# Cloud Data Stores

## Why Cloud Data Stores?

- On premise data stores = your time + energy for storage maintenance
  - Hire a DBA in your team?
- On cloud data stores are highly scalable
- On cloud data stores are usually built using no SQL technology
- Cloud data stores are accessible to both **web** and **mobile** clients

## SQL vs. no SQL

- Relation model
- Data are stored into two or more tables on **a single server**
- Schema: relationship between tables and fields
- Popular examples
  - Oracle
  - DB2
  - MySQL
  - PostGreSQL

- Non-relational
- Data may be stored **distributed across cluster of servers**
- No schema
- Cloud Computing and Cloud Storage
- Rapid Development
- Popular examples
  - MongoDB
  - CouchDB
  - BigTable
  - Firebase

## Entity Relationship


### Relational Databases

- Three tables
  - `Order` (with **primary key**)
  - `Delivery` (**foreign key**)
  - `Product` (**foreign key**)
- The order and its associated data are split across several places
- Potentially more I/O operations, DBMS may have to access **three separate files**

## Entity Relationship


**Non-Relational Databases**

- Place related data in one place
- The order and its associated data are **placed together** (potentially in the same file)
- Fewer I/O operations

# No SQL Data Models

- Key-Value: given a key, the database returns its associated value
- Document Database
  - extending the key-value concept
  - Given a key, the database returns its associated **document**
- *Other models (not relevant to Firebase)*

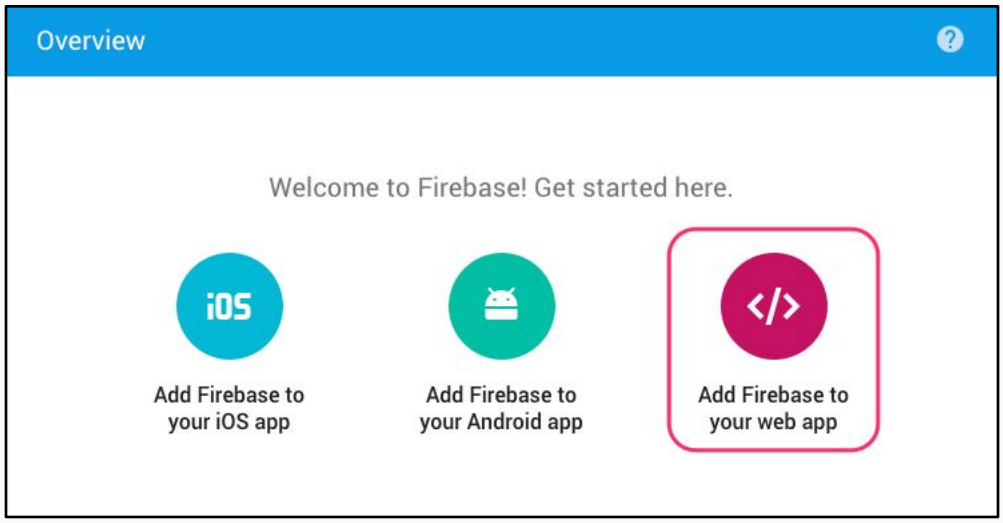
# Firebase

- Document data model
- JSON representation
  - The entire database is just a **giant** JSON tree
- http://firebase.google.com
- Supported Platforms
  - Mobile: iOS, Android
  - Web
  - Unity, C++


