Author: Madhurima Rawat

Designing and Implementing a Data Warehouse Report

This experiment involves designing and executing SQL queries to generate insightful reports from a data warehouse, utilizing business intelligence tools for data analysis and visualization.

📝 What is Business Intelligence (BI)?

Business Intelligence (BI) refers to a set of technologies, tools, and practices that collect, integrate, analyze, and present business data. The goal is to help organizations make **data-driven decisions** by transforming raw data into actionable insights.

🛠️ What are BI Tools?

BI Tools are software applications used to analyze an organization's raw data. These tools help in:

- Extracting data from multiple sources (like databases, data warehouses, etc.)
- Processing and cleaning data
- Creating **interactive reports**, **dashboards**, and **visualizations**
- Identifying trends, patterns, and insights that support **strategic business decisions**

Common BI Tools Include:

- Power BI
- Tableau
- QlikView
- Python (when used with libraries like pandas, matplotlib, seaborn, etc.)
- SQL (for querying and preparing data)

In this experiment, **Python** was used as the BI tool to analyze and visualize data, replacing traditional BI platforms.

✅ Why is BI Useful?

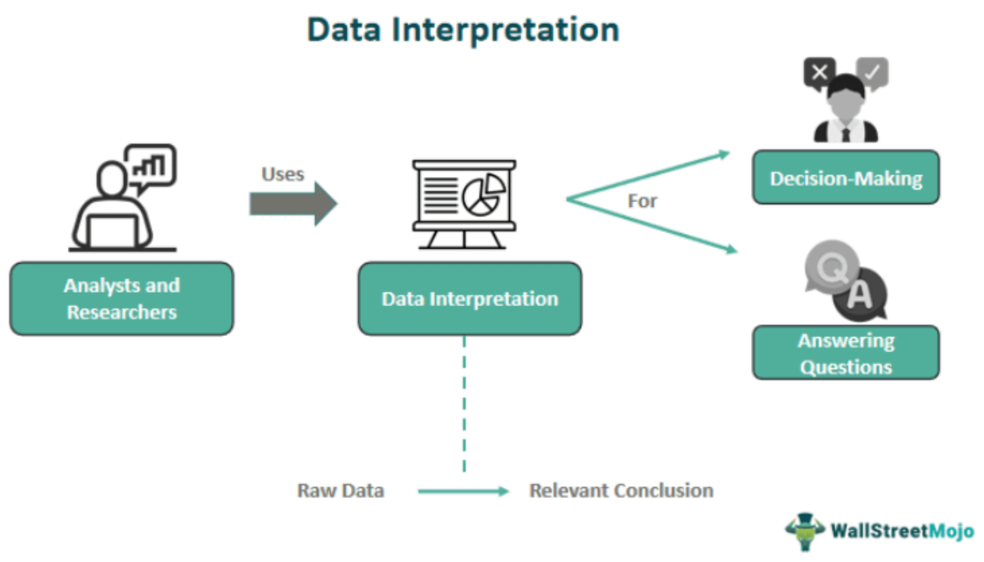
1. **Informed Decision-Making**
   BI provides data-driven insights, enabling businesses to make better strategic decisions.

2. **Improved Efficiency**
   BI tools automate data collection and reporting processes, saving time and resources.

3. **Competitive Advantage**
   By analyzing market trends and customer behavior, organizations can stay ahead of competitors.

4. **Better Customer Insights**
   Understanding customer preferences leads to improved products and services.

5. **Data Visualization**
   Visual dashboards and reports make complex data easy to understand for stakeholders at all levels.



🌍 Real-Life Applications of BI Tools

1. Retail Sector

- **Example**: Walmart uses BI tools to analyze customer purchase behavior.
- **Outcome**: Optimizes inventory management, predicts demand, and personalizes marketing strategies.

2. Healthcare

- **Example**: Hospitals use BI to monitor patient data and predict disease outbreaks.
- **Outcome**: Improves patient care and resource management.

3. Banking and Finance

- **Example**: Banks use BI to detect fraudulent transactions by analyzing transaction patterns.
- **Outcome**: Reduces fraud risk and enhances customer trust.

4. Education

- **Example**: Universities use BI to track student performance and optimize course offerings.
- **Outcome**: Improves student retention and learning outcomes.

5. Manufacturing

- **Example**: Companies analyze production data to detect bottlenecks in the supply chain.
- **Outcome**: Enhances productivity and reduces downtime.

🔍 Case Study: Netflix

Netflix uses BI tools and advanced analytics to:

- Recommend personalized content to users.
- Analyze viewing patterns to create original shows.
- Optimize streaming quality based on location and bandwidth.

