**Offline Storage-HTML5**

Table of Contents

[INTRODUCTION](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451939) 4

[OLDER OFFLINE TECHNIQUIES](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451940) 5

[Cookies](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451941) 5

[Plugin Based Storage](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451942) 5

[Broswer Specific Features](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451942) 5

[Disadvantages](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 6

[WHY USE OFFLINE STORAGE?](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451940) 7

[Browser Support](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 7

[OFFLINE STORAGE TYPES](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451940) 8

[**Web Storage**](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 8

[LocalStorage](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 9

[Syntax](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 9

[Output](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 9

[SessionStorage](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 8

[Syntax](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 10

[Output](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 10

[Difference Between LocalStorage and SessionStorage](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 11

[Advantages](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 11

[Disadvantages](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 12

[**WebSQL**](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 13

[Syntax](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 13

[Create](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 14

[Insert](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 14

[HTML Input form](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 15

Output16

[Select](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 16

[Update](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 16

[Delete](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 16

[Advantages](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 17

[Disadvantages](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 17

[**IndexedDB**](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 18

[Syntax](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 18

[Create](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 19

[Output](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 19

[Insert](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 20

Select20

[Delete](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 20

[Advantages](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 21

[Disadvantages](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 21

[Differnce between WebSQL and IndexedDB](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 21

[**CONCLUSION**](file:///D:\Desktop%20Backup_06_06_2016\Paper\whitepapaer-palani\Palanivelu%20S-9112-Offline%20Storage-HTML5.docx#_Toc423451943) 22

# Introduction

This white paper explains about main problem with HTTP as the main transport layer of the Web is that it is stateless. It means that when you use an application and then close it, its state will be reset the next time you open it. If you close an application on your desktop and re-open it, its most recent state is restored. This is why, as a developer, you need to store the state of your interface somewhere. Normally, this is done server-side, and you would check the user name to know which state to revert to. But what if you don’t want to force people to sign up? This is where local storage comes in. You would keep a key on the user’s computer and read it out when the user returns.

One of the biggest evolutions happening around HTML5 today is the availability of persistent storage on browsers. Before HTML5, we had no choice other than using cookies to store some kind of data on client side. With HTML5, you have several choices for storing your data, depending on what you want to store.

The most broadly used technology is called **Web storage**, an API for persistent key-value data storage which most major browsers have supported for a while. Web Storage has persistent storage called **LocalStorage**, and temporary storage called **SessionStorage** which will be cleared after a session.

**WebSQL Database** is a relational database solution for browsers and **Indexed Database** is an object store. Since it's considered to be the final solution for storing structured data on browsers, its support and usage is gradually expanding.

# Older Offline Storage Techniques

While there's only one kind of application cache, there's a whole flock of offline storage capabilities available, or becoming available. Let's begin with a quick survey of older storage techniques, those that have been around several years and are not generally associated with the "HTML5" label.

* Cookies,
* Plugin Based Storage,
* Browser-specific features

### **Cookies**

Cookies have been around since the early days of the web. They were originally intended to associate a little data with the user, though these days, most applications only store identifying information, and store the rest of the user's data on the server. Still, the fact that cookies can store some data does put them in the category of offline storage.

### **Plugin Based Storage**

Plugin based storage has been supported by several plugins, notably Flash, Java, and Google Gears. Since Flash 6, there has been the concept of a local shared object. Java applets have also been an option, since Java can write to local files. Google Gears is a third example of a plugin, and in its case, offline storage was one of the main selling points right from its inception.

### **Browser-specific features**

Browser-specific features are capabilities that are available, but outside the realm of standard web technologies. They typically have an "underground" nature, being little-known and lightly documented. For example, TiddlyWiki and the derivative twFile JQuery plugin use IE-specific and Firefox-specific APIs to access the underlying file system on those respective browsers, falling back to Java applets where that's not possible. IE, since version 5, also supports the User Data API.

**Disadvantages**

The data capacity is extremely limited, as low as 20 cookies limited to 4KB each according to specification. Cookies slow down network activity because they are transferred to and from the server inside HTTP headers.

The main downside of plugin based storage is the obvious one: You have to assume the plugin is present, and with the right version. It's not uncommon for company firewalls to block plugins and for plugin releases to lag behind, or not exist at all, on certain operating systems. You also have to have faith in the plugin manufacturer keeping up plugin quality and market share going forth, versus the easy substitutability of open-source, open-standards based, browsers.