# Demo 1: New Firebase Project

# Demo 2: New Firebase App (web)

## Creating a new WebApp



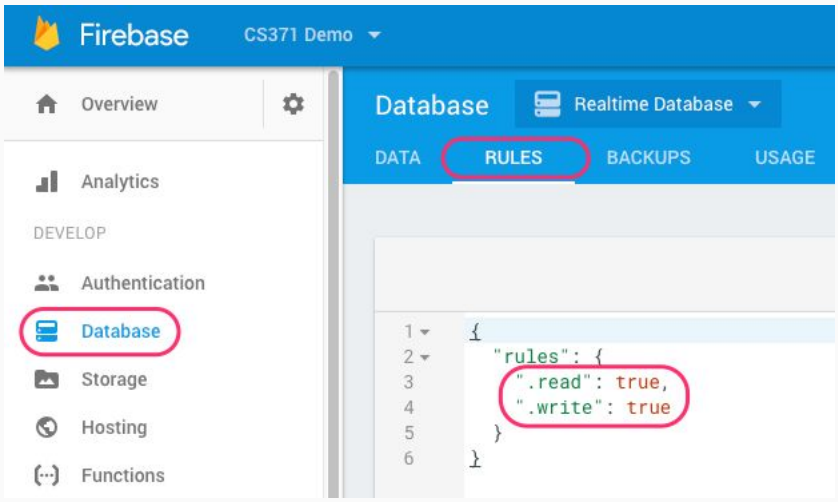
## Initialization Code

```
<!-- copy this snippet into your HTML file -->
<script src="https://www.gstatic.com/firebasejs/4.5.0/firebase.js"></script>
<script>
  // Initialize Firebase
  var config = {
    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: "80758575596"
  };
  firebase.initializeApp(config);  // firebase is a GLOBAL variable, use it
                                   // throughout your code
</script>
```

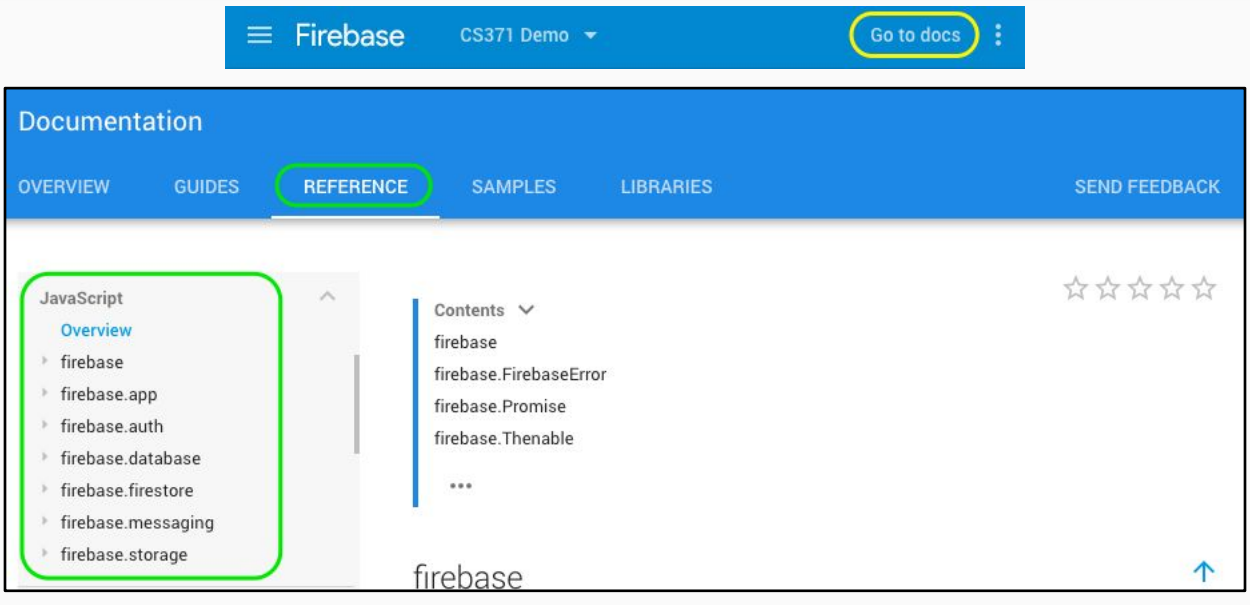
# Database Dashboard

- 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

## Database Security Rules




## Firebase Online Docs



## Firebase API References

Frequently Used Classes

- firebase.app
- firebase.database
- firebase.database.Reference (reference to data nodes)
- firebase.database.DataSnapShot
- firebase.database.Query

## Firebase DB = a giant JSON tree

*Operations are applied to data nodes in the tree*

## Transition from SQL to Firebase

| SQL | Firebase |
|---|---|
| Tables | Immediate JSON Nodes of ROOT |
| Keys | Autogenerated Keys |
| Columns | JSON key-value pairs |

