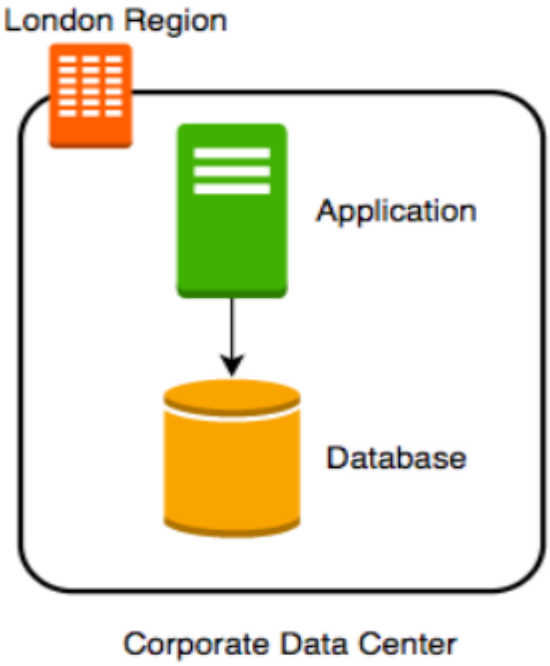
Database Fundamentals

Databases Primer

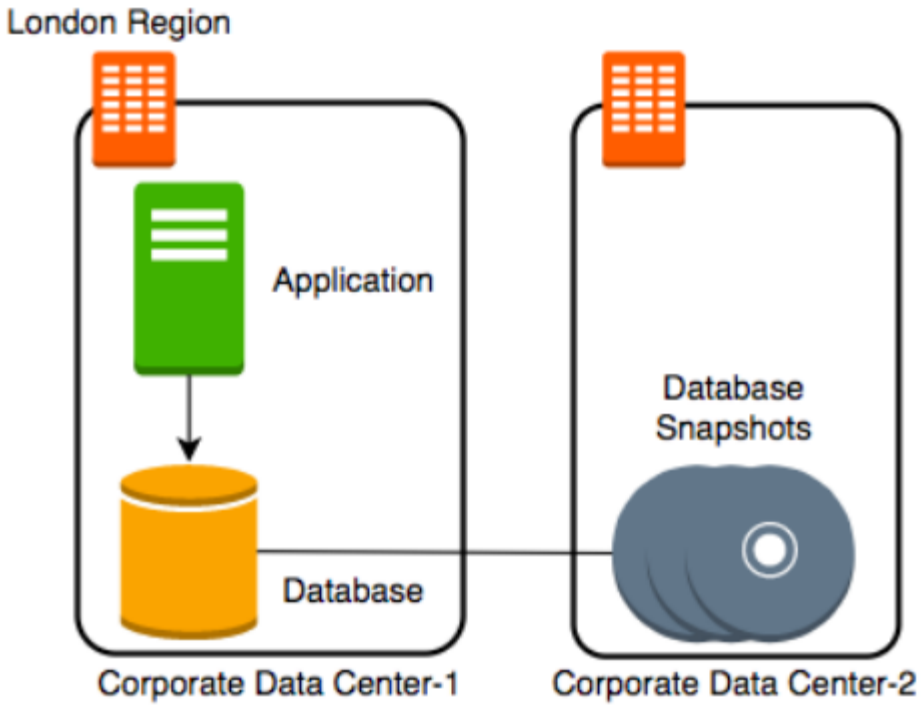
- Databases provide organized and persistent storage for your data.
- To choose between different database types, we would need to understand:
- Availability
- Durability
- RTO
- RPO
- Consistency
- Transactions etc
- Let's get started on a simple journey to understand these

Database - Getting Started



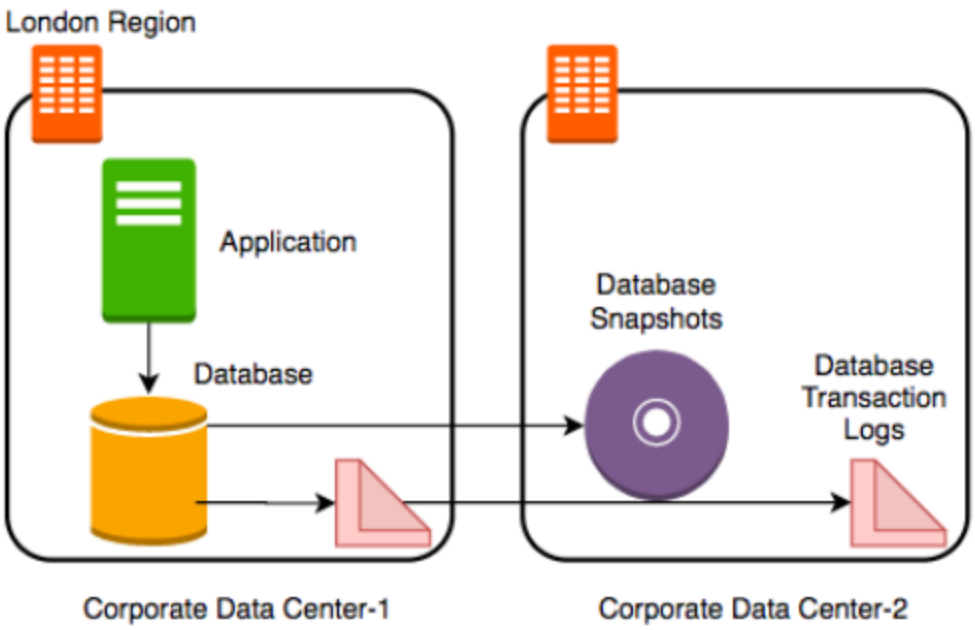
- Imagine a database deployed in a data center in London.
- Let's consider some challenges:
- Challenge 1: Your database will go down if the data center crashes or the server storage fails
- Challenge 2: You will lose data if the database crashes

