HOUR 21
Web Storage in HTML5

What You'll Learn in This Hour:

- How to use local storage
- How to detect local storage support
- When to use local storage versus cookies
- How to add local databases using Web SQL and Indexed DB
- How to build an application that uses local storage

Before HTML5 the only way you could store information was either by adding it to the DOM (Document Object Model) or using cookies. But both have drawbacks.

Storing data in the HTML is fairly easy to do but it requires JavaScript, and it can only store data as long as the browser is open and on the application. As soon as the user's session ends, that data is lost. Plus, data that is stored in the HTML is visible to anyone.

Cookies are limited to around 4KB of data, which is not a lot of space to store information. Plus, cookies are sent with every HTTP request, which can slow down your application. Cookies are unencrypted, so data stored that way can be sniffed unless your entire application is served on an SSL (Secure Sockets Layer) connection. SSL slows pages down, too.

This hour you will learn about web storage and how you can use space on your users' hard drives to store information about preferences, runtime state, and more. HTML5 local storage provides you with more space that is associated with the user agent and is only transferred to and from the server when the application asks for it.

However, web storage involves more than just local storage. You also have two ways to create local databases and store information in them: Web SQL and Indexed Database or Indexed DB. This hour will show you more about these two specifications and how you can use them.

What Is Web Storage?

Web storage is a separate API that was originally part of the HTML5 specification, but now is a separate document within the aegis of HTML5. It is sometimes called *Local Storage* or *DOM Storage* depending upon the browser manufacturer.

By the Way

Several Plug-ins Are Available to Create Persistent Storage

Browser manufacturers and plug-in writers have made many attempts to create a form of persistent storage. Microsoft created `userData` (http://msdn.microsoft.com/en-us/library/ms531424(VS.85).aspx). Adobe created Local Shared Objects in Flash (www.adobe.com/products/flashplayer/articles/lso/). The Dojo Toolkit has dojox.storage integrated into it (http://dojotoolkit.org/api/1.6/dojox/storage/manager). Google Gears, Adobe AIR, and others also provide forms of local persistent storage, but all of these require either a specific browser or a plug-in for people to use.

Web storage allows web developers to store data from web applications as key/value pairs on the user's local machine. This data persists even if the user leaves the application, closes the browser, or turns off the machine. When the user comes back to the web application, that web storage data is available to the application.

Storing data is useful for many web applications. The simplest applications use web storage to store user data so that users can pick up right where they left off when they return to the site. Another way to use web storage is to store data locally when a user is offline that would normally be stored on a server (see Hour 20, "Offline Web Applications").

How Web Storage Is Different from Cookies

Many people make the mistake of thinking that web storage is the same thing as a cookie, only bigger. Although it's true that web storage provides a lot more space than cookies, other differences exist as well:

- Web storage is only delivered to the web page when the client requests it, whereas cookies are sent with every HTTP request.
- Web storage can only be accessed by the client (JavaScript), whereas cookies can be accessed by server scripts such as PHP as well as JavaScript.
- Cookies are specific to the browser and domain, but don't make a distinction between windows in the same browser, whereas web storage (specifically, session storage) distinguishes between windows as well as browser and domain.

Remember that web storage is not just a better cookie. Cookies serve a purpose and you should continue to use them. For example, cookies have two flags, `HttpOnly` and `secure`, that help make cookies safer. The `HttpOnly` flag tells the browser that this cookie should not be available to client-side scripts. The `secure` flag requires that the cookie only be served over a secure (SSL) connection.

You should only store session data in cookies that have been secured with at least `HttpOnly` and preferably the `secure` flag, as well. Hackers can use session information to get information about your customers.

Thinking of web storage as a place to store application preferences, non-sensitive user information, and things such as saved games and saved content is best. Also, web storage isn't as widely supported as cookies are. Cookies have been supported in browsers since 1995. Web storage is supported by all modern browsers, but only specific versions:

- Android 2.0
- Chrome 4
- Firefox 3.5
- Internet Explorer 8.0
- iOS 2.0
- Opera 10.5
- Safari 4.0

Session Storage and Local Storage

The two different types of web storage in the Web Storage specification are local storage and session storage.

