BigQuery - Michaël Bettan

Definition: Fully managed, Serverless (no-ops), powers fast SQL queries. Essential for analytics, cloud-based data warehouse

Availability: Region (no cross-region replication with multi-region)

SLA: 99.99%

Use Cases:
- **Large-scale data warehousing:** Store and analyze massive datasets for historical trends and insights.
- **Interactive data exploration**: Easily explore and analyze Big Data using SQL queries.
- **Fast, ad-hoc analytics**: Get quick answers to complex questions with petabyte-scale data.
- **Powerful BI reporting**: Build interactive dashboards and reports for data-driven decision making.

Billing: Load, copy, export, re-cluster, delete/metadata operations is Free
1. **Storage** (amount of data stored): **Active storage** (<= 90 days or data modified) or **Long-term storage** (> 90 days ; automatically)
2. **Querying**: **Pay-per-query (aka On-Demand)** amount of data scanned with 2000 fixed slots capacity or **Pay-per-slot_hour (aka Editions)** amount of slot hours with different features set in Standard, Enterprise, Ent Plus. with optionally slot commitments
3. **Ingestion**: **Streaming Inserts** and **Storage Write API** ; amount of data processed

Architecture

- *Designed as a **cloud-native EDW** (first party service)*
- ***Serverless**: automated scaling & zero infra management*
- ***Decoupled compute and storage**: scale independently*
- ***OLAP** (not OLTP): optimized for analytical queries, not frequent updates.*
- *Supports **ANSI SQL Queries***
- ***Multi-region location**: no cross-region replication nor regional redundancy ⟶ no increase in dataset availability in the event of a regional outage. Data is stored in a single region within the geographic location.*
- ***Columnar-Based Storage**: stores data column by column, with all the values for each column stored together.*
  - ***Faster Queries**: quickly scan through the relevant columns without reading unnecessary data.*
  - ***Better Compression**: values in a single column have similar data types and patterns (less disk space)*
- *Slots for predictable costs ⟶ High, consistent query volume*
- *Pay-per-query ⟶ Infrequent queries, unpredictable usage*
- ***Autoscaler** dynamically adjusts your reservation allocation (# of slots) based on workloads: minimum guaranteed with **baseline**, scales up for peak loads to the **maximum**.*
- ***Application Development:** building apps can be challenging due to query latency (multi-seconds) and potential cost implications for frequent and high-volume queries.*
- ***Query Queue:** previous limit of 100 concurrent queries has been replaced with dynamic concurrency queuing.*
- ***VPC Service Controls**: service perimeter preventing data exfiltration and unauthorized access and movement.*

Data Model

- ***Project**: logical grouping of datasets for billing purposes.*
- ***Datasets**: top-level container for **tables** and **views**.*
- ***Authorized dataset** authorizes all views within a dataset to access data in another (streamlining view authorization)*
  - *Protect Sensitive data*
  - *Simplify access control management*
- ***Table**: individual records into rows. Each is composed of columns with an enforced data type. Native or External table*
- ***View** is a virtual table defined by a SQL query.*

***Authorized Views** and **Materialized Views** let you share query results without giving them access to the underlying source data:*
- ***Authorized Views** is a Virtual table defined by a SQL query, but the definition cannot be seen by users / groups*
  - *Protect sensitive data and fine-grained access control mechanisms.*
  - *Created **within the same dataset** as the source*
- ***Materialized Views**: precomputed views that periodically cache the results of a query for increased performance and efficiency (zero maintenance, fresh data, smart tuning)*
- ***Job**: asynchronous action executed on your behalf (query, extract, load, copy operations)*
- ***Data manipulation language (DML)** to update, insert, and delete data from the tables*
  - ***MERGE** ⟶ INSERT, UPDATE, and DELETE operations in a single statement based on a join condition.*
  - *Limits on the number of DML statements you can execute per day and per project and query size*
- ***Data definition language (DDL)** create, alter, delete resources: tables, table clones, table snapshots, views, user-defined functions (UDFs), and row-level access policies.*
- ***Schema Evolution** supports schema changes without requiring table re-creation.*

