LA TROBE UNIVERSITY

All kinds of clever

# CSE2MAD

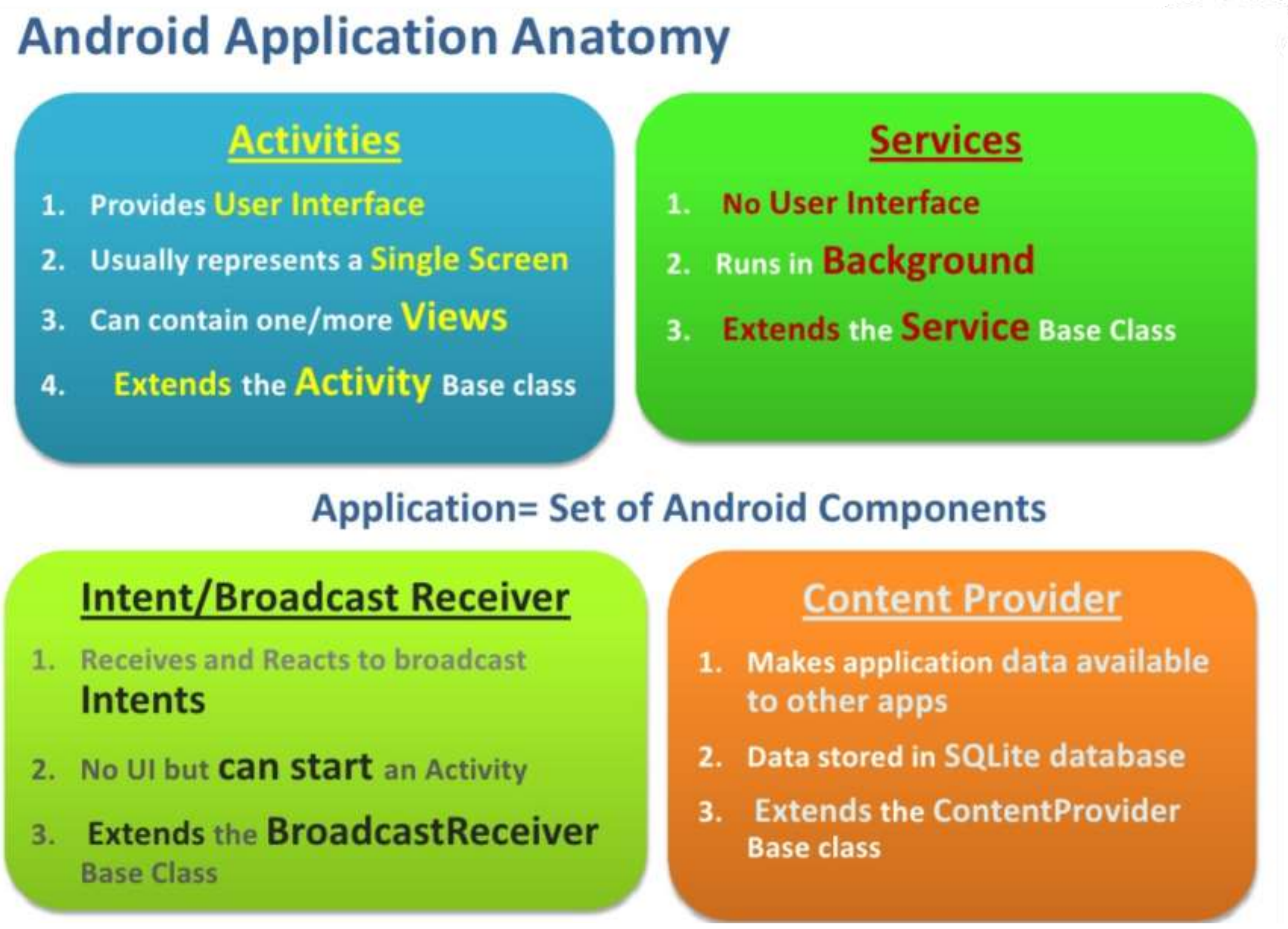
Mobile Application Development

Lecture 6

## Outline

- Content Providers
- SQLite DB
- Firebase
- Location

## Recap:


Android Application Anatomy

Activities
1. Provides User Interface
2. Usually represents a Single Screen
3. Can contain one/more Views
4. Extends the Activity Base class

Services
1. No User Interface
2. Runs in Background
3. Extends the Service Base Class

Application= Set of Android Components

Intent/Broadcast Receiver
1. Receives and Reacts to broadcast Intents
2. No UI but can start an Activity
3. Extends the BroadcastReceiver Base Class

Content Provider
1. Makes application data available to other apps
2. Data stored in SQLite database
3. Extends the ContentProvider Base class

## Content Providers

- Goal is to provide a mechanism to access data within and across applications.
- The CP's encapsulate and give a scheme for implementing security
- Abstract so the storage system may change but your apps don't have to
    - Eg. Using a different storage type or DB provider

Overview diagram of how content providers manage access to storage.

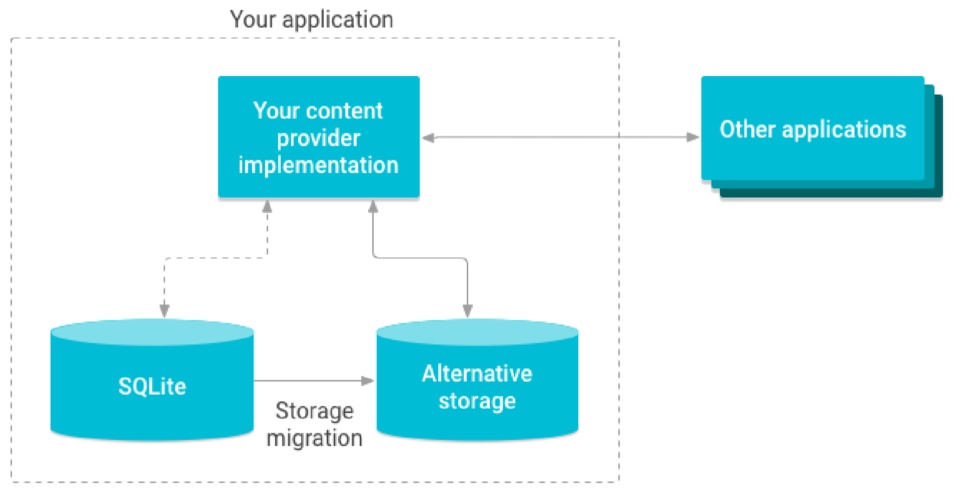
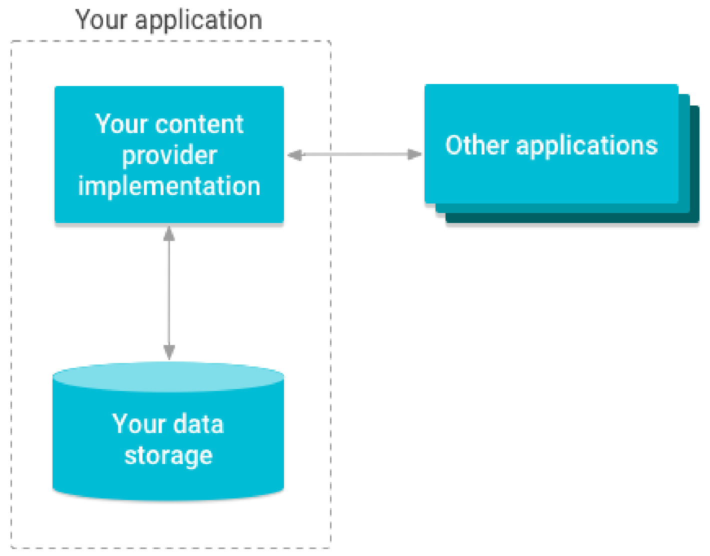


Illustration of migrating content provider storage.



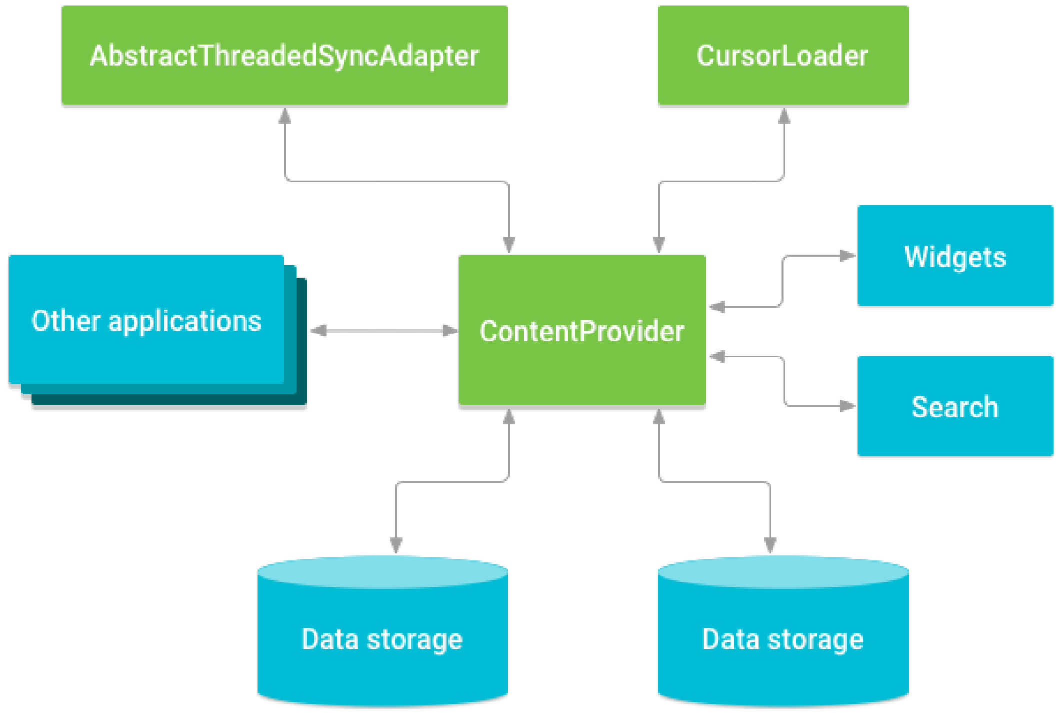
https://developer.android.com/guide/topics/providers/content-providers

# Content Providers

A content provider coordinates access to the data storage layer in your application for a number of different APIs and components.

Sharing access to your application data with other applications

- Sending data to a widget
- Returning custom search suggestions for your application through the search framework using SearchRecentSuggestionsProvider
- Synchronizing application data with your server using an implementation of AbstractThreadedSyncAdapter
- Loading data in your UI using a CursorLoader



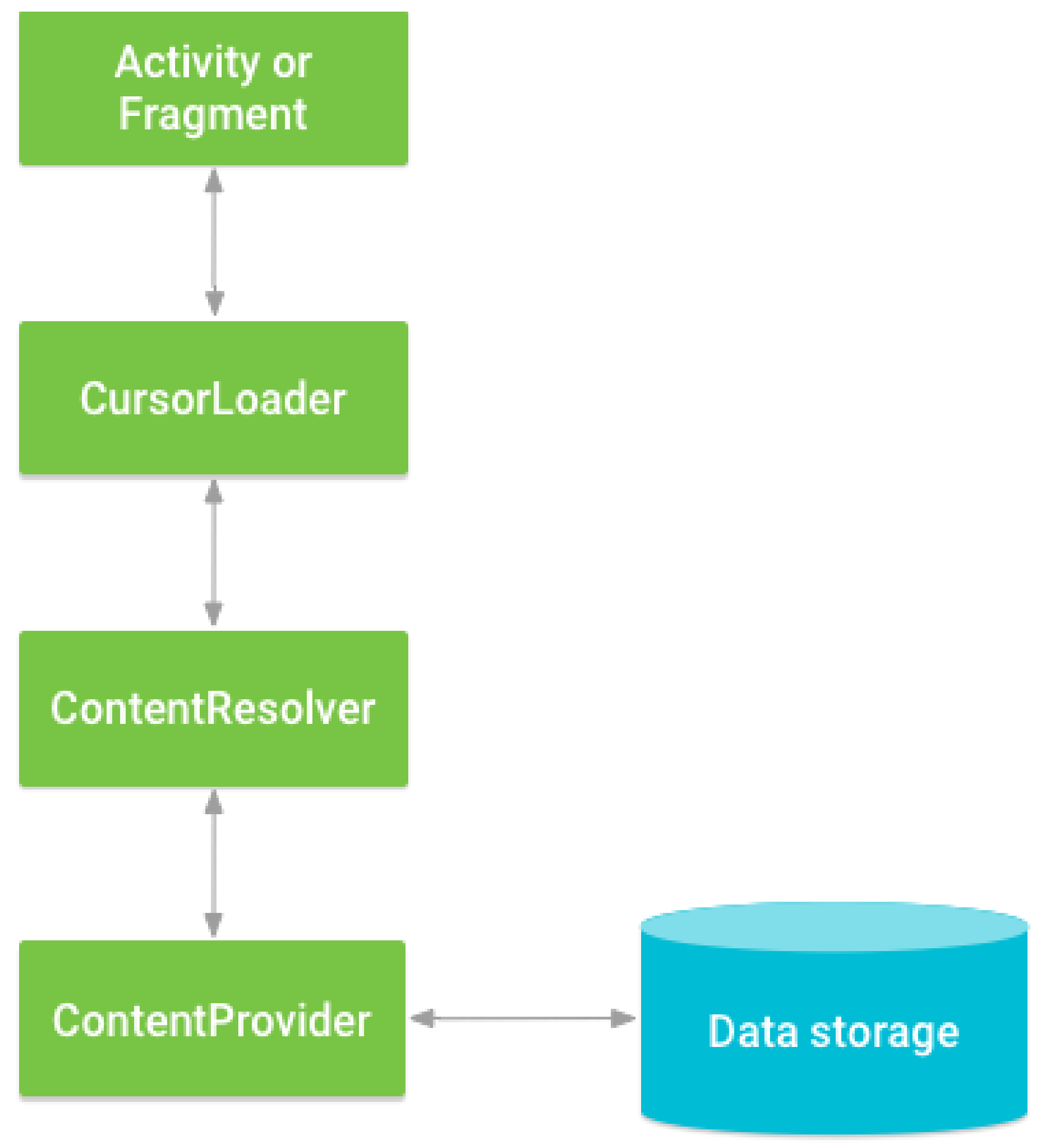
https://developer.android.com/guide/topics/providers/content-provider-basics

# Accessing a provider

A common pattern for accessing a ContentProvider from your UI uses a CursorLoader to run an asynchronous query in the background.

The Activity or Fragment in your UI call a CursorLoader to the query, which in turn gets the ContentProvider using the ContentResolver.

This allows the UI to continue to be available to the user while the query is running.



https://developer.android.com/guide/topics/providers/content-provider-basics

## Some Default Content Providers

- browser
- call log
- contacts
- mediastore
- settings
- userdictionary

Full list in the android.provider package: http://developer.android.com/reference/android/provider/package-summary.html

## What’s inside a content provider? (structure - typical)

One of the built-in providers in the Android platform is the user dictionary, which stores the spellings of non-standard words that the user wants to keep. The table illustrates what the data might look like in this provider's table:

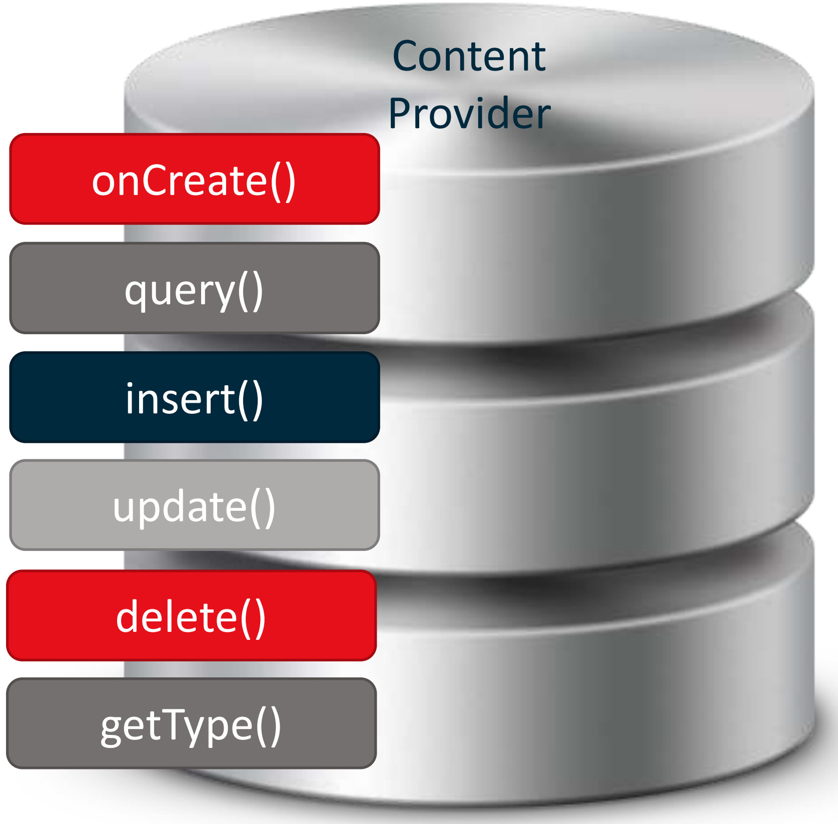
- a table: a collection of rows
- like a relational database tables but fewer constraints

Sample user dictionary table.

| word | app id | frequency | locale | _ID |
|---|---|---|---|---|
| mapreduce | user1 | 100 | en_US | 1 |
| precompiler | user14 | 200 | fr_FR | 2 |
| applet | user2 | 225 | fr_CA | 3 |
| const | user1 | 255 | pt_BR | 4 |
| int | user5 | 100 | en_UK | 5 |

https://developer.android.com/guide/topics/providers/content-provider-basics

# What can we do with a content provider? (operations)



The primary methods that need to be implemented are:

- onCreate() which is called to initialize the provider
- query(Uri, String[], String, String[], String) **which returns data to the caller**
- insert(Uri, ContentValues) which inserts new data into the content provider
- update(Uri, ContentValues, String, String[]) which updates existing data in the content provider
- delete(Uri, String, String[]) which deletes data from the content provider
- getType(Uri) which returns the MIME type of data in the content provider

https://developer.android.com/guide/topics/providers/content-provider-basics

# Getting data via query()

To retrieve data from a provider, follow these basic steps:

1. Request the read access permission for the provider.
2. Define the code that sends a query to the provider.

## 1. Requesting read access permission

To retrieve data from a provider, your application needs "read access permission" for the provider. You can't request this permission at run-time; instead, you have to specify that you need this permission in your manifest, using the **<uses-permission>** element and the exact permission name defined by the provider.

