Manual completo de SQL

Bases de datos de 6 a 18 años · Sin necesidad de plataforma

Versión: manual autónomo · 3 módulos por edad
Para: alumnos, padres, ludotecas, centros educativos y tutores

Cómo usar este manual

SQL es el idioma para **preguntar** y **organizar** datos en tablas: listas de alumnos, inventarios, puntuaciones… Este manual funciona **sin plataforma online**.

Analogía para niños

Imagina una **hoja de cálculo gigante** guardada en el ordenador:

- Cada **tabla** es una hoja (por ejemplo «Alumnos»).
- Cada **fila** es un registro (un alumno).
- Cada **columna** es un dato (nombre, edad…).

SQL sirve para decir: «Muéstrame solo los alumnos de 10 años, ordenados por nombre».

Estructura

Módulo	Edades	Duración orientativa	Contenido
A	6–12	8–10 sesiones × 30–45 min	`SELECT`, `WHERE`, `ORDER BY`
B	12–14	10–12 sesiones × 45–60 min	`INSERT`, `UPDATE`, `DELETE`, agregaciones
C	14–18	12–14 sesiones × 60 min	`JOIN`, subconsultas, diseño de tablas

Para el adulto: No hace falta ser experto en bases de datos. Instala **SQLite** una vez, ejecuta el script de datos de cada módulo y deja que el alumno escriba las consultas. Si algo falla, lee el mensaje de error en voz alta: suele indicar un nombre mal escrito o una coma olvidada.

Qué necesitas

1. Un ordenador (Windows, Mac o Linux).
2. **SQLite** (motor de base de datos ligero, sin servidor).
3. Una de estas formas de trabajar:
   - **DB Browser for SQLite** (recomendado para ludotecas): sqlitebrowser.org
   - **Terminal:** comando `sqlite3` (viene en Mac/Linux; en Windows descarga desde sqlite.org)
4. Carpeta de proyecto, por ejemplo `mi-sql/`.

Configuración inicial (una sola vez)

Opción A — DB Browser (visual):

1. Instala DB Browser for SQLite.

2. Archivo → Nuevo proyecto de base de datos → guarda `academia.db`.
3. Pestaña **Ejecutar SQL** → pega y ejecuta el script de datos del módulo.

Opción B — Terminal:

```
cd mi-sql
sqlite3 academia.db
```

Dentro de SQLite verás el prompt `sqlite>`. Pega el script de datos y pulsa Enter.

Para salir: `.quit`

> **Sin instalar:** sqliteonline.com permite crear tablas y ejecutar SQL en el navegador. Copia los scripts de este manual allí si no puedes instalar nada.

Script de datos — Módulo A

Ejecuta esto **una vez** al empezar el módulo A:

```
DROP TABLE IF EXISTS alumnos;

CREATE TABLE alumnos (
  id INTEGER PRIMARY KEY,
  nombre TEXT NOT NULL,
  edad INTEGER NOT NULL,
  curso TEXT NOT NULL
);

INSERT INTO alumnos (nombre, edad, curso) VALUES
  ('Luna', 10, 'Python'),
  ('Alex', 12, 'JavaScript'),
  ('María', 10, 'HTML'),
  ('Luis', 11, 'Python'),
  ('Sofía', 12, 'SQL'),
  ('Diego', 10, 'JavaScript');
```

Comprueba que funciona:

```
SELECT * FROM alumnos;
```

Deberías ver 6 filas.

Reglas de oro

1. **Un concepto por sesión.**
2. SQL **no distingue** mayúsculas en nombres de tablas, pero escribe los comandos en MAYÚSCULAS por costumbre (`SELECT` , `FROM` ...).
3. Cada consulta termina con `;`
4. Las soluciones están al final de cada módulo.

Módulo A · 6–12 años

Módulo A · Leer datos

Objetivos

- Entender qué es una tabla, fila y columna.
- Usar `SELECT` y `FROM` .
- Filtrar con `WHERE` .
- Ordenar con `ORDER BY` .

