Uber Data Analytics Project

→ **Date :** 18 August, 2024

→ **Reference :** https://youtu.be/WpQECq5Hx9g?si=kxJC1Y8_FBDdIZLM

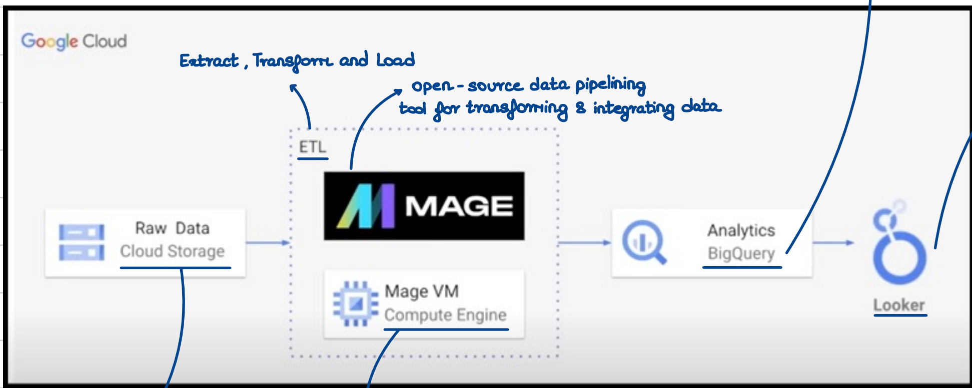
→ Architecture

Differences from Operational Databases

- **Operational Databases:** Used for day-to-day transaction processing (e.g., e-commerce transactions, inventory management). They are optimized for fast writes and real-time processing but may not perform as well with complex analytical queries.
- **Data Warehouses:** Designed for aggregating and analyzing large volumes of historical data. They support complex queries and reporting, and are typically read-heavy rather than write-heavy.

Google manages infrastructure, provisioning, backend, backups, patching, scaling ...

- Fully-managed, Serverless Data Warehouse → specialized database → querying & reporting
- SQL-based Querying — vs Traditional DBs optimized for transactional process
- Different from GCS (storing unstructured data) vs Analyzing & Querying Data



Extract, Transform and Load

open-source data pipelining tool for transforming & integrating data

Looker Studio (formerly known as Google Data Studio) is a data visualization and business intelligence (BI) tool provided by Google. It allows users to create interactive dashboards and reports that can visualize data from various sources. Here's a detailed overview of Looker Studio:

Key Features of Looker Studio

1. **Data Integration:**
   - **Connectors:** Looker Studio supports a wide range of data connectors, including Google products like Google Analytics, Google Ads, BigQuery, and Google Sheets, as well as third-party sources such as SQL databases, cloud storage, and APIs.
   - **Data Blending:** Users can combine data from different sources to create comprehensive reports and dashboards.
2. **Customizable Dashboards:**
   - **Interactive Elements:** Create interactive elements like filters, date range selectors, and dynamic controls to allow users to explore data in various ways.
   - **Visualizations:** Offers a variety of visualization types including charts, graphs, tables, and geo maps.
3. **Collaboration:**
   - **Sharing and Permissions:** Share reports and dashboards with other users and control access levels (view, edit, comment).
   - **Real-time Collaboration:** Multiple users can work on the same report simultaneously, similar to Google Docs.
4. **Data Transformation:**
   - **Calculated Fields:** Create new metrics and dimensions by applying custom calculations to existing data.
   - **Data Blending and Aggregation:** Combine and aggregate data from different sources to generate comprehensive insights.
5. **User-Friendly Interface:**
   - **Drag-and-Drop:** Provides an intuitive drag-and-drop interface for designing reports and dashboards without needing extensive technical knowledge.
   - **Templates:** Offers pre-built templates and themes to streamline report creation.
6. **Customization and Branding:**
   - **Custom Themes:** Customize the appearance of reports to match your brand's look and feel.
   - **Embedded Reporting:** Embed interactive reports and dashboards in websites or applications.

