CRUD Databases

(Create, Read, Update and Delete)

Part 4 - Traveler's Database using Web SQL and SQLite

1 Introduction

A demonstration of **CRUD** database programming, based on using **Web SQL with SQLite** as the database format. CRUD stands for **create, read, update and delete**. These are the four basic operations every database must perform.

Web SQL is an API for managing databases. [74] It uses a version of **SQL** (**Structured Query Language**). Most implementations of Web SQL use **SQLite** [1] as the underlying technology. [32]

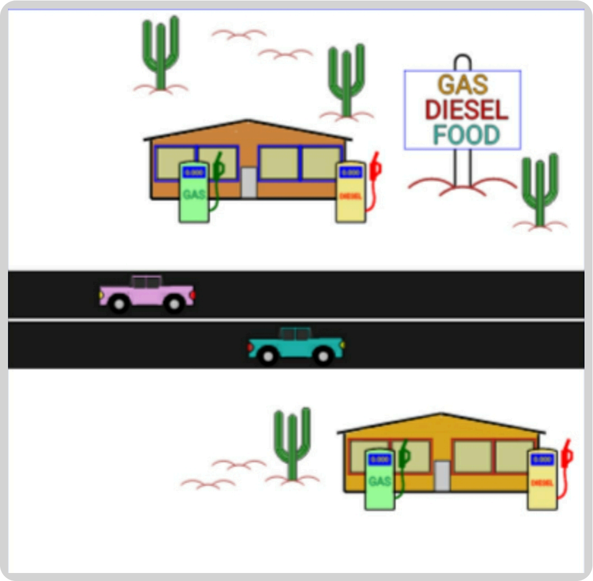

Figure 1: Travel on a long road trip.

Scenario - You are driving through the desert on a long road trip. Gas stations are few and far between, so it's important to know where you can get fuel so you don't run out. You also need to know where you can find food and a refreshing beverage. Calculate the distance to these things relative to your location. The data is **dynamically displayed** by querying the database using SQL.

When a query occurs, the app searches the database and returns the database entries which are the closest based on the location indicated in the query. Web SQL allows us to embed the database in either a Web or mobile device app.

The database will store the following information:

- **name**
- **longtitude**
- **lattitude**
- **GAS** or **NO GAS**
- **DIESEL** or **NO DIESEL**
- **FOOD** or **NO FOOD**

At the beginning of the demo, and after every GPS update, the closest items in the database will be displayed.

2 Database Access

Create The database is ***read-only***, so we aren't going to create any new records.

Read We read the records any time there is a GPS update.

Update The database is ***read-only***, so we aren't going to update any records.

Delete The database is ***read-only***, so we aren't going to delete any records.

3 Command Line vs Browser

SQLite runs on **Android**, **iOS**, **Linux**, **Mac**, and **Windows**, as well as other operating systems. [98] It runs in the following browsers: **Android Browser** (versions 2.1 - 4.4.4, 108), **Chrome** (versions 4 - 111), **Chrome for Android** (version 108), **Edge** (versions 79 - 108), **Safari** (versions 3.1 - 12.1) and **Safari on iOS** (versions 3.2 - 12.5). [99] Various APIs are under development to provide **persistent storage**. [100]

On **Windows** it is possible to open a command line shell for SQLite. [11] You can then type SQL commands similar to the way you use **phpMyAdmin** [49] with MySQL on a **LAMP** (**Linux**, **Apache**, **MySQL**, **PHP**) server. The app for the command line interface is called **sqlite3**.

The demo was tested under **Android™** and **Windows** using the **Chrome** browser. It was also tested on **Windows** using the **sqlite3** command line interface.

Since there are several versions of SQLite in use, it is possible that a particular command might not work during the demo. When that happens, an alternate conmand that works will be shown when possible.

The demo is Copyright © 2022 - 2024. All rights reserved.

4 Web SQL and SQLite

Web SQL is an API for managing databases. [74] It uses a version of **SQL** (**Structured Query Language**). Most implementations of Web SQL use **SQLite** [1] as the underlying technology. [32] Web SQL has been deprecated from the **HTML5** specification, but it is still widely in use. [3] [24] [32] **Although it has been deprecated, the developers of SQLite have committed to support the product through the year 2050**. [4] The U.S. **Library Of Congress** [5] has recommended SQLite as a storage format for the **preservation of digital content**. [4] SQLite is used in the **Android™**, **iOS**, **Mac**, and **Windows10** operating systems. It is found in the **Chrome**, **Firefox** and **Safari** web browsers. Some of the applications which use SQLite are **iTunes**, **PHP**, **Python** and **Skype**. [3] Some of the companies which use SQLite are **Adobe**, **Airbus**, **Apple**, **Facebook**, **Google**, and **Microsoft**. [8] All of the code and documentation in SQLite is in the **public domain**. It is **free to use for any commercial or non-commercial purpose**. The code in SQLite was written from scratch, and is **free of licensed code from other projects**. [7] SQLite may be **downloaded** [10] from

https://www.sqlite.org/download.html .

Downloads are available for the **source code** (multiple formats), **documentation**, and **binaries** for Android™, Linux, Mac OS X (x86), .NET, WebAssembly / JavaScript and Windows. [10] A **quick start** guide is available. [9]

5 When is SQLite a good choice?

SQLite is an **embedded database**, so it can run within your applications. [55] [57] It is a good choice for embedded platforms.[42]

SQLite **does not include authentication** by default, but it is possible to add this. When authentication has been activated, a new table named **sqlite_user** will be present. The sqlite_user table is normally **inaccessible** (unreadable and unwriteable) to non-admin users and is **read-only** for admin users. [58]

6 SQLite vs Other SQL Databases

SQLite	MySQL	SQL Server	PostgreSQL
no server	server	server	server
embedded database	client / server	client / server	client / server
Fast.	Fast.	Fast.	Fast.
Library: 250 - 500 KB single file. cross-platform.	Server: around 600 MB.	Server: 512 MB minimum.	Server: more than 200 MB.
Dynamic typing Supports: Blob, Integer, Null, Text, Real.	Static typing Supports: Tinyint, Smallint, Mediumint, Int, Bigint, Double, Float, Real, Decimal, Double precision, Numeric, Timestamp, Date, Datetime, Char, Varchar, Year, Tinytext, Tinyblob, Blob, Text, MediumBlob, MediumText, Enum, Set, Longblob, Longtext	Static typing Supports: bigint, numeric, bit, smallint, decimal, smallmoney, int, tinyint, money, float, real, date, datetimeoffset, datetime2, smalldatetime, datetime, time, char, varchar, text, nchar, nvarchar, ntext, binary, varbinary, image, cursor, rowversion, hiearchyid, uniqueidentifier, sql_variant, xml, Spatial Geometry Types, Spatial Geography Types, table	Static typing Supports: bigint, bigserial, double precision, integer, real, smallint, smallserial, serial, character, varchar, text, date, interval, time, time without time zone, time with time zone, timestamp, timestamp without time zone, timestamp with time zone, box, circle, line, lseg, path, point, polygon, cidr, inet, macaddr, bit, bit varying, tsquery, tsvector, json, jsonb, boolean, bytea, money, pg_lsn, txid_snapshot, uuid, xml
Supports "if exists" in drop statements.	Supports "if exists" in drop statements.	Supports "if exists" in drop statements. **(SQL Server 2016 and higher)**	Supports "if exists" in drop statements.
Does not support TOP.	**Does not support TOP.**	Supports TOP.	**Does not support TOP.**
Supports LIMIT.	Supports LIMIT.	Supports LIMIT.	Supports LIMIT.
Does not support right or full outer joins.	**Does not support full outer joins.** Supports right join.	Supports right or full outer joins.	Supports right or full outer joins.
Supports cross joins, inner joins, and left outer joins.	Supports cross joins, inner joins, and left outer joins.	Supports cross joins, inner joins, and left outer joins.	Supports cross joins, inner joins, and left outer joins.
No configurations.	Requires configuration.	Requires configuration.	Requires configuration.
Does not support simultaneous multiple users.	Supports simultaneous multiple users.	Supports simultaneous multiple users.	Supports simultaneous multiple users.
Authentication not included, may be added.	Includes authentication (username, password, and SSH).	Includes authentication.	Includes authentication, many security features.

