CRUD Operations Using WebSQL In HTML5 And jQuery

- Updated date Jan 21, 2020

Introduction

"Web SQL" feature is new in HTML5. Web SQL is an API, which helps the developers to do some database operations on the client-side, like creating a database, opening the transaction, creating the tables, inserting values to the tables, deleting the values and reading the data. If you need any other way to save some data on the client-side, you can use the storage mechanisms, introduced in HTML5.

Browser Support

A Web SQL database will work in the latest version of Safari, Chrome, and Opera.

Differents types of Methods are,

1. *openDatabase*

   Creates the database object, either using an existing database or creating a new one.

2. *transaction*

   Controls a transaction and performs either a commit or rollback on the basis of the situation.

3. *executeSql*

   Executes an SQL query.

Create/Open Web SQL Database

To create a Web SQL database, we can use a function called an open database, which has four parameters as follows:

- Database name
- Version Number
- Description
- Size
- Creation callback.

The last argument (Creation callback) will be called if the database is being created.

1. var Database_Name = 'MyDatabase';

```
2.        var Version = 1.0;
3.        var Text_Description = 'My First Web-SQL Example';
4.        var Database_Size = 2 * 1024 * 1024;
5.        var dbObj = openDatabase(Database_Name, Version, Text_Description,
    Database_Size);
```

Creating transaction

To create a transaction we can use the following syntax. We can use the transaction method from our database instance.

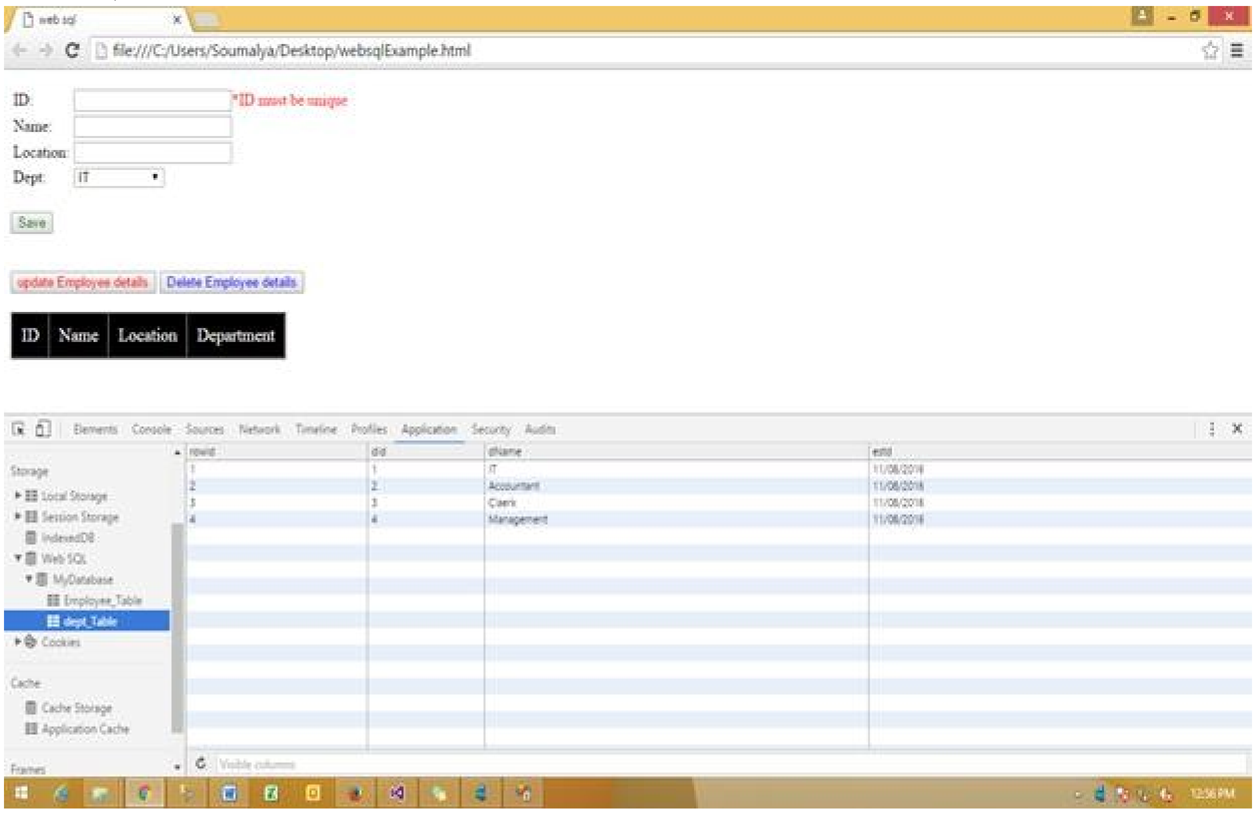
```
1. dbObj.transaction(function (tx)
2.  {
3. // do SQL here using the tx object
4. });
```

Execute Sql

The executeSql method is used for both reads and write statements. To execute a SQL query, we can use the following syntax:

- **Table creation**

```
1. dbObj.transaction(function (tx)
2.  {
3. tx.executeSql('CREATE TABLE IF NOT EXISTS Employee_Table (id unique,
   Name, Location,did)');
4.          tx.executeSql('CREATE TABLE IF NOT EXISTS dept_Table (did unique,
   dName,estd)');
5. });
```



ID	Name	Location	Department

rowid	did	dName	estd
1	1	IT	11/08/2016
2	2	Accountant	11/08/2016
3	3	Clerk	11/08/2016
4	4	Management	11/08/2016

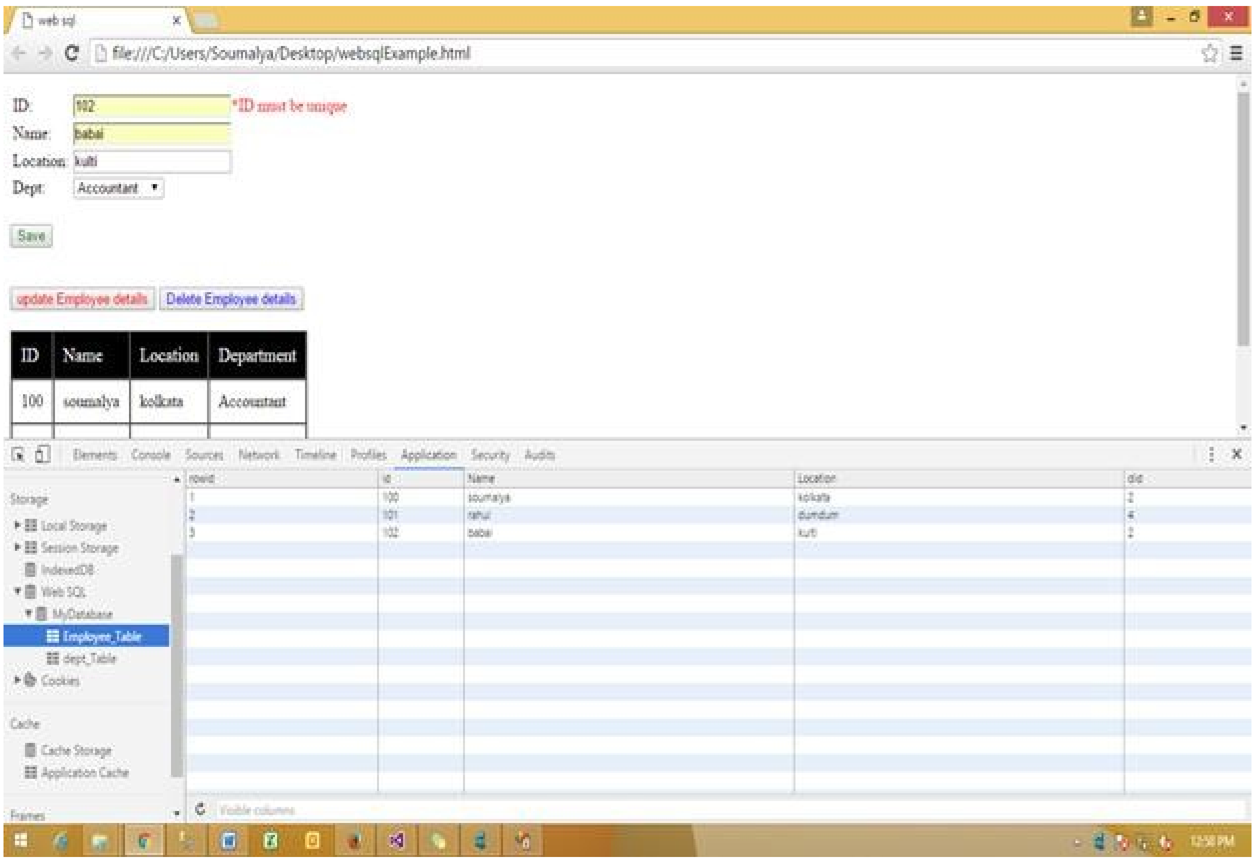
Note, if the database already exists, the transaction will fail. Thus, any successive SQL will not run. Thus, we can either use another transaction or we can only create the table if it doesn't exist.

- **Insert**

Now, let's see, how to perform an insert operation.

```
var id = document.getElementById("tbID").value;
        var name = document.getElementById("tbName").value;
        var location = document.getElementById("tbLocation").value;
        var did = document.getElementById("tbLdept").value;
        dbObj.transaction(function (tx) {
            tx.executeSql('insert into Employee_Table(id, Name, Location,did) values(' + id + ',"' + name + '","' + location + '",' + did + ')');
 });
```

After insert



web sql

file:///C:/Users/Soumalya/Desktop/websqlExample.html

ID:	102 *ID must be unique
Name:	babai
Location:	kulti
Dept:	Accountant

Save

update Employee details Delete Employee details

ID	Name	Location	Department
100	soumalya	kolkata	Accountant

Elements Console Sources Network Timeline Profiles Application Security Audits

Storage: Local Storage, Session Storage, IndexedDB, Web SQL, MyDatabase, Employee_Table, dept_Table, Cookies

Cache: Cache Storage, Application Cache

Frames

rowid	id	Name	Location	did
1	100	soumalya	kolkata	2
2	101	rahul	dumdum	4
3	102	babai	kulti	2