Identity and Access Management (IAM)

***Security identities** = users, groups, and service accounts*
Granting an IAM role to an entity at the different levels:
- *Organization, Folder, Project, Dataset, Table, View*

Roles can't be applied to Routines & Models resources typed
***Tag** (key-value pair) to conditionally apply IAM to tables & datasets.*
- ***BigQuery Admin**: Full access to all resources.*
- ***BigQuery Data Owner**: Can manage data and metadata, including creating and deleting datasets and tables.*
- ***BigQuery Data Editor**: Can query data, create and modify tables, and load data.*
- ***BigQuery Data Viewer**: Can only query data.*
- ***BigQuery User**: Can run jobs, including queries.*
- ***BigQuery Job User**: Can only run jobs, but cannot access data directly ⟶ used for automated processes and apps*
  - *Automatic: best effort ~ 5 min of base table change*
- ***BigQuery Metadata Viewer:** Allows users to view metadata*

about resources, such as datasets, tables, and views.

Partitioning

- ***Purpose**: Divides large tables into smaller, more manageable **segments** called partitions:*
  - ***Query performance:** faster by scanning only relevant partitions for data.*
  - ***Cost reduction**: Charges are based only on the processed data within queried partitions.*
- *Each partitioned table can have up to 10,000 partitions (~27 years of data). The daily limit is 30,000 partition modifications per table, per day, per project.*
  The rate limit: 50 partition updates every 10 seconds.
- ***Ingestion Time** is based on the ingestion date. Each date-based partitioned table has a pseudo-column _PARTITIONTIME (date-based timestamp)*
- ***Time-Unit Column** is based on date or timestamp column. Each partition = one time unit = hour, day, month.*
- ***Integer range** ranges of values in a specific INTEGER column (start, end range, interval)*
- ***Daily** (default), **hourly**, **monthly**, or **yearly** partitioning*
- *Requires Partition Filter at table level (+ WHERE clause)*
- ***Excessive partitioning** (thousands of partitions) can negatively impact performance.*
- ***TABLE_DATE_RANGE**: specify a date range instead of manually listing individual partitions (simplify querying)*
- ***Partition decorators** reference a partition in a table. You can use them to write data to a specific partition. **table_name$partition_id** where the format of the partition_id segment depends on the type of partitioning*

Sharding

- ***Sharding** is a similar concept (less preferred) with a single table per day. UNION in query to scan multiple tables.*
- ***Partitioning vs. Sharding**: Partitioning within a single table is easier to manage and leverages query optimizations more effectively than creating multiple sharded tables.*

Clustering

- ***Purpose**: Sort data within partitions based on one or more columns (clustering keys). Further improves query performance, especially for queries with filtering and aggregation, by co-locating related data.*
- ***Automatic reclustering** to maintain efficiency (free)*
- ***Hierarchical Clustering**: cluster a table by multiple columns to create a hierarchical organization.*
  - ***Example**: Cluster by country, then by state/region, and then by city.*
- *Partitioning and clustering are powerful when used together. For example, partition by date and cluster by a frequently filtered column.*

