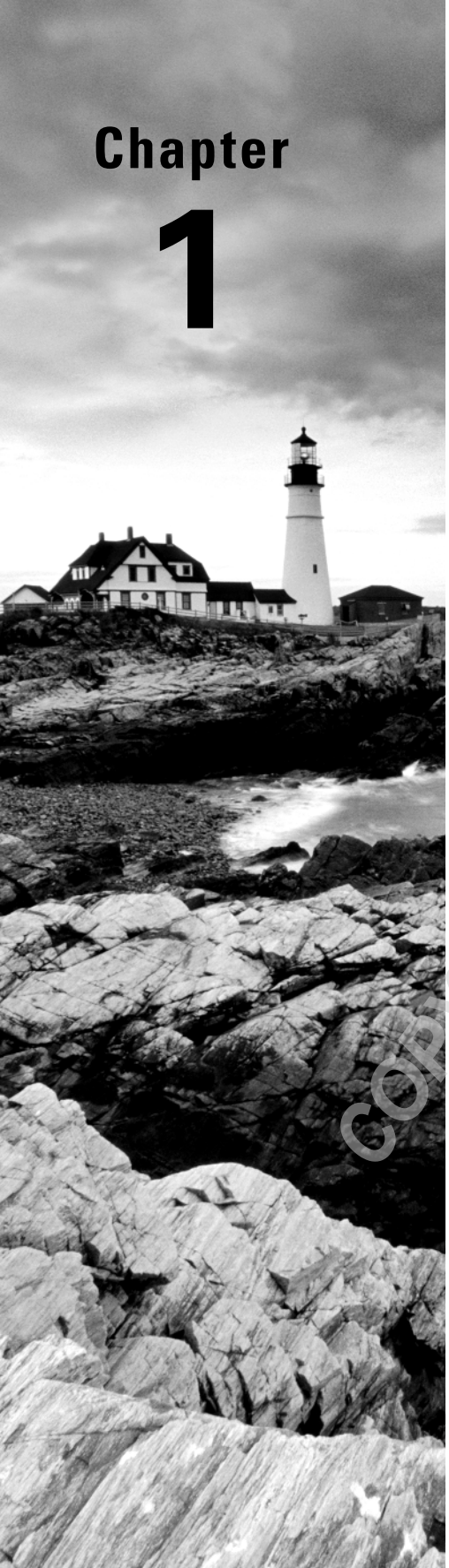


Chapter

1

Selecting Appropriate Storage Technologies

GOOGLE CLOUD PROFESSIONAL DATA ENGINEER EXAM OBJECTIVES COVERED IN THIS CHAPTER INCLUDE THE FOLLOWING:

1. Designing data processing systems

✓ **1.1 Selecting the appropriate storage technologies**

- Mapping storage systems to business requirements
- Data modeling
- Tradeoffs involving latency, throughput, transactions
- Distributed systems
- Schema design

Data engineers choose how to store data for many different situations. Sometimes data is written to a temporary staging area, where it stays only seconds or less before it is read by an application and deleted. In other cases, data engineers arrange long-term archival storage for data that needs to be retained for years. Data engineers are increasingly called on to work with data that streams into storage constantly and in high volumes. Internet of Things (IoT) devices are an example of streaming data.

Another common use case is storing large volumes of data for batch processing, including using data to train machine learning models. Data engineers also consider the range of variety in the structure of data. Some data, like the kind found in online transaction processing, is highly structured and varies little from one datum to the next. Other data, like product descriptions in a product catalog, can have a varying set of attributes. Data engineers consider these and other factors when choosing a storage technology.

This chapter covers objective 1.1 of the Google Cloud Professional Data Engineer exam—Selecting appropriate storage technologies. In this chapter, you will learn about the following:

- The business aspects of choosing a storage system
- The technical aspects of choosing a storage system
- The distinction between structured, semi-structured, and unstructured data models
- Designing schemas for relational and NoSQL databases

By the end of this chapter, you should understand the various criteria data engineers consider when choosing a storage technology. In Chapter 2, "Building and Operationalizing Storage Systems," we will delve into the details of Google Cloud storage services.

From Business Requirements to Storage Systems

Business requirements are the starting point for choosing a data storage system. Data engineers will use different types of storage systems for different purposes. The specific storage system you should choose is determined, in large part, by the stage of the data lifecycle for which the storage system is used.

The data lifecycle consists of four stages:

- Ingest
- Store
- Process and analyze
- Explore and visualize

Ingestion is the first stage in the data lifecycle, and it entails acquiring data and bringing data into the Google Cloud Platform (GCP). The *storage stage* is about persisting data to a storage system from which it can be accessed for later stages of the data lifecycle. The *process and analyze stage* begins with transforming data into a usable format for analysis applications. *Explore and visualize* is the final stage, in which insights are derived from analysis and presented in tables, charts, and other visualizations for use by others.

Ingest

The three broad ingestion modes with which data engineers typically work are as follows:

- Application data
- Streaming data
- Batch data

Application Data

Application data is generated by applications, including mobile apps, and pushed to backend services. This data includes user-generated data, like a name and shipping address collected as part of a sales transaction. It also includes data generated by the application, such as log data. Event data, like clickstream data, is also a type of application-generated data. The volume of this kind of data depends on the number of users of the application, the types of data the application generates, and the duration of time the application is in use. This size of application data that is sent in a single operation can vary widely. A clickstream event may have less than 1KB of data, whereas an image upload could be multiple megabytes. Examples of application data include the following:

- Transactions from an online retail application
- Clickstream data from users reading articles on a news site
- Log data from a server running computer-aided design software
- User registration data from an online service

Application data can be ingested by services running in Compute Engine, Kubernetes Engine, or App Engine, for example. Application data can also be written to Stackdriver Logging or one of the managed databases, such as Cloud SQL or Cloud Datastore.

