FREE eBook

LEARNING indexeddb

Free unaffiliated eBook created from
Stack Overflow contributors.

#indexeddb

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: indexeddb

It is an unofficial and free indexeddb ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official indexeddb.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with indexeddb

Remarks

This section provides an overview of what indexeddb is, and why a developer might want to use it.

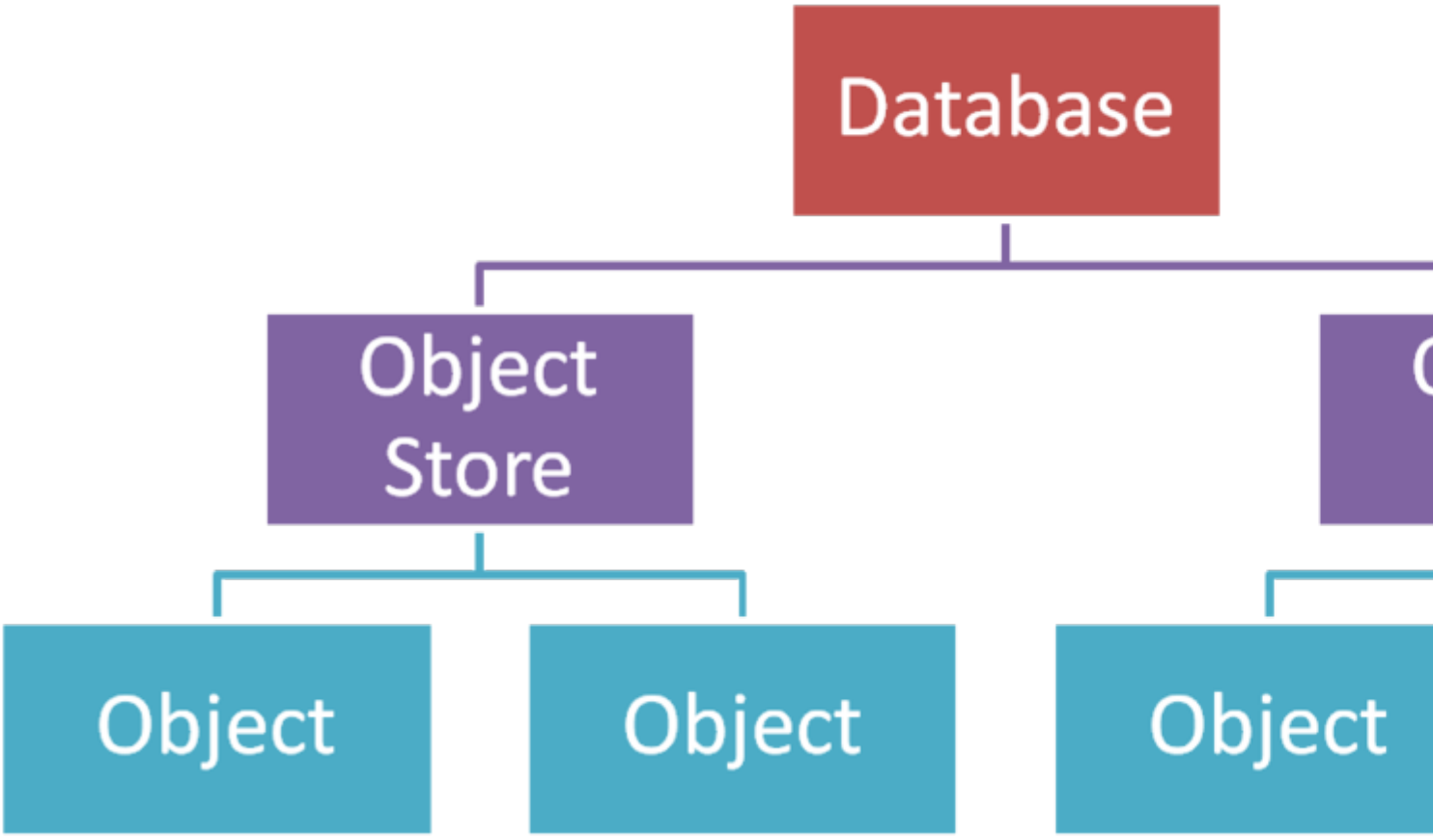
It should also mention any large subjects within indexeddb, and link out to the related topics. Since the Documentation for indexeddb is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Detailed instructions on getting indexeddb set up or installed.

Indexed DB Schema



As it can be seen from the picture above, on a single application we can create:

- Multiple databases
- Each database can have multiple object stores (tables)

- Each object store can have stored multiple objects

Overview

IndexedDB is a low-level API for client-side storage of significant amounts of structured data, including files/blobs. This API uses indexes to enable high-performance searches of this data. While Web Storage is useful for storing smaller amounts of data, it is less useful for storing larger amounts of structured data. The IndexedDB standard was created to enable scalable, performant storage and retrieval of Javascript objects in a browser.