Result: Increased user engagement and retention, making Netflix one of the most successful streaming platforms globally.

🚀 Conclusion

BI tools, whether traditional software like **Power BI** and **Tableau** or programming-based solutions like **Python**, are essential in today's data-driven world. They transform vast amounts of raw data into meaningful insights, helping organizations make informed decisions, streamline operations, and gain a competitive edge.

BI using MySQL and Python

1. Create a Database

```sql
CREATE DATABASE linkedin_data;
```

✅ Explanation:

- This command creates a new database named `linkedin_data`.
- A database is like a container where tables and other database objects are stored.

🖥️ Output:

```
Query OK, 1 row affected (0.05 sec)
```

🔎 Significance:

- The message confirms that the database was successfully created.
- The `1 row affected` indicates the operation was successful and took approximately `0.05` seconds.

2. Select and Use the Database

```sql
USE linkedin_data;
```

✅ Explanation:

- Switches to the `linkedin_data` database so that all subsequent commands apply to this database.

🖥️ Output:

```
Database changed
```

🔎 Significance:

- Confirms you're now working inside the `linkedin_data` database.

3. Create a Table

```sql
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    followers INT,
    content_type VARCHAR(100),
    gender VARCHAR(10),
    age INT,
    organization VARCHAR(255)
);
```

✅ Explanation:

- Creates a table named `users` inside the `linkedin_data` database.
- Each column stores specific types of data:
    - `id` : Unique identifier for each user, auto-incremented.
    - `name` : Name of the user (up to 255 characters).
    - `followers` : Number of followers they have.
    - `content_type` : Type of content they create (text, video, etc.).
    - `gender` : Gender of the user.
    - `age` : Age of the user.
    - `organization` : Company/organization they are associated with.

💻 Output:

```
Query OK, 0 rows affected (0.05 sec)
```

🔎 Significance:

- Confirms that the `users` table was successfully created.
- `0 rows affected` means no data rows were changed because the command only created the table structure.

4. Retrieve Insights Using Aggregation

```
SELECT
    content_type,
    AVG(followers) AS avg_followers,
    COUNT(*) AS total_users
FROM users
GROUP BY content_type;
```

✅ Explanation:

- Retrieves:
    - The **type of content** each user creates (`content_type`).
    - The **average number of followers** for each content type (`AVG(followers)`).
    - The **total number of users** creating that content (`COUNT(*)`).
- `GROUP BY content_type` groups the data by each unique content type.

💻 Output:

```
Empty set (0.02 sec)
```

🔎 **Significance:**

- No data has been inserted into the `users` table yet, so there's nothing to analyze.
- Once data is added, this query would provide valuable insights into which content type is more popular or has more followers.

5. Check the Current User

```
SELECT USER();
```

✅ **Explanation:**

- Displays the username and host of the currently connected user.

💻 **Output:**

```
+----------------+
| USER()         |
+----------------+
| username@localhost |
+----------------+
```

🔎 **Significance:**

- Shows you're connected as the `username` user from the `localhost` machine.
- Important for knowing your current privileges (username typically has full access).

6. List All Databases

```
SHOW DATABASES;
```

✅ **Explanation:**

- Displays all databases available on the MySQL server.

💻 **Output (example):**

```
+---------------------+
| Database            |
+---------------------+
| classdb             |
| company             |
| employee           |
| hospital           |
| information_schema |
| linkedin_data      |
| movie              |
| mydatabase         |
| mysql              |
| olap               |
| performance_schema |
| retaildatawarehouse |
| username                |
| shopping           |
| student            |
| sys                |
| utd                |
+--------------------+
17 rows in set (0.07 sec)
```

🔎 **Significance:**

- Lists all the databases you have access to.
- Confirms that your newly created `linkedin_data` database exists.

7. Switch to the `linkedin_data` Database Again

```
USE linkedin_data;
```

✅ **Explanation:**

- Reconfirms the current active database (just to ensure you're in the correct database).

💻 **Output:**

```
Database changed
```

🔎 **Significance:**

- Ensures any further commands apply to `linkedin_data`.

8. Show Tables Inside the Database

```
SHOW TABLES;
```

✅ **Explanation:**

- Lists all tables in the current active database (`linkedin_data`).

💻 **Output:**

```
+-------------------------+
| Tables_in_linkedin_data |
+-------------------------+
| users                   |
+-------------------------+
1 row in set (0.03 sec)
```

🔎 **Significance:**

- Confirms the `users` table was successfully created and exists in the `linkedin_data` database.

9. Grant Privileges to the username User

```
GRANT ALL PRIVILEGES ON linkedin_data.* TO 'username'@'localhost';
```

✅ **Explanation:**

- Gives full permissions (`ALL PRIVILEGES`) on the `linkedin_data` database and all its tables to the `username` user accessing from `localhost` .

