

# Offline Storage

# Introduction

- ❖ HTML 5.0 supports a well structured offline storage solution.
- ❖ There are two types of offline storage possibilities. The first is the possibility to store session data separately for each tab. The second is the possibility to store data common to all tabs and all windows.

# Session Storage

- ❖ The session storage extends the capabilities we get when using cookies.
- ❖ Unlike the cookies mechanism that limits us for storing up to 4kilobytes of data the session storage allows us much more space.
- ❖ Unlike the cookies mechanism, the session data isn't sent automatically to the server every HTTP request.

# Session Storage

- ❖ Unlike cookies, session storage is tied to the browser tab. It isn't tied to the browser window. Each tab maintains its own session information.

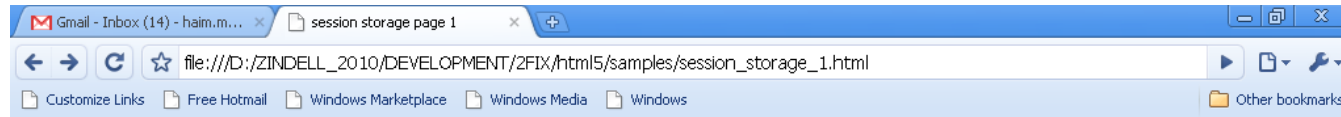
# Session Storage

```
<html>
  <head>
    <title>session storage page 1</title>
  </head>
  <body>
    <h2>page 1</h2>
    <script language="javascript">
      sessionStorage.setItem('company', 'Zindell Technologies');
    </script>
    The 'company' (id) and 'Zindell Technologies' (value) were set as a key
    value pair in the session storage mechanism.
    <p>
      <a href="session_storage_2.html">next</a>
    </p>
  </body>
</html>
```

# Session Storage

```
<html>
  <head>
    <title>session storage page 2</title>
  </head>
  <body>
    <h2>page 2</h2>
    <script language="javascript">
      function showData()
      {
        alert(sessionStorage.getItem('company'));
      }
    </script>
    <form>
      <input type="button" value="click me" onClick="showData()">
    </form>
  </body>
</html>
```

# Session Storage



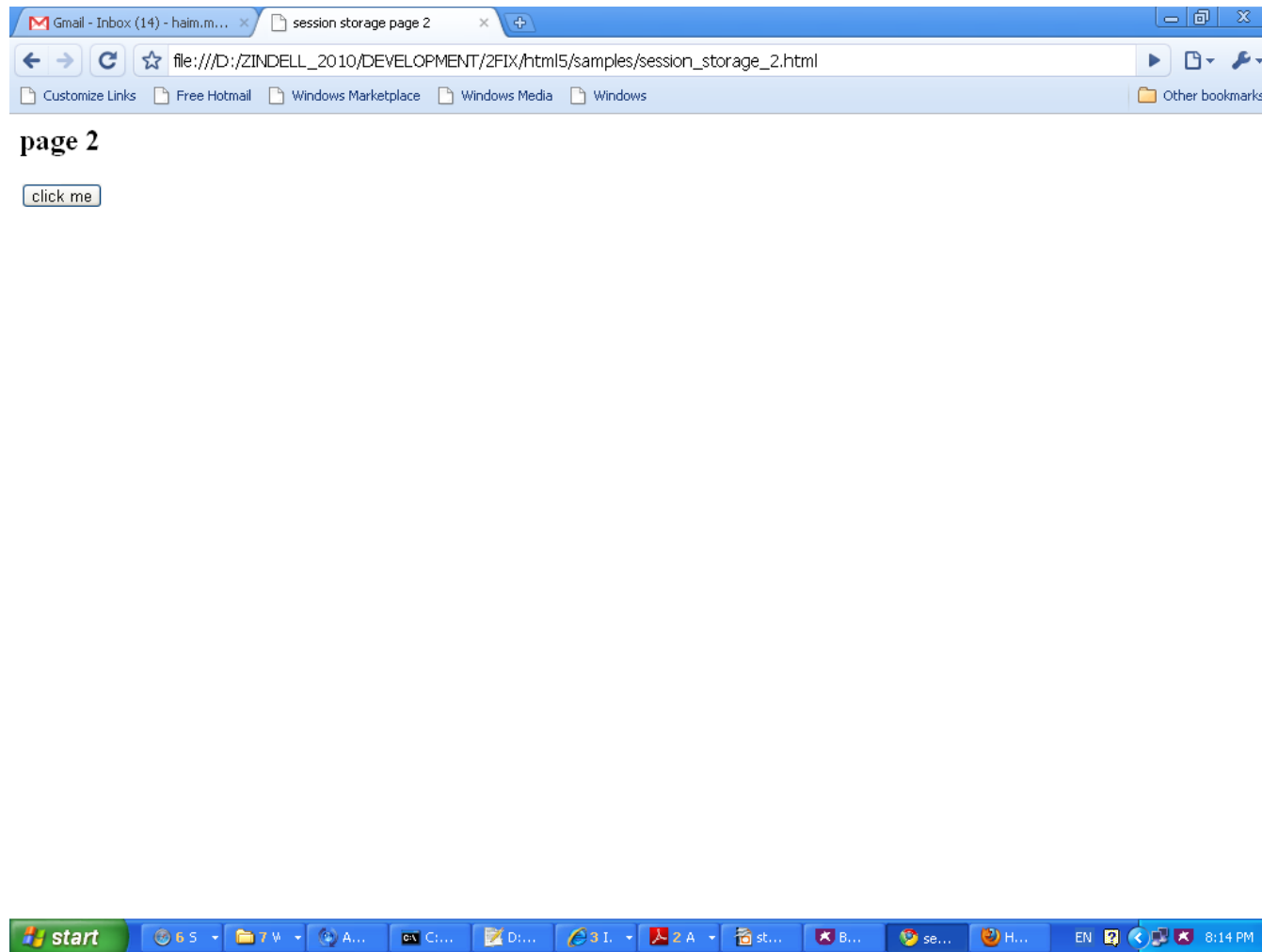
## page 1

The 'company' (id) and 'Zindell Technologies' (value) were set as a key value pair in the session storage mechanism.

