## **BigQuery - Geospatial and ML functionality is with BQ**

The exam was heavy on BigQuery questions too. Questions related to Updates on Bigquery, IAM roles, Slots, Storage, etc.

* Update DML: How to use Update DML in BigQuery, how you can handle quota error exceeded in your project. How many simultaneous updates can you run on a daily basis? Best ways to update a table (for example, if you have a partitioned table, etc.).
* Authorized View: I recommend studying all about authorized views, and how you can share queries with your data science team without the necessity of querying the entire columns of a table.
* Allocated Slots and Available Slots: What can you do if you don’t have more slots available and you don’t want to create a new project in your organization?
* Storage: Questions related to the best way to store raw data, for example between BigQuery or Storage. This will depend on the context of the question and if price or performance is the priority.
* BigQuery Data Transfer Service and the connection available with BI tools.
* Partition and clustering
* HASH, Merge, Data manipulation. 2 questions related to this.
* Integration with Google Identity and Access Management (IAM) roles
* Basic understanding of the GCP Key Management Sevice (KMS) and keys (google-managed, customer-supplied, and customer-managed)
* Partitioned tables, specifically as used in SQL commands
* Wildcards
* Federated tables
* Integration with Google Cloud Storage (GCS)
* BigQuery (BQ) data transfer service and connectors
* When to use normalized and denormalized data
* Loading different data formats into BQ, including a good understanding of the Apache Avro™, CSV, Apache Parquet, and JSON formats
* Pricing with slots
* Cached queries

Google BigQuery provides a fully managed data warehouse with a familiar SQL interface, so you can store your IoT data alongside any of your other enterprise analytics and logs. The performance and cost of BigQuery means you might keep your valuable data longer, instead of deleting it just to save disk space.

By creating an authorized view, one assures that the data is current and avoids taking more storage space (and cost) in order to share a dataset.

BigQuery interfaces include Google Cloud console interface and the BigQuery command-line tool. Developers and data scientists can use client libraries with familiar programming including Python, Java, JavaScript, and Go, as well as BigQuery's REST API and RPC API to transform and manage data. ODBC and JDBC drivers provide interaction with existing applications including third-party tools and utilities.

BigQuery stores data using a columnar storage format that is optimized for analytical queries. BigQuery presents data in tables, rows, and columns and provides full support for database transaction semantics (ACID). BigQuery storage is automatically replicated across multiple locations to provide high availability.

**Cloud dataflow is used to stream data to bigquery in near real-time**.

**Clustering**:

In general, there are two typical usage patterns for clustering within a data warehouse: Clustering on columns that have a very high number of distinct values, like userId or transactionId. Clustering on multiple columns that are frequently used together. When clustering by multiple columns, the order of columns you specify is important. The order of the specified columns determines the sort order of the data. You can filter by any prefix of the clustering columns and get the benefits of clustering, like regionId, shopId and productId together; or regionId and shopId; or just regionId.

Refer: <https://cloud.google.com/blog/products/data-analytics/skip-the-maintenance-speed-up-queries-with-bigquerys-clustering>

Based on Google’s best practices, using external tables for ETL is better than loading data to BigQuery.

A partitioned table is a special table that is divided into segments, called partitions, that make it easier to manage and query your data. By dividing a large table into smaller partitions, you can improve query performance, and you can control costs by reducing the number of bytes read by a query.  
There are two types of tables partitioning in BigQuery:  
 – Tables partitioned by ingestion time: Tables partitioned based on the data’s ingestion (load) date or  
 arrival date.  
 – Partitioned tables: Tables that are partitioned based on a TIMESTAMP or DATE column.  
As an alternative to partitioned tables, you can shard tables using a time-based naming approach such as [PREFIX]\_YYYYMMDD. This is referred to as creating date-sharded tables.

Reference(s):  
[https://cloud.google.com/bigquery/docs/partitioned- tables#partitioning\_versus\_sharding](https://cloud.google.com/bigquery/docs/partitioned-%C2%A0tables#partitioning_versus_sharding)

If you split a table into multiple tables (such as one table for each day), then you can limit your query to the data in specific tables (such as for particular days). A better method is to use a partitioned table, as long as your data can be separated by the day.  
If you use the LIMIT clause, BigQuery will still process the entire table.  
Reference: <https://cloud.google.com/bigquery/docs/partitioned-tables>

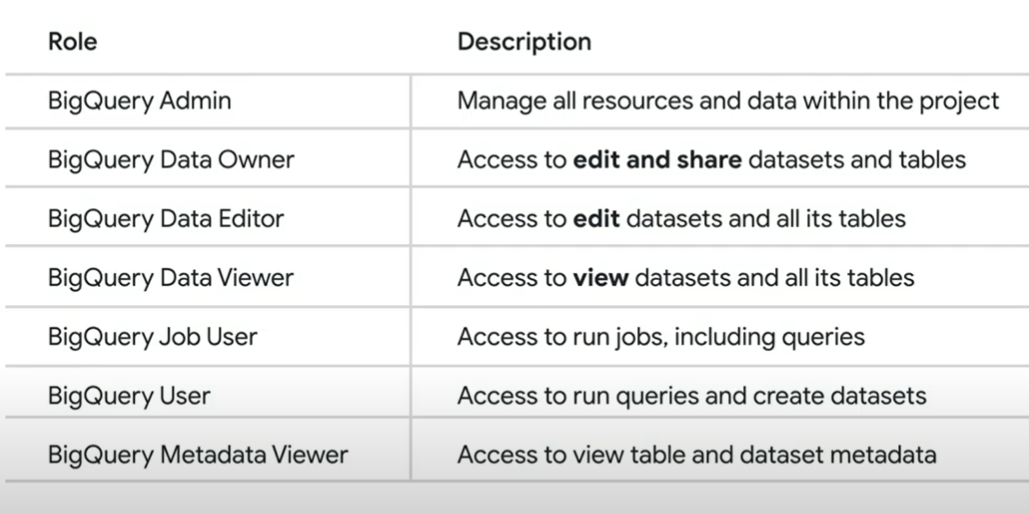
You cannot change an existing table into a partitioned table. You must create a partitioned table from scratch. Then you can either stream data into it every day and the data will automatically be put in the right partition, or you can load data into a specific partition by using “$YYYYMMDD” at the end of the table name.  
Reference: <https://cloud.google.com/bigquery/docs/partitioned-tables>

