

Given topic: "Creating database opening and closing with different extension".

Tool: DB Browser for SQLite

You: Expert in using tool, good guide to explain step by step, covers all the topic lies with the given topic, little theory to make beginner to understand the complex, Provide code from beginner to advance, provide all the detail about it without asking me. where I will be statisfied with all the information related to given topic given in one response.

Sure! Let's go deep into the topic "**Creating, Opening, and Closing a Database with Different Extensions in DB Browser for SQLite**".

What is DB Browser for SQLite?

DB Browser for SQLite is a high-quality, visual, open-source tool to create, design, and edit database files compatible with SQLite. SQLite itself is a lightweight, serverless database engine widely used in mobile apps, desktop software, and small-scale applications.

Understanding the Topic

♦ Goal:

1. Create a new database.
2. Use **different file extensions** (.db, .sqlite, .sqlite3, .db3, etc.)
3. Open existing databases with these extensions.

4. Close databases properly.
 5. Understand the implications of extensions and practical usage.
 6. Example SQL queries to verify data inside the created databases.
-

1. Creating a Database with DB Browser for SQLite

Steps to Create a Database

1. Launch DB Browser for SQLite
2. Click “File” → “New Database”
3. A file dialog box will open to **save your new database**:
 - Provide the **name** (e.g., `students.db`)
 - Choose the **extension**:
 - Common extensions:
 - `.db` – Standard
 - `.sqlite` – Friendly for SQLite recognition
 - `.sqlite3` – Newer version SQLite3
 - `.db3` – Variation, used in specific tools
 - `.data`, `.s3db`, `.database` – less common but supported
4. Click “**Save**”
5. The **Create Table** dialog will appear.
6. Optionally create a table or **cancel** and go to the **Execute SQL** tab.

Important Note:

- All extensions function the same in SQLite.
 - Choosing an extension is mostly **for human readability and organization**.
 - The **internal structure** of the file remains **SQLite format**, regardless of the extension.
-

2. Opening a Database File (Any Extension)

Steps to Open Existing Database:

1. Open **DB Browser for SQLite**
 2. Click **“File”** → **“Open Database”**
 3. Choose a file like:
 - `college.db`
 - `records.sqlite`
 - `employee.sqlite3`
 - `studentList.db3`
 4. You can now:
 - View tables
 - Edit data
 - Run SQL queries
 - Modify schema
-

3. Closing a Database Properly

✓ How to Close:

1. Click “**File**” → “**Close Database**”
 2. OR Use shortcut: **Ctrl + W** (**Cmd + W** for Mac)
 3. This ensures no memory leaks or locked files.
 4. Changes not saved will prompt you to “**Save Changes**”
-

4. Where Are These Databases Stored?

Wherever you save them — Desktop, a project folder, or any drive.

The file is **self-contained**, meaning:

- Entire database is in **one file**
 - Easy to move/copy/share
-

5. Theory Behind Extensions

SQLite doesn’t enforce file extensions. Internally, it reads the binary structure, not the file type.

Extension	Meaning
.db	Common for general use
.sqlite	Preferred for clarity
.sqlite3	Indicates version 3 (current standard)
.db3	Another alternative, readable
.data	Sometimes used in embedded systems

.s3db

Seen in Windows CE or mobile DBs

⚠ **Don't confuse extension with compatibility. All extensions open the same in DB Browser.**



6. SQL Code Samples to Verify and Work with Databases

Once you've created a **.db** or **.sqlite** file, try the following inside the **Execute SQL** tab:



Beginner Level SQL

-- Create table

```
CREATE TABLE users (  
  id INTEGER PRIMARY KEY,  
  name TEXT NOT NULL,  
  email TEXT UNIQUE NOT NULL  
);
```

-- Insert data

```
INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');  
INSERT INTO users (name, email) VALUES ('Bob', 'bob@example.com');
```

-- View data

```
SELECT * FROM users;
```



Intermediate Level SQL

-- Add a new column

```
ALTER TABLE users ADD COLUMN age INTEGER;
```

-- Update data

```
UPDATE users SET age = 25 WHERE name = 'Alice';
```

-- Add index for performance

```
CREATE INDEX idx_email ON users (email);
```



Advanced Level SQL

```
-- Create another table
CREATE TABLE orders (
  order_id INTEGER PRIMARY KEY,
  user_id INTEGER,
  product TEXT,
  FOREIGN KEY(user_id) REFERENCES users(id)
);
```

```
-- Join two tables
SELECT u.name, o.product
FROM users u
JOIN orders o ON u.id = o.user_id;
```