next

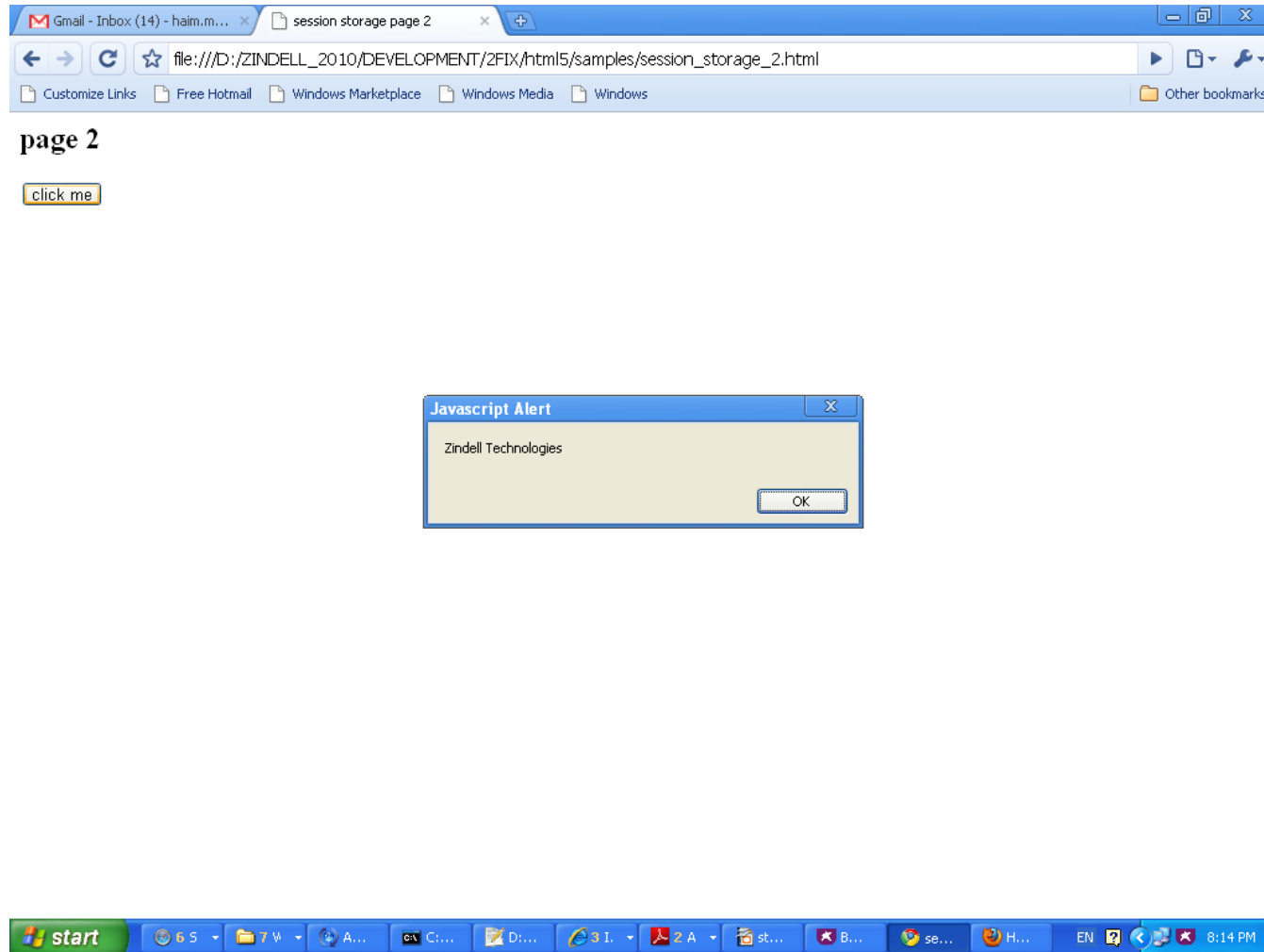


# Session Storage





# Session Storage



# Local Storage

- ❖ The local storage allows us to save data shared by all tabs. That data is kept even when the user closes the web browser.
- ❖ Instead of using the `sessionStorage` we should use the `localStorage`. The syntax is the same.

# Relational Database

- ❖ The HTML5 specifications provides us with a local relational database.
- ❖ We can access that database using the `window.openDatabase` property. This property holds a function that returns an object that represents a specific database on our web browser.

# Relational Database

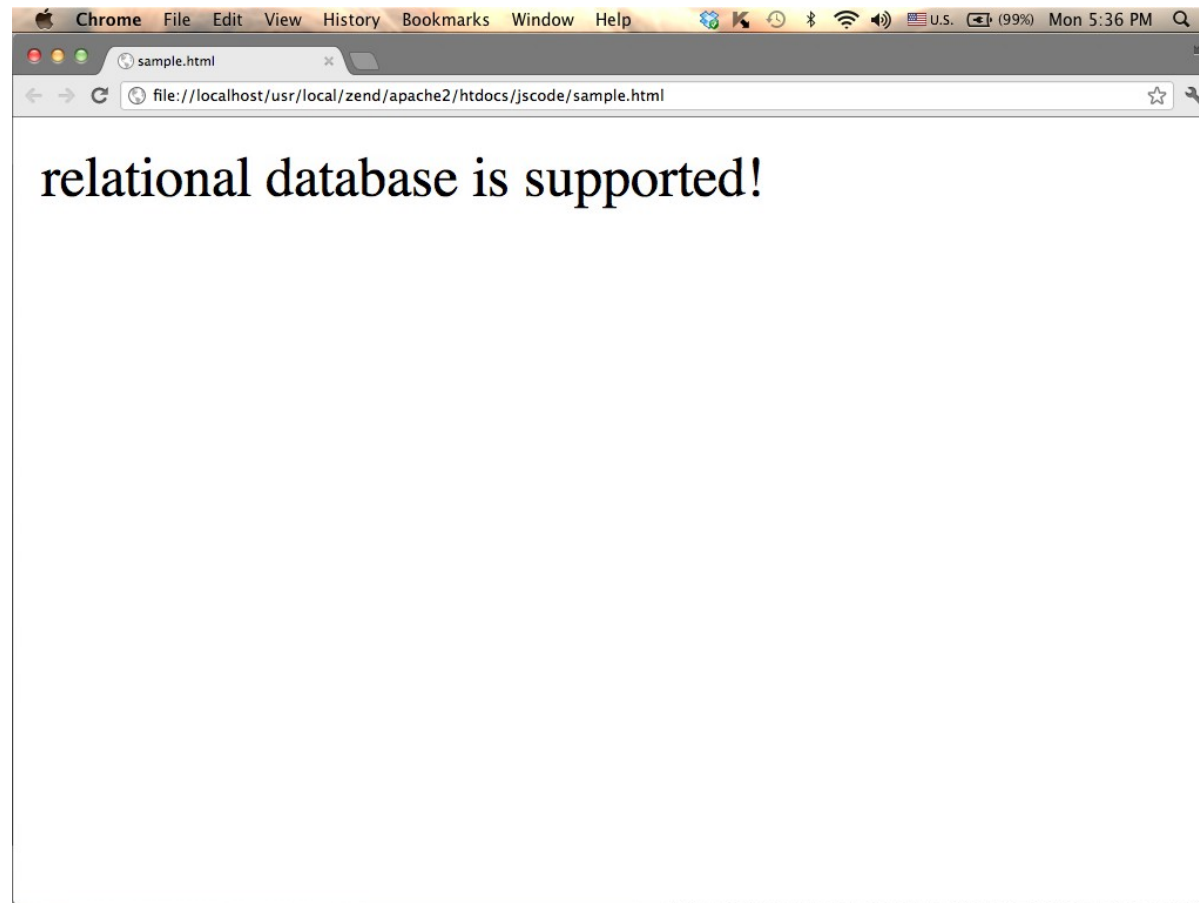
- ❖ We can easily check whether the web browser supports the relational database (or not) by referring the `openDatabase` property. If the property doesn't exist the returned value will be equivalent to `false`.

# Relational Database

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title></title>
    <script type="text/javascript">
        if(window.openDatabase)
        {
            document.write("relational database is supported!");
        }
        else
        {
            document.write("relational database IS NOT supported!");
        }
    </script>
</head>
<body>
</body>
</html>
```



# Relational Database



# Relational Database

- ❖ We can create a new relational database by calling the `window.openDatabase` function passing over the following arguments: the database short name, its version number, its long detailed name and its maximum size in bytes.
- ❖ The returned value is a reference for a new object that represents the new relational database on our web browser.

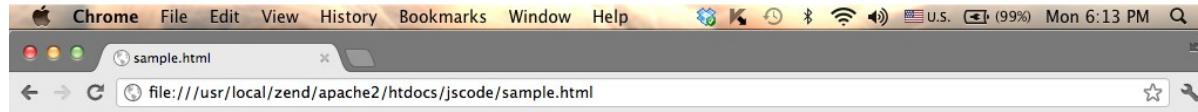
# Relational Database

