

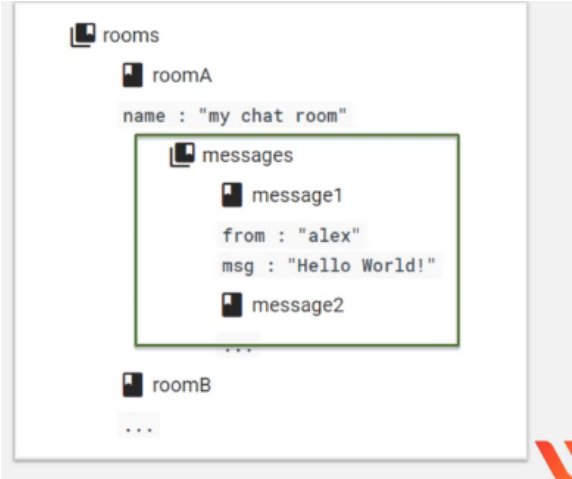
# Datastore/Firestore

25 July 2021 20:46

## Key Points Datastore/Firestore

- 1. Document and collection
- 2. Firestore Support nested document with each having document id autogenerated and sub-collection. Primitive data type like string and complex data type like list supported.
- 3. In firestore all fields are by default indexed (Single-field index -> a sorted mapping of field value)
- 4. We can exempt a field from single-field index by using single-field-index exemption. (Firestore)
- 5. Firestore support Composit index can be used to based on field order.
- 6. Previously eventual consistent but now strong consistency with firestore storage layer.
- 7. Contention can be resolved by failing one transaction at mobile sdk or by placing a lock in document at server side.
- 8. Support millions of concurrent client
- 9. Firestore support max 10,000 req per sec for datastore no limit.
- 10. Offline sync support present in firestore
- 11. Entity and Kind are used for datastore.
- 12. Scale automatically
- 13. Used for gaming and user profile data, indexing for objects in cloud storage
- 14. Batch operation are more suited
- 15. Choice of firestore is permanent
- 16. With multi-region 99.999% and with single region 99.99% availability
- 17. All data by default encrypted at rest.
- 18. Datastore provide sql like query with no join support
- 19. In datastore mode we can have a root entity with related entity under it knows as entity group. For example a product entity and all its info
- 20. With firestore storage layer 1 transaction/sec per entity group limitation removed. Transaction are not limited to 25 entity group.
- 21. Firestore is real-time database for realtime notification.

## Sub-collection structure



	Native mode	Datastore mode
	Enable all of Cloud Firestore's features, with offline support and real-time synchronization.	Leverage Cloud Datastore's system behavior on top of Cloud Firestore's powerful storage layer.
	<a href="#">SELECT NATIVE MODE</a>	<a href="#">SELECT DATASTORE MODE</a>
API	Firestore	Datastore
Scalability	Automatically scales to millions of concurrent clients	Automatically scales to millions of writes per second
App engine support	Not supported in the App Engine standard Python 2.7 and PHP 5.5 runtimes	All runtimes
Max writes per second	10,000	No limit
Real-time updates	✓	✗

Mobile/web client libraries with offline data persistence	✓	✗
Query consistency	Strong	Strong
Data model	Documents / collections	Entities / kinds
Web console	Firestore page in Google Cloud Platform and Firebase	Datastore page in Google Cloud Platform

### Limit and Costing

#### Datastore limits

If you have not yet upgraded from Datastore to [Firestore in Datastore mode](#), the following limits also apply to your database instance:

Limit	Amount
Maximum number of entity groups that can be accessed in a transaction	25
Maximum rate of transactions reading from or writing to an entity group	<a href="#">1 per sec</a>
Maximum write rate to an entity group. Note you can batch writes together for an entity group. This allows you to write multiple entities to an entity group within this limit.	1 per second
No of write per second 10,000	

#### Firestore in Datastore mode limits


Maximum size for a transaction: 10MB  
Maximum depth of nested entity values: 20

### Datastore/Firestore structure

Concept	Cloud Datastore	Cloud Firestore	Relational database
Category of object	Kind	Collection group	Table
One object	Entity	Document	Row
Individual data for an object	Property	Field	Column
Unique ID for an object	Key	Document ID	Primary key