Lección A1 · ¿Qué es una base de datos?

Duración: 25 min

Una base de datos guarda información de forma **ordenada**. La tabla `alumnos` tiene columnas `nombre` , `edad` y `curso` .

Para **ver** datos usamos `SELECT` (elegir columnas) y `FROM` (de qué tabla):

```
SELECT nombre FROM alumnos;
```

Esto devuelve solo la columna `nombre` de todas las filas.

Para ver **todo**:

```
SELECT * FROM alumnos;
```

El asterisco `*` significa «todas las columnas».

Práctica A1: Muestra solo las columnas `nombre` y `edad`. Luego muestra solo la columna `curso`.

Lección A2 · Elegir columnas

Duración: 30 min

Puedes elegir varias columnas separadas por comas:

```
SELECT nombre, edad FROM alumnos;
```

El orden de las columnas en el resultado es el que escribas.

```
SELECT curso, nombre FROM alumnos;
```

Práctica A2: Consulta que muestre `nombre`, `curso` y `edad` en ese orden.

Lección A3 · Filtrar con WHERE

Duración: 35 min

`WHERE` filtra **filas** que cumplan una condición:

```
SELECT nombre, edad FROM alumnos WHERE edad = 10;
```

Operadores útiles:

Operador	Significado	Ejemplo
`=`	Igual	`edad = 10`
`>`	Mayor	`edad > 10`
`<`	Menor	`edad < 12`
`⩾`	Mayor o igual	`edad ⩾ 11`
`≠`	Distinto	`curso ≠ 'Python'`

Texto entre comillas simples: `'Python'`

```sql
SELECT nombre FROM alumnos WHERE curso = 'Python';
```

Práctica A3: Alumnos mayores de 10 años. Luego alumnos del curso «JavaScript».

Lección A4 · Ordenar con ORDER BY

Duración: 35 min

```sql
SELECT nombre, edad FROM alumnos ORDER BY nombre;
```

Orden alfabético ascendente (A→Z) por defecto.

```sql
SELECT nombre, edad FROM alumnos ORDER BY edad DESC;
```

`DESC` = descendente (de mayor a menor).

Combinar filtro y orden:

```sql
SELECT nombre, edad FROM alumnos
WHERE edad ⩾ 11
ORDER BY edad;
```

Práctica A4: Lista de alumnos del curso «Python», ordenados por edad de menor a mayor.

Lección A5 · LIMIT — los primeros resultados

Duración: 30 min

Para ver solo los **primeros** N resultados:

```sql
SELECT nombre, edad FROM alumnos ORDER BY edad DESC LIMIT 3;
```

Los 3 alumnos de mayor edad.

```sql
SELECT nombre FROM alumnos WHERE curso = 'Python' LIMIT 2;
```

Práctica A5: Los dos alumnos más jóvenes (menor edad). Pista: `ORDER BY edad` sin `DESC`.

Proyecto A · Buscador de compañeros

Escribe **cuatro consultas** sobre la tabla `alumnos`:

1. Todos los nombres ordenados alfabéticamente.
2. Alumnos de exactamente 12 años.
3. Alumnos que **no** están en el curso «SQL».
4. Los 3 alumnos con mayor edad (nombre y edad).

Soluciones módulo A

Práctica A3:

```sql
SELECT nombre FROM alumnos WHERE edad > 10;
SELECT nombre FROM alumnos WHERE curso = 'JavaScript';
```

Proyecto A (ejemplo):

```sql
SELECT nombre FROM alumnos ORDER BY nombre;

SELECT nombre FROM alumnos WHERE edad = 12;

SELECT nombre FROM alumnos WHERE curso ≠ 'SQL';

SELECT nombre, edad FROM alumnos ORDER BY edad DESC LIMIT 3;
```

Módulo B · 12–14 años

Módulo B · Modificar y resumir

Objetivos

- Insertar, actualizar y borrar filas.
- Contar y sumar con `COUNT`, `SUM`, `AVG`.
- Agrupar con `GROUP BY` y filtrar grupos con `HAVING`.

