Module – 1 Big Query for Data Analysis

Understanding the Roles

Data analysts, data engineers, and data scientists all work with datasets to gain insights. However, their roles are different:

Role	Primary Responsibility	Tools/Techniques
Data Analyst	Gather and interpret data to solve problems and support business decisions.	SQL, static modelling techniques, reporting, visualization
Data Engineer	Build systems for collecting, validating, and preparing data. Develop and maintain data pipelines.	Data pipelines, ETL processes
Data Scientist	Use dynamic techniques like machine learning to predict future outcomes and gain deeper insights.	Machine learning, statistical modelling

It's important to note that these roles **may not work in silos**.

For instance:

- A **data analyst** might collaborate with a **data engineer** to source and ingest data.
- Sometimes, **one person** might perform all three roles.

Common Challenges for Data Analysts

Two major barriers for data analysts:

- **Too much data**
- **Disconnected data**

Three recurring themes:

Theme	Challenges
Querying	Queries taking too long, overly complex logic
Infrastructure	Scalability and maintenance issues
Storage	Capacity concerns, pricing issues, non-centralized data

Note: Data engineers and scientists face similar challenges.

How Google Cloud Helps

Google Cloud addresses these common problems with:

- Powerful **storage** solutions
- **Elastic infrastructure** that scales automatically
- Focus on **query writing** and **data analysis** instead of resource management

Storage Evolution

Year	Storage Details	Cost
1981	10 MB hard drive	$300,000
2023	1 TB hard drive	Less than $20

While hard drives have become cheaper, handling **Big Data** still needs:

- Computing power
- Networking
- Admin and hardware maintenance
- Software licenses

Even after investing in these, finding meaningful insights becomes challenging because **traditional databases** can't handle the **scale of modern data** effectively.

Focus Areas:

Instead of spending time on:

- Managing infrastructure
- Manipulating and monitoring data

You should be focusing on:

- **Extracting valuable insights**

That's exactly why **Google** built systems that:

1. **Scale** with exploding data volumes
2. Are **fully managed** to avoid complexity
3. Let you **focus on insights**, not operations

Don't Build It Yourself — Leverage Google's Infrastructure

Google Cloud's Global Network

- Continually expanding regions and zones
- View updates: Google Cloud Locations
- Points of presence (PoPs) ensure:
    - Low-latency network performance

- Global reach

Infrastructure Advantages:

On-Premises	Google Cloud
Pay for idle CPUs	Pay for what you use
Manual scaling	Automatic elastic scaling
Co-located storage and compute	Separate storage and compute
High upfront costs	Flexible, scalable pricing

With Cloud:

- You can process 1 TB of data in 1 second using the power of 120 machines, **only when needed**.
- Elastic scaling ensures no waste or over-provisioning.

Key Feature: Decoupling Storage and Compute

Traditional systems have storage and compute tightly coupled, meaning:

- You pay for both, even if you're not using the compute power.

In Google Cloud:

- You pay **only** for **storage** or **compute** — depending on what you actually use.
- This leads to **optimized costs** and **better resource management**.

Introduction to BigQuery

BigQuery is Google Cloud's fully managed enterprise data warehouse. It helps you manage and analyze large datasets effortlessly.

Feature	Details
Architecture	Serverless and fully managed
Query Language	SQL
Performance	Query terabytes in seconds, petabytes in minutes
Scalability	Separate storage and compute for maximum flexibility

BigQuery lets analysts focus purely on **asking business questions** and **analyzing data**, instead of worrying about infrastructure or performance tuning.

Data insights using Big Query

In this section, we will highlight the **five common tasks** of any data analyst and discuss how **BigQuery** can help with these tasks.

Earlier, we briefly described the role of a **data analyst** and compared it with other roles like **data engineers** and **data scientists**. Ideally, a data analyst wants to spend more time on the last two tasks—**analysing** and **visualizing data**. We will spend considerable time discussing these areas.

However, the data analyst may also need to deal with:

- Ingesting the right data
- Cleaning and transforming the data
- Storing data for faster retrieval

Each of these tasks brings its own challenges. For example:

- **Analysis** may become harder because queries are slow.
- **Large volumes of data** or **complex transformation logic** can delay insights.

All of this can make it difficult to extract useful, actionable, and timely insights.

BigQuery supports each of these tasks, helping data analysts in a **scalable, economical, and impactful** manner.

BigQuery Features and Benefits

With BigQuery, you benefit from Google's fully managed **datacentre-backed infrastructure**. You don't need to optimize hardware or networks; your focus remains on **asking smart questions** and **finding insights**.

Key Features of BigQuery:

Feature	Description
Fully Managed	No need for hardware provisioning, cluster management, or server maintenance (NoOps).
Real-Time Analysis	Near real-time interactive analysis of massive datasets.
Replication	Data is replicated across multiple data centres for reliability and peace of mind.
Pay-for-Use	You are billed only for the storage and compute resources you actually use.
Security	Access Control Lists (ACLs), Identity and Access Management (IAM), and encryption in transit and at rest.
Auditing	Cloud Audit Logs track admin activity, data access, and system events for accountability.
Scalability	Virtually unlimited storage and highly parallel processing for faster query performance.

Flexible Data Handling	Supports real-time streaming ingestion and mixing datasets from various sources, including public datasets.
Ease of Use	Simple denormalized tables, columnar storage, no indexes/keys required, and a familiar SQL interface.
Open Standards	Easily integrate with your preferred tools and third-party applications.

How You Can Interact with BigQuery

BigQuery offers four ways to interact with the system:

1. **BigQuery Web UI**
    - Primary tool for this course.
    - Examine tables, build queries, view estimated data processed.
    - Simple mouse clicks to get started quickly.
2. **Command-Line Interface (CLI)**
    - Robust set of commands to run queries and explore features.
    - Ideal for those who prefer interactive, text-based workflows.
3. **REST API**
    - Used by programming languages like **Python** or **Java**.
    - BigQuery receives HTTP requests and returns JSON responses.
    - Web UI and CLI internally use the REST API.
4. **Third-party GUI SQL Editors**
    - Tools like **Looker, Appsmith, Retool, Cogniti, SuperQuery, PopSQL, and Zing Data** can connect to BigQuery.
    - However, detailed usage of REST API and third-party tools is beyond the scope of this course.

BigQuery acts as a two-in-one service:

- **Storage Layer**: Provides economical and efficient storage options and supports ingestion from many formats and sources.
- **Analysis Engine**: Optimized for fast analytic queries on massive datasets (terabytes in seconds, petabytes in minutes).

Storage Organization in BigQuery:

- **Tables** reside within a **dataset**.
- **Datasets** exist inside a **project**.
- You can also store other resources like **views** and **routines** inside datasets.

You can think of a dataset as a **collection of tables, views, and other resources**.

Organization of Resources in BigQuery

Google Cloud Project

- **Dataset 1**
  - Table A
  - Table B
  - View 1
  - Routine 1
- **Dataset 2**
  - Table C
  - Table D
- **Dataset 3**
  - Table E
  - View 2

Module -2 Exploring and Preparing Your Data with BigQuery

Exploring Large Datasets in BigQuery

There are three main ways to explore datasets:

1. **SQL Queries** – Use SQL in editors like the BigQuery Web UI.
2. **Data Prep/BI Tools** – Examples include Dataprep, Looker's SQL Runner, Looker Studio, and Cloud Data Fusion's Wrangler.
3. **Visualization Tools** – Utilize third-party tools for raw data visualization.

Among these, SQL remains a fundamental and essential skill for data analysts, offering speed, flexibility, and direct interaction with data.

Before writing SQL queries, good analysts:

- Understand the **type of data** they're interested in (e.g., financial, non-profit).
- Define **specific goals** (e.g., analysing revenue).
- Decide **how results should be presented** (e.g., sorted by highest revenue).
- Review **available datasets**, structure, important fields, and volume.
- Translate questions into queries by identifying needed fields, cleaning data, applying aggregations, or joining tables.

Effective exploration requires both technical SQL skills and a strategic approach to data understanding and querying.

Basic of Query

BigQuery uses SQL that aligns closely with the ANSI 2011 standard. Common SQL elements like SELECT, WHERE, JOIN, GROUP BY, and ORDER BY are supported, along with aggregation functions such as COUNT, SUM, MIN, and MAX. BigQuery also offers advanced features like arrays, structs, geography types, and BigQuery ML.

SQL Basics in BigQuery:

- A basic query uses SELECT, FROM, and optional WHERE, ORDER BY, and LIMIT clauses.
- Use WHERE to filter rows, and ORDER BY to sort results (default is ascending; use DESC for descending).
- LIMIT restricts the number of returned rows — helpful for data exploration.
- Use SELECT DISTINCT to eliminate duplicates.
- Use aliases (AS) to rename output columns, especially when using functions like ROUND for cleaner output.
- Aliases **can** be used in ORDER BY, GROUP BY, and HAVING, but **not** in WHERE.

