This page describes how data is replicated in Cloud Spanner, the different types of Cloud Spanner replicas and their roles in reads and writes, and the benefits of replication.

Cloud Spanner automatically gets replication at the byte level from the underlying distributed filesystem that it's built on (as described in Life of Cloud Spanner Reads and Writes (/spanner/docs/whitepapers/life-of-reads-and-writes#aside-distributed-filesystems)). Cloud Spanner writes database mutations to files in this filesystem, and the filesystem takes care of replicating and recovering the files when a machine or disk fails.

Even though the underlying distributed filesystem that Cloud Spanner is built on already provides byte-level replication, Cloud Spanner also replicates data to provide the additional benefits of data availability and geographic locality. At a high level, all data in Cloud Spanner is organized into rows. Cloud Spanner creates multiple copies, or "replicas," of these rows, then stores these replicas in different geographic areas. Cloud Spanner uses a synchronous, Paxos-based replication scheme, in which voting replicas (explained in detail below (#replica_types)) take a vote on every write request before the write is committed. This property of globally synchronous replication gives you the ability to read the most up-to-date data from any Cloud Spanner read-write or read-only replica.

Cloud Spanner creates replicas of each database split (/spanner/docs/schema-and-data-model#database-splits), to tie the above concepts to the terminology and concepts introduced in Schema and data model (/spanner/docs/schema-and-data-model). A split is a set of contiguous rows in a top-level table, where the rows are ordered by primary key. All of the data in a split is physically stored together in the replica, and Cloud Spanner serves each replica out of an independent failure zone.

A set of splits is stored and replicated using Paxos (https://en.wikipedia.org/wiki/Paxos_(computer_science)). Within each Paxos replica set, one replica is elected to act as the **leader**. Leader replicas are responsible for handling writes, while any read-write or read-only replica can serve a read request without communicating with the leader (though if a strong read is requested, the leader will typically be consulted to ensure that the read-only replica has received all recent mutations). More details on the roles of leader and non-leader replicas in reads and writes are discussed below (#replicas_roles_in_writes_and_reads).

As mentioned above, the benefits of Cloud Spanner replication include:

- **Data availability**: Having more copies of your data makes the data more available to clients that want to read it. Also, Cloud Spanner can still serve writes even if some of the replicas are unavailable, because only a majority of voting replicas are required in order to commit a write.
- **Geographic locality**: Having the ability to place data across different regions and continents with Cloud Spanner means data can be geographically closer (and hence faster to access), to the users and services that need it.
- **Single database experience**: Because of the synchronous replication and global strong consistency, at any scale Cloud Spanner behaves the same, delivering a single database experience.
- **Easier application development**: Cloud Spanner's ACID transactions with global strong consistency means developers don't have to add extra logic in the applications to deal with eventual consistency, making application development and subsequent maintenance faster and easier.

Cloud Spanner has three types of replicas: **read-write replicas**, **read-only replicas**, and **witness replicas**. Single-region instances use only read-write replicas, while multi-region instance configurations use a combination of all three types. (For a detailed explanation of why, see: Why read-only and witness replicas? (#aside-why-read-only-and-witness-replicas))

The following table summarizes the types of Cloud Spanner replicas and their properties, and the sections below describe each type in more detail.

Replica type	Can vote	Can become leader	Can serve reads
Read-write	yes	yes	yes
Read-only	no	no	yes
Witness	yes	no	no

Read-write replicas support both reads and writes. These replicas:

- Maintain a full copy of your data.
- Serve reads.
- Can vote whether to commit a write.
- Participate in leadership election.
- Are eligible to become a leader.
- Are the only type used in single-region instances.