## SQL "NOT NULL"

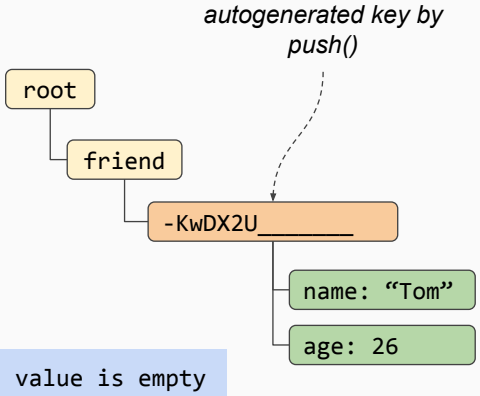
- A data node in Firebase JSON tree must have **at least one child** node
- When all the children of a node (P) is removed, then node P itself is automatically removed by Firebase
- The removal of nodes propagates upwards (from child to parent to grandparent to grand-grandparent …. *ad infinitum*)

# INSERT or UPDATE

## Firebase: INSERT === push().set({....})

```
//SQL
INSERT INTO friend (name, age) VALUES (“Tom”, 26)
```

```
<!-- after Firebase is initialized -->
<script>
var rootRef = firebase.database().ref();
friendRef = rootRef.child(“friend”);
friendRef.push().set({ name: “Tom”, age: 26 });
</script>
```




push(): creates a new node with a unique key, but its value is empty

set(): overwrites a new content into an existing node.

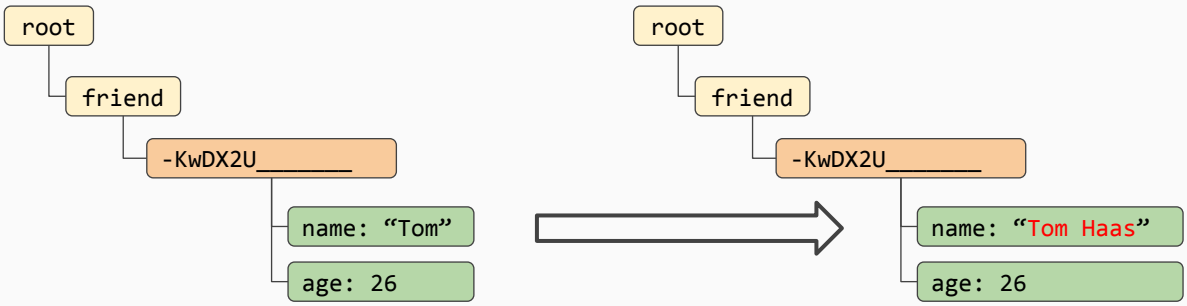
## Using ref() and child()

```
// without arg, ref() points to the ROOT node
var topRef = firebase.database().ref();
var cisRef = topRef.child("dept").child("cis");
```

```
var topRef = firebase.database().ref();
var cisRef = topRef.child("dept/cis");
```

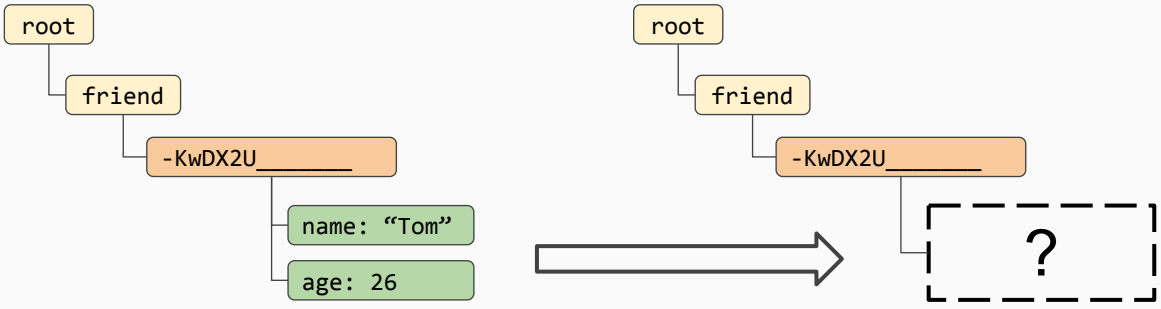
```
// this snippet has the same effect as the above
// with arg, ref() points to the specified node
var cisRef = firebase.database().ref("dept/cis");
```

