Cloud DataBase

Firebase Cloud Firestore

Why Cloud Data Stores?

- Highly scalable
- Usually built using No SQL technology
- Accessible to both **web** and **mobile** clients

No SQL = No DB Schema

3

SQL vs. no SQL

SQL

- Relational model
- **Schema**: relationship between tables and fields
- Popular examples
  - Oracle
  - DB2
  - MySQL
  - PostGreSQL

no SQL

- Non-relational
- **Schemaless Datastore**
- Cloud Computing and Cloud Storage
- Rapid Development
- Popular examples
  - MongoDB
  - CouchDB
  - BigTable
  - Firebase Realtime DB
  - **Firebase Cloud Firestore**

4

Schema or Schemaless?

First	Last	G#	Major
Alice	Smith	12345678	Statistics
Brad	Jordan	23456789	History

Must redefine the SCHEMA to add a new column.

First	Last	G#	Major	MusGenre	FavClr	SocMedia	???
Alice	Smith	12345678	Statistics		Green		
Brad	Jordan	23456789	History			IG, FB	
Gary	deGroot	72551834	Biology			TW	
Ann	Hunt	78921631	Physics	Classical			
Fay	Ross	72631235	English		Blue	LinkedIn	

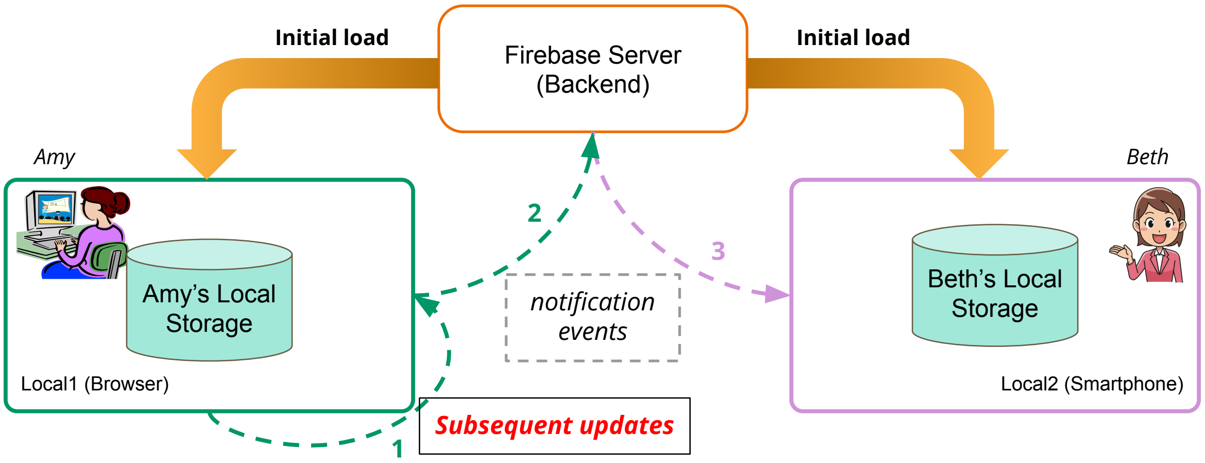
Firebase

A collection of many products

- **Cloud Firestore** (beta since 2017, GA since 2019)
- **Authentication**
- **Cloud Storage**
- **Realtime DB** (beta since 2012, GA since 2014?)
- Cloud Messaging
- ML Kit
- Cloud Functions

	Realtime DB	Cloud Firestore
Autogenerated Key	Time-based	Not time-based
Write operations	Max 1000 writes/second	Max 10,000 writes/second
Offline support	iOS and Android clients	iOS, Android, and Web clients
Concurrent Connections	Max 200,000	Max 1,000,000
Data Model	Giant JSON tree	Hierarchy of Collections (“Tables”) and Documents (“Records”)
Queries	Deep (***slower performance***), fetching a node will return the entire subtree of the node	Shallow (***better performance***), it is possible to fetch a document without its “children”
	Queries can use sorting **or** filtering (but not both)	Queries can use sorting **and** filtering

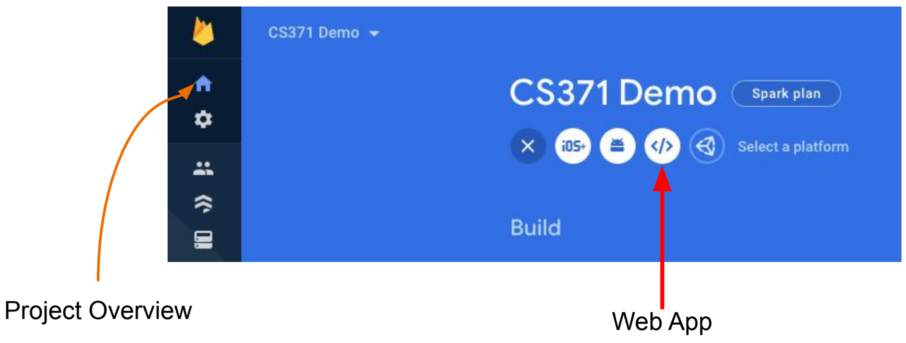
Local Storage, Local Events, & Global Events