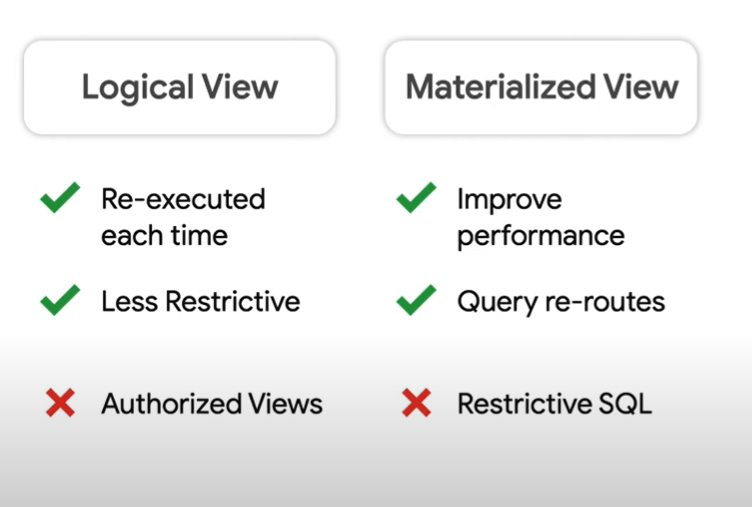
**Optimize query performance using:**

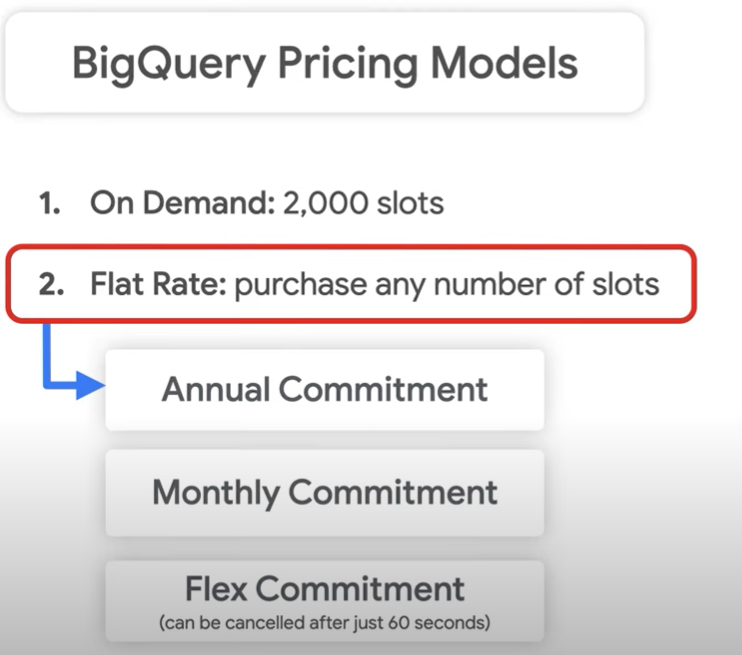
Partitioned tables: Prune large tables based on time or integer ranges.

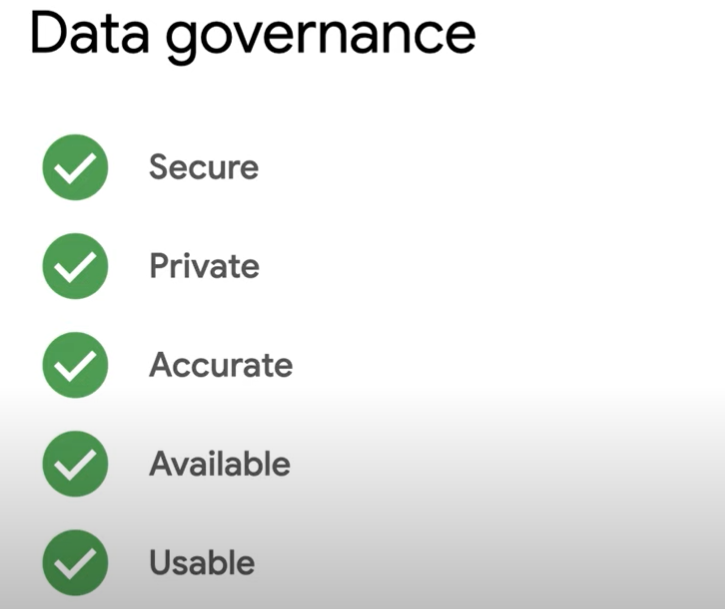
Materialized views: Define cached views to optimize queries or provide persistent results.