Figure 2: Comparison of SQLite with other SQL databases. Derived from [2] [55] [56] [57] [58] [61] [62] [63] [64] [65] [66] [69] [70] [71] [72] [73] [75] [76] [77] [78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] [90].

7 Open a File

On **Windows** it is possible to open a command line shell for SQLite. [11] You can then type SQL commands similar to the way you use **phpMyAdmin** [49] with MySQL on a **LAMP** (**Linux**, **Apache**, **MySQL**, **PHP**) server. The app for the command line interface is called **sqlite3**. So if you wanted to open a database in the command line interface, you would first start sqlite3, then you would type the following at the sqlite> prompt

```
.open _path_\_database_name_.db
```

If you open a database file that does not exist, sqlite3 will create the file. **Using JavaScript in your browser**, we would open a database like this:

```
var db = openDatabase( dbID, dbVersion, dbName, dbSize );
```

8 A Typical Transaction

Once we have opened a database, we perform transactions. Transactions in JavaScript basically follow the format of the example shown here. **Using JavaScript in your browser**, let's create a table in the database we just opened.

JavaScript:

```
db.transaction(function (tx) {
// Perform database transaction
// (tx = transaction)

// Create a table named DATA
tx.executeSql('CREATE TABLE IF NOT EXISTS DATA (id unique, text)', [],
function (tx, results) {
// command successful

    // Process the results

// end command successful
},
commandFailed

// end tx.executeSql
);

// end db.transaction
});


function commandFailed(tx, error) {

    // Process the error

// end function commandFailed
}
```

sqlite3:

```
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>

sqlite> CREATE TABLE IF NOT EXISTS DATA (id unique, text);
sqlite>
```

9 Database Tables

After we create the table, we can query the database for the names of the tables. [2] [13] [41]

JavaScript:

```
SELECT name FROM sqlite_schema WHERE type="table" ORDER BY name

Rows returned: 2

Row 0     DATA
Row 1     WebKitDatabaseInfoTable

End of data returned by query
Command results: success.
```

sqlite3:

```
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>

sqlite> CREATE TABLE IF NOT EXISTS DATA (id unique, text);
sqlite>

sqlite> .tables
DATA

sqlite> SELECT name FROM sqlite_master WHERE type="table" ORDER BY name;
DATA
sqlite>
```

10 Database Indexes

We can query the database for the names of the indexes.

JavaScript:

```
SELECT name FROM sqlite_master WHERE type="index" ORDER BY name

Rows returned: 2

Row 0     sqlite_autoindex_DATA_1
Row 1     sqlite_autoindex___WebKitDatabaseInfoTable___1

End of data returned by query
Command results: success.
```

sqlite3:

```
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>

sqlite> .indexes

sqlite_autoindex_DATA_1
sqlite>
```

11 View the original SQL

We can use the .SCHEMA command in sqlite3 to recall the SQL that was used to create the table. SCHEMA is preceded by a period (.). sqlite3 has many such commands, but they do not appear to work in JavaScript.

sqlite3:

```
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>

sqlite> CREATE TABLE DATA (id unique, text);


sqlite> .schema
CREATE TABLE DATA (id unique, text);
sqlite>
```

Another way [12] [14] to view the SQL that created the table is to issue the command

```
SELECT sql FROM sqlite_master WHERE tbl_name = ' _table_name_';
```

We obtain the following results:

JavaScript:

```
SELECT sql FROM sqlite_master WHERE tbl_name = "DATA"

Rows returned: 2

Row 0    CREATE TABLE DATA (id unique, text)
Row 1    null

End of data returned by query
Command results: success.
```

sqlite3:

```
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>

sqlite> select sql from sqlite_master where tbl_name="DATA";
Error: no such table: sqlite_master


sqlite> select sql from sqlite_master where type="table";
CREATE TABLE DATA (id unique, text)
sqlite>
```

12 Insert Entries

We can populate the table we just created using the following commands:

JavaScript:

```
INSERT INTO DATA (id, text) VALUES (0, "text to insert 0")
Command results: success.


INSERT INTO DATA (id, text) VALUES (1, "text to insert 1")
Command results: success.


INSERT INTO DATA (id, text) VALUES (2, "text to insert 2")
Command results: success.
```

```
INSERT INTO DATA (id, text) VALUES (3, "text to insert 3")
Command results: success.

INSERT INTO DATA (id, text) VALUES (4, "text to insert 4")
Command results: success.
```

We can now query the table with the following results:

JavaScript:

```
SELECT * FROM DATA

Rows returned: 5

Row 0     text to insert 0
Row 1     text to insert 1
Row 2     text to insert 2
Row 3     text to insert 3
Row 4     text to insert 4

End of data returned by query
Command results: success.
```

sqlite3:

```
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>

sqlite> SELECT * FROM DATA;
0|text to insert 0
1|text to insert 1
2|text to insert 2
3|text to insert 3
4|text to insert 4
sqlite>
```

13 Duplicate Entries

If we attempt to enter a duplicate entry, we get the following results:

JavaScript:

```
INSERT INTO DATA (id, text) VALUES (3, "text to insert 3")
Command results: failed.
Error message:
could not execute statement due to a constraint failure (19 UNIQUE constraint failed: DATA.id)
```

sqlite3:

```
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>

sqlite> INSERT INTO DATA (id, text) VALUES (3, "line of text 3");
Error: UNIQUE constraint failed: DATA.id
sqlite>
```

14 Dynamic vs Static Typing

SQLite uses dynamic typing while other versions of SQL use static typing. [2] [56] Data of any type may be inserted into any column. The only exception is that columns of type INTEGER PRIMARY KEY may only contain an integer. [2] For example, let's use JavaScript to insert some data into the table using a character for the index instead of a number.

JavaScript:

```
INSERT INTO DATA (id, text) VALUES (z, "text to insert z")
Command results: failed.
Error message:
could not prepare statement (1 no such column: z)
```

If we enclose z within quotes so it represents a string, we can now successfully insert the value into the table.