Script de datos — Módulo B

Ejecuta al empezar el módulo B (puedes usar la misma `academia.db`):

```
DROP TABLE IF EXISTS libros;
DROP TABLE IF EXISTS alumnos_xp;

CREATE TABLE libros (
  id INTEGER PRIMARY KEY,
  titulo TEXT NOT NULL,
  autor TEXT NOT NULL,
  paginas INTEGER NOT NULL
);

INSERT INTO libros (titulo, autor, paginas) VALUES
  ('El principito', 'Saint-Exupéry', 96),
  ('Harry Potter 1', 'Rowling', 320),
  ('Harry Potter 2', 'Rowling', 341),
  ('Cien años de soledad', 'García Márquez', 471);

CREATE TABLE alumnos_xp (
  id INTEGER PRIMARY KEY,
  nombre TEXT NOT NULL,
  equipo TEXT NOT NULL,
  xp INTEGER NOT NULL
);

INSERT INTO alumnos_xp (nombre, equipo, xp) VALUES
  ('Luna', 'Rojos', 120),
  ('Alex', 'Azules', 200),
  ('María', 'Rojos', 95),
  ('Luis', 'Azules', 180),
  ('Sofía', 'Verdes', 210);
```

Lección B1 · INSERT — añadir filas

Duración: 40 min

```sql
INSERT INTO libros (titulo, autor, paginas)
VALUES ('Don Quijote', 'Cervantes', 863);
```

Comprueba:

```sql
SELECT * FROM libros;
```

Varias filas a la vez:

```sql
INSERT INTO libros (titulo, autor, paginas) VALUES
  ('Matilda', 'Dahl', 240),
  ('Charlie', 'Dahl', 176);
```

Práctica B1: Añade un libro inventado con título, autor y páginas. Muéstralo con `SELECT`.

Lección B2 · UPDATE y DELETE

Duración: 45 min

Actualizar filas existentes:

```sql
UPDATE libros
SET paginas = 100
WHERE titulo = 'El principito';
```

⚠️ **Siempre** usa `WHERE` en `UPDATE` y `DELETE`, o cambiarás/borrarás **todas** las filas.

Borrar filas:

```sql
DELETE FROM libros WHERE titulo = 'Charlie';
```

Práctica B2: Cambia el equipo de «Luna» a «Azules» en `alumnos_xp`. Borra el libro «Matilda» si lo insertaste.

Lección B3 · COUNT, SUM y AVG

Duración: 45 min

Funciones que **resumen** datos:

```
SELECT COUNT(*) AS total_libros FROM libros;
```

```
SELECT SUM(paginas) AS paginas_totales FROM libros;
```

```
SELECT AVG(paginas) AS media_paginas FROM libros;
```

`AS` pone un **alias** al resultado (nombre de columna más legible).

```
SELECT MIN(paginas) AS minimo, MAX(paginas) AS maximo FROM libros;
```

Práctica B3: ¿Cuántos alumnos hay en `alumnos_xp`? ¿Cuál es el XP total y la media?

Lección B4 · GROUP BY

Duración: 50 min

Agrupa filas que comparten un valor:

```
SELECT autor, COUNT(*) AS num_libros
FROM libros
GROUP BY autor;
```

Páginas totales por autor:

```
SELECT autor, SUM(paginas) AS paginas_totales
FROM libros
GROUP BY autor;
```

XP total por equipo:

```sql
SELECT equipo, SUM(xp) AS xp_total
FROM alumnos_xp
GROUP BY equipo
ORDER BY xp_total DESC;
```

Práctica B4: Media de XP por equipo. ¿Qué equipo tiene más XP en total?