- Online file storage service
- Store and Retrieve Data from Cloud

- Service providing VMs for running applications & services
- Create, Configure and Manage VMs with various OS

→ Fact & Dimension Table

Fact Table:

- Contains quantitative measures or metrics that are used for analysis
- Typically contains foreign keys that link to dimension tables
- Contains columns that have high cardinality and change frequently
- Contains columns that are not useful for analysis by themselves, but are necessary for calculating metrics

Dimension Table:

- Contains columns that describe attributes of the data being analyzed
- Typically contains primary keys that link to fact tables
- Contains columns that have low cardinality and don't change frequently
- Contains columns that can be used for grouping or filtering data for analysis

https://builtin.com/articles/fact-table-vs-dimension-table

Fact Table vs. Dimension Table Defined

- **Fact table:** A fact table contains the primary keys of the referenced dimension tables along with some quantitative metrics. Examples of a fact table include customer orders or time-series financial data.
- **Dimension table:** A dimension table holds the descriptive information for the related fields that are in the fact table's records. It typically represents a physical entity like "customer" or "product."

→ Given Data : Taxi Trip Details

Sample : uber_data.csv

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_and_fwd_flag	dropoff_
0	1	2016-03-01 00:00:00	2016-03-01 00:07:55	1	2.50	-73.976746	40.765152	1	N	-
1	1	2016-03-01 00:00:00	2016-03-01 00:11:06	1	2.90	-73.983482	40.767925	1	N	-
2	2	2016-03-01 00:00:00	2016-03-01 00:31:06	2	19.98	-73.782021	40.644810	1	N	-
3	2	2016-03-01 00:00:00	2016-03-01 00:00:00	3	10.78	-73.863419	40.769814	1	N	-
4	2	2016-03-01 00:00:00	2016-03-01 00:00:00	5	30.43	-73.971741	40.792183	3	N	-

→ **Google Cloud Steps**

→ Google Cloud Storage → Create Bucket → Upload file → Fine-grained access
→ Edit Access → Public → URL

→ Compute Engine → Create Instance → Hardware + OS Configuration
→ Allow HTTP / HTTPS traffic
→ SSH Connect , Install Packages and Dependencies

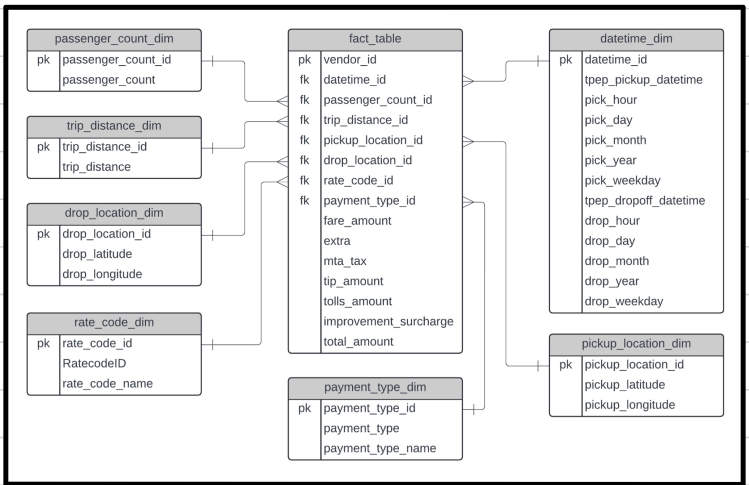
→ Virtual Private Cloud → Create Firewall Rule
→ Target, Source IP : 0.0.0.0/0 (open for any incoming traffic)
→ Open TCP port 6789 → Mage Access

→ Service Account → Create → Grant BigQuery Admin Role
→ Create Key → Download JSON file