Read-only replicas only support reads (not writes). These replicas do not vote for leaders or for committing writes, so they allow you to scale your read capacity without increasing the quorum size needed for writes. Read-only replicas:

- Are only used in multi-region instances.
- Maintain a full copy of your data, which is replicated from read-write replicas.
- Serve reads.
- Do not participate in voting to commit writes. Hence, the location of the read-only replicas never contributes to write latency.
- Can usually serve stale reads without needing a round-trip to the default leader region, assuming staleness is at least 15 seconds. Strong reads may require a round-trip to the leader replica; this communication is handled automatically by the system. (Learn more about stale and strong reads below (#in_reads).)
- Are not eligible to become a leader.

Witness replicas don't support reads but do participate in voting to commit writes. These replicas make it easier to achieve quorums for writes without the storage and compute resources that are required by read-write replicas to store a full copy of data and serve reads. Witness replicas:

- Are only used in multi-region instances.
- Do not maintain a full copy of data.

- Do not serve reads.
- Vote whether to commit writes.
- Participate in leader election but are not eligible to become leader.

This section describes the role of replicas in Cloud Spanner writes and reads, which is helpful in understanding why Cloud Spanner uses read-only and witness replicas in multi-region configurations.

Client write requests always go to the leader replica first, even if there is a non-leader replica that's closer to the client, or if the leader replica is geographically distant from the client.

The leader replica logs the incoming write, and forwards it, in parallel, to the other replicas that are eligible to vote on that write. Each eligible replica completes its write, and then responds back to the leader with a vote on whether the write should be committed. The write is committed when a majority of voting replicas (or "write quorum") agree to commit the write. In the background, all remaining (non-witness) replicas log the write. If a read-write or read-only replica falls behind on logging writes, it can request the missing data from another replica to have a full, up-to-date copy of the data.

Client read requests might be executed at or require communicating with the leader replica, depending on the concurrency mode of the read request.

- Reads that are part of a read-write transaction (/spanner/docs/transactions#read-write_transactions) are served from the leader replica, because the leader replica maintains the locks required to enforce serializability.
- Single read methods (a read outside the context of a transaction) and reads in read-only transactions (/spanner/docs/transactions#read-only_transactions) might require communicating with the leader, depending on the concurrency mode of the read. (Learn more about these concurrency modes in Read types (/spanner/docs/reads#read_types).)
    - Strong read requests can go to any read-write or read-only replica. If the request goes to a non-leader replica, that replica must communicate with the leader in order to execute the read.

- Stale read requests go to the closest available read-only or read-write replica that is caught up to the timestamp of the request. This can be the leader replica if the leader is the closest replica to the client that issued the read request.

egion configurations use a combination of read-write, read-only, and witness replicas, whereas regional configuratio ly read-write replicas. The reasons for this difference have to do with the varying roles of replicas in writes and reads bed above (#replicas_roles_in_writes_and_reads). For writes, Cloud Spanner needs a majority of voting replicas to ag mit in order to commit a mutation. In other words, every write to a Cloud Spanner database requires communication en voting replicas. To minimize the latency of this communication, it is desirable to use the fewest number of voting s, and to place these replicas as close together as possible. That's why regional configurations contain exactly three eplicas, each of which contains a full copy of your data and is able to vote: if one replica fails, the other two can still f quorum, and because replicas in a regional configuration are within the same data center, network latencies are minir

egion configurations contain more replicas by design, and these replicas are in different datacenters (so that clients heir data quickly from more locations). What characteristics should these additional replicas have? They could all be eplicas, but that would be undesirable because adding more read-write replicas to a configuration increases the size quorum (which means potentially higher network latencies due to more replicas communicating with each other, espe eplicas are in geographically distributed locations) and also increases the amount of storage needed (because read- s contain a full copy of data). Instead of using more read-write replicas, multi-region configurations contain two add of replicas that have fewer responsibilities than read-write replicas.

Read-only replicas do not vote for leaders or for committing writes, so they allow you to scale your read capacity wit increasing the quorum size needed for writes.

Witness replicas vote for leaders and for committing writes, but do not store a full copy of the data, can't become the leader, and can't serve reads. They make it easier to achieve quorums for writes without the storage and compute resources that are required by read-write replicas to store a full copy of data and serve reads.