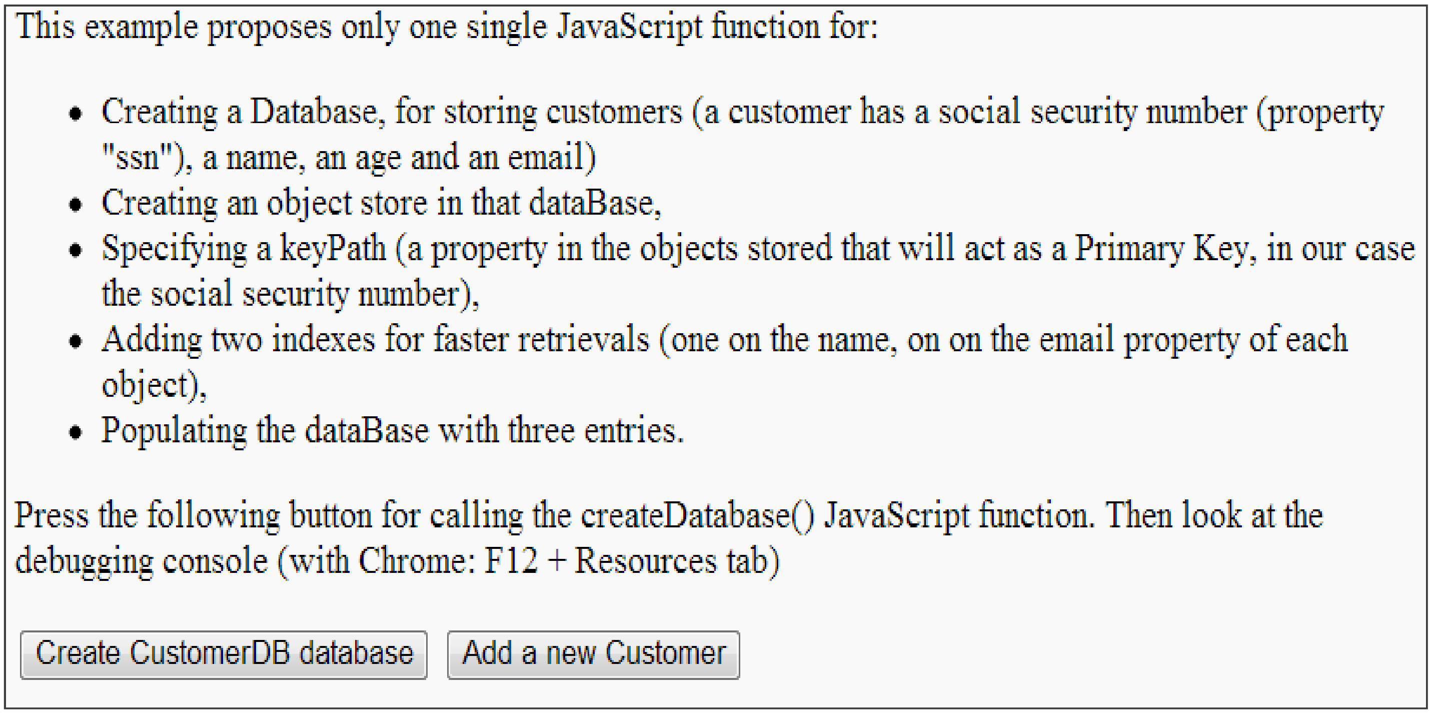
Inserting data in an object store

EXAMPLE 1: BASIC STEPS

Online example at JSBin:

Execute this example and look at the IndexedDB object store content from the Chrome dev tools (F12 or cmd-alt-i). One more customer should have been added.

Be sure to click on the "create database" button before clicking the "insert new customer" button.