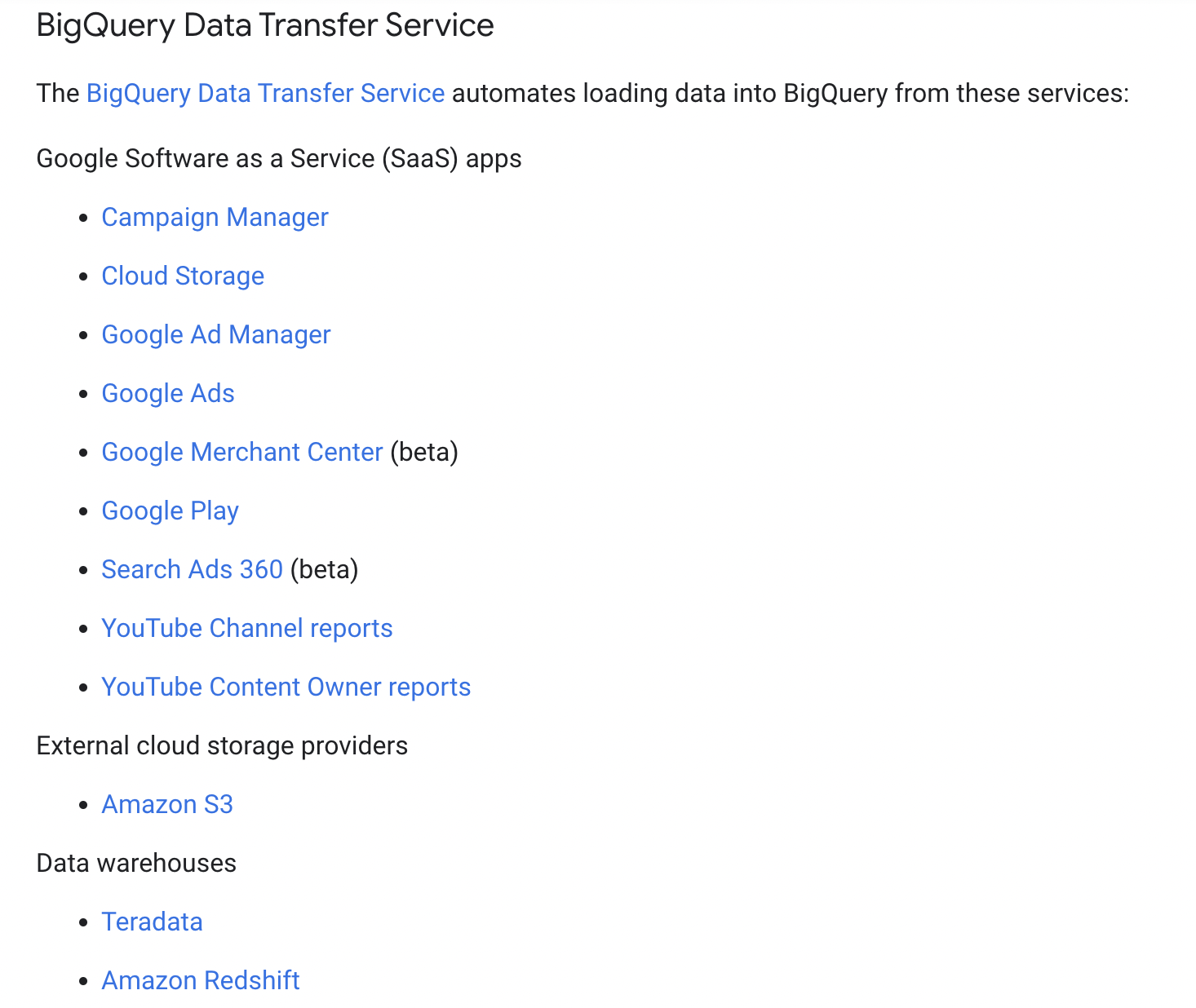
BI Engine: BigQuery's fast, in-memory analysis service.

