

# 01 Firestore - KirkYagami

---

## 1. Introduction to Firestore

---

- ◆ **Definition:** Firestore is a fully managed, serverless NoSQL database by Google Cloud, designed for high-performance, low-latency mobile, web, and server development. It scales automatically, supports real-time updates, and integrates deeply with other Google Cloud services.
- ◆ **Types of Firestore:**
  - ◆ **Native Mode:** Optimized for mobile/web app development, supporting hierarchical data models and real-time syncing.
  - ◆ **Datastore Mode:** A more traditional database mode that is backward compatible with Google Cloud Datastore, suitable for server applications.

## 2. Core Features

---

- ◆ **Document-Oriented:** Firestore stores data in the form of **documents** which are grouped into **collections**. Documents can contain fields (key-value pairs), arrays, nested objects, and references to other documents.
- ◆ **Scalability:** It automatically scales to handle millions of concurrent connections, transactions, and terabytes of data.
- ◆ **Real-Time Synchronization:** Firestore supports real-time listeners, allowing apps to update in real-time as data changes.
- ◆ **Offline Support:** Both web and mobile SDKs provide local persistence and sync capabilities, enabling apps to work offline.
- ◆ **Security:** Firestore integrates with **Firebase Authentication** for user-based security rules and offers role-based access control via **Google IAM** for enterprise-grade access management.
- ◆ **ACID Transactions:** Supports atomic operations across multiple documents or collections, ensuring data consistency.
- ◆ **Strong Consistency:** Firestore provides strongly consistent reads, ensuring that once data is written, any future read operations reflect the most recent update.

## 3. Firestore Data Model

---

- ◆ **Documents:** The smallest unit of storage in Firestore. A document is essentially a JSON-like structure with a maximum size of 1 MB. It contains fields (key-value pairs) and can also reference other documents.
- ◆ **Collections:** Documents are stored within collections, forming a hierarchy. Collections cannot store primitive data types but only documents.
- ◆ **Subcollections:** Each document can contain one or more subcollections, allowing for hierarchical data structures.

- ◆ **Nested Data:** Documents can include nested objects, arrays, and even references to other documents, making Firestore highly flexible.

## 4. Firestore Architecture

---

- ◆ **Global Distribution:** Firestore is a multi-region, globally distributed database designed to maintain high availability. Data is automatically replicated across regions, ensuring durability and quick access from anywhere in the world.
- ◆ **Serverless:** It removes the need for server management. Google Cloud manages the underlying infrastructure, automatically handling scaling, availability, and updates.
- ◆ **Sharding:** Firestore automatically shards data based on access patterns, ensuring balanced workloads and preventing hotspots, which is essential for large-scale apps with unpredictable query patterns.

## 5. Firestore Querying

---

- ◆ **Indexed Queries:** Firestore indexes every field in a document automatically, allowing for fast queries. It also supports **composite indexes**, where multiple fields are indexed together for complex query performance.
- ◆ **Range Queries:** Supports range queries such as **greater than**, **less than**, **equal to**, and **not equal to** operators.
- ◆ **Complex Queries:**
  - ◆ Compound queries (e.g., combining filters).
  - ◆ Array queries (e.g., checking if a document contains a specific item in an array).
  - ◆ Pagination (via cursors).
  - ◆ Real-time queries, which notify apps of changes to queried data.
- ◆ **Limitations:** Firestore limits querying to ensure scalability (e.g., no support for **OR** queries, limited ability to sort data by multiple fields).

## 6. Firestore Security and Access Control

---

- ◆ **Firebase Security Rules:** Fine-grained control over access to data. You can define rules for reads and writes based on document properties, request properties (like the authenticated user's ID), and more.
- ◆ **Google IAM:** Firestore in Datastore mode uses Google Cloud IAM roles to manage access. This is more suitable for server-side applications where role-based access control (RBAC) is critical.

## 7. Firestore Use Cases

---

- ◆ **Real-Time Applications:** Firestore is ideal for apps that require real-time data synchronization, such as chat applications, collaborative document editing, or live gaming leaderboards.
- ◆ **Mobile and Web Apps:** With strong SDK support for mobile and web development, Firestore is a good fit for apps that require offline access and low-latency data.

synchronization.

- ◆ **E-commerce Platforms:** Firestore's ability to handle large volumes of transactions while ensuring data consistency makes it suitable for e-commerce systems.
- ◆ **IoT and Sensor Data:** Firestore can handle high-velocity data generated from IoT devices, ensuring the data is processed, stored, and synchronized in real-time.
- ◆ **Social Networks:** With its hierarchical data model and real-time features, Firestore is excellent for building social platforms that handle user posts, notifications, and chats.

## 8. Real-World Example: Building a Real-Time Chat Application

---

- ◆ **Requirements:**
  - ◆ Users should be able to send messages and see updates in real-time.
  - ◆ Offline support should ensure messages are saved locally and synchronized when the user is back online.
- ◆ **Firestore Implementation:**
  - ◆ Each chat room is represented by a collection, with each document representing a message.
  - ◆ Use **real-time listeners** to automatically update the chat UI when a new message is sent.
  - ◆ Enable **Firebase Authentication** for user management and apply security rules to ensure only authenticated users can send messages.

