12. Encuentra el fallo: la media que sale entera

Medio · Consultas SQL Avanzadas · apuntesdam.com/subject/bases-datos/topic/consultas

Solución (sql)

```
SELECT SUM(cantidad) * 1.0 / COUNT(DISTINCT pedido_id) AS media
FROM DetallePedido;
```

Cómo se resuelve

- ¿De qué tipo son SUM(cantidad) y COUNT(DISTINCT pedido_id)?
- En SQLite, como en otros gestores, dividir dos enteros da un entero: $13 / 5 = 2$.
- Convierte uno de los dos a decimal: SUM(cantidad) * 1.0, o CAST(SUM(cantidad) AS REAL).

13. Encuentra el fallo: el pedido fantasma

Medio · Consultas SQL Avanzadas · apuntesdam.com/subject/bases-datos/topic/consultas

Solución (sql)

```
SELECT c.nombre, COUNT(p.pedido_id) AS pedidos
FROM Clientes c
LEFT JOIN Pedidos p ON p.cliente_id = c.cliente_id
GROUP BY c.cliente_id
ORDER BY pedidos DESC, c.nombre;
```

Cómo se resuelve

- Ejecuta la consulta sin GROUP BY ni COUNT: ¿qué fila produce el LEFT JOIN para Marta?
- Para un cliente sin pedidos, el LEFT JOIN genera una fila con las columnas de Pedidos a NULL, y COUNT(*) cuenta filas, así que la cuenta.
- COUNT(columna) no cuenta los NULL: usa COUNT(p.pedido_id).

14. Encuentra el fallo: el LEFT JOIN que pierde clientes

Difícil · Consultas SQL Avanzadas · apuntesdam.com/subject/bases-datos/topic/consultas

Solución (sql)

```
SELECT c.nombre, COUNT(p.pedido_id) AS entregados
FROM Clientes c
LEFT JOIN Pedidos p ON p.cliente_id = c.cliente_id AND p.estado = 'entregado'
GROUP BY c.cliente_id
ORDER BY c.cliente_id;
```

Cómo se resuelve

- ¿Qué valor tiene p.estado en la fila que el LEFT JOIN genera para Marta? ¿Y en la de Lucía, cuyo único pedido está pendiente?

- El WHERE se aplica después del JOIN: p.estado = 'entregado' descarta las filas con NULL o con otro estado, y el LEFT JOIN acaba comportándose como un INNER JOIN.
- Mueve la condición del estado a la cláusula ON: LEFT JOIN Pedidos p ON p.cliente_id = c.cliente_id AND p.estado = 'entregado'.

15. Encuentra el fallo: las bajas que lo borran todo

Muy difícil · Consultas SQL Avanzadas · apuntesdam.com/subject/bases-datos/topic/consultas

Solución (sql)

```
SELECT nombre
FROM Clientes c
WHERE c.email IS NOT NULL
AND NOT EXISTS (SELECT 1 FROM Bajas b WHERE b.email = c.email)
ORDER BY nombre;
```

Cómo se resuelve

- Prueba a quitar de la subconsulta la fila con NULL: ¿cambia el resultado?
- email NOT IN ('carlos@email.com', NULL) equivale a email <> 'carlos@email.com' AND email <> NULL, y lo segundo siempre es desconocido: ninguna fila pasa el filtro.
- Quita los NULL en la subconsulta (WHERE email IS NOT NULL) o usa NOT EXISTS, que no tiene este problema (y entonces pide además email IS NOT NULL en el cliente).

16. Encuentra el fallo: la actualización que no actualiza nada

Fácil · Modificación de Datos en SQL · apuntesdam.com/subject/bases-datos/topic/modificacion

Solución (sql)

```
UPDATE Productos
SET stock = 0
WHERE categoria = 'Audio';
```

Cómo se resuelve

- Mira cómo está escrita exactamente la categoría en la tabla.
- En SQLite (y en PostgreSQL) la comparación con = distingue mayúsculas de minúsculas: 'audio' no es 'Audio'.
- Escribe el valor tal y como está guardado, o compara con LOWER(categoria) = 'audio'.

