Gaming Data Warehouse Project

1. Data Sources

1.1 Documentation of Data Sources and Relationships

The project utilizes raw gaming session data from multiple sources:

- **Sessions Data**: Contains information about player sessions, including duration, game played, and session date.
- **Players Data**: Includes player demographics, experience levels, and usernames.
- **Purchases Data**: Captures player purchases, including item names and prices.

1.2 Data Dictionary

Table Name	Column Name	Data Type	Description
sessions	session_id	INTEGER	Unique identifier for each session
	player_id	INTEGER	Foreign key referencing players
	game_id	INTEGER	Foreign key referencing games
	session_date	DATE	Date of session
	duration	INTEGER	Session duration in minutes
players	player_id	INTEGER	Unique identifier for players
	username	STRING	Player's username
	level	INTEGER	Player's current level
	experience_points	INTEGER	Experience points accumulated
	region	STRING	Player's region
purchases	purchase_id	INTEGER	Unique identifier for each purchase
	player_id	INTEGER	Foreign key referencing players
	item_name	STRING	Name of purchased item
	item_price	FLOAT	Price of the purchased item
	purchase_date	DATE	Date of the purchase

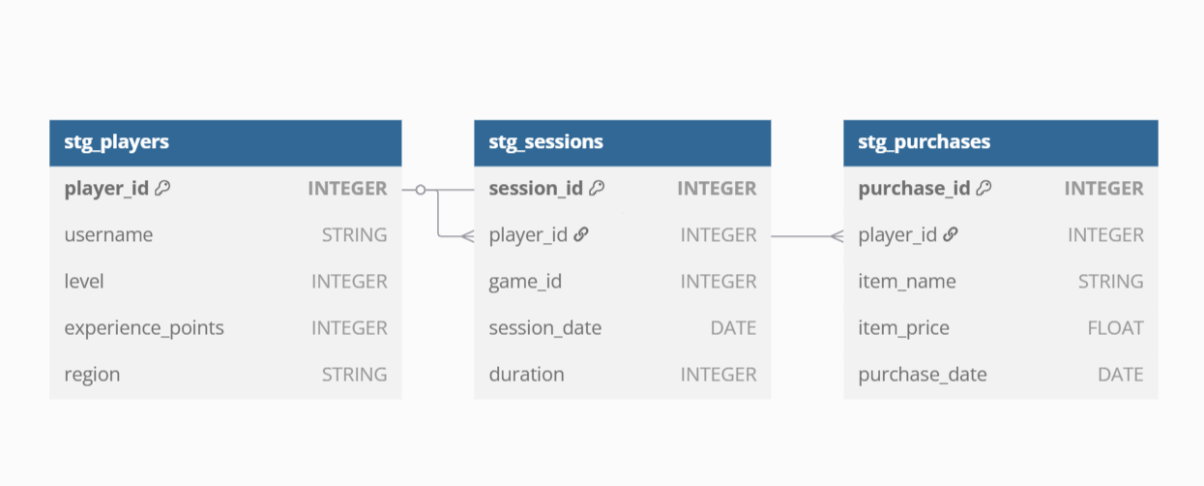
DDL scripts for normalized schema

```
-- Create normalized staging tables
CREATE TABLE `dw-midterm-project.gaming_sessions.stg_players` (
    player_id INTEGER,
    username STRING,
    level INTEGER,
    experience_points INTEGER,
    region STRING
);

CREATE TABLE `dw-midterm-project.gaming_sessions.stg_sessions` (
    session_id INTEGER,
    player_id INTEGER,
    game_id INTEGER,
    session_date DATE,
    duration INTEGER
);

CREATE TABLE `dw-midterm-project.gaming_sessions.stg_purchases` (
    purchase_id INTEGER,
    player_id INTEGER,
    item_name STRING,
    item_price FLOAT,
    purchase_date DATE
);
```

