Database Final Project: SQLite

Josh Regalla

Zachary Brown

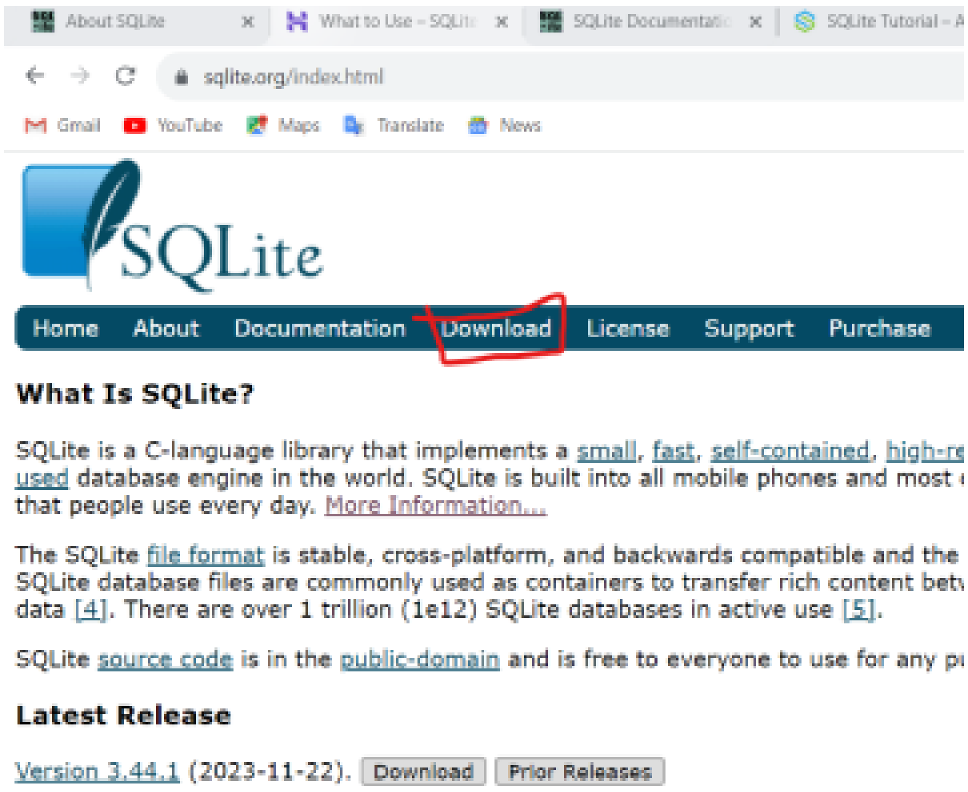
Chaminade University

Dr. Chong

December 1, 2023

How to Download and Open SQLite: 5 Easy Steps

Step 1: Go to www.sqlite.org and click the download tab boxed in red.



What Is SQLite?

SQLite is a C-language library that implements a small, fast, self-contained, high-re
used database engine in the world. SQLite is built into all mobile phones and most
that people use every day. More Information...

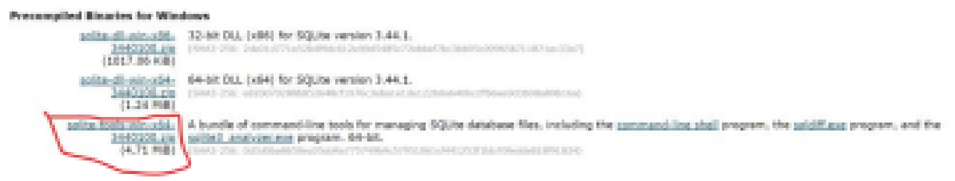
The SQLite file format is stable, cross-platform, and backwards compatible and the
SQLite database files are commonly used as containers to transfer rich content betw
data [4]. There are over 1 trillion (1e12) SQLite databases in active use [5].

SQLite source code is in the public-domain and is free to everyone to use for any pu

Latest Release

Version 3.44.1 (2023-11-22). Download Prior Releases

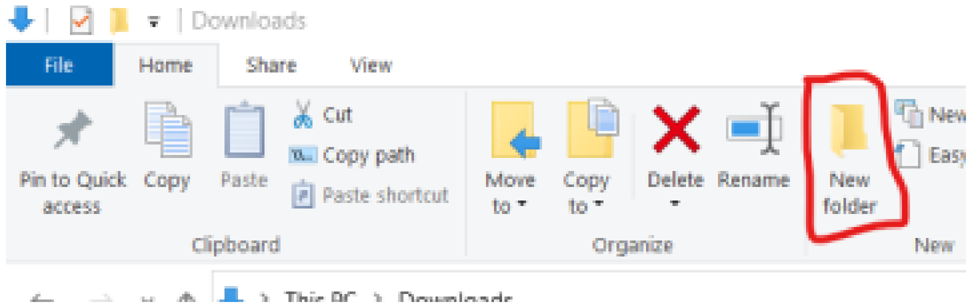
Step 2: Scroll down until you see Precompiled Binaries for Windows. Double click the download link boxed in red.

