Web SQL basics

This article explains how to take the WebSQL in use and some basic operations.

Introduction

Web SQL Database is an interesting feature, even though it is not anymore is not part of the HTML5 specification. It can still provide some good help for developing web applications. Basically the use of Web SQL Database has a disclaimer: use it at your own risk. Certain desktop and mobile browsers support Web SQL Database, but it is always better to test it on the target you are developing aswell as checking the latest support information in the internet. The example code in this article will demonstrate how to create a database, how to create a database table, how to insert data into the table and finally how to display data that was inserted into the table.

Creating the database

To open an existing database or to create a new one, the openDatabase method is used. It will take these arguments, in the given order:

- Name of the database (if the database with the given name will exist it will be opened and if it does not exist, a new database will be created with the given name)

- Version of the database (this is needed and must match the database version used by the client).

- Textual description of the database

- Size of the database to create (recommended size is 5MB per origin/user agent. More can be used).

Note: there may be a user request to accept authorization for more space by the browser.

Example code

In the example, the database does not exist when run first time, so it will be created. The next step is to add some data into the database and it is done by creating a new table. This will happen in form of a database transaction. Every SQL statement is executed as a separate, single operation and is fully isolated from any other transaction - this to secure the database consistency. Should something go wrong during the transaction, there will be a "rollback" to the state that occurred before the failed transaction took place.

```
    // Create a new database
    var db = openDatabase('mydb','1.0','testdatabase',1024);
```

After we have the reference to the database, we will execute a transaction against it:

```
    // Create a new database table
   db.transaction(function(query){
  query.executeSql('CREATE TABLE IF NOT EXISTS user (id unique, usr, pass)');
    });
   db.transaction(function(query){
  query.executeSql('CREATE TABLE IF NOT EXISTS locations (id unique, lat, lon)');

    });
```

The transaction method can take up to three arguments. First is for the transaction callback and the second one for ther error callback. Third is meant for handling a successful transaction callback. Callbacks are optional.

Syntax:

```
executeSql(sqlStatement, arguments, callback, errorCallback)
```


(C) Copyright Nokia 2014. All rights reserved.

The sqlStatement argument can contain SQL arguments represented by question marks '?'. These question marks are then mapped to values contained in the arguments argument which is an array of values.

In the database table creation transaction, we will use executeSql method for the transaction object, in order to create the tables "user" and "locations".

As with the SQL, you could use e.g. "id unique primary key autoincrement" for the locations table, so that the row number would not be needed.

When the table has been created and the rows have been inserted into the database, we will examine how to issue an query against the database with SELECT statement and get the query results displayed. For this, we need to define the success callback, and it takes two arguments: a SQLTransaction object and a SQLResultSet object.

SQLResultSet object set definition by specification:

```
interface SQLResultSet {
  readonly attribute long insertId;
  readonly attribute long rowsAffected;
  readonly attribute SQLResultSetRowList rows;
};
```

The "insertId" will return the row ID of the row that was inserted into the table and database. The "rowsAffected" returns the number of rows that were modified by the SQL statement. The "rows" refer to the SQLResultSetRowList object, that contains all the rows returned by a SELECT statement.

E.g. In the article example code, the following insert clauses will add rows to the "locations" table and one row into the "user" table.

```
 });
...
 // Insert data
  db.transaction(function(query){
 query.executeSql('INSERT INTO user VALUES (1, "maveric","mypassword")');
  });
  db.transaction(function(query){
 query.executeSql('INSERT INTO locations VALUES (1, "52.111111","10.11111")');
  });
  db.transaction(function(query){
 query.executeSql('INSERT INTO locations VALUES (2, "53.111111","10.11111")');
  });
...
```

And here is the full code:

```
<!doctype html>
<html>
   <head>
   <title>WebSQL intro</title>
   <script type = "text/javascript">
   // Create a new database
   var db = openDatabase('mydb','1.0','testdatabase',1024);
   // Create a new database table
  db.transaction(function(query){
 query.executeSql('CREATE TABLE IF NOT EXISTS user (id unique, usr, pass)');
   });
  db.transaction(function(query){
 query.executeSql('CREATE TABLE IF NOT EXISTS locations (id unique, lat, lon)');
```


(C) Copyright Nokia 2014. All rights reserved.

```
   });
 // Insert data
  db.transaction(function(query){
 query.executeSql('INSERT INTO user VALUES (1, "maveric","mypassword")');
  });
  db.transaction(function(query){
 query.executeSql('INSERT INTO locations VALUES (1, "52.111111","10.11111")');
  });
  db.transaction(function(query){
 query.executeSql('INSERT INTO locations VALUES (2, "53.111111","10.11111")');
  });
  db.transaction(function(query){
 query.executeSql('INSERT INTO locations VALUES (3, "54.111111","10.11111")');
  });
  db.transaction(function(query){
 query.executeSql('INSERT INTO locations VALUES (4, "55.111111","10.11111")');
  });
  db.transaction(function(query){
 query.executeSql('INSERT INTO locations VALUES (5, "56.111111","10.11111")');
  });
  db.transaction(function(query){
 query.executeSql('INSERT INTO locations VALUES (6, "57.111111","10.11111")');
  });
  db.transaction(function(query){
 query.executeSql('INSERT INTO locations VALUES (7, "58.111111","10.11111")');
  });
  db.transaction(function(query){
 query.executeSql('INSERT INTO locations VALUES (8, "59.111111","10.11111")');
  });
  db.transaction(function(query){
 query.executeSql('INSERT INTO locations VALUES (9, "61.111111","10.11111")');
  });
  db.transaction(function(query){
 query.executeSql('INSERT INTO locations VALUES (10, "62.111111","10.11111")');
  });
 //Show Data
 db.transaction(function(query)
 {
  query.executeSql('SELECT * FROM locations',[],function(u, results)
     {
       document.write('Locations table has  '+results.rows.length+' rows.')
    }
  );
});
 </script>
 </head>
</html>
```

The Show Data section will execute an SQL "SELECT" command against the "location" table, requesting count of all insterted rows, then writes the result on screen.

Tested on

Google Chrome

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.