**TraceIt**

**Database Design Document**

**V 2.0**

**By**

|  |  |  |
| --- | --- | --- |
| 1 | Usman Ghani | Roll No 24 |
| 2 | Muhammad Shariq | Roll No 39 |
| 3 | Rayan Badar | Roll No 18 |



**Department of Computer Sciences**

**Namal University Mianwali, Pakistan**

**Submission Date: 2nd June, 2025**

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Approved by** |
| 18/04/25 | V 2.0 | This version includes updates based on the approved ERD creation Milestone submitted earlier, covering complete Relational Schema design and all other deliverables. | **Miss Asiya Batool** |
| 31/05/25 | V 1.0 | This version includes updates based on the approved project proposal submitted earlier, covering complete ERD design, data dictionary, relationships, and in-text citations. | **Miss Asiya Batool** |

**TABLE OF CONTENTS**

[**CHAPTER 1: PROJECT OVERVIEW 5**](#_Toc199772769)

[**1.1. INTRODUCTION: 5**](#_Toc199772770)

[**1.2. PROBLEM STATEMENT: 5**](#_Toc199772771)

[**1.3. PROJECT OBJECTIVES: 5**](#_Toc199772772)

[**1.4. DOCUMENT OBJECTIVES: 5**](#_Toc199772773)

[**CHAPTER 2: DETAILED DATABASE DESIGN 6**](#_Toc199772774)

[**2.1 ENTITY [3]: 6**](#_Toc199772776)

[**2.2 DATA DICTIONARY [3]: 6**](#_Toc199772777)

[**2.2.1 USER 6**](#_Toc199772778)

[**2.2.2 ADMIN 7**](#_Toc199772779)

[**2.2.3 ITEM 7**](#_Toc199772780)

[**2.2.4 REPORT 8**](#_Toc199772781)

[**2.2.5 VERIFICATION 8**](#_Toc199772782)

[**2.2.6 MATCHING\_LOG 8**](#_Toc199772783)

[**2.2.7 NOTIFICATION 9**](#_Toc199772784)

[**2.2.8 FEEDBACK 9**](#_Toc199772785)

[**2.2.9 CLAIM 9**](#_Toc199772786)

[**2.2.10 ITEM\_RETURN\_LOG 10**](#_Toc199772787)

[**2.3 RELATIONSHIPS 10**](#_Toc199772788)

[**2.4 ENTITY RELATIONSHIP DIAGRAM [3]: 11**](#_Toc199772789)

[**CHAPTER 3: LOGICAL DATABASE DESIGN 12**](#_Toc199772790)

[**3.1 RELATIONAL SCHEMA: 12**](#_Toc199772791)

[**3.2 FUNCTIONAL DEPENDENCIES: 13**](#_Toc199772792)

[**3.3 NORMALIZATION: 14**](#_Toc199772793)

[**3.3.1 STEP-BY-STEP NORMALIZATION PROCESS 15**](#_Toc199772794)

[**3.3.2 FINAL 3NF RELATIONS 15**](#_Toc199772795)

[**CHAPTER 4: PHYSICAL DATABASE DESIGN 16**](#_Toc199772796)

[**4.1 STRUCTURE OF THE TABLES: 16**](#_Toc199772797)

[**4.1.1 ADMIN 16**](#_Toc199772798)

[**4.1.2 CLAIM 16**](#_Toc199772799)

[**4.1.3 FEEDBACK 16**](#_Toc199772800)

[**4.1.4 ITEM 17**](#_Toc199772801)

[**4.1.5 ITEM\_RETURN\_LOG 17**](#_Toc199772802)

[**4.1.6 MATCHING\_LOG 18**](#_Toc199772803)

[**4.1.7 USER 18**](#_Toc199772804)

[**4.1.8 NOTIFICATION 18**](#_Toc199772805)

[**4.1.9 VERIFICATION 19**](#_Toc199772806)

[**4.1.10 REPORT 19**](#_Toc199772807)

[**4.2 DATA SAMPLES INSIDE TABLES: 19**](#_Toc199772808)

[**4.2.1 USER 20**](#_Toc199772809)

[**4.2.2 ITEM 20**](#_Toc199772810)

[**4.2.3 ADMIN 20**](#_Toc199772811)

[**4.2.4 ITEM\_RETURN\_LOG 21**](#_Toc199772812)

[**4.2.5 MATCHING\_LOG 21**](#_Toc199772813)

[**4.2.6 REPORT 22**](#_Toc199772814)

[**4.2.7 VERIFICATION 22**](#_Toc199772815)

[**4.2.8 NOTIFICATION 23**](#_Toc199772816)

[**4.2.9 CLAIM 23**](#_Toc199772817)

[**4.2.10 FEEDBACK 24**](#_Toc199772818)

[**4.3 QUERIES RESULTS: 24**](#_Toc199772819)

[**CHAPTER 5: REFERENCES 30**](#_Toc199772820)

[**5.1 APPENDIX: AI PROMPTS 30**](#_Toc199772821)

# PROJECT OVERVIEW

## INTRODUCTION:

The **TraceIt** Lost & Found Management System is a mobile application backed by a cloud database, designed to digitize the lost-and-found process within institutions like universities or offices. It enables users to report lost or found items, automates match suggestions using algorithm-based logic, and allows administrators to verify and approve item claims. The system promotes transparency, improves recovery rates, and streamlines the item tracking process by eliminating traditional, inefficient, paper-based methods [1] [2].

## PROBLEM STATEMENT:

Lost and found items are frequently mismanaged [1] due to traditional manual logging systems that are inefficient, error-prone, and non-centralized. These systems lead to unclaimed items, delayed recoveries, and miscommunication. The lack of real-time notifications, proper verification, and data-driven tracking impairs the overall efficiency. The desired future state is a cloud-based mobile system that allows users to report, search, and recover items, while enabling administrators to verify and manage the overall process securely and efficiently [2].

## PROJECT OBJECTIVES:

The primary objective of **TraceIt** is to digitize and optimize the lost-and-found process by leveraging modern database management, automation, and secure authentication methods.

The key objectives include:

1. **Providing a centralized cloud-based database** for storing and managing lost and found item records.
2. **Enabling users to report lost items** and search for found objects through a user-friendly mobile application.
3. **Providing an automated matching mechanism** that suggests potential item matches based on entered details.
4. **Providing admins with verification tools** to validate reports and ensure authenticity.
5. **Integrating real-time notifications** to alert users about potential item matches.

## DOCUMENT OBJECTIVES:

This document aims to:

1. **Define** the core database entities and their relationships.
2. **Provide** a detailed data dictionary for each table.
3. **Describe** the logical structure of the ERD based on system functionalities.
4. **Act as a** guide for database implementation and future modifications.
5. **Ensure** clarity, consistency, and maintainability of the data model.