- **Select**

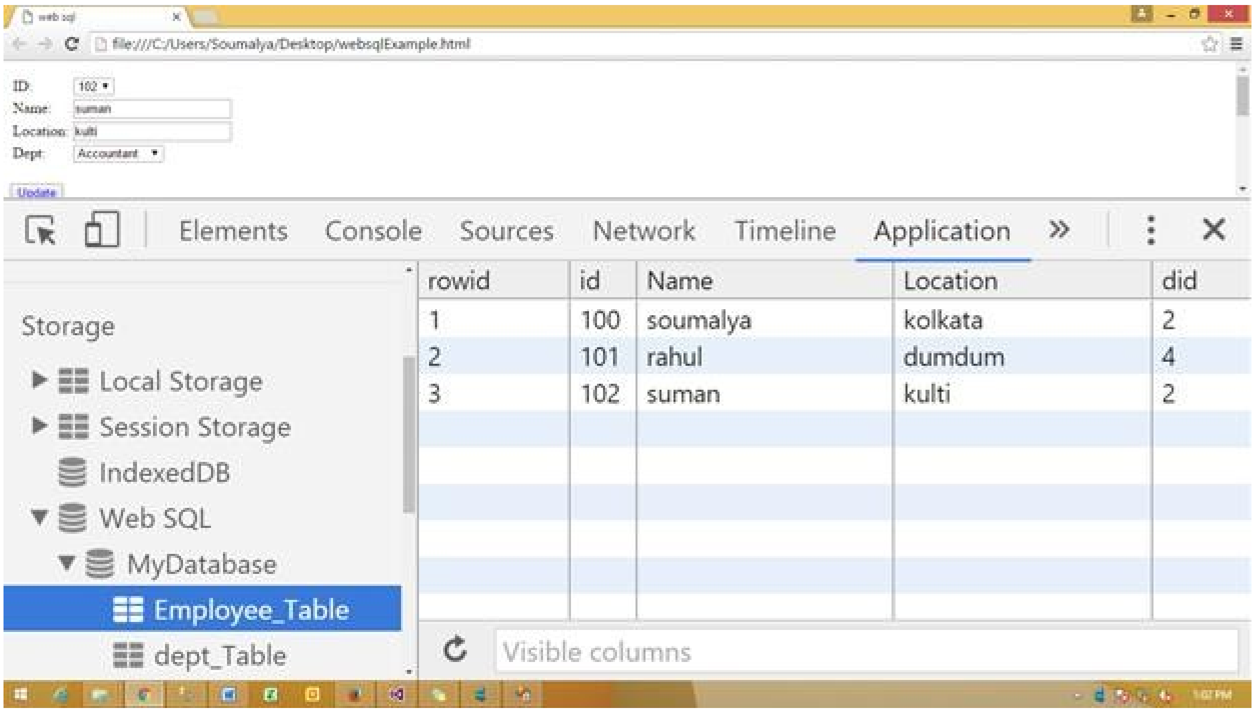
Now, let's see, how to retrieve the inserted record.

```
dbObj.transaction(function (tx) {
            tx.executeSql('SELECT e.id,e.Name,e.Location,d.dName,d.did FROM Employee_Table e inner join dept_Table d on e.did=d.did ', [], function (tx, results) {
```

```
            var len = results.rows.length, i;
            // document.getElementById("tblGrid").innerHTML = ";
            $("#tblGrid").find("tr:gt(0)").remove();
            var str = ";
            for (i = 0; i < len; i++) {
               str += "<tr>";
               str += "<td>" + results.rows.item(i).id + "</td>";
               str += "<td>" + results.rows.item(i).Name + "</td>";
               str += "<td>" + results.rows.item(i).Location + "</td>";
               str += "<td>" + results.rows.item(i).dName + "</td>";
               str += "</tr>";
               document.getElementById("tblGrid").innerHTML += str;
               str = ";
            }
```

- **Update**

```
var id = document.getElementById("ddlid").value;
        var name = document.getElementById("tbName").value;
        var location = document.getElementById("tbLocation").value;
        var did = document.getElementById("tbLdept").value;

        dbObj.transaction(function (tx) {
           tx.executeSql('update Employee_Table set Name="' + name + '",Location="' + location + '",did=' + did + ' where id=' + id + '');
        });
```


file:///C:/Users/Soumalya/Desktop/websqlExample.html

ID: 102
Name: suman
Location: kulti
Dept: Accountant

Update

rowid	id	Name	Location	did
1	100	soumalya	kolkata	2
2	101	rahul	dumdum	4
3	102	suman	kulti	2

Now, let's see, how to perform an update operation.

- **Delete**

Now, let's see, how to perform delete operation.

```
var id = document.getElementById("ddlid").value;
    //  alert(id);
      dbObj.transaction(function (tx) {
          tx.executeSql('delete from Employee_Table where id=' + id + '');
      });
```

Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Open DataBase</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>

    <script>

        var Database_Name = 'MyDatabase';
        var Version = 1.0;
        var Text_Description = 'My First Web-SQL Example';
        var Database_Size = 2 * 1024 * 1024;
        var dbObj = openDatabase(Database_Name, Version, Text_Description, Database_Size);
        dbObj.transaction(function (tx) {

            tx.executeSql('CREATE TABLE IF NOT EXISTS Employee_Table (id unique, Name, Location,did)');
            tx.executeSql('CREATE TABLE IF NOT EXISTS dept_Table (did unique, dName,estd)');


            var today = new Date();
            var dd = today.getDate();
            var mm = today.getMonth() + 1; //January is 0!

            var yyyy = today.getFullYear();
            if (dd < 10) {
                dd = '0' + dd
            }
            if (mm < 10) {
                mm = '0' + mm
            }
            var today = dd + '/' + mm + '/' + yyyy;

            tx.executeSql('insert into dept_Table(did, dName, estd) values(1,"IT","' + today + '")');
            tx.executeSql('insert into dept_Table(did, dName, estd) values(2,"Accountant","' + today + '")');
            tx.executeSql('insert into dept_Table(did, dName, estd) values(3,"Claerk","' + today + '")');
            tx.executeSql('insert into dept_Table(did, dName, estd) values(4,"Management","' + today + '")');
            alldetails();
        });


        function Insert() {

            var id = document.getElementById("tbID").value;
            var name = document.getElementById("tbName").value;
            var location = document.getElementById("tbLocation").value;
            var did = document.getElementById("tbLdept").value;
            dbObj.transaction(function (tx) {
                tx.executeSql('insert into Employee_Table(id, Name, Location,did) values(' + id + ',"' + name + '","' + location + '",' + did + ')');


            });

            alldetails();
        }

        function del() {
            var id = document.getElementById("ddlid").value;
          //  alert(id);
            dbObj.transaction(function (tx) {
                tx.executeSql('delete from Employee_Table where id=' + id + '');
            });

            empidbind();

            alldetails();
        }


        function myFunction()
        {
            var id = document.getElementById("ddlid").value;

            dbObj.transaction(function (tx) {
                tx.executeSql('SELECT * from Employee_Table where id=' + id + '', [], function (tx, results)
                {
                    document.getElementById("tbName").value = results.rows.item(0).Name;
                    document.getElementById("tbLocation").value = results.rows.item(0).Location;
                    document.getElementById("tbLdept").value = results.rows.item(0).did;


                }, null);
            });

        }

        function showdel() {

            empidbind();

            $('#tdorginal').hide();
            $('#tdid').show();
            $('#btnupdate').hide();
            $('#btnInsert').hide();
            $('#btndel').show();
            $('#btninsertshw').show();
            $('#btnupdateshw').show();
            $('#btndeleteshw').hide();
            ////////////
            $('#rowName').hide();
            $('#rowLocation').hide();
            $('#rowdept').hide();
          }

          function showin()
          {
            $('#tdid').hide();
            $('#tdorginal').show();

            $('#btnupdate').hide();
            $('#btnInsert').show();
            $('#btndel').hide();
            $('#btninsertshw').hide();
            $('#btnupdateshw').show();
            $('#btndeleteshw').show();
            ////////////
            $('#rowName').show();
            $('#rowLocation').show();
            $('#rowdept').show();

            /////////////
```

```
      document.getElementById("tbID").value='';
      document.getElementById("tbName").value='';
      document.getElementById("tbLocation").value='';
      document.getElementById("tbLdept").value='1';

           empidbind();
    }


    function empidbind()
    {
        dbObj.transaction(function (tx) {
            tx.executeSql('SELECT * from Employee_Table', [], function (tx, results) {
                var len = results.rows.length, i;
                document.getElementById("ddlid").innerHTML = '';
                var str = '';
                for (i = 0; i < len; i++) {
                    str += "<option value=" + results.rows.item(i).id + ">" + results.rows.item(i).id + "</option>";
                    document.getElementById("ddlid").innerHTML += str;
                    str = '';
                }
            }, null);
        });

    }

    function showupdte()
    {
        empidbind();


        $('#tdorginal').hide();
        $('#tdid').show();
        $('#btnupdate').show();
        $('#btnInsert').hide();
        $('#btndel').hide();
        $('#btninsertshw').show();
        $('#btnupdateshw').hide();
        $('#btndeleteshw').show();
        $('#rowName').show();
        $('#rowLocation').show();
        $('#rowdept').show();

    }

```

```
function updte() {

    var id = document.getElementById("ddlid").value;
    var name = document.getElementById("tbName").value;
    var location = document.getElementById("tbLocation").value;
    var did = document.getElementById("tbLdept").value;

    dbObj.transaction(function (tx) {
        tx.executeSql('update Employee_Table set Name="' + name + '",Location="' + location + '",did=' + did + ' where id=' + id + ");
    });

    alldetails();
}

function alldetails()
{
    dbObj.transaction(function (tx) {
        tx.executeSql('SELECT e.id,e.Name,e.Location,d.dName,d.did FROM Employee_Table e inner join dept_Table d on e.did=d.did ', [], function (tx, results) {
            var len = results.rows.length, i;
            // document.getElementById("tblGrid").innerHTML = ";
            $("#tblGrid").find("tr:gt(0)").remove();
            var str = ";
            for (i = 0; i < len; i++) {
                str += "<tr>";
                str += "<td>" + results.rows.item(i).id + "</td>";
                str += "<td>" + results.rows.item(i).Name + "</td>";
                str += "<td>" + results.rows.item(i).Location + "</td>";
                str += "<td>" + results.rows.item(i).dName + "</td>";
                str += "</tr>";
                document.getElementById("tblGrid").innerHTML += str;
                str = ";
            }
        }, null);
    });

}

dbObj.transaction(function (tx) {
    tx.executeSql('SELECT * from dept_Table', [], function (tx, results) {
        var len = results.rows.length, i;
        var str = ";
        for (i = 0; i < len; i++) {
            str += "<option value=" + results.rows.item(i).did + ">" + results.rows.item(i).dName + "</option>";
```

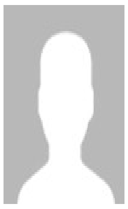
```
                    document.getElementById("tbLdept").innerHTML += str;
                    str = '';
                }
            }, null);
        });



    </script>
</head>
<body>
    <p id="hh"></p>
    <form id="frm1">
        <table id="tblinsert">
            <tr>
                <td>ID:</td>
                <td id="tdorginal"><input type="text" id="tbID" /><span style="color:red">*ID must be unique</span></td>
                <td id="tdid" style="display:none">
                    <select id="ddlid" onchange="myFunction()"></select>
                </td>
            </tr>
            <tr id="rowName">
                <td>Name:</td>
                <td><input type="text" id="tbName" /></td>

            </tr>
            <tr id="rowLocation">
                <td>Location:</td>
                <td><input type="text" id="tbLocation" /></td>
            </tr>

            <tr id="rowdept">
                <td>Dept:</td>
                <td>
                    <select id="tbLdept"></select>
                </td>
            </tr>

            <tr>

            </tr>
        </table>
    </form>
    <br />
```

```
<button id="btnInsert" onclick="Insert()" style="color:green;display:block">Save</button>
<button id="btnupdate" onclick="updte()" style="color:blue;display:none">Update</button>
<button id="btndel" onclick="del()" style="color:red;display:none">Delete</button>

<br /><br />
<button id="btnupdateshw" onclick="showupdte()" style="color:red">update Employee details</button>
<button id="btndeleteshw" onclick="showdel()" style="color:blue">Delete Employee details</button>
<button id="btninsertshw" onclick="showin()" style="color:green;display:none">save Employee details</button>
<br /><br />
<table id="tblGrid" cellpadding="10px" cellspacing="0" border="1">
    <tr style="background-color:black;color:white;font-size:18px;">
        <td >
            ID
        </td>
        <td >
            Name
        </td>
        <td >
            Location
        </td>
        <td >
            Department
        </td>
    </tr>
</table>
<br />


</body>
</html>
```

Conclusion

In this article, we studied CRUD Operations Using WebSQL In HTML5 And jQuery.
https://www.c-sharpcorner.com/article/crud-operations-using-websql-in-html5-and-jquery/

CRUD Operations Using WebSQL In HTML5 And jQuery



@sudhanshu.tripathi

over 4 years ago

- 0
- 0
- 0
- 0
- 0
- 0
- 0
- 0
- 664

Comment on it

Hello!!

HTML 5 provides an API, called "Web SQL" API. It is new feature in HTML5 and it also helps to the developer for doing CRUD operation on the client side. It give the permission to client side for doing some operation:-

- Create a Database.
- Transaction.
- Execute sql query.
- Creating a Table.
- Inserting Data to Table.
- Deleting Data from Table.
- Reading Data.

Different Type of Method:-

1. openDatabase:- It has the some parameter like Database Name, Version, Description, Size and Creation Callback.
2. transaction.
3. executeSql.

Here is the example which describe all the operation in mention above:-

Code:-