E.g. The User Dictionary Provider defines the permission android.permission.READ_USER_DICTIONARY in its manifest file, so an application that wants to read from the provider must request this permission.

```
<uses-permission android:name="android.permission.READ_USER_DICTIONARY">
```

https://developer.android.com/guide/topics/providers/content-provider-basics#Permissions

## Getting data via query()

**query(Uri,projection,selection,selectionArgs,sortOrder)**

### 2. Define the code that sends a query to the provider

The next step in retrieving data from a provider is to construct a query. This first snippet defines some variables for accessing the User Dictionary Provider:

```
// A "projection" defines the columns that will be returned for each row
String[] mProjection =
{
    UserDictionary.Words._ID,    // Contract class constant for the _ID column name
    UserDictionary.Words.WORD,   // Contract class constant for the word column name
    UserDictionary.Words.LOCALE  // Contract class constant for the locale column name
};

// Defines a string to contain the selection clause
String selectionClause = null;

// Initializes an array to contain selection arguments
String[] selectionArgs = {""};
```

https://developer.android.com/guide/topics/providers/content-provider-basics

## Getting data via query()

**query(Uri,projection,selection,selectionArgs,sortOrder)**

query(..) returns a cursor, representing a result set

| query() argument | SELECT keyword/parameter | Notes |
|---|---|---|
| Uri | FROM table_name | Uri maps to the table in the provider named table_name. ***content://authority/optionalPath/optionalId*** |
| projection | col,col,col,... | projection is an array of columns that should be included for each row retrieved. |
| selection | WHERE col = value | selection specifies the criteria for selecting rows. |
| selectionArgs | (No exact equivalent. Selection arguments replace ? placeholders in the selection clause.) | |
| sortOrder | ORDER BY col,col,... | sortOrder specifies the order in which rows appear in the returned Cursor. |

https://developer.android.com/guide/topics/providers/content-provider-basics

## Displaying query results

The ContentResolver.query() client method always returns a Cursor containing the columns specified by the query's projection for the rows that match the query's selection criteria.

A Cursor object provides random read access to the rows and columns it contains. Using Cursor methods, you can iterate over the rows in the results, determine the data type of each column, get the data out of a column, and examine other properties of the results.

- If no rows match the selection criteria, the provider returns a Cursor object for which **Cursor.getCount() is 0** (an empty cursor).
- If an internal error occurs, the results of the query depend on the particular provider. It may choose to return **null**, or it may throw an Exception.
- Since a Cursor is a "list" of rows, a good way to display the contents of a Cursor is to link it to a ListView via a SimpleCursorAdapter.

https://developer.android.com/guide/topics/providers/content-provider-basics

https://developer.android.com/reference/android/database/Cursor

# Displaying query results

```
// Defines a list of columns to retrieve from the Cursor and load into an output row
String[] wordListColumns =
{
    UserDictionary.Words.WORD,   // Contract class constant containing the word column name
    UserDictionary.Words.LOCALE  // Contract class constant containing the locale column name
};

// Defines a list of View IDs that will receive the Cursor columns for each row
int[] wordListItems = { R.id.dictWord, R.id.locale};

// Creates a new SimpleCursorAdapter
cursorAdapter = new SimpleCursorAdapter(
    getApplicationContext(),               // The application's Context object
    R.layout.wordlistrow,                  // A layout in XML for one row in the ListView
    mCursor,                               // The result from the query
    wordListColumns,                       // A string array of column names in the cursor
    wordListItems,                         // An integer array of view IDs in the row layout
    0);                                    // Flags (usually none are needed)

// Sets the adapter for the ListView
wordList.setAdapter(cursorAdapter);
```

## the method insert(..)

```
// Defines a new Uri object that receives the result of the insertion
Uri newUri;
...

// Defines an object to contain the new values to insert
ContentValues newValues = new ContentValues();

/*
 * Sets the values of each column and inserts the word. The arguments to the "put"
 * method are "column name" and "value"
 */
newValues.put(UserDictionary.Words.APP_ID, "example.user");
newValues.put(UserDictionary.Words.LOCALE, "en_US");
newValues.put(UserDictionary.Words.WORD, "insert");
newValues.put(UserDictionary.Words.FREQUENCY, "100");

newUri = getContentResolver().insert(
    UserDictionary.Words.CONTENT_URI,   // the user dictionary content URI
    newValues                          // the values to insert
```

The content URI returned in newUri identifies the newly-added row, with the following format:

```
content://user_dictionary/words/<id_value>
```

To get the value of _ID from the returned Uri, call ContentUris.parseId().

# the method update(..)

To update a row, you use a ContentValues object with the updated values just as you do with an insertion, and selection criteria just as you do with a query.

- The client method you use is ContentResolver.update().
- You only need to add values to the ContentValues object for columns you're updating.
- If you want to clear the contents of a column, set the value to null.
- You should also sanitize user input when you call ContentResolver.update().

```
// Defines an object to contain the updated values
ContentValues updateValues = new ContentValues();

// Defines selection criteria for the rows you want to update
String selectionClause = UserDictionary.Words.LOCALE +  " LIKE ?";
String[] selectionArgs = {"en_%"};

// Defines a variable to contain the number of updated rows
int rowsUpdated = 0;
...
/*
 * Sets the updated value and updates the selected words.
 */
updateValues.putNull(UserDictionary.Words.LOCALE);

rowsUpdated = getContentResolver().update(
    UserDictionary.Words.CONTENT_URI,   // the user dictionary content URI
    updateValues,                      // the columns to update
    selectionClause,                   // the column to select on
    selectionArgs                      // the value to compare to
);
```

## the method delete(..)

Deleting rows is similar to retrieving row data.

- Specify selection criteria for the rows you want to delete
- The client method returns the number of deleted rows
- You should also sanitize user input when you call ContentResolver.delete().

```
// Defines selection criteria for the rows you want to delete
String selectionClause = UserDictionary.Words.APP_ID + " LIKE ?";
String[] selectionArgs = {"user"};

// Defines a variable to contain the number of rows deleted
int rowsDeleted = 0;

...

// Deletes the words that match the selection criteria
rowsDeleted = getContentResolver().delete(
    UserDictionary.Words.CONTENT_URI,   // the user dictionary content URI
    selectionClause,                    // the column to select on
    selectionArgs                       // the value to compare to
);
```

https://developer.android.com/guide/topics/providers/content-provider-basics#Injection

## Calendar example

https://developer.android.com/guide/topics/providers/calendar-provider

# Create a Content Provider

Suppose you have or want to create data store, storing rows  of data, files (e.g., images), or heterogenous data objects (BLOBS), and want to make that available for access, then?

- Create a class that implements ContentProvider
- Implement required methods
- Amend manifest

| | |
|---|---|
| query() | Retrieve data from your provider. Use the arguments to select the table to query, the rows and columns to return, and the sort order of the result. Return the data as a Cursor object. |
| insert() | Insert a new row into your provider. Use the arguments to select the destination table and to get the column values to use. Return a content URI for the newly-inserted row. |
| update() | Update existing rows in your provider. Use the arguments to select the table and rows to update and to get the updated column values. Return the number of rows updated. |
| delete() | Delete rows from your provider. Use the arguments to select the table and the rows to delete. Return the number of rows deleted. |
| getType() | Return the MIME type corresponding to a content URI. This method is described in more detail in the section Implementing content provider MIME types. |
| onCreate() | Initialize your provider. The Android system calls this method immediately after it creates your provider. Notice that your provider is not created until a ContentResolver object tries to access it. |

https://developer.android.com/guide/topics/providers/content-provider-creating

## Demo

```
public class ExampleProvider extends ContentProvider

    // Defines a handle to the Room database
    private AppDatabase appDatabase;

    // Defines a Data Access Object to perform the database operations
    private UserDao userDao;

    // Defines the database name
    private static final String DBNAME = "mydb";

    public boolean onCreate() {

        // Creates a new database object.
        appDatabase = Room.databaseBuilder(getContext(), AppDatabase.class, DBNAME).build();

        // Gets a Data Access Object to perform the database operations
        userDao = appDatabase.getUserDao();

        return true;
    }
…
    // Implements the provider's insert method
    public Cursor insert(Uri uri, ContentValues values) {
        // Insert code here to determine which DAO to use when inserting data, handle error conditions, etc.
    }
}
```

https://developer.android.com/guide/topics/providers/content-provider-creating

# Storing your own persistent data on an Android Device

- App-specific storage: Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage. Use the directories within internal storage to save sensitive information that other apps shouldn't access.
- Shared storage: Store files that your app intends to share with other apps, including media, documents, and other files.
- Preferences: Store private, primitive data in key-value pairs.
- Databases: Store structured data in a private database using the Room persistence library.

https://developer.android.com/training/data-storage#pref

# Storing your own persistent data on an Android Device

| | Type of content | Access method | Permissions needed | Can other apps access? | Files removed on app uninstall? |
|---|---|---|---|---|---|
| App-specific files | Files meant for your app's use only | From internal storage, getFilesDir() or getCacheDir()<br><br>From external storage, getExternalFilesDir() or getExternalCacheDir() | Never needed for internal storage<br><br>Not needed for external storage when your app is used on devices that run Android 4.4 (API level 19) or higher | No, if files are in a directory within internal storage<br><br>Yes, if files are in a directory within external storage | Yes |
| Media | Shareable media files (images, audio files, videos) | MediaStore API | READ_EXTERNAL_STORAGE or WRITE_EXTERNAL_STORAGE when accessing other apps' files on Android 10 (API level 29) or higher<br><br>Permissions are required for all files on Android 9 (API level 28) or lower | Yes, though the other app needs the READ_EXTERNAL_STORAGE permission | No |
| Documents and other files | Other types of shareable content, including downloaded files | Storage Access Framework | None | Yes, through the system file picker | No |
| App preferences | Key-value pairs | Jetpack Preferences library | None | No | Yes |
| Database | Structured data | Room persistence library | None | No | Yes |

https://developer.android.com/training/data-storage

## Shared Preferences

- Information stored as a set of attributes (or keys) and values (basic types such as ints, floats, strings, etc)
- Default preference values can be stored in an XML document
- Shared preference key-value pairs stored in an XML file in the application’s “data” subdirectory

https://developer.android.com/guide/topics/ui/settings/use-saved-values#sharedpreferences

## Write shared preferences

```
// Storing data into SharedPreferences

SharedPreferences sharedPreferences = getSharedPreferences("MySharedPref", MODE_PRIVATE);

// Creating an Editor object  to edit(write to the file)

SharedPreferences.Editor myEdit  = sharedPreferences.edit();

// Storing the key and its value as the data fetched from edittext

myEdit.putString( "name", name.getText().toString());
myEdit.putInt( "age", Integer.parseInt(age.getText().toString()));

// Once the changes have been made, we need to apply those changes made, otherwise, it will throw an error

myEdit.apply();
```

https://developer.android.com/guide/topics/ui/settings/use-saved-values#sharedpreferences

## Read shared preferences

```
// Retrieving the value using its keys the file name must be same in both saving and retrieving the data

SharedPreferences sh
                = getSharedPreferences("MySharedPref", MODE_PRIVATE);

// The value will be default as empty string because for the very first time when the app is opened,there is nothing to show

String s1 = sh.getString("name", "");
int a = sh.getInt("age", 0);

// We can then use the data

name.setText(s1);
age.setText(String.valueOf(a));
```

https://developer.android.com/guide/topics/ui/settings/use-saved-values#sharedpreferences

## Shared preferences example

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture6_SharedPrefExample

## App Specific Storage - Internal

Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage. Use the directories within internal storage to save sensitive information that other apps shouldn't access.

You can use the File API to access and store files:

```
File file = new File(context.getFilesDir(), filename);
```

https://developer.android.com/training/data-storage/app-specific

## App Specific Storage - External

If internal storage doesn't provide enough space to store app-specific files, consider using external storage instead. The system provides directories within external storage where an app can organize files that provide value to the user only within your app.

- One directory is designed for your app's persistent files, and another contains your app's cached files.
- The files stored in these directories are removed when your app is uninstalled.

Caution: The files in these directories aren't guaranteed to be accessible, such as when a removable SD card is taken out of the device. If your app's functionality depends on these files, you should instead store the files within internal storage.

https://developer.android.com/training/data-storage/app-specific

# App Specific Storage - External

To access the different locations, call **ContextCompat.getExternalFilesDirs().** As shown in the code snippet, the first element in the returned array is considered the primary external storage volume. Use this volume unless it's full or unavailable.

```
File[] externalStorageVolumes =
        ContextCompat.getExternalFilesDirs(getApplicationContext(), null);
File primaryExternalStorage = externalStorageVolumes[0];
```

To access app-specific files from external storage, call getExternalFilesDir(), as shown in the following code snippet:

```
File appSpecificExternalDir = new File(context.getExternalFilesDir(), filename);
```

https://developer.android.com/training/data-storage/app-specific#external

## SQLite

Saving data to a database is ideal for repeating or structured data, such as contact information.

- Key classes: SQLiteOpenHelper and SQLiteDatabase

```
DatabaseHelper dbh;
SQLiteDatabase db = dbh.getWritableDatabase();

db.insert(...);

db.query(...);

db.delete(...)
```

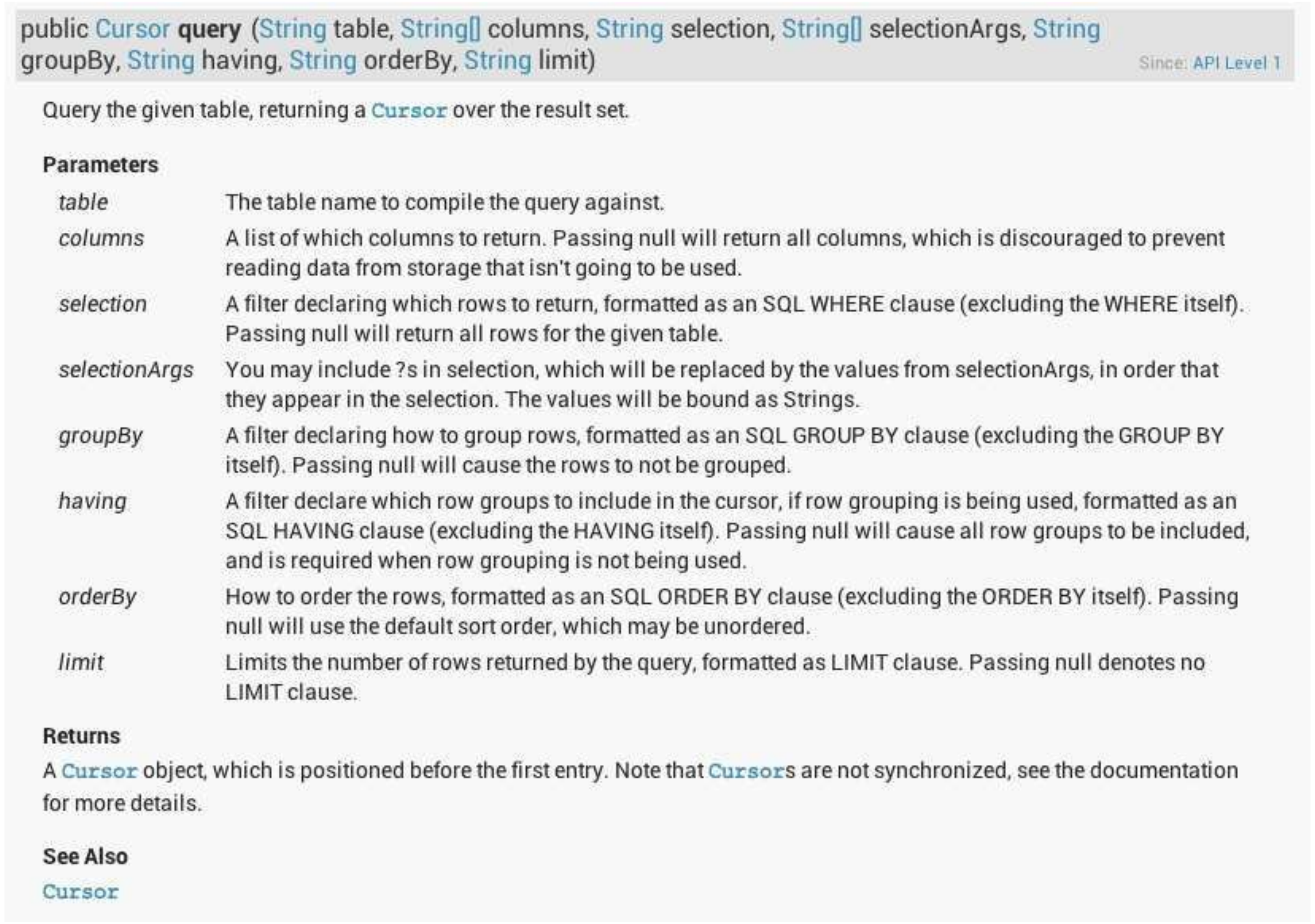
https://developer.android.com/training/data-storage/sqlite

# Query database

```
SQLiteDatabase db = dbHelper.getReadableDatabase();

// Define a projection that specifies which columns from the database
// you will actually use after this query.
String[] projection = {
    BaseColumns._ID,
    FeedEntry.COLUMN_NAME_TITLE,
    FeedEntry.COLUMN_NAME_SUBTITLE
    };

// Filter results WHERE "title" = 'My Title'
String selection = FeedEntry.COLUMN_NAME_TITLE + " = ?";
String[] selectionArgs = { "My Title" };

// How you want the results sorted in the resulting Cursor
String sortOrder =
    FeedEntry.COLUMN_NAME_SUBTITLE + " DESC";

Cursor cursor = db.query(
    FeedEntry.TABLE_NAME,   // The table to query
    projection,             // The array of columns to return (pass null to get all)
    selection,              // The columns for the WHERE clause
    selectionArgs,          // The values for the WHERE clause
    null,                   // don't group the rows
    null,                   // don't filter by row groups
    sortOrder               // The sort order
    );
```

https://developer.android.com/training/data-storage/sqlite

# Query database



public Cursor **query** (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)

Since: API Level 1

Query the given table, returning a `Cursor` over the result set.

**Parameters**

| | |
|---|---|
| *table* | The table name to compile the query against. |
| *columns* | A list of which columns to return. Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used. |
| *selection* | A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given table. |
| *selectionArgs* | You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings. |
| *groupBy* | A filter declaring how to group rows, formatted as an SQL GROUP BY clause (excluding the GROUP BY itself). Passing null will cause the rows to not be grouped. |
| *having* | A filter declare which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause (excluding the HAVING itself). Passing null will cause all row groups to be included, and is required when row grouping is not being used. |
| *orderBy* | How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered. |
| *limit* | Limits the number of rows returned by the query, formatted as LIMIT clause. Passing null denotes no LIMIT clause. |

**Returns**

A `Cursor` object, which is positioned before the first entry. Note that `Cursor`s are not synchronized, see the documentation for more details.

