# Chapter 2 Building and Operationalizing Storage Systems

**Google Cloud Professional Data Engineer Exam objectives covered in this chapter include the following:**

1. **Designing data processing systems**
1. ✓ **1.2 Designing data pipelines. Considerations include:**
1. Data publishing and visualization (e.g., BigQuery)
2. Batch and streaming data (e.g., Cloud Dataflow, Cloud Dataproc, Apache Beam, Apache Spark and Hadoop ecosystem, Cloud Pub/Sub, Apache Kafka)
3. Online (interactive) vs. batch predictions
4. Job automation and orchestration (e.g., Cloud Composer)

Not only are data engineers expected to understand when to use various ingestion and storage technologies, but they should also be familiar with deploying and operating those systems. In this chapter, you will learn how to deploy storage systems and perform data management operations, such as importing and exporting data, configuring access controls, and doing performance tuning. The services included in this chapter are as follows:

- Cloud SQL
- Cloud Spanner
- Cloud Bigtable
- Cloud Firestore
- BigQuery
- Cloud Memorystore
- Cloud Storage

The chapter also includes a discussion of working with unmanaged databases, storage costs and performance, and data lifecycle management.

## Cloud SQL

*Cloud SQL* is a fully managed relational database service that supports MySQL, PostgreSQL, and SQL Server databases. As of this writing, SQL Server is in beta release. A managed database is one that does not require as much administration and operational support as an unmanaged database because Google will take care of core operational tasks, such as creating databases, performing backups, and updating the operating system of database instances. Google also manages scaling disks, configuring for failover, monitoring, and authorizing network connections.

Cloud SQL supports regional-level databases of up to 30 TB. If you need to store more data or need multi-regional support, consider using Cloud Spanner.

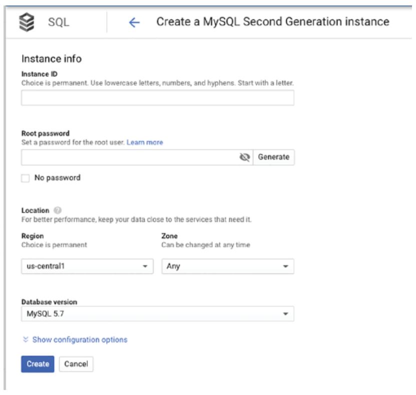
> **NOTE** Cloud SQL currently offers First Generation and Second Generation instances. Second Generation instances are generally faster, larger, and less costly than First Generation instances. Support for First Generation ended on January 30, 2020. Unless otherwise stated, the following will describe Second Generation features and operations.

### CONFIGURING CLOUD SQL

The first step to deploying a Cloud SQL database is choosing which of the three relational database management systems (RDBMSs) you will use. The configuration process is similar for the various types of databases and include specifying the following:

- An instance ID
- A password
- A region and zone
- A database version

With Cloud SQL, you can choose your machine type, which determines the number of virtual CPUs (vCPUs) and the amount of memory available. Second Generation instances support up to 64 CPUs and 416 GB of memory. Note that these limits may have changed by the time you read this (see Figure 2.1).



SQL — Create a MySQL Second Generation instance

Instance info

**Instance ID**
Choice is permanent. Use lowercase letters, numbers, and hyphens. Start with a letter.

**Root password**
Set a password for the root user. Learn more
Generate

☐ No password

**Location**
For better performance, keep your data close to the services that need it.

**Region**
Choice is permanent
us-central1

**Zone**
Can be changed at any time
Any

**Database version**
MySQL 5.7

Show configuration options

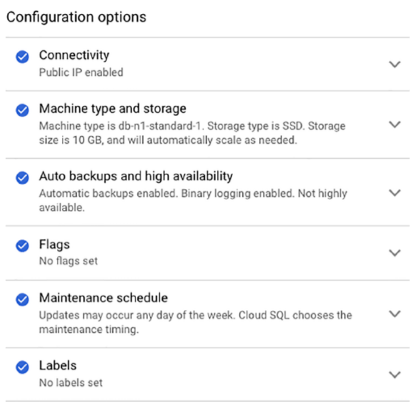
Create Cancel

**Figure 2.1** Basic Cloud SQL configuration

Optionally, you can also specify the use of a public or private IP address for the instance; a public IP is assigned by default. When using a public IP, you will need to specify one or more external networks that connect to the database instance. Alternatively, you could connect to the database using Cloud SQL Proxy, which provides secure access to Second Generation instances without having to create allow lists or to configure SSL. The proxy manages authentication and automatically encrypts data.

Cloud SQL will perform daily backups, and you can specify a four-hour time window during the time that the backup will occur. You can also specify the day and time for a maintenance window. Backups are deleted when an instance is deleted. Keep in mind that you cannot restore individual tables from a backup. To maintain a copy of the data after an instance is deleted or to restore individual tables or databases, use exports, which are discussed in a moment.

Although Google manages the creation of a database, you do have the option of specifying database flags, which are RDBMS specific (see Figure 2.2).



Configuration options

Connectivity
Public IP enabled

Machine type and storage
Machine type is db-n1-standard-1. Storage type is SSD. Storage size is 10 GB, and will automatically scale as needed.

Auto backups and high availability
Automatic backups enabled. Binary logging enabled. Not highly available.

Flags
No flags set

Maintenance schedule
Updates may occur any day of the week. Cloud SQL chooses the maintenance timing.

Labels
No labels set

**Figure 2.2** Optional configuration parameters in Cloud SQL

By default, a Cloud SQL instance is created with one instance in a single zone. If you want to be sure that your application can still access data when a single instance is down for any reason, which is recommended for production systems, you should choose the High Availability option. That will create a second instance in a second zone and synchronously replicate data from the primary instance to the standby instance. If the primary instance becomes nonresponsive, Cloud SQL will fail over or switch to using the secondary instance.

It is a good practice to shard your data into smaller databases rather than have a single monolithic database. The best way to shard will depend on the use case. If data is often queried using time as a selection criterion, sharding by time makes sense. In other cases, sharding by some other attribute, such as geographic region, might make more sense. Smaller databases are easier to manage. Also, if there is an incident involving one instance, the impact is limited to data on the one instance.

Databases should not have more than 10,000 tables. All tables should have a primary key to facilitate row-based replication.

#### IMPROVING READ PERFORMANCE WITH READ REPLICAS

You can improve the performance of read-heavy databases by using read replicas. A *read replica* is a copy of the primary instance's data that is maintained in the same region as the primary instance. Applications can read from the read replica and allow the primary instance to handle write operations. Here are some things to keep in mind about read replicas:

- Read replicas are not designed for high availability; Cloud SQL does not fail over to a read replica.
- Binary logging must be enabled to support read replicas.
- Maintenance on read replicas is not limited to maintenance windows—it can occur at any time and disrupt operations.
- A primary instance can have multiple read replicas, but Cloud SQL does not load-balance between them.
- You cannot perform backups on a read replica.
- Read replicas can be promoted to a standalone Cloud SQL. The operation is irreversible.
- The primary instance cannot be restored from backup if a read replica exists; it will have to be deleted or promoted.
- Read replicas must be in the same region as the primary instance.

#### IMPORTING AND EXPORTING DATA

Data can be imported and exported from Cloud SQL databases. Each RDBMS has an import and an export program.

- MySQL provides `mysqldump`.
- PostgreSQL provides `pg_dump`.
- SQL Server provides `bcp`, which stands for bulk copy program.

In general, data is imported from and exported to Cloud Storage buckets. Files can be in either CSV or SQL dump format.

It should be noted that when you are using the command line to export a database, you must use particular flags to create a file that can later be imported. For Cloud SQL to import correctly, you will need to exclude views, triggers, stored procedures, and functions. Those database objects will have to be re-created using scripts after a database has been imported.