```
<!DOCTYPE html>  <html>  <head>      <title>Open DataBase</title>      <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></
script>          <script>            var Database_Name = 'MyDatabase';
var Version = 1.0;         var Text_Description = 'My First Web-SQL Example';
var Database_Size = 2 * 1024 * 1024;         var dbObj =
openDatabase(Database_Name, Version, Text_Description, Database_Size);
dbObj.transaction(function (tx) {               tx.executeSql('CREATE TABLE IF
NOT EXISTS Employee_Table (id unique, Name, Location,did)');
tx.executeSql('CREATE TABLE IF NOT EXISTS dept_Table (did unique, dName,estd)');
var today = new Date();              var dd = today.getDate();              var
mm = today.getMonth() + 1; //January is 0!                var yyyy =
today.getFullYear();              if (dd < 10) {                  dd = '0' + dd
}              if (mm < 10) {                  mm = '0' + mm              }
var today = dd + '/' + mm + '/' + yyyy;                tx.executeSql('insert
into dept_Table(did, dName, estd) values(1,"IT","' + today + '")');
tx.executeSql('insert into dept_Table(did, dName, estd) values(2,"Accountant","'
+ today + '")');              tx.executeSql('insert into dept_Table(did, dName,
estd) values(3,"Claerk","' + today + '")');             tx.executeSql('insert
into dept_Table(did, dName, estd) values(4,"Management","' + today + '")');
alldetails();          });              function Insert() {
var id = document.getElementById("tbID").value;              var name =
document.getElementById("tbName").value;              var location =
document.getElementById("tbLocation").value;            var did =
document.getElementById("tbLdept").value;
dbObj.transaction(function (tx) {                  tx.executeSql('insert into
Employee_Table(id, Name, Location,did) values(' + id + ',"' + name + '","' +
location + '",' + did + ')');                  });               alldetails();
}            function del() {            var id =
document.getElementById("ddlid").value;            //  alert(id);
dbObj.transaction(function (tx) {                  tx.executeSql('delete from
Employee_Table where id=' + id + '');              });
empidbind();                alldetails();         }              function
myFunction()          {             var id =
document.getElementById("ddlid").value;
dbObj.transaction(function (tx) {                  tx.executeSql('SELECT * from
Employee_Table where id=' + id + '', [], function (tx, results)
{                      document.getElementById("tbName").value =
results.rows.item(0).Name;
document.getElementById("tbLocation").value = results.rows.item(0).Location;
document.getElementById("tbLdept").value = results.rows.item(0).did;
}, null);             });                      }            function showdel()
{              empidbind();                $('#tdorginal').hide();
$('#tdid').show();              $('#btnupdate').hide();              $
('#btnInsert').hide();            $('#btndel').show();             $
('#btninsertshw').show();            $('#btnupdateshw').show();             $
('#btndeleteshw').hide();            ////////////             $
('#rowName').hide();             $('#rowLocation').hide();             $
('#rowdept').hide();         }            function showin()         {
$('#tdid').hide();              $('#tdorginal').show();
$('#btnupdate').hide();              $('#btnInsert').show();             $
('#btndel').hide();             $('#btninsertshw').hide();             $
('#btnupdateshw').show();            $
('#btndeleteshw').show();            ////////////             $
('#rowName').show();             $('#rowLocation').show();             $
('#rowdept').show();              /////////////
document.getElementById("tbID").value='';
document.getElementById("tbName").value='';
document.getElementById("tbLocation").value='';
document.getElementById("tbLdept").value='1';                  empidbind();
}              function empidbind()
```

```
{               dbObj.transaction(function (tx)
{                   tx.executeSql('SELECT * from Employee_Table', [], function
(tx, results) {                      var len = results.rows.length, i;
document.getElementById("ddlid").innerHTML = '';                      var str =
'';                      for (i = 0; i < len; i++)
{                          str += "<option value=" + results.rows.item(i).id +
">" + results.rows.item(i).id + "</option>";
document.getElementById("ddlid").innerHTML += str;                            str
= '';                      }                  }, null);            });
}             function showupdte()          {             empidbind();
$('#tdorginal').hide();             $('#tdid').show();             $
('#btnupdate').show();             $('#btnInsert').hide();             $
('#btndel').hide();             $('#btninsertshw').show();             $
('#btnupdateshw').hide();             $('#btndeleteshw').show();             $
('#rowName').show();             $('#rowLocation').show();             $
('#rowdept').show();                        }             function updte() {
var id = document.getElementById("ddlid").value;             var name =
document.getElementById("tbName").value;             var location =
document.getElementById("tbLocation").value;             var did =
document.getElementById("tbLdept").value;
dbObj.transaction(function (tx) {                  tx.executeSql('update
Employee_Table set Name="' + name + '",Location="' + location + '",did=' + did +
' where id=' + id + '');             });
alldetails();          }             function alldetails()          {
dbObj.transaction(function (tx) {                  tx.executeSql('SELECT
e.id,e.Name,e.Location,d.dName,d.did FROM Employee_Table e inner join dept_Table
d on e.did=d.did ', [], function (tx, results) {                      var len =
results.rows.length, i;                      //
document.getElementById("tblGrid").innerHTML = '';                      $
("#tblGrid").find("tr:gt(0)").remove();                      var str = '';
for (i = 0; i < len; i++) {                          str += "<tr>";
str += "<td>" + results.rows.item(i).id + "</td>";                          str
+= "<td>" + results.rows.item(i).Name + "</td>";                          str +=
"<td>" + results.rows.item(i).Location + "</td>";                          str
+= "<td>" + results.rows.item(i).dName + "</td>";                          str
+= "</tr>";
document.getElementById("tblGrid").innerHTML += str;
str = '';                      }                  }, null);             });
}             dbObj.transaction(function (tx)
{             tx.executeSql('SELECT * from dept_Table', [], function (tx,
results) {                 var len = results.rows.length, i;
var str = '';                 for (i = 0; i < len; i++)
{                     str += "<option value=" + results.rows.item(i).did + ">"
+ results.rows.item(i).dName + "</option>";
document.getElementById("tbLdept").innerHTML += str;                      str =
'';                 }             }, null);         });
</script>  </head>  <body>      <p id="hh"></p>      <form id="frm1">
<table id="tblinsert">             <tr>                 <td>ID:</td>
<td id="tdorginal"><input type="text" id="tbID" /><span style="color:red">*ID
must be unique</span></td>                 <td id="tdid" style="display:none">
<select id="ddlid" onchange="myFunction()"></select>                 </td>
</tr>             <tr id="rowName">                 <td>Name:</td>
<td><input type="text" id="tbName" /></td>             </tr>             <tr
id="rowLocation">                 <td>Location:</td>
<td><input type="text" id="tbLocation" /></td>             </tr>
<tr id="rowdept">                 <td>Dept:</td>                 <td>
<select id="tbLdept"></select>                 </td>             </tr>
<tr>                             </tr>          </table>      </form>      <br
/>      <button id="btnInsert" onclick="Insert()"
style="color:green;display:block">Save</button>      <button id="btnupdate"
onclick="updte()" style="color:blue;display:none">Update</button>      <button
id="btndel" onclick="del()" style="color:red;display:none">Delete</button>
<br /><br />      <button id="btnupdateshw" onclick="showupdte()"
style="color:red">update Employee details</button>      <button
```

```
id="btndeleteshw" onclick="showdel()" style="color:blue">Delete Employee
details</button>      <button id="btninsertshw" onclick="showin()"
style="color:green;display:none">save Employee details</button>      <br
/><br />      <table id="tblGrid" cellpadding="10px" cellspacing="0" border="1">
<tr style="background-color:black;color:white;font-size:18px;">              <td
>                  ID              </td>              <td >
Name              </td>              <td >                  Location
</td>              <td >                 Department              </td>
</tr>      </table>      <br />          </body>  </html
```

For Create Database:-

```
var Database_Name = 'MyDatabase';          var Version = 1.0;          var
Text_Description = 'My First Web-SQL Example';          var Database_Size = 2 *
1024 * 1024;          var dbObj = openDatabase(Database_Name, Version,
Text_Description, Database_Size);
```

Creating transaction:-

```
dbObj.transaction(function (tx)   {  tx.executeSql('CREATE TABLE IF NOT EXISTS
Employee_Table (id unique, Name, Location,did)');
tx.executeSql('CREATE TABLE IF NOT EXISTS dept_Table (did unique, dName,estd)');
});
```

Execute Sql:-

```
dbObj.transaction(function (tx)   {  tx.executeSql('CREATE TABLE IF NOT EXISTS
Employee_Table (id unique, Name, Location,did)');
tx.executeSql('CREATE TABLE IF NOT EXISTS dept_Table (did unique, dName,estd)');
});
```

CRUD using indexedDB/WebSQL sample code

#indexeddb
#javascript
#websql

This sample code is part of an app I am working at. I am kind of newbie, so please give some constructive feedback. The code is based on this sample code: http://nparashuram.com/trialtool/#example=/IndexedDB/trialtool/index.html&selected=

For WebSQL compatibility use his IndexedDBShim (http://nparashuram.com/IndexedDBShim/)

I think it is much easier to pick this code and adapt it for your own scenario. You are welcome to come with improvements.

If this is difficult to read get the script from this link: https://www.dropbox.com/s/kc5rjp8dmwcl1ze/myIndexedDB.js

```
/* -------------------------- */
/* myindexedDB
/* -------------------------- */

var indexedDBModule = (function(){

//'global' index;
var i = 0;
//keep a refference to the database name in dbName
var dbName;

//prefixes of implementation that we want to Test
var indexedDB = window.indexedDB || window.mozIndexedDB ||
window.webkitIndexedDB || window.msIndexedDB;
//not used yet
var IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction ||
window.msIDBTransaction;
var IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange ||
window.msIDBKeyRange;

var init = function(db,store) {
    if (!indexedDB) {
        alert("Sorry!Your browser doesn't support IndexedDB");
    } else {
        initDB(db,store);
    }
};

var destroy = function(db){
    try {
        //request to delete the entire database
        if(!db || db=='')
            var dbDeleteRequest = indexedDB.deleteDatabase(DB_NAME);
        else var dbDeleteRequest = indexedDB.deleteDatabase(db);
        dbDeleteRequest.onsuccess = function(e) {
            alert('Database successfully deleted');
            /*
             * I found out that when I need to destroy the database and recreate
             * it, there are some issues. I tried to destroy database, create
database,
             * insert items, deleteAllItems, count the items in the objectStore
etc...
             * after destroying the database, a location.reaload() helps
refreshing these operations
             */
            location.reload();
        };
        dbDeleteRequest.onupgradeneeded = function(e) {
            console.log('Database upgrade needed ');
        };
        dbDeleteRequest.onerror = function(e) {
            console.log('Error deleting database ');
        };
        dbDeleteRequest.onblocked = function(e) {
            console.log('Deleting Database Blocked... Try closing the database
and then deleting it ');
        };
    } catch (e) {
        console.log(e);
        alert('Error on destroy -> deleteDatabase ');
    }
};

var createItem = function(id,data,store,update){
    try {
        //request to open the database
        var dbOpenRequest = indexedDB.open(dbName);
        dbOpenRequest.onsuccess = function(event) {
            var thisDB = dbOpenRequest.result;
            thisDB.onversionchange = function(e) {
                console.log('Version change triggered, so closing database
connection ' + e.oldVersion + ' ' + e.newVersion + ' ' +thisDB);
                thisDB.close();
            };
            try{
                var transaction = thisDB.transaction([store], "readwrite");
                transaction.oncomplete = function(e) {
                    console.log('transaction is complete ');
                };
                transaction.onabort = function(e) {
                    var error = e.target.error; // DOMError
                    if (error.name == 'QuotaExceededError') {
                        alert('createItem -> transaction['+store+'] -> onabort -
QuotaExceededError ');
                    }
                    console.log('transaction aborted ');
                };
                transaction.onerror = function(e) {
                    console.log('transaction error');
                };
                try {
                    var request;
                    var objectStore = transaction.objectStore(store);
                    if(update && update==true){
                        //request to UPDATE new entry in the object store
                        request = objectStore.put(data);
                    } else{
                        //request to ADD entry in the object store
                        request = objectStore.add(data);
                    }
                    request.onsuccess = function(event) {
                        //var element = event.target.result;
                        if(update && update == true)
```