Local storage is name/value pairs that are stored on the local disk. Every site has a separate storage area. The data stored in this area persists even if the user leaves the site or closes the web browser. The data is only available to the domain that added it, and that data is only sent when a client on that domain requests it.

Did you Know?

Web Storage Stores All Data as Strings

When you store data in web storage, it will be stored as a string. This means that if you have structured data (such as database information) or numbers (integers and floating point), you need to convert them to strings so that they can be stored in web storage. Don't forget to use the `parseInt()` and `parseFloat()` methods to convert your integers and floats back when you retrieve the data.

Local storage is best for storing non-sensitive data that takes up a lot of space. It is for data that needs to be saved across sessions so that users have access to it when they return to the site.

Session storage is name/value pairs that are stored on the local disk for the length of the page session. Session storage will deliver stored data to user agents on the same domain and in the same window as when the data was stored. After that window closes (except in certain cases), the stored data is removed. If a new tab or window is opened to the same site, a new instance of session storage is created.

Did you Know?

Session Storage Can Survive Reloads

If a browser has support for resuming sessions after a reload, then the session storage can be saved as well. For example, in Firefox, if the browser crashes, Firefox will restart and offer to restore your previous session. Chrome, Opera, and Safari have similar features, and plug-ins are available to add this feature to IE. In these browsers, data stored in session storage will be restored as well.

Session storage was created to solve a problem that cookies have. If you go to a website that uses cookies to store data, and then open that site in a second window, the cookie data from the first window can be accessed and contaminate the cookie data in the second window.

Session storage is most useful for temporary data that should be saved and restored if the browser session is accidentally refreshed. Like local storage, you should not store sensitive data in session storage.

Using Web Storage

As with all new HTML5 features, you should check to make sure that the browser supports web storage. To do this you can use Modernizr as mentioned in Hour 4, “Detecting Mobile Devices and HTML5 Support,” or you can use this function:

```
function supportsWebStorage() {
    try {
        return 'localStorage' in window && window['localStorage'] !== null;
    } catch (e) {
        return false;
    }
}
```

When using web storage, the first thing you do is define a storage object. This tells the user agent whether you’re using local storage or session storage. These objects are

- `window.localStorage`
- `window.sessionStorage`

As with other objects, the `window` portion is assumed.

Try It Yourself ▼

Saving Gift Status in an E-commerce Store

In online stores, a user can easily accidentally click away or close the browser window and lose his or her choices. This Try It Yourself shows how to use the `sessionStorage` object to save the gift status of a purchase in case of accidental browser refresh.

1. Add a checkbox, a `div` to hold the status, and a button to the page:

```
<p>
<label>
<input type=checkbox id=gift>This is a gift
</label>
<p>
<button id=checkGift>Check Gift Status</button>
<p><div id="isItaGift"></div>
```

2. In a `<script>` element at the bottom of the document, put a script to check for web storage support:

```
<script>
function supportsWebStorage() {
    try {
        return 'localStorage' in window && window['localStorage']
➥!== null;
    } catch (e) {
        return false;
    }
}
</script>
```

3. Inside the jQuery document ready function, add the check for support `if` statement:

```
$(document).ready(function(e) {
    if (supportsWebStorage()) {
        // web storage functions
    } else {
        // fallback functions
    }
});
```

4. In the web storage functions area, add the `sessionStorage` object for when the user clicks the checkbox:

```
$("#gift").click(function() {
    if ($(this).attr('checked')) {
       sessionStorage.setItem("gift", "yes");
    } else {
        sessionStorage.setItem("gift", "no");
    }
});
```

5. Add a function to check the web storage when the user clicks the button:

```
$("#checkGift").click(function() {
    var giftStatus = sessionStorage.getItem("gift");
    if (giftStatus == "yes") {
        $("#isItaGift").html("<p>This item is a gift.");
        $("#gift").attr("checked", "checked");
    } else {
        $("#isItaGift").html("<p>This item is not a gift.");
        $("#gift").removeAttr("checked");
    }
});
```

To check that this code is working, click the checkbox and then refresh the page. The browser should clear the checkbox, but if you click the button, it will update the page with the correct status. You can view this session storage example online at www.html5in24hours.com/examples/web-storage-example.html.