Data Ingestion

- *Batch vs. Real-time factors: data volume, frequency, and latency requirements*
- ***Batch Loading** for large datasets (free, shared resources)*
- ***Streaming Loading** for real-time use cases*
  - ***Streaming Inserts** (disable insertId for high volume ⟶ best effort deduplication)*
  - ***BQ Storage Write API:** high throughput*
- ***Streaming Buffer:** data is first held in a temporary staging area to ensure data durability by ack successful writes before the data is persisted in storage.*
- ***Streaming buffer**: available for querying immediately after ack a tabledata.insertAll request. Up to 90 min for table copy.*
- ***File formats (fastest to slowest):** Avro, Parquet, ORC, uncompressed CSV & JSON (Newline), compressed (gzip)*
  - ***Avro**: row-based, snappy/deflate (faster write)*
  - ***Parquet**: columnar, snappy/gzip/LZO (faster read)*
- ***Encoding:** UTF-8 (default) for accurate data representation*
- ***Schema**: Auto-detection for CSV/JSON, or define manually*
- ***Data sources**: GCS (Firestore export), local file, Drive, BigTable, BigQuery Data Transfer Service (DTS)*
- ***Dataflow** and **Pub/Sub** can stream/write data directly*
- ***Partitioning**: Supports Hive partitioning for data in GCS*
- ***Schema Flexibility**: add columns without rewriting the entire table.*
- ***Nested & Repeated Fields** for complex data structures effectively. How to query these structures using dot notation (e.g., field.attr) and UNNEST.*
  - *Nested data (columns within columns)*
- ***Data Transformation during Ingestion**: Dataflow or Dataproc or Create a table from a query result (CTAS)*

Querying Data

- ***Language:** GoogleSQL (ANSI SQL) or Legacy SQL*
- ***Query jobs**: asynchronous actions: load, export, query, copy*
- ***Results:** permanent or temporary tables*
- ***Query Types**:*
  - ***Interactive** (default, immediate execution, counts toward daily/concurrent usage)*
  - ***Batch** (queued, starts when idle resources are available, only counts for daily and switches to interactive if idle for 24 hours)*
- ***Federated Queries:** Allows querying data from external data sources (e.g., Cloud Spanner, Cloud SQL, etc.)*
  - ***Limitations**: no consistency guarantees, potential performance impact, no caching.*
  - ***Use Cases**: ETL operations. Query small and frequently changing data without the need to reload it. Access data that is being ingested periodically.*

Analytics Hub

***Built-in data exchange platform** for secure & scalable **data sharing** across organizations. Facilitates **collaboration** and **data discovery**.*

- ***Publish and subscribe model:** Share datasets without duplicating data. Publishers control access and manage subscriptions.*
- ***Shared datasets and exchanges:** Organize data for sharing through curated datasets and access control via exchanges (private or public)*
- ***Enhanced data security:** Granular access controls and built-in security measures protect sensitive data.*
- ***Cost-effective sharing:** Publishers only pay for storage, subscribers only pay for querying shared data.*
- ***Discovery and access to datasets:** Find and access internal and external data assets, inc/ datasets from Google.*

***Analytics Hub** supports a **Data Mesh** by enabling **secure, compliant, and efficient data exchanges** within and between* *organizations, reducing silos and empowering teams to leverage data more effectively.*

- ***Cross-domain Data Consumption**: Users can easily access data products across different domains without duplication.*
- ***Data Discoverability**: Data consumers can search for datasets across the organization in a standardized way, improving data availability.*
- ***Governance**: By using Analytics Hub in conjunction with Dataplex, teams can maintain high data quality, privacy, and compliance through federated governance policies.*

Data Clean Room

***Data Clean Rooms**: analyzing sensitive data with multiple parties while ensuring privacy and security without directly sharing the underlying data itself:*

- ***Query controls**: restrictions on the types of queries that can be executed.*
- ***Output privacy mechanisms via analysis rule:***
  - ***Aggregation threshold** enforces the minimum number of distinct entities that must be present in a set of data, so that statistics on that set of data are included in the results of a query.*
  - ***Differential privacy** is an anonymization technique that limits the personal information that is revealed by an output. Differential privacy is commonly used to allow inferences and to share data while preventing someone from learning information about an entity in that dataset.*
  - ***List overlap** enforces an INNER JOIN and returns the exact rows where data intersects between two or more tables.*