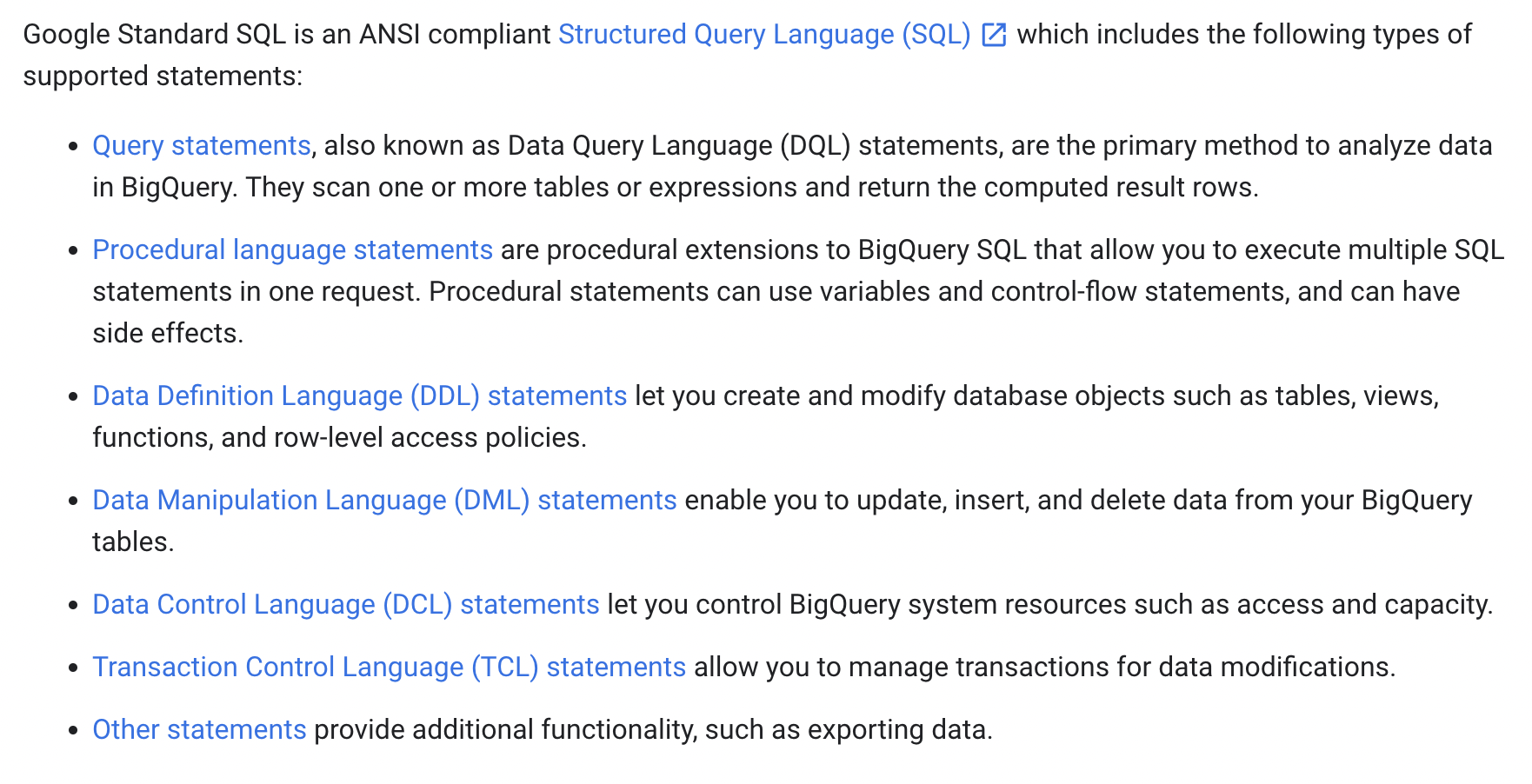


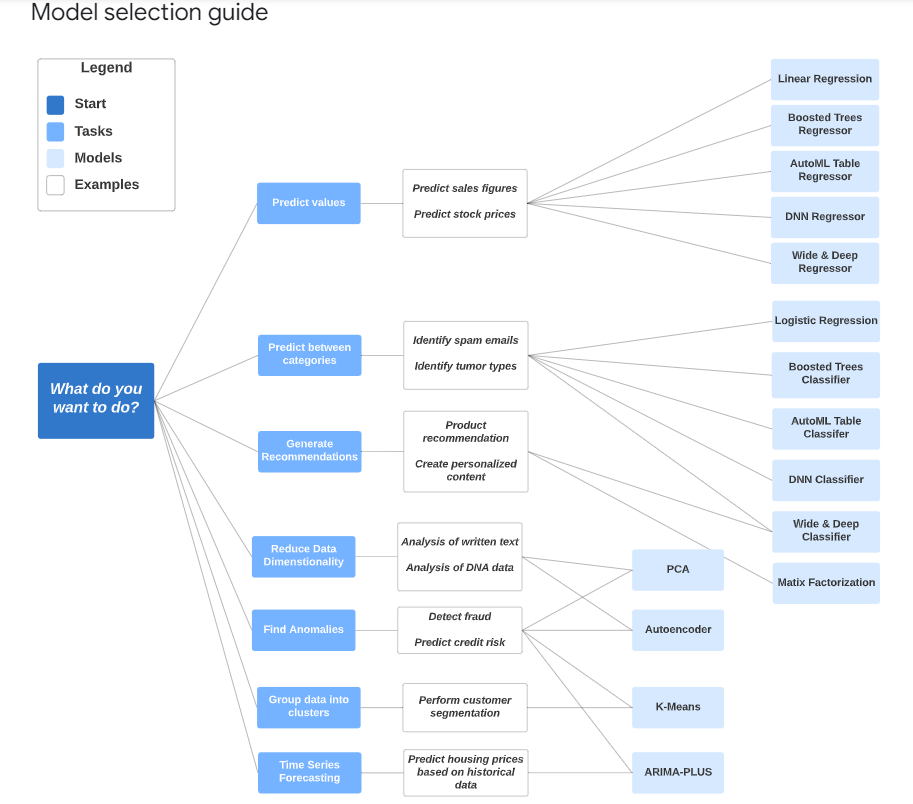












<https://cloud.google.com/bigquery/docs/loading-data>

Normalise and Denormalise

To maintain relationships while denormalizing your data, you can use nested and repeated fields instead of completely flattening your data.