```
                            console.log('The record with ID = ' + id + ' has
been updated in the database ');
                        else console.log('A new record with ID = ' + id + ' has
been added to the database ');
                    };
                    request.onerror = function(event) {
                        if (update && update == true){
                            console.log('Error. Could not update record with id
= ' + id);
                        }
                        else {
                            console.log('Error ' + event + ' This record is
already in the database. ');
                        }
                    };
                } catch (e) {
                    console.log('Could not operate on '+ '['+store+']' +'. You
may have to create it first ');
                };
            } catch(e){
                console.log('Error ' + e);
            };
        };
        dbOpenRequest.onupgradeneeded = function(e) {
            console.log('Database upgrade needed ');
            //not used
            //var db = dbOpenRequest.result;
            //var transaction = dbOpenRequest.transaction;
        };
        dbOpenRequest.onerror = function(e){
            console.log('Error ');
        };
        dbOpenRequest.onblocked = function(e) {
            console.log('Database open request blocked ');
        };
    } catch(e){
        console.log('Error ');
    };
};

var readItem = function(id,store) {
    try {
        var dbOpenRequest = indexedDB.open(dbName);
        dbOpenRequest.onsuccess = function(event) {
            var thisDB = dbOpenRequest.result;
            thisDB.onversionchange = function(e) {
                console.log('Version change triggered, so closing database
connection ' + e.oldVersion + ' ' + e.newVersion + ' ' +thisDB);
                thisDB.close();
            };
            try{
                var transaction = thisDB.transaction([store], "readonly");
                transaction.oncomplete = function(e) {
                    console.log('transaction is complete ');
                };
                transaction.onabort = function(e) {
                    console.log('transaction aborted ');
                };
                transaction.onerror = function(e) {
                    console.log('transaction error ');
                };

                try {
                    var objectStore = transaction.objectStore(store);
                    var request = objectStore.get(id);
                    request.onsuccess = function(event) {
                        var element = event.target.result;
                        if(element!==undefined){
                            console.log('Elemtent with id = ' + id + ' was
found\n' + JSON.stringify(element));
                        }
                        else{
                            console.log('Element with id = ' + id + ' does not
exist in the IndexedDB ');
                        }
                    };
                    request.onerror = function(event){
                        console.log('Error - could not request data from the
object store ' + event);
                    };
                } catch (e) {
                    console.log('Could not operate on '+ '['+store+']' +'. You
may have to create it first ');
                }
            } catch(e){
                console.log('Error ');
            }
        };
        dbOpenRequest.onupgradeneeded = function(e) {
            console.log('Database upgrade needed ');
            //var db = dbOpenRequest.result;
            //var transaction = dbOpenRequest.transaction;
        };
        dbOpenRequest.onerror = function(e){
            console.log('Error ');
        };
        dbOpenRequest.onblocked = function(e) {
            console.log('Database open request is blocked ');
        };
    } catch(e){
        console.log('Error ');
    };
};

var updateItem = function(id,data,store){
    createItem(id,data,store,true);
};

var deleteItem = function(id,store) {
    try {
        var dbOpenRequest = indexedDB.open(dbName);
        dbOpenRequest.onsuccess = function(event) {
            var thisDB = dbOpenRequest.result;
            thisDB.onversionchange = function(e) {
                console.log('Version change triggered, so closing database
connection ' + e.oldVersion + ' ' + e.newVersion + ' ' +thisDB);
                thisDB.close();
            };
            try{
                //attempt to create a transaction
                var transaction = thisDB.transaction([store], "readwrite");
                transaction.oncomplete = function(e) {
                    console.log('transaction is complete ');
                };
                transaction.onabort = function(e) {
                    console.log('transaction aborted ');
                };
                transaction.onerror = function(e) {
                    console.log('Transaction error -> [deleteItem] ');
                };

                try {
                    var objectStore = transaction.objectStore(store);
                    var getRequest = objectStore.get(id);
                    getRequest.onsuccess = function(e){
                        if (e.target.result != null) {
                            console.log('Object Exist');
                            //request to delete entry from the object store
                            var request = objectStore.delete(id);
                            request.onsuccess = function(event){
                                console.log('Entry id = ' + id + ' was deleted
from the database');
                            };
                            request.onerror = function(event){
                                console.log('Error [deleteItem] ->
delete('+id+') ');
                            };
                        }else{
                            console.log('Object doesn\'t exist ');
                        }
                    };
                    getRequest.onerror = function(e){
                        console.log('Error [deleteItem] -> delete('+id+') ');
                    };
                } catch (e) {
                    console.log('Could not operate on '+ '['+store+']' +'. You
may have to create the entry first ');
                }
            } catch(e){
                console.log('Error ');
            }
        };
        dbOpenRequest.onupgradeneeded = function(e) {
            console.log('Database upgrade needed ');
            //var db = dbOpenRequest.result;
            //var transaction = dbOpenRequest.transaction;
        };
        dbOpenRequest.onerror = function(e){
            console.log('Error ');
        };
        dbOpenRequest.onblocked = function(e) {
            console.log('Database open request is blocked ');
        };
    } catch(e){
        console.log('Error ');
    }
};

var insertItems = function(store,obj,length){
    //inserts the same item multiple times in the same transaction.
    //just a test
    try {
        //request to open the database
        var dbOpenRequest = indexedDB.open(dbName);
        dbOpenRequest.onsuccess = function(event) {
            var thisDB = dbOpenRequest.result;
            thisDB.onversionchange = function(e) {
                console.log('Version change triggered, so closing database
connection ' + e.oldVersion + ' ' + e.newVersion + ' ' +thisDB);
                thisDB.close();
            };
            try{
                //creating transaction to open an object store in "readwrite"
mode
                var transaction = thisDB.transaction([store], 'readwrite');
                transaction.oncomplete = function(e) {
                    console.log('transaction is complete ');
                };
                transaction.onabort = function(e) {
                    var error = e.target.error; // DOMError
                    console.log('transaction aborted ' + error);
                };
                transaction.onerror = function(e) {
                    console.log('transaction error ');
                };

                try {
                    var request;
                    var objectStore = transaction.objectStore(store);

                    putNext();

                    function putNext() {
                        if (i<length) {
                            objectStore.add(JSON.parse(obj)).onsuccess =
putNext;
                            //console.log('add entry with id = ' + i);
                            ++i;
                        } else {
                            // complete
                            console.log('insertItems -> DONE');
                            //callback();
                        }
                    }
                } catch (e) {
                    console.log('Could not operate on '+ '['+store+']' +'. You
may have to create it first ');
                };
            } catch(e){
                console.log('Error ');
            };
            //when done
            //callback();
        };
        dbOpenRequest.onupgradeneeded = function(e) {
            console.log('Database upgrade needed ');
        };
        dbOpenRequest.onerror = function(e){
            console.log('Error ');
        };
        dbOpenRequest.onblocked = function(e) {
            console.log('Database open request blocked ');
        };
    } catch(e){
        console.log('Error ');
    };
};

var readAllItems = function(store) {
    try {
        var dbOpenRequest = indexedDB.open(dbName);
        dbOpenRequest.onsuccess = function(event) {
            var thisDB = dbOpenRequest.result;
            thisDB.onversionchange = function(e) {
                console.log('Version change triggered, so closing database
connection ' + e.oldVersion + ' ' + e.newVersion + ' ' +thisDB);
```

```
                thisDB.close();
            };
            try{
                var transaction = thisDB.transaction([store], "readonly");
                transaction.oncomplete = function(e) {
                    console.log('transaction is complete ');
                };
                transaction.onabort = function(e) {
                    console.log('transaction aborted ');
                };
                transaction.onerror = function(e) {
                    console.log('transaction error ');
                };
                try {
                    var objectStore = transaction.objectStore(store);
                    //request cursor on the object store
                    var request = objectStore.openCursor();
                    var count = 0;
                    request.onsuccess = function(event) {
                        var cursor = event.target.result;
                        if (cursor) {
                            console.log('id = ' + cursor.key + ' is ' +
cursor.value);
                            count++;
                            cursor['continue']();
                        } else console.log('Total entries = ' + count);
                    };
                    request.onerror = function(e){
                        console.log('transaction error on readAllItems ->
oenCursor -> onerror ');
                    };
                } catch (e) {
                    console.log('Could not operate on '+ '['+store+']' +'. You
may have to create it first ');
                }
            } catch(e){
                console.log('Error ');
            }
        };
        dbOpenRequest.onupgradeneeded = function(e) {
            console.log('Database upgrade needed ');
            //var db = dbOpenRequest.result;
            //var transaction = dbOpenRequest.transaction;
        };
        dbOpenRequest.onerror = function(e){
            console.log('Error ');
        };
        dbOpenRequest.onblocked = function(e) {
            console.log('Database open request is blocked ');
        };
    } catch(e){
        console.log('Error ');
    }
};

var deleteAllItems = function(store,count){
    try {
        var dbOpenRequest = indexedDB.open(dbName);
        dbOpenRequest.onsuccess = function(event) {
            var thisDB = dbOpenRequest.result;
            thisDB.onversionchange = function(e) {
                console.log('Version change triggered, so closing database
connection ' + e.oldVersion + ' ' + e.newVersion + ' ' +thisDB);
                thisDB.close();
            };
            try{
                var transaction = thisDB.transaction([store], "readwrite");
                transaction.oncomplete = function(e) {
                    console.log('transaction is complete ');
                };
                transaction.onabort = function(e) {
                    console.log('transaction aborted ');
                };
                transaction.onerror = function(e) {
                    console.log('transaction error ');
                };
                try {
                    //open object store
                    var objectStore = transaction.objectStore(store);
                    try{
                        //count items in the object store
                        var requestCount = objectStore.count();
                        requestCount.onsuccess = function(event){
                            if(requestCount.result){
                                console.log('There are '+requestCount.result+'
entries in the database.');
                            }
                            else{
                                if(requestCount.result!=0){
                                    //request to clear the object store
                                    var request = objectStore.clear();
                                    request.onsuccess = function(event) {
                                        console.log('No more entries to delete
in the object store!');

                                        //reset 'global' index
                                        i = 0;
                                    };
                                    request.onerror = function(event) {
                                        console.log('Error clearing the object
store! ' + event);
                                    };
                                }
                                else {
                                    console.log('Object store is already
empty');
                                }
                            }
                        };
                        requestCount.onerror = function(event){
                            console.log('Error counting elements in the store
');
                        };
                    } catch(e){
                        console.log('Error ');
                    }
                } catch (e) {
                    console.log('Could not operate on '+ '['+store+']' +'. You
may have to create it first ');
                }
            } catch(e){
                console.log('Error ');
            }
        };
        dbOpenRequest.onupgradeneeded = function(e) {
            console.log('Database upgrade needed ');
        };
        dbOpenRequest.onerror = function(e){
            console.log('Error ');
        };
        dbOpenRequest.onblocked = function(e) {
            console.log('Database open request is blocked ');
        };
    } catch(e){
        console.log('Error ');
    }
};

/************ helper functions ****************/
function initDB(db,store) {
    try {
        var dbOpenRequest = indexedDB.open(db,1);
        dbOpenRequest.onsuccess = function(event) {
            var thisDB = dbOpenRequest.result;
            // Need to create this variable since the variable db is assigned to
other things later
            thisDB.onversionchange = function(e) {
                console.log('Version change triggered, so closing database
connection ' + e.oldVersion + ' ' + e.newVersion + ' ' + thisDB);
                thisDB.close();
            };
            console.log('Database '+db+' is created with objectStore '+store);
            dbName = db;
        };
        dbOpenRequest.onupgradeneeded = function(e) {
            var thisDB = dbOpenRequest.result;
            var transaction = dbOpenRequest.transaction;
            try {
                var storeOptions = {autoIncrement : true}; //possible pass
storeOptions in init and further to initDB
                var objectStore = thisDB.createObjectStore(store,storeOptions);