Database – Snapshots



- Let's automate taking copy of the database (take a snapshot) every hour to another data center in London.
- Let's consider some challenges:
- Challenge 1: Your database will go down if the data center crashes
- Challenge 2 (PARTIALLY SOLVED): You will lose data if the database crashes
- You can setup database from latest snapshot. But depending on when failure occurs you can lose up to an hour of data
- Challenge 3(NEW): Database will be slow when you take snapshots

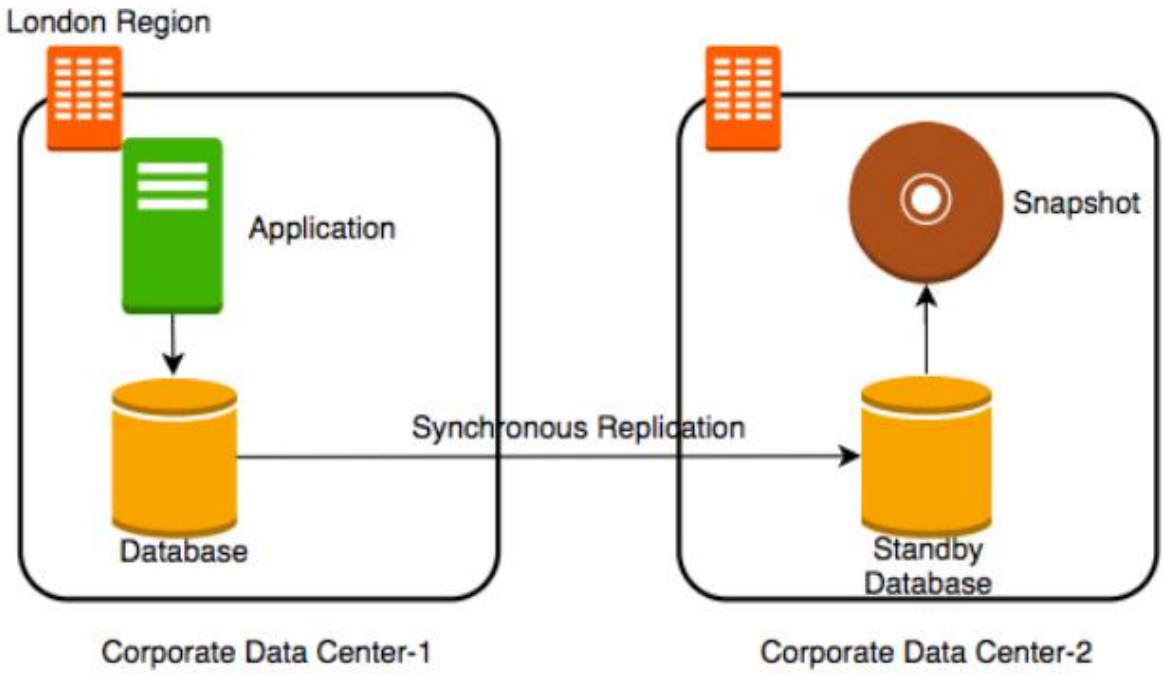
Database - Transaction Logs



- Let's add transaction logs to database and create a process to copy it over to the second data center
- Let's consider some challenges:
- Challenge 1: Your database will go down if the data center crashes
- Challenge 2 (SOLVED): You will lose data if the database crashes
- You can setup database from latest snapshot and apply transaction logs

- Challenge 3: Database will be slow when you take snapshots

Data base - Add a Standby



- L et's add a standby database in the second data center with replication
- Let's consider some challenges:
- Challenge 1 (SOLVED): Your database will go down if t he data center crashes
- You can switch to the stand by database

- Challenge 2 (S O LV E D): You will lose data if the database crashes
- Challenge 3 (S O LV E D): Data base will be slow when you take snapshot s
- Take snapshots from stand by.
- Applications connecting to master will get good perform ance always

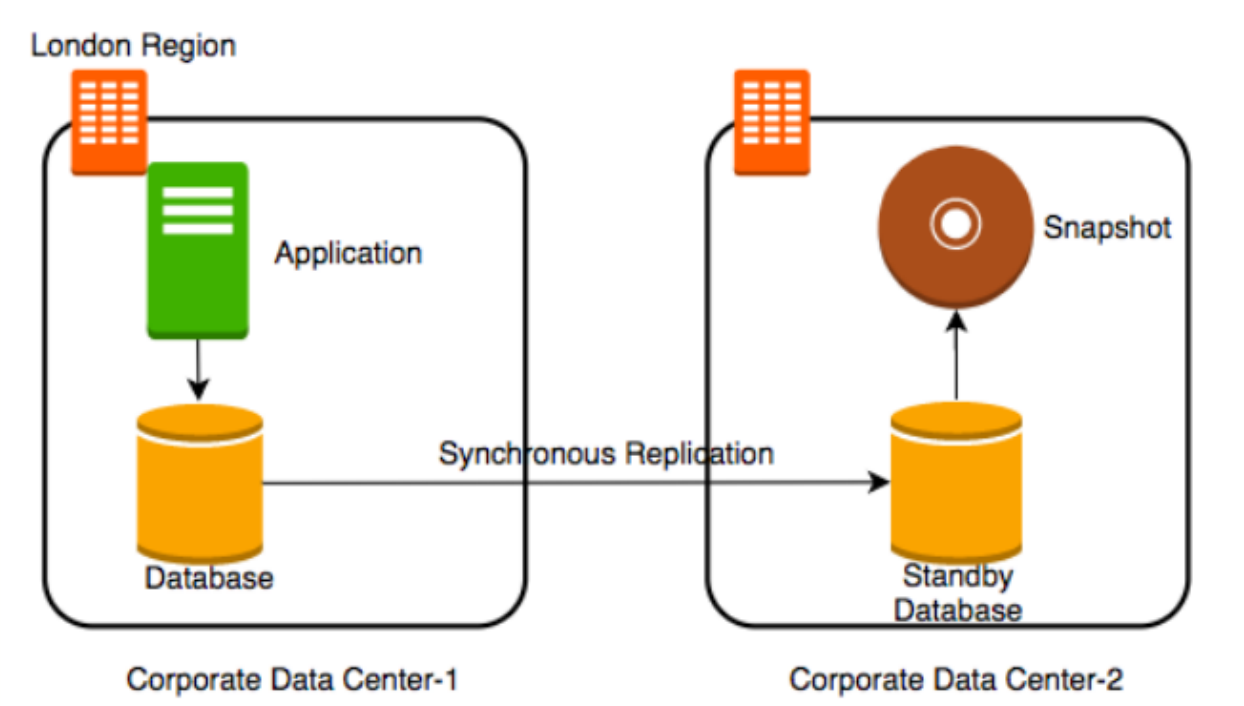
Availability and Durability

- Availability
- Will I be able to access my data now and when I need it?
- Percentage of time an application provides the operations expected of it
- Durability
- Will my data be available after 10 or 100 or 1000 years?
- Examples of measuring availability and durability:
- 4 9's - 99.99
- 11 9's - 99.999999999
- Typically, an availability of four 9's is considered very good
- Typically, a durability of eleven 9's is considered very good

