Using SQLite

by Sophia

WHAT'S COVERED

This lesson explores the unique features of SQLite, in four parts. Specifically, this lesson will cover:

1. Introduction

SQLite is a unique database with some approaches that distinguish it from other databases. With SQLite, you can manage structured data simply and efficiently with a powerful, lightweight, and self-contained relational database management system.

HINT

There is no need to install a database server for SQLite to run, as it is a serverless database. It can be directly integrated into applications without complex setup or administration. A SQLite database is written in C, has a small footprint and minimal overhead, and is highly efficient. This makes it ideal for applications that run on mobile devices, embedded systems, and resource-constrained desktops.

Although SQLite is small and straightforward, it has a rich feature set and supports a variety of SQL commands and data types. The system is fully ACID-compliant, ensuring the integrity and reliability of data, and it supports concurrent access with robust locking mechanisms. Among the many industries and applications that use SQLite are mobile app development (Android and iOS), web browsers, Internet of Things devices, desktop software, embedded systems, and embedded data storage. Developers seeking an easy-to-use, portable, and versatile database solution that seamlessly integrates into their projects prefer its ease of use, portability, and versatility.

SQLite has been ported to various platforms like Windows, macOS, Linux, iOS, Android, and many others. The applications that use the SQLite database don't have to be written in a specific language, as you would typically

see with other databases. It works correctly as long as there is some way to bind and work with the external libraries. All of the source code for SQLite is public domain so that it can be reused in other programs with no restrictions.

2. Data Types

SQLite has only a few data types:

- REAL: A decimal number with a relatively low level of precision.
- BLOB: A binary file such as an image file. BLOB is an acronym for Binary Large Object.
- NULL: The absence of a value or the unknown.
- INTEGER: A whole number.
- TEXT: Character strings.

DID YOU KNOW

SQLite is quite forgiving of the kind of data that you enter. If a column has the data type of an integer, and you try to insert a text string that contains only numeric digits into that column, SQLite will try to convert the text string into an integer. For example, if the user enters a text string 9876 into an integer column, that value is converted to an integer of 9876 and stored in the column. If you try inserting a non-numeric string like “abcd” into an integer column, most other databases will throw an error. However, SQLite will just store the string value in the column.

In the database you will work with later in the course, many columns have a VARCHAR(40) that allows up to 40 characters. (VARCHAR is a data type for variable length values; you will learn about it later.) If you were to try to insert more than 40 characters into such a column, many databases would either throw an error or truncate the string to 40 characters. SQLite instead stores the entire string without the loss of information. This is viewed as a feature of SQLite rather than a bug. However, it makes it very difficult for applications built using SQLite to be ported to other databases, as such situations can create problems.

Booleans are another instance where SQLite does not store a separate data type. Instead, true and false are represented by integers of 0 and 1.

Dates are also different in SQLite, where they are stored in a TEXT string. If a date is stored as an integer, SQLite stores the number of seconds since midnight on January 1, 1970. There are built-in date and time features to change between those values.

Tables can be created without any data types at all, which can be confusing. For example, you could run a SQL command like this:

```
CREATE TABLE myTable (a, b, c);
```

This would create a table named myTable with the column names a, b, and c that have no data types defined. Anything of any size could be stored in those columns.

3. Primary Keys

Each record in a table is uniquely identified by its primary key. Primary keys in SQLite are defined as columns or column combinations.

KEY CONCEPT

Each row in the table must have at least one unique value. In SQLite, the uniqueness constraint is enforced automatically, so duplicate values are not inserted into the primary key column(s).

The primary key column(s) of SQLite are automatically indexed. This significantly enhances query performance when searching for records using primary key values. Column indexes can also be created to speed up data retrieval, particularly for large tables.

The SQLite database also supports composite keys, which use multiple columns to create unique identifiers for each row. A composite key is defined by listing the columns separated by commas within the PRIMARY KEY constraint.

In SQLite, a primary key can be a null value. This was originally a bug in the program. However, when the bug was identified, so many databases had already used that bug as a feature that the bug was kept. If the primary key field is of an INTEGER type, however, null values are not allowed. Even though it is technically possible to have a null value in a primary key field, it is not recommended because it defeats the purpose of the primary key field containing a unique identifier for each row.

