Travel & Tourism Data Warehouse Project Report

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3355315970359751/1092239607842669/305967476956713/latest.html

1. Project Overview and Objective

The Travel & Tourism Data Warehouse project was designed to integrate and analyze diverse datasets related to tourist demographics, flight bookings, hotel stays, and attraction visits. The primary goal was to create a scalable, efficient data warehouse that enables analytical reporting, business intelligence (BI), and informed decision-making for stakeholders in the tourism industry.

By leveraging PySpark for ETL processes and implementing Normalized and Dimensional Models (Star Schema with Slowly Changing Dimensions - SCD Type 2), the project ensures data quality, integrity, and performance.

Project Approach and Implementation

1. Data Sources and Collection

Each data source captures an essential aspect of the tourist's journey, allowing analysts to gain a 360-degree view of the customer experience. Understanding where tourists are traveling, staying, and what attractions they visit provides critical insights for improving services, optimizing offerings, and increasing profitability.

- Datasets Used:
    - Tourist Demographics (JSON)
    - Flight Bookings (CSV)
    - Hotel Stays (CSV)
    - Attractions Visited (CSV)
- All datasets were synthetically generated using the Faker library in Python to simulate realistic data.

Flight Bookings Dataset

- **File Name:** flight_bookings.csv
- **Format:** CSV
- **Records:** 1,000+
- **Primary Key:** booking_id
- **Description:** This dataset captures flight reservation details made by tourists. Each booking includes essential travel details such as departure and arrival cities, airlines, ticket prices, and travel dates. The table provides valuable insights into travel routes, popular destinations, and tourist spending on air travel.
- **Relevance:** Enables analysis of customer preferences for airlines, routes, and departure/arrival trends. It helps in understanding travel flows and can assist in optimizing airline partnerships and route planning.

Attributes:

- booking_id (INTEGER): Unique identifier for each flight booking
- customer_id (INTEGER): Links to the Tourist Demographics dataset
- flight_number (TEXT): Alphanumeric flight number assigned to the flight
- airline (TEXT): Name of the airline
- origin_city (TEXT): Departure city for the flight
- destination_city (TEXT): Arrival city for the flight
- departure_date (DATE): Date when the flight departs
- arrival_date (DATE): Date when the flight arrives
- booking_date (DATE): Date when the flight was booked
- ticket_price (DECIMAL): Price paid for the flight ticket

Hotel Stays Dataset

- **File Name:** hotel_stays.csv
- **Format:** CSV
- **Records:** 1,000+
- **Primary Key:** booking_id
- **Description:** This dataset contains hotel reservation records, capturing the details of a tourist's accommodation, including location, room type, and total stay amount. It helps track where tourists are staying and how much they are spending on accommodation.

- **Relevance:** Provides insights into customer accommodation preferences, room occupancy rates, and hotel popularity by location. Essential for hospitality strategy, pricing, and partnership decisions.

Attributes:

- booking_id (INTEGER): Unique identifier for each hotel booking
- customer_id (INTEGER): Links to the Tourist Demographics dataset
- hotel_name (TEXT): Name of the hotel
- city (TEXT): City where the hotel is located
- room_type (TEXT): Type of room booked (Single, Double, Suite)
- check_in_date (DATE): Date when the customer checks into the hotel
- check_out_date (DATE): Date when the customer checks out of the hotel
- booking_date (DATE): Date when the hotel booking was made
- total_amount (DECIMAL): Total amount paid for the hotel stay

Tourist Demographics Dataset

- **File Name:** tourist_demographics.json
- **Format:** JSON
- **Records:** 1,000+
- **Primary Key:** customer_id
- **Description:** This dataset holds personal and demographic information about tourists. It provides the context for analyzing tourist behavior and preferences, allowing segmentation by age, gender, nationality, and travel purpose.
- **Relevance:** Forms the basis of customer profiling and segmentation strategies. It enables targeted marketing campaigns and personalization of travel offerings based on demographic attributes.

Attributes:

- customer_id (INTEGER): Unique identifier for the tourist
- first_name (TEXT): First name of the tourist
- last_name (TEXT): Last name of the tourist
- gender (TEXT): Gender of the tourist (Male, Female, Other)
- age (INTEGER): Age of the tourist
- nationality (TEXT): Country of origin for the tourist
- travel_purpose (TEXT): Purpose of the tourist's travel (Leisure, Business, Education, Medical)

Attractions Visited Dataset

- **File Name:** attractions_visited.csv
- **Format:** CSV
- **Records:** 1,000+
- **Primary Key:** attraction_id
- **Description:** This dataset tracks visits to popular tourist attractions, detailing the number of visitors, revenue generated, and ratings. It offers an understanding of tourist preferences and the popularity of different destinations.
- **Relevance:** Helps identify top-performing attractions and understand visitor preferences. Crucial for destination management, marketing, and improving tourist experiences at attractions.

Attributes:

- attraction_id (INTEGER): Unique identifier for the attraction record
- attraction_name (TEXT): Name of the attraction visited
- city (TEXT): City where the attraction is located
- visit_date (DATE): Date when the attraction was visited
- visitors_count (INTEGER): Number of visitors to the attraction on that date
- revenue (DECIMAL): Total revenue generated from visitors on that date
- average_rating (DECIMAL): Average visitor rating for the attraction (scale of 1.0 to 5.0)

Relationships and Significance

These relationships enable powerful insights into traveler behavior, preferences, and trends, making it possible to optimize marketing, enhance customer experience, and improve operational efficiency.

- **customer_id:** A common key that connects Flight Bookings, Hotel Stays, and Tourist Demographics. It allows the creation of a full tourist profile for personalized insights.
- **city:** Present in Hotel Stays and Attractions Visited. Useful for destination-based analysis and cross-referencing tourist accommodation with nearby attractions.
- **visit_date, booking_date, departure_date:** Time dimensions that enable trend and seasonality analysis across flights, hotels, and attractions.

I used jupyter notebook to create data source from faker -

```
pip install faker pandas numpy
```

```
Collecting faker
  Downloading faker-37.0.0-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.12/site-packages (2.2.2)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.12/site-packages (1.26.4)
Requirement already satisfied: tzdata in /opt/anaconda3/lib/python3.12/site-packages (from faker) (2023.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/anaconda3/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/anaconda3/lib/python3.12/site-packages (from pandas) (2024.1)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.12/site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Downloading faker-37.0.0-py3-none-any.whl (1.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 6.0 MB/s eta 0:00:0000:0100:01
Installing collected packages: faker
Successfully installed faker-37.0.0
Note: you may need to restart the kernel to use updated packages.
```

Created the first data source – flight_booking.csv

```
# ------------------------------------------------------------------------------
# 1. Flight Bookings CSV (1000 records)
flight_records = []
for i in range(1, 1001):
    customer_id = randint(1, 1000)
    origin = choice(cities)
    destination = choice([city for city in cities if city != origin])
    booking_date = fake.date_between(start_date='-1y', end_date='today')

    # Departure within 60 days of booking
    departure_date = fake.date_between(start_date=booking_date, end_date=booking_date + timedelta(days=
    arrival_date = departure_date  # same day arrival for simplicity

    record = [
        i,
        customer_id,
        f"FL{randint(1000,9999)}",
        choice(airlines),
        origin,
        destination,
        departure_date,
        arrival_date,
        booking_date,
        round(np.random.uniform(100, 1500), 2)
    ]
    flight_records.append(record)

df_flight = pd.DataFrame(flight_records, columns=[
    'booking_id', 'customer_id', 'flight_number', 'airline',
    'origin_city', 'destination_city', 'departure_date',
    'arrival_date', 'booking_date', 'ticket_price'
])

flight_file_path = os.path.join(downloads_path, 'flight_bookings.csv')
df_flight.to_csv(flight_file_path, index=False)
print(f"✅ Flight Bookings saved to {flight_file_path}")
```

Created the second data source – hotel_stays.csv

```
# ------------------------------------------------------------------------------
# 2. Hotel Stays CSV (1000 records)
hotel_records = []
for i in range(1, 1001):
    customer_id = randint(1, 1000)
    city = choice(cities)
    booking_date = fake.date_between(start_date='-1y', end_date='today')

    # Check-in within 30 days after booking
    check_in = fake.date_between(start_date=booking_date, end_date=booking_date + timedelta(days=30))

    # Stay duration between 1 to 10 days
    stay_duration = randint(1, 10)
    check_out = check_in + timedelta(days=stay_duration)

    record = [
        i,
        customer_id,
        choice(hotels),
        city,
        choice(room_types),
        check_in,
        check_out,
        booking_date,
        round(np.random.uniform(50, 1000), 2)
    ]
    hotel_records.append(record)

df_hotel = pd.DataFrame(hotel_records, columns=[
    'booking_id', 'customer_id', 'hotel_name', 'city',
    'room_type', 'check_in_date', 'check_out_date',
    'booking_date', 'total_amount'
])

hotel_file_path = os.path.join(downloads_path, 'hotel_stays.csv')
df_hotel.to_csv(hotel_file_path, index=False)
print(f"✅ Hotel Stays saved to {hotel_file_path}")
```

Created the third data source – tourist_demographics.json

```
# ------------------------------------------------------------------------------
# 3. Tourist Demographics JSON (1000 records)
demographic_records = []
for i in range(1, 1001):
    record = {
        "customer_id": i,
        "first_name": fake.first_name(),
        "last_name": fake.last_name(),
        "gender": choice(['Male', 'Female', 'Other']),
        "age": randint(18, 70),
        "nationality": fake.country(),
        "travel_purpose": choice(purposes)
    }
    demographic_records.append(record)

demographics_file_path = os.path.join(downloads_path, 'tourist_demographics.json')
df_demographics = pd.DataFrame(demographic_records)
df_demographics.to_json(demographics_file_path, orient='records', indent=4)
print(f"✅ Tourist Demographics saved to {demographics_file_path}")
```