Lección B5 · HAVING — filtrar grupos

Duración: 40 min

`WHERE` filtra **filas** antes de agrupar.
`HAVING` filtra **grupos** después de `GROUP BY`:

```sql
SELECT autor, COUNT(*) AS num_libros
FROM libros
GROUP BY autor
HAVING COUNT(*) ⩾ 2;
```

Solo autores con 2 o más libros.

```sql
SELECT equipo, AVG(xp) AS media_xp
FROM alumnos_xp
GROUP BY equipo
HAVING AVG(xp) > 150;
```

Práctica B5: Equipos cuyo XP total supera 200.

Proyecto B · Informe de biblioteca y ranking

Escribe consultas que respondan:

1. ¿Cuántos libros hay en total?
2. Lista de libros con más de 200 páginas, ordenados por páginas.
3. Autor con más páginas escritas (suma de `paginas`).
4. Ranking de equipos por XP total (de mayor a menor).
5. Equipos con al menos 2 miembros.

Soluciones módulo B

Práctica B4:

```
SELECT equipo, AVG(xp) AS media_xp FROM alumnos_xp GROUP BY equipo;
SELECT equipo, SUM(xp) AS xp_total FROM alumnos_xp GROUP BY equipo ORDER BY xp_to
```

Proyecto B (ejemplo):

```
SELECT COUNT(*) FROM libros;

SELECT titulo, paginas FROM libros WHERE paginas > 200 ORDER BY paginas;

SELECT autor, SUM(paginas) AS total FROM libros GROUP BY autor ORDER BY total DES

SELECT equipo, SUM(xp) AS xp_total FROM alumnos_xp GROUP BY equipo ORDER BY xp_to

SELECT equipo, COUNT(*) AS miembros FROM alumnos_xp GROUP BY equipo HAVING COUNT(
```

Módulo C · 14–18 años

Módulo C · Relaciones y diseño

Objetivos

- Diseñar tablas con claves primarias y foráneas.
- Combinar tablas con `JOIN`.
- Usar subconsultas.
- Proyecto: informe completo de academia.

Script de datos — Módulo C

```sql
DROP TABLE IF EXISTS progreso;
DROP TABLE IF EXISTS matriculas;
DROP TABLE IF EXISTS cursos;
DROP TABLE IF EXISTS alumnos_c;

CREATE TABLE alumnos_c (
  id INTEGER PRIMARY KEY,
  nombre TEXT NOT NULL,
  xp INTEGER NOT NULL DEFAULT 0
);

CREATE TABLE cursos (
  id INTEGER PRIMARY KEY,
  titulo TEXT NOT NULL,
  nivel TEXT NOT NULL
);

CREATE TABLE matriculas (
  id INTEGER PRIMARY KEY,
  alumno_id INTEGER NOT NULL,
  curso_id INTEGER NOT NULL,
  nota REAL,
  FOREIGN KEY (alumno_id) REFERENCES alumnos_c(id),
  FOREIGN KEY (curso_id) REFERENCES cursos(id)
);

CREATE TABLE progreso (
  id INTEGER PRIMARY KEY,
  alumno_id INTEGER NOT NULL,
  curso_id INTEGER NOT NULL,
  lecciones_completadas INTEGER NOT NULL DEFAULT 0,
  FOREIGN KEY (alumno_id) REFERENCES alumnos_c(id),
  FOREIGN KEY (curso_id) REFERENCES cursos(id)
);

INSERT INTO alumnos_c (nombre, xp) VALUES
  ('Luna', 240), ('Alex', 310), ('María', 180), ('Luis', 95), ('Sofía', 400);

INSERT INTO cursos (titulo, nivel) VALUES
  ('Python Starter', 'Principiante'),
  ('JavaScript Web', 'Intermedio'),
  ('SQL Datos', 'Intermedio');

INSERT INTO matriculas (alumno_id, curso_id, nota) VALUES
  (1, 1, 8.5), (1, 3, 7.0),
  (2, 2, 9.0), (2, 3, 8.0),
  (3, 1, 6.5),
  (4, 2, 4.0);
```

```sql
INSERT INTO progreso (alumno_id, curso_id, lecciones_completadas) VALUES
  (1, 1, 12), (1, 3, 8),
  (2, 2, 15), (2, 3, 10),
  (3, 1, 6),
  (5, 1, 20);
```