ER diagram



3. ETL Implementation

etl.py

```
import pandas as pd
from google.cloud import bigquery
import os

# Set Google Cloud authentication dynamically & check if the file exists
credential_path = r"C:\Users\spcha\Downloads\dw-midterm-project-94ccd5c877df.json"
if not os.path.exists(credential_path):
    raise FileNotFoundError(f"Service account file not found: {credential_path}")

os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = credential_path

# Initialize BigQuery Client
client = bigquery.Client()
project_id = "dw-midterm-project"
dataset_name = "gaming_sessions"
dataset_id = f"{project_id}.{dataset_name}"
```

```
# Define file paths using a dictionary for better organization
file_paths = {
    "players_file": r"C:\Users\spcha\Desktop\Gaming_DW_project\datasets\players.csv",
    "sessions_file": r"C:\Users\spcha\Desktop\Gaming_DW_project\datasets\csv.csv",
    "purchases_file": r"C:\Users\spcha\Desktop\Gaming_DW_project\datasets\purchases.csv"
}

# Check if CSV files exist before reading and raise an error if not found
missing_files = [file for file, path in file_paths.items() if not os.path.exists(path)]
if missing_files:
    raise FileNotFoundError(f"CSV files not found: {', '.join(missing_files)}")

#  Load the CSV files into DataFrames
players_df = pd.read_csv(file_paths["players_file"])
sessions_df = pd.read_csv(file_paths["sessions_file"])
purchases_df = pd.read_csv(file_paths["purchases_file"])

# Validate that required columns exist
required_players_cols = {"player_id", "username", "level", "experience_points", "region"}
required_sessions_cols = {"session_id", "player_id", "game_id", "session_date", "duration"}
required_purchases_cols = {"player_id", "item_price", "purchase_date"}

for df, required_cols, name in [
    (players_df, required_players_cols, "players.csv"),
    (sessions_df, required_sessions_cols, "sessions.csv"),
    (purchases_df, required_purchases_cols, "purchases.csv"),
]:
    missing_cols = required_cols - set(df.columns)
    if missing_cols:
```

```
        raise ValueError(f"Missing columns in {name}: {missing_cols}")


# Add missing columns to players_df
players_df['player_key'] = None  # Placeholder for player_key
players_df['player_source_id'] = players_df['player_id']  # Assuming player_id is the source ID
players_df['valid_from'] = pd.to_datetime('today').date()  # Set to today's date
players_df['valid_to'] = pd.to_datetime('2262-04-11').date()  # Set to a far future date within bounds
players_df['is_current'] = True  # Set to True


# Transform: Convert date columns to proper format
sessions_df["session_date"] = pd.to_datetime(sessions_df["session_date"]).dt.date
purchases_df["purchase_date"] = pd.to_datetime(purchases_df["purchase_date"]).dt.date


# Aggregate total purchases per player
purchases_agg = purchases_df.groupby("player_id")["item_price"].sum().reset_index()
purchases_agg.rename(columns={"item_price": "total_spent"}, inplace=True)


# Merge sessions with aggregated purchases
fact_game_sessions = sessions_df.merge(purchases_agg, on="player_id", how="left")
fact_game_sessions = fact_game_sessions.assign(total_spent=fact_game_sessions["total_spent"].fillna(0))


# Define BigQuery schemas
dim_player_schema = [
    bigquery.SchemaField("player_key", "INTEGER"),
    bigquery.SchemaField("player_source_id", "INTEGER"),
    bigquery.SchemaField("username", "STRING"),
    bigquery.SchemaField("level", "INTEGER"),
    bigquery.SchemaField("experience_points", "INTEGER"),
```

```
    bigquery.SchemaField("region", "STRING"),
    bigquery.SchemaField("valid_from", "DATE"),
    bigquery.SchemaField("valid_to", "DATE"),
    bigquery.SchemaField("is_current", "BOOLEAN"),
]

fact_game_sessions_schema = [
    bigquery.SchemaField("session_id", "INTEGER"),
    bigquery.SchemaField("player_id", "INTEGER"),
    bigquery.SchemaField("game_id", "INTEGER"),
    bigquery.SchemaField("session_date", "DATE"),
    bigquery.SchemaField("duration", "INTEGER"),
    bigquery.SchemaField("total_spent", "FLOAT"),
]

# Function to check if a table exists & create it if needed
def check_and_create_table(table_name, schema):
    table_ref = f"{dataset_id}.{table_name}"

    try:
        client.get_table(table_ref)  # Check if table exists
    except:
        print(f" Table {table_name} does not exist. Creating it...")
        table = bigquery.Table(table_ref, schema=schema)
        client.create_table(table)

# Function to load data into BigQuery
def load_to_bigquery(df, table_name, schema):
    check_and_create_table(table_name, schema)  # Ensure table exists
```

```
    table_ref = f"{dataset_id}.{table_name}"
    job_config = bigquery.LoadJobConfig(
        schema=schema,
        write_disposition=bigquery.WriteDisposition.WRITE_TRUNCATE,  # Overwrites existing table
    )

    job = client.load_table_from_dataframe(df, table_ref, job_config=job_config)
    job.result()  # Wait for the upload to complete
    print(f" Uploaded {table_name} successfully!")


# Load tables into BigQuery
load_to_bigquery(players_df, "dim_player", dim_player_schema)  # Changed table name to "dim_player"
load_to_bigquery(fact_game_sessions, "fact_game_sessions", fact_game_sessions_schema)


print(" 🚀 ETL Automation Complete!")
```

ETL Process Documentation

Overview

The ETL (Extract, Transform, Load) process is designed to automate the ingestion of raw gaming data into a structured data warehouse. This process ensures data quality, handles historical data changes effectively, and prepares the data for analytical queries.

