# HTML5 - Web SQL Database

The Web SQL Database API isn't actually part of the HTML5 specification but it is a separate specification which introduces a set of APIs to manipulate client-side databases using SQL.

I'm assuming you are a great web developer and if that is the case then no doubt, you would be well aware of SQL and RDBMS concepts. If you still want to have a session with SQL then, you can go through our [SQL Tutorial](https://www.tutorialspoint.com/sql/index.htm).

Web SQL Database will work in latest version of Safari, Chrome and Opera.

## The Core Methods

There are following three core methods defined in the spec that I am going to cover in this tutorial −

* **openDatabase** − This method creates the database object either using existing database or creating new one.
* **transaction** − This method gives us the ability to control a transaction and performing either commit or rollback based on the situation.
* **executeSql** − This method is used to execute actual SQL query.

## Opening Database

The *openDatabase* method takes care of opening a database if it already exists, this method will create it if it already does not exist.

To create and open a database, use the following code −

var db = openDatabase('mydb', '1.0', 'Test DB', 2 \* 1024 \* 1024);

The above method took the following five parameters −

* Database name
* Version number
* Text description
* Size of database
* Creation callback

The last and 5th argument, creation callback will be called if the database is being created. Without this feature, however, the databases are still being created on the fly and correctly versioned.

## Executing queries

To execute a query you use the database.transaction() function. This function needs a single argument, which is a function that takes care of actually executing the query as follows −

var db = openDatabase('mydb', '1.0', 'Test DB', 2 \* 1024 \* 1024);

db.transaction(function (tx) {

tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');

});

The above query will create a table called LOGS in 'mydb' database.

## INSERT Operation

To create enteries into the table we add simple SQL query in the above example as follows −

var db = openDatabase('mydb', '1.0', 'Test DB', 2 \* 1024 \* 1024);

db.transaction(function (tx) {

tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');

tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "foobar")');

tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "logmsg")');

});

We can pass dynamic values while creating entering as follows −

var db = openDatabase('mydb', '1.0', 'Test DB', 2 \* 1024 \* 1024);

db.transaction(function (tx) {

tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');

tx.executeSql('INSERT INTO LOGS (id,log) VALUES (?, ?'), [e\_id, e\_log];

});

Here **e\_id** and **e\_log** are external variables, and executeSql maps each item in the array argument to the "?"s.

## READ Operation

To read already existing records we use a callback to capture the results as follows −

var db = openDatabase('mydb', '1.0', 'Test DB', 2 \* 1024 \* 1024);

db.transaction(function (tx) {

tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');

tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "foobar")');

tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "logmsg")');

});

db.transaction(function (tx) {

tx.executeSql('SELECT \* FROM LOGS', [], function (tx, results) {

var len = results.rows.length, i;

msg = "<p>Found rows: " + len + "</p>";

document.querySelector('#status').innerHTML += msg;

for (i = 0; i < len; i++) {

alert(results.rows.item(i).log );

}

}, null);

});

## Final Example

So finally, let us keep this example in a full-fledged HTML5 document as follows and try to run it with Safari browser.

[Live Demo](http://tpcg.io/rMAhs5)

<!DOCTYPE HTML>

<html>

<head>

<script type = "text/javascript">

var db = openDatabase('mydb', '1.0', 'Test DB', 2 \* 1024 \* 1024);

var msg;

db.transaction(function (tx) {

tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');

tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "foobar")');

tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "logmsg")');

msg = '<p>Log message created and row inserted.</p>';

document.querySelector('#status').innerHTML = msg;

})

db.transaction(function (tx) {

tx.executeSql('SELECT \* FROM LOGS', [], function (tx, results) {

var len = results.rows.length, i;

msg = "<p>Found rows: " + len + "</p>";

document.querySelector('#status').innerHTML += msg;

for (i = 0; i < len; i++) {

msg = "<p><b>" + results.rows.item(i).log + "</b></p>";

document.querySelector('#status').innerHTML += msg;

}

}, null);

});

</script>

</head>

<body>

<div id = "status" name = "status">Status Message</div>

</body>

</html>

# A Very Simple Example of HTML 5 OFFLINE Database (indexedDB and WebSql)

## Introduction

This application is for reference and not to teach. I am sharing a small part of my code in this article for beginners like me. I would be pleased to get your suggestions or any comments.

This article tries to explain the basics about the browser databases that comes in handy when working with the HTML 5 Offline application. How can a developer develop a web application that would interact with the local databases of browsers, insert new records, modify or edit the currently existing records of a table, get all or a specific record from the table and delete the records of a table.