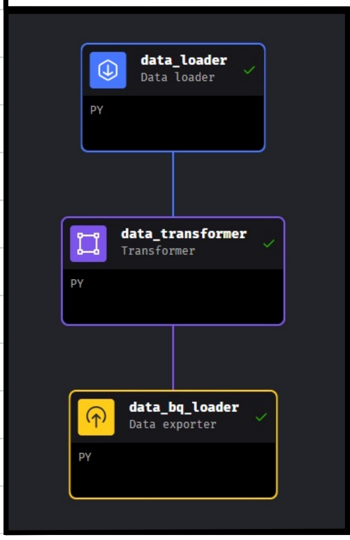
→ BigQuery → Create Dataset → Note Dataset ID

→ **Mage Pipeline** (Access : external_ip:6789)

Planned
Data Structure
Tranformation



Pipeline



→ Code Explanation

PY DATA LOADER data_loader ←× Edit parents

```python
import io
import pandas as pd
import requests
if 'data_loader' not in globals():
    from mage_ai.data_preparation.decorators import data_loader
if 'test' not in globals():
    from mage_ai.data_preparation.decorators import test


@data_loader
def load_data_from_api(*args, **kwargs):
    """
    Template for loading data from API
    """
    url = 'https://storage.googleapis.com/my_uber_data_analytics_project/uber_data.csv'
    response = requests.get(url)

    return pd.read_csv(io.StringIO(response.text), sep=',')


@test
def test_output(output, *args) -> None:
    """
    Template code for testing the output of the block.
    """
    assert output is not None, 'The output is undefined'
```

URL for Cloud stored data

PY TRANSFORMER data_transformer ←o 1 parent

```python
import pandas as pd
if 'transformer' not in globals():
    from mage_ai.data_preparation.decorators import transformer
if 'test' not in globals():
    from mage_ai.data_preparation.decorators import test


@transformer
def transform(df, *args, **kwargs):
    """
    Template code for a transformer block.

    Add more parameters to this function if this block has multiple parent blocks.
    There should be one parameter for each output variable from each parent block.

    Args:
        data: The output from the upstream parent block
        args: The output from any additional upstream blocks (if applicable)

    Returns:
        Anything (e.g. data frame, dictionary, array, int, str, etc.)
    """
    # Specify your transformation logic here

    # Converting objects to datetime for required fields

    df['tpep_pickup_datetime']  = pd.to_datetime(df['tpep_pickup_datetime'])
    df['tpep_dropoff_datetime'] = pd.to_datetime(df['tpep_dropoff_datetime'])

    df = df.drop_duplicates().reset_index(drop=True)
    df['trip_id'] = df.index
```

PY TRANSFORMER data_transformer ←o 1 parent

```python
    # Creating datetime_dim table

    datetime_dim = df[['tpep_pickup_datetime','tpep_dropoff_datetime']].reset_index(drop=True)
    datetime_dim['tpep_pickup_datetime'] = datetime_dim['tpep_pickup_datetime']
    datetime_dim['pick_hour']            = datetime_dim['tpep_pickup_datetime'].dt.hour
    datetime_dim['pick_day']             = datetime_dim['tpep_pickup_datetime'].dt.day
    datetime_dim['pick_month']           = datetime_dim['tpep_pickup_datetime'].dt.month
    datetime_dim['pick_year']            = datetime_dim['tpep_pickup_datetime'].dt.year
    datetime_dim['pick_weekday']         = datetime_dim['tpep_pickup_datetime'].dt.weekday

    datetime_dim['tpep_dropoff_datetime'] = datetime_dim['tpep_dropoff_datetime']
    datetime_dim['drop_hour']             = datetime_dim['tpep_dropoff_datetime'].dt.hour
    datetime_dim['drop_day']              = datetime_dim['tpep_dropoff_datetime'].dt.day
    datetime_dim['drop_month']            = datetime_dim['tpep_dropoff_datetime'].dt.month
    datetime_dim['drop_year']             = datetime_dim['tpep_dropoff_datetime'].dt.year
    datetime_dim['drop_weekday']          = datetime_dim['tpep_dropoff_datetime'].dt.weekday

    datetime_dim['datetime_id'] = datetime_dim.index
    # Ordering the Columns
    datetime_dim = datetime_dim[['datetime_id', 'tpep_pickup_datetime', 'pick_hour', 'pick_day', 'pick_month', 'pick_year', 'pick_weekday',
                                 'tpep_dropoff_datetime', 'drop_hour', 'drop_day', 'drop_month', 'drop_year', 'drop_weekday']]

    # Creating passenger_count_dim table

    passenger_count_dim = df[['passenger_count']].reset_index(drop=True)
    passenger_count_dim['passenger_count_id'] = passenger_count_dim.index
    passenger_count_dim = passenger_count_dim[['passenger_count_id','passenger_count']]

    # Creating trip_distance_dim table

    trip_distance_dim = df[['trip_distance']].reset_index(drop=True)
    trip_distance_dim['trip_distance_id'] = trip_distance_dim.index
    trip_distance_dim = trip_distance_dim[['trip_distance_id','trip_distance']]
```