---

4. Table Management

Creating, modifying, and maintaining tables within a SQLite database is all part of table management. Using the CREATE TABLE SQL statement, developers specify the table name, column names, data types, and constraints for a new table.

As you learned earlier in this lesson, SQLite stores different data types, including INTEGER, TEXT, REAL, and BLOB. To ensure data integrity, the table can also be configured with constraints like PRIMARY KEY, UNIQUE, NOT NULL, and FOREIGN KEY. Using the INSERT INTO statement, you can insert records (rows) and retrieve data with the SELECT statement. The ALTER TABLE statement enables developers to modify existing tables by adding or dropping columns, altering column data types, or applying constraints. Database structures can be designed and managed efficiently using SQLite's intuitive and flexible table management system. You will learn to use all of these commands later in the course.

SQLite enables developers to efficiently create and modify tables and manage table data. In SELECT queries, developers can filter and sort the data to retrieve specific data, update existing records using UPDATE statements, and delete records using DELETE statements. With SQLite, multiple operations can be grouped as a single unit, ensuring data consistency and reliability.

Developers can define triggers, which perform actions in response to specific database events, such as insertion, deletion, or updating of data. As a whole, SQLite's table management capabilities offer a comprehensive solution to managing table structures, manipulating data, and maintaining data integrity.

5. Joins

Developers can use SQLite joins to combine data from multiple tables based on related columns. Among the types of joins supported by SQLite are INNER JOIN, LEFT JOIN (or LEFT OUTER JOIN), RIGHT JOIN (or RIGHT OUTER JOIN), and FULL JOIN (or FULL OUTER JOIN). You will learn how to use all these join commands later in the course. Based on the condition specified, an INNER JOIN returns only matching rows from both tables. A LEFT JOIN returns all rows from the left table along with matching rows from the right table, while a RIGHT JOIN does the opposite, returning all rows from the right table along with matching rows from the left table. When a FULL JOIN is performed, all rows from both tables are returned, including unmatched rows, with NULL values for nonmatching rows. By joining data from multiple tables, developers can generate comprehensive results by combining related information. The “ON” keyword defines column relationships between tables using join conditions. The use of joins is beneficial when dealing with complex data models, normalized databases, or situations involving multiple data tables.

A developer must carefully consider the performance implications when using joins in SQLite, as inefficient joins can slow down the execution of queries. Join conditions can be improved significantly by indexing the columns involved. Developers can use the keyword JOIN or shorthand notations like INNER JOIN, LEFT JOIN, and so on, according to their preference. When SQLite joins are used effectively, developers can retrieve data from multiple tables in a single query, which helps facilitate comprehensive data analysis. In developing sophisticated database applications, combining and presenting data from different tables meaningfully requires a flexible and efficient way of working with related data.

SUMMARY

In this lesson, you learned in the **introduction** that an embedded SQLite database management system is lightweight, self-contained, and easy to use. This is a serverless database, meaning it operates directly within the application and does not require a separate server process. This compact and efficient database program is written in C programming language, making it an ideal choice for resource-constrained environments like mobile devices, embedded systems, and desktop applications. You learned that each record is recorded in a table using **primary keys**, and that SQLite only has a few **data types**, though it offers reliable and ACID-compliant transaction support despite its small footprint. You also learned that SQLite can help with **table management**, and developers can use SQLite **joins**, such as INNER JOIN, LEFT JOIN (or LEFT OUTER JOIN), RIGHT JOIN (or RIGHT OUTER JOIN), and FULL JOIN (or FULL OUTER JOIN), to combine data from multiple tables based on related columns. Several applications use SQLite, including mobile apps (on Android and iOS), desktop software, web browsers, IoT devices, and embedded systems, where lightweight and self-contained database capabilities are required.

Source: THIS TUTORIAL WAS AUTHORED BY DR. VINCENT TRAN, PHD (2020) AND Faithe Wempen (2024) FOR SOPHIA LEARNING. PLEASE SEE OUR **TERMS OF USE**.