The previous Try It Yourself section shows you how to use two of the four methods on the web storage interface. These methods are

- **`getItem(`*`key`*`)`**—This method gets the value of the item identified by the `key` from storage.
- **`setItem(`*`key, value`*`)`**—This method creates the item identified by the `key` and sets the value to `value` in storage.
- **`removeItem(`*`key`*`)`**—This method removes the item identified by the key from storage.
- **`clear()`**—This method completely empties all items from storage for the current object.
- **`key(`*`n`*`)`**—This method iterates through all the currently stored keys and gets the total number of values in the storage area. You can also use this to call a `key` by index value (`n`).

The storage object also has a `length` attribute that, when called, returns the number of key/value pairs associated with the object.

Watch Out!

Web Storage Can Be Hacked

As with cookies, web storage can be hacked if your site is not secured against cross site scripting (XSS). The name of any items you store in local storage is written in clear text inside your JavaScript, so a hacker can easily use XSS to collect that data.[1] To protect your readers, you should always use a `<meta charset>` element at the top of your pages before any text. Don't use web storage for session IDs; use session storage rather than local storage, and *do not* store sensitive data in local storage.

If you call the `setItem()` method with a `key` that already exists, the data will be overwritten with the new value. If you call the `getItem()` method and no `key` is stored the browser will return `null`.

The Web Storage API also provides an event to keep track of when the storage area changes: `storage`. This event creates a `StorageEvent` object that has the following properties:

- **`key`**—The named key that was added, removed, or modified
- **`oldValue`**—The previous value (including `null`) of the item
- **`newValue`**—The new value (including `null` if the item was removed) of the item
- **`url`**—The page that called a method that triggered the change

The last thing you should know about the `storage` event is that it cannot be cancelled. This is just an event to tell you what happened in the browser.

Web SQL and Indexed DB

One drawback to web storage is that it provides only storage for key/value pairs. You cannot save structured data. However, two APIs are in use now that allow you to store complex relational data and query that data using SQL: Web SQL and Indexed DB.

Web SQL Database

Web SQL Database is a specification that allows you to make SQL calls on the client side and store SQL databases. It is based on SQLite. Web SQL is supported by Android 2.1, Chrome 4, iOS 3.2, Opera 10.5, and Safari 3.2.

[1] "HTML5, Local Storage, and XSS." Application Security. July 12, 2010. http://michael-coates.blogspot.com/2010/07/html5-local-storage-and-xss.html. Referenced June 21, 2011.

Watch Out!

Web SQL Has Been Discontinued

On November 18, 2010, the W3C decided to discontinue development of the Web SQL specification. It is still supported by some browsers, and it could be revisited in the future, but right now it is not being worked on and there are no plans to work on it.

First you should detect whether the browser supports Web SQL DB:

```
function hasWebSql() {
    return !!window.openDatabase;
}
```

You can then open a database. To create a Web SQL database use the `openDatabase()` method. If the database doesn't exist, Web SQL DB will create it. This method takes five attributes:

- The name of the database
- The version number of the database
- A text description
- An estimated size for the database in bytes
- The creation callback script

You open a database like this:

```
var db = openDatabase("addressBook", "0.1", "an online address book",
➥2000000);
```

Remember that the version number (0.1 in the above example) is required, and to open an existing database, you must use the same version number. Plus, although a `changeVersion()` method exist, it is not widely supported.

The creation callback is an optional script that runs when the database is created. Most of the time, you don't need this option, and in the preceding example it is not included.