PY TRANSFORMER data_transformer ←o 1 parent

```python
    # Rate Code Type Dictionary

    rate_code_type = {
        1:"Standard rate",
        2:"JFK",
        3:"Newark",
        4:"Nassau or Westchester",
        5:"Negotiated fare",
        6:"Group ride"
    }

    rate_code_dim = df[['RatecodeID']].reset_index(drop=True)
    rate_code_dim['rate_code_id']   = rate_code_dim.index
    rate_code_dim['rate_code_name'] = rate_code_dim['RatecodeID'].map(rate_code_type)
    rate_code_dim = rate_code_dim[['rate_code_id','RatecodeID','rate_code_name']]

    # Creating pickup_location_dim table

    pickup_location_dim = df[['pickup_longitude','pickup_latitude']].reset_index(drop=True)
    pickup_location_dim['pickup_location_id'] = pickup_location_dim.index
    pickup_location_dim = pickup_location_dim[['pickup_location_id','pickup_latitude','pickup_longitude']]

    # Creating dropoff_location_dim table

    dropoff_location_dim = df[['dropoff_longitude','dropoff_latitude']].reset_index(drop=True)
    dropoff_location_dim['dropoff_location_id'] = dropoff_location_dim.index
    dropoff_location_dim = dropoff_location_dim[['dropoff_location_id','dropoff_latitude','dropoff_longitude']]
```

PY TRANSFORMER data_transformer 1 parent

```python
    # Payment Type Dictionary

    payment_type_name = {
        1:"Credit card",
        2:"Cash",
        3:"No charge",
        4:"Dispute",
        5:"Unknown",
        6:"Voided trip"
    }

    # Creating payment_type_dim table

    payment_type_dim = df[['payment_type']].reset_index(drop=True)
    payment_type_dim['payment_type_id']   = payment_type_dim.index
    payment_type_dim['payment_type_name'] = payment_type_dim['payment_type_id'].map(payment_type_name)
    payment_type_dim = payment_type_dim[['payment_type_id','payment_type','payment_type_name']]

    fact_table = df.merge(passenger_count_dim, left_on='trip_id', right_on='passenger_count_id') \
                 .merge(trip_distance_dim     , left_on='trip_id', right_on='trip_distance_id') \
                 .merge(rate_code_dim         , left_on='trip_id', right_on='rate_code_id') \
                 .merge(pickup_location_dim   , left_on='trip_id', right_on='pickup_location_id') \
                 .merge(dropoff_location_dim  , left_on='trip_id', right_on='dropoff_location_id')\
                 .merge(datetime_dim          , left_on='trip_id', right_on='datetime_id') \
                 .merge(payment_type_dim      , left_on='trip_id', right_on='payment_type_id') \
                 [['trip_id','VendorID', 'datetime_id', 'passenger_count_id',
                'trip_distance_id', 'rate_code_id', 'store_and_fwd_flag', 'pickup_location_id', 'dropoff_location_id',
                'payment_type_id', 'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
                'improvement_surcharge', 'total_amount']]

    return {"datetime_dim":datetime_dim.to_dict(orient="dict"),
            "passenger_count_dim":passenger_count_dim.to_dict(orient="dict"),
            "trip_distance_dim":trip_distance_dim.to_dict(orient="dict"),
            "rate_code_dim":rate_code_dim.to_dict(orient="dict"),
            "pickup_location_dim":pickup_location_dim.to_dict(orient="dict"),
            "dropoff_location_dim":dropoff_location_dim.to_dict(orient="dict"),
            "payment_type_dim":payment_type_dim.to_dict(orient="dict"),
            "fact_table":fact_table.to_dict(orient="dict")}
```

