Indexed DB in Firefox OS

Data Access using No SQL technology based on Open Web Platform

By A.R. Kazemi, Reference: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Using_IndexedDB

SQL vs No SQL Data Access

- SQL-based DBMSs
  - Use SQL for querying the DB
  - Data Model:
    - Relational
    - Object Relational Mapping (ORM)
  - Examples
    - PostgresSql, Oracle, MS-SQL Server, MySQL, DB-II, Sybase, …
- No-SQL DBs
  - Use an API function call to access data
  - Data Model
    - Object-based
  - Examples:
    - MongoDB, …

Indexed DB in Web Platform

```
// In the following line, you should include the prefixes of implementations you want to test.
window.indexedDB = window.indexedDB || window.mozIndexedDB || window.webkitIndexedDB || window.msIndexedDB;
// DON'T use "var indexedDB = ..." if you're not in a function.
// Moreover, you may need references to some window.IDB* objects:
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange || window.msIDBKeyRange;
// (Mozilla has never prefixed these objects, so we don't need window.mozIDB*)
```

```
if (!window.indexedDB) {
    window.alert("Your browser doesn't support a stable version of IndexedDB.");
}
```

Opening a Database

- Open it uisng:

```
// Let us open our database
var request = window.indexedDB.open("MyTestDatabase", 3);
```

- Handle the open request by defining handlers as callback functions

```
request.onerror = function(event) {
  // Do something with request.errorCode!
};
request.onsuccess = function(event) {
  // Do something with request.result!
};
```

Opening a Database

- Open it uisng:

```
// Let us open our database
var request = window.indexedDB.open("MyTestDatabase", 3);
```

- Handle the open request by defining handlers as callback functions

```
request.onerror = function(event) {
  // Do something with request.errorCode!
};
request.onsuccess = function(event) {
  // Do something with request.result!
};
```

Error Events Bubble up!

Different Positions for Handling Error Events

- Per Request Handler

```
var db;
var request = indexedDB.open("MyTestDatabase");
request.onerror = function(event) {
  alert("Why didn't you allow my web app to use IndexedDB?!");
};
request.onsuccess = function(event) {
  db = event.target.result;
};
```

- Generic Handler on DB object

```
db.onerror = function(event) {
  // Generic error handler for all errors targeted at this database's
  // requests!
  alert("Database error: " + event.target.errorCode);
};
```

Creating or updating the version of the database

```
// This event is only implemented in recent browsers
request.onupgradeneeded = function(event) {
  var db = event.target.result;

  // Create an objectStore for this database
  var objectStore = db.createObjectStore("name", { keyPath: "myKey" });
};
```

Structure of the Database

Key Path (**keyPath**)	Key Generator (**autoIncrement**)	Description
No	No	This object store can hold any kind of value, even primitive values like numbers and strings. You must supply a separate key argument whenever you want to add a new value.
Yes	No	This object store can only hold JavaScript objects. The objects must have a property with the same name as the key path.
No	Yes	This object store can hold any kind of value. The key is generated for you automatically, or you can supply a separate key argument if you want to use a specific key.
Yes	Yes	This object store can only hold JavaScript objects. Usually a key is generated and the value of the generated key is stored in the object in a property with the same name as the key path. However, if such a property already exists, the value of that property is used as key rather than generating a new key.

- Example records:

```
// This is what our customer data looks like.
const customerData = [
  { ssn: "444-44-4444", name: "Bill", age: 35, email: "bill@company.com" },
  { ssn: "555-55-5555", name: "Donna", age: 32, email: "donna@home.org" }
];
```

Creating DB and filling some Records (Objects)

```
const dbName = "the_name";

var request = indexedDB.open(dbName, 2);

request.onerror = function(event) {
  // Handle errors.
};
request.onupgradeneeded = function(event) {
  var db = event.target.result;

  // Create an objectStore to hold information about our customers. We're
  // going to use "ssn" as our key path because it's guaranteed to be
  // unique - or at least that's what I was told during the kickoff meeting.
  var objectStore = db.createObjectStore("customers", { keyPath: "ssn" });

  // Create an index to search customers by name. We may have duplicates
  // so we can't use a unique index.
  objectStore.createIndex("name", "name", { unique: false });

  // Create an index to search customers by email. We want to ensure that
  // no two customers have the same email, so use a unique index.
  objectStore.createIndex("email", "email", { unique: true });

  // Use transaction oncomplete to make sure the objectStore creation is
  // finished before adding data into it.
  objectStore.transaction.oncomplete = function(event) {
    // Store values in the newly created objectStore.
    var customerObjectStore = db.transaction("customers", "readwrite").objectStore("customers");
    for (var i in customerData) {
      customerObjectStore.add(customerData[i]);
    }
  }
};
```