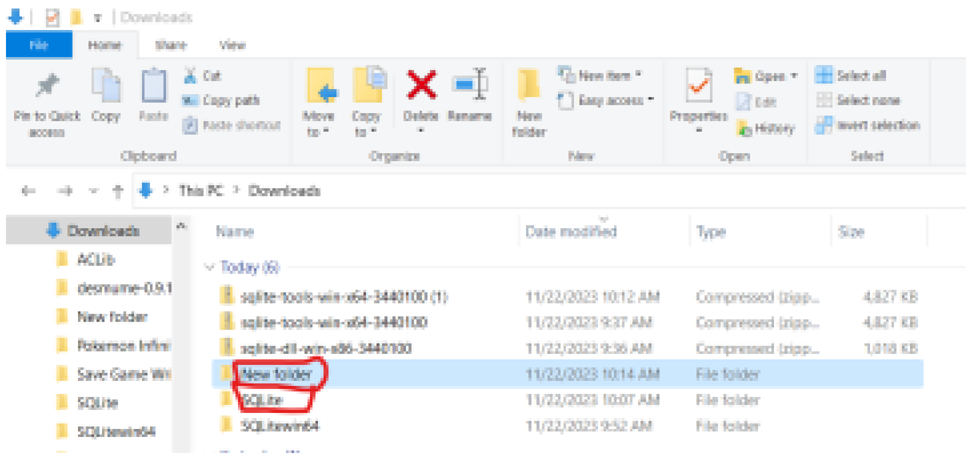


Precompiled Binaries for Windows

sqlite-dll-win-x86-3440200.zip (1017.86 KiB)	32-bit DLL (x86) for SQLite version 3.44.1. [illegible]
sqlite-dll-win-x64-3440200.zip (1.24 MiB)	64-bit DLL (x64) for SQLite version 3.44.1. [illegible]
sqlite-tools-win-x64-3440200.zip (4.71 MiB)	A bundle of command-line tools for managing SQLite database files, including the command-line shell program, the sqldiff.exe program, and the sqlite3_analyzer.exe program. 64-bit. [illegible]

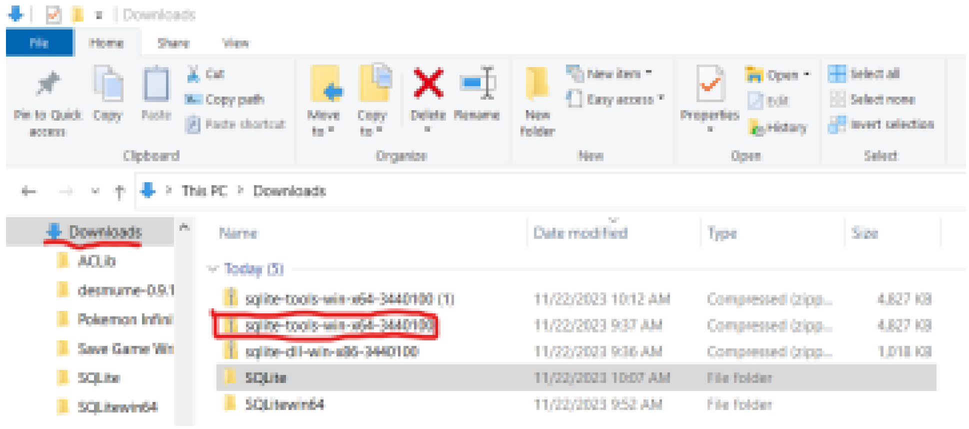
Step 3: Go to your files and create a new folder and name it SQLite.



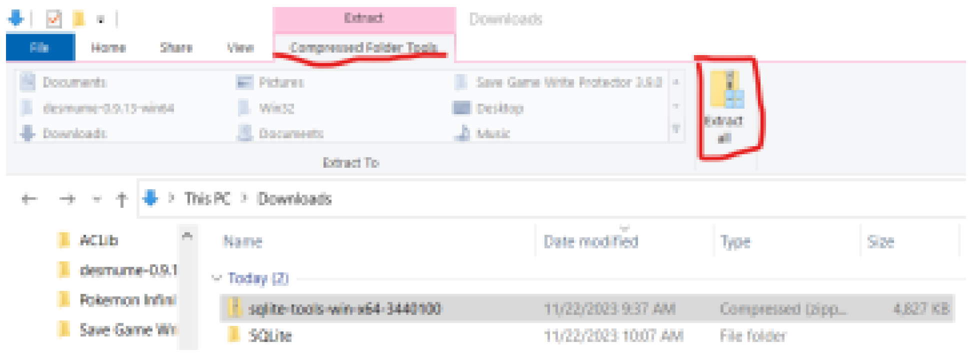


Name	Date modified	Type	Size
Today (6)			
sqlite-tools-win-x64-3440100 (1)	11/22/2023 10:12 AM	Compressed (zipp...	4,827 KB
sqlite-tools-win-x64-3440100	11/22/2023 9:37 AM	Compressed (zipp...	4,827 KB
sqlite-dll-win-x86-3440100	11/22/2023 9:36 AM	Compressed (zipp...	1,018 KB
New folder	11/22/2023 10:14 AM	File folder	
SQLite	11/22/2023 10:07 AM	File folder	
SQLitewin64	11/22/2023 9:52 AM	File folder	