Basics

indexedDB is designed to store Javascript object literals such as `{prop1 : value, prop2 : value}`. In addition, more recent implementations support storing large binary objects (BLOBs), such as images, audio files, and video files. In addition, indexedDB can store objects that contain other objects (nested objects), such as `{prop1 : value, prop2 : {nestedprop1 : value, nestedprop2 : value}}`.

The following are some basic concepts:

- **Database**: a container of object stores and indices. Every database has a name and a version.
- **Object store**: a container of objects. This is analogous to a table in a relational database. In indexedDB, records correspond to Javascript objects, and columns correspond to Javascript object properties. Objects added to the store are stored in the order added. Queries against the store retrieve objects in the same order. You can insert, update, or delete objects in an object store.
- **Index**: a special container for specific objects contained within an object store. Indices are also analogous to tables, and can be understood as object stores with special constraints. When an object is inserted into an object store, it may, if it meets certain criteria, also be inserted into a corresponding index store. Objects in an index are stored in an order defined by the index. Queries against an index retrieve objects in the order defined by the index (although queries can be configured to work differently). You cannot insert, update, or delete objects in an index (you can only do so indirectly by inserting the object into the store upon which the index is based).
- **Cursor**: cursors are analogous to queries. A cursor iterates over the objects in either an object store or an index. Cursors can move forwards or backwards, seek (jump or advance past objects), and jump to the next or previous 'unique' object in the underlying store/index.
- **Key path**: key paths are analogous to primary keys (or compound primary keys) of a table in a relational database. In the general case, when you instruct indexedDB to create an object store in a particular database, you also define the key path for the store. You can use the key path to quickly get a particular object, which is similar to using a primary key to select a record in a relational table. You can, optionally, use keys to ensure that later attempts to insert an object into an object store that already contains an object with the same key will produce an error.
- **Transactions and requests**: requests are analogous to individual SQL queries. There are

specific API methods for inserting an object, deleting an object, updating an object, and iterating over one or more objects. Each method call corresponds to a single request. Each request occurs within the context of a transaction. In other words, multiple requests can occur in one transaction. Individual requests can fail for a variety of reasons. When performing multiple requests in a single transaction, the requests are not fully committed until all the requests are considered successful. In this manner, if a problem occurs in a later request, the entire transaction can be "rolled back" so that the state of the underlying object store is the same as it was before the occurrence of the first request in the transaction.

Async vs Sync

indexedDB's Javascript API uses asynchronous techniques. When directly interacting with the API, and not some higher level third party library, the API requires the use of Javascript callbacks. The asynchronous design helps prevent larger data processing operations from blocking the main Javascript thread, which helps prevent the user interface (what you see in the browser) from appearing frozen/jerky/laggy.