17. Encuentra el fallo: el stock que se vuelve NULL

Difícil · Modificación de Datos en SQL · apuntesdam.com/subject/bases-datos/topic/modificacion

Solución (sql)

```
UPDATE Productos
SET stock = stock - (
    SELECT d.cantidad FROM DetallePedido d
    WHERE d.pedido_id = 3 AND d.producto_id = Productos.producto_id
)
WHERE producto_id IN (SELECT producto_id FROM DetallePedido WHERE pedido_id = 3);
```

Cómo se resuelve

- Un UPDATE sin WHERE cambia todas las filas de la tabla.
- Para un producto que no está en el pedido, la subconsulta no devuelve ninguna fila, es decir, NULL, y stock - NULL es NULL.
- Limita el UPDATE con WHERE producto_id IN (SELECT producto_id FROM DetallePedido WHERE pedido_id = 3), o resta COALESCE(subconsulta, 0).

18. Encuentra el fallo: el disparador que descuenta a todos

Difícil · Programación en Bases de Datos · apuntesdam.com/subject/bases-datos/topic/programacion-bdd

Solución (sql)

```
CREATE TRIGGER descontar_stock
AFTER INSERT ON DetallePedido
BEGIN
    UPDATE Productos
    SET stock = stock - NEW.cantidad
    WHERE producto_id = NEW.producto_id;
END;

INSERT INTO DetallePedido VALUES (4, 2, 5, 24.99);
```

Cómo se resuelve

- ¿Para qué filas de Productos es verdadera la condición `producto_id = producto_id`?
- Dentro del disparador, `producto_id` sin más es la columna de la fila de Productos que se está mirando: compararla consigo misma siempre es verdad.
- La fila recién insertada se lee con `NEW`: `WHERE producto_id = NEW.producto_id`.

19. Productos baratos

Fácil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
SELECT nombre, precio
FROM Productos
WHERE precio < 50
ORDER BY precio;
```

Cómo se resuelve

- Filtra con `WHERE precio < 50`.
- Ordena con `ORDER BY precio` (ASC es el valor por defecto).

20. Clientes sin email

Fácil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
SELECT nombre, ciudad
FROM Clientes
WHERE email IS NULL;
```

Cómo se resuelve

- Un valor desconocido es `NULL`, y `NULL` no se compara con `=`.
- Usa `WHERE email IS NULL`.

21. Búsqueda por patrón

Fácil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
SELECT nombre
FROM Productos
WHERE LOWER(nombre) LIKE '%portátil%' OR nombre LIKE 'Disco%';
```

Cómo se resuelve

- `LIKE '%texto%'` busca el texto en cualquier posición; `LIKE 'Disco%'` al principio.
- Combina las dos condiciones con `OR`. Para ignorar mayúsculas usa `LOWER(nombre)`.

22. Alta de un producto

Fácil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
INSERT INTO Productos (producto_id, nombre, categoria, precio, stock, proveedor, fecha_alta)
VALUES (9, 'Alfombrilla XL', 'Periféricos', 14.95, 80, 'Logitech', '2024-07-01');
```

Cómo se resuelve

- INSERT INTO Productos (columnas...) VALUES (valores...);
- Los textos y las fechas van entre comillas simples; los números sin comillas.

23. Estadísticas del catálogo

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
SELECT categoria, COUNT(*) AS productos, MIN(precio) AS minimo, MAX(precio) AS maximo, SUM(stock) AS
stock
FROM Productos
GROUP BY categoria
ORDER BY categoria;
```

Cómo se resuelve

- Agrupa con GROUP BY categoria.
- Usa COUNT(*), MIN(precio), MAX(precio) y SUM(stock).

24. Pedidos con su cliente

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
SELECT p.pedido_id, p.fecha, c.nombre, c.ciudad
FROM Pedidos p
JOIN Clientes c ON c.cliente_id = p.cliente_id
ORDER BY p.fecha;
```

Cómo se resuelve

- Necesitas unir Pedidos y Clientes por cliente_id.
- SELECT p.pedido_id, p.fecha, c.nombre, c.ciudad FROM Pedidos p JOIN Clientes c ON ...