Availability

Availability	Downtime (in a month)	Comment
99.95%	22 minutes	
99.99% (4 9's)	4 and 1/2 minutes	Typically online apps aim for 99.99% (4 9's) availability
99.999% (5 9's)	26 seconds	Achieving 5 9's availability is tough

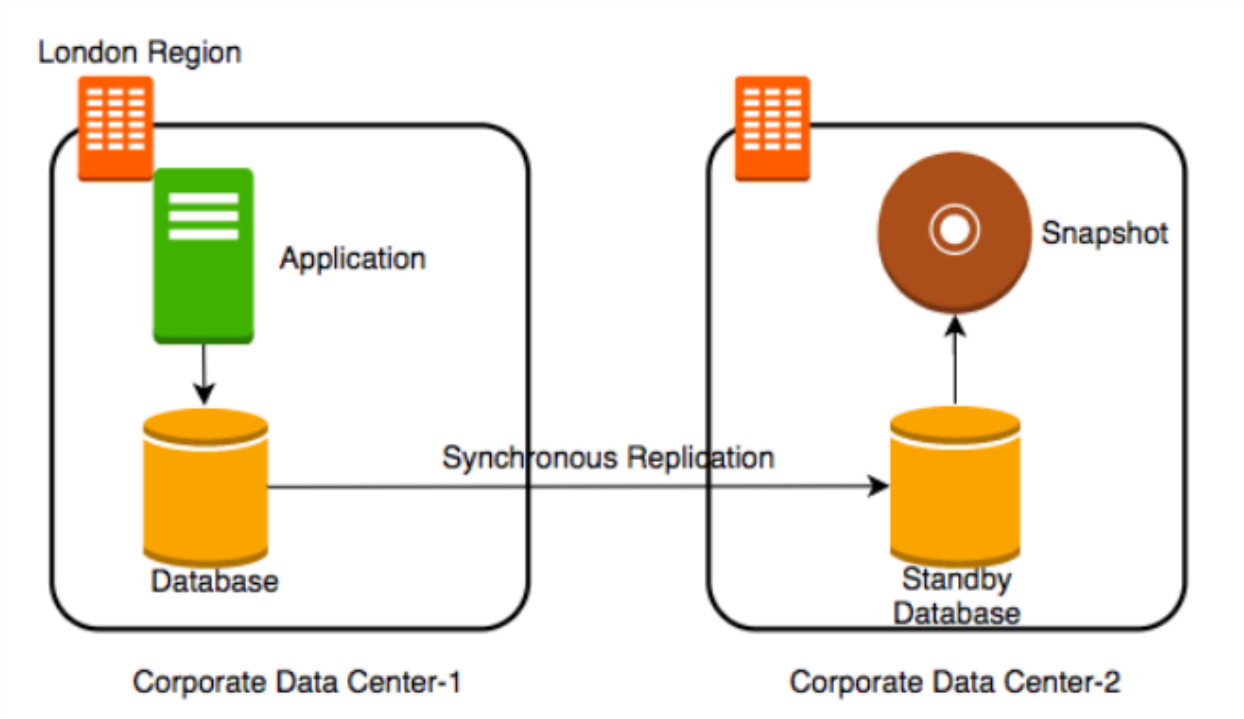
Durability



- What does a durability of 11 9's mean?

- If you store one million files for ten million years, you would expect to lose one file
- Why should durability be high?
- Because we hate losing data
- Once we lose data, it is gone

Increasing Availability and Durability of Databases



Increasing Availability:

- Have multiple standbys available OR distribute the database.
- in multiple Zones
- in multiple Regions
- **Increasing Durability:** Multiple copies of data (standbys, snapshots, transaction logs and replicas)
- in multiple Zones
- in multiple Regions
- **Replicating data** comes with its own challenges!
- We will talk about them a little later

Database Terminology : RTO and RPO

- Imagine a financial transaction being lost
- Imagine a trade being lost
- Imagine a stock exchange going down for an hour
- Typically businesses are fine with some downtime but they hate losing data
- Availability and Durability are technical measures
- How do we measure how quickly we can recover from failure?
- RPO (Recovery Point Objective): Maximum acceptable period of data loss
- RTO (Recovery Time Objective): Maximum acceptable downtime

- Achieving minimum RTO and RPO is expensive
- Trade-off based on the criticality of the data

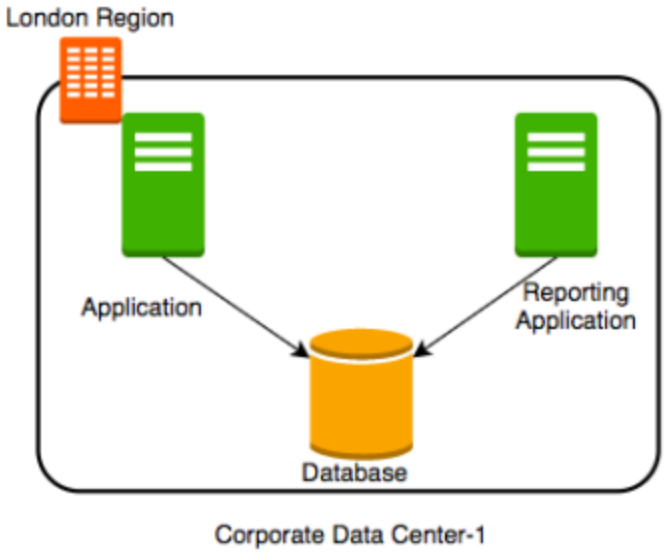
Question - RTO and RPO

- You are running an application in VM instance storing its data on a persistent data storage.
- You are taking snapshots every 48 hours. If the VM instance crashes, you can manually bring it back up in 45 minutes from the snapshot.
- What is your RTO and RPO?
- RTO - 45 minutes
- RPO - 48 hours