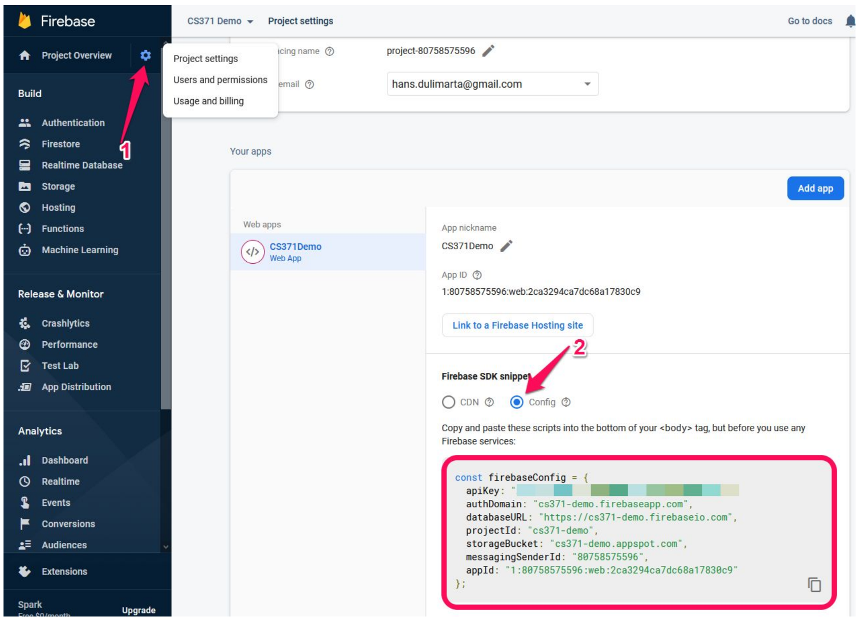


Demo 1: New Firebase Project & Web App

Creating a new WebApp

1. Use a *personal* Google account to login to Firebase Console
   a. GVSU Google Mail account does not work
2. Create a new project'
3. Add an app to the project





Project settings

icing name: project-80758575596

email: hans.dulimarta@gmail.com

Your apps

Web apps

CS371Demo
Web App

App nickname: CS371Demo

App ID: 1:80758575596:web:2ca3294ca7dc68a17830c9

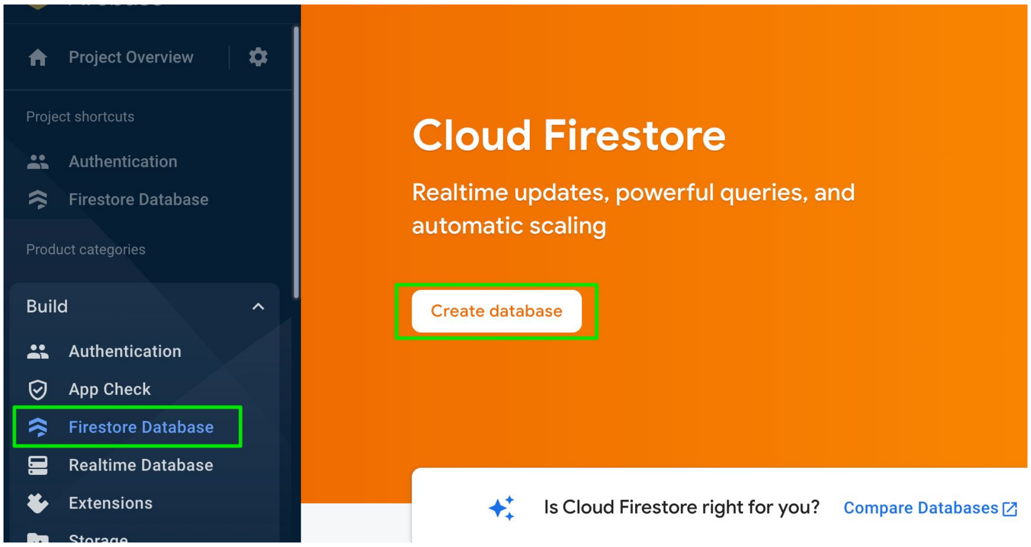
Firebase SDK snippet

CDN [✓ Config]

Copy and paste these scripts into the bottom of your <body> tag, but before you use any Firebase services:

```
const firebaseConfig = {
  apiKey: "[redacted]
  authDomain: "cs371-demo.firebaseapp.com",
  databaseURL: "https://cs371-demo.firebaseio.com",
  projectId: "cs371-demo",
  storageBucket: "cs371-demo.appspot.com",
  messagingSenderId: "80758575596",
  appId: "1:80758575596:web:2ca3294ca7dc68a17830c9"
};
```

Initialize Firestore



Cloud Firestore

Realtime updates, powerful queries, and automatic scaling

Is Cloud Firestore right for you?