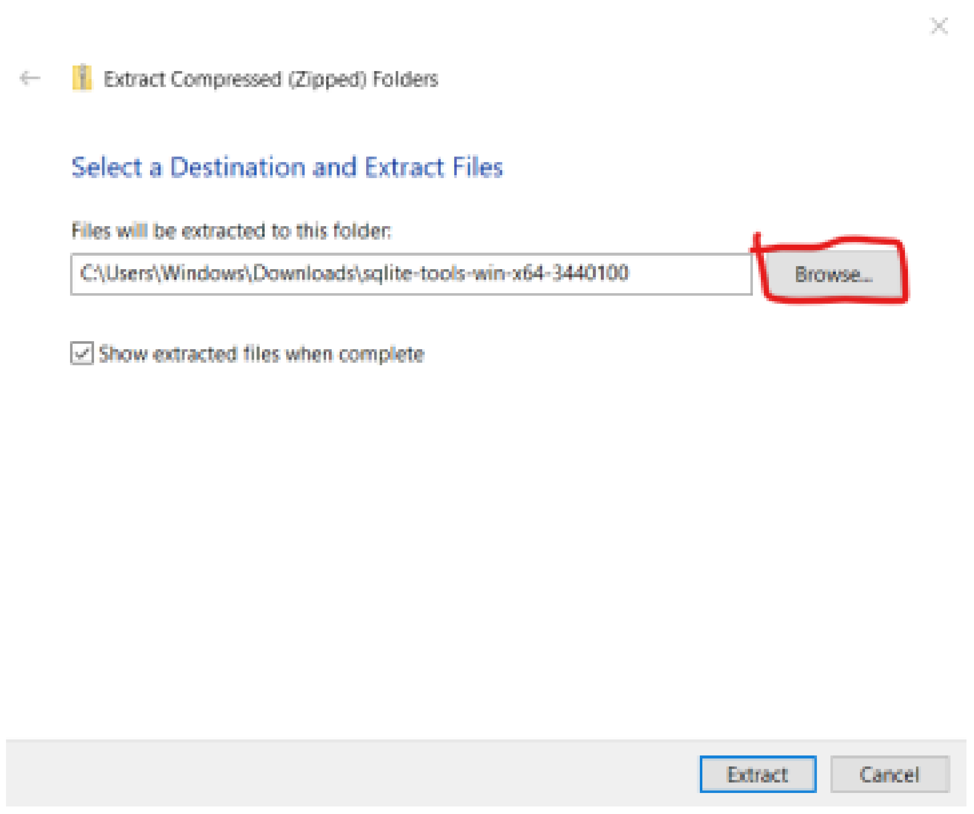
Step 4: Extract the zip file into the SQLite folder you just created.



Name	Date modified	Type	Size
Today (5)			
sqlite-tools-win-x64-3440100 (1)	11/22/2023 10:12 AM	Compressed (zipp...	4,827 KB
sqlite-tools-win-x64-3440100	11/22/2023 9:37 AM	Compressed (zipp...	4,827 KB
sqlite-dll-win-x86-3440100	11/22/2023 9:36 AM	Compressed (zipp...	1,018 KB
SQLite	11/22/2023 10:07 AM	File folder	
SQLitewin64	11/22/2023 9:52 AM	File folder	

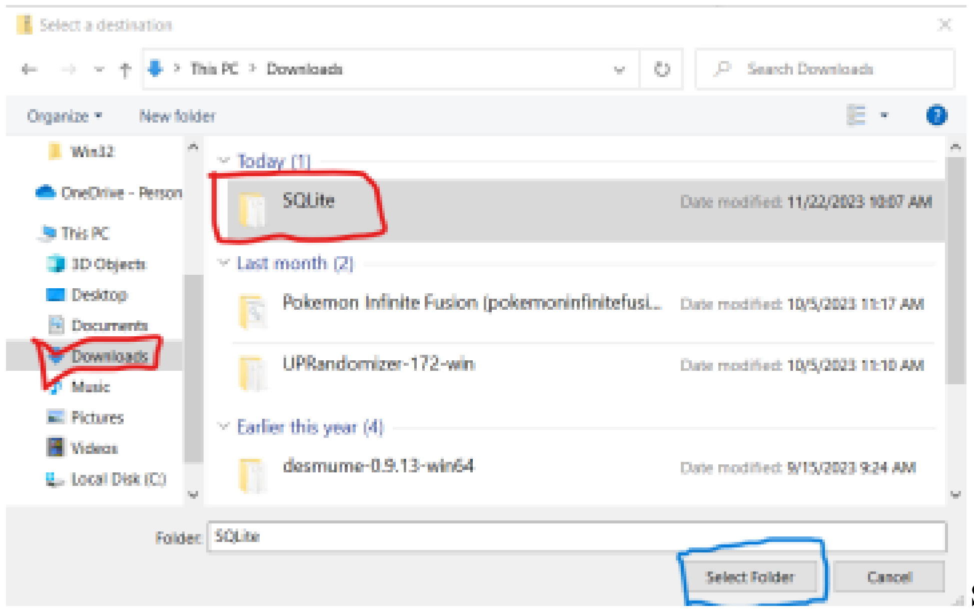


Name	Date modified	Type	Size
Today (2)			
sqlite-tools-win-x64-3440100	11/22/2023 9:37 AM	Compressed (zipp...	4,827 KB
SQLite	11/22/2023 10:07 AM	File folder	