```
<html>
<head>
  <title></title>
  <script type="text/javascript">
    try
    {
      var db = window.openDatabase(
        'webstore',
        '1.0',
        'database for web store',
        100000);
      document.write("db version " +db.version+ " is ready to use!");
    }
    catch(ex)
    {
      document.write("ex="+ex);
    }
  </script>
</head>
<body></body>
</html>
```





# Relational Database



db version 1.0 is ready to use!

# Relational Database

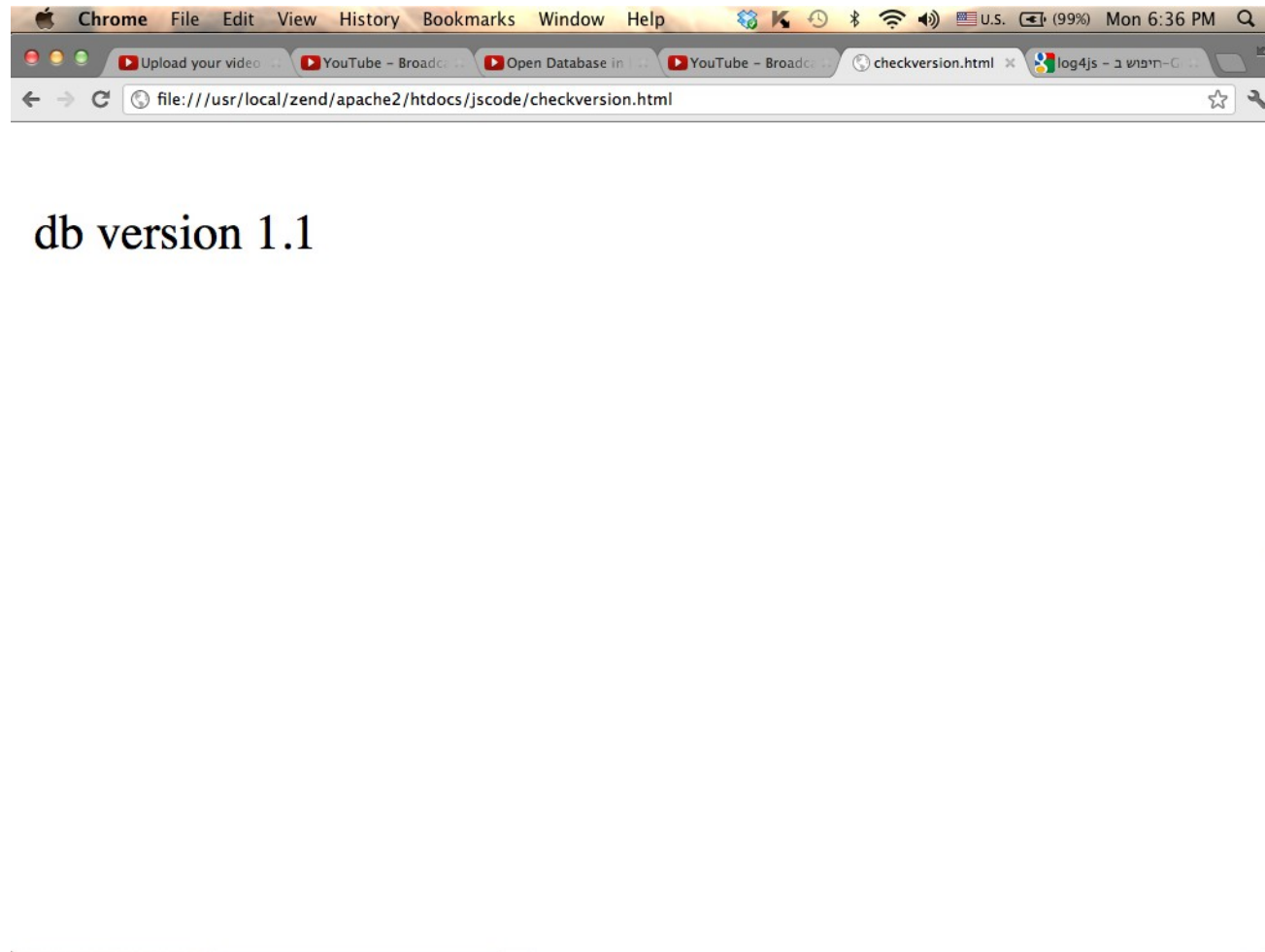
- ❖ We can pass over an empty string as the database version, get the object that represents the data base, check its version and if necessary call the `changeVersion` function in order to change it.

# Relational Database

```
<html>
<head>
  <title></title>
  <script type="text/javascript">
    try
    {
      var db = window.openDatabase(
        'webstore',
        "",
        'database for web store',
        100000);
      document.write("<br>db version " +db.version);
      db.changeVersion("1.0","1.1");
    }
    catch(ex)
    {
      document.write("ex="+ex);
    }
  </script>
</head>
<body></body>
</html>
```



# Relational Database



# Relational Database

- ❖ In order to perform SQL command on our database we should call the `transaction` function on the object that represents the database.
- ❖ We should pass over a function with one argument. That function will be called and an object that represents the transaction will be passed over to it. One of the functions we can call on that object is `executeSql`.

# Relational Database

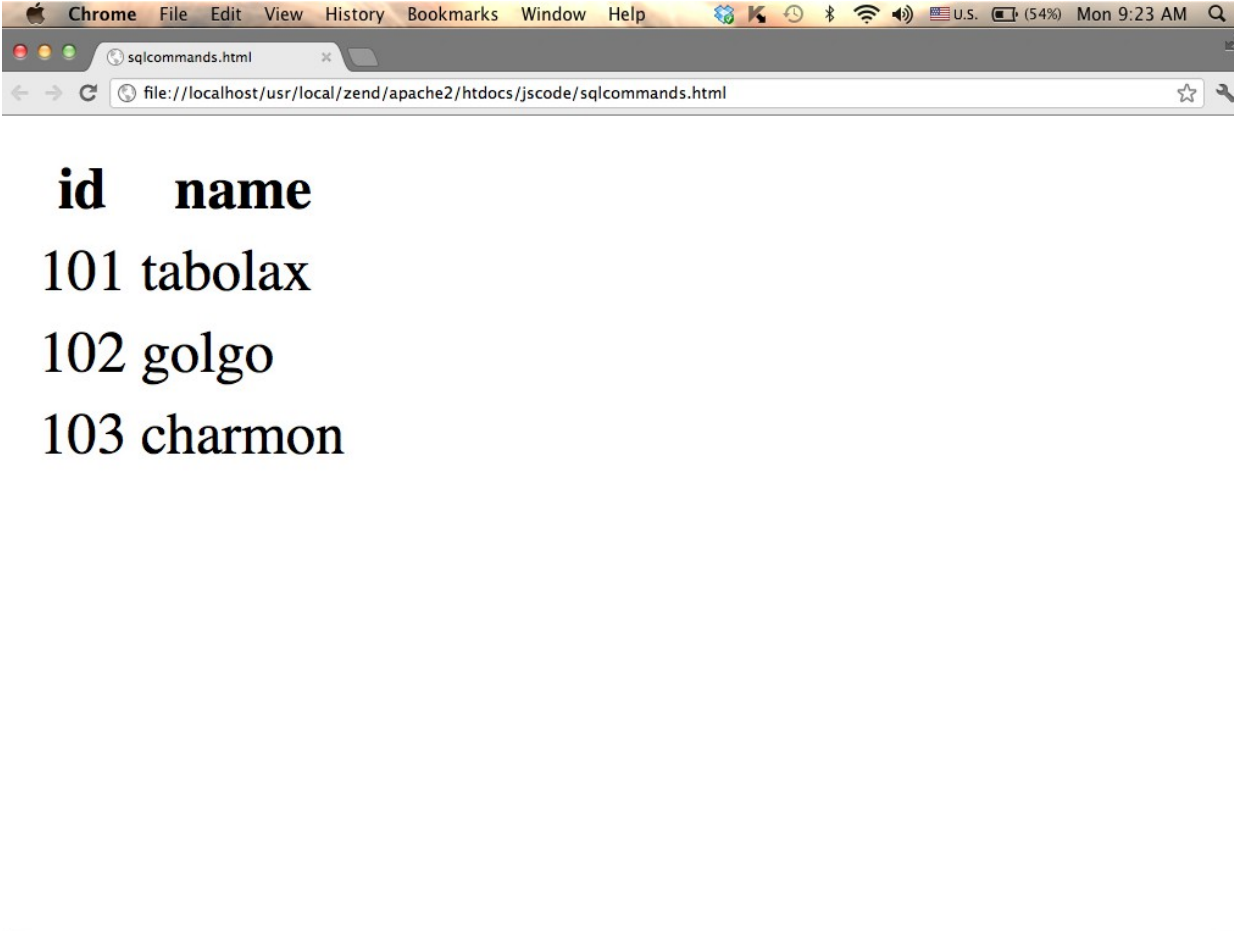