**See Also**

`Cursor`

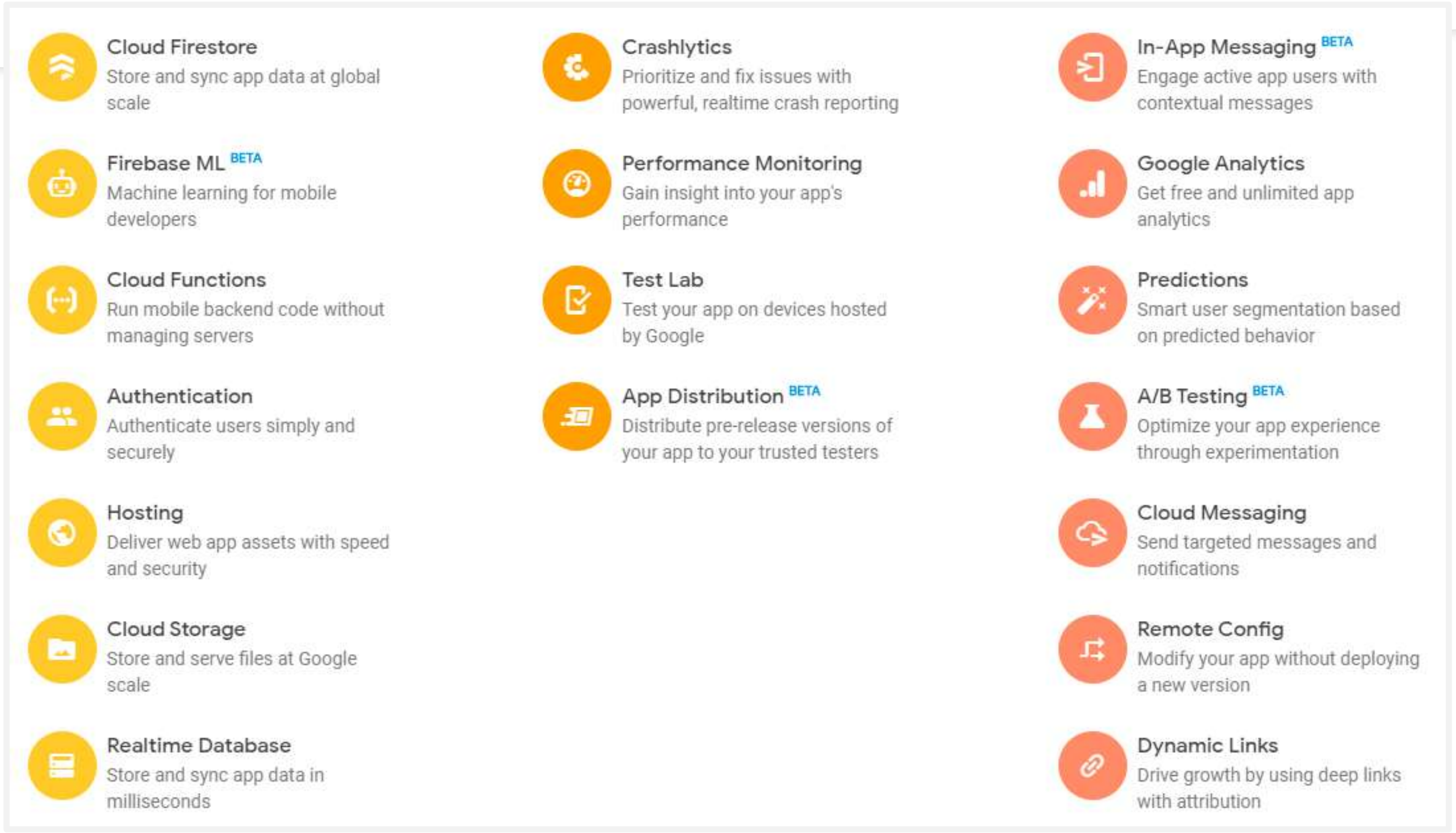
https://developer.android.com/training/data-storage/sqlite

# SQLite example

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture6_SQLiteDemo

# Firebase

- Mobile and web application platform to handle back-end application support (i.e. server side)
- Owned by Google since 2014



Cloud Firestore
Store and sync app data at global scale

Firebase ML BETA
Machine learning for mobile developers

Cloud Functions
Run mobile backend code without managing servers

Authentication
Authenticate users simply and securely

Hosting
Deliver web app assets with speed and security

Cloud Storage
Store and serve files at Google scale

Realtime Database
Store and sync app data in milliseconds

Crashlytics
Prioritize and fix issues with powerful, realtime crash reporting

Performance Monitoring
Gain insight into your app's performance

Test Lab
Test your app on devices hosted by Google

App Distribution BETA
Distribute pre-release versions of your app to your trusted testers

In-App Messaging BETA
Engage active app users with contextual messages

Google Analytics
Get free and unlimited app analytics

Predictions
Smart user segmentation based on predicted behavior

A/B Testing BETA
Optimize your app experience through experimentation

Cloud Messaging
Send targeted messages and notifications

Remote Config
Modify your app without deploying a new version

Dynamic Links
Drive growth by using deep links with attribution

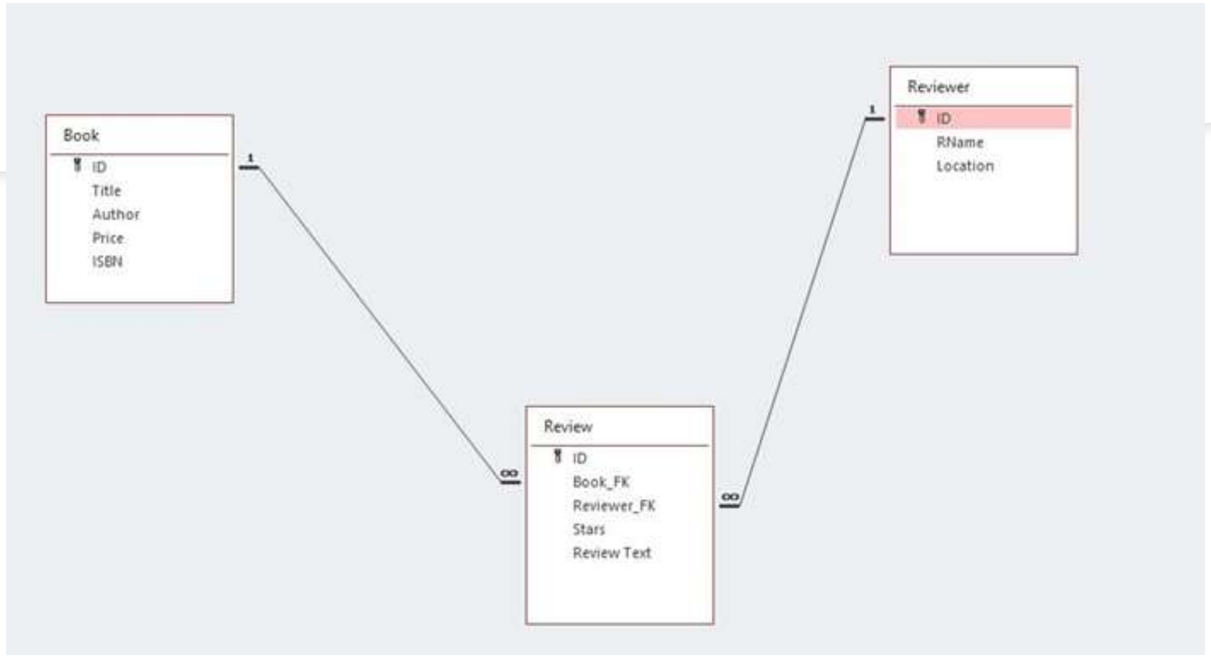
https://firebase.google.com/

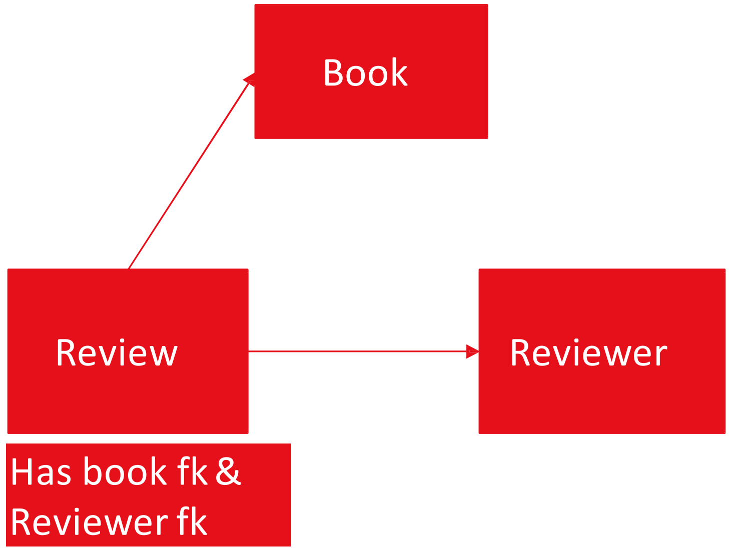
## Firestore

- Firestore is a flexible, scalable, realtime database. It's simple enough for rapid prototyping yet scalable and flexible enough to grow with you to any size.
- Firestore is a realtime database, meaning that clients can listen to data in Firestore and be notified in real time as it changes. This feature lets you build responsive apps that work regardless of network latency or Internet connectivity.
- Firestore is a cloud-hosted NoSQL database. You store data in documents, which contain fields mapping to values. These documents are stored in collections, which are simply containers that help you organize and query your documents. You can learn more about the data model in the Structuring Data documentation.

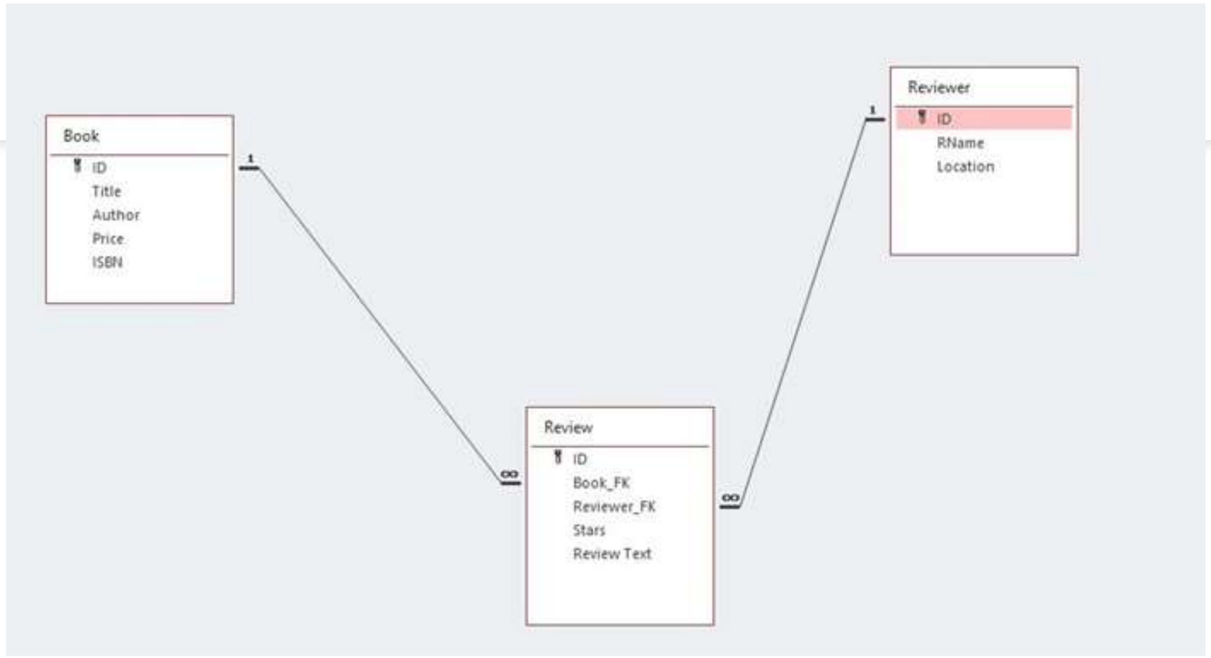
# SQL Databases

- In SQL, an object is represented as a table
- Object association is done by adding foreign keys to the table to reference the other object
- SQL has a schema
  - Rules for the table { columns and their data types }
- E.g. for a book retailer
  - A book has a review supplied by a reviewer





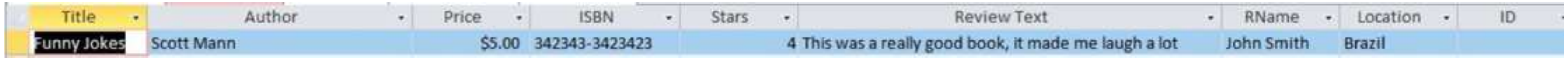
# SQL Way



SELECT Book.Title, Book.Author, Book.Price, Book.ISBN, Review.Stars, Review.[Review Text],  Reviewer.RName, Reviewer.Location, Book.ID

FROM Reviewer INNER JOIN (Book INNER JOIN Review ON Book.ID = Review.Book_FK) ON  Reviewer.ID = Review.Reviewer_FK

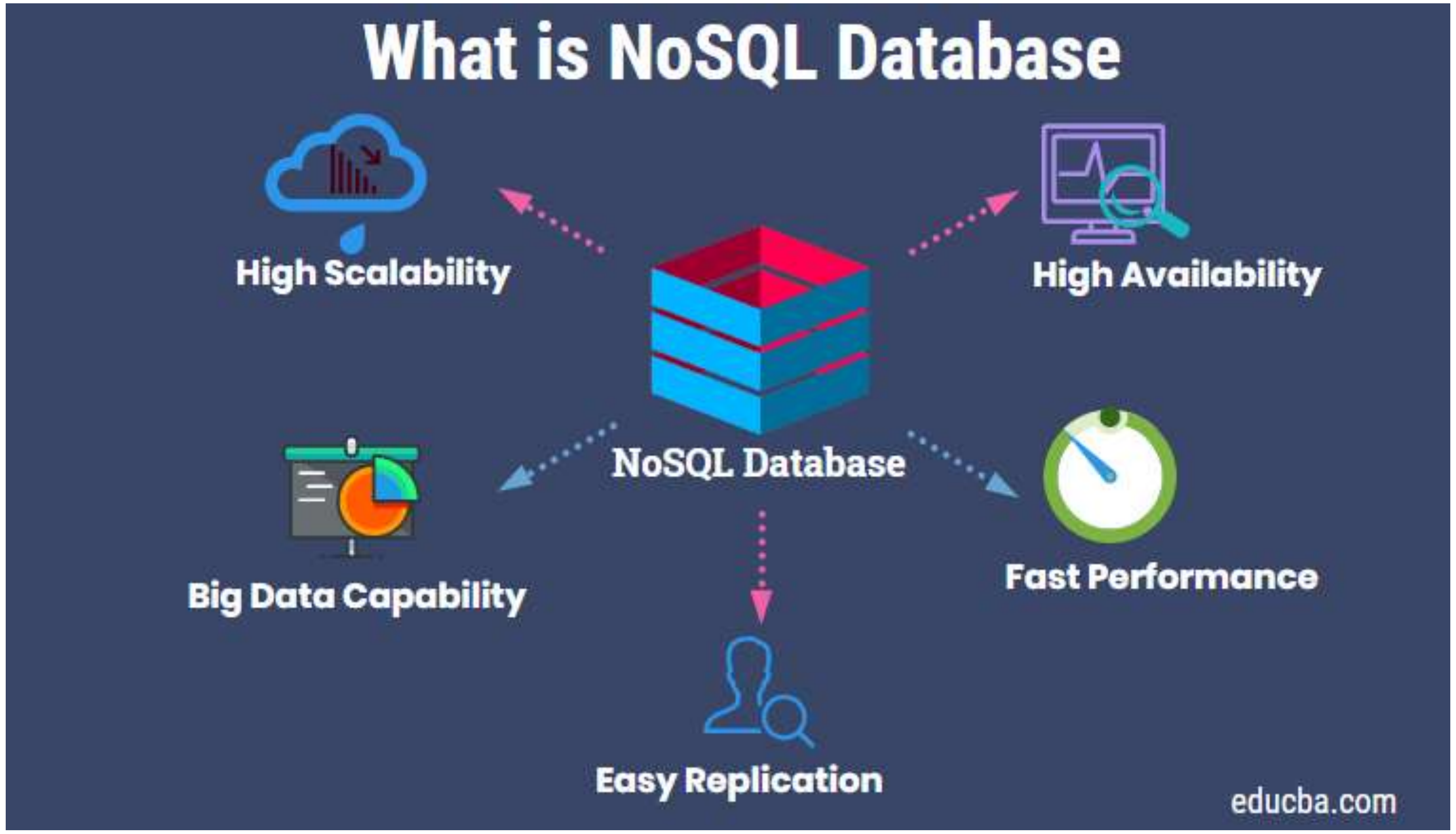
WHERE (((Book.ID)=1));



| Title | Author | Price | ISBN | Stars | Review Text | RName | Location | ID |
|---|---|---|---|---|---|---|---|---|
| Funny Jokes | Scott Mann | $5.00 | 342343-3423423 | 4 | This was a really good book, it made me laugh a lot | John Smith | Brazil | |

## Why NoSQL?

- Writes require data duplication
- Reads are really fast
- Most apps read more than they write
- NoSQL is a valid choice
- Optimised for read heavy applications



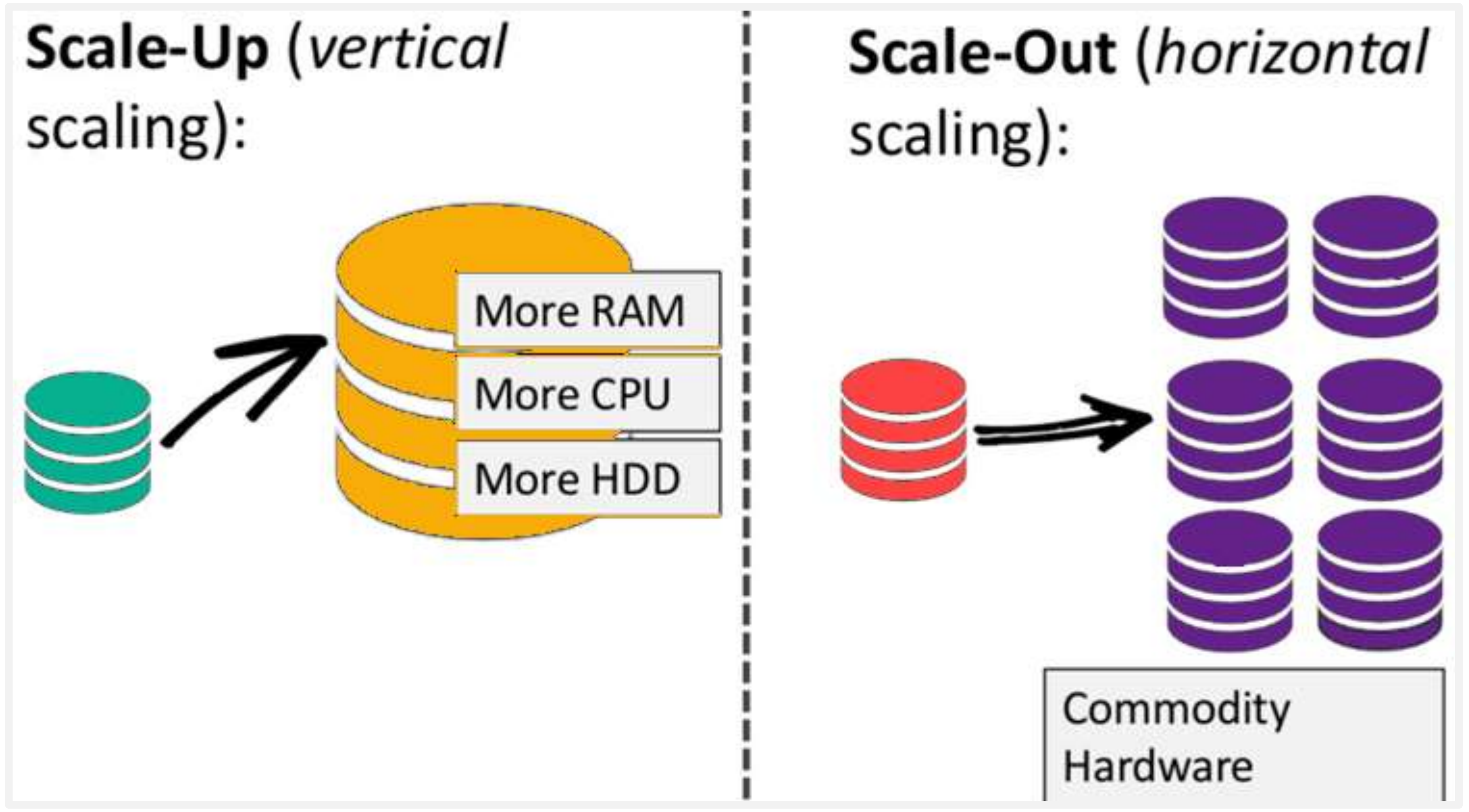
https://www.educba.com/what-is-nosql-database/