25. Importe de cada pedido

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
SELECT pedido_id, ROUND(SUM(cantidad * precio_unitario), 2) AS importe
FROM DetallePedido
GROUP BY pedido_id
ORDER BY importe DESC;
```

Cómo se resuelve

- Los importes están en DetallePedido.
- SUM(cantidad * precio_unitario) agrupado por pedido_id; ROUND(..., 2).

26. Clientes que nunca han comprado

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
SELECT c.nombre
FROM Clientes c
LEFT JOIN Pedidos p ON p.cliente_id = c.cliente_id
WHERE p.pedido_id IS NULL;
```

Cómo se resuelve

- Con LEFT JOIN, los clientes sin pedidos tienen NULL en las columnas de Pedidos.
- WHERE p.pedido_id IS NULL, o bien WHERE NOT EXISTS (SELECT 1 FROM Pedidos p WHERE p.cliente_id = c.cliente_id).

27. Ciudades de clientes y proveedores

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
SELECT ciudad AS nombre FROM Clientes WHERE ciudad IS NOT NULL
UNION
SELECT proveedor FROM Productos
ORDER BY nombre;
```

Cómo se resuelve

- Une dos consultas de una columna con UNION (elimina duplicados).
- El ORDER BY va al final de la consulta completa.

28. Subida de precios

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
UPDATE Productos
SET precio = ROUND(precio * 1.05, 2)
WHERE proveedor = 'Samsung';
```

Cómo se resuelve

- UPDATE ... SET precio = ... WHERE proveedor = 'Samsung';
- ROUND(precio * 1.05, 2).

29. Crear una vista

Medio · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
CREATE VIEW StockBajo AS
SELECT nombre, categoria, stock
FROM Productos
WHERE stock < 25;
```

Cómo se resuelve

- CREATE VIEW nombre AS SELECT ...;
- La vista se comprueba consultándola: SELECT * FROM StockBajo.

30. Productos más caros que la media de su categoría

Difícil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
SELECT p.nombre, p.categoria, p.precio
FROM Productos p
WHERE p.precio > (SELECT AVG(p2.precio) FROM Productos p2 WHERE p2.categoria = p.categoria);
```

Cómo se resuelve

- Necesitas una subconsulta correlacionada: la media depende de la fila exterior.
- WHERE precio > (SELECT AVG(p2.precio) FROM Productos p2 WHERE p2.categoria = p.categoria).

31. Mejor cliente por gasto

Difícil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
SELECT c.nombre, ROUND(SUM(d.cantidad * d.precio_unitario), 2) AS gasto
FROM Clientes c
JOIN Pedidos p ON p.cliente_id = c.cliente_id
JOIN DetallePedido d ON d.pedido_id = p.pedido_id
GROUP BY c.cliente_id, c.nombre
ORDER BY gasto DESC
LIMIT 1;
```

Cómo se resuelve

- Une Clientes, Pedidos y DetallePedido y agrupa por cliente.
- Ordena por el total descendente y quédate con la primera fila con LIMIT 1.

32. Categorías con más de 100 € vendidos

Difícil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
SELECT pr.categoria, ROUND(SUM(d.cantidad * d.precio_unitario), 2) AS vendido
FROM DetallePedido d
JOIN Productos pr ON pr.producto_id = d.producto_id
GROUP BY pr.categoria
HAVING SUM(d.cantidad * d.precio_unitario) > 100
ORDER BY vendido DESC;
```

Cómo se resuelve

- Primero JOIN de DetallePedido con Productos para conocer la categoría de cada línea.
- Filtra grupos con HAVING SUM(...) > 100 (WHERE no sirve para agregados).

33. Borrado con subconsulta

Difícil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
DELETE FROM DetallePedido
WHERE pedido_id IN (SELECT pedido_id FROM Pedidos
                    WHERE estado = 'pendiente'
                    AND cliente_id IN (SELECT cliente_id FROM Clientes WHERE ciudad = 'Madrid'));

