# Forge of LLM Empires

**Chungman Lee**
**23205535**
5115076a@gmail.com

## Table of Contents

## List of Figures

# 1. Introduction

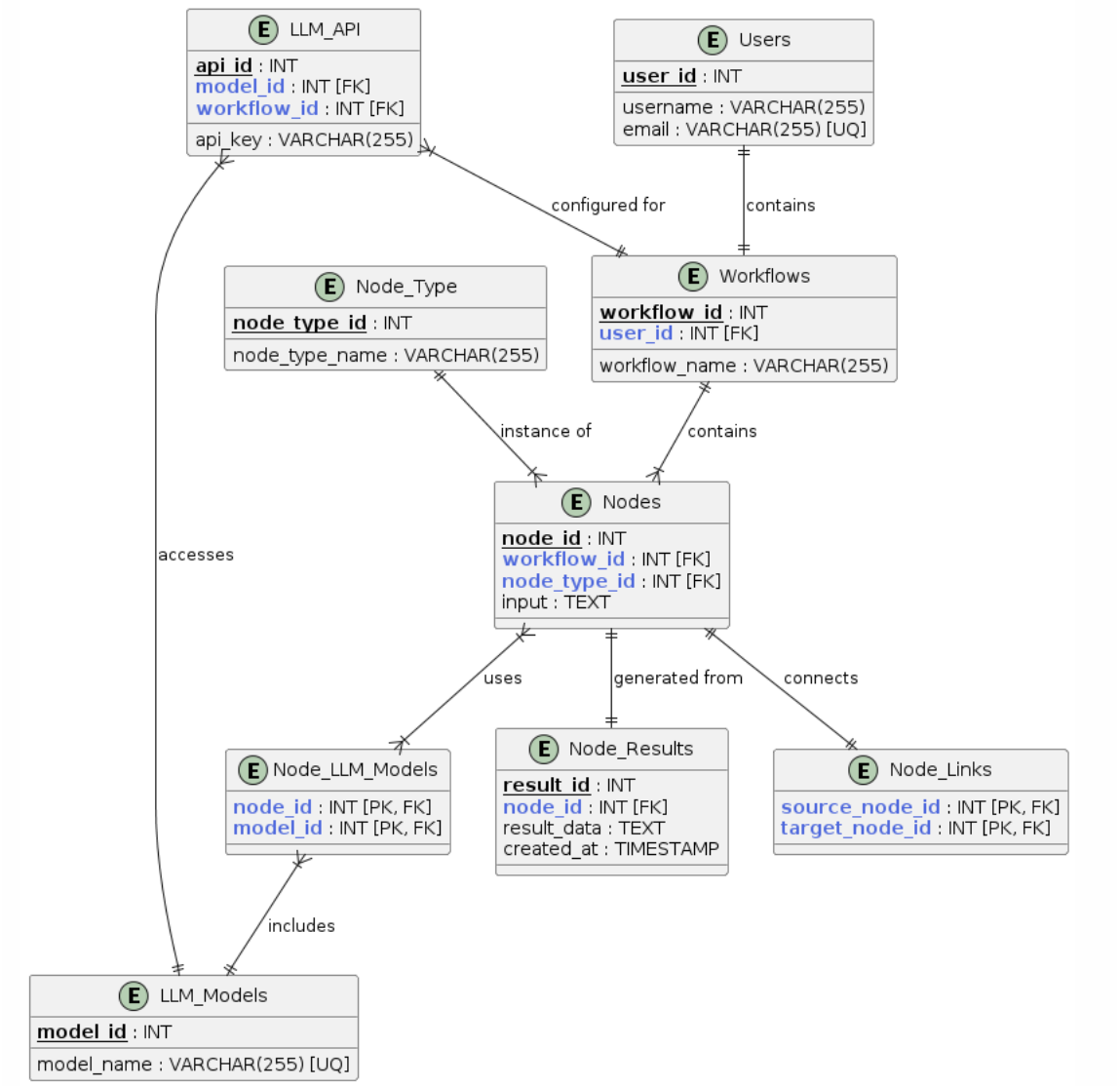
The domain of this project is database structure with sample datasets of ChainForge.

The data of this project is generated with python code. The tables of this database consist of Users, LLM_Models that can be used in the ChainForge, LLM_API to store API keys of users, Workflows that is used by users, nodes that is generated, Node_Type, Node_Links that describe the communication of the nodes, Node_LLM_Models which shows LLM each node uses, and Node_Type each node generated.

The number of user of the sample data is 50, so the number of workflow is 50 also because one user can only use one work flow project. I generated 2~30 nodes per a workflow randomly (777 in total) and node_links(632 in total) reflecting the nature of the node type because some of the nodes cannot be a source node or a target node. LLMs are used that is used in ChainForge (16) and created 0~3 api_keys per a workflow(77 in total). Then I generated Node_LLM_Models for each prompt node and LLM evaluator node (95 in total). Lastly, for the result of the data, 1~5 results has been created per a node(2311).