7. Practical Tips

Task	Tip
Naming Files	Use underscores: <code>student_records_2025.sqlite3</code>
Backup	Just copy the <code>.db</code> file – it's portable
Multiple Databases	You can open one at a time in DB Browser
Syntax Errors	Use <code>Execute SQL</code> tab to test commands one-by-one
Corruption Prevention	Always close properly; avoid killing the app suddenly



8. Testing with Different Extensions

To truly understand, try this:

Extension	Test
<code>test1.db</code>	Create table, insert rows
<code>test2.sqli</code>	Try JOIN queries
<code>te</code>	

`test3.sqli` Use foreign keys
`te3`

`test4.db3` Use `PRAGMA` settings (e.g., `PRAGMA foreign_keys = ON;`)

`test5.data` Observe any issues (you won't find any)

All will **function the same**, confirming that **extension doesn't affect DB logic**.

9. Bonus: Export & Import

Export to SQL File

- **File → Export → Database to SQL file**
- Useful for versioning and backup.

Import SQL File

- **File → Import → Database from SQL file**
- Restore or clone structure/data.

Conclusion

You've now learned:

- ✓ How to **create**, **open**, and **close** SQLite databases in DB Browser
- ✓ The meaning and flexibility of **different file extensions**
- ✓ Practical **SQL examples** to get hands-on
- ✓ How to **test and verify** your understanding

- ✓ Real-world **tips and workflows**
-

Let me know if you want a **video guide**, **mini project**, or **PDF notes** based on this topic!

Given topic: "Creating Tables"

- Go to "Create Table" tab
- Add fields: name, type (e.g., INTEGER, TEXT), primary key, not null
- Click OK and Write Changes".

Tool: DB Browser for SQLite

You: Expert in using tool, good guide to explain step by step on how to use the tool, covers all the topic lies with the given topic, little theory to make beginner to understand the complex, Provide code from beginner to advance, provide all the detail about it without asking me. where I will be statisfied with all the information related to given topic given in one response.

Absolutely! Here's your complete **step-by-step guide** on the topic “**Creating Tables**” using **DB Browser for SQLite**, written from an expert’s point of view while still beginner-friendly. This includes both the **UI-based method** and **SQL method**, with proper explanations and sample code from beginner to advanced level.

◆ Introduction to Creating Tables in SQLite

In SQLite, a **table** is a structure that holds data in rows and columns, much like a spreadsheet. Each column has a **name** and a **data type** (like **INTEGER**, **TEXT**, etc.). The table also may have **constraints** such as:

- **PRIMARY KEY**: Uniquely identifies a row.
 - **NOT NULL**: Prevents empty (null) values in that column.
 - **UNIQUE**: Ensures all values in a column are different.
-

Tool Used: DB Browser for SQLite

This is a GUI-based tool to visually interact with SQLite databases. It simplifies tasks like table creation, data entry, and SQL execution.

Step-by-Step Guide: Creating Tables Using GUI

Step 1: Open/Create a Database

- Launch **DB Browser for SQLite**.
 - Click **New Database** or open an existing **.sqlite** file using **Open Database**.
-

Step 2: Go to "Create Table" Tab

- Click the ****Database Structure**** tab.
 - Press **Create Table**.
-

Step 3: Add Fields (Columns)

A new dialog window will pop up.

Example Table Name: **students**

You'll be prompted to enter:

- **Table name** → **students**
- **Fields** (Add each one row-by-row):

Field Name	Data Type	PK (Primary Key)	Not Null	Unique
id	INTEGER	✓ Yes	✓ Yes	✓
name	TEXT	✗ No	✓ Yes	✗
age	INTEGER	✗ No	✗ No	✗
email	TEXT	✗ No	✓ Yes	✓

Click **Add** to insert each field.

Step 4: Click OK and Save Changes

- After entering all fields and constraints, click **OK**.
- Then press **Write Changes** (disk icon) to save the table structure into the database.

You've now created a new table successfully!

SQL Code Equivalent (Manual Method)

You can do the same thing using SQL. In the **Execute SQL** tab, paste and run this:

```
CREATE TABLE students (  
  id INTEGER PRIMARY KEY NOT NULL UNIQUE,  
  name TEXT NOT NULL,  
  age INTEGER,  
  email TEXT NOT NULL UNIQUE  
);
```

 Explanation:

- **id**: Integer primary key, not null, and unique (all IDs must be different).
- **name**: Must have a value (not null).
- **age**: Optional field.
- **email**: Must have a value and be unique.