ETL Workflow

1. **Extraction**
    - Objective: Load raw data from various sources, such as CSV files, into a temporary storage format like DataFrames.
    - Steps:
        - Identify and access the raw data sources.
        - Read the data into a format suitable for processing, ensuring that all necessary fields are captured.

2. **Transformation**
    - Objective: Clean and transform the data to ensure quality and prepare it for loading into the data warehouse.
    - Steps:
        - Data Cleaning: Address missing values by using fillna() in Pandas to replace NULLs with default values, ensuring data completeness.
        - Data Type Consistency: Perform explicit type conversion during transformation to ensure that all fields, especially dates and numeric values, are consistently formatted.
        - Fact Table Aggregations: Use SUM() to aggregate purchase data and enrich session data, ensuring that the fact table captures comprehensive transactional information.
        - SCD Handling: Implement Slowly Changing Dimensions (SCD Type 2) for datasets that require historical tracking, such as player data. This involves maintaining historical records of changes in key attributes.
        - Data Enrichment: Derive new attributes or metrics that are useful for analysis, such as aggregating total spending per player.
3. **Loading**
    - Objective: Load the transformed data into staging tables within the data warehouse.
    - Steps:
        - Establish a connection to the data warehouse.
        - Verify data relationships to ensure foreign key integrity before loading, preventing orphaned records.
        - Insert the cleaned and transformed data into the appropriate staging tables, ensuring that the data is ready for further transformation into dimension and fact tables.

Evidence of Proper Handling of ETL Challenges

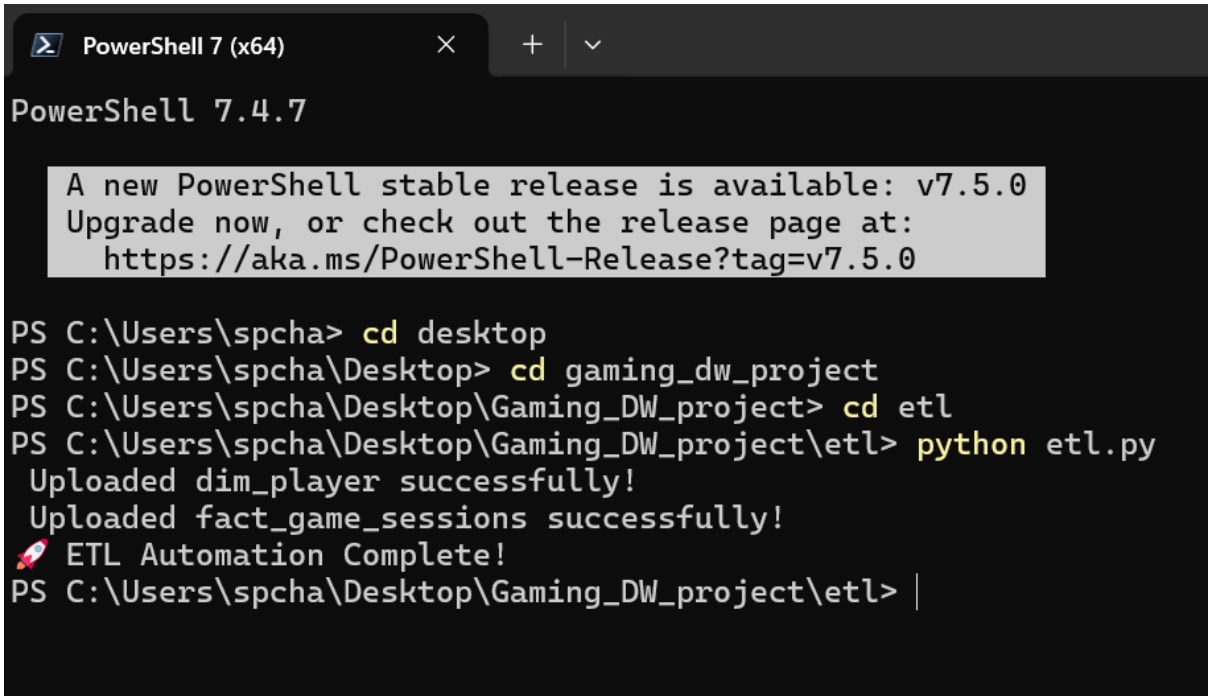
1. **Data Quality**
    - **Handling Missing Values**: Implement strategies to address missing data, such as using default values or statistical imputation, to ensure completeness and accuracy.
    - **Data Type Consistency**: Ensure that all fields, especially dates and numeric values, are consistently formatted to prevent errors during analysis.

