SQLite Course: Beginner to Advanced

Introduction to SQLite

- **What is SQLite?**
  - Overview of SQLite as a lightweight, serverless database.
  - Key features: Zero-configuration, self-contained, and cross-platform.
  - Use cases: Mobile apps, embedded systems, prototyping.
- **Installing SQLite**
  - Installation steps for Windows, macOS, and Linux.
  - Setting up SQLite CLI.
  - Verifying installation.
- **SQLite Ecosystem**
  - Tools and libraries for SQLite (e.g., DB Browser for SQLite, SQLiteStudio).

Section 1: Getting Started with SQLite

1.1 SQLite Basics

- Creating and connecting to a database:

```
sqlite3 mydatabase.db
```

- Basic SQLite commands:

```
.databases  -- List databases
.tables     -- List tables
.schema     -- Show schema of a table
.exit       -- Exit SQLite CLI
```

1.2 Creating and Managing Tables

- Creating a table:

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

- Viewing table schema:

```
.schema students
```

- Dropping a table:

```
DROP TABLE students;
```

1.3 Basic CRUD Operations

- Inserting data:

```
INSERT INTO students (name, age, grade) VALUES ('John Doe', 15,
'10th');
```

- Reading data:

```
SELECT * FROM students;
```

- Updating data:

```
UPDATE students SET grade = '11th' WHERE name = 'John Doe';
```

- Deleting data:

```
DELETE FROM students WHERE name = 'John Doe';
```

Section 2: Intermediate SQLite

2.1 SQLite Data Types

- Overview of SQLite type affinity: INTEGER, TEXT, BLOB, REAL, and NULL.
- Dynamic typing in SQLite and how it differs from strict typing.

2.2 Constraints

- Using constraints:
  - PRIMARY KEY, UNIQUE, NOT NULL, DEFAULT, CHECK.
- Example:

```
CREATE TABLE teachers (
    id INTEGER PRIMARY KEY,
    name TEXT UNIQUE NOT NULL,
    subject TEXT DEFAULT 'Math'
);
```

2.3 Querying with Conditions

- Using WHERE clause with operators:

```
SELECT * FROM students WHERE age > 15;
```

- Pattern matching with LIKE:

```
SELECT * FROM students WHERE name LIKE 'J%';
```

- Using logical operators: AND, OR, NOT.

2.4 Joins

- Understanding joins in SQLite:
  - INNER JOIN:

```
SELECT students.name, grades.score
FROM students
```

```
INNER JOIN grades ON students.id = grades.student_id;
```

- LEFT JOIN, CROSS JOIN.

Section 3: Advanced SQLite

3.1 Indexing

- Creating indexes to optimize queries:

```
CREATE INDEX idx_students_name ON students (name);
```

- Viewing indexes:

```
PRAGMA index_list('students');
```

- Dropping indexes:

```
DROP INDEX idx_students_name;
```

3.2 Views

- Creating views:

```
CREATE VIEW student_grades AS
SELECT students.name, grades.subject, grades.score
FROM students
JOIN grades ON students.id = grades.student_id;
```

- Querying views:

```
SELECT * FROM student_grades;
```

- Dropping views:

```
DROP VIEW student_grades;
```

3.3 Triggers

- Creating triggers:

```
CREATE TRIGGER update_time
AFTER UPDATE ON students
BEGIN
    UPDATE students SET updated_at = CURRENT_TIMESTAMP WHERE id
= NEW.id;
END;
```

- Listing triggers:

```
SELECT name FROM sqlite_master WHERE type = 'trigger';
```

- Dropping triggers:

```
DROP TRIGGER update_time;
```

3.4 Transactions

- Using transactions:

```
BEGIN TRANSACTION;
INSERT INTO students (name, age, grade) VALUES ('Jane Doe', 16,
'11th');
ROLLBACK; -- or COMMIT;
```

Section 4: SQLite Administration

4.1 Backup and Restore

- Backing up a database:

```
sqlite3 mydatabase.db .dump > backup.sql
```

- Restoring a database:

```
sqlite3 newdatabase.db < backup.sql
```

4.2 Analyzing Database Performance

- Using EXPLAIN to analyze queries:

```
EXPLAIN QUERY PLAN SELECT * FROM students WHERE age > 15;
```

- Optimizing queries with indexes and avoiding full table scans.

4.3 Security in SQLite

- Best practices for securing SQLite databases:
  - Using file permissions to restrict access.
  - Enabling encryption with SQLite Encryption Extension (SEE) or other third-party tools.

Section 5: Real-World Projects

5.1 To-Do List Application

- Designing tables for tasks, categories, and priorities.
- Writing queries for task management.

5.2 Expense Tracker

- Designing tables for transactions, categories, and budgets.
- Using aggregate functions for monthly expense reports.

Conclusion

- Recap of SQLite features and capabilities.

- Best practices for working with SQLite.
- Resources for further learning.

Appendix

- Common SQLite commands reference.
- Troubleshooting common errors.