Streaming Data

Streaming data is a set of data that is typically sent in small messages that are transmitted continuously from the data source. Streaming data may be sensor data, which is data generated at regular intervals, and event data, which is data generated in response to a particular event. Examples of streaming data include the following:

- Virtual machine monitoring data, such as CPU utilization rates and memory consumption data
- An IoT device that sends temperature, humidity, and pressure data every minute
- A customer adding an item to an online shopping cart, which then generates an event with data about the customer and the item

Streaming data often includes a timestamp indicating the time that the data was generated. This is often called the *event time*. Some applications will also track the time that data arrives at the beginning of the ingestion pipeline. This is known as the *process time*. Time-series data may require some additional processing early in the ingestion process. If a stream of data needs to be in time order for processing, then late arriving data will need to be inserted in the correct position in the stream. This can require buffering of data for a short period of time in case the data arrives out of order. Of course, there is a maximum amount of time to wait before processing data. These and other issues related to processing streaming data are discussed in Chapter 4, "Designing a Data Processing Solution."

Streaming data is well suited for Cloud Pub/Sub ingestion, which can buffer data while applications process the data. During spikes in data ingestion in which application instances cannot keep up with the rate data is arriving, the data can be preserved in a Cloud Pub/Sub topic and processed later after application instances have a chance to catch up. Cloud Pub/Sub has global endpoints and uses GCP's global frontend load balancer to support ingestion. The messaging service scales automatically to meet the demands of the current workload.

Batch Data

Batch data is ingested in bulk, typically in files. Examples of batch data ingestion include uploading files of data exported from one application to be processed by another. Examples of batch data include the following:

- Transaction data that is collected from applications may be stored in a relational database and later exported for use by a machine learning pipeline
- Archiving data in long-term storage to comply with data retention regulations
- Migrating an application from on premises to the cloud by uploading files of exported data

Google Cloud Storage is typically used for batch uploads. It may also be used in conjunction with Cloud Transfer Service and Transfer Appliance when uploading large volumes of data.

Once data enters the GCP platform through ingestion, it can be stored for longer-term access by other applications or services.

Store

The focus of the storage stage of the data lifecycle is to make data available for transformation and analysis. Several factors influence the choice of storage system, including

- How the data is accessed—by individual record (row) or by an aggregation of columns across many records (rows)
- The way access controls need to be implemented, at the schema or database level or finer-grained level
- How long the data will be stored

These three characteristics are the minimum that should be considered when choosing a storage system; there may be additional criteria for some use cases. (Structure is another factor and is discussed later in this chapter.)

Data Access Patterns

Data is accessed in different ways. Online transaction processing systems often query for specific records using a set of filtering parameters. For example, an e-commerce application may need to look up a customer shipping address from a data store table that holds tens of thousands of addresses. Databases, like Cloud SQL and Cloud Datastore, provide that kind of query functionality.

In another example, a machine learning pipeline might begin by accessing files with thousands of rows of data that is used for training the model. Since machine learning models are often trained in batch mode, all of the training data is needed. Cloud Storage is a good option for storing data that is accessed in bulk.

If you need to access files using filesystem operations, then Cloud Filestore is a good option.

Access Controls

Security and access control in particular also influence how data is stored.

Relational databases, like Cloud SQL and Cloud Spanner, provide mechanisms to restrict access to tables and views. Some users can be granted permission to update data, whereas others can only view data, and still others are not allowed any direct access to data in the database. Fine-grained security can be implemented at the application level or by creating views that limit the data available to some users.

Some access controls are coarse grained. For example, Cloud Storage can limit access based on bucket permissions and access control lists on objects stored in a bucket. If a user has access to a file in the bucket, then they will have access to all the data in that file. Cloud Storage treats files as atomic objects; there is no concept of a row of data, for example, in Cloud Storage as there is in a relational database.

In some cases, you may be able to use other security features of a service along with access controls. BigQuery, for example, is an analytical database used for data warehousing, data analytics, and machine learning. Data is organized into datasets, which are groups of tables and views. At the current time, BigQuery supports dataset-level access controls but not access controls on tables or views directly. One way to work around these limitations is to create authorized views in one dataset that reference tables in another dataset. The dataset with the authorized views can have one set of access controls whereas the dataset with the source tables can have more restrictive access controls.

When choosing a data store, it is important to consider access control requirements and how well a storage system supports those requirements.

Time to Store

Consider how long data will be stored when choosing a data store. Some data is transient. For example, data that is needed only temporarily by an application running on a Compute Engine instance could be stored on a local solid-state drive (SSD) on the instance. As long as the data can be lost when the instance shuts down, this could be a reasonable option.

Data is often needed longer than the lifetime of a virtual machine instance, so other options are better fits for those cases. Cloud Storage is a good option for long-term storage, especially if you can make use of storage lifecycle policies to migrate older data to Nearline or Coldline storage. For long-lived analytics data, Cloud Storage or BigQuery are good options, since the costs are similar.

Nearline storage is used for data that is accessed less than once per 30 days. Coldline storage is used to store data accesses less than once per year.

Data that is frequently accessed is often well suited for either relational or NoSQL databases. As data ages, it may not be as likely to be accessed. In those cases, data can be deleted or exported and archived. If the data is not likely to be used for other purposes, such as machine learning, and there are no regulations that require you to keep the older data, then deleting it may be the best option. In cases where the data can be useful for other purposes or you are required to retain data, then exporting and storing it in Cloud Storage is an option. Then, if the data needs to be accessed, it can be imported to the database and queried there.

Process and Analyze

During the process and analyze stage, data is transformed into forms that make the data readily available to ad hoc querying or other forms of analysis.

Data Transformations

Transformations include data cleansing, which is the process of detecting erroneous data and correcting it. Some cleansing operations are based on the data type of expected data.

For example, a column of data containing only numeric data should not have alphabetic characters in the column. The cleansing process could delete rows of data that have alphabetic characters in that column. It could alternatively keep the row and substitute another value, such as a zero, or treat the value as NULL.

In other cases, business logic is applied to determine incorrect data. Some business logic rules may be simple, such as that an order date cannot be earlier than the date that the business began accepting orders. An example of a more complex rule is not allowing an order total to be greater than the credit limit assigned to a customer.