Query Best Practices:

- Always specify the full table path using project. Dataset table for accuracy.
- Avoid SELECT * on large tables without LIMIT or WHERE, as this increases query cost.
- Use the **Schema** or **Preview** tabs for cost-free exploration.
- Comment your queries to improve readability and maintainability.
- Use the **query validator** to estimate cost before running a query.

BigQuery Cost Management:

- You get **1 TB/month of free query processing**.
- Costs are based on bytes processed, not results returned.
- Use the **Google Cloud pricing calculator** for estimates.
- Repeated queries may use **cached results** (valid for ~24 hours), which are free unless the query changes.

Other Tips:

- Use functions (e.g., ROUND, CURRENT_TIMESTAMP) to transform data.
- If using a computed field in WHERE, repeat the expression or use a subquery.
- For "top N" analysis (e.g., top 10 products), combine ORDER BY and LIMIT.

For demos and example queries, visit: GitHub: Explore Data with SQL

Working with Functions

String Manipulation Functions in BigQuery

String manipulation functions are useful for changing a string's format or applying simple transformations—such as converting text to uppercase or extracting the first five characters.

For example, the CONCAT() function allows you to concatenate one or more STRING or BYTES values into a single result. You can also use a formatting function to make data more readable, such as displaying a revenue column with commas.

Common String Functions

- CONCAT(): Combines two or more strings.
- ENDS_WITH(): Returns TRUE if the second string is a suffix of the first.
- LOWER(): Converts a string to lowercase.
- Regular Expression Matching: Returns TRUE if a value matches a regex pattern.

For more advanced use cases, check the documentation on BigQuery string and regex functions.

LIKE Operator for Simple Pattern Matching

You can use the LIKE operator within the WHERE clause for basic pattern matching. For example:

```
WHERE LOWER(product_name) LIKE '%shirt%'
```

Here, LOWER() ensures consistent comparison regardless of the casing in the product_name field. However, note that this approach has high overhead since every value must be transformed before the comparison.

Aggregation Functions

Aggregation functions are used to perform calculations over a set of values. Examples include:

- SUM()
- COUNT()

- MIN()
- MAX()

Example: Counting the total number of unique users by aggregating visitor_id values using COUNT(DISTINCT visitor_id).

GROUP BY Clause

To perform aggregations by a specific column (e.g., country), use the GROUP BY clause:

```
GROUP BY country
```

This enables insights like the top 5 countries by number of unique visitors.

HAVING Clause

To filter results **after** aggregation, use the HAVING clause. For example, to find users who visited a website more than once:

```
HAVING COUNT(*) > 1
```

Think of HAVING as the WHERE clause for aggregated results.

Data Type Conversion with CAST and SAFE_CAST

The CAST() function converts one data type to another, such as:

- Converting a number to a string
- Converting a string to a DATE type

This is useful with functions like EXTRACT(), where you might need to extract a specific part of a date.

In cases where invalid values might cause errors (e.g., casting a string to INT), use SAFE_CAST(). This returns NULL instead of throwing an error.

Handling NULL Values

NULL represents missing or unknown data. Key things to remember:

- NULL is not equal to anything—not even another NULL
- To filter NULL values, use:
- WHERE transaction_id IS NOT NULL

By default, NULLs are respected in aggregations, but their behavior depends on the function.

String Aggregation Functions

When multiple rows share identical values (like visitor ID or transaction ID), and only one column (e.g., product) varies, aggregation can help.

STRING_AGG()

Concatenates string values from different rows into a single string:

```
STRING_AGG(product_name)
```

Result: One row per transaction_id, with a string of all related products.

ARRAY_AGG()

Combines values into an array for better readability:

```
ARRAY_AGG(product_name)
```

You can also include quantities:

```
ARRAY_AGG(STRUCT(product_name, quantity))
```

Using the WITH Clause (CTEs)

The WITH clause defines one or more **Common Table Expressions (CTEs)**. These act like temporary tables used within a single query:

```
WITH top_countries AS (
  SELECT country, COUNT(*) AS total
  FROM visitors
  GROUP BY country
)
SELECT * FROM top_countries WHERE total > 100
```

Using Unions and Joins to Enrich Your Queries

Dataset Structure

- There is a separate table for **daily weather temperatures** for each year since **1929**.
- For example, there is one table for 1929, one for 1930, and so on.
- That’s a lot of tables to query and combine — but don’t worry, it’s manageable.

The **weather station location details** (latitude, longitude, state, station name, etc.) are stored in a **single lookup table**.

Important fields like **Country** and **State** are **not** present in the daily temperature tables due to a concept called **normalization**, but we can retrieve them by joining the tables together.

Identifying the Linking Field

Before we can join the tables, we need to determine the **common field** used for linking.

- Is the unique identifier for weather stations **USAF** or **WBAN**?
- Neither is unique **by itself**.
- However, the **combination** of USAF and WBAN provides a **unique identifier**.

We can use this **concatenated key** to join the temperature tables and station details.

Using UNION

UNION is used to **append** data from multiple tables — stacking them **vertically**.

Example: You want to retrieve temperature data for **1929** and **1930**. You can:

```
SELECT * FROM `dataset.gsod1929`
UNION DISTINCT
SELECT * FROM `dataset.gsod1930`
```

- UNION DISTINCT removes duplicates.
- Use UNION ALL if you want to keep duplicates.
- **Important:** All SELECT statements in a UNION must have the **same schema**: same number of columns, same order, and compatible data types.

Using Wildcard Tables

Typing UNIONs manually for **every year** can be tedious.

BigQuery allows **wildcard tables**, which represent a **union of all matching tables**.

Example:

```
SELECT * FROM `bigquery-public-data.noaa_gsod.gsod*`
```

This automatically unions all tables that start with gsod.

You can also **filter** specific tables using the special column _TABLE_SUFFIX:

```
SELECT *
FROM `bigquery-public-data.noaa_gsod.gsod*`
WHERE _TABLE_SUFFIX BETWEEN '2000' AND '2023'
AND _TABLE_SUFFIX NOT IN ('2010', '2015')
```

Schema Considerations with UNION

- When using UNION, schemas **must match**.
- If column numbers, names, or data types differ, the query **will fail**.
- Be mindful of changes in schema over the years.

Using JOIN to Enrich Data

Joins help **enrich** your data by **adding columns** from other tables.

Example: To get station name, state, and location, you'll need to JOIN the temperature data with the station lookup table.

Understanding Primary and Foreign Keys

Let's consider two simple tables:

Employee Table

- EID (Employee ID) is a **primary key** – unique and not null.

Department Table

- DID (Department ID) is also a **primary key**.

In the **Employee** table, DID is a **foreign key** referencing the Department table.

- **Primary Key**: unique, not null.
- **Foreign Key**: matches a primary key in another table, may have duplicates.

Joining Weather Data

We can join temperature and station tables using:

```
ON temp.USAF = station.USAF AND temp.WBAN = station.WBAN
```

By **concatenating USAF and WBAN**, we create a **unique join key**.

Types of JOINs

- **INNER JOIN**: Returns only rows that match in both tables.
- **LEFT OUTER JOIN**: All rows from the left table and matched rows from the right.
- **RIGHT OUTER JOIN**: All rows from the right table and matched rows from the left.
- **FULL OUTER JOIN**: All rows from both tables, matched or not.
- **CROSS JOIN**: Returns the Cartesian product — all combinations of both tables.

Importance of Join Conditions

A missing or incomplete join condition can create **explosions** in row count.

Example:

- If you join on just USAF (not both USAF and WBAN), you may get **128x more rows**.
- This behaves like a **CROSS JOIN** due to poor join key selection.

Best Practices

- Understand your **data relationships** before joining.
- Always check which fields are **primary** and which are **foreign** keys.
- Use **multiple join conditions** if needed.
- Be aware of **dirty data** that can pollute your results.

Module -3 Cleaning and Transforming your Data

5 Principles of data integrity

High-quality datasets will yield high-quality insights.

So let’s spend some time in this lesson to talk about the **5 principles of dataset integrity**.

The majority of data quality lectures will have one central theme: **Garbage in, garbage out.** If you want to build amazing machine learning models, you can't feed them with garbage. You've got to feed them with really good data to begin with.

The same goes with data analysis. You have to get **extremely good, clean data** in order to get **good, useful, actionable insights**.

So let's talk about what makes quality data and some of the things that we can do to bring data to that level. High quality datasets follow these strict rules, and there are **five of them**:

1. **Valid** – they conform to your business rules.
2. **Accurate** – they conform to some true objective value.
3. **Complete** – you can see the whole picture; no data elements are filtered, truncated, or lost.
4. **Consistent** – there is harmony across data systems; no contradictory facts.
5. **Uniform** – same units, same formats, same standards across datasets.

Validity

When modern database technology is used to design data-capture systems, validity is fairly easy to ensure.