## Background

To make our web application work offline, we require two things:

* CACHE.APPCACHE: To get all the required files (HTML pages, JavaScript files, images, etc.) on the local machine browser
* DATABASE: A database to store user work locally

#### CACHE.APPCACHE

First point can be implemented using the cache.appcache file, a new feature of HTML 5. You can easily find some useful tutorials for this cache.appcache file, and this is very easy to implement cache using this file. For our project, we are not going to write a hard coded file, in this project, we will implement an Action Result which will be a server for this file.

#### DATABASE

Now this is the most important part of this article. Any application requires database to store information/records like SQL server on the server side. As we are implementing an offline application, we need any database so that we can store the information/records. For offline applications, we have two types of databases supported by different browsers.

* WebSql for Safari like browsers
* IndexedDB for Mozilla, Chrome, Internet Explorer like browsers.

WebSql: This is a query based database like SQL server. You need to write queries to insert, update and delete the records like you did in SQL server. This database is used for the Safari browser.

IndexedDB: This database works on objects, we are not required to write queries for this database. It simply works to add, update and delete the objects. For this database, we will use a db.js wrapper file provided by aaron powell. We will use this database for all the browsers other than Safari.

## Using the Code

### Implementation for cache manifest Action

JavaScript

Shrink ▲   Copy Code

// Cache Action

public ActionResult Manifest()

{

var manifest = "CACHE MANIFEST" + Environment.NewLine +

"# App Version: " + System.IO.File.GetLastWriteTime

(Server.MapPath("~/Views/Home/Index.cshtml")) + Environment.NewLine +

"# Server Assembly Version: " + this.GetType().Assembly.GetName().Version +

Environment.NewLine +

"NETWORK:" + Environment.NewLine +

"\*" + Environment.NewLine +

"CACHE:" + Environment.NewLine +

Url.Action("Index", "Home") + Environment.NewLine +

Url.Content("~/Content/site.css") + Environment.NewLine +

Url.Content("~/Content/bootstrap.min.css") + Environment.NewLine +

Url.Content("~/scripts/jquery-1.7.1.js") + Environment.NewLine +

Url.Content("~/scripts/bootstrap.min.js") + Environment.NewLine +

Url.Content("~/scripts/bootbox.min.js") + Environment.NewLine +

Url.Content("~/scripts/db.js") + Environment.NewLine +

Url.Content("~/scripts/Config.js") + Environment.NewLine +

Url.Content("~/scripts/DbManager.js") + Environment.NewLine +

Url.Content("~/scripts/index.js") + Environment.NewLine +

Url.Content("~/scripts/jquery.blockUI.js") + Environment.NewLine +

Url.Content("~/scripts/cache.js") + Environment.NewLine;

return Content(manifest, "text/cache-manifest");

}

This Action result has three main lines of code.

This line sets the version for the cache.appcache file. Whenever you change anything in your main view, this will force the browser to get the new files from the server.

JavaScript

Copy Code

"# App Version: " + System.IO.File.GetLastWriteTime

(Server.MapPath("~/Views/Home/Index.cshtml")) + Environment.NewLine +

To add new files, you can add the files using this line of code:

JavaScript

Copy Code

Url.Content("~/Content/site.css") + Environment.NewLine +

The return type of this Action is set of type "text/cache-manifest".

JavaScript

Copy Code

return Content(manifest, "text/cache-manifest");

Now to add this manifest to your layout or page, you simply add this line to your page html tag.

JavaScript

Copy Code

html manifest="@Url.Action("Manifest", "Home")"

That's it, we are done with the cache manifest code.

### Implementation for Databases

Now we will have two code block sections, one for websql and the other for indexedDb. First of all, we are required to recognise whether it a Safari browser or not. If it is a Safari browser, we will use websql, otherwise indexedDB.

JavaScript

Copy Code

if (navigator.userAgent.indexOf("Safari") >= 0

&& navigator.userAgent.indexOf("Chrome") < 0) {

//SAFARI BROWSER SO WEBSQL IMPLEMENTATION HERE

}else{

//OTHER BROWSERS SO INDEXEDDB IMPLEMENTATION HERE

}

Now, we need to initialize our databases:

JavaScript

Copy Code

if (navigator.userAgent.indexOf("Safari") >= 0

&& navigator.userAgent.indexOf("Chrome") < 0) {

var DataBaseManager = {

Offlinedb: openDatabase("OfflineDB", '1', 'my first database',

2 \* 1024 \* 1024)//this accept four parameters 1. database name,

//2. version, 3. Comments, 4.initial Size

}

}else{

//For indexedDB the db.js file automatically initialize the database.