Overall, what this database should do is to store usage of LLM in ChainForge. This can monitor the usage of each LLMs in ChainForge and save/view data of the workflow information. Further details of the database structure is following in the next.

## 2. Database Plan: A Schematic View




I used PlantUML to generate E-R diagram describing the primary key and foreign key of each table with relationships. In terms of the motivation, I focused on the information of the workflow's information and core parts of the system. Focusing on normalizing the tables, the relationships has been made clearly based on the use in ChainForge.

Nodes and workflows are separated to make sure the scalability of the system by adding different types of each entity. Moreover, I separated users and API key, which can improve security and control data access carefully.

I didn't consider the detailed figures of the node types. Instead, just added the name of the node types used in ChainForge.

Some descriptions about each entity in the next.

## 3. Database Structure: A Normalized View

Some descriptions of tables are below:

| Entity | Attributes | Description |
|---|---|---|
| **Users** | **user_id** (PK), **username**, **email** | Represents system users, each capable of owning one workflow. |
| **Workflows** | **workflow_id** (PK), **user_id** (FK), **workflow_name** | Represents a sequence of operations belonging to a specific user, containing multiple nodes that perform specific tasks. |
| **Nodes** | **node_id** (PK), **workflow_id** (FK), **node_type_id** (FK), **input** | Individual units of work within a workflow, each node performs a specific function based on its type. |
| **Node_Type** | **node_type_id** (PK), **node_type_name** | Defines the type of each node, indicating the specific functionality of the node. |
| **Node_Links** | **source_node_id** (PK, FK), **target_node_id** (PK, FK) | Represents the connections between nodes, defining the flow of data. |
| **Node_LLM_Models** | **node_id** (PK, FK), **model_id** (PK, FK) | Links nodes to the LLM models they utilize. |
| **LLM_Models** | **model_id** (PK), **model_name** | Represents the available LLM models that nodes can reference. |
| **LLM_API** | **api_id** (PK), **model_id** (FK), **workflow_id** (FK), **api_key** | Stores API keys for accessing LLM models, each key is associated with a specific workflow. |
| **Node_Results** | **result_id** (PK), **node_id** (FK), **result_data**, **created_at** | Stores the outcomes of node executions, each result is timestamped upon creation. |

As seen in the previous E-R diagram, this database meets BCNF normal form and 3NF. Firstly, to satisfy 1NF, all columns must be atomic, and there must be no repeating groups within the table. All tables in this database contain atomic values without complex types or nested data. For example, node_id, model_name, workflow_name, etc. are all atomic.

Secondly, to satisfy 2NF, the database should meet 1NF, and all non-primary attributes must be fully functionally dependent on the primary key. This dataset's tables' non-key attributes are fully dependent on the primary key. For example, in the Nodes table, input depends only on the primary key node_id and not depend on any other attributes.

Thirdly, to satisfy 3NF, a database should meet 2NF, and no non-primary key

attribute should be transitively dependent on any other non-primary key attribute. For instance, in Users table, username and email directly depend on the primary key user_id, and doesn't transitively depend on any other non-primary key.

Lastly, to satisfy BCNF, the database should meet 3NF, and every determinant must be a candidate key. For example, in LLM_Models, model_id is the only determinant and is the primary key.

This kind of normalization I applied can ensures efficiency and integrity by reducing redundancy and preventing anomalies in data management.

## 4. Database Views

Overall, the main purpose of creating view is related to the normalization of the tables. If all the related tables are in a table, the efficiency of the storage can decrease. View was useful for this database because it decreases the complexity of the normalized tables.

View 1) Work_Flow_Results

```
CREATE VIEW Work_flow_Results AS
SELECT
    w.workflow_id,
    w.workflow_name,
    n.node_id,
    nr.result_data,
    nr.created_at
FROM
    Workflows w
JOIN
    Nodes n ON w.workflow_id = n.workflow_id
JOIN
    Node_Results nr ON n.node_id = nr.node_id
ORDER BY
    w.workflow_id, n.node_id;
```

Purpose: The first view that I made for this database is a view that shows the results of workflow. This will make it handy to show the result of the workflow to the user. This will provide a quick look-up for results associated with specific workflows and nodes within the workflow.

Targer Users: The users of this view can system administrators and workflow manager who need to monitor and analyze the output and performance of various workflows and nodes. Also, the system can provide results to the user with shorter query.

Rationale for View Over Table: Implementing this view as a view rather than a table provides made the user access to the recent result without additional storage or manual updates.