# DETAILED DATABASE DESIGN



## ENTITY [3]:

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Entity Name** | **Description** |
| 01 | USER | Represents individuals who use the app to report or recover items. |
| 02 | ADMIN | Authorized personnel responsible for verifying item matches and reports. |
| 03 | ITEM | Represents a physical object reported as lost or found. |
| 04 | REPORT | Contains lost or found reports submitted by users or admins. |
| 05 | VERIFICATION | Records the admin’s decision for verifying item claims. |
| 06 | MATCHING\_LOG | Logs system-suggested item matches with a score and review status. |
| 07 | NOTIFICATION | Stores alerts sent to users/admins regarding match suggestions or actions. |
| 08 | FEEDBACK | Captures user feedback for item experience or system usage. |
| 09 | CLAIM | Shows the process of Claim by an user. |
| 10 | ITEM\_RETURN\_LOG | Stores the data of returning the item to user. |

## DATA DICTIONARY [3]:

### **USER**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | UserID | Integer | Primary Key | Unique identifier for user. |
| 02 | Name | String | Not Null | Full name of the user. |
| 03 | Email | String | Unique | Email address. |
| 04 | PasswordHash | String | Not Null | Hashed password for security. |
| 05 | ContactInfo | String | Optional | Phone number or other contact. |

### **ADMIN**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | AdminID | Integer | Primary Key | Unique identifier for the admin. |
| 02 | Name | String | Not Null | Full name of the admin. |
| 03 | Email | String | Unique, Not Null | Admin’s email for login/notifications. |
| 04 | PasswordHash | String | Not Null | Encrypted password for authentication. |
| 05 | ContactInfo | String | Optional | Contact number of the admin. |

### **ITEM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Attribute** | **Data Type** | **Constraint** | **Description** |
| 01 | ItemID | Integer | Primary Key | Unique identifier. |
| 02 | Category | String | Not Null | Type/category of item. |
| 03 | Description | String | Optional | Detailed explanation of the item. |
| 04 | Color | String | Optional | Primary color of the item. |
| 05 | Brand | String | Optional | Brand/manufacturer of the item. |
| 06 | Size | String | Optional | Item size or dimensions. |
| 07 | ItemCondition | String | Optional | Physical condition of the item. |
| 08 | Material | String | Optional | Material composition (e.g., leather). |
| 09 | TagNumber | String | Not Null | Unique label, barcode, or tag (optional). |
| 10 | Location | String | Not Null | Where the item was reported. |
| 11 | DateReported | DateTime | Not Null | Timestamp of report. |
| 12 | Status | String | Not Null | 'Returning', 'matched', ‘Resolved’ etc. |
| 13 | ImageURL | String | Optional | Link to item image (optional). |
| 14 | ItemType | String | Not Null | Lost or found. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | ReportID | Integer | Primary Key | Unique ID for each report. |
| 02 | ItemID | Integer | --- | References the reported item. |
| 03 | AdminID | Integer | --- | ID of Admin who made the report. |
| 04 | UserID | Integer | --- | ID of user who made report. |
| 05 | ReportedAt | DateTime | Not Null | Timestamp of when the report was filed. |

### **REPORT**

### **VERIFICATION**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | VerificationID | Integer | Primary Key | Unique ID for each verification request. |
| 02 | ClaimID | Integer | --- | References the claim item under verification. |
| 03 | AdminID | Integer | --- | Admin responsible for verifying the item. |
| 04 | VerifiedAt | DateTime | Optional | Timestamp of verification action. |
| 05 | Status | String | Not Null | ‘Pending’, ‘Approved’, or ‘Rejected’. |

### **MATCHING\_LOG**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | MatchID | Integer | Primary Key | Unique ID for each AI-based item match. |
| 02 | LostItemID | Integer | --- | Refers to the item reported as lost. |
| 03 | FoundItemID | Integer | --- | Refers to the item reported as found. |
| 04 | MatchScore | Float | Not Null | AI-calculated score of similarity. |
| 05 | MatchedAt | DateTime | Optional | Timestamp when the match was logged. |
| 06 | Status | String | Not Null | Match status: ‘Active’ or ‘Resolved’. |

### **NOTIFICATION**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | NotificationID | Integer | Primary Key | Unique ID for each notification. |
| 02 | UserID | Integer | --- | ID of the user receiving the notification. |
| 03 | AdminID | Integer | --- | ID of the admin receiving the notification. |
| 04 | Message | String | Not Null | Content of the notification. |
| 05 | Timestamp | DateTime | Not Null | When the notification was generated. |
| 06 | ClaimID | Integer | --- | Information about matching\_log. |
| 07 | MatchID | Integer | --- | Information about items (lost or found). |

### **FEEDBACK**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | FeedbackID | Integer | Primary Key | Unique identifier for each feedback entry. |
| 02 | UserID | Integer | --- | User who gave the feedback. |
| 03 | FeedbackType | String | Not Null | Type: ‘Bug Report’, ‘Feature Request’, etc. |
| 04 | Comment | String | Optional | Text of the feedback. |
| 05 | Rating | Integer | Range 1–5 | Numeric rating provided by the user. |
| 06 | CreatedAt | DateTime | Not Null | Timestamp when feedback was submitted. |

### **CLAIM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | ClaimID | Integer | Primary Key | Unique ID for each claim by user. |
| 02 | UserID | Integer | --- | ID of the user claiming the item. |
| 03 | MatchID | Integer | --- | ID of the Matching log that matches for specific item. |
| 04 | ClaimedAt | DateTime | Not Null | When the item is claimed, i.e., time. |

### **ITEM\_RETURN\_LOG**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | ReturnedID | Integer | Primary Key | Unique ID for each item return. |
| 02 | ItemID | Integer | --- | ID of the item being return. |
| 03 | AdminID | Integer | --- | ID of the admin recording the return. |
| 04 | Remarks | String | Not Null | Content of the return. |
| 05 | ReturnedAt | DateTime | Not Null | When the return was done, i.e., time. |
| 06 | Status | String | Not Null | Received or not received status. |
| 07 | UserID | Integer | --- | ID of user who receives item. |

## RELATIONSHIPS

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** | **Participating Entities** | **Relation** | **Business Rule** |
| 01 | User, Report | User creates Report | A user can report multiple items; each report belongs to one user. |
| 02 | Admin, Verification | Admin approves Verification | An admin verifies item claims through the verification module. |
| 03 | Item, Matching\_Log | Item appears in Match | A single item can be part of multiple match logs (lost or found sides). |
| 04 | User, Notification | User receives Notification | A user can receive multiple notifications. |
| 05 | Admin, Notification | Admin receives Notification | An admin can receive multiple notifications. |
| 06 | User, Feedback | User gives Feedback | A user can give feedback on items or system experiences. |
| 07 | User, Item | User claims Items | A user can claim items if found matched. |
| 08 | Admin, Record | Admin Checks Record | An Admin can check records of items claimed by user. |