Created the fourth data source -

```
# ------------------------------------------------------------------------------
# 4. Attractions Visited CSV (1000 records)
attraction_records = []
for i in range(1, 1001):
    attraction_name = choice(attractions)

    city_map = {
        'Eiffel Tower': 'Paris',
        'Great Wall': 'Beijing',
        'Statue of Liberty': 'New York',
        'Sydney Opera House': 'Sydney',
        'Burj Khalifa': 'Dubai'
    }

    city = city_map[attraction_name]
    visit_date = fake.date_between(start_date='-1y', end_date='today')

    visitors_count = randint(50, 1000)
    revenue = visitors_count * round(np.random.uniform(10, 100), 2)
    avg_rating = round(np.random.uniform(3, 5), 2)

    record = [
        i,
        attraction_name,
        city,
        visit_date,
        visitors_count,
        revenue,
        avg_rating
    ]
    attraction_records.append(record)

df_attractions = pd.DataFrame(attraction_records, columns=[
    'attraction_id', 'attraction_name', 'city',
    'visit_date', 'visitors_count', 'revenue', 'average_rating'
])

attractions_file_path = os.path.join(downloads_path, 'attractions_visited.csv')
df_attractions.to_csv(attractions_file_path, index=False)
print(f"✅ Attractions Visited saved to {attractions_file_path}")
```

Data Dictionary -

data_dictionary

Table Name	Field Name	Data Type	Description
flight_bookings	booking_id	INTEGER	Unique identifier for each flight booking
flight_bookings	customer_id	INTEGER	Unique identifier for the customer making the booking
flight_bookings	flight_number	TEXT	Alphanumeric flight number assigned to the flight
flight_bookings	airline	TEXT	Name of the airline
flight_bookings	origin_city	TEXT	Departure city for the flight
flight_bookings	destination_city	TEXT	Arrival city for the flight
flight_bookings	departure_date	DATE	Date when the flight departs
flight_bookings	arrival_date	DATE	Date when the flight arrives
flight_bookings	booking_date	DATE	Date when the flight was booked
flight_bookings	ticket_price	DECIMAL(10,2)	Price paid for the flight ticket
hotel_stays	booking_id	INTEGER	Unique identifier for each hotel booking
hotel_stays	customer_id	INTEGER	Unique identifier for the customer making the hotel reservation
hotel_stays	hotel_name	TEXT	Name of the hotel
hotel_stays	city	TEXT	City where the hotel is located
hotel_stays	room_type	TEXT	Type of room booked (Single, Double, Suite)
hotel_stays	check_in_date	DATE	Date when the customer checks into the hotel
hotel_stays	check_out_date	DATE	Date when the customer checks out of the hotel
hotel_stays	booking_date	DATE	Date when the hotel booking was made
hotel_stays	total_amount	DECIMAL(10,2)	Total amount paid for the hotel stay
tourist_demographics	customer_id	INTEGER	Unique identifier for the tourist
tourist_demographics	first_name	TEXT	First name of the tourist
tourist_demographics	last_name	TEXT	Last name of the tourist
tourist_demographics	gender	TEXT	Gender of the tourist (Male, Female, Other)
tourist_demographics	age	INTEGER	Age of the tourist
tourist_demographics	nationality	TEXT	Country of origin for the tourist
tourist_demographics	travel_purpose	TEXT	Purpose of the tourist's travel (Leisure, Business, Education, Medical)
attractions_visited	attraction_id	INTEGER	Unique identifier for the attraction record
attractions_visited	attraction_name	TEXT	Name of the attraction visited
attractions_visited	city	TEXT	City where the attraction is located
attractions_visited	visit_date	DATE	Date when the attraction was visited
attractions_visited	visitors_count	INTEGER	Number of visitors to the attraction on that date
attractions_visited	revenue	DECIMAL(10,2)	Total revenue generated from visitors on that date
attractions_visited	average_rating	DECIMAL(3,2)	Average visitor rating for the attraction (scale of 1.0 to 5.0)

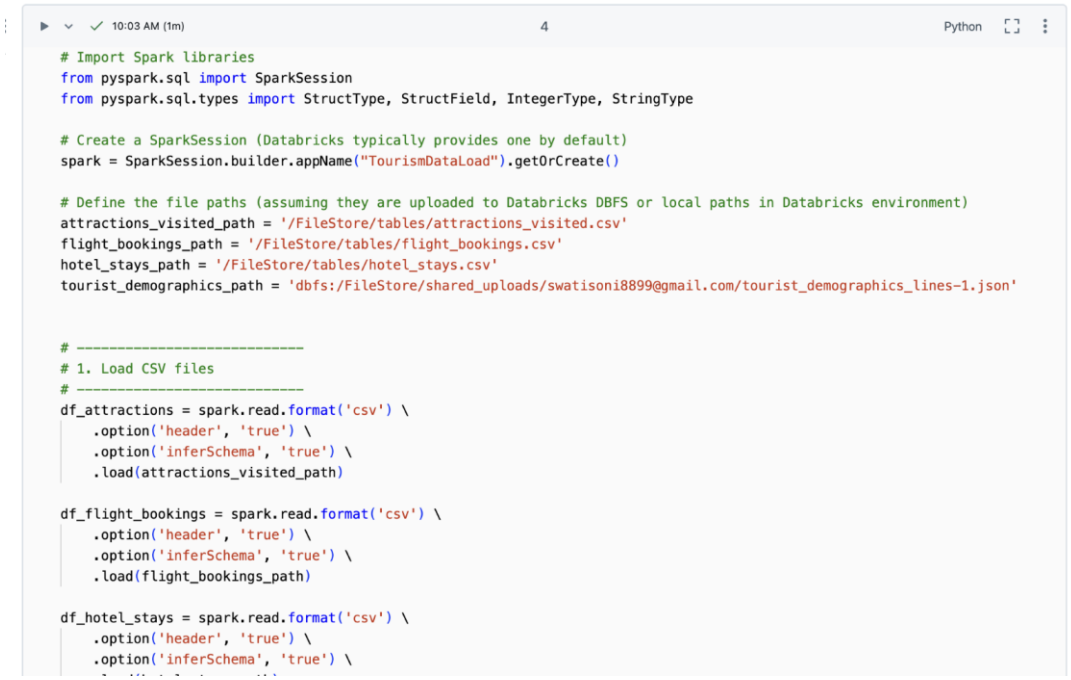
2. Data Preprocessing and Normalization

Normalized Database Layer within the Data Warehousing Solution for the tourism dataset. It serves as the foundational layer, ensuring data quality and integrity through normalization techniques (up to 3rd Normal Form).

The normalized schema structures data from four distinct raw sources into clean, organized tables that minimize redundancy, enforce data consistency, and prepare the data for dimensional modeling and analytical querying in subsequent stages.

- Data normalized to Third Normal Form (3NF) to reduce redundancy and ensure referential integrity.
- Separate dimension and fact tables created for cities, hotels, nationalities, tourists, etc.
- Surrogate keys generated using monotonically_increasing_id() in PySpark.

Code Snippet: Extract Raw Data



```
10:03 AM (1m)                                4                                Python

# Import Spark libraries
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

# Create a SparkSession (Databricks typically provides one by default)
spark = SparkSession.builder.appName("TourismDataLoad").getOrCreate()

# Define the file paths (assuming they are uploaded to Databricks DBFS or local paths in Databricks environment)
attractions_visited_path = '/FileStore/tables/attractions_visited.csv'
flight_bookings_path = '/FileStore/tables/flight_bookings.csv'
hotel_stays_path = '/FileStore/tables/hotel_stays.csv'
tourist_demographics_path = 'dbfs:/FileStore/shared_uploads/swatisoni8899@gmail.com/tourist_demographics_lines-1.json'


# -----------------------------
# 1. Load CSV files
# -----------------------------
df_attractions = spark.read.format('csv') \
    .option('header', 'true') \
    .option('inferSchema', 'true') \
    .load(attractions_visited_path)

df_flight_bookings = spark.read.format('csv') \
    .option('header', 'true') \
    .option('inferSchema', 'true') \
    .load(flight_bookings_path)

df_hotel_stays = spark.read.format('csv') \
    .option('header', 'true') \
    .option('inferSchema', 'true') \
```

```
df_flight_bookings = spark.read.format('csv') \
    .option('header', 'true') \
    .option('inferSchema', 'true') \
    .load(flight_bookings_path)

df_hotel_stays = spark.read.format('csv') \
    .option('header', 'true') \
    .option('inferSchema', 'true') \
    .load(hotel_stays_path)

# -----------------------------
# 2. Define Schema for JSON and Load
# -----------------------------
demographics_schema = StructType([
    StructField("customer_id", IntegerType(), True),
    StructField("first_name", StringType(), True),
    StructField("last_name", StringType(), True),
    StructField("gender", StringType(), True),
    StructField("age", IntegerType(), True),
    StructField("nationality", StringType(), True),
    StructField("travel_purpose", StringType(), True)
])

df_tourist_demographics = spark.read \
    .schema(demographics_schema) \
    .json(tourist_demographics_path)

# -----------------------------
```

```
# ------------------------------
# 3. Save DataFrames as Delta Tables
# ------------------------------

df_attractions.write.format("delta").mode("overwrite").saveAsTable("attractions_visited")
df_flight_bookings.write.format("delta").mode("overwrite").saveAsTable("flight_bookings")
df_hotel_stays.write.format("delta").mode("overwrite").saveAsTable("hotel_stays")
df_tourist_demographics.write.format("delta").mode("overwrite").saveAsTable("tourist_demographics")

# ------------------------------
# 4. Verify the tables
# ------------------------------

# Show all tables in your current database
spark.sql("SHOW TABLES").show()

# Optional: Preview the data in tables
spark.sql("SELECT * FROM attractions_visited LIMIT 10").show()
spark.sql("SELECT * FROM flight_bookings LIMIT 10").show()
spark.sql("SELECT * FROM hotel_stays LIMIT 10").show()
spark.sql("SELECT * FROM tourist_demographics LIMIT 10").show()
```

▸ (56) Spark Jobs

Final Output -

```
# Show all tables in your current database


# Optional: Preview the data in tables
spark.sql("SELECT * FROM attractions_visited LIMIT 10").show()
spark.sql("SELECT * FROM flight_bookings LIMIT 10").show()
spark.sql("SELECT * FROM hotel_stays LIMIT 10").show()
spark.sql("SELECT * FROM tourist_demographics LIMIT 10").show()
```