```
<script type="text/javascript">
try
{
    var db = window.openDatabase(
        'newwebstore',
        "1.0",
        'database for web store',
        100000);
    db.transaction(function(ob)
    {
        ob.executeSql(
            "CREATE TABLE IF NOT EXISTS products (pid INTEGER, pname TEXT)");
        ob.executeSql("INSERT INTO products VALUES(101,'tabolax')");
        ob.executeSql("INSERT INTO products VALUES(102,'golgo')");
        ob.executeSql("INSERT INTO products VALUES(103,'charmon')");
        ob.executeSql("SELECT * FROM products",
            [],dataHandler,errorHandler);
    }
    );
}
```

# Relational Database

```
catch(ex)
{
    document.write("ex="+ex);
}
function errorHandler(error)
{
    document.write("handling error "+error);
}
function dataHandler(transaction,data)
{
    document.write("<table>");
    document.write("<tr><th>id</th><th>name</th></tr>")
    size = data.rows.length;
    for(i=0;i<size;i++)
    {
        row = data.rows.item(i);
        document.write(
            "<tr><td>"+row['pid']+"</td><td>"+row['pname']+"</td></tr>");
    }
    document.write("</table>");
}
</script>
```

# Relational Database



A screenshot of a web browser window. The browser is Google Chrome, with the address bar showing a local file path: file:///localhost/usr/local/zend/apache2/htdocs/jrcode/sqlcommands.html. The page content displays a table with two columns, 'id' and 'name', and three rows of data.

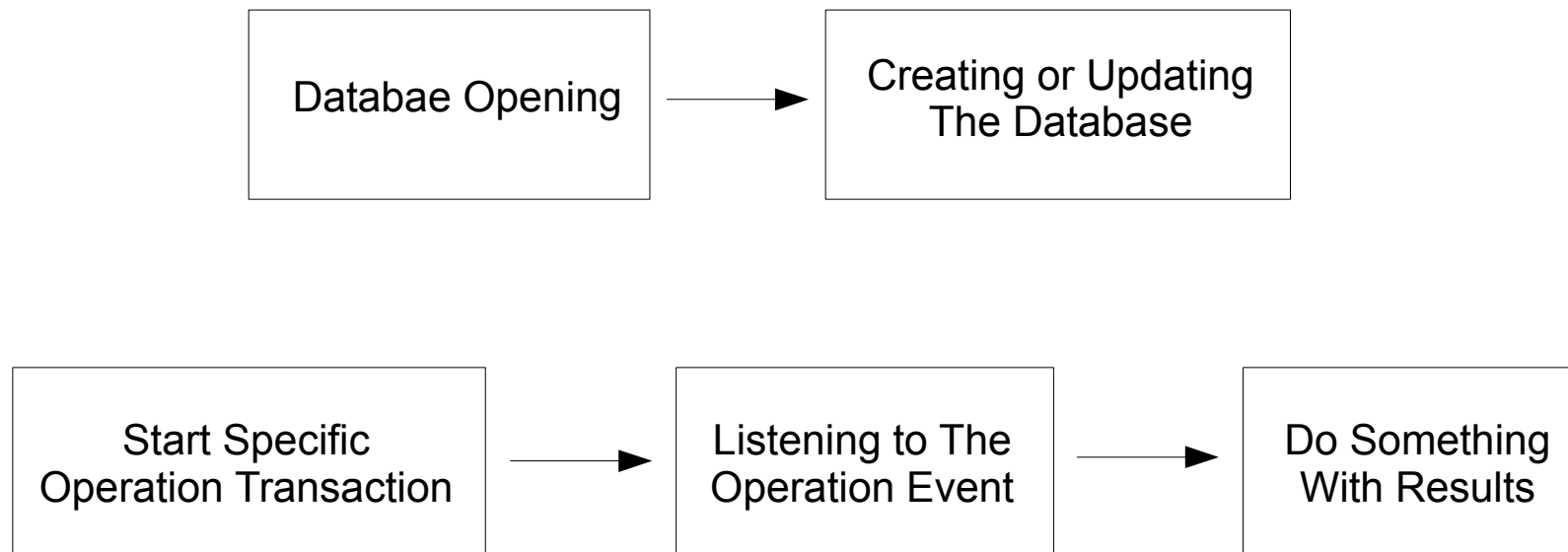
id	name
101	tabolax
102	golgo
103	charmon



# IndexedDB Database

- ❖ The IndexedDB allows us to develop offline web applications that store data into the web browser and have it available for the next time the user chooses to run the web application.
- ❖ The IndexedDB organizes the data in documents. Each and every document is identified using a unique id.
- ❖ When acquiring the object store that will keep the documents we set the identifier we are going to use.

# Using The IndexedDB Database



# The `indexedDB` Global Property

- ❖ This `window.indexedDB` global property holds a reference for an object that represents the IndexedDB database.

```
window.indexedDB = window.indexedDB ||  
                  window.mozIndexedDB ||  
                  window.webkitIndexedDB ||  
                  window.msIndexedDB;  
  
if (!window.indexedDB)  
{  
    console.log("The browser doesn't support IndexedDB")  
}
```

# The IDBOpenDBRequest Object

- ❖ In order to open an indexedDB database we should invoke the `open` function on the `IndexedDB` object we can access using the `window.indexedDB` property.
- ❖ Calling this function indirectly starts the process of opening the database. It doesn't open it immediately.

# The IDBOpenDBRequest Object

- ❖ The open function returns an object that represents the opening process of an IndexedDB database. It is an object of the `IDBOpenDBRequest` type. We can refer its `onerror`, `onsuccess` and `onupgradeneeded` properties in order to specify the exact functions we want to be invoked when these events take place.

# The IDBDatabase Object

- ❖ The successful result of the database opening process is getting an `IDBDatabase` object that represents the database.
- ❖ The `result` property in the `IDBOpenDBRequest` object, that represents the opening process, will hold the reference for the `IDBDatabase` object.

<https://developer.mozilla.org/en-US/docs/Web/API/IDBDatabase>

# The IDBDatabase Object

- ❖ We should assign the `onsuccess` property in our `IDBOpenDBRequest` object in order to refer the `result` property of the `IDBOpenDBRequest` object for getting the `IDBDatabase` object that represents the database.