The beauty of Web SQL DB is that it is a fairly simple API. After you have an open database, you use the `transaction()` method on that database to execute SQL commands with the `executeSql()` method. The `executeSql()` method takes SQL commands. If you don't know SQL, you should start by reviewing the documentation on SQLite at its website (http://sqlite.org/). To add a table and two entries, you write:

```
db.transaction(function(tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS names (id unique text)');
    tx.executeSql('INSERT INTO names (id, text) VALUES (1, "Joe")');
    tx.executeSql('INSERT INTO names (id, text) VALUES (2, "Sarah")');
});
```

Watch Out!

SQLite Ignores Data Types

When you are inserting data into a database with Web SQL, you are inserting it into an SQLite database, and SQLite ignores data types. It just uses simple strings. So you should always do data verification to ensure that the data you are inserting is the type of data you want to insert—such as dates, integers, floating point numbers, and so on.

If you wanted to add external data, such as from a form, you would want to check it for malicious code (such as SQL injection) first and then pass the `executeSQL` method a variable. You enter dynamic values like this:

```
tx.executeSql('INSERT INTO names (id, text) VALUES (?,?)'
➥[uniqueId, username]);
```

The `uniqueId` and `username` are variables that are mapped to the query.

Then, if you want to get data from the database, you use the SQL `SELECT` function:

```
db.transaction(function(tx) {
    tx.executeSql('SELECT * FROM addr', [], function(tx, names) {
        var len = names.row.length, i;
        alert(len + " names found");
    });
});
```

Indexed Database API

The Indexed Database API (or IndexedDB or IDB) exposes the object store. This can have databases with records and fields that you use just like a database, but rather than accessing them through SQL calls, you use methods on the object store. IndexedDB is supported by Firefox 4 and Chrome 11.

As usual, first you detect for IndexedDB support:

```
function hasIndexedDb() {
    return !!window.indexedDB;
}
```

You then open a database with the `open()` method:

```
var req = window.indexedDB.open("addressBook", "an online address book");
```

Watch Out!

IndexedDB Requires Browser Prefixes

Although the specification lists `window.indexedDB.open()` as the method to open indexedDB databases, you need to use browser-specific prefixes to get it to work reliably. Firefox version 3 and lower uses `moz_` (with an underscore), and Firefox 4 and up uses `moz` (without). Safari and Chrome use the `webkit` prefix. So, when you work with IDB it helps to initialize the object itself:

```
window.indexedDB = window.indexedDB || window.mozIndexedDB ||
window.webkitIndexedDB;
window.IDBKeyRange = window.IDBKeyRange ||
window.webkitIDBKeyRange;
window.IDBTransaction = window.IDBTransaction ||
window.webkitIDBTransaction;
```

You have to set up the database for a first-time user:

```
req.onsuccess = function(e) {
var db = e.result;
if (db.version != "1") {
    //first visit, initialize database
    var createdObjectStoreCount = 0;
    var objectStores = [
        { name: "fnames", keypath: "id", autoIncrement: true },
        { name: "lnames", keypath: "id", autoIncrement: true },
        { name: "emails", keypath: "id", autoIncrement: true }
    ];
    function objectStoreCreated(e) {
        if (++createdObjectStoreCount == objectStores.length) {
            db.setVersion("1").onsuccess = function(e) {
                loadData(db);
            };
        }
    }
    for (var i=0; i< objectStores.length; i++) {
        var params = objectStores[i];
        req = db.createObjectStore(params.name, params.keyPath,
➥params.autoIncrement);
        req.onsuccess = objectStoreCreated;
    }
} else {
    //returning user, no initialization
    loadData(db);
}
};
```

Then to add an item to the database, you use the `add()` method:

```
req.onsuccess = function(e) {
    var objectStore = e.result.objectStore("fnames");
    objectStore.add("Jennifer").onsuccess = function(e) {
        alert("'Jennifer' added to database with id " + e.result);
    };
};
```