DELETE FROM Pedidos
WHERE estado = 'pendiente'
AND cliente_id IN (SELECT cliente_id FROM Clientes WHERE ciudad = 'Madrid');
```

Cómo se resuelve

- Primero DELETE FROM DetallePedido WHERE pedido_id IN (...), después DELETE FROM Pedidos WHERE

- La subconsulta de pedidos: `SELECT pedido_id FROM Pedidos WHERE estado = 'pendiente' AND cliente_id IN (SELECT cliente_id FROM Clientes WHERE ciudad = 'Madrid')`.

34. Trigger de auditoría

Difícil · Ejercicios de Bases de Datos · apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos

Solución (sql)

```
CREATE TABLE HistorialPrecios (producto_id INTEGER, precio_anterior REAL, precio_nuevo REAL);

CREATE TRIGGER auditar_precio
AFTER UPDATE OF precio ON Productos
WHEN OLD.precio <> NEW.precio
BEGIN
    INSERT INTO HistorialPrecios VALUES (NEW.producto_id, OLD.precio, NEW.precio);
END;

UPDATE Productos SET precio = precio + 10 WHERE producto_id = 3;
```

Cómo se resuelve

- `CREATE TRIGGER nombre AFTER UPDATE OF precio ON Productos ... BEGIN ... END;`
- Dentro del trigger, `OLD.precio` y `NEW.precio` son los valores antes y después del cambio.

35. La base de datos de una academia de idiomas

Difícil · SQL · 75 minutos · apuntesdam.com/ejercicios/sql/academia-de-idiomas

Apartado 1 (sql)

```
SELECT nombre, email
FROM alumnos
WHERE ciudad = 'Madrid' AND email IS NOT NULL
ORDER BY nombre;
```

Apartado 2 (sql)

```
SELECT c.idioma, c.nivel, p.nombre, c.precio_mes
FROM cursos c
JOIN profesores p ON p.profesor_id = c.profesor_id
ORDER BY c.idioma, c.nivel;
```

Apartado 3 (sql)

```
SELECT c.idioma, c.nivel, COUNT(m.alumno_id) AS matriculados
FROM cursos c
LEFT JOIN matriculas m ON m.curso_id = c.curso_id
GROUP BY c.curso_id
ORDER BY c.idioma, c.nivel;
```

Apartado 4 (sql)

```
SELECT c.idioma, c.nivel, COUNT(m.alumno_id) AS matriculados, c.plazas,
    ROUND(100.0 * COUNT(m.alumno_id) / c.plazas, 1) AS ocupacion
FROM cursos c
LEFT JOIN matriculas m ON m.curso_id = c.curso_id
GROUP BY c.curso_id
ORDER BY ocupacion DESC, c.idioma, c.nivel;
```

Apartado 5 (sql)

```
SELECT c.idioma, COUNT(m.nota) AS evaluados, ROUND(AVG(m.nota), 2) AS nota_media
FROM matriculas m
JOIN cursos c ON c.curso_id = m.curso_id
WHERE m.nota IS NOT NULL
GROUP BY c.idioma
ORDER BY nota_media DESC;
```

Apartado 6 (sql)

```
SELECT nombre, total, pagado, total - pagado AS deuda
FROM (
  SELECT a.nombre,
    (SELECT SUM(c.precio_mes) FROM matriculas m JOIN cursos c ON c.curso_id = m.curso_id
     WHERE m.alumno_id = a.alumno_id) AS total,
    COALESCE((SELECT SUM(p.importe) FROM pagos p WHERE p.alumno_id = a.alumno_id), 0) AS pagado
  FROM alumnos a
)
WHERE total - pagado > 0
ORDER BY deuda DESC;
```

Apartado 7 (sql)

```
SELECT p.nombre, p.idioma, COUNT(DISTINCT m.alumno_id) AS alumnos
FROM profesores p
LEFT JOIN cursos c ON c.profesor_id = p.profesor_id
LEFT JOIN matriculas m ON m.curso_id = c.curso_id
GROUP BY p.profesor_id
ORDER BY alumnos DESC, p.nombre;
```

Apartado 8 (sql)

```
SELECT c.idioma, a.nombre, m.nota
FROM matriculas m
JOIN cursos c ON c.curso_id = m.curso_id
JOIN alumnos a ON a.alumno_id = m.alumno_id
WHERE m.nota = (
  SELECT MAX(m2.nota)
  FROM matriculas m2
  JOIN cursos c2 ON c2.curso_id = m2.curso_id
  WHERE c2.idioma = c.idioma
)
ORDER BY c.idioma;
```

Apartado 9 (sql)