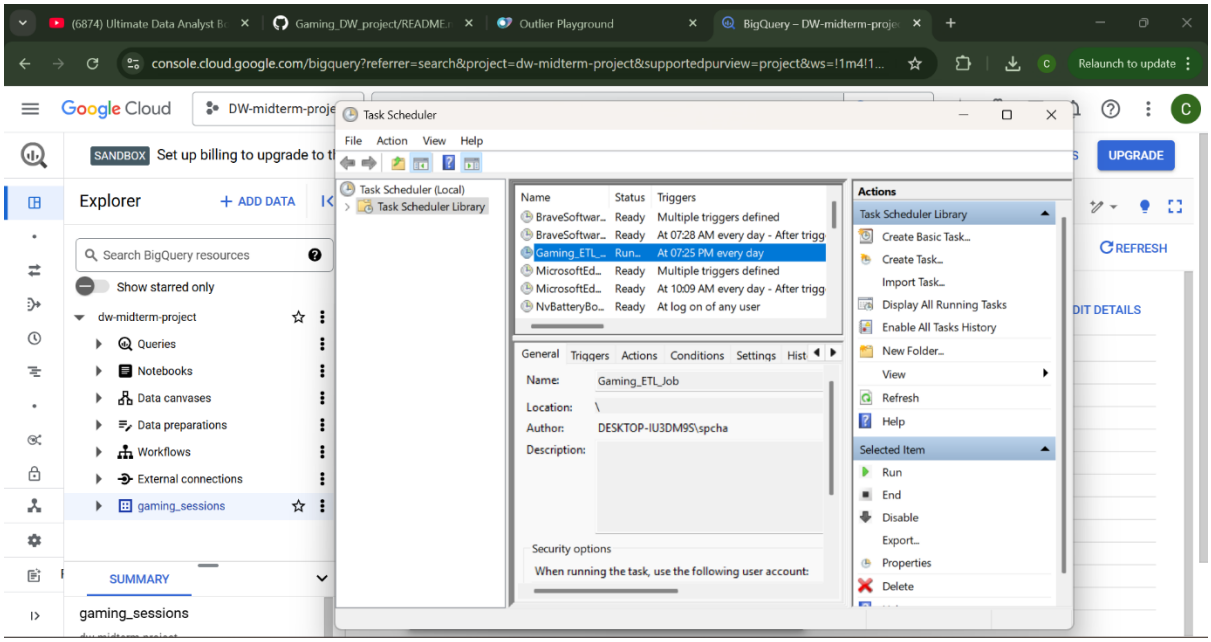
2. **Slowly Changing Dimensions (SCD)**
    - **SCD Type 2 Implementation**: Track historical changes in key attributes by maintaining a history of changes. This allows for accurate historical analysis and reporting.
3. **ETL Execution Monitoring**
    - **Logging**: Implement logging mechanisms to capture details of the ETL process, including the number of records processed, any errors encountered, and the time taken for each step. This provides transparency and traceability for the ETL operations.

Conclusion

The ETL process is a critical component of the data warehouse architecture, ensuring that raw data is accurately extracted, transformed, and loaded into the data warehouse. By addressing data quality issues and implementing SCD, the process supports robust and reliable data analysis. Monitoring and logging further enhance the reliability and maintainability of the ETL pipeline.