Data Recovery & Backup

- ***Table clones**: create a temporary, read-only copy of an existing table (only the changed data is billed)*
  - *Creating copies of production tables that you can use for development and testing.*
  - *Creating sandboxes for users to generate their own analytics and data manipulations, without physically copying all of the production data.*
- ***Table Snapshots** preserve the contents of a base table at a particular time up to 7d (changes billed). Not ideal for backup*
- ***Time travel:** Retrieves full table data for the past 2-7 days (configurable). Quick rollbacks of recent modifications or deletions but no selective restore.*
- ***Export to GCS** in various formats (e.g., Avro, CSV) for long-term archival or disaster recovery (vs. BQ full datasets)*
- ***Fail-Safe:** retains deleted data for an additional 7 days after the time travel window via support for emergency recovery*
- ***Copy dataset:** create a complete copy of the dataset, including tables and metadata. Costly, not suited long-term*
- ***ELT pipelines**: Organize your data in separate tables for each month, export, compress, and store the data in GCS. Organizing data into monthly tables allows you to restore specific time periods if an error affects only a portion*
- ***Regional outage**: BigQuery does not offer automatic durability or availability. Data loss is possible in a catastrophic regional failure. Users need to implement their own backup and recovery strategies to define their RPO. Exporting data to Cloud Storage in a different geographic region is the recommended approach.*
- ***Accidental deletion/corruption**: time travel allows querying data from any point within the last 2-7 days, providing an RPO of 0 within that window. Snapshots can extend this, but don't replicate data.*

BigLake

Query all your **structured & unstructured data** across analytics & AI engines, with built in security, governance & management

- ***Open Formats:** Built on open formats (Parquet, ORC, Iceberg) → interoperability and avoid vendor lock-in.*
- ***Multi-Cloud Governance:** discover and manage your BigLake tables through Data Catalog.*
- ***Fine-Grained Security:** Enforce consistent security policies (row filtering, column masking) across supported query engines (BigQuery, Spark, Trino, Presto, and more), removing the need to manage file-level permissions.*
- ***Simplified Analytics:** Query data using SQL interfaces, regardless of the underlying storage or analytics engine.*
- ***Metadata caching** (names, partitions, row counts) from GCS/S3 will allow to speed up queries, especially those with many files or Hive partition filters*
- ***Moving cold (archive) data** to GCS, BigLake will query the data even in the lower-cost storage tiers → cost-effective*
- ***BigLake over Omni**: Eliminate the need for data movement and ETL pipelines by directly querying data residing in S3 and Azure Data Lake.*

Query & Cost Optimization

- ***Avoid SELECT * (full scan)**, select only necessary columns to reduce data scanned → **SELECT * EXCEPT (dim1, dim2)***
- *Sample data using **Preview Table** (free) and **Preview queries** to estimate costs --=dry run*
- ***Use max bytes billed** to limit query costs - Don't use LIMIT clause to limit costs (as it is still a full scan)*
- ***Use filters and WHERE clauses**: Filter data early in the query to limit the amount of data processed.*
- ***Join tables** with most efficient join type, Largest table, then smallest and then by decreasing size. WHERE clauses should be executed as soon as possible. Use filters on both tables*
- ***Use approximate aggregate functions** such as APPROX_COUNT_DISTINCT for acceptable accuracy results.*
- ***Use query plan visualization**: Analyze the query execution plan to identify bottlenecks and optimize query structure.*
- ***Monitor costs** using dashboards and audit logs - Partition data by date*
- *Break query results into stages*
- *Use **default table expiration** to delete unneeded data*
- *Use Streaming Inserts or Write API wisely (potentially costly)*
- *Set hard limit on bytes (members) processed per day*
- *Set a quota per user, but it applies to all users in a project*
- ***Aggregate late and rarely,** especially for queries with sub-SELECTs. Move your aggregation functions up to the*

top to avoid multiple GROUP BY executions.

