**Data Warehouse Project for Anime Recommendation Database (MySQL)**

This project will guide you through creating a **data warehouse** using MySQL based on the **Anime Recommendation Database (2020)** from Kaggle. The data will be organized across multiple layers, and we will use procedures, triggers, views, and different tables to build a well-structured system. Let’s break it down step by step.

**Project Overview**

1. **Dataset**:  
   We will use the Anime Recommendation Database, which contains information about anime ratings, genres, users, etc.  
   [Anime Recommendation Database 2020](https://www.kaggle.com/datasets/hernan4444/anime-recommendation-database-2020)
2. **Goal**:  
   Build a **Data Warehouse** with different layers of data. These layers will hold the data as we move it from raw, unprocessed data to final, usable data.
3. **Layers of the Warehouse**:
   * **Raw Layer**: The original, unprocessed data (optional).
   * **Stage Layer**: Cleaned and normalized data (ready for analysis).
   * **Hist Layer**: Historical data where we track changes over time using Slowly Changing Dimensions (SCD Type 2).
4. **Schema Type**:  
   We will organize the tables using either a **Star Schema** or a **Snowflake Schema**, depending on the relationships between the tables.

**Steps for Building the Data Warehouse**

**1. Raw Layer (Optional)**

The **Raw Layer** holds the data exactly as you download it from Kaggle. You might skip this layer if you want to jump directly to cleaning the data.

**2. Stage Layer**

In the **Stage Layer**, we clean and prepare the data:

* **Cleaning**: Handle missing values (nulls), remove unwanted columns, and ensure data types are correct.
* **Normalization**: Separate data into different tables (e.g., one table for users, one table for anime).

We’ll store this cleaned data in **Stage Tables**.

Example of tables in the Stage Layer:

* **Stage\_Anime**: Contains anime information (title, genre, episodes, etc.).
* **Stage\_User**: Contains user information (username, age, location, etc.).
* **Stage\_Ratings**: Contains ratings given by users to anime (user\_id, anime\_id, rating).

**3. Hist Layer**

The **Hist Layer** stores historical data. Here, we track changes over time using Slowly Changing Dimensions (SCD Type 2), which means when data about an anime or user changes, we keep the old version and add a new version.

Example of tables in the Hist Layer:

* **Hist\_Anime**: Same as Stage\_Anime, but with extra columns for tracking versions.
* **Hist\_User**: Same as Stage\_User, with tracking for changes.
* **Fact\_Ratings**: The fact table that contains anime ratings linked to users and anime.

The data is moved from **Stage Layer to Hist Layer** using **Stored Procedures**.

**Star Schema Design (Example)**

In a Star Schema, you have:

* **Fact Tables**: Store measurable data (like ratings).
* **Dimension Tables**: Store details (like anime titles, user info).

Example of tables:

* **Fact\_Ratings**: Contains all anime ratings.
  + Columns: anime\_id, user\_id, rating, timestamp.
* **Dim\_Anime**: Stores information about anime.
  + Columns: anime\_id, title, genre, episodes, release\_year.
* **Dim\_User**: Stores information about users.
  + Columns: user\_id, username, age\_group.

**Implementation Steps**

**Step 1: Cleaning Data (Raw to Stage)**

1. **Extract the Data** from Kaggle and load it into the Raw Layer (optional).
2. **Clean and Normalize** the data before moving it into the Stage Layer.
   * Remove unwanted columns or rows with too many missing values.
   * Normalize the data into different tables (e.g., user data in one table, anime data in another).

**Step 2: Move Data to Hist Layer**

1. **Insert the data** from the Stage Layer to the Hist Layer, keeping track of any changes in the data.
2. **Slowly Changing Dimensions (SCD Type 2)**:
   * This is a method of tracking changes. For example, if the title or genre of an anime changes, we add a new row in the **Hist\_Anime** table and mark the old row as expired.

**Step 3: Create Triggers**

Triggers automatically execute actions when something happens in the database. You need to create 3 triggers:

* **After Insert**: Automatically log when a new record is added.
* **After Update**: Automatically update history when data changes.
* **After Delete**: Automatically log when a record is deleted.

**Step 4: Views and Aggregations**

You will create views to filter and aggregate data:

* **Filter Views**: Show only specific data (e.g., anime released after 2015).
* **Aggregate Views**: Summarize data (e.g., average ratings for each anime).