## Updating Node Contents Using set()



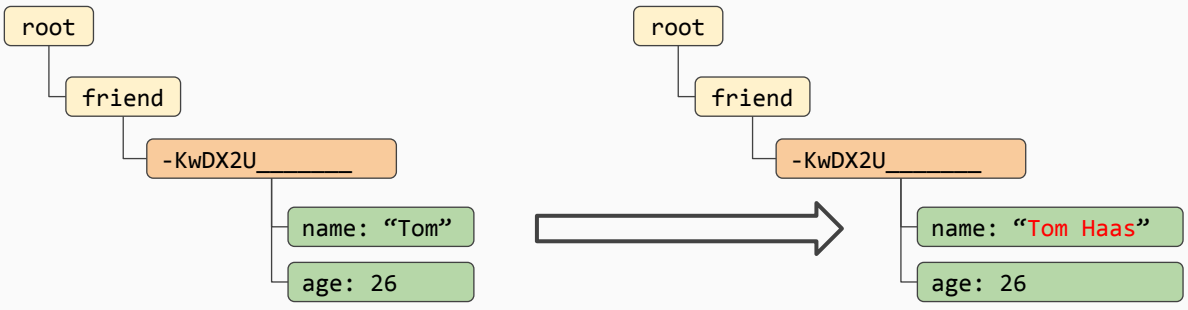

```
<script>
var topRef = firebase.database().ref();
nameRef = topRef.child("friend/-KwDX2U_____/name");
nameRef.set("Tom Haas");   // Update to new string
</script>
```

## Incorrect Use of `set()`




```
<!-- What's wrong with this code? -->
<script>
var topRef = firebase.database().ref();
nameRef = topRef.child("friend/-KwDX2U_____");
nameRef.set({ name: "Tom Haas"});
</script>
```

## `update()`: selectively update a node's children




```
<script>
var topRef = firebase.database().ref();
nameRef = topRef.child("friend/-KwDX2U_____");
nameRef.update({ name: "Tom Haas" });   // Update to new string
</script>
```

```
SELECT * FROM table
```

## Firebase Event Listeners

- `on()`: activate listener until `off()` is called
- `off()`: deactivate listener
- `once()`: activate listener once
- Listeners can be attached to any data node in the JSON tree

## Event Types

| Event | When Event is Triggerred? |
|---|---|
| Value | Initial data stored at a given node or when the data changes |
| Child Added | Once for each child added to a node (the parent) |
| Child Removed | Once for each child removed from a node (the parent) |
| Child Changed | Once for each child (*or any of its descendants*) is updated |
| Child Moved | The child sort order (i.e. relative position to its sibling) changes |

# Demo:
# Show Events on Dashboard

## Firebase equivalent of: SELECT * FROM table

```
//SQL
SELECT * FROM friend
```

```
<script>
var tabRef = firebase.database().ref().child("friend");
tabRef.on("child_added", function(snapshot) {
  console.log(snapshot.key);    // key of the node
  console.log(snapshot.val());  // the object associated with the key
});
</script>
```

```
<script>   // Using JavaScript arrow function
var tabRef = firebase.database().ref().child("friend");
tabRef.on("child_added", snapshot => {
  console.log(snapshot.key);    // key of the node
  console.log(snapshot.val());  // the object associated with the key
});
</script>
```

## off(): removing Event Listeners

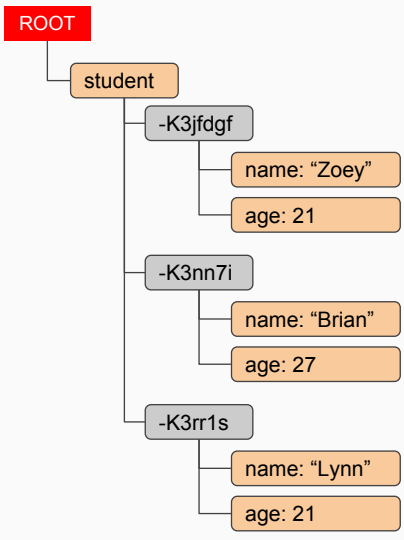
```
var nodeRef = firebase.database.ref("path/to/your/node");

var chldListener = nodeRef.on("child_added", snapshot => {
  /* this function listens for new child insertions */
});
```

```
/* much later in code when the listener is not needed anymore */
nodeRef.off("child_added", chldListener);
```