- *[Nest repeated data](): create a repeated column that contains a list of all the elements*
- *[Window (analytic) functions]() eliminate full table scans and temporary tables for intermediate calculations. Avoid self-joins (anti-pattern)*
  - ***Standard Aggregations:** SUM, AVG, COUNT*
  - ***Navigation Functions:** Access rows relative to the current row (e.g., LAG, LEAD, FIRST_VALUE, LAST_VALUE).*
  - ***Ranking & Numbering Functions:** Assign ranks or row numbers within the window (e.g., RANK, ROW_NUMBER, DENSE_RANK).*
- ***Denormalization** (combining data into fewer tables with redundant data) can speed up queries by reducing joins. Beneficial for read-heavy dashboards. Trade-offs:*
  - ***Increased storage costs** (data duplication).*
  - ***Data redundancy** (potential inconsistencies if not managed carefully).*
- *[Cached queries](): If you run the exact same query again, results from the cached table, if it exists. Temporary, cached results tables are maintained per-user, per-project (24 hours)*
- ***Partitioning** large tables by date or other criteria*
- ***Clustering** tables based on frequently used columns*
- ***Handle Data Skew:** Uneven data distribution (e.g., most rows having the same 'country') can lead to inefficient workload distribution across slots, slowing down queries:*
  - ***Pre-Aggregation**: on the skewed column to reduce the number of rows with the same value.*
  - ***SHUFFLE Clause**: can help redistribute data more evenly before joining or aggregating*

Auditing & Logging

- *[Cloud Audit Logs (via Cloud Logging)]():*
  - *Identify users: who has accessed specific tables*
  - *Analyze query execution: insights in usage patterns.*
  - *Track changes to resources (table schemas, IAM)*
  - *Track data access and ensure security compliance*
  - *Troubleshooting and Debugging*
- ***Security Command Center** for security events and alerts*
- ***Cloud Logging & Monitoring** to monitor BigQuery usage, query performance, and identify potential issues.*

SQL Function Reference

Working with arrays and JSON data: Useful for handling semi-structured data.
Geospatial analysis: Enables location-based queries.
Security functions for data encryption and access control.
Approximate and HyperLogLog++ functions for advanced analytics and efficient counting of distinct values.

Monitoring

- *[Slot consumptions metrics](): # of slots used by a query via:*
  - ***Cloud Monitoring** (dashboards and alerts)*
  - ***INFORMATION_SCHEMA.JOBS_****
  - *Operational health dashboard (preview)*
  - ***Audit logs**: metrics on slot usage (Jobs API method)*
- ***Query metrics**: Execution time, bytes processed, slots used*
- ***Job metrics**: # of jobs started, failed, and completed*
- ***Storage metrics**: Bytes stored, active storage*
- ***Streaming metrics**: # of rows inserted, latency*
- ***Custom quota per user,** but it applies to all users in a project*
  - *Query usage per day (default: unlimited)*
  - *Query usage per day per user (default: unlimited)*

External Data Sources

- *[External Data Sources (5)](): Bigtable. Cloud Spanner, Cloud SQL, Cloud Storage, Drive*
- ***External Tables** act like a standard table. The table metadata, including the table schema, is stored in BigQuery storage, but the data itself resides in the external source (BigTable, Cloud Storage, Drive)*
- ***Federated Query** is a way to send a query statement to an external database (Cloud SQL or Cloud Spanner) and get the result back as a temporary table.*

Data Governance, Quality & Lineage

- ***Protect sensitive data** using data masking in table columns and Sensitive Data Protection (DLP) via Sensitive data profiling and On-Demand inspection.*
- ***Control data ingress and egress** using VPC Service Controls*
- ***Manage encryption**: Google-Managed, CMEK. Cloud KMS*
- ***AEAD encryption:** create keysets that contain keys for encryption and decryption to encrypt and decrypt individual values in a table, and rotate keys within a keyset.*
- *BigQuery uses **Dataplex** to **define continuous data checks**, **monitor results**, and **troubleshoot** issues with data quality.*
- ***Data lineage**: track how data moves through your systems: where it comes from, where it is passed to, and what transformations are applied to it.*

Data Lineage

- *Definition: Tracking the origin, transformations, and dependencies of data.*
- *Importance: Ensures data quality, compliance, and reproducibility.*