JavaScript:

```
INSERT INTO DATA (id, text) VALUES ("z", "text to insert z")
Command results: success.
```

We can now query the table with the following results:

JavaScript:

```
SELECT * FROM DATA

Rows returned: 6

Row 0     text to insert 0
Row 1     text to insert 1
Row 2     text to insert 2
Row 3     text to insert 3
Row 4     text to insert 4
Row 5     text to insert z

End of data returned by query
Command results: success.
```

*** NOTE ***
In JavaScript the index appears as a number instead of a character. In the Windows command line interface (sqlite3), the index will appear as z.

sqlite3:

```
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>

sqlite> INSERT INTO DATA (id, text) VALUES (z, "line of text z");
Error: no such column: z

sqlite> INSERT INTO DATA (id, text) VALUES ("z", "line of text z");

sqlite> SELECT * FROM DATA;
0|text to insert 0
1|text to insert 1
2|text to insert 2
3|text to insert 3
4|text to insert 4
z|text to insert z
sqlite>
```

15 Foreign Keys

Support for SQL foreign key constraints was added in SQLite version 3.6.19 (2009-10-14). If SQLite was compiled with SQLITE_OMIT_FOREIGN_KEY or SQLITE_OMIT_TRIGGER defined, then foreign keys will not be supported. Even if SQLite was compiled with foreign key support, foreign keys are not enabled by default. [91] To determine if foreign keys are enabled or to enable foreign key support, it is necessary to use a PRAGMA statement. [92]

To determine if foreign keys are currently enabled:

```
sqlite> PRAGMA foreign_keys;
0
```

(0 = disabled, 1 = enabled)

If this command returns no data, then SQLite does not support foreign keys. Either it is an older version, or it was compiled without foreign key support. [91]

To enable foreign key support:

```
sqlite> PRAGMA foreign_keys = ON;
```

(PRAGMA foreign_keys=1;)

```
sqlite> PRAGMA foreign_keys;
1
```

To disable foreign key support:

```
sqlite> PRAGMA foreign_keys = OFF;
```

(PRAGMA foreign_keys=0;)

```
sqlite> PRAGMA foreign_keys;
0
```

16 Database Without Foreign Keys

JavaScript:

```
PRAGMA foreign_keys
Command results: failed.
Error message:
could not prepare statement (23 not authorized)
```

```
PRAGMA foreign_keys=1
Command results: failed.
Error message:
could not prepare statement (23 not authorized)
```

```
PRAGMA foreign_keys=ON
Command results: failed.
Error message:
could not prepare statement (23 not authorized)
```

It looks like foreign keys are not supported in this particular browser. We'll have to design the database so that we avoid using them. Let's create a table using the data from our travelers database.

JavaScript:

```
CREATE TABLE IF NOT EXISTS restStop ( restStop_id unique,
restStopName VARCHAR(28),
longtitude INTEGER,
lattitude INTEGER,
gas VARCHAR(6),
diesel VARCHAR(9),
food VARCHAR(8),
PRIMARY KEY(restStop_id) )
Command results: success.
```

17 Insert Entries

We can populate the table we just created using the following commands:

JavaScript:

```
INSERT INTO restStop VALUES (0, "Petrol King #1", 450, 900, "GAS", "DIESEL", "FOOD")
Command results: success.

INSERT INTO restStop VALUES (1, "Oil City #2", 457, 900, "NO GAS", "NO DIESEL", "NO FOOD")
Command results: success.

INSERT INTO restStop VALUES (2, "Burger Stop #3", 462, 900, "NO GAS", "NO DIESEL", "FOOD")
Command results: success.

INSERT INTO restStop VALUES (3, "Fill 'Er Up #4", 462, 900, "GAS", "DIESEL", "NO FOOD")
Command results: success.

INSERT INTO restStop VALUES (4, "Taco Town #5", 467, 900, "NO GAS", "NO DIESEL", "FOOD")
Command results: success.

INSERT INTO restStop VALUES (5, "Oil City #6", 467, 900, "GAS", "DIESEL", "NO FOOD")
Command results: success.

INSERT INTO restStop VALUES (6, "Petrol King #7", 475, 900, "GAS", "DIESEL", "FOOD")
Command results: success.

INSERT INTO restStop VALUES (7, "Fill 'Er Up #8", 482, 900, "GAS", "DIESEL", "FOOD")
Command results: success.

INSERT INTO restStop VALUES (8, "Burger Stop #9", 487, 900, "NO GAS", "NO DIESEL", "FOOD")
Command results: success.

INSERT INTO restStop VALUES (9, "Petrol King #10", 487, 900, "GAS", "DIESEL", "FOOD")
Command results: success.

INSERT INTO restStop VALUES (10, "Oil City #11", 492, 900, "NO GAS", "NO DIESEL", "NO FOOD")
Command results: success.

INSERT INTO restStop VALUES (11, "Burger Stop #12", 492, 900, "NO GAS", "NO DIESEL", "FOOD")
Command results: success.

INSERT INTO restStop VALUES (12, "Fill 'Er Up #13", 492, 900, "GAS", "DIESEL", "FOOD")
Command results: success.

INSERT INTO restStop VALUES (13, "Taco Town #14", 502, 900, "NO GAS", "NO DIESEL", "FOOD")
Command results: success.

INSERT INTO restStop VALUES (14, "Oil City #15", 502, 900, "GAS", "DIESEL", "FOOD")
Command results: success.

INSERT INTO restStop VALUES (15, "Petrol King #16", 510, 900, "GAS", "DIESEL", "NO FOOD")
Command results: success.

INSERT INTO restStop VALUES (16, "Burger Stop #17", 518, 900, "NO GAS", "NO DIESEL", "FOOD")
Command results: success.

INSERT INTO restStop VALUES (17, "Fill 'Er Up #18", 518, 900, "GAS", "DIESEL", "NO FOOD")
Command results: success.
```

We can now query the contents of the table we just created.

JavaScript:

```
SELECT * FROM restStop

Rows returned: 18

Row 0 | 0 | Petrol King #1 | 450 | 900 | GAS | DIESEL | FOOD |
Row 1 | 1 | Oil City #2 | 457 | 900 | NO GAS | NO DIESEL | NO FOOD |
Row 2 | 2 | Burger Stop #3 | 462 | 900 | NO GAS | NO DIESEL | FOOD |
Row 3 | 3 | Fill 'Er Up #4 | 462 | 900 | GAS | DIESEL | NO FOOD |
Row 4 | 4 | Taco Town #5 | 467 | 900 | NO GAS | NO DIESEL | FOOD |
Row 5 | 5 | Oil City #6 | 467 | 900 | GAS | DIESEL | NO FOOD |
Row 6 | 6 | Petrol King #7 | 475 | 900 | GAS | DIESEL | FOOD |
Row 7 | 7 | Fill 'Er Up #8 | 482 | 900 | GAS | DIESEL | FOOD |
Row 8 | 8 | Burger Stop #9 | 487 | 900 | NO GAS | NO DIESEL | FOOD |
Row 9 | 9 | Petrol King #10 | 487 | 900 | GAS | DIESEL | FOOD |
Row 10 | 10 | Oil City #11 | 492 | 900 | NO GAS | NO DIESEL | NO FOOD |
Row 11 | 11 | Burger Stop #12 | 492 | 900 | NO GAS | NO DIESEL | FOOD |
Row 12 | 12 | Fill 'Er Up #13 | 492 | 900 | GAS | DIESEL | FOOD |
Row 13 | 13 | Taco Town #14 | 502 | 900 | NO GAS | NO DIESEL | FOOD |
Row 14 | 14 | Oil City #15 | 502 | 900 | GAS | DIESEL | FOOD |
Row 15 | 15 | Petrol King #16 | 510 | 900 | GAS | DIESEL | NO FOOD |
Row 16 | 16 | Burger Stop #17 | 518 | 900 | NO GAS | NO DIESEL | FOOD |
Row 17 | 17 | Fill 'Er Up #18 | 518 | 900 | GAS | DIESEL | NO FOOD |

End of data returned by query
Command results: success.
```

We only want to see the entries which fall within our search window. Let's set the search window for entries where the longtitude is between 467 and 492.

