**DAY- 8(Data Warehousing)**

**TOPIC: Data Warehousing Fundamentals**

1. Design a data warehouse schema for a retail company that includes dimension tables for products, customers, and time. Implement the schema using a relational database management system (RDBMS) of your choice.

Schema Design:

* Product Dimension Table:
  + product\_id (Primary Key)
  + product\_name
  + category
  + brand
  + price
* Customer Dimension Table:
  + customer\_id (Primary Key)
  + customer\_name
  + address
  + email
* Time Dimension Table:
  + date\_id (Primary Key)
  + date
  + day\_of\_week
  + month
  + quarter
  + year
* Sales Fact Table:
  + sales\_id (Primary Key)
  + product\_id (Foreign Key referencing Product Dimension Table)
  + customer\_id (Foreign Key referencing Customer Dimension Table)
  + date\_id (Foreign Key referencing Time Dimension Table)
  + quantity
  + revenue

CREATE TABLE ProductDimension (

product\_id INT IDENTITY(1,1) PRIMARY KEY,

product\_name VARCHAR(255),

category VARCHAR(50),

brand VARCHAR(50),

price DECIMAL(10, 2),

-- Additional columns

);

CREATE TABLE CustomerDimension (

customer\_id INT IDENTITY(1,1) PRIMARY KEY,

customer\_name VARCHAR(255),

address VARCHAR(255),

email VARCHAR(255)

);

CREATE TABLE TimeDimension (

date\_id INT IDENTITY(1,1) PRIMARY KEY,

date DATE,

day\_of\_week VARCHAR(20),

month VARCHAR(20),

quarter VARCHAR(20),

year INT);

CREATE TABLE SalesFact (

sales\_id INT IDENTITY(1,1) PRIMARY KEY,

product\_id INT FOREIGN KEY REFERENCES ProductDimension(product\_id),

customer\_id INT FOREIGN KEY REFERENCES CustomerDimension(customer\_id),

date\_id INT FOREIGN KEY REFERENCES TimeDimension(date\_id),

quantity INT,

revenue DECIMAL(10, 2)

);

2. Create a fact table that captures sales data, including product ID, customer ID, date, and sales amount. Populate the fact table with sample data.

3. Write SQL queries to retrieve sales data from the data warehouse, including aggregations and filtering based on different dimensions.

1. Retrieve total revenue for each product:

SELECT

p.product\_id,

p.product\_name,

SUM(s.revenue) AS total\_revenue

FROM

SalesFact s

INNER JOIN ProductDimension p ON s.product\_id = p.product\_id

GROUP BY

p.product\_id, p.product\_name;

1. Retrieve total revenue for each customer:

SELECT

c.customer\_id,

c.customer\_name,

SUM(s.revenue) AS total\_revenue

FROM

SalesFact s

INNER JOIN CustomerDimension c ON s.customer\_id = c.customer\_id

GROUP BY

c.customer\_id, c.customer\_name;

1. Retrieve total revenue for each product category in a specific year:

SELECT

p.category,

SUM(s.revenue) AS total\_revenue

FROM

SalesFact s

INNER JOIN ProductDimension p ON s.product\_id = p.product\_id

INNER JOIN TimeDimension t ON s.date\_id = t.date\_id

WHERE

t.year = 2023 r

GROUP BY

P.category;

**TOPIC: ETL and Data Integration**

1. Design an ETL process using a programming language (e.g., Python) to extract data from a source system (e.g., CSV files), transform it by applying certain business rules or calculations, and load it into a data warehouse.

**Extract Data from CSV Files:**

import pandas as pd

# Define the path to the CSV files

csv\_file\_path = "path/to/csv/files/"

# Read the CSV files into Pandas DataFrames

df\_products = pd.read\_csv(csv\_file\_path + "products.csv")

df\_customers = pd.read\_csv(csv\_file\_path + "customers.csv")

df\_sales = pd.read\_csv(csv\_file\_path + "sales.csv")

**Transform Data:**

# Add a new column to calculate total revenue for each sales record

df\_sales['total\_revenue'] = df\_sales['quantity'] \* df\_sales['price']

# Merge the DataFrames to combine relevant information

df\_merged = pd.merge(df\_sales, df\_products, on='product\_id', how='inner')

df\_merged = pd.merge(df\_merged, df\_customers, on='customer\_id', how='inner')

# Drop unnecessary columns

df\_transformed = df\_merged.drop(['price'], axis=1)

# Rename columns if necessary

df\_transformed = df\_transformed.rename(columns={'product\_name': 'product', 'customer\_name': 'customer'})

