Offline web apps

working with indexed db

So far, you've seen two extremes for client-side data storage. Web storage provides a simple key/value persistence model but lacks some of the features that are important when working with a database. The other extreme, Web SQL, provides many of the features associated with a fully functional relational database but brings with it all the manual work required for set- ting up and maintaining the persistence structure.

- IndexedDB provides a compromise between the two alternatives. It's a key/value database in which values can range from simple strings to complex object structures. To provide for fast retrieval and searching, it includes an easy way to create indexes for each of your object stores. Much like Web SQL, interfacing with the database is transaction-based and requires minimal effort.

- Because the W3C announced that it will not continue development of the Web SQL specification, IndexedDB has gained even more support. Although it's not yet supported by every major browser, it does have wider adoption than Web SQL.

- To work with IndexedDB, you need to use methods that might contain browser - specific prefixes because of the continuing development of the specification. To make your examples cross browser–friendly, include the following x at the top of your scripts to avoid the need to put browser-specific logic in each of your methods. All subsequent examples will be based on this code.

```
window.indexedDB = window.indexedDB || window.mozIndexedDB
      || window.webkitIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction;
window.IDBCursor = window.IDBCursor || window.webkitIDBCursor;
window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange;
```

Creating and opening the database

The first step in working with IndexedDB is to create and open a database. You need to access the browser's indexedDB object, which, in the previous example, was assigned to a consistent variable.

```
var indexedDB = window.indexedDB;
```

This *indexedDB* variable is an IDBFactory object that provides access to your databases through the open method, which has the following parameters.

- **name** The name of the object store
- **version** Optional; the version of the object store

```
var indexedDB = window.indexedDB;
var openRequest = indexedDB.open('Library', 1);
var db;

openRequest.onsuccess = function(response) {
    db = openRequest.result;
};

openRequest.onerror = function (response) {
    alert("Error code: " + response.target.errorCode);
};
```

Using object stores

In standard relational databases, tables are created that are defined by rigid schemas. Each table contains a set of columns, each of which has a name and a data type. This doesn't allow for much flexibility because it requires a lot of work when schema changes are needed. Therefore, instead of these table structures, IndexedDB uses spaces called object stores, which are key/value storage areas.

Adding indexes

Although the key will be the primary index for object stores, you can specify other indexes. This can provide a performance boost if properties other than the key might be commonly used in sorting or filtering. To do so, use the createIndex method on the object store, which has the following parameters.

- **name** The index name.
- **keyPath** Specifies the property on the value object for which the index will be created.
- **optionalParameters** Optional parameter that can contain an object with properties used for advanced index settings. Currently, IndexedDB supports two advanced settings. The first is 'unique', which when true adds a constraint to the property that prohibits two records from having the same value. The second property that can be set is 'multiEntry', which indicates how the index should behave when the keyPath is an array. If set to true, an index entry is created for each value in the array. If set to false, a single index entry is created for the array as a whole.

```
var openRequest = indexedDB.open('Library', 2);
openRequest.onupgradeneeded = function(response) {
    var store = response.currentTarget.transaction.objectStore("authors");
    store.createIndex('lastName', 'lastName', { unique: false });
};
```

Removing object stores

As you probably guessed, the steps to remove an object store are very similar to the steps for creating one. You create a new migration that uses the database's deleteObjectStore() method as follows.

```
var openRequest = indexedDB.open('Library', 4);
openRequest.onupgradeneeded = function(response) {
    response.currentTarget.result.deleteObjectStore("authors");
};
```

using transactions

- **objectStoreNames** Specifies the object stores with which the transaction will work. If only one object store is needed, the parameter can be a single string. If multiple object stores are needed, pass an array of strings. The following is an example of opening a transaction for a single object store.

```
var trans = db.transaction('authors');
```

Here is an example of opening a transaction for multiple object stores.

```
var trans = db.transaction(['authors', 'books']);
```

using transactions

- **mode** Optional when possible values are *readonly* and *readwrite*. If not specified, the transaction will be defaulted to *readonly*. If left in readonly mode, multiple transactions can be run concurrently. The following is an example of a transaction being opened in readonly mode.

```
var trans = db.transaction('authors', 'readonly');
```

Example of a transaction being opened in readwrite mode.

```
var trans = db.transaction('authors', 'readwrite');
```

inserting a new keyword

```
var openRequest = indexedDB.open('Library', 1);
var db;

openRequest.onsuccess = function(response) {
    db = openRequest.result;
    addAuthor();
};

function addAuthor() {
    var trans = db.transaction('authors', 'readwrite');
    var authors = trans.objectStore("authors");
    var request = authors.add({firstName: 'Daniel', lastName: 'Defoe'});
    request.onsuccess = function(response) {
        alert('New record id: ' + request.result);
    };

    request.onerror = function(response) {  // display error };
}
```

Deleting a record

To remove a stored object, you only need to know its key value, which is passed to the delete method of the object store.

```
function deleteAuthor() {
    var trans = db.transaction('authors', 'readwrite');
    var authors = trans.objectStore("authors");
    var request = authors.delete(1);

    request.onsuccess = function(response) {  // success! };
    request.onerror = function(response) {  // display error };
}
```