SQLite in C#

What is SQLite?

SQLite = a self-contained, serverless database stored in a single .db file. No installation, no server process, no config. Perfect for desktop apps, local tools, mobile, and learning.

Same ADO.NET concepts you just learned — just swap SqlConnection → SqliteConnection and point to a file instead of a server.

SQL Server vs SQLite

SQL Server	SQLite
Needs a server running	Just a .db file on disk
Heavy, enterprise-grade	Lightweight, embedded
SqlConnection	SqliteConnection
Server=localhost;Database=X	Data Source=school.db
Great for production APIs	Great for local/desktop apps

Step 1 — Install the NuGet Package

bash

```
# In your project folder
dotnet add package Microsoft.Data.Sqlite
```

Or in the .csproj:

xml

```
<PackageReference Include="Microsoft.Data.Sqlite" Version="8.0.0" />
```

Step 2 — Project Structure

```
/StudentApp
 ├── Models/
 |   └── Student.cs
 ├── DAL/
 |   ├── IStudentDAO.cs
 |   └── StudentSqliteDAO.cs     ← NEW (replaces SqlDAO)
```

```
├── Services/
│   └── StudentService.cs
├── Database/
│   └── DbHelper.cs          ← manages connection + table creation
└── Program.cs
```

Step 3 — DbHelper — Connection + Table Setup

```csharp
// Database/DbHelper.cs
using Microsoft.Data.Sqlite;

public class DbHelper
{
    // Single .db file — created automatically if it doesn't exist
    private const string DbPath = "school.db";

    public static string ConnectionString => $"Data Source={DbPath}";

    // Call once at app startup — creates table if not exists
    public static void Initialize()
    {
        using var con = new SqliteConnection(ConnectionString);
        con.Open();

        string sql = @"
            CREATE TABLE IF NOT EXISTS Students (
                Id    INTEGER PRIMARY KEY AUTOINCREMENT,
                Name  TEXT    NOT NULL,
                Age   INTEGER NOT NULL,
                Dept  TEXT    NOT NULL
            )";
```

```
        using var cmd = new SqliteCommand(sql, con);
        cmd.ExecuteNonQuery();

        Console.WriteLine("✅ Database initialized — school.db ready.");
    }
}
```

CREATE TABLE IF NOT EXISTS — safe to call every startup, won't duplicate.

Step 4 — Model

```
csharp
// Models/Student.cs
public class Student
{
    public int    Id   { get; set; }
    public string Name { get; set; }
    public int    Age  { get; set; }
    public string Dept { get; set; }

    public override string ToString()
        => $"[{Id}] {Name} | Age: {Age} | Dept: {Dept}";
}
```

Step 5 — Interface

```
csharp
// DAL/IStudentDAO.cs
public interface IStudentDAO
{
    void         Insert(Student student);
    Student      GetById(int id);
    List<Student> GetAll();
```

```
    void        Update(Student student);
    void        Delete(int id);
}
```