Local Project Setup
(On Your Computer)

Project Setup & Initialization (Version 9.x)

```
yarn init -y

yarn add firebase        # version 9.x
```

```
npm init -y

npm install firebase     # version 9.x
```

```
import {initializeApp, FirebaseApp} from “firebase/app”
import {getFirestore, Firestore} from “firebase/firestore”
const config = { // COPY this from your Firebase Console
  apiKey: "your-api-key-goes-here",
  authDomain: "your-project-name-here.firebaseapp.com",
  databaseURL: "https://your-project-name-here.firebaseio.com",
  projectId: "your-project-name-here",
  storageBucket: "your-project-name.appspot.com",
  messagingSenderId: "xxxxxxxx"
};

const myapp: FirebaseApp = initializeApp(config);
const db:Firestore = getFirestore(myApp);
// OR  const db:Firestore = getFirestore();     // Called without myApp
```

Database Dashboard

- Browse and Modify Data
- Security Rules (default settings: user authentication required)
- Change read/write access to “true” during your initial experiment
  - `“.read”: “auth != null” ⇒   “.read”: true`
  - `“.write”: “auth != null” ⇒   “.write”: true`

Part 1: Cloud Firestore

Data Model: Hierarchy of Collections-Documents

- Hierarchical structure
  - The "root" holds one or more collections
  - A collection consists of one or more documents
  - A document is one or more key-value pairs
  - A value in a document may refer to a subcollection (1-to-many relationships)
- Data Types in a document
  - string, number, boolean, array, timestamp, map (kv-pairs), geolocation
  - Reference to a subcollection

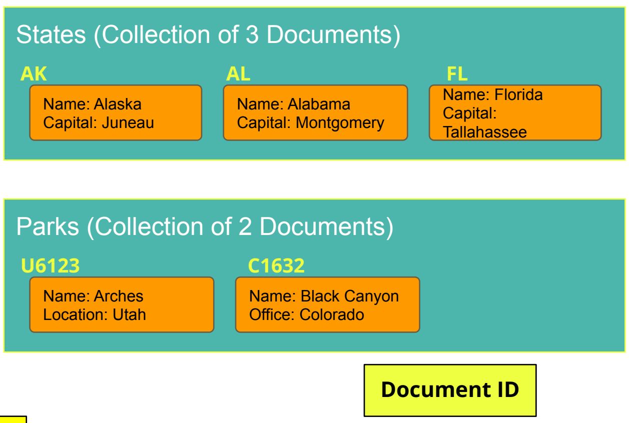
SQL	Cloud Firestore
Tables	Collections
Rows	Documents
Primary Key	Document ID
Fields	key-value pairs

State (SQL table)

Abbrev (PK)	Name	Capital
AK	Alaska	Juneau
AL	Alabama	Montgomery
FL	Florida	Tallahassee

NatlPark (SQL table)

Code (PK)	Name	Location
U6123	Arches	Utah
C1632	Black Canyon	Colorado



• SQL Table	⇒ Firestore Collection
• SQL Primary Key	⇒ Firestore Document **ID/"name"**
• SQL Table Row	⇒ Firestore Document

All Firestore Collection/Doc Manipulation Functions return a Promise

Firestore Functions (version 9.x)

- Functions for creating references
  - `collection(`*`refToFirestore`*`, “path/to/collection”)`
  - `doc(refToFirestoreOrCollection, “path/to/your/document”)`
  - `query(refToCollection, ____)`
- Retrieval functions
  - `getDoc(`*`refToDoc`*`)`
  - `getDocs(`*`refToCollection`*`)`
- Manipulation functions
  - `addDoc(refToColl, { new_content_object })`
  - `setDoc(refToDoc, { new_content_object })`
  - `updateDoc(refToDoc, { new_content_object })`
  - `deleteDoc(refToDoc)`
- Update listener `onSnapShot()` (**specific to Firebase**)

C
R
U
D

CRUD Operations (Summary)

	Collection	Document

Create	*Implied when a doc is created*	`// Option #1`
		`const collPar = collection(db, “cName”);`
		`addDoc(collPar, { /* new content here */ });`
		
		`// Option #2`
		`const myDoc = doc(db, “cName”, “docName”);`
		`setDoc(myDoc, { /* new content here */ })`

Read	`const myC = collection(db,“cName”)`
	`getDocs(myC).then(___);`
		`const myDoc = doc(____, ____, ___);`
		`getDoc(myDoc).then(___);`

Update	N/A	`const myDoc = doc(____, ____, ___);`
		`updateDoc(myDoc, {/* content */}).then(___);`

Delete	N/A. Use `getDocs()` together with `deleteDoc()`	`const myDoc = doc(____, ____, ___);`
		`deleteDoc(myDoc).then(___);`

SQL vs Firebase Firestore

SQL	Firestore 8.x	Firestore 9.x

	`const myColl = db.collection(“xyz”)`
	`const myDoc = db.doc(“xyz/def”)`
	`const myDoc = db.collection(“xyz”)`
	`.doc(“def”)`
		`const myColl = collection(db, “xyz”)`
		`const myDoc = doc(db, “xyz”, “def”)`
		`const myDoc = doc(db, “xyz/def”)`

`SELECT * FROM myTable`	`myColl.`**`getDocs`**`().then(___)`	**`getDocs`**`(myCollection).then(___)`

`INSERT INTO myTable`	`const myData = { /* TS object */ }`
	`myColl.`**`addDoc`**`(myData).then(___)`
	`myDoc.`**`setDoc`**`(myData).then(___)`
		`const myData = { /* TS object */ }`
		`addDoc``(myColl, myData).then(___)`
		`setDoc``(myDoc, myData).then(___)`