            } catch (e) {
                console.log('Error occured while creating object store ');
            }
        };
        dbOpenRequest.onerror = function(e) {
            console.log('Error ');
        };
        dbOpenRequest.onblocked = function(e) {
            console.log('Blocked ');
        };
    } catch (e){
        console.log('Error ');
    }
}
/******** helpers **********/

return{
    init : init,
    destroy : destroy,
    createItem : createItem,
    readItem : readItem,
    updateItem : updateItem,
    deleteItem : deleteItem,
    insertItems : insertItems,
    readAllItems : readAllItems,
    deleteAllItems : deleteAllItems
};
}());
```

https://coderwall.com/p/6it6hw/crud-using-indexeddb-websql-sample-code

Create a CRUD web app using JQuery Mobile & WebSQL

Anele 'Mashy' Mbanga
28 Mar 2015CPOL
Demonstrates how to use SQLite database for mobile development

Introduction

I seek to demonstrate how to one can create a CRUD web mobile application using JQuery Mobile and WebSQL. Whilst the future of WebSQL seems to be dwingling due to it no longer being supported by W3C as more is being put on IndexedDB. WebSQL is a relational database that sits on a browser and IE and FireFox does not support it unfortunately but any webkit related web browser has its functionality.

To make this work, we will create a javascript file to handle all the database operations, like creating a database, creating a table, creating indexes, inserting records, updating records and deleting records and returning all records from a table.

Some silly assumptions: I will assume that you know some WebSQL and how to perform SQL queries. I will also assume that you know some JQuery Mobile Development and how to link javascript files via code to your html files.

Download AddressBookSQL.zip

Background

CRUD applications are developed for basically three things, CR - create records, U - update records and D - delete records. In the past, I have written a couple of articles that basically spoke about how one can perform those functionalities using different frameworks. These are:

1. CRUD using LocalStorage
2. CRUD using single JSON with PHP for a web server

and now this framework, using WebSQL. For more details about WebSQL, you can read this wiki document here.

Using the code

WEBSQL.JS - database maintenance

Using a separation of concerns, I have separated the code that reads and writes to the database in a single javascript file. This ensures that only objects can be passed to the WebSQL file to manage the record manipulation. I will first talk about the attached **websql.js** file attached in the source code and how you can use it in your own apps.

```
// variables for database table maintenance
var DB_REAL = "REAL";
var DB_INTEGER = "INTEGER";
var DB_BLOB = "BLOB";
var DB_TEXT = "TEXT";
var DB_FLOAT = "FLOAT";
var DB_NUMERIC = "NUMERIC";
```

These are the various field types that one can create for the WebSQL database, however the one most used in the example application here is DB_TEXT. I declare these as constants that will be accessed by my app code.

```
function Left(str, n) {
    // return a left part of a string
    var s = str + '';
    var iLen = s.length;
    if (n <= 0) {
        return "";
        } else if (n >= iLen) {
        return str;
        } else {
        return s.substr(0, n);
    }
}
```

This is helper function that will return the Left part of a string, e.g. Left("My Name", 2) will return "My".

```
function Len(str) {
    // return the length of a string
    if (typeof (str) === 'object') {
        return str.length;
    }
    str += '';
    return str.length;
}
```

This is also another helper function to return the length of a string, e.g. Len("Anele Mbanga") will return 12. It is used by the WebSQL script file to maintain the databases.

Creating and Opening the database

```
function SqlOpenDb(shortName, version, displayName, maxSize) {
    // code to open the database and returns a variable, one can open different
    // databases, database size is 1MB, increase dbsize to <= 5
    var db, dbsize = 1;
    try {
        if (!window.openDatabase) {
            return 0;
            } else {
            if (typeof (shortName) === 'undefined') {
                return 0;
            }
            if (typeof (version) === 'undefined') version = "";
            if (typeof (displayName) === 'undefined') displayName = shortName;
            if (typeof (maxSize) === 'undefined') maxSize = dbsize * (1024 *
1024);
            db = openDatabase(shortName, version, displayName, maxSize);
        }
        } catch (e) {
        return 0;
    }
    return db;
}
```

The SqlOpenDb script is the main script that creates and opens the database if it does not exist. The maximum database size for websql as on writing is 5MB. The usage of this function is that an object is returned by this function which will hold all functions to pass to the database. If the browser does not support WebSQL, a zero is returned.

To open the database, one can call this function like this...

```
var db = SqlOpenDb("AddressBook");
```

this will create a 1MB database on the browser for storing all your records. After opening a database you need to create the tables that will store the information and then insert records therein. I forgot to mention, the way the database maintenance functions are called here is though using a **deferred** method of JQuery. I found this helpful not to create extra functions to call for the **success** or **failure** of execute methods. This then means that a database transaction is called and when thats done other functions can be performed depending on success or failure. You will see this soon.

Creating a Table

```
function SqlCreateTable(db, TableName, FieldsAndTypes, PrimaryKey,
AutoIncrement) {
    // code to create a table in the websql database
    // fieldsandtypes is a json object
    // autoincrement is the field name to autoincrement
    var sb = "(";
    for (item in FieldsAndTypes) {
        sb += "[" + item + "] " + FieldsAndTypes[item];
        if (item == PrimaryKey) {
            sb += " NOT NULL PRIMARY KEY";
        }
        if (item == AutoIncrement) {
            sb += " AUTOINCREMENT";
        }
        sb += ", ";
    }
    sb = Left(sb, (Len(sb) - 2));
    sb += ")";
    sb = "CREATE TABLE IF NOT EXISTS [" + TableName + "] " + sb + ";";
    return Execute(db, sb);
}
```

This method above is the method used to create a table in the database. This method checks if the database table exists and if not its created. As you can see, the **Left** and **Len** method defined previously are used in this function.

To create a table, you pass the database object that was used in SqlOpenDb function, then the table name to create and then the **FieldsAndTypes**, the field name that will be the primarykey and the fieldname that will autoincrement. Remember, javascript is case sensitive. This function loops through the object passed, gets the field names, field types and builds up the *CREATE TABLE* statement. If the PrimaryKey given matches any of the given field names this will be marked as so and also same as the auto increment field.

Specifying Field Names and Types

When creating tables, you need to specify the field names and types first before calling the method. This is actually easier that you think, however if you checked the source code above, you will know that you just pass an object that can be looped. Lets define the object to pass, in this example below. We will create a contact table with some few fields.

```
var FT = {};

FT.FullName = DB_TEXT;

FT.MobileNumber = DB_NUMERIC;

SqlCreateTable(db, "Contacts", FT, "FullName", "");
```

Putting that in a function will easily create your table, and like I said, if it does not exist.

Insert a Record

We have opened a database, created a table and now its time to add records to it. The same approach to create a table is followed here.

```
function SqlInsertRecord(db, tblName, tblRecord) {
    // code to insert a record into the database
    // fields are passed as parameters
    var qry, flds = "", vals = "", avals = [];
    for (var key in tblRecord) {
        flds += "[" + key + "],";
        vals += "?,";
        avals.push(tblRecord[key]);
    }
    flds = Left(flds, Len(flds) - 1);
    vals = Left(vals, Len(vals) - 1);
    qry = "INSERT INTO [" + tblName + "] (" + flds + ") VALUES (" + vals + ");";
    return Execute(db, qry, avals);
}
```

SqlInsertRecord gets passed the database object, the table name to update and then the record object to add. One defines the tblRecord like this.

```
var tblRec = {};

tblRec.FullName = "Anele Mbanga";

tblRec.MobileNumber = 123456789

SqlInsertRecord(db, "Contacts", tblRec);
```

As you can see, the database methods are fairly straightforward but what will differ will be the objects passed to them to maintain the records in your database.

Get Records from a table

Now that you have opened a database, created a table and added up a record, how do you read it? Records read from a table are returned as a json object. You can read a single record or return all records from a table.

```
function SqlGetRecordWhere(db, tblName, tblWhere) {
    // code to get a record from database using a where clause
    // tblWhere should be objects
    var qry = "", vals = "", avals = [];
    for (item in tblWhere) {
        vals += "[" + item + "] = ? AND ";
        avals.push(tblWhere[item]);
    }
    vals = Left(vals, Len(vals) - 5);
    qry = "SELECT * FROM [" + tblName + "] WHERE " + vals + ";";
    return Execute(db, qry, avals);
}
```

SqlGetRecordWhere will return a single / multiple records from a table depending on your **WHERE** clause. The tblWhere clause is also an object that we parse to return our records. This uses a **parametised** query to extract the records from the database. Also you pass the database object, the table name and the where object clause. For example, to return records with FullName = "Anele Mbanga", you would define a where clause like this.

```
var rWhere = {};

rWhere.FullName = "Anele Mbanga";

SqlGetRecordWhere(db, "Contacts", rWhere);

// return all records from a table sorted by primary key

function SqlGetRecords(db, TableName, PrimaryKey) {
    // return all records from a table ordered by primary key
    var qry = "SELECT * FROM [" + TableName + "] ORDER BY [" + PrimaryKey +"]";
    return Execute(db, qry);
};
```

To return all records from the database then our code would be:

```
SqlGetRecords(db, "Contacts", "FullName");
```

Update an existing record

To update an existing record, we pass SqlUpdateRecordWhere some few objects, 1, the fields we want to update and the WHERE clause like above, the database object and the table name.

```
function SqlUpdateRecordWhere(db, tblName, tblRecord, tblWhere) {
    // code to update a record on a database
    // tblRecord and tblWhere should be objects
    var qry = "", vals = "", wvals = "", avals = [];
    for (item in tblRecord) {
        vals += "[" + item + "] = ?,";
        avals.push(tblRecord[item]);
    }
    for (item in tblWhere) {
        wvals += "[" + item + "] = ? AND ";
        avals.push(tblWhere[item]);
    }
    vals = Left(vals, Len(vals) - 1);
    wvals = Left(wvals, Len(wvals) - 5);
    qry = "UPDATE [" + tblName + "] SET " + vals + " WHERE " + wvals + ";";
    return Execute(db, qry, avals);
}
```

This creates the UPDATE statement to update the table and then runs Execute to update the table. To update the MobileNumber of our contact for example, we would define our process like this.

```
var rM = {}, rW = {};

rM.MobileNumber = 98765432

rW.FullName = "Anele Mbanga";

SqlUpdateRecordWhere(db, "Contacts", rM, rW);
```

Deleting existing record(s).

To delete existing records, you also pass the object to delete just like the examples above.

```
function SqlDeleteRecordWhere(db, tblName, tblWhere) {
    // delete a record from a table using a where clause
    // pass the where fields as parameters
    var qry, wvals = "", avals = [];
    for (item in tblWhere) {
        wvals += "[" + item + "] = ? AND ";
        avals.push(tblWhere[item]);
    }
    // remove last ' AND '
    wvals = Left(wvals, Len(wvals) - 5);
    qry = "DELETE FROM [" + tblName + "] WHERE " + wvals + ";";
    return Execute(db, qry, avals);
};
```

The function loops through each field specified and builds the DELETE statement. An example to delete a record would be:

```
var dR = {};

dR.FullName = "Anele Mbanga";

SqlDeleteRecordWhere(db, "Contacts", dR);
```

Returning Distinct Fields

At times you might want to return a distinct field from the database,

```
function SqlGetDistinctField(db, TableName, FldName) {
    // return distinct records from a table
    var qry = "SELECT DISTINCT [" + FldName + "] FROM [" + TableName + "] ORDER
BY [" + FldName +"]";
    return Execute(db, qry);
};
```

SqlGetDistinctField will help with that. As you can see you can define your own SQL Statements that you can pass to the Execute function. Let's explain the Execute function then.

The Execute method is the main function that executes the SQL statements in the database in our deferred way here. Let me explain it.