## ENTITY RELATIONSHIP DIAGRAM [3]:

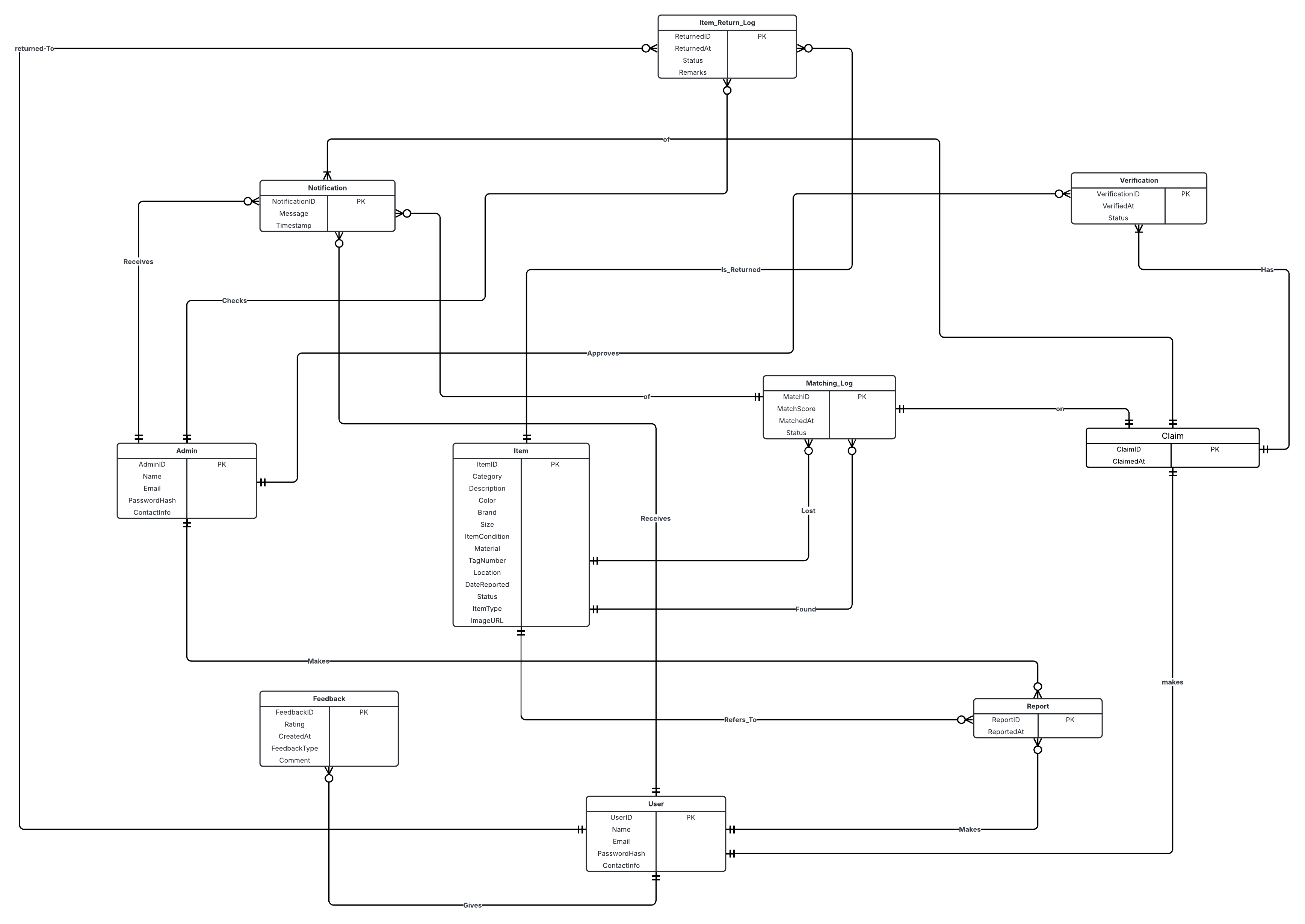


Fig 1.1: Entity Relationship Diagram (Version 2.0)

# CHAPTER 3: LOGICAL DATABASE DESIGN

* 1. **RELATIONAL SCHEMA:**

The process of converting our **Entity-Relationship Diagram (ERD)** into a **Relational Schema** involved analyzing each entity, identifying primary keys, and mapping relationships through appropriate Foreign Keys. We ensured that the resulting schema was **fully normalized to Third Normal Form (3NF)** to eliminate redundancy and preserve data integrity.

Each entity in the ERD was translated into a relation (table), with attributes becoming columns and **Primary Keys (PK)** clearly defined. For relationships between entities, **Foreign Keys (FK)** were introduced to maintain referential integrity. Where applicable, **associative entities** were created to handle many-to-many relationships, such as between User, Item, and Claim.