//So we just need to provide the schema and database name during our database operations.

}

Now before going further, first we will declare a global schema object for indexedDB, in this we have UserData Table with UserID as autoincrement true.

JavaScript

Copy Code

var OfflineConfiguration = {

Db\_VERSION: 1,

DB\_NAME: "OfflineDB",

SCHEMA: {

UserData: { key: { keyPath: 'UserID', autoIncrement: true } }

}

};

Now we have a dataserver to store our application database, we need to define the schema for database. For WebSql, we will define our table structure with query and for IndexedDb, we will define the schema with objects. Our application uses a single table, i.e., UserData and UserID is an autoincrement identity column.

JavaScript

Copy Code

if (navigator.userAgent.indexOf("Safari") >= 0

&& navigator.userAgent.indexOf("Chrome") < 0) {

var DataBaseManager = {

Offlinedb: openDatabase("OfflineDB", '1', 'my first database', 2 \* 1024 \* 1024),

initializeDataBase: function () {

var self = this;

self.Offlinedb.transaction(function (tx) {

tx.executeSql('CREATE TABLE IF NOT EXISTS UserData

(UserID INTEGER PRIMARY KEY AUTOINCREMENT, Name, Email, Technology)');

});

},

}

}else{

var DataBaseManager = {

//For indexedDB we will not have any initialize function

//as we already define the schema for this globally.

}

}

### Data Operations

Now, we have our table so we can add our data operations ADD, GET, UPDATE and DELETE.

#### 1. Add User

JavaScript

Shrink ▲   Copy Code

if (navigator.userAgent.indexOf("Safari") >= 0

&& navigator.userAgent.indexOf("Chrome") < 0) {

var DataBaseManager = {

Offlinedb: openDatabase("OfflineDB", '1', 'my first database', 2 \* 1024 \* 1024),

initializeDataBase: function () {

var self = this;

self.Offlinedb.transaction(function (tx) {

tx.executeSql('CREATE TABLE IF NOT EXISTS UserData

(UserID INTEGER PRIMARY KEY AUTOINCREMENT, Name, Email, Technology)');

});

},

AddnewUser: function (data, callback) {//data: contains the object of user ,

// callback: is a function will execute after the addition

this.initializeDataBase();

var self = this;

self.Offlinedb.transaction(function (tx) {

var query = "insert into UserData(Name,Email,Technology) values(?,?,?)";

tx.executeSql(query, [data.Name, data.Email, data.Technology],

function (tx, results) {

if (callback) callback("User Saved");

});

});

}

}

}else{

var DataBaseManager = {

//For indexedDB we will not have any initialize function as we already define

//the schema for this globally.

AddnewUser: function (data, callback) {//data: contains the object of user ,

// callback: is a function will execute after the addition

db.open({

server: OfflineConfiguration.DB\_NAME,//database name defined globally

version: OfflineConfiguration.Db\_VERSION,//version defined globally

schema: OfflineConfiguration.SCHEMA//schema defined globally

}).done(function (s) {

self.Server = s;

self.Server.UserData.add(data).done(function (results) {

if (callback) callback("Data added into UserData");

});

});

},

}

}

#### 2. Get User

JavaScript

Shrink ▲   Copy Code

if (navigator.userAgent.indexOf("Safari") >=0

&& navigator.userAgent.indexOf("Chrome") < 0) {

var DataBaseManager = {

.....

GetSingleUser: function (data, callback) {//data: contains the UserID ,

// callback: is a function will execute after the selection

try {

var self = this;

this.initializeDataBase();

var query1 = "SELECT \* from UserData where UserID=" + data;

self.Offlinedb.transaction(function (tx) {

tx.executeSql(query1, [], function (tx, results) {

if (results.rows.length > 0) {

var v = results.rows.item(0);

if (callback) callback(results.rows.item(0));

} else {

if (callback) callback("Not Found");

}

});

});

}

catch (e) {

console.log(" error occurred in selecting data");

}

},

....

}

}else{

var DataBaseManager = {

....