Key Generators

```
// Open the indexedDB.
var request = indexedDB.open(dbName, 3);

request.onupgradeneeded = function (event) {

    var db = event.target.result;

    // Create another object store called "names" with the autoIncrement flag set as true.
    var objStore = db.createObjectStore("names", { autoIncrement : true });

    // Because the "names" object store has the key generator, the key for the name value is generated
    // The added records would be like:
    // key : 1 => value : "Bill"
    // key : 2 => value : "Donna"
    for (var i in customerData) {
        objStore.add(customerData[i].name);
    }
}
```

Adding, Retrieving, Editing, Removing, Data

- Creating a ReadWrite transaction

```
var transaction = db.transaction(["customers"], "readwrite");
// Note: Older experimental implementations use the deprecated constant IDBTransaction.READ_WRITE instead
// In case you want to support such an implementation, you can write:
// var transaction = db.transaction(["customers"], IDBTransaction.READ_WRITE);
```

- Adding Data

```
// Do something when all the data is added to the database.
transaction.oncomplete = function(event) {
  alert("All done!");
};

transaction.onerror = function(event) {
  // Don't forget to handle errors!
};

var objectStore = transaction.objectStore("customers");
for (var i in customerData) {
  var request = objectStore.add(customerData[i]);
  request.onsuccess = function(event) {
    // event.target.result == customerData[i].ssn;
  };
}
```

Adding, Retrieving, Editing, Removing, Data

- Removing Data

```
var request = db.transaction(["customers"], "readwrite")
                .objectStore("customers")
                .delete("444-44-4444");
request.onsuccess = function(event) {
  // It's gone!
};
```

- Retrieving Data

```
var transaction = db.transaction(["customers"]);
var objectStore = transaction.objectStore("customers");
var request = objectStore.get("444-44-4444");
request.onerror = function(event) {
  // Handle errors!
};
request.onsuccess = function(event) {
  // Do something with the request.result!
  alert("Name for SSN 444-44-4444 is " + request.result.name);
};
```

Adding, Retrieving, Editing, Removing, Data

- Updating an entry

```
var objectStore = db.transaction(["customers"], "readwrite").objectStore("customers");
var request = objectStore.get("444-44-4444");
request.onerror = function(event) {
  // Handle errors!
};
request.onsuccess = function(event) {
  // Get the old value that we want to update
  var data = request.result;

  // update the value(s) in the object that you want to change
  data.age = 42;

  // Put this updated object back into the database.
  var requestUpdate = objectStore.put(data);
   requestUpdate.onerror = function(event) {
     // Do something with the error
   };
   requestUpdate.onsuccess = function(event) {
     // Success - the data is updated!
   };
};
```

Adding, Retrieving, Editing, Removing, Data

- Using Data Cursors

```
var objectStore = db.transaction("customers").objectStore("customers");

objectStore.openCursor().onsuccess = function(event) {
  var cursor = event.target.result;
  if (cursor) {
    alert("Name for SSN " + cursor.key + " is " + cursor.value.name);
    cursor.continue();
  }
  else {
    alert("No more entries!");
  }
};
```

Adding, Retrieving, Editing, Removing, Data

- Using Data Cursors

```
1  var customers = [];
2
3  objectStore.openCursor().onsuccess = function(event) {
4    var cursor = event.target.result;
5    if (cursor) {
6      customers.push(cursor.value);
7      cursor.continue();
8    }
9    else {
10     alert("Got all customers: " + customers);
11   }
12 };
```

Get All Records

- This is Mozilla (FF) Specific
  - This option is hidden by default settings and its use is not recommended.

```
1 objectStore.getAll().onsuccess = function(event) {
2   alert("Got all customers: " + event.target.result);
3 };
```

