
FREE eBook
LEARNING
firebase-database
Free unaffiliated eBook created from
Stack Overflow contributors.
#firebase-
database

Table of Contents

About

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

It is an unofficial and free firebase-database 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 firebase-database.

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 firebase-database

Remarks

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

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

Versions

Platform SDK	Version	Release date
Firebase JavaScript SDK	3.7.0	2017-03-01
Firebase C++ SDK	3.0.0	2107-02-27
Firebase Unity SDK	3.0.0	2107-02-27
Firebase iOS SDK	3.14.0	2017-02-23
Firebase Android SDK	10.2	2017-02-15
Firebase Admin Node.js SDK	4.1.1	2017-02-14
Firebase Admin Java SDK	4.1.2	2017-02-14

Examples

Add Firebase to Your Android Project

Here the steps required to create a Firebase project and to connect it with an Android app.

Add Firebase to your app

1. Create a Firebase project in the Firebase console and click **Create New Project**.
2. Click **Add Firebase to your Android app** and follow the setup steps.
3. When prompted, enter your **app's package name**.

It's important to enter the package name your app is using; this can only be set when you add an app to your Firebase project.

4. At the end, you'll download a `google-services.json` file. You can download this file again at any time. (this file is located under project setting in Firebase console).

5. Switch android studio View to Project and paste google-service.json file under app folder

The next step is to Add the SDK to integrate the Firebase libraries in the project.

Add the SDK

To integrate the Firebase libraries into one of your own projects, you need to perform a few basic tasks to prepare your Android Studio project. You may have already done this as part of adding Firebase to your app.

1. Add rules to your root-level `build.gradle` file, to include the **google-services plugin**:

```
buildscript {
    // ...
    dependencies {
        // ...
        classpath 'com.google.gms:google-services:3.1.1'
    }
}
```

Then, in your module Gradle file (usually the `app/build.gradle`), add the apply plugin line at the bottom of the file to enable the Gradle plugin:

```
apply plugin: 'com.android.application'

android {
  // ...
}

dependencies {
  // ...
  compile 'com.google.firebase:firebase-core:9.4.0'//THIS IS FOR ANALYTICS
  compile "com.google.firebase:firebase-database:11.0.2"
}

// BELOW STATEMENT MUST BE WRITTEN IN BOTTOM
apply plugin: 'com.google.gms.google-services'
```

Notes:

- Data cannot be read/write without Authenticating. If you want it without authentication. Change rules in Database firebase console.

  { "rules": { ".read": true, ".write": true } }

- Add internet permission in Manifest

- Upgrade Google Play Services and Google Repository

Writing simple value into database

First, complete the installation and setup to connect your app to Firebase. Then from anywhere in your class, you can write:

```
// Write a message to the database
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference("message");

myRef.setValue("Hello, World!");
```

It will write `Hello, Wold!` into `message` node, like seen below:

```
"your-project-parent" : {
    "message" : "Hello, World!"
}
```

Explanation

```
FirebaseDatabase database = FirebaseDatabase.getInstance();
```

Above code will assign `FirebaseDatabase` instance into `database` object for further use.

```
DatabaseReference myRef = database.getReference("message");
```

Above code will reference `myRef` object into `"message"` child of your project's parent (in this example, it is `"your-project-parent"`). So it is `"your-project-parent/message"`

```
myRef.setValue("Hello, World!");
```

Above code will set `"Hello, World!"` into path referenced by `myRef`

Automatically map custom model to data structure

After you have set a few data to database and have get a structure consisting of several nodes like this;

```
"user" : {
    "-KdbKcU2ptfYF2xKb5aO" : {
      "firstName" : "Arthur",
      "lastName" : "Schopenhauer",
      "userName" : "AphorismMan",
      "phone" : "+9022-02-1778",
      "gender": "M",
      "age" : 25
    },
    "-KdbQFjs9BDviuqQVHjY" : {
      "firstName" : "Werner",
      "lastName" : "Heisenberg",
```

```
      "userName" : "whereAmI",
      "phone" : "+9005-12-1901",
      "gender": "M",
      "age" : 75
    }
  }
```

you can categorize data structures.