Browser-specific features have problem with these is significant: You're requiring the user to be working with a particular browser and you're putting faith in the browser manufacturer going forward. This is different from using features only present in just one or two browsers so far, where the feature is part of an open standards process and other manufacturers are likely to include it in the future.

These are all overcome by HTML5 offline storage

# Why use Offline Storage?

There are several reasons to use offline storage. First, you can make your app work when the user is offline, possibly sync'ing data back once the network is connected again. Second, it's a performance booster; you can show a large corpus of data as soon as the user clicks on to your site, instead of waiting for it to download again. Third, it's an easier programming model, with no server infrastructure required. Of course, the data is more vulnerable and the user can't access it from multiple clients, so you should only use it for non-critical data, in particular cached versions of data that's also "in the cloud". Advantages of offline storage

* Significantly speed up display times
* Cache data from RPC calls
* Load cached data on startup (faster startup)
* Save temporary state
* Restore state upon app reentry
* Prevent work loss from network disconnects
* Unlike cookies, items in Storage are not sent along in requests, which helps reduce network traffic.

**Browser Support**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Storage Types | Chrome | Firebox | Safari | Opera | IE | iOS | Android |
| Web Storage- named/value pairs | 4+ | 3.5+ | 4+ | 10.5+ | 8+ | 3.2+ | 2.1+ |
| IndexedDB | 23+ | 10+ | - | 15+ | - | - | 4.4+ |
| Web SQL | 4+ | - | 3.1+ | 10.5+ | - | 3.2+ | 2.1+ |

# Offline Storage Types

An offline HTML5 store generally classified into three major types

* Web Storage
* WebSQL
* IndexedDB

These are all explained following topics.

**Web Storage**

HTML5 Web Storage defines two types of key-value storage types:

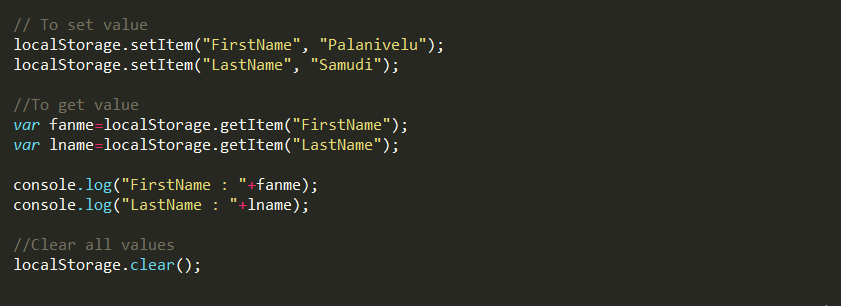
1. LocalStorage.
2. SessionStorage

The primary behavioral difference is how long the values persist and how they are shared.

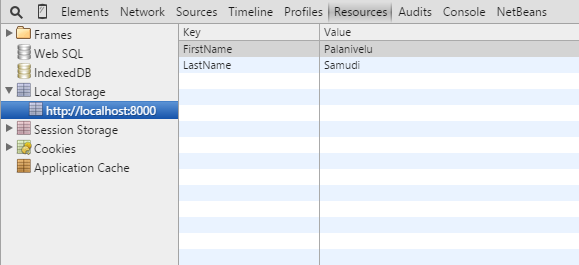
**LocalStorage**

The Local Storage is designed for storage that spans multiple windows, and lasts beyond the current session. In particular, Web applications may wish to store megabytes of user data, such as entire user-authored documents or a user's mailbox, on the client side for performance reasons. Cookies do not handle this case well, because they are transmitted with every request