Extract Compressed (Zipped) Folders
Select a Destination and Extract Files
Browse...
Extract
Cancel

Files will be extracted to this folder:
C:\Users\Windows\Downloads\sqlite-tools-win-x64-3440100

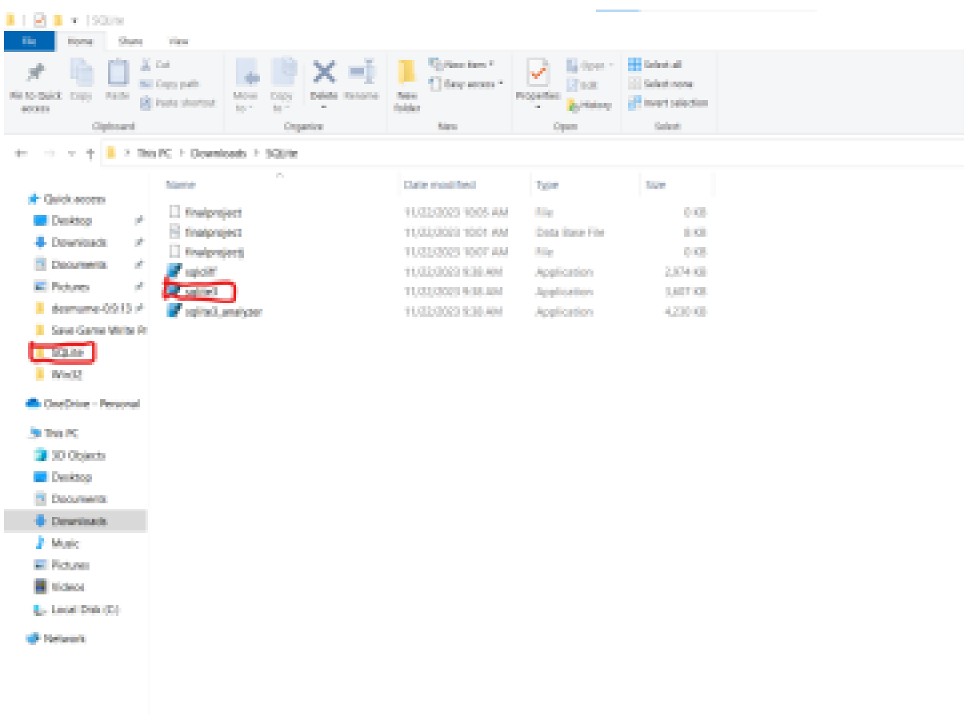
Show extracted files when complete



Today (1)	
SQLite	Date modified: 11/22/2023 10:07 AM
Last month (2)	
Pokemon Infinite Fusion (pokemoninfinitefusi...	Date modified: 10/5/2023 11:17 AM
UPRandomizer-172-win	Date modified: 10/5/2023 11:10 AM
Earlier this year (4)	
desmume-0.9.13-win64	Date modified: 9/15/2023 9:24 AM