Achieving RTO and RPO - Failover Examples

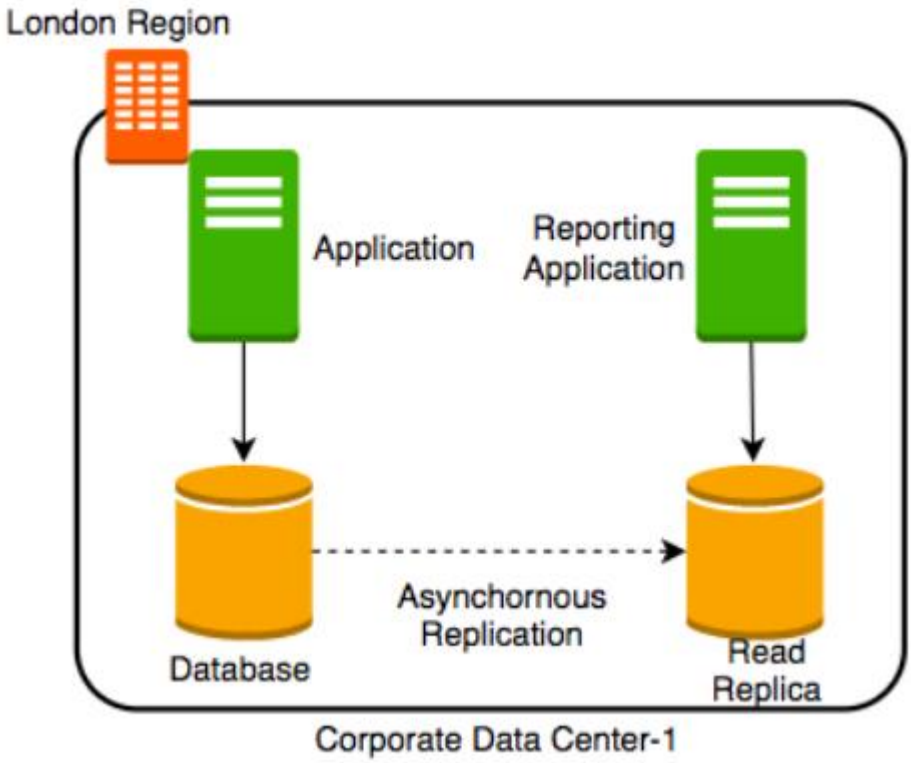
Scenario	Solution
Very small data loss (RPO - 1 minute) **Very small downtime (RTO - 5 minutes)**	**Hot standby** - Automatically synchronize data Have a standby ready to pick up load Use automatic failover from master to standby
Very small data loss (RPO - 1 minute) **BUT I can tolerate some downtimes (RTO - 15 minutes)**	**Warm standby** - Automatically synchronize data Have a standby with minimum infrastructure Scale it up when a failure happens
Data is critical (RPO - 1 minute) but I can tolerate downtime of a few hours (RTO - few hours)	Create regular data **snapshots and transaction logs** Create database from snapshots and transactions logs when a failure happens
Data can be lost without a problem (for example: cached data)	Failover to a completely new server

(New Scenario) Reporting and Analytics Applications



- New reporting and analytics applications are being launched using the same database.
- These applications will ONLY read data.
- Within a few days you see that the database performance is impacted
- How can we fix the problem?
- Vertically scale the database - increase CPU and memory
- Create a database cluster (Distribute the database) - Typically database clusters are expensive to setup
- Create read replicas - Run read only applications against read replicas

Database - Read Replicas



- Add read replica
- Connect reporting and analytics applications to read replica
- Reduces load on the master databases
- Upgrade read replica to master database (supported by some databases)
- Create read replicas in multiple regions
- Take snapshots from read replicas

