SQLite Tutorial: Managing a Library System

Dr. Frank Xu

March 3, 2025

Introduction

In this tutorial, we will create a simple SQLite database to manage a library system. The database will store information about books, including their title, author, publication year, and genre.

Step 1: Create a New SQLite Database

Open your terminal or command prompt and create a new SQLite database called `library.db`:

```
sqlite3 library.db
```

Once you run the above command, you should see something like this in your terminal:

```
user@linux:~$ sqlite3 library.db
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
sqlite>
```

This indicates that SQLite has successfully created the `library.db` file and opened the SQLite shell. You can now start executing SQL commands directly in this shell.

Step 2: Create a Table

Now, let's create a table called `books` to store information about the books in our library. The table will have the following columns:

- `id`: A unique identifier for each book (Primary Key).
- `title`: The title of the book.
- `author`: The author of the book.
- `year`: The year the book was published.

- `genre`: The genre of the book.

Run the following SQL command to create the table:

```
CREATE TABLE books (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    author TEXT NOT NULL,
    year INTEGER,
    genre TEXT
);
```

After running the command, you won't see any output, but the table will be created. To verify the table schema, use the '.schema' command:

```
sqlite> .schema books
CREATE TABLE books (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    author TEXT NOT NULL,
    year INTEGER,
    genre TEXT
);
```

The '.schema' command displays the exact SQL statement used to create the table, allowing you to confirm its structure.

You use a different command 'PRAGMA table_info' to inspect the structure of a table.

```
sqlite> PRAGMA table_info(books);
```

Step 3: Insert Data into the Table

Now that we have a table, let's insert some sample data into it.

```
INSERT INTO books (title, author, year, genre)
VALUES ('To Kill a Mockingbird', 'Harper Lee', 1960, 'Fiction');

INSERT INTO books (title, author, year, genre)
VALUES ('1984', 'George Orwell', 1949, 'Dystopian');

INSERT INTO books (title, author, year, genre)
VALUES ('The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Classic')
    ;

INSERT INTO books (title, author, year, genre)
VALUES ('Pride and Prejudice', 'Jane Austen', 1813, 'Romance');
```

After inserting the data, you won't see any output, but you can verify the data by running a 'SELECT' query:

```
sqlite> SELECT * FROM books;
1|To Kill a Mockingbird|Harper Lee|1960|Fiction
2|1984|George Orwell|1949|Dystopian
3|The Great Gatsby|F. Scott Fitzgerald|1925|Classic
4|Pride and Prejudice|Jane Austen|1813|Romance
```

Step 4: Query the Data

Now that we have some data in the table, let's query it.

1. Select All Books

To retrieve all the books in the library, run:

```
SELECT * FROM books;
```

The result will look like this:

```
sqlite> SELECT * FROM books;
1|To Kill a Mockingbird|Harper Lee|1960|Fiction
2|1984|George Orwell|1949|Dystopian
3|The Great Gatsby|F. Scott Fitzgerald|1925|Classic
4|Pride and Prejudice|Jane Austen|1813|Romance
```

2. Select Books Published After 1900

To find books published after the year 1900, run:

```
SELECT * FROM books WHERE year > 1900;
```

The result will look like this:

```
sqlite> SELECT * FROM books WHERE year > 1900;
1|To Kill a Mockingbird|Harper Lee|1960|Fiction
2|1984|George Orwell|1949|Dystopian
3|The Great Gatsby|F. Scott Fitzgerald|1925|Classic
```

3. Select Books by Genre

To find all books in the "Fiction" genre, run:

```
SELECT * FROM books WHERE genre = 'Fiction';
```

The result will look like this:

```
sqlite> SELECT * FROM books WHERE genre = 'Fiction';
1|To Kill a Mockingbird|Harper Lee|1960|Fiction
```

Step 5: Update Data

Let's say we want to update the genre of "1984" from "Dystopian" to "Science Fiction". We can do this with the `UPDATE` statement:

```
UPDATE books SET genre = 'Science Fiction' WHERE title = '1984';
```

After running the command, you won't see any output, but you can verify the change by querying the data again:

```
sqlite> SELECT * FROM books WHERE title = '1984';
2|1984|George Orwell|1949|Science Fiction
```

Step 6: Delete Data

If we want to remove a book from the library, we can use the `DELETE` statement. For example, to delete "Pride and Prejudice":

```
DELETE FROM books WHERE title = 'Pride and Prejudice';
```

After running the command, you won't see any output, but you can verify the deletion by querying the data again:

```
sqlite> SELECT * FROM books;
1|To Kill a Mockingbird|Harper Lee|1960|Fiction
2|1984|George Orwell|1949|Science Fiction
3|The Great Gatsby|F. Scott Fitzgerald|1925|Classic
```

Step 7: Drop the Table (Optional)

If you want to delete the entire table (including all its data), you can use the `DROP TABLE` command:

```
DROP TABLE books;
```

After running the command, you won't see any output, but you can verify that the table is gone by listing all tables:

```
sqlite> .tables
```

Since the table has been dropped, no tables will be listed.

Step 8: Other Useful Commands

You can try other useful commands

```
SELECT * FROM books WHERE genre like '%F%';
SELECT * FROM books WHERE lower(genre) like '%f%';
SELECT * FROM books WHERE lower(genre) like '%f%' limit 2;
```

Step 9: Exit SQLite

Once you're done working with the database, you can exit the SQLite shell by typing:

```
.exit
```

The terminal will return to the normal command prompt:

```
sqlite> .exit
user@linux:~$
```

Conclusion

In this tutorial, we created a simple SQLite database to manage a library system. We learned how to create a table, verify its schema, insert data, query data, update records, and delete records. SQLite is a lightweight, serverless database engine that is perfect for small applications, prototyping, and learning SQL.

Feel free to experiment with more complex queries and operations as you become more comfortable with SQLite!