Invalid data mostly arises in legacy contexts (where constraints were not implemented in software),
or where inappropriate data-capture tools were used (e.g., spreadsheets without cell validation).

Common **data constraints** include:

- Data-type constraints
- Range constraints
- Mandatory constraints
- Set-membership constraints
- Foreign-key constraints
- Regular expression patterns
- Cross-field validation
- Unique constraints

Speaking of uniqueness, consider the following images:

- A phone number
- A car license plate
- A mailbox (or address)

What do they have in common? Each is a **unique identifier**. Why would duplicate license plates be a problem? Because **someone could get a speeding ticket intended for another person**.
Likewise, duplicate phone numbers or shared addresses can cause massive issues.

In data, **duplicate or stale records** (like an old address) are problematic. They lead to inflated or inaccurate query results, such as with outer joins that multiply results due to duplication.

Range Constraints

Example:
You have a table of **dice rolls**. Which value looks out of place: **1 or 7**?

- If you rolled **one die**, 7 is out of range.
- If you rolled **two dice**, 1 is out of range (lowest possible total is 2).

It depends on context. **Range and data type** are just two types of constraints. There are many others that also determine validity.

Accuracy

Data must match a **source of truth**.

Example:
You see this list of US states:

Washington, Alaska, California, Hot Dog, Florida, Texas

"Hot Dog" may be a valid string, but it's **not a real U.S. state**.

Sources of truth vary—some include U.S. territories like **Guam** or **Puerto Rico**, others don't. **Accuracy** depends on your chosen definitions and data standards.

Data cleansing can't usually guarantee accuracy, because it would require external "gold standard" sources, which are often **unavailable**.

Completeness

With just **partial data**, it's hard to understand the full picture.

Imagine two vague images—can you tell where you are? Maybe not, but once the full image is revealed, you know you're in **London**.

We often **trust row/column data too much**. But some data may be **missing, filtered, or truncated**.

Consequences of incomplete data:

- Biased insights
- Overfitting in machine learning
- Misinterpreted patterns
- Over-dependence on limited context (e.g., "Lamp and Clock Land")

Consistency

Example:
Two databases contain ownership info for a house at **123 ABC St**.

- In **Database A**, it's linked to **Owner ID 12**.
- In **Database B**, **Owner ID 12** is linked to another address, and **123 ABC St** is linked to **Owner ID 15**.

This is a **referential integrity issue**. Which owner is correct?

Inconsistencies arise when systems contradict each other. To resolve it, you might:

- Check timestamps
- Evaluate source reliability
- Contact the data subject

Uniformity

Uniform data means using **the same measurement units and formats**.

A real-world example: On **November 10, 1999**, NASA lost its **$125 million Mars Climate Orbiter**.
Why? Because one team used **imperial units**, the other used **metric units**. This lack of uniformity caused the orbiter to **burn up** in Mars' atmosphere.

Uniform data is essential for correct interpretation and integration.

Data issues can include:

- Conflicting data types
- Out-of-range values
- Duplicates
- Missing or stale data
- Misaligned measurement systems

Dataset Shape

Datasets can come in different shapes:

1. **Tall but narrow** (lots of rows, few columns)
2. **Wide but short** (many columns, few rows)
3. **Small datasets** (average columns and rows)
4. **Ideal datasets** (sufficient records and relevant columns)

No shape is "perfect"—it depends on your **analytical goals**. You need enough observations to make **justifiable inferences**.

Dataset Skew

Skew refers to the **distribution of values**.

Examples:

- A dataset where 90% of employees are in **California**
- A sales dataset focused on only **one country**, despite being for an international business

Questions to ask:

- Did we **collect everything**?
- Are we **missing rows**?
- Are the results **representative**, or is skew masking problems?

Skew can affect:

- Interpretation of trends
- SQL query results (e.g., **GROUP BY**, **ORDER BY**)
- Model accuracy and generalization

Skewed data isn't always bad, but it must be understood in context.

Cleaning and transforming data using SQL

Data Integrity with SQL and BigQuery

In this lesson, we revisit each of the five data integrity rules we covered in the previous lesson, and demonstrate how SQL can be used to clean, prepare, and transform datasets to ensure high-quality data.

1. Validity

Validity ensures that your data conforms to business rules and constraints.

- SQL and BigQuery offer tools to enforce these rules.
- You can define whether a column allows NULL values.
- Required fields like Name or ID—which are unique identifiers—should be set as NOT NULL.
- These constraints can be defined during table creation to ensure that only valid rows are ingested from upstream systems.
- Alternatively, data can be validated *after* ingestion using conditional SQL logic like IF...THEN...ELSE and CASE WHEN.

Even when your data types and null constraints are satisfied, it’s essential to verify that the data is accurate and makes sense.

- Calculated fields can be used for testing data values.
- Lookup comparisons against objective datasets (e.g., a table of the 50 U.S. states) can confirm data accuracy.
- SQL joins, subqueries, and Common Table Expressions (CTEs) are useful for these validations.
- The IN operator is helpful for checking if values match expected entries.

2. Completeness

Completeness refers to identifying missing or incomplete data.

- Analysts look for data skew and null values.
- The COALESCE() function allows fallback values when encountering NULL.
- Data can be enriched by combining it with other datasets using JOIN and UNION.
    - For example, combining multiple yearly temperature datasets using UNION.
    - Enriching temperature readings with station metadata via JOIN.

3. Consistency

To handle dirty or inconsistent data, SQL provides various functions:

- **Date Parsing**: Normalizing date formats is common.
- **String Functions**:
    - SUBSTRING()
    - REPLACE()
      These help clean up inconsistencies, especially when data comes from multiple sources.

Importantly, you don't need to overwrite your raw data. These transformations can be part of a repeatable data-cleaning workflow applied downstream.

4. Uniformity

Uniformity ensures that data types and formats are standardized across the dataset.

- This is essential when comparing or visualizing data (e.g., avoiding "apples to oranges").
- Always document your code. For instance, note whether you’re using the metric or imperial system.
- Use functions like:
    - FORMAT()
    - CAST()
      These standardize values.

Common issues like time zone differences can be handled using:

- DATE(), TIME(), and DATETIME() functions that support time zone parameters.
- Functions to convert Unix epoch time to standard timestamps also aid in uniformity.

5. Example: Handling NULL vs. Blank Strings

Consider weather data pulled from NOAA. Suppose your SQL query includes:

```
WHERE state IS NOT NULL
LIMIT 10
```

Yet, blank state values still appear in your results.

Why?
Because a blank string ('') is not the same as NULL.

- A blank string is a valid data value.
- NULL represents the absence of any value.

For example:

- Latitude and Longitude columns show NULL when no data is present.
- A blank string appears empty but is technically *not* NULL.

Solution:
Update your WHERE clause to filter out both nulls and blank strings:

```
WHERE state IS NOT NULL AND state != ''
```

This ensures you're only selecting rows where the state field is truly populated with meaningful data.

Cleaning and transforming data other than SQL

Data Cleaning and Transformation Tools Beyond SQL

Tools Overview

We will explore the following tools:

- **Cloud Data Fusion** and its **Wrangler** tool
- **Dataflow**
- **Dataprep**
- **Dataproc**
- **Dataform** (introduced briefly, covered in more detail later)

Cloud Data Fusion and Wrangler

Cloud Data Fusion is a fully managed, cloud-native, enterprise data integration service. It helps quickly build and manage data pipelines through a graphical, no-code interface. It’s ideal when:

- You can't easily export all data to Excel or Sheets
- Metadata is inconsistent
- There is no business glossary
- Operationalizing data preparation is difficult

Wrangler is a powerful tool within Cloud Data Fusion that allows users to:

- View and explore data
- Apply transformations on a 10MB sample
- Visualize the effect of transformations before scaling them
- Work with data from Cloud Storage, Kafka, databases, and more
- Derive schemas and operationalize data with point-and-click simplicity

Transformations can be saved as a **recipe**, and once finalized, you can build a pipeline with sources, transformations, and output sinks such as BigQuery.

Dataflow

Dataflow is a Google Cloud service for unified batch and stream data processing at scale. It enables:

- ETL operations
- Data movement through a pipeline of stages (e.g., read, transform, aggregate, write)
- Simple to complex processing scenarios

Dataflow is based on **Apache Beam**, an open-source model that allows:

- Unified programming for both batch and stream processing
- Abstraction of low-level distributed computing tasks
- Execution on different runners (e.g., local machine, cloud)

You define your logic with an Apache Beam SDK, and Dataflow manages orchestration, scalability, and infrastructure.

Dataprep

Dataprep is a GUI-based tool operated by Trifacta and built on **Dataflow**, offering:

- Auto-scalability for massive datasets
- Integration with BigQuery, Cloud Storage, and local files (CSV, JSON, tables)
- Direct exports to Looker Studio, Vertex AI, and more

Common use cases:

- **Data onboarding** – Integrate third-party data into existing formats
- **Data science & ML** – Prepare clean, structured data for model training
- **Marketing analytics** – Enrich datasets from sales, finance, and support for cross-functional analysis

You can build transformation steps with wranglers, preview the flow, and execute the job, which triggers a Dataflow job under the hood. Dashboards allow monitoring and metric analysis.