### Different kind of query in datastore

- **Ancestor queries:** An ancestor query limits its results to the specified entity and its descendants
- **Kindless queries:** A query with no kind and no ancestor retrieves all of the entities of an application from Datastore mode
- **Projection queries:** Projection queries allow you to query for specific properties of an entity that you actually need
- **Keys-only queries:** A keys-only query returns just the keys of the result entities instead of the entities

	<p>themselves</p>
Stackdriver logging supported	<ul style="list-style-type: none"> <li>• <b>Requests:</b> Basic traffic monitoring for volume and any changes</li> <li>• <b>Read Size:</b> Size of entity reads grouped by type</li> <li>• <b>Write Size:</b> Size of entity writes grouped by type</li> <li>• <b>Index Writes:</b> Count of index writes from inbound writes and volume/changes</li> </ul> 
Supported data type firestore	<p><u>Supported data types</u></p> <ul style="list-style-type: none"> <li>• Array</li> <li>• Boolean</li> <li>• Bytes</li> <li>• Date and time</li> <li>• Floating point number</li> <li>• Geographical point</li> <li>• Integer</li> <li>• Map – e.g. {a: "foo", b: "bar", c: "qux"}.</li> <li>• Null</li> <li>• Reference – e.g. projects/[PROJECT_ID]/databases/[DATABASE_ID]</li> <li>• Text string</li> </ul>
<p><b>Cloud Datastore</b></p> <p><b>Data Contention</b></p> <p>Data contention</p> <p>When two or more operations compete to control the same document. For example, one transaction might require a document to remain consistent while a concurrent operation tries to update that document's field values.</p> <p>Firestore resolves data contention by delaying or failing one of the operations. The Firestore client libraries automatically retry transactions that fail due to data contention. After a finite number of retries, the transaction operation fails and returns an error message:</p> <p>ABORTED: Too much contention on these documents. Please try again.</p>	<p><b>Cloud Datastore and Firestore</b></p> <ul style="list-style-type: none"> <li>• <b>Datastore</b> - Highly scalable NoSQL Document Database <ul style="list-style-type: none"> <li>▪ Automatically scales and partitions data as it grows</li> <li>▪ Recommended for upto a few TBs of data <ul style="list-style-type: none"> <li>◦ For bigger volumes, BigTable is recommended</li> </ul> </li> <li>▪ Supports Transactions, Indexes and SQL like queries (GQL) <ul style="list-style-type: none"> <li>◦ Does NOT support Joins or Aggregate (sum or count) operations</li> </ul> </li> <li>▪ For use cases needing flexible schema with transactions <ul style="list-style-type: none"> <li>◦ Examples: User Profile and Product Catalogs</li> </ul> </li> <li>▪ Structure: Kind &gt; Entity (Use namespaces to group entities)</li> <li>▪ You can export data ONLY from gcloud (NOT from cloud console) <ul style="list-style-type: none"> <li>◦ Export contains a metadata file and a folder with the data</li> </ul> </li> </ul> </li> <li>• <b>Firestore</b> = Datastore++ : Optimized for multi device access <ul style="list-style-type: none"> <li>▪ Offline mode and data synchronization across multiple devices - mobile, IOT etc</li> <li>▪ Provides client side libraries - Web, iOS, Android and more</li> <li>▪ Offers <b>Datastore</b> and Native modes</li> </ul> </li> </ul>
Play with firestore	<p><u>Select Firestore Mode</u></p>

- Choice of firestore mode is permanent for the project. We can not change later.
- In Firestore a collection is a table and a document is a row
- Inside a document we can add field or a nested collection
- Indexing is by default added to all the fields , but we can create composite index

1
Select a Cloud Firestore mode
2
Choose where to store your data

Cloud Firestore is the next generation of Cloud Datastore. You can use Cloud Firestore in either Native mode or Datastore mode, each with distinct system behavior optimized for different types of projects. [Pricing](#) for both modes is based on location, stored data, operations, and network egress with a daily free quota for each. [Learn more about choosing a mode](#)

**The mode you select here will be permanent for this project**

	<b>Native mode</b> Enable all of Cloud Firestore's features, with offline support and real-time synchronization.  SELECT NATIVE MODE	<b>Datastore mode</b> Leverage Cloud Datastore's system behavior on top of Cloud Firestore's powerful storage layer.  SELECT DATASTORE MODE
API	Firestore	Datastore
Scalability	Automatically scales to millions of	Automatically scales to millions of

### Select Location

1
Select a Cloud Firestore mode
2
Choose where to store your data

You selected Cloud Firestore in Native mode. Now choose a database location.