Output:

| workflow_id | workflow_name | node_id | result_data | created_at |
|---|---|---|---|---|
| 1 | Workflow_user1 | 1 | The result of node_id: 1 result_1. | 2024-05-08 17:14:57 |
| 1 | Workflow_user1 | 1 | The result of node_id: 1 result_2. | 2024-05-20 16:56:44 |
| 1 | Workflow_user1 | 1 | The result of node_id: 1 result_3. | 2024-05-12 07:13:50 |
| 1 | Workflow_user1 | 2 | The result of node_id: 2 result_1. | 2024-06-02 23:29:13 |
| 1 | Workflow_user1 | 2 | The result of node_id: 2 result_2. | 2024-05-14 12:00:11 |
| 1 | Workflow_user1 | 2 | The result of node_id: 2 result_3. | 2024-05-17 05:25:20 |
| 1 | Workflow_user1 | 2 | The result of node_id: 2 result_4. | 2024-05-20 20:50:07 |
| 1 | Workflow_user1 | 2 | The result of node_id: 2 result_5. | 2024-05-23 06:39:19 |
| 1 | Workflow_user1 | 3 | The result of node_id: 3 result_1. | 2024-05-11 20:18:00 |
| 1 | Workflow_user1 | 3 | The result of node_id: 3 result_2. | 2024-05-10 08:53:36 |
| 1 | Workflow_user1 | 3 | The result of node_id: 3 result_3. | 2024-05-09 02:28:04 |
| 1 | Workflow_user1 | 4 | The result of node_id: 4 result_1. | 2024-05-12 00:13:24 |
| 1 | Workflow_user1 | 4 | The result of node_id: 4 result_2. | 2024-06-01 12:33:31 |
| 1 | Workflow_user1 | 4 | The result of node_id: 4 result_3. | 2024-06-02 08:46:24 |
| 1 | Workflow_user1 | 4 | The result of node_id: 4 result_4. | 2024-05-30 09:22:18 |
| 1 | Workflow_user1 | 5 | The result of node_id: 5 result_1. | 2024-05-18 03:49:30 |
| 1 | Workflow_user1 | 5 | The result of node_id: 5 result_2. | 2024-05-17 18:11:24 |
| 1 | Workflow_user1 | 5 | The result of node_id: 5 result_3. | 2024-06-06 14:58:25 |
| 1 | Workflow_user1 | 5 | The result of node_id: 5 result_4. | 2024-05-11 14:45:40 |
| 1 | Workflow_user1 | 5 | The result of node_id: 5 result_5. | 2024-05-22 22:01:24 |
| 1 | Workflow_user1 | 6 | The result of node_id: 6 result_1. | 2024-05-09 23:43:09 |
| 1 | Workflow_user1 | 7 | The result of node_id: 7 result_1. | 2024-06-02 22:52:12 |
| 1 | Workflow_user1 | 7 | The result of node_id: 7 result_2. | 2024-05-23 10:31:11 |
| 1 | Workflow_user1 | 7 | The result of node_id: 7 result_3. | 2024-05-21 21:56:03 |
| 1 | Workflow_user1 | 8 | The result of node_id: 8 result_1. | 2024-05-15 09:20:28 |
| 1 | Workflow_user1 | 9 | The result of node_id: 9 result_1. | 2024-05-28 04:57:17 |
| 1 | Workflow_user1 | 9 | The result of node_id: 9 result_2. | 2024-05-22 02:12:50 |
| 1 | Workflow_user1 | 10 | The result of node_id: 10 result_1. | 2024-05-31 08:08:04 |
| 1 | Workflow_user1 | 10 | The result of node_id: 10 result_2. | 2024-05-25 01:06:42 |
| 1 | Workflow_user1 | 10 | The result of node_id: 10 result_3. | 2024-06-03 12:01:04 |
| 2 | Workflow_user2 | 11 | The result of node_id: 11 result_1. | 2024-05-17 05:43:26 |
| 2 | Workflow_user2 | 11 | The result of node_id: 11 result_2. | 2024-05-31 11:15:45 |
| 2 | Workflow_user2 | 12 | The result of node_id: 12 result_1. | 2024-05-27 11:45:17 |
| 2 | Workflow_user2 | 12 | The result of node_id: 12 result_2. | 2024-05-27 19:12:36 |
| 2 | Workflow_user2 | 13 | The result of node_id: 13 result_1. | 2024-05-16 15:09:29 |

Etc.

View 2) LLM_usage

```
CREATE VIEW LLM_useage AS
SELECT
    m.model_id,
    m.model_name,
    COUNT(nr.result_id) AS usage_count,
    RANK() OVER (ORDER BY COUNT(nr.result_id) DESC) AS LLM_node_rank
FROM
    LLM_Models m
JOIN
    node_llm_models nlm ON m.model_id = nlm.model_id
JOIN
    node_results nr ON nlm.node_id = nr.node_id
GROUP BY
    m.model_id, m.model_name
ORDER BY
    usage_count DESC;
```

Purpose: View 2 to 4 aim to evaluate the usage of the LLM and check which LLM is popular for user also. View 2 helps the user check the ranking of LLM models based on their usage across various nodes, helping identify the most frequently used models.