```
UPDATE cursos
SET precio_mes = ROUND(precio_mes * 1.05, 2)
WHERE idioma = 'inglés'
AND (SELECT COUNT(*) FROM matriculas m WHERE m.curso_id = cursos.curso_id) > 2;
```

Los apartados 3, 4 y 7 tienen el mismo esqueleto: la tabla de la que queremos todas las filas a la izquierda, `LEFT JOIN` hacia la tabla que se cuenta y `COUNT` de una columna de esa tabla, que vale 0 cuando no hay coincidencias. En el 7 además hace falta `DISTINCT`, porque un alumno puede estar en dos cursos del mismo profesor.

En el 4 la trampa es la división entera: `COUNT(...)` y `plazas` son enteros, y `3 / 12` vale 0. Multiplicar por `100.0` antes de dividir convierte la operación en decimal. El orden por `ocupacion DESC, idioma, nivel` deshace los empates de forma predecible.

El 6 es el clásico error de multiplicar filas al unir dos relaciones uno a muchos con la misma tabla: cada alumno tiene varios cursos y puede tener varios pagos. Calculando cada total en su propia subconsulta no hay repeticiones; `COALESCE` convierte en 0 la suma de quien no ha pagado nada, y el filtro `total - pagado > 0` descarta también a Álvaro, cuyo total es `NULL`.

El 8 resuelve el problema de «el máximo y de quién es» con una subconsulta correlacionada: para cada matrícula se calcula el máximo de su propio idioma. Si hubiera un empate en la nota máxima, saldrían los dos alumnos, que es lo correcto.

En el UPDATE, la subconsulta cuenta los matriculados del curso que se está modificando (`cursos.curso_id`). Solo cumplen la condición los cursos 1 (3 alumnos) y 2 (4 alumnos): 55 pasa a 57.75 y 60 a 63.

36. Clínica: citas, especialidades y facturación

Difícil · SQL · 80 minutos · apuntesdam.com/ejercicios/sql/clinica-citas-y-facturacion

Apartado 1 (sql)

```
SELECT nombre,
       (strftime('%Y', '2026-10-01') - strftime('%Y', nacimiento))
       - (strftime('%m-%d', nacimiento) > '10-01') AS edad
FROM pacientes
WHERE ciudad = 'Madrid' AND aseguradora IS NULL
ORDER BY edad DESC;
```

Apartado 2 (sql)

```
SELECT c.hora, p.nombre, m.nombre, e.nombre
FROM citas c
JOIN pacientes p ON p.paciente_id = c.paciente_id
JOIN medicos m ON m.medico_id = c.medico_id
JOIN especialidades e ON e.especialidad_id = m.especialidad_id
WHERE c.fecha = '2026-10-05'
ORDER BY c.hora;
```

Apartado 3 (sql)

```
SELECT e.nombre, COUNT(c.cita_id) AS citas
FROM especialidades e
LEFT JOIN medicos m ON m.especialidad_id = e.especialidad_id
LEFT JOIN citas c ON c.medico_id = m.medico_id AND c.estado = 'realizada'
GROUP BY e.especialidad_id
ORDER BY citas DESC, e.nombre;
```

Apartado 4 (sql)

```
SELECT m.nombre, COUNT(*) AS citas, SUM(c.importe) AS total
FROM citas c
JOIN medicos m ON m.medico_id = c.medico_id
WHERE c.estado = 'realizada' AND c.fecha BETWEEN '2026-09-01' AND '2026-09-30'
GROUP BY m.medico_id
HAVING SUM(c.importe) > 100
ORDER BY total DESC;
```

Apartado 5 (sql)

```
SELECT p.nombre
FROM pacientes p
WHERE EXISTS (SELECT 1 FROM citas c WHERE c.paciente_id = p.paciente_id)
AND NOT EXISTS (SELECT 1 FROM citas c WHERE c.paciente_id = p.paciente_id AND c.estado = 'cancelada')
ORDER BY p.nombre;
```

Apartado 6 (sql)