After writing, click **Execute All** and press **Write Changes**.

Modifying an Existing Table

- Go to the **Database Structure** tab.
 - Right-click the table name → **Modify Table**.
 - Add, remove, or update fields.
 - Click **OK** and then **Write Changes**.
-

Supported Data Types in SQLite

SQLite supports these common types:

SQLite Type	Description
INTEGER	Whole numbers
REAL	Floating point numbers
TEXT	Character strings
BLOB	Binary data (e.g., images)

NUMERIC

Can store values as integer, real, or text depending on format



Common Constraints

Constraint	Description
PRIMARY KEY	Unique + Not Null by default
NOT NULL	Field must have a value
UNIQUE	No duplicate values allowed
DEFAULT	Assigns default value if none is provided
CHECK	Enforces a condition on values
FOREIGN KEY	Links to a column in another table (used for relationships)



Advanced Example with More Constraints

```
CREATE TABLE courses (  
  course_id INTEGER PRIMARY KEY,  
  course_name TEXT NOT NULL,  
  credits INTEGER NOT NULL CHECK (credits BETWEEN 1 AND 5),  
  department TEXT DEFAULT 'General',  
  UNIQUE(course_name, department)  
);
```




Notes:

- **CHECK:** Ensures `credits` is between 1 to 5.
- **DEFAULT:** Assigns `'General'` to `department` if nothing is entered.
- **UNIQUE(course_name, department):** Composite unique constraint.

Using Foreign Key Example

```
CREATE TABLE enrollments (  
  enrollment_id INTEGER PRIMARY KEY,  
  student_id INTEGER NOT NULL,  
  course_id INTEGER NOT NULL,  
  FOREIGN KEY (student_id) REFERENCES students(id),  
  FOREIGN KEY (course_id) REFERENCES courses(course_id)  
);
```

 This sets up a relational database system with:

- `students` table
- `courses` table
- `enrollments` table linking them

To enable foreign keys in DB Browser:

Go to `Execute SQL` and run:

```
PRAGMA foreign_keys = ON;
```

-

Best Practices

- Always define a **Primary Key**.
- Use **NOT NULL** where applicable to maintain data integrity.
- Use **constraints** like `UNIQUE`, `CHECK`, and `DEFAULT` for safety.
- Avoid using too many `TEXT` fields unless necessary.
- Always click `Write Changes` after editing or executing anything.

Conclusion

You now have a **complete understanding** of how to:

1. Create a table using GUI and SQL in DB Browser for SQLite.
2. Define fields with appropriate data types and constraints.
3. Understand beginner to advanced table structure concepts.

You can now build complex relational databases using DB Browser efficiently.

If you want me to generate some practice exercises or mini-project ideas using this knowledge, let me know—I'd be happy to help!

Given topic: "Table, Columns, Primary Key, Data Types (SQLite-supported types: INTEGER, REAL, TEXT, BLOB)".

Tool: DB Browser for SQLite

You: Expert in using tool, good guide to explain step by step on how to use the tool, covers all the topic lies with the given topic, little theory to make beginner to understand the complex, Provide code from beginner to advance, provide all the detail about it without asking me. where I will be statisfied with all the information related to given topic given in one response.

Absolutely! Here's a **complete, step-by-step beginner-to-advanced guide** on how to use **DB Browser for SQLite** to understand and work with:

- **Tables**
- **Columns**
- **Primary Keys**
- **Data Types** (INTEGER, REAL, TEXT, BLOB)

Let's dive in with a blend of **clear theory, practical steps using DB Browser for SQLite**, and **SQL code examples**.

1. What is DB Browser for SQLite?

DB Browser for SQLite is a free, open-source tool that allows you to **create, design, and browse SQLite databases** in a GUI format. No coding needed for simple operations, but full SQL scripting is supported.

Download: <https://sqlitebrowser.org/>

2. SQLite Database Essentials

♦ What is a Table?