```
demo.request.onsuccess = function(event)
{
    demo.db = demo.request.result;
    console.log("success: "+ demo.db);
};
```

# The IDBDatabase Object

- ❖ We can find the `onupgradedneeded` property in our `IDBOpenDBRequest` object. We should assign this property with the function that will be executed when there is a need to create the database or to upgrade the existing one.



# The IDBDatabase Object

- ❖ The function we assign this property with has a parameter that holds a reference for an object with the `request` property that refers another object with the `result` property that refers an `IDBDatabase` object, on which we can invoke the `createObjectStore()` function.

# The IDBDatabase Object

```
demo.request.onupgradeneeded = function(event) {  
    var database = event.target.result;  
    var objectStore = database.createObjectStore(  
        "cars", {keyPath: "id"});  
    for (var i in demo.data) {  
        objectStore.add(demo.data[i]);  
    }  
}
```

# The IDBTransaction Object

- ❖ The `transaction` method can be invoked on the `IDBDatabase` object that represents the database in order to start a new transaction for performing a specific operation.
- ❖ When calling this method we should pass over the names of the object stores we want to access.
- ❖ Calling the `transaction` method we will immediately get an `IDBTransaction` object that represents the transaction.

# The IDBTransaction Object

- ❖ Calling the `objectStore` method on our `IDBTransaction` object passing over the name of a specific object store we will get the reference for an `IDBObjectStore` object that represents that specific object store.

# The IDBObjectStore Object

- ❖ Calling the `objectStore` method on our `IDBTransaction` object passing over the name of a specific object store we will get the reference for an object that represents that specific object store.
- ❖ There are various methods we can invoke on an `IDBObjectStore` object that represents an object store.

<https://developer.mozilla.org/en-US/docs/Web/API/IDBObjectStore>

# IndexedDB Database Sample

```
<!DOCTYPE html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>IndexedDB Simple Demo</title>
  <script type="text/javascript">
    //creating our demo namespace
    var demo = {};
    //different web browsers might have different implementations
    window.indexedDB = window.indexedDB || window.mozIndexedDB ||
      window.webkitIndexedDB
      || window.msIndexedDB;

    //checking whether the web browser supports the IndexedDB database
    //if it doesn't then showing a message saying so
    if (!window.indexedDB) {
      console.log("The web browser doesn't support IndexedDB")
    }
  </script>
</head>
```

# IndexedDB Database Sample

```
//the data we want to store in our indexeddb database
demo.data = [
  { id: "1232523", brand: "Toyota", year: 2012, model: "Corola" },
  { id: "2343434", brand: "Mazda", year: 2008, model: "6" },
  { id: "2234345", brand: "Fiat", year: 2014, model: "500 Large"},
  { id: "2234333", brand: "Fiat", year: 2011, model: "Bravo"}
];
demo.db;
demo.request = window.indexedDB.open("cars", 1);
demo.request.onerror = function(event) {
  console.log("error: ");
};
demo.request.onsuccess = function(event) {
  demo.db = demo.request.result;
  console.log("success: "+ demo.db);
};
demo.request.onupgradeneeded = function(event) {
  demo.db = event.target.result;
  var objectStore = demo.db.createObjectStore("cars", {keyPath: "id"});
  for (var i in demo.data) {
    objectStore.add(demo.data[i]);
  }
}
```

# IndexedDB Database Sample

```
function readItem() {  
    var transaction = demo.db.transaction(["cars"]);  
    var objectStore = transaction.objectStore("cars");  
    var request = objectStore.get("12121212");  
    request.onerror = function(event) {  
        console.log("readItem(): cannot find the data item");  
    };  
    request.onsuccess = function(event) {  
        if(request.result) {  
            console.log("readItem(): "+ request.result.brand + ", "  
                +request.result.id + ", "+request.result.model);  
        } else {  
            console.log("readItem(): cannot find the item")  
        }  
    };  
}
```



# IndexedDB Database Sample

```
function readAllItems() {  
    var objectStore = demo.db.transaction("cars").objectStore("cars");  
  
    objectStore.openCursor().onsuccess = function(event) {  
        var cursor = event.target.result;  
        if (cursor) {  
            console.log("readAllItems(): key=" + cursor.key + " brand=" +  
                + cursor.value.brand + " model=" + cursor.value.model + " id=" +  
                + cursor.value.id);  
            cursor.continue();  
        }  
        else {  
            console.log("readAllItems(): no more entries!");  
        }  
    };  
}
```

# IndexedDB Database Sample

```
function addItem() {  
    var request = demo.db.transaction(["cars"], "readwrite")  
        .objectStore("cars")  
        .add({ id: "12121212", brand: "BMW", year: 2009, model: "318" });  
  
    request.onsuccess = function(event) {  
        console.log("addItem(): the new data item was added to your  
            database.");  
    };  
  
    request.onerror = function(event) {  
        console.log("addItem(): problem with adding the new data item to the  
            database ");  
    }  
}
```

# IndexedDB Database Sample

```
function removeItem() {  
    var request = demo.db.transaction(["cars"], "readwrite")  
        .objectStore("cars")  
        .delete("12121212");  
    request.onsuccess = function(event) {  
        console.log("removeItem(): the data item was removed from the  
            database");  
    };  
    request.onerror = function(event) {  
        console.log("removeItem(): problem with removing a data item from the  
            database");  
    }  
}  
  
</script>
```

# IndexedDB Database Sample

```
</head>

<body>
<form>
  <input type="button" onclick="removeItem()"
    value="deleting an item" />
  <input type="button" onclick="readItem()"
    value="reading a single item" />
  <input type="button" onclick="readAllItems()"
    value="reading all items" />
  <input type="button" onclick="addItem()"
    value="adding an item" />
</form>
</body>
</html>
```

# IndexedDB Database

deleting an item    reading a single item    reading all items

adding an item

Elements Network Sources Timeline Profiles Resources Audits Console

<top frame>

```
readAllItems(): no more entries!
readItem(): BMW, 12121212, 318
addItem(): problem with adding the new data item to the database
removeItem(): the data item was removed from the database
readItem(): cannot find the item
addItem(): the new data item was added to your database.
readItem(): BMW, 12121212, 318
removeItem(): the data item was removed from the database
```

Console Search Emulation Rendering

# Offline Storage

© 2010 Haim Michael. All Rights Reserved.

## Introduction

- ❖ HTML 5.0 supports a well structured offline storage solution.
- ❖ There are two types of offline storage possibilities. The first is the possibility to store session data separately for each tab. The second is the possibility to store data common to all tabs and all windows.

© 2010 Haim Michael. All Rights Reserved.

## Session Storage

- ❖ The session storage extends the capabilities we get when using cookies.
- ❖ Unlike the cookies mechanism that limits us for storing up to 4kilobytes of data the session storage allows us much more space.
- ❖ Unlike the cookies mechanism, the session data isn't sent automatically to the server every HTTP request.

© 2010 Haim Michael. All Rights Reserved.



## Session Storage

- ❖ Unlike cookies, session storage is tied to the browser tab. It isn't tied to the browser window. Each tab maintains its own session information.

© 2010 Haim Michael. All Rights Reserved.

## Session Storage

```
<html>
  <head>
    <title>session storage page 1</title>
  </head>
  <body>
    <h2>page 1</h2>
    <script language="javascript">
      sessionStorage.setItem('company', 'Zindell Technologies');
    </script>
    The 'company' (id) and 'Zindell Technologies' (value) were set as a key
    value pair in the session storage mechanism.
    <p>
      <a href="session_storage_2.html">next</a>
    </p>
  </body>
</html>
```

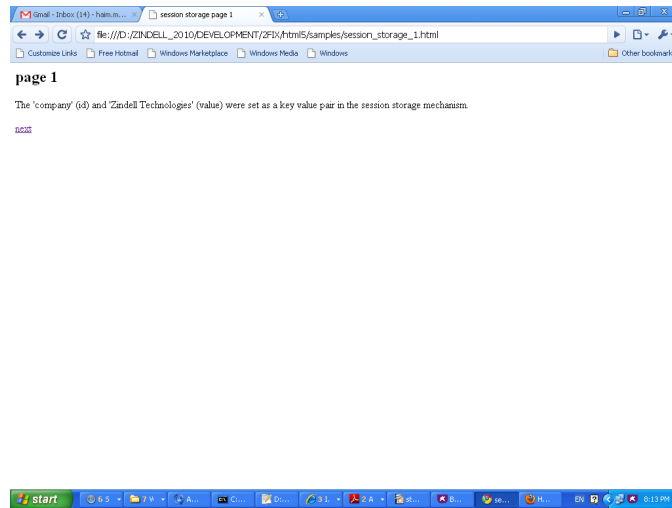
© 2010 Haim Michael. All Rights Reserved.

## Session Storage