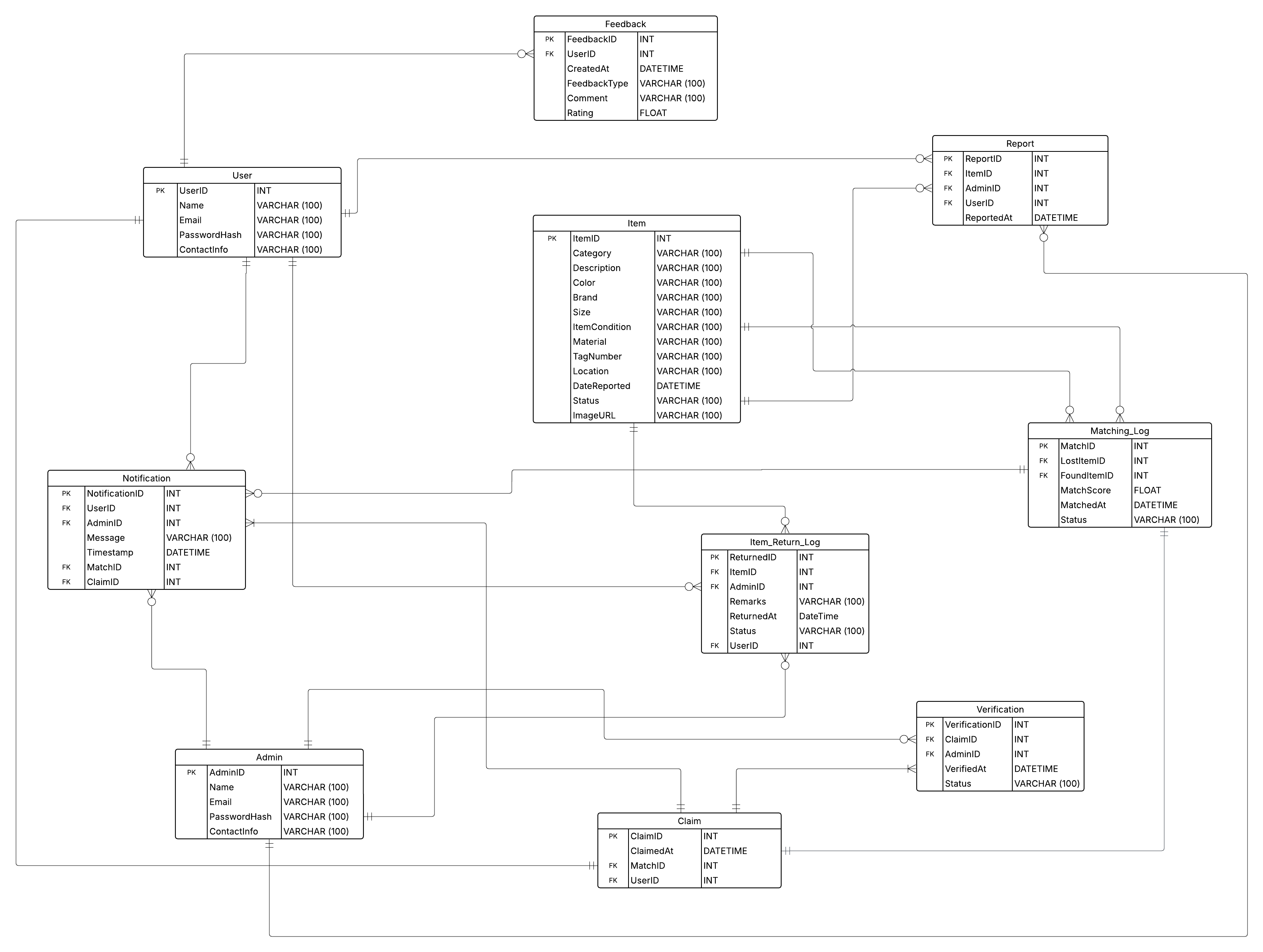


Fig 1.2: Relational Schema Diagram (Version 1.0)

This relational schema effectively captures the **conceptual design** of our system and is ready for implementation in any relational DBMS such as MySQL. It ensures consistency, supports complex queries, and enforces integrity across all interrelated data.

* 1. **FUNCTIONAL DEPENDENCIES:**

In a relational database, a functional dependency (FD) exists when the value of one attribute (or a combination of attributes) determines the value of another attribute. Below, we list the functional dependencies for each table in our schema, along with concrete examples to clarify how they work in real-world scenarios.

#### **USER**

1. **Functional Dependencies**:

UserID → Name, Email, PasswordHash, ContactInfo

1. **Example**:

If UserID = 2, we can determine that Name = "Sara", Email = "sara@example.com".

#### **ADMIN**

1. **Functional Dependencies**:

AdminID → Name, Email, PasswordHash, ContactInfO

1. **Example**:

AdminID = 3 determines a unique admin and their details.

#### **ITEM**

1. **Functional Dependencies**:

ItemID → Category, Description, Color, Brand, Size, ItemCondition, Material, TagNumber, Location, DateReported, Status, ImageURL, ItemType

1. **Example**:

If ItemID = 10, we can determine that it's a "Red Nike Bag" reported on "2025-05-12".

#### **FEEDBACK**

1. **Functional Dependencies**:

FeedbackID → UserID, CreateAt, FeedbackType, Comment, Rating

1. **Example**:

FeedbackID = 7 implies UserID = 3, FeedbackType = "General", and Rating = 4.5

#### **REPORT**

1. **Functional Dependencies**:

ReportID → ItemID, AdminID, UserID, ReportedAt

1. **Example**:

ReportID = 5 determines which item was reported, by which user, and to which admin.

#### **NOTIFICATION**

1. **Functional Dependencies**:

NotificationID → UserID, AdminID, ClaimID, MatchID, Message, Timestamp

1. **Example**:

NotificationID = 4 tells us that user UserID = 8 received "Item matched" at 2025-05-15 10:00:00.

#### **MATCHING\_LOG**

1. **Functional Dependencies**:

MatchID → LostItemID, FoundItemID, MatchScore, MatchedAt, Status

1. **Example**:

MatchID = 12 tells us LostItemID = 5, FoundItemID = 11, MatchScore = 0.92.

#### **CLAIM**

1. **Functional Dependencies**:

ClaimID → ClaimedAt, MatchID, UserID

1. **Example**:

ClaimID = 8 tells us the claim was made by UserID = 4 for MatchID = 8 on a specific date.

#### **VERIFICATION**

1. **Functional Dependencies**:

VerificationID → ClaimID, AdminID, VerifiedAt, Status

1. **Example**:

VerificationID = 5 determines that AdminID = 2 verified ClaimID = 5.

#### **ITEM\_RETURN\_LOG**

1. **Functional Dependencies**:

ReturnID → ItemID, AdminID, ReturnedAt, Remarks, Status

1. **Example**:

ReturnID = 3 tells us that ItemID = 7 was returned by AdminID = 1 with remarks "Delivered".

## NORMALIZATION:

The Lost & Found Management System (TraceIt) underwent a thorough normalization process using the identified functional dependencies. The goal was to achieve **Third Normal Form (3NF)** for all relations.

### **STEP-BY-STEP NORMALIZATION PROCESS**

#### **First Normal Form (1NF)**

All attributes contain only **atomic** values, and each record is **uniquely identifiable** by a primary key.  
All tables in the schema satisfy 1NF because:

1. There are no repeating groups or arrays.
2. Each column stores a single piece of data.

#### **Second Normal Form (2NF)**

All tables in the schema satisfy 2NF because:

1. Table is in 1NF.
2. All tables use **single-column primary keys.**
3. No **partial dependencies** (i.e., no non-prime attribute depends on a part of a composite key).

#### **Third Normal Form (3NF)**

All tables in the schema satisfy 3NF because:

1. Table is in 3NF.
2. No **transitive dependencies** (i.e., non-key attributes should not depend on other non-key attributes).
3. And all non-key attributes in each table depend only on the primary key.

### **FINAL 3NF RELATIONS**

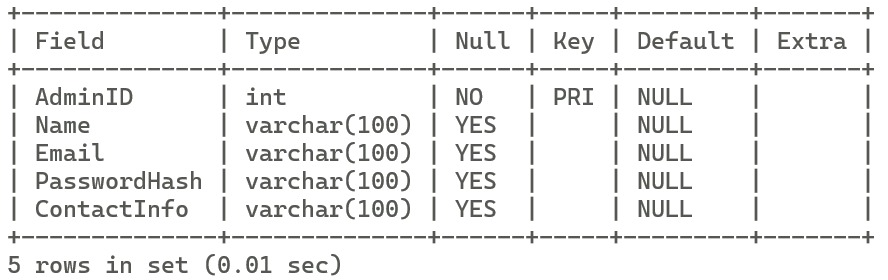
|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** | **Relation Name** | **Primary Key** | **Foreign Keys** |
| 01 | **User** | UserID | ---- |
| 02 | **Admin** | AdminID | ---- |
| 03 | **Item** | ItemID | ---- |
| 04 | **Feedback** | FeedbackID | UserID → User |
| 05 | **Report** | ReportID | UserID → User, AdminID → Admin, ItemID → Item |
| 06 | **Notification** | NotificationID | UserID → User, AdminID → Admin, ClaimID → Claim, MatchID → Matching\_Log |
| 07 | **Matching\_Log** | MatchID | LostItemID → Item, FoundItemID → Item |
| 08 | **Claim** | ClaimID | MatchID → Matching\_Log, UserID → User |
| 09 | **Verification** | VerificationID | ClaimID → Claim, AdminID → Admin |
| 10 | **Item\_Return\_Log** | ReturnID | ItemID → Item, AdminID → Admin, UserID → User |