```
SELECT m.nombre, COUNT(*) AS citas,
       SUM(c.estado = 'cancelada') AS canceladas,
       ROUND(100.0 * SUM(c.estado = 'cancelada') / COUNT(*), 1) AS porcentaje
FROM medicos m
JOIN citas c ON c.medico_id = m.medico_id
GROUP BY m.medico_id
HAVING COUNT(*) >= 3
ORDER BY porcentaje DESC, m.nombre;
```

Apartado 7 (sql)

```
SELECT p.nombre, COUNT(DISTINCT m.especialidad_id) AS especialidades
FROM pacientes p
JOIN citas c ON c.paciente_id = p.paciente_id AND c.estado = 'realizada'
JOIN medicos m ON m.medico_id = c.medico_id
GROUP BY p.paciente_id
HAVING COUNT(DISTINCT m.especialidad_id) > 1
ORDER BY p.nombre;
```

Apartado 8 (sql)

```
WITH ordenadas AS (  
  SELECT c.*, ROW_NUMBER() OVER (PARTITION BY c.paciente_id ORDER BY c.fecha DESC, c.hora DESC) AS n  
  FROM citas c  
  WHERE c.estado = 'realizada'  
)  
SELECT p.nombre, o.fecha, m.nombre  
FROM ordenadas o  
JOIN pacientes p ON p.paciente_id = o.paciente_id  
JOIN medicos m ON m.medico_id = o.medico_id  
WHERE o.n = 1  
ORDER BY p.nombre;
```

Apartado 9 (sql)

```
UPDATE citas  
SET estado = 'no presentado'  
WHERE estado = 'pendiente' AND fecha < '2026-10-01';
```

La edad se calcula restando años y corrigiendo con una comparación de 'MM-DD': Rosa nació un 2 de octubre y el 1 de octubre aún tiene 50 años, no 51. En SQLite una comparación vale 1 o 0 y se puede restar directamente.

El apartado de especialidades muestra la trampa más frecuente del `LEFT JOIN`: un `WHERE` sobre la tabla de la derecha lo convierte en un `JOIN` normal. La condición del estado tiene que ir en el `ON` para que Cardiología salga con 0.

`EXISTS` y `NOT EXISTS` responden preguntas sobre el conjunto de filas de cada paciente («alguna», «ninguna»), que no se pueden responder mirando las filas de una en una con `WHERE`.

`ROW_NUMBER` con `PARTITION BY` resuelve el problema del «máximo por grupo con el resto de su fila», que con `GROUP BY` solo se puede hacer con subconsultas correlacionadas más difíciles de leer.

37. Liga de baloncesto: clasificación y funciones de ventana

Muy difícil · SQL · 90 minutos · apuntesdam.com/ejercicios/sql/liga-de-baloncesto-funciones-de-ventana

Apartado 1 (sql)

```
WITH resultados AS (  
  SELECT local_id AS equipo_id, puntos_local AS favor, puntos_visitante AS contra FROM partidos  
  UNION ALL  
  SELECT visitante_id, puntos_visitante, puntos_local FROM partidos  
)  
SELECT e.nombre, COUNT(*) AS jugados,  
       SUM(r.favor > r.contra) AS victorias,  
       SUM(r.favor < r.contra) AS derrotas,  
       SUM(r.favor) AS favor, SUM(r.contra) AS contra,  
       SUM(r.favor) - SUM(r.contra) AS diferencia  
FROM resultados r  
JOIN equipos e ON e.equipo_id = r.equipo_id  
GROUP BY e.equipo_id  
ORDER BY victorias DESC, diferencia DESC, e.nombre;
```

Apartado 2 (sql)

```
WITH ranking AS (  
  SELECT s.partido_id, j.nombre, s.puntos,  
         ROW_NUMBER() OVER (PARTITION BY s.partido_id ORDER BY s.puntos DESC, j.nombre) AS n  
  FROM estadisticas s  
  JOIN jugadores j ON j.jugador_id = s.jugador_id  
)  
SELECT partido_id, nombre, puntos FROM ranking WHERE n = 1 ORDER BY partido_id;
```

Apartado 3 (sql)