Dataproc

Dataproc is a fully managed, scalable service to run open-source frameworks such as:

- Apache Hadoop
- Apache Spark
- Apache Flink
- Presto

Use cases include:

- Data lake modernization
- Secure and large-scale ETL and data science tasks

Dataproc provides:

- Autoscaling clusters
- Integration with BigQuery, Cloud Storage, Bigtable, Logging, and Monitoring

For example, Dataproc can be used to ETL terabytes of raw logs directly into BigQuery for business intelligence.

For detailed guides on each product, refer to the official Google Cloud documentation.

Module – 4 Ingesting and storing New Big Query Dataset

Permanent vs. Temporary Tables

When you run queries in BigQuery, the results are always saved either to:

- A **permanent table** (if you specify one), or
- A **temporary table** (even if you don't explicitly create one).

Many data analysts run SQL queries on raw data sources and save the results into **new reporting tables** for future analysis.

Before running your query, you can specify a **destination table**, and choose one of the following options:

- **Append** data to an existing table
- **Overwrite** data in an existing table
- **Create** a new table if none exists by that name

If you **don't specify a destination table**, the results are copied into a **temporary table**, which is available for **24 hours**.

You can **copy a temporary table to a permanent table** by clicking the **"Save Results"** button in the results window and choosing **BigQuery Table**.

Query Cache and Temporary Tables

Recall the discussion on **query cache**:

- It refers to the **temporary table** created when a query runs.
- Identical queries run again within a short time may reuse cached results, if:
    - The **underlying table has not been modified**, and
    - The query does **not** use **nondeterministic functions** (e.g., CURRENT_DATE())

Views in BigQuery

A **view** is a **virtual table** defined by a SQL query. Think of it as a **saved SELECT statement**.

When you query a view, you're essentially running a SELECT statement that fetches data from one or more tables.

Views **do not store data**. They execute the underlying query each time they are accessed.

You can query views via:

- BigQuery **Web UI**
- **Command-line tool**
- **API**
- As a data source in visualization tools like **Looker** or **Looker Studio**

Since views are **not materialized**, querying a view means running the full query every time, which may impact performance and cost.

Queries are billed based on the total data **referenced** by the query—not just the final result.

Creating Views with DDL

You can create a view using the CREATE VIEW DDL statement:

```
CREATE VIEW [IF NOT EXISTS] dataset.view_name AS
```

```
SELECT ...
```

- Use OR REPLACE to **overwrite** an existing view:
- CREATE OR REPLACE VIEW dataset.view_name AS ...
- Use IF NOT EXISTS to **avoid overwriting** an existing view:
- CREATE VIEW IF NOT EXISTS dataset.view_name AS ...

Creating Tables with DDL

To create an **empty table** with a specific schema, use the CREATE TABLE statement:

```
CREATE TABLE dataset.table_name (
  column1 STRING,
  column2 INT64,
  ...
);
```

To insert data later, use the INSERT statement (DML):

```
INSERT INTO dataset.table_name (column1, column2)
VALUES ('value1', 123);
```

You can also use a **load job** from the command line to populate tables from external sources such as **CSV files**.

Materialized Views in BigQuery

Materialized views are **precomputed** views that **cache query results** for faster access and reduced resource use.

Key Features:

- BigQuery automatically uses the **cached result** from a materialized view when possible.
- When the **base table changes**, only the **delta** is processed to refresh the view.
- Queries using materialized views are usually **faster and cheaper** than those querying base tables.

Example Workflow:

1. **Step 1**: A materialized view is created and kept fresh.
2. **Step 2**: Queries still reference the base table.
3. **Step 3**: BigQuery begins using the materialized view for more efficient query execution.

Things to Know:

- You can **grant access** to a materialized view at:
  - Dataset level
  - View level
  - Column level
- You can also manage access using **IAM hierarchy**.
- You can **manually refresh** a materialized view.
- If the **base table is partitioned**, you can **partition the materialized view** too.
- By default:
  - Materialized views are **automatically refreshed within 5 minutes** of changes to the base table.
  - But they are **not refreshed more frequently than every 30 minutes**.

Loading Data into BigQuery

EL, ELT, and ETL Methods

- **EL (Extract and Load)** is used when data is imported *as-is*, and the source and target have the same schema.
- **ELT (Extract, Load, Transform)** is used when raw data is loaded directly into BigQuery and transformations are applied there.
- **ETL (Extract, Transform, Load)** is used when transformations are done in an intermediate service before loading the data into BigQuery.

EL – The Simplest Case

The simplest case is **EL**. If the data is usable in its original form, there's no need for transformation—just load it.

In the hands-on lab at the end of this module, you will practice ingesting data from a variety of sources.

You can also use **BigQuery Data Transfer Service** to automate data movement into BigQuery on a scheduled and managed basis.

ELT – Transform in BigQuery

Sometimes, raw data from the source system is not usable in its original form. In such cases, you can:

1. Load the raw data into BigQuery.
2. Use SQL to build and apply the necessary transformations.

This is known as the **ELT** approach. BigQuery is quickly becoming the center of the data stack, and many organizations are moving from traditional ETL tools like Spark to ELT with SQL pipelines in BigQuery.

Data Formats and Ingestion Options

When EL or ELT is preferred, BigQuery can ingest datasets in various formats. Once data is inside BigQuery's native storage, it is fully managed by Google—this includes replication, backups, autoscaling, and more.

You also have the option to:

- **Query external data sources directly**, bypassing BigQuery's managed storage (more on this in the next lesson).
- **Stream records into BigQuery** using tools like **Dataflow**, **Data Fusion**, or the **BigQuery API**.

Streaming Use Case: Event Logging

One common use case for streaming is event logging. For example:

If you have a mobile app that tracks events, your app or backend servers can record user interactions or system errors and stream them into BigQuery in real time.

You can then analyze this data to:

- Identify high interaction areas.
- Monitor system issues.
- Detect and respond to error conditions as they happen.

To learn more about building real-time dashboards with streaming data, refer to the BigQuery documentation.

ETL – Complex Transformations

Sometimes, the transformations required are complex and numerous. In such cases, **ETL** may be the better option.

You can use services like:

- **Cloud Data Fusion**
- **Dataflow**
- **Dataproc**

These allow you to build pipelines and perform the necessary transformations *before* loading the data into BigQuery.

Working with External Data Sources in BigQuery

An external data source is a source that you can **query directly from BigQuery**, even though the data is **not stored in BigQuery's managed storage**.

For example, you might have data:

- In a different Google Cloud database,
- In files on Cloud Storage, or
- In a different cloud product altogether.

You may want to analyze this data in BigQuery without migrating it.

Mechanisms for Querying External Data

BigQuery supports two main mechanisms for working with external data:

1. **External Tables**
2. **Federated Queries**

Use Cases for External Data Sources

You might use external data sources in the following situations:

- Performing **ad-hoc queries** where performance isn't critical.
- Accessing **short-term or infrequently accessed data**.
- Reading **constantly changing data**, like from a **Google Sheet**.
- Joining **BigQuery tables** with frequently updated external data.
- Running **ELT (Extract, Load, Transform)** workloads by transforming data during ingestion with a CREATE TABLE ... AS SELECT query.
- Accessing data stored in **AWS** or **Azure** with **BigQuery Omni remote tables**.

Comparison: External Tables vs. Federated Queries

Feature	External Tables	Federated Queries
Supported Data Sources	Cloud Bigtable, Cloud Storage, Google Drive	Cloud Spanner, Cloud SQL
Metadata Location	Stored in BigQuery	Fetched live from external database
Security	Supports **column- and row-level access controls**	Limited options
Use Case Example	Querying CSVs in Cloud Storage	Querying relational data in Cloud Spanner
Cross-cloud Support	Yes, via **BigQuery Omni**	No

Considerations for External Tables

- **Performance** is generally **slower** compared to native BigQuery tables.
- **Caching** and **query size estimates** are **not supported**.
- **Region compatibility** matters. You must ensure your Cloud Storage bucket region matches the BigQuery dataset's location.
- **Refer to: BigQuery Data Location Documentation**

Limitations of External Data Sources

- **No guarantee of consistency**: Changes to the source data during a query can lead to unexpected behavior.
- **Lower query performance** than native tables.
- **Query performance varies** by storage type:
  - GCS is **faster** than Google Drive.
  - Performance is roughly equivalent to reading from the external system directly.
- **You cannot**:
  - Export data from an external source via BigQuery job.
  - Use **wildcard table queries** on external sources.

Module – 5 Visualizing Your Insights from BigQuery

Why Use Data Visualization?

Data visualization helps uncover insights **faster** than raw data analysis. Instead of scanning thousands of rows, interactive tools allow you to **explore trends visually**.