Step 6 — StudentSqliteDAO — Full CRUD

csharp

```
// DAL/StudentSqliteDAO.cs
using System;
using System.Collections.Generic;
using Microsoft.Data.Sqlite;

public class StudentSqliteDAO : IStudentDAO
{
    private readonly string _conn = DbHelper.ConnectionString;

    // ── Helper: Map a row to a Student object ──────────
    private Student MapRow(SqliteDataReader r) => new Student
    {
        Id   = r.GetInt32(0),
        Name = r.GetString(1),
        Age  = r.GetInt32(2),
        Dept = r.GetString(3)
    };

    // ── INSERT ─────────────────────────────────────────
    public void Insert(Student s)
    {
        using var con = new SqliteConnection(_conn);
        con.Open();

        string sql = @"INSERT INTO Students (Name, Age, Dept)
```

```
                VALUES (@Name, @Age, @Dept)";

    using var cmd = new SqliteCommand(sql, con);
    cmd.Parameters.AddWithValue("@Name", s.Name);
    cmd.Parameters.AddWithValue("@Age",  s.Age);
    cmd.Parameters.AddWithValue("@Dept", s.Dept);

    cmd.ExecuteNonQuery();

    // Get the auto-generated ID
    cmd.CommandText = "SELECT last_insert_rowid()";
    s.Id = (int)(long)cmd.ExecuteScalar();

    Console.WriteLine($"✅ Inserted: {s}");
}

// ── GET BY ID ──────────────────────────────────────
public Student GetById(int id)
{
    using var con = new SqliteConnection(_conn);
    con.Open();

    using var cmd = new SqliteCommand(
        "SELECT * FROM Students WHERE Id = @Id", con);
    cmd.Parameters.AddWithValue("@Id", id);

    using var reader = cmd.ExecuteReader();

    return reader.Read() ? MapRow(reader) : null;
}
```

```
// ── GET ALL ──────────────────────────────────────────────
public List<Student> GetAll()
{
    var list = new List<Student>();

    using var con = new SqliteConnection(_conn);
    con.Open();

    using var cmd    = new SqliteCommand("SELECT * FROM Students", con);
    using var reader = cmd.ExecuteReader();

    while (reader.Read())
        list.Add(MapRow(reader));

    return list;
}

// ── GET BY DEPT (Bonus — extra filter method) ─────
public List<Student> GetByDept(string dept)
{
    var list = new List<Student>();

    using var con = new SqliteConnection(_conn);
    con.Open();

    using var cmd = new SqliteCommand(
        "SELECT * FROM Students WHERE Dept = @Dept", con);
    cmd.Parameters.AddWithValue("@Dept", dept);

    using var reader = cmd.ExecuteReader();
    while (reader.Read())
```

```
        list.Add(MapRow(reader));

    return list;
}

// ── UPDATE ──────────────────────────────────────────
public void Update(Student s)
{
    using var con = new SqliteConnection(_conn);
    con.Open();

    string sql = @"UPDATE Students
                   SET Name = @Name, Age = @Age, Dept = @Dept
                   WHERE Id = @Id";

    using var cmd = new SqliteCommand(sql, con);
    cmd.Parameters.AddWithValue("@Name", s.Name);
    cmd.Parameters.AddWithValue("@Age",  s.Age);
    cmd.Parameters.AddWithValue("@Dept", s.Dept);
    cmd.Parameters.AddWithValue("@Id",   s.Id);

    int rows = cmd.ExecuteNonQuery();
    Console.WriteLine(rows > 0
        ? $"✅ Updated: {s}"
        : $"❌ Student ID {s.Id} not found.");
}

// ── DELETE ──────────────────────────────────────────
public void Delete(int id)
{
    using var con = new SqliteConnection(_conn);
```

```
    con.Open();

    using var cmd = new SqliteCommand(
        "DELETE FROM Students WHERE Id = @Id", con);
    cmd.Parameters.AddWithValue("@Id", id);

    int rows = cmd.ExecuteNonQuery();
    Console.WriteLine(rows > 0
        ? $"🗑️ Deleted student ID {id}"
        : $"❌ Student ID {id} not found.");
}

// ── TRANSACTION: Bulk Insert ──────────────────────────
public void BulkInsert(List<Student> students)
{
    using var con = new SqliteConnection(_conn);
    con.Open();

    using var txn = con.BeginTransaction();  // Start transaction

    try
    {
        foreach (var s in students)
        {
            using var cmd = new SqliteCommand(
                "INSERT INTO Students(Name,Age,Dept) VALUES(@N,@A,@D)",
                con, txn);
            cmd.Parameters.AddWithValue("@N", s.Name);
            cmd.Parameters.AddWithValue("@A", s.Age);
            cmd.Parameters.AddWithValue("@D", s.Dept);
            cmd.ExecuteNonQuery();
```

```
        }

        txn.Commit();
        Console.WriteLine($"✅ Bulk inserted {students.Count} students.");
    }
    catch (Exception ex)
    {
        txn.Rollback();
        Console.WriteLine($"❌ Bulk insert failed. Rolled back.\n{ex.Message}");
    }
  }
}
```