```
SELECT RANK() OVER (ORDER BY SUM(s.puntos) DESC) AS puesto,
       j.nombre, e.nombre, SUM(s.puntos) AS puntos
FROM estadisticas s
JOIN jugadores j ON j.jugador_id = s.jugador_id
JOIN equipos e ON e.equipo_id = j.equipo_id
GROUP BY j.jugador_id
ORDER BY puesto, j.nombre;
```

Apartado 4 (sql)

```
SELECT j.nombre, p.jornada, s.puntos,
       SUM(s.puntos) OVER (PARTITION BY j.jugador_id ORDER BY p.jornada) AS acumulado
FROM estadisticas s
JOIN jugadores j ON j.jugador_id = s.jugador_id
JOIN partidos p ON p.partido_id = s.partido_id
WHERE j.equipo_id = 1
ORDER BY j.nombre, p.jornada;
```

Apartado 5 (sql)

```
SELECT j.nombre, p.jornada, s.puntos,
       s.puntos - LAG(s.puntos) OVER (PARTITION BY j.jugador_id ORDER BY p.jornada) AS diferencia
FROM estadisticas s
JOIN jugadores j ON j.jugador_id = s.jugador_id
JOIN partidos p ON p.partido_id = s.partido_id
WHERE j.equipo_id = 3
ORDER BY j.nombre, p.jornada;
```

Apartado 6 (sql)

```
WITH totales AS (
  SELECT j.equipo_id, j.nombre, SUM(s.puntos) AS puntos
  FROM estadisticas s
  JOIN jugadores j ON j.jugador_id = s.jugador_id
  GROUP BY j.jugador_id
)
SELECT e.nombre, t.nombre, t.puntos,
       ROUND(100.0 * t.puntos / SUM(t.puntos) OVER (PARTITION BY t.equipo_id), 1) AS porcentaje
FROM totales t
JOIN equipos e ON e.equipo_id = t.equipo_id
ORDER BY e.nombre, porcentaje DESC;
```

Apartado 7 (sql)

```
WITH medias AS (
  SELECT j.nombre, j.posicion, AVG(s.puntos) AS media
  FROM estadisticas s
  JOIN jugadores j ON j.jugador_id = s.jugador_id
  GROUP BY j.jugador_id
), comparadas AS (
  SELECT nombre, posicion, media, AVG(media) OVER (PARTITION BY posicion) AS media_posicion
  FROM medias
)
SELECT nombre, posicion, ROUND(media, 1), ROUND(media_posicion, 1)
FROM comparadas
WHERE media > media_posicion
ORDER BY posicion, media DESC;
```

La clasificación muestra un truco general: cuando una fila habla de dos cosas (local y visitante), se «desdobla» con **UNION ALL** para tener una fila por equipo y partido. A partir de ahí todo es un **GROUP BY** normal.

ROW_NUMBER, **RANK** y **DENSE_RANK** numeran de forma distinta los empates: en el ranking, Jon Ibarra y Pablo Romero comparten el 7.º puesto y el siguiente es el 9.º. Elegir la función es elegir la regla de los empates.

Con **ORDER BY** dentro de **OVER**, **SUM** se convierte en un acumulado, y **LAG** mira la fila anterior del mismo grupo. Son las dos funciones de ventana que más se usan en informes de evolución (ventas mes a mes, puntos jornada a jornada).

Los dos últimos apartados combinan agregación y ventanas en varias CTE: primero se resume por jugador y después se compara cada jugador con su grupo. Encadenar CTE con nombres claros mantiene legible una consulta que, con subconsultas anidadas, sería difícil de seguir.

38. Banco: vistas, disparadores, UPSERT y transacciones

Muy difícil · SQL · 100 minutos · apuntesdam.com/ejercicios/sql/banco-vistas-disparadores-y-transacciones

Apartado 1 (sql)

```
CREATE VIEW v_saldos AS
SELECT cl.nombre, c.numero, c.tipo, c.saldo
FROM cuentas c
JOIN clientes cl ON cl.cliente_id = c.cliente_id
WHERE c.cerrada IS NULL;
```

Apartado 2 (sql)

```
CREATE TRIGGER trg_saldo AFTER INSERT ON movimientos
BEGIN
    UPDATE cuentas SET saldo = ROUND(saldo + NEW.importe, 2) WHERE cuenta_id = NEW.cuenta_id;
END;