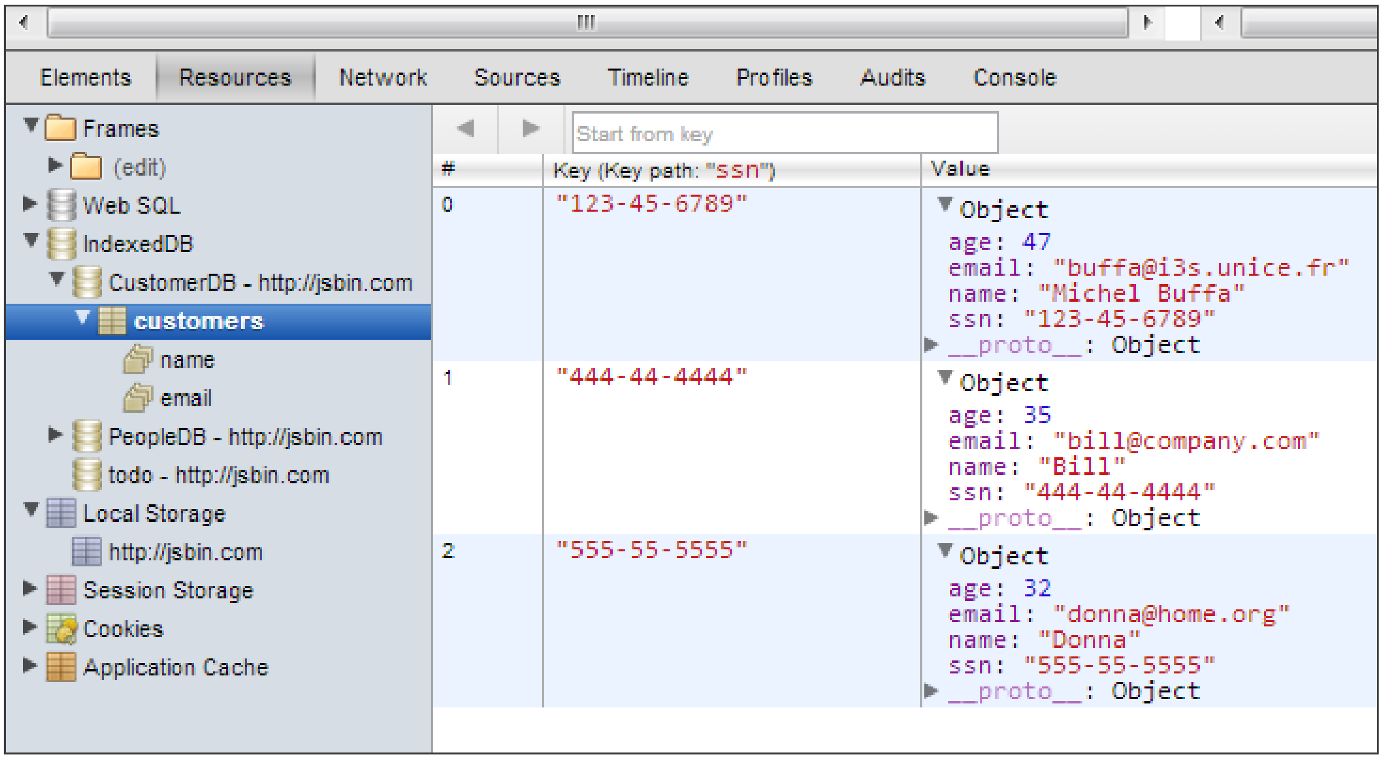


This example proposes only one single JavaScript function for:

- Creating a Database, for storing customers (a customer has a social security number (property "ssn"), a name, an age and an email)
- Creating an object store in that dataBase,
- Specifying a keyPath (a property in the objects stored that will act as a Primary Key, in our case the social security number),
- Adding two indexes for faster retrievals (one on the name, on on the email property of each object),
- Populating the dataBase with three entries.