```
function Execute(db, qry, args){
    // execute a query against the database using defer
    if (typeof (args) === 'undefined') args = [];
    return $.Deferred(function (d) {
        db.transaction(function (tx) {
            tx.executeSql(qry, args, successWrapper(d), failureWrapper(d));
        });
    });
};
```

The execute method gets passed the database object, the query string we want to execute and then an array that has the arguements that we want to process. This returns a JQuery Deferred object that is then processed with **successWrapper** and **failureWrapper** when our operation is successful or fails respecively.

```
function successWrapper(d) {
    // when sql query succeeds
    return (function (tx, data) {
        d.resolve(data)
    })
};

function failureWrapper(d) {
    // when sql query fails
    return (function (tx, error) {
        d.reject(error)
    })
};
```

When our database operation is succesful, a **ResultSet** is returned as the data object and this can be passed and processed. In most cases, you will want to return all the data of the resultset as a simple object. As I wanted my code to be compatible easily whether I developed for LocalStorage or the Php ajax calls as demonstrated in the previous articles, I followed the same partern here and convert it to a JSON array.

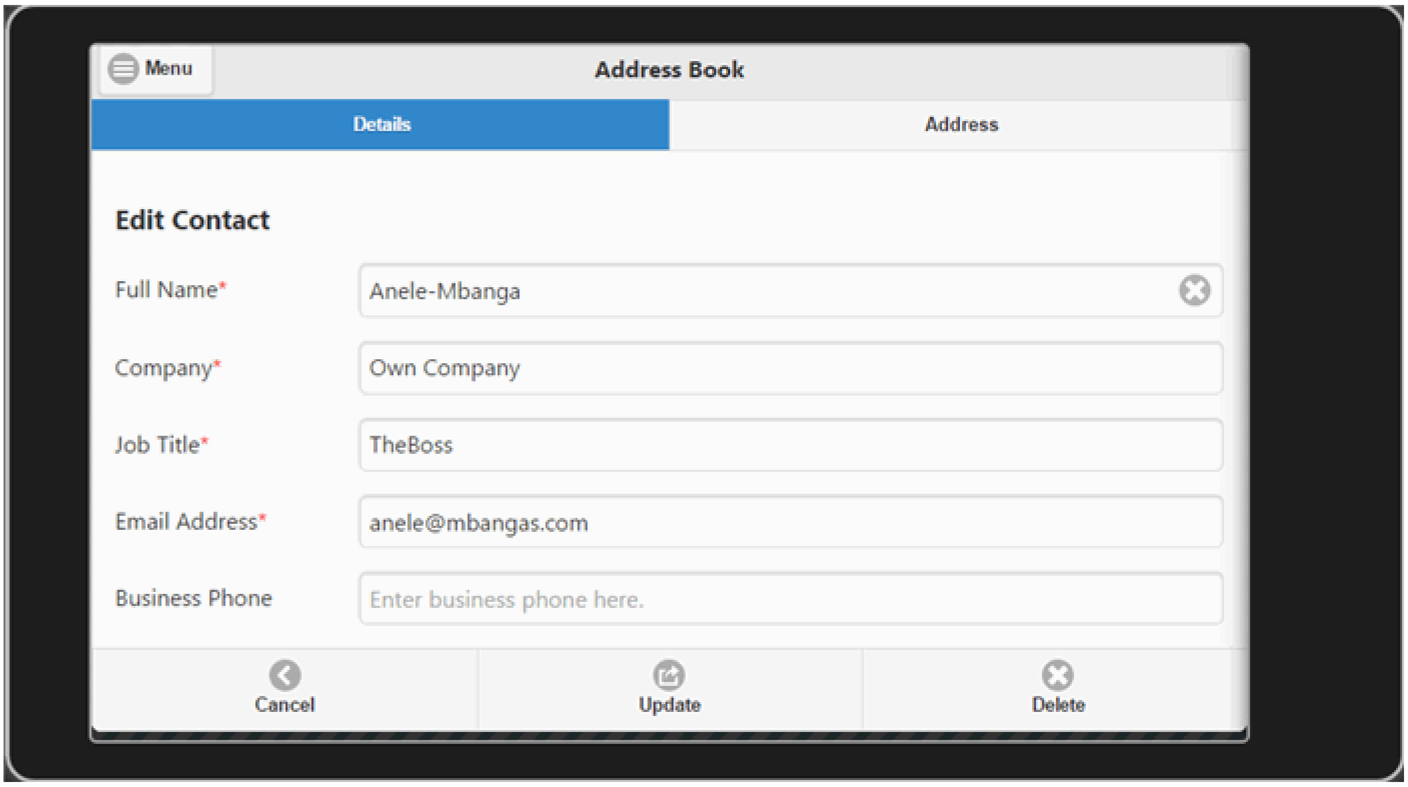
```
function ResultSetToJSON(results, PrimaryKey) {
    // process data returned by successWrapper;
    // return it as a json object using primary key as key
    var Records = {};
    var len = results.rows.length - 1, priKey, i, row;
    // loop through each row
    for (i = 0; i <= len; i++) {
        // get the row
        row = results.rows.item(i);
        // get the primary key
        priKey = row[PrimaryKey];
        // cleanse the primary key
        priKey = priKey.split(' ').join('-');
        // set row to object using primary key
        Records[priKey] = row;
    }
    return Records;
}
```

with this function, data returned with successWrapper, is passed to **ResultSetToJSON** specifying the primary key to return a JSON object that will be accessible with each primary key of all records in your table.

We have in detail explained the database operations, now lets create our user interface to use these operations. Let's create a simple Address Book CRUD app.

The User Interface

Figure 1 - Create/Update a new Contact Details



Edit Contact

Full Name*	Anele-Mbanga
Company*	Own Company
Job Title*	TheBoss
Email Address*	anele@mbangas.com
Business Phone	Enter business phone here.

Figure 2 - Contact Address Details

Figure 3 - Address Book Listing

The figures above indicate the user interface we will create to demonstrate our CRUD app. We want to capture address details and save them and also be able to delete an address when we want.

HTML Definition - Add Contacts Screen

```
<div id="pgAddContact" data-role="page">
                <div data-position="left" data-display="reveal" data-position-
fixed="true" id="pgAddContactPnl" data-role="panel">
                    <ul data-role="listview" id="pgAddContactPnlLV">
                        <li ><a data-transition="slide"
href="#pgAddContact">New</a></li>
                        <li ><a data-transition="slide"
href="#pgContactMindCity">Relationships > City</a></li>
                        <li ><a data-transition="slide"
href="#pgRptContact">Report</a></li>
                        <li ><a data-transition="slide"
href="#pgContact">Back</a></li>
                    </ul>
                </div>

                <header id="pgAddContactHdr" data-role="header" data-
position="fixed">
                    <h1>Address Book</h1>
                    <a data-role="button" id="pgAddContactMenu"
href="#pgAddContactPnl" data-icon="bars" class="ui-btn-left">Menu</a>
                    <div id="pgAddContactHdrNavBar" data-role="navbar">
                        <ul>
                            <li><a href="#" data-href="pgAddContactDetails"
id="pgAddContactDetailsBtn" class="ui-btn-active">Details</a>
                            </li>
                            <li><a href="#" data-href="pgAddContactAddress"
id="pgAddContactAddressBtn">Address</a>
                            </li>
                        </ul>
                    </div>
                </header>
                <div id="pgAddContactCnt" data-role="content">
```

```
                    <h3>Add Contact</h3><div id="pgAddContactDetails" 
class="tab-content">
                        <div data-role="fieldcontain">
                            <label for="pgAddContactFullName">Full Name<span 
style='color:red;'>*</span></label>
                            <input title="Enter full name here." type="text" 
name="pgAddContactFullName" id="pgAddContactFullName" placeholder="Enter full 
name here." autocomplete="off" data-clear-btn="true" class="required"></input>
                        </div>
                        <div data-role="fieldcontain">
                            <label for="pgAddContactCompany">Company<span 
style='color:red;'>*</span></label>
                            <input title="Enter company here." type="text" 
name="pgAddContactCompany" id="pgAddContactCompany" placeholder="Enter company 
here." autocomplete="off" data-clear-btn="true" class="required"></input>
                        </div>
                        <div data-role="fieldcontain">
                            <label for="pgAddContactJobTitle">Job Title<span 
style='color:red;'>*</span></label>
                            <input title="Enter job title here." type="text" 
name="pgAddContactJobTitle" id="pgAddContactJobTitle" placeholder="Enter job 
title here." autocomplete="off" data-clear-btn="true" class="required"></input>
                        </div>
                        <div data-role="fieldcontain">
                            <label for="pgAddContactEmailAddress">Email 
Address<span style='color:red;'>*</span></label>
                            <input title="Enter email address here." 
type="email" name="pgAddContactEmailAddress" id="pgAddContactEmailAddress" 
placeholder="Enter email address here." autocomplete="off" data-clear-btn="true"
class="required"></input>
                        </div>
                        <div data-role="fieldcontain">
                            <label for="pgAddContactBusinessPhone">Business 
Phone</label>
                            <input type="tel" pattern="\d\d\d \d\d\d \d\d\d\d" 
name="pgAddContactBusinessPhone" id="pgAddContactBusinessPhone" 
placeholder="Enter business phone here." autocomplete="off" data-clear-
btn="true"></input>
                        </div>
                        <div data-role="fieldcontain">
                            <label for="pgAddContactBusinessFax">Business 
Fax</label>
                            <input type="tel" pattern="\d\d\d \d\d\d \d\d\d\d" 
name="pgAddContactBusinessFax" id="pgAddContactBusinessFax" placeholder="Enter 
business fax here." autocomplete="off" data-clear-btn="true"></input>
                        </div>
                        <div data-role="fieldcontain">
                            <label for="pgAddContactMobilePhone">Mobile 
Phone</label>
                            <input type="tel" pattern="\d\d\d \d\d\d \d\d\d\d" 
name="pgAddContactMobilePhone" id="pgAddContactMobilePhone" placeholder="Enter 
mobile phone here." autocomplete="off" data-clear-btn="true"></input>
                        </div>
                        <div data-role="fieldcontain">
                            <label for="pgAddContactHomePhone">Home 
Phone</label>
                            <input type="tel" pattern="\d\d\d \d\d\d \d\d\d\d" 
name="pgAddContactHomePhone" id="pgAddContactHomePhone" placeholder="Enter home 
phone here." autocomplete="off" data-clear-btn="true"></input>
                        </div>
                    </div>
                    <div id="pgAddContactAddress" class="tab-content">
                        <div data-role="fieldcontain">
```

```
                            <label for="pgAddContactStreetAddress1">Street
Address 1</label>
                            <input type="text" name="pgAddContactStreetAddress1"
id="pgAddContactStreetAddress1" placeholder="Enter street address 1 here."
autocomplete="off" data-clear-btn="true"></input>
                        </div>
                        <div data-role="fieldcontain">
                            <label for="pgAddContactStreetAddress2">Street
Address 2</label>
                            <input type="text" name="pgAddContactStreetAddress2"
id="pgAddContactStreetAddress2" placeholder="Enter street address 2 here."
autocomplete="off" data-clear-btn="true"></input>
                        </div>
                        <div data-role="fieldcontain">
                            <label for="pgAddContactCity">City</label>
                            <input type="text" name="pgAddContactCity"
id="pgAddContactCity" placeholder="Enter city here." autocomplete="off" data-
clear-btn="true"></input>
                        </div>
                        <div data-role="fieldcontain">
                            <label for="pgAddContactState">State</label>
                            <input type="text" name="pgAddContactState"
id="pgAddContactState" placeholder="Enter state here." autocomplete="off" data-
clear-btn="true"></input>
                        </div>
                        <div data-role="fieldcontain">
                            <label for="pgAddContactProvince">Province</label>
                            <input type="text" name="pgAddContactProvince"
id="pgAddContactProvince" placeholder="Enter province here." autocomplete="off"
data-clear-btn="true"></input>
                        </div>
                        <div data-role="fieldcontain">
                            <label for="pgAddContactPostalCode">Postal
Code</label>
                            <input type="number" name="pgAddContactPostalCode"
id="pgAddContactPostalCode" placeholder="Enter postal code here."
autocomplete="off" data-clear-btn="true"></input>
                        </div>
                    </div>
                </div>

                <footer id="pgAddContactFtr" data-role="footer" data-
position="fixed">
                    <div id="pgAddContactFtrNavBar" data-role="navbar">
                        <ul>
                            <li><a id="pgAddContactBack" data-icon="carat-
l">Cancel</a>
                            </li>
                            <li><a type="submit" id="pgAddContactSave" data-
icon="action">Save</a>
                            </li>
                        </ul>
                    </div>
                </footer></div>
```

The past two articles discussed in detail how to create the views and the same approach has been followed here. This view has a left panel that shows when the menu is selected. The footer has a navigation bar with Cancel and Save button. Cancel takes one to the Contacts listing whilst Save will insert a new Contact to the database.

The code that gets executed when the save button is clicked on the Add Contact page does this:

JavaScript - Add Contact

```
// Save click event on Add page
            $('#pgAddContactSave').on('click', function (e) {
                e.preventDefault();
                e.stopImmediatePropagation();
                // save the Contact
                var ContactRec;
                //get form contents into an object
                ContactRec = pgAddContactGetRec();
                //save object to SQL
                app.addContact(ContactRec);
            });
```

The data input by end user is read into an object and stored into pgAddContactGetRec, this is later passed to app.addContact which inserts the record to the database.