JavaScript:

```
SELECT * FROM restStop WHERE longtitude BETWEEN 467 AND 492

Rows returned: 9

Row 0 | 4 | Taco Town #5 | 467 | 900 | NO GAS | NO DIESEL | FOOD |
Row 1 | 5 | Oil City #6 | 467 | 900 | GAS | DIESEL | NO FOOD |
Row 2 | 6 | Petrol King #7 | 475 | 900 | GAS | DIESEL | FOOD |
Row 3 | 7 | Fill 'Er Up #8 | 482 | 900 | GAS | DIESEL | FOOD |
Row 4 | 8 | Burger Stop #9 | 487 | 900 | NO GAS | NO DIESEL | FOOD |
Row 5 | 9 | Petrol King #10 | 487 | 900 | GAS | DIESEL | FOOD |
Row 6 | 10 | Oil City #11 | 492 | 900 | NO GAS | NO DIESEL | NO FOOD |
Row 7 | 11 | Burger Stop #12 | 492 | 900 | NO GAS | NO DIESEL | FOOD |
Row 8 | 12 | Fill 'Er Up #13 | 492 | 900 | GAS | DIESEL | FOOD |

End of data returned by query
Command results: success.
```

Now that we have the entries within our search window, the application can use the data to update the display.

18 Database Using Foreign Keys

sqlite3:

```
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
sqlite> PRAGMA foreign_keys;
0

sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
COMMIT;
sqlite>

sqlite> PRAGMA foreign_keys=ON;
sqlite>

sqlite> PRAGMA foreign_keys;
1

sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
COMMIT;
sqlite>

sqlite> PRAGMA foreign_keys=OFF;
sqlite>

sqlite> PRAGMA foreign_keys;
0

sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
COMMIT;
sqlite>

sqlite> PRAGMA foreign_keys=ON;
sqlite>

sqlite> PRAGMA foreign_keys;
1

sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
COMMIT;
sqlite>
```

Let's create a table using foreign keys. We'll use the data from our travelers database. First, we enable foreign keys.

```
sqlite> PRAGMA foreign_keys=1;
sqlite>

sqlite> PRAGMA foreign_keys;
1

sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
COMMIT;
sqlite>
```

Then we create and populate the tables which will be referenced.

We need to reference the following items:

- **The NAME of the rest stop.**
- **Whether GAS is available.**
- **Whether DIESEL is available.**
- **Whether FOOD is available.**

We create the table for the individual rest stops.

```
sqlite> CREATE TABLE REST_STOP ( restStop_id INTEGER AUTO_INCREMENT,
restStopName VARCHAR(28),
longtitude INTEGER,
lattitude INTEGER,
gas INTEGER,
diesel INTEGER,
food INTEGER,
PRIMARY KEY(restStop_id),
FOREIGN KEY (gas) REFERENCES GAS (gas_id)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (diesel) REFERENCES DIESEL (diesel_id)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (food) REFERENCES FOOD (food_id)
ON DELETE CASCADE ON UPDATE CASCADE
);
sqlite>
```

We insert entries into the table.

```
sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(0,'Petrol King #1',450,900,0,0,0);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(1,'Oil City #2',457,900,1,1,1);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(2,'Burger Stop #3',462,900,1,1,0);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(3,'Fill ''Er Up #4',462,900,0,0,1);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(4,'Taco Town #5',467,900,1,1,0);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(5,'Oil City #6',467,900,0,0,1);
sqlite>
sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(6,'Petrol King #7',475,900,0,0,0);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(7,'Fill ''Er Up #8',482,900,0,0,0);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(8,'Burger Stop #9',487,900,1,1,0);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(9,'Petrol King #10',487,900,0,0,0);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(10,'Oil City #11',492,900,1,1,1);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(11,'Burger Stop #12',492,900,1,1,0);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(12,'Fill ''Er Up #13',492,900,0,0,0);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(13,'Taco Town #14',502,900,1,1,0);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(14,'Oil City #15',502,900,0,0,0);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(15,'Petrol King #16',510,900,0,0,1);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(16,'Burger Stop #17',518,900,1,1,0);
sqlite>

sqlite> INSERT INTO REST_STOP (restStop_id, restStopName, longtitude, lattitude, gas, diesel,
food)
...> VALUES(17,'Fill ''Er Up #18',518,900,0,0,1);
sqlite>
```

Now that the table is populated, let's query the data.

```
sqlite> select * from REST_STOP;
0|Petrol King #1|450|900|0|0|0
1|Oil City #2|457|900|1|1|1
2|Burger Stop #3|462|900|1|1|0
3|Fill ''Er Up #4|462|900|0|0|1
4|Taco Town #5|467|900|1|1|0
5|Oil City #6|467|900|0|0|1
6|Petrol King #7|475|900|0|0|0
7|Fill ''Er Up #8|482|900|0|0|0
8|Burger Stop #9|487|900|1|1|0
9|Petrol King #10|487|900|0|0|0
10|Oil City #11|492|900|1|1|1
11|Burger Stop #12|492|900|1|1|0
12|Fill ''Er Up #13|492|900|0|0|0
13|Taco Town #14|502|900|1|1|0
14|Oil City #15|502|900|0|0|0
15|Petrol King #16|510|900|0|0|1
16|Burger Stop #17|518|900|1|1|0
17|Fill ''Er Up #18|518|900|0|0|1
sqlite>
```

19 Formatting The Output

There are some dot commands which are used to format the output of the SELECT command. (A list of all the commands is available by typing .help at the sqlite3 prompt.) These commands are:

`.header on`
(turns on the headers)

`.mode column`
(display the output in columns using left alignment)

`.width num num`
(sets the column widths for the columns)

For this view we set the width as follows:

```
.width 12 12 12 12 12 12 12
```

sqlite3:

```
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

```
sqlite> .header on
sqlite>
sqlite> .mode column
sqlite>
```

```
sqlite> .width 12 12 12 12 12 12 12
sqlite>
```

```
sqlite> .show
echo: off
eqp: off
explain: auto
headers: on
mode: column
nullvalue: ""
output: stdout
colseparator: "|"
rowseparator: "\n"
stats: off
width: 12 12 12 12 12 12 12
filename: travelers.db
sqlite>
```

```
sqlite> sqlite> select * from rest_stop;
restStop\_id  restStopName  longtitude  lattitude  gas  diesel  food
------------ ------------ ------------ ------------ ------------ ------------ ------------
0            Petrol King #1      450        900        0       0       0
1            Oil City #2         457        900        1       1       1
2            Burger Stop #3      462        900        1       1       0
3            Fill ''Er Up #4     462        900        0       0       1
4            Taco Town #5        467        900        1       1       0
5            Oil City #6         467        900        0       0       1
6            Petrol King #7      475        900        0       0       0
7            Fill ''Er Up #8     482        900        0       0       0
8            Burger Stop #9      487        900        1       1       0
9            Petrol King #10     487        900        0       0       0
10           Oil City #11        492        900        1       1       1
11           Burger Stop #12     492        900        1       1       0
12           Fill ''Er Up #13    492        900        0       0       0
13           Taco Town #14       502        900        1       1       0
14           Oil City #15        502        900        0       0       0
15           Petrol King #16     510        900        0       0       1
16           Burger Stop #17     518        900        1       1       0
17           Fill ''Er Up #18    518        900        0       0       1
sqlite>
```

We're using foreign keys, so the data is not very clear. Let's correct that.

Let's query the database where the foreign indexes are converted to their actual values. We'll also only want to return the results which are within our search window. Let's set the search window for entries where the longtitude is between 467 and 492. [103]

20 Read Database With Foreign Keys