▸ (38) Spark Jobs

▸ df_attractions: pyspark.sql.dataframe.DataFrame = [attraction_id: integer, attraction_name: string ... 5 more fields]
▸ df_flight_bookings: pyspark.sql.dataframe.DataFrame = [booking_id: integer, customer_id: integer ... 8 more fields]
▸ df_hotel_stays: pyspark.sql.dataframe.DataFrame = [booking_id: integer, customer_id: integer ... 7 more fields]
▸ df_tourist_demographics: pyspark.sql.dataframe.DataFrame = [customer_id: integer, first_name: string ... 5 more fields]

```
|          7|         388| Oceanview Resort| Toronto|    Suite|   2024-03-21|    2024-03-24|  2024-03-13|      643.37|
|          8|         732| Oceanview Resort| Toronto|   Double|   2025-02-21|    2025-02-28|  2025-02-18|      814.98|
|          9|         192| Oceanview Resort|New York|    Suite|   2024-07-23|    2024-07-28|  2024-07-04|      291.78|
|         10|         959|City Lights Hotel|New York|    Suite|   2024-08-24|    2024-08-25|  2024-08-19|      920.48|
+-----------+------------+-----------------+--------+---------+-------------+--------------+------------+------------+


+-----------+----------+---------+------+---+--------------------+--------------+
|customer_id|first_name|last_name|gender|age|         nationality|travel_purpose|
+-----------+----------+---------+------+---+--------------------+--------------+
|          1|      John| Phillips|  Male| 55|              Sweden|     Education|
|          2|     David|   Sawyer|Female| 55|             Belarus|      Business|
|          3|      Juan|     Bell|  Male| 56|           Greenland|      Business|
|          4|     Paula|  Sherman|Female| 46|         El Salvador|       Medical|
|          5|   Anthony|   Briggs|Female| 27|         Saint Lucia|       Medical|
|          6|    Tyrone|  Roberts|  Male| 52|        Sierra Leone|     Education|
|          7|      John|  Jackson|Female| 60|   Wallis and Futuna|       Medical|
|          8|  Jonathan|   Jacobs|Female| 33|               Niger|     Education|
|          9|     Steve|    Lyons|Female| 61|          Uzbekistan|     Education|
|         10|     Jacob|  Donovan|  Male| 40|Palestinian Terri...|       Medical|
+-----------+----------+---------+------+---+--------------------+--------------+
```

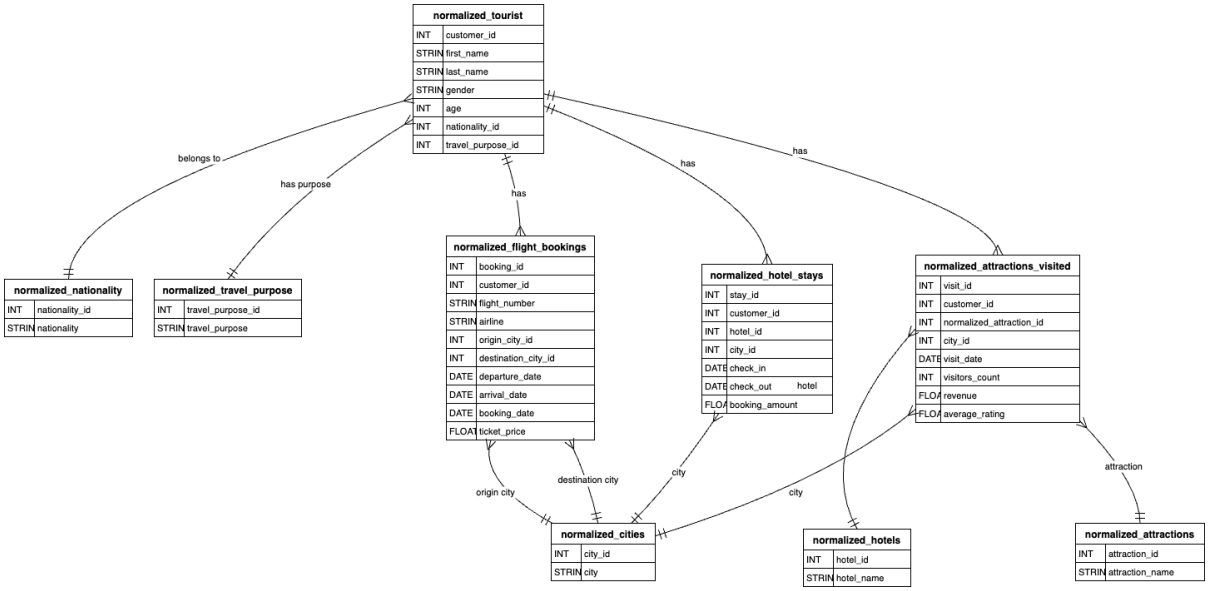
Normalized Schema Design

We applied normalization up to the Third Normal Form (3NF), systematically organizing raw data from heterogeneous sources (CSV and JSON) into clean, structured tables. Dimension entities such as nationalities, cities, hotels, and attractions were separated into distinct lookup tables, each assigned surrogate primary keys. Fact tables—capturing transactional events like flight bookings and hotel stays—were linked to these dimensions through foreign keys. This approach minimized data duplication, ensured consistency across datasets, and laid a robust foundation for subsequent dimensional modeling and analytical querying.

Entity-Relationship (ER) Diagram

The diagram shows the relationships between dimension tables (such as cities, nationalities) and fact tables (tourists, flight bookings).

ER Diagram



Code Snippet: Normalization

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import trim, monotonically_increasing_id, col

# Initialize Spark Session
spark = SparkSession.builder.appName("Normalization3NF").getOrCreate()

# ---------------------------
# Load Raw Datasets (Staging Layer)
# ---------------------------
tourist_df = spark.read.json('dbfs:/FileStore/shared_uploads/swatisoni8899@gmail.com/tourist_demographics_lines-1.json')
flight_bookings_df = spark.read.option("header", "true").option("inferSchema", "true").csv('/FileStore/tables/flight_bookings.csv')
hotel_stays_df = spark.read.option("header", "true").option("inferSchema", "true").csv('/FileStore/tables/hotel_stays.csv')
attractions_visited_df = spark.read.option("header", "true").option("inferSchema", "true").csv('/FileStore/tables/attractions_visited.csv')

# ---------------------------
# (Normalized)
# ---------------------------
```

```
# Nationality Dimension
nationality_df = tourist_df.select(trim(col("nationality")).alias("nationality")).distinct() \
.withColumn("nationality_id", monotonically_increasing_id())

# Travel Purpose Dimension
travel_purpose_df = tourist_df.select(trim(col("travel_purpose")).alias("travel_purpose")).distinct() \
.withColumn("travel_purpose_id", monotonically_increasing_id())

# Cities Dimension
cities_from_flights = flight_bookings_df.select(trim(col("origin_city")).alias("city")).union(
flight_bookings_df.select(trim(col("destination_city")).alias("city"))
).distinct()

cities_from_hotels = hotel_stays_df.select(trim(col("city")).alias("city")).distinct()
cities_from_attractions = attractions_visited_df.select(trim(col("city")).alias("city")).distinct()

cities_df = cities_from_flights.union(cities_from_hotels).union(cities_from_attractions).distinct() \
.withColumn("city_id", monotonically_increasing_id())

# Hotels Dimension
hotels_df = hotel_stays_df.select(trim(col("hotel_name")).alias("hotel_name")).distinct() \
.withColumn("hotel_id", monotonically_increasing_id())

# Attractions Dimension
attractions_df = attractions_visited_df.select(trim(col("attraction_name")).alias("attraction_name")).distinct() \
.withColumn("attraction_id", monotonically_increasing_id())

# --------------------------
# Tourist Table (Normalized)
# --------------------------
tourist_clean_df = tourist_df.alias("tourist") \
.join(nationality_df.alias("nationality"), trim(col("tourist.nationality")) == col("nationality.nationality"), "left") \
.join(travel_purpose_df.alias("purpose"), trim(col("tourist.travel_purpose")) == col("purpose.travel_purpose"),
"left") \
.select(
col("tourist.customer_id"),
trim(col("tourist.first_name")).alias("first_name"),
trim(col("tourist.last_name")).alias("last_name"),
col("tourist.gender"),
col("tourist.age"),
col("nationality.nationality_id"),
col("purpose.travel_purpose_id")
)

# --------------------------
# Flight Bookings Table (Normalized)
# --------------------------
```

```
cities_df_origin = cities_df.alias("origin_city")
cities_df_destination = cities_df.alias("destination_city")

flight_bookings_clean_df = flight_bookings_df.alias("flight") \
.join(cities_df_origin, trim(col("flight.origin_city")) == trim(col("origin_city.city")), "left") \
.join(cities_df_destination, trim(col("flight.destination_city")) == trim(col("destination_city.city")), "left") \
.select(
col("flight.booking_id"),
col("flight.customer_id"),
col("flight.flight_number"),
col("flight.airline"),
col("origin_city.city_id").alias("origin_city_id"),
col("destination_city.city_id").alias("destination_city_id"),
col("flight.departure_date"),
col("flight.arrival_date"),
col("flight.booking_date"),
col("flight.ticket_price")
)

# ---------------------------
# Hotel Stays Table (Normalized)
# ---------------------------
hotels_df_alias = hotels_df.alias("hotels")
cities_df_alias = cities_df.alias("hotel_city")

hotel_stays_clean_df = hotel_stays_df.alias("hotel_stay") \
.join(hotels_df_alias, trim(col("hotel_stay.hotel_name")) == trim(col("hotels.hotel_name")), "left") \
.join(cities_df_alias, trim(col("hotel_stay.city")) == trim(col("hotel_city.city")), "left") \
.select(
col("hotel_stay.booking_id").alias("stay_id"), # Renamed booking_id to stay_id
col("hotel_stay.customer_id"),
col("hotels.hotel_id"),
col("hotel_city.city_id"),
col("hotel_stay.check_in_date").alias("check_in"), # Aliased column for clarity
col("hotel_stay.check_out_date").alias("check_out"), # Aliased column for clarity
col("hotel_stay.total_amount").alias("booking_amount") # Aliased column for clarity
)

# ---------------------------
# Attractions Visited Table (Normalized)
# ---------------------------
# Add surrogate visit_id column (since the raw data doesn't have one)
attractions_visited_df = attractions_visited_df \
.withColumn("visit_id", monotonically_increasing_id())

attractions_df_alias = attractions_df.alias("attractions")
cities_df_alias_2 = cities_df.alias("attraction_city")
```