Press the following button for calling the createDatabase() JavaScript function. Then look at the debugging console (with Chrome: F12 + Resources tab)

The next screenshot shows the IndexedDB object store in Chrome dev. tools (use the "Resources" tab). Clicking the "Create CustomerDB" database creates or opens the database, and clicking "Add a new Customer" button adds a customer named "Michel Buffa" into the object store:



```
▼ Frames
  ▶ (edit)
▶ Web SQL
▼ IndexedDB
  ▼ CustomerDB - http://jsbin.com
    ▼ customers
        name
        email
  ▶ PeopleDB - http://jsbin.com
    todo - http://jsbin.com
▼ Local Storage
    http://jsbin.com
▶ Session Storage
▶ Cookies
▶ Application Cache
```

#	Key (Key path: "ssn")	Value
0	"123-45-6789"	▼Object age: 47 email: "buffa@i3s.unice.fr" name: "Michel Buffa" ssn: "123-45-6789" ▶ __proto__: Object
1	"444-44-4444"	▼Object age: 35 email: "bill@company.com" name: "Bill" ssn: "444-44-4444" ▶ __proto__: Object
2	"555-55-5555"	▼Object age: 32 email: "donna@home.org" name: "Donna" ssn: "555-55-5555" ▶ __proto__: Object

Code from the example, explanations:

We just added a single function into the example from the previous section - the function `AddACustomer()` that adds one customer:

```
{ ssn: "123-45-6789", name: "Michel Buffa",age: 47, email:
 "buffa@i3s.unice.fr" }
```

Here is the complete source code of the `addACustomer` function:

```
function addACustomer() {
   // 1 - get a transaction on the "customers" object store
   // in readwrite, as we are going to insert a new object

 var transaction =db.transaction(["customers"], "readwrite");
   // Do something when all the data is added to the
database.
   // This callback is called after transaction has been
completely
   // executed (committed)
```

```
       transaction.oncomplete = function(event) {
           alert("All done!");
11.    };
       // This callback is called in case of error (rollback)
       transaction.onerror = function(event) {

    console.log("transaction.onerror" +event.target.errorCode);
       };
       // 2 - Init the transaction on the objectStore
       var objectStore =transaction.objectStore("customers");
21.    // 3 - Get a request from the transaction for adding a new
    object
       var request = objectStore.add({ ssn: "123-45-6789",
                                       name:"Michel Buffa",
                                       age: 47,

     email:"buffa@i3s.unice.fr" });
       // The insertion was ok
       request.onsuccess = function(event) {
           console.log("Customer with ssn=
    " +event.target.result + "
                               added.");
       };
       // the insertion led to an error (object already in the
    store,
       // for example)
       request.onerror = function(event) {
36.        console.log("request.onerror, could not insert
    customer,
37.                            errcode = " +event.target.errorCode);
       };
    }
```

Explanations:

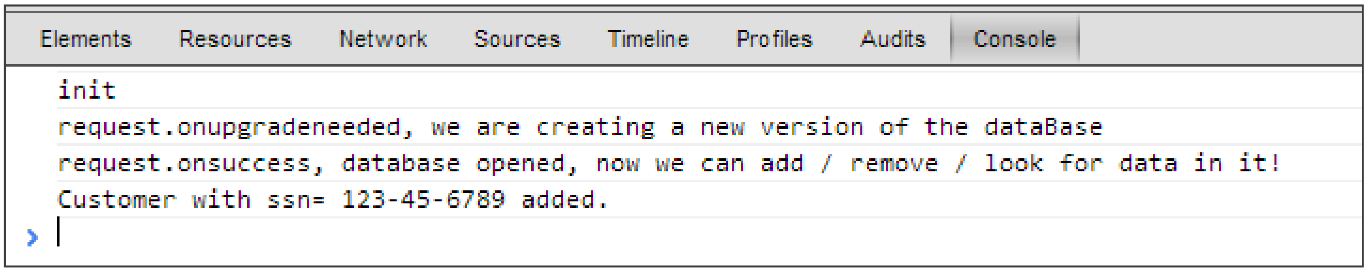
In the code above, in lines 4, 19 and 22 are the main calls you have to perform in order to add a new object to the store:

1. Create a transaction.
2. Map the transaction on the object store.
3. Create an "add" request that will take part in the transaction.