Creating Class

Create a model class to set to database.

```
@IgnoreExtraProperties
public class User {
    public String firstName;
    public String lastName;
    public String userName;
    public String phone;
    public String gender;
    public int age;

    public User() {
    }

    public User(String firstName, String lastName, String userName, String phone, String
gender, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.userName = userName;
        this.phone = phone;
        this.gender = gender;
        this.age = age;
    }
}
```

Some things to remember when creating a model class that you want to map to your data:

1. You have to have an empty constructor
2. Scope of Variables/Fields must be public, so that the DataSnapshot returning from the firebase can access these fields. If you don't do that, when you want to get data, DataSnapshot can't access to your model in callback and that will cause an exception.
3. Names of Variables/Fields should match to those in your data structure.

Sending to Firebase

Create a User object

```
User user = new User ( "Arthur","Schopenhauer","AphorismMan","+9022-02-1778","M",25)
```

and reference

```
DatabaseReference databaseReference = FirebaseDatabase.getInstance().getReference();
```

Now you have the reference of your database. Create an `user` node with `databaseReference.child("user")`. If you do `.push()` your models will locate under randomly created unique ids like above, `"-KdbKcU2ptfYF2xKb5aO", "-KdbQFjs9BDviuqQVHjY"`.

```
databaseReference.child("user").push().setValue(user, new
DatabaseReference.CompletionListener() {
            @Override
            public void onComplete(DatabaseError databaseError, DatabaseReference
databaseReference) {
                Toast.makeText(getActivity(), "User added.", Toast.LENGTH_SHORT).show();

            }
        });
```

If you want to set your datas under your specific key, do it with `.child("yourSpecificKey")` instead of `.push()`.

```
databaseReference.child("user").child("yourSpecificKey").setValue(user,...
```

Read Getting started with firebase-database online: https://riptutorial.com/firebase-database/topic/3044/getting-started-with-firebase-database

Chapter 2: Firebase Query

Introduction

Firebase Query can be used to order a collection of data based on some attributes as well as restricted to the large list of items (for like chat data) down to a number suitable for synchronizing to the client.

Just as with a Reference, you can receive data from a Query by using the on() method. You will only receive events and DataSnapshots for the subset of the data that matches your query.

Examples

Firebase Query Example

```
private void loadData(){
        DatabaseReference dbRef = FirebaseDatabase.getInstance().getReference();

        Query dataQuery = dbRef.child("chat").orderByChild("id").equalTo("user1");
        dataQuery.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                if (dataSnapshot.exists()) {
                    // dataSnapshot is the "issue" node with all children with id 0
                    for (DataSnapshot issue : dataSnapshot.getChildren()) {
                        // do something with the individual "issues"
                    }
                }
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {

            }
        });
    }
```

Read Firebase Query online: https://riptutorial.com/firebase-database/topic/10100/firebase-query

Chapter 3: Firebase Realtime Database Rules

Remarks

Firebase Realtime Database Rules determine who has read and write access to your database, how your data is structured, and what indexes exist. These rules live on the Firebase servers and are enforced automatically at all times. Every read and write request will only be completed if your rules allow it. By default, your rules are set to allow only authenticated users full read and write access to your database. This is to protect your database from abuse until you have time to customize your rules or set up authentication.

Firebase Database Rules have a JavaScript-like syntax and come in four types:

Rule Types	
.read	Describes if and when data is allowed to be read by users.
.write	Describes if and when data is allowed to be written.
.validate	Defines what a correctly formatted value will look like, whether i and the data type.
.indexOn	Specifies a child to index to support ordering and querying.

Examples

Authorization

Identifying your user is only part of security. Once you know who they are, you need a way to control their access to data in your database. Firebase Database Rules allow you to control access for each user. For example, here's a set of security rules that allows anyone to read the path `/foo/`, but no one to write to it:

```
{
  "rules": {
    "foo": {
      ".read": true,
      ".write": false
    }
  }
}
```

`.read` and `.write` rules cascade, so this ruleset grants read access to any data at path /foo/ as well as any deeper paths such as `/foo/bar/baz`. Note that `.read` and `.write` rules that permit access will

override other rules in the database that do not allow access; in other words all applicable, `.read` and `.write` rules are ORed together). So read access to `/foo/bar/baz` would still be granted in this example even if a rule at the path `/foo/bar/baz` evaluated to false.