Both SQL Dump and CSV formats can be compressed to save on storage costs. It is possible to import from a compressed file and export to a compressed file.

Cloud SQL is well suited for regional applications that do not need to store more than 30 TB of data in a single instance. Cloud Spanner is used for more demanding database systems.

### Cloud Spanner

*Cloud Spanner* is Google's relational, horizontally scalable, global database. It is a relational database, so it supports fixed schemas and is ANSI SQL 2011 compliant. Cloud Spanner provides strong consistency, so all parallel processes see the same state of the database. This consistency is different from NoSQL databases, which are generally eventually consistent, allowing parallel processes to see different states of the database.

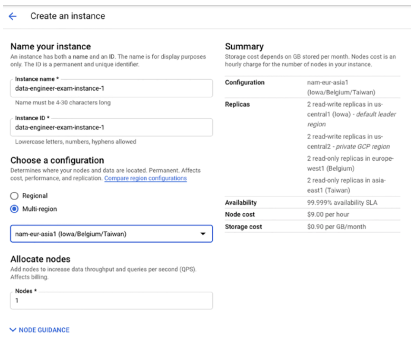
Cloud Spanner is highly available and does not require failover instances in the way that Cloud SQL does. It also manages automatic replication.

#### CONFIGURING CLOUD SPANNER

Cloud Spanner configuration is similar to Cloud SQL. When creating a Cloud Spanner instance, you specify the following:

- An instance name
- An instance ID
- Regional or Multi-Regional
- Number of nodes
- Region in which to create the instance

The choice of regional or multi-regional and the number of nodes determines the cost of an instance. A single node in a regional instance in us-west1 currently costs $0.90/hour, whereas a multi-regional instance in nam-eur-asia1 costs $9.00/hour. The number of nodes you choose is usually dictated by your workload requirements. Google recommends keeping CPU utilization below 65 percent in regional instances and below 45 percent in multi-regional instances. Also, each node can store up to 2 TB of data (see Figure 2.3).



Create an instance

**Name your instance**
An instance has both a name and an ID. The name is for display purposes only. The ID is a permanent and unique identifier.

Instance name *: data-engineer-exam-instance-1
Name must be 4-30 characters long

Instance ID *: data-engineer-exam-instance-1
Lowercase letters, numbers, hyphens allowed

**Choose a configuration**
Determines where your nodes and data are located. Permanent. Affects cost, performance, and replication. Compare region configurations

○ Regional
● Multi-region

nam-eur-asia1 (Iowa/Belgium/Taiwan)

**Allocate nodes**
Add nodes to increase data throughput and queries per second (QPS). Affects billing.

Nodes *: 1

NODE GUIDANCE

**Summary**
Storage cost depends on GB stored per month. Nodes cost is an hourly charge for the number of nodes in your instance.

| | |
|---|---|
| **Configuration** | nam-eur-asia1 (Iowa/Belgium/Taiwan) |
| **Replicas** | 2 read-write replicas in us-central1 (Iowa) - *default leader region*<br>2 read-write replicas in us-central2 - *private GCP region*<br>2 read-only replicas in europe-west1 (Belgium)<br>2 read-only replicas in asia-east1 (Taiwan) |
| **Availability** | 99.999% availability SLA |
| **Node cost** | $9.00 per hour |
| **Storage cost** | $0.90 per GB/month |

**Figure 2.3** Configuring Cloud Spanner

## REPLICATION IN CLOUD SPANNER

Cloud Spanner maintains multiple replicas of rows of data in multiple locations. Since Cloud Spanner implements globally synchronous replication, you can read from any replica to get the latest data in a row. Rows are organized into splits, which are contiguous blocks of rows that are replicated together. One of the replicas is designated the leader and is responsible for write operations. The use of replicas improves data availability as well as reducing latency because of geographic proximity of data to applications that need the data. Again, the benefit of adding nodes should be balanced with the price of those additional nodes.

The distributed nature of Cloud Spanner creates challenges for writing data. To keep all replicas synchronized, Cloud Spanner uses a voting mechanism to determine writes. Cloud Spanner uses a voting mechanism to determine the latest write-in case of a conflict value.

There are three types of replicas in Cloud Spanner:

- Read-write replicas
- Read-only replicas
- Witness replicas

Regional instances use only read-only replicas; multi-regional instances use all three types. Read-write replicas maintain full copies of data and serve read operations, and they can vote on write operations. Read-only replicas maintain full copies of data and serve read operations, but they do not vote on write operations. Witness replicas do not keep full copies of data but do participate in write votes. Witness replicas are helpful in achieving a quorum when voting.

Regional instances maintain three read-write replicas. In multi-regional instances, two regions are considered read-write regions and contain two replicas. One of those replicas is considered the leader replica. A witness replica is placed in a third region.

## DATABASE DESIGN CONSIDERATIONS

Like NoSQL databases, Cloud Spanner can have hotspots where many read or write operations are happening on the same node instead of in parallel across multiple nodes. This can occur using sequential primary keys, such as auto-incrementing counters or timestamps. If you want to store sequential values and use them for primary keys, consider using the hash of the sequential value instead. That will evenly distribute writes across multiple nodes. This is because a hash value will produce apparently random values for each input, and even a small difference in an input can lead to significantly different hash values.

Relational databases are often normalized, and this means that joins are performed when retrieving data. For example, orders and order line items are typically stored in different tables. If the data in different tables is stored in different locations on the persistent data store, the database will spend time reading data blocks from two different areas of storage. Cloud Spanner allows for interleaving data from different tables. This means that an order row will be stored together with order line items for that order. You can take advantage of interleaving by specifying a parent-child relationship between tables when creating the database schema.

Please note that row size should be limited to 4 GB, and that includes any interleaved rows.

## IMPORTING AND EXPORTING DATA

Data can be imported to or exported from Cloud Storage into Cloud Spanner. Exported files use the Apache Avro or CSV file formats. The export process is implemented by a Cloud Dataflow connector.

The performance of import and export operations is affected by several factors, including the following:

- Size of the database
- Number of secondary indexes
- Location of data
- Load on Cloud Spanner and number of nodes

Since Cloud Spanner can use the Avro format, it is possible to import data into Cloud Spanner from a file that was created by another application.

If you prefer to export data to CSV format, you can run a Dataflow job using the Cloud Spanner to Cloud Storage Text template.

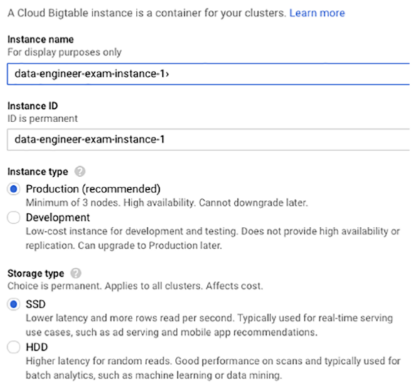
## Cloud Bigtable

*Cloud Bigtable* is a wide-column NoSQL database used for high-volume databases that require low millisecond (ms) latency. Cloud Bigtable is used for IoT, time-series, finance, and similar applications.

### CONFIGURING BIGTABLE

Bigtable is a managed service, but it is not a NoOps service: like Cloud SQL and Cloud Spanner, you have to specify instance information.

In addition to providing an instance name and instance ID, you will have to choose if you are deploying a production or development cluster. A production cluster provides high availability and requires at least three nodes. A development cluster is a lower-cost option for developing and testing your applications (see Figure 2.4).