The decision to keep the row or delete it will depend on the particular use case. A set of telemetry data arriving at one-minute intervals may include an invalid value. In that case, the invalid value may be dropped without significantly affecting hour-level aggregates. A customer order that violates a business rule, however, might be kept because orders are significant business events. In this case, the order should be processed by an exception-handling process.

Transformations also include normalizing or standardizing data. For example, an application may expect phone numbers in North America to include a three-digit area code. If a phone number is missing an area code, the area code can be looked up based on the associated address. In another case, an application may expect country names specified using the International Organization for Standardization (ISO) 3166 alpha-3 country code, in which case data specifying Canada would be changed to CAN.

Cloud Dataflow is well suited to transforming both stream and batch data. Once data has been transformed, it is available for analysis.

Data Analysis

In the analyze stage, a variety of techniques may be used to extract useful information from data. Statistical techniques are often used with numeric data to do the following:

- Describe characteristics of a dataset, such as a mean and standard deviation of the dataset.
- Generate histograms to understand the distribution of values of an attribute.
- Find correlations between variables, such as customer type and average revenue per sales order.
- Make predictions using regression models, which allow you to estimate one attribute based on the value of another. In statistical terms, regression models generate predictions of a dependent variable based on the value of an independent variable.
- Cluster subsets of a dataset into groups of similar entities. For example, a retail sales dataset may yield groups of customers who purchase similar types of products and spend similar amounts over time.

Text data can be analyzed as well using a variety of techniques. A simple example is counting the occurrences of each word in a text. A more complex example is extracting entities, such as names of persons, businesses, and locations, from a document.

Cloud Dataflow, Cloud Dataproc, BigQuery, and Cloud ML Engine are all useful for data analysis.

Explore and Visualize

Often when working with new datasets, you'll find it helpful to explore the data and test a hypothesis. Cloud Datalab, which is based on Jupyter Notebooks (`http://jupyter.org`), is a GCP tool for exploring, analyzing, and visualizing data sets. Widely used data science and machine learning libraries, such as pandas, scikit-learn, and TensorFlow, can be used with Datalab. Analysts use Python or SQL to explore data in Cloud Datalab.

Google Data Studio is useful if you want tabular reports and basic charts. The drag-and-drop interface allows nonprogrammers to explore datasets without having to write code.

As you prepare for the Google Cloud Professional Data Engineer exam, keep in mind the four stages of the data lifecycle—ingestion, storage, process and analyze, and explore and visualize. They provide an organizing framework for understanding the broad context of data engineering and machine learning.

Technical Aspects of Data: Volume, Velocity, Variation, Access, and Security

GCP has a wide variety of data storage services. They are each designed to meet some use cases, but certainly not all of them. Earlier in the chapter, we considered data storage from a business perspective, and in this section, we will look into the more technical aspects of data storage. Some of the characteristics that you should keep in mind when choosing a storage technology are as follows:

- The volume and velocity of data
- Variation in structure
- Data access patterns
- Security requirements

Knowing one of these characteristics will not likely determine the single storage technology you should use. However, a single mismatch between the data requirements and a storage service's features can be enough to eliminate that service from consideration.

Volume

Some storage services are designed to store large volumes of data, including petabyte scales, whereas others are limited to smaller volumes.

Cloud Storage is an example of the former. An individual item in Cloud Storage can be up to 5 TB, and there is no limit to the number of read or write operations. Cloud Bigtable, which is used for telemetry data and large-volume analytic applications, can store up to 8 TB per node when using hard disk drives, and it can store up to 2.5 TB per node when

using SSDs. Each Bigtable instance can have up to 1,000 tables. BigQuery, the managed data warehouse and analytics database, has no limit on the number of tables in a dataset, and it may have up to 4,000 partitions per table. Persistent disks, which can be attached to Compute Engine instances, can store up to 64 TB.

Single MySQL First Generation instances are limited to storing 500 GB of data. Second Generation instances of MySQL, PostgreSQL, and SQL Server can store up to 30 TB per instance. In general, Cloud SQL is a good choice for applications that need a relational database and that serve requests in a single region.

The limits specified here are the limits that Google has in place as of this writing. They may have changed by the time you read this. Always use Google Cloud documentation for the definitive limits of any GCP service.

Velocity

Velocity of data is the rate at which it is sent to and processed by an application. Web applications and mobile apps that collect and store human-entered data are typically low velocity, at least when measured by individual user. Machine-generated data, such IoT and time-series data, can be high velocity, especially when many different devices are generating data at short intervals of time. Here are some examples of various rates for low to high velocity:

- Nightly uploads of data to a data
- Hourly summaries of the number of orders taken in the last hour
- Analysis of the last three minutes of telemetry data
- Alerting based on a log message as soon as it is received is an example of real-time processing

If data is ingested and written to storage, it is important to match the velocity of incoming data with the rate at which the data store can write data. For example, Bigtable is designed for high-velocity data and can write up to 10,000 rows per second using a 10-node cluster with SSDs. When high-velocity data is processed as it is ingested, it is a good practice to write the data to a Cloud Pub/Sub topic. The processing application can then use a pull subscription to read the data at a rate that it can sustain. Cloud Pub/Sub is a scalable, managed messaging service that scales automatically. Users do not have to provision resources or configure scaling parameters.

At the other end of the velocity spectrum are low-velocity migrations or archiving operations. For example, an organization that uses the Transfer Appliance for large-scale migration may wait days before the data is available in Cloud Storage.

Variation in Structure

Another key attribute to consider when choosing a storage technology is the amount of variation that you expect in the data structure. Some data structures have low variance. For example, a weather sensor that sends temperature, humidity, and pressure readings at regular time intervals has virtually no variation in the data structure. All data sent to the storage system will have those three measures unless there is an error, such as a lost network packet or corrupted data.

Many business applications that use relational databases also have limited variation in data structure. For example, all customers have most attributes in common, such as name and address, but other business applications may have name suffixes, such as M.D. and Ph.D., stored in an additional field. In those cases, it is common to allow NULL values for attributes that may not be needed.

Not all business applications fit well into the rigid structure of strictly relational databases. NoSQL databases, such as MongoDB, CouchDB, and OrientDB, are examples of document databases. These databases use sets of key-value pairs to represent varying attributes. For example, instead of having a fixed set of attributes, like a relational database table, they include the attribute name along with the attribute value in the database (see Table 1.1).

