Data model

Firestore is a NoSQL, document-oriented database. Unlike a SQL database, there are no tables or rows. Instead, you store data in *documents*, which are organized into *collections*.

Each *document* contains a set of key-value pairs. Firestore is optimized for storing large collections of small documents.

All documents must be stored in collections. Documents can contain *subcollections* and nested objects, both of which can include primitive fields like strings or complex objects like lists.

Collections and documents are created implicitly in Firestore. Simply assign data to a document within a collection. If either the collection or document does not exist, Firestore creates it.

Documents

In Firestore, the unit of storage is the document. A document is a lightweight record that contains fields, which map to values. Each document is identified by a name.

A document representing a user `alovelace` might look like this:

alovelace

```
first : "Ada"
last : "Lovelace"
born : 1815
```

Note: Firestore supports a variety of data types for values: boolean, number, string, geo point, binary blob, and timestamp. You can also use arrays or nested objects, called maps, to structure data within a document.

Complex, nested objects in a document are called maps. For example, you could structure the user's name from the example above with a map, like this:

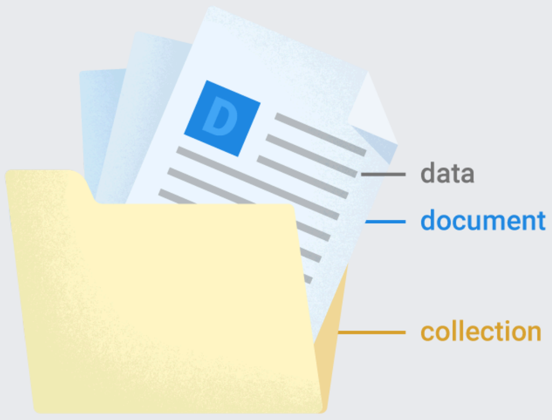
alovelace

```
name :
  first : "Ada"
  last : "Lovelace"
born : 1815
```

You may notice that documents look a lot like JSON. In fact, they basically are. There are some differences (for example, documents support extra data types and are limited in size to 1 MB), but in general, you can treat documents as lightweight JSON records.

Collections

Documents live in collections, which are simply containers for documents. For example, you could have a `users` collection to contain your various users, each represented by a document:



users

alovelace

```
first : "Ada"
last : "Lovelace"
born : 1815
```

aturing

```
first : "Alan"
last : "Turing"
born : 1912
```

Firestore is schemaless, so you have complete freedom over what fields you put in each document and what data types you store in those fields. Documents within the same collection can all contain different fields or store different types of data in those fields. However, it's a good idea to use the same fields and data types across multiple documents, so that you can query the documents more easily.

A collection contains documents and nothing else. It can't directly contain raw fields with values, and it can't contain other collections. (See Hierarchical Data (#hierarchical-data) for an explanation of how to structure more complex data in Firestore.)

The names of documents within a collection are unique. You can provide your own keys, such as user IDs, or you can let Firestore create random IDs for you automatically.

You do not need to "create" or "delete" collections. After you create the first document in a collection, the collection exists. If you delete all of the documents in a collection, it no longer exists.

References

Every document in Firestore is uniquely identified by its location within the database. The previous example showed a document `alovelace` within the collection `users`. To refer to this location in your code, you can create a *reference* to it.

Web version 9 (modular) Web version 8 (namespaced) Swift (#swift) Objective-C (#objective-c) Kotlin+KTX Android

Learn more (//firebase.google.com/docs/web/learn-more#modular-version) about the tree-shakeable modular Web API and upgrade (//firebase.google.com/docs/web/modular-upgrade) from the namespaced API.

```
import { doc } from "firebase/firestore";

const alovelaceDocumentRef = doc(db, 'users', 'alovelace');
```
f02285779974620ad3ff260f5ac/snippets/firestore-next/test-firestore/doc_reference.js#L8-L10)

A reference is a lightweight object that just points to a location in your database. You can create a reference whether or not data exists there, and creating a reference does not perform any network operations.