To list all the items in a table you would use the `openCursor()` method, and enumerate through it until it returned `null`:

```
req.onsuccess = function(e) {
    req = e.result.objectStore("fnames").openCursor();
    req.onsuccess = function(e) {
        var cursor = e.result;
        if (!cursor) {
            return
        }
        var element = $("#fnameList").append("<p>"+cursor.value.name);
        cursor.continue();
    };
};
```

Try It Yourself ▼

Building a Birthday Application for iOS and Android

This application lets iOS and Android users store and retrieve birthdays of friends and family in a local Web SQL database. Note that this application won't work in Firefox because the application uses only Web SQL, and Firefox only supports IndexedDB. It also doesn't work in Internet Explorer because IE doesn't support any web databases.

1. In your HTML5 document, write a form to enter the birthday data and display the month's birthdays:

```
<article>
<hgroup>
<h1>Birthday List</h1>
</hgroup>
<p>Store and retrieve birthdays of friends and family.
<h3>
<button id="prev">&amp;lt; Previous</button>
Birthdays in <span id=thisMonth>this Month</span>
<button id="next">Next &amp;gt;</button>
</h3>
<table id="birthdays"></table>
</article>
<section id="input">
<h3>Store a Birthday</h3>
<p>Name: <input id="fullname" placeholder="full name" required
➥pattern="[A-Z,a-z, ]+">
(only letters and spaces allowed)
```

```
<p>Birthday:
<select id="birthdayMonth" required>
    <option value="0">January
    <option value="1">February
    <option value="2">March
    <option value="3">April
    <option value="4">May
    <option value="5">June
    <option value="6">July
    <option value="7">August
    <option value="8">September
    <option value="9">October
    <option value="10">November
    <option value="11">December
</select>
<input type="number" id="birthdayDay" min="1" max="31"
size="4" step="1" required>
<p><button id="addBirthday">Store Birthday</button>
</section>
<section id="output">
<table id="submitted"></table>
</section>
```

2. Add links to script files for general functions and the Web SQL–specific functions:

```
<script src="birthday-list-websql.js"></script>
<script src="birthday-list.js"></script>
```

3. In the `birthday-list.js` file, check for Web SQL support and add code to display a message if the device doesn't support it:

```
if (hasWebSql()) {
    //Open or create the web SQL database
    var db = openWebSqlDb();
    //Initialize the database
    initWebSqlDb(db);
} else {
    // use plain web storage
    $("hgroup").append("<h2 id=warning>This Application Requires Web
➥SQL.</h2>");
}
// begin general functions
function hasWebSql() {
    return !!window.openDatabase;
}
```

4. In the `birthday-list-websql.js` file, write the `openWebSqlDb` and `initWebSqlDb` functions:

```
function openWebSqlDb() {
    var db=openDatabase('MyBirthdayDb','1.0','my birthdays app',
➥2 * 1024 * 1024);
    return db;
}
```

```
function initWebSqlDb(db) {
    db.transaction(function(tx) {
        tx.executeSql('CREATE TABLE IF NOT EXISTS birthdays(
➥id integer primary key autoincrement, fullname, birthdayMonth,
➥birthdayDay)');
    });
    // load current month's birthdays
    var thisMonth = new Date().getMonth();
    window.onload = listBirthdaysWebS(thisMonth);
}
```

5. Upon loading, the script will load any birthdays stored in the database, so add the `listBirthdaysWebS` function to the `birthday-list-websql.js` file:

```
function listBirthdaysWebS(month) {
    db.transaction(function(tx) {
        tx.executeSql('SELECT * FROM birthdays', [], function(tx,
➥results) {
            var len=results.rows.length, i;
            for(i=0; i<len; i++) {
                if(results.rows.item(i).birthdayMonth == month) {
                    var prettyMonth =
➥getMonthName(parseInt(results.rows.item(i).birthdayMonth));
                    createTableRow(results.rows.item(i).id,
➥results.rows.item(i).fullname,prettyMonth,
➥results.rows.item(i).birthdayDay,"birthdays");
                }
            }
        });
    });
}
```

