

# Ejercicios de SQL resueltos

Los 29 ejercicios de SQL de la web en una sola hoja: 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

### Bases de Datos

- |                                                                                      |                                                                            |
|--------------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| <a href="#"><u>1. Encuentra el fallo: los clientes sin email</u></a>                 | <a href="#"><u>2. Encuentra el fallo: las prioridades de AND y OR</u></a>  |
| <a href="#"><u>3. Encuentra el fallo: la media que sale entera</u></a>               | <a href="#"><u>4. Encuentra el fallo: el pedido fantasma</u></a>           |
| <a href="#"><u>5. Encuentra el fallo: el LEFT JOIN que pierde clientes</u></a>       | <a href="#"><u>6. Encuentra el fallo: las bajas que lo borran todo</u></a> |
| <a href="#"><u>7. Encuentra el fallo: la actualización que no actualiza nada</u></a> | <a href="#"><u>8. Encuentra el fallo: el stock que se vuelve NULL</u></a>  |
| <a href="#"><u>9. Encuentra el fallo: el disparador que descuenta a todos</u></a>    | <a href="#"><u>10. Productos baratos</u></a>                               |
| <a href="#"><u>11. Clientes sin email</u></a>                                        | <a href="#"><u>12. Búsqueda por patrón</u></a>                             |
| <a href="#"><u>13. Alta de un producto</u></a>                                       | <a href="#"><u>14. Estadísticas del catálogo</u></a>                       |
| <a href="#"><u>15. Pedidos con su cliente</u></a>                                    | <a href="#"><u>16. Importe de cada pedido</u></a>                          |
| <a href="#"><u>17. Clientes que nunca han comprado</u></a>                           | <a href="#"><u>18. Ciudades de clientes y proveedores</u></a>              |
| <a href="#"><u>19. Subida de precios</u></a>                                         | <a href="#"><u>20. Crear una vista</u></a>                                 |
| <a href="#"><u>21. Productos más caros que la media de su categoría</u></a>          | <a href="#"><u>22. Mejor cliente por gasto</u></a>                         |
| <a href="#"><u>23. Categorías con más de 100 € vendidos</u></a>                      | <a href="#"><u>24. Borrado con subconsulta</u></a>                         |
| <a href="#"><u>25. Trigger de auditoría</u></a>                                      |                                                                            |
- 
- |                                                                                     |                                                                                |
|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| <b>Ejercicios largos</b>                                                            |                                                                                |
| <a href="#"><u>26. La base de datos de una academia de idiomas</u></a>              | <a href="#"><u>27. Clínica: citas, especialidades y facturación</u></a>        |
| <a href="#"><u>28. Liga de baloncesto: clasificación y funciones de ventana</u></a> | <a href="#"><u>29. Banco: vistas, disparadores, UPSERT y transacciones</u></a> |

**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);
```

# Bases de Datos

---

## 1. Encuentra el fallo: los clientes sin email

Fácil · Consultas SQL Avanzadas · [apuntesdam.com/subject/bases-datos/topic/consultas](https://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;
```

## 2. Encuentra el fallo: las prioridades de AND y OR

Medio · Consultas SQL Avanzadas · [apuntesdam.com/subject/bases-datos/topic/consultas](https://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;
```

## 3. Encuentra el fallo: la media que sale entera

Medio · Consultas SQL Avanzadas · [apuntesdam.com/subject/bases-datos/topic/consultas](https://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;
```

## 4. Encuentra el fallo: el pedido fantasma

Medio · Consultas SQL Avanzadas · [apuntesdam.com/subject/bases-datos/topic/consultas](https://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;
```

## 5. Encuentra el fallo: el LEFT JOIN que pierde clientes

Difícil · Consultas SQL Avanzadas · [apuntesdam.com/subject/bases-datos/topic/consultas](https://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;
```

## 6. Encuentra el fallo: las bajas que lo borran todo

Muy difícil · Consultas SQL Avanzadas · [apuntesdam.com/subject/bases-datos/topic/consultas](https://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;
```

## 7. Encuentra el fallo: la actualización que no actualiza nada

Fácil · Modificación de Datos en SQL · [apuntesdam.com/subject/bases-datos/topic/modificacion](https://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';
```

## 8. Encuentra el fallo: el stock que se vuelve NULL

Difícil · Modificación de Datos en SQL · [apuntesdam.com/subject/bases-datos/topic/modificacion](https://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
);
```

## 9. Encuentra el fallo: el disparador que descuenta a todos

Difícil · Programación en Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/programacion-bdd](https://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);
```

## 10. Productos baratos

Fácil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 11. Clientes sin email

Fácil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 12. Búsqueda por patrón

Fácil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 13. Alta de un producto