```
<html>
  <head>
    <title>session storage page 2</title>
  </head>
  <body>
    <h2>page 2</h2>
    <script language="javascript">
      function showData()
      {
        alert(sessionStorage.getItem('company'));
      }
    </script>
    <form>
      <input type="button" value="click me" onClick="showData()">
    </form>
  </body>
</html>
```

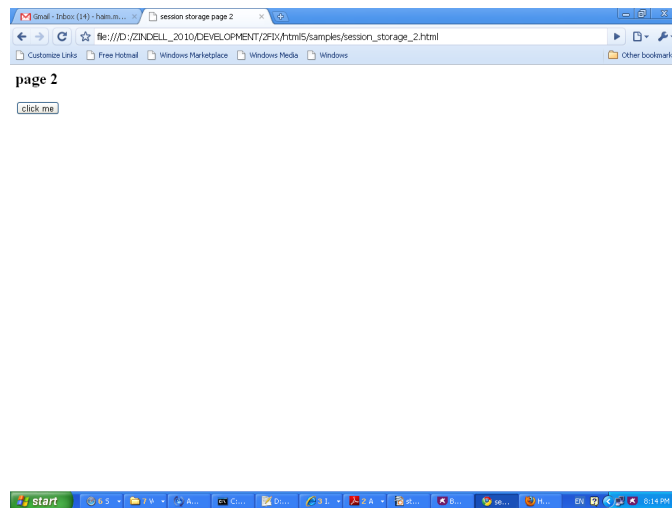
© 2010 Haim Michael. All Rights Reserved.

# Session Storage



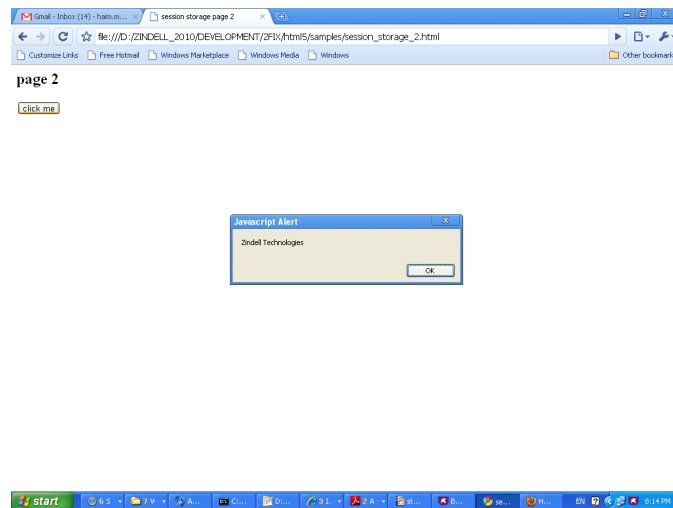
© 2010 Haim Michael. All Rights Reserved.

# Session Storage



© 2010 Haim Michael. All Rights Reserved.

# Session Storage



© 2010 Haim Michael. All Rights Reserved.

## Local Storage

- ❖ The local storage allows us to save data shared by all tabs. That data is kept even when the user closes the web browser.
- ❖ Instead of using the `sessionStorage` we should use the `localStorage`. The syntax is the same.

© 2010 Haim Michael. All Rights Reserved.

## Relational Database

- ❖ The HTML5 specifications provides us with a local relational database.
- ❖ We can access that database using the `window.openDatabase` property. This property holds a function that returns an object that represents a specific database on our web browser.

© 2010 Haim Michael. All Rights Reserved.



## Relational Database

- ❖ We can easily check whether the web browser supports the relational database (or not) by referring the `openDatabase` property. If the property doesn't exist the returned value will be equivalent to `false`.

© 2010 Haim Michael. All Rights Reserved.

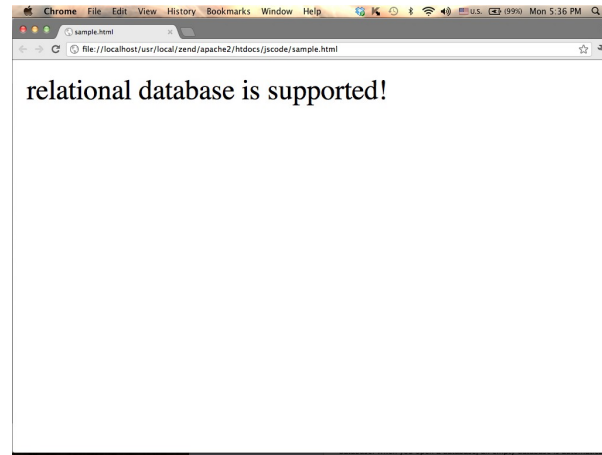
# Relational Database

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title></title>
  <script type="text/javascript">
    if(window.openDatabase)
    {
      document.write("relational database is supported!");
    }
    else
    {
      document.write("relational database IS NOT supported!");
    }
  </script>
</head>
<body>
</body>
</html>
```



© 2010 Haim Michael. All Rights Reserved.

# Relational Database



© 2010 Haim Michael. All Rights Reserved.

## Relational Database

- ❖ We can create a new relational database by calling the `window.openDatabase` function passing over the following arguments: the database short name, its version number, its long detailed name and its maximum size in bytes.
- ❖ The returned value is a reference for a new object that represents the new relational database on our web browser.

© 2010 Haim Michael. All Rights Reserved.

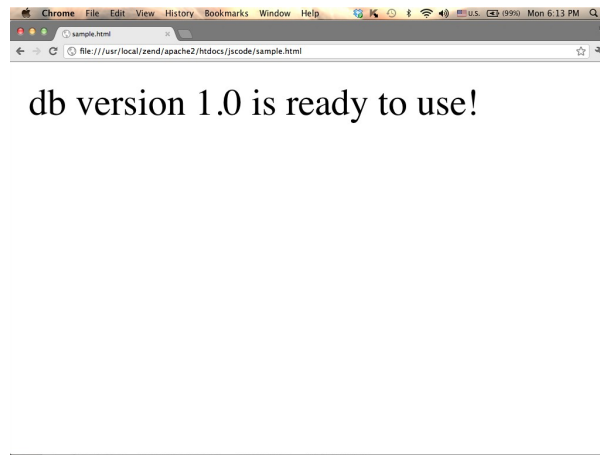
# Relational Database

```
<html>
<head>
  <title></title>
  <script type="text/javascript">
    try
    {
      var db = window.openDatabase(
        'webstore',
        '1.0',
        'database for web store',
        100000);
      document.write("db version " +db.version+ " is ready to use!");
    }
    catch(ex)
    {
      document.write("ex="+ex);
    }
  </script>
</head>
<body></body>
</html>
```



© 2010 Haim Michael. All Rights Reserved.

# Relational Database



© 2010 Haim Michael. All Rights Reserved.

## Relational Database

- ❖ We can pass over an empty string as the database version, get the object that represents the data base, check its version and if necessary call the `changeVersion` function in order to change it.

© 2010 Haim Michael. All Rights Reserved.

