The SQL Language

The SQL Language

- Almost universal means of interacting with a relational database
- Designed for configuring, structuring, reading, writing, filtering, sorting, calculating information
- **SQL is a command language**
  - Assumes we know what we are doing. It does not ask for confirmation to perform the commands we issue
- We'll cover some basics of SQL using commands such as **CREATE TABLE, INSERT** and **SELECT**

The SQL Language

Portability

- Almost every database product(including SQLite) has enhancements to the core language that help differentiate it form other products
  - Typically these are **performance enhancements**
- **Learning the standard syntax of SQL give us a degree of portability**
  - We still need to learn the specifics of the product we are using; SQLite in our case

SQL - Basic Syntax

- SQL consists of a number of different commands such as **CREATE TABLE** or **INSERT**
- They are processed one at a time
- SQL is case-insensitive
  - It is customary to capitalize these commands and keywords to make easier to identify
- SQL is whitespace insensitive, including line breaks
- Individual statements are separated by a semicolon
  - A semicolon indicates the end of a statement

SQL - Basic Syntax – Data Types

Literals - also called constants, denote explicit values

- **Numeric literals**, are represented in
  - **Integer**
  - **Decimal**
  - **Scientific notation**

    -10, -10, 5.234, 4.0123223E23

  (SQLite requires the decimal point to always be represented as a period (.) regardless of the current international setting)

SQL - Basic Syntax - Data Types

- **String/text literals**
  - – one or more alphanumeric characters enclosed in single quotes (‘ ‘)

    'James'

    'hello'If single quotes are part of the string value, use two successive single quotes

Example:

```
SELECT 'IT''S A GOOD IDEA';
```

A comma is used to separate statements and expressions

Example:

```
SELECT 2*2 ,  3*5 ;
```

SQL - Basic Syntax

Identifiers and keywords

- Identifiers refer to specific objects in a database
  - table names, column names, etc.
- Keywords are words with specific meaning in SQL
  - **SELECT, UPDATE, INSERT**, etc.

- SQLite is case sensitive with respect to string values
  - The value **‘Terry**’ is not the same as the value **‘terry’**

Three-Value Logic

Example: Check if the value in a column named ‘city’ is ‘Sarasota’

1. If the value of Sarasota is in city, then this expression evaluates **true**
2. If the value in city is Tampa, then this expression evaluates **false**
3. If there no value has been stored in ‘city’, then the value for ‘city’ is **unknown**

   The correct syntax in SQLite would be:

```
SELECT * FROM tableName WHERE city = ‘Sarasota’
SELECT * FROM tableName WHERE city IS NULL
SELECT * FROM tableName WHERE city NOT NULL
```

Basic Operators

- Produce some kind of result
- Take one or more values as input and produce a value as output

Operator	Type	Action/Meaning
||	String	Concatenation
*	Arithmetic	Multiply
/	Arithmetic	Divide
%	Arithmetic	Modulus
+	Arithmetic	Add
-	Arithmetic	Subtract
&	Logical	And
|	Logical	Or

Basic Operators

<	Relational	Less than
<=	Relational	Less than or equal to
>	Relational	Greater than
>=	Relational	Greater than or equal to
=	Relational	Equal to
==	Relational	Equal to
<>	Relational	Not equal to
!=	Relational	Not equal to

Basic Operators

IN	Logical	
AND	Logical	
OR	Logical	
LIKE	Relational	String matching

Tables – Column Types

(SQLite supports only **five concrete data types**)

- **NULL**
  - Does not hold a value
- **Integer**
  - 8 bytes in length
  - Range between **-9,223,372,036,854,775,808** and **+9,223,372,036,854,775,8087** (roughly 19 digits)
- **Float**
  - 8 bytes
  - Numeric digits that include a decimal point

Tables – Column Types

- **Text**
  - Variable length strings
  - Literal text values represented using character strings in single quotes
- **BLOB**
  - Binary Large Object (like an image or a file)
  - The data that actually gets stored in the database is the bytes that make up an image or a file
  - Literal BLOBs are represented as hexadecimal text string preceded by an x
    - Example: x’1234ABCD’ represents a 4-byte BLOB

Dropping Tables

- DROP TABLE command
  - Deletes a table and all of its data
  - The table definition is also removed from the database catalogs
  - It also drops any indexes associated with the table
  - Syntax

    DROP TABLE table_name

Data Manipulation Language (DML)

- Used to get user data in and out of a database
- SQLite supports two DML categories
  - **Update commands**
    - INSERT, DELETE, UPDATE
  - **Query commands**
    - Used to extract data from the database
    - The only command is SELECT

Update commands (Insert)

- These **are row modification commands**
- INSERT
  - Creates a new row in the specified table
  - Syntax

INSERT INTO table_name (column_name1, column_name2…) VALUES (value1, value2…);

  - The list of column names and the list of values must have the same number of items
  - Columns can be listed in any order as long as their values are entered in the correct order to match them

Update commands (Insert)

- long as their values are entered in the correct order to match them
  - Technically, the list of column names is optional
    - The number and order of values must match the order of column names

    INSERT INTO table_name VALUES (value1, value2…);
  - Examples:

INSERT INTO parts (name, stock, status) VALUES (‘Widget’, 17, ‘IN STOCK’);

or

INSERT INTO parts   VALUES (‘Widget’, 17, ‘IN STOCK’);

(see Appendix C for details)

Update commands (Update)

- **UPDATE**
  - Used to assign new values to one or more columns of existing rows in a table
  - All of the rows being updated must be part of the same table
  - Syntax