You can also create references to *collections*:

Web version 9 (modular) Web version 8 (namespaced) Swift (#swift) Objective-C (#objective-c) Kotlin+KTX Android

Learn more (//firebase.google.com/docs/web/learn-more#modular-version) about the tree-shakeable modular Web API and upgrade (//firebase.google.com/docs/web/modular-upgrade) from the namespaced API.

```
import { collection } from "firebase/firestore";

const usersCollectionRef = collection(db, 'users');
```
5779974620ad3ff260f5ac/snippets/firestore-next/test-firestore/collection_reference.js#L8-L10)

Note: *Collection references* and *document references* are two distinct types of references and let you perform different operations. For example, you could use a collection reference for querying the documents in the collection, and you could use a document reference to read or write an individual document.

For convenience, you can also create references by specifying the path to a document or collection as a string, with path components separated by a forward slash (`/`). For example, to create a reference to the `alovelace` document:

Web version 9 (modular) Web version 8 (namespaced) Swift (#swift) Objective-C (#objective-c) Kotlin+KTX Android

Learn more (//firebase.google.com/docs/web/learn-more#modular-version) about the tree-shakeable modular Web API and upgrade (//firebase.google.com/docs/web/modular-upgrade) from the namespaced API.

```
import { doc } from "firebase/firestore";

const alovelaceDocumentRef = doc(db, 'users/alovelace');
```
)74620ad3ff260f5ac/snippets/firestore-next/test-firestore/doc_reference_alternative.js#L8-L10)

Hierarchical Data

To understand how hierarchical data structures work in Firestore, consider an example chat app with messages and chat rooms.

You can create a collection called `rooms` to store different chat rooms:

rooms

roomA

`name : "my chat room"`

roomB

`...`

Now that you have chat rooms, decide how to store your messages. You might not want to store them in the chat room's document. Documents in Firestore should be lightweight, and a chat room could contain a large number of messages. However, you can create additional collections within your chat room's document, as subcollections.

Subcollections

The best way to store messages in this scenario is by using subcollections. A subcollection is a collection associated with a specific document.

Note: You can query across subcollections with the same collection ID by using Collection Group Queries (/firestore/docs/query-data/queries#collection-group-query).

You can create a subcollection called `messages` for every room document in your `rooms` collection:

- rooms
  - roomA
    `name : "my chat room"`
    - messages
      - message1
        `from : "alex"`
        `msg : "Hello World!"`
      - message2
        `...`
  - roomB
    `...`

In this example, you would create a reference to a message in the subcollection with the following code:

Web version 9 (modular) | Web version 8 (namespaced) | Swift (#swift) | Objective-C (#objective-c) | Kotlin+KTX Android

Learn more (//firebase.google.com/docs/web/learn-more#modular-version) about the tree-shakeable modular Web API and upgrade (//firebase.google.com/docs/web/modular-upgrade) from the namespaced API.

```
import { doc } from "firebase/firestore";

const messageRef = doc(db, "rooms", "roomA", "messages", "message1");
```
9974620ad3ff260f5ac/snippets/firestore-next/test-firestore/subcollection_reference.js#L8-L10)

Notice the alternating pattern of collections and documents. Your collections and documents must always follow this pattern. You cannot reference a collection in a collection or a document in a document.

Subcollections allow you to structure data hierarchically, making data easier to access. To get all messages in `roomA`, you can create a collection reference to the subcollection `messages` and interact with it like you would any other collection reference.

Documents in subcollections can contain subcollections as well, allowing you to further nest data. You can nest data up to 100 levels deep.

Warning: Deleting a document does not delete its subcollections!

When you delete a document that has subcollections, those subcollections are not deleted. For example, there may be a document located at `coll/doc/subcoll/subdoc` even though the document `coll/doc` no longer exists. If you want to delete documents in subcollections when deleting a parent document, you must do so manually, as shown in Delete Collections (/firestore/docs/manage-data/delete-data#collections).

Last updated 2024-04-01 UTC.