`UPDATE myTable WHERE`	`myDoc.`**`updateDoc`**`(newData).then(___)`	**`updateDoc`**`(myDoc, newData).then(___)`
`DELETE from WHERE`	`myDoc.`**`deleteDoc`**`().then(___)`	**`deleteDoc`**`(myDoc).then(___)`

Live Demo #1:
firestore/src/create-funcs.ts

```
# Option #1: create a ref to doc and “primaryKey”
const aDoc = doc(dbRoot, “nameOfCol”, “primKey”);
setDoc(aDoc, { /* detailed contents here */ });

# Option #2: create a ref to doc with auto ID
const aColl = collection(dbRoot, “nameOfCol”);
addDoc(aColl, { /* detailed contents here */ });
```

StackBlitz Playground

https://stackblitz.com/edit/typescript-u91bjq

CRUD Operations: Create Doc (own Doc ID)

SQL

```
// Use “AK” as the primary key for the tuple
INSERT INTO states (abbrev, name, capital) VALUES(“AK”, “Alaska”, “Juneau”)
```

State (SQL table)

Abbrev (PK)	Name	Capital
AK	Alaska	Juneau
FL	Florida	Tallahassee
MI	Michigan	Lansing

Firestore in TS

```
import {DocumentReference, setDoc, doc} from
  “firebase/firestore”;

// Option #1: Use file name syntax for doc path
// Primary key “AK” becomes doc id
const doc1: DocumentReference = doc(db, ”states”, “AK”)
setDoc(doc1, { name: “Alaska”, capital: “Juneau”})
  .then(() => {
    console.log(“New doc added”);
  })
  .catch((err:any) => { /* your code here */ });
```

CRUD Operations: Create Doc (automatic Doc ID)

SQL

```
INSERT INTO states (name, capital) VALUES(“Alaska”, “Juneau”)
```

State (SQL table)

Name	Capital
Alaska	Juneau
Florida	Tallahassee
Michigan	Lansing

Firestore in TS

```
import {CollectionReference, addDoc, doc} from
  “firebase/firestore”;

// Option #1: Use file name syntax for doc path
// Primary key “AK” becomes doc id
const myColl: CollectionReference = collection(db, ”states”)
addDoc(myColl, { name: “Alaska”, capital: “Juneau”})
  .then(() => {
    console.log(“New doc added”);
  })
  .catch((err:any) => { /* your code here */ });
```

CRUD Operations: Create Docs from Array

Firebase TS

```
import {DocumentReference, setDoc, doc, collection, addDoc} from “firebase/firestore”;

const stateArr = [
  {abbrev: “CA”, name: “California”, capital: “Sacramento”}
  {abbrev: “CO”, name: “Colorado”, capital: “Denver”},
  // more data here]

// Option 1: Use state abbreviation as document ID
stateArr.forEach(async (st:any) => {
    const stateDoc = doc(db, “states”, z.abbrev); // Us Abbreviation as document ID
    await setDoc(stateDoc, {name: st.name, capital: st.capital});
  })

// Option 2: Let Firestore generates automatic
const myStateColl = collection(db, “states”)    // Do this outside .forEach
stateArr.forEach(async (st:any) => {
    await addDoc(myStateColl, {name: st.name, capital: st.capital});
  }
)
```

Live Demo #2:
firestore/src/read-funcs.ts

```
# Get details of a document
const aDoc = doc(dbRoot, “path/to/your/doc”);
getDoc(aDoc).then((snap:DocumentSnapshot) => {
  // doc details in snap.data()
});

# Get all documents in a collection
const aColl = collection(dbRoot, “path/to/your/collection”);
getDocs(aColl).then((snap:QuerySnapshot) => {
  for (let z in snap) {
    // doc details in z.data()
  }
});
```

CRUD Operations: Read All Documents

SQL

```
SELECT * FROM states
```

State (SQL table)

Abbrev (PK)	Name	Capital
FL	Florida	Tallahassee
MI	Michigan	Lansing

TS

```
import {CollectionReference, collection, QuerySnapshot,
    QueryDocumentSnapshot, getDocs} from “firebase/firestore”;

const myStateColl:CollectionReference  = collection(db, “states”);

getDocs(myStateColl).then(
  (qs: QuerySnapshot) => {
    qs.forEach((qd:QueryDocumentSnapshot) => {
      const stateData = qd.data() as StateType
      cost docId = qd.id
      // More code here to manipulate stateData
    })
  })
```

CRUD Operations: Read A Specific Document

SQL

```
// Select a tuple with a known primary key
SELECT * FROM states WHERE abbrev = “MI”
```

State (SQL table)

Abbrev (PK)	Name	Capital
FL	Florida	Tallahassee
MI	Michigan	Lansing

```
// Assume saved data has the
// following structure
type StateType = {
  // abbrev: string;
  name: string;
  capital: string;
}
```

TS

```
import {DocumentReference, doc, DocumentSnapshot,
    getDoc} from “firebase/firestore”;

// MI is a document ID
const myDoc:DocumentReference  = doc(db, “states/MI”);

getDoc(myDoc).then(
  (qd:DocumentSnapshot) => {
    if (qd.exists()) {
      const stateData = qd.data() as StateType
      // More code here to manipulate stateData
    }
  })
```

CRUD Operations: Fetch Document(s) Where...

