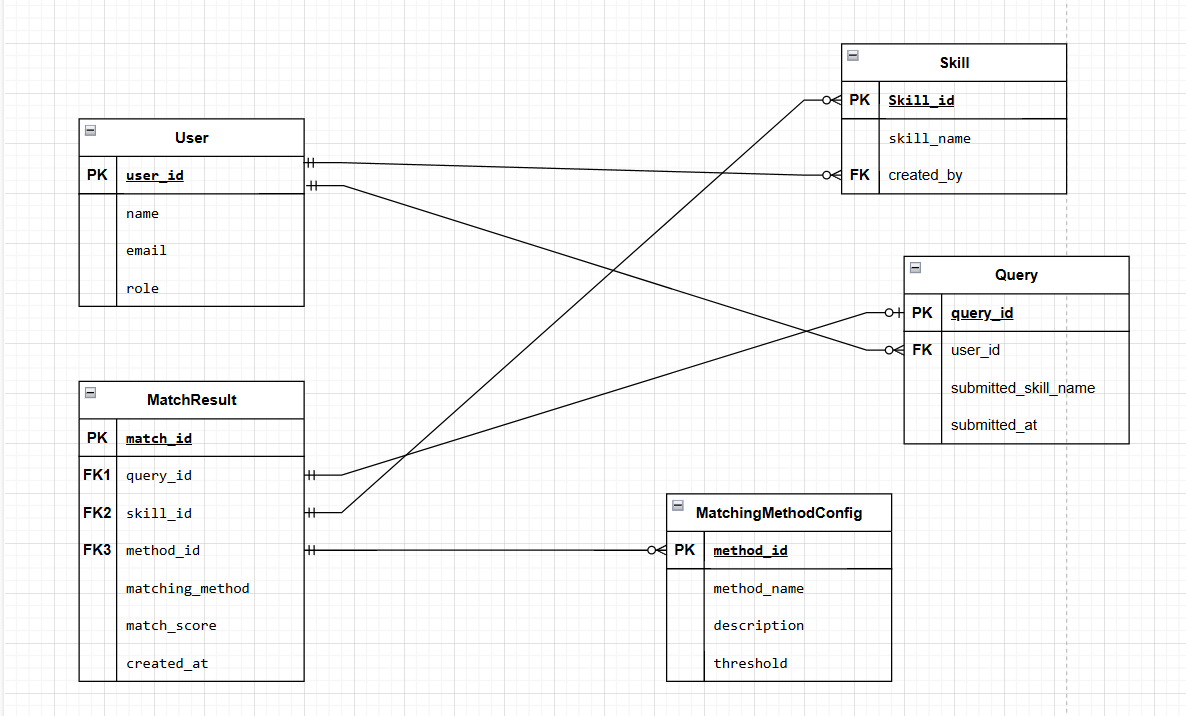
**Teamo AI - ER Diagram Documentation**

**ER Diagram**

****

**1. Introduction**

The **Teamo AI Entity-Relationship (ER) Diagram** represents the database structure for a skill-matching system, where users submit queries to find relevant skills from an administrator-curated skill list. The system applies multiple matching methods to compare the user’s submitted skill with predefined skills, generating match results with similarity scores. The ER diagram illustrates the key entities, relationships, and constraints required to manage users, queries, skills, match results, and matching methods efficiently.

The database is designed to ensure data integrity, scalability, and performance while supporting administrators who manage the skill list and users who submit queries. Each query can produce multiple match results, linking user-submitted skills with predefined skills based on different matching algorithms. The MatchResult entity acts as a bridge between queries, skills, and matching methods, ensuring accurate tracking of matches and algorithm performance.

**2. Challenges:**

* Users submit skills with **inconsistent naming** (e.g., "ML" vs. "Machine Learning").
* Matching methods need to support **fuzzy matching, NLP, and similarity scoring**.
* The system should differentiate between **users and administrators**.
* Query results should be stored for **analytics and improvement.**

**3. Proposed Solution**

The **Teamo AI ER Diagram** provides a well-structured relational model that ensures:

* **Efficient skill storage** by allowing only administrators to manage the skill list.
* **Accurate skill matching** through multiple similarity methods.
* **User-friendly query handling** to improve search efficiency.
* **Scalability and performance optimization** by structuring relationships properly.

**4. ER Diagram Overview**

**The ER diagram consists of five main entities:**

* **User:** Stores all user details and differentiates between regular users and administrators**.**
* **Skill:** Maintains a list of administrator-defined skills**.**
* **Query:** Tracks user-submitted skill searches**.**
* **MatchResult:** Stores skill matches, linking queries to skills with similarity scores**.**
* **MatchingMethodConfig:** Stores details of different matching algorithms used**.**

Each entity has been carefully selected to optimize **data retrieval, prevent redundancy, and allow future enhancements.**

**5. Entities and Their Attributes**

* **User Entity 🧑‍💻 (Stores all users, including admins)**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| user\_id (PK) | INT | **Primary Key** (Unique ID for each user). |
| name | VARCHAR | Stores the full name of the user. |
| email | VARCHAR UNIQUE | Stores the user's email (Ensures uniqueness and allows authentication) |
| role | VARCHAR | Differentiates between regular users and administrators. |