# Relational Database

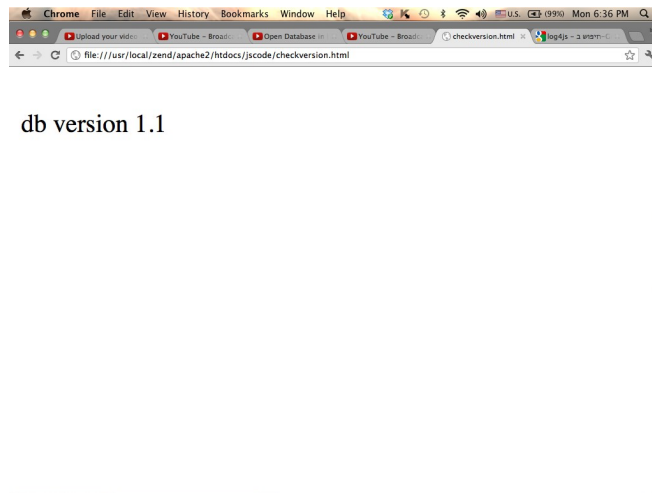
```
<html>
<head>
  <title></title>
  <script type="text/javascript">
    try
    {
      var db = window.openDatabase(
        'webstore',
        "",
        'database for web store',
        100000);
      document.write("<br>db version " +db.version);
      db.changeVersion("1.0","1.1");
    }
    catch(ex)
    {
      document.write("ex="+ex);
    }
  </script>
</head>
<body></body>
</html>
```



© 2010 Haim Michael. All Rights Reserved.



# Relational Database



© 2010 Haim Michael. All Rights Reserved.

## Relational Database

- ❖ In order to perform SQL command on our database we should call the `transaction` function on the object that represents the database.
- ❖ We should pass over a function with one argument. That function will be called and an object that represents the transaction will be passed over to it. One of the functions we can call on that object is `executeSql`.

© 2010 Haim Michael. All Rights Reserved.

# Relational Database

```
<script type="text/javascript">
try
{
    var db = window.openDatabase(
        'newwebstore',
        "1.0",
        'database for web store',
        100000);
    db.transaction(function(ob)
    {
        ob.executeSql(
            "CREATE TABLE IF NOT EXISTS products (pid INTEGER, pname TEXT)");
        ob.executeSql("INSERT INTO products VALUES(101,'tabolax')");
        ob.executeSql("INSERT INTO products VALUES(102,'golgo')");
        ob.executeSql("INSERT INTO products VALUES(103,'charmon')");
        ob.executeSql("SELECT * FROM products",
            [],dataHandler,errorHandler);
    }
    );
}
```



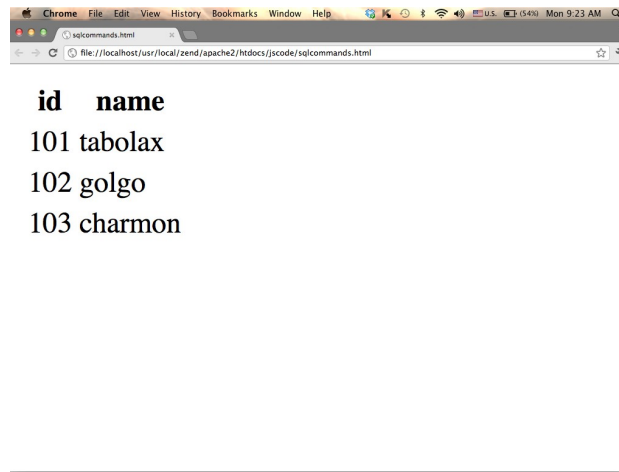
© 2010 Haim Michael. All Rights Reserved.

## Relational Database

```
catch(ex)
{
    document.write("ex="+ex);
}
function errorHandler(error)
{
    document.write("handling error "+error);
}
function dataHandler(transaction,data)
{
    document.write("<table>");
    document.write("<tr><th>id</th><th>name</th></tr>")
    size = data.rows.length;
    for(i=0;i<size;i++)
    {
        row = data.rows.item(i);
        document.write(
            "<tr><td>"+row['pid']+ "</td><td>"+row['pname']+ "</td></tr>");
    }
    document.write("</table>");
}
</script>
```

© 2010 Haim Michael. All Rights Reserved.

# Relational Database



A screenshot of a web browser window displaying a table. The browser's address bar shows the file path: file:///localhost:/usr/local/Zend/apache2/htdocs/jscodes/sqlcommands.html. The table has two columns, 'id' and 'name', and contains three rows of data.

id	name
101	tabolax
102	golgo
103	charmon

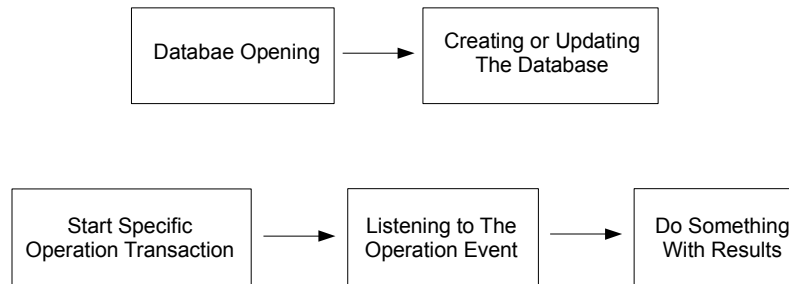
© 2010 Haim Michael. All Rights Reserved.

## IndexedDB Database

- ❖ The IndexedDB allows us to develop offline web applications that store data into the web browser and have it available for the next time the user chooses to run the web application.
- ❖ The IndexedDB organizes the data in documents. Each and every document is identified using a unique id.
- ❖ When acquiring the object store that will keep the documents we set the identifier we are going to use.

© 2010 Haim Michael. All Rights Reserved.

## Using The IndexedDB Database



© 2010 Haim Michael. All Rights Reserved.

## The indexedDB Global Property

- ❖ This `window.indexedDB` global property holds a reference for an object that represents the IndexedDB database.

```
window.indexedDB = window.indexedDB ||
                  window.mozIndexedDB ||
                  window.webkitIndexedDB ||
                  window.msIndexedDB;

if (!window.indexedDB)
{
    console.log("The browser doesn't support IndexedDB")
}
```

© 2010 Haim Michael. All Rights Reserved.



## The IDBOpenDBRequest Object

- ❖ In order to open an indexedDB database we should invoke the `open` function on the `IndexedDB` object we can access using the `window.indexedDB` property.
- ❖ Calling this function indirectly starts the process of opening the database. It doesn't open it immediately.

© 2010 Haim Michael. All Rights Reserved.

## The `IDBOpenDBRequest` Object

- ❖ The open function returns an object that represents the opening process of an IndexedDB database. It is an object of the `IDBOpenDBRequest` type. We can refer its `onerror`, `onsuccess` and `onupgradeneeded` properties in order to specify the exact functions we want to be invoked when these events take place.

© 2010 Haim Michael. All Rights Reserved.

## The IDBDatabase Object

- ❖ The successful result of the database opening process is getting an `IDBDatabase` object that represents the database.
- ❖ The `result` property in the `IDBOpenDBRequest` object, that represents the opening process, will hold the reference for the `IDBDatabase` object.

<https://developer.mozilla.org/en-US/docs/Web/API/IDBDatabase>

© 2010 Haim Michael. All Rights Reserved.

## The IDBDatabase Object

- ❖ We should assign the `onsuccess` property in our `IDBOpenDBRequest` object in order to refer the `result` property of the `IDBOpenDBRequest` object for getting the `IDBDatabase` object that represents the database.

```
demo.request.onsuccess = function(event)
{
    demo.db = demo.request.result;
    console.log("success: "+ demo.db);
};
```

© 2010 Haim Michael. All Rights Reserved.

## The IDBDatabase Object

- ❖ We can find the `onupgradedneeded` property in our `IDBOpenDBRequest` object. We should assign this property with the function that will be executed when there is a need to create the database or to upgrade the existing one.