TABLE 1.1 Example of structured, relational data

First_name	Last_name	Street_Address	City	Postal_Code
Michael	Johnson	334 Bay Rd	Santa Fe	87501
Wang	Li	74 Alder St	Boise	83701
Sandra	Connor	123 Main St	Los Angeles	90014

The data in the first row would be represented in a document database using a structure something like the following:

```
{
'first_name': 'Michael',
'last_name': 'Johnson'.
'street'_address': '334 Bay Rd',
'city': 'Santa Fe',
'postal_code': '87501'
}
```

Since most rows in a table of names and addresses will have the same attributes, it is not necessary to use a data structure like a document structure. Consider the case of a product

catalog that lists both appliances and furniture. Here is an example of how a dishwasher and a chair might be represented:

```
{
{'id': '123456',
'product_type': 'dishwasher',
'length': '24 in',
'width': '34 in',
'weight': '175 lbs',
'power': '1800 watts'
}
{'id':'987654',
'product_type': 'chair',
'weight': '15 kg',
'style': 'modern',
'color': 'brown'
}
}
```

In addition to document databases, wide-column databases, such as Bigtable and Cassandra, are also used with datasets with varying attributes.

Data Access Patterns

Data is accessed in different ways for different use cases. Some time-series data points may be read immediately after they are written, but they are not likely to be read once they are more than a day old. Customer order data may be read repeatedly as an order is processed. Archived data may be accessed less than once a year. Four metrics to consider about data access are as follows:

- How much data is retrieved in a read operation?
- How much data is written in an insert operation?
- How often is data written?
- How often is data read?

Some read and write operations apply to small amounts of data. Reading or writing a single piece of telemetry data is an example. Writing an e-commerce transaction may also entail a small amount of data. A database storing telemetry data from thousands of sensors that push data every five seconds will be writing large volumes, whereas an online transaction processing database for a small online retailer will also write small individual units of data but at a much smaller rate. These will require different kinds of databases. The telemetry data, for example, is better suited to Bigtable, with its low-latency writes, and the retailer transaction data is a good use case for Cloud SQL, with support for sufficient I/O operations to handle relational database loads.

Cloud Storage supports ingesting large volumes of data in bulk using tools such as the Cloud Transfer Service and Transfer Appliance. (Cloud Storage also supports streaming transfers, but bulk reads and writes are more common.) Data in Cloud Storage is read at the object or the file level. You typically don't, for example, seek a particular block within a file as you can when storing a file on a filesystem.

It is common to read large volumes of data in BigQuery as well; however, in that case we often read a small number of columns across a large number of rows. BigQuery optimizes for these kinds of reads by using a columnar storage format known as Capacitor. *Capacitor* is designed to store semi-structured data with nested and repeated fields.

Data access patterns can help identify the best storage technology for a use case by highlighting key features needed to support those access patterns.

Security Requirements

Different storage systems will have different levels of access controls. Cloud Storage, for example, can have access controls at the bucket and the object level. If someone has access to a file in Cloud Storage, they will have access to all the data in that file. If some users have access only to a subset of a dataset, then the data could be stored in a relational database and a view could be created that includes only the data that the user is allowed to access.

Encrypting data at rest is an important requirement for many use cases; fortunately, all Google Cloud storage services encrypt data at rest.

When choosing a storage technology, the ability to control access to data is a key consideration.

Types of Structure: Structured, Semi-Structured, and Unstructured

For the purposes of choosing a storage technology, it is helpful to consider how data is structured. There are three widely recognized categories:

- Structured
- Semi-structured
- Unstructured

These categories are particularly helpful when choosing a database.

Structured Data *Structured data* has a fixed set of attributes that can be modeled in a table of rows and columns.

Semi-Structured Data *Semi-structured data* has attributes like structured data, but the set of attributes can vary from one instance to another. For example, a product description of

an appliance might include length, width, height, weight, and power consumption. A chair in the same catalog might have length, width, height, color, and style as attributes. Semi-structured data may be organized using arrays or sets of key-value pairs.

Unstructured Data *Unstructured data* does not fit into a tabular structure. Images and audio files are good examples of unstructured data. In between these two extremes lies semi-structured data, which has characteristics of both structured and unstructured.

Structured: Transactional vs. Analytical

Structured data can be represented in tables of columns and rows, where columns are attributes and rows are records or entities. Table 1.1 showed an example of structured data. Structured data may be oriented to transactional processes or analytical use cases.

Transactional structured data is often operated on one row at a time. For example, a business application may look up a customer's account information from a customer table when displaying data about the customer's shipping address. Multiple columns from a single row will be used, so it is efficient to store all row attributes together in a data block. Retrieving a single data block will retrieve all the needed data. This is a common pattern in transactional databases such as Cloud SQL and Cloud Spanner, which use row-oriented storage.

Now consider a data warehousing example. A business analyst is working with a sales data mart and wants to understand how sales last month compare to the same period last year. The data mart has one row for each product on each date, which include the following attributes in addition to product and date: number of units sold, total revenue for units sold, average unit price, average marginal revenue, and total marginal revenue. The analyst is only interested in the monthly sums of total revenue for units sold for each product. In this case, the analyst would query many rows and only three columns. Instead of retrieving the full row for all rows selected, it is more efficient to retrieve only the date, product, and total revenue of units sold columns. This is a common pattern in analytical applications and the reason why BigQuery uses a column-oriented storage mechanism.

Semi-Structured: Fully Indexed vs. Row Key Access

Semi-structured data, as noted earlier, does not follow a fixed tabular format and instead stores schema attributes along with the data. In the case of document databases, this allows developers to add attributes as needed without making changes to a fixed database schema. Two ways of storing semi-structured data are as documents or as wide columns. An important distinction between the two is how data is retrieved from them.