* **Skill Entity 🎓 (Stores predefined skills managed by administrators)**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| skill\_id (PK) | INT | **Primary Key** (Unique ID for each skill |
| skill\_name | VARCHAR UNIQUE | Stores the name of the skill. |
| created\_by (FK) | INT | **Foreign Key** linking the skill to the admin who created it. |

* **Query Entity** 🔍 (**Stores user-submitted skill searches**)

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| query\_id (PK) | INT | **Primary Key** (Unique ID for each query). |
| user\_id (FK) | INT | **Foreign Key** linking the query to the user who submitted it |
| submitted\_skill\_name | VARCHAR | the skill name that the user searched for. |
| submitted\_at | TIMESTAMP | Timestamp of when the query was submitted. |

* **MatchResult Entity 🏆** (**Stores matching results for each query**)

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| match\_id (PK) | INT | **Primary Key** (Unique ID for each match). |
| query\_id (FK) | INT | **Foreign Key** linking to the query that generated this match. |
| skill\_id (FK) | INT | **Foreign Key** linking to the skill that was matched. |
| method\_id (FK) | INT | **Foreign Key** linking to the matching method used. |
| matching\_method | VARCHAR | The algorithm/method used for matching. |
| match\_score | FLOAT | The similarity score between the query and skill. |
| created\_at | TIMESTAMP | Timestamp of when the match result was generated. |

* **MatchingMethodConfig Entity (Stores details of matching methods)**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| method\_id(PK) | INT | **Primary Key** (Unique ID for each matching method). |
| method\_name | VARCHAR UNIQUE | Stores the name of the matching algorithm (e.g., Levenshtein, TF-IDF). |
| description | VARCHAR | Describes how the matching method works. |
| threshold | FLOAT | Defines the minimum similarity score required for a match. |

**6. Relationships and Cardinality**

|  |  |  |
| --- | --- | --- |
| **Relationship** | **Cardinality** | **Total Participation (**Mandatory or Optional) |
| **User → Query** | 1:M | **User:** Optional (0..M), **Query:** Mandatory (1) |
| **User → Skill (Admins Only)** | 1:M | **User:** Optional (0..M), **Skill:** Mandatory (1) |
| **Query → MatchResult** | 1:M | **Query:** Optional (0..M), **MatchResult:** Mandatory (1) |
| **MatchResult → Skill** | M:1 | **MatchResult:** Mandatory (1), **Skill:** Optional (0..M) |
| **MatchResult → MatchingMethodConfig** | M:1 | **MatchResult:** Mandatory (1), **MatchingMethodConfig:** Optional (0..M) |

**User → Query (One-to-Many 1:M)**

**Explanation**

* **Cardinality: 1:M →** A single admin can create multiple queries, but each query belongs to only one user.
* **Total Participation:**
  + User side: Partial participation (Not all users must submit queries).
  + Query side: Total participation (Every query must belong to a user).

**User → Skill (One-to-Many 1:M: Admins only)**

**Explanation**

* **Cardinality: 1:M →** A single admin can create multiple skills, but each skill is created by only one admin
* **Total Participation:**
  + User side: Partial participation (Not all users create skills; only admins do)
  + Skill side: Total participation (Every skill must be created by an admin).

**Query → Match result (One-to-Many 1:M)**

**Explanation**

* **Cardinality: 1:M →** A single query can generate multiple match results, but each match result belongs to only one query.
* **Total Participation:**
  + Query side: Partial participation (Some queries may not return any match results).
  + Match result side: Total participation (Every match result must be linked to a query).

**MatchResult → Skill (Many-to-One M:1)**

**Explanation**

* **Cardinality: M:1 →** A single skill can be matched by multiple match results, but each match result references only one skill**.**
* **Total Participation:**
  + MatchResult side: Total participation (Every match result must reference a valid skill).
  + Skill side: Partial participation (Not all skills may appear in a match result).

**MatchResult → MatchingMethodConfig (Many-to-One M:1)**

**Explanation**

* **Cardinality: M:1 →** A single matching method can be used for multiple match results, but each match result is generated using one specific method**.**
* **Total Participation:**
  + MatchResult side: Total participation (Every match result must use a matching method).
  + MatchingMethodConfig side: Partial participation (Not all methods must be used in match results).

**7. Challenges in Designing the ER Diagram**

Designing the Teamo AI ER Diagram involved several complexities:

* **Ensuring proper relationship cardinality**: Defining One-to-Many (1:M) and Many-to-One (M:1) relationships correctly was crucial to maintaining data integrity.
* **Handling weak entities**: The MatchResult entity had to be carefully structured since it depends on multiple foreign keys (query\_id, skill\_id, method\_id).
* **Avoiding redundancy**: Ensuring that queries, skills, and match results were properly normalized to avoid duplication.
* **Flexible MatchingMethodConfig**: Creating a scalable design that allows new matching algorithms to be added dynamically without altering the core schema.