# CHAPTER 4: PHYSICAL DATABASE DESIGN

* 1. **STRUCTURE OF THE TABLES:**

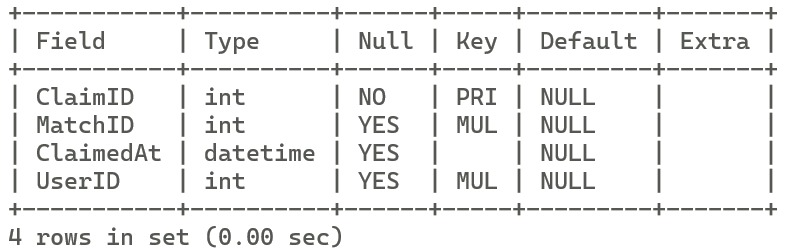
### **ADMIN**

**Query:** Describe Admin;



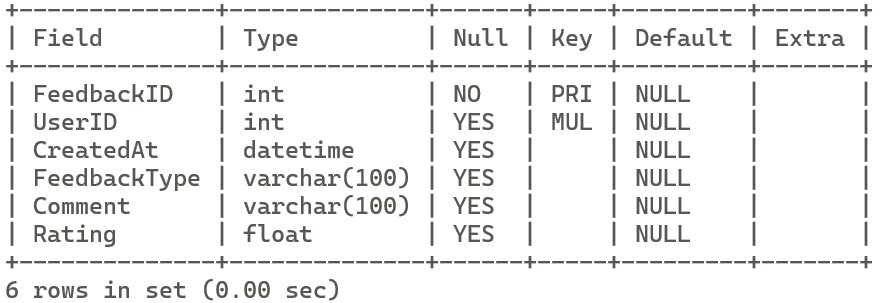
### **CLAIM**

**Query:** Describe Claim;



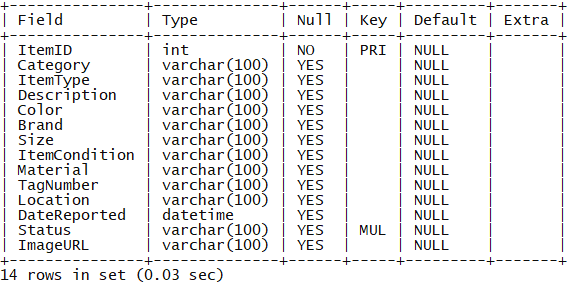
### **FEEDBACK**

**Query:** Describe Feedback;



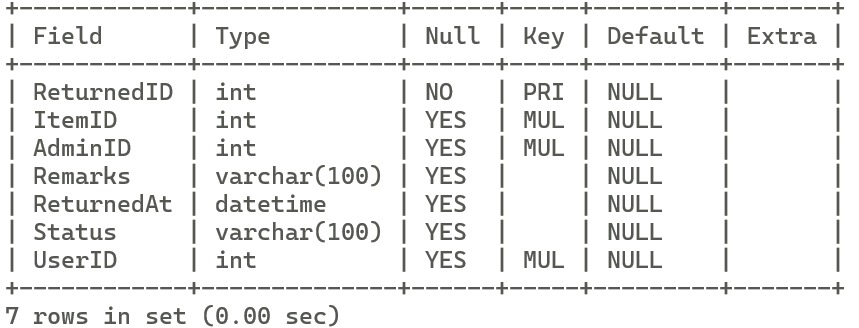
### **ITEM**

**Query:** Describe Item;



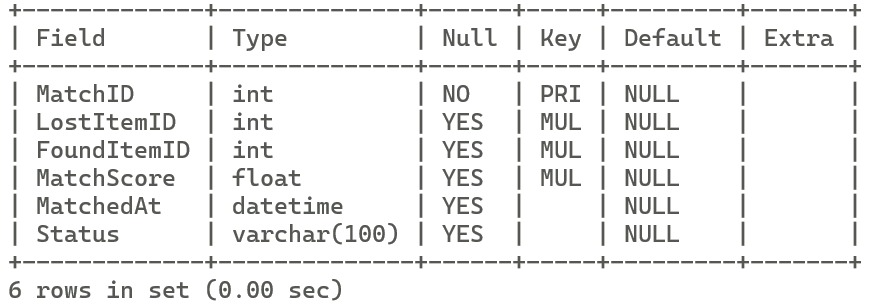
### **ITEM\_RETURN\_LOG**

**Query:** Describe Item\_Return\_Log;



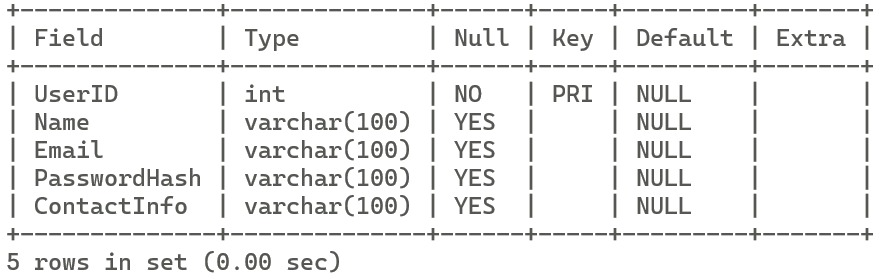
### **MATCHING\_LOG**

**Query:** Describe Matching\_Log;



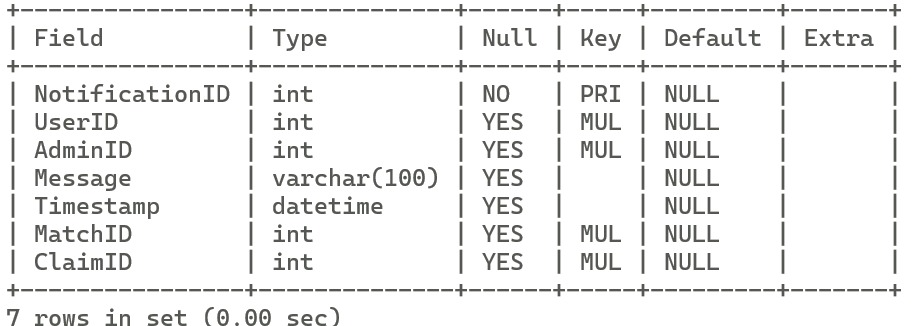
### **USER**

**Query:** Describe User;



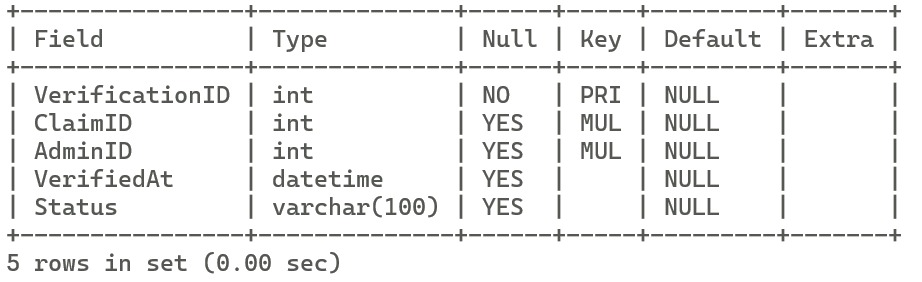
### **NOTIFICATION**

**Query:** Describe Notification;



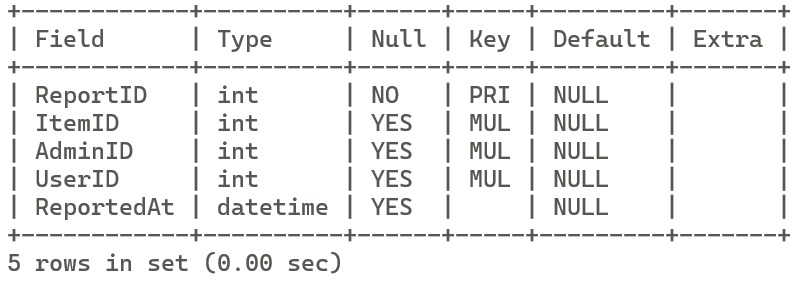
### **VERIFICATION**

**Query:** Describe Verification;



### **REPORT**

**Query:** Describe Report;

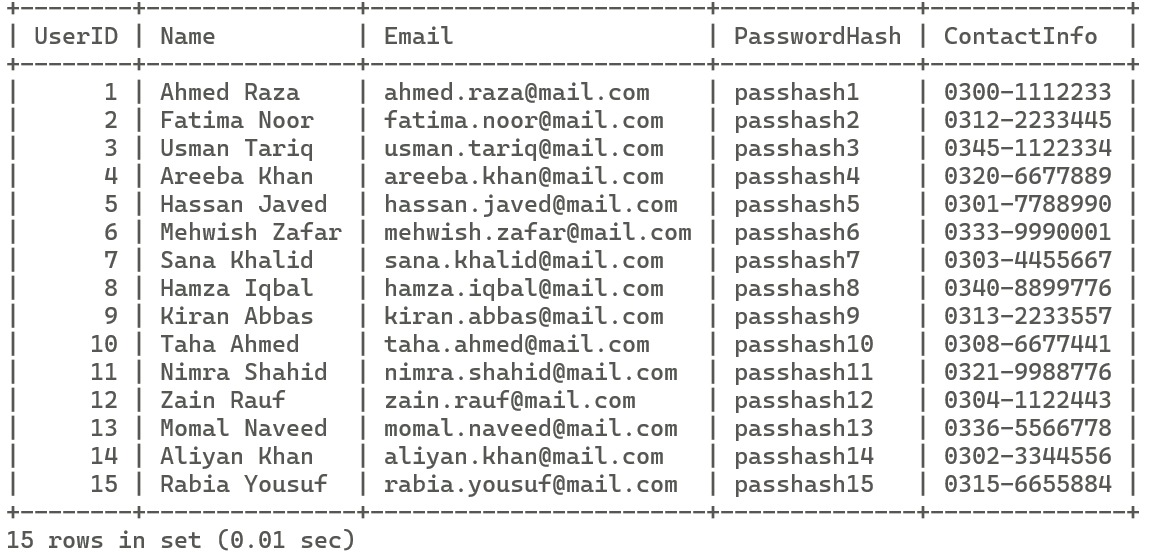


* 1. **DATA SAMPLES INSIDE TABLES:**

To confirm that the database has been correctly populated, the following SELECT \* queries were executed on each table. The sample output shows that each table contains at least 15 rows of data, as required.