Key Benefits

Benefit	Description
Faster Insights	Spot trends and anomalies quickly.
Interactivity	Drill down into details by clicking data points.
Storytelling	Present findings in a **clear and engaging** way.
Summarized Data	Convert billions of rows into meaningful aggregates.
Performance Gains	Looker Studio and other tools **process queries in BigQuery**, ensuring **faster rendering**.

Example: A **time-series dashboard** lets you instantly **zoom into spikes** and uncover anomalies without needing SQL expertise.

The Science of Visualization

Humans don't just see with their **eyes**; they **interpret with their brains**. Our ability to recognize patterns instantly—called **preattentive processing**—allows us to **process visuals much faster** than raw text or numbers.

Processing Type	Characteristics	Example
Preattentive Processing	**Fast, automatic, subconscious**	Instantly recognizing **bolded** text.
Attentive Processing	**Slow, requires focus**	Searching for a specific word in a dense paragraph.

Example: If you're searching for the number **"5"** in a grid of numbers, it's hard. But **bolding, coloring, or highlighting** them makes it **instantaneous**.

Using Visual Encoding to Guide Attention

To help viewers **focus on key insights**, apply **visual encoding techniques**:

Technique	How It Helps
Color & Contrast	Highlights key data points.
Size & Shape	Emphasizes differences between values.
Positioning	Organizes information for clarity.
Borders & Grouping	Directs focus to related elements.

Example: A **heatmap** uses color intensity to show areas of high or low activity, making it **easier to spot trends** at a glance.

Choosing the Right Visualization

Selecting the right chart type is **crucial** for **effective communication**.

Chart Type	Best For	When to Avoid
Line Chart	Time-series data trends.	Comparing categories.
Bar Chart	Comparing values across groups.	Too many categories.
Pie/Donut Chart	Showing proportions in simple datasets.	More than 5–6 categories.
Scatter Plot	Showing relationships between variables.	Small datasets.

Example: A **donut chart** is great for **gender comparison in book characters**, but **not ideal for tracking book sales over time**—a **line chart** works better for that.

Impact of Visualization in Decision-Making

The final **dashboard or report** is often the **only thing stakeholders see**—not the SQL behind it.
A well-designed visualization ensures **quick, data-driven decisions**.

Key Takeaways:

- **Simplify the Message** – Highlight key insights, not raw numbers.
- **Optimize Readability** – Use contrasting colors and meaningful labels.
- **Encourage Exploration** – Enable drill-downs for deeper analysis.

Example: Instead of listing **thousands of transactions**, show a **summary table** with **top trends and outliers**.

Dimensions vs. Measures in Visualization Tools

Concept	Definition	Examples
Dimensions	**Qualitative** values that categorize data.	Names, Dates, Locations.
Measures	**Quantitative** values used for analysis.	Revenue, Count, Average.

Note: **Not all numbers are measures**—a **Zip Code** is numeric but should be treated as a **dimension**.

Looker Studio in Action

Looker Studio allows you to:

- **Visualize data** in charts, tables, and maps.
- **Restrict access** with user roles.
- **Share reports** with teams.
- **Enhance decision-making** with interactive dashboards.

Example: A Looker Studio dashboard may include:

- A **histogram** showing user distribution.
- A **line chart** tracking sales trend.
- A **geographic heatmap** highlighting market performance.

Connected Sheets: Accessing BigQuery Data with Familiar Tools

Connected Sheets merges the power of Google Cloud with the ease of Google Workspace, enabling users to analyze data at scale without needing advanced technical skills.

Key Benefits of Connected Sheets

Benefit Category	Description
Accessibility	Allows non-technical users to access, prepare, analyze, and share BigQuery data via Google Sheets.
Reduced Dependency	Minimizes reliance on data scientists by enabling self-service analysis.
Improved Efficiency	Reduces workload for specialized roles by empowering broader teams to perform their own analysis.
Scalable Analysis	Lets users analyze billions of rows of data without using BigQuery directly.
Familiar Tools	Users can apply known Sheets features like pivot tables, charts, and formulas.
Security and Governance	Data remains in its original secure source; access is managed and collaboration is controlled.

How to Connect Google Sheets to BigQuery

Step Action

1 Open a blank Google Sheet.

2 Go to Data > Data connectors > Connect to BigQuery.

3 Select the correct GCP project and dataset (or use a public dataset).

4 Choose the desired table and connect.

5 Analyze the populated data using familiar tools like charts and pivot tables.

Alternate Method via BigQuery UI:

- From the BigQuery Schema tab, click Export > Explore with Sheets.
- To query specific rows, run a SQL query, then click Explore data > Explore with Sheets.

Important Distinction:

- Using **"Save results"** exports static data to Sheets with **no live connection**, similar to copy-paste.

Analytical Features Available in Connected Sheets

Feature	Description
Charts	Create visuals like pie or bar charts (e.g., payment types used for taxi rides).
Functions	Use standard spreadsheet functions for transformation and calculation.
Pivot Tables	Summarize large datasets with dynamic grouping and aggregation.
Data Extracts	Filter and isolate relevant data for specific views or reports.

Data Refresh Options

Type	Details
Manual Refresh	Default; requires users to refresh to update the data.

Object-Level Refresh	Refresh a specific chart or pivot table.
Full Sheet Refresh	Refresh all data in the connected sheet.
Scheduled Refresh	Can be set up (maximum frequency: once per hour).

Security & Performance Advantages

- Avoids the common pitfalls of using spreadsheets as databases (corruptibility, size limitations).
- Maintains data security—no local copies are stored, reducing risk of data loss or manipulation.
- Enables live connections, ensuring up-to-date insights without exporting datasets.

Common Data Visualization Pitfall: Data Freshness vs. Performance

Key Question: Where is Your Data Coming From?

Data Source Type	Description	Pros	Cons
Static Tables	Dashboard pulls data directly from precomputed tables.	Fast loading	Data may become outdated
Live Views (SQL)	Dashboard runs complex queries on-the-fly via views.	Always fresh	Performance intensive; slower response times
Scheduled Reporting Tables	Intermediate solution—queries populate tables at scheduled intervals.	Balances freshness and performance	Not real-time but still timely

Scheduled Queries: A Balanced Solution

To reduce performance load and still maintain data freshness, use **scheduled queries** in BigQuery.

Feature	Explanation
What They Are	Recurring queries that populate tables on a predefined schedule.
Support for SQL Types	Can include both Data Definition Language (DDL) and Data Manipulation Language (DML).
Time-Based Organization	Queries can be parameterized to include date/time fields in table names or results.
Multiple Schedules	Create different schedules for different reporting needs.
Time Conversion	All schedules are converted from your local time to UTC, which does not observe daylight saving.

BigQuery BI Engine: Speeding Up Dashboards

To build real-time dashboards with **sub-second latency**, use **BigQuery BI Engine**, an in-memory analysis service integrated with BigQuery.

Feature	Description
High Performance	Enables sub-second response times without needing custom OLAP cubes.
In-Memory Caching	Maintains an intelligent cache layer to accelerate query results.
Built-in Integration	Part of the BigQuery architecture—no separate infrastructure is needed.
Cost-Effective	Reduces complexity and management overhead compared to traditional BI solutions.

Refer to the GitHub demo repository for examples of scheduled queries in action.

Introduction to Looker Studio

Looker Studio (formerly *Data Studio*) is a powerful web-based tool that helps create interactive dashboards and reports using data from various sources, supporting smarter business decisions. It is distinct from **Looker**, which is a more advanced business intelligence platform used for real-time analytics and exploration.

Looker vs. Looker Studio

Tool	Description
Looker	A business intelligence platform that helps users explore, analyze, and share real-time data.
Looker Studio	A simpler web interface for creating interactive dashboards and reports from multiple sources.

Getting Started with Looker Studio

Creating a Report

You can create a new report in two ways:

Method	Action
From Template Gallery	Click **Blank Report** in the center of the home screen.
From Navigation Pane	Click the **Create** button on the left side of the screen.

Looker Studio supports multiple data sources—including Google and Partner connectors—within a single report.

Data Source Permissions

Role	Access Implications
Viewer	May see all data from connected sources unless restricted.
Editor	Can use all fields from any connected data source to build or modify charts.

To add a data source, click **"Add to report"**, then choose your desired dataset and specify the **Dimensions** (descriptive attributes) and **Metrics** (quantitative measures).

Working with Data

Term	Meaning
Dimension	Describes attributes (e.g., Country, Product).
Metric	Measures or aggregated data (e.g., Revenue, Count), also referred to as *measures*.

You can edit fields by clicking the pencil icon near the data source name or create **calculated fields**—either data source-level or chart-specific (also known as **chart-level**). These fields can perform operations and use **CASE statements** for conditional logic.

