# Offline web applications

We will look at Web SQL, IndexedDB, the FileSystem API, and the HTTP cache

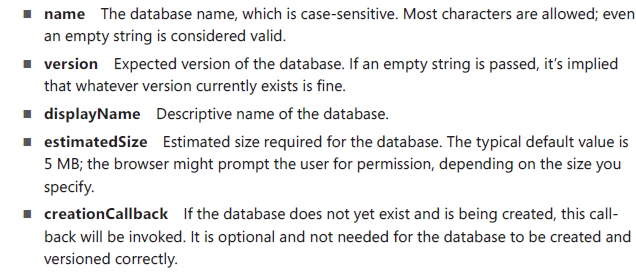
## Lesson 1: Working with Web SQL

**Serious Note**: W3C *no longer supports Web SQL*, but some browsers have continued the support of it. See http://caniuse.com/#feat=sql-storage for a full list of browsers (If you do plan on developing with Web SQL, consider the lack of browser support).

### Creating and opening the database

- Use the *openDatabase* method to start communication with a database (it returns a Database object). Note that if you attempt to open a database that doesn't exist, it will be automatically created for you.

The following are the *openDatabase* parameters:



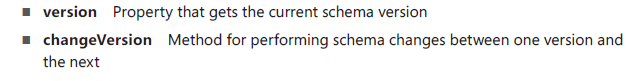
The following example creates a database named 'Library' with an estimated size of 5MB (it returns a Database object that supports transactional operations):

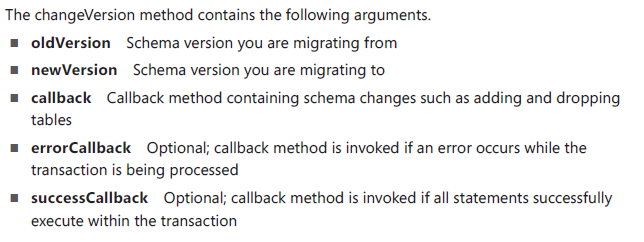
var db = openDatabase('Library', '1.0', 'My Library', 5 \* 1024 \* 1024);

**Closing a connection**: with Web SQL, closing a connection is automatically handled for you

### Performing schema updates

You may need to add new tables, drop existing ones, or even change particular columns. The *Database* object provides the following hooks.



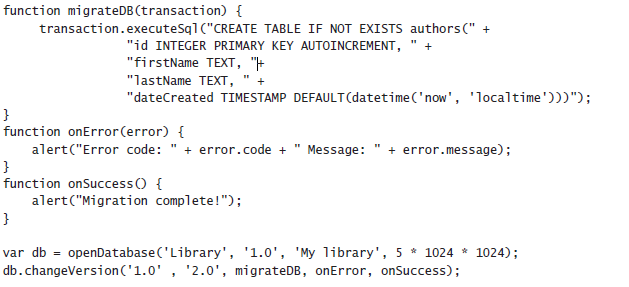


#### Adding a table

To add a table, need a callback method that accepts a *transaction* object, which executes the CREATE TABLE script.

**Note**: The *transaction* object allows multiple actions within it, and it automatically rolls back all changes if any fail.

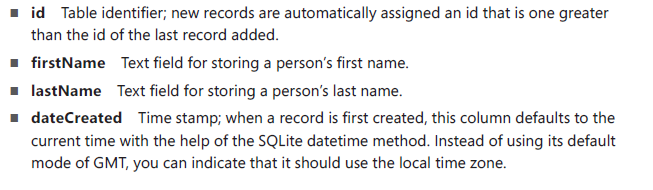
Javascript Example:



JSFiddle: https://jsfiddle.net/bs4r5ooa/ (**The code did not work on JsFiddle, or on my local computer - Forget this garbage**)

**To read the database version:** alert("Current schema: " + db.version);

**After the migration**, the current database has the following fields:



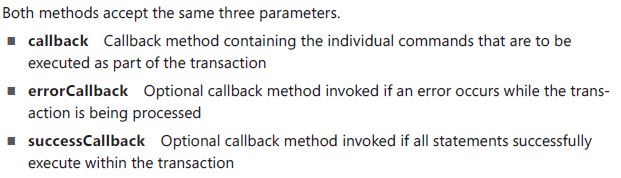
#### Using transactions

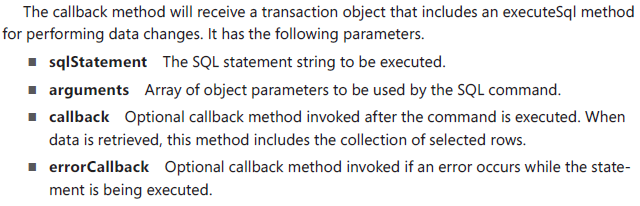
Use transactions to execute SQL statements.