```
sqlite> select restStop_id, restStopName, longtitude, lattitude, c.status, d.status, e.status from
REST_STOP a INNER JOIN GAS c,DIESEL d,FOOD e where longtitude between 467 and 492 GROUP BY
restStopName ORDER BY restStop_id;
restStop_id  restStopName  longtitude  lattitude  gas  diesel  food
------------ ------------ ------------ ------------ ------------ ------------ ------------
4          Taco Town #5          467       900       GAS        DIESEL       FOOD
5          Oil City #6           467       900       GAS        DIESEL       FOOD
6          Petrol King #7        475       900       GAS        DIESEL       FOOD
7          Fill ''Er Up #8       482       900       GAS        DIESEL       FOOD
8          Burger Stop #9        487       900       GAS        DIESEL       FOOD
9          Petrol King #10       487       900       GAS        DIESEL       FOOD
10         Oil City #11          492       900       GAS        DIESEL       FOOD
11         Burger Stop #12       492       900       GAS        DIESEL       FOOD
12         Fill ''Er Up #13      492       900       GAS        DIESEL       FOOD
sqlite>
```

We're using foreign keys, so why aren't they showing up correctly for the GAS, DIESEL and FOOD?

It's not clear if this is because the SQL query is not formatted correctly or if the version of sqlite3 we are using does not fully support foreign keys.

Let's try SQLite on a different device. We'll access SQLite using JavaScript and we'll make a slight modification to the SELECT statement.

JavaScript:

```
SELECT SQLITE_VERSION()
Command results: The SQLite version number is 3.7.7.1
```

```
PRAGMA foreign_keys
Command results: *** Foreign keys are not enabled. ***
```

Enable foreign keys.

```
PRAGMA foreign_keys=1
Command results: success.
```

```
PRAGMA foreign_keys
Command results: *** Foreign keys are enabled. ***
```

Query the database using the modified SELECT statement.

```
SELECT a.restStop_id, a.restStopName, a.longtitude, a.lattitude,
c.gas_status, d.diesel_status, e.food_status FROM REST_STOP a
JOIN GAS c
ON a.gas = c.gas_id
JOIN DIESEL d
ON a.diesel = d.diesel_id
JOIN FOOD e
ON a.food = e.food_id
GROUP BY
restStopName ORDER BY restStop_id;
Rows returned: 18
Data returned:
  row   id    name             longtitude   lattitude    gas       diesel         food
   0     0    Petrol King #1     450          900       GAS       DIESEL         FOOD
   1     1    Oil City #2        457          900       NO GAS    NO DIESEL      NO FOOD
   2     2    Burger Stop #3     462          900       NO GAS    NO DIESEL      FOOD
   3     3    Fill 'Er Up #4     462          900       GAS       DIESEL         NO FOOD
   4     4    Taco Town #5       467          900       NO GAS    NO DIESEL      FOOD
   5     5    Oil City #6        467          900       GAS       DIESEL         NO FOOD
   6     6    Petrol King #7     475          900       GAS       DIESEL         FOOD
   7     7    Fill 'Er Up #8     482          900       GAS       DIESEL         FOOD
   8     8    Burger Stop #9     487          900       NO GAS    NO DIESEL      FOOD
   9     9    Petrol King #10    487          900       GAS       DIESEL         FOOD
   10    10   Oil City #11       492          900       NO GAS    NO DIESEL      NO FOOD
   11    11   Burger Stop #12    492          900       NO GAS    NO DIESEL      FOOD
   12    12   Fill 'Er Up #13    492          900       GAS       DIESEL         FOOD
   13    13   Taco Town #14      502          900       NO GAS    NO DIESEL      FOOD
   14    14   Oil City #15       502          900       GAS       DIESEL         FOOD
   15    15   Petrol King #16    510          900       GAS       DIESEL         NO FOOD
   16    16   Burger Stop #17    518          900       NO GAS    NO DIESEL      FOOD
   17    17   Fill 'Er Up #18    518          900       GAS       DIESEL         NO FOOD

End of data returned by query
Command results: success.
```

The foreign keys are now working correctly. We only want to return the results which are within our search window. Let's set the search window for entries where the longtitude is between 490 and 510. [103]

```
SELECT a.restStop_id, a.restStopName, a.longtitude, a.lattitude,
c.gas_status, d.diesel_status, e.food_status FROM REST_STOP a
JOIN GAS c
ON a.gas = c.gas_id
JOIN DIESEL d
ON a.diesel = d.diesel_id
JOIN FOOD e
ON a.food = e.food_id
WHERE longtitude BETWEEN 490 AND 510 GROUP BY
restStopName ORDER BY restStop_id;
Rows returned: 6
Data returned:
  row   id    name             longtitude   lattitude    gas       diesel         food
   0    10    Oil City #11       492          900       NO GAS    NO DIESEL      NO FOOD
   1    11    Burger Stop #12    492          900       NO GAS    NO DIESEL      FOOD
   2    12    Fill 'Er Up #13    492          900       GAS       DIESEL         FOOD
   3    13    Taco Town #14      502          900       NO GAS    NO DIESEL      FOOD
   4    14    Oil City #15       502          900       GAS       DIESEL         FOOD
   5    15    Petrol King #16    510          900       GAS       DIESEL         NO FOOD