A Cloud Bigtable instance is a container for your clusters. Learn more

**Instance name**
For display purposes only
data-engineer-exam-instance-1›

**Instance ID**
ID is permanent
data-engineer-exam-instance-1

**Instance type**
[✓ Production (recommended)]
Minimum of 3 nodes. High availability. Cannot downgrade later.
Development
Low-cost instance for development and testing. Does not provide high availability or replication. Can upgrade to Production later.

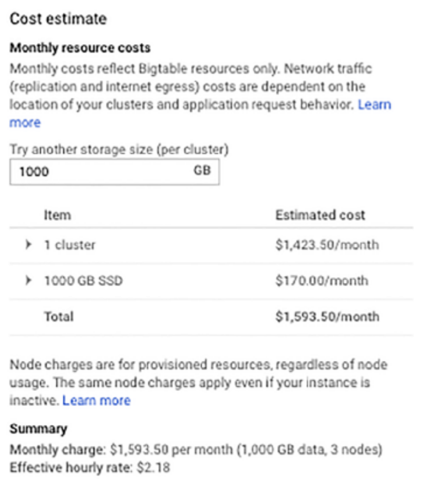
**Storage type**
Choice is permanent. Applies to all clusters. Affects cost.
[✓ SSD]
Lower latency and more rows read per second. Typically used for real-time serving use cases, such as ad serving and mobile app recommendations.
HDD
Higher latency for random reads. Good performance on scans and typically used for batch analytics, such as machine learning or data mining.

**Figure 2.4** Configuring a Bigtable cluster

When creating a cluster, you will have to provide a cluster ID, region, zone, and number of nodes. Bigtable performance scales linearly with the number of nodes. For example, a three-node production cluster using SSDs can read and write 30,000 rows per second. Doubling the number of nodes to six increases read and write performance to 60,000 operations per second.

For multi-regional high availability, you can create a replicated cluster in another region. All data is replicated between clusters. In the event that a cluster becomes unresponsive, it can fail over either manually or automatically. Bigtable instances have an application profile, also called an *app profile*, which specifies how to handle incoming requests. If all requests are routed to single cluster, you will have to perform a manual failover. If the app profile uses multicluster routing, then an automatic failover occurs.

Be aware that Bigtable is an expensive service. As shown in Figure 2.5, the cost of a three-node production cluster is over $1,500 a month, or over $2 an hour.



Cost estimate

**Monthly resource costs**

Monthly costs reflect Bigtable resources only. Network traffic (replication and internet egress) costs are dependent on the location of your clusters and application request behavior. Learn more

Try another storage size (per cluster)

1000 GB

| Item | Estimated cost |
|---|---|
| 1 cluster | $1,423.50/month |
| 1000 GB SSD | $170.00/month |
| Total | $1,593.50/month |

Node charges are for provisioned resources, regardless of node usage. The same node charges apply even if your instance is inactive. Learn more

**Summary**

**Monthly charge:** $1,593.50 per month (1,000 GB data, 3 nodes)
**Effective hourly rate:** $2.18

**Figure 2.5** Cost of a three-node Bigtable production cluster

When migrating from Hadoop and HBase, you may want to take advantage of the fact that Bigtable provides an HBase shell and HBase client for Java. Alternatively, you can work with Bigtable from the command line using the `cbt` command-line utility.

Bigtable tables can be accessed from BigQuery. (This feature is in beta as of this writing). Data is not stored in BigQuery, but Bigtable data is accessed from a table external to BigQuery. This enables you to store frequently changing data, such as IoT data, in Bigtable and not have to export the data to BigQuery before analyzing it or using the data from machine learning.

#### DATABASE DESIGN CONSIDERATIONS

Designing tables for Bigtable is fundamentally different from designing them for relational databases. Bigtable tables are denormalized, and they can have thousands of columns. There is no support for joins in Bigtable or for secondary indexes. Data is stored in Bigtable lexicographically by *row-key*, which is the one indexed column in a Bigtable table. Keeping related data in adjacent rows can help make reads more efficient.

All operations are atomic at the row level, not a transaction level. For example, if an application performs operations on two rows, it is possible for one of those operations to succeed and another one to fail. This can leave the database in an inconsistent state. To avoid this, store all the data that needs to be updated together in the same row.

Since Bigtable does not have secondary indexes, queries are executed using either row-key–based lookups or full table scans. The latter are highly inefficient and should be avoided. Instead, row-key lookups or a range of row-key lookups should be used. This requires carefully planning the row-key around the way that you will query the data. The goal when designing a row-key is to take advantage of the fact that Bigtable stores data in a sorted order.

Characteristics of a good row-key include the following:

- Using a prefix for multitenancy. This isolates data from different customers. This makes scans and reads more efficient since data blocks will have data from one customer only and customers query only their own data.
- Columns that are not frequently updated, such as a user ID or a timestamp.
- Nonsequential value in the first part of the row-key, which helps avoid hotspots.

Another way to improve performance is to use *column families*, which are sets of columns with related data. They are stored together and retrieved together, making reads more efficient.

#### IMPORTING AND EXPORTING

Like Cloud Spanner, Bigtable import and export operations are performed using Cloud Dataflow. Data can be exported to Cloud Storage and stored in either Avro format or SequenceFile format. Data can be imported from Avro and SequenceFile. Data can be imported from CSV files using a Dataflow process as well.

### Cloud Firestore

*Cloud Firestore* is the managed document database that is replacing Cloud Datastore. Document databases are used when the structure of data can vary from one record to another. Cloud Firestore has features not previously available in Cloud Datastore, including

- Strongly consistent storage layer
- Real-time updates
- Mobile and web client libraries

- A new collection and document data model

  Cloud Firestore operates in one of two modes: Native Mode and Cloud Datastore Mode. In Datastore Mode, Firestore operates like Datastore but uses the Firestore storage system. This provides for strong consistency instead of eventual consistency. In addition, Datastore had a limit of 25 entity groups, or ancestor/descendent relations, and a maximum of one write per second to an entity group. Those limits are removed. The new data model, real-time updates, and mobile and web client library features are available only in Native Mode.

### CLOUD FIRESTORE DATA MODEL

Cloud Firestore in Datastore Mode uses a data model that consists of entities, entity groups, properties, and keys.

*Entities* are analogous to tables in a relational database. They describe a particular type of thing, or *entity kind*. Entities have identifiers, which can be assigned automatically or specified by an application. If identifiers are randomly assigned in Cloud Datastore Mode, the identifiers are randomly generated from a uniformly distributed set of identifiers. The random distribution is important to avoid hot spotting when a large number of entities are created in a short period of time.

Entities have properties, which are name-value pairs. For example, `’color’:’red’` is an instance of a `color` property with the value `red`. The property of values can be different types in different entities. Properties are not strongly typed. A property can also be indexed. Note that this is a difference with the other managed NoSQL databases—Bigtable does not have secondary indexes.

Properties can have one value or multiple values. Multiple values are stored as *array properties*. For example, a news article might have a property `’topics’` and values `{’technology’, ’business’, ’finance’}`.

The value of a property can be another entity. This allows for hierarchical structures. Here is an example of a hierarchical structure of an order:

```
{’order_id’: 1,
  ’Customer’: "Margaret Smith",
  ’Line_item’: {
                           {’line_id’: 10,
                             ’description’: "Blue coat",
                             ’quantity’: 3},
                           {’line_id’: 20,
                             ’description’: "Green shirt",
                             ’quantity’: 1},
                           {’line_id’: 30,
                             ’description’: "Black umbrella",
                             ’quantity’: 1}
                       }
}
```

In this example, the line items are entities embedded in the order entity. The order is the parent of the line item entities. The order entity is the root identity, since it does not have a parent entity. To reference an entity within another entity, you specify a path, known as the *ancestor path*, which includes the kind-identifier from the root to the descendent entity. For example, to refer to the line item describing the blue coat line item, you would use something like the following:

```
[order:1, line_item: 10]
```

To summarize, entities have properties that can be atomic values, arrays, or entities. Atomic values include integers, floating value numbers, strings, and dates. These are used when a property can be described by a single value, such as the date an order is placed. Arrays are used for properties that have more than one value, such as topics of news articles. Child entities are entities within another entity and are used to represent complex objects that are described by properties that may be atomic values, arrays, or other entities.