Visualizing Data

Action	Description
Switch View	Change data table view to chart using the **Chart** option in the properties panel.
Add Chart	Insert additional charts using the **Add a chart** option from the toolbar.
Customize Layout	Resize and arrange elements like tables and charts on the canvas.
Set Fields	Define Dimensions and Metrics by selecting from the Data list.
Reorder Fields	Change display order by dragging fields within the **Metric** section.
Name Your Report	Reports can have duplicate names, as Looker Studio saves them in Google Drive.
View Mode	Use the **View** toggle to preview the report in viewer mode, where it becomes non-editable.

When users mouse over a chart in View mode, they can interact with **live data** (e.g., viewing that in 2000 there were 31 disasters due to extreme temperatures).

Sharing and Permissions

- **Sharing reports** works similarly to sharing files in Google Drive.
- **Important**: Sharing a report does **not** automatically share its underlying data sources—they must be shared separately.

References:

- Looker Studio Demo Guide (GitHub)

Using Notebooks for Data Analysis and Machine Learning on Google Cloud

Why Standard Tools Fall Short

Traditional software development tools are not optimized for data analysis and machine learning. These tasks involve:

- Frequent visualization through plots
- Iterative execution of small code segments
- Constantly printing and inspecting outputs

Running entire scripts repeatedly for such tasks becomes inefficient, which led to the rise of **notebooks**—a more interactive and flexible environment.

Benefits of Notebook Environments

Feature	Description
Interactive Cells	Code, text, and plots are organized into individual, executable blocks.
Seamless Integration	Combine narrative, analysis, and visualization in one document.
Collaboration	Easily shared and edited like Google Docs or online tax platforms.

For example, colleagues can adjust a plot by simply modifying the relevant cell in the notebook.

Cloud-Hosted Notebooks vs. Traditional Servers

Previously, notebooks ran on individual servers, which posed challenges such as availability and scalability. Now, with **cloud-hosted notebooks**, you benefit from:

- Always-on availability
- Real-time collaboration
- Scalable compute infrastructure

Setting Up Notebooks in Google Cloud

Spinning up a notebook in **Google Cloud Vertex AI** is as simple as one click. These notebooks:

Feature	Details
Based on Jupyter Notebooks	Built on industry-standard **JupyterLab** interface
Pre-installed Libraries	Comes with pre-configured ML libraries like TensorFlow, Scikit-learn
Compute Engine Integration	Treated as regular VM instances under your Google Cloud project
Flexible Hardware	Choose and modify machine type, memory, and accelerator settings
Easy Sharing	Others can access the notebook by connecting via a shared URL

Google Colab and Colab Enterprise

Tool	Description
Google Colab	Hosted Jupyter Notebooks with free GPU/TPU, no setup required
Colab Enterprise	Integrated in **Vertex AI** with added features for enterprises

Colab is particularly suited for **machine learning**, **data science**, and **education**.

BigQuery Integration with Notebooks

You can use **magic functions** (prefixed with % or %%) in Jupyter notebooks to interact with your environment or external systems.

BigQuery Magic Function

```
%%bigquery df
SELECT * FROM my_dataset.my_table LIMIT 100
```

This runs a BigQuery query and stores the result in a **Pandas DataFrame** called df, allowing further manipulation and analysis.

Working with Pandas DataFrames

Feature	Description
Tabular Data Display	Structured rows and columns similar to spreadsheets
Built-in Summary Functions	Quickly describe, sort, and summarize numeric data
Supports Plotting	Create visualizations like bar plots and line charts from within the notebook

Caution: Avoid pulling large (multi-terabyte) datasets directly—sample instead to stay within memory limits.

Example: Query to Visualization

1. Query BigQuery and save into DataFrame
2. Generate a simple bar plot using df.plot.bar()

This two-step process is efficient and visual.

Visualization Theory Recap

Data visualization is both **an art and a science**. Earlier, we discussed:

- **Preattentive vs. attentive processing** for better design
- **Bad vs. good chart practices** for clarity
- Tools like **Connected Sheets** and **Looker Studio** for dashboard creation

Reference:

- **Google Cloud Training Notebooks Demo GitHub**

Module – 6 Developing scalable data transformations pipelines in BigQuery with Dataform

Dataform: SQL-Based Data Transformation in BigQuery

From ETL to ELT in the Cloud

Organizations are increasingly shifting from traditional **ETL (Extract, Transform, Load)** workflows in dedicated environments (like Spark) to **ELT (Extract, Load, Transform)** using SQL pipelines in **BigQuery**.

Why the shift to ELT in BigQuery?

Reason	Description
Serverless Architecture	No infrastructure or Spark clusters to manage
Wider Skill Availability	SQL skills are more common than expertise in Spark or ETL tools
Centralized Raw Data	Data from disparate sources lands in BigQuery, ready for transformation

Data Transformation Needs in BigQuery

Raw data accumulation leads to **thousands of incompatible tables** that require transformation through **multiple layers** of curation:

1. **Raw Ingestion Layer** – Centralized but unstructured
2. **Transformation Layers** – Gradual refinement and alignment
3. **Curated Layer (Single Source of Truth)** – Clean, reliable data for:
    - Analytics
    - Reporting dashboards
    - Visualization tools

Challenges Without a Unified Framework

Problem	Impact
SQL reuse limitations	Rewriting similar queries across scripts
Lack of testing support	No native way to validate data consistency
Dependency management	Requires external tools
Poor documentation integration	Metadata often managed in separate catalogs

What is Dataform?

Dataform helps teams **develop, test, document, and orchestrate SQL pipelines in BigQuery**—using only SQL. It provides an integrated environment for building scalable data transformations, removing the need for complex infrastructure or manual processes.

Key Benefits of Dataform

- Build a **centralized, production-ready data model**
- Implement **CI/CD**, **version control**, and **collaboration workflows**
- Perform **data quality testing** and **automated documentation**
- Eliminate reliance on external orchestration and metadata tools

Dataform in Practice

Feature	Description
Infrastructure-Free Pipelines	Write and deploy transformation logic directly in BigQuery
End-to-End Development	Create, document, and test tables in one place
Team Collaboration	Use Git-based versioning, environments, and workflows
Self-Service Analytics	Analysts can build production pipelines independently

Dataform Core and CLI

Dataform Core is an **open-source framework** with a **CLI** (Command-Line Interface) that enables local development outside Google Cloud.

Supported Data Warehouses

- BigQuery
- Snowflake
- Redshift
- Azure SQL Data Warehouse
- PostgreSQL

As open source, **Dataform Core** can be extended with custom integrations.

SQLX: SQL Made Better

SQLX is the primary development language in Dataform. It's an **open-source extension of SQL**, offering enhanced capabilities while remaining fully backward compatible with regular SQL.

SQLX Enhancements

Feature	Description
Dependency Management	Define and manage inter-table relationships

Automated Data Quality Tests	Write assertions to ensure reliable transformations
In-Query Documentation	Embed table and field documentation directly within your SQL code

Reference:

- Dataform Documentation (Google Cloud)
- Dataform GitHub

Getting Started with Dataform

In this lesson, we'll explore how to begin using **Dataform**, and you'll conclude this module with a hands-on lab to practice what you've learned.

Understanding the Dataform Code Lifecycle

Once you've extracted raw data from source systems and loaded it into **BigQuery**, Dataform helps transform that data into a **clean, tested, and documented suite of tables** using code.

Key Stages in the Dataform Lifecycle

Stage	Description
Write SQL Workflow Code	Use SQLX and JavaScript to create transformation logic
Compilation in Real Time	Dataform compiles SQL code to SQL using dataform.json settings
Execution in BigQuery	Execute the compiled workflow directly in BigQuery
Git Integration	Collaborate through linked repositories in GitHub, GitLab, or Bitbucket
Manual or Scheduled Runs	Control when workflows execute, either manually or on a schedule

Setting Up Dataform

Step 1: Create a Repository

A **repository** contains one Dataform project. It includes:

- SQLX and JavaScript files
- Configuration files (e.g., dataform.json)
- Optional packages

You can link this repository to a remote Git repository hosted by **GitHub**, **GitLab**, or **Bitbucket Cloud**.

Step 2: Use a Development Workspace

A **development workspace** is your personal copy of a repository. It allows you to:

- Create, edit, and delete code
- Commit and push changes to the shared repository
- Preserve changes across sessions

Development workspaces let multiple users work on the same repository without interfering with each other.

Step 3: Configure dataform.json

This file defines key compilation settings:

Setting	Purpose
projectId	Specifies the Google Cloud project
defaultSchema	Sets the schema prefix used for tables
tablePrefix	Applies a prefix to created tables
schemaSuffix	Adds suffixes for organizing environments (can be overridden)

Note: The region selected during repository creation does **not** impact BigQuery data storage locations.

Required Permissions

To allow your code to access BigQuery datasets, appropriate **IAM roles** must be granted. This step is covered in the lab that follows this module.

Workflow Development in Dataform

Once setup is complete, most of your time will be spent developing and executing your data transformation pipelines.