A **table** in a database is like an Excel sheet:

- **Rows** = Data records (entries)
- **Columns** = Fields (attributes)

Each table has:

- A **name**
 - One or more **columns**
 - Optionally a **Primary Key**
-

3. Understanding SQLite Data Types

SQLite uses **dynamic typing**, but it supports these **affinity types**:

Data Type	Description
INTEGER	Whole numbers (e.g., age, id)
REAL	Floating point (e.g., price, GPA)
TEXT	String/text (e.g., name, address)
BLOB	Binary data (e.g., images, files)

4. Creating Tables in DB Browser for SQLite

► Step-by-Step using GUI

1. Open DB Browser for SQLite
2. Click "New Database"
 - Choose a location and name like `my_database.db`
3. Go to "Database Structure" tab → Click "Create Table"
4. In the popup:
 - **Table Name:** `students`
 - **Add columns:**
 - `id` → `INTEGER`, check **PK** (Primary Key), check **Not NULL**, check **Auto Increment**
 - `name` → `TEXT`, check **Not NULL**
 - `age` → `INTEGER`
 - `gpa` → `REAL`
 - `profile_picture` → `BLOB`
5. Click **OK**
6. Click "**Write Changes**" to save

 You've just created a table with all 4 major SQLite data types.

5. Creating Tables Using SQL Code (Beginner to Advanced)

Beginner SQL

```
CREATE TABLE students (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  name TEXT NOT NULL,  
  age INTEGER,  
  gpa REAL,  
  profile_picture BLOB  
);
```

- **PRIMARY KEY** ensures unique row id
- **AUTOINCREMENT** automatically increases **id**
- **TEXT**, **REAL**, **BLOB** used to demonstrate SQLite data types

Explanation:

- **INTEGER PRIMARY KEY**: Special in SQLite; becomes **rowid**
 - **NOT NULL**: Disallows empty values
 - **BLOB**: Store files like images (base64 or binary)
-

6. Modifying Tables Later

GUI:

- Right-click a table in **Database Structure** tab
- Choose **Modify Table**
- Add, delete, or rename columns

SQL:

```
ALTER TABLE students ADD COLUMN email TEXT;
```



7. Inserting Data into Tables

GUI:

- Go to **"Browse Data"** → Select Table → Click **"New Record"** → Enter data

SQL:

```
INSERT INTO students (name, age, gpa) VALUES ('Alice', 20, 3.8);
```

```
INSERT INTO students (name, age, gpa) VALUES ('Bob', 22, 3.2);
```

For BLOB:

```
-- Assuming you have binary image data in code (advanced use case)
```



8. Reading and Querying Data

Simple Query:

```
SELECT * FROM students;
```

Filter:

```
SELECT name, gpa FROM students WHERE age > 20;
```



9. Testing Data Types (Advanced Examples)

-- Testing INTEGER

```
INSERT INTO students (name, age, gpa) VALUES ('Charlie', 25, 3.9);
```

-- Testing REAL

```
INSERT INTO students (name, age, gpa) VALUES ('David', 23, 3.95);
```

-- Testing TEXT

```
INSERT INTO students (name, age, gpa) VALUES ('Eve', 21, 3.7);
```

-- Testing BLOB (advanced - done in application with prepared statements)

10. Understanding PRIMARY KEY

Key Points:

- Uniquely identifies each row
- You can have **only one PRIMARY KEY** per table
- Best practice: Use **INTEGER PRIMARY KEY AUTOINCREMENT**

Example with Custom Primary Key

```
CREATE TABLE employees (  
    emp_id TEXT PRIMARY KEY,  
    name TEXT NOT NULL  
);
```

11. Exporting and Importing

- Go to **File > Export > Database to SQL file**
 - Or **File > Import > Table from CSV File** for loading external data
-

12. Practical Use of BLOBs

To actually **store images/files** in BLOB:


- Use programming languages like Python or Java

- Use **prepared statements** to read binary files and insert

Summary Table

Feature	How to Use in DB Browser	SQL Code Example
Create Table	"Create Table" in GUI	<code>CREATE TABLE students (...)</code>
Add Column	Modify Table in GUI	<code>ALTER TABLE students ADD COLUMN ...</code>
Insert Data	"Browse Data" > New Record	<code>INSERT INTO students (...) VALUES (...)</code>
Query Data	"Execute SQL" tab	<code>SELECT * FROM students</code>
Primary Key	Check PK during column creation	<code>PRIMARY KEY AUTOINCREMENT</code>
Data Types	Choose from dropdown during creation	<code>INTEGER, REAL, TEXT, BLOB</code>

Pro Tip for Learning

 **Experiment actively!** Try inserting wrong types, see how SQLite accepts them. For example:

```
INSERT INTO students (name, age, gpa) VALUES ('Zoe', 'twenty-two', 'A+'); -- Will insert!
```

Why? Because SQLite uses dynamic typing (it doesn't strictly enforce types unless CHECK constraints or strict typing is applied).