An entity and all its descendent entities are known as *entity groups*.

Entities also have keys, which uniquely identify an entity. A key consists of the following:

- A namespace, which is used to support multitenancy
- The kind of entity, such as order or line item
- An identifier
- An optional ancestor path

  Keys can be used to look up entities and their properties. Alternatively, entities can be retrieved using queries that specify properties and values, much like using a `WHERE` clause in SQL. However, to query using property values, properties need to be indexed.

### INDEXING AND QUERYING

Cloud Firestore uses two kinds of indexes: built-in indexes and composite indexes. *Built-in indexes* are created by default for each property in an entity. *Composite indexes* index multiple values of an entity.

Indexes are used when querying and must exist for any property referenced in filters. For example, querying for `"color" = ’red’` requires an index on the color property. If that index does not exist, Cloud Firestore will not return any entities, even if there are entities with a `color` property and value of `red`. Built-in indexes can satisfy simple equality and inequality queries, but more complex queries require composite indexes.

Composite indexes are used when there are multiple filter conditions in a query. For example, the filter `color = 'red'` and `size = 'large'` requires a composite index that includes both the color and size properties.

Composite indexes are defined in a configuration file called `index.yaml`. You can also exclude properties from having a built-in index created using the `index.yaml` configuration file. You may want to exclude properties that are not used in filter conditions. This can save storage space and avoid unnecessary updating of an index that is never used.

Cloud Firestore uses a SQL-like language called GQL (GraphQL). Queries consist of an entity kind, filters, and optionally a sort order specification. Here is an example query:

```
SELECT * FROM orders
WHERE item_count > 1 AND status = 'shipping'
ORDER BY item_count DESC
```

This query will return entities from the orders collection that have a value greater than 1 for `item_count` and a value of `'shipping'` for status. The results will be sorted in descending order by `item_count`. For this query to function as expected, there needs to be an index on `item_count` and `'shipping'` and sorted in descending order.

#### IMPORTING AND EXPORTING

Entities can be imported and exported from Cloud Firestore. Exports contain entities but not indexes. Those are rebuilt during import. Exporting requires a Cloud Storage bucket to store the exported data. You can specify a filter when exporting so that only a subset of entity kinds are exported.

Exported entities can be imported into BigQuery using the `bq` command, but only if an entity filter was used when creating the export. There are a number of restrictions on importing Cloud Firestore exports to BigQuery:

- The Cloud Storage URI of the export file must not have a wildcard.
- Data cannot be appended to an existing table with a defined schema.
- Entities in the export have a consistent schema.
- Any property value greater than 64 KB is truncated to 64 KB on export.

Cloud Firestore in Datastore Mode is a managed document database that is well suited for applications that require semi-structured data but that do not require low-latency writes (< 10 ms). When low-latency writes are needed, Bigtable is a better option.

### BigQuery

*BigQuery* are fully managed, petabyte-scale, low-cost analytics data warehouse databases. As with other databases, some common tasks are as follows:

- Interacting with data sets
- Importing and exporting data
- Streaming inserts
- Monitoring and logging
- Managing costs
- Optimizing tables and queries

BigQuery is an important service for the Process and Analyze stage of the data lifecycle.

#### BIGQUERY DATASETS

*Datasets* are the basic unit of organization for sharing data in BigQuery. A dataset can have multiple tables. When you create a dataset in BigQuery, you specify the following:

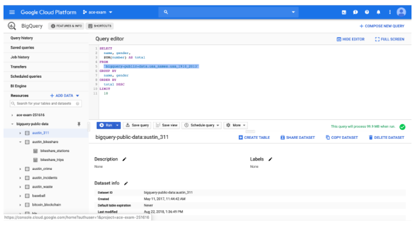
- A dataset ID
- Data location
- Default table expiration

A *dataset ID* is a name of the dataset, and the *data location* is a region that supports BigQuery. The default table expiration allows you to set a time to live for the dataset. This is useful when you know that you will keep data for some period of time and then delete it. You can create tables using the BigQuery user interface or the command line. Once you have created a dataset, you can load data into it. We'll describe how to do that in the next section, but first let's describe how to use SQL to query a dataset.

Figure 2.6 shows an example BigQuery interactive interface. The query is as follows:

```
SELECT
  name, gender,
  SUM(number) AS total
FROM
  'bigquery-public-data.usa_names.usa_1910_2013'
GROUP BY
```

```
    name, gender
  ORDER BY
    total DESC
  LIMIT
    10
```



```
SELECT
  name, gender,
  SUM(number) AS total
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
GROUP BY
  name, gender
ORDER BY
  total DESC
LIMIT
  10
```

This query will process 99.9 MB when run.

bigquery-public-data:austin_311

Description
None

Labels
None

Dataset info

| | |
|---|---|
| Dataset ID | bigquery-public-data:austin_311 |
| Created | May 11, 2017, 11:44:42 AM |
| Default table expiration | Never |
| Last modified | Aug 22, 2018, 1:36:49 PM |

**Figure 2.6** BigQuery interactive interface with sample query

This query selects the name, gender, and number of children born with that name for each gender from a table called `usa_1910_2013`. That table is in a dataset named `usa_names` and in a project called `bigquery-public-data`. Note that when fully specifying a table name, the pattern is the project name, followed by the dataset name, then followed by the table name. Also note that backticks, not single quotes, are used when specifying a table name.

You could also query a dataset using the `bq` command line, as shown in the following example. The bodies of the `SELECT` statements are wrapped in single quotes, whereas the table names are wrapped in backticks.

```
  bq query --nouse_legacy_sql \' SELECT
    name, gender,
    SUM(number) AS total
  FROM
    'bigquery-public-data.usa_names.usa_1910_2013'
  GROUP BY
    name, gender
  ORDER BY
    total DESC
  LIMIT
    10'
```

BigQuery supports two dialects of SQL, legacy and standard, and the `bq` query command takes a parameter indicating which SQL dialect to use when parsing. As the names imply, standard SQL is the preferred dialect, but both are currently supported. Standard SQL supports advanced SQL features, such as correlated subqueries, `ARRAY` and `STRUCT` data types, as well as complex join expressions.

BigQuery uses the concept of slots for allocating computing resources to execute queries. For most users, the default number of slots is sufficient, but very large queries, or a large number of concurrent queries, could see a performance benefit with additional slots. The default number of slots is shared across a project, and as long as you are not processing more than 100 GB at once, you will probably be using less than the maximum number of slots, which is 2,000. The 2,000-slot limit only applies to on-demand pricing; Google also offers flat-rate pricing for enterprises with a need for more slots.

### LOADING AND EXPORTING DATA

With an existing dataset, you can create tables and load data into those tables. As with querying, you can use either the UI or the command line. In both cases, you will need to specify the following:

- The type of data source, which can be Google Cloud Storage, an uploaded file from local drive, G Drive, or a Cloud Bigtable table, and the data transfer service (DTS).
- The data source, such as a URI to a Cloud Storage file, a Bigtable table, or a path to a local filename.
- File format—one of: Avro, CSV, JSON (newline delimited), ORC, or Parquet. If you are loading from Cloud Storage, you can also load Cloud Datastore and Cloud Firestore exports.
- Destination table, including project, dataset, and table name.
- Schema, which can be auto-detected, specified in text, or entered one column at a time, specifying the column name, type, and mode. Mode may be `NULLABLE`, `REQUIRED`, or `REPEATED`.
- Partitioning, which can be no partitioning or partitioning by ingestion time. If a table is partitioned, you can use clustering order, which optimizes the way that column data is stored and can optimize the way that queries are executed. You can also specify that any query on a partitioned table needs to specify a partition filter in the `WHERE` clause. This can significantly reduce cost and improve performance by limiting the amount of data scanned when executing the query.