The different callbacks are in lines 9 and 14 for the transaction, and in lines 28 and 35 for the request.

You may have several requests for the same transaction. Once all requests have finished, the `transaction.oncomplete` callback is called. In any other case the `transaction.onerror` callback is called, and the datastore remains unchanged.

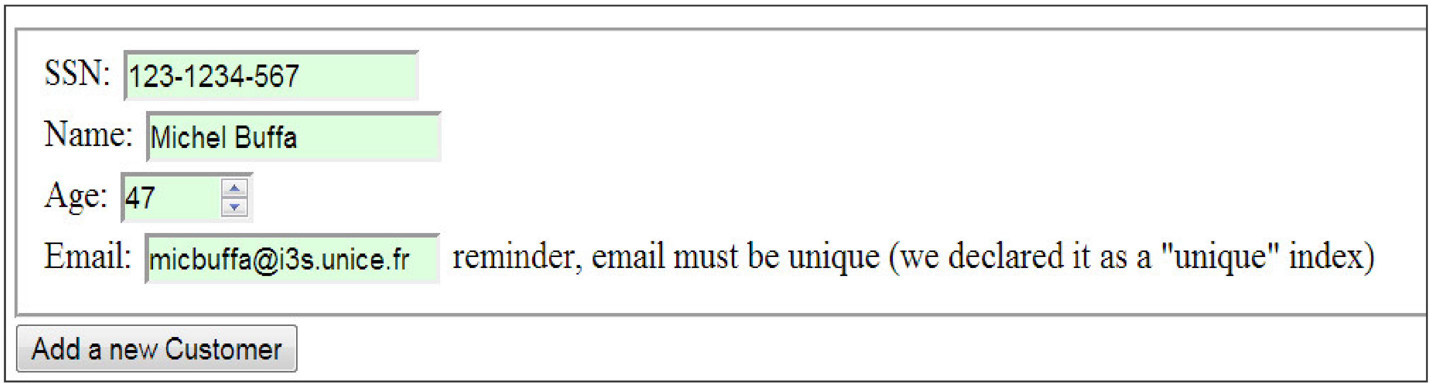
Here is the trace from the dev tools console:



```
init
request.onupgradeneeded, we are creating a new version of the dataBase
request.onsuccess, database opened, now we can add / remove / look for data in it!
Customer with ssn= 123-45-6789 added.
```

EXAMPLE 2: ADDING A FORM AND VALIDATING INPUTS

Online example available at JSBin:



SSN:	123-1234-567
Name:	Michel Buffa
Age:	47
Email:	micbuffa@i3s.unice.fr reminder, email must be unique (we declared it as a "unique" index)

You can try this example by following these steps:

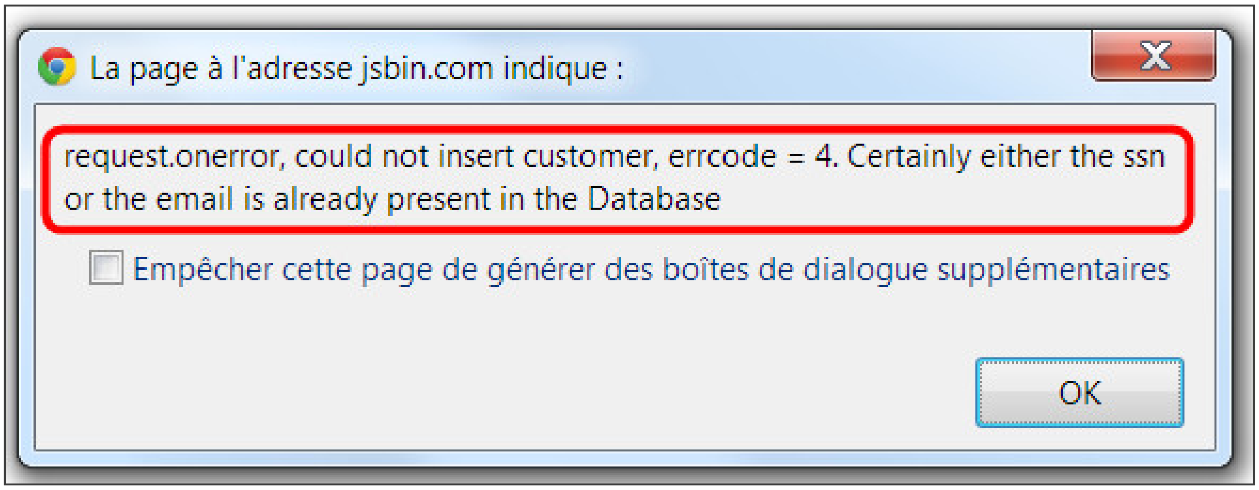
1. Press first the "Create database" button