Jobs & Reservation

- *# of slots = unit of compute = ~ throughput (CPU, RAM, NET)*
- *[Flex Slots]() available for bursting*
- *[Reservations]() (bucket of slots) could be allocated to split any slot commitment to org, folder and project levels*
- *Idle slots are shared and available for other reservations in your organization to use.*
- *Fair scheduler to limit resource-heavy query consumption*
- *Load, export, copy by default is on free shared pool of slots*
- *3 types of Assignments: QUERY, PIPELINE, ML_EXTERNAL*
- *Large **UPDATE** statements via batching updates, time-based partitioning, clustering (worse case: increased quotas)*

Schema Design

Choose Data Types Wisely:

- *Select the most specific data type for your needs to*

optimize storage and query performance.
- *Avoid overly generic types (like STRING for everything).*

Nested and Repeated Fields: *removes the need for a separate table and costly join operation*
- ***Nested Fields (STRUCTs):*** *Group related data into logical units, improving readability and query efficiency (think of embedded tables within a table).*
- ***Repeated Fields (ARRAYs):*** *Store multiple values within a single row, reducing data redundancy.*

Schema Flexibility: *You can have nested structs within repeated fields or repeated fields within structs, providing the flexibility to model complex data relationships and track historical changes.*

UNNEST operator *for nested and repeated fields. Flattens the data for easier querying and treats each item in a repeated field as a separate row.*

Data Types

Choosing the right data type *(e.g., INT64 vs. FLOAT64, TIMESTAMP, GEOGRAPHY) for performance and cost:*

Data type	***Use case with examples***
NUMERIC BIGNUMERIC	***Financial analysis*** *for precise currency calculations, stock prices, and financial reporting*
FLOAT64	***Sensor data f****or capturing temperature readings, pressure, or other sensor values*
TIMESTAMP	***Event logging*** *for storing event timestamps with accurate time zone information*
STRUCT	***Product catalog****: Representing a product with attributes name, description, price, etc.*
GEOGRAPHY	***Location-based services*** *for storing user locations, geofencing, and spatial analysis.*
JSON	***Exchange Log events*** *Representing structured data in a flexible format*

Error Handling and Data Quality

Common Errors:
- ***Invalid Schema:*** *Data type mismatches between source data and table schema.*
- ***Query Timeouts****: Queries exceeding configured limits or resource contention.*
- ***Ingestion Issues****: Problems loading data, often due to format errors or permissions.*

Troubleshooting Tips:
- ***Query Plans****: Analyze using EXPLAIN statements to identify bottlenecks.*
- ***Cloud Logging:*** *Examine logs for error messages and performance insights.*
- ***Job Details****: job history in the console for errors and stats.*

Data Validation:
- ***CHECK Constraints****: Define rules at the table level to enforce data integrity (e.g., data type, range).*
- ***ETL Validation:*** *Incorporate data quality checks within your ETL pipelines to catch errors early.*

BI Engine

BI Engine *is an in-memory analysis service that accelerates BigQuery queries for business intelligence (BI) and data visualization tools like Looker Studio and Looker. It optimizes performance by storing data in memory, providing fast, sub-second query responses, making it ideal for interactive reporting and dashboards. BI Engine automatically integrates with BigQuery and requires minimal configuration, delivering high performance with the ability to scale seamlessly while maintaining real-time insights and reducing costs associated with long-running queries.*
- ***BI Engine SQL*** *for any BI tooling, e.g., Looker or Tableau*