INSERT INTO movimientos (cuenta_id, fecha, concepto, importe) VALUES
    (3, '2026-10-03', 'Gimnasio', -39.90),
    (5, '2026-10-03', 'Ingreso', 200);
```

Apartado 3 (sql)

```
CREATE TRIGGER trg_ahorro BEFORE INSERT ON movimientos
WHEN (SELECT tipo FROM cuentas WHERE cuenta_id = NEW.cuenta_id) = 'ahorro'
AND (SELECT saldo FROM cuentas WHERE cuenta_id = NEW.cuenta_id) + NEW.importe < 0
BEGIN
    SELECT RAISE(IGNORE);
END;

INSERT INTO movimientos (cuenta_id, fecha, concepto, importe) VALUES (2, '2026-10-04', 'Retirada',
    -9000);
INSERT INTO movimientos (cuenta_id, fecha, concepto, importe) VALUES (5, '2026-10-04', 'Retirada', -500);
INSERT INTO movimientos (cuenta_id, fecha, concepto, importe) VALUES (1, '2026-10-04', 'Recibo', -2000);
```

Apartado 4 (sql)

```
UPDATE cuentas
SET saldo = ROUND(saldo * 1.015, 2)
WHERE tipo = 'ahorro' AND cerrada IS NULL AND saldo > 1000;
```

Apartado 5 (sql)

```
INSERT INTO movimientos (cuenta_id, fecha, concepto, importe)
SELECT cuenta_id, '2026-10-31', 'Comisión de mantenimiento', -2
FROM cuentas
WHERE tipo = 'corriente' AND cerrada IS NULL AND saldo < 1500;
```

Apartado 6 (sql)

```
DELETE FROM movimientos
WHERE cuenta_id IN (SELECT cuenta_id FROM cuentas WHERE cerrada < '2021-10-01');
```

Apartado 7 (sql)

```
INSERT INTO resumen (cuenta_id, mes, ingresos, gastos)
SELECT cuenta_id, '2026-09',
    ROUND(SUM(CASE WHEN importe > 0 THEN importe ELSE 0 END), 2),
    ROUND(SUM(CASE WHEN importe < 0 THEN -importe ELSE 0 END), 2)
FROM movimientos
WHERE fecha LIKE '2026-09-%'
GROUP BY cuenta_id
ON CONFLICT (cuenta_id, mes) DO UPDATE SET ingresos = excluded.ingresos, gastos = excluded.gastos;
```

Apartado 8 (sql)

```
BEGIN;
UPDATE cuentas SET saldo = saldo - 150 WHERE numero = 'C-001';
UPDATE cuentas SET saldo = saldo + 150 WHERE numero = 'A-002';
INSERT INTO movimientos (cuenta_id, fecha, concepto, importe)
  SELECT cuenta_id, '2026-10-05', 'Traspaso enviado', -150 FROM cuentas WHERE numero = 'C-001';
INSERT INTO movimientos (cuenta_id, fecha, concepto, importe)
  SELECT cuenta_id, '2026-10-05', 'Traspaso recibido', 150 FROM cuentas WHERE numero = 'A-002';
COMMIT;
```

Las vistas guardan consultas, no datos: `v_saldos` siempre muestra los saldos actuales. Los disparadores mueven reglas de negocio a la base de datos, de modo que se cumplen aunque los datos lleguen desde varias aplicaciones distintas.

`RAISE(IGNORE)` y `RAISE(ABORT, ...)` son las dos formas de frenar una operación desde un disparador: ignorarla en silencio o rechazarla con un error. En una aplicación real se prefiere el error, para que el usuario sepa por qué no se ha hecho la retirada.

`INSERT ... SELECT`, `UPDATE` y `DELETE` con subconsultas, y el `UPSERT`, operan sobre conjuntos de filas en una sola sentencia: es mucho más rápido y seguro que traer los datos a un programa y modificarlos fila a fila.

El traspaso son cuatro sentencias que solo tienen sentido juntas. La transacción garantiza la atomicidad (todo o nada): sin ella, un fallo después del primer `UPDATE` haría desaparecer 150 €.