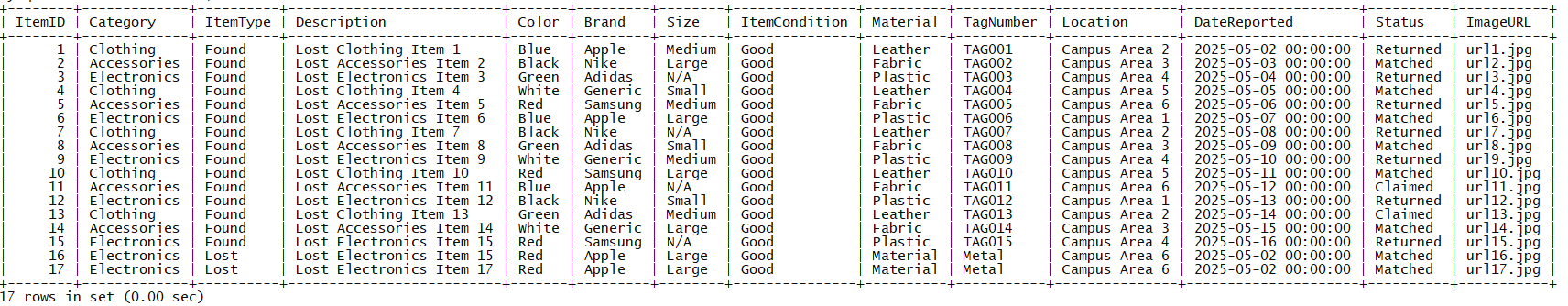
### **USER**

**Query:** Select \* from User;



### **ITEM**

**Query:** Select \* from Item;



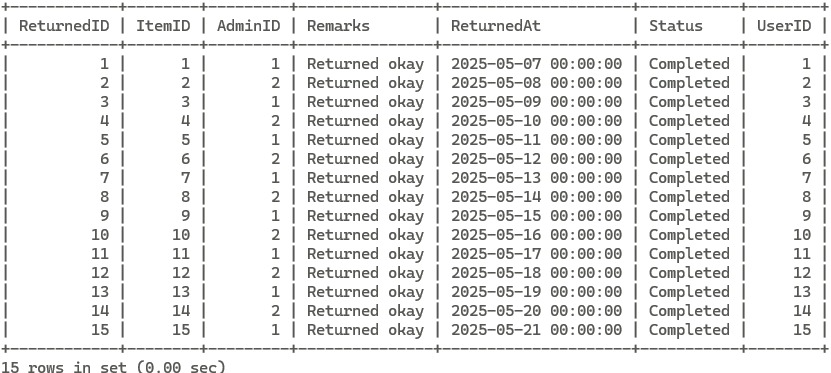
### **ADMIN**

**Query:** Select \* from Admin;



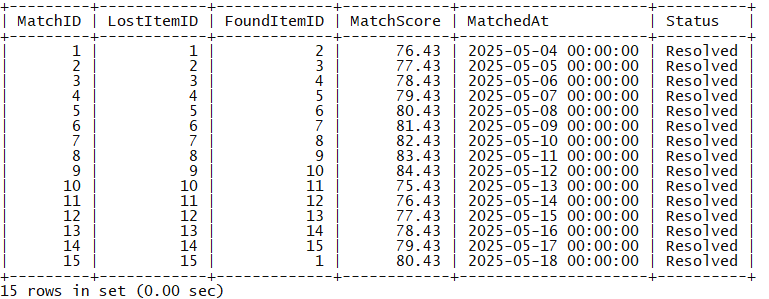
### **ITEM\_RETURN\_LOG**

**Query:** Select \* from Item\_Return\_Log;



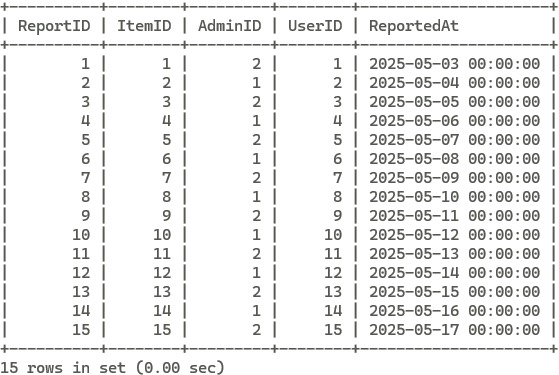
### **MATCHING\_LOG**

**Query:** Select \* from matching\_log;



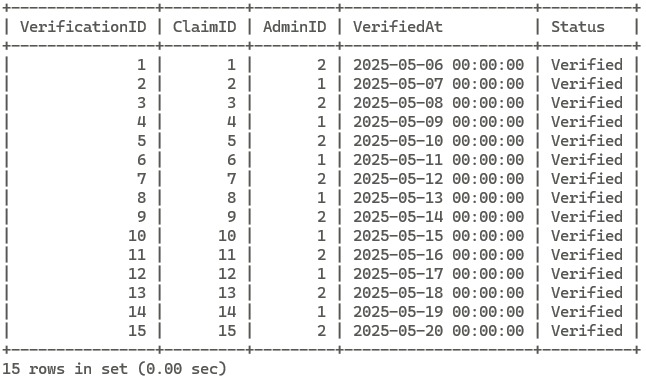
### **REPORT**

**Query:** Select \* from Report Limit 15;



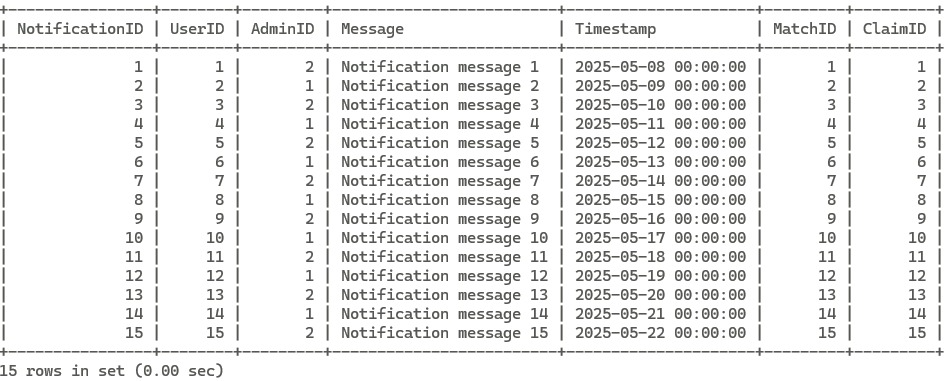
### **VERIFICATION**

**Query:** Select \* from Verification;



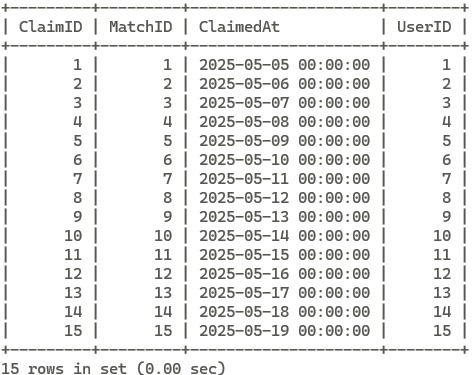