Fully Indexed, Semi-Structured Data

Let's consider the simple product catalog example again. There are many ways that shoppers might want to search for information about products. If they are looking for a

dishwasher, for example, they might want to search based on size or power consumption. When searching for furniture, style and color are important considerations.

```
{
{'id': '123456',
'product_type': 'dishwasher',
'length': '24 in',
'width': '34 in',
'weight': '175 lbs',
'power': '1800 watts'
}
{'id':'987654',
'product_type': 'chair',
'weight': '15 kg',
'style': 'modern',
'color': 'brown'
}
}
```

To search efficiently by attributes, document databases allow for indexes. If you use Cloud Datastore, for example, you could create indexes on each of the attributes as well as a combination of attributes. Indexes should be designed to support the way that data is queried. If you expect users to search for chairs by specifying style and color together, then you should create a style and color index. If you expect customers to search for appliances by their power consumption, then you should create an index on power.

Creating a large number of indexes can significantly increase the amount of storage used. In fact, it is not surprising to have total index storage greater than the amount of storage used to store documents. Also, additional indexes can negatively impact performance for insert, update, and delete operations, because the indexes need to be revised to reflect those operations.

Row Key Access

Wide-column databases usually take a different approach to querying. Rather than using indexes to allow efficient lookup of rows with needed data, wide-column databases organize data so that rows with similar row keys are close together. Queries use a row key, which is analogous to a primary key in relational databases, to retrieve data. This has two implications.

Tables in wide-column databases are designed to respond to particular queries. Although relational databases are designed according to forms of normalization that minimize the risk of data anomalies, wide-column databases are designed for low-latency reads and writes at high volumes. This can lead to duplication of data. Consider IoT sensor data stored in a wide-column database. Table 1.2 shows IoT data organized by sensor ID and timestamp (milliseconds since January 1, 1970 00:00:00 UTC). Future rows would feature the same sensor ID but different corresponding timestamps, and the row key would be determined by both.

TABLE 1.2 IoT data by sensor ID and timestamp

Sensor ID	Timestamp	Temperature	Relative humidity	Pressure
789	1571760690	40	35	28.2
790	1571760698	42.5	50	29.1
791	1571760676	37	61	28.6

Table 1.2 is organized to answer queries that require looking up data by sensor ID and then time. It is not well suited for looking up data by time—for example, all readings over the past hour. Rather than create an index on timestamp, wide-column databases duplicate data in a different row key order. Table 1.3, for example, is designed to answer time range queries. Note that a new table must be created with the desired schema to accomplish this—there is no index that is used to support the query pattern.

TABLE 1.3 IoT data by timestamp and sensor ID

Timestamp	Sensor ID	Temperature	Relative humidity	Pressure
1571760676	791	37	61	28.6
1571760690	789	40	35	28.2
1571760698	790	42.5	50	29.1

Unstructured Data

The distinguishing characteristic of unstructured data is that it does not have a defined schema or data model. Structured data, like relational database tables, has a fixed data model that is defined before data is added to the table. Semi-structured databases include a schema with each row or document in the database. Examples of unstructured data include the following:

- Text files of natural language content
- Audio files
- Video files
- Binary large objects (BLOBs)

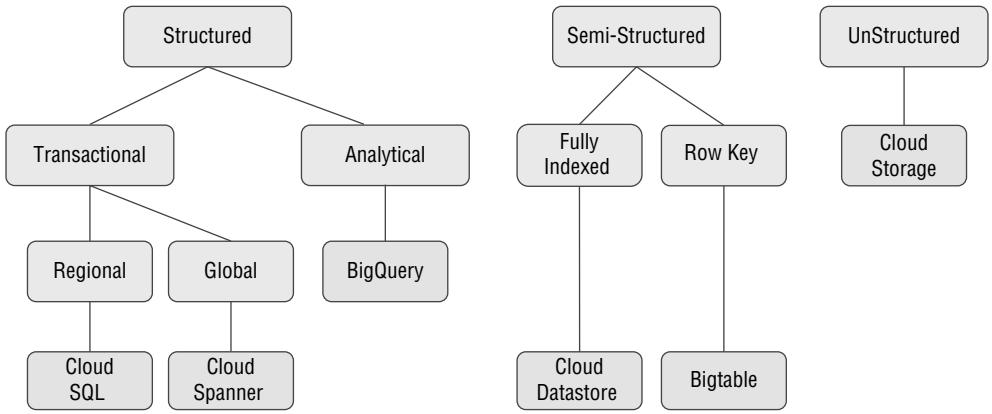
It should be pointed out that data is considered unstructured if it does not have a schema that influences how the data is stored or accessed. Unstructured data may have an internal structure that is not relevant to the way it is stored. For example, natural language is highly

structured according to the syntax rules of languages. Audio and video files may have an internal format that includes metadata as well as content. Here again, there is structure within the file, but that structure is not used by storage systems, and that is the reason why this kind of data is classified as unstructured.

Google's Storage Decision Tree

Google has developed a decision tree for choosing a storage system that starts with distinguishing structured, semi-structured, and unstructured data. Figure 1.1 is based on the decision tree published at `https://cloud.google.com/solutions/data-lifecycle-cloud-platform`.

FIGURE 1.1 Choosing a storage technology in GCP



Schema Design Considerations

Structured and semi-structured data has a schema associated with it. Structured data is usually stored in relational databases whereas semi-structured data is often stored in NoSQL databases. The *schema* influences how data is stored and accessed, so once you have determined which kind of storage technology to use, you may then need to design a schema that will support optimal storage and retrieval.

The distinction between relational and NoSQL databases is becoming less pronounced as each type adopts features of the other. Some relational databases support storing and querying JavaScript Object Notation (JSON) structures, similar to the way that document databases do. Similarly, some NoSQL databases now support ACID (atomicity, consistency, isolation, durability) transactions, which are a staple feature of relational databases.

Relational Database Design

Data modeling for relational databases begins with determining which type of relational database you are developing: an online transaction processing (OLTP) database or an online analytical processing (OLAP) database.