Step 7 — Service Layer (unchanged from DAO topic!)

```
csharp
// Services/StudentService.cs
public class StudentService
{
  private readonly IStudentDAO _dao;

  public StudentService(IStudentDAO dao) { _dao = dao; }

  public void Enroll(string name, int age, string dept)
  {
    if (string.IsNullOrWhiteSpace(name))
      throw new ArgumentException("Name is required.");
    if (age < 16 || age > 40)
      throw new ArgumentException("Age must be 16–40.");

    _dao.Insert(new Student { Name = name, Age = age, Dept = dept });
  }
```

```
    public void ShowAll()
    {
        var all = _dao.GetAll();
        if (all.Count == 0) { Console.WriteLine("No students."); return; }
        Console.WriteLine("\n—— All Students ——");
        all.ForEach(s => Console.WriteLine(s));
    }

    public void ShowById(int id)
    {
        var s = _dao.GetById(id);
        Console.WriteLine(s != null ? s.ToString() : $"Not found: {id}");
    }

    public void ChangeDept(int id, string dept)
    {
        var s = _dao.GetById(id);
        if (s == null) return;
        s.Dept = dept;
        _dao.Update(s);
    }

    public void Remove(int id) => _dao.Delete(id);
}
```

Step 8 — Program.cs — Wire Everything

```
csharp
// Program.cs
using System.Collections.Generic;
```

```
class Program
{
    static void Main()
    {
        // 1. Init DB (creates school.db + table if not exists)
        DbHelper.Initialize();

        // 2. Wire up DAO + Service
        IStudentDAO    dao = new StudentSqliteDAO();
        StudentService svc = new StudentService(dao);

        // 3. Single inserts
        svc.Enroll("Alice",   21, "CSE");
        svc.Enroll("Bob",     22, "ECE");
        svc.Enroll("Charlie", 20, "IT");

        // 4. Show all
        svc.ShowAll();

        // 5. Update
        svc.ChangeDept(2, "MECH");

        // 6. Show one
        svc.ShowById(2);

        // 7. Delete
        svc.Remove(1);

        // 8. Bulk insert via transaction
        var batch = new List<Student>
        {
```

```
            new Student { Name = "Diana", Age = 21, Dept = "CSE" },
            new Student { Name = "Evan",  Age = 23, Dept = "IT"  }
        };
        ((StudentSqliteDAO)dao).BulkInsert(batch);

        // 9. Final list
        svc.ShowAll();
    }
}
```

Output:

✅ Database initialized — school.db ready.

✅ Inserted: [1] Alice | Age: 21 | Dept: CSE

✅ Inserted: [2] Bob | Age: 22 | Dept: ECE

✅ Inserted: [3] Charlie | Age: 20 | Dept: IT

——— All Students ———

[1] Alice | Age: 21 | Dept: CSE

[2] Bob | Age: 22 | Dept: ECE

[3] Charlie | Age: 20 | Dept: IT

✅ Updated: [2] Bob | Age: 22 | Dept: MECH

[2] Bob | Age: 22 | Dept: MECH

🗑️ Deleted student ID 1

✅ Bulk inserted 2 students.

