Project Report: CallHub – Phone Directory Management System

Github Link: https://github.com/arpitaa-k/CallHub-CS-432

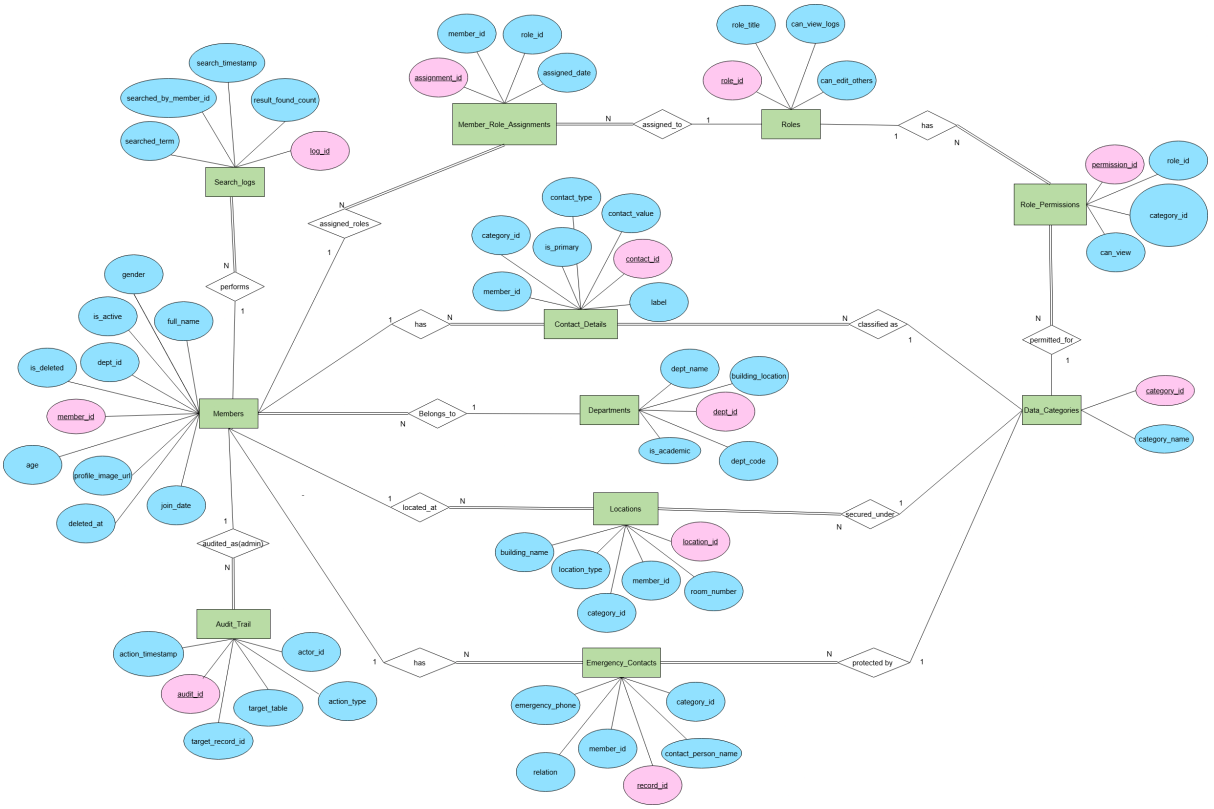
Entity-Relationship (ER) Diagram

Visual Representation Logic:

The ER model is designed with a centralized core entity structure, where the Members entity serves as the primary entity and is connected to multiple dependent entities.

Text-Based Diagram Flow:

- Central Node: Members
- Security Layer: Members connect to Roles, which connect to Data_Categories.
- Data Layer: Members connect to Contact_Details, Locations, and Emergency_Contacts.
- Audit Layer: Members connect to Search_Logs and Audit_Trail.



Relationship Justifications

This section details the cardinality and the logic behind every connection in the database.

A. Core Hierarchy & Assignment

a. **Departments ↔ Members**
   i. Relation Name: belongs_to
   ii. Cardinality: 1: N (One-to-Many)
   iii. Justification: A strict hierarchical rule is enforced, requiring every member (Student, Staff, or Faculty) to belong to exactly one home department (e.g., Computer Science). However, a single department functions as an organizational unit containing many members.

b. **Members ↔ Roles** (via Member_Role_Assignments)
   i. Relation Name: assigned_role / assigned_to
   ii. Cardinality: M : N (Many-to-Many, resolved via Bridge Table)
   iii. Justification: A direct 1:N relationship is insufficient because a single person may hold multiple functional titles simultaneously (e.g., a "Professor" who is also a "Warden"). The Member_Role_Assignments table resolves this by linking one Member to many Roles, and one Role to many Members.

c. **Roles ↔ Data_Categories** (via Role_Permissions)
   i. Relation Name: has_permission / permitted_for
   ii. Cardinality: M : N (Many-to-Many, resolved via Bridge Table)
   iii. Justification: This relationship forms the core of the "Keychain Security Model." It maps who (Role) can see what (Category). A Role (e.g., Warden) needs access to multiple categories (Public + Residential), and a single category (e.g., Public) must be accessible by multiple roles.

B. Directory Data & Security Categorization

d. **Members ↔ Contact_Details**
   i. Relation Name: has_contact
   ii. Cardinality: 1: N (One-to-Many)
   iii. Justification: Modern users have multiple communication channels (Personal Email, Work Email, Mobile, Landline). Normalizing this into a