**Load Data into the Data Warehouse:**

import sqlalchemy

# Define the connection string to the data warehouse (replace with your own connection details)

database\_connection = "postgresql://username:password@localhost:5432/database\_name"

# Establish a connection to the data warehouse

engine = sqlalchemy.create\_engine(database\_connection)

# Load the transformed data into the data warehouse

df\_transformed.to\_sql('sales\_fact', engine, if\_exists='replace', index=False)

2. Implement the ETL process by writing code that performs the extraction, transformation, and loading steps.

import pandas as pd

import sqlalchemy

# Define the path to the CSV files

csv\_file\_path = "path/to/csv/files/"

# Define the connection string to the data warehouse (replace with your own connection details)

database\_connection = "postgresql://username:password@localhost:5432/database\_name"

# Extract Data from CSV Files

df\_products = pd.read\_csv(csv\_file\_path + "products.csv")

df\_customers = pd.read\_csv(csv\_file\_path + "customers.csv")

df\_sales = pd.read\_csv(csv\_file\_path + "sales.csv")

# Transform Data

df\_sales['total\_revenue'] = df\_sales['quantity'] \* df\_sales['price']

df\_merged = pd.merge(df\_sales, df\_products, on='product\_id', how='inner')

df\_merged = pd.merge(df\_merged, df\_customers, on='customer\_id', how='inner')

df\_transformed = df\_merged.drop(['price'], axis=1)

df\_transformed = df\_transformed.rename(columns={'product\_name': 'product', 'customer\_name': 'customer'})

# Load Data into the Data Warehouse

engine = sqlalchemy.create\_engine(database\_connection)

df\_transformed.to\_sql('sales\_fact', engine, if\_exists='replace', index=False)

**TOPIC: Dimensional Modeling and Schemas**

1. Design a star schema for a university database, including a fact table for student enrollments and dimension tables for students, courses, and time. Implement the schema using a database of your choice.

Schema Design:

* Student Dimension Table:
  + student\_id (Primary Key)
  + student\_name
  + student\_age
  + student\_major
* Course Dimension Table:
  + course\_id (Primary Key)
  + course\_name
  + course\_department
  + course\_credits
* Time Dimension Table:
  + date\_id (Primary Key)
  + date
  + day\_of\_week
  + month
  + quarter
  + year
* Enrollment Fact Table:
  + enrollment\_id (Primary Key)
  + student\_id (Foreign Key referencing Student Dimension Table)
  + course\_id (Foreign Key referencing Course Dimension Table)
  + date\_id (Foreign Key referencing Time Dimension Table)
  + grade

CREATE TABLE StudentDimension (

student\_id INT IDENTITY(1,1) PRIMARY KEY,

student\_name VARCHAR(255),

student\_age INT,

student\_major VARCHAR(50)

);

CREATE TABLE CourseDimension (

course\_id INT IDENTITY(1,1) PRIMARY KEY,

course\_name VARCHAR(255),

course\_department VARCHAR(50),

course\_credits INT

);

CREATE TABLE TimeDimension (

date\_id INT IDENTITY(1,1) PRIMARY KEY,

date DATE,

day\_of\_week VARCHAR(20),

month VARCHAR(20),

quarter VARCHAR(20),

year INT

);

CREATE TABLE EnrollmentFact (

enrollment\_id INT IDENTITY(1,1) PRIMARY KEY,

student\_id INT FOREIGN KEY REFERENCES StudentDimension(student\_id),

course\_id INT FOREIGN KEY REFERENCES CourseDimension(course\_id),

date\_id INT FOREIGN KEY REFERENCES TimeDimension(date\_id),

grade VARCHAR(2)

);

2. Write SQL queries to retrieve data from the star schema, including aggregations and joins between the fact table and dimension tables.

**Retrieve total enrollments for each course:**

SELECT

cd.course\_id,

cd.course\_name,

COUNT(\*) AS total\_enrollments

FROM

EnrollmentFact ef

INNER JOIN CourseDimension cd ON ef.course\_id = cd.course\_id

GROUP BY

cd.course\_id, cd.course\_name;

**Retrieve the average grade for each student:**

SELECT

sd.student\_id,

sd.student\_name,

AVG(CAST(ef.grade AS FLOAT)) AS average\_grade

FROM

EnrollmentFact ef

INNER JOIN StudentDimension sd ON ef.student\_id = sd.student\_id

GROUP BY

sd.student\_id, sd.student\_name;

**Retrieve the number of enrollments by quarter and year:**

SELECT

td.quarter,

td.year,

COUNT(\*) AS enrollments\_count

FROM

EnrollmentFact ef

INNER JOIN TimeDimension td ON ef.date\_id = td.date\_id

GROUP BY

td.quarter, td.year;

**TOPIC: Performance Optimization and Querying**

1. Scenario: You need to improve the performance of your data loading process in the data warehouse. Write a Python script that implements the following optimizations:

1. Utilize batch processing techniques to load data in bulk instead of individual row insertion.

b) Implement multi-threading or multiprocessing to parallelize the data loading process.

c) Measure the time taken to load a specific amount of data before and after implementing these optimizations.

