**Big concepts**

If you have assumptions from working with other types of databases, you might get thrown off when working with IndexedDB. So keep the following important concepts in mind:

* **IndexedDB databases store key-value pairs.** The values can be complex structured objects, and keys can be properties of those objects. You can create indexes that use any property of the objects for quick searching, as well as sorted enumeration. Keys can be binary objects.
* **IndexedDB is built on a transactional database model**. Everything you do in IndexedDB always happens in the context of a [transaction](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Basic_Concepts_Behind_IndexedDB#gloss_transaction). The IndexedDB API provides lots of objects that represent indexes, tables, cursors, and so on, but each of these is tied to a particular transaction. Thus, you cannot execute commands or open cursors outside of a transaction. Transactions have a well-defined lifetime, so attempting to use a transaction after it has completed throws exceptions. Also, transactions auto-commit and cannot be committed manually.

This transaction model is really useful when you consider what might happen if a user opened two instances of your web app in two different tabs simultaneously. Without transactional operations, the two instances could interfere with each other's modifications. If you are not familiar with transactions in a database, read the [Wikipedia article on transactions](https://en.wikipedia.org/wiki/Database_transaction). Also see [transaction](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Basic_Concepts_Behind_IndexedDB#gloss_transaction) under the Definitions section.

* **The IndexedDB API is mostly asynchronous.** The API doesn't give you data by returning values; instead, you have to pass a callback function. You don't "store" a value into the database, or "retrieve" a value out of the database through synchronous means. Instead, you "request" that a database operation happens. You get notified by a DOM event when the operation finishes, and the type of event you get lets you know if the operation succeeded or failed. This sounds a little complicated at first, but there are sanity measures baked in. It's not that different from the way that [XMLHttpRequest](https://developer.mozilla.org/en/DOM/XMLHttpRequest) works.
* **IndexedDB uses a lot of requests.**Requests are objects that receive the success or failure DOM events that were mentioned previously. They have onsuccess and onerror properties, and you can call addEventListener() and removeEventListener() on them. They also have readyState, result, and errorCode properties that tell you the status of the request. The result property is particularly magical, as it can be many different things, depending on how the request was generated (for example, an IDBCursor instance, or the key for a value that you just inserted into the database).
* **IndexedDB uses DOM events to notify you when results are available.** DOM events always have a type property (in IndexedDB, it is most commonly set to "success" or "error"). DOM events also have a target property that indicates where the event is headed. In most cases, the target of an event is the IDBRequest object that was generated as a result of doing some database operation. Success events don't bubble up and they can't be canceled. Error events, on the other hand, do bubble, and can be cancelled. This is quite important, as error events abort whatever transactions they're running in, unless they are cancelled.
* **IndexedDB is object-oriented.** IndexedDB is not a relational database with tables representing collections of rows and columns. This important and fundamental difference affects the way you design and build your applications.

In a traditional relational data store, you would have a table that stores a collection of rows of data and columns of named types of data. IndexedDB, on the other hand, requires you to create an object store for a type of data and simply persist JavaScript objects to that store. Each object store can have a collection of indexes that makes it efficient to query and iterate across. If you are not familiar with object-oriented database management systems, read the [Wikipedia article on object database](https://en.wikipedia.org/wiki/Object_database).

* **IndexedDB does not use Structured Query Language (SQL).** It uses queries on an index that produces a cursor, which you use to iterate across the result set. If you are not familiar with NoSQL systems, read the [Wikipedia article on NoSQL](https://en.wikipedia.org/wiki/NoSQL).
* **IndexedDB adheres to a same-origin policy**. An origin is the domain, application layer protocol, and port of a URL of the document where the script is being executed. Each origin has its own associated set of databases. Every database has a name that identifies it within an origin.