BigQuery expects data to be encoded using UTF-8. If a CSV file is not in UTF-8, BigQuery will try to convert it. The conversion is not always correct, and there can be differences in some bytes. If your CSV file does not load correctly, specify the correct encoding. If you are loading JSON files, they need to be in UTF-8 encoding.

Avro is the preferred format for loading data because blocks of data can be read in parallel, even if the file is compressed and there are no encoding issues, such as with CSV. Parquet also stores data using a column model.

Uncompressed CSV and JSON files load faster than compressed files because they can be loaded in parallel, but this can lead to higher storage costs when using Cloud Storage.

The BigQuery Data Transfer Service automates loading data from Google's software as a service (SaaS) offerings, such as Google Ad Manager, Google Ads, Google Play, and YouTube Channel Reports. It can also be used to load data from Amazon S3.

Cloud Dataflow can load directly into BigQuery.

#### CLUSTERING, PARTITIONING, AND SHARDING TABLES

BigQuery provides for creating clustered tables. In clustered tables, data is automatically organized based on the contents of one or more columns. Related data is collocated in a clustered table. This kind of organization can improve the performance of some queries, specifically the queries that filter rows using the columns used to cluster data. Clustered tables can be partitioned.

It is a good practice to break large tables into smaller ones or partitions to improve query efficiency. When splitting data by date or timestamp, you can use partitions, and to split data into multiple tables by other attributes, you can try *sharding*.

Partitions can be based on ingestion time or by date or timestamp in the input data. When data is partitioned by ingestion time, BigQuery creates a new partition for each day.

Shards can be based on the value in a partition column, such as customer ID. The column does not have to be a date or timestamp column. It's a best practice to use partitioning by time instead of sharding by a date. The former is more efficient with BigQuery due to the backend optimizations it creates with timestamps.

Sharding can make use of template tables, which are tables that have a schema defined in a template and that template is used to create one or more tables that have a target table name and a table suffix. The target table name is the same for all tables created with the template, but the suffix is different for each table created.

#### STREAMING INSERTS

The loading procedures just described are designed for batch loading. BigQuery also supports streaming inserts that load one row at a time. Data is generally available for analysis within a few seconds, but it may be up to 90 minutes before data is available for copy and export operations. This is not intended for transactional workloads, but rather analytical ones.

Streaming inserts provide best effort de-duplication. When inserting a row, you can include an `insertID` that uniquely identifies a record. BigQuery uses that identifier to detect duplicates. If no `insertID` is provided with a row, then BigQuery does not try to de-duplicate data. If you do provide an `insertID` and employ de-duplication, you are limited to 100,000 rows per second and 100 MB per second. If de-duplication is not enabled, you can insert up to 1,000,000 rows per second and 1 GB per second.

The advantage of using template tables is that you do not have to create all tables in advance. For example, if you are streaming in data from a medical device and you want to have table for each device, you could use the device identifier as the suffix, and when the first data from that device arrives, a table will be created from the template.

Standard SQL makes it easy to query across template tables by allowing wildcards in a table name. For example, if you have a set of medical device tables named `’medical_device_’ + <device id>`, such as `’medical_device_123’`, `’medical_device_124’`, `’medical_device_125’`, and so forth, you could query across those tables by using a `FROM` clause as follows:

```
FROM ’med_project.med_dataset.medical_device*’
```

Wildcards cannot be used with views or external tables.

#### MONITORING AND LOGGING IN BIGQUERY

Stackdriver is used for monitoring and logging in BigQuery. Stackdriver Monitoring provides performance metrics, such query counts and time to run queries. Stackdriver Logging is used to track events, such as running jobs or creating tables.

Stackdriver Monitoring collects metrics on a range of operations, including

- Number of scanned bytes
- Query time
- Slots allocated
- Slots available
- Number of tables in a dataset
- Uploaded rows

You can build dashboards in Stackdriver Monitoring to help track key performance indicators, such as top long-running queries and 95th percentile query time.

Stackdriver Logging tracks log entries that describe events. Events have resource types, which can be projects or datasets, and type-specific attributes, like a location for storage events. Events that are tracked include the following:

- Inserting, updating, patching, and deleting tables
- Inserting jobs
- Executing queries

Logs are useful for understanding who is performing actions in BigQuery, whereas monitoring is useful for understanding how your queries and jobs are performing.

#### BIGQUERY COST CONSIDERATIONS

BigQuery costs are based on the amount of data stored, the amount of data streamed, and the workload required to execute queries. Since the prices of these various services can change, it is not important to know specific amounts, but it is helpful to understand the relative costs. That can help when choosing among different options.

BigQuery data is considered active if it was referenced in the last 90 days; otherwise, it is considered long-term data. Active Storage is currently billed at $0.20/GB a month, and long-term data is billed at $0.10/GB a month. The charge for long-term storage in BigQuery is currently equal to the cost of Nearline storage, so there is no cost advantage to storing long-term data in Cloud Storage unless you were to store it Coldline storage, which is currently billed at $0.07/GB a month.

Streaming inserts are billed at $0.01 per 200 MB, where each row is considered at least 1 KB.

On-demand queries are billed at $5.00 per TB scanned. Monthly flat rate billing is $10,000 per 500 slots per month. Annual flat rate billing is $8,500 a month for 500 slots.

There is no charge for loading, copying, or exporting data, but there are charges for the storage used.

There are separate charges for using BigQuery ML machine learning service (BQML), for using BigQuery's native machine learning capabilities, and for using the BigQuery Data Transfer service.

#### TIPS FOR OPTIMIZING BIGQUERY

One way to keep costs down is to optimize the way that you use BigQuery. Here are several ways to do this:

- Avoid using `SELECT *`.
- Use `--dry-run` to estimate the cost of a query.
- Set the maximum number of bytes billed.
- Partition by time when possible.
- Denormalize data rather than join multiple tables.

Avoid using `SELECT *` queries. These scan all the columns in a table or view; instead, list the specific columns that you want and limit the list only to columns that are needed. If you need to view examples of data in all columns, use the Preview option in the BigQuery GUI or run the `bq head` command from the command line, which functions like the Linux `head` command and displays the first rows of a table. Preview or `bq head` is a better option than running a `SELECT` query with a `LIMIT` clause because `LIMIT` limits the number of rows returned only, not the number of rows scanned.

If you would like to know what it would cost to run a query, you can view the query validator in the BigQuery GUI, or you can use the `--dry-run` option with the `bq` query command.

You can also set a maximum number of bytes billed for a query. If the query scans more than that number of bytes, the query fails, and you are not billed for the query. Maximum bytes billed can be specified in the GUI or in a `bq` query command using the `--maximum_bytes_billed` parameter.

Partition by time when possible. This will help reduce the amount of data that needs to be scanned. BigQuery creates a pseudo-column on partitioned tables called `_PARTITIONTIME`, which can be used in `WHERE` clauses to limit the amount of data scanned. For example, the following `WHERE` clause will return only rows that are in partitions holding data from January 1, 2019, to January 31, 2019.

```
WHERE _PARTITIONTIME
BETWEEN TIMESTAMP("20190101")
    AND TIMESTAMP("20190131")
```

BigQuery supports nested and repeated structures in rows. Nested data is represented in `STRUCT` type in SQL, and repeated types are represented in `ARRAY` types in SQL.