**Syntax**

****

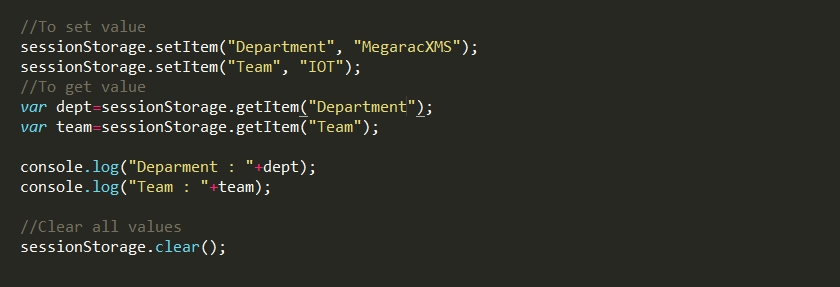
**Output**



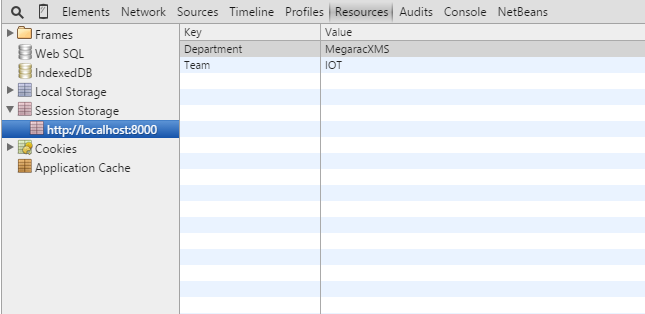
**Session Storage**

SessionStorage attribute which would be used by the sites to add data to the session storage, and it will be accessible to any page from the same site opened in that window i.e., session and as soon as you close the window, session would be lost.

**Syntax**

****

**Output**



**Difference between Local Storage and Session Storage**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Storage Type | Size | Persistence | Availability | Data Type |
| LocalStorage | 5MB per app per browser. According to the HTML5 spec, this limit can be increased by the user when needed; however, only a few browsers support this | On disk until deleted by user (delete cache) or by the app | Shared across every window and tab of one browser running same web app | String only, as key-value pairs |
| SessionStorage | Limited only by system memory | Survives only as long as its originating window or tab | Accessible only within the window or tab that created it | String only, as key-value pairs |

**Advantages**

* Supported on all modern browsers, as well as on iOS and Android, for several years (IE since version 8).
* Simple API signature.
* Simple call flow, being a synchronous API.
* Semantic events available to keep other tabs/windows in sync.

**Disadvantages**

* Poor performance for large/complex data, when using the synchronous API (which is the most widely supported mode).
* Poor performance when searching large/complex data, due to lack of indexing. (Search operations have to manually iterate through all items.)
* Poor performance when storing and retrieving large/complex data structures, because it's necessary to manually serialize and de-serialize to/from string values. The major browser implementations only support string values (even though the spec says otherwise).
* Need to ensure data consistency and integrity, since data is effectively unstructured.

# WebSQL

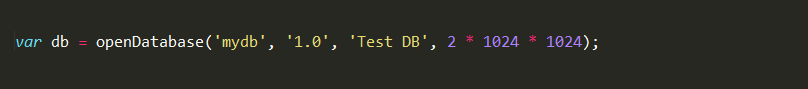
**WebSQL Database** is a relational database solution for browsers. Even though it is essentially deprecated and the W3C no longer maintains the specification, meaning browsers may or may not continue to support it, people still use it because it is the only common answer for structured data storage on mobile browsers.

There are three core methods in WebSQL

* openDatabase
* transaction
* executeSql

OpenDatabase method creates the database object either using existing database or creating new one. Transaction method give us the ability to control a transaction and performing either commit or roll-back based on the situation. Execute method is used to execute actual SQL query.

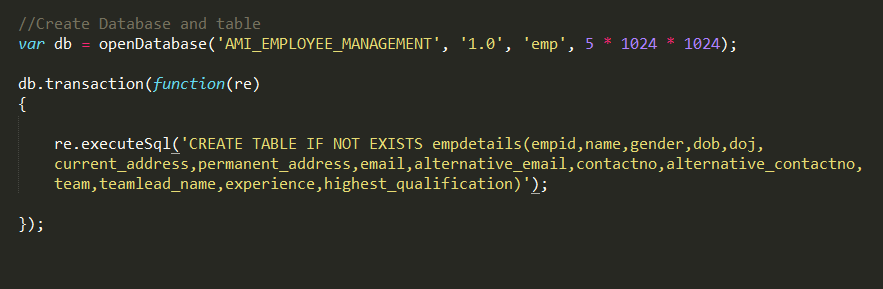
**Syntax**

****

Where**mydb**represent Database name, **1.0** represent version number, **TestDB**represent Text description,**2 \* 1024 \* 1024**representsize of database and we can add callback function as fifth parameter.

Let consider real-time example for Employee management system, which help to get all employee details. The following queries are explained basic CRUD operations.