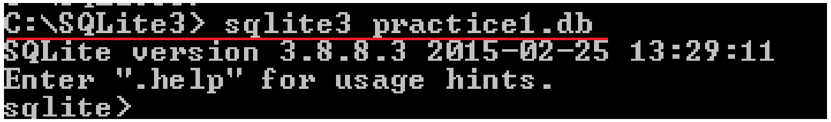
UPTATE table_name SET column_name = new_value [,…] WHERE expression

- If WHERE is not used, the command will attempt to update the designated columns in <u>every row of a table</u>

Create a database and a table

Create a database and name it practice1.db

From the command line, at the sqlite3 prompt type: **sqlite3 practice.db**



```
C:\SQLite3> sqlite3 practice1.db
SQLite version 3.8.8.3 2015-02-25 13:29:11
Enter ".help" for usage hints.
sqlite>
```

Create a table, parts, with the following structure/schema:

```
CREATE TABLE parts (part_id  INTEGER   PRIMARY KEY,  stock   INTEGER,  desc TEXT                 );
```

Insert a record into the parts table:

```
INSERT INTO parts (part_id, stock, desc) VALUES (100, 501, 'coffee cup');
```

Practice entering 2 more records

Update commands (Delete)

- **DELETE**
  - Used to remove one or more rows from a single table
  - The command only requires a table name and a conditional expression to pick up the rows
  - WHERE is used to select specific rows to delete
  - If no WHERE condition is used, the DELETE command will attempt to delete every row of a table

Query command

- The most basic form of SELECT is

SELECT output_list FROM input_table WHERE row_filter

Examples assuming a table has been created with the following:

CREATE TABLE tbl1 (n1 INTEGER, n2 INTEGER, n3 INTEGER, id INTEGER PRIMARY KEY);

INSERT INTO tbl1 (n1, n2, n3) VALUES (10,20,30);

INSERT INTO tbl1 (n1, n2, n3) VALUES (11,21,31);

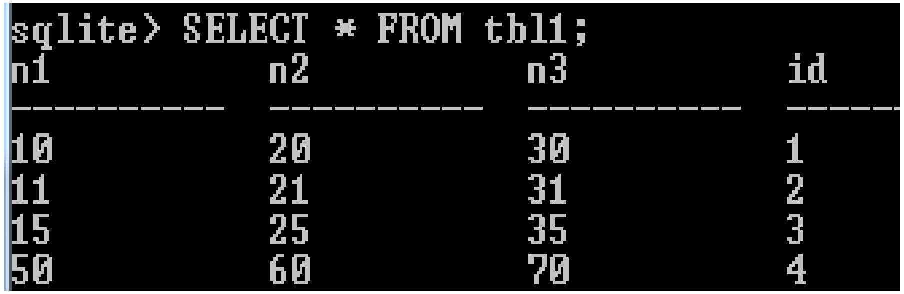
INSERT INTO tbl1 (n1, n2, n3) VALUES (15,25,35);

INSERT INTO tbl1 (n1, n2, n3) VALUES (50,60,70);

Note: these dot commands where used to display data in columns with headers

.headers on

.mode column



```
sqlite> SELECT * FROM tbl1;
n1          n2          n3          id
----------  ----------  ----------  -----
10          20          30          1
11          21          31          2
15          25          35          3
50          60          70          4
```

Query command

Examples

- Return all records from this table

SELECT * FROM tbl1;

- Return the values for the first two columns

SELECT n1, n2 FROM tbl1;

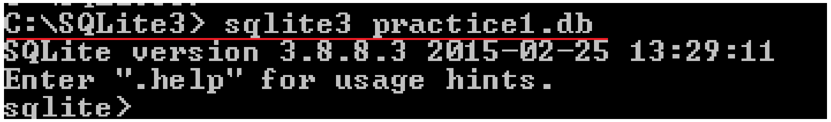
- Return rows where the value in n2 is greater than 20

SELECT * FROM tbl1 WHERE n2 > 20;

Practice Exercise

1. Create a database and name it practice1.db

From the command line, at the sqlite3 prompt type**: sqlite3 practice.db**



```
C:\SQLite3> sqlite3 practice1.db
SQLite version 3.8.8.3 2015-02-25 13:29:11
Enter ".help" for usage hints.
sqlite>
```

2. Create a table, employee, with the following structure/schema:

```
CREATE TABLE employee (EmployeeId INTEGER PRIMARY KEY,
LastName TEXT, FirstName TEXT, CellPhone TEXT, ExperienceLevel TEXT);
```

3. Insert the following records into the employee table

```
INSERT INTO employee VALUES(NULL, 'Murray', 'Dale', '206-254-3456', 'Senior');
INSERT INTO employee VALUES(NULL, 'Murphy', 'Jerry', '585-545-8765', 'Master');
INSERT INTO employee VALUES(NULL, 'Fontaine', 'Joan', '585-545-8765', 'Junior');
INSERT INTO employee VALUES(NULL, 'Evanston', 'John', '206-254-2345', 'Junior');
INSERT INTO employee VALUES(NULL, 'Smith', 'Sam', '206-254-1234', 'Master');
```

(note: NULL is used as the first value because the EmployeeId will be automatically generated due to its data type definition: INTEGER PRIMARY KEY)

Practice Exercise

4. Display all records from the employee table in ascending order by LastName;

```
SELECT * from employee order by LastName ;
```

5. Select records from the employee table where ExperienceLevel is equal to Master

```
SELECT FROM employee WHERE ExperienceLevel = 'Master ';
```

6. Delete records with the value of ‘Junior’ in the ExperienceLevel column:

```
DELETE FROM employee WHERE ExperienceLevel ='Junior';
```

Note: If you do not provide a criteria (WHERE ….), and just type DELETE FROM employee, ALL RECORDS WILL BE DELETED