# Firebase Queries

Query: SELECT * FROM **student** WHERE **age >= 25**;




```
var rootRef = firebase.database().ref(“student”);
rootRef.orderByChild(“age”).startAt(25).on(“child_added”,
   snapshot => {
      var st = snapshot.val();

      // access  st.name     and     st.age

   });
```

```
rootRef.orderByChild(“age”).endAt(30).on(“child_added”,
   snapshot => {
      // students whose age <= 30
   });
```

```
rootRef.orderByChild(“age”).equalTo(21).on(“child_added”,
  snapshot => {
      // students whose age == 21
  });
```

**Firebase allows only ONE orderByXXXX() call per query!**

# Improving performance of orderBy___()

- Change security rule of your DB
- Add .indexOn entry on the node(s) that use orderBy___()

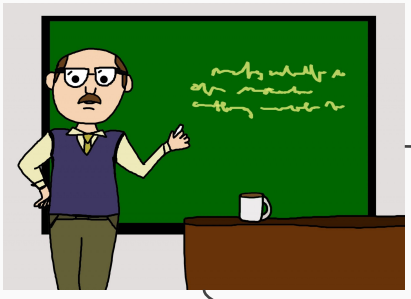
```
“rules”: {
  “student”: {
    “.indexOn”: [“age”, “gpa”]
  }
}
```

# Query Related Functions

- orderByChild(), orderByKey(), orderByValue()
- startAt(), endAt(), equalTo()
  - Filter data to certain range
- limitToFirst(N), limitToLast(N)
  - Sets the maximum number of items returned by a query to N