The location of your database affects its cost, availability, and durability. Choose a regional location (lower write I/O) or a multi-region location (higher availability, higher cost). [Learn more](#)

**Choose carefully: your location selection is permanent and will also apply to this project's App Engine**

Select a location

asia-south1 (Mumbai)

To improve performance, store your data close to the users and services that need it

CREATE DATABASE BACK

Google Cloud Platform
My First Project
Data
/
Root
+ START COLLECTION

### Start a collection

A collection is a set of one or more documents that contain data. Start a collection at this path by adding its first document. [Learn more](#)

Give the collection an ID

Parent path  
/

Collection ID \*  
to

[https://onedrive.live.com/redir?resid=762A2E8BFA6817EE%211993&page=Edit&wd=target%28Google Vcloud.one%7Cb20e3d52-1b36-42f2-9e7d-3feed4218d22%2FDatastore%5C%2FFires...](https://onedrive.live.com/redir?resid=762A2E8BFA6817EE%211993&page=Edit&wd=target%28Google%20Vcloud.one%7Cb20e3d52-1b36-42f2-9e7d-3feed4218d22%2FDatastore%5C%2FFires...) 4/11

OneNote

Choose an ID that describes the documents you'll add to this collection.

Add its first document

Document ID

Assigned automatically. Customize if needed.

Field name

Field type

string

Field value

SAVE

SAVE & ADD ANOTHER

CANCEL

nlzfMpFi

todos

+ ADD DOCUMENT

3M9DzZVFmak4MzSsgva

MrHawHEjopGgUNwiLhHj

PCLCZKxl8SZqnlzfMpFi

PCLCZKxl8SZqnlzfMpFi

+ START COLLECTION

+ ADD FIELD

description: "Learn AWS and Azure"

target\_date: February 26, 2021 at 5:11...

Create Indexing

Products and resources

E FIELD

multiple fields by indexing

INDEX

Query scope

the indexes you

desired query in your a

Create a composite index

Cloud Firestore uses composite indexes for compound queries not already supported by single field indexes (ex: combining equality and range operators). [Learn more](#)

Tip: instead of creating indexes here, run queries in your code – if you're missing any indexes, you'll see errors with links to create them.

[Learn more](#)

Collection ID \*

to|

Fields to index

Field paths

Index options

Ascending

Ascending

+ ADD FIELD

Query scope

Collection

For queries in a specific collection

CREATE

CANCEL

Add filter on document to query

https://onedrive.live.com/redir?resid=762A2E8BFA6817EE%211993&page=Edit&wd=target%28Google%20Vlound.one%7Cb20e3d52-1b36-42f2-9e7d-3feed4218d22%2FDatastore%5C%2FFires... 5/11

and resource

**Add filters**

Choose a field to filter by

description

**Add a condition (optional)**

Only show documents where the specified field is...

(==) equal to

String

Learn GCP

**Sort results**

☐ Ascending order

☐ Descending order

**Preview query code**

```
.collection("todos")
  .where("description", "==", "Learn GCP")
```

## Import/ Export Firestore

## Export

## ← Export

☒ Export entire database

Acts as a full backup of your database at this point in time. Large amounts of data take longer to export.

☐ Export one or more collection groups

You can choose up to 60 collection groups per export.

Choose collection group(s)

**Choose Destination**

Choose a Google Cloud Storage bucket or folder to export into. Your data from project glowing-furnace-304608 will be stored in asia-south1.

Bucket

BROWSE

Choose a bucket to export into.