End of data returned by query
Command results: success.
```

Now that we have the entries within our search window, the application can use the data to update the display.

This page intentionally left blank.

References

[1] “SQLite Home Page.” *SQLite.org*, 27-Dec-2022. [Online]. Available: https://www.sqlite.org/index.html. [Accessed: 03-Jan-2023].

[2] “SQLite Frequently Asked Questions.” *SQLite.org*, 27-Jul-2021. [Online]. Available: https://www.sqlite.org/faq.html. [Accessed: 03-Jan-2023].

[3] “Most Widely Deployed SQL Database Engine.” *SQLite.org*, 03-Jun-2021. [Online]. Available: https://www.sqlite.org/mostdeployed.html. [Accessed: 03-Jan-2023].

[4] “Long Term Support.” *SQLite.org*, 01-Dec-2020. [Online]. Available: https://www.sqlite.org/lts.html. [Accessed: 03-Jan-2023].

[5] “Home | Library of Congress,” *Library of Congress*. [Online]. Available: https://www.loc.gov/. [Accessed: 03-Jan-2023].

[6] “Alphabetical List Of SQLite Documents.” *SQLite.org*. [Online]. Available: https://www.sqlite.org/doclist.html. [Accessed: 03-Jan-2023].

[7] “SQLite Copyright.” *SQLite.org*, 10-Nov-2021. [Online]. Available: https://www.sqlite.org/copyright.html. [Accessed: 03-Jan-2023].

[8] “Well-Known Users Of SQLite.” *SQLite.org*, 18-Jun-2022. [Online]. Available: https://www.sqlite.org/famous.html. [Accessed: 03-Jan-2023].

[9] “SQLite In 5 Minutes Or Less.” *SQLite.org*, 04-Apr-2016. [Online]. Available: https://www.sqlite.org/quickstart.html. [Accessed: 03-Jan-2023].

[10] “SQLite Download Page.” *SQLite.org*. [Online]. Available: https://www.sqlite.org/download.html. [Accessed: 03-Jan-2023].

[11] “Command Line Shell For SQLite.” *SQLite.org*, 27-Dec-2022. [Online]. Available: https://www.sqlite.org/cli.html. [Accessed: 03-Jan-2023].

[12] “The Schema Table.” *SQLite.org*, 15-Feb-2022. [Online]. Available: https://www.sqlite.org/schematab.html. [Accessed: 03-Jan-2023].

[13] “SQLite Show Tables | Basic Syntax and the Different Examples,” *EDUCBA*, 05-May-2021. [Online]. Available: https://www.educba.com/sqlite-show-tables/. [Accessed: 03-Jan-2023].

[14] “4 Ways to Get Information about a Table’s Structure in SQLite.” *database.guide*, 30-May-2020. [Online]. Available: https://database.guide/4-ways-to-get-information-about-a-tables-structure-in-sqlite/. [Accessed: 03-Jan-2023].

[15] R. Peterson, “SQLite Database Tutorial for Beginners: Learn with Examples,” *Guru99*, 24-Dec-2022. [Online]. Available: https://www.guru99.com/sqlite-tutorial.html. [Accessed: 03-Jan-2023].

[16] J. Zhao, “Html5 Web SQLite Database Example,” *dev2qa.com*, 04-Apr-2022. [Online]. Available: https://www.dev2qa.com/html5-web-sqlite-database-example/. [Accessed: 03-Jan-2023].

[17] R. Sharp, “Introducing Web SQL Databases | HTML5 Doctor.” *HTML5 Doctor*, 24-Feb-2010. [Online]. Available: http://html5doctor.com/introducing-web-sql-databases/. [Accessed: 03-Jan-2023].

[18] S. Bose, “Working SqLite Javascript,” *Gist*. [Online]. Available: https://gist.github.com/siddharthabose03/8450242. [Accessed: 03-Jan-2023].

[19] “Use Query Parameters | Reporting | DevExpress Documentation.” *DevExpress*, 28-Aug-2022. [Online]. Available: https://docs.devexpress.com/XtraReports/17387/detailed-guide-to-devexpress-reporting/bind-reports-to-data/sql-database/specify-query-parameters. [Accessed: 03-Jan-2023].

[20] “Web SQL API Tutorial - Zebra Technologies TechDocs.” *Zebra TechDocs*, 2020. [Online]. Available: https://techdocs.zebra.com/enterprise-browser/3-0/tutorial/websql/. [Accessed: 03-Jan-2023].

References (continued)

[21] T. Steiner, “Deprecating and removing Web SQL,” *Chrome Developers*, 13-Dec-2022. [Online]. Available: https://developer.chrome.com/blog/deprecating-web-sql/. [Accessed: 04-Jan-2023].

[22] K. Puls, “Pass Parameters to SQL Queries,” *Excelguru*, 28-Apr-2016. [Online]. Available: https://excelguru.ca/pass-parameters-to-sql-queries/. [Accessed: 04-Jan-2023].

[23] A. Ravikiran, “SQL Insert: The Best Way to Populate Database Tables [Updated],” *Simplilearn*, 27-Oct-2022. [Online]. Available: https://www.simplilearn.com/tutorials/sql-tutorial/sql-insert. [Accessed: 04-Jan-2023].

[24] “What is WEB SQL?,” *GeeksforGeeks*, 11-Oct-2022. [Online]. Available: https://www.geeksforgeeks.org/what-is-web-sql/. [Accessed: 04-Jan-2023].

[25] A. Choudhary, “Handling multiple records in Web Sql by recursive method,” *Oodles Technologies*, 22-Jun-2015. [Online]. Available: https://www.oodlestechnologies.com/blogs/handling-multiple-records-in-web-sql-by-recursive-method/. [Accessed: 04-Jan-2023].

[26] A. Choudhary, “Use of Web SQL in Phonegap,” *Oodles Technologies*, 01-Sep-2014. [Online]. Available: https://www.oodlestechnologies.com/blogs/Use%20of%20Web%20SQL%20in%20Phonegap/. [Accessed: 04-Jan-2023].

[27] S. Dixit, “Dev.Opera — Taking Your Web Apps Offline: A Tale of Web Storage, Application Cache and WebSQL.” *Opera Software AS*, 22-Mar-2011. [Online]. Available: https://dev.opera.com/articles/offline-web-apps/. [Accessed: 04-Jan-2023].

[28] M. Mansuriya, “WEBSQL, SQL at the client’s end,” *Medium*, 27-Jul-2020. [Online]. Available: https://madhavmansuriya40.medium.com/websql-sql-at-the-clients-end-e70579a0401f. [Accessed: 04-Jan-2023].

[29] “SQLite DROP TABLE Statement with Examples,” *SQLite Tutorial*. [Online]. Available: https://www.sqlitetutorial.net/sqlite-drop-table/. [Accessed: 04-Jan-2023].

[30] “Connecting To SQLite Database Using Node.js,” *SQLite Tutorial*. [Online]. Available: https://www.sqlitetutorial.net/sqlite-nodejs/connect/. [Accessed: 04-Jan-2023].

[31] “HTML5 - Web SQL Database.” *Tutorials Point*. [Online]. Available: https://www.tutorialspoint.com/html5/html5_web_sql.htm. [Accessed: 04-Jan-2023].

[32] “Web SQL by james-priest.” *github.io*. [Online]. Available: https://james-priest.github.io/100-days-of-code-log-r2/CH16-Offline1-WebSQL.html. [Accessed: 04-Jan-2023].

[33] “Web SQL Database | Tizen Docs.” *Tizen Project*, 2023. [Online]. Available: https://docs.tizen.org/application/web/guides/w3c/storage/websql/. [Accessed: 04-Jan-2023].

[34] S. Somani, “HTML 5 Web SQL Database.” *C# Corner*, 2023. [Online]. Available: https://www.c-sharpcorner.com/uploadfile/75a48f/html-5-web-sql-database/. [Accessed: 04-Jan-2023].

[35] “HTML5 - Web SQL Database.” *Tutorials Point*. [Online]. Available: https://www.tutorialspoint.com/html5/html5_web_sql.htm. [Accessed: 04-Jan-2023].

[36] S. Jaiswal, “What is Web SQL - javatpoint,” *Javatpoint*. [Online]. Available: https://www.javatpoint.com/what-is-web-sql. [Accessed: 04-Jan-2023].