Lección C1 · Claves primarias y foráneas

Duración: 45 min

- **Clave primaria (`PRIMARY KEY`):** identifica cada fila de forma única (`id`).
- **Clave foránea (`FOREIGN KEY`):** enlaza una tabla con otra (`alumno_id` → `alumnos_c.id`).

Así evitamos datos huérfanos: una matrícula siempre apunta a un alumno y un curso reales.

```sql
CREATE TABLE ejemplo (
  id INTEGER PRIMARY KEY,
  alumno_id INTEGER NOT NULL,
  FOREIGN KEY (alumno_id) REFERENCES alumnos_c(id)
);
```

Práctica C1: Dibuja en papel el diagrama de las 4 tablas del script y señala las claves foráneas.

Lección C2 · INNER JOIN

Duración: 55 min

`JOIN` combina filas de **dos tablas** relacionadas:

```sql
SELECT a.nombre, c.titulo, m.nota
FROM alumnos_c a
INNER JOIN matriculas m ON m.alumno_id = a.id
INNER JOIN cursos c ON c.id = m.curso_id;
```

Alias: `alumnos_c a` → usamos `a.nombre` en lugar de `alumnos_c.nombre`.

Solo aparecen filas con coincidencia en **ambas** tablas.

```sql
SELECT a.nombre, c.titulo, m.nota
FROM alumnos_c a
INNER JOIN matriculas m ON m.alumno_id = a.id
INNER JOIN cursos c ON c.id = m.curso_id
WHERE m.nota ⩾ 7
ORDER BY m.nota DESC;
```

Práctica C2: Alumnos matriculados en «Python Starter» con su nota.

Lección C3 · LEFT JOIN

Duración: 45 min

`LEFT JOIN` incluye **todas** las filas de la tabla izquierda, aunque no haya coincidencia:

```sql
SELECT a.nombre, m.nota
FROM alumnos_c a
LEFT JOIN matriculas m ON m.alumno_id = a.id;
```

Alumnos **sin** ninguna matrícula:

```sql
SELECT a.nombre
FROM alumnos_c a
LEFT JOIN matriculas m ON m.alumno_id = a.id
WHERE m.id IS NULL;
```

(`Sofía` y `Luis` pueden aparecer según los datos: Luis tiene matrícula, Sofía no.)

Práctica C3: Alumnos que tienen progreso en algún curso (usa `progreso` + `alumnos_c`).

Lección C4 · Subconsultas

Duración: 50 min

Una consulta **dentro** de otra:

```sql
SELECT nombre, xp
FROM alumnos_c
WHERE xp > (SELECT AVG(xp) FROM alumnos_c);
```

Alumnos con XP por encima de la media.

En `FROM` (tabla derivada):

```sql
SELECT curso_id, AVG(lecciones_completadas) AS media
FROM progreso
GROUP BY curso_id;
```

Práctica C4: Cursos cuya media de lecciones completadas (tabla `progreso`) es mayor que 10.

Lección C5 · CREATE TABLE y buenas prácticas

Duración: 45 min

Diseño antes de escribir SQL:

1. Lista las **entidades** (alumno, curso, nota).
2. Define **columnas** y tipos (`TEXT`, `INTEGER`, `REAL`).
3. Marca claves primarias y foráneas.
4. Inserta datos de prueba.

```sql
CREATE TABLE IF NOT EXISTS logros (
  id INTEGER PRIMARY KEY,
  alumno_id INTEGER NOT NULL,
  titulo TEXT NOT NULL,
  fecha TEXT NOT NULL,
  FOREIGN KEY (alumno_id) REFERENCES alumnos_c(id)
);
```

Nombres en singular o plural, pero **consistentes**. Documenta qué significa cada tabla.