For the fastest query performance, load data into BigQuery, but if you can tolerate some longer latency, then keeping data in external data stores can minimize the amount of data loading you need to do.

BigQuery is a popular GCP service because it requires little operational overhead, supports large volumes of data, and is highly performant, especially when queries are tuned to take advantage of BigQuery's architecture.

### Cloud Memorystore

*Cloud Memorystore* is a managed Redis service, which is commonly used for caching. Redis instances can be created using the Cloud Console or `gcloud` commands. There are only a small number of basic configuration parameters with Cloud Memorystore:

- Instance ID.
- Size specification.
- Region and zone.
- Redis version; currently the options are 3.2 and 4.0; 4.0 is recommended.
- Instance tier, which can be basic and is not highly available, or standard, which is includes a failover replica in a different zone.
- Memory capacity, which ranges from 1 to 300 GB.

You also have the option of specifying Redis configuration parameters, such as maximum memory policy, and an eviction policy, such as least frequently used.

Cloud Memorystore provides support for importing and exporting from Redis; this feature is in beta as of this writing. Exporting will create a backup file of the Redis cache in a Cloud Storage bucket. During export, read and write operations can occur, but administration operations, like scaling, are not allowed. Import reads export files and overwrites the contents of a Redis cache. The instance is not available for read or write operations during the import.

Redis instances in Cloud Memorystore can be scaled to use more or less memory. When scaling a Basic Tier instance, reads and writes are blocked. When the resizing is complete, all data is flushed from the cache. Standard Tier instances can scale while continuing to support read and write operations. During a scaling operation, the replica is resized first and then synchronized with the primary. The primary then fails over to the replica. Write operations are supported when scaling Standard Tier instances, but too much write load can significantly slow the resizing operation.

When the memory used by Redis exceeds 80 percent of system memory, the instance is considered under memory pressure. To avoid memory pressure, you can scale up the instance, lower the maximum memory limit, modify the eviction policy, set time-to-live (TTL) parameters on volatile keys, or manually delete data from the instance. The TTL parameter specifies how long a key should be kept in the cache before it becomes eligible for eviction. Frequently updated values should have short TTLs whereas keys with values that don't change very often can have longer TTLs. Some eviction policies target only keys with TTLs whereas other policies target all keys. If you find that you are frequently under memory pressure, your current eviction policy applies only to keys with TTLs, and there are keys without TTLs, then switching to an eviction policy that targets all keys may relieve some of that memory pressure.

Redis provides a number of eviction policies that determine which keys are removed from the cache when the maximum memory limit is reached. By default, Redis evicts the least recently used keys with TTLs set. Other options include evicting based on least frequently used keys or randomly selecting keys.

Although Cloud Memorystore is a managed service, you should still monitor the instance, particularly memory usage, duration periods of memory overload, cache-hit ratio, and the number of expirable keys.

### Cloud Storage

NOTE This section first appeared in Chapter 5 of my book, *Official Google Cloud Certified Professional Cloud Architect Study Guide* (Wiley, 2019).

*Google Cloud Storage* is an object storage system. It is designed for persisting unstructured data, such as data files, images, videos, backup files, and any other data. It is unstructured in the sense that objects—that is, files stored in Cloud Storage—are treated as atomic. When you access a file in Cloud Storage, you access the entire file. You cannot treat it as file on a block storage device that allows for seeking and reading specific blocks in the file. There is no presumed structure within the file that Cloud Storage can exploit.

#### ORGANIZING OBJECTS IN A NAMESPACE

Also, there is minimal structure for hierarchical structures. Cloud Storage uses buckets to group objects. A *bucket* is a group of objects that share access controls at the bucket level. For example, the service account assigned to a virtual machine may have permissions to write to one bucket and read from another bucket. Individual objects within buckets can have their own access controls as well.

Google Cloud Storage uses a global namespace for bucket names, so all bucket names must have unique names. Object names do not have to be unique. A bucket is named when it is created and cannot be renamed. To simulate renaming a bucket, you will need to copy the contents of the bucket to a new bucket with the desired name and then delete the original bucket.

Google recommends the following suggestions for bucket naming:

- Do not use personally identifying information, such as names, email addresses, IP addresses, and so forth in bucket names. That kind of information could be useful to an attacker.
- Follow DNS naming conventions because bucket names can appear in a CNAME record in DNS.
- Use globally unique identifiers (GUIDs) if creating a large number of buckets.
- Do not use sequential names or timestamps if uploading files in parallel. Files with sequentially close names will likely be assigned to the same server. This can create a hotspot when writing files to Cloud Storage.
- Bucket names can also be subdomain names, such as `mybucket.example.com`.

To create a domain name bucket, you will have to verify that you are the owner of the domain.

The Cloud Storage service does not use a filesystem. This means that there is no ability to navigate a path through a hierarchy of directories and files. The object store does support a naming convention that allows for the naming of objects in a way that looks similar to the way that a hierarchical filesystem would structure a file path and filename. If you would like to use Google Cloud Storage as a filesystem, the Cloud Storage FUSE open source project provides a mechanism to map from object storage systems to filesystems (`https://cloud.google.com/storage/docs/gcs-fuse`).

### STORAGE TIERS

Cloud Storage offers four tiers or types of storage. It is essential to understand the characteristics of each tier and when it should be used for the Cloud Professional Data Engineer exam. The four types of Cloud Storage are as follows:

- Regional
- Multi-regional
- Nearline
- Coldline

*Regional storage* stores multiple copies of an object in multiple zones in one region. All Cloud Storage options provide high durability, which means that the probability of losing an object during any particular period of time is extremely low. Cloud Storage provides 99.999999999 percent (eleven 9s) annual durability.

This level of durability is achieved by keeping redundant copies of the object. *Availability* is the ability to access an object when you want it. An object can be durably stored but unavailable. For example, a network outage in a region would prevent you from accessing an object stored in that region, although it would continue to be stored in multiple zones.

*Multi-regional storage* mitigates the risk of a regional outage by storing replicas of objects in multiple regions. This can also improve access time and latency by distributing copies of objects to locations that are closer to the users of those object. Consider a user in California in the western United States accessing an object stored in us-west1, which is a region located in the northwest state of Oregon in the United States. That user can expect under 5 ms latency with a user in New York, in the United States northeast, and would likely experience latencies closer to 30 ms.

For more, see Windstream Services IP Latency Statistics, `https://ipnetwork.windstream.net/`, accessed May 8, 2019.

Multi-regional storage is also known as *geo-redundant storage*. Multi-regional Cloud Storage buckets are created in one of the multi-regions—asia, eu, or us—for data centers in Asia, the European Union, and the United States, respectively.

The latencies mentioned here are based on public Internet network infrastructure. Google offers two network tiers: Standard and Premium. With the *Standard network tier*, data is routed between regions using public Internet infrastructure and is subject to network conditions and routing decisions beyond Google's control. The *Premium network tier* routes data over Google's global high-speed network. Users of Premium tier networking can expect lower latencies.

Nearline and Coldline storage are used for storing data that is not frequently accessed. Data that is accessed less than once in 30 days is a good candidate for *Nearline storage*. Data that is accessed less than once a year is a good candidate for *Coldline storage*. All storage classes have the same latency to return the first byte of data, but the costs to access data and the per-operation costs are higher than regional storage.

Multi-regional storage has a 99.95 percent availability SLA. Regional storage has a 99.9 percent availability SLA. Nearline and Coldline storage have 99.9 percent availability SLA in multi-regional locations and 99.0 percent availability in regional locations.