[37] “SQLite Tutorial - An Easy Way to Master SQLite Fast,” *SQLite Tutorial*. [Online]. Available: https://www.sqlitetutorial.net/. [Accessed: 04-Jan-2023].

[38] “How To Download & Install SQLite Tools,” *SQLite Tutorial*. [Online]. Available: https://www.sqlitetutorial.net/download-install-sqlite/. [Accessed: 04-Jan-2023].

[39] “SQLite Sample Database And Its Diagram (in PDF format),” *SQLite Tutorial*. [Online]. Available: https://www.sqlitetutorial.net/sqlite-sample-database/. [Accessed: 04-Jan-2023].

[40] “SQLite Cheat Sheet,” *SQLite Tutorial*. [Online]. Available: https://www.sqlitetutorial.net/sqlite-cheat-sheet/. [Accessed: 04-Jan-2023].

References (continued)

[41] “SQLite Show Tables: Listing All Tables in a Database,” *SQLite Tutorial*, 2022. [Online]. Available: https://www.sqlitetutorial.net/sqlite-show-tables/. [Accessed: 05-Jan-2023].

[42] “Appropriate Uses For SQLite.” *SQLite.org*, 14-Dec-2022. [Online]. Available: https://www.sqlite.org/whentouse.html. [Accessed: Dec. 05-Jan-2023].

[43] 262588213843476, “Reset WebSQL database dropping every tables,” *Gist*. [Online]. Available: https://gist.github.com/partageit/d091039f1942baed910a3f54f491056c. [Accessed: Dec. 05-Jan-2023].

[44] N. Lawson, “Web SQL Database: In Memoriam,” *Read the Tea Leaves*, Apr. 26, 2014. [Online]. Available: https://nolanlawson.com/2014/04/26/web-sql-database-in-memoriam/. [Accessed: 05-Jan-2023].

[45] user2250152, “How to read metadata from Sqlite database,” *Database Administrators Stack Exchange*, 05-Apr-2017. [Online]. Available: https://dba.stackexchange.com/q/169246. [Accessed: 05-Jan-2023].

[46] “Blind WebSQL and Storage extraction for HTML5 Apps.” *Blueinfy's blog*, 2012. [Online]. Available: http://blog.blueinfy.com/2012/01/blind-websql-and-storage-extraction-for.html. [Accessed: 05-Jan-2023].

[47] “HTML5 - Web SQL Database.” *Tutorials Point*, 2022. [Online]. Available: https://www.tutorialspoint.com/html5/html5_web_sql.htm. [Accessed: 05-Jan-2023].

[48] “Hosting SQLite databases on Github Pages - (or any static file hoster) - phiresky’s blog.” *phiresky's blog*, 03-May-2021. [Online]. Available: https://phiresky.github.io/blog/2021/hosting-sqlite-databases-on-github-pages/. [Accessed: 05-Jan-2023].

[49] phpMyAdmin contributors, “phpMyAdmin,” *phpMyAdmin*, 2023. [Online]. Available: https://www.phpmyadmin.net/. [Accessed: 05-Jan-2023].

[50] “SQL Introduction.” *W3Schools*, 2023. [Online]. Available: https://www.w3schools.com/sql/sql_intro.asp. [Accessed: 05-Jan-2023].

[51] P. Loshin and J. Sirkin, “What is Structured Query Language (SQL)?,” *TechTarget*, 2023. [Online]. Available: https://www.techtarget.com/searchdatamanagement/definition/SQL. [Accessed: 05-Jan-2023].

[52] C. Brooks, “What Is SQL, and How Is It Used? - businessnewsdaily.com,” *Business News Daily*, 16-Nov-2022. [Online]. Available: https://www.businessnewsdaily.com/5804-what-is-sql.html. [Accessed: 05-Jan-2023].

[53] A. Kozubek-Krycuń, “The History of SQL – How It All Began,” *LearnSQL.com*, 17-Nov-2020. [Online]. Available: https://learnsql.com/blog/history-of-sql/. [Accessed: 05-Jan-2023].

[54] E. F. Codd, “A relational model of data for large shared data banks,” *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970, doi: 10.1145/362384.362685. [Online]. Available: https://doi.org/10.1145/362384.362685. [Accessed: 05-Jan-2023].

[55] E. S, “SQLite vs MySQL – What’s the Difference,” *Hostinger Tutorials*, 27-Dec-2022. [Online]. Available: https://www.hostinger.com/tutorials/sqlite-vs-mysql-whats-the-difference/. [Accessed: 05-Jan-2023].

[56] T. Wiseman, “Comparing some differences of SQL Server to SQLite.” *MSSQLTips.com*, 23-Mar-2017. [Online]. Available: https://www.mssqltips.com/sqlservertip/4777/comparing-some-differences-of-sql-server-to-sqlite/. [Accessed: 05-Jan-2023].

[57] N. Samuel, “SQLite vs PostgreSQL: 8 Critical Differences - Learn | Hevo,” *Hevo*, 18-May-2021. [Online]. Available: https://hevodata.com/learn/sqlite-vs-postgresql/. [Accessed: 05-Jan-2023].

[58] “SQLite: Documentation.” *SQLite.org*, 02-Jan-2023. [Online]. Available: https://www.sqlite.org/src/doc/trunk/ext/userauth/user-auth.txt. [Accessed: 05-Jan-2023].

[59] “Defense Against The Dark Arts.” *SQLite.org*, 07-Nov-2022. [Online]. Available: https://www.sqlite.org/security.html. [Accessed: 05-Jan-2023].

[60] “SQLite Database Speed Comparison.” *SQLite.org*, 01-Apr-2014. [Online]. Available: https://www.sqlite.org/speed.html. [Accessed: 05-Jan-2023].

References (continued)

[61] D. Team, “MySQL vs PostgreSQL vs SQLite: A comparison of 3 popular RDBMS,” *Devathon*, 15-Jan-2021. [Online]. Available: https://devathon.com/blog/mysql-vs-postgresql-vs-sqlite/. [Accessed::06-Jan-2022].

[62] H. Zahid, “MySQL vs SQLite – Compared.” *Linux Hint*. [Online]. Available: https://linuxhint.com/mysql-vs-sqlite/. [Accessed: 06-Jan-2023].

[63] ostezer and M. Drake, “SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems | DigitalOcean.” *DigitalOcean*, 10-Mar-2022. [Online]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. [Accessed: 06-Jan-2023].

[64] M. Ray *et al.*, “Data types (Transact-SQL) - SQL Server.” *Microsoft Learn*, 19-Nov-2022. [Online]. Available: https://learn.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql. [Accessed: 06-Jan-2023].

[65] V. To *et al.*, “TOP (Transact-SQL) - SQL Server.” *Microsoft Learn*, 31-Dec-2022. [Online]. Available: https://learn.microsoft.com/en-us/sql/t-sql/queries/top-transact-sql. [Accessed: 06-Jan-2023].

[66] “SQL SELECT TOP, LIMIT, FETCH FIRST ROWS ONLY, ROWNUM.” *W3Schools*, 2023. [Online]. Available: https://www.w3schools.com/sql/sql_top.asp. [Accessed: 06-Jan-2023].

[67] D. Jalli, “How To Use The SQL NOT EXISTS and EXISTS Operator?,” *Janbasktraining*, 17-Apr-2022. [Online]. Available: https://www.janbasktraining.com/blog/sql-exists-operator. [Accessed: 06-Jan-2023].

[68] H. Zahid, “How to create table in MySQL using ‘if not exists’ technique.” *Linux Hint*. [Online]. Available: https://linuxhint.com/create-table-if-not-exist-mysql/. [Accessed: 06-Jan-2023].

[69] “Drop Tables in PostgreSQL Database.” *TutorialsTeacher*, 2023. [Online]. Available: https://www.tutorialsteacher.com/postgresql/drop-tables. [Accessed: 06-Jan-2023].

[70] V. Kaplarevic, “MySQL DROP TABLE: With Examples & Options,” *phoenixNAP*, 30-Jun-2020. [Online]. Available: https://phoenixnap.com/kb/mysql-drop-table. [Accessed: 06-Jan-2023].