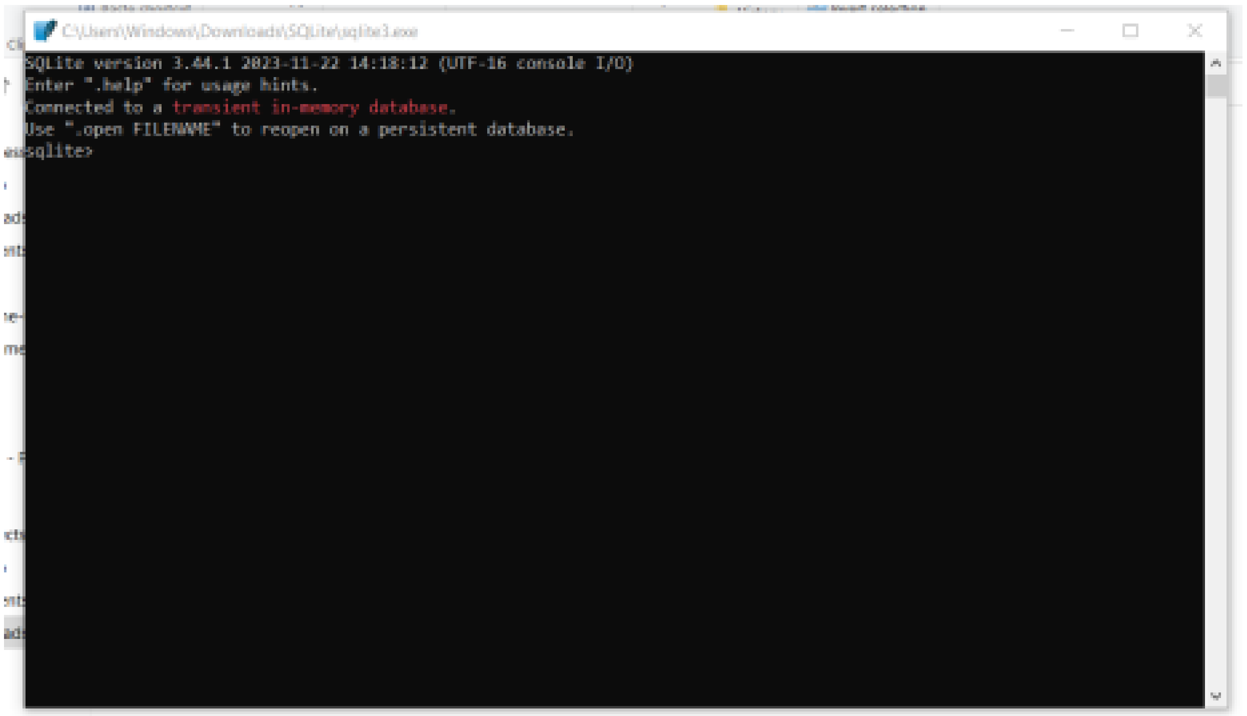
Folder: SQLite

Step 5: Open the SQLite folder and then double click the application named sqlite3 boxed in red.



This PC > Downloads > SQLite

Name	Date modified	Type	Size
finalproject	11/22/2023 10:05 AM	File	0 KB
finalproject	11/22/2023 10:01 AM	Data Base File	8 KB
finalprojectj	11/22/2023 10:07 AM	File	0 KB
sqldiff	11/22/2023 9:38 AM	Application	2,874 KB
sqlite3	11/22/2023 9:38 AM	Application	3,607 KB
sqlite3_analyzer	11/22/2023 9:38 AM	Application	4,230 KB