## Horizontally Scaling

- NoSQL data is easily distributed across multiple machines

- SQL requires vertical scaling
  - ➢ Use a higher performance device (1) to host the DB

- NoSQL scales horizontally
  - ➢ Data is distributed across several machines (many) to host the database
  - ➢ Cloud model for scalability e.g. Amazon Web Services (AWS)



https://www.guru99.com/nosql-tutorial.html

# Firestore

- Not table based (No Tables/Rows) you store data in **documents**, which are organized into **collections**.
- Each **document** contains a set of **key-value pairs**
- Nested Tree
- Collection of JSON objects
- No Schema
- DB won’t restrict the data types
- Updating database design is much easier
- Add/Edit fields without breaking the database

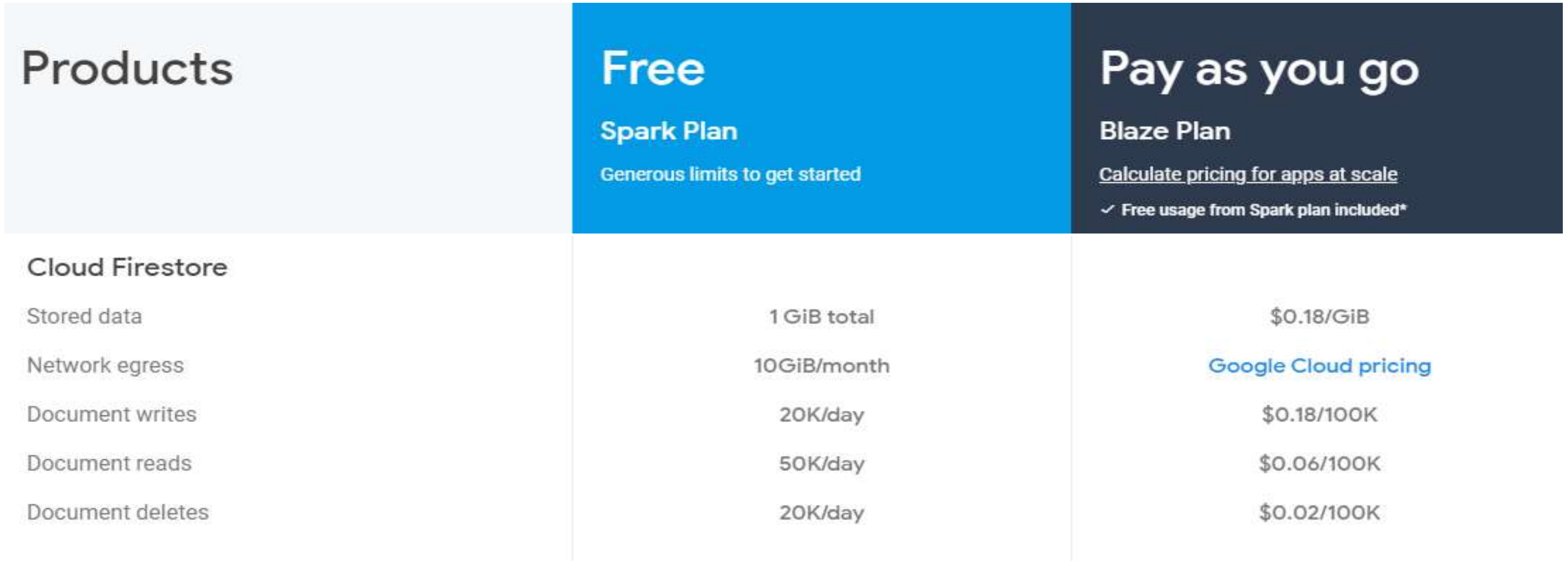
- Add a new field to Book for ‘Publisher’ and won’t expect previous entries to be forced to have this data

Cloud Firestore

https://cloud.google.com/firestore/docs/data-model

## Pricing

https://firebase.google.com/pricing/



| Products | Free<br>Spark Plan<br>Generous limits to get started | Pay as you go<br>Blaze Plan<br>Calculate pricing for apps at scale<br>✓ Free usage from Spark plan included* |
|---|---|---|
| **Cloud Firestore** | | |
| Stored data | 1 GiB total | $0.18/GiB |
| Network egress | 10GiB/month | Google Cloud pricing |
| Document writes | 20K/day | $0.18/100K |
| Document reads | 50K/day | $0.06/100K |
| Document deletes | 20K/day | $0.02/100K |

Correct as of 23/08/2020 please ensure you read carefully!

Background: https://www.youtube.com/watch?v=v_hR4K4auoQ

# NoSQL & Firestore

- Not table based (No Tables/Rows) you store data in **documents**, which are organized into **collections**.
- Each **document** contains a set of **key-value pairs**
- Nested Tree
- Similar to a Collection of JSON objects
- No Schema
- DB won’t restrict the data types
- Updating database design is much easier
- Add/Edit fields without breaking the database

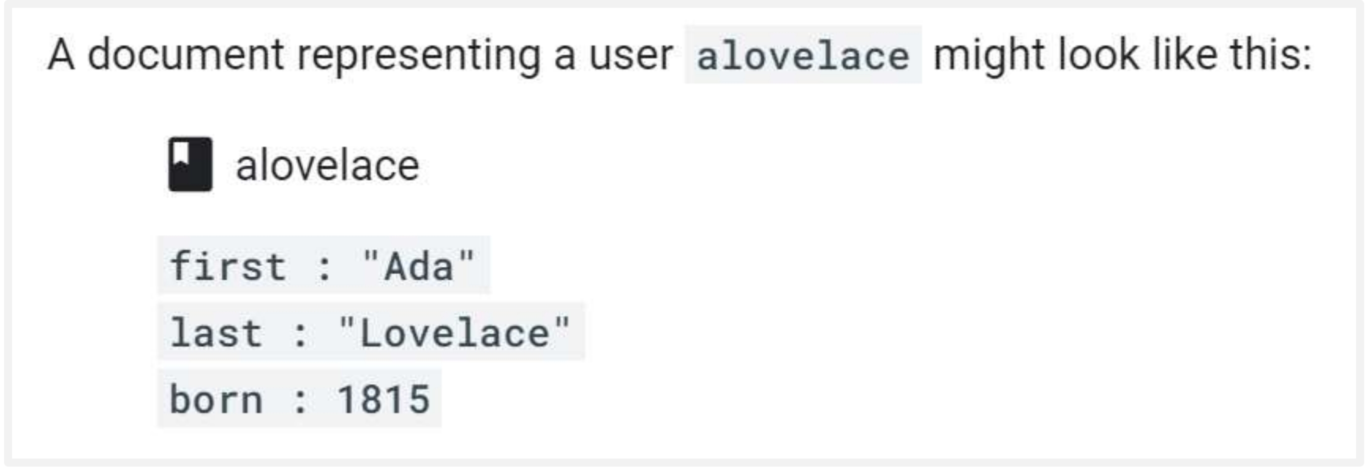
Add a new field to Book for ‘Publisher’ and won’t expect previous entries to be forced to have this data

# 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.
- Can only be <= 1MB in size
- Similar to JSON object
- Key/Value pairs (termed ‘Fields’)
- Document cannot contain a document

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.


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

alovelace

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

JSON (JavaScript Object Notation)

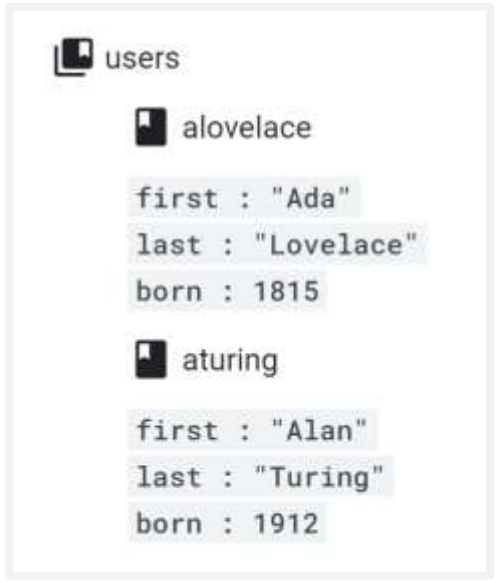
```
{
"employees":[
   {"firstName":"John", "lastName":"Doe"},
   {"firstName":"Anna", "lastName":"Smith"},
   {"firstName":"Peter", "lastName":"Jones"}
]
}
```

https://www.w3schools.com/whatis/whatis_json.asp

https://cloud.google.com/firestore/docs/data-model

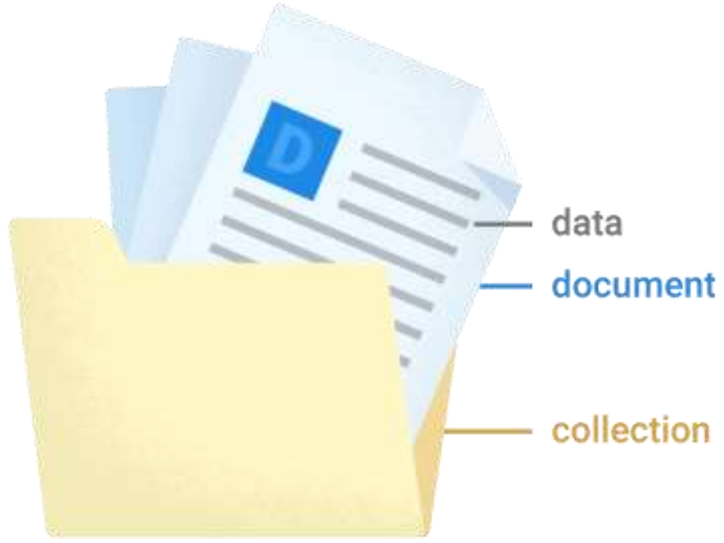
# Collections

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
```

- Can only contain documents
- Firestore root can only contain collections
- The names of documents within a collection are unique
- You do not need to "create" or "delete" collections.



# 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.

```
DocumentReference alovelaceDocumentRef = db.collection("users").document("alovelace");
```

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:

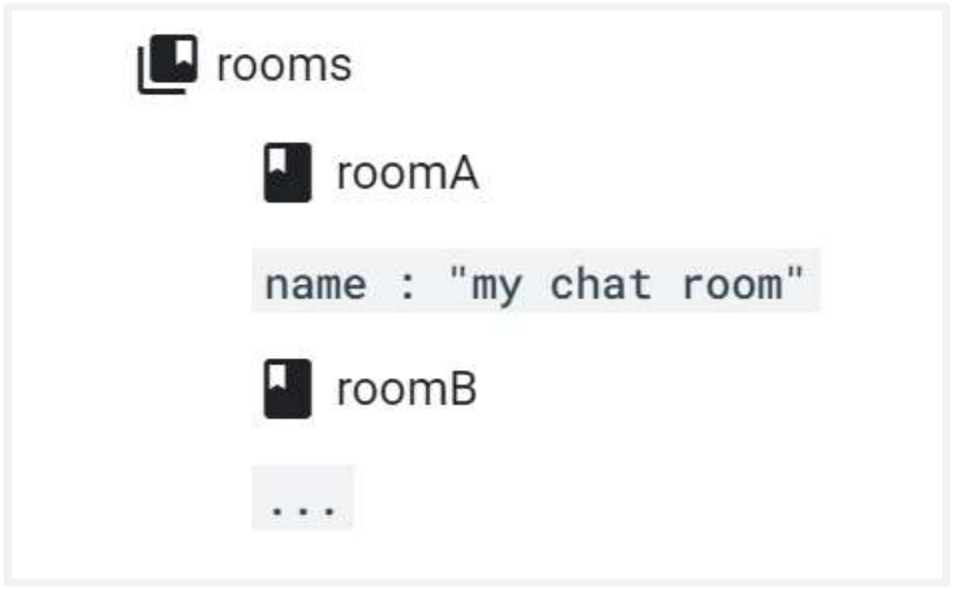
```
CollectionReference usersCollectionRef = db.collection("users");
```

https://cloud.google.com/firestore/docs/data-model#references

## 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

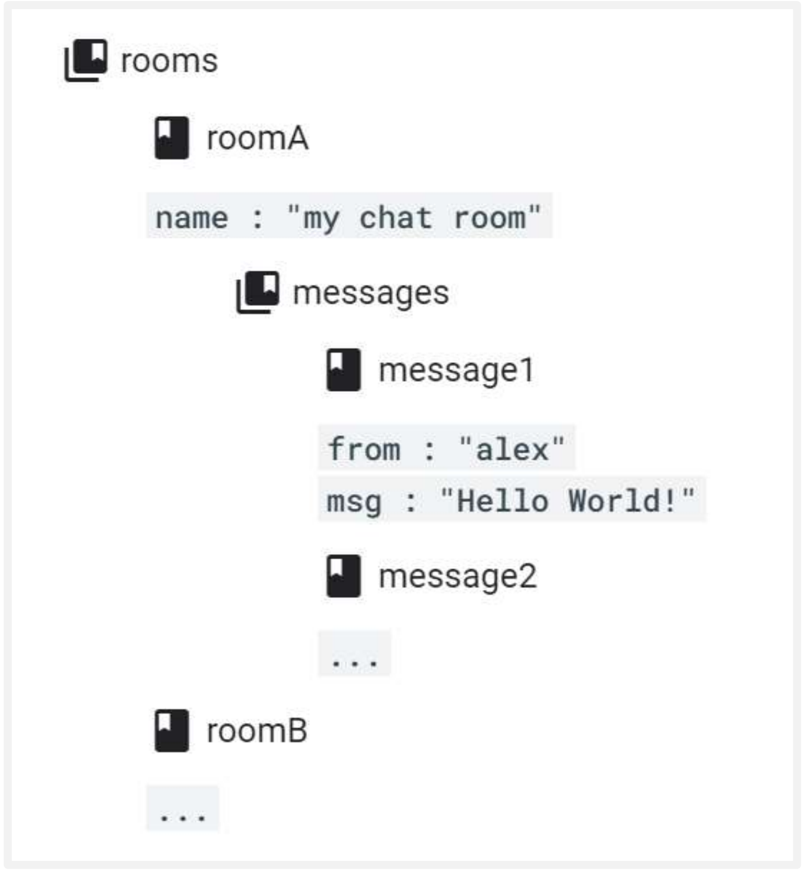
```
...
```

https://cloud.google.com/firestore/docs/data-model#hierarchical-data

# Subcollections

A subcollection is a collection associated with a specific document.

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