[71] J. Gavin, “SQL Server DROP TABLE IF EXISTS Examples.” *MSSQLTips*, 23-Mar-2021. [Online]. Available: https://www.mssqltips.com/sqlservertip/6769/sql-server-drop-table-if-exists/. [Accessed: 06-Jan-2023].

[72] M. Ray, J. Roth, R. Konidena and R. West, “SQL Server 2019: Hardware & software requirements - SQL Server.” *Microsoft Learn*, 15-Dec-2022. [Online]. Available: https://learn.microsoft.com/en-us/sql/sql-server/install/hardware-and-software-requirements-for-installing-sql-server-2019. [Accessed: 06-Jan-2023].

[73] “Difference Between SQL Vs MySQL Vs SQL Server (with Examples)” *SoftwareTestingHelp*, 05-Dec-2022. [Online]. Available: https://www.softwaretestinghelp.com/sql-vs-mysql-vs-sql-server/. [Accessed: 06-Jan-2023].

[74] I. Hickson and Google, Inc., “Web SQL Database W3C Working Group Note” *W3C*, 18-Nov-2010. [Online]. Available: http://www.w3.org/TR/webdatabase/. [Accessed: 06-Jan-2023].

[75] “7.6. LIMIT and OFFSET,” *The PostgreSQL Global Development Group*, 10-Nov-2022. [Online]. Available: https://www.postgresql.org/docs/15/queries-limit.html. [Accessed: 06-Jan-2023].

[76] “SQL Server TOP and FETCH and PostgreSQL LIMIT and OFFSET - SQL Server to Aurora PostgreSQL Migration Playbook.” *Amazon Web Services, Inc.*, 2023. [Online]. Available: https://docs.aws.amazon.com/dms/latest/sql-server-to-aurora-postgresql-migration-playbook/chap-sql-server-aurora-pg.tsql.topfetch.html. [Accessed: 06-Jan-2023].

[77] “SQL Features That SQLite Does Not Implement.” *SQLite.org*, 13-Apr-2022. [Online]. Available: https://www.sqlite.org/omitted.html. [Accessed: 06-Jan-2023].

[78] “SQL: SELECT LIMIT Statement.” *TechOnTheNet.com*, 2023. [Online]. Available: https://www.techonthenet.com/sql/select_limit.php. [Accessed: 06-Jan-2023].

[79] RajuKumar19, “PostgreSQL - LIMIT with OFFSET clause,” *GeeksforGeeks*, 28-Aug-2020. [Online]. Available: https://www.geeksforgeeks.org/postgresql-limit-with-offset-clause/. [Accessed: 06-Jan-2023].

References (continued)

[80] “How To Do a Full Outer Join in MySQL,” *Ubiq*, 03-Feb-2021. [Online]. Available: https://ubiq.co/database-blog/how-to-do-a-full-outer-join-in-mysql/. [Accessed: 06-Jan-2023].

[81] “MySQL RIGHT JOIN Explained By Practical Examples,” *MySQLTutorial.org*, 2022. [Online]. Available: https://www.mysqltutorial.org/mysql-right-join/. [Accessed: 06-Jan-2023].

[82] “SQL Server Full Outer Join Explained By Practical Examples,” *sqlservertutorial.net*, 2022. [Online]. Available: https://www.sqlservertutorial.net/sql-server-basics/sql-server-full-outer-join/. [Accessed: 06-Jan-2023].

[83] “dbForge SQL Complete - Powerful T-SQL Formatting Tool,” *Devart*, 2023. [Online]. Available: https://www.devart.com/dbforge/sql/sqlcomplete/. [Accessed: 06-Jan-2023].

[84] “PostgreSQL - JOINS.” *Tutorials Point*. [Online]. Available: https://www.tutorialspoint.com/postgresql/postgresql_using_joins.htm. [Accessed: 06-Jan-2023].

[85] R. Peterson, “MySQL JOINS Tutorial: INNER, OUTER, LEFT, RIGHT, CROSS,” *Guru99*, 02-Nov-2022. [Online]. Available: https://www.guru99.com/joins.html. [Accessed: 06-Jan-2023].

[86] “PostgreSQL Joins: A Visual Explanation of PostgreSQL Joins.” *PostgreSQL Tutorial*, 2022. [Online]. Available: https://www.postgresqltutorial.com/postgresql-joins/. [Accessed: 06-Jan-2023].

[87] R. West *et al.*, “Server Configuration Options (SQL Server) - SQL Server.” *Microsoft Learn*, 27-Dec-2022. [Online]. Available: https://learn.microsoft.com/en-us/sql/database-engine/configure-windows/server-configuration-options-sql-server. [Accessed: 06-Jan-2023].

[88] “Chapter 20. Server Configuration,” *The PostgreSQL Global Development Group*, 2023. [Online]. Available: https://www.postgresql.org/docs/15/runtime-config.html. [Accessed: 06-Jan-2023].

[89] I. N. SA, “🚀 MySQL: Maximum number of simultaneous connections - Infomaniak.” *Infomaniak*, 2022. [Online]. Available: https://www.infomaniak.com/en/support/faq/471/mysql-maximum-number-of-simultaneous-connections. [Accessed: 06-Jan-2023].

[90] R. West *et al.*, “Configure the user connections Server Configuration Option - SQL Server.” *Microsoft Learn*, 19-Nov-2022. [Online]. Available: https://learn.microsoft.com/en-us/sql/database-engine/configure-windows/configure-the-user-connections-server-configuration-option. [Accessed: 06-Jan-2023].

[91] “SQLite Foreign Key Support.” *SQLite.org*, 20-Jan-2022. [Online]. Available: https://sqlite.org/foreignkeys.html. [Accessed: 06-Jan-2023].

[92] “Pragma statements supported by SQLite.” *SQLite.org*, 25-Dec-2022. [Online]. Available: https://www.sqlite.org/pragma.html. [Accessed: 06-Jan-2023].

[93] “SELECT.” *SQLite.org*, 26-Oct-2022. [Online]. Available: https://www.sqlite.org/lang_select.html. [Accessed: 06-Jan-2023].

[94] “SQLite WHERE - Filter Rows in a Result Set,” *SQLite Tutorial*, 2022. [Online]. Available: https://www.sqlitetutorial.net/sqlite-where/. [Accessed: 06-Jan-2023].

[95] “SQLite - WHERE Clause.” *Tutorials Point*, 2022. [Online]. Available: https://www.tutorialspoint.com/sqlite/sqlite_where_clause.htm. [Accessed: 06-Jan-2023].

[96] “SQLite SELECT DISTINCT - Removing Duplicate in Result Set,” *SQLite Tutorial*, 2022. [Online]. Available: https://www.sqlitetutorial.net/sqlite-distinct/. [Accessed: 06-Jan-2023].

[97] “SQLite - DISTINCT Keyword.” *Tutorials Point*, 2022. [Online]. Available: https://www.tutorialspoint.com/sqlite/sqlite_distinct_keyword.htm. [Accessed: 06-Jan-2023].

[98] “Features Of SQLite.” *SQLite.org*, 21-Feb-2022. [Online]. Available: https://www.sqlite.org/features.html. [Accessed: 08-Jan-2023].

References (continued)

[99] A. Deveria, L. Schoors and individual contributors, "'SQLite' | Can I use... Support tables for HTML5, CSS3, etc." *Can I use*. [Online]. Available: https://caniuse.com/?search=SQLite. [Accessed: 08-Jan-2023].

[100] "Persistent Storage Options." *SQLite.org*. [Online]. Available: https://sqlite.org/wasm/doc/trunk/persistence.md. [Accessed: 08-Jan-2023].

[101] "sql.js." *sql.js.org*. [Online]. Available: https://sql.js.org/. [Accessed: 08-Jan-2023].

[102] "Home." *sql.js.org*. [Online]. Available: https://sql.js.org/documentation/. [Accessed: 08-Jan-2023].

[103] "Quirks, Caveats, and Gotchas In SQLite." *SQLite.org*, 16-Nov-2022. [Online]. Available: https://www.sqlite.org/quirks.html#aggregate_queries_can_contain_non_aggregate_result_columns_that_are_not_in_the_group_by_clause. [Accessed: 10-Jan-2023].

References (continued)

This page intentionally left blank.

End of document.