PY DATA EXPORTER data_bq_loader 1 parent

```python
from mage_ai.data_preparation.repo_manager import get_repo_path
from mage_ai.io.bigquery import BigQuery
from mage_ai.io.config import ConfigFileLoader
from pandas import DataFrame
from os import path

if 'data_exporter' not in globals():
    from mage_ai.data_preparation.decorators import data_exporter


@data_exporter
def export_data_to_big_query(data, **kwargs) -> None:
    """
    Template for exporting data to a BigQuery warehouse.
    Specify your configuration settings in 'io_config.yaml'.

    Docs: https://docs.mage.ai/design/data-loading#bigquery

    """
    config_path = path.join(get_repo_path(), 'io_config.yaml')
    config_profile = 'default'

    for key, value in data.items():
        table_id = 'project-uberdataanalytics.Uber_Transformed_Dataset.{}'.format(key)
        BigQuery.with_config(ConfigFileLoader(config_path, config_profile)).export(
            DataFrame(value),
            table_id,
            if_exists='replace',  # Specify resolution policy if table name already exists
        )
```

→ SQL Query

SQL-Query   RUN   SAVE QUERY   DOWNLOAD   SHARE   SCHEDULE   MORE

```sql
CREATE OR REPLACE TABLE `project-uberdataanalytics.Uber_Transformed_Dataset.AnalysisReport` AS (
 SELECT
   f.VendorID,
   dt.tpep_pickup_datetime,
   dt.tpep_dropoff_datetime,
   p.passenger_count,
   td.trip_distance,
   rc.RatecodeID,
   rc.rate_code_name,
   f.store_and_fwd_flag,
   pl.pickup_latitude,
   pl.pickup_longitude,
   dl.dropoff_latitude,
   dl.dropoff_longitude,
   pt.payment_type,
   f.fare_amount,
   f.extra,
   f.mta_tax,
   f.tip_amount,
   f.tolls_amount,
   f.improvement_surcharge,
   f.total_amount
 FROM
   `project-uberdataanalytics.Uber_Transformed_Dataset.fact_table` f
   JOIN `project-uberdataanalytics.Uber_Transformed_Dataset.passenger_count_dim` p
     ON f.passenger_count_id = p.passenger_count_id
   JOIN `project-uberdataanalytics.Uber_Transformed_Dataset.trip_distance_dim` td
     ON f.trip_distance_id = td.trip_distance_id
   JOIN `project-uberdataanalytics.Uber_Transformed_Dataset.rate_code_dim` rc
     ON f.rate_code_id = rc.rate_code_id
   JOIN `project-uberdataanalytics.Uber_Transformed_Dataset.pickup_location_dim` pl
     ON f.pickup_location_id = pl.pickup_location_id
   JOIN `project-uberdataanalytics.Uber_Transformed_Dataset.dropoff_location_dim` dl
     ON f.dropoff_location_id = dl.dropoff_location_id
   JOIN `project-uberdataanalytics.Uber_Transformed_Dataset.payment_type_dim` pt
     ON f.payment_type_id = pt.payment_type_id
   JOIN `project-uberdataanalytics.Uber_Transformed_Dataset.datetime_dim` dt
     ON f.datetime_id = dt.datetime_id
);
```

→ Data Visualization Tool – Looker Studio

https://lookerstudio.google.com/u/0/reporting/20849a3e-fcc5-4455-9942-291da35781d4/page/l2C9D