OLTP

Online transaction processing (OLTP) databases are designed for transaction processing and typically follow data normalization rules. There are currently 10 recognized forms of normalization, but most transaction processing systems follow no more than three of those forms:

- The *first form of normalization* requires that each column in the table have an atomic value, no repeating groups, and a primary key, which is one or more ordered columns that uniquely identify a row.
- The *second form of normalization* includes the first form and creates separate tables for values that apply to multiple rows and links them using foreign keys. A *foreign key* is one or more ordered columns that correspond to a primary key in another table.
- The *third form of normalization*, which includes the second form, eliminates any columns from a table that does not depend on the key.

These rules of normalization are designed to reduce the risk of data anomalies and to avoid the storage of redundant data. Although they serve those purposes well, they can lead to high levels of I/O operations when joining tables or updating a large number of indexes. Using an OLTP data model requires a balance between following the rules of normalization to avoid anomalies and designing for performance.

Denormalization—that is, intentionally violating one of the rules of normalization—is often used to improve query performance. For example, repeating customer names in both the customer table and an order table could avoid having to join the two tables when printing invoices. By denormalizing, you can reduce the need to join tables since the data that would have been in another table is stored along with other data in the row of one table.

OLAP

Online analytical processing (OLAP) data models are often used for data warehouse and data mart applications. OLAP models are also called *dimensional models* because data is organized around several dimensions. OLAP models are designed to facilitate the following:

- Rolling up and aggregating data
- Drilling down from summary data to detailed data
- Pivoting and looking at data from different dimensions—sometimes called *slicing and dicing*

OLAP can be implemented in relational database or in specialized multidimensional data stores.

SQL Crash Course

In-depth knowledge of SQL is not necessarily required to pass the Google Cloud Professional Data Engineer exam, but knowledge of SQL may help if a question includes a SQL statement.

SQL has three types of statements that developers use:

- *Data definition language (DDL) statements*, which are used to create and modify database schemas
- *Data manipulation language (DML) statements*, which are used to insert, update, delete, and query data
- *Data query language (DQL) statements*, which is a single statement: `SELECT`

Table 1.4 shows examples of data definition statements and their function. Table 1.5 shows data manipulation examples, and Table 1.6 shows query language examples.

TABLE 1.4 Data definition language examples

DDL statement	Example	Explanation
`CREATE TABLE`	`CREATE TABLE address (address_id INT PRIMARY KEY, street_name VARCHAR(50), city VARCHAR(50), state VARCHAR(2) );`	Creates a table with four columns. The first is an integer and the primary key; the other three are variable-length character strings.
`CREATE INDEX`	`CREATE INDEX addr_idx ON address(state);`	Creates an index on the state column of the address table.
`ALTER TABLE`	`ALTER TABLE address ADD (zip VARCHAR(9));`	Adds a column called zip to the address table. `ALTER` is also used to modify and drop entities.
`DROP INDEX`	`DROP INDEX addr_idx;`	Deletes the index `addr_idx`.

TABLE 1.5 Data manipulation language examples

Data Manipulation Language

DML Statement	Example	Explanation
INSERT	`INSERT INTO address VALUES (1234, '56 Main St', 'Seattle', 'WA');`	Adds rows to the table with the specified values, which are in column order
UPDATE	`UPDATE address SET state = 'OR'`	Sets the value of the state column to `'OR'` for all rows
DELETE	`DELETE FROM address WHERE state = 'OR'`	Removes all rows that have the value `'OR'` in the state column

TABLE 1.6 Data query language examples

Data Query Language

DDL statement	Example	Explanation
SELECT … FROM	`SELECT address_id, state FROM address`	Returns the `address_id` and state values for all rows in the address table
SELECT … FROM … WHERE	`SELECT address_id, state FROM address WHERE state = 'OR'`	Returns the `address_id` and state values for all rows in the address table that have the value `'OR'` in the state column
SELECT … FROM … GROUP BY	`SELECT state, COUNT(*) FROM address GROUP BY state`	Returns the number of addresses in each state
SELECT … FROM … GROUP BY … HAVING	`SELECT state, COUNT(*) FROM address GROUP BY state HAVING COUNT(*) > 50`	Returns the number of addresses in each state that has at least 50 addresses

NoSQL Database Design

NoSQL databases are less structured than relational databases, and there is no formal model, like relational algebra and forms of normalization, that apply to all NoSQL databases. The four types of NoSQL databases available in GCP are

- Key-value
- Document
- Wide column
- Graph

Each type of NoSQL database is suited for different use cases depending on data ingestion, entity relationships, and query requirements.

Key-Value Data Stores

Key-value data stores are databases that use associative arrays or dictionaries as the basic datatype. Keys are data used to look up values. An example of key-value data is shown in Table 1.7, which displays a mapping from names of machine instances to names of partitions associated with each instance.

TABLE 1.7 Examples of key-value data

Key	Value
Instance1	PartitionA
Instance2	PartitionB
Instance3	PartitionA
Instance4	PartitionC

Key-value data stores are simple, but it is possible to have more complex data structures as values. For example, a JSON object could be stored as a value. This would be reasonable use of a key-value data store if the JSON object was only looked up by the key, and there was no need to search on items within the JSON structure. In situations where items in the JSON structure should be searchable, a document database would be a better option.

Cloud Memorystore is a fully managed key-value data store based on Redis, a popular open source key-value datastore. As of this writing, Cloud Memorystore does not support persistence, so it should not be used for applications that do not need to save data to persistent storage. Open source Redis does support persistence. If you wanted to use Redis for a key-value store and wanted persistent storage, then you could run and manage your own Redis service in Compute Engine or Kubernetes Engine.

Document Databases

Document stores allow complex data structures, called *documents*, to be used as values and accessed in more ways than simple key lookup. When designing a data model for *document databases*, documents should be designed to group data that is read together.

Consider an online game that requires a database to store information about players' game state. The player state includes

- Player name
- Health score
- List of possessions
- List of past session start and end times
- Player ID

The player name, health score, and list of possessions are often read together and displayed for players. The list of sessions is used only by analysts reviewing how players use the game. Since there are two different use cases for reading the data, there should be two different documents. In this case, the first three attributes should be in one document along with the player ID, and the sessions should be in another document with player ID.

When you need a managed document database in GCP, use Cloud Datastore. Alternatively, if you wish to run your own document database, MongoDB, CouchDB, and OrientDB are options.