© 2010 Haim Michael. All Rights Reserved.

## The IDBDatabase Object

- ❖ The function we assign this property with has a parameter that holds a reference for an object with the `request` property that refers another object with the `result` property that refers an `IDBDatabase` object, on which we can invoke the `createObjectStore()` function.

© 2010 Haim Michael. All Rights Reserved.

## The IDBDatabase Object

```
demo.request.onupgradeneeded = function(event) {  
  var database = event.target.result;  
  var objectStore = database.createObjectStore(  
    "cars", {keyPath: "id"});  
  for (var i in demo.data) {  
    objectStore.add(demo.data[i]);  
  }  
}
```

© 2010 Haim Michael. All Rights Reserved.

## The IDBTransaction Object

- ❖ The `transaction` method can be invoked on the `IDBDatabase` object that represents the database in order to start a new transaction for performing a specific operation.
- ❖ When calling this method we should pass over the names of the object stores we want to access.
- ❖ Calling the `transaction` method we will immediately get an `IDBTransaction` object that represents the transaction.

© 2010 Haim Michael. All Rights Reserved.



## The IDBTransaction Object

- ❖ Calling the `objectStore` method on our `IDBTransaction` object passing over the name of a specific object store we will get the reference for an `IDBObjectStore` object that represents that specific object store.

© 2010 Haim Michael. All Rights Reserved.

## The IDBObjectStore Object

- ❖ Calling the `objectStore` method on our `IDBTransaction` object passing over the name of a specific object store we will get the reference for an object that represents that specific object store.
- ❖ There are various methods we can invoke on an `IDBObjectStore` object that represents an object store.

<https://developer.mozilla.org/en-US/docs/Web/API/IDBObjectStore>

© 2010 Haim Michael. All Rights Reserved.

## IndexedDB Database Sample

```
<!DOCTYPE html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>IndexedDB Simple Demo</title>
  <script type="text/javascript">
    //creating our demo namespace
    var demo = {};
    //different web browsers might have different implementations
    window.indexedDB = window.indexedDB || window.mozIndexedDB ||
      window.webkitIndexedDB
      || window.msIndexedDB;

    //checking whether the web browser supports the IndexedDB database
    //if it doesn't then showing a message saying so
    if (!window.indexedDB) {
      console.log("The web browser doesn't support IndexedDB")
    }
  </script>
</head>
```

© 2010 Haim Michael. All Rights Reserved.

## IndexedDB Database Sample

```
//the data we want to store in our indexeddb database
demo.data = [
  { id: "1232523", brand: "Toyota", year: 2012, model: "Corola" },
  { id: "2343434", brand: "Mazda", year: 2008, model: "6" },
  { id: "2234345", brand: "Fiat", year: 2014, model: "500 Large"},
  { id: "2234333", brand: "Fiat", year: 2011, model: "Bravo"}
];
demo.db;
demo.request = window.indexedDB.open("cars", 1);
demo.request.onerror = function(event) {
  console.log("error: ");
};
demo.request.onsuccess = function(event) {
  demo.db = demo.request.result;
  console.log("success: "+ demo.db);
};
demo.request.onupgradeneeded = function(event) {
  demo.db = event.target.result;
  var objectStore = demo.db.createObjectStore("cars", {keyPath: "id"});
  for (var i in demo.data) {
    objectStore.add(demo.data[i]);
  }
}
```

© 2010 Haim Michael. All Rights Reserved.

## IndexedDB Database Sample

```
function readItem() {  
    var transaction = demo.db.transaction(["cars"]);  
    var objectStore = transaction.objectStore("cars");  
    var request = objectStore.get("12121212");  
    request.onerror = function(event) {  
        console.log("readItem(): cannot find the data item");  
    };  
    request.onsuccess = function(event) {  
        if(request.result) {  
            console.log("readItem(): "+ request.result.brand + ", "  
                +request.result.id + ", "+request.result.model);  
        } else {  
            console.log("readItem(): cannot find the item")  
        }  
    };  
}
```

© 2010 Haim Michael. All Rights Reserved.

## IndexedDB Database Sample

```
function readAllItems() {  
    var objectStore = demo.db.transaction("cars").objectStore("cars");  
  
    objectStore.openCursor().onsuccess = function(event) {  
        var cursor = event.target.result;  
        if (cursor) {  
            console.log("readAllItems(): key=" + cursor.key + " brand=" +  
                cursor.value.brand + " model=" + cursor.value.model + " id=" +  
                cursor.value.id);  
            cursor.continue();  
        }  
        else {  
            console.log("readAllItems(): no more entries!");  
        }  
    };  
}
```

© 2010 Haim Michael. All Rights Reserved.

## IndexedDB Database Sample

```
function addItem() {
    var request = demo.db.transaction(["cars"], "readwrite")
        .objectStore("cars")
        .add({ id: "12121212", brand: "BMW", year: 2009, model: "318" });

    request.onsuccess = function(event) {
        console.log("addItem(): the new data item was added to your
            database.");
    };

    request.onerror = function(event) {
        console.log("addItem(): problem with adding the new data item to the
            database ");
    }
}
```

© 2010 Haim Michael. All Rights Reserved.

## IndexedDB Database Sample

```
function removeItem() {  
    var request = demo.db.transaction(["cars"], "readwrite")  
        .objectStore("cars")  
        .delete("12121212");  
    request.onsuccess = function(event) {  
        console.log("removeItem(): the data item was removed from the  
            database");  
    };  
    request.onerror = function(event) {  
        console.log("removeItem(): problem with removing a data item from the  
            database");  
    }  
}  
</script>
```

© 2010 Haim Michael. All Rights Reserved.

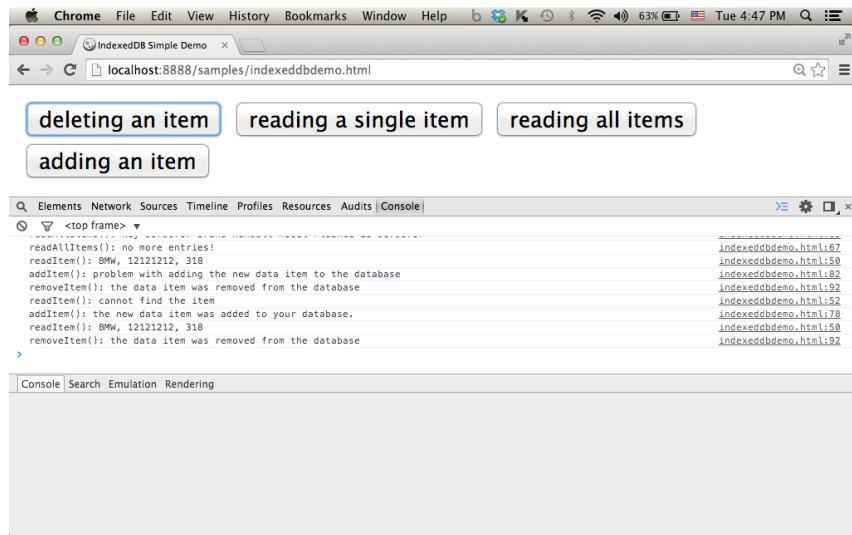


## IndexedDB Database Sample

```
</head>
<body>
<form>
  <input type="button" onclick="removeItem()"
    value="deleting an item" />
  <input type="button" onclick="readItem()"
    value="reading a single item" />
  <input type="button" onclick="readAllItems()"
    value="reading all items" />
  <input type="button" onclick="addItem()"
    value="adding an item" />
</form>
</body>
</html>
```

© 2010 Haim Michael. All Rights Reserved.

# IndexedDB Database



© 2010 Haim Michael. All Rights Reserved.