Target Users: This view is useful for data scientists and analysts supporting LLM or ChainForge company itself looking to optimize resource allocation and for managers needing insights into model popularity and usage patterns.

Rationale for View Over Table: LLM_usage is implemented as a view to facilitate real-time analytics and reporting on dynamically changing data as other views do. As usage data grows and changes, the view updates automatically, providing always-updated rankings without the overhead of updating a physical table.

Ouput:

| model_id | model_name | usage_count | LLM_node_rank |
|---|---|---|---|
| 14 | Bedrock Mistral Mistral | 31 | 1 |
| 12 | Bedrock Amazon Titan | 25 | 2 |
| 4 | Cloude | 24 | 3 |
| 11 | Bedrock AI21 Jurassic 2 | 22 | 4 |
| 13 | Bedrock Cohere Command Text 14 | 22 | 4 |
| 1 | OpenAI GPT3.5 | 19 | 6 |
| 10 | Bedrock Anthropic Claude | 19 | 6 |
| 16 | Bedrock Meta Llmama2 Chat | 17 | 8 |
| 15 | Bedrock Mistral Mixtral | 15 | 9 |
| 8 | Aleph Alpha | 14 | 10 |
| 2 | OpenAI GPT4 | 13 | 11 |
| 3 | OpenAI Dall-E | 12 | 12 |
| 6 | HuggingFace Mistral.B | 12 | 12 |
| 9 | Azure OpenAI | 12 | 12 |
| 7 | HuggingFace Falcon.7B | 8 | 15 |
| 5 | Gemini | 5 | 16 |

View 3) Weekly_LLM_Usage_Trend

```sql
# View 3
CREATE VIEW Weekly_LLM_Usage_Trend AS
SELECT
    t.model_id,
    t.model_name,
    t.week_number,
    t.weekly_usage_count,
    IFNULL(t.previous_usage, 0) AS previous_usage,
    t.weekly_usage_count - IFNULL(t.previous_usage, 0) AS usage_change,
    CASE
        WHEN t.weekly_usage_count > IFNULL(t.previous_usage, 0) THEN 'Increase'
        WHEN t.weekly_usage_count < IFNULL(t.previous_usage, 0) THEN 'Decrease'
        ELSE 'Stable'
    END AS trend,
    CASE
        WHEN t.weekly_usage_count > IFNULL(t.previous_usage, 0) THEN
            CASE
                WHEN (t.weekly_usage_count - IFNULL(t.previous_usage, 0)) > 4 THEN 20
                WHEN (t.weekly_usage_count - IFNULL(t.previous_usage, 0)) > 2 THEN 15
                ELSE 10
            END
        WHEN t.weekly_usage_count < IFNULL(t.previous_usage, 0) THEN
            CASE
                WHEN (IFNULL(t.previous_usage, 0) - t.weekly_usage_count) > 4 THEN -20
                WHEN (IFNULL(t.previous_usage, 0) - t.weekly_usage_count) > 2 THEN -15
                ELSE -10
            END
        ELSE 0
    END AS score
FROM (
    SELECT
        m.model_id,
        m.model_name,
        WEEK(nr.created_at) AS week_number,
        COUNT(nr.node_id) AS weekly_usage_count,
        LAG(COUNT(nr.node_id), 1) OVER (PARTITION BY m.model_id ORDER BY WEEK(nr.created_at)) AS previous_usage
    FROM
        node_results nr
    JOIN
        node_llm_models nlm ON nr.node_id = nlm.node_id
    JOIN
        LLM_Models m ON nlm.model_id = m.model_id
    GROUP BY
        m.model_id, m.model_name, WEEK(nr.created_at)
) AS t;
```

Purpose: This view tracks weekly usage trends of LLM models, showing whether usage is increasing, stable, or decreasing with a quantified score. The larger increase will give more points.

Target Users: The targe user can be strategic planners and operational managers who need to understand trend in model usage over time to make informed decisions about resource planning and model deployment.

Rationale for View Over Table: Maintaining this as a view ensures that the information remains up-to-date with each query, reflecting the latest data about weekly usage. This dynamic capability is crucial for trend analysis.