### **NOTIFICATION**

**Query:** Select \* from notification;



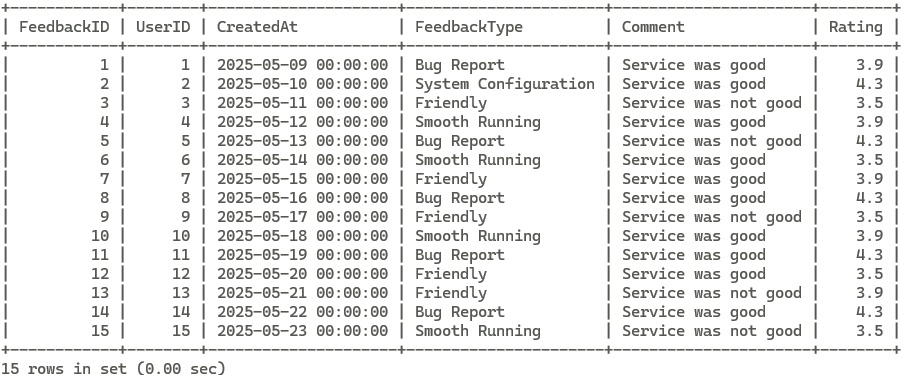
### **CLAIM**

**Query:** Select \* from claim;



### **FEEDBACK**

**Query:** Select \* from feedback;



* 1. **QUERIES RESULTS:**

**Use Case 1:** List all lost items reported with user details

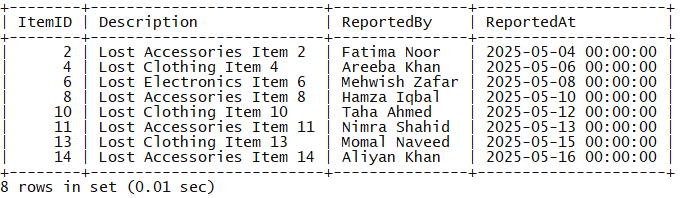
**Query:** SELECT I.ItemID, I.Description, U.Name AS ReportedBy, R.ReportedAt

        FROM Item AS I

        JOIN Report R ON I.ItemID = R.ItemID

        JOIN User U ON R.UserID = U.UserID

        WHERE I.ItemType = 'Lost';



**Use Case 2:** Show details of all claimed items with their matched items

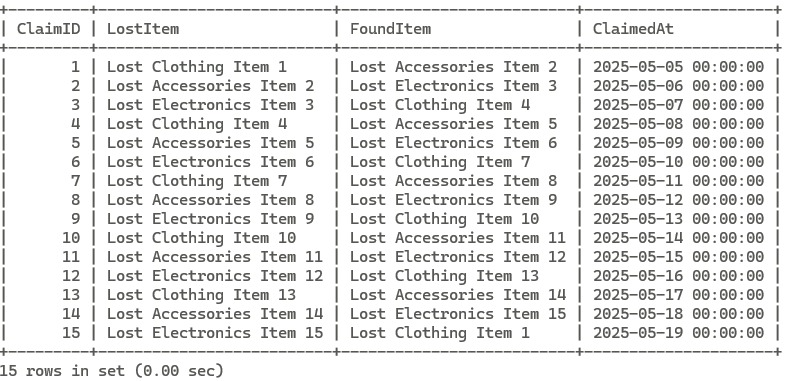
**Query:** SELECT C.ClaimID, L.Description AS LostItem, F.Description AS FoundItem, C.ClaimedAt