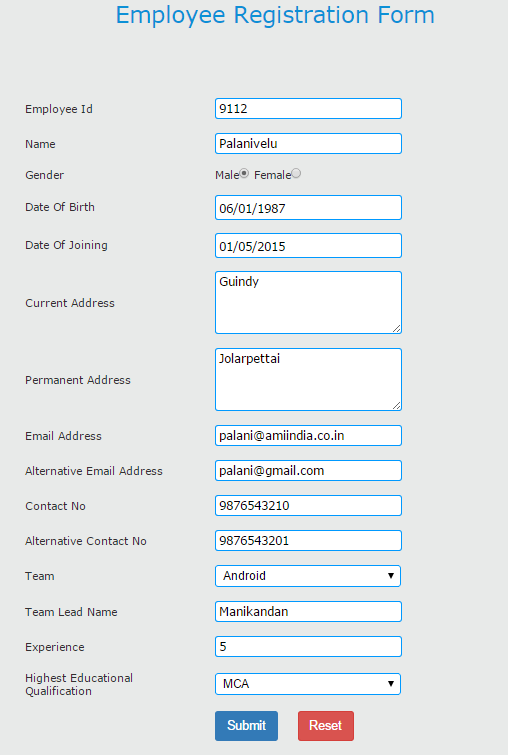
**Create**



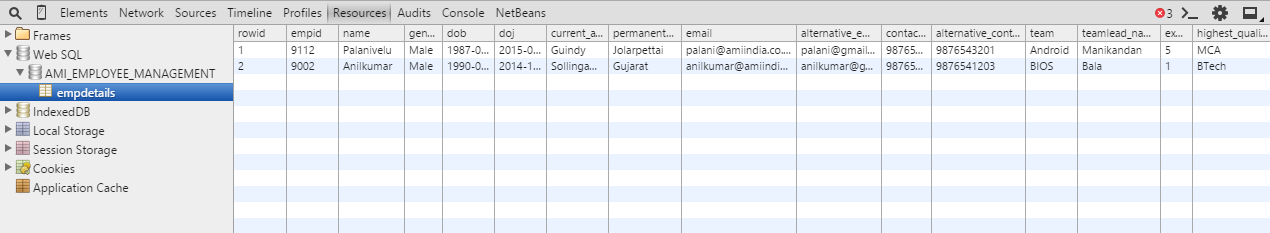
**Insert**

****

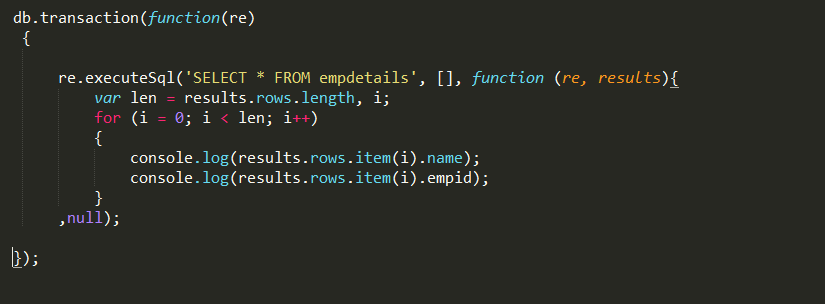
**HTML input form**

******

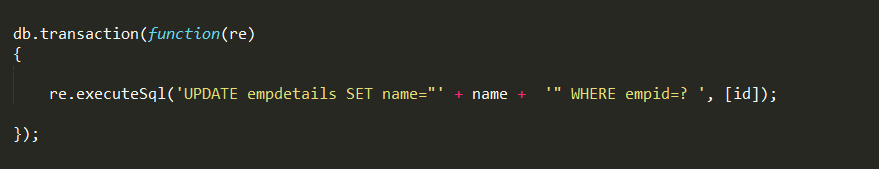
**Output**



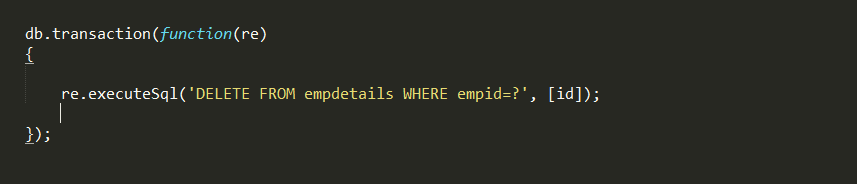
**Select**

****

**Update**

**

**Delete**

**

**Advantages**

* Supported on major mobile browsers (Android Browser, Mobile Safari, Opera Mobile) as well as several desktop browsers (Chrome, Safari, Opera).
* Good performance generally, being an asynchronous API. Database interaction won't lock up the user interface. (Synchronous API is also available for WebWorkers.)
* Good search performance, since data can be indexed according to search keys.
* Robust, since it supports a transactional database model.
* Easier to maintain integrity of data, due to rigid data structure.