## 9. Firestore Pricing

---

- ◆ **Firestore Pricing is based on:**
  - ◆ **Document Reads:** Each time a document is read (e.g., via queries or real-time listeners), you are charged for a read operation.
  - ◆ **Document Writes:** You are charged for each document written or updated.
  - ◆ **Document Deletes:** You are charged when documents are deleted.
  - ◆ **Network Egress:** Charges apply based on the amount of data sent outside Google Cloud regions.
- ◆ **Free Tier:** Firestore offers a generous free tier, making it cost-effective for small applications. The free tier includes a certain number of reads, writes, and deletes per day, along with 1 GB of storage.

## 10. Best Practices for Firestore

---

- ◆ **Minimize Document Size:** Keep documents under the 1 MB size limit, avoiding storing large blobs or deeply nested objects.
- ◆ **Use Batching:** For multiple writes or updates, use **batch operations** to reduce latency and cost.
- ◆ **Optimize Querying:** Avoid using complex queries unnecessarily and leverage Firestore's indexing capabilities to ensure queries are fast.
- ◆ **Data Modeling:** Think of how your data will be queried and design collections/documents accordingly, avoiding unnecessary subcollections or deeply nested structures.

- ♦ **Firestore vs Bigtable:** Use Firestore for real-time applications and Bigtable for high-throughput analytics workloads.

## 11. Conclusion

---

Firestore's ease of use, scalability, real-time capabilities, and strong security make it one of the most powerful NoSQL databases available. It is ideal for modern applications that need real-time updates and seamless integration with other Google Cloud services. Understanding Firestore's architecture, querying capabilities, and best practices is critical for leveraging its full potential in data engineering and cloud application development.



## FireStore

---

- ♦ Only Firestore can be used in a project; Datastore is deprecated.
- ♦ It is a next-generation datastore.
- ♦ Uses a collection and document model.
- ♦ Native mode is Firestore mode.
- ♦ Having more indexes will lead to greater storage sizes.
- ♦ Good replacement for **MongoDB**.

FireStore	RDBMS
Collection	Table
Document ID	PK
Document	Row
Field	Column



## Cloud Firestore

---

### Cloud Firestore:

---

Fully managed, scalable, and serverless document database.

- ♦ **Two modes:**
  - ♦ **Native Mode (Firestore):** Recommended for all servers, mobile apps, and web apps.
  - ♦ **Datastore Mode:** Use Datastore mode if your app requires the Datastore API.



**Serverless: No need to manage infrastructure; Firestore handles it automatically.**

---

- ♦ **Scales up or down:** Firestore automatically scales to handle any demand, from small to large applications.
- ♦ **No maintenance windows or downtime:** Firestore ensures high availability without scheduled downtime.

## Replication:

- ♦ **Automatic regional replication:** Data is automatically replicated within the same region for high availability with SLA availability of 99.99%.
- ♦ **Automatic multi-regional replication:** Data can be replicated across multiple regions for even higher availability with SLA availability of 99.999%.

	Native mode	Datastore mode
	<p>Fully managed, scalable, and serverless document database with offline support and Real-time synchronization.</p> <p><a href="#">Learn more</a> </p> <p><b>SELECT</b></p>	<p>Fully managed, scalable NoSQL database.</p> <p><a href="#">Learn more</a> </p> <p><b>SELECT</b></p>
API	Firestore	Datastore
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 and Firebase	Datastore page in Google Cloud

## Cloud Firestore

- ♦ **A1 Integrations:**
  - ♦ Seamless integrations with AI services with a few clicks.
  - ♦ Enables AI use cases such as automated language translations, image classification, and more.
- ♦ **Strong Consistency:**
  - ♦ Allows running sophisticated ACID transactions against document data.

- ◆ Provides flexibility in structuring data while ensuring data integrity.
- ◆ **Rich Development Library Support:**
  - ◆ **Client Side:** Web, iOS, Android, Flutter, C++, and Unity.
  - ◆ **Server Side:** Node.js, Java, Go, Ruby, and PHP.

## Cloud Firestore

---

- ◆ **Offline Data Persistence Mode (Built-in) and Live Synchronization:**
  - ◆ **Offline Data Persistence:** Cloud Firestore supports offline data persistence, allowing apps to access cached data even when offline.
  - ◆ **Live Synchronization:** Changes made offline are synced with the cloud backend upon reconnection, ensuring data consistency.
- ◆ **Features of Offline Data Persistence:**
  - ◆ **Cached Data Support:** Enables write, read, listen, and query operations even when offline.
  - ◆ **Automatic Sync:** Firestore client library automatically syncs local changes with the cloud backend when online.
  - ◆ **No Code Changes Required:** Enabling offline persistence in Cloud Firestore does not require any code changes; the Firestore client library manages offline and online data access seamlessly.

Cloud Firestore's offline data persistence mode ensures uninterrupted app functionality without internet connectivity by caching data locally. It supports seamless synchronization of local changes with the cloud backend upon reconnection, providing a robust solution for applications requiring reliable offline access.