```
name : "my chat room"
from : "alex"
msg : "Hello World!"
...
...
```

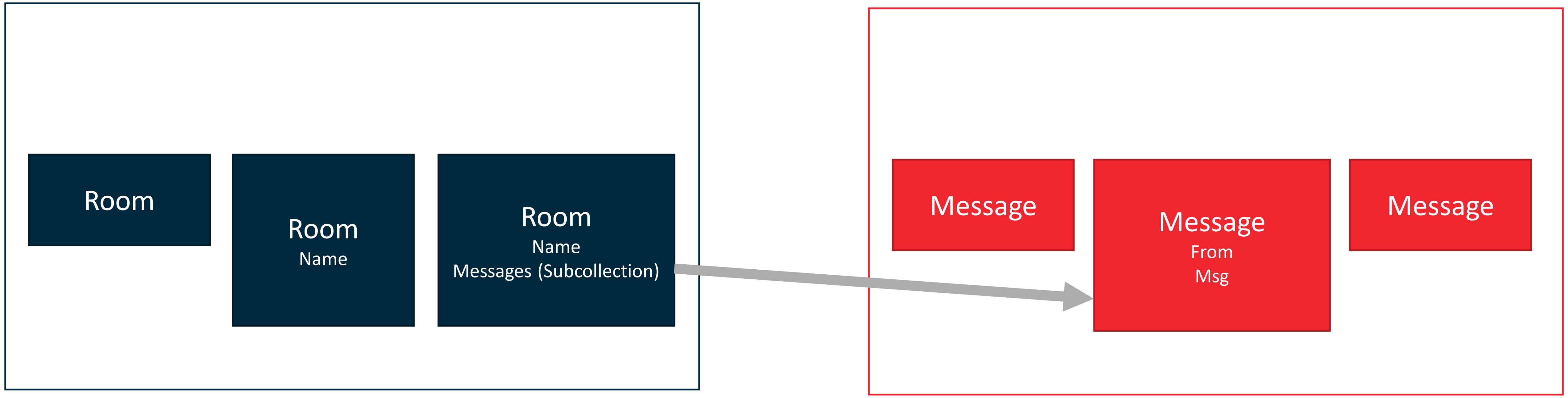
https://cloud.google.com/firestore/docs/data-model#subcollections

## Querying

- Each document in a collection has an index
- Therefore performance is consistent no matter the size of the results
- ALL QUERIES ARE FAST
- Good at equality == > or < inequality type queries
- Composite indexing is supported for AND type clauses

https://firebase.google.com/docs/firestore/query-data/queries

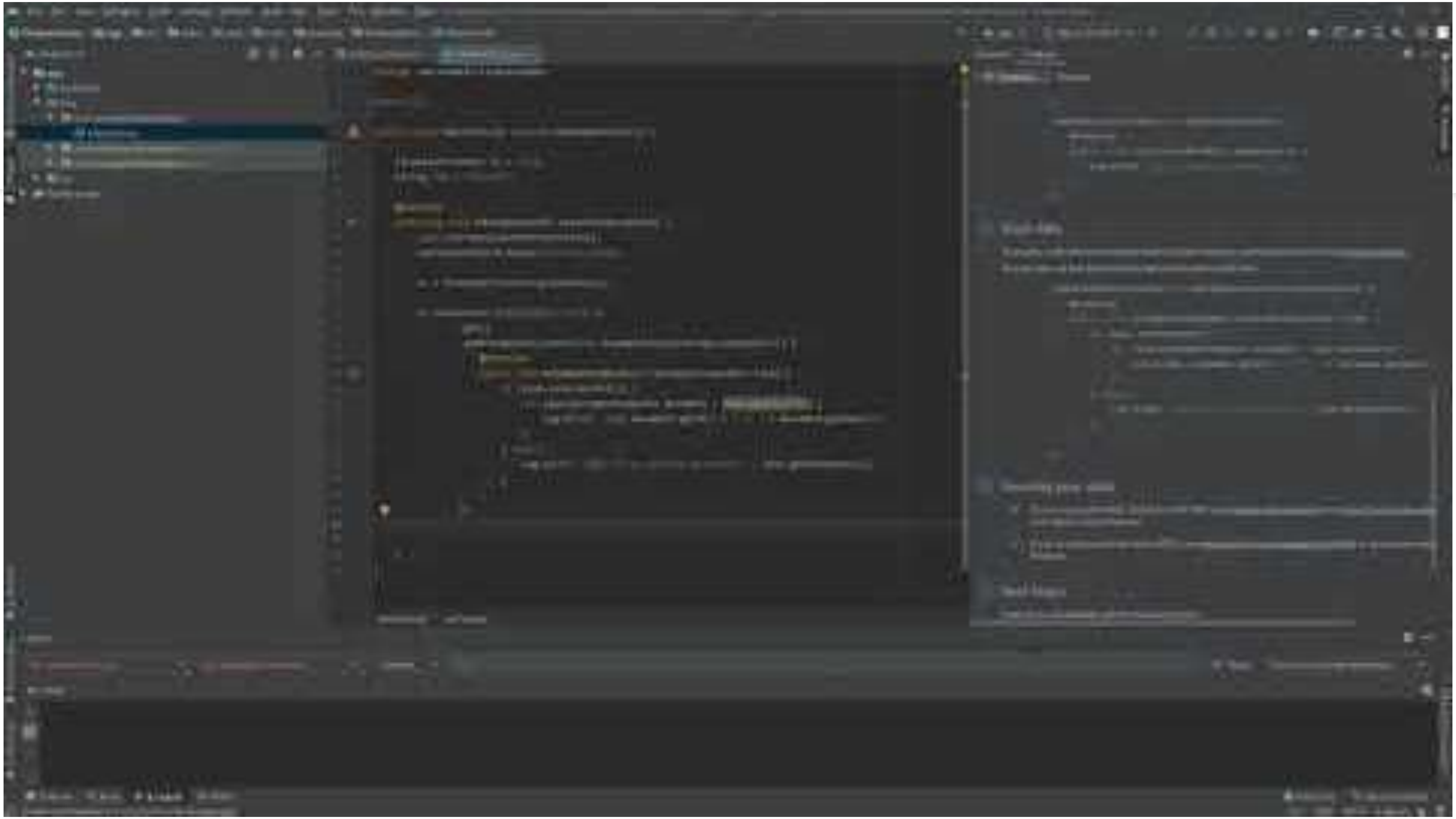
# NoSQL of example



Query for a set of documents in a collection by default (shallow query), e.g. only matching book documents, you don't get the subcollections

# Firestore example

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture6_FirebaseStore

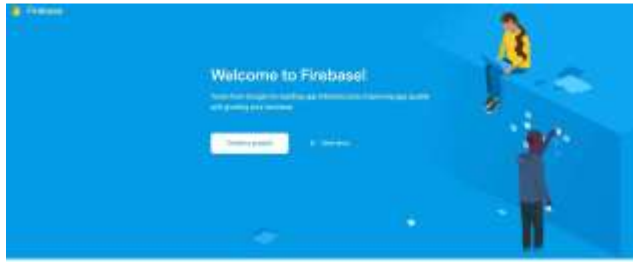


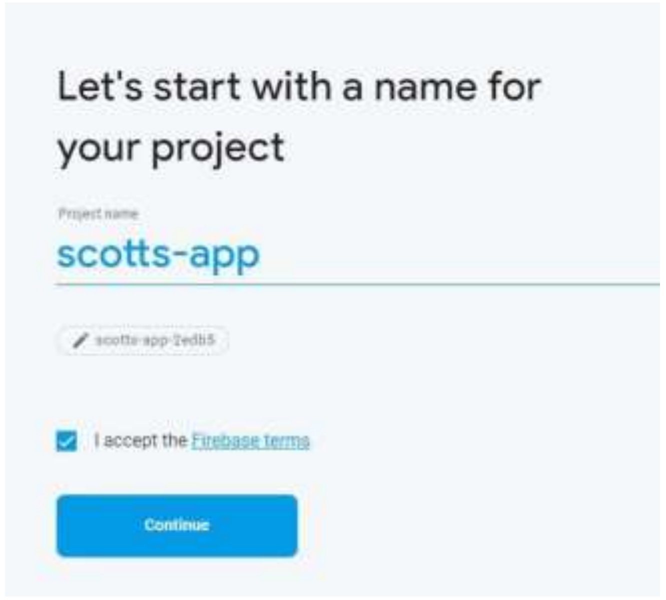
https://www.youtube.com/watch?v=MJHoLVyGoa0

# Adding Firestore to your App

**Create a Firebase project**

https://console.firebase.google.com/



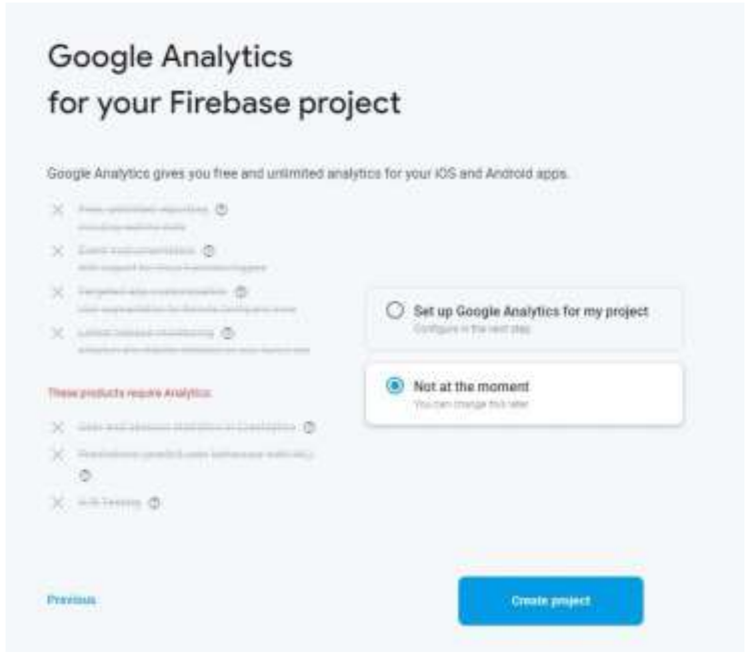


Let's start with a name for your project

Project name
scotts-app

scotts-app-2edb5

[✓] I accept the Firebase terms



Google Analytics for your Firebase project

Google Analytics gives you free and unlimited analytics for your iOS and Android apps.

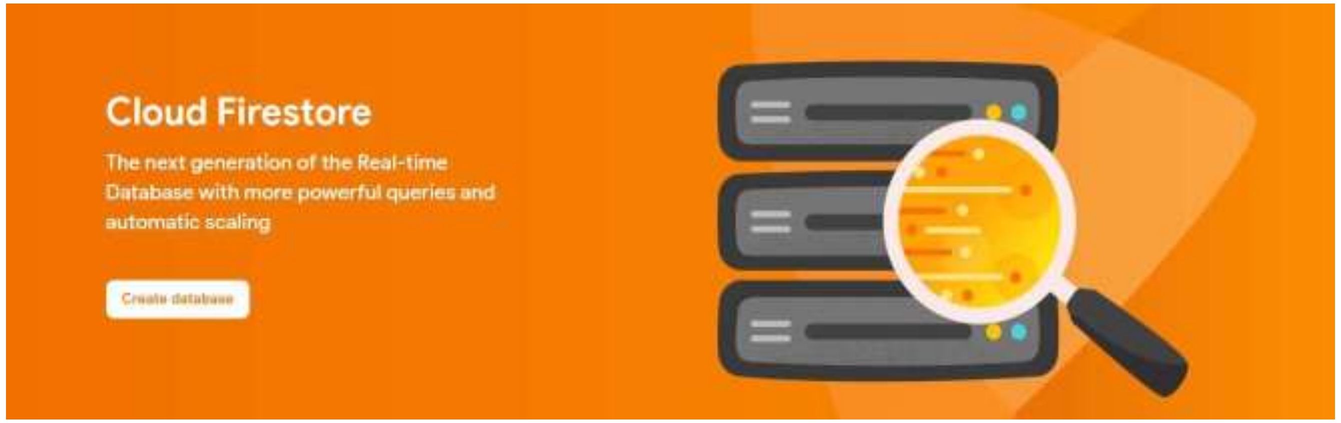
These products require Analytics:

( ) Set up Google Analytics for my project

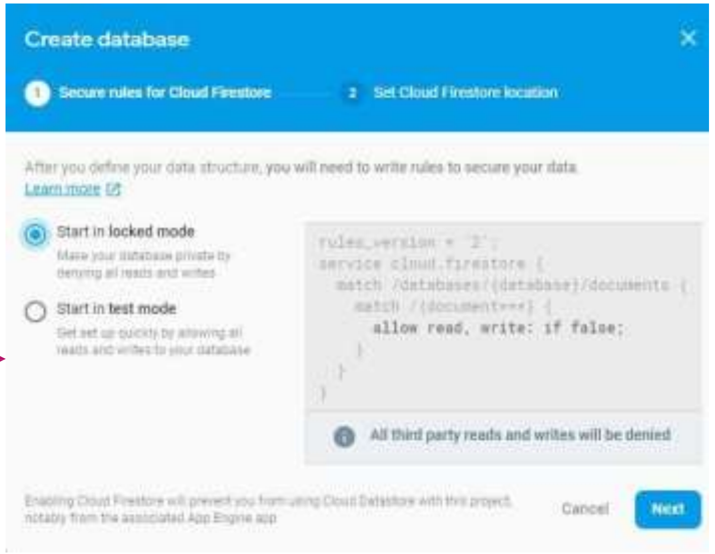
[✓ Not at the moment]

https://firebase.google.com/docs/android/setup#create-firebase-project

# Set up Firestore


Cloud Firestore

The next generation of the Real-time Database with more powerful queries and automatic scaling


Create database

1 Secure rules for Cloud Firestore — 2 Set Cloud Firestore location

After you define your data structure, you will need to write rules to secure your data.
Learn more

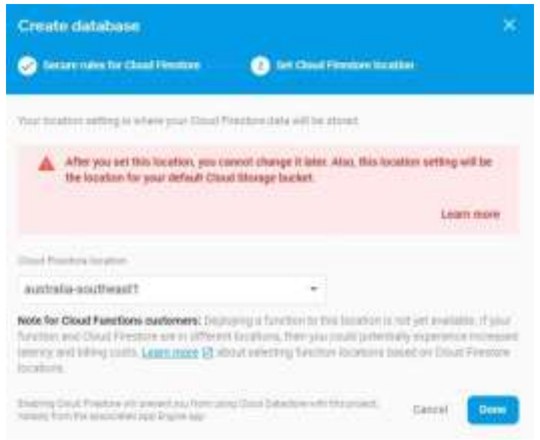
[✓ Start in locked mode]
Make your database private by denying all reads and writes

Start in test mode
Get set up quickly by allowing all reads and writes to your database

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if false;
    }
  }
}
```

All third party reads and writes will be denied

Enabling Cloud Firestore will prevent you from using Cloud Datastore with this project, notably from the associated App Engine app


Create database

✓ Secure rules for Cloud Firestore — 2 Set Cloud Firestore location

Your location setting is where your Cloud Firestore data will be stored.

After you set this location, you cannot change it later. Also, this location setting will be the location for your default Cloud Storage bucket.

Learn more

Cloud Firestore location: australia-southeast1

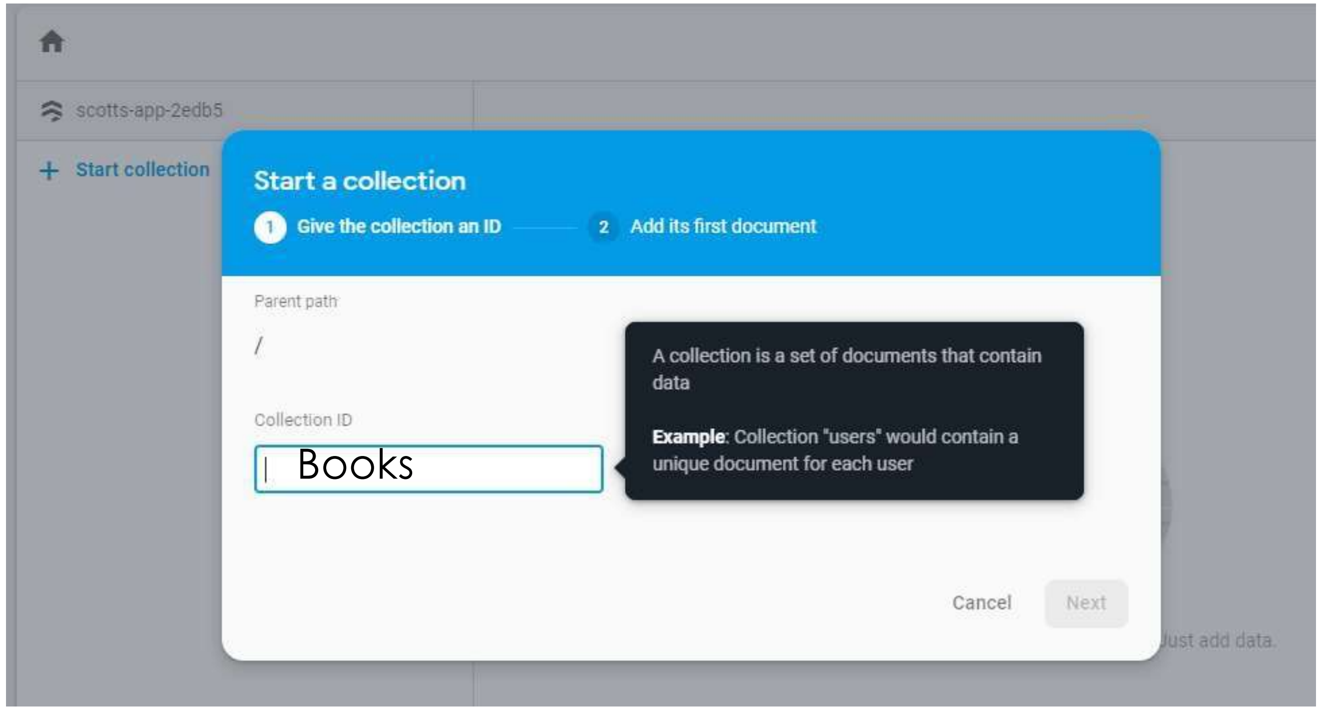
**Note for Cloud Functions customers:** Deploying a function to this location is not yet available. If your function and Cloud Firestore are in different locations, then you could potentially experience increased latency and billing costs. Learn more about selecting function locations based on Cloud Firestore locations.

Enabling Cloud Firestore will prevent you from using Cloud Datastore with this project, notably from the associated App Engine app

Firestore is a NoSQL, horizontally scaling, document model database in the cloud

https://console.firebase.google.com/

## NoSQL Example in Firestore


scotts-app-2edb5

Start collection

Start a collection

1 Give the collection an ID — 2 Add its first document

Parent path

/

Collection ID

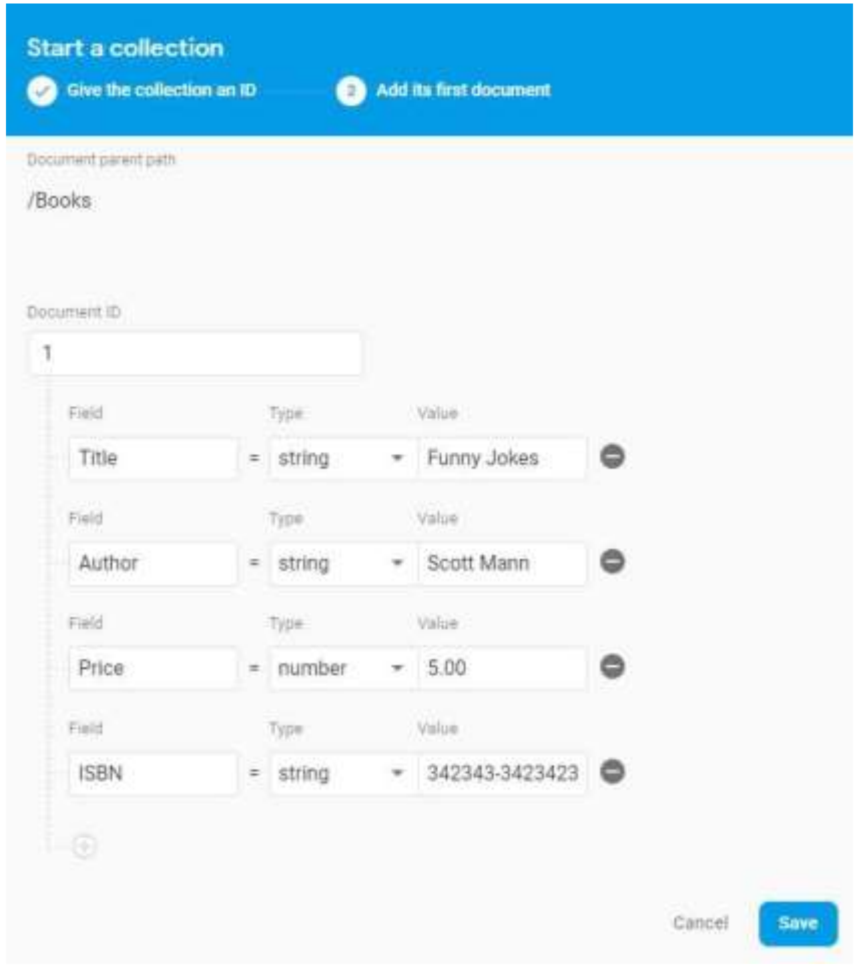
Books

A collection is a set of documents that contain data

**Example**: Collection "users" would contain a unique document for each user

Cancel Next

Just add data.