#### CLOUD STORAGE USE CASES

Cloud Storage is used for a few broad use cases:

- Storage of data shared among multiple instances that does not need to be on persistent attached storage. For example, log files may be stored in Cloud Storage and analyzed by programs running in a Cloud Dataproc Spark cluster.
- Backup and archival storage, such as persistent disk snapshots, backups of on-premises systems, and data kept for audit and compliance requirements but not likely to be accessed.
- As a staging area for uploaded data. For example, a mobile app may allow users to upload images to a Cloud Storage bucket. When the file is created, a Cloud Function could trigger to initiate the next steps of processing.

Each of these examples fits well with Cloud Storage's treatment of objects as atomic units. If data within the file needs to be accessed and processed, that is done by another service or application, such as a Spark analytics program.

Different tiers are better suited for some use cases. For example, Coldline storage is best used for archival storage, but multi-regional storage may be the best option for uploading user data, especially if users are geographically dispersed.

#### DATA RETENTION AND LIFECYCLE MANAGEMENT

Data has something of a life as it moves through several stages, starting with creation, active use, infrequent access but kept online, archived, and deleted. Not all data goes through all of the stages, but it is important to consider lifecycle issues when planning storage systems.

The choice of storage system technology usually does not directly influence data lifecycles and retention policies, but it does impact how the policies are implemented. For example, Cloud Storage lifecycle policies can be used to move objects from Nearline storage to Coldline storage after some period of time. When partitioned tables are used in BigQuery, partitions can be deleted without affecting other partitions or running time-consuming jobs that scan full tables for data that should be deleted.

If you are required to store data, consider how frequently and how fast the data must be accessed:

- If submillisecond access time is needed, use a cache such as Cloud Memorystore.
- If data is frequently accessed, may need to be updated, and needs to be persistently stored, use a database. Choose between relational and NoSQL based on the structure of the data. Data with flexible schemas can use NoSQL databases.
- If data is less likely to be accessed the older it gets, store data in time-partitioned tables if the database supports partitions. Time-partitioned tables are frequently used in BigQuery, and Bigtable tables can be organized by time as well.
- If data is infrequently accessed and does not require access through a query language, consider Cloud Storage. Infrequently used data can be exported from a database, and the export files can be stored in Cloud Storage. If the data is needed, it can be imported back into the database and queried from there.
- When data is not likely to be accessed but it must still be stored, use the Coldline storage class in Cloud Storage. This is less expensive than multi-regional, regional, or Nearline classes of storage.

Cloud Storage provides object lifecycle management policies to make changes automatically to the way that objects are stored in the object datastore. These policies contain rules for manipulating objects and are assigned to buckets. The rules apply to objects in those buckets. The rules implement lifecycle actions, including deleting an object and setting the storage class. Rules can be triggered based on the age of the object, when it was created, the number of newer versions, and the storage class of the object.

Another control for data management are *retention policies*. A retention policy uses the Bucket Lock feature of Cloud Storage buckets to enforce object retention. By setting a retention policy, you ensure that any object in the bucket or future objects in the bucket are not deleted until they reach the age specified in the retention policy. This feature is particularly useful for compliance with government or industry regulations. Once a retention policy is locked, it cannot be revoked.

### Unmanaged Databases

Although GCP offers a range of managed database options, there may be use cases in which you prefer to manage your own database. These are sometimes referred to as *unmanaged databases*, but self-managed is probably a better term.

When you manage your own databases, you will be responsible for an array of database and system administration tasks, including

- Updating and patching the operating system
- Updating and patching the database system
- Backing up and, if needed, recovering data
- Configuring network access
- Managing disk space
- Monitoring database performance and resource utilization
- Configuring for high availability and managing failovers
- Configuring and managing read replicas

For the purpose of the Cloud Professional Data Engineer exam, it is important to appreciate the role of Stackdriver to understanding the state of a database system. The two Stackdriver components that are used with unmanaged databases are Stackdriver Monitoring and Stackdriver Logging.

Instances have built-in monitoring and logging. Monitoring includes CPU, memory, and I/O metrics. Audit logs, which have information about who created an instance, is also available by default. If you would like insights into application performance, in this case into database performance, you should install Stackdriver Monitoring and Stackdriver Logging agents.

Once the Stackdriver Logging agent is installed, it can collect application logs, including database logs. Stackdriver Logging is configured with Fluentd, an open source data collector for logs.

Once the Stackdriver Monitoring agent is installed, it can collect application performance metrics. Monitoring a specific database may require a plug-in designed for the particular database, such as MySQL or PostgreSQL.

## Exam Essentials

**Cloud SQL supports MySQL, PostgreSQL, and SQL Server (beta).** Cloud SQL instances are created in a single zone by default, but they can be created for high availability and use instances in multiple zones. Use read replicas to improve read performance. Importing and exporting are implemented via the RDBMS-specific tool.

**Cloud Spanner is configured as regional or multi-regional instances.** Cloud Spanner is a horizontally scalable relational database that automatically replicates data. Three types of replicas are read-write replicas, read-only replicas, and witness replicas. Avoid hotspots by not using consecutive values for primary keys.

**Cloud Bigtable is a wide-column NoSQL database used for high-volume databases that require sub-10 ms latency.** Cloud Bigtable is used for IoT, time-series, finance, and similar applications. For multi-regional high availability, you can create a replicated cluster in another region. All data is replicated between clusters. Designing tables for Bigtable is fundamentally different from designing them for relational databases. Bigtable tables are denormalized, and they can have thousands of columns. There is no support for joins in Bigtable or for secondary indexes. Data is stored in Bigtable lexicographically by row-key, which is the one indexed column in a Bigtable table. Keeping related data in adjacent rows can help make reads more efficient.

**Cloud Firestore is a document database that is replacing Cloud Datastore as the managed document database.** The Cloud Firestore data model consists of entities, entity groups, properties, and keys. Entities have properties that can be atomic values, arrays, or entities. Keys can be used to lookup entities and their properties. Alternatively, entities can be retrieved using queries that specify properties and values, much like using a `WHERE` clause in SQL. However, to query using property values, properties need to be indexed.

**BigQuery is an analytics database that uses SQL as a query language.** Datasets are the basic unit of organization for sharing data in BigQuery. A dataset can have multiple tables. BigQuery supports two dialects of SQL: legacy and standard. Standard SQL supports advanced SQL features such as correlated subqueries, `ARRAY` and `STRUCT` data types, and complex join expressions. BigQuery uses the concepts of slots for allocating computing resources to execute queries. BigQuery also supports streaming inserts, which load one row at a time. Data is generally available for analysis within a few seconds, but it may be up to 90 minutes before data is available for copy and export operations. Streaming inserts provide for best effort de-duplication. Stackdriver is used for monitoring and logging in BigQuery. Stackdriver Monitoring provides performance metrics, such query counts and time, to run queries. Stackdriver Logging is used to track events, such as running jobs or creating tables. BigQuery costs are based on the amount of data stored, the amount of data streamed, and the workload required to execute queries.

**Cloud Memorystore is a managed Redis service. Redis instances can be created using the Cloud Console or `gcloud` commands.** Redis instances in Cloud Memorystore can be scaled to use more or less memory. When scaling a Basic Tier instance, reads and writes are blocked. When the resizing is complete, all data is flushed from the cache. Standard Tier instances can scale while continuing to support read and write operations. When the memory used by Redis exceeds 80 percent of system memory, the instance is considered under memory pressure. To avoid memory pressure, you can scale up the instance, lower the maximum memory limit, modify the eviction policy, set time-to-live (TTL) parameters on volatile keys, or manually delete data from the instance.

**Google Cloud Storage is an object storage system.** It is designed for persisting unstructured data, such as data files, images, videos, backup files, and any other data. It is unstructured in the sense that objects—that is, files stored in Cloud Storage—use buckets to group objects. A bucket is a group of objects that share access controls at the bucket level. The four storage tiers are Regional, Multi-regional, Nearline, and Coldline.

