10.0: SQLite primer

Contents:

This course assumes that you are familiar with databases in general, SQL databases in particular, and the SQL language used to interact with them. This chapter is a refresher and quick reference only.

SQL databases

SQL databases store data in tables of rows and columns:

- The intersection of a row and column is called a *field*.
- Fields contain data, references to other fields, or references to other tables.
- Rows are identified by unique IDs.
- Columns are identified by names that are unique per table.

Think of a database as a spreadsheet with rows, columns, and cells, where cells can contain data, references to other cells, and links to other sheets.

SQLite

SQLite is a software library. SQLite implements an SQL database engine that has the following characteristics:

- Self-contained (requires no other components)
- Serverless (requires no server backend)
- Zero-configuration (does not need to be configured for your app)
- Transactional (changes within a single transaction in SQLite either occur completely or not at all)

SQLite is the most widely deployed database engine in the world. The source code for SQLite is in the public domain.

For details of the SQLite database, see the SQLite website.

Example table

SQLite stores data in tables.

Assume the following:

- A database named `DATABASE NAME`
- A table named `WORD LIST_TABLE`
- Columns for `_id`, `word`, and `definition`

After inserting the words `alpha` and `beta`, where `alpha` has two definitions, the `DATABASE_NAME` database table named `WORD_LIST_TABLE` might look like this:

WORD_LIST_TABLE		
_id	word	definition
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

You can find what's in a specific row using the `_id`, or you can retrieve rows by formulating queries that select rows from the table by specifying constraints. You use the SQL query language discussed below to create queries.

Transactions

A *transaction* is a sequence of operations performed as a single logical unit of work. A logical unit of work must exhibit four properties, called the atomicity, consistency, isolation, and durability (ACID) properties, to qualify as a transaction:

- **Atomicity.** Either all of a transaction's data modifications are performed, or none of them are performed. This is true even if the act of writing the change to the disk is interrupted by a program crash, operating system crash, or power failure.
- **Consistency.** When completed, a transaction must leave all data in a consistent state.
- **Isolation.** Modifications made by concurrent transactions must be isolated from the modifications made by any other concurrent transactions. A transaction either recognizes data in the state it was in before another concurrent transaction modified it, or it recognizes the data after the second transaction has completed, but it does not recognize an intermediate state.
- **Durability.** After a transaction has completed, its effects are permanently in place in the system. The modifications persist even in the event of a system failure.

Examples of transactions:

- Transferring money from a savings account to a checking account.
- Entering a term and definition into dictionary.
- Committing a changelist to the master branch.

More on transactions.

Query language

You use a special SQL query language to interact with the database. Queries can be very complex, but there are four basic operations:

- Inserting rows
- Deleting rows
- Updating values in rows
- Retrieving rows that meet given criteria

On Android, the database object provides convenient methods for inserting, deleting, and updating the database. You only need to understand SQL for retrieving data.

Full description of the query language.

Query structure

An SQL query is highly structured and contains the following basic parts:

- `SELECT word, description FROM WORD_LIST_TABLE WHERE word="alpha"`

Generic version of sample query:

- `SELECT` *columns* `FROM` *table* `WHERE` *column="value"*

Parts:

- `SELECT` *columns*: Select the columns to return. Use `*` to return all columns.
- `FROM` *table*: Specify the table from which to get results.
- `WHERE`: Keyword that precedes conditions that have to be met, for example *column="value"*. Common operators are `=`, `LIKE`, `<`, and . To connect multiple conditions, use `AND` or `OR`.
- `ORDER BY`: Specify `ASC` for ascending, or `DESC` for descending. For default order, omit `ORDER BY`.
- `LIMIT`: Very useful keyword if you want to only get a limited number of results.

Sample queries

1	`SELECT * FROM WORD_LIST_TABLE`	Gets the whole table.
2	`SELECT word. definition FROM WORD_LIST_TABLE WHERE _id > 2`	Returns `[["alpha", "particle"]]`
3	`SELECT id FROM WORD LIST TABLE WHERE word="alpha" AND definition LIKE "%art%"`	Returns the `id` of the word `alpha` with the substring `art` in the definition. `[["3"]]`
4	`SELECT * FROM WORD LIST TABLE ORDER BY word DESC LIMIT 1`	Sorts in reverse and gets the first item. This gives you the last item per sort order. Sorting is by the first column, in this case, the `_id`. `[["3", "alpha", "particle"]]`
5	`SELECT * FROM WORD_LIST_TABLE LIMIT 2,1`	Returns 1 item starting at position 2. Position counting starts at 1 (not zero!). Returns `[["2", "beta", "second letter"]]`

You can practice creating and querying databases at this Fiddle website.

Queries for Android SQLite

You can send queries to the SQLite database of the Android system as raw queries or as parameters.

The `rawQuery(String sql, String[] selectionArgs)` method runs the provided SQL and returns a `Cursor` of the result set. The following table shows how the first two sample queries from above would look as raw queries:

1	`String query = "SELECT * FROM WORD_LIST_TABLE"; rawQuery(query, null);`
2	`query = "SELECT word, definition FROM WORD_LIST_TABLE WHERE _id> ? "; String[] selectionArgs = new String[]{"2"} rawQuery(query, selectionArgs) ;`

The `query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)` method queries the given table, returning a `Cursor` over the result set. Here's a query showing how to fill in the arguments:

```
SELECT * FROM WORD_LIST_TABLE
WHERE word="alpha"
ORDER BY word ASC
LIMIT 2,1;
```

This query returns:

```
[["alpha", "particle"]]
```

Argument examples:

```
String table = "WORD_LIST_TABLE"
String[] columns = new String[]{"*"};
String selection = "word = ?"
String[] selectionArgs = new String[]{"alpha"};
String groupBy = null;
String having = null;
String orderBy = "word ASC"
String limit = "2,1"

query(table, columns, selection, selectionArgs, groupBy, having, orderBy, limit);
```

Note that in real code, you wouldn't create variables for `null` values. See the SQLiteDatabase documentation for versions of this method with different parameters.

Cursors

A *cursor* is a pointer into a row of structured data. You can think of a cursor as a pointer to a table row.

A query returns a `Cursor` object that points to the first element in the query result. The `Cursor` class provides methods for moving the cursor through the query result, and methods to get the data from the columns of each row in the result.

When a method returns a `Cursor` object, you iterate over the result, extract the data, do something with the data, and close the cursor to release the memory.