Using Index

- Get objects using an index

```
var index = objectStore.index("name");

index.get("Donna").onsuccess = function(event) {
  alert("Donna's SSN is " + event.target.result.ssn);
};
```

Two Different Types of Cursors

- Whole Record Cursor and Key Cursor

```
// Using a normal cursor to grab whole customer record objects
index.openCursor().onsuccess = function(event) {
  var cursor = event.target.result;
  if (cursor) {
    // cursor.key is a name, like "Bill", and cursor.value is the whole object.
    alert("Name: " + cursor.key + ", SSN: " + cursor.value.ssn + ", email: " + cursor.value.email);
    cursor.continue();
  }
};

// Using a key cursor to grab customer record object keys
index.openKeyCursor().onsuccess = function(event) {
  var cursor = event.target.result;
  if (cursor) {
    // cursor.key is a name, like "Bill", and cursor.value is the SSN.
    // No way to directly get the rest of the stored object.
    alert("Name: " + cursor.key + ", SSN: " + cursor.value);
    cursor.continue();
  }
};
```

Specifying the Range in Cursors

- Single Record, Lower Bound, Upper Bound, Lower-Upper Bound

```
// Only match "Donna"
var singleKeyRange = IDBKeyRange.only("Donna");

// Match anything past "Bill", including "Bill"
var lowerBoundKeyRange = IDBKeyRange.lowerBound("Bill");

// Match anything past "Bill", but don't include "Bill"
var lowerBoundOpenKeyRange = IDBKeyRange.lowerBound("Bill", true);

// Match anything up to, but not including, "Donna"
var upperBoundOpenKeyRange = IDBKeyRange.upperBound("Donna", true);

// Match anything between "Bill" and "Donna", but not including "Donna"
var boundKeyRange = IDBKeyRange.bound("Bill", "Donna", false, true);

// To use one of the key ranges, pass it in as the first argument of openCursor()/openKeyCursor()
index.openCursor(boundKeyRange).onsuccess = function(event) {
  var cursor = event.target.result;
  if (cursor) {
    // Do something with the matches.
    cursor.continue();
  }
};
```

Specifying the Bound & Direction

- Specifying Bound & Traverse Direction

```
objectStore.openCursor(boundKeyRange, "prev").onsuccess = function(event) {
  var cursor = event.target.result;
  if (cursor) {
    // Do something with the entries.
    cursor.continue();
  }
};
```

- Only Specifying the Direction

```
objectStore.openCursor(null, "prev").onsuccess = function(event) {
  var cursor = event.target.result;
  if (cursor) {
    // Do something with the entries.
    cursor.continue();
  }
};
```

- Traversing Unique Keys

```
index.openKeyCursor(null, "nextunique").onsuccess = function(event) {
  var cursor = event.target.result;
  if (cursor) {
    // Do something with the entries.
    cursor.continue();
  }
};
```

Support in Different Web Browsers (WebViews)

- Very old browsers just have **cookie** as client-side storage
  - Which is so limited
- Newer versions have implemented **localStorage**
- More recent versions have implemented **WebSQL**
- Current Versions have also **indexedDB**

- **So what we should do about browser compatibility?**
  - **Solution:** use **localForge**
  - a JS library which provides data storage using indexedDB and if it is not available fall backs on WebSQL and then localStorage.

Version Change While WebApp Is Open in Another Tab

- Use the onblocked callback to detect such a situation

```
var openReq = mozIndexedDB.open("MyTestDatabase", 2);

openReq.onblocked = function(event) {
  // If some other tab is loaded with the database, then it needs to be closed
  // before we can proceed.
  alert("Please close all other tabs with this site open!");
};

openReq.onupgradeneeded = function(event) {
  // All other databases have been closed. Set everything up.
  db.createObjectStore(/* ... */);
  useDatabase(db);
}

openReq.onsuccess = function(event) {
  var db = event.target.result;
  useDatabase(db);
  return;
}

function useDatabase(db) {
  // Make sure to add a handler to be notified if another page requests a version
  // change. We must close the database. This allows the other page to upgrade the database.
  // If you don't do this then the upgrade won't happen until the user closes the tab.
  db.onversionchange = function(event) {
    db.close();
    alert("A new version of this page is ready. Please reload!");
  };

  // Do stuff with the database.
}
```