```
// add a new record to SQL storage.
        app.addContact = function (ContactRec) {
            // define a record object to store the current details
            var FullName = ContactRec.FullName;
            // cleanse the record key of spaces.
            FullName = FullName.split(' ').join('-');
            ContactRec.FullName = FullName;
            // store the json object in the database
            $.when(SqlInsertRecord(dbAddressBook, "Contact",
ContactRec)).done(function () {
                //show a toast message that the record has been saved
                toastr.success('Contact record successfully saved.', 'Address
Book');
                //find which page are we coming from, if from sign in go back to
it
                var pgFrom = $('#pgAddContact').data('from');
                switch (pgFrom) {
                    case "pgSignIn":
                    $.mobile.changePage('#pgSignIn', {transition:
pgtransition});
                    break;
                    default:
                    // clear the edit page form fields
                    pgAddContactClear();
                    //stay in the same page to add more records
                }
                }).fail(function (err) {
                //show a toast message that the record has not been saved
                toastr.success('Contact record NOT successfully saved.',
'Address Book');
            });
        };

// store the json object in the database
            $.when(SqlInsertRecord(dbAddressBook, "Contact",
ContactRec)).done(function () {
```

The crucial line that does the magic of inserting the record to the database is the one above. Because we are using the deferred method of websql maintenance, we call the database method to execute passing it inbetween **$.when .done** and **.fail** statements. This basically means that when the SqlInsertRecord method is done executing run the code in the function after done, if there is a failure perform the action in the failure. In this case, when the web methods finalises, show a toast that the message was saved, if failed, show a toast that the record could not be saved.

HTML Definition - Update Contact Screen

```
<div data-url="FullName" id="pgEditContact" data-role="page">
                    <div data-position="left" data-display="reveal" data-
position-fixed="true" id="pgEditContactPnl" data-role="panel">
                        <ul data-role="listview" id="pgEditContactPnlLV">
                            <li ><a data-transition="slide"
href="#pgAddContact">New</a></li>
                            <li ><a data-transition="slide"
href="#pgContactMindCity">Relationships > City</a></li>
                            <li ><a data-transition="slide"
href="#pgRptContact">Report</a></li>
                            <li ><a data-transition="slide"
href="#pgContact">Back</a></li>
                        </ul>
                    </div>

                    <header id="pgEditContactHdr" data-role="header" data-
position="fixed">
                        <h1>Address Book</h1>
                        <a data-role="button" id="pgEditContactMenu"
href="#pgEditContactPnl" data-icon="bars" class="ui-btn-left">Menu</a>
                        <div id="pgEditContactHdrNavBar" data-role="navbar">
                            <ul>
                                <li><a href="#" data-href="pgEditContactDetails"
id="pgEditContactDetailsBtn" class="ui-btn-active">Details</a>
                                </li>
                                <li><a href="#" data-href="pgEditContactAddress"
id="pgEditContactAddressBtn">Address</a>
                                </li>
                            </ul>
```

```
                            <div data-role="fieldcontain">
                                <label
for="pgEditContactProvince">Province</label>
                                <input type="text" name="pgEditContactProvince"
id="pgEditContactProvince" placeholder="Enter province here." autocomplete="off"
data-clear-btn="true"></input>
                            </div>
                            <div data-role="fieldcontain">
                                <label for="pgEditContactPostalCode">Postal
Code</label>
                                <input type="number"
name="pgEditContactPostalCode" id="pgEditContactPostalCode" placeholder="Enter
postal code here." autocomplete="off" data-clear-btn="true"></input>
                            </div>
                        </div>
                    </div>

                    <footer id="pgEditContactFtr" data-role="footer" data-
position="fixed">
                        <div id="pgEditContactFtrNavBar" data-role="navbar">
                            <ul>
                                <li><a id="pgEditContactBack" data-icon="carat-
l">Cancel</a>
                                </li>
                                <li><a type="submit" id="pgEditContactUpdate"
data-icon="action">Update</a>
                                </li>
                                <li><a id="pgEditContactDelete" data-
icon="delete">Delete</a>
                                </li>
                            </ul>
                        </div>
                    </footer></div>
```

To access the Contact Update screen, one needs to select a Contact from the ListView listing and this screen will appear. A user can then Update the Contact details or select Delete to delete a contact.

JavaScript - Update Contact

When a user updates a contact, the details of the screen are read and saved as an object which gets passed to the update method inside our app.js files.

```
// Update click event on Edit Page
            $('#pgEditContactUpdate').on('click', function (e) {
                e.preventDefault();
                e.stopImmediatePropagation();
                // save the Contact
                var ContactRecNew;
                //get contents of Edit page controls
                ContactRecNew = pgEditContactGetRec();
                //save updated records to SQL
                app.updateContact(ContactRecNew);
            });
```

As soon as the Update button is clicked, the contact details are saved with app.updateContact, lets look at what app.updateContact does.

```
//update an existing record and save to SQL
        app.updateContact = function (ContactRec) {
            // lookup specific Contact
            var FullName = ContactRec.FullName;
            //cleanse the key of spaces
            FullName = FullName.split(' ').join('-');
            //define the record to update
            var ContactUpdate = {};
            ContactUpdate.FullName = FullName;
            $.when(SqlUpdateRecordWhere(dbAddressBook, "Contact", ContactRec,
ContactUpdate)).done(function () {
                //record has been saved
                toastr.success('Contact record updated.', 'Address Book');
                // clear the edit page form fields
                pgEditContactClear();
                // show the records listing page.
                $.mobile.changePage('#pgContact', {transition: pgtransition});
                }).fail(function (err) {
                toastr.error('Contact record not updated, please try again.',
'Address Book');
                return;
            });
        };
```

Also here, we are deferring the execution until it finishes. $.when SqlUpdateRecordWhere, a contact details are updated using the FullName of the person is done, show a toast that the contact details are updated, if not toast that the user details are not updated.

JavaScript - Delete Contact

From the update contact screen, if an end user opts to delete a contact, clicking the Delete button will follow a series of events. The end user will be prompted if they want to delete a contact and if they click Yes, the contact details will be erased from the websql database. Message Boxes were discussed in the previous articles and will not be redone here, so please refer to that.

A contact is deleted by primary key which is the FullName in this case.

```
//delete a record from SQL using record key
        app.deleteContact = function (FullName) {
            FullName = FullName.split(' ').join('-');
            //define the record to delete
            var ContactDelete = {};
            ContactDelete.FullName = FullName;
            $.when(SqlDeleteRecordWhere(dbAddressBook, "Contact",
ContactDelete)).done(function () {
                //record has been deleted
                toastr.success('Contact record deleted.', 'Address Book');
                // show the page to display after a record is deleted, this case
listing page
                $.mobile.changePage('#pgContact', {transition: pgtransition});
                }).fail(function (err) {
                toastr.error('Contact record not deleted, please try again.',
'Address Book');
                return;
            });
        };
```

we build the where clause first using the FullName and then execute SqlDeleteRecordWhere. Thus when a contact is deleted, a toast is shown and then the end user is taken to the contact listing screen.

You can refer to the previous articles in terms of how to list records, but for the sake of this article, we will just talk about getting all records from the database with the CheckStorageMethod.

Before all contacts are listed in the ListView, the checkForContactStorage function is executed. This reads all records from the database and loads them to the listview.

JavaScript - Get All Contacts & Display Them

```
//display records if they exist or tell user no records exist.
        app.checkForContactStorage = function () {
            //get records from SQL.
            //when returned, parse then as json object
            var ContactObj = {};
            $.when(SqlGetRecords(dbAddressBook, "Contact",
"FullName")).done(function (dta) {
                // return json object of all records
                ContactObj = ResultSetToJSON(dta, "FullName");
                // are there existing Contact records?
                if (!$.isEmptyObject(ContactObj)) {
                    // yes there are. pass them off to be displayed
                    app.displayContact(ContactObj);
                    } else {
                    // nope, just show the placeholder
                    $('#pgContactList').html(ContactHdr +
noContact).listview('refresh');
                }
                }).fail(function (err) {
                //just show the placeholder
                $('#pgContactList').html(ContactHdr +
noContact).listview('refresh');
            });
        };
```

$.when *SqlGetRecords* is .done, ***dta*** returns the ResultSet which is passed to *ResultSetToJSON* containing all records in the contacts database, when these are found, these are displayed with *app.displayContact(ContactObj);* with the listview refreshed.

JavaScript - Create Database & Tables

Before all these CRUD operations are performed, the database for this app should be created and the respective tables. This should be on init.

1. We opened the database

```
//open the websql database
            dbAddressBook = SqlOpenDb("AddressBook");
```

2. We created the table, first defining it and then calling the function to create it.