Fácil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 14. Estadísticas del catálogo

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 15. Pedidos con su cliente

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 16. Importe de cada pedido

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 17. Clientes que nunca han comprado

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 18. Ciudades de clientes y proveedores

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 19. Subida de precios

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 20. Crear una vista

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 21. Productos más caros que la media de su categoría

Difícil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 22. Mejor cliente por gasto

Difícil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 23. Categorías con más de 100 € vendidos

Difícil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 24. Borrado con subconsulta

Difícil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 25. Trigger de auditoría

Difícil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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

---

### 26. La base de datos de una academia de idiomas

Difícil · 75 minutos · [apuntesdam.com/ejercicios/sql/academia-de-idiomias](https://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 cuele.

#### 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).

## 27. Clínica: citas, especialidades y facturación

Difícil · 80 minutos · [apuntesdam.com/ejercicios/sql/clinica-citas-y-facturacion](https://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).

## 28. Liga de baloncesto: clasificación y funciones de ventana

Muy difícil · 90 minutos · [apuntesdam.com/ejercicios/sql/liga-de-baloncesto-funciones-de-ventana](https://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.

## 29. Banco: vistas, disparadores, UPSERT y transacciones

Muy difícil · 100 minutos · [apuntesdam.com/ejercicios/sql/banco-vistas-disparadores-y-transacciones](https://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. Encuentra el fallo: los clientes sin email

Fácil · Consultas SQL Avanzadas · [apuntesdam.com/subject/bases-datos/topic/consultas](https://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).

## 2. Encuentra el fallo: las prioridades de AND y OR

Medio · Consultas SQL Avanzadas · [apuntesdam.com/subject/bases-datos/topic/consultas](https://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').

## 3. Encuentra el fallo: la media que sale entera

Medio · Consultas SQL Avanzadas · [apuntesdam.com/subject/bases-datos/topic/consultas](https://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).

## 4. Encuentra el fallo: el pedido fantasma

Medio · Consultas SQL Avanzadas · [apuntesdam.com/subject/bases-datos/topic/consultas](https://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).

## 5. Encuentra el fallo: el LEFT JOIN que pierde clientes

Difícil · Consultas SQL Avanzadas · [apuntesdam.com/subject/bases-datos/topic/consultas](https://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'.

## 6. Encuentra el fallo: las bajas que lo borran todo

Muy difícil · Consultas SQL Avanzadas · [apuntesdam.com/subject/bases-datos/topic/consultas](https://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).

## 7. Encuentra el fallo: la actualización que no actualiza nada

Fácil · Modificación de Datos en SQL · [apuntesdam.com/subject/bases-datos/topic/modificacion](https://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'.

## 8. Encuentra el fallo: el stock que se vuelve NULL

Difícil · Modificación de Datos en SQL · [apuntesdam.com/subject/bases-datos/topic/modificacion](https://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).

## 9. Encuentra el fallo: el disparador que descuenta a todos

Difícil · Programación en Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/programacion-bdd](https://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.

## 10. Productos baratos

Fácil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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).

## 11. Clientes sin email

Fácil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 12. Búsqueda por patrón

Fácil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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).

## 13. Alta de un producto

Fácil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 14. Estadísticas del catálogo

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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).

## 15. Pedidos con su cliente

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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 ...

## 16. Importe de cada pedido

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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).

## 17. Clientes que nunca han comprado

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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).

## 18. Ciudades de clientes y proveedores

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 19. Subida de precios

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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).

## 20. Crear una vista

Medio · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 21. Productos más caros que la media de su categoría

Difícil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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).

## 22. Mejor cliente por gasto

Difícil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 23. Categorías con más de 100 € vendidos

Difícil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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).

## 24. Borrado con subconsulta

Difícil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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').

## 25. Trigger de auditoría

Difícil · Ejercicios de Bases de Datos · [apuntesdam.com/subject/bases-datos/topic/ejercicios-bases-datos](https://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.

## 26. La base de datos de una academia de idiomas

Difícil · 75 minutos · [apuntesdam.com/ejercicios/sql/academia-de-idiomas](https://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.

## 27. Clínica: citas, especialidades y facturación

Difícil · 80 minutos · [apuntesdam.com/ejercicios/sql/clinica-citas-y-facturacion](https://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.

## 28. Liga de baloncesto: clasificación y funciones de ventana

Muy difícil · 90 minutos · [apuntesdam.com/ejercicios/sql/liga-de-baloncesto-funciones-de-ventana](https://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 «desdobra» 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.

## 29. Banco: vistas, disparadores, UPSERT y transacciones

Muy difícil · 100 minutos · [apuntesdam.com/ejercicios/sql/banco-vistas-disparadores-y-transacciones](https://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 €.