SQL
```
// Select tuples satisfying some conditions (other than primary key)
SELECT * FROM states WHERE name = “Florida”
```

State (SQL table)

Name	Capital	Population
Florida	Tallahassee	26_222_943
Michigan	Lansing	8_432_911
California	Sacramento	39_123_612

```
// Assume saved data has the
// following structure
type StateType = {
  name: string;
  capital: string;
  population: number;
}
```

Firebase TS
```
import {Query, getdocs, collections, where} from
“firebase/firestore”;

const aboveTenMil:Query = query(
  collection(db, “states”),
  where(“name”, “==”, “Florida”))

getDocs(aboveTenMill).then(
  (qs:QuerySnapshot) => {
    qs.forEach((qd:QueryDocumentSnapshot) => {
      const stateData = qd.data() as StateType
     // More code here to manipulate stateData
    })
  })
```

CRUD Operations: Fetch Document(s) Where....

SQL
```
// Select tuples satisfying some conditions
SELECT * FROM states WHERE population > 10_000_000
```

State (SQL table)

Name	Capital	Population
Florida	Tallahassee	26_222_943
Michigan	Lansing	8_432_911
California	Sacramento	39_123_612

```
// Assume saved data has the
// following structure
type StateType = {
  name: string;
  capital: string;
  population: number;
}
```

Firebase TS
```
import {Query, getdocs, collections, where} from
“firebase/firestore”;

const aboveTenMil:Query = query(
  collection(db, “states”),
  where(“population”, “>”, 10_000_000))

getDocs(aboveTenMill).then(
  (qs:QuerySnapshot) => {
    qs.forEach((qd:QueryDocumentSnapshot) => {
      const stateData = qd.data() as StateType
     // More code here to manipulate stateData
    })
  })
```

CRUD Operations: Fetch Document(s) Where....

SQL

```
// Select tuples satisfying some conditions
SELECT * FROM states WHERE population > 10_000_000 AND population < 15_000_000
```

State (SQL table)

Name	Capital	Population
Florida	Tallahassee	26_222_943
Michigan	Lansing	8_432_911
California	Sacramento	39_123_612

```
// Assume saved data has the
// following structure
type StateType = {
  name: string;
  capital: string;
  population: number;
}
```

Firebase TS

```
import {Query, getdocs, collections, where} from
“firebase/firestore”;

const aboveTenMil:Query = query(
  collection(db, “states”),
  where(“population”, “>”, 10_000_000),
  where(“population”, “<”, 15_000_000))

getDocs(aboveTenMill).then(
  (qs:QuerySnapshot) => {
    qs.forEach((qd:QueryDocumentSnapshot) => {
      const stateData = qd.data() as StateType
     // More code here to manipulate stateData
    })
  })
```

Available Query Where Operators

Operator	Example	SQL Equivalent
<, <=, ==, >=, >	where(“population”, “>”, 20_000_000)	WHERE population > 20000000
!=	where(“name”, “!=”, “Andy”)	WHERE name != “Andy”
in	where(“city”, “in”, [“Ada”, “Flint”])	WHERE city == “Ada” OR city == “Flint”
not-in	where(“city”, “not-in”, [“Ada”, “Flint”])	WHERE city != “Ada” AND city != “Flint”

Operator	Example (courses must be an ARRAY)
array-contains	// Has this student taken MTH200? where(“courses”, “array-contains”, “MTH200”)
array-contains-any	// Has this student taken either MTH200 or STA215? where(“courses”, “array-contains-any”, [“MTH200”, “STA215”])

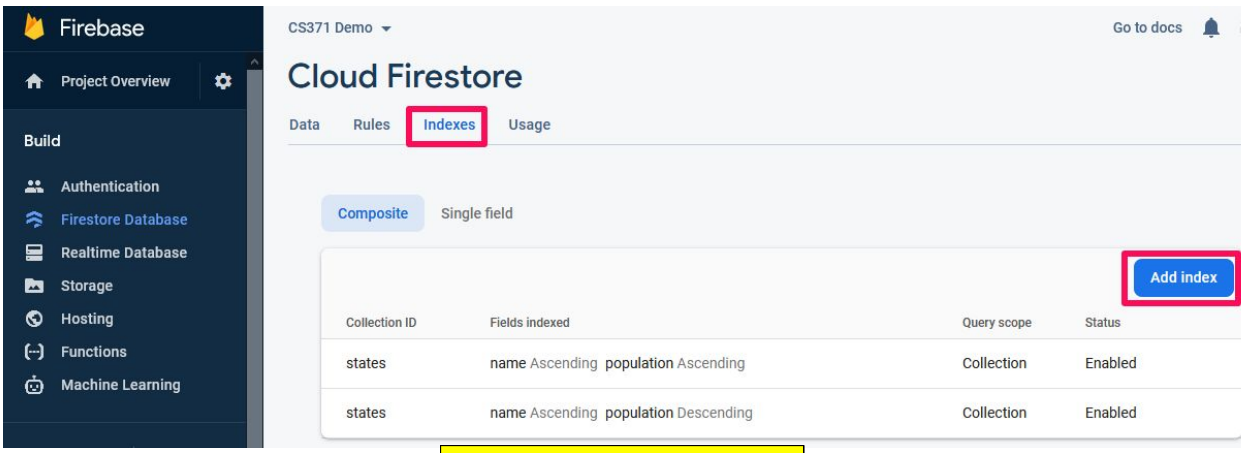
Query Limitations

```
// Multiple .where() on the same field
const q = query(collection(__, 'states'),
        where("population", ">=", 5_000_000),
        where("population", "<=", 10_000_000));
getDocs(q).then(() => { /* code */ });
```

```
// Multiple .where on different fields
// require a composite index on both fields
// At most one inequality comparison!!
const q = query(collection(__, "students"),
        where("major", "==", "MATH")
        where("gpa", ">=", 3.0));
getDocs(q)
    .then(/* more code */);
```

```
// Can't use inequalities on two different fields
query(collection(__, 'states'),
        where("population", ">=", 5_000_000),
        where("area", "<=", 200_000));
```