Ouput:

| model_id | model_name | week_number | weekly_usage_count | previous_usage | usage_change | trend | score |
|---|---|---|---|---|---|---|---|
| 1 | OpenAI GPT3.5 | 18 | 2 | 0 | 2 | Increase | 10 |
| 1 | OpenAI GPT3.5 | 19 | 5 | 2 | 3 | Increase | 15 |
| 1 | OpenAI GPT3.5 | 20 | 2 | 5 | -3 | Decrease | -15 |
| 1 | OpenAI GPT3.5 | 21 | 6 | 2 | 4 | Increase | 15 |
| 1 | OpenAI GPT3.5 | 22 | 4 | 6 | -2 | Decrease | -10 |
| 2 | OpenAI GPT4 | 18 | 5 | 0 | 5 | Increase | 20 |
| 2 | OpenAI GPT4 | 19 | 4 | 5 | -1 | Decrease | -10 |
| 2 | OpenAI GPT4 | 20 | 3 | 4 | -1 | Decrease | -10 |
| 2 | OpenAI GPT4 | 22 | 1 | 3 | -2 | Decrease | -10 |
| 3 | OpenAI Dall-E | 18 | 2 | 0 | 2 | Increase | 10 |
| 3 | OpenAI Dall-E | 19 | 2 | 2 | 0 | Stable | 0 |
| 3 | OpenAI Dall-E | 20 | 1 | 2 | -1 | Decrease | -10 |
| 3 | OpenAI Dall-E | 21 | 2 | 1 | 1 | Increase | 10 |
| 3 | OpenAI Dall-E | 22 | 5 | 2 | 3 | Increase | 15 |
| 4 | Cloude | 18 | 6 | 0 | 6 | Increase | 20 |
| 4 | Cloude | 19 | 2 | 6 | -4 | Decrease | -15 |
| 4 | Cloude | 20 | 5 | 2 | 3 | Increase | 15 |
| 4 | Cloude | 21 | 8 | 5 | 3 | Increase | 15 |
| 4 | Cloude | 22 | 3 | 8 | -5 | Decrease | -20 |
| 5 | Gemini | 19 | 1 | 0 | 1 | Increase | 10 |
| 5 | Gemini | 20 | 2 | 1 | 1 | Increase | 10 |
| 5 | Gemini | 21 | 1 | 2 | -1 | Decrease | -10 |
| 5 | Gemini | 22 | 1 | 1 | 0 | Stable | 0 |
| 6 | HuggingFace ... | 18 | 2 | 0 | 2 | Increase | 10 |
| 6 | HuggingFace ... | 19 | 5 | 2 | 3 | Increase | 15 |
| 6 | HuggingFace ... | 20 | 2 | 5 | -3 | Decrease | -15 |
| 6 | HuggingFace ... | 22 | 3 | 2 | 1 | Increase | 10 |
| 7 | HuggingFace F... | 18 | 2 | 0 | 2 | Increase | 10 |
| 7 | HuggingFace F... | 19 | 2 | 2 | 0 | Stable | 0 |
| 7 | HuggingFace F... | 20 | 1 | 2 | -1 | Decrease | -10 |
| 7 | HuggingFace F... | 22 | 3 | 1 | 2 | Increase | 10 |

Etc.

View 4) LLM_Final_Scores

```
CREATE VIEW LLM_Final_Scores AS
SELECT
    u.model_id,
    u.model_name,
    u.usage_count,
    t.week_number,
    t.weekly_usage_count,
    t.score AS weekly_score,
    u.usage_count + IFNULL(t.score, 0) AS final_score
FROM
    LLM_useage u
JOIN
    Weekly_LLM_Usage_Trend t ON u.model_id = t.model_id
WHERE
    t.week_number = (SELECT MAX(week_number) FROM Weekly_LLM_Usage_Trend WHERE model_id = u.model_id)
ORDER BY
    final_score DESC;
```

Purpose: This view combines usage counts and weekly trend scores to provide a comprehensive final score for each LLM model. This will aid in evaluation and comparison across models.

Target Users: This view serves senior management and decision-makers who require a synthesized view of model performance and trends to guide strategic decisions as the second and third views.

Rationale for View Over Table: Like the other views, LLM_Final_Scores is kept as a view to ensure it reflects the most current data across multiple tables without the need for manual updates, which would be necessary if it were a standalone table.

Output:

| model_id | model_name | usage_count | week_number | weekly_usage_count | weekly_score | final_score |
|---|---|---|---|---|---|---|
| 16 | Bedrock Meta Llmama2 Chat | 17 | 22 | 2 | 10 | 27 |
| 3 | OpenAI Dall-E | 12 | 22 | 5 | 15 | 27 |
| 15 | Bedrock Mistral Mixtral | 15 | 21 | 6 | 10 | 25 |
| 6 | HuggingFace Mistral.B | 12 | 22 | 3 | 10 | 22 |
| 7 | HuggingFace Falcon.7B | 8 | 22 | 3 | 10 | 18 |
| 13 | Bedrock Cohere Command Text 14 | 22 | 22 | 3 | -10 | 12 |
| 9 | Azure OpenAI | 12 | 22 | 2 | 0 | 12 |
| 14 | Bedrock Mistral Mistral | 31 | 22 | 4 | -20 | 11 |
| 12 | Bedrock Amazon Titan | 25 | 22 | 2 | -15 | 10 |
| 1 | OpenAI GPT3.5 | 19 | 22 | 4 | -10 | 9 |
| 5 | Gemini | 5 | 22 | 1 | 0 | 5 |
| 4 | Cloude | 24 | 22 | 3 | -20 | 4 |
| 10 | Bedrock Anthropic Claude | 19 | 22 | 3 | -15 | 4 |
| 8 | Aleph Alpha | 14 | 22 | 3 | -10 | 4 |
| 2 | OpenAI GPT4 | 13 | 22 | 1 | -10 | 3 |
| 11 | Bedrock AI21 Jurassic 2 | 22 | 22 | 1 | -20 | 2 |