Final Thoughts

With this guide:

- You can **create, modify, and explore** SQLite tables confidently
- Understand all **core data types**
- Use **DB Browser** effectively for real-world projects
- Move from **basic usage to advanced operations**

Let me know if you'd like the **Python integration with SQLite for BLOB storage**, or want to simulate **file uploads**, or create **constraints and relationships**!

Insert Data (Manually) • Go to Browse Data > Select Table • Click New Record to add rows

Sure! Here's a **complete guide** to the topic:

◆ **Insert Data (Manually) in DB Browser for SQLite**

🧠 **What is DB Browser for SQLite?**

DB Browser for SQLite is a high-quality, open-source, visual tool to create, design, and edit SQLite database files. It allows you to interact with SQLite databases without writing SQL code—perfect for both beginners and professionals.

🎯 **Objective of This Topic**

We are focusing on how to **insert data manually** using the **Graphical User Interface (GUI)** of DB Browser for SQLite—**no SQL knowledge required**.

🧩 **Key Steps: Insert Data Manually**

✓ Step-by-Step Process

♦ 1. Open or Create a Database

If you already have a database:

- Click **File > Open Database** and select your `.sqlite` or `.db` file.

If not:

- Click **File > New Database**, choose a location and name, then click **Save**.
 - Define a table with fields by clicking **Create Table**.
-

♦ 2. Navigate to the Table

Go to the **Browse Data** tab at the top.



Then:

- From the dropdown next to **Table**, choose the table where you want to add data.

Example: Suppose your table is `students`.

♦ 3. Click 'New Record' to Insert Data


- Click the **"New Record"** button at the bottom-left of the screen.
- A new empty row will appear.
 - Fill in values for each column.

Example:

- `id: 1`
 - `name: John Doe`
 - `grade: A`
- Repeat for each row you want to insert.
-

◆ 4. Save the Data

After inserting records:

- Click **"Write Changes"** ( icon or **File > Write Changes**) to save the new data to your database file.
-

Little Theory Behind the GUI Insertion

Under the hood, this GUI action executes an **SQL INSERT INTO** statement like this:

```
INSERT INTO students (id, name, grade)
VALUES (1, 'John Doe', 'A');
```

But you don't have to write this—DB Browser does it for you.

Example: Create Table + Manual Insert

Step 1: Create a Table

If you want to create a table from scratch (only for beginners):

- Go to **Database Structure > Create Table**
- Table Name: `students`
- Add Fields:
 - `id` – Integer – Primary Key
 - `name` – Text
 - `grade` – Text

Click **OK**.

Step 2: Insert Data Manually

Now follow the earlier steps:

- Go to **Browse Data > Select `students`**
- Click **New Record**
- Add:
 - `id = 1`
 - `name = Alice`
 - `grade = A`
- Click **Write Changes**

Repeat as needed.



Sample Table Data

id	name	grade
----	------	-------

1	Alice	A
2	Bob	B
3	Charlie	C

Tips for Beginners

- **Auto Increment ID:** If `id` is set to `INTEGER PRIMARY KEY`, you can leave it blank—it will auto-increment.
 - **Don't Forget to Save:** Always click **Write Changes** after editing. Unsaved changes will be lost.
 - **Empty Row?** You can click a row and hit **Delete Record** if you added one by mistake.
-

Intermediate: Use SQL for Custom Insert

You can also go to **Execute SQL** tab and write:

```
INSERT INTO students (id, name, grade)
VALUES (4, 'David', 'B');
```

This is **optional** but helpful as you grow.

Advanced Pro Tip

To import data quickly in bulk:

- Use **File > Import > Table from CSV file**
- Select a `.csv` file and map the columns.

But this is outside the “manual insert” scope.

Summary

Feature	What to Do
Open Database	File > Open Database
Select Table	Browse Data > Select Table dropdown
Add New Row	Click New Record
Enter Data	Type in each column of the new row
Save Data	Click Write Changes
Optional: Execute SQL	Use INSERT INTO in Execute SQL tab

Conclusion

This guide covered:

- Manual data insertion via GUI.
- Visual steps using Browse Data.
- What happens under the hood with SQL.
- Beginner to advanced tips.

This is all you need to **fully master manual data insertion** in DB Browser for SQLite.

Let me know if you want a cheat sheet or tutorial video link too.