2. then enter a new customer in the form
3. click the "add a new Customer" button
4. then press F12 or cmd-alt-i to use the Chrome dev tools and check for the IndexedDB store content. Sometimes it is necessary to refresh the view (right click on IndexedDB/refresh), and sometimes it is necessary to close/open the dev. tools to have a view that shows the changes (press F12 or cmd-alt-i twice). Chrome dev. tools are a bit strange from time to time.

This time we added some tests for checking that the database is open before trying to insert an element + we added a small form for entering a new customer.

Notice that the insert will fail and display an alert with an error message if:

- The `ssn` is already present in the database. This property has been declared as the keyPath (a sort of primary key) in the object store schema, and it should be unique:`db.createObjectStore("customers", { keyPath: "ssn" });`
- The `email` is already present in the object store. Remember that in our schema, the `email` property is an index that we declared unique: `objectStore.createIndex("email", "email", { unique: true });`
- Try to insert the same customer twice, or different customers with the same `ssn`. An alert like this should pop up:


La page à l'adresse jsbin.com indique :

request.onerror, could not insert customer, errcode = 4. Certainly either the ssn or the email is already present in the Database

Empêcher cette page de générer des boîtes de dialogue supplémentaires
OK

Here is the updated version of the HTML code of this example:

```
<fieldset>
  SSN: <input type="text" id="ssn"placeholder="444-44-4444"
              required/><br>
  Name: <input type="text" id="name"/><br>
  Age: <input type="number" id="age" min="1"max="100"/><br>
  Email:<input type="email" id="email"/>reminder, email must
be
               unique (we declared it as a "unique" index)
<br>
</fieldset>
<button onclick="addACustomer();">Add a new Customer</button>
```

And here is the new version of the `addACustomer()` JavaScript function:

```
function addACustomer() {
   if(db === null) {
      alert('Database must be opened, please click the Create
             CustomerDB Database first');
      return;
   }

 var transaction =db.transaction(["customers"], "readwrite");
   // Do something when all the data is added to the
database.
   transaction.oncomplete = function(event) {
     console.log("All done!");
   };
   transaction.onerror = function(event) {

 console.log("transaction.onerror" +event.target.errorCode);
   };
   var objectStore =transaction.objectStore("customers");
   // adds the customer data
   var newCustomer={};
   newCustomer.ssn =document.querySelector("#ssn").value;
   newCustomer.name =document.querySelector("#name").value;
   newCustomer.age =document.querySelector("#age").value;
   newCustomer.email =document.querySelector("#email").value;
   alert('adding customer ssn=' +newCustomer.ssn);

   var request =objectStore.add(newCustomer);
   request.onsuccess = function(event) {
     console.log("Customer with ssn=
" +event.target.result + "
                  added.");
     };

  request.onerror = function(event) {
     alert("request.onerror, could not insert customer,
errcode = "
           + event.target.errorCode +
           ". Certainly either the ssn or the email is
already
           present in the Database");
  };
}
```