Start a collection

Give the collection an ID — 2 Add its first document

Document parent path

/Books

Document ID

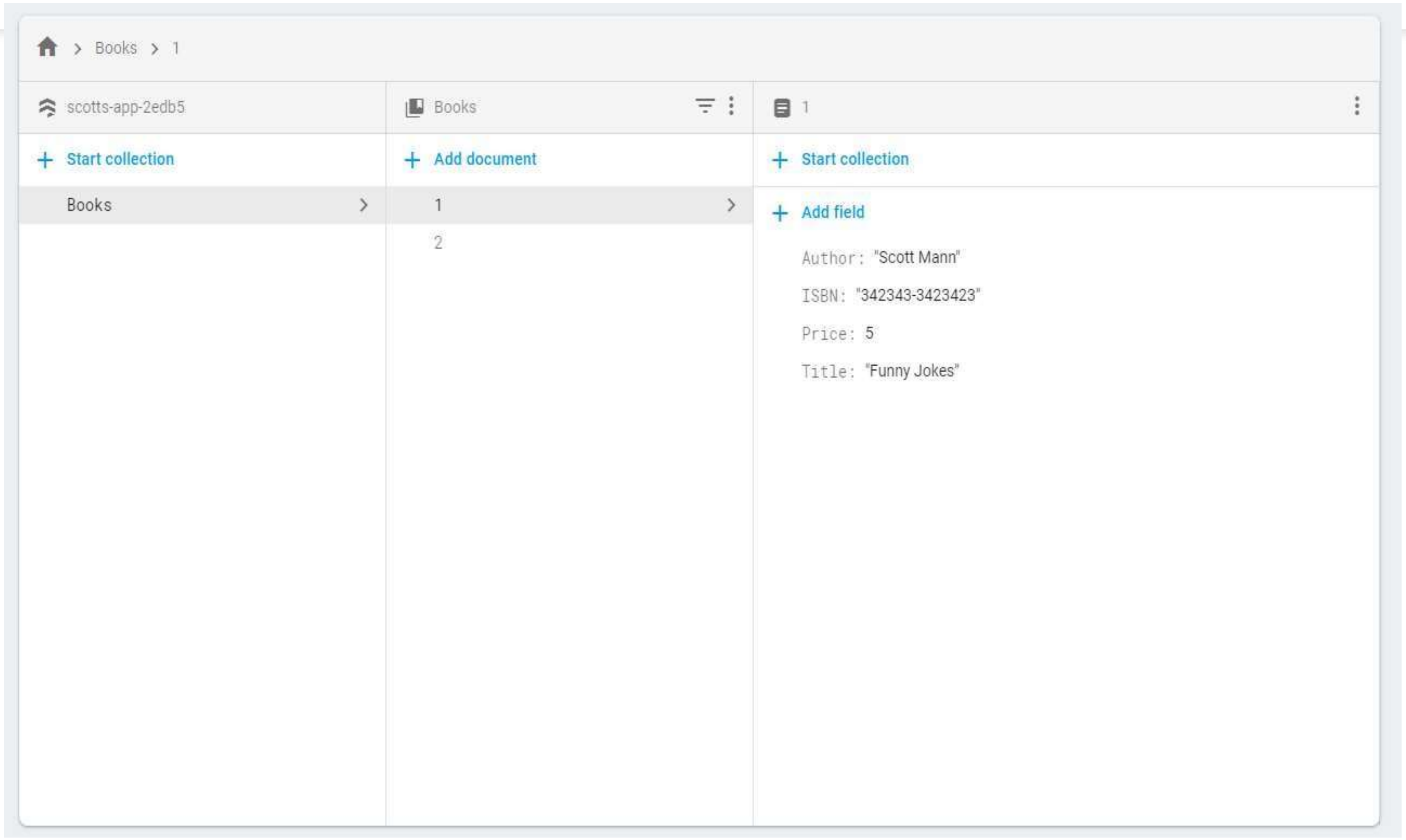
1

| Field | | Type | Value |
|---|---|---|---|
| Title | = | string | Funny Jokes |
| Author | = | string | Scott Mann |
| Price | = | number | 5.00 |
| ISBN | = | string | 342343-3423423 |

Cancel Save

https://console.firebase.google.com/

## Setting up the Book collection



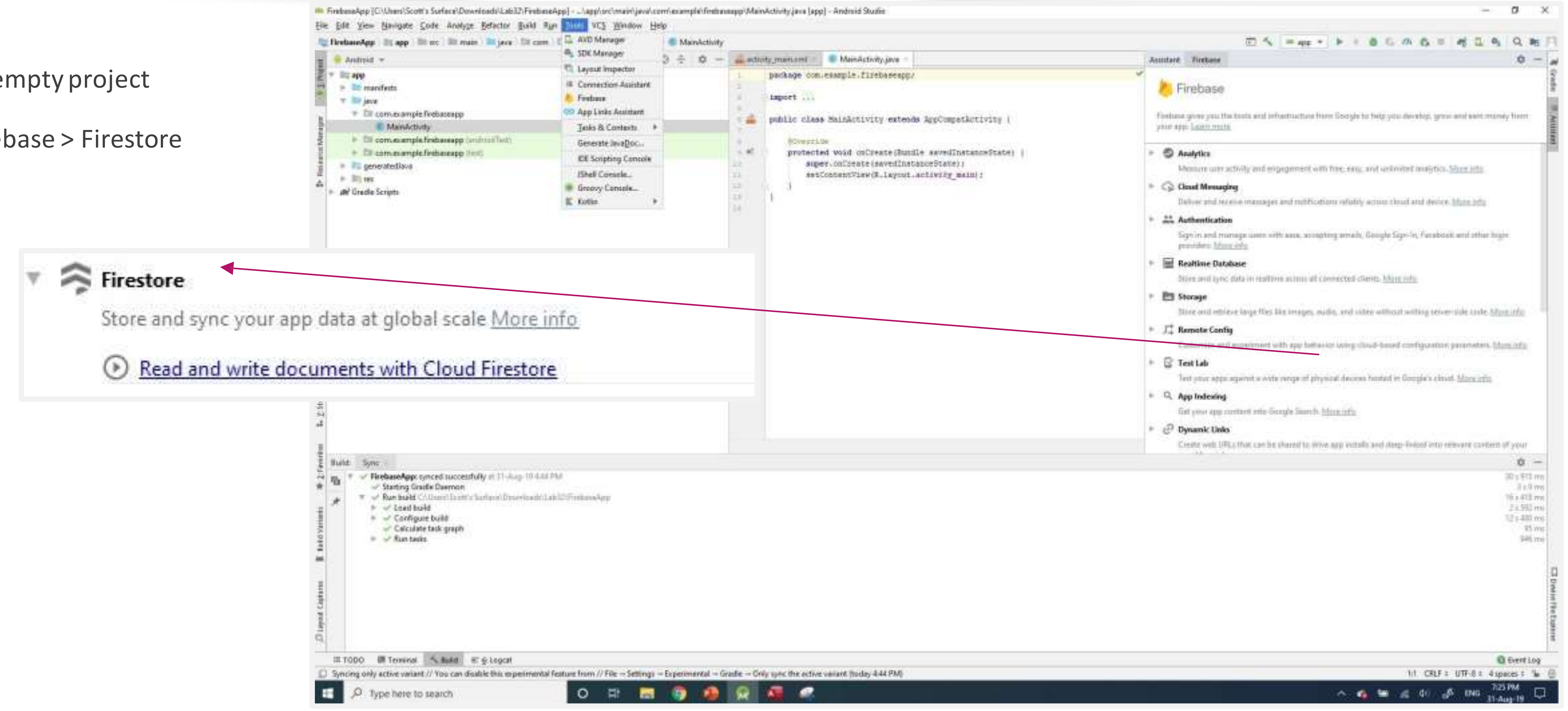
Author: "Scott Mann"
ISBN: "342343-3423423"
Price: 5
Title: "Funny Jokes"

https://console.firebase.google.com/

# Integrating Firebase into your project

Create an empty project

Tools > Firebase > Firestore



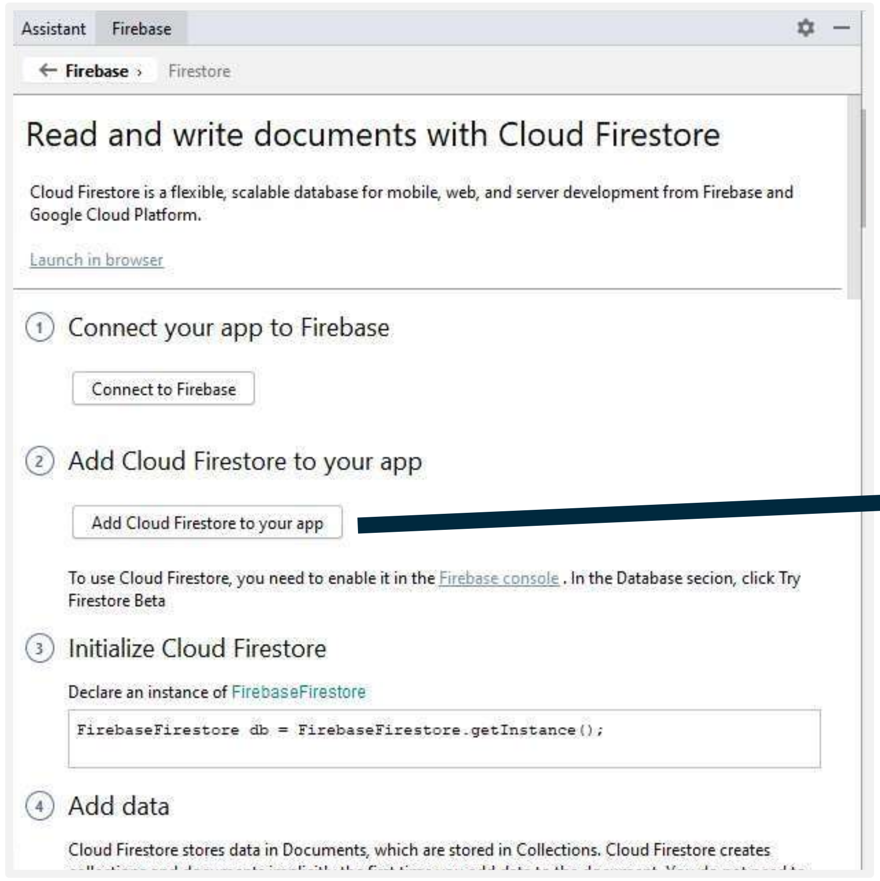
Firestore

Store and sync your app data at global scale More info

Read and write documents with Cloud Firestore

```
package com.example.firebaseapp;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

# Connect to Firebase



Assistant | Firebase

← Firebase › Firestore

Read and write documents with Cloud Firestore

Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud Platform.

Launch in browser

1. Connect your app to Firebase

   Connect to Firebase

2. Add Cloud Firestore to your app

   Add Cloud Firestore to your app

   To use Cloud Firestore, you need to enable it in the Firebase console . In the Database secion, click Try Firestore Beta

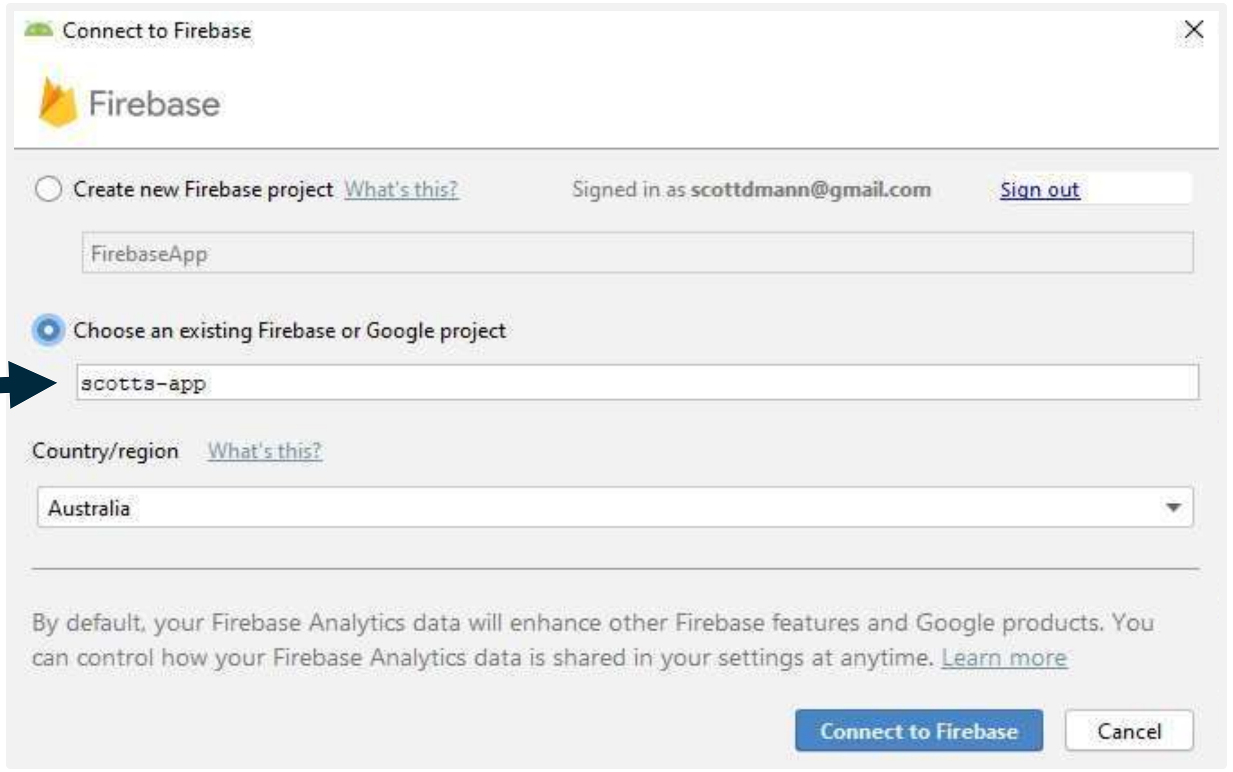
3. Initialize Cloud Firestore

   Declare an instance of FirebaseFirestore

```
FirebaseFirestore db = FirebaseFirestore.getInstance();
```

4. Add data

   Cloud Firestore stores data in Documents, which are stored in Collections. Cloud Firestore creates



Connect to Firebase

Firebase

Create new Firebase project What's this? Signed in as **scottdmann@gmail.com** Sign out

FirebaseApp

[✓ Choose an existing Firebase or Google project]

scotts-app

Country/region What's this?

Australia

By default, your Firebase Analytics data will enhance other Firebase features and Google products. You can control how your Firebase Analytics data is shared in your settings at anytime. Learn more

Connect to Firebase | Cancel

Follow the steps, depending on your Android studio version the ‘Connect to Firebase’ option might open a web browser page or an in-app page (as per image)

## Initialize Cloud Firestore

```java
public class MainActivity extends AppCompatActivity {

    FirebaseFirestore db = null;
    String TAG = "MainAct";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        db = FirebaseFirestore.getInstance();
    }
}
```

Get a firestore instance

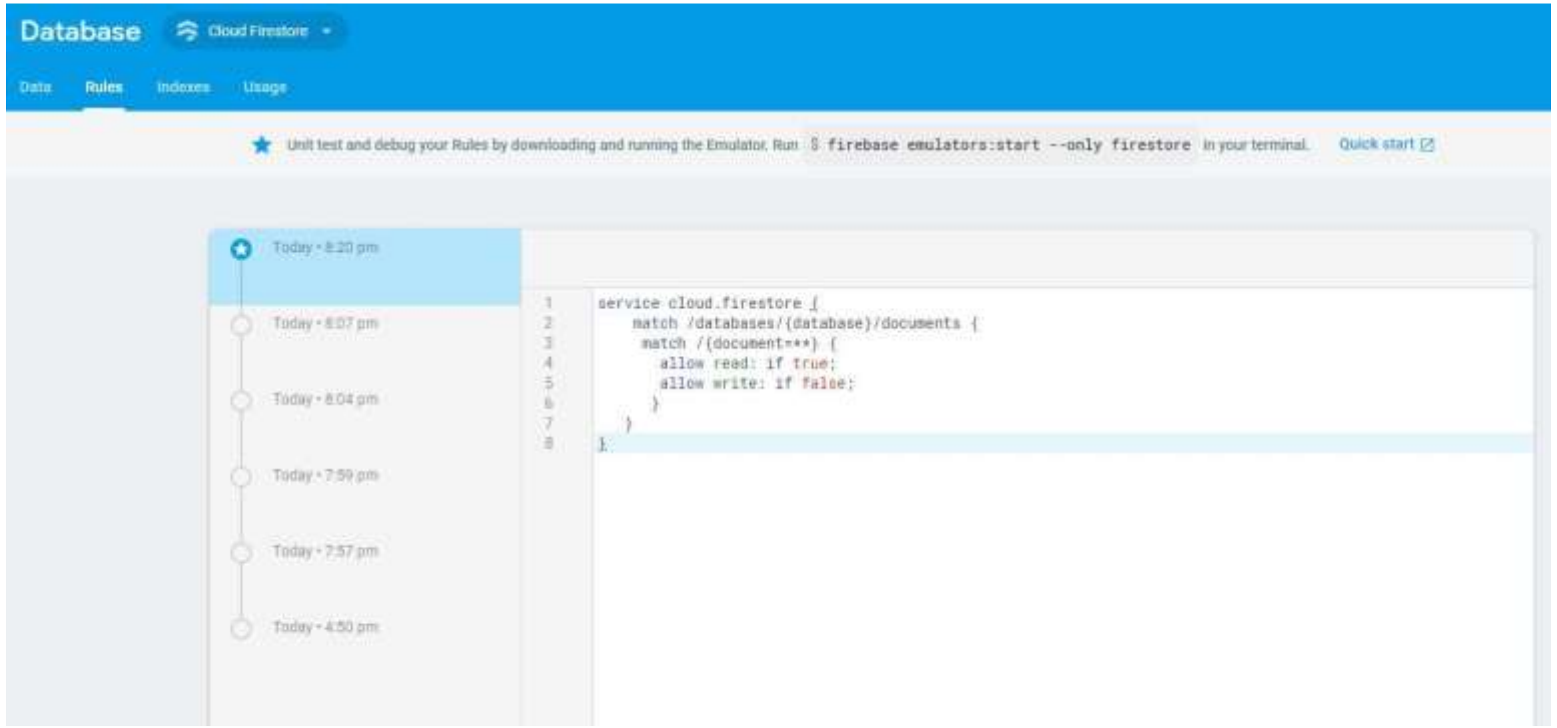
## Query Firestore

Add to onCreate, a get() query for all documents(book) in the Books collection

```
db.collection("Books")
        .get()
        .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
            @Override
            public void onComplete(@NonNull Task<QuerySnapshot> task) {
                if (task.isSuccessful()) {
                    for (QueryDocumentSnapshot document : Objects.requireNonNull(task.getResult())) {
                        Log.d(TAG, document.getId() + " => " + document.getData());
                    }
                } else {
                    Log.w(TAG, "Error getting documents.", task.getException());
                }
            }
        });
```

# Enable read without authentication

Change in your Firestore Rules tab



Unit test and debug your Rules by downloading and running the Emulator. Run `$ firebase emulators:start --only firestore` in your terminal.