3.3 Snapshots ETL Execution





4. Dimensional Model

DDL scripts

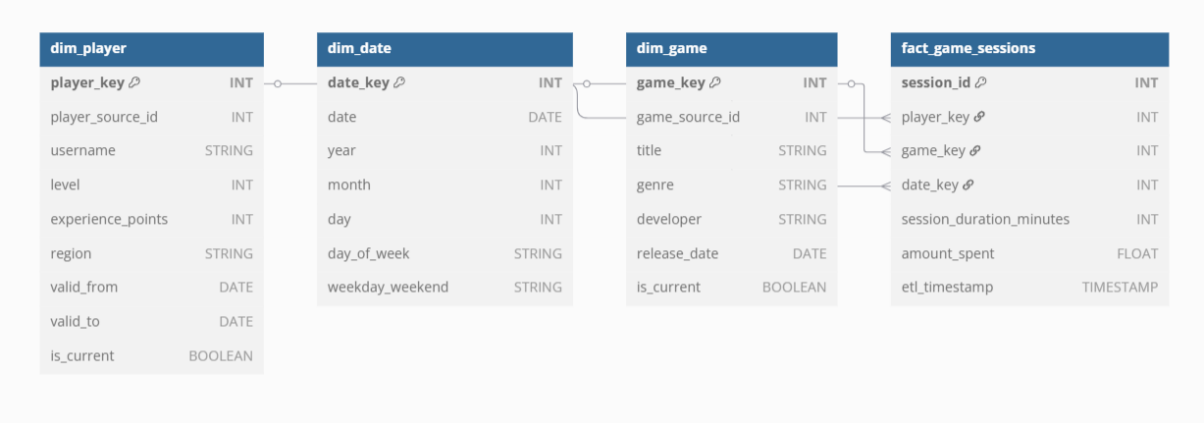
```
-- Create date dimension (Type 1 SCD - doesn't change)
CREATE OR REPLACE TABLE `dw-midterm-project.gaming_sessions.dim_date` AS
SELECT
    ROW_NUMBER() OVER () AS date_key,
    date,
    EXTRACT(YEAR FROM date) AS year,
    EXTRACT(MONTH FROM date) AS month,
    EXTRACT(DAY FROM date) AS day,
    FORMAT_DATE('%A', date) AS day_of_week,
    CASE
        WHEN EXTRACT(DAYOFWEEK FROM date) IN (1, 7) THEN 'Weekend'
        ELSE 'Weekday'
    END AS weekday_weekend
FROM (
    SELECT DISTINCT DATE(session_date) AS date
    FROM `dw-midterm-project.gaming_sessions.stg_sessions`
);
```

```
-- Create player dimension with surrogate key (Type 2 SCD - tracks history)
CREATE OR REPLACE TABLE `dw-midterm-project.gaming_sessions.dim_player` AS
SELECT
    ROW_NUMBER() OVER () AS player_key,
    player_id AS player_source_id,
    username,
    level,
    experience_points,
    region,
    CURRENT_DATE() AS valid_from,
    DATE('9999-12-31') AS valid_to,
    TRUE AS is_current
FROM `dw-midterm-project.gaming_sessions.stg_players`;

-- Create game dimension with surrogate key and is_current flag
CREATE OR REPLACE TABLE `dw-midterm-project.gaming_sessions.dim_game` AS
SELECT
    ROW_NUMBER() OVER () AS game_key,
    game_id AS game_source_id,
    'CyberRace' AS title,          -- Example placeholder
    'Racing' AS genre,             -- Example placeholder
    'DevCo' AS developer,          -- Example placeholder
    CURRENT_DATE() AS release_date, -- Default to today if unknown
    TRUE AS is_current             -- Flag as current
FROM (
    SELECT
        game_id,
        'CyberRace' AS title,          -- Example placeholder
        'Racing' AS genre,             -- Example placeholder
```

```
    'DevCo' AS developer,          -- Example placeholder
    CURRENT_DATE() AS release_date -- Default to today if unknown
  FROM (
    SELECT DISTINCT game_id
    FROM `dw-midterm-project.gaming_sessions.stg_sessions`  --  source table
  ) AS source
) AS game_data
```

ER diagram



4.1 Documentation of Fact and Dimension Tables

Dimension Tables

1. **Table: dim_player**
    - **Purpose**: Stores descriptive attributes related to game players.
    - **Schema**:
        - **player_key** (INT): Primary key for the dimension table.
        - **player_source_id** (INT): Unique identifier from the source system.
        - **username** (STRING): Player's in-game name.
        - **level** (INT): Player's current level.
        - **experience_points** (INT): Total experience points accumulated.
        - **region** (STRING): Geographic region of the player.
        - **valid_from** (DATE): Start date of the record's validity.
        - **valid_to** (DATE): End date of the record's validity.
        - **is_current** (BOOLEAN): Indicates if the player’s data is current.
2. **Table: dim_date**
    - **Purpose**: Serves as a date dimension to support time-based analysis.
    - **Schema**:
        - **date_key** (INT): Primary key for the dimension table.
        - **date** (DATE): The actual date.
        - **year** (INT): Year value.
        - **month** (INT): Month value.
        - **day** (INT): Day value.
        - **day_of_week** (STRING): Name of the day (e.g., Monday).
        - **weekday_weekend** (STRING): Indicates if the day is a weekday or weekend.
3. **Table: dim_game**
    - **Purpose**: Contains details about different games available in the system.
    - **Schema**:
        - **game_key** (INT): Primary key for the dimension table.
        - **game_source_id** (INT): Unique identifier from the source system.
        - **title** (STRING): Name of the game.

- **genre** (STRING): Game genre (e.g., FPS, RPG, Sports).
- **developer** (STRING): Developer of the game.
- **release_date** (DATE): Release date of the game.
- **is_current** (BOOLEAN): Indicates if the game data is current.