```
attractions_visited_clean_df = attractions_visited_df.alias("visit") \
.join(attractions_df_alias, trim(col("visit.attraction_name")) == trim(col("attractions.attraction_name")), "left")
\
.join(cities_df_alias_2, trim(col("visit.city")) == trim(col("attraction_city.city")), "left") \
.select(
col("visit.visit_id"),
col("visit.attraction_id"), # existing column in dataset
col("attractions.attraction_id").alias("normalized_attraction_id"),
col("attraction_city.city_id"),
col("visit.visit_date"),
col("visit.visitors_count"),
col("visit.revenue"),
col("visit.average_rating")
)

# --------------------------
# Save Normalized Tables as Delta Tables
# --------------------------

# Dimension Tables
nationality_df.write.format("delta").mode("overwrite").saveAsTable("normalized_nationality")
travel_purpose_df.write.format("delta").mode("overwrite").saveAsTable("normalized_travel_purpose")
cities_df.write.format("delta").mode("overwrite").saveAsTable("normalized_cities")
hotels_df.write.format("delta").mode("overwrite").saveAsTable("normalized_hotels")
attractions_df.write.format("delta").mode("overwrite").saveAsTable("normalized_attractions")

# Fact Tables (Normalized)
tourist_clean_df.write.format("delta").mode("overwrite").saveAsTable("normalized_tourist")
flight_bookings_clean_df.write.format("delta").mode("overwrite").saveAsTable("normalized_flight_bookings")
hotel_stays_clean_df.write.format("delta").mode("overwrite").saveAsTable("normalized_hotel_stays")
attractions_visited_clean_df.write.format("delta").mode("overwrite").saveAsTable("normalized_attractions_visi
ted")

# --------------------------
# Verify Tables Created
# --------------------------
spark.sql("SHOW TABLES").show()

# OPTIONAL: Preview the data in tables
spark.sql("SELECT * FROM normalized_tourist LIMIT 10").show()
spark.sql("SELECT * FROM normalized_flight_bookings LIMIT 10").show()
spark.sql("SELECT * FROM normalized_hotel_stays LIMIT 10").show()
spark.sql("SELECT * FROM normalized_attractions_visited LIMIT 10").show()
```

📸 **Screenshot #3 – Normalized Dimension Table Previews**

```
+--------+--------------------+-----------+
|database|           tableName|isTemporary|
+--------+--------------------+-----------+
| default| attractions_visited|      false|
| default|     flight_bookings|      false|
| default|         hotel_stays|      false|
| default|normalized_attrac...|      false|
| default|normalized_attrac...|      false|
| default|   normalized_cities|      false|
| default|normalized_flight...|      false|
| default|normalized_hotel_...|      false|
| default|   normalized_hotels|      false|
| default|normalized_nation...|      false|
| default|  normalized_tourist|      false|
| default|normalized_travel...|      false|
| default|tourist_demographics|      false|
+--------+--------------------+-----------+
```

3. Data Quality Assurance

The purpose of the Data Quality Assurance (DQA) process is to ensure that the data integrated into the Tourism Data Warehouse is accurate, complete, consistent, and reliable. This process plays a critical role in maintaining trust in the data for downstream analytical and business intelligence (BI) applications.

Comprehensive data quality checks were performed across the **staging layer (normalized tables)** and the **dimensional model (star schema)** to validate data integrity before final consumption.

1. Null, Blank, and NaN Checks

Purpose

To detect and handle missing, incomplete, or blank data entries that could compromise data integrity and analytic accuracy.

2. Duplicate Primary Key Checks

Purpose

To ensure uniqueness and prevent duplication of records across dimension and fact tables where surrogate or natural primary keys are defined.

3. Foreign Key Integrity Checks

Purpose

To validate referential integrity between fact and dimension tables by ensuring that all foreign key values exist in the respective dimension tables.

4. Domain and Range Validation

Purpose

To verify that numerical and date-based data fields fall within acceptable, predefined ranges.

Output -

4. Dimensional Modeling (Star Schema)

- Kimball's methodology adopted with a Star Schema for the analytical layer.
- Includes dimension tables (tourist, city, attraction, etc.) and fact tables (flight bookings, hotel stays, attractions visits).
- SCD Type 2 applied to dim_tourist to track historical changes.

Key Features of the Design:

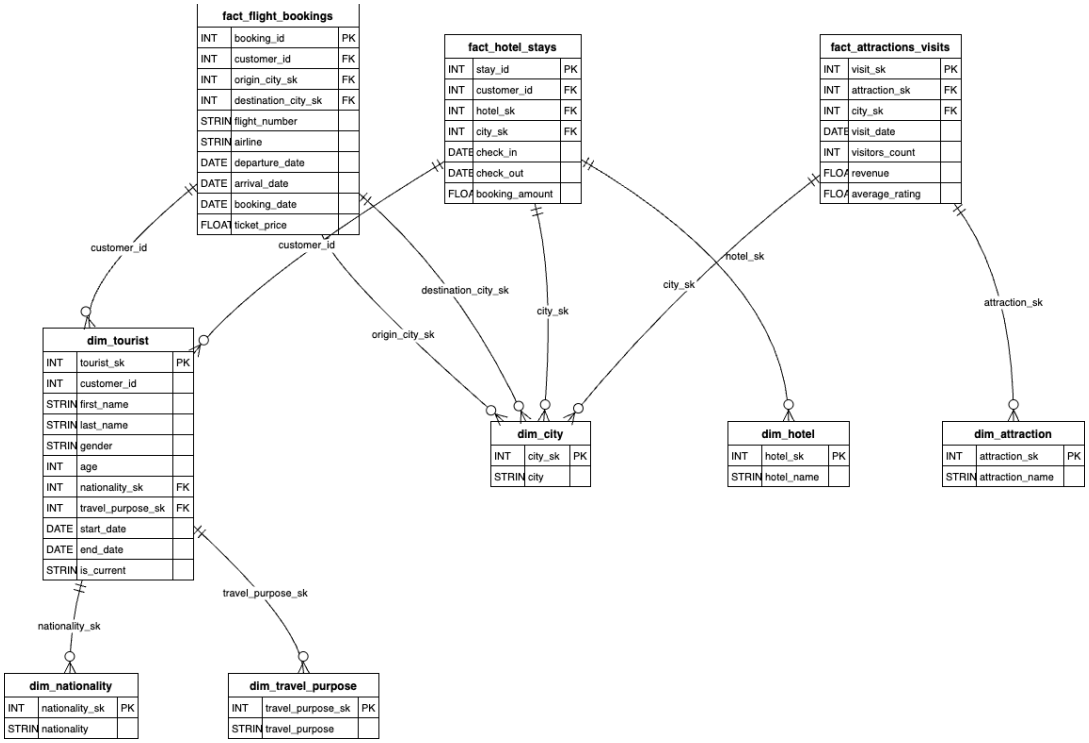
- **Fact Tables** store measurable, quantitative data about business processes.
- **Dimension Tables** provide descriptive attributes (context) for facts.
- **Surrogate Keys** ensure data integrity, uniqueness, and efficient joins.
- **Slowly Changing Dimension Type 2 (SCD2)** manages and preserves historical changes in dimension data.

Star Schema Overview

The **Star Schema** consists of:

- **Central Fact Tables**, representing events or transactions (e.g., flight bookings, hotel stays, attraction visits).
- **Connected Dimension Tables**, providing descriptive context (e.g., tourists, cities, nationalities).

Star Schema ERD Diagram with Facts and Dimensions.



A **Surrogate Key** is:

- A unique, system-generated identifier (often an integer).
- Independent of the business key or natural key (like customer ID or city name).

Why Use Surrogate Keys?

- Business keys can change (e.g., customer ID updates).
- Surrogate keys provide stability and simplicity for joins.
- Essential for implementing **Slowly Changing Dimensions** and maintaining historical accuracy.

In Spark, we generated surrogate keys using monotonically_increasing_id() for each dimension table.

Slowly Changing Dimension (SCD) Type 2 Explained

A **Slowly Changing Dimension (SCD)** manages changes in dimension attributes over time.

SCD Type 1

- Overwrites old data.
- No historical data is preserved.

SCD Type 2 (Used in This Project)

- **Preserves history** by adding new rows for each change.
- Each row includes **effective dates** (start_date and end_date).
- **Current record** is flagged (is_current = 'Y').
- Prior versions have is_current = 'N'.

Why SCD Type 2?

Businesses need to:

- Analyze **point-in-time** data.
- Answer questions like "What was the tourist's travel purpose in 2023?"
- Track **historical changes** in customer demographics for accurate reporting.

Dimension Tables

dim_nationality

Column Name	Data Type	Description

nationality_sk	BIGINT	Surrogate primary key.
nationality	STRING	Country/Nationality.

dim_travel_purpose

Column Name	Data Type	Description
travel_purpose_sk	BIGINT	Surrogate primary key.
travel_purpose	STRING	Purpose (Leisure, Business).

dim_city

Column Name	Data Type	Description
city_sk	BIGINT	Surrogate primary key.
city	STRING	Name of the city.

dim_hotel

Column Name	Data Type	Description
hotel_sk	BIGINT	Surrogate primary key.
hotel_name	STRING	Name of the hotel.

dim_attraction

Column Name	Data Type	Description
attraction_sk	BIGINT	Surrogate primary key.
attraction_name	STRING	Name of the attraction.

dim_tourist (SCD2)

Column Name	Data Type	Description
tourist_sk	BIGINT	Surrogate primary key.
customer_id	STRING	Business/Natural key for tourists.
first_name	STRING	Tourist's first name.

last_name	STRING	Tourist's last name.
gender	STRING	Gender of the tourist.
age	INT	Age of the tourist.
nationality_sk	BIGINT	FK to dim_nationality.
travel_purpose_sk	BIGINT	FK to dim_travel_purpose.
start_date	DATE	Start date of this version (SCD2).
end_date	DATE	End date of this version (SCD2).
is_current	STRING	'Y' for current record, 'N' otherwise.

Fact Tables – Detailed Design

Fact tables capture **measurable events** or transactions and link to dimensions via **foreign keys** (FKs to surrogate keys).

fact_flight_bookings

Column Name	Data Type	Description
booking_id	STRING	Unique flight booking identifier.
customer_id	STRING	FK to dim_tourist.
origin_city_sk	BIGINT	FK to dim_city (origin).
destination_city_sk	BIGINT	FK to dim_city (destination).
flight_number	STRING	Flight number.
airline	STRING	Airline company name.
departure_date	DATE	Date of departure.
arrival_date	DATE	Date of arrival.
booking_date	DATE	Date when the flight was booked.
ticket_price	DECIMAL	Price of the flight ticket.

fact_hotel_stays

Column Name	Data Type	Description
stay_id	STRING	Unique hotel stay identifier.
customer_id	STRING	FK to dim_tourist.
hotel_sk	BIGINT	FK to dim_hotel.
city_sk	BIGINT	FK to dim_city.
check_in	DATE	Check-in date.
check_out	DATE	Check-out date.
booking_amount	DECIMAL	Total amount paid for the stay.

fact_attractions_visits

Column Name	Data Type	Description
visit_sk	BIGINT	Surrogate key for each visit.
attraction_sk	BIGINT	FK to dim_attraction.
city_sk	BIGINT	FK to dim_city.
visit_date	DATE	Date of the visit.
visitors_count	INT	Number of visitors on that date.
revenue	DECIMAL	Revenue generated by the attraction visit.
average_rating	DECIMAL	Average rating given by visitors.

ETL Process

- Extract: Raw datasets loaded into Spark DataFrames.
- Transform:
  - Cleaned data (trimming, deduplication, handling missing values).
  - Generated surrogate keys and ensured referential integrity.
  - Applied SCD Type 2 for historical tracking.
- Load:
  - Data saved into Delta Tables for optimized querying and scalability.
  -

The purpose of this ETL (Extract, Transform, Load) process is to **integrate data from multiple heterogeneous sources**, **clean and transform it** into meaningful structures, and **load it** into a **data warehouse** for **analytical querying and reporting**.

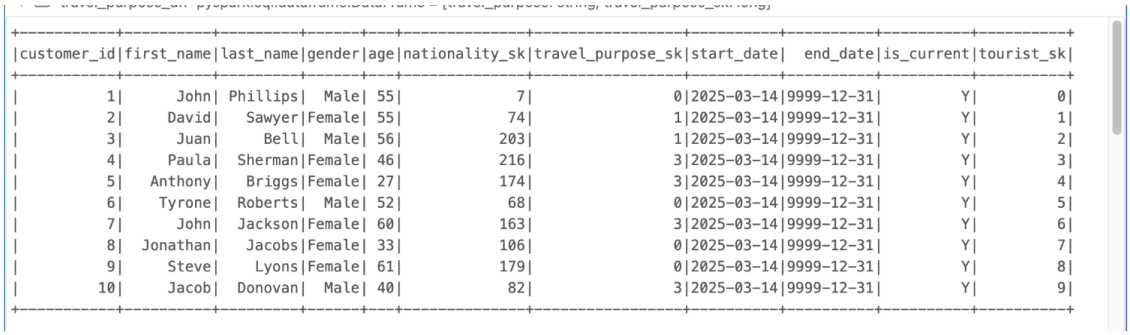
This process ensures data:

- Is accurate, complete, and consistent.
- Is transformed into a **normalized** structure for staging.
- Is organized into a **dimensional model (Star Schema)** optimized for BI tools and reporting.

Code Snippet: Building Dimension Tables with SCD Type 2