## 5. Procedural Elements

Procedural elements like triggers and stored procedures are used in this database design to automate routine tasks, ensure data integrity, and simplify application logic. These elements reduce the need for external intervention and complex SQL commands in application code. This will lead to a more robust, efficient, and error-resistant system. These are particularly important in environments where actions need to be taken automatically in response to changes in the database, such as creating default settings for new users or cleaning up data relations like the ChainForge. By embedding business logic within the database, the system becomes more self-managing and less inconsistencies.

Overall, these procedural elements are integral to ensuring the database operates smoothly and remains consistent with the business rules defined for the system. Therefore, these elements play a role in maintaining the integrity and usability of the database over time in ChainForge database. Basically, I came up with the following triggers by using ChainForge by myself.

Procedure/triger 1) Auto-Create Workflow Trigger After User Creation

```
DELIMITER //
CREATE TRIGGER CreateDefaultWorkflow AFTER INSERT ON Users
FOR EACH ROW
BEGIN
    INSERT INTO Workflows (user_id, workflow_name)
    VALUES (NEW.user_id, CONCAT(NEW.username, '\'s basic workflow'));
END; //
DELIMITER ;
```

Purpose: Tjos troger automatically creates a default workflow for each new user added to the Users table.

Motivation: This trigger immediately provides a starting workflow, which is especially useful in systems where a workflow is necessary to begin any task. Automating this process avoids the need for users or administrators to manually create an initial workflow, streamlining user onboarding.

Procedure/trigger 2) Auto-Delete Node Links Trigger Upon Node Deletion

```
DELIMITER //
CREATE TRIGGER DeleteNodeLinks BEFORE DELETE ON Nodes
FOR EACH ROW
BEGIN
    DELETE FROM Node_Links
    WHERE source_node_id = OLD.node_id OR target_node_id = OLD.node_id;
END; //
DELIMITER ;
```

Purpose: This trigger ensures that when a node is deleted, any links associated with that node in the Node_Links table are also removed.

Motivation: Maintains referential integrity by removing orphan links that could lead to errors or inconsistencies when data is queried. This trigger prevents errors related to missing nodes which ensures that all links in the workflow are valid. This trigger is crucial for maintaining the accuracy and reliability of workflow executions.

Procedure/triger 3) Record Result Addition Time Procedure

```
DELIMITER //
CREATE PROCEDURE RecordResult(IN node_id INT, IN result_data TEXT)
BEGIN
    INSERT INTO Node_Results (node_id, result_data, created_at)
    VALUES (node_id, result_data, CURRENT_TIMESTAMP);
END; //
DELIMITER ;
```

Purpose: This procedure inserts a result entry into the Node_Results table.

Motivation: Provides a standardized way to record node results with timestamps, for tracking the execution history and outcomes of nodes. This procedure encapsulates the insertion logic, making the application code cleaner and more maintainable by reducing redundancy and centralizing the data entry logic.

Procedure/triger 4) Update LLM Model Usage Aggregate Procedure

```
#Procedure/triger 4
DELIMITER //
CREATE TRIGGER DeleteOldAPIKey BEFORE INSERT ON LLM_API
FOR EACH ROW
BEGIN
    DELETE FROM LLM_API
    WHERE model_id = NEW.model_id AND workflow_id = NEW.workflow_id;
END; //
DELIMITER ;
```

Purpose:

This trigger automatically deletes old API keys in the LLM_API table when a new API key for the same model_id and workflow_id is inserted. This ensures that each combination of model and workflow retains only the most recent API key.

Motivation:

It prevents the accumulation of outdated or possibly conflicting API keys, which can lead to errors or security issues if old keys are inadvertently used. By ensuring that only the latest API key is available, the system enhances security measures and simplifies management without any need to manually clean up old keys. Additionally, this trigger ensures that any process or system interacting with the models through these API keys will always have the most current access credentials, enhancing efficiency and reliability in operations.