```
service cloud.firestore {
  match /databases/{database}/documents {
   match /{document=**} {
     allow read: if true;
     allow write: if false;
   }
  }
};
```

Output to Logcat after running app

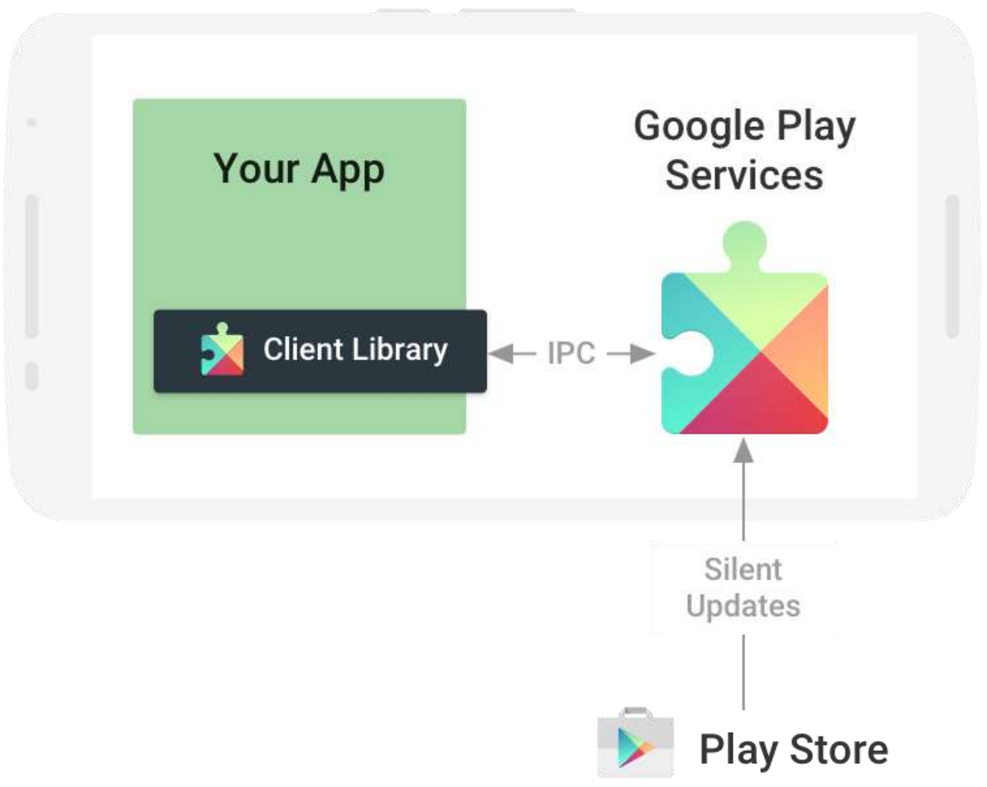
```
08-31 20:20:35.459 3268-3268/? D/FB_READ: 1 => {ISBN=342343-3423423, Price=5, Title=Funny Jokes, Author=Scott Mann}
08-31 20:20:35.459 3268-3268/? D/FB_READ: 2 => {ISBN=344352-3497324, Price=30, Title=Mobile Apps, Author=Shaarang Tanpure}
```

# Location-Based Services in Android

# Google Play Services

- Pushed as an application therefore not as an OS upgrade
    - ➢ Helps get around fragmentation in the ecosystem

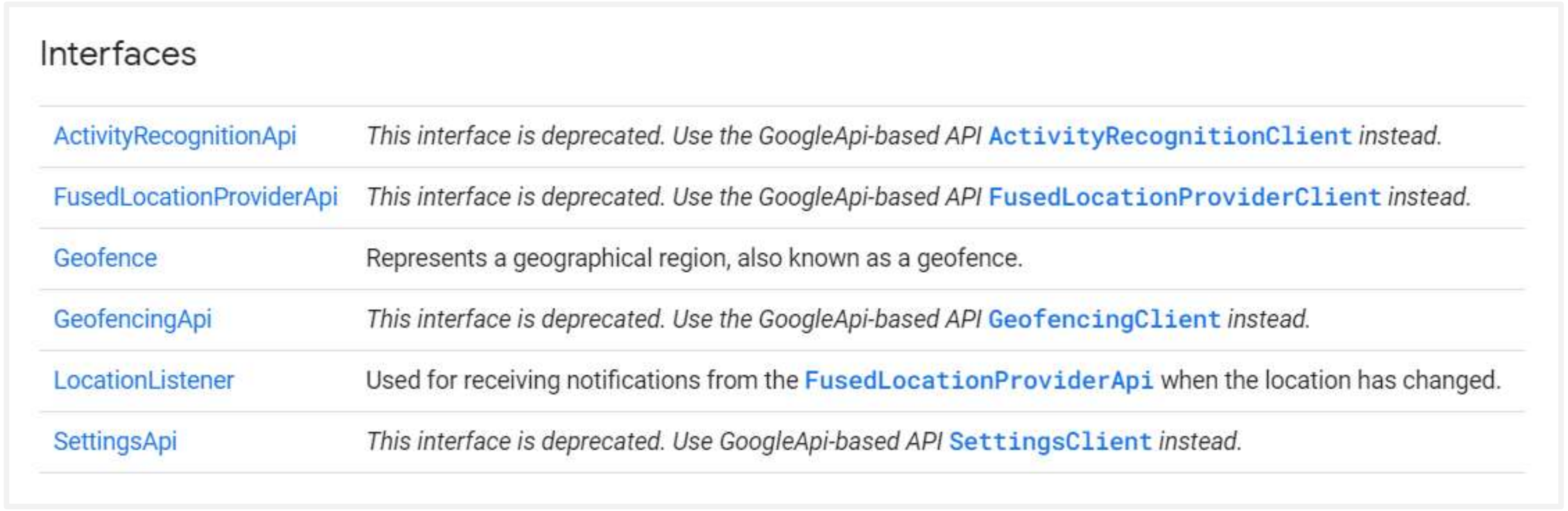
- Services released as an application



https://developers.google.com/android/guides/overview

## Google Play services location API

com.google.android.gms.location



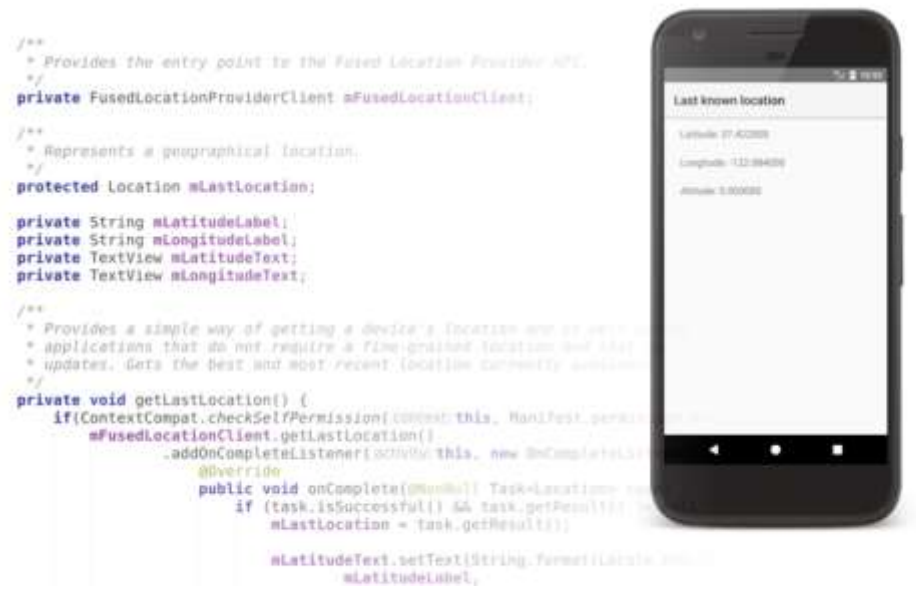
Interfaces

| | |
|---|---|
| ActivityRecognitionApi | *This interface is deprecated. Use the GoogleApi-based API* `ActivityRecognitionClient` *instead.* |
| FusedLocationProviderApi | *This interface is deprecated. Use the GoogleApi-based API* `FusedLocationProviderClient` *instead.* |
| Geofence | Represents a geographical region, also known as a geofence. |
| GeofencingApi | *This interface is deprecated. Use the GoogleApi-based API* `GeofencingClient` *instead.* |
| LocationListener | Used for receiving notifications from the `FusedLocationProviderApi` when the location has changed. |
| SettingsApi | *This interface is deprecated. Use GoogleApi-based API* `SettingsClient` *instead.* |

# FusedLocationProviderClient

Obtaining the location information for a device can be complicated: Devices contain different types of GPS hardware, and a satellite or network connection (cell or Wi-Fi) is not always available. Activating GPS and other hardware components uses power.

To find the device location efficiently without worrying about which provider or network type to use, use the *FusedLocationProviderClient* interface. Before you can use FusedLocationProviderClient, you need to set up Google Play services.



```
/**
 * Provides the entry point to the Fused Location Provider API.
 */
private FusedLocationProviderClient mFusedLocationClient;

/**
 * Represents a geographical location.
 */
protected Location mLastLocation;

private String mLatitudeLabel;
private String mLongitudeLabel;
private TextView mLatitudeText;
private TextView mLongitudeText;

/**
 * Provides a simple way of getting a device's location and is well suited for
 * applications that do not require a fine-grained location and that
 * updates. Gets the best and most recent location currently available
 */
private void getLastLocation() {
    if(ContextCompat.checkSelfPermission( context: this, Manifest.permission
        mFusedLocationClient.getLastLocation()
                .addOnCompleteListener( activity: this, new OnCompleteListener
                    @Override
                    public void onComplete(@NonNull Task<Location> task
                        if (task.isSuccessful() && task.getResult()
                            mLastLocation = task.getResult();

                            mLatitudeText.setText(String.format(Locale
                                    mLatitudeLabel,
```

https://developers.google.com/location-context/fused-location-provider/

## FusedLocationProviderClient

**Last Known Location**

Using the fused location provider API, your app can request the last known location of the user's device. Getting the last known location is usually a good starting point for apps that require location information.

**Location Updates**

In addition to the last known location, the fused location provider API can deliver location updates to a callback in your app at specific intervals. You can specify the desired interval as a parameter of the quality of service. By using location updates, your app can provide additional information such as direction and velocity.

https://developer.android.com/training/location

## Set Up Google Play Services

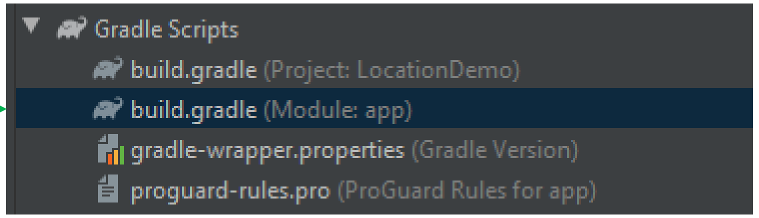
To develop an app using the Google Play services APIs, you need to set up your project with the Google Play services SDK, which is available from the Google maven repository.

To test your app when using the Google Play services SDK, you must use either:

- A compatible Android device that runs Android 4.1 or higher and includes Google Play Store.
- The Android emulator with an AVD that runs the Google APIs platform based on Android 4.2.2 or higher.

# Add Google Play Services to your Project

1. Open the build.gradle file inside your application module directory.



```
Gradle Scripts
    build.gradle (Project: LocationDemo)
    build.gradle (Module: app)
    gradle-wrapper.properties (Gradle Version)
    proguard-rules.pro (ProGuard Rules for app)
```

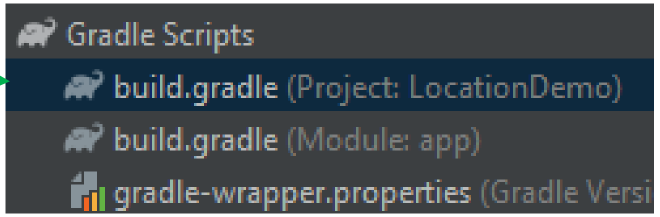
2. Add a new build rule under dependencies for the latest version of play-services, using the latest API Version number

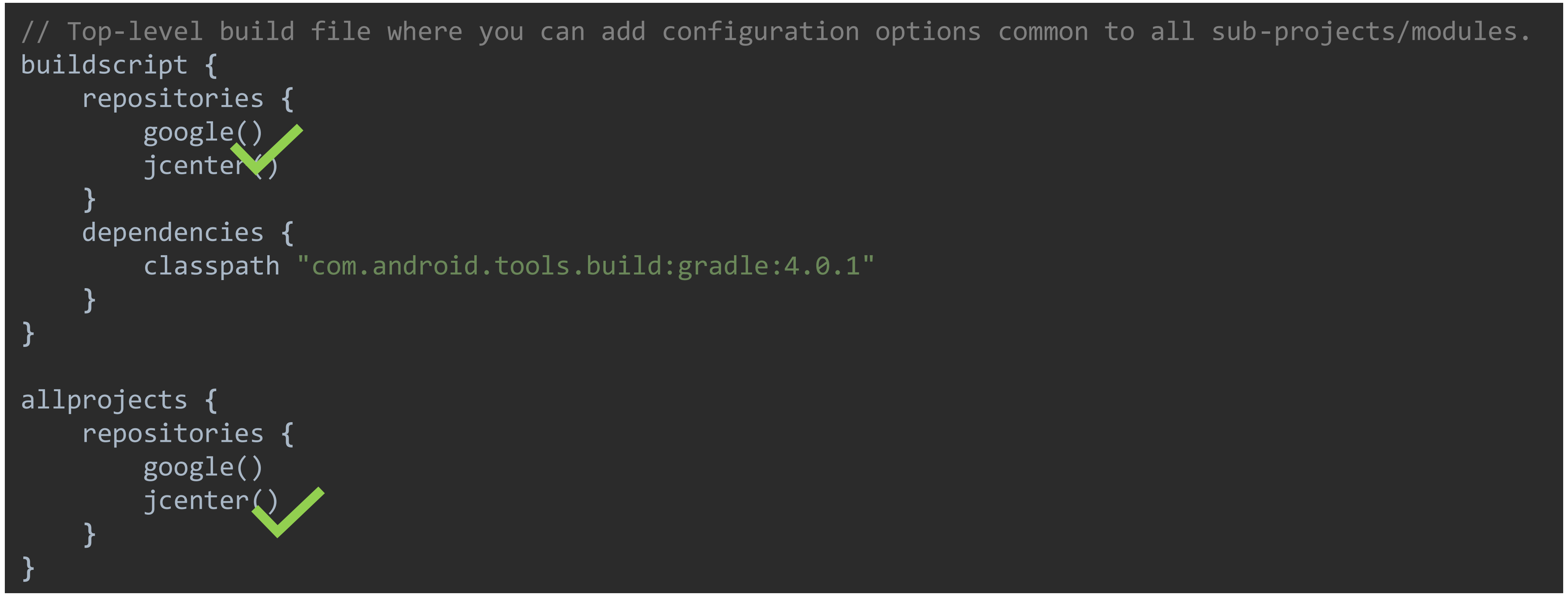
```
apply plugin: 'com.android.application’

...

dependencies {
    implementation 'com.google.android.gms:play-services-location:17.0.0'
```

https://developers.google.com/android/guides/setup

# Add Google Play Services to your Project

3. Ensure that your top-level build.gradle contains a reference to the google() repo or to maven { url "https://maven.google.com" }.


Gradle Scripts
build.gradle (Project: LocationDemo)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Versi



```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
buildscript {
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:4.0.1"
    }
}

allprojects {
    repositories {
        google()
        jcenter()
    }
}
```

4. Save the changes, and click Sync Project with Gradle Files in the toolbar.

## Location Permissions & Type

To protect user privacy, apps that use location services must request location permissions.

When you request location permissions, follow the same best practices as you would for any other runtime permission.

One important difference when it comes to location permissions is that the system includes multiple permissions related to location. Which permissions you request, and how you request them, depend on the location requirements for your app's use case.

### Types of location access

Android's location permissions deal with the following categories of location access:

- Foreground location
- Background location

https://developer.android.com/training/location/permissions

# Foreground location

If your app contains a feature that shares or receives location information only once, or for a defined amount of time, then that feature requires foreground location access. Some examples include the following:

- Within a navigation app, a feature allows users to get turn-by-turn directions.
- Within a messaging app, a feature allows users to share their current location with another user.

```
<manifest ... >
  <!-- To request foreground location access, declare one of these permissions. -->
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
</manifest>
```

ACCESS_COARSE_LOCATION - Provides location accuracy to within a city block.

ACCESS_FINE_LOCATION - Provides a more accurate location.

https://developer.android.com/training/location/permissions

## Background location

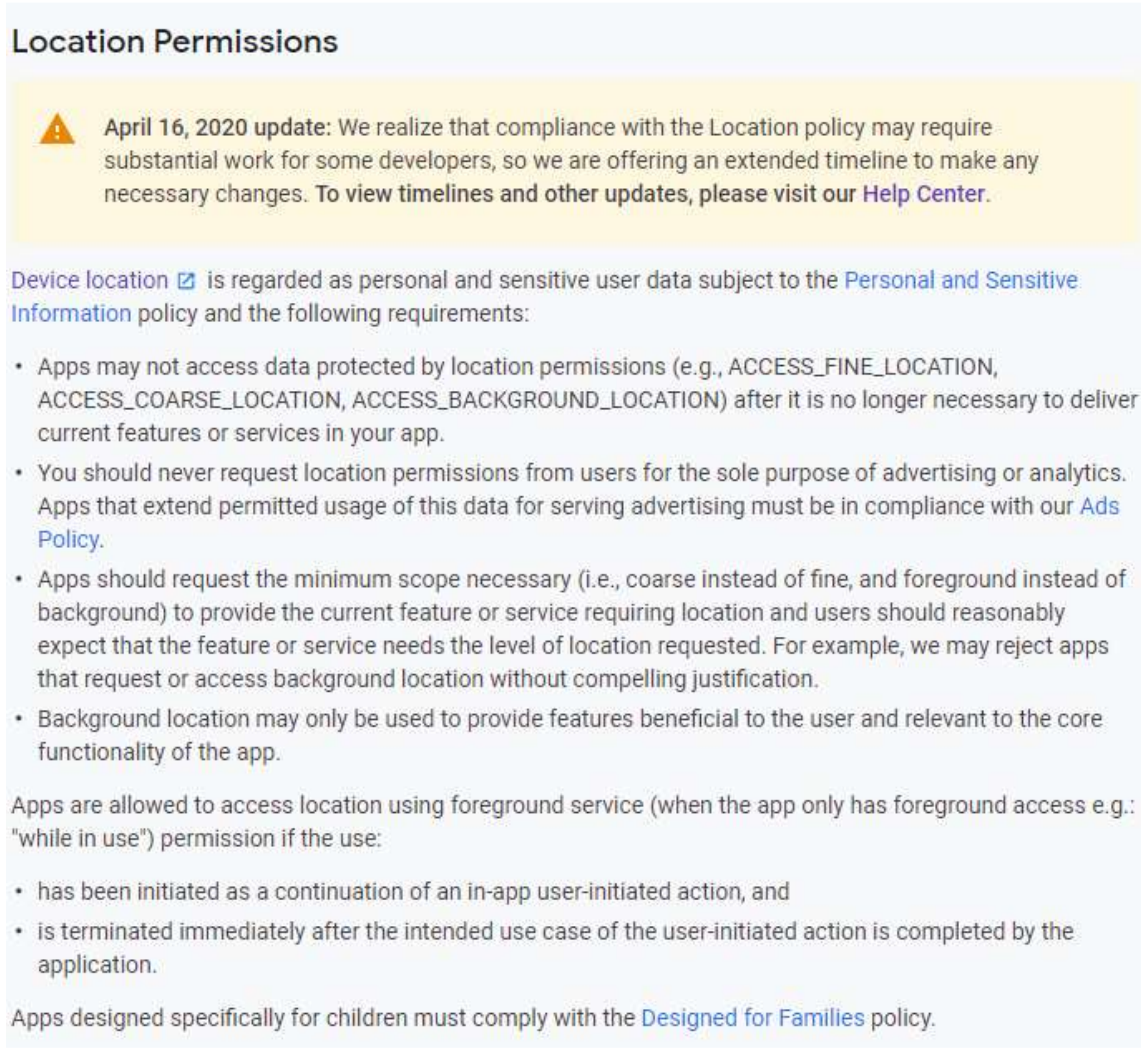
An app requires background location access if a feature within the app constantly shares location with other users or uses the Geofencing API. Several examples include the following:

- Within a family location sharing app, a feature allows users to continuously share location with family members.
- Within an IoT app, a feature allows users to configure their home devices such that they turn off when the user leaves their home and turn back on when the user returns home.