# **Strategies for optimizing your BigQuery queries:**

Necessary columns only

Prune data with partitions and clusters

Aggregations in BigQuery

Late aggregation

Nest repeated data

Shuffle Joins in BigQuery

Broadcast Joins in BigQuery

JOIN order

Filter before joins

Cluster on join keys

WHERE clause order matters

Which Google Cloud Platform service is an alternative to Hadoop with Hive?

Apache Hive is a data warehouse software project built on top of Apache Hadoop for providing data summarization, query, and analysis.  
Google BigQuery is an enterprise data warehouse.  
Reference: <https://en.wikipedia.org/wiki/Apache_Hive>

Schema Auto-detection: Schema auto-detection is available when you load data into BigQuery, and when you query an external data source. When auto-detection is enabled, BigQuery starts the inference process by selecting the file in the data source and scanning up to 100 rows of data to use as a representative sample. BigQuery then examines each field and attempts to assign a data type to that field based on the values in the sample. BigQuery makes the best-effort attempt to automatically infer the schema for CSV and JSON files.

BigQuery allows partial modification on an existing table’s schema definition. The following actions are allowed:  
Adding columns to a schema definition.  
Relaxing a column’s mode from REQUIRED to NULLABLE.

In BigQuery, the following schema modifications are unsupported and require manual workarounds:  
• Changing a column’s name.  
• Changing a column’s data type.  
• Changing the column’s mode (aside from relaxing REQUIRED columns to NULLABLE).

• Deleting a column.

[https://cloud.google.com/bigquery/docs/managing-table- schemas](https://cloud.google.com/bigquery/docs/managing-table-%C2%A0schemas)

Native tables in BigQuery are tables that import the full data inside Google BigQuery as you do in any other common database system. In contrast, external tables are tables that do not store the data in Google BigQuery, instead, they reference the data from an external source, such as a data lake. The advantages of creating external tables are that they are fast to create so you skip the part of importing data and no additional monthly billing storage costs are accrued to your account since you only get charged for the data that is stored in the data lake, which is comparatively cheaper than storing it in BigQuery.

BigQuery doesn’t provide high throughput and low latency **competent to Bigtable**. Moreover, you are **unable to increase BigQuery’s performance**, as opposed to Bigtable which you can add more nodes for linear performance return.

If you have multiple BigQuery projects and users, you can manage costs by requesting a custom quota that specifies a limit on the amount of query data processed per day.  
Creating a custom quota on query data allows you to control costs at the project-level or at the user- level.  
• Project-level custom quotas limit the aggregate usage of all users in that project.  
• User-level custom quotas are separately applied to each user or service account within a project.

An external data source (also known as a federated data source) is a data source that allows you to query directly even though the data is not stored in BigQuery. Instead of loading or streaming the data, you create a table that references the external data source.

Querying an external data source using a temporary table is useful for one-time, ad-hoc queries over external data, or for extract, transform, and load (ETL) processes.  
In summary, using external tables in BigQuery is useful for such cases:  
- Perform ETL operations on data.  
- Frequently changed data.  
- Data is being ingested periodically.

BigQuery supports UTF-8 encoding for both nested (repeated) and flat data. BigQuery supports ISO- 8859-1 encoding for flat data only for CSV files.  
By default, the BigQuery service expects all source data to be UTF-8 encoded. Optionally, if you have CSV files with data encoded in ISO-8859-1 format, you should explicitly specify the encoding when you import your data so that BigQuery can properly convert your data to UTF-8 during the import process.

When you apply a BigQueryIO.Write transform in batch mode to write to a single table, Dataflow invokes a BigQuery load job. When you apply a BigQueryIO.Write transform in streaming mode or in batch mode using a function to specify the destination table, Dataflow uses BigQuery’s streaming inserts

**Denormalization** increases query speed for tables with billions of rows because BigQuery’s performance degrades when doing JOINs on large tables, but with a denormalized data structure, you don’t have to use JOINs, since all of the data has been combined into one table. Denormalization also makes queries simpler because you do not have to use JOIN clauses.  
Denormalization increases the amount of data processed and the amount of storage required because it creates redundant data.  
Reference: <https://cloud.google.com/solutions/bigquery-data-warehouse#denormalizing_data>

The conventional method of denormalizing data involves simply writing a fact, along with all its dimensions, into a flat table structure. For example, if you are dealing with sales transactions, you would write each individual fact to a record, along with the accompanying dimensions such as order and customer information.  
The other method for denormalizing data takes advantage of BigQuerys native support for nested and repeated structures in JSON or Avro input data. Expressing records using nested and repeated structures can provide a more natural representation of the underlying data. In the case of the sales order, the outer part of a  
JSON structure would contain the order and customer information, and the inner part of the structure would contain the individual line items of the order, which would be represented as nested, repeated elements.  
Reference: <https://cloud.google.com/solutions/bigquery-data-warehouse#denormalizing_data>

**Exporting**:

Data can be exported in CSV, JSON, or Avro format. If you are exporting nested or repeated data, then CSV format is not supported.  
Reference: <https://cloud.google.com/bigquery/docs/exporting-data>

**Caching**:

When query results are retrieved from a cached results table, you are not charged for the query.  
BigQuery caches query results for 24 hours, not 48 hours.  
Query results are not cached if you specify a destination table.  
A query’s results are always cached except under certain conditions, such as if you specify a destination table.  
Reference: <https://cloud.google.com/bigquery/querying-data#query-caching>