Consistency

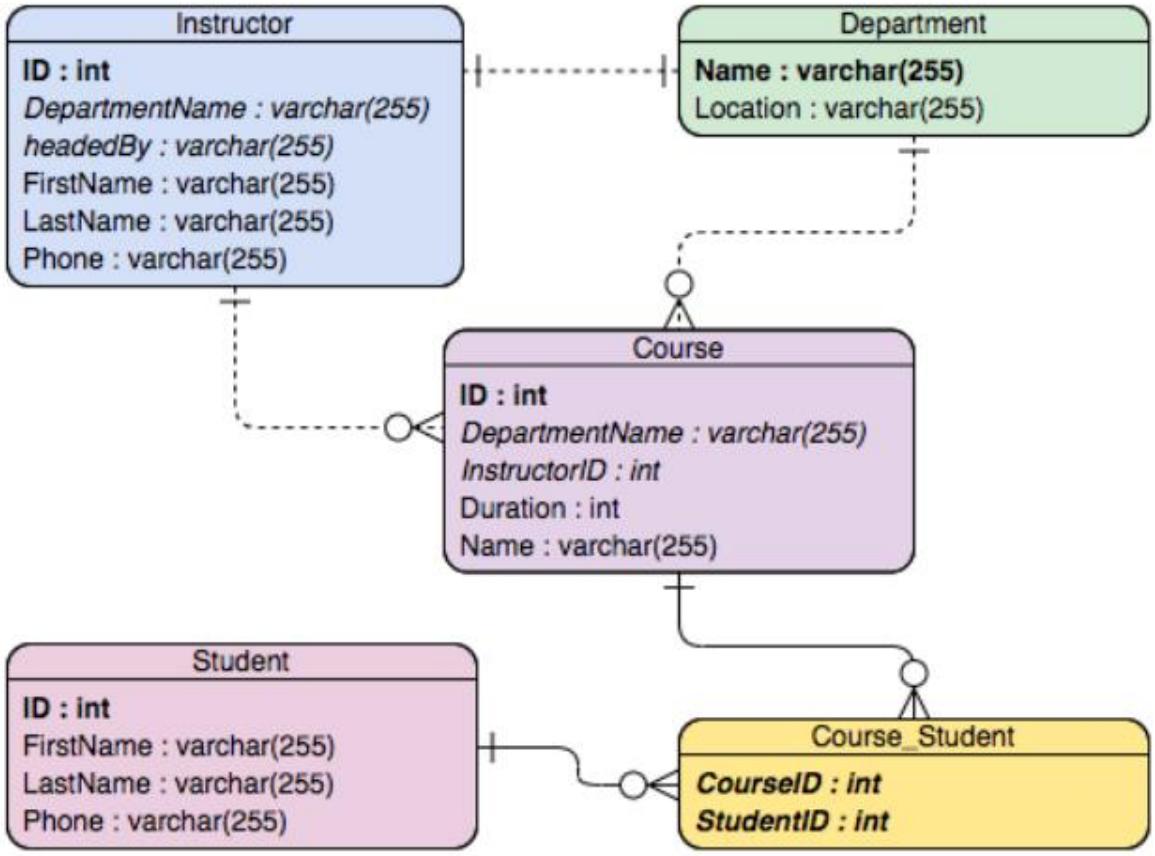
- How do you ensure that data in multiple database instances (standbys and replicas) is updated simultaneously?
- Strong consistency - Synchronous replication to all replicas
- Will be slow if you have multiple replicas or standbys
- Eventual consistency - Asynchronous replication. A little lag - few seconds - before the change is available in all replicas
- In the intermediate period, different replicas might return different values
- Used when scalability is more important than data integrity
- Examples : Social Media Posts - Facebook status messages, Twitter tweets, Linked in posts etc
- Read-after-Write consistency - Inserts are immediately available
- However, updates would have eventual consistency

Database Categories

* There are several categories of databases:

- Relational (OLTP and OLAP), Document, Key Value, Graph, In Memory among others
- Choosing type of database for your use case is not easy. A few factors:
- Do you want a fixed schema?
- Do you want flexibility in defining and changing your schema? (schemaless)
- What level of transaction properties do you need? (atomicity and consistency)
- What kind of latency do you want? (seconds, milliseconds or microseconds)
- How many transactions do you expect? (hundreds or thousands or millions of transactions per second)
- How much data will be stored? (MBs or GBs or TBs or PBs)
- and a lot more...

Relational Databases



- This was the only option until a decade back!
- Most popular (or unpopular) type of databases
- Predefined schema with tables and relationships
- Very strong transactional capabilities
- Used for
- OLTP (Online Transaction Processing) use cases and
- OLAP (Online Analytics Processing) use cases

Relational Database - OLTP (Online Transaction Processing)

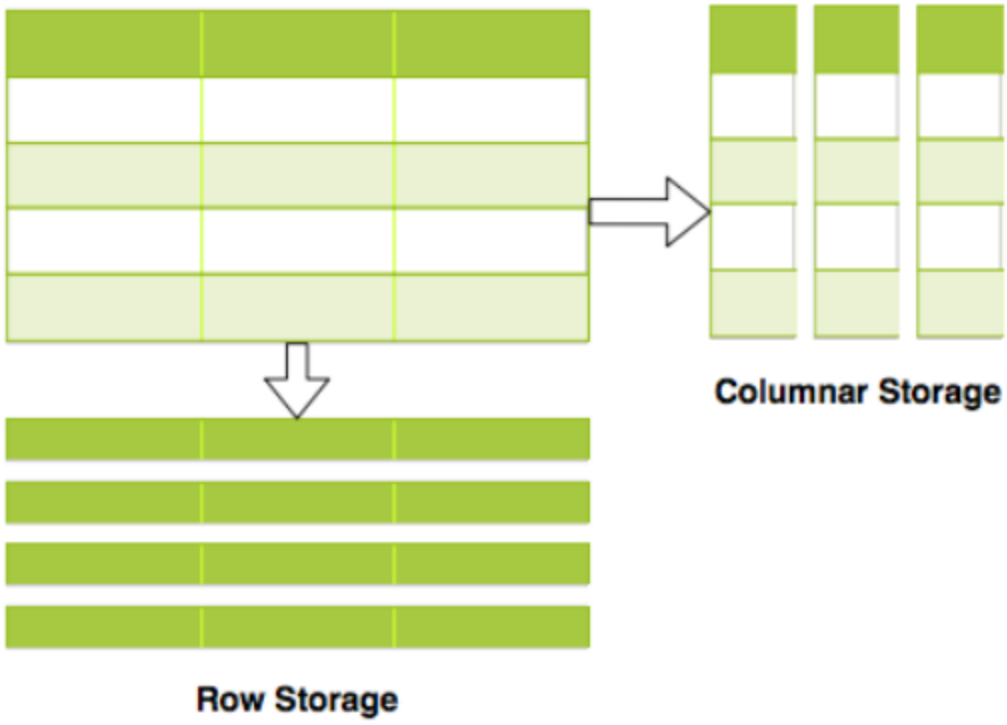
- Applications where large number of users make large number of small transactions
- small data reads, updates and deletes
- Use cases:
- Most traditional applications, ERP, CRM, e-commerce, banking applications
- Popular databases:
- MySQL, Oracle, SQL Server etc
- Recommended Google Managed Services:
- Cloud SQL : Supports PostgreSQL, MySQL, and SQL Server for regional relational databases (upto a few TBs)
- Cloud Spanner: Unlimited scale (multiple PBs) and 99.999% availability for global applications with horizontal scaling