1. Declarations

You can **declare any BigQuery table** as a source in your Dataform project. Declarations can include:

- **Descriptions** and **tags** (visible in BigQuery)
- **Dependencies**, defining execution order between objects

2. Transformations

This is where you **write your SQL logic** to transform raw data into clean, analytics-ready datasets. Use all your existing SQL knowledge to build powerful transformations.

3. Assertions

Assertions are **data quality tests** that detect violations of defined rules.

Feature	Description
Query-Based Test	SQL query returns rows that violate specified conditions
Assertion Failure	If rows are returned, the assertion fails
Automatic Checks	Assertions are run automatically with each SQL workflow update
Alerts	Alerts are generated if any assertions fail

Executing and Monitoring Workflows

Use the **Executions** tab in your development workspace to **run** and **monitor** workflows.

- You can execute the entire workflow or select specific elements.
- Execution logs include detailed error messages and system information.

Example: An execution may fail due to insufficient privileges on the service account. This information is clearly shown in the product, though redacted in training visuals.

Dataform: Getting Started Steps

Step 1: Launch Dataform in Google Cloud Console

1. Open the **Google Cloud Console**.
2. Use the **navigation menu** to go to **BigQuery**.
3. In BigQuery, click **Dataform** on the left-hand side menu.
4. If prompted, **enable the Dataform service**.
5. You'll be redirected to the **Dataform UI**.

Step 2: Create a Dataform Repository

1. Click **Create Repository**.
2. Provide a **name**, choose a **region**, and note the **default service account**.
3. Click **Create**.

Tip: Copy the **service account email** to your clipboard—you'll need it to assign IAM roles later.

Step 3: Create a Development Workspace

1. In your repository, click **Create Workspace**.
2. Provide a name and click **Create**.
3. You'll now see various tabs:

- **Code**
- **Compiled Graph**
- **Executions**
- **Start Execution**

Step 4: Initialize the Workspace

1. Initialize the workspace to generate the default file structure.
2. In the **Files pane**, you'll see folders like:
   - definitions – where you create SQLX files.
   - includes, tests, and dataform.json.

Example: A pre-created SQLX file in the definitions folder demonstrates how to create a view.

Step 5: Explore the dataform.json File

This configuration file sets up your project. It includes settings like:

Key	Description
projectId	Google Cloud project ID
defaultSchema	Schema prefix for your tables
tablePrefix	Optional prefix added to all table names
schemaSuffix	Used to organize environments (can be overridden)

Step 6: Create and Edit SQLX Files

Create a New SQLX File (View)

1. Click the dropdown next to the **Files pane**, then select **Create File**.
2. Name the file (e.g., quickstart_view.sqlx) and click **Create File**.
3. Add a SELECT statement to create a view.
4. Notice that the code compiles immediately in the **right-hand pane**.

Create Another SQLX File (Table)

1. Create a new file (e.g., quickstart_table.sqlx).
2. Write code to **create a table** using the previously created view as input.
3. The compiled SQL and object type (table) appear on the right.

Ignore the temporary error—permissions need to be configured first.

Step 7: Assign IAM Roles to the Dataform Service Account

1. Go to **IAM & Admin > IAM** in the navigation menu.
2. Click **Grant Access**.

3. Paste the **service account** copied earlier.
4. Assign the following roles:

Role	Purpose
BigQuery Job User	Allows running BigQuery jobs
BigQuery Data Editor	Grants write access to datasets
BigQuery Data Viewer	Grants read access to datasets

Step 8: Execute the SQL Workflow

1. Return to your **Dataform repository**, then go into the **development workspace**.
2. Click **Start Execution**, select **All Actions**, and then click **Start Execution**.
3. To view results:
    - Click the **Executions tab**, or
    - Click the **details link** at the bottom of the execution summary.

Step 9: Review Execution Logs and Outputs

- You'll see the list of SQLX files executed.
- Clicking **View Details** shows the execution status (e.g., Success) and details.
- Click the **object name** to open the generated table or view in BigQuery.

Example: The quickstart_table object is created and listed in your BigQuery dataset. Preview the table to see the rows.

Module – 7 Big Query Studio

Introduction to BigQuery Studio

Organizations around the world recognize that the shift to **AI and data-driven strategies** is already underway. Recent research shows that companies effectively using **data and AI** are:

- More **profitable** than their competitors
- Achieving improved **performance across key business metrics**

However, many organizations are still **struggling to unlock the full business value of data**. Even though **BigQuery** is part of a **unified, open, and intelligent data ecosystem**, data practitioners continue to face several challenges.

Challenges Faced by Data Practitioners

1. Tool Fragmentation

- Data teams often rely on **multiple disjointed tools**, causing frequent **context switching**, which reduces efficiency.

2. Complexity of SQL

- While **SQL is powerful**, it can be:
  - **Unintuitive**
  - **Time-consuming**
  - **Costly** to write and maintain
- On average:
  - **Data engineers and scientists** spend **25% to 50%** of their time writing SQL.
  - **Data analysts** often spend **even more time** writing SQL.

3. Barriers to Adopting Generative AI

- Leveraging new technologies like **Generative AI** traditionally requires:
  - Learning new **concepts and coding paradigms**
  - Managing increasing demands for **security, compliance, and governance**

BigQuery Studio is the **evolution of the BigQuery workspace**. It offers a **unified GUI experience** to help users:

- **Discover**, **explore**, **analyze**, and **predict** using data in BigQuery
- Perform **end-to-end tasks** in a single, purpose-built platform

Key Features of BigQuery Studio

BigQuery Studio integrates with the **broader Data Analytics + AI platform**, providing a unified experience across several capabilities:

1. Code Development Support

- **SQL and Python editing**
- **Notebook development** powered by **Colab Enterprise**

2. Resource Exploration and Discovery

- Powered by **Dataplex**, allowing users to easily discover and understand data assets

3. Centralized Source Control and Version History

- Built-in integration for **source control**
- Supports **CI/CD** and asset versioning

4. Collaborative Asset Management

- **Powered by Dataform**, enabling efficient **SQL pipeline creation and maintenance**

5. Assisted Data Experiences

- **Duet AI** provides:
  - **Contextual chat**
  - **Code assistance**
  - **Natural language query support**

Benefits of BigQuery Studio

- Offers a **single workspace** for:
  - **SQL scripts**
  - **Python scripts**
  - **Notebooks**
  - **Data pipelines**
- Enables **end-to-end tasks** such as:
  - **Data ingestion**
  - **Pipeline creation**
  - **Predictive analytics**
  - All in the **language of your choice**
- **Boosts collaboration** by:
  - Applying **software development best practices** to data analytics
  - Supporting **external code repository** integration to prevent code drift

AI-Powered Collaboration with Duet AI

BigQuery Studio includes **Duet AI**, an **AI-powered assistant** that enhances user productivity:

- **Understands context** of user data and environment
- Offers:
  - **Function suggestions**
  - **Auto-completion**
  - **Code block generation** for SQL and Python
- A **chat-based interface** lets users:
  - Ask questions in **natural language**
  - Get **real-time, personalized guidance**
  - Reduce reliance on documentation and trial-and-error

BigQuery Studio: A Unified Interface for Data-to-AI Workflows

With **BigQuery Studio**, you now have a **single interface** for all your **data-to-AI workflows**, combining the **scalability of BigQuery** with the power of a **data science notebook**.

Native Python Notebooks in BigQuery Studio

- **Python notebooks** are now **natively available** in BigQuery Studio.
- They appear in the **resource hierarchy** on the left pane alongside **SQL queries**.
- This enables **data scientists** and **analytics users** to work in a **familiar Colab notebook environment**—directly inside BigQuery—while operating at **petabyte scale**.

Key Features:

- **Dataset browsing**
- **Schema autocompletion**
- **Querying and transformation** of data
- Seamless transition to **Vertex AI** for:
    - **Model training**
    - **Customization**
    - **Deployment**
    - **MLOps**

BigFrames: Scalable Python on BigQuery

When working with large datasets, memory and performance can become a concern. **BigQuery Studio** addresses this through support for **BigFrames**.

What is BigFrames?

- **BigFrames** (BigQuery DataFrames) is an **open-source Python library**.
- It translates popular Python APIs such as **pandas** and **scikit-learn** into **BigQuery SQL**, enabling **scalable execution** on top of:
    - **BigQuery-managed storage**
    - **BigLake tables**

What It Enables:

- Python developers can:
    - **Discover**, **describe**, and **analyze** BigQuery data using a familiar API.
    - Work on **BigQuery-scale datasets** without worrying about memory limitations.
- Move from **exploration to production** using:
    - **BigQuery remote stored procedures**
    - **BigQuery ML (BQL) models**

    - **BigQuery remote functions**
- All actions are performed via the **BigFrames API** and follow the **BigQuery permissions model**, ensuring consistent and secure access.