C:\Users\Windows\Downloads\SQLite\sqlite3.exe

```
SQLite version 3.44.1 2023-11-22 14:18:12 (UTF-16 console I/O)
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

You are now ready to code in SQLite.

1. Description: SQLite Overview

SQLite is a lightweight, open-source, serverless relational database management system. Unlike traditional database systems, SQLite is self-contained, meaning there is no need for a separate server process and it operates directly on the user's device. Its design simplicity and minimal setup make it an ideal choice for various applications, especially in embedded systems and scenarios where resources are limited.

Key Points:

- **Lightweight Nature:** SQLite is designed to be lightweight and efficient. It is a compact library that can be embedded into applications, keeping the overall system footprint to a minimum.
- **Serverless Architecture:** Unlike client-server databases like MySQL, SQLite operates as a file-based database system. The entire database is stored in a single disk file, simplifying deployment and eliminating the need for a dedicated database server.
- **Zero Configuration:** There is no complex setup or administration required. SQLite databases are created on-the-fly, and the absence of a separate server process streamlines the deployment process.
- **Suitability for Embedded Systems:** SQLite's small size and simplicity make it well suited for embedded systems with limited resources. It's commonly used in scenarios where a full-fledged database server would be impractical or unnecessary.
- **ACID Compliant:** Despite its lightweight design, SQLite maintains ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring data integrity and reliability.
- **Cross-Platform Compatibility:** SQLite is compatible with various operating systems, making it versatile for use across different platforms, including Windows, macOS, and Linux.

2. Key Features Compared to MySQL:

a. File-Based Architecture: **SQLite** • **File-Based Storage:** SQLite databases are self-contained in a single file. The entire database, including tables, schema, and indices, is stored in a single disk file. This simplicity facilitates easy sharing and portability.	**MySQL** • **Client-Server Architecture:** MySQL follows a client-server model where the data is stored on a server, and clients connect to it. This architecture requires a separate MySQL server process running independently.
b. Serverless Design: **SQLite** • **Serverless:** SQLite operates as a serverless database. It doesn't require a dedicated server process to handle requests. Applications interact with the SQLite library directly, making it suitable for embedded systems and standalone applications.	**MySQL** • **Client-Server Model:** MySQL employs a client-server architecture where clients send requests to the server, and the server processes those requests. This design introduces additional complexity compared to SQLite's serverless approach.
c. Deployment and Setup:	**MySQL**

SQLite • **Zero Configuration:** SQLite databases are created dynamically, and there is minimal setup required. No separate server installation or configuration is needed, making it straightforward for development and deployment.	• **Installation and Configuration:** MySQL requires a separate server installation and configuration. Setting up and managing a MySQL server involves additional steps, including user authentication, database creation, and server tuning.
d. Resource Utilization: **SQLite** • **Low Resource Footprint:** SQLite is designed to be lightweight, making it suitable for scenarios with limited resources. It is often used in embedded systems, mobile applications, and scenarios where memory and processing power are constrained.	**MySQL** • **Resource Intensive:** MySQL, being a traditional client-server database, may require more resources, especially in scenarios with high concurrent connections or large datasets.

e. Use Cases **SQLite**	**MySQL**
• **Embedded Systems:** Well-suited for embedded systems, mobile applications, and scenarios where a lightweight, serverless database is sufficient. • **Single-User Applications:** Ideal for single-user applications or small-scale projects where simplicity is a priority.	• **Enterprise Applications:** Suited for large-scale enterprise applications with high concurrent connections and complex data relationships. • **Multi-User Environments:** Designed to handle multiple simultaneous connections in environments where scalability is crucial.

In summary, SQLite's file-based architecture, serverless design, and minimal setup make it a practical choice for projects where simplicity, portability, and low resource utilization are essential. MySQL, with its client-server architecture, is better suited for larger, enterprise-level applications with more complex requirements and scalability needs.

Helpful links:

- SQLite Documentation
- SQLite Tutorial

6. Short Summary:

SQLite, with its unique characteristics, has proven to be an invaluable database solution throughout our project. Its lightweight nature and serverless design significantly simplify the development and deployment process. The following are key thoughts on SQLite, along with a brief comparison to MySQL:

Advantages of SQLite:

- **Simplicity and Zero Configuration:** SQLite's simplicity, combined with zero configuration, allows for a seamless development experience. The absence of a dedicated server streamlines deployment and eliminates the need for complex setup steps.
- **Portability and File-Based Architecture:** The file-based architecture of SQLite brings unparalleled portability. The entire database is encapsulated in a single file, making it easy to share and transport. This feature is particularly advantageous in scenarios where mobility and distribution of the database are crucial.
- **Well-Suited for Embedded Systems:** SQLite's small footprint and efficiency make it an ideal choice for embedded systems and resource-constrained environments. It excels in scenarios where a lightweight, serverless database is preferred.
- **ACID Compliance:** Despite its simplicity, SQLite maintains ACID compliance, ensuring data integrity and reliability. This is a crucial aspect, especially when dealing with mission-critical applications.

Challenges Faced:

- **Scalability Considerations:** While SQLite excels in simplicity, it may face challenges in highly concurrent, large-scale applications. In scenarios demanding extensive scalability, a traditional client-server database like MySQL might be more appropriate.
- **Complex Data Relationships:** SQLite is well-suited for simpler data models. In cases

where complex relationships and intricate queries are prevalent, a more robust database system like MySQL could provide better support.

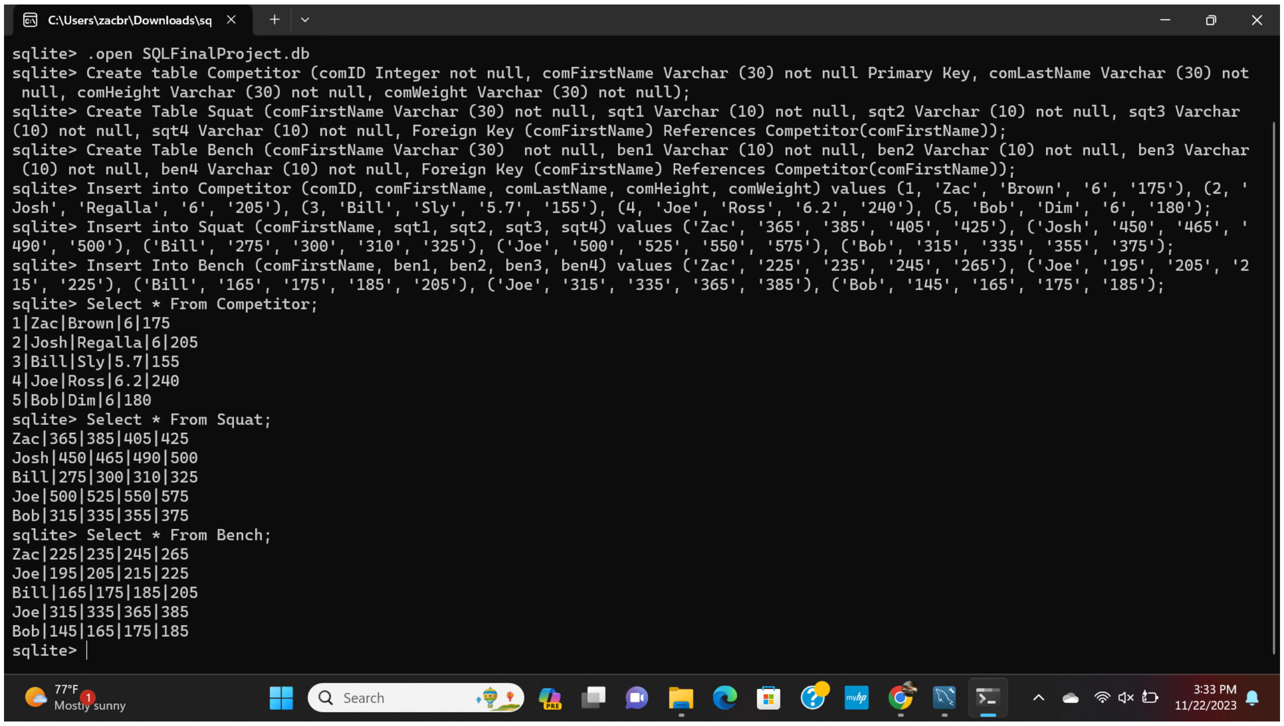
- **Clean Database:** While the database is super simple there is no way of deleting things. This means that if you are trying to take a picture of the code, or show somebody the code so they can use it, and you want it to look clean you have to restart the database every time because there is no deleting.

Comparison to MySQL:

- **Ease of Use:** SQLite shines in terms of ease of use. Its zero-configuration setup and straightforward file-based architecture make it accessible for developers at all levels. MySQL, with its client-server model, involves more setup steps, which might be a consideration in smaller projects.
- **Suitability for Applications:**
- **SQLite:** Ideal for single-user applications, mobile development, and projects with simplicity as a priority.
- **MySQL:** Better suited for enterprise-level applications with higher concurrency, complex data relationships, and scalability requirements.

In conclusion, SQLite is a powerful tool for specific use cases, offering simplicity and efficiency. Its suitability for embedded systems and single-user applications makes it an excellent choice for certain projects. However, understanding the specific needs of the application is crucial, and for larger, more complex systems, MySQL's scalability and robust feature set may be more appropriate.

Code Case 1:



```
sqlite> .open SQLFinalProject.db
sqlite> Create table Competitor (comID Integer not null, comFirstName Varchar (30) not null Primary Key, comLastName Varchar (30) not
 null, comHeight Varchar (30) not null, comWeight Varchar (30) not null);
sqlite> Create Table Squat (comFirstName Varchar (30) not null, sqt1 Varchar (10) not null, sqt2 Varchar (10) not null, sqt3 Varchar
(10) not null, sqt4 Varchar (10) not null, Foreign Key (comFirstName) References Competitor(comFirstName));
sqlite> Create Table Bench (comFirstName Varchar (30)  not null, ben1 Varchar (10) not null, ben2 Varchar (10) not null, ben3 Varchar
 (10) not null, ben4 Varchar (10) not null, Foreign Key (comFirstName) References Competitor(comFirstName));
sqlite> Insert into Competitor (comID, comFirstName, comLastName, comHeight, comWeight) values (1, 'Zac', 'Brown', '6', '175'), (2, '
Josh', 'Regalla', '6', '205'), (3, 'Bill', 'Sly', '5.7', '155'), (4, 'Joe', 'Ross', '6.2', '240'), (5, 'Bob', 'Dim', '6', '180');
sqlite> Insert into Squat (comFirstName, sqt1, sqt2, sqt3, sqt4) values ('Zac', '365', '385', '405', '425'), ('Josh', '450', '465', '
490', '500'), ('Bill', '275', '300', '310', '325'), ('Joe', '500', '525', '550', '575'), ('Bob', '315', '335', '355', '375');
sqlite> Insert Into Bench (comFirstName, ben1, ben2, ben3, ben4) values ('Zac', '225', '235', '245', '265'), ('Joe', '195', '205', '2
15', '225'), ('Bill', '165', '175', '185', '205'), ('Joe', '315', '335', '365', '385'), ('Bob', '145', '165', '175', '185');
sqlite> Select * From Competitor;
1|Zac|Brown|6|175
2|Josh|Regalla|6|205
3|Bill|Sly|5.7|155
4|Joe|Ross|6.2|240
5|Bob|Dim|6|180
sqlite> Select * From Squat;
Zac|365|385|405|425
Josh|450|465|490|500
Bill|275|300|310|325
Joe|500|525|550|575
Bob|315|335|355|375
sqlite> Select * From Bench;
Zac|225|235|245|265
Joe|195|205|215|225
Bill|165|175|185|205
Joe|315|335|365|385
Bob|145|165|175|185
sqlite> |
```

Line 1: creating the database

Line 2: creating the competitor table, not null – means you cannot leave data blank, Primary Key – used to reference into another table – all primary keys must be not null, Integer – value inserted into the table must be a number, Varchar (x) - can type any characters up to the x amount.

Line 3: creating the squat table, Foreign key – your primary key becomes a foreign key when you insert it into another table – you cannot have a foreign key without a primary key- you must reference the foreign key back to its original table.

Line 4: creating the bench table

Line 5: inserting data into the competitor table

Line 6: inserting data into the squat table

Line 7: inserting data into the bench table

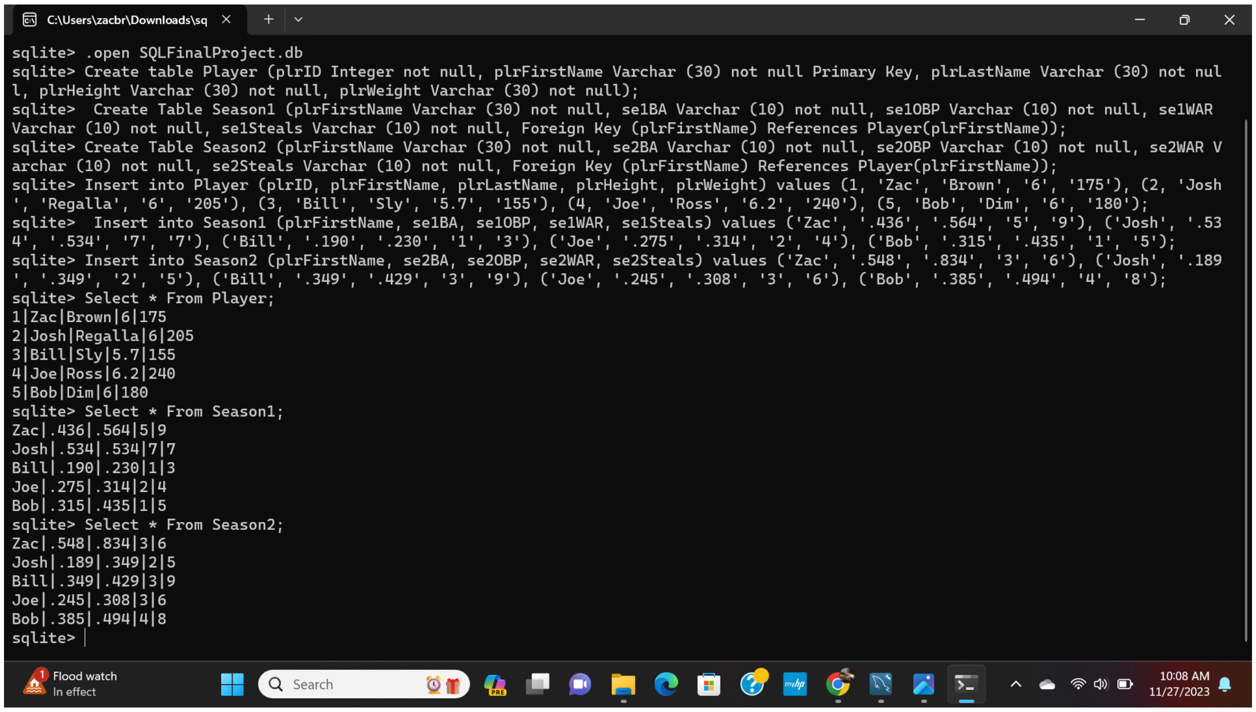
Line 8: querying from the competitor table, now you see each person with their height and weight.

Line 9: querying from the squat table, now you see each competitor and all their squat attempts

Line 10: querying from the bench table, now you see each competitor and all their bench attempts

Saving Project: Once you are done with the database you can close the tab and it will automatically save in the file SQLite.

Code case 2:



```
sqlite> .open SQLFinalProject.db
sqlite> Create table Player (plrID Integer not null, plrFirstName Varchar (30) not null Primary Key, plrLastName Varchar (30) not nul
l, plrHeight Varchar (30) not null, plrWeight Varchar (30) not null);
sqlite>  Create Table Season1 (plrFirstName Varchar (30) not null, se1BA Varchar (10) not null, se1OBP Varchar (10) not null, se1WAR
Varchar (10) not null, se1Steals Varchar (10) not null, Foreign Key (plrFirstName) References Player(plrFirstName));
sqlite> Create Table Season2 (plrFirstName Varchar (30) not null, se2BA Varchar (10) not null, se2OBP Varchar (10) not null, se2WAR V
archar (10) not null, se2Steals Varchar (10) not null, Foreign Key (plrFirstName) References Player(plrFirstName));
sqlite> Insert into Player (plrID, plrFirstName, plrLastName, plrHeight, plrWeight) values (1, 'Zac', 'Brown', '6', '175'), (2, 'Josh
', 'Regalla', '6', '205'), (3, 'Bill', 'Sly', '5.7', '155'), (4, 'Joe', 'Ross', '6.2', '240'), (5, 'Bob', 'Dim', '6', '180');
sqlite>  Insert into Season1 (plrFirstName, se1BA, se1OBP, se1WAR, se1Steals) values ('Zac', '.436', '.564', '5', '9'), ('Josh', '.53
4', '.534', '7', '7'), ('Bill', '.190', '.230', '1', '3'), ('Joe', '.275', '.314', '2', '4'), ('Bob', '.315', '.435', '1', '5');
sqlite> Insert into Season2 (plrFirstName, se2BA, se2OBP, se2WAR, se2Steals) values ('Zac', '.548', '.834', '3', '6'), ('Josh', '.189
', '.349', '2', '5'), ('Bill', '.349', '.429', '3', '9'), ('Joe', '.245', '.308', '3', '6'), ('Bob', '.385', '.494', '4', '8');
sqlite> Select * From Player;
1|Zac|Brown|6|175
2|Josh|Regalla|6|205
3|Bill|Sly|5.7|155
4|Joe|Ross|6.2|240
5|Bob|Dim|6|180
sqlite> Select * From Season1;
Zac|.436|.564|5|9
Josh|.534|.534|7|7
Bill|.190|.230|1|3
Joe|.275|.314|2|4
Bob|.315|.435|1|5
sqlite> Select * From Season2;
Zac|.548|.834|3|6
Josh|.189|.349|2|5
Bill|.349|.429|3|9
Joe|.245|.308|3|6
Bob|.385|.494|4|8
sqlite>
```

Line 1: create the database

Line 2: creating the player table, uses not null, primary key, integer and varchar – reference above

Line 3: creating the season1 table

Line 4: creating the season2 table

Line 5: inserting data into the player table

Line 6: inserting data into the season1 table

Line 7: inserting data into the season2 table

Line 8: querying the player table, you will now see all the players and their height and weight

Line 9: querying the season1 table, you will now see all the players and their stats for season1

Line 10: querying the season2 table, you will now see all the players and their stats for season2

Saving Project: Once you are done with the database you can close the tab and it will automatically save in the file SQLite.

Works Cited

1. SQLite documentation. (n.d.). https://www.sqlite.org/docs.html

2. SQLite tutorial. (2023, July 20). https://www.sqlitetutorial.net/