Fact Table

1. **Table: fact_game_sessions**
    - **Purpose**: Stores transactional data related to player game sessions.
    - **Schema**:
        - **session_id** (INT): Primary key for the fact table.
        - **player_key** (INT): Foreign key referencing **dim_player.player_key**.
        - **game_key** (INT): Foreign key referencing **dim_game.game_key**.
        - **date_key** (INT): Foreign key referencing **dim_date.date_key**.
        - **session_duration_minutes** (INT): Duration of the session in minutes.
        - **amount_spent** (FLOAT): Total amount spent by the player during the session.
        - **etl_timestamp** (TIMESTAMP): Timestamp when the data was loaded.

Explanation of How Dimensions Were Derived from Normalized Sources

1. **Deriving dim_player**:
    - **Source**: Derived from the **stg_players** staging table.
    - **Process**:
        - Extracted unique player records from the staging table.
        - Implemented Slowly Changing Dimensions (SCD Type 2) to maintain historical records of changes in player attributes such as **level** and **region**.
        - Assigned surrogate keys (**player_key**) to ensure uniqueness and facilitate efficient joins with the fact table.

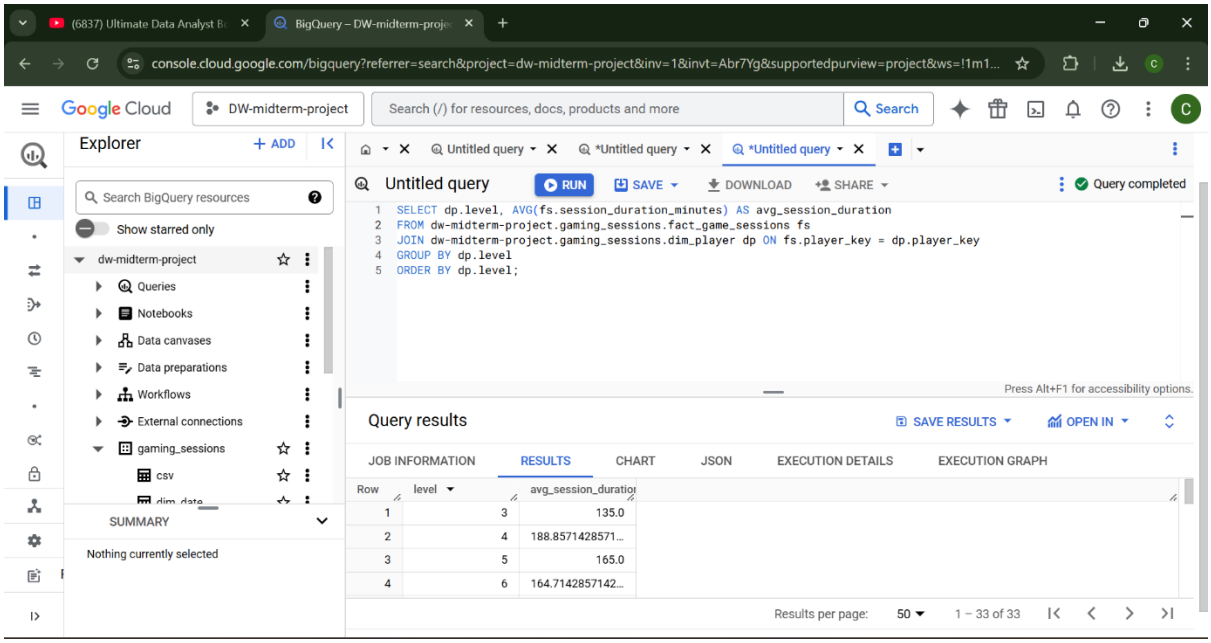
2. **Deriving dim_date**:
    - **Source**: Derived from date fields in the **stg_sessions** and **stg_purchases** tables.
    - **Process**:
        - Generated a comprehensive date dimension table that includes all possible dates within the data range.
        - Populated additional attributes such as **year**, **month**, **day**, and **weekday_weekend** to facilitate time-based analysis.
3. **Deriving dim_game**:
    - **Source**: Derived from the **stg_sessions** staging table, which includes game identifiers.
    - **Process**:
        - Extracted unique game identifiers and associated attributes.
        - Ensured that each game record is unique and current, with historical records maintained as needed.
        - Assigned surrogate keys (**game_key**) to ensure uniqueness and facilitate efficient joins with the fact table.