6. You will also need functions to write the table rows and display month names in the `birthday-list.js` file:

```
function createTableRow
➥(insertId,inputFullName,inputBirthdayMonth,inputBirthdayDay,tableId) {
    var birthdayRow = $("<tr id=b"+insertId+"></tr>");
    var id = $("<td><p>"+insertId+"</td>");
    var fullname = $("<td><p>"+inputFullName+"</td>");
    var birthdayMonth = $("<td><p>"+inputBirthdayMonth+"</td>");
    var birthdayDay = $("<td><p>"+inputBirthdayDay+"</td>");
    // if you add an indexeddb section,
    // you will need to create a separate remove button for that db
    var removeButton = $('<td><p><button onclick="removeBirthdayWebS('
➥+ insertId + ')">Delete</button></td>');
    birthdayRow.append(fullname)
               .append(birthdayMonth)
               .append(birthdayDay)
               .append(removeButton);
    $("#"+tableId).append(birthdayRow);

}
```

```
function getMonthName(month) {
    month = parseInt(month);
    switch(month) {
        case 0:
            month = "January";
            break;
        case 1:
            month = "February";
            break;
        case 2:
            month = "March";
            break;
        case 3:
            month = "April";
            break;
        case 4:
            month = "May";
            break;
        case 5:
            month = "June";
            break;
        case 6:
            month = "July";
            break;
        case 7:
            month = "August";
            break;
        case 8:
            month = "September";
            break;
        case 9:
            month = "October";
            break;
        case 10:
            month = "November";
            break;
        case 11:
            month = "December";
            break;
    }
    return month;
}
```

7. The last thing to go in the `birthday-list.js` file is the document ready function. In it add jQuery to change the text "`this Month`" to the current month name:

```
$(document).ready(function(){
    var monthNum = new Date().getMonth();
    var thisMonthIs = getMonthName(monthNum);
    $("#prev").attr("class", monthNum);
    $("#next").attr("class", monthNum);
    $("#thisMonth").html(thisMonthIs);
});
```

8. Add the "next" and "previous" button support into the document ready function:

```
$("#prev").click( function(e) {
    var curMonth = $(this).attr("class");
    var newMonth = parseInt(curMonth) -1;
    if (curMonth == "0") {
        newMonth = "11";
    }
    $("#prev").attr("class", newMonth);
    $("#next").attr("class", newMonth);
    $("#birthdays tr").remove();
    $("#thisMonth").html(getMonthName(newMonth));
    listBirthdaysWebS(newMonth);
}); // end previous month

$("#next").click( function(e) {
    var curMonth = $(this).attr("class");
    var newMonth = parseInt(curMonth) +1;
    if (curMonth == "11") {
        newMonth = "0";
    }
    $("#next").attr("class", newMonth);
    $("#prev").attr("class", newMonth);
    $("#birthdays tr").remove();
    $("#thisMonth").html(getMonthName(newMonth));
    listBirthdaysWebS(newMonth);
}); // end next month
```

9. Add the functions to add and delete birthdays to your `birthday-list-web-sql.js` file:

```
function addBirthdayWebS(db,inputFullName,inputBirthdayMonth,
➥inputBirthdayDay) {
    var prettyMonth = getMonthName(parseInt(inputBirthdayMonth));
    var curMonth = $("#next").attr("class");
    db.transaction(function(tx) {
        tx.executeSql('INSERT INTO birthdays(
➥fullname,birthdayMonth,birthdayDay) VALUES (?,?,?)',
➥[inputFullName,inputBirthdayMonth,inputBirthdayDay],
➥function(tx, results) {
            createTableRow(results.insertId,inputFullName,
➥prettyMonth,inputBirthdayDay,"submitted");
            if (!$('#stored').length) {
                $("#submitted").before("<h3 id=stored>Stored</h3>");
            }
            // add current month additions to current birthday list
            if (inputBirthdayMonth == curMonth) {
                createTableRow(results.insertId,inputFullName,
➥prettyMonth,inputBirthdayDay,"birthdays");
            }
        });
    });
}

function removeBirthdayWebS(id) {
    db.transaction(function(tx) {
        tx.executeSql('DELETE FROM birthdays WHERE id=?', [id],
➥function() {
            //Dynamically remove the birthday from the list
            $("#b"+id).remove();
        });
    });
}
```