The Firebase Database Rules include built-in variables and functions that allow you to refer to other paths, server-side timestamps, authentication information, and more. Here's an example of a rule that grants write access for authenticated users to `/users/<uid>/`, where is the ID of the user obtained through Firebase Authentication.

```
{
  "rules": {
    "users": {
      "$uid": {
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

Data validation

The Firebase Realtime Database is schemaless. This makes it easy to change things as you develop, but once your app is ready to distribute, it's important for data to stay consistent. The rules language includes a `.validate` rule which allows you to apply validation logic using the same expressions used for `.read` and `.write` rules. The only difference is that all relevant validation rules must evaluate to true in order for the write to be allowed (in other words, all applicable `.validate` rules are ANDed together to allow a database write).

These rule enforce that data written to `/foo/` must be a string less than 100 characters:

```
{
  "rules": {
    "foo": {
      ".validate": "newData.isString() && newData.val().length < 100"
    }
  }
}
```

Validation rules have access to all of the same built-in functions and variables as `.read` and `.write` rules. You can use these to create validation rules that are aware of data elsewhere in your database, your user's identity, server time, and much more.

Defining database indexes

The Firebase Realtime Database allows ordering and querying data. For small data sizes, the database supports ad hoc querying, so indexes are generally not required during development. Before launching your app though, it is important to specify indexes for any queries you have to ensure they continue to work as your app grows.

Indexes are specified using the `.indexOn` rule. Here is an example index declaration that would

index the height and length fields for a list of dinosaurs:

```
{
  "rules": {
    "dinosaurs": {
      ".indexOn": ["height", "length"]
    }
  }
}
```

Read Firebase Realtime Database Rules online: https://riptutorial.com/firebase-database/topic/3348/firebase-realtime-database-rules

Chapter 4: Firebase Realtime Database Transactions

Introduction

Transactions provide a mechanism to coordinate between multiple parties that might be accessing the same data at the same time. These "parties" might be different instances of the same code like different users running the same application or nodes in a server cluster, parts of the same program or event different programs like an administration application, a "end user" application and/or "backend" server logic.

Examples

A distributed counter

Imagine many users all running a web application that is trying to increment a counter in the database. Each user must read the current count, add one and write out the updated value. To make sure no one reads the counter while someone else is is adding one we use a transaction:

```
ref.transaction(function(value){
  if (value === null) {
    // the counter doesn't exist yet, start at one
    return 1;
  } else if (typeof value === 'number') {
    // increment - the normal case
    return value + 1;
  } else {
    // we can't increment non-numeric values
    console.log('The counter has a non-numeric value: ' + value)
    // letting the callback return undefined cancels the transaction
  }
});
```

Read Firebase Realtime Database Transactions online: https://riptutorial.com/firebase-database/topic/9612/firebase-realtime-database-transactions

Chapter 5: Firebase Real-Time Database with Android

Examples

Integrate Firebase Real-Time database with an Android application

How to implement Firebase Real-Time database in Android applications.

Setup/Installation:

1. First, install the Firebase SDK (guide)
2. Register your project using the Firebase console
3. After successfuly completing the two steps above, add the following dependency in your application level gradel.

```
compile 'com.google.firebase:firebase-database:9.2.1'
```

4. [Optional] Configure your database security rules (reference).

Implementation Sample:

1. Declare and initialize the database reference

```
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference("message");
```

You can later create different references to access different nodes

6. Write new data to the database

```
myRef.setValue("Writing Demo");
```

7. Read data from the database

```
myRef.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(DataSnapshot dataSnapshot) {
    // This method is called once with the initial value and again
    // whenever data at this location is updated.
    String value = dataSnapshot.getValue(String.class);
    Log.d(TAG, "Value is: " + value);
}

@Override
public void onCancelled(DatabaseError error) {
```

```
        // Failed to read value
        Log.w(TAG, "Failed to read value.", error.toException());
    }
});
```

Read Firebase Real-Time Database with Android online: https://riptutorial.com/firebase-database/topic/6341/firebase-real-time-database-with-android

Chapter 6: FirebaseRealtime database with Android

Examples

Add the Realtime Database in Android

1. Complete the Installation and setup to connect your app to Firebase.
   This will create the project in Firebase.

2. Add the dependency for Firebase Realtime Database to your module-level `build.gradle` file:

```
compile 'com.google.firebase:firebase-database:9.2.1'
```

3. Configure Firebase Database Rules

Now you are ready to work with the Realtime Database in Android.

For example you write a `Hello World` message to the database under the `message` key.