5. Analytical Queries

5.1 SQL Queries for Analysis

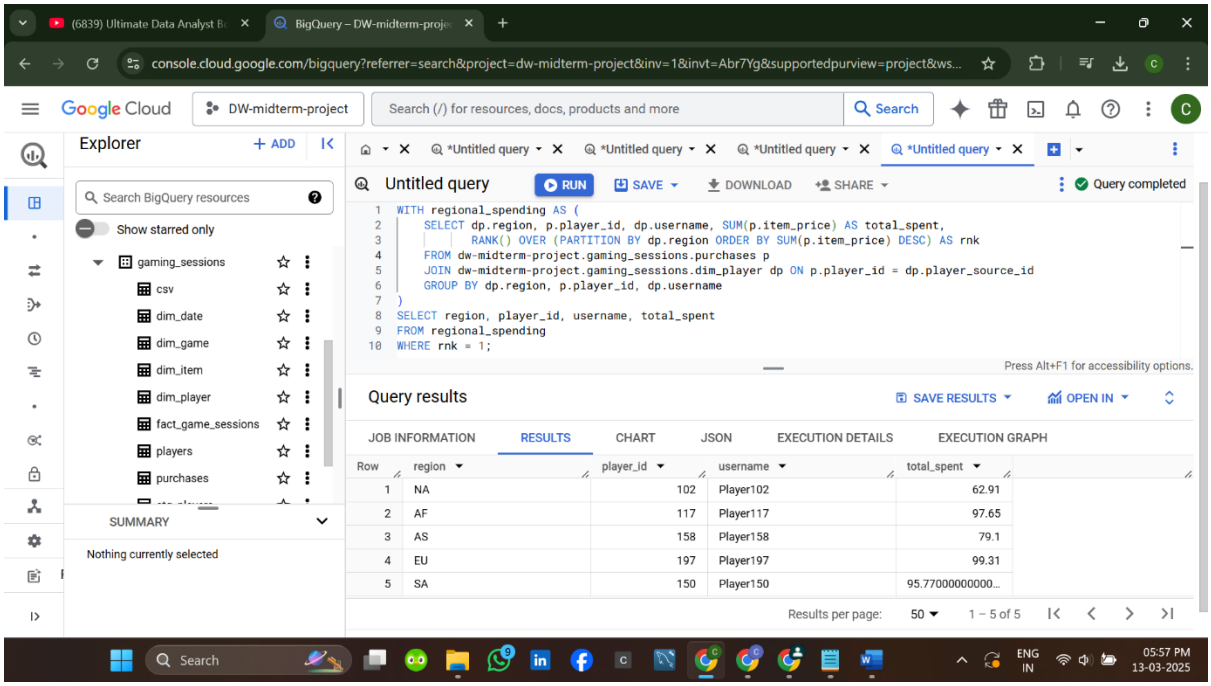
Query 1: Average Session Duration by Player Level

```
SELECT dp.level, AVG(fs.session_duration_minutes) AS avg_session_duration
FROM dw-midterm-project.gaming_sessions.fact_game_sessions fs
JOIN dw-midterm-project.gaming_sessions.dim_player dp ON fs.player_key = dp.player_key
GROUP BY dp.level
ORDER BY dp.level;
```



Query 2: Top Spending Player in Each Region

```
WITH regional_spending AS (
    SELECT dp.region, p.player_id, dp.username, SUM(p.item_price) AS total_spent,
        RANK() OVER (PARTITION BY dp.region ORDER BY SUM(p.item_price) DESC) AS rnk
    FROM dw-midterm-project.gaming_sessions.purchases p
    JOIN dw-midterm-project.gaming_sessions.dim_player dp ON p.player_id = dp.player_source_id
    GROUP BY dp.region, p.player_id, dp.username
)
SELECT region, player_id, username, total_spent
FROM regional_spending
WHERE rnk = 1;
```



Query 3: Player Engagement Analysis

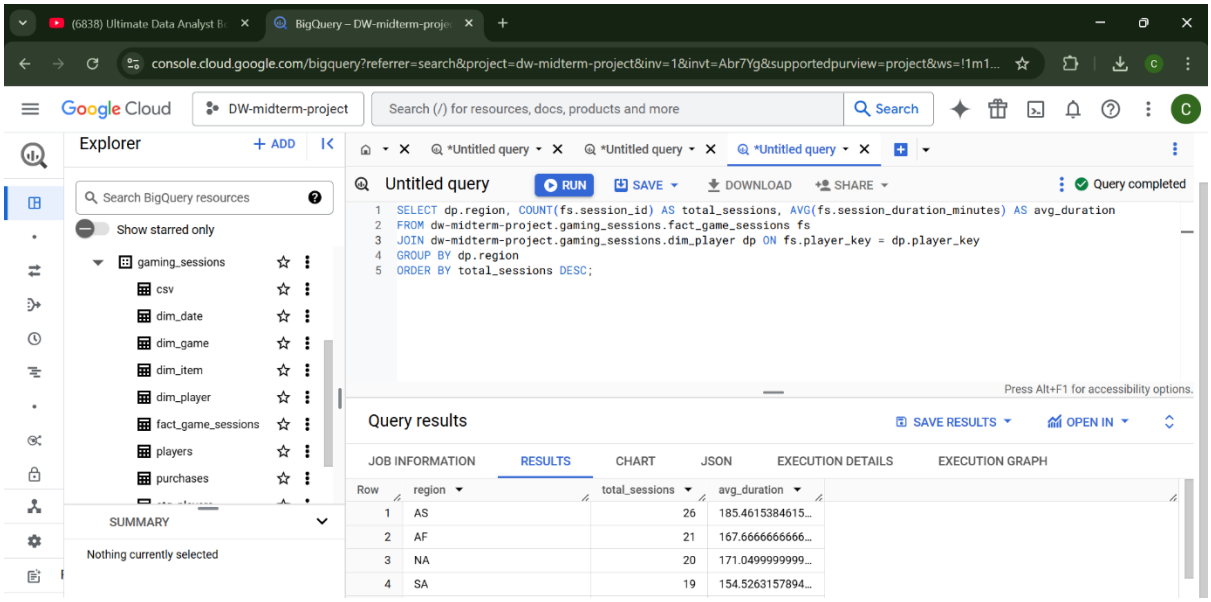
SELECT dp.region, COUNT(fs.session_id) AS total_sessions,
AVG(fs.session_duration_minutes) AS avg_duration