**Ingesting data in to BQ:**

You can load data into BigQuery from a file upload, Google Cloud Storage, Google Drive, or Google Cloud Bigtable. It is not possible to load data into BigQuery directly from Google Cloud SQL. One way to get data from Cloud SQL to BigQuery would be to export data from Cloud SQL to Cloud Storage and then load it from there.  
Reference: <https://cloud.google.com/bigquery/loading-data>

You can load data with nested and repeated fields using the Web UI.  
You cannot use the Web UI to:  
– Upload a file greater than 10 MB in size  
– Upload multiple files at the same time  
– Upload a file in SQL format  
All three of the above operations can be performed using the “bq” command.  
Reference: <https://cloud.google.com/bigquery/loading-data>

**Standard SQL vs Legacy SQL**:

You do not set a query language for each dataset. It is set each time you run a query and the default query language is Legacy SQL.  
Standard SQL has been the preferred query language since BigQuery 2.0 was released.  
In legacy SQL, to query a table with a project-qualified name, you use a colon, :, as a separator. In standard SQL, you use a period, ., instead.  
Due to the differences in syntax between the two query languages (such as with project-qualified table names), if you write a query in Legacy SQL, it might generate an error if you try to run it with Standard SQL.  
Reference:  
<https://cloud.google.com/bigquery/docs/reference/standard-sql/migrating-from-legacy-sql>

For BigQuery roles, the lowest permission available is the dataset level. You CANNOT set permissions on the table level. To restrict access to a table, you may use authorized views. An authorized view allows you to share query results with particular users and groups without giving them access to the underlying tables. You can also use the view’s SQL query to restrict the columns (fields) the users are able to query. When you create the view, it must be created in a dataset separate from the source data, queried by the view. Because you can assign access controls only at the dataset level, if the view is created in the same dataset as the source data, your users would have access to both the view and the data.

BigQuery UI does not support exporting stats to a file. There is no “slot utilization metrics” feature in BigQuery UI.

to support occasionally (schema) changing JSON files and aggregate ANSI SQL queries: you need to use BigQuery, and it is quickest to use ‘Automatically detect’ for schema changes.

Schema auto-detection is available when you load data into BigQuery, and when you query an external data source.  
When auto-detection is enabled, BigQuery starts the inference process by selecting a random file in the data source and scanning up to 100 rows of data to use as a representative sample. BigQuery then examines each field and attempts to assign a data type to that field based on the values in the sample.

BigQuery can export Avro data natively to Cloud Storage.

After you’ve loaded your data into BigQuery, you can export the data in several formats. BigQuery can export up to 1 GB of data to a single file. If you are exporting more than 1 GB of data, you must export your data to multiple files. When you export your data to multiple files, the size of the files will vary.  
You cannot export data to a local file or to Google Drive, but you can save query results to a local file. The only supported export location is Google Cloud Storage.  
For Export format, choose the format for your exported data: CSV, JSON (Newline Delimited), or Avro.

<https://cloud.google.com/bigquery/streaming-data-into-bigquery#manually_removing_duplicates>

To remove duplicates, perform the following query. You should specify a destination table, allow large results, and disable result flattening.  
#standardSQL SELECT \* EXCEPT(row\_number) FROM ( SELECT \*, ROW\_NUMBER() OVER (PARTITION BY ID\_COLUMN) row\_number FROM `TABLE\_NAME`) WHERE row\_number = 1

IAM cannot be applied on columns or tables, but only on datasets and views. best way to limit and expose number of columns and access is to create a View. With BigQuery, the access can only be controlled on Datasets and Views, but not on tables.

column type cannot be changed and the column needs to casting loaded into a new table using either SQL Query or import/export.  
Refer GCP documentation – BigQuery Changing Schema:- <https://cloud.google.com/bigquery/docs/manually-changing-schemas#changing_a_columns_data_type>

SELECT with partition would limit the data for querying.  
Refer GCP documentation – BigQuery Cost Best **Practices:-** [**https://cloud.google.com/bigquery/docs/best-practices-costs**](https://cloud.google.com/bigquery/docs/best-practices-costs)

dataset location cannot be changed once created. The dataset needs to be copied using Cloud Storage.  
Refer GCP documentation – BigQuery Exporting Data:- <https://cloud.google.com/bigquery/docs/exporting-data>

You cannot change the location of a dataset after it is created. Also, you cannot move a dataset from one location to another. If you need to move a dataset from one location to another, follow this process:  
1. Export the data from your BigQuery tables to a regional or multi-region Cloud Storage bucket in the same location as your dataset. For example, if your dataset is in the EU multi-region location, export your data into a regional or multi-region bucket in the EU.There are no charges for exporting data from BigQuery, but you do incur charges for storing the exported data in Cloud Storage. BigQuery exports are subject to the limits on export jobs.  
2. Copy or move the data from your Cloud Storage bucket to a regional or multi-region bucket in the new location. For example, if you are moving your data from the US multi-region location to the Tokyo regional location, you would transfer the data to a regional bucket in Tokyo. Note that transferring data between regions incurs network egress charges in Cloud Storage.  
3. After you transfer the data to a Cloud Storage bucket in the new location, create a new BigQuery dataset (in the new location). Then, load your data from the Cloud Storage bucket into BigQuery.You are not charged for loading the data into BigQuery, but you will incur charges for storing the data in Cloud Storage until you delete the data or the bucket. You are also charged for storing the data in BigQuery after it is loaded. Loading data into BigQuery is subject to the limits on load jobs.