Práctica C5: Crea tabla `logros` e inserta un logro para «Luna».

Proyecto C · Informe de academia

Con las tablas del script, escribe consultas para:

1. Lista de alumnos con cursos matriculados y nota (nombre, título curso, nota).
2. Media de lecciones completadas **por curso** (título del curso y media).
3. Top 3 alumnos por XP.
4. Alumnos matriculados en algún curso pero con nota < 5.
5. Alumnos con progreso que **no** tienen matrícula en ese mismo curso (LEFT JOIN + `IS NULL`).

Soluciones módulo C

Práctica C2:

```sql
SELECT a.nombre, m.nota
FROM alumnos_c a
INNER JOIN matriculas m ON m.alumno_id = a.id
INNER JOIN cursos c ON c.id = m.curso_id
WHERE c.titulo = 'Python Starter';
```

Proyecto C (ejemplo):

```sql
-- 1
SELECT a.nombre, c.titulo, m.nota
FROM alumnos_c a
INNER JOIN matriculas m ON m.alumno_id = a.id
INNER JOIN cursos c ON c.id = m.curso_id
ORDER BY a.nombre;

-- 2
SELECT c.titulo, AVG(p.lecciones_completadas) AS media_lecciones
FROM cursos c
INNER JOIN progreso p ON p.curso_id = c.id
GROUP BY c.titulo;

-- 3
SELECT nombre, xp FROM alumnos_c ORDER BY xp DESC LIMIT 3;

-- 4
SELECT DISTINCT a.nombre, m.nota
FROM alumnos_c a
INNER JOIN matriculas m ON m.alumno_id = a.id
WHERE m.nota < 5;

-- 5 (progreso sin matrícula en ese curso)
SELECT a.nombre, c.titulo
FROM progreso p
INNER JOIN alumnos_c a ON a.id = p.alumno_id
INNER JOIN cursos c ON c.id = p.curso_id
LEFT JOIN matriculas m ON m.alumno_id = p.alumno_id AND m.curso_id = p.curso_id
WHERE m.id IS NULL;
```

Apéndice · Errores frecuentes

Error	Causa	Solución
`no such table`	Tabla no creada	Ejecuta el script de datos del módulo
`no such column`	Nombre mal escrito	Revisa mayúsculas y guiones bajos
`syntax error`	Falta `;` o coma	Revisa la línea indicada
`UPDATE` borra todo	Sin `WHERE`	Añade siempre `WHERE`
Resultado vacío	Filtro muy estricto	Prueba sin `WHERE` primero
`FOREIGN KEY constraint failed`	Id inexistente	Inserta el padre antes (alumno/curso)

Apéndice · Glosario

Término	Significado
Tabla	Conjunto de datos en filas y columnas
Fila (registro)	Una entrada en la tabla
Columna (campo)	Un tipo de dato en cada fila
`SELECT`	Leer datos
`INSERT`	Añadir filas
`UPDATE`	Modificar filas
`DELETE`	Borrar filas
`JOIN`	Combinar tablas relacionadas
Clave primaria	Identificador único de fila
Clave foránea	Referencia a otra tabla
Agregación	Resumen (`COUNT` , `SUM` , `AVG`)

Apéndice · Chuleta rápida

```
SELECT col1, col2 FROM tabla WHERE condicion ORDER BY col1 LIMIT 10;
INSERT INTO tabla (col1, col2) VALUES (val1, val2);
UPDATE tabla SET col1 = val WHERE condicion;
DELETE FROM tabla WHERE condicion;
SELECT col, COUNT(*) FROM tabla GROUP BY col HAVING COUNT(*) > 1;
SELECT a.nom, b.titulo FROM a INNER JOIN b ON a.id = b.a_id;
```

Opcional: La plataforma Academia (`sql-starter` , Editor, SQL Builder) refuerza este manual con ejercicios interactivos.