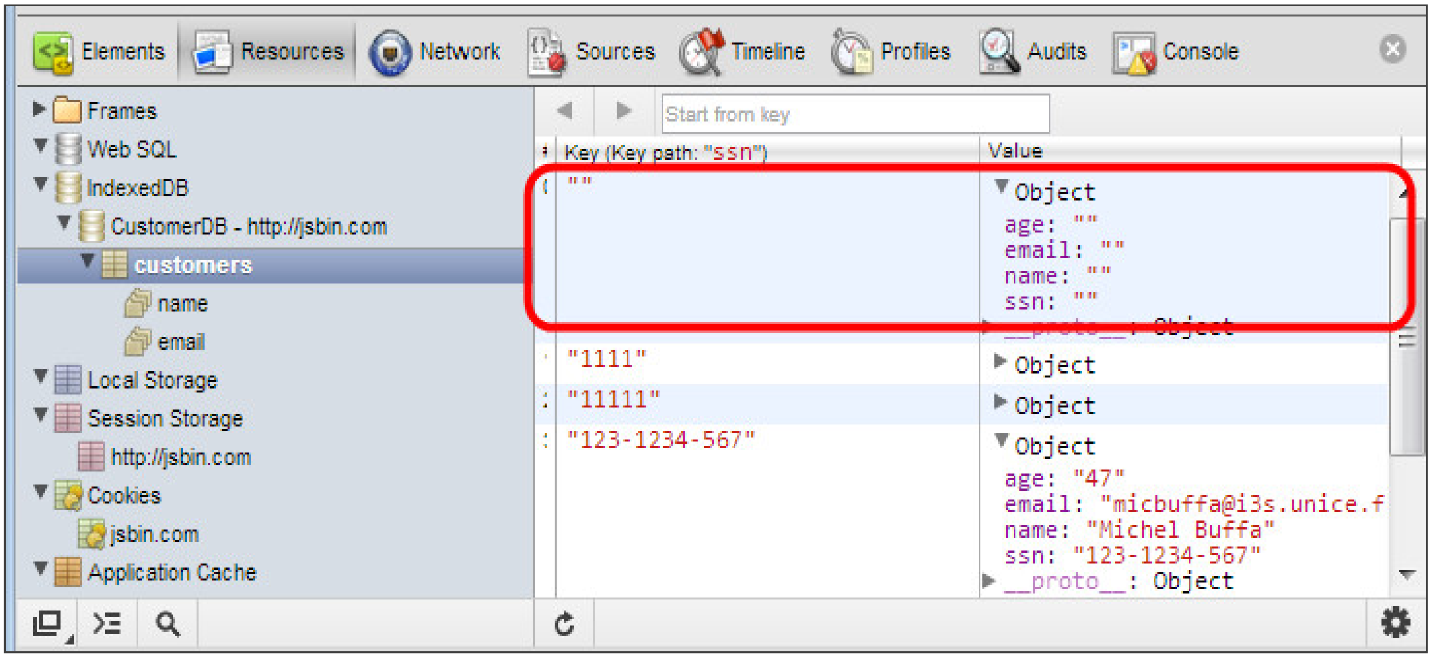
It is also possible to shorten the code of the above function by chaining the different operations using the "." operator (getting a transaction from the db, opening the store, adding a new customer, etc.).

Here is the short version:

```
var request = db.transaction(["customers"],"readwrite")
 .objectStore("customers")
 .add(newCustomer);
```

The above code does not perform all the tests, but you may encounter such a way of coding (!).

Also note that it works if you try to insert empty data:



```
Key (Key path: "ssn")     Value
""                        ▼Object
                            age: ""
                            email: ""
                            name: ""
                            ssn: ""
                          ▶__proto__: Object
"1111"                    ▶Object
"11111"                   ▶Object
"123-1234-567"            ▼Object
                            age: "47"
                            email: "micbuffa@i3s.unice.f
                            name: "Michel Buffa"
                            ssn: "123-1234-567"
                          ▶__proto__: Object
```

Indeed, entering an empty value for the keyPath or for indexes is a valid value (in the IndexedDB sense). In order to avoid this, you should add more JavaScript code. We will let you do this as an exercise.