        FROM Claim C

        JOIN Matching\_Log M ON C.MatchID = M.MatchID

        JOIN Item L ON M.LostItemID = L.ItemID

        JOIN Item F ON M.FoundItemID = F.ItemID;



**Use Case 3:** List users who have given more than one feedback with average rating

**Query:** SELECT U.Name, COUNT(F.FeedbackID) AS FeedbackCount, AVG(F.Rating)

        FROM Feedback F

        JOIN User U ON F.UserID = U.UserID

        GROUP BY U.Name

        HAVING COUNT(F.FeedbackID) > 1

        ORDER BY AvgRating DESC;



**Use Case 4:** Count number of items reported by each user, ordered by count descending.

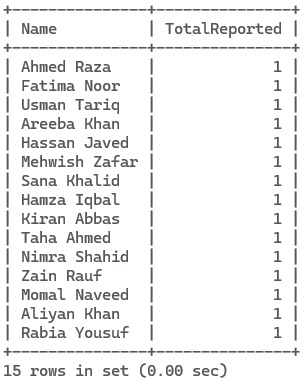
**Query:** SELECT U.Name, COUNT(R.ItemID) AS TotalReported

      FROM User U

        JOIN Report R ON U.UserID = R.UserID

        GROUP BY U.Name

        ORDER BY TotalReported DESC;



**Use Case 5:** List users who claimed items that were verified.

**Query:** SELECT Name FROM User

        WHERE UserID IN (

            SELECT C.UserID FROM Claim C

            JOIN Verification V ON C.ClaimID = V.ClaimID

            WHERE V.Status = 'Verified' );



**Use Case 6:** Show notifications sent by Admins related to verified claims only.

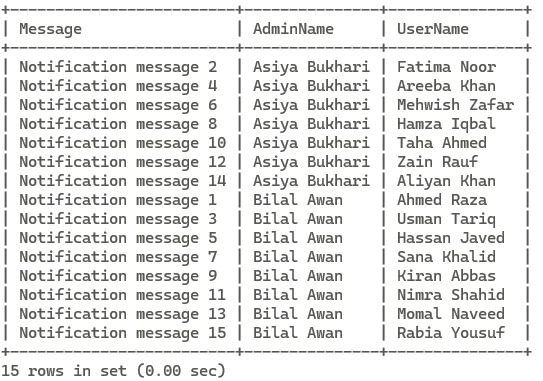
**Query:** SELECT N.Message, A.Name AS AdminName, U.Name AS UserName

        FROM Notification N

        JOIN Admin A ON N.AdminID = A.AdminID

        JOIN User U ON N.UserID = U.UserID

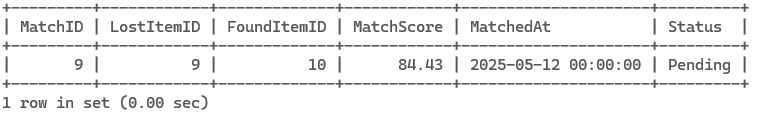
      WHERE N.ClaimID IN (SELECT ClaimID FROM Verification WHERE Status = 'Verified');



**Use Case 7:** Find item(s) with the highest match score.

**Query:** SELECT \* FROM Matching\_Log

        WHERE MatchScore = (SELECT MAX(MatchScore) FROM Matching\_Log);



**Use Case 8:** Show the latest 5 returned items with their users and admins

**Query:** SELECT IRL.ReturnedID, I.Description, U.Name AS ReturnedTo, A.Name AS HandledBy, IRL.ReturnedAt

        FROM Item\_Return\_Log IRL

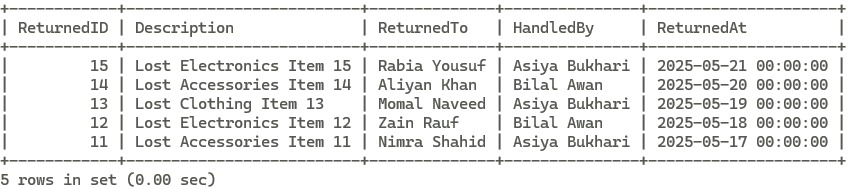
        JOIN Item I ON IRL.ItemID = I.ItemID

        JOIN User U ON IRL.UserID = U.UserID

        JOIN Admin A ON IRL.AdminID = A.AdminID

        ORDER BY IRL.ReturnedAt DESC

        LIMIT 5;



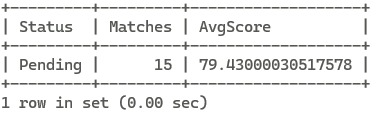
**Use Case 9:** Average match score for each status type in Matching\_Log

**Query:** SELECT Status, COUNT(\*) AS Matches, AVG(MatchScore) AS AvgScore

        FROM Matching\_Log

        GROUP BY Status

        ORDER BY AvgScore DESC;

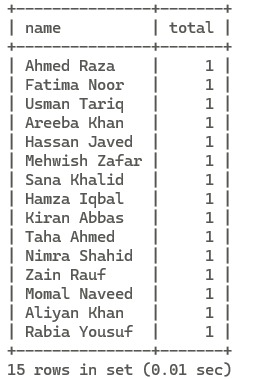


**Use Case 10:** List users with the total number of claims they made (include users with 0 claims)

**Query:** SELECT U.Name, (

            SELECT COUNT(\*) FROM Claim C WHERE C.UserID = U.UserID) AS TotalClaims

        FROM User U;



# CHAPTER 5: REFERENCES

|  |  |
| --- | --- |
| [1] | A. K. H. F. &. S. S. Silberschatz, Database System Concepts, McGraw-Hill, 2019. |
| [2] | Firebase, "Firebase Authentication Docs," 2024. [Online]. Available: https://firebase.google.com/docs/auth. [Accessed 2025]. |
| [3] | R. B. M. S. K. Usman Ghani, "Database Design Document (V 1.0)," 2025. |

## APPENDIX: AI PROMPTS

Our group responsibly used **AI tools (i.e., ChatGPT by OpenAI)** for the purpose of **analyzing, interpreting**,and **validating** specific elements of our database project **TraceIt**. The AI was used solely to assist in understanding complex design concepts, reviewing relational structures, and exploring alternative perspectives — all work submitted was **authored, structured, and implemented independently** by the team.

Below is a list of prompts we engaged with for analytical clarity:

1. "Can you analyze our ERD and guide us on how to convert it into a relational schema?"
2. "Review our functional dependencies and explain how to structure them clearly."
3. "Help interpret the normalization process for our schema (1NF → 3NF)."
4. "Suggest realistic values for INSERT statements using Pakistani user data."
5. "Analyze our SQL queries and suggest improvements using JOINs and subqueries."
6. "Suggest more formal alternatives to the heading 'Task 1' in academic SQL documentation."
7. "Guide us on the appropriate format and structure of the Queries Result section in a database report."
8. "What is the best way to reference the use of AI tools in academic project reports?"