Building Composite Index



Collection ID	Fields indexed	Query scope	Status
states	name Ascending population Ascending	Collection	Enabled
states	name Ascending population Descending	Collection	Enabled

***Order** of index build does matter!!!*

Live Demo #3: firestore/src/update-funcs.ts

```
# Create a ref to doc
const aDoc = doc(dbRoot, “path/to/your/doc”);
updateDoc(aDoc, { /* details of update here */ })
  .then(() => {
    // work after document is updated
  });
```

CRUD: Update Doc (change a simple field)

SQL

```
// Update a record with a known primary key
UPDATE students SET phone = “616-616-6161” WHERE gnumber = “G71884”
```

Gnumber	Name	Phone
G81291	Abby	517-123-4567
G71884	Ally	269-333-4444
G53181	Annie	616-777-3332

```
// Assume saved data has the
// following structure
type StudentType = {
  //gnumber: string; // PrimaryKey
  name: string;
  phone: string;
}
```

TS

```
// After initialization
const docRef: DocumentReference = doc(db, ‘students/G71884’)

// add a new simple data
updateDoc(docRef, { phone: “616-616-6161” })
.then(() => {
   console.debug(“Update successful”);
})
```

CRUD: Update Doc (change a simple field)

SQL

```
// Update a record when primary key is UNKNOWN
UPDATE students SET phone = “616-616-6161” WHERE name = “Abby”
```

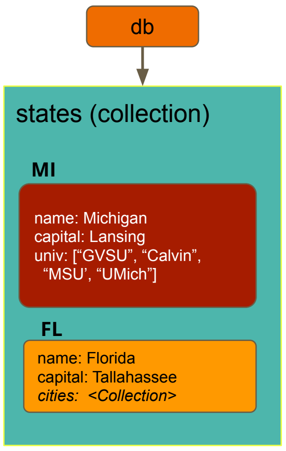
Gnumber	Name	Phone
G81291	Abby	517-123-4567
G71884	Ally	269-333-4444
G53181	Annie	616-777-3332

```
// Assume saved data has the
// following structure
type StudentType = {
  //gnumber: string; // PrimaryKey
  name: string;
  phone: string;
}
```

FirestoreTS

```
const myColl: CollectionReference = collection(db, ‘students’)

const qr = query(myCol, where(“name”, “==”, “Abby”))

getDocs(qr).then((qs:QuerySnapshot) => {
  qs.forEach(async (qd:QueryDocumentSnapshot) => {
    const myDoc = doc(db, qd.id);
    await updateDoc(myDoc, { phone: “616-616-6161” })
  })
})
```

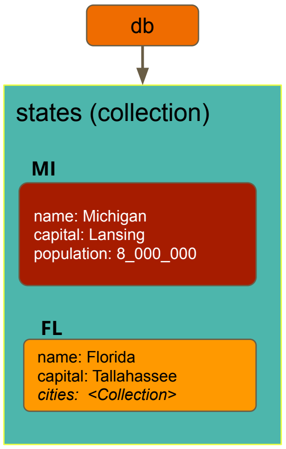
CRUD: Update array field in a Document



TS

```
// After initialization
import {updateDoc, arrayRemove, arrayUnion} from “firebase/firestore”
const mich: DocumentReference = doc(db, ‘states/MI’)

// add a new JS/TS array
updateDoc(mich, { universities: [“GVSU”, “Calvin”, “XYZ”] })
.then(() => { console.debug(“Update successful”);})

// updated erroneous entry in the array
updateDoc(mich, {
  universities: arrayRemove(“XYZ”)
})
.then(() => { console.debug(“Update successful”);})

// Add more entries in the array
updateDoc(mich, {
  universities: arrayUnion(“MSU”, “UMich”)
})
.then(() => { console.debug(“Update successful”);})
```

CRUD: Update numeric field in a Document



TS

```
import {updateDoc, increment} from “firebase/firestore”

const mich: DocumentReference = doc(db, ‘states/MI’)

updateDoc(mich, {
  // Add 1234 to the current population
  population: increment(1234)
})
.then(() => { console.debug(“Update successful”);})
```

Live Demo #4:
firestore/src/delete-funcs.ts

```
# Create a ref to doc
const aDoc = doc(dbRoot, “path/to/your/doc”);
deleteDoc(aDoc)
  .then(() => {
    // work after document is deleted
  });
```