Extensibility:

- Partners can **extend BigQuery functionality** for their specific applications.
- **Spark jobs on BigQuery** offer additional integration, while **open-source-first customers** can continue using **Dataproc** as an alternative.

Enhanced Usability Features in BigQuery Studio

BigQuery Studio includes several new **interface enhancements** that improve productivity:

1. Easier Resource Access

- Quickly retrieve **recently accessed resources**, such as notebooks.

2. Enhanced Query Result Handling

- **Sort query results** directly within the interface.
- View **estimated cost** for sort operations before execution.

3. Data Visualization

- Generate **various chart types** directly from query results for visual insights.

Asset Management

Enhancing Asset and Code Management for Modern Data Workflows

In this lesson, we explore how BigQuery integrates with **DataPlex** and **Dataform** to enhance asset and code management for data practitioners. These integrations enable enterprises to work more efficiently across distributed data environments while maintaining governance, quality, and collaboration.

Why DataPlex Matters

Many enterprises operate in complex data environments where information is scattered across data lakes, warehouses, and marts. **DataPlex** helps unify this ecosystem by allowing organizations to **discover, curate, unify, and manage data** without physically moving it. It enables teams to organize data based on business needs while enforcing centralized policies for metadata, security, classification, and lifecycle management.

Data Exploration and Governance in BigQuery Studio

With BigQuery's integration into DataPlex, users now benefit from powerful features directly in the BigQuery Studio interface:

- **Data Profiling**: Offers statistical summaries and trends that help users understand their data in greater depth. You can identify data drifts and gain insights right within the BigQuery UI.
- **Data Quality Monitoring**: Through the *Data Quality* tab, you can define and apply validation rules, trigger alerts, and monitor metrics at the table level to build trust in your data pipelines.
- **Data Lineage**: Provides end-to-end tracking of data sources and destinations across BigQuery, BigLake, and BigQuery Omni. This visibility supports impact analysis, issue tracing, and policy compliance.

Together, these features guide users from initial discovery through data understanding, issue identification, and resolution.

Quick Overview of Capabilities

Feature	Benefit
Data Profiling	Understand data trends and anomalies through detailed statistics
Data Quality	Enforce rule-based checks and view metrics directly in BigQuery UI
Data Lineage	Visualize data flow to understand origin, transformation, and impact

Dataform Integration for Code and Asset Management

BigQuery's integration with **Dataform** brings structured code management capabilities to SQL development. Dataform allows teams to write, organize, and version control SQLX files using Git integration—similar to how software developers manage application code.

This integration supports:

- **Source Control**: Version and share SQL code across teams
- **Project Structuring**: Maintain clean, modular projects for repeatability
- **Collaboration**: Teams can review, test, and deploy SQL pipelines with CI/CD practices

Code Management Features at a Glance

Feature	Description
Version Control	Track changes and revert as needed using Git
Collaboration	Share SQL logic across teams in a controlled environment
Governance	Enforce consistent standards for pipeline development

Duet AI Integration in BigQuery Studio

AI-Powered Collaboration for Smarter, Faster Analytics

BigQuery Studio not only provides a powerful data workspace but also introduces an AI-powered collaborator—**Duet AI**—designed to streamline workflows through contextual chat and code assistance. Duet AI in BigQuery understands both the **user context** and **data context**, enabling it to auto-suggest SQL and Python functions and code blocks, making data analysis faster and more intuitive.

Duet AI: What It Is

Duet AI is an umbrella term used to describe Google's use of **generative AI** to deliver intelligent, contextual assistance across its cloud products—including Google Workspace and Google Cloud Platform. Built on **Google's large foundation models**, Duet AI is specially trained to boost productivity while maintaining **data privacy and security**.

Leveraging **Vertex AI**, Duet AI provides personal, always-on, generative assistance for a wide range of roles—developers, data scientists, and operators—integrated directly into tools such as BigQuery Studio.

For the scope of this course, we'll focus specifically on **Duet AI in BigQuery**, although Duet AI is also available across other Google Cloud services.

How Duet AI Enhances BigQuery Workflows

Duet AI enhances data workflows in BigQuery Studio through a series of intelligent, real-time capabilities:

1. Explaining SQL Queries in Plain Language

If you're working with SQL queries written by someone else, Duet AI can explain them in **simple English**, helping you understand the logic, syntax, schema, and business purpose of the query.

- **How to use**: Highlight the query and click the **"Explain"** button. An explanation appears in a panel on the right.

2. SQL Generation from Natural Language Prompts

Even if you have limited SQL knowledge or are unsure of the schema, Duet AI can generate SQL queries from natural language.

- **How to use**: Start a comment in the SQL editor with a #, then type your goal in plain English. Press **Enter**, and Duet AI will generate a SQL query.
- You can accept the suggestion with the **Tab** key or make edits before running it.

3. Contextual SQL Code Completion

Duet AI also helps you write SQL faster by **autocompleting code** as you type.

- For instance, if you can't recall the syntax for a function, Duet AI will suggest completions as you type, which you can accept or modify as needed.

Recap: Why BigQuery Studio Stands Out

Capability	Description
Unified Workspace	Offers both SQL and notebook interfaces in one place
Language Flexibility	Supports Python and SQL for data ingestion, transformation, and analytics
Data Exploration & Discovery	Enables schema browsing and metadata search, powered by Data X
Centralized Code Management	With Dataform integration for version control, asset management, and CI/CD
AI Assistance via Duet AI	Provides code explanations, generation, and intelligent completions

BigQuery Studio, with Duet AI, empowers data engineers, analysts, and scientists to **analyze, predict, and collaborate** more effectively in a scalable, governed, and intelligent environment.

BigQuery Studio Steps: Getting Started with the New UI

1. Launching BigQuery Studio

- In the **Cloud Console**, navigate to **BigQuery** and click **BigQuery Studio** to launch it.
- Upon launch, you'll see the **"Create SQL Query"** button.
- After enabling the required APIs, a second button will appear allowing you to **create Python notebooks**.

2. Using Duet AI in Cloud Console

Enabling Duet AI

- Start by enabling the **Cloud AI Companion API**.
- Once enabled, Duet AI becomes available for assistance via natural language prompts.

Asking Duet AI for Help

- Example Prompt: *"How do I get started with BigQuery?"*
    - Duet AI responds with steps such as:
        1. Start a Google Cloud Project.
        2. Enable the BigQuery API.
        3. Create a dataset.

Creating a Dataset

- Next to your project, follow the steps to:
  - **Create a Dataset**
  - Enter a **Dataset ID**
  - Leave other options as default
  - Click **Create Dataset**

Querying Available Data

- Ask Duet AI: *"How do I learn which datasets and tables are available to me in BigQuery?"*
- Duet AI provides guidance via:
  - **Console steps**
  - **API usage**
  - **Information schema views**
  - **Example SQL queries**

3. Using Duet AI in BigQuery Studio

Enabling Duet AI in BigQuery

- Join the **Trusted Tester Program**.
- Enable required APIs and grant permissions.
- Click the **Duet AI in BigQuery** button to explore features like:
  - **SQL Explanation**
  - **SQL Generation**
  - **SQL Completion**

4. SQL Explanation Feature

- Paste a complex SQL query into the query editor.
- Highlight it, right-click, and choose **"Explain current selection."**
- Duet AI provides:
  - A summary of what the query does.
  - Detailed breakdown:
    - Data sources used
    - Join conditions
    - Grouping and sorting
    - Limit clause
    - Final summary

5. SQL Generation from Natural Language

- Open the **SQL Generation widget**.

- Type your goal in **natural language** (e.g., "Get total sales by product").
- Click **Generate**.
- Review the generated SQL.
- Click **Insert** to move it to the query editor.
- The original prompt is displayed at the top.

Note: You may need to tweak column names (e.g., product_id might actually be id), which can be easily verified by checking the table schema.

6. Running Queries and Creating Charts

- Run the generated query.
- Once results populate, use the built-in chart feature to visualize results:
    - Example: Change a busy **line graph** to a **bar chart**.
    - Modify dimensions and settings to improve clarity.

7. Sorting Results in the UI

- Sort query results directly from the UI by clicking the column headers.
- BigQuery Studio also provides an **estimated cost** for the sort operation.

8. SQL Code Completion

- Begin typing a query in the editor.
- Duet AI offers **autocomplete suggestions**.
- Hit **Tab** to accept suggestions and continue editing as needed.
- Run the query once finalized.

9. Working with Python Notebooks

Using Vertex AI Notebooks

- Navigate to **Vertex AI > Colab Enterprise** in the Cloud Console.
- Enable required APIs.
- Create a new notebook and name it.

Accessing Notebooks in BigQuery Studio

- Return to BigQuery Studio.
- You will now see the option to **create Python notebooks** directly in the UI.
- Expand your project under **Notebooks** to:
    - Access the previously created notebook
    - Create and manage new notebooks
    - Share notebooks across teams
    - Access them both via BigQuery Studio and Vertex AI