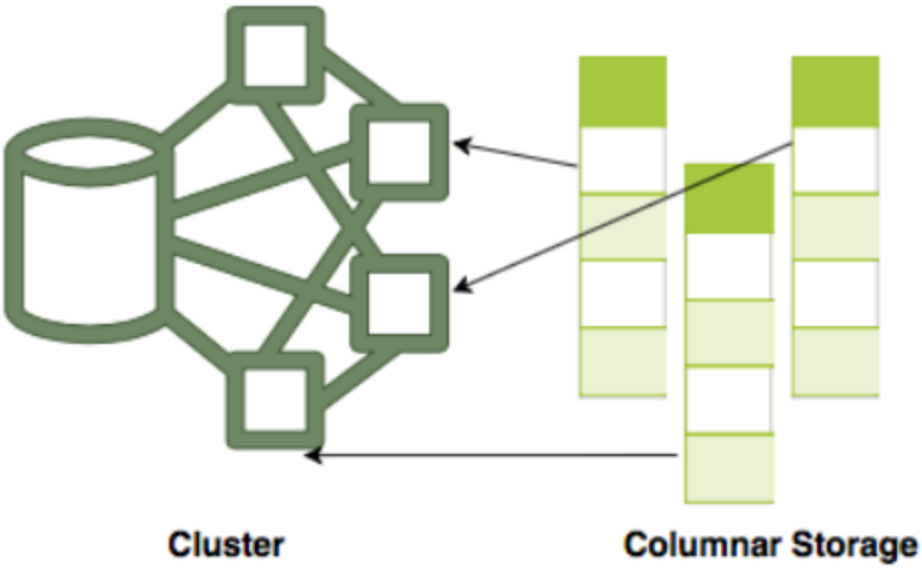
Relational Database - OLAP (Online Analytics Processing)

- Applications allowing users to analyze petabytes of data
- Examples : Reporting applications, Data ware houses, Business intelligence applications, Analytics systems
- Sample application : Decide insurance premiums analyzing data from last hundred years

- Data is consolidated from multiple (transactional) databases
- Recommended GCP Managed Service
- BigQuery: Petabyte-scale distributed data ware house

Relational Databases - OLAP vs OLTP





- OLAP and OLTP use similar data structures

- BUT very different approach in how data is stored
- OLTP databases use row storage
- Each table row is stored together
- Efficient for processing small transactions
- OLAP databases use columnar storage
- Each table column is stored together
- High compression - store petabytes of data efficiently
- Distribute data - one table in multiple cluster nodes
- Execute single query across multiple nodes - Complex queries can be executed efficiently.

NoSQL Databases

- New approach (actually NOT so new!) to building your databases
- NoSQL = not only SQL
- Flexible schema
- Structure data the way your application needs it
- Let the schema evolve with time
- Horizontally scale to petabytes of data with millions of TPS
- NOT a 100% accurate generalization but a great starting point:

- Typical NoSQL databases trade-off "Strong consistency and SQL features" to achieve "scalability and high-performance"
- Google Managed Services:
- Cloud Firestore (Datastore)
- Cloud BigTable

Cloud Firestore (Datastore) vs Cloud BigTable

- Cloud Datastore - Managed serverless NoSQL document database
- Provides ACID transactions, SQL-like queries, indexes
- Designed for transactional mobile and web applications
- Firestore (next version of Datastore) adds:
- Strong consistency
- Mobile and Web client libraries
- Recommended for small to medium databases (0 to a few Terabytes)
- Cloud BigTable - Managed, scalable NoSQL wide column database
- NOT serverless (You need to create instances)
- Recommend for data size > 10 Terabytes to several Petabytes
- Recommended for large analytical and operational workloads:

- NOT recommended for transactional workloads (Does NOT support multi row transactions - supports ONLY Single-row transactions)

In-memory Databases

- Retrieving data from memory is much faster than retrieving data from disk.
- In-memory databases like Redis deliver microsecond latency by storing persistent data in memory.
- Recommended GCP Managed Service
- Memory Store
- Use cases : Caching, session management, gaming leader boards, geospatial applications.

Databases – Summary

Database Type	GCP Services	Description
Relational OLTP databases	Cloud SQL, Cloud Spanner	Transactional usecases needing **predefined schema** and very **strong transactional** capabilities (Row storage) **Cloud SQL**: MySQL, PostgreSQL, SQL server DBs **Cloud Spanner**: Unlimited scale and 99.999% availability for global applications with horizontal scaling
Relational OLAP databases	BigQuery	Columnar storage with predefined schema. Datawarehousing & BigData workloads
NoSQL Databases	Cloud Firestore (Datastore) , Cloud BigTable	Apps needing **quickly evolving** structure (**schema-less**) **Cloud Firestore** - Serverless transactional document DB supporting mobile & web apps. Small to medium DBs (0 - few TBs) **Cloud BigTable** - Large databases(10 TB - PBs). Streaming (IOT), analytical & operational workloads. NOT serverless.
In memory databases/caches	Cloud Memorystore	Applications needing **microsecond** responses

Databases – Scenarios

Scenario	Solution
A start up with quickly evolving schema (table structure)	Cloud Datastore/Firestore
Non relational db with less storage (10 GB)	Cloud Datastore
Transactional global database with predefined schema needing to process million of transactions per second	CloudSpanner
Transactional local database processing thousands of transactions per second	Cloud SQL
Cache data (from database) for a web application	MemoryStore
Database for analytics processing of petabytes of data	BigQuery
Database for storing huge volumes stream data from IOT devices	BigTable
Database for storing huge streams of time series data	BigTable

Cloud SQL

- Fully Managed Relational Database service.
- Configure your needs and do NOT worry about managing the database.
- Supports MySQL, PostgreSQL, and SQL Server.
- Regional Service providing High Availability (99.95%).
- Use SSDs or HDDs (For best performance: use SSDs).
- Upto 416 GB of RAM and 30 TB of data storage.
- Use Cloud SQL for simple relational use cases:
- To migrate local MySQL, PostgreSQL, and SQL Server databases.