💻 **Output:**

```
Query OK, 0 rows affected (0.01 sec)
```

🔎 **Significance:**

- No rows were changed (expected for a privilege command).
- Ensures the `username` user has unrestricted access to manage the database.

10. Apply Privilege Changes Immediately

```
FLUSH PRIVILEGES;
```

✅ **Explanation:**

- Forces MySQL to reload the grant tables.
- Applies any privilege changes made by `GRANT`, `REVOKE`, or direct updates to the user tables.

🖥️ Output:

```
Query OK, 0 rows affected (0.01 sec)
```

🔎 Significance:

- Confirms privilege changes are active without needing to restart the MySQL server.

15. Check MySQL Server Port Number

```
SHOW VARIABLES LIKE 'port';
```

✅ Explanation:

- Retrieves the current port number on which the MySQL server is listening.
- Useful to confirm or troubleshoot the connection configuration.

🖥️ Output:

```
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| port          | 3305  |
+---------------+-------+
1 row in set (0.03 sec)
```

🔎 Significance:

- Ensures you are connecting to the correct port.
- Helpful when running multiple MySQL instances or using non-default ports.

16. Change username User Password and Authentication Plugin

```
ALTER USER 'username'@'localhost' IDENTIFIED WITH
mysql_native_password BY 'your_password';
```

✅ Explanation:

- Updates the `username` user password to `your_password` .
- Changes the authentication plugin to `mysql_native_password` for compatibility with older clients.

🖥️ Output:

```
Query OK, 0 rows affected (0.01 sec)
```

🔎 Significance:

- Secures the username account with a new password.
- Ensures compatibility with applications that do not support the default `caching_sha2_password` plugin.

17. Apply Privilege Changes Immediately

```
FLUSH PRIVILEGES;
```

✅ Explanation:

- Reloads MySQL's in-memory privilege tables.
- Ensures that user, permission, or authentication changes take effect immediately.

🖥️ Output:

```
Query OK, 0 rows affected (0.00 sec)
```

🔎 Significance:

- Critical after manual modifications to user tables.
- No need to restart the MySQL server for changes to apply.

18. Insert Multiple Records into the Users Table

```
INSERT INTO users (name, followers, content_type,
gender, age, organization)
VALUES
('Alice Johnson', 8500, 'Blog', 'Female', 28, 'Google'),
('Bob Smith', 4300, 'Video', 'Male', 35, 'Microsoft'),
('Charlie Brown', 9200, 'Article', 'Male', 41, 'Amazon'),
('David Lee', 1200, 'Post', 'Male', 24, 'Meta'),
('Emily Davis', 6700, 'Infographic', 'Female', 30, 'Tesla'),
('Fatima Khan', 3100, 'Video', 'Female', 27, 'OpenAI'),
('George Martin', 1500, 'Blog', 'Male', 37, 'Google'),
('Hannah White', 4900, 'Post', 'Female', 22, 'Microsoft'),
('Ivan Petrov', 7400, 'Article', 'Male', 45, 'Amazon'),
('Jasmine Li', 6100, 'Infographic', 'Female', 33, 'Meta');
```

✅ Explanation:

- Adds 10 new records to the `users` table with various attributes.
- Populates the table with sample data for testing or analysis.

💻 Output:

```
Query OK, 10 rows affected (0.00 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

🔎 Significance:

- Provides a dataset for running SELECT, UPDATE, and DELETE queries.
- Useful for demonstrating or testing MySQL queries and functions.

19. View Table Structure of the Users Table

```
DESCRIBE users;
```

✅ Explanation:

- Displays the column definitions of the `users` table.
- Shows data types, NULL constraints, keys, defaults, and extra attributes like auto-increment.

💻 Output:

```
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| id           | int          | NO   | PRI | NULL    | auto_increment |
| name         | varchar(255) | YES  |     | NULL    |                |
| followers    | int          | YES  |     | NULL    |                |
| content_type | varchar(100) | YES  |     | NULL    |                |
| gender       | varchar(10)  | YES  |     | NULL    |                |
| age          | int          | YES  |     | NULL    |                |
| organization | varchar(255) | YES  |     | NULL    |                |
+--------------+-------------+-----+-----+---------+----------------+
7 rows in set (0.02 sec)
```

🔎 Significance:

- Essential for understanding the table schema before inserting or querying data.
- Ensures you use the correct data types and column names in your SQL statements.

The Python script was used to send data and generate plots. For the complete code and detailed explanations, visit the notebook: *Designing and Implementing a Data Warehouse Report.ipynb*