```
# ----------------------------
# Save Dimension Tables (with SCD2 tourist)
# ----------------------------
nationality_df.write.format("delta").option("overwriteSchema", "true").mode("overwrite").saveAsTable("dim_nationality")
travel_purpose_df.write.format("delta").option("overwriteSchema", "true").mode("overwrite").saveAsTable
("dim_travel_purpose")
cities_df.write.format("delta").option("overwriteSchema", "true").mode("overwrite").saveAsTable("dim_city")
hotels_df.write.format("delta").option("overwriteSchema", "true").mode("overwrite").saveAsTable("dim_hotel")
attractions_df.write.format("delta").option("overwriteSchema", "true").mode("overwrite").saveAsTable("dim_attraction")
tourist_dim_df.write.format("delta").option("overwriteSchema", "true").mode("overwrite").saveAsTable("dim_tourist")
```

📸 Screenshot #4 – SCD2 in dim_tourist Table



customer_id	first_name	last_name	gender	age	nationality_sk	travel_purpose_sk	start_date	end_date	is_current	tourist_sk
1	John	Phillips	Male	55	7	0	2025-03-14	9999-12-31	Y	0
2	David	Sawyer	Female	55	74	1	2025-03-14	9999-12-31	Y	1
3	Juan	Bell	Male	56	203	1	2025-03-14	9999-12-31	Y	2
4	Paula	Sherman	Female	46	216	3	2025-03-14	9999-12-31	Y	3
5	Anthony	Briggs	Female	27	174	3	2025-03-14	9999-12-31	Y	4
6	Tyrone	Roberts	Male	52	68	0	2025-03-14	9999-12-31	Y	5
7	John	Jackson	Female	60	163	3	2025-03-14	9999-12-31	Y	6
8	Jonathan	Jacobs	Female	33	106	0	2025-03-14	9999-12-31	Y	7
9	Steve	Lyons	Female	61	179	0	2025-03-14	9999-12-31	Y	8
10	Jacob	Donovan	Male	40	82	3	2025-03-14	9999-12-31	Y	9

Fact Table Construction

- Fact tables record **transactions** linked to **dimensions** using **surrogate keys**.
- Surrogate keys simplify joins and ensure historical consistency.

Fact Tables Created

- **fact_flight_bookings**
- **fact_hotel_stays**
- **fact_attractions_visits**

Code Snippet: Building Fact Tables

```
# ---------------------------
# Fact_Flight_Bookings
# ---------------------------
cities_df_origin = cities_df.alias("origin_city")
cities_df_destination = cities_df.alias("destination_city")


fact_flight_bookings_df = flight_bookings_df.alias("flight") \
.join(cities_df_origin, trim(col("flight.origin_city")) == trim(col("origin_city.city")), "left") \
.join(cities_df_destination, trim(col("flight.destination_city")) == trim(col("destination_city.city")), "left") \
.select(
col("flight.booking_id"),
col("flight.customer_id"),
col("origin_city.city_sk").alias("origin_city_sk"),
col("destination_city.city_sk").alias("destination_city_sk"),
col("flight.flight_number"),
col("flight.airline"),
col("flight.departure_date"),
col("flight.arrival_date"),
col("flight.booking_date"),
col("flight.ticket_price")
```

```
)

# --------------------------
# Fact_Hotel_Stays
# --------------------------
hotels_df_alias = hotels_df.alias("hotels")
cities_df_alias = cities_df.alias("hotel_city")

fact_hotel_stays_df = hotel_stays_df.alias("hotel") \
.join(hotels_df_alias, trim(col("hotel.hotel_name")) == trim(col("hotels.hotel_name")), "left") \
.join(cities_df_alias, trim(col("hotel.city")) == trim(col("hotel_city.city")), "left") \
.select(
col("hotel.booking_id").alias("stay_id"),
col("hotel.customer_id"),
col("hotels.hotel_sk"),
col("hotel_city.city_sk"),
col("hotel.check_in_date").alias("check_in"),
col("hotel.check_out_date").alias("check_out"),
col("hotel.total_amount").alias("booking_amount")
)

# --------------------------
# Fact_Attractions_Visits (No customer_id)
# --------------------------
attractions_df_alias = attractions_df.alias("attractions")
cities_df_alias_2 = cities_df.alias("attraction_city")

fact_attractions_visits_df = attractions_visited_df.alias("visit") \
.join(attractions_df_alias, trim(col("visit.attraction_name")) == trim(col("attractions.attraction_name")), "left")
\
.join(cities_df_alias_2, trim(col("visit.city")) == trim(col("attraction_city.city")), "left") \
.withColumn("visit_sk", monotonically_increasing_id()) \
.select(
col("visit_sk"),
col("attractions.attraction_sk"),
col("attraction_city.city_sk"),
col("visit.visit_date"),
col("visit.visitors_count"),
col("visit.revenue"),
col("visit.average_rating")
)
```

📸 Screenshot #5 – Fact Table Data Preview