# Case Study: EZPoll

## EZPoll Session Management



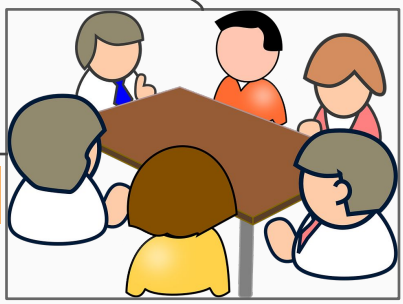
Step 1: Create Session

Step 2: Join Session

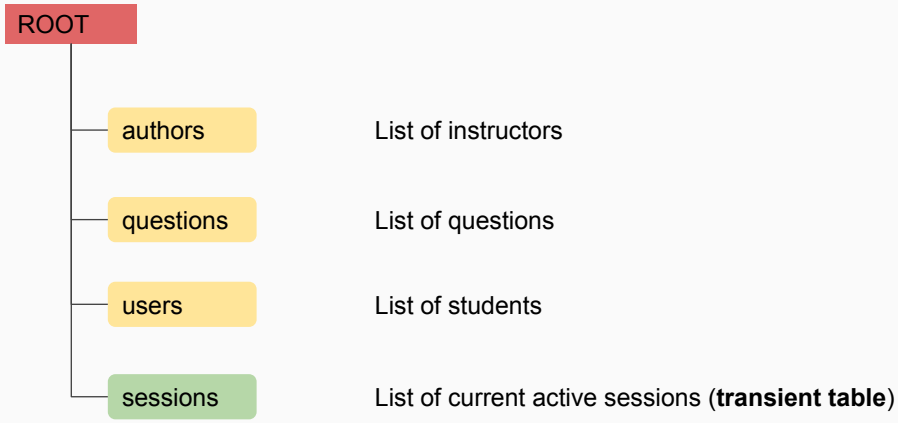
Firebase DB

Step 3: Post Question

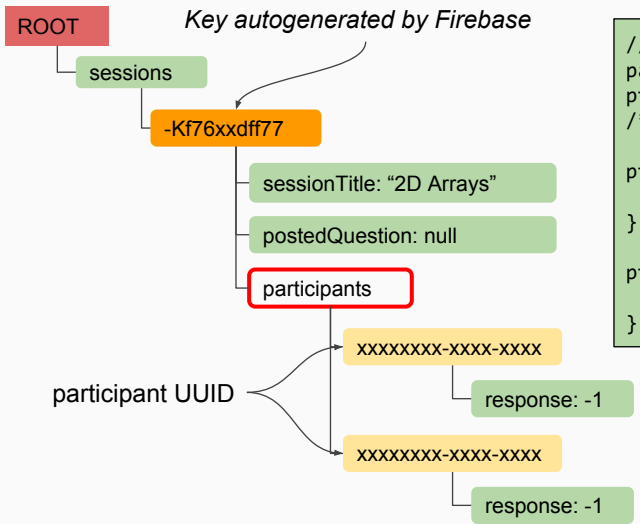
Step 4: Submit Answer



## EZPoll “tables”




## Step 1: Instructor creates a new session and Step 2: Students join the session



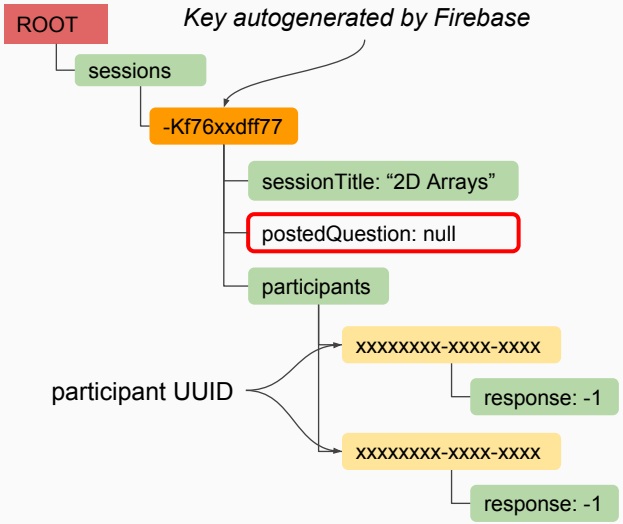

```
// author logic for monitoring incoming students
path = “sessions/-Kf76xxdff77/participants”;
ptRef = firebase.database().ref(path);
/* Use child added event listener */

ptRef.on(“child_added”, snapshot => {
  /* update instructor dashboard */
});

ptRef.on(“child_removed”, snapshot => {
  /* update instructor dashboard */
});
```

```
// student logic for joining a session
path = “sessions/-Kf76xxdff77/participants”;
ptRef = firebase.database().ref(path);
uuid = generate_my_uuid();
ptRef.child(uuid).set({ response: -1 });
```

## Step 3: Instructor posts a question




```
// author updates question
path = “sessions/-Kf76xxdff77/postedQuestion”;
qstRef = firebase.database().ref(path);
qstRef.set (key_of_selected_question);
```

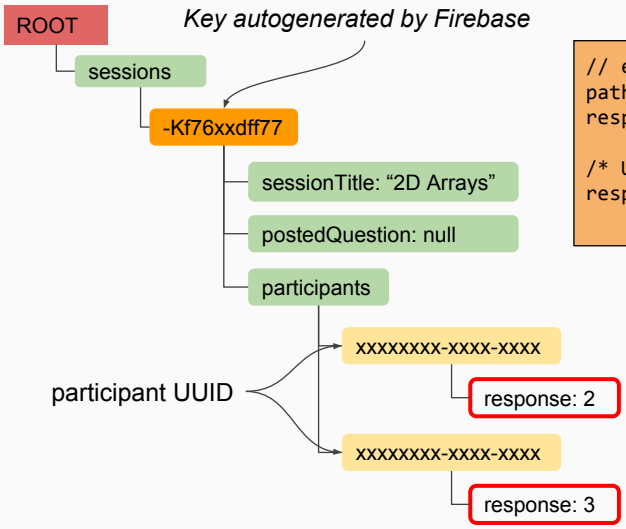
```
// student listening for new questions
path = “sessions/-Kf76xxdff77/postedQuestion”;
qstRef = firebase.database().ref(path);

/* Use value event listener */
ptRef.on(“value”, snapshot => {

  // retrieve question details

});
```

## Step 4: Students submit answer




```
// each student submits his/her answer
path = “sessions/-Kf76xxdff77/participants/”;
respRef = firebase.database().ref(path + myUUID);

/* Update response */
respRef.update({ response : myanswer });
```