BigQuery charges for Storage, Queries and Streaming inserts. Loading and Exporting of data are free operations and not charged by BigQuery.  
Refer GCP documentation – **BigQuery Pricing:-** [**https://cloud.google.com/bigquery/pricing**](https://cloud.google.com/bigquery/pricing)

BigQuery is a petabyte-scale analytics data warehouse that you can use to run SQL queries over vast amounts of data in near realtime.  
Giving a view access to a dataset is also known as creating an authorized view in BigQuery. An authorized view allows you to share query results with particular users and groups without giving them access to the underlying tables. You can also use the view’s SQL query to restrict the columns (fields) the users are able to query.  
When you create the view, it must be created in a dataset separate from the source data queried by the view. Because you can assign access controls only at the dataset level, if the view is created in the same dataset as the source data, your data analysts would have access to both the view and the data.

BigQuery Authorized View:- https://cloud.google.com/bigquery/docs/share-access-views

Bigquery and Dataflow: the best option is to create a new table with the updated columns. Dataflow provides a serverless NoOps option to convert data.

Best practice: If possible, materialize your query results in stages.  
If you create a large, multi-stage query, each time you run it, BigQuery reads all the data that is required by the query. You are billed for all the data that is read each time the query is run.  
Instead, break your query into stages where each stage materializes the query results by writing them to a destination table. Querying the smaller destination table reduces the amount of data that is read and lowers costs. The cost of storing the materialized results is much less than the cost of processing large amounts of data.

Data Analysts have OWNER roles to the projects, the logs need to be exported to a separate project which only the Auditor team has access to. Also, as there are multiple projects aggregated export sink can be used to export data access logs from all projects. BigQuery Auditing and Aggregated Exports:- <https://cloud.google.com/logging/docs/export/aggregated_exports>You can create an aggregated export sink that can export log entries from all the projects, folders, and billing accounts of an organization. As an example, you might use this feature to export audit log entries from an organization’s projects to a central location.

export needs to be in separate project. You need to use aggregated sink instead of project sink, as it would capture logs from all projects.

performance issue is because the data is stored in a non-optimal format in an external storage medium. BigQuery External Data Sources:- <https://cloud.google.com/bigquery/external-data-sources>

Query performance for external data sources may not be as high as querying data in a native BigQuery table. If query speed is a priority, load the data into BigQuery instead of setting up an external data source. The performance of a query that includes an external data source depends on the external storage type. For example, querying data stored in Cloud Storage is faster than querying data stored in Google Drive. In general, query performance for external data sources should be equivalent to reading the data directly from the external storage.

BigQuery Nested Query:- <https://cloud.google.com/bigquery/docs/reference/standard-sql/migrating-from-legacy-sql>#standardSQL SELECT page.title FROM `bigquery-public-data.samples.github\_nested`, UNNEST(payload.pages) AS page WHERE page.page\_name IN (‘db\_jobskill’, ‘Profession’);

BigQuery supports both batch & streaming data. However, due to the mentioned budget restrictions, the solution would choose the cheaper approach, which is batching data to BigQuery. Batching data to BigQuery is free of charge. Streaming data, on the other hand, is charged by size.

<https://cloud.google.com/bigquery/streaming-data-into-bigquery><https://cloud.google.com/bigquery/batch><https://cloud.google.com/bigquery/pricing>

BigQuery writes all query results to a table. The table is either explicitly identified by the user (a destination table), or it is a temporary, cached results table. Temporary, cached results tables are maintained per-user, per-project. There are no storage costs for temporary tables, but if you write query results to a permanent table, you are charged for storing the data.  
When you run a query, a temporary, cached results table is created in a special dataset referred to as an “anonymous dataset”. Unlike regular datasets that inherit permissions from the IAM resource hierarchy model (project and organization permissions), access to anonymous datasets is restricted to the dataset owner. The owner of an anonymous dataset is the user who ran the query that produced the cached result.  
When an anonymous dataset is created, the user that runs the query job is explicitly given bigquery.dataOwner access to the anonymous dataset. bigquery.dataOwner access gives only the user who ran the query job full control over the dataset. This includes full control over the cached results tables in the anonymous dataset. If you intend to share query results, do not use the cached results stored in an anonymous dataset. Instead, write the results to a named destination table.  
Though the user that runs the query has full access to the dataset and the cached results table, using them as inputs for dependent jobs is strongly discouraged.  
The names of anonymous datasets begin with an underscore. This hides them from the datasets list in the GCP Console and the classic BigQuery web UI. You can list anonymous datasets and audit anonymous dataset access controls by using the CLI or the API.

<https://cloud.google.com/bigquery/docs/cached-results>

In BigQuery, cached results are not supported for queries against multiple tables using a wildcard even if the “Use Cached Results” option is checked. If you run the same wildcard query multiple times, you are billed for each query.  
If the query uses non-deterministic functions; for example, date and time functions such as CURRENT\_TIMESTAMP() and NOW(), and other functions such as CURRENT\_USER() return different values depending on when a query is executed  
For the complete list of query cases not cached in BigQuery, check “BigQuery – Using Cached Query Results” below.