```
----+------------+
|booking_id|customer_id|origin_city_sk|destination_city_sk|flight_number|       airline|departure_date|arrival_date|booking_
date|ticket_price|
+----------+-----------+--------------+-------------------+-------------+--------------+--------------+------------+--------
----+------------+
|         1|        500|             1|                  0|       FL6844|United Airlines|    2025-04-03|  2025-04-03|  2025-0
2-15|      819.79|
|         2|        380|             0|                  1|       FL8071|      Emirates|    2024-08-06|  2024-08-06|  2024-0
8-06|       565.6|
|         3|        699|             4|                  0|       FL6921|      Emirates|    2024-10-08|  2024-10-08|  2024-0
8-22|      513.88|
|         4|        391|             0|                  1|       FL4809|     Lufthansa|    2025-01-22|  2025-01-22|  2025-0
1-08|      852.24|
|         5|        156|             4|                  3|       FL1878| Qatar Airways|    2024-08-04|  2024-08-04|  2024-0
6-11|     1453.13|
|         6|        278|             2|                  4|       FL8064| Qatar Airways|    2024-06-15|  2024-06-15|  2024-0
5-24|     1034.11|
|         7|        968|             3|                  2|       FL3004|      Emirates|    2025-01-17|  2025-01-17|  2024-1
2-07|      727.48|
|         8|        582|             5|                  6|       FL6441| Qatar Airways|    2025-02-08|  2025-02-08|  2025-0
1-14|      317.86|
```

stay_id	customer_id	hotel_sk	city_sk	check_in	check_out	booking_amount
1	796	1	0	2024-09-19	2024-09-26	990.34
2	876	2	6	2024-09-20	2024-09-26	235.59
3	174	3	4	2024-06-21	2024-06-28	922.14
4	433	2	3	2024-07-25	2024-07-31	756.29
5	356	3	6	2024-09-25	2024-10-01	79.74
6	298	1	5	2024-09-02	2024-09-09	665.91
7	388	0	4	2024-03-21	2024-03-24	643.37
8	732	0	4	2025-02-21	2025-02-28	814.98
9	192	0	6	2024-07-23	2024-07-28	291.78
10	959	3	6	2024-08-24	2024-08-25	920.48

visit_sk	attraction_sk	city_sk	visit_date	visitors_count	revenue	average_rating
0	0	3	2024-11-10	64	4195.84	3.35
1	2	6	2024-05-14	884	40133.6	3.68
2	0	3	2025-02-17	192	17383.68	4.02
3	3	5	2024-08-30	257	10752.880000000001	4.68
4	0	3	2025-02-20	848	52236.8	3.45
5	3	5	2024-05-25	399	16315.11	3.85
6	2	6	2024-09-06	777	47319.299999999996	4.48
7	0	3	2025-01-24	971	89535.90999999999	4.25
8	0	3	2024-03-17	100	9276.0	3.07
9	2	6	2025-01-17	927	75504.15000000001	3.26

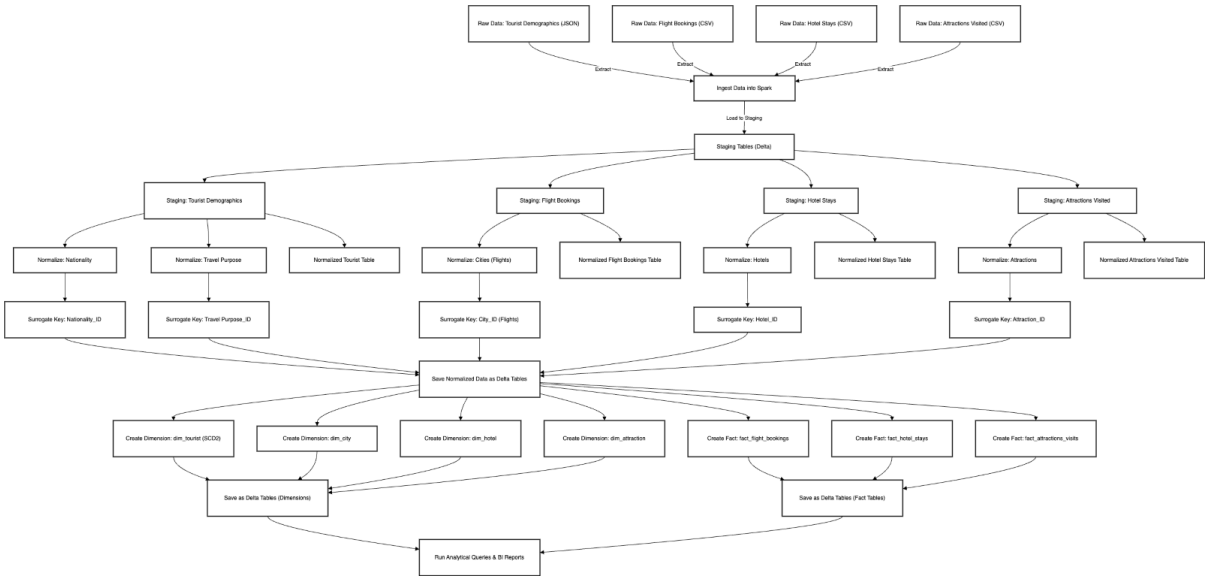
5. ETL Architecture Overview

The ETL process follows the **Kimball methodology**:

1. **Extract** raw data from source files.

2. **Transform** it by cleansing, normalizing, and enriching it.
3. **Load** it into:
    a. Normalized staging tables.
    b. Star schema dimensional models.

📸 **Screenshot #1 – ETL Architecture Diagram**



6. Analytical Queries and Reporting

- Developed analytical queries for:
    - Sales performance
    - Customer segmentation
    - Geographic sales analysis
    - Complex multi-dimensional analysis
- Visualizations created to support insights.

Purpose and Objective of Analytical Queries

The goal of these **analytical queries** is to transform raw and processed data into **business insights** that help stakeholders **make informed decisions**. By leveraging the **dimensional star schema** built in our data warehouse, we can easily generate reports, analyze trends, and extract actionable intelligence.

Description of Analytical Queries and Business Insights

🔑 Query 1: Top 5 Destination Cities by Flight Bookings and Revenue

Query Purpose

This query identifies the **top 5 destination cities** based on:

- **Total number of flight bookings**
- **Total revenue from ticket sales**

It focuses on bookings from **January 1, 2024**, to the present. This insight helps track recent travel trends.

SQL Code

Query 1: Top 5 Cities by Tourist Flight Bookings and Revenue

✓ 23 hours ago (3s) 20

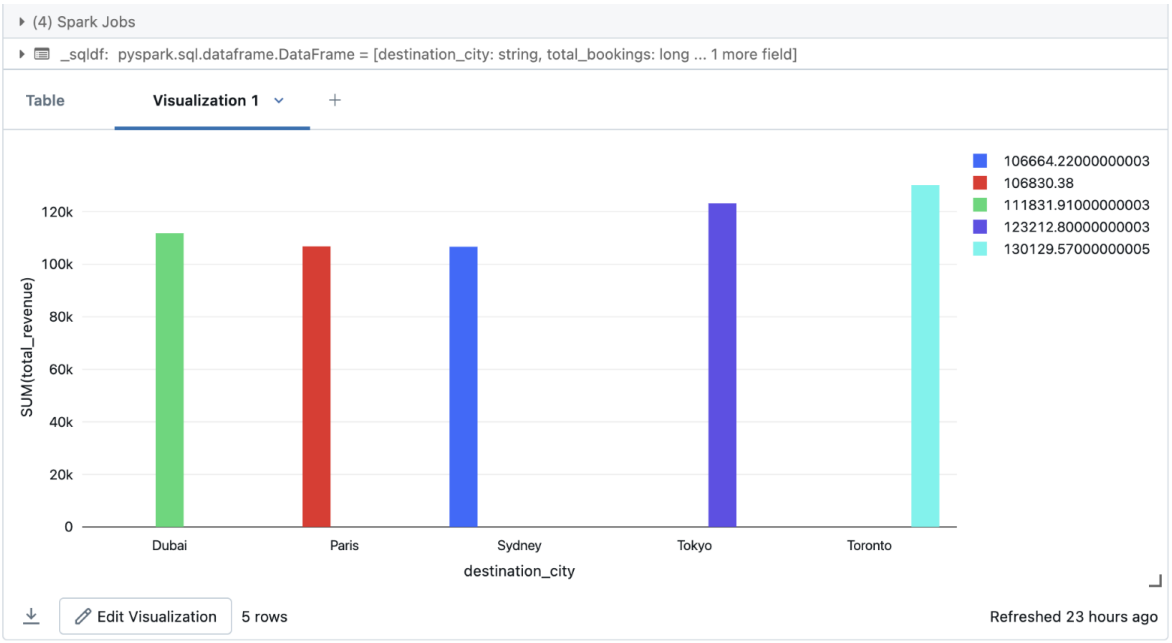
```
%sql
SELECT
    dc.city AS destination_city,
    COUNT(ffb.booking_id) AS total_bookings,
    SUM(ffb.ticket_price) AS total_revenue
FROM
    fact_flight_bookings ffb
JOIN
    dim_city dc ON ffb.destination_city_sk = dc.city_sk
WHERE
    ffb.booking_date >= '2024-01-01'  -- Filter: Recent bookings (adjust as needed)
GROUP BY
    dc.city
ORDER BY
    total_bookings DESC
LIMIT 5;
```

▸ (4) Spark Jobs

▸ _sqldf: pyspark.sql.dataframe.DataFrame = [destination_city: string, total_bookings: long ... 1 more field]

Business Insight

- Highlights **popular tourist destinations** that attract the most travelers.
- Provides insight into **where to allocate marketing budgets**, focusing on high-demand cities.
- Assists airlines and tourism boards in **capacity planning** for future seasons.



🔑 Query 2: Nationality-wise Hotel Stay Analysis

Query Purpose

This query analyzes **hotel stays** segmented by **tourist nationality**:

- **Number of stays**
- **Average stay duration (in days)**
- **Total hotel spend**

It evaluates **only current tourist records** (using is_current = 'Y' from the SCD Type 2 dimension).

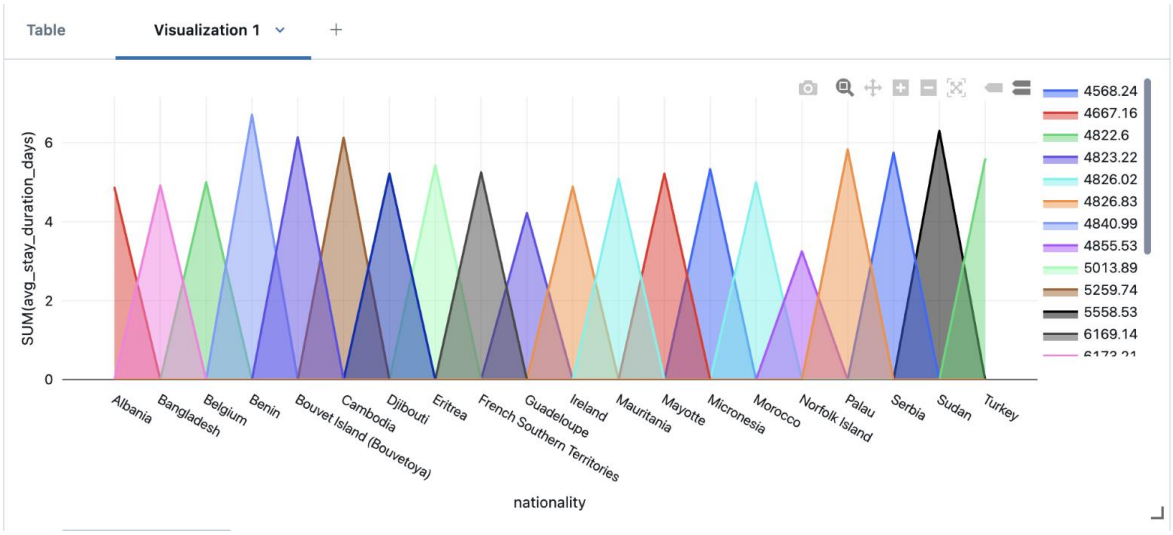
SQL Code

Query 2: Average Stay Duration and Spending by Nationality