**Disadvantages**

* Deprecated. Will not be supported on IE or Firefox, and will probably be phased out from the other browsers at some stage.
* Steep learning curve, requiring knowledge of relational databases and SQL.
* Suffers from object-relational impedance mismatch.
* Diminishes agility, as database schema must be defined upfront, with all records in a table matching the same structure.

# IndexedDB

Indexed Database is somewhere in between Web Storage and Web SQL Database. Like Web Storage, it's a straightforward key-value mapping, but it supports indexes like those of relational databases, so searching objects matching a particular field is fast; you don't have to manually iterate through every object in the store

The W3C has announced that the Web SQL database is a deprecated local storage specification so web developer should not use this technology any more. indexeddb is an alternative for web SQL data base and more effective than older technologies.

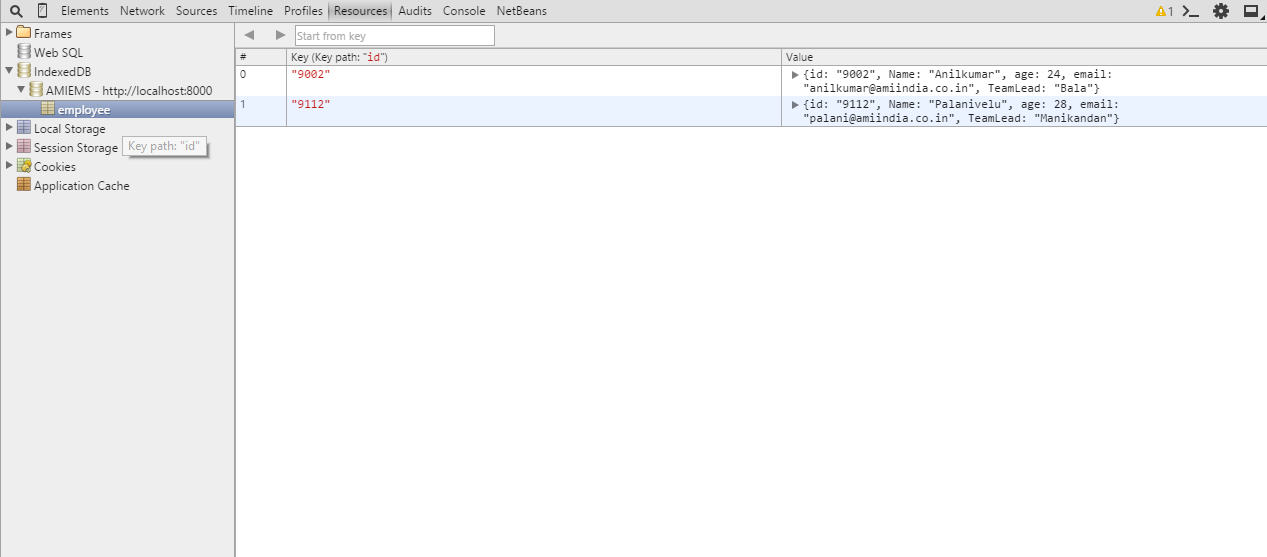
**Syntax**

Setup takes some work, because it enforces a database version system. When we make a connection to our database we specify which version we're expecting, if the current database uses a previous version, or hasn't been created yet, the onupgradeneeded event is fired, and onsuccess is called once the upgrade is complete. If no upgrade is needed onsuccess is called straight away.