Miscellaneous

- ***BigQuery Omni for multi-cloud*** *with External Tables*
- ***User-Defined-Functions (UDF)****: SQL or Javascript function that accepts input, performs actions, and returns a value as a result. Temporary or Persistent.*
- ***Stored Procedure*** *is a persistent script that you can invoke from inside a SQL statement. A procedure can take input arguments and could return values as output.*
- ***Routines*** *are a resource type: Stored procedures, User-defined functions (UDFs), inc/ remote functions, Table functions = re-use logic and handle your data in a unique way*
- ***Scripting*** *= send multiple statements to BigQuery in one request.*
- ***Geospatial Analytics (GIS)****: geography data types and geography functions to analyze and visualize geospatial data*
- ***Geospatial analysis****: Optimized for querying and analyzing geographic data. GEOGRAPHY data type and related*
- ***Cached queries:*** *All query results, interactive and batch, are cached in temporary tables for ~ 24 hours, some exceptions.*
- ***Time travel*** *at no extra cost via AS OF SYSTEM TIME, going back 7 days*
- *Real-time* ***Streaming Ingest*** *(or BQ Storage Write API)*
- ***ROW_NUMBER*** *generates a series of temporary values that are assigned to figures, and is calculated dynamically based on when the query is executed*
- ***Wildcard tables*** *to query multiple tables using concise SQL, useful when a dataset contains multiple, similarly named tables that have compatible schemas.*
- ***BigQuery ML****: Enables building and training machine learning models directly on BigQuery data.*
- ***Geography Functions*** *to perform spatial analysis.*
- ***Business intelligence tools****: Integrate seamlessly with BI tools for data visualization and reporting (Tableau, Looker)*
- ***BigQuery Tags****: assign metadata to datasets in the form of key-value pairs. Used to apply IAM policies conditionally based on specific tag values.*

Primary and Foreign Keys

While BigQuery doesn't enforce primary key and foreign key constraints in the same way as RDBMS,

Primary Key*:*

- ***Definition****: uniquely identifies each row in a table. It must contain a non-null value and cannot have duplicates.*
- ***Declaration****: define constraint during table creation*
- ***Enforcement****: doesn't prevent you from inserting duplicate or NULL values in a primary key column. However, if you declare a primary key, BigQuery can optimize query performance, especially for joins involving the primary key column.*

Foreign Key:

- ***Definition****: column or a set of columns in a table that references the primary key of another table*
- ***Declaration****: during table creation using DDL.*
- ***Enforcement****: doesn't automatically enforce referential integrity for foreign keys. This means you can insert a foreign key value that doesn't exist in the referenced table's primary key. You need to implement your own checks and validation mechanisms to maintain referential integrity (e.g., using MERGE statements, pre-insert validation queries).*

VPC Service Controls

Security feature that defines a security perimeter around resources to mitigate the risk of data exfiltration.

- ***Data Protection*** *protects sensitive data by restricting access and establishing a defined perimeter around resources.*
- ***Access Levels*** *determine which users or services can access resources ensuring only authorized users can interact with your datasets.*
- ***Service Perimeters****: encompass multiple services, including BigQuery, ensuring that data does not leave the defined boundaries.*
- ***Audit Logs****: detailed audit logs for activities within the perimeter, monitor access and detect security threats.*
- ***Deny Rules****: Allows the creation of deny rules to restrict access to resources from specified IP ranges, enhancing security against unauthorized access.*
- ***Data Exfiltration Protection****: Reduces the risk of accidental or intentional data exfiltration by enforcing policies that control how data can be accessed and shared outside the organization.*

Change Data Capture (CDC)

- *CDC tracks changes from transactional databases to BigQuery in near-real time.*
- ***Log-based CDC*** *is preferred for efficient replication of inserts, updates, and deletes.*
- ***Use staging tables*** *to capture changes before applying them to reporting tables.*
- *Apply changes in batches* ***using DML MERGE*** *to optimize performance and reduce overhead.*
- ***Stream CDC events in real tim****e for low-latency data availability.*
- ***Transactional integrity in log-based CDC*** *ensures data consistency.*
- ***BigQuery supports schema evolution****, allowing dynamic structure changes.*
- ***Minimize DML on the reporting table*** *to optimize performance.*
- ***Implement error handling*** *for recovery and consistency in the pipeline.*