Wide-Column Databases

Wide-column databases are used for use cases with the following:

- High volumes of data
- Need for low-latency writes
- More write operations than read operations
- Limited range of queries—in other words, no ad hoc queries
- Lookup by a single key

Wide-column databases have a data model similar to the tabular structure of relational tables, but there are significant differences. Wide-column databases are often sparse, with the exception of IoT and other time-series databases that have few columns that are almost always used.

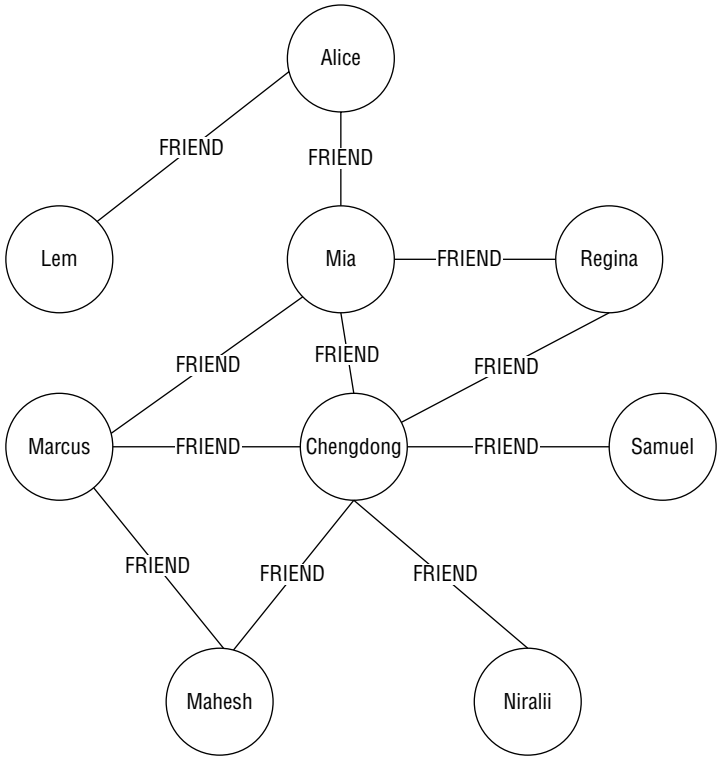
Bigtable is GCP's managed wide-column database. It is also a good option for migrating on-premises Hadoop HBase databases to a managed database because Bigtable has an HBase interface. If you wish to manage your own wide column, Cassandra is an open source option that you can run in Compute Engine or Kubernetes Engine.

Graph Databases

Another type of NoSQL database are *graph databases*, which are based on modeling entities and relationships as nodes and links in a graph or network. Social networks are a good

example of a use case for graph databases. People could be modeled as nodes in the graph, and relationships between people are links, also called *edges*. For example, Figure 1.2 shows an example graph of friends showing Chengdong with the most friends, 6, and Lem with the fewest, 1.

FIGURE 1.2 Example graph of friends



Data is retrieved from a graph using one of two types of queries. One type of query uses SQL-like declarative statements describing patterns to look for in a graph, such as the following the Cypher query language. This query returns a list of persons and friends of that person's friends:

```
MATCH (n:Person)-[:FRIEND]-(f)
MATCH (n)-[:FRIEND]-()-[:FRIEND]-(fof)
RETURN n, fof
```

The other option is to use a traversal language, such as Gremlin, which specifies how to move from node to node in the graph.

GCP does not have a managed graph database, but Bigtable can be used as the storage backend for HGraphDB (`https://github.com/rayokota/hgraphdb`) or JanusGraph (`https://janusgraph.org`).

Exam Essentials

Know the four stages of the data lifecycle: ingest, storage, process and analyze, and explore and visualize. Ingestion is the process of bringing application data, streaming data, and batch data into the cloud. The storage stage focuses on persisting data to an appropriate storage system. Processing and analyzing is about transforming data into a form suitable for analysis. Exploring and visualizing focuses on testing hypotheses and drawing insights from data.

Understand the characteristics of streaming data. Streaming data is a set of data that is sent in small messages that are transmitted continuously from the data source. Streaming data may be telemetry data, which is data generated at regular intervals, and event data, which is data generated in response to a particular event. Stream ingestion services need to deal with potentially late and missing data. Streaming data is often ingested using Cloud Pub/Sub.

Understand the characteristics of batch data. Batch data is ingested in bulk, typically in files. Examples of batch data ingestion include uploading files of data exported from one application to be processed by another. Both batch and streaming data can be transformed and processed using Cloud Dataflow.

Know the technical factors to consider when choosing a data store. These factors include the volume and velocity of data, the type of structure of the data, access control requirements, and data access patterns.

Know the three levels of structure of data. These levels are structured, semi-structured, and unstructured. Structured data has a fixed schema, such as a relational database table. Semi-structured data has a schema that can vary; the schema is stored with data. Unstructured data does not have a structure used to determine how to store data.

Know which Google Cloud storage services are used with the different structure types. Structured data is stored in Cloud SQL and Cloud Spanner if it is used with a transaction processing system; BigQuery is used for analytical applications of structured data. Semi-structured data is stored in Cloud Datastore if data access requires full indexing; otherwise, it can be stored in Bigtable. Unstructured data is stored in Cloud Storage.

Know the difference between relational and NoSQL databases. Relational databases are used for structured data whereas NoSQL databases are used for semi-structured data. The four types of NoSQL databases are key-value, document, wide-column, and graph databases.

Review Questions

You can find the answers in the appendix.