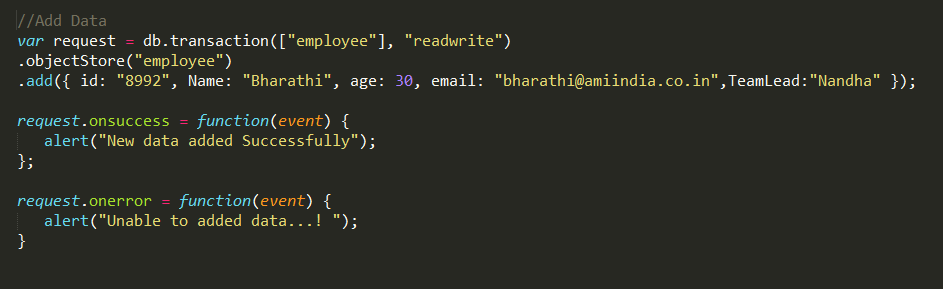
**Create**

****

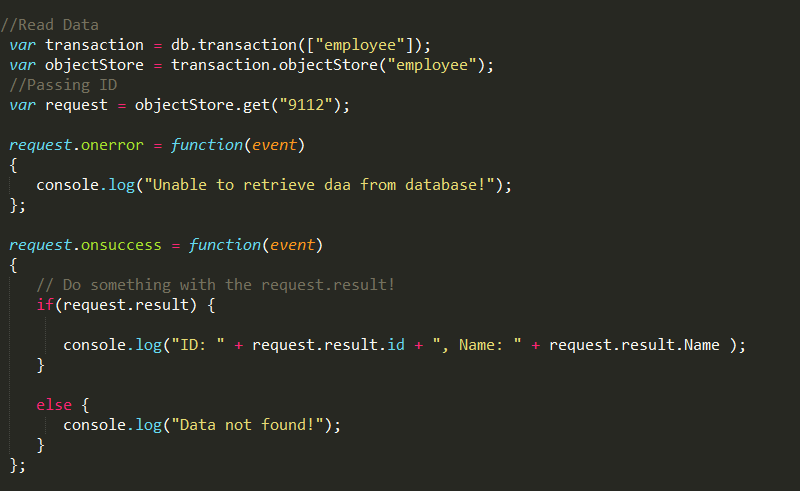
**Output**

****

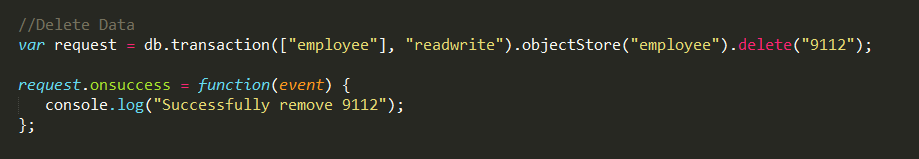
**Insert**

****

**Select**

****

**Delete**

****

**Advantages**

* Good performance generally, being an asynchronous API. Database interaction won't lock up the user interface. (Synchronous API is also available for WebWorkers.)
* Good search performance, since data can be indexed according to search keys.
* Supports versioning.
* Robust, since it supports a transactional database model.
* Fairly easy learning curve, due to a simple data model.
* Decent browser support: Chrome, Firefox, mobile FF, IE10.

**Disadvantages**

* Very complex API resulting in large amounts of nested callbacks.

**Difference between WebSQL and IndexedDB**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Storage Type | Location | Query | Transaction | Commits |
| WebSQL | Tables contain columns and rows | SQL | Lock can happen on databases, tables, or rows on 'readwrite' transactions | Transaction creation is explicit. Default is to rollback unless we call commit. |
| IndexedDB | objectStore contains Javascript objects and keys | [Cursor APIs](http://www.w3.org/TR/IndexedDB/#cursor-concept), [Key Range APIs](http://www.w3.org/TR/IndexedDB/#range-concept), and Application Code | Lock can happen on database 'versionchange' transaction, on an objectStore 'readonly' and 'readwrite' transactions. | Transaction creation is explicit. Default is to commit unless we call abort or there is an error that is not caught. |

# Conclusion

Offline Storage is a really great feature that makes it possible for web apps and games to work offline.

* [Web Storage](http://dev.w3.org/html5/webstorage/) simply provides a key-value mapping, e.g. localStorage.name = username;. Unfortunately, present implementations only support string-to-string mappings, so you need to serialise and de-serialise other data structures. You can do so using JSON.stringify() and JSON.parse().
* [Web SQL Database](http://dev.w3.org/html5/webdatabase/) gives you all the power - and effort - of a structured SQL relational database.
* [Indexed Database](http://www.w3.org/TR/IndexedDB/) is somewhere in between Web Storage and Web SQL Database. Like Web Storage, it's a straightforward key-value mapping, but it supports indexes like those of relational databases, so searching objects matching a particular field is fast; you don't have to manually iterate through every object in the store.