Simple Queries • Go to Execute SQL `SELECT * FROM tablename;`  Concepts: • `SELECT`, `WHERE`, `ORDER BY`

Absolutely! Here's a **complete, beginner-to-advanced guide** for the topic:

Topic: Simple Queries in DB Browser for SQLite

 Tool Used: DB Browser for SQLite

Introduction: What is SQL?

SQL (Structured Query Language) is a standard language for managing and manipulating databases. With SQL, we can:

- Retrieve data
 - Insert, update, or delete records
 - Organize data using conditions
-

Concepts Covered

Concept	Description
<code>SELECT</code>	Retrieves data from the database
<code>WHERE</code>	Filters records that fulfill a specified condition

ORDER Sorts the result set in ascending or descending
BY order

Step-by-Step Guide: Using DB Browser for SQLite

Step 1: Install and Open DB Browser for SQLite

1. Open the software.
 2. Go to **"File > Open Database"** and choose your `.sqlite` or `.db` file.
 3. Once open, go to the **"Execute SQL"** tab.
-

Sample Table

For demonstration, consider a table named `students` with the following structure:

id	name	age	grade
1	Alice	20	A
2	Bob	21	B
3	Charlie	20	C
4	Diana	22	A

1. **SELECT**: Fetch All Data

`SELECT * FROM students;`

- `*` means **all columns**.
- This will return every row and every column from the `students` table.

✓ 2. **SELECT**: Fetch Specific Columns

SELECT name, grade FROM students;

- Returns only the **name** and **grade** columns.
-

✓ 3. **WHERE**: Filter Records

🧠 Syntax:

SELECT column1, column2 FROM table_name WHERE condition;

👉 Examples:

Filter by Age:

SELECT * FROM students WHERE age = 20;

Filter by Name:

SELECT * FROM students WHERE name = 'Alice';

Filter by Grade:

SELECT * FROM students WHERE grade = 'A';

✓ 4. Using Logical Operators with WHERE

a. AND

SELECT * FROM students WHERE age = 20 AND grade = 'A';

- Both conditions must be true.

b. OR

```
SELECT * FROM students WHERE grade = 'A' OR grade = 'B';
```

- Either condition can be true.
-

✓ 5. ORDER BY: Sort Results

🧠 Syntax:

```
SELECT * FROM table_name ORDER BY column_name ASC|DESC;
```

👉 Examples:

Sort by Name (Ascending):

```
SELECT * FROM students ORDER BY name ASC;
```

Sort by Age (Descending):

```
SELECT * FROM students ORDER BY age DESC;
```

✓ 6. Combined Usage: SELECT + WHERE + ORDER BY

💡 Example 1: Students aged 20, ordered by name

```
SELECT * FROM students  
WHERE age = 20  
ORDER BY name ASC;
```

💡 Example 2: Students with grade A or B, sorted by age descending

```
SELECT * FROM students  
WHERE grade = 'A' OR grade = 'B'  
ORDER BY age DESC;
```

✓ 7. Advanced Filters Using **IN**, **NOT IN**, and **BETWEEN**

a. Using **IN**

```
SELECT * FROM students WHERE grade IN ('A', 'B');
```

b. Using **NOT IN**

```
SELECT * FROM students WHERE grade NOT IN ('C');
```

c. Using **BETWEEN**

```
SELECT * FROM students WHERE age BETWEEN 20 AND 22;
```

🧠 Tips in DB Browser for SQLite:

- After writing SQL, click **"Execute All"** or press **Ctrl + Enter**.
 - Check the **"Browse Data"** tab to view the table before writing queries.
 - You can copy/paste the output or export it as CSV.
-

🧪 Extra Practice Queries

Query Description	Query
Get students with grade 'B'	<pre>SELECT * FROM students WHERE grade = 'B';</pre>
Get students older than 20	<pre>SELECT * FROM students WHERE age > 20;</pre>
Get students younger than 22 and in grade 'A'	<pre>SELECT * FROM students WHERE age < 22 AND grade = 'A';</pre>

Order students by grade then name

```
SELECT * FROM students ORDER BY  
grade, name;
```

Get names of students who are 20 or
22

```
SELECT name FROM students WHERE age  
IN (20, 22);
```

Conclusion

You now understand how to:

1. Use **SELECT** to retrieve data.
2. Filter using **WHERE**.
3. Sort using **ORDER BY**.
4. Combine them for powerful queries.

These are **core SQL fundamentals**, and mastering them is crucial for working with any database!

If you want, I can provide **sample .db file**, create **ER diagrams**, or give you a **query challenge sheet** next! Just let me know.