Reference(s):  
<https://cloud.google.com/bigquery/docs/cached-results><https://cloud.google.com/bigquery/docs/querying-wildcard-tables><https://cloud.google.com/bigquery/docs/cached-results>

Currently, cached results are not supported for queries against multiple tables using a wildcard even if the “Use Cached Results” option is checked. If you run the same wildcard query multiple times, you are billed for each query.  
If the query uses non-deterministic functions; for example, date and time functions such as CURRENT\_TIMESTAMP() and NOW(), and other functions such as CURRENT\_USER() return different values depending on when a query is executed

Resource(s):  
<https://cloud.google.com/bigquery/docs/querying-wildcard-tables><https://cloud.google.com/bigquery/docs/cached-results>

If you have multiple BigQuery projects and users, you can manage costs by requesting a custom quote that specifies a limit on the amount of query data processed per day.  
Creating a custom quota on query data allows you to control costs at the project-level or at the user- level.  
Project-level custom quotas limit the aggregate usage of all users in that project.  
User-level custom quotas are separately applied to each user or service account within a project.

Setting project-level quota is not the best approach for this scenario because this will not set user limit quotas and when a project reaches the limit set it will disallow all users from running queries. Note that, as stated, all datasets reside in a single cloud project.  
Flat-rate can be a possible approach. However, BigQuery does not provide flexible flat-rate pricing and the cheapest is $10,000 for 500 slots, which may not be a desirable option for small-medium businesses.

[https://cloud.google.com/bigquery/docs/custom- quotas#controlling\_query\_costs\_using\_bigquery\_custom\_quotas](https://cloud.google.com/bigquery/docs/custom-%C2%A0quotas#controlling_query_costs_using_bigquery_custom_quotas)

<https://cloud.google.com/bigquery/pricing#monthly-flat-rate>

Schema auto-detection: Schema auto-detection is available when you load data into BigQuery, and when you query an external data source.  
When auto-detection is enabled, BigQuery starts the inference process by selecting the file in the data source and scanning up to 100 rows of data to use as a representative sample. BigQuery then examines each field and attempts to assign a data type to that field based on the values in the sample. BigQuery makes a best-effort attempt to automatically infer the schema for CSV and JSON files.

<https://cloud.google.com/bigquery/docs/schema-detect>

Materialized Views: In BigQuery, materialized views are precomputed views that periodically cache the results of a query for increased performance and efficiency. BigQuery leverages precomputed results from materialized views and whenever possible reads only delta changes from the base tables to compute up-to-date results.

Materialized views can be queried directly or can be used by the BigQuery optimizer to process queries to the base tables.

BigQuery Materialized Views can optimize queries with high computation cost and small dataset results. Processes that benefit from materialized views include online analytical processing (OLAP) operations that require significant processing with predictable and repeated queries like those in from extract, transform, load (ETL) processes or business intelligence (BI) pipelines.

Materialized views can improve query performance if you frequently require the following:

* Pre-aggregate data. Aggregation of streaming data.
* Pre-filter data. Run queries that only read a particular subset of the table.
* Pre-join data. Query joins, especially between large and small tables.
* Recluster data. Run queries that would benefit from a clustering scheme that differs from the base tables.

Limitations:

1. Manipulating materialized view data directly is not supported. This includes the following actions:

* Copy a materialized view, either as a source or destination of a copy job.
* Export a materialized view.
* Load data into a materialized view
* Write a query result into a materialized view
* Run DML statements over a materialized view.

1. A materialized view must reside in the same organization as the base tables, or in the same project if the project does not belong to an organization.
2. Each base table can be referenced by up to 20 materialized views from the same dataset, up to 100 materialized views from the same project, and up to 500 materialized views from the whole organization.
3. Only materialized views from the same dataset are considered for automatic query rewrite (or smart tuning).
4. Materialized views use a restricted SQL syntax and a limited set of aggregation functions. For more information, see Supported materialized views.
5. Materialized views cannot be nested on other materialized views.
6. Materialized views cannot query external tables.
7. Only the standard SQL dialect is supported for materialized views.
8. If you delete a base table without first deleting the materialized view, queries over the materialized view fail, as do refreshes. If you decide to recreate the base table, you must also recreate the materialized view.

BigQuery supports UTF-8 encoding for both nested or repeated and flat data. BigQuery supports ISO- 8859-1 encoding for flat data only for CSV files.  
By default, the BigQuery service expects all source data to be UTF-8 encoded. Optionally, if you have CSV files with data encoded in ISO-8859-1 format, you should explicitly specify the encoding when you import your data so that BigQuery can properly convert your data to UTF-8 during the import process.

[https://cloud.google.com/blog/products/gcp/handling-invalid- inputs-in-dataflow](https://cloud.google.com/blog/products/gcp/handling-invalid-%C2%A0inputs-in-dataflow)

You receive a daily comma-separated (CSV) file which should be imported to BigQuery. You need to scan the file in case of incomplete or improperly aligned column values which will cause importing to BigQuery fail. What should you do to handle invalid inputs?

----A better way to solve this problem would be to have a dead letter file where all of the failing inputs are written for later analysis and reprocessing. We can use a side output in Dataflow to accomplish this goal.

Stack Driver doesn’t have native support for detecting BigQuery import failures. Also, this does not help fix a corrupted file.

*materialized views periodically cache the results of a query for increased performance. Materialized views are suited to small datasets that are frequently queried. When underlying table data changes, the materialized view invalidates the affected portions and re-reads them.*

*BigQuery ML requires a lot of data to build an accurate model, and you don't have much data.*