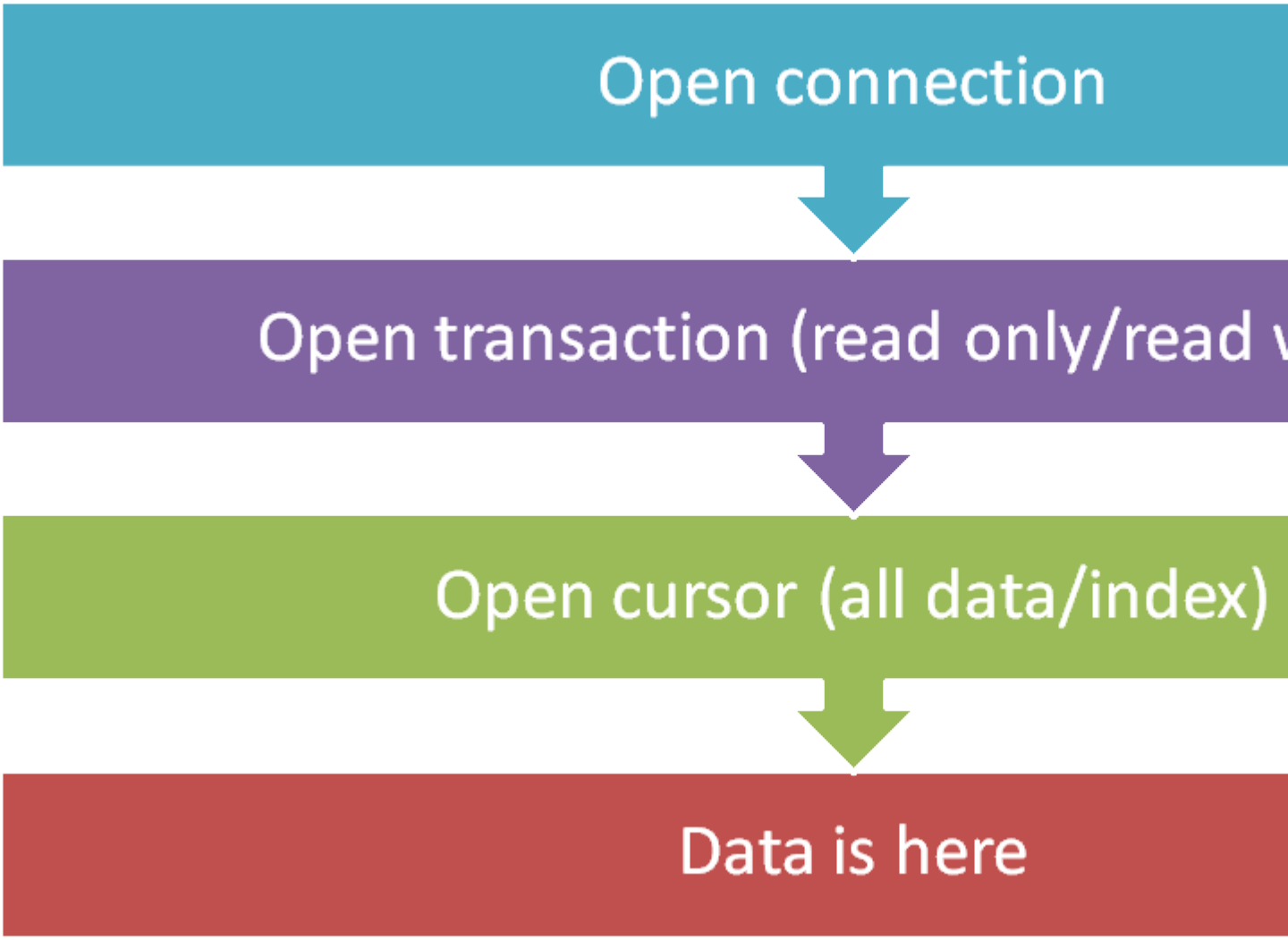
Support

Visit http://caniuse.com/#feat=indexeddb.

Learn More

- *Using IndexedDB* article on MDN
- W3 spec
- Example on html5rocks.com. (**Warning:** this example is outdated, for instance it uses `setVersion` instead of `onupgradeneeded`, and thus it might not run in modern browsers.)

Indexed DB Request overview



As it can be seen on the picture above, in indexed db in order to access the data you need to have:

1. Open a connection to the desired database
2. Open a transaction which can be read only or read write
3. Open a cursor or index which can be used for filtering the data
4. In the cursor request - onsuccess event you can access your data

Read Getting started with indexeddb online: https://riptutorial.com/indexeddb/topic/10356/getting-started-with-indexeddb

Chapter 2: Indexed DB examples

Examples

Open a database

```
function openDatabase(dbName) {
  var request = indexedDB.open(dbName);
  request.onsuccess = function (e) {
    var database = request.result;
    if (database) {
      console.log("Database initialized.");
    } else {
      console.error("Database is not initialized!");
    }

    //Your code goes here
    database.close();
  }
  request.onerror = function (e) {
    console.error( e.target.error.message);
  }
}
```

Initialize database - create table - with known db version

In order to trigger a "upgradeneeded" event you need to request the database with version higher than the current version - otherwise the event won't be triggered.

```
function createTable(dbName, dbversion, tableName) {
  var request = indexedDB.open(dbName, dbversion);
  request.onupgradeneeded = function (e) {
    var database = e.target.result;
    var objectStore = database.createObjectStore(tableName, {
        keyPath: 'id'
    });
    console.log("Object Store Created");
  };
  request.onsuccess = function (e) {
    var database = e.target.result;

    //code to verify that the table was created
    database.objectStoreNames.contains(storeName);

    database.close();
  }
  request.onerror = function (e) {
    console.error(e.target.error.message);
  }
}
```

Initialize database - create table without knowing db version

This is a more generic solution applicable in system where the user has option to add indexes to the table that he uses:

```
function createTable(dbName, tableName) {
  var request = indexedDB.open(dbName);
  request.onsuccess = function (e){
    var database = e.target.result;
    var version =  parseInt(database.version);
    database.close();
    var secondRequest = indexedDB.open(dbName, version+1);
    secondRequest.onupgradeneeded = function (e) {
        var database = e.target.result;
        var objectStore = database.createObjectStore(storeName, {
            keyPath: 'id'
        });
    };
    secondRequest.onsuccess = function (e) {
        e.target.result.close();
    }
  }
}
```

Read Indexed DB examples online: https://riptutorial.com/indexeddb/topic/10446/indexed-db-examples

Credits

S. No	Chapters	Contributors
1	Getting started with indexeddb	Community, Deni Spasovski
2	Indexed DB examples	Deni Spasovski