```
%sql
SELECT
    dn.nationality,
    COUNT(fhs.stay_id) AS total_stays,
    ROUND(AVG(DATEDIFF(fhs.check_out, fhs.check_in)), 2) AS avg_stay_duration_days,
    ROUND(SUM(fhs.booking_amount), 2) AS total_spent
FROM
    fact_hotel_stays fhs
JOIN
    dim_tourist dt
    ON fhs.customer_id = dt.customer_id
JOIN
    dim_nationality dn
    ON dt.nationality_sk = dn.nationality_sk
WHERE
    dt.is_current = 'Y'  -- SCD2: Only current records
GROUP BY
    dn.nationality
ORDER BY
    total_spent DESC
LIMIT 20;
```

Business Insight

- Identifies **which nationalities** contribute most to hotel revenues.
- Guides **marketing campaigns** towards countries with tourists who spend more or stay longer.
- Helps hotels **customize services** to cater to specific nationalities (e.g., language, cuisine).



🔑 Query 3: High-Value Tourists Spending Analysis

Query Purpose

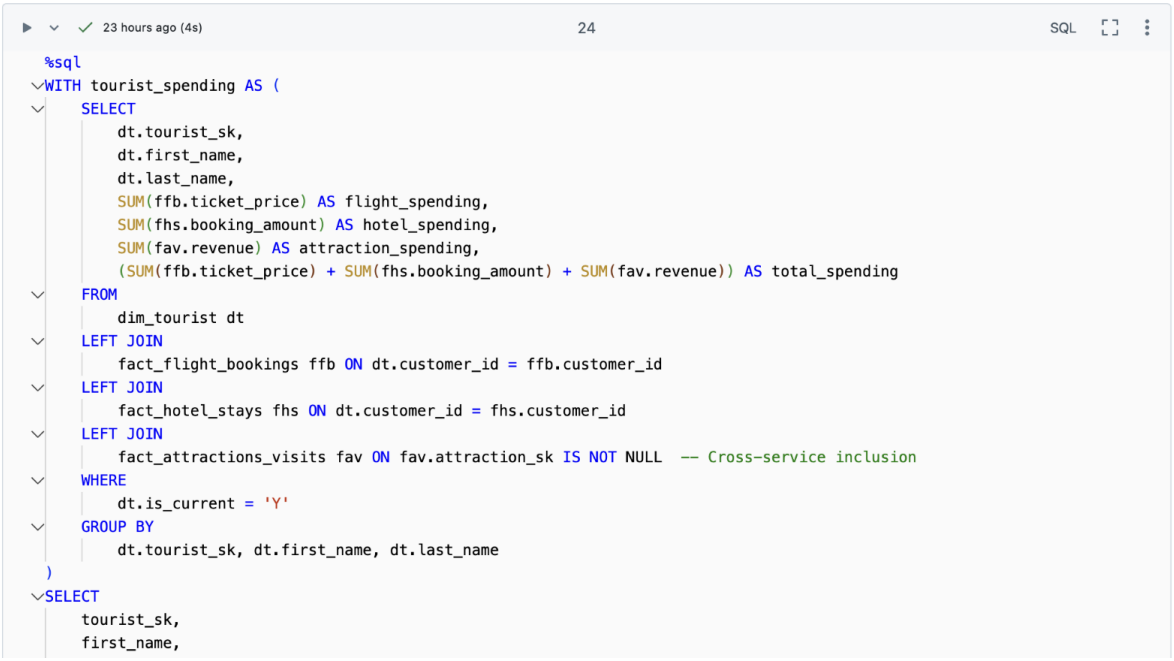
This query finds **top-spending tourists** across:

- **Flights**
- **Hotels**
- **Attractions**

It aggregates their **total spend** and identifies tourists with **spending over 10,000 units**, marking them as **VIP/High-Value Customers**.

SQL Code

Query 3 (Complex): High-Value Tourists and Their Multi-Activity Engagement



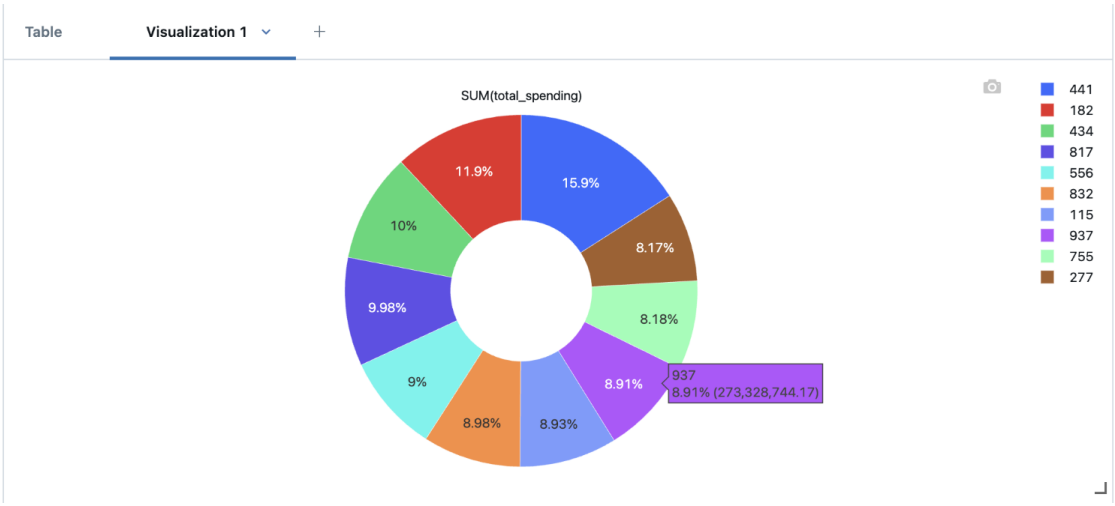
```
%sql
WITH tourist_spending AS (
    SELECT
        dt.tourist_sk,
        dt.first_name,
        dt.last_name,
        SUM(ffb.ticket_price) AS flight_spending,
        SUM(fhs.booking_amount) AS hotel_spending,
        SUM(fav.revenue) AS attraction_spending,
        (SUM(ffb.ticket_price) + SUM(fhs.booking_amount) + SUM(fav.revenue)) AS total_spending
    FROM
        dim_tourist dt
    LEFT JOIN
        fact_flight_bookings ffb ON dt.customer_id = ffb.customer_id
    LEFT JOIN
        fact_hotel_stays fhs ON dt.customer_id = fhs.customer_id
    LEFT JOIN
        fact_attractions_visits fav ON fav.attraction_sk IS NOT NULL  -- Cross-service inclusion
    WHERE
        dt.is_current = 'Y'
    GROUP BY
        dt.tourist_sk, dt.first_name, dt.last_name
)
SELECT
    tourist_sk,
    first_name,
```