CRUD Operations: Delete one Document

SQL

```
// Delete a record with a known primary key
DELETE FROM students WHERE gnumber = "G71884"
```

Firebase TS

```
import {deleteDoc, updateDoc, deleteField} from
"firebase/firestore";

// Delete the entire document
const toRemove = doc(db, "students/G71884")
deleteDoc(toRemove)
  .then(() => { console.debug("Student G71884 removed") });
```

CRUD: Delete One Document (unknown Doc ID)

SQL

```
// Update a record when primary key is UNKNOWN
DELETE FROM students WHERE name = "Abby"
```

Gnumber	Name	Phone
G81291	Abby	517-123-4567
G71884	Ally	269-333-4444
G53181	Annie	616-777-3332

```
// Assume saved data has the
// following structure
type StudentType = {
  //gnumber: string; // PrimaryKey
  name: string;
  phone: string;
}
```

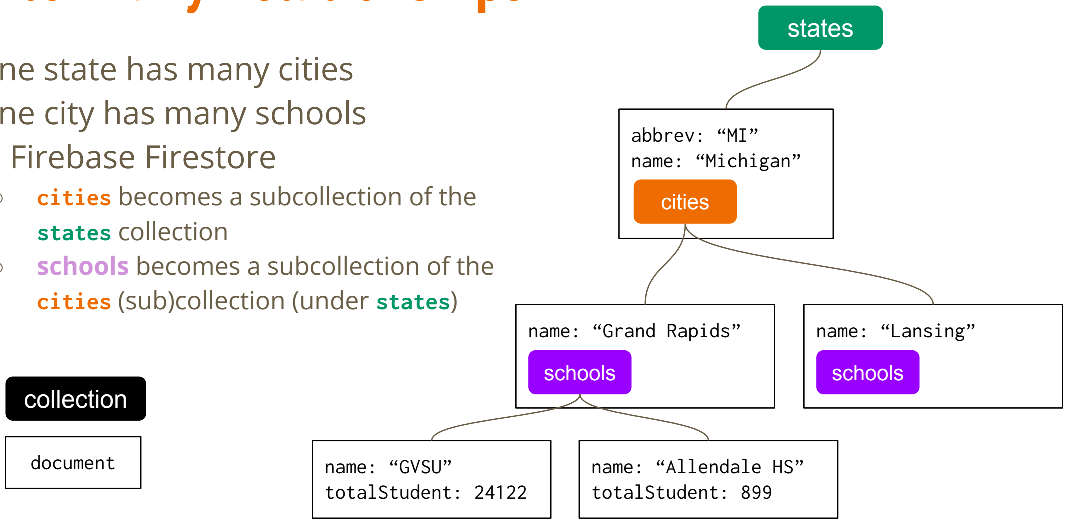
FirestoreTS

```
const myColl: CollectionReference = collection(db, 'students')

const qr = query(myCol, where("name", "==", "Abby"))

getDocs(qr).then((qs:QuerySnapshot) => {
  qs.forEach(async (qd:QueryDocumentSnapshot) => {
    const myDoc = doc(db, qd.id);
    await deleteDoc(myDoc)
  })
})
```

SubCollections: One-to-Many Relationship

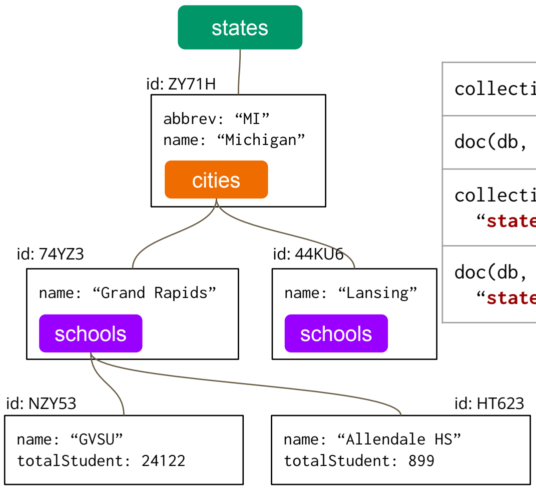
One-to-Many Relationships

- One state has many cities
- One city has many schools
- In Firebase Firestore
  - `cities` becomes a subcollection of the `states` collection
  - `schools` becomes a subcollection of the `cities` (sub)collection (under `states`)



Sub-Collections

collection document



`collection(db, "states/ZY71H/cities")`	Cities in Michigan
`doc(db, "states/Z71H/cities/74Y23")`	City of Grand Rapids
`collection(db, "states/ZY71H/cities/74Y23/schools")`	Schools in GR
`doc(db, "states/ZY71H/cities/74Y23/schools/NZY53")`	GVSU

Operations on SubCollections

`collection(db, "states/ZY71H/cities")`	Cities in Michigan
`doc(db, "states/Z71H/cities/74Y23")`	City of Grand Rapids
`collection(db, "states/ZY71H/cities/74Y23/schools")`	Schools in GR
`doc(db, "states/ZY71H/cities/74Y23/schools/NZY53"`	GVSU

```
// Add a new city in Michigan
const miCities = collection(db, "states/ZY71H/cities")
await addDoc(miCities, {
  name: "Holland",
  /* more details on Holland */}
```