**import pandas as pd**

**import time**

**from concurrent.futures import ThreadPoolExecutor, ProcessPoolExecutor**

**import sqlalchemy**

**# Define the path to the CSV file**

**csv\_file\_path = "path/to/csv/file.csv"**

**# Define the connection string to the data warehouse (replace with your own connection details)**

**database\_connection = "postgresql://username:password@localhost:5432/database\_name"**

**# Define the batch size for batch processing**

**batch\_size = 1000**

**# Function to load data in batches**

**def load\_data\_in\_batches(data, table\_name):**

**engine = sqlalchemy.create\_engine(database\_connection)**

**with engine.begin() as connection:**

**for i in range(0, len(data), batch\_size):**

**batch\_data = data[i:i+batch\_size]**

**batch\_data.to\_sql(table\_name, connection, if\_exists='append', index=False)**

**# Function to load data using multi-threading**

**def load\_data\_with\_multithreading(data, table\_name, num\_threads):**

**with ThreadPoolExecutor(max\_workers=num\_threads) as executor:**

**futures = []**

**for i in range(0, len(data), batch\_size):**

**batch\_data = data[i:i+batch\_size]**

**future = executor.submit(load\_data\_in\_batches, batch\_data, table\_name)**

**futures.append(future)**

**# Wait for all threads to complete**

**for future in futures:**

**future.result()**

**# Function to load data using multiprocessing**

**def load\_data\_with\_multiprocessing(data, table\_name, num\_processes):**

**with ProcessPoolExecutor(max\_workers=num\_processes) as executor:**

**futures = []**

**for i in range(0, len(data), batch\_size):**

**batch\_data = data[i:i+batch\_size]**

**future = executor.submit(load\_data\_in\_batches, batch\_data, table\_name)**

**futures.append(future)**

**# Wait for all processes to complete**

**for future in futures:**

**future.result()**

**# Measure the time taken to load data**

**def measure\_loading\_time(data, table\_name):**

**start\_time = time.time()**

**load\_data\_in\_batches(data, table\_name)**

**end\_time = time.time()**

**loading\_time = end\_time - start\_time**

**print(f"Time taken to load data without optimization: {loading\_time} seconds")**

**start\_time = time.time()**

**load\_data\_with\_multithreading(data, table\_name, num\_threads=4)**

**end\_time = time.time()**

**loading\_time = end\_time - start\_time**

**print(f"Time taken to load data with multi-threading: {loading\_time} seconds")**

**start\_time = time.time()**

**load\_data\_with\_multiprocessing(data, table\_name, num\_processes=4)**

**end\_time = time.time()**

**loading\_time = end\_time - start\_time**

**print(f"Time taken to load data with multiprocessing: {loading\_time} seconds")**

**# Read the data from the CSV file**

**df\_data = pd.read\_csv(csv\_file\_path)**

**# Measure the time taken to load data before and after optimization**

**measure\_loading\_time(df\_data, "sales\_fact")**

**Submission Guidelines:**

1. Answer all the questions in a single Jupyter Notebook file (.ipynb).

2. Include necessary code, comments, and explanations to support your answers and implementation.

3. Ensure the notebook runs without errors and is well-organized.

4. Create a GitHub repository to host your assignment files.

5. Rename the Jupyter Notebook file using the format "date\_month\_topic.ipynb" (e.g., "12\_July\_DataWarehousing.ipynb").

6. Place the Jupyter Notebook file in the repository.

7. Commit and push any additional files or resources required to run your code (if applicable) to the repository.

8. Ensure the repository is publicly accessible.

9. Submit the link to your GitHub repository as the assignment submission.

**Grading Criteria:**

1. Understanding and completeness of answers: 40%

2. Clarity and depth of explanations: 25%

3. Correct implementation and evaluation of matrix operations: 15%

4. Proper code implementation and organization: 10%

5. Overall presentation and adherence to guidelines: 10%

**Note:- Create your assignment in Jupyter notebook and upload it to GitHub & share that uploaded assignment file link through your dashboard. Make sure the repository is public.**