```
SELECT
    tourist_sk,
    first_name,
    last_name,
    ROUND(flight_spending, 2) AS flight_spending,
    ROUND(hotel_spending, 2) AS hotel_spending,
    ROUND(attraction_spending, 2) AS attraction_spending,
    ROUND(total_spending, 2) AS total_spending
FROM
    tourist_spending
WHERE
    total_spending > 10000  -- High-value threshold
ORDER BY
    total_spending DESC
LIMIT 10;
```

Business Insight

- Identifies **high-value tourists** for:
    - **Personalized offers** (luxury packages, exclusive deals)
    - **Loyalty programs**
- Supports **Customer Lifetime Value (CLV)** modeling.
- Helps **maximize retention** and **repeat business** from top customers.

Recommended Visuals / Screenshot #3



🔎 Query 4: Monthly Airline Revenue and Booking Analysis

Query Purpose

Tracks **monthly trends** in:

- **Total bookings**
- **Total revenue** by **airline**, starting from **January 2024**.

SQL Code

Monthly Flight Revenue and Top Airlines

22 hours ago (2s) 26 SQL

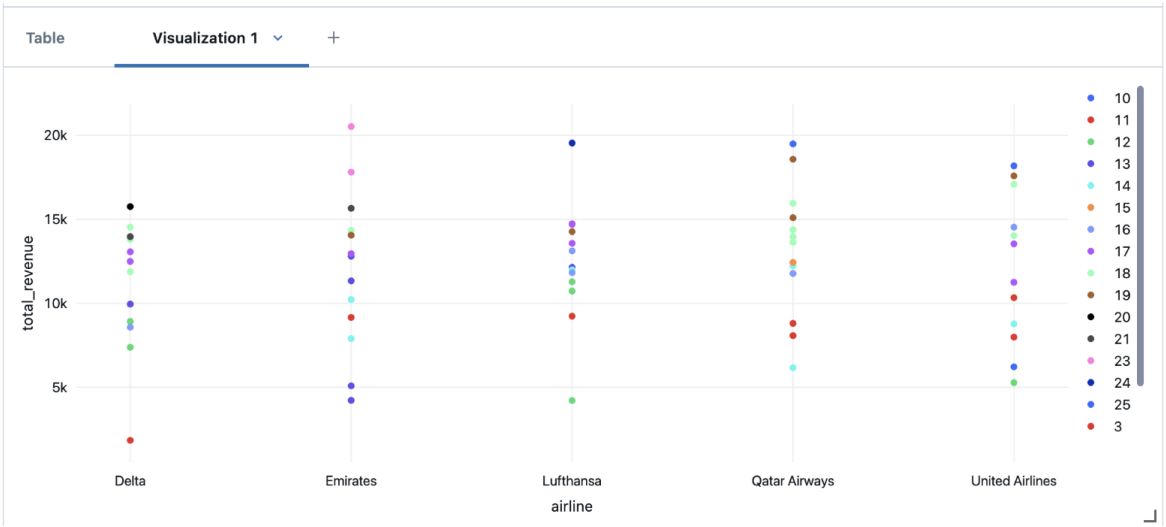
```sql
%sql
SELECT
    DATE_FORMAT(ffb.booking_date, 'yyyy-MM') AS booking_month,
    ffb.airline,
    COUNT(ffb.booking_id) AS total_bookings,
    ROUND(SUM(ffb
FROM
    fact_flight_bookings ffb
JOIN
    dim_city dc_origin ON ffb.origin_city_sk = dc_origin.city_sk
JOIN
    dim_city dc_dest ON ffb.destination_city_sk = dc_dest.city_sk
WHERE
    ffb.booking_date >= '2024-01-01'
GROUP BY
    DATE_FORMAT(ffb.booking_date, 'yyyy-MM'),
    ffb.airline
ORDER BY
    booking_month ASC,
    total_revenue DESC;
```

Loading...

Business Insight

- Identifies **seasonal trends** in flight bookings.
- Reveals **top-performing airlines** by month.
- Helps airlines and travel agencies plan:
    - **Promotions** in low seasons.
    - **Resource allocation** in peak months.

Recommended Visuals / Screenshot #4



🔑 Query 5: Top Tourist Attractions by Revenue, Rating, and Visitors

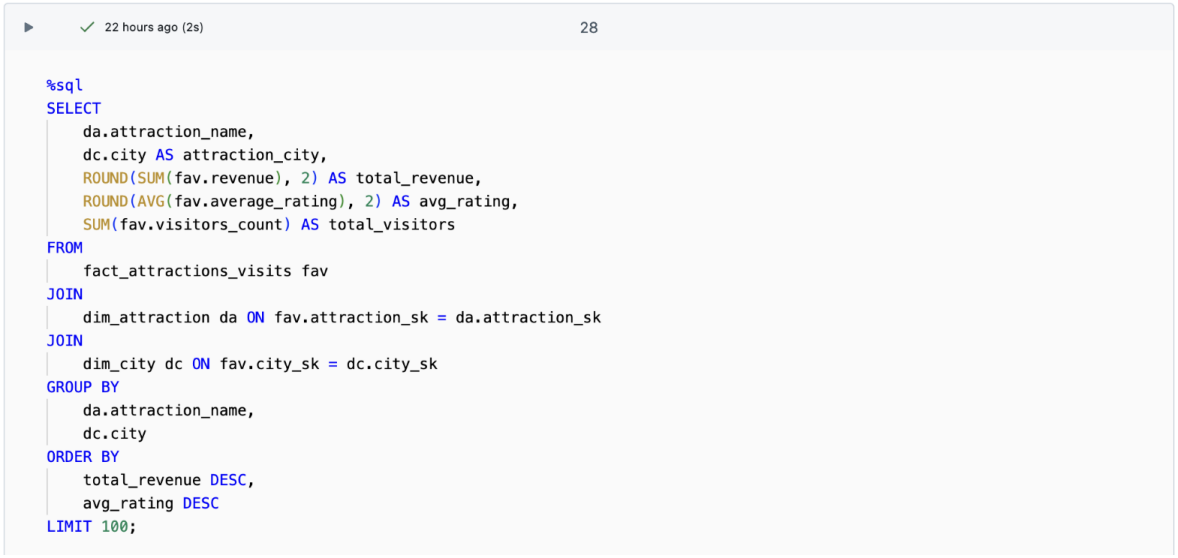
Query Purpose

Lists the **top tourist attractions**, ranked by:

- **Total revenue**
- **Average visitor ratings**
- **Total number of visitors**

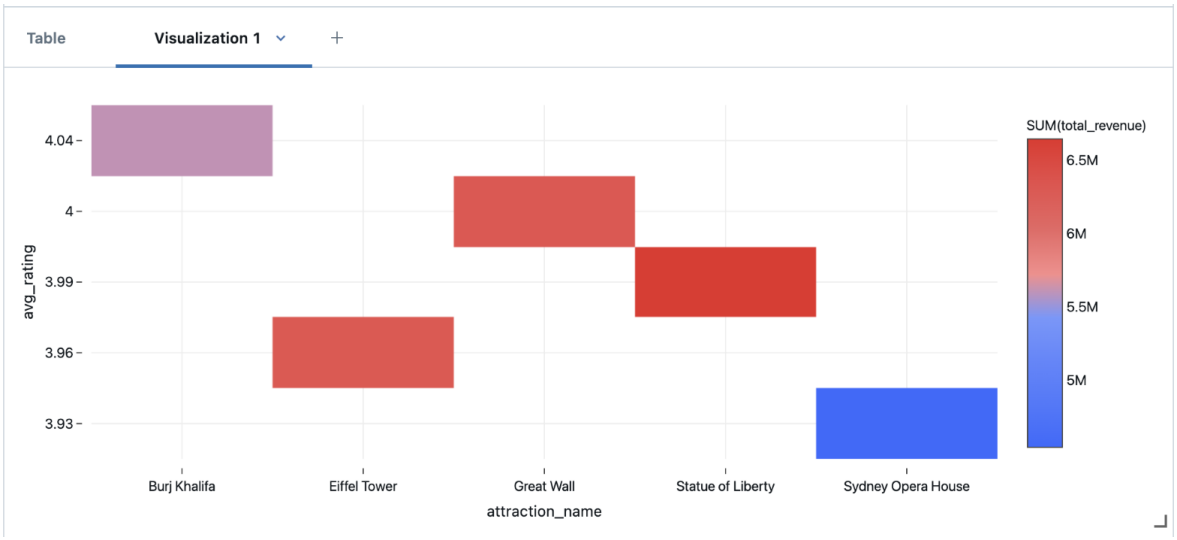
SQL Code

Top Attractions by Revenue and Visitor Ratings



Business Insight

- Pinpoints **top-earning attractions** for **investment** and **promotional focus**.
- Reveals which attractions have **highest ratings**, suggesting **customer satisfaction**.
- Helps in **destination planning**, **partnerships**, and **resource allocation** for tourism boards.



7. Challenges Encountered and Solutions Applied

- **Heterogeneous Data Formats (CSV, JSON):** To handle different file formats, we utilized Spark's read.option() for CSV files and read.json() for JSON files. This approach allowed us to efficiently load and validate heterogeneous data sources.
- **Data Quality Issues (Duplicates, Nulls):** We addressed data quality challenges by implementing a comprehensive cleansing process. This included using trim() to remove unnecessary whitespace, applying distinct() to eliminate duplicate records, and handling null values during the transformation stage to ensure clean and reliable data.
- **Surrogate Key Generation at Scale:** To generate unique surrogate keys at scale, we leveraged Spark's monotonically_increasing_id() function. This ensured each record had a distinct identifier, which was critical for maintaining data integrity in our dimensional models.
- **Historical Tracking in Dimensional Tables (SCD Type 2):** For historical data tracking, we implemented Slowly Changing Dimension Type 2 (SCD2). We added start_date, end_date, and is_current columns to the dim_tourist table, enabling the system to capture and maintain a full history of changes over time.

8. Analysis of the Effectiveness of the Dimensional Model

- **Query Performance:** The use of a star schema combined with surrogate keys significantly enhanced query performance. This structure enabled fast and efficient joins, allowing analytical queries to run with minimal latency even on large datasets.
- **Historical Data Handling:** By implementing Slowly Changing Dimension Type 2 (SCD2), we successfully maintained historical accuracy for key dimensions such as tourists. This allowed the system to capture and reflect changes over time without losing valuable historical information.
- **Business-Friendly Data Structure:** The dimensional model was designed to be simple and intuitive, making it highly accessible for business users and BI tools. The straightforward structure facilitated self-service reporting and analysis, reducing dependency on technical teams.
- **Support for Complex Analytics:** The model supported complex analytical requirements, including customer lifetime value (CLV) calculations and sales trend analysis. It provided a robust foundation for advanced metrics and KPIs, driving more informed business decisions.
- **Data Quality and Consistency:** Referential integrity was enforced across all fact and dimension tables, ensuring data quality and consistency. This maintained trust in the data and produced reliable metrics for analysis and reporting.
- **Flexibility and Scalability:** Leveraging Delta Lake storage offered flexibility for incremental data loads and scalable ETL workflows. This approach prepared the data warehouse for future growth, allowing seamless integration of additional data sources and increased data volumes.

9. Business Insights Derived

- Identified top tourist nationalities, travel purposes, and spending behaviors.
- Uncovered popular destinations, attractions, and hotels.
- Tracked seasonality trends in travel bookings and attraction visits.
- Enabled customer segmentation for targeted marketing and lifetime value calculations.
- Provided regional performance metrics for logistics optimization and future expansion.

10. Recommendations for Future Improvements

- **Implement dim_date and dim_time Dimensions:** Introduce dedicated date and time dimension tables to simplify time-based analysis. This will allow for more efficient filtering and aggregation by date attributes such as day, month, quarter, and year, streamlining temporal reporting.
- **Partition Fact Tables by Date Fields:** Apply partitioning strategies on fact tables using date fields like booking_date. This will significantly enhance query performance, especially for time-series analysis, by reducing the amount of data scanned during queries.
- **Incremental ETL Pipeline:** Develop an incremental ETL process to enable real-time or near-real-time data updates. This approach minimizes the need for full data reloads, reducing processing time and downtime, while ensuring the data warehouse remains current.
- **Data Quality Monitoring Framework:** Implement an automated data quality framework that applies validation rules and checks during ETL processes. This should include record count verifications, foreign key constraint checks, and other data integrity validations to maintain high-quality data.
- **Advanced Customer Segmentation (ML):** Leverage machine learning models to perform advanced customer segmentation. Predictive segmentation and customer behavior modeling can enhance marketing strategies, personalize customer experiences, and drive more targeted business decisions.
- **Dashboard Development in BI Tools:** Build interactive and intuitive dashboards in business intelligence tools such as Tableau or Power BI. These dashboards will provide stakeholders with real-time insights and facilitate data-driven decision-making across the organization.
- **Incorporate External Data Sources:** Integrate additional data sources such as weather data, social media sentiment analysis, or travel advisories. These external datasets can provide richer context and deeper insights, leading to more comprehensive and actionable analytics.
- **Enhanced SCD Management Tools:** Utilize Delta Lake's Change Data Feed (CDF) to improve the management of Slowly Changing Dimensions (SCD). CDF enables advanced change tracking, ensuring more efficient handling of data updates and historical data management.

11. Conclusion

The Travel & Tourism Data Warehouse project successfully consolidated disparate data sources, ensuring data integrity and enabling advanced analytical reporting. The combination of a normalized database, a star schema dimensional model, and a robust ETL pipeline provided a strong foundation for scalable, efficient analytics.

By addressing real-world challenges and implementing best practices in data warehousing, this project offers a framework for tourism businesses to make data-driven decisions, improve customer experiences, and drive profitability.

8. Appendices (Optional)

- ER Diagrams (Normalized Schema & Star Schema)
- Sample Queries and Results
- Screenshots of Tables & Previews
- ETL Code Snippets