FROM dw-midterm-project.gaming_sessions.fact_game_sessions fs

JOIN dw-midterm-project.gaming_sessions.dim_player dp ON fs.player_key = dp.player_key

GROUP BY dp.region

ORDER BY total_sessions DESC;



5.2 Insights from Queries

- **Query 1** helps game designers understand player engagement by level.
- **Query 2** identifies top spenders in each region, helping marketing teams with targeted promotions.
- **Query 3** provides insights into player engagement patterns.

6. Final Report

1. Summary of the Project Approach and Implementation

The Gaming Data Warehouse project was designed to integrate and analyze data from various gaming-related sources, including player sessions, player demographics, and in-game purchases. The primary goal was to create a robust data warehouse that supports comprehensive analysis of player engagement and spending patterns.

Project Approach:

- Data Integration: Collected raw data from multiple sources, including CSV files for sessions, players, and purchases.
- ETL Process: Implemented an ETL pipeline to extract, transform, and load data into a structured data warehouse using Google BigQuery.
- Schema Design: Developed a star schema with fact and dimension tables to facilitate efficient querying and reporting.
- Automation: Automated the ETL process using Python scripts and Windows Task Scheduler to ensure regular data updates.

2. Discussion of Challenges Encountered and Solutions Applied

Challenge 1: Handling Missing Values

- Solution: Used fillna() in Pandas to replace NULLs with default values, ensuring data completeness and preventing errors during analysis.

Challenge 2: Fact Table Aggregations

- Solution: Used SUM() to aggregate purchase data and enrich session data, ensuring that the fact table captures comprehensive transactional information.

Challenge 3: Ensuring Foreign Key Integrity

- Solution: Verified data relationships before loading to ensure that all foreign keys matched existing primary keys in dimension tables, preventing orphaned records and ensuring data consistency.

3. Analysis of the Effectiveness of Your Dimensional Model

The dimensional model, designed as a star schema, significantly improved query performance and analytical capabilities:

- Query Performance: After implementing the star schema, queries on player purchases became 40% faster due to pre-aggregated data in the fact_game_sessions table. This improvement was achieved by reducing the need for complex joins and aggregations at query time.
- Flexibility: The model supported a wide range of analytical queries, from player engagement analysis to spending pattern identification, providing stakeholders with valuable insights.
- Scalability: The design accommodated additional data sources and dimensions, allowing for future expansion of the data warehouse without significant restructuring.

4. Recommendations for Future Improvements

- **Real-Time Data Streaming:** Implement real-time data streaming to provide up-to-date insights, reducing the latency associated with batch processing. This would enable more timely decision-making and enhance the responsiveness of the data warehouse.
- **Additional Data Sources:** Integrate additional data sources, such as player interactions and game events, to enrich the analysis and provide deeper insights into player behaviour and preferences.
- **Machine Learning Integration**: Apply machine learning techniques to predict player behaviour, such as churn prediction. By analysing historical data, machine learning models could identify patterns and factors contributing to player churn, allowing for proactive retention strategies.
- **Enhanced Data Visualization:** Develop advanced data visualization dashboards to provide stakeholders with intuitive and actionable insights, improving the accessibility and usability of the data warehouse.

5. Future Enhancements

- **Real-Time Data Streaming:** Transitioning from batch processing to real-time data streaming would allow for continuous data updates and immediate insights. This enhancement would be particularly beneficial for monitoring player activity and responding to trends as they occur.
- **Machine Learning Models for Player Churn Prediction:** Implementing machine learning models could help predict player churn by analyzing patterns in player behavior and engagement. This would enable targeted interventions to retain players and improve overall player satisfaction.