10. Finally, call the `addBirthday` function when a user clicks on the Store Birthday button in the document ready function:

```
$("#addBirthday").click( function(e) {
    var inputFullName=$("#fullname").val();
    var inputBirthdayMonth=$("#birthdayMonth").val();
    var inputBirthdayDay=$("#birthdayDay").val();
    addBirthdayWebS(db,inputFullName,inputBirthdayMonth,
➥inputBirthdayDay);
}); // end addBirthday
```

You can see an example of this application at www.html5in24hours.com/examples/birthday-list.html.

Summary

This hour you learned about web storage and how you can store more data on the client side than you can with cookies. Web storage consists of local storage and session storage.

Local storage allows you to store a lot of data in name=value pairs that are stored locally on the hard drive. Items stored in this area are kept even if the browser is shut down or the computer is turned off.

Session storage allows you to store data in name=value pairs for the duration of a session. Session storage is specific to the window that it is opened in, so a user opening two windows to your website opens two different session storage instances.

This hour also covered two ways to store more structured data. These are Web SQL, which stores data using SQLite, and IndexedDB, which uses a JavaScript API to store data.

Q&A

Q. ***If web storage is supported by all major browsers, shouldn't I just use it instead of cookies?***

A. As mentioned in this hour, cookies still have a valid purpose. They can be more secure, and so using them for session IDs is important. Plus, cookies are more familiar to most developers and so easier to implement. Also, users can delete local storage (just like they can cookies) and may be more likely to if it takes up too much space.

Q. ***Do I have to know SQL to use Web SQL and IndexedDB?***

A. Because Web SQL uses SQLite to select and query the database, you need to write SQL calls to create and query the database. IndexedDB is easier to use because it doesn't require that you learn SQL.

Q. ***If Web SQL Database was discontinued in 2010, why should I use it now? Won't it just be obsolete?***

A. Several browsers support Web SQL Database and not all of them currently support other client-side database options. Plus, it is a good way to familiarize yourself with the idea of client-side databases.

You should also keep in mind that Web SQL is the only database currently supported on mobile devices. If your application needs to work on mobile, then you should use it instead of IndexedDB. Ultimately, you should use the technology that best fits your users.

Q. ***Are there easy ways to create structured data for storing in local storage?***

A. Many developers use `JSON.stringify` and `JSON.parse` to convert JavaScript string objects from JavaScript (`stringify`) and back (`parse`).

Q. ***Can I use the same storage API to access different data stores?***

A. Data stores are protected in a similar fashion to cookies. If your script is on the same domain as the stores, then you can access it, even from different web pages, but you cannot access a data store set by another domain.

Workshop

The workshop contains quiz questions to help you process what you've learned in this chapter. Try to answer all the questions before you read the answers. See Appendix A, "Answers to Quizzes," for answers.

Quiz

1. What are three differences between web storage and cookies?
2. What is the difference between local storage and session storage?
3. What is the benefit to using Web SQL or IndexedDB instead of web storage?
4. How do you detect Web SQL or IndexedDB support?

Exercises

1. Add search functionality to the Birthday list application you built. Let your users search by name or month.
2. Add IndexedDB functionality to the Birthday list application. This will let the application work in Firefox. Add fallback support for browsers that don't support Web SQL or IndexedDB using basic web storage.
3. Convert the to-do list that you created in Hour 14, "Editing Content and User Interaction with HTML5," into a to-do list that saves the data to the local drive. The items should be stored in persistent storage and be retrieved when the user returns to the page.