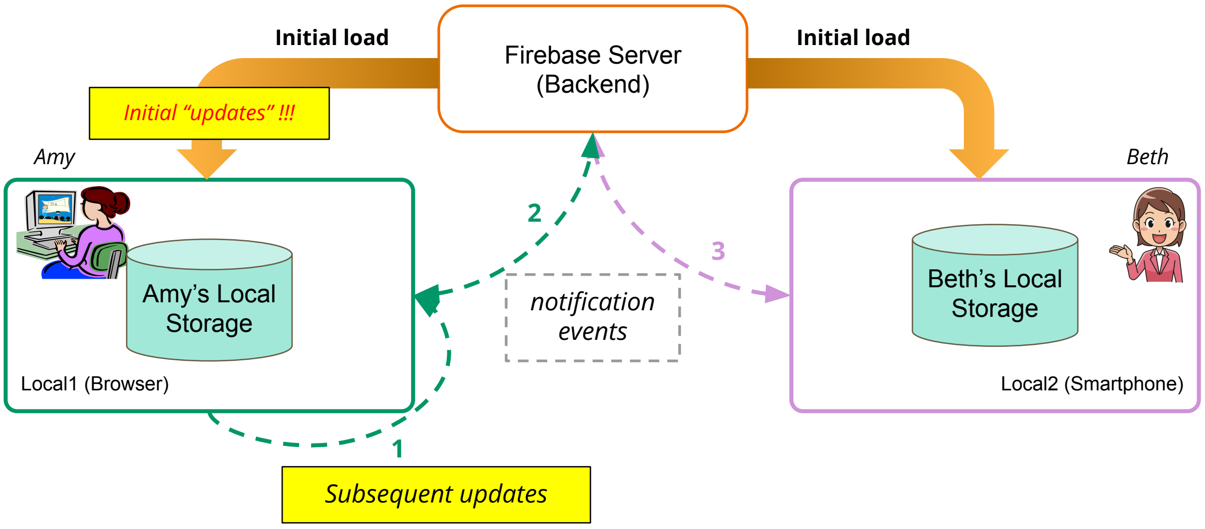
```
// Update Grand Rapids details
const grDoc = doc(db, "states/Z71H/cities/74Y23")
await updateDoc(grDoc, { subwayAvailable: false })
```

```
// Get all schools in Grand Rapids
const grSchools = collection(db, "states/ZY71H/cities/74Y23/schools")
getDocs(grSchools).then((qs:QuerySnapshot) => {
  qs.forEach((qd:QueryDocumentSnapshot) => {
     const skool = qd.data() as SchoolType
  })
})
```

Collection/Doc Update Listener(s)

Benefit: Interactive Web App

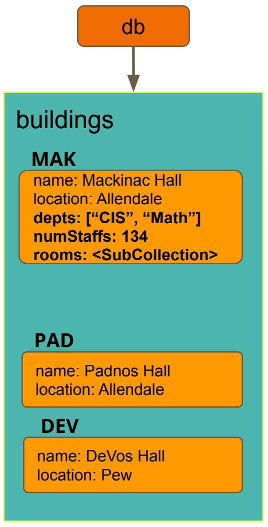
Local Storage, Local Events & Global Events



Live Demo #5: firestore/src/listen-funcs.ts

```
# Get a ref to the document
const aDoc = doc(dbRoot, “path/to/your/doc”);
onSnapshot(aDoc, (snap:DocumentSnapshot) => {
  // updated doc in snap.data()
});

# Get a ref to the collection
const aColl = collection(dbRoot, “path/to/your/collection”);
onSnapshot(aColl, (snap:QuerySnapshot) => {
  for (let chg in snap.docChanges()) {
    // doc details in chg
  });
```

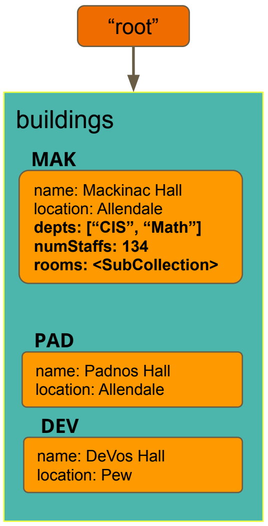
Listening to Field Updates on a SINGLE doc



```
import {doc, onSnapshot, DocumentSnapshot} from “firebase/firestore”;
const mac = doc(db, “buildings/MAK”);

// Listen to updates on a single document
onSnapshot(mac,
  (snapshot:DocumentSnapshot) => {
     const newData = snapshot.data();
     console.log(“Document has been updated to”, newData);
  });
```

*Will NOT receive notifications on updates of the doc subcollections (**rooms** in the example)*

Listening to Updates on a Collection of Docs



```
import {collection, onSnapshot, QuerySnapshot} from "firebase/firestore";

const bldColl = collection(db, 'buildings');

onSnapshot(bldColl,
  (s:QuerySnapshot) => {
    for (let chg of s.docChanges()) {
      const newData = chg.doc.data();
      const updateAction = chg.type;  // "added", "modified", "removed"
      console.log(chg.doc.id, "has been", updateAction, newData);

  });
```