```
//create a websql table for SQL-Contact
            app.ContactSqlCreateTable = function () {
                var tblStructure = {};
                tblStructure.FullName = DB_TEXT;
                tblStructure.Company = DB_TEXT;
                tblStructure.JobTitle = DB_TEXT;
                tblStructure.EmailAddress = DB_TEXT;
                tblStructure.BusinessPhone = DB_TEXT;
                tblStructure.BusinessFax = DB_TEXT;
                tblStructure.MobilePhone = DB_TEXT;
                tblStructure.HomePhone = DB_TEXT;
                tblStructure.StreetAddress1 = DB_TEXT;
                tblStructure.StreetAddress2 = DB_TEXT;
                tblStructure.City = DB_TEXT;
                tblStructure.State = DB_TEXT;
                tblStructure.Province = DB_TEXT;
                tblStructure.PostalCode = DB_TEXT;
                SqlCreateTable(dbAddressBook, "Contact", tblStructure,
"FullName", "");
            };
            app.ContactSqlCreateTable();
```

Points of Interest

Finding the deferred function of JQuery to manipulate function calls helped me in ensuring that I minimise the code to maintain the database. The websql.js code can be used in any WebSQL database that one can decide to use as one just passes objects to the functions. Whilst there is more functionality added in this app to display relationships in relation to City for the contacts we have not explained that as the focus here was basically for CRUD functionality. You can also create such CRUD apps using the three step JQM.Show mobile app that is a RAD tool that writes all this code for you to create your prototypes.

History

This is the third installment of our articles in creating CRUD web apps using different frameworks. We spoke about LocalStorage, then web related JSON and now WebSQL. The next steps will be MySQL and depending on interest and use other databases. You can leave me comments of what you would also like to see in these articles so that I can explorer and relate more information. If you find anything useful, please vote and also recommend any enhancements that I might need to add in any of my articles. Thank you for taking time to go through this. #Enjoy

License

https://www.codeproject.com/Articles/891263/Create-a-CRUD-web-app-using-JQuery-Mobile-WebSQL?display=Print

Chapter 16 - Offline Web Applications

<– back to Chapter 16 - Offline Web Applications

Lesson 1: Web SQL

Web SQL is arguably one of the most powerful options available to you. It provides a full relational database that includes many of the features you've come to enjoy from server-side database offerings.

Most implementations are built on SQLite, which is one of the most widely used lightweight database engines. It's an open-source solution with a vibrant community backing it.

1. Applicability of Web SQL

Before starting with Web SQL, be aware that the W3C has stated that Web SQL is no longer on its recommendation track and that it does not intend to maintain the specification.

What this means depends on the browser you're targeting. While the spec is no longer being maintained, some browsers (Safari and Chrome) have continued their support.

Therefore, Web SQL might be a viable option if you're building specifically for a platform such as iOS for iPad or iPhone. Another common use is in building Google Chrome extensions.

If you need a browser-agnostic solution then you might be better off with IndexedDB instead.

2. Creating & opening the database

In the following sections, examine the syntax used to create or open a database, start a transaction, and execute a SQL command. Most of these command should look very familiar if you've used other relational databases.

To start communication with a database, use the `openDatabase` method, which returns a Database object. If you attempt to open a database that doesn't exist, it will be automatically created for you, so you won't need to execute any extra steps.

The following are the `openDatabase` parameters.

- **name** The database name, which is case-sensitive. Most characters are allowed; even an empty string is considered valid.
- **version** Expected version of the database. If an empty string is passed, it's implied that whatever version currently exists is fine.
- **displayName** Descriptive name of the database.
- **estimatedSize** Estimated size required for the database. The typical default value is 5 MB; the browser might prompt the user for permission, depending on the size you specify.

- **creationCallback** If the database does not yet exist and is being created, this callback will be invoked. It is optional and not needed for the database to be created and versioned correctly.

In the following example, a database named `Library` is created with an estimated size of 5 MB. It returns a Database object that supports transactional operations.

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

If you're familiar with traditional database connections, you might be expecting a need to close a connection. With Web SQL, however, that's automatically handled, so you don't have to close the connection manually.

3. Performing schema updates

As your application grows, your data requirements change. You might need to add new tables, drop existing ones, or even change particular columns. The Database object provides the following hooks for making those changes.

- **version** Property that gets the current schema version
- **changeVersion** Method for performing schema changes between one version and the next
  - **oldVersion** Schema version you are migrating from
  - **newVersion** Schema version you are migrating to
  - **callback** Callback method containing schema changes such as adding and dropping tables
  - **errorCallback** Optional; callback method is invoked if an error occurs while the transaction is being processed
  - **successCallback** Optional; callback method is invoked if all statements successfully execute within the transaction

4. Adding a table

You can add an `authors` table to the `Library` created earlier. You need a callback method that accepts a transaction object, which executes the `CREATE TABLE` script.

The transaction object allows multiple actions within it, and it automatically rolls back all changes if any fail. For now, this example keeps the idea simple by adding just one table.

```
function migrateDB(transaction) {
    transaction.executeSql("CREATE TABLE IF NOT EXISTS authors(" +
        "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
        "firstName TEXT, " +
        "lastName TEXT, " +
        "dateCreated TIMESTAMP DEFAULT(datetime('now', 'localtime')))"
    );
}
function onError(error) {
    console.log('Error Code:', error.code, 'Message: ', error.message );
}
function onSuccess() {
    console.log('Migration complete!');
}
```

```
var db = openDatabase('Library', '1.0', 'My  library', 5 * 1024 * 1024);
db.changeVersion('1.0', '2.0', migrateDB, onError, onSuccess);
```

Live Sample - Web SQL - Create database, add table, insert, update, delete

Later in the chapter, you can read the version property of the Database object to determine the schema version with which you are working.

Note that version updates are applied asynchronously, so if the following line was placed immediately after the `db.changeVersion()` call in the preceding code, it would still display 1.0 because the `console.log()` method would fire before the migrations had a chance to complete.

```
console.log("Current schema:", db.version);
```

Now that the migration has been applied, you have a new table in your database with the following columns.

- **id** Table identifier; new records are automatically assigned an `id` that is one greater than the id of the last record added.
- **firstName** Text field for storing a person’s first name.
- **lastName** Text field for storing a person’s last name.
- **dateCreated** Time stamp; when a record is first created, this column defaults to the current time with the help of the SQLite datetime method. Instead of using of using its default mode of GMT, you can indicate that it should use the local time zone.

5. Using transactions

Now that you have a schema in place, you can use transactions to execute SQL statements. To do this, the Database object provides the following two methods.

- **transaction** Starts a new transaction that executes SQL statements; allows both read and write command
- **readTransaction** Works similarly to the transaction method but allows read commands only

Both methods accept the same three parameters.

- **callback** Callback method containing the individual commands that are to be executed as part of the transaction
- **errorCallback** Optional callback method invoked if an error occurs while the transaction is being processed
- **successCallback** Optional callback method invoked if all statements successfully execute within the transaction

The callback method will receive a transaction object that includes an `executeSql` method for performing data changes. It has the following parameters.

- **sqlStatement** The SQL statement string to be executed
- **arguments** Array of object parameters to be used by the SQL command.
- **callback** Optional callback method invoked after the command is executed. When data is retrieved, this method includes the collection of selected rows.

- **errorCallback** Optional callback method invoked if an error occurs while the statement is being executed

In the next section, you see how you can use the transactions to execute some of the most commonly used SQL commands.

6. Inserting a new record

Now that you have a database and table in place, add a new record. Like creating a new table, do this by using the `executeSql` method on the transaction instance.

```
var db = openDatabase('Library', '2.0', 'My library', 5 * 1024 * 1024);

db.transaction(function(t) {
    t.executeSql("INSERT INTO authors (firstName, lastName) " +
        " VALUES ('James', 'Priest')");
});
```

However, in general, it’s a good idea to useSQL parameters when working with dynamic SQL. The preceding statement can be rewritten to take advantage of an optional second parameter on the `executeSql` method, which accepts an array of field values.

Note the use of the question marks to indicate that the value will be populated from the array being passed in.

```
var firstName = 'James';
var lastName = 'Priest';
db.transaction(function(t) {
    t.executeSql("INSERT INTO authors (firstName, lastName) VALUES(?, ?)",
        [firstName, lastName]);
});
```

You can go a step further by adding a callback to the `executeSql` method, which enables you to capture the `Id` of the newly created row.

```
function itemInserted(transaction, results) {
    console.log("Id:", results.insertId);
}

var firstName = 'James';
var lastName = 'Priest';
db.transaction(function(t) {
    t.executeSql("INSERT INTO authors (firstName, lastName) VALUES(?, ?)",
        [firstName, lastName],
        itemInserted);
});
```

Live Sample - Web SQL - Create database, add table, insert, update, delete

7. Updating an existing record

In the following example, the `lastName` of the author, which has an `id` of 1, is updated. Besides the SQL syntax differences, it’s very similar to the code used for adding a new record.

```
var db = openDatabase('Library', '2.0', 'My library', 5 * 1024 * 1024);
var authorId = 1;
var lastName = 'Smith';
```

```
db.transaction(function(t) {
    t.executeSql("UPDATE authors SET lastName = ? WHERE id = ?"
        , [lastName, authorId]);
});
```

Live Sample - Web SQL - Create database, add table, insert, update, delete

8. Deleting a record

Removing records is also fairly straightforward. The following example deletes the author record with an `id` of 1.

```
var db = openDatabase('Library', '2.0', 'My library', 5 * 1024 * 1024);
var authorId = 1;
db.transaction(function(t) {
    t.executeSql("DELETE FROM authors WHERE id = ?", [authorId]);
});
```

Live Sample - Web SQL - Create database, add table, insert, update, delete

9. Reading values from the database

Now that you know how to add data to the database, you can read and display those records back to the user. Create a simple SELECT statement to read all values from the authors table. When `executeSql` is called this time, a callback method is passed that accepts a transaction object and a results set containing the rows returned from the SQL statement.

As the `displayResults` method iterates through the rows, it formats the person's name in a list item and adds it to an unordered list with an `id` of items. To access the individual column values within the row, use dot notation, which reads each as a property on the object.

```
function displayResults(transaction, results) {
  for (var i = 0; i < results.rows.length; i++) {
    var item = results.rows.items(i);
    $('#items').append('<li>' + item.firstName + ' ' + item.lastName + '</li>');
  }
}

var db = openDatabase('library', '2.0', 'My library', 5 * 1024 * 1024);
db.transaction(function(t) {
    t.executeSql("SELECT * FROM authors", [], displayResults)
});
```

Because you are only retrieving data, you just as easily could have used the `readTransaction` method instead of the `transaction` method.

```
db.readTransaction(function(t) {
    t.executeSql("SELECT * FROM  authors", [], displayResults);
})
```

Quick check

- The following statement has a syntax error in the second step of the transaction in this migration script (misspelled CREATE as CRATE). What do you expect will happen because of this migration script?

```
function migrateDB(transaction) {
    transaction.executeSql("CREATE TABLE authors(firstName TEXT)");
    transaction.executeSql("CREATE TABLE books(title TEXT)");
}

var db = openDatabase('Library', '1.0', 'My library', 5 * 1024 * 1024);
db.changeVersion('1.0', '2.0', migrateDB);
```

Answer

- Neither table will be created.

10. Filtering results

You rarely want to read every row from a database table; most of the time, you need to limit those results to specific criteria. Because current implementations are based on SQLite, you have all the power of a mature database engine to help you.

For example, you can add a WHERE clause to return only records with a specific `lastName` value, as follows.

```
var db = openDatabase('Library', '2.0', 'My library', 5 * 1024 * 1024);
var lastName = 'Priest';
db.transaction(function(t) {
    t.executeSql("SELECT * FROM authors WHERE lastName = ?"
    , [lastName], displayResults);
});
```

You might like to find all `authors` whose last name starts with the letter D. To do so, use the LIKE keyword along with the `%` wildcard.

```
var lastName = 'D%';
db.transaction(function(t) {
    t.executeSql("SELECT * FROM authors WHERE lastName LIKE ?"
    , [lastName], displayResults);
});
```

11. Using JOIN commands

Web SQL includes support for traditional JOIN statements (such as INNER JOIN and LEFT JOIN), which can be used to include columns from multiple tables within a single SELECT statement.

Assume you added a `books` table to your library database and would now like to modify your earlier query to include the title of each book in the results.

```
var db = openDatabase('Library', '1.0', 'My library', 5 * 1024 * 1024);
var lastName = 'D%';
db.transaction(function(t) {
    t.executeSql("SELECT a.firstName, a.lastName, b.title " +
        "FROM authors a " +
        "INNER JOIN books b on a.id = b.authorId " +
        "WHERE lastName LIKE ?"
        , [lastName], displayResults);
});
```

12. Aggregating functions

Another useful feature of Web SQL is the ability to group results, which enables the use of more advanced functions such as COUNT(x), MIN(x), MAX(x), and SUM(x) within your SELECT statements. For example, the following is a new query that finds the number of books written by each author.

```
db.transaction(function(t) {
    t.executeSql("SELECT a.firstName, a.lastName, COUNT(b.id) as numOfBooks " +
        "FROM authors a " +
        "INNER JOIN books b on a.id = b.authorId " +
        "GROUP BY a.id"
        , [], displayResults);
});
```

13. Lesson summary

- The W3C has stated that the Web SQL specification is no longer on tit recommendation track. It may still be used when targeting specific platforms that have continued support, but other options such as **IndexedDB** and **web storage** should be considered when possible.
- Current browser implementations are based on SQLite, which gives you all the power of a full relational database.
- Database communication is started by calling the `openDatabase()` command. If the database does not exist, it will be created automatically.
- Schema migration support is available by using the `changeVersion()` method.
- Web SQL supports a common SQL syntax for create, retrieve, update, & delete (CRUD) operations
- If one statement in a transaction fails, all actions are rolled back.

https://james-priest.github.io/100-days-of-code-log-r2/CH16-Offline1-WebSQL.html