1. A developer is planning a mobile application for your company's customers to use to track information about their accounts. The developer is asking for your advice on storage technologies. In one case, the developer explains that they want to write messages each time a significant event occurs, such as the client opening, viewing, or deleting an account. This data is collected for compliance reasons, and the developer wants to minimize administrative overhead. What system would you recommend for storing this data?
   - **A.** Cloud SQL using MySQL
   - **B.** Cloud SQL using PostgreSQL
   - **C.** Cloud Datastore
   - **D.** Stackdriver Logging

2. You are responsible for developing an ingestion mechanism for a large number of IoT sensors. The ingestion service should accept data up to 10 minutes late. The service should also perform some transformations before writing the data to a database. Which of the managed services would be the best option for managing late arriving data and performing transformations?
   - **A.** Cloud Dataproc
   - **B.** Cloud Dataflow
   - **C.** Cloud Dataprep
   - **D.** Cloud SQL

3. A team of analysts has collected several CSV datasets with a total size of 50 GB. They plan to store the datasets in GCP and use Compute Engine instances to run RStudio, an interactive statistical application. Data will be loaded into RStudio using an RStudio data loading tool. Which of the following is the most appropriate GCP storage service for the datasets?
   - **A.** Cloud Storage
   - **B.** Cloud Datastore
   - **C.** MongoDB
   - **D.** Bigtable

4. A team of analysts has collected several terabytes of telemetry data in CSV datasets. They plan to store the datasets in GCP and query and analyze the data using SQL. Which of the following is the most appropriate GCP storage service for the datasets?
   - **A.** Cloud SQL
   - **B.** Cloud Spanner
   - **C.** BigQuery
   - **D.** Bigtable

5. You have been hired to consult with a startup that is developing software for self-driving vehicles. The company's product uses machine learning to predict the trajectory of persons and vehicles. Currently, the software is being developed using 20 vehicles, all located in the same city. IoT data is sent from vehicles every 60 seconds to a MySQL database running on a Compute Engine instance using an n2-standard-8 machine type with 8 vCPUs and 16 GB of memory. The startup wants to review their architecture and make any necessary changes to support tens of thousands of self-driving vehicles, all transmitting IoT data every second. The vehicles will be located across North America and Europe. Approximately 4 KB of data is sent in each transmission. What changes to the architecture would you recommend?

   A. None. The current architecture is well suited to the use case.

   B. Replace Cloud SQL with Cloud Spanner.

   C. Replace Cloud SQL with Bigtable.

   D. Replace Cloud SQL with Cloud Datastore.

6. As a member of a team of game developers, you have been tasked with devising a way to track players' possessions. Possessions may be purchased from a catalog, traded with other players, or awarded for game activities. Possessions are categorized as clothing, tools, books, and coins. Players may have any number of possessions of any type. Players can search for other players who have particular possession types to facilitate trading. The game designer has informed you that there will likely be new types of possessions and ways to acquire them in the future. What kind of a data store would you recommend using?

   A. Transactional database

   B. Wide-column database

   C. Document database

   D. Analytic database

7. The CTO of your company wants to reduce the cost of running an HBase and Hadoop cluster on premises. Only one HBase application is run on the cluster. The cluster currently supports 10 TB of data, but it is expected to double in the next six months. Which of the following managed services would you recommend to replace the on-premises cluster in order to minimize migration and ongoing operational costs?

   A. Cloud Bigtable using the HBase API

   B. Cloud Dataflow using the HBase API

   C. Cloud Spanner

   D. Cloud Datastore

8. A genomics research institute is developing a platform for analyzing data related to genetic diseases. The genomics data is in a specialized format known as FASTQ, which stores nucleotide sequences and quality scores in a text format. Files may be up to 400 GB and are uploaded in batches. Once the files finish uploading, an analysis pipeline runs, reads the data in the FASTQ file, and outputs data to a database. What storage system is a good option for storing the uploaded FASTQ data?

   A. Cloud Bigtable

   B. Cloud Datastore

   C. Cloud Storage

   D. Cloud Spanner

9. A genomics research institute is developing a platform for analyzing data related to genetic diseases. The genomics data is in a specialized format known as FASTQ, which stores nucleotide sequences and quality scores in a text format. Once the files finish uploading, an analysis pipeline runs, reads the data in the FASTQ file, and outputs data to a database. The output is in tabular structure, the data is queried using SQL, and typically queries retrieve only a small number of columns but many rows. What database would you recommend for storing the output of the workflow?

A. Cloud Bigtable

B. Cloud Datastore

C. Cloud Storage

D. BigQuery

10. You are developing a new application and will be storing semi-structured data that will only be accessed by a single key. The total volume of data will be at least 40 TB. What GCP database service would you use?

A. BigQuery

B. Bigtable

C. Cloud Spanner

D. Cloud SQL

11. A group of climate scientists is collecting weather data every minute from 10,000 sensors across the globe. Data often arrives near the beginning of a minute, and almost all data arrives within the first 30 seconds of a minute. The data ingestion process is losing some data because servers cannot ingest the data as fast as it is arriving. The scientists have scaled up the number of servers in their managed instance group, but that has not completely eliminated the problem. They do not wish to increase the maximum size of the managed instance group. What else can the scientists do to prevent data loss?

A. Write data to a Cloud Dataflow stream

B. Write data to a Cloud Pub/Sub topic

C. Write data to Cloud SQL table

D. Write data to Cloud Dataprep

12. A software developer asks your advice about storing data. The developer has hundreds of thousands of 1 KB JSON objects that need to be accessed in sub-millisecond times if possible. All objects are referenced by a key. There is no need to look up values by the contents of the JSON structure. What kind of NoSQL database would you recommend?

A. Key-value database

B. Analytical database

C. Wide-column database

D. Graph database

13. A software developer asks your advice about storing data. The developer has hundreds of thousands of 10 KB JSON objects that need to be searchable by most attributes in the JSON structure. What kind of NoSQL database would you recommend?
    - **A.** Key-value database
    - **B.** Analytical database
    - **C.** Wide-column database
    - **D.** Document database

14. A data modeler is designing a database to support ad hoc querying, including drilling down and slicing and dicing queries. What kind of data model is the data modeler likely to use?
    - **A.** OLTP
    - **B.** OLAP
    - **C.** Normalized
    - **D.** Graph

15. A multinational corporation is building a global inventory database. The database will support OLTP type transactions at a global scale. Which of the following would you consider as possible databases for the system?
    - **A.** Cloud SQL and Cloud Spanner
    - **B.** Cloud SQL and Cloud Datastore
    - **C.** Cloud Spanner only
    - **D.** Cloud Datastore only