```
// Write a message to the database
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference("message");

myRef.setValue("Hello, World!");
```

Using setValue to save data

The`setValue()` method overwrites data at the specified location, including any child nodes.

You can use this method to:

1. Pass types that correspond to the available JSON types as follows:

- String
- Long
- Double
- Boolean
- Map<String, Object>
- List

2. Pass a custom Java object, if the class that defines it has a default constructor that takes no arguments and has public getters for the properties to be assigned.

This is an example with a CustomObject.
First define the object.

```
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default constructor required for calls to DataSnapshot.getValue(User.class)
    }

    public User(String username, String email) {
        this.username = username;
        this.email = email;
    }
```

Then get the Database reference and set the value:

```
   User user = new User(name, email);
   DatabaseReference mDatabase mDatabase = FirebaseDatabase.getInstance().getReference();
   mDatabase.child("users").child(userId).setValue(user);
```

Example for data insert or data retrieve from Firebase

Before understand require to follow some setup for project integrate with firebase.

1. Create your project in Firebase Console and download google-service.json file from console and put it in app level module of your project, Follow link for Create Project in console
2. After this we require to add some dependency in our project,

- First add class path in our project level gradle,

  ```
  classpath 'com.google.gms:google-services:3.0.0'
  ```

- And after that apply plugin in app level gradel,write it below of dependancy section,

  ```
  apply plugin: 'com.google.gms.google-services
  ```

- There are to more dependancy which require to add in app level gradle in dependancy section

  ```
  compile 'com.google.firebase:firebase-core:9.0.2'

  compile 'com.google.firebase:firebase-database:9.0.2'
  ```

- Now start to insert data in firebase database, First require to create instance of

  ```
  FirebaseDatabase database = FirebaseDatabase.getInstance();
  ```

  after creation of FirebaseDatabase object we going to create our DatabaseReference for insert data in database,

  ```
  DatabaseReference databaseReference = database.getReference().child("student");
  ```

  Here `student` is the table name if table is exist in database then insert data into table

otherwise create new one with student name, after this you can insert data using `databaseReference.setValue();` function like following,

```
HashMap<String,String> student=new HashMap<>();

student.put("RollNo","1");

student.put("Name","Jayesh");

databaseReference.setValue(student);
```

Here I am inserting data as hasmap But you can set as model class also,

- Start how to retrieve data from firebase, We are using here addListenerForSingleValueEvent for read value from database,

```
  senderRefrence.addListenerForSingleValueEvent(new ValueEventListener() {
          @Override
          public void onDataChange(DataSnapshot dataSnapshot) {
              if(dataSnapshot!=null && dataSnapshot.exists()){
                  HashMap<String,String>
studentData=dataSnapshot.getValue(HashMap.class);
                  Log.d("Student Roll Num "," : "+studentData.get("RollNo"));
                  Log.d("Student Name "," : "+studentData.get("Name"));
              }
          }

          @Override
          public void onCancelled(DatabaseError databaseError) {

          }
      });
```

Get value/s from firebase

1. Create class and add imports to parse information:

```
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.IgnoreExtraProperties;

//Declaration of firebase references
private DatabaseReference mDatabase;

//Declaration of firebase atributtes
public String uID;
public String username;
public String email;

@IgnoreExtraProperties
public class User {

    //Default constructor
    public User() {

        //Default constructor required for calls to DataSnapshot.getValue(User.class)
        mDatabase = FirebaseDatabase.getInstance().getReference();
```

```
        //...
    }

    //...
}
```

2. Add `addListenerForSingleValueEvent()` to our database reference:

```
//Add new imports
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.ValueEventListener;

//...

public void getUser(String uID){

    //The uID it's unique id generated by firebase database
    mDatabase.child("users").child(uID).addListenerForSingleValueEvent(
            new ValueEventListener () {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                // ...
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                // Getting Post failed, log a message
            }
    });
}
```

3. Inflate our class with firebase information in `onDataChange()` event:

```
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {

            //Inflate class with dataSnapShot
            Users user = dataSnapshot.getValue(Users.class);

            //...
        }
```

4. Finally we can get diferent atributtes from firebase class as normally:

```
//User inflated
Users user = dataSnapshot.getValue(Users.class);

//Get information
this.uID = user.uID;
this.username = user.username;
this.email = user.email;
```

Best practices

1. The firebase supports 32 different child levels, then is simple to write wrong a references, to evade this create a final private references:

```
//Declaration of firebase references
//...
final private DatabaseReference userRef =
mDatabase.child("users").child("premium").child("normal").getRef();

//...

public void getUser(String uID){

    //Call our reference
    userRef.child(uID).addListenerForSingleValueEvent(
        new ValueEventListener () {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                // ...
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                // Getting Post failed, log a message
            }
    });
}
```

2. The `onCancelled()` event is called when the user doesn't have access this reference by database rules. Add pertinent code to control this exception if you need.

For more information visit official documentation

Read FirebaseRealtime database with Android online: https://riptutorial.com/firebase-database/topic/6220/firebaserealtime-database-with-android

Chapter 7: Hello World!

Examples

Hello World in Android

1. Complete the Installation and setup part. This will create the project in Firebase console and will also install the base SDK in your Android App.
2. Add the dependency for Firebase Realtime Database to your app-level `build.gradle` file:

```
compile 'com.google.firebase:firebase-database:9.4.0'
```

3. Now write a `Hello World` message to the database under the `message` key.

```
// Write a message to the database
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference("message");

myRef.setValue("Hello, World!");
```

4. Check to see if the message showed up in the `Realtime Database`.

Hello World in IOS

1. After you have set up Firebase to your IOS Project following the documentation for setting up firebase for IOS in other firebase related documentation you can get going with firebase.

2. If you haven't already added the database pod to your Podfile do so now by adding `pod Firebase/Database` in order to get Database functionality for Firebase

3. Push 'Hello World' to the database but first you have to go and edit the Rules for your database in the Dashboard of your app in firebase and change it to

   ```
   {
     "rules": {
       ".read": true,
       ".write": true
     }
   }
   ```

   NOTE: Make sure you change this beofre you go into production because this means anyone can read and write into your database which is a major security flaw

4. Now all you have to do is import Firebase into the file you are using firbase in using `import Firebase` write the following line and you should see a 'Test' node in the database, expand it and it should have a child of "Hello World"!!!!!!

```
FIRDatabase.database().reference().child("Test").setValue("Hello World")
```

CONGRATS ON YOUR FIRST DATABASE PUSH

Read Hello World! online: https://riptutorial.com/firebase-database/topic/8885/hello-world-

Chapter 8: Reading data

Examples

Understanding which data referenced by getReference()

In this example, we use this database:

```
"your-project-name" : {
    "users" : {
        "randomUserId1" : {
            "display-name" : "John Doe",
            "gender" : "male"
        }
        "randomUserId2" : {
            "display-name" : "Jane Dae",
            "gender" : "female"
        }
    },
    "books" {
        "bookId1" : {
            "title" : "Adventure of Someone"
        },
        "bookId1" : {
            "title" : "Harry Potter"
        },
        "bookId1" : {
            "title" : "Game of Throne"
        }
    }
}
```

If you use above database then:

- `FirebaseDatabase.getInstance().getReference()`

  will point at your project's parent, `"your-project-name"` data. So the `dataSnapshot` you acquired will contain all of data inside it, including all of `"users"` data and `"books"` data.

- `FirebaseDatabase.getInstance().getReference("users")` and
  `FirebaseDatabase.getInstance().getReference().child("users")`

  will have the same result, pointing at `"your-project-name/users"`

- `FirebaseDatabase.getInstance().getReference("users/randomUserId1")` and
  `FirebaseDatabase.getInstance().getReference().child("users/randomUserId1")` and
  `FirebaseDatabase.getInstance().getReference().child("users").child("randomUserId1")`

  will have the same result, pointing at `"your-project-name/users/randomUserId1"`

  Note: this example is needed to fully understand which data is inside dataSnapshot object

Understanding which data is inside dataSnapshot object

> Note: You need to know which data referenced by getReference() first before you can completely understand this example.

There are three common method to get your data from Firebase Realtime Database:

- `addValueEventListener()`
- `addListenerForSingleValueEvent()`
- `addChildEventListener()`

When we talk about which data is inside `dataSnapshot` object, then `addValueEventListener()` and `addListenerForSingleValueEvent()` is basically the same. The only difference is `addValueEventListener()` keep listen to changes made in the referenced data while `addListenerForSingleValueEvent()` is not.

So consider we have this database:

```
"your-project-name" : {
    "users" : {
        "randomUserId1" : {
            "display-name" : "John Doe",
            "gender" : "male"
        }
        "randomUserId2" : {
            "display-name" : "Jane Dae",
            "gender" : "female"
        }
    },
    "books" {
        "bookId1" : {
            "title" : "Adventure of Someone"
        },
        "bookId1" : {
            "title" : "Harry Potter"
        },
        "bookId1" : {
            "title" : "Game of Throne"
        }
    }
}
```

DataSnapshot produced by addValueEventListener and addListenerForSingleValueEvent

`dataSnapshot` produced by `addValueEventListener()` and `addListenerForSingleValueEvent()` will contain value(s) of the exact data it is referenced into. Like when `ref` is point to `"your-project-name"` then `dataSnapshot` should be :

```
... onDataChange(DataSnapshot dataSnapshot) {
    dataSnapshot.getKey(); // will have value of String: "your-project-name"
    for (DataSnapshot snapshot : dataSnapshot) {
        snapshot.getKey(); // will have value of String: "users", then "books"
        for (DataSnapshot deeperSnapshot : dataSnapshot) {
```

```
            snapshot.getKey();
            // if snapshot.getKey() is "users", this will have value of String:
"randomUserId1", then "randomUserId2"
            // If snapshot.getKey() is "books", this will have value of String: "bookId1",
then "bookId2"
        }
    }
}
```

DataSnapshot produced by addChildEventListener

`dataSnapshot` produced by `addChildEventListener()` will contain value(s) of data one level deeper inside the data it is referenced into. Like in these cases:

When `ref` is point to `"your-project-name"` then `dataSnapshot` should be :

```
... onChildAdded(DataSnapshot dataSnapshot, String s) {
    dataSnapshot.getKey(); // will have value of String: "users", then "books"
    for (DataSnapshot snapshot : dataSnapshot) {
        snapshot.getKey();
        // if dataSnapshot.getKey() is "users", this will have value of String:
"randomUserId1", then "randomUserId2"
        // If dataSnapshot.getKey() is "books", this will have value of String: "bookId1",
then "bookId2"
        for (DataSnapshot deeperSnapshot : dataSnapshot) {
            snapshot.getKey();
            // if snapshot.getKey() is "randomUserId1" or "randomUserId1", this will have
value of String: "display-name", then "gender"
            // But the value will be different based on key
            // If snapshot.getKey() is "books", this will have value of String: "title", but
the value will be different based on key
        }
    }
}
// dataSnapshot inside onChildChanged, onChildMoved, and onChildRemoved will have the same
data as onChildAdded
```

I know most likely we will want to use `.getValue()` instead of `getKey()`. But in here we use `getKey` because it will always contain one String and no need to convert into custom object, or Map, or other. Basically, when you know which key `dataSnapshot` is pointing into, you can easily know which value it contains and parse it into your own custom obeject (or anything)

Read Reading data online: https://riptutorial.com/firebase-database/topic/9242/reading-data

Credits

S. No	Chapters	Contributors
1	Getting started with firebase-database	Abdul Wasae, Adarsh Madrecha, AtaerCaner , Community, ErstwhileIII, Gabriele Mariotti, koceeng, Nepster, Shiven, TwiterZX, Veeresh Charantimath
2	Firebase Query	Dhaval Solanki
3	Firebase Realtime Database Rules	Adarsh Madrecha, mckoss
4	Firebase Realtime Database Transactions	Mike
5	Firebase Real-Time Database with Android	Dhaval Solanki, Krishna Kumar, ThunderStruct
6	FirebaseRealtime database with Android	Dhaval Solanki, Gabriele Mariotti, Merlí Escarpenter Pérez, ThunderStruct
7	Hello World!	noob, RyanM, ThunderStruct
8	Reading data	koceeng