**When you manage your own databases, you will be responsible for an array of database and system administration tasks.** The two Stackdriver components that are used with unmanaged databases are Stackdriver Monitoring and Stackdriver Logging. Instances have built-in monitoring and logging. Monitoring includes CPU, memory, and I/O metrics. Audit logs, which have information about who created an instance, are also available by default. Once the Stackdriver Logging agent is installed, it can collect application logs, including database logs. Stackdriver Logging is configured with Fluentd, an open source data collector for logs. Once the Stackdriver Monitoring agent is installed, it can collect application performance metrics.

## Review Questions

You can find the answers in the appendix.

1. A database administrator (DBA) who is new to Google Cloud has asked for your help configuring network access to a Cloud SQL PostgreSQL database. The DBA wants to ensure that traffic is encrypted while minimizing administrative tasks, such as managing SQL certificates. What would you recommend?

1. Use the TLS protocol
2. Use Cloud SQL Proxy
3. Use a private IP address
4. Configure the database instance to use auto-encryption

2. You created a Cloud SQL database that uses replication to improve read performance. Occasionally, the read replica will be unavailable. You haven't noticed a pattern, but the disruptions occur once or twice a month. No DBA operations are occurring when the incidents occur. What might be the cause of this issue?

1. The read replica is being promoted to a standalone Cloud SQL instance.
2. Maintenance is occurring on the read replica.
3. A backup is being performed on the read replica.
4. The primary Cloud SQL instance is failing over to the read replica.

3. Your department is experimenting with using Cloud Spanner for a globally accessible database. You are starting with a pilot project using a regional instance. You would like to follow Google's recommendations for the maximum sustained CPU utilization of a regional instance. What is the maximum CPU utilization that you would target?

1. 50%
2. 65%
3. 75%
4. 45%

4. A Cloud Spanner database is being deployed in us-west1 and will have to store up to 20 TB of data. What is the minimum number of nodes required?

1. 10
2. 20
3. 5
4. 40

5. A software-as-a-service (SaaS) company specializing in automobile IoT sensors collects streaming time-series data from tens of thousands of vehicles. The vehicles are owned and operated by 40 different companies, who are the primary customers of the SaaS company. The data will be stored in Bigtable using a multitenant database; that is, all customer data will be stored in the same database. The data sent from the IoT device includes a sensor ID, which is globally unique; a timestamp; and several metrics about engine efficiency. Each customer will query their own data only. Which of the following would you use as a row-key?

1. Customer ID, timestamp, sensor ID

2. Customer ID, sensor ID, timestamp
3. Sensor ID, timestamp, customer ID
4. Sensor ID, customer ID, timestamp

6. A team of game developers is using Cloud Firestore to store player data, including character description, character state, and possessions. Descriptions are up to a 60-character alphanumeric string that is set when the character is created and not updated. Character state includes health score, active time, and passive time. When they are updated, they are all updated at the same time. Possessions are updated whenever the character acquires or loses a possession. Possessions may be complex objects, such as bags of items, where each item may be a simple object or another complex object. Simple objects are described with a character string. Complex objects have multiple properties. How would you model player data in Cloud Firestore?

1. Store description and character state as strings and possessions as entities
2. Store description, character state, and possessions as strings
3. Store description, character state, and possessions as entities
4. Store description as a string; character state as an entity with properties for health score, active time, and passive time; and possessions as an entity that may have embedded entities

7. You are querying a Cloud Firestore collection of order entities searching for all orders that were created today and have a total sales amount of greater than $100. You have not excluded any indexes, and you have not created any additional indexes using `index.yaml`. What do you expect the results to be?

1. A set of all orders created today with a total sales amount greater than $100
2. A set of orders created today and any total sales amount
3. A set of with total sales amount greater than $100 and any sales date
4. No entities returned

8. You are running a Redis cache using Cloud Memorystore. One day, you receive an alert notification that the memory usage is exceeding 80 percent. You do not want to scale up the instance, but you need to reduce the amount of memory used. What could you try?

1. Setting shorter TTLs and trying a different eviction policy.
2. Switching from Basic Tier to Standard Tier.
3. Exporting the cache.
4. There is no other option—you must scale the instance.

9. A team of machine learning engineers are creating a repository of data for training and testing machine learning models. All of the engineers work in the same city, and they all contribute datasets to the repository. The data files will be accessed frequently, usually at least once a week. The data scientists want to minimize their storage costs. They plan to use Cloud Storage; what storage class would you recommend?

1. Regional
2. Multi-regional
3. Nearline
4. Coldline

10. Auditors have informed your company CFO that to comply with a new regulation, your company will need to ensure that financial reporting data is kept for at least three years. The CFO asks for your advice on how to comply with the regulation with the least administrative overhead. What would you recommend?

1. Store the data on Coldline storage
2. Store the data on multi-regional storage
3. Define a data retention policy
4. Define a lifecycle policy

11. As a database administrator tasked with migrating a MongoDB instance to Google Cloud, you are concerned about your ability to configure the database optimally. You want to collect metrics at both the instance level and the database server level. What would you do in addition to creating an instance and installing and configuring MongoDB to ensure that you can monitor key instances and database metrics?

1. Install Stackdriver Logging agent.
2. Install Stackdriver Monitoring agent.
3. Install Stackdriver Debug agent.
4. Nothing. By default, the database instance will send metrics to Stackdriver.

12. A group of data scientists have uploaded multiple time-series datasets to BigQuery over the last year. They have noticed that their queries—which select up to six columns, apply four SQL functions, and group by the day of a timestamp—are taking longer to run and are incurring higher BigQuery costs as they add data. They do not understand why this is the case since they typically work only with the most recent set of data loaded. What would you recommend they consider in order to reduce query latency and query costs?

    1. Sort the data by time order before loading
    2. Stop using Legacy SQL and use Standard SQL dialect
    3. Partition the table and use clustering
    4. Add more columns to the `SELECT` statement to use data fetched by BigQuery more efficiently

13. You are querying a BigQuery table that has been partitioned by time. You create a query and use the `--dry_run` flag with the `bq` query command. The amount of data scanned is far more than you expected. What is a possible cause of this?

    1. You did not include `_PARTITIONTIME` in the `WHERE` clause to limit the amount of data that needs to be scanned.
    2. You used CSV instead of AVRO file format when loading the data.
    3. Both active and long-term data are included in the query results.
    4. You used JSON instead of the Parquet file format when loading the data.

14. Your department is planning to expand the use of BigQuery. The CFO has asked you to investigate whether the company should invest in flat-rate billing for BigQuery. What tools and data would you use to help answer that question?

    1. Stackdriver Logging and audit log data
    2. Stackdriver Logging and CPU utilization metrics
    3. Stackdriver Monitoring and CPU utilization metrics
    4. Stackdriver Monitoring and slot utilization metrics

15. You are migrating several terabytes of historical sensor data to Google Cloud Storage. The data is organized into files with one file per sensor per day. The files are named with the date followed by the sensor ID. After loading 10 percent of the data, you realize that the data loads are not proceeding as fast as expected. What might be the cause?

    1. The filenaming convention uses dates as the first part of the file name. If the files are loaded in this order, they may be creating hotspots when writing the data to Cloud Storage.
    2. The data is in text instead of Avro or Parquet format.
    3. You are using a `gcloud` command-line utility instead of the REST API.
    4. The data is being written to regional instead of multi-regional storage.

Support / Sign Out

PREV
Chapter 1 Selecting Appropriate Storage Technologies

NEXT
Chapter 3 Designing Data Pipelines