——— All Students ———

[2] Bob | Age: 22 | Dept: MECH

[3] Charlie | Age: 20 | Dept: IT

[4] Diana | Age: 21 | Dept: CSE

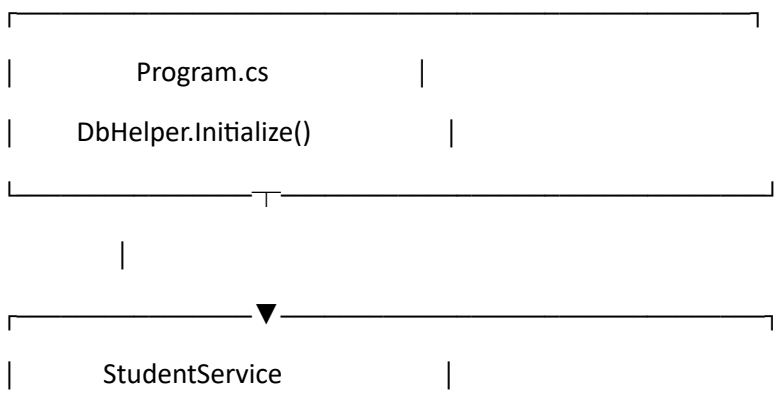
[5] Evan | Age: 23 | Dept: IT

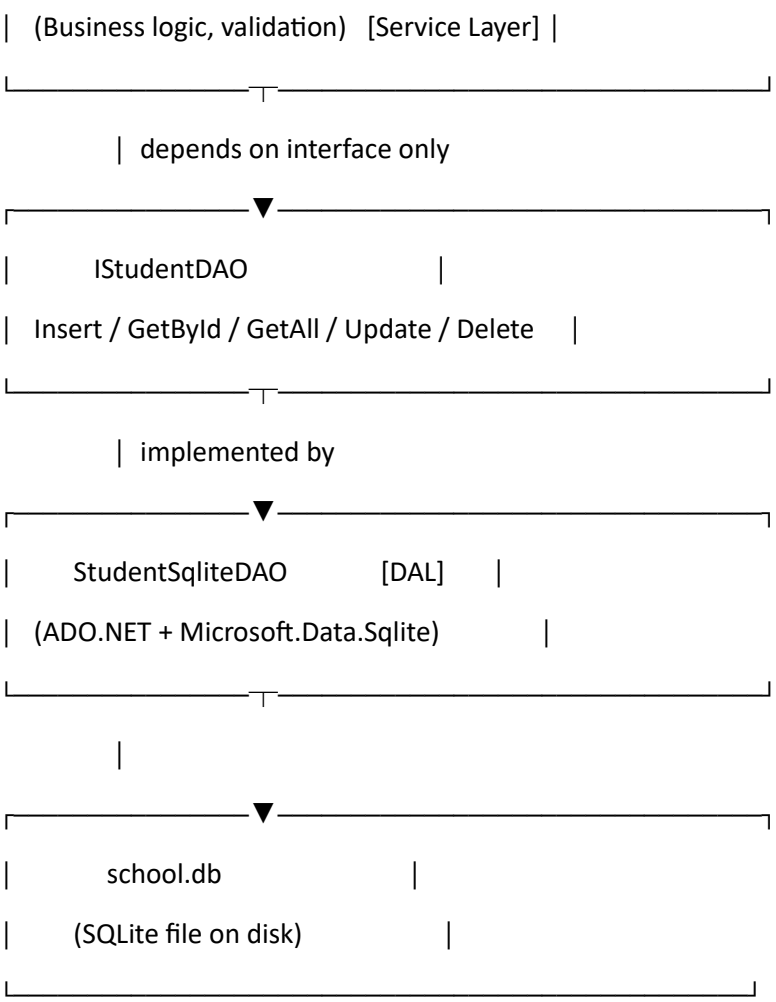
SQLite-Specific Things to Know

csharp

```
// ── 1. last_insert_rowid() → get auto ID after insert ──
cmd.CommandText = "SELECT last_insert_rowid()";
long id = (long)cmd.ExecuteScalar();   // SQLite returns long, not int
int  intId = (int)id;                  // cast safely

// ── 2. SQLite types are flexible (Type Affinity) ────────
// INTEGER → maps to long in C#
// TEXT    → maps to string
// REAL    → maps to double
// BLOB    → maps to byte[]

// ── 3. In-memory DB (great for unit testing!) ──────────
string testConn = "Data Source=:memory:";
using var con = new SqliteConnection(testConn);
con.Open();   // DB lives only as long as connection is open

// ── 4. WAL mode → better performance for concurrent reads
using var cmd = new SqliteCommand("PRAGMA journal_mode=WAL;", con);
cmd.ExecuteNonQuery();
```

Full Picture — All 4 Topics Together





MCQ Bank 🎯

Q1. What NuGet package is used for SQLite in C#?

- A) System.Data.SQLite
- B) Microsoft.Data.Sqlite ✅
- C) SQLite.NET
- D) System.SQLite.Core

Q2. What does CREATE TABLE IF NOT EXISTS do?

- A) Throws an error if table exists
- B) Drops and recreates the table
- C) Creates the table only if it doesn't already exist ✅
- D) Creates a temporary table

Q3. What does last_insert_rowid() return in SQLite?

- A) Total row count
- B) The ID of the last inserted row ✅
- C) The max ID in the table
- D) Always returns 1

Q4. What C# type does SQLite's INTEGER map to when read via ExecuteScalar()?

- A) int
- B) object
- C) long ✅
- D) double

Q5. Which connection string creates an in-memory SQLite database?

- A) "Data Source=memory"
- B) "Data Source=:memory:" ✅
- C) "Server=:memory:"
- D) "Database=RAM"

Q6. What is the main advantage of using a transaction for bulk inserts?

- A) It makes each insert faster individually
- B) It auto-creates missing tables
- C) All inserts succeed together or all are rolled back ✅
- D) It bypasses parameterized queries