The Database Object provides the ff two methods:



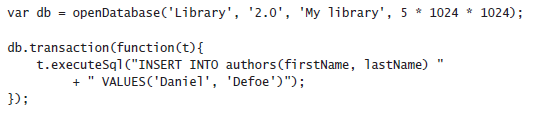




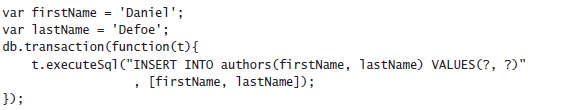


#### Inserting a new record

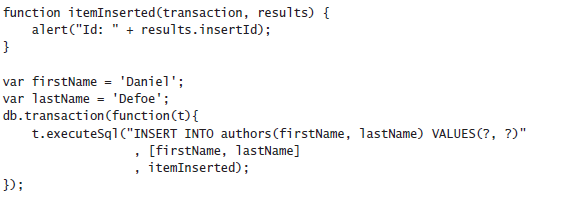
**Without using SQL parameters:**



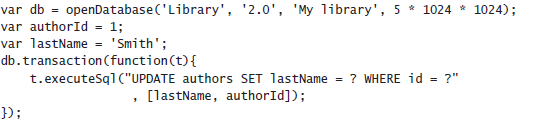
**Using SQL parameters:**

****

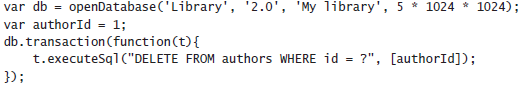
**Add a callback function to the executeSql method, which allows us capture the Id of the newly created row:**

****

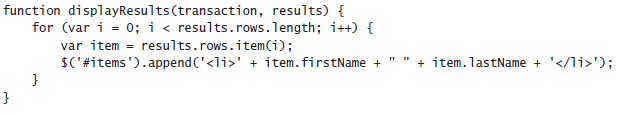
#### Updating an existing record

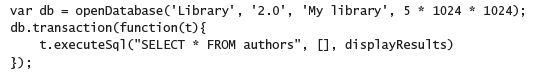


#### Deleting a record



#### Reading values from the database

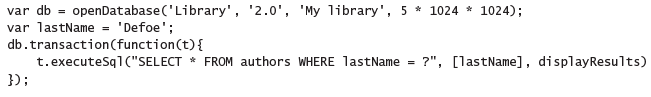




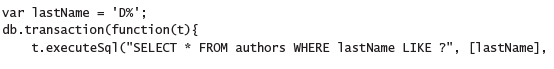
NOTE: Because you are only retrieving data, you just as easily could have used the *readTransaction* method instead of the transaction method



#### Filtering results (using WHERE)

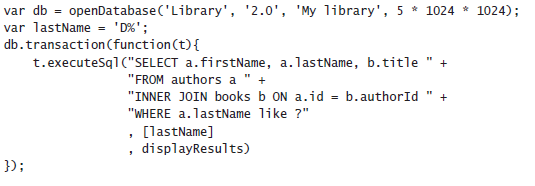


**Using Wildcard symbol** (%)

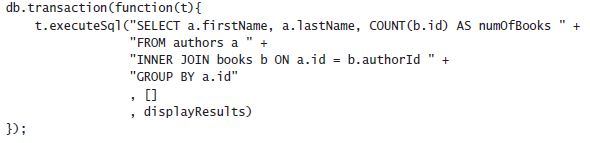




#### Using JOIN commands



#### using Aggregating functions



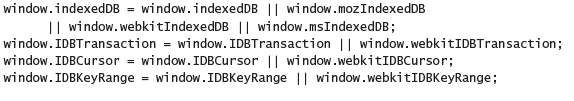
## Lesson 2: Working with IndexedDB

**What is IndexedDB**: A key/value database in which values can range from simple strings to complex object structures

### Using browser-specific code

IndexedDB is still under development hence we need to use browser-specific prefixes.

**Note**: To **make your IndexedDB code cross-browser-friendly**, include the following code at the top of your page

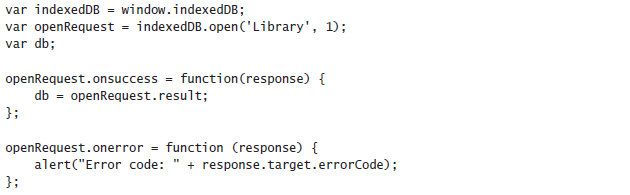


### Creating and opening the database

**To access the browser's indexedDB object,** use:

var indexedDB = window.indexedDB;

**To open an indexedDB object store**, use



**Live code**: Run "index.html". Look at the "testIndexedDBOpen.js" file to see working code.

**Note about using the 'open' indexedDB method**: This method returns an *IDBRequest* object and begins an *asynchronous process* of opening a connection. Hence we have to listen to the 'onsuccess' and 'onerror' methods to determine when the database has been opened or when an error occurs.

### Using object stores

IndexedDB uses 'Object stores', which are key/value storage areas.

#### Understanding versioning

Before creating a new 'object store', you need to understand how IndexedDB handles versioning.

E.g. Consider the code below



In the code, a **version number is passed as the second parameter**. The *'request object'* returned contains an *'onupgradeneeded'* event that will be **triggered if the version requested doesn't match the current version of the existing database (or if the database does not yet exist)**.

**Note**: the ' *onupgradeneeded'* event will be fired BEFORE the *'onsuccess'* event

**How to allocate a new storage area**

Within the *onupgradeneeded* event handler, use the *createObjectStore* method to allocate a new storage area.

This method requires an *object store name* and an *object* containing any extra parameters to use in configuring the store.

#### Using the keypath property

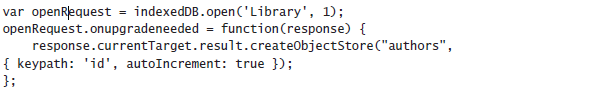
Used to specify which property on the *value* object should be used as the key.

(The key is used as the primary index for stored object instances)

**NOTE**: If the property specified by the keypath does not exist on the value object, you must use a key generator such as autoIncrement, which creates auto incrementing numeric keys

e.g.

**JS**:



**Live Code**: open 'keypathTest.html' in a browser. See 'testIndexedDBKeypath.js' for the code

**To use an existing field in the value object as the *key*** (e.g. using the 'email' property as a key)*,* use:

****

#### Adding indexes