```
<manifest ... >
  <!-- Required only when requesting background location access on
       Android 10 (API level 29). -->
  <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
</manifest>
```

https://developer.android.com/training/location/permissions

# Location Permission Changes 2020



## Location Permissions

**April 16, 2020 update:** We realize that compliance with the Location policy may require substantial work for some developers, so we are offering an extended timeline to make any necessary changes. **To view timelines and other updates, please visit our Help Center.**

Device location is regarded as personal and sensitive user data subject to the Personal and Sensitive Information policy and the following requirements:

- Apps may not access data protected by location permissions (e.g., ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION, ACCESS_BACKGROUND_LOCATION) after it is no longer necessary to deliver current features or services in your app.
- You should never request location permissions from users for the sole purpose of advertising or analytics. Apps that extend permitted usage of this data for serving advertising must be in compliance with our Ads Policy.
- Apps should request the minimum scope necessary (i.e., coarse instead of fine, and foreground instead of background) to provide the current feature or service requiring location and users should reasonably expect that the feature or service needs the level of location requested. For example, we may reject apps that request or access background location without compelling justification.
- Background location may only be used to provide features beneficial to the user and relevant to the core functionality of the app.

Apps are allowed to access location using foreground service (when the app only has foreground access e.g.: "while in use") permission if the use:

- has been initiated as a continuation of an in-app user-initiated action, and
- is terminated immediately after the intended use case of the user-initiated action is completed by the application.

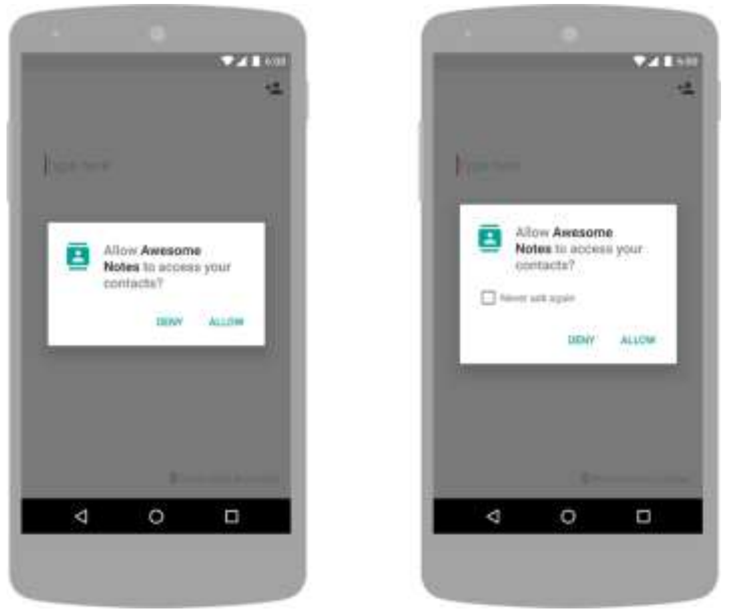
Apps designed specifically for children must comply with the Designed for Families policy.

https://support.google.com/googleplay/android-developer/answer/9888170

# 💀 Dangerous Permissions!!

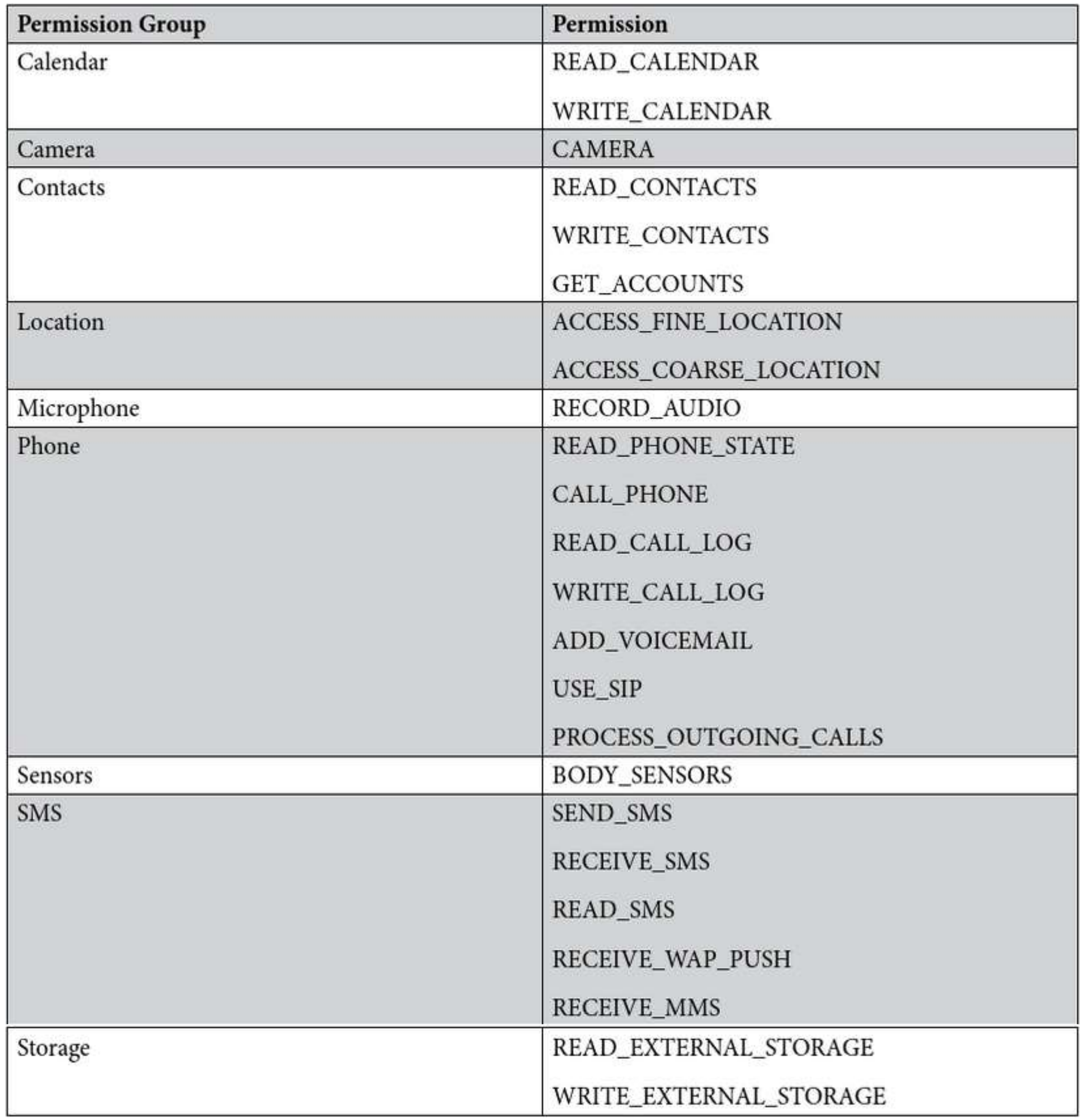
The way Android asks the user to grant dangerous permissions depends on the version of Android running on the user's device, and the system version targeted by your app.

- Runtime requests (Android 6.0 and higher)
  - ➢ If the device is running Android 6.0 (API level 23) or higher, and the app's targetSdkVersion is 23 or higher, the user isn't notified of any app permissions at install time. Your app must ask the user to grant the dangerous permissions at runtime.
- Install-time requests (Android 5.1.1 and below)
  - ➢ If the device is running Android 5.1.1 (API level 22) or lower, or the app's targetSdkVersion is 22 or lower while running on any version of Android, the system automatically asks the user to grant all dangerous permissions for your app at install-time.


Allow Awesome Notes to access your contacts?
DENY
ALLOW
Allow Awesome Notes to access your contacts?
Never ask again
DENY
ALLOW

https://developer.android.com/guide/topics/permissions/overview#dangerous-permission-prompt

# Dangerous Permissions



| Permission Group | Permission |
|---|---|
| Calendar | READ_CALENDAR<br>WRITE_CALENDAR |
| Camera | CAMERA |
| Contacts | READ_CONTACTS<br>WRITE_CONTACTS<br>GET_ACCOUNTS |
| Location | ACCESS_FINE_LOCATION<br>ACCESS_COARSE_LOCATION |
| Microphone | RECORD_AUDIO |
| Phone | READ_PHONE_STATE<br>CALL_PHONE<br>READ_CALL_LOG<br>WRITE_CALL_LOG<br>ADD_VOICEMAIL<br>USE_SIP<br>PROCESS_OUTGOING_CALLS |
| Sensors | BODY_SENSORS |
| SMS | SEND_SMS<br>RECEIVE_SMS<br>READ_SMS<br>RECEIVE_WAP_PUSH<br>RECEIVE_MMS |
| Storage | READ_EXTERNAL_STORAGE<br>WRITE_EXTERNAL_STORAGE |

https://youtu.be/C8lUdPVSzDk

# Get the last known location

1. Set up Google Play services ✓
2. Specify app permissions ✓
3. Create location services client

```
private FusedLocationProviderClient fusedLocationClient;

// ..

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...

    fusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
}
```

https://developer.android.com/training/location/retrieve-current

# Get the last known location (continued)

4. Get the last known location

```
fusedLocationClient.getLastLocation()
        .addOnSuccessListener(this, new OnSuccessListener<Location>() {
            @Override
            public void onSuccess(Location location) {
                // Got last known location. In some rare situations this can be null.
                if (location != null) {
                    // Logic to handle location object
                }
            }
        });
```

https://developer.android.com/training/location/retrieve-current

# Get the last known location (continued)

The getLastLocation() method returns a Task that you can use to get a Location object with the latitude and longitude coordinates of a geographic location. (See docs below for all location object methods).

**https://developer.android.com/reference/android/location/Location**

The location object may be null in the following situations:

- Location is turned off in the device settings. The result could be null even if the last location was previously retrieved because disabling location also clears the cache.
- The device never recorded its location, which could be the case of a new device or a device that has been restored to factory settings.
- Google Play services on the device has restarted, and there is no active Fused Location Provider client that has requested location after the services restarted. To avoid this situation you can create a new client and request location updates yourself.

https://developer.android.com/training/location/retrieve-current

## Last known location example

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture6_LastLocationDemo

## Optimize location for battery

Location gathering and battery drain are directly related in the following aspects:

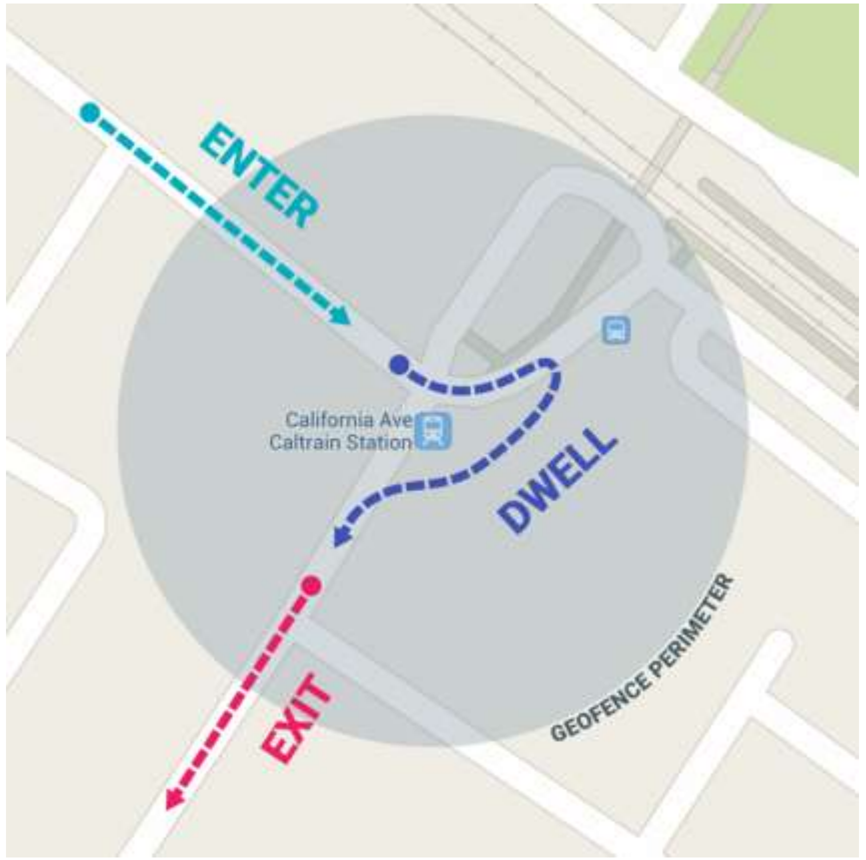
- Accuracy: The precision of the location data. In general, the higher the accuracy, the higher the battery drain.
- Frequency: How often location is computed. The more frequent location is computed, the more battery is used.
- Latency: How quickly location data is delivered. Less latency usually requires more battery.

https://developer.android.com/guide/topics/location/battery

# Geofence API

The geofencing API allows you to define perimeters, also referred to as geofences, which surround the areas of interest. Your app gets a notification when the device crosses a geofence, which allows you to provide a useful experience when users are in the vicinity.

- Provide contextual experiences when users enter or leave an area of interest
- Receive notifications when users trigger your geofences



https://developer.android.com/training/location/geofencing

# Google Maps API

With the Maps SDK for Android, you can add maps based on Google Maps data to your application. The API automatically handles access to Google Maps servers, data downloading, map display, and response to map gestures. You can also use API calls to add markers, polygons, and overlays to a basic map, and to change the user's view of a particular map area. These objects provide additional information for map locations, and allow user interaction with the map. The API allows you to add these graphics to a map:

- Icons anchored to specific positions on the map (Markers).
- Sets of line segments (Polylines).
- Enclosed segments (Polygons).
- Bitmap graphics anchored to specific positions on the map (Ground Overlays).
- Sets of images which are displayed on top of the base map tiles (Tile Overlays).

https://developer.android.com/training/maps
https://developers.google.com/maps/documentation/android-sdk/overview

# Adding a GoogleMap to your app

1. **Create a Google Maps project**

Create a new project as follows:

- If you see the Welcome to Android Studio dialog, choose Start a new Android Studio project, available under 'Quick Start' on the right of the dialog.
- Otherwise, click File in the Android Studio menu bar, then New, New Project.

- In the Choose your project dialog, select the tab that corresponds to the platform you intended to develop for. Most users will want to keep the default Phone and Tablet.

- Select Google Maps Activity, then click Next.

- Enter your app name, package name, and project location, and the minimum Android API level supported by your app, then click Finish.

https://developers.google.com/maps/documentation/android-sdk/start

# Adding a GoogleMap to your app

**2. Get a Google Maps API key**

- Use the link provided in the google_maps_api.xml file that Android Studio created for you.
- Copy the link provided in the google_maps_api.xml file and paste it into your browser. The link takes you to the Google Cloud Platform Console and supplies the required information to the Google Cloud Platform Console via URL parameters, thus reducing the manual input required from you.
- Follow the instructions to create a new project on the Google Cloud Platform Console or select an existing project.
- Create an Android-restricted API key for your project.
- Copy the resulting API key, go back to Android Studio, and paste the API key into the <string> element in the google_maps_api.xml file.

```
<string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">AIzaxxxxxxxxxxxxxxxxxxxx</string>
```

## Adding a GoogleMap to your app

**3. Adjust MapsActivity.java**

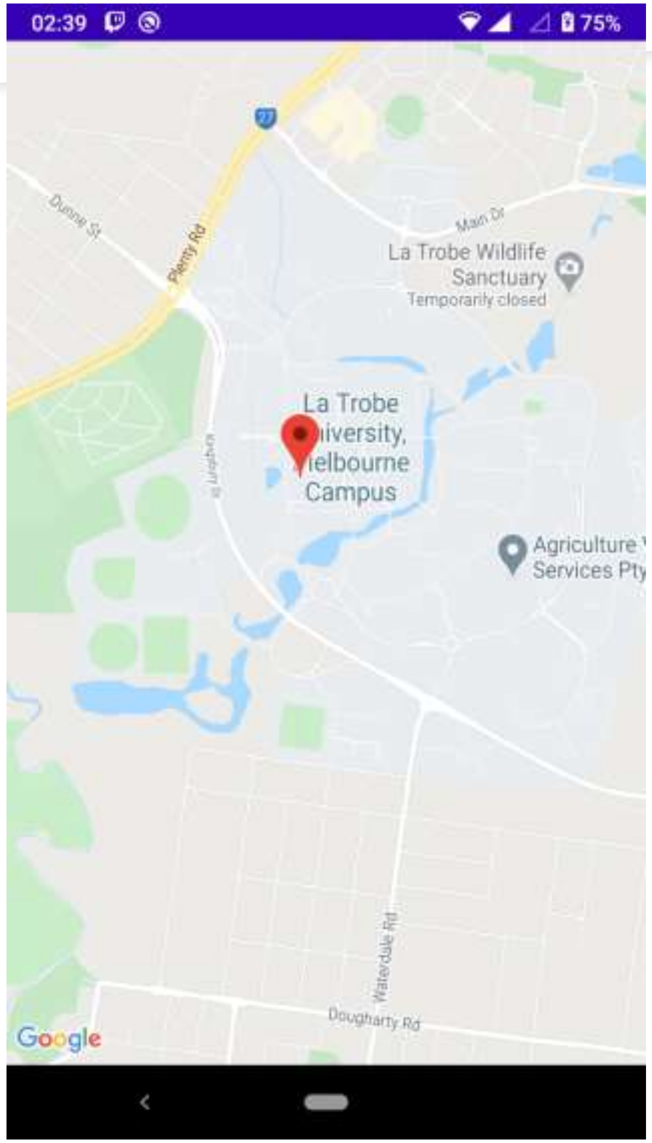
- Adjust the code to make a marker appear over Latrobe or any other location (you can use https://www.latlong.net/ to find the coordinates)

```java
// Add a marker at Latrobe Bundoora and move the camera
LatLng latrobe = new LatLng(-37.721476, 145.046851);
mMap.addMarker(new MarkerOptions().position(latrobe).title("Marker at Latrobe"));
mMap.moveCamera(CameraUpdateFactory.newLatLng(latrobe));
```

# Adding a GoogleMap to your app

**4. Build and run your app**

- In Android Studio, click the Run menu option (or the play button icon) to run your app.
- When prompted to choose a device, choose one of the following options:
- Select the Android device that's connected to your computer.
- Alternatively, select the Launch emulator radio button and choose the virtual device that you've previously configured.



# Maps Demo

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture6_GoogleMapsDemo