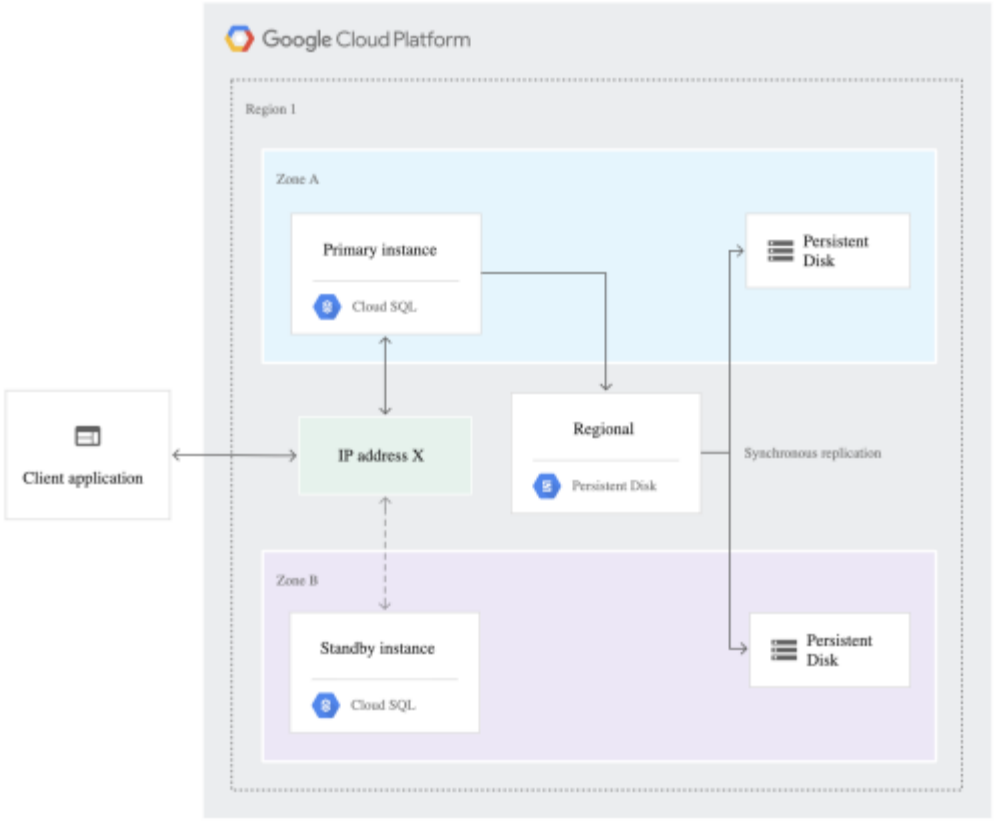
- To reduce your maintenance cost for a simple relational database.
- (REMEMBER) Use Cloud Spanner(Expensive $$$$) instead of Cloud SQL if:
- You have huge volumes of relational data (TBs) OR
- You need infinite scaling for a growing application (to TBs) OR
- You need a Global (distributed across multiple regions) Database OR
- You need higher availability (99.999%)

Cloud SQL – Features

- Important Cloud SQL Features:
- Automatic encryption (tables/backups), maintenance and updates
- High availability and failover:
- Create a Standby with automatic failover
- Pre requisites: Automated backups and Binary logging
- Read replicas for read workloads:
- Options: Cross-zone, Cross-region and External (NON Cloud SQL DB)
- Pre requisites: Automated backups and Binary logging
- Automatic storage increase without downtime (for newer versions)
- Point-in-time recovery: Enable binary logging

- Backups (Automated and on-demand backups)
- Supports migration from other sources
- Use Database Migration Service (DMS)
- You can export data from UI (console) or gcloud with formats:
- SQL (Recommended if you import data into other databases) and CSV

Cloud SQL - High Availability



- Create a High Availability (HA) Configuration.
- Choose Primary and Secondary zones within a region.

- You will have two instances : Primary and Secondary instances.
- Changes from primary are replicated synchronously to secondary.
- In case of Zonal failure, automatic failover to secondary instance:
- If Primary zone becomes available, failover does not revert automatically
- (Remember) High Availability setup CANNOT be used as a Read Replica.

Cloud Spanner

- Fully managed, mission critical, relational(SQL), globally distributed database with VERY high availability (99.999%).
- Strong transactional consistency at global scale.
- Scales to PBs of data with automatic sharding.
- Cloud Spanner scales horizontally for reads and writes.
- Configure no of nodes.
- (REMEMBER) In comparison, Cloud SQL provides read replicas:
- BUT you cannot horizontally scale write operations with Cloud SQL!
- Regional and Multi-Regional configurations
- Expensive (compared to Cloud SQL): Pay for nodes & storage

- Data Export: Use Cloud Console to export data
- Other option is to use Data flow to automate export
- No gcloud export option

Cloud Datastore and Firestore

- Datastore - Highly scalable NoSQL Document Database.
- Automatically scales and partitions data as it grows
- Recommended for upto a few TBs of data
- For bigger volumes, BigTable is recommended
- Supports Transactions, Indexes and SQL like queries (GQL)
- Does NOT support Joins or Aggregate (sum or count) operations
- For use cases needing flexible schema with transactions
- Examples: User Profile and Product Catalogs
- Structure: Kind > Entity (Use namespaces to group entities)
- You can export data ONLY from gcloud (NOT from cloud console)
- Export contains a metadata file and a folder with the data
- Firestore = Datastore++ : Optimized for multi device access
- Offline mode and data synchronization across multiple devices - mobile, IOT etc
- Provides client side libraries - Web, iOS, Android and more

- Offers Datastore and Native modes

Cloud BigTable

- Petabyte scale, wide column NoSQL DB (HBase API compatible)
- Designed for huge volumes of analytical and operational data
- IOT Streams, Analytics, Time Series Data etc
- Handle millions of read/write TPS at very low latency
- Single row transactions (multi row transactions NOT supported)
- NOT serverless: You need to create a server instance (Use SSD or HDD)
- Scale horizontally with multiple nodes (No downtime for cluster resizing)
- CANNOT export data using cloud console or gcloud:
- Either use a Java application (java -jar JAR export\import) OR
- Use HBase commands
- Use cbt command line tool to work with BigTable (NOT gcloud)
- Ex: cbt createtable my-table