GetSingleUser: function (data, callback) {//data: contains the UserID ,

//callback: is a function will execute after the selection

db.open({

server: OfflineConfiguration.DB\_NAME,

version: OfflineConfiguration.Db\_VERSION,

schema: OfflineConfiguration.SCHEMA

}).done(function (s) {

self.Server = s

self.Server.UserData

.query()

.all()

.filter(function (result) {

return result.UserID === parseInt(data);

})

.execute()

.done(function (finalResult) {

if (callback) callback(finalResult);

});

});

},

....

}

}

#### 3. Update User

JavaScript

Shrink ▲   Copy Code

if (navigator.userAgent.indexOf("Safari") >=0

&& navigator.userAgent.indexOf("Chrome") < 0) {

var DataBaseManager = {

.....

UpdateUser: function (data, callback) {//data: contains the object of user ,

// callback: is a function will execute after the updation

try {

this.initializeDataBase();

var self = this;

var query1 = "update UserData set Name=?,Email=?,Technology=? where UserID=?";

self.Offlinedb.transaction(function (tx) {

tx.executeSql(query1, [data.Name, data.Email,

data.Technology, parseInt(data.UserID)], function (tx, results) {

if (callback) callback("Response updated");

});

});

}

catch (e) {

console.log(" error occurred in selecting data");

}

},

....

}

}else{

var DataBaseManager = {

....

UpdateUser: function (data, callback) {//data: contains the object of user ,

// callback: is a function will execute after the updation

db.open({

server: OfflineConfiguration.DB\_NAME,

version: OfflineConfiguration.Db\_VERSION,

schema: OfflineConfiguration.SCHEMA

}).done(function (s) {

self.Server = s

self.Server.UserData.update(data).done(function (item) {

if (callback) callback("response updated to database");

});

});

},

....

}

}

#### 4. Delete User

JavaScript

Shrink ▲   Copy Code

if (navigator.userAgent.indexOf("Safari") >=0

&& navigator.userAgent.indexOf("Chrome") < 0) {

var DataBaseManager = {

.....

DeleteUser: function (data, callback) {//data: contains the UserID of user ,

// callback: is a function will execute after the deletion

try {

this.initializeDataBase();

var self = this;

self.Offlinedb.transaction(function (tx) {

tx.executeSql("Delete from UserData where UserID=?",

[data], function (tx, results) {

if (callback) callback("Data deleted");

});

});

}

catch (e) {

}

}

....

}

}else{

var DataBaseManager = {

....

DeleteUser: function (data, callback) {//data: contains the UserID of user ,

// callback: is a function will execute after the deletion

var self = this;

db.open({

server: OfflineConfiguration.DB\_NAME,

version: OfflineConfiguration.Db\_VERSION,

schema: OfflineConfiguration.SCHEMA

}).done(function (s) {

self.Server = s

self.Server.UserData.remove(data).done(function (a) {

if (callback) callback("Data deleted");

});

});

}

....

}

}

That's it! We are done with our four data operations.

### Use of These Operations

You can refer to the index.js file in which I have implemented the use of these operations.

#### 1. Use of Add New User Operation

JavaScript

Copy Code

var data = '{"Name":"Vinu","Email":"itvinay12@gmail.com",

"Technology":".NET"}';//user object

data = JSON.parse(data);

DataBaseManager.AddnewUser(data, GetAllUser);// GetAllUser is a

//function defined in index.js you can refer it.

#### 2. Use of Get User Operation

JavaScript

Copy Code

DataBaseManager.GetSingleUser(1, anycallbackfunction);// Get the details of user with id 1.

#### 3. Use of Update User Operation

JavaScript

Copy Code

var data = '{"UserID":1,"Name":"Vinu",

"Email":"itvinay12@gmail.com","Technology":".NET"}';//user object with UserID

data = JSON.parse(data);

DataBaseManager.UpdateUser(data, GetAllUser);// GetAllUser is a

//function defined in index.js you can refer it.

#### 4. Use of Delete User Operation

JavaScript

Copy Code

DataBaseManager.DeleteUser(1,GetAllUser);// delete the details of user with id 1.

## Points of Interest

I just want to suggest a point that before starting these types of applications, please choose the best JavaScript framework and read about the compatibility of the browsers.

## History

* 18th September, 2014: First post

## License

This article, along with any associated source code and files, is licensed under [The Code Project Open License (CPOL)](http://www.codeproject.com/info/cpol10.aspx)

https://www.codeproject.com/Articles/820391/A-Very-Simple-Example-of-HTML-OFFLINE-Database-ind

https://www.findbestopensource.com/tagged/websql