## 6. Example Queries: Your Database In Action

Query 1)

```
SELECT w.workflow_id, w.workflow_name, COUNT(n.node_id) AS node_count
FROM Workflows w
JOIN Nodes n ON w.workflow_id = n.workflow_id
GROUP BY w.workflow_id, w.workflow_name;
```

Outupt:

| workflow_id | workflow_name | node_count |
|---|---|---|
| 1 | Workflow_user1 | 10 |
| 2 | Workflow_user2 | 3 |
| 3 | Workflow_user3 | 18 |
| 4 | Workflow_user4 | 18 |
| 5 | Workflow_user5 | 8 |
| 6 | Workflow_user6 | 23 |
| 7 | Workflow_user7 | 7 |
| 8 | Workflow_user8 | 15 |
| 9 | Workflow_user9 | 21 |
| 10 | Workflow_user10 | 10 |

This query is to retrieve and display the count of nodes associated with each workflow in the database. It lists each workflow by its ID and name, along with the number of nodes that belong to it. The query performs a join between the Workflows and Nodes tables on the workflow_id to link each workflow with its corresponding nodes. By grouping the results by workflow_id and workflow_name, it ensures that the output is organized by individual workflows, providing a clear overview of how many nodes each workflow contains.

Query 2)

```
# Query 2
SELECT
    nl.source_node_id,
    'Source Node' AS node_role,
    n1.node_type_id AS source_node_type_id,
    nt1.node_type_name AS source_node_type_name,
    nl.target_node_id,
    'Target Node' AS node_role,
    n2.node_type_id AS target_node_type_id,
    nt2.node_type_name AS target_node_type_name
FROM
    Node_Links nl
JOIN
    Nodes n1 ON nl.source_node_id = n1.node_id
JOIN
    Node_Type nt1 ON n1.node_type_id = nt1.node_type_id
JOIN
    Nodes n2 ON nl.target_node_id = n2.node_id
JOIN
    Node_Type nt2 ON n2.node_type_id = nt2.node_type_id
WHERE
    nl.source_node_id = 1
ORDER BY
    nl.target_node_id;
```

Output:

| source_node_id | node_role | source_node_type_id | source_node_type_name | target_node_id | node_role | target_node_type_id | target_node_type_name |
|---|---|---|---|---|---|---|---|
| 2 | Source Node | 8 | Python Evaluator | 7 | Target Node | 12 | Inspect Node |

This query provides detailed information about all nodes that are directly connected to a specific source node, identified by node_id '2'. It fetches and displays attributes for both the source and the target nodes linked by the Node_Links table. The attributes include the node ID, node type ID, and node type name for both source and target nodes.

Query 3)

```
SELECT
    u.user_id,
    u.username,
    w.workflow_id,
    w.workflow_name,
    l.api_id,
    l.model_id,
    m.model_name,
    l.api_key
FROM
    Users u
JOIN
    Workflows w ON u.user_id = w.user_id
JOIN
    LLM_API l ON w.workflow_id = l.workflow_id
JOIN
    LLM_Models m ON l.model_id = m.model_id
WHERE
    u.user_id = 1
ORDER BY
    l.api_id;
```

Output:

| user_id | username | workflow_id | workflow_name | api_id | model_id | model_name | api_key |
|---|---|---|---|---|---|---|---|
| 1 | user1 | 1 | Workflow_user1 | 1 | 4 | Cloude | jiZDxkgPMhZUHiU7kdowiHlP5qr0S6DQ |

This query is to provide a comprehensive overview of the connections between nodes in a system. It specifically retrieves details about both source and target nodes connected via specified node links. The output includes node IDs, their roles as either source or target, and the types of these nodes, allowing users to understand the relationships and data flow between different components of a workflow.

Query 4) Triger 1 test

```
SELECT workflow_name, node_id, result_data, created_at
FROM Work_flow_Results
WHERE workflow_id = 1;
```

Output:

| workflow_name | node_id | result_data | created_at |
|---|---|---|---|
| Workflow_user1 | 1 | The result of node_id: 1 result_1. | 2024-05-08 17:14:57 |
| Workflow_user1 | 1 | The result of node_id: 1 result_2. | 2024-05-20 16:56:44 |
| Workflow_user1 | 1 | The result of node_id: 1 result_3. | 2024-05-12 07:13:50 |
| Workflow_user1 | 2 | The result of node_id: 2 result_1. | 2024-06-02 23:29:13 |
| Workflow_user1 | 2 | The result of node_id: 2 result_2. | 2024-05-14 12:00:11 |
| Workflow_user1 | 2 | The result of node_id: 2 result_3. | 2024-05-17 05:25:20 |
| Workflow_user1 | 2 | The result of node_id: 2 result_4. | 2024-05-20 20:50:07 |
| Workflow_user1 | 2 | The result of node_id: 2 result_5. | 2024-05-23 06:39:19 |
| Workflow_user1 | 3 | The result of node_id: 3 result_1. | 2024-05-11 20:18:00 |
| Workflow_user1 | 3 | The result of node_id: 3 result_2. | 2024-05-10 08:53:36 |
| Workflow_user1 | 3 | The result of node_id: 3 result_3. | 2024-05-09 02:28:04 |
| Workflow_user1 | 4 | The result of node_id: 4 result_1. | 2024-05-12 00:13:24 |
| Workflow_user1 | 4 | The result of node_id: 4 result_2. | 2024-06-01 12:33:31 |
| Workflow_user1 | 4 | The result of node_id: 4 result_3. | 2024-06-02 08:46:24 |
| Workflow_user1 | 4 | The result of node_id: 4 result_4. | 2024-05-30 09:22:18 |
| Workflow_user1 | 5 | The result of node_id: 5 result_1. | 2024-05-18 03:49:30 |
| Workflow_user1 | 5 | The result of node_id: 5 result_2. | 2024-05-17 18:11:24 |
| Workflow_user1 | 5 | The result of node_id: 5 result_3. | 2024-06-06 14:58:25 |
| Workflow_user1 | 5 | The result of node_id: 5 result_4. | 2024-05-11 14:45:40 |
| Workflow_user1 | 5 | The result of node_id: 5 result_5. | 2024-05-22 22:01:24 |
| Workflow_user1 | 6 | The result of node_id: 6 result_1. | 2024-05-09 23:43:09 |
| Workflow_user1 | 7 | The result of node_id: 7 result_1. | 2024-06-02 22:52:12 |
| Workflow_user1 | 7 | The result of node_id: 7 result_2. | 2024-05-23 10:31:11 |
| Workflow_user1 | 7 | The result of node_id: 7 result_3. | 2024-05-21 21:56:03 |
| Workflow_user1 | 8 | The result of node_id: 8 result_1. | 2024-05-15 09:20:28 |
| Workflow_user1 | 9 | The result of node_id: 9 result_1. | 2024-05-28 04:57:17 |
| Workflow_user1 | 9 | The result of node_id: 9 result_2. | 2024-05-22 02:12:50 |
| Workflow_user1 | 10 | The result of node_id: 10 result_1. | 2024-05-31 08:08:04 |
| Workflow_user1 | 10 | The result of node_id: 10 result_2. | 2024-05-25 01:06:42 |
| Workflow_user1 | 10 | The result of node_id: 10 result_3. | 2024-06-03 12:01:04 |

This query utilizing the first view is particularly useful for workflow managers or system users who need to audit or review the outcomes and execution history of a particular workflow. It provides a comprehensive look at the results generated by each node within the workflow.

Query 5) Extra queries to test triggers

```sql
-- Insert a new user
INSERT INTO Users (username, email) VALUES ('newuser', 'newuser@example.com');
-- Query to check if the workflow was automatically created
SELECT *
FROM Workflows
WHERE user_id = (SELECT user_id FROM Users WHERE username = 'newuser');

-- Assume a node with ID 10 exists; delete this node
DELETE FROM Nodes WHERE node_id = 10;
-- Check if the links have been removed
SELECT *
FROM Node_Links
WHERE source_node_id = 10 OR target_node_id = 10;

-- Call the procedure with sample data
CALL RecordResult(1, 'Sample result data for node 1');
-- Check if the result was added
SELECT *
FROM Node_Results
WHERE node_id = 1;

-- Insert a new API key for a given model and workflow
INSERT INTO LLM_API (model_id, workflow_id, api_key) VALUES (1, 1, 'NEWAPIKEY123');
-- Check if old API keys have been removed
SELECT *
FROM LLM_API
WHERE model id = 1 AND workflow id = 1;
```

Test queries for testing triggers were all working well.

## 7. Conclusions

This project can be a good start of the full database of the ChainForge system. By integrating with the application level of the system, more procedures and triggers can be added.

As I didn't add tables for each specific node, each type of node tables can be added with specific configuration and structures, and other aspects of ChainForge system. Furthermore, this database needs to be optimized for the larger data and complicate queries consistently. For example, the database structure can be distributed increasing the amount of the data or can introduce the big data techs.

The mechanism of the link of nodes can be improved. For instance, procedure that is passing the result of the source node once it's creating to the target node as an input can be added.

For the further design, I would like to try 4NF and 5NF to test the performance of the database with the larger dataset.

Overall, even though I increased the consistency and integrity by adding some procedures, security measures should be added more with encryption techniques and access control mechanisms. By doing this project, I felt the importance of design process because I had to change the structure over trying the queries because I realized some problems.

## Acknowledgements

Purcell, J. [freeCodeCamp.org]. (2019, June 3). Data Normalization Explained in 5 Minutes [Video]. YouTube. https://www.youtube.com/watch?v=RYIiOG4LsvQ

## References

1. Nambiar, K.K., Gopinath, B., Nagaraj, W. and Manjunath, S., 1997. Boyce-Codd Normal Form Decomposition. Computers Math. Applic., [online] 33(4), pp.1-3. Available at: https://www.sciencedirect.com/science/article/pii/S0898122197000011 [29.04.24].