Cloud BigTable - Wide Column Database

Rowid	Column Family 1			Column Family 2			Column Family 3		
	col1	col2	col3	col1	col2	col3	col1	col2	col3
1									
2									
3									

- At the most basic level, each table is a sorted key/value map
- Each value in a row is indexed using a key - row key
- Related columns are grouped into column families
- Each column is identified by using column-family:column-qualifer(or name)
- This structure supports high read and write throughput at low latency
- Advantages : Scalable to petabytes of data with millisecond responses upto millions of TPS
- Use cases : IOT streams, graph data and real time analytics (time-series data, financial data - transaction histories, stock prices etc)
- Cloud Dataflow : Used to export data from BigTable to CloudStorage

- For ex: Cloud Bigtable to Cloud Storage Avro, Cloud Bigtable to Cloud Storage Parquet

Memorystore

- In-memory datastore service: Reduce access times
- Fully managed (Provisioning, Replication, Failover and Patching)
- Highly available with 99.9% availability SLA
- Monitoring can be easily setup using Cloud Monitoring
- Support for Redis and Memcached:
- Use Memcached for Caching
- Reference data, database query caching, session store etc
- Use Redis for low latency access with persistence and high availability
- Gaming Leader Boards, Player Profiles, In memory Stream Processing etc

BigQuery – Datawarehouse

- Exabyte scale modern Datawarehousing solution from GCP
- Relational database (SQL, schema, consistency etc)
- Use SQL-like commands to query massive datasets
- Traditional (Storage + Compute) + Modern (Realtime + Serverless)

- When we are talking about a Datawarehouse, importing and exporting data (and formats) becomes very important:
- Load data from a variety of sources, incl. streaming data
- Variety of import formats - CSV/JSON/Avro/Parquet/ORC/Datastore backup
- Export to Cloud Storage (long term storage) & Data Studio (visualization)
- Formats - CSV/JSON (with Gzip compression), Avro (with deflate or snappy compression)
- Automatically expire data (Configurable Table Expiration)
- Query external data sources without storing data in BigQuery
- Cloud Storage, Cloud SQL, BigTable, Google Drive
- Use Permanent or Temporary external tables

BigQuery - Accessing and Querying Data

- Access databases using:
- Cloud Console
- bq command-line tool (NOT gcloud)
- BigQuery Rest API OR
- HBase API based libraries (Java, .NET & Python)
- (Remember) BigQuery queries can be expensive as you are running them on large data sets!

- (BEST PRACTICE) Estimate BigQuery queries before running:
- 1: Use UI(console)/bq(--dry-run) - Get scanned data volume (estimate)
- 2: Use Pricing Calculator: Find price for scanning 1 MB data. Calculate cost.

Relational Database - Import and Export

- Cloud SQL : to/from Cloud Storage (gcloud sql export/import csv/sql)
- From Console/gcloud/REST API
- SQL and CSV formats
- For large databases, use serverless mode
- Reduces performance impact of export on the live database
- Cloud Spanner: to/from Cloud Storage
- From Console (uses Cloud Data Flow)
- BigQuery: to/from Cloud Storage and Others (bq extract/load)
- From Console/bq
- Formats - CSV/JSON (with Gzip compression), Avro (with deflate or snappy compression)
- Variety of options to import data:

- Load data from Cloud Storage
- Example Use Case: Data Store > Cloud Storage > Big Query
- Batch Loading with BigQuery Data Transfer Service
- Use Dataflow to setup streaming pipeline

NoSQL Databases - Import and Export

* Cloud Datastore/Firestore: to/from Cloud Storage

- From Console/gcloud/REST API
- gcloud datastore/firestore export/import --kinds --namespaces
- Cloud BigTable: to/from Cloud Storage
- Create Dataflow jobs
- Formats: Avro/Parquet/SequenceFiles
- (REMEMBER) Ensure that service accounts have access to Cloud Storage Buckets
- ACL (gsutil acl ch -U SERVICE_ACCOUNT :W BUCKET) OR
- Roles Storage Admin or Storage Object Admin or Storage Object Creator

Cloud SQL – CommandLine

* gcloud sql

- gcloud sql instances create/clone/delete/describe/patch

- gcloud sql instances create INSTANCE
- gcloud sql instances patch --backup-start-time
- gcloud sql databases create/delete/describe/list/patch
- gcloud sql databases create DATABASE --instance=INSTANCE
- gcloud sql connect INSTANCE [--database=DATABASE --user=root]
- gcloud sql backups create/describe/list
- gcloud sql backups create --async --instance [INSTANCE] (one time backup)

BigQuery - Command Line

- BigQuery (bq)
- bq show bigquery-public-data:samples.shakespeare
- bq query 'QUERY-STRING'
- --dry-run - To estimate the bytes scanned by a query
- bq extract (export data)
- bq load (load data)
- (Remember) Use the standard way to set the project gcloud config set project my-project

cbt tool - Cloud Bigtable – CommandLine

- cbt - CLI for Cloud Bigtable (NOT gcloud)

- Installing - gcloud components install cbt
- Verify Installation - cbt listinstances
- Create .cbtrc file with the configuration
- echo project = project-id > ~/.cbtrc
- echo instance = quickstart-instance >> ~/.cbtrc
- Commands (cbt):
- cbt createinstance - Create an instance
- cbt createcluster - Create a cluster within configured instance!
- cbt createtable/deleteinstance/deletecluster/deletetable
- cbt listinstances/listclusters
- cbt ls - list tables and column families
- (Remember) You can configure your project with cbt in .cbtrc file

Databases – Remember

- (Remember) BigQuery, Datastore, Firebase does NOT need VM configuration.
- whereas Cloud SQL and BigTable need VM configuration
- Relational Databases
- Small Local Databases - Cloud SQL
- Highly scalable global databases - Cloud Spanner
- Datawarehouse - BigQuery

- NoSQL Databases Transactional database for a few Terabytes of data - Cloud Datastore
- Huge volumes of IOT or streaming analytics data - Cloud BigTable