You'll be billed based on entity reads and writes, and for storage. [Learn more](#)

Import

← Import

**Choose a file to import**

You can only import files that were previously exported from Firestore. Data will be imported into project **glowing-furnace-304608** in **asia-south1** via upsert. [Learn more](#)

Filename

BROWSE



You'll be billed based on entity reads and writes, and for storage. [Learn more](#)

IMPORT

CANCEL

**Datastore best practices**

## Understanding Cloud Datastore Best Practices

- **Cloud Datastore is a document store with flexible schema**
  - Recommended for storing things like user profiles
  - Another Use Case: Index for objects stored in Cloud Storage
    - You want to allow users to upload their profile pictures:
      - Store objects (pictures) in Cloud Storage
      - Enable quick search by storing metadata (like ids and cloud storage bucket, object details) in Cloud Datastore
- **Design your keys and indexes carefully:**
  - Avoid monotonically increasing values as keys
    - NOT RECOMMENDED - 1, 2, 3, ..., OR "Customer1", "Customer2", "Customer3", ... or timestamps
    - RECOMMENDED - Use `allocatIds()` for well-distributed numeric IDs
  - Create indexes only if they would be used in queries
    - For ad hoc queries on large datasets without pre-defined indexes, BigQuery is recommended!
- **Prefer batch operations (to single read, write or delete operations):**
  - More efficient as multiple operations are performed with same overhead as one operation

**Datastore emulator for local debugging**

The Datastore emulator provides local emulation of the production Datastore environment. You can use the emulator to develop and test your application locally.

**Datastore mode**

Get started

✓ Select a Cloud Firestore mode — 2 Choose where to store your data

You selected Cloud Firestore in Datastore mode. Now choose a database location.

The location of your database affects its cost, availability, and durability. Choose a regional location (lower write latency, lower cost) or a multi-region location (higher availability, higher cost). [Learn more](#)

⚠ Choose carefully: your location selection is permanent and will also apply to this project's App Engine app

Select a location

eur3 (Europe)

To improve performance, store your data close to the users and services that need it

CREATE DATABASE

BACK

## ← Create an entity

Namespace

[default]



Kind

Task



Key identifier

Numeric ID (auto-generated)



### ✓ SPECIFY PARENT

## Properties

ADD PROPERTY

CREATE

CANCEL

Entities

+ CREATE ENTITY

DELETE

QUERY BY KIND

QUERY BY GQL

Kind



OneNote

Task

FILTER ENTITIES

Name/ID ↑

[id=5634161670881280](#)

← Edit entity

REFRESH

DELETE

Namespace ?

[default]

Kind ?

Task

Key ?

Task id:5634161670881280

Key literal ?

Key(Task, 5634161670881280)

URL-safe key ?

ahdlfmRhdGFzdG9yZS1kZW1vLTl0Njc5MHIRCxiEVGFzaxiAglDo14eBCGw

Properties

New property

^

Name \*

description

Type

String

Value

Studying for the exam

☒ Index this property

CANCEL

DONE

ADD PROPERTY

SAVE

CANCEL

← Edit entity

REFRESH

DELETE

Namespace ?



[default]

Kind ?

Task

Key ?

Task id:5634161670881280

Key literal  Key(Task, 5634161670881280)  
URL-safe key  ahdIfmRh dGFzdG9yZS1kZW1vLT10Njc xMHIRCxIEVGZaxiAgIDo14eBCgw

## Properties

description: Studying for the exam  
Indexed

date: 2019-07-14 (11:49:00.000) UTC+1  
Indexed (Not saved)

## New property

Name \*  
done

Type  
Boolean

Value  
False

☒ Index this property

CANCEL DONE

ADD PROPERTY

SAVE

CANCEL

```
1 select * from Task where done=false
```

RUN QUERY

CLEAR QUERY

[GQL query help](#)

<input type="checkbox"/>	Name/ID ↑	date ↑	description	doi
<input type="checkbox"/>	id=5634161670881280	2019-07-14 (11:49:00.000) BST	Studying for the exam	fal

### Metrics Explorer

**METRIC** VIEW OPTIONS Line

Find resource type and metric ⓘ

Resource type: **Datastore Request** ✕

Select a metric

Metrics

- Index writes**  
datastore.googleapis.com/index/write\_count
- Requests**  
datastore.googleapis.com/api/request\_count
- Sizes of read entities**  
datastore.googleapis.com/entity/read\_sizes
- Sizes of written entities**  
datastore.googleapis.com/entity/write\_sizes

**Metric:** datastore.googleapis.com/index/write\_count  
**Description:** Count of Datastore Index writes.  
**Resource type:** datastore\_request  
**Unit:** number **Kind:** Delta **Value type:** Int64

11 AM 11:05 11:10 11:15