**8.Strong Entities vs Weak Entities in the ER-Diagram**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | **Entity** |  |  | | --- | |  | | **Type** | **Reason** |
| **User** | Strong | Exists independently with user\_id as PK. |
| **Skill** | Strong | Has a unique skill\_id and does not require another entity. |
| **Query** | Strong | Defined by query\_id, independent of other entities. |
| **MatchResult** | Weak | Depends on Query, Skill, and MatchingMethodConfig. |
| **MatchingMethodConfig** | Strong | Exists independently with method\_id as PK. |

Why is MatchResult a Weak Entity?

* **MatchResult** depends on three other entities for its existence:
* **Query** → The user’s search request that initiated the match.
* **Skill** → The skill from the predefined administrator list that was matched.
* **MatchingMethodConfig** → The algorithm used to compute the match.
* Without valid entries in these three tables, **MatchResult** cannot exist, making it a dependent weak entity.
* **Primary Key Selection**→ The **MatchResult** table uses a **surrogate key (match\_id)** instead of a **composite primary key** (query\_id, skill\_id, method\_id) for **better indexing and scalability**.
* AI Assistance: ChatGPT provided guidance on defining the primary key of **MatchResult** as a surrogate key (**match\_id**) instead of a composite primary key (**query\_id, skill\_id, method\_id**), ensuring better indexing, scalability, and simpler referencing.

**10. Conclusion**

The Teamo AI ER Diagram models the skill-matching system by defining essential entities, relationships, and constraints that uphold data consistency, scalability, and accuracy. The modular structure supports future enhancements, such as integrating new matching algorithms or expanding user roles, ensuring the system remains adaptable to evolving needs. This ER model serves as a solid foundation for AI-powered skill matching, optimizing data retrieval, minimizing redundancy, and improving search precision.

**Creative Part**

**Scenario - Using Teamo to Boost Efficiency**

**What AI tools have I used to assist in solving this problem, and what types of problems have they helped me address?**

I have been advised by Teamlift to utilize various tools such as Draw.io, DbVisualizer, DGM, SmartDraw, and Microsoft Visio for creating an ER diagram based on the activities I defined while using the application. After exploring these options, I chose to use Draw.io as I found it the most intuitive and comfortable, though I have also tested the other tools.

**How AI Assisted in Designing the ER Diagram**

In designing the Teamo AI ER Diagram, ChatGPT played a role as an AI tool in structuring the database, defining key entities, and establishing relationships. The assistance was particularly valuable in identifying the need for the MatchingMethodConfig entity, defining its attributes, and explaining its importance in the system as well as the provided guidance on defining the primary key of **MatchResult.**

* **Justifying the Need for MatchingMethodConfig as an Entity**

Initially, the concept of storing matching methods was unclear. ChatGPT provided a structured analysis that highlighted the need for a dedicated entity to store different matching algorithms. The reasoning was based on the following key points:

* The system needs to support multiple matching techniques (e.g., fuzzy matching, NLP-based similarity, and TF-IDF cosine similarity).
* Instead of hardcoding matching logic, storing these methods in a database allows for flexibility and future expansion.
* It enables tracking of which matching method was used for each result, helping to analyze and improve algorithm performance.
* **Defining Attributes for MatchingMethodConfig**

Once the need for MatchingMethodConfig was established, ChatGPT helped define its key attributes to ensure it served its intended purpose effectively:

This structure ensures that new matching techniques can be added dynamically without modifying the database schema.

* **Establishing Relationships from MatchingMethodConfig**

ChatGPT provided detailed reasoning for the relationship between MatchingMethodConfig and MatchResult:

* A single matching method can be used in multiple match results (M:1 relationship).
* Every match result must have a reference to a matching method to track how it was generated.
* Not all methods may be used immediately, so partial participation was applied to MatchingMethodConfig.

**Dilema** – whether to keep **matching\_method** in Match Result. ChatGPT provided instruction whether I should keep it or not.

|  |  |
| --- | --- |
| **Scenario** | **Do We Need matching\_method?** |
| Faster Queries Without Joins | Yes |
| Redundant Data Storage (Normalization Issue) | No |
| Easier Analytics & Readability | Yes |
| If MatchingMethodConfig is Small (<10 rows) | Yes |
| If MatchingMethodConfig Can Change Often | No |

**Did AI help me in writing the code**?

I was advised by Teamlift to explore various learning tools such as **SmartScripter, CodeWhisperer, GPT Engineer, and Coursera**. However, I chose **GeeksforGeeks** as my primary learning platform, as its detailed explanations helped me better understand and implement key concepts such as Cosine Similarity, Levenshtein Distance (Fuzzy Matching), TF-IDF + Cosine Similarity, and Weighted Hybrid Scoring. While I was already familiar with these techniques from my academic assignments, **GeeksforGeeks** provided a more structured and comprehensive approach that was particularly beneficial for this problem.

Additionally, I utilized **ChatGPT** to optimize my script by implementing efficient NLP processing with **spaCy**, specifically precomputing embeddings and storing them instead of repeatedly computing skill vectors. Although this optimization may not have been strictly necessary given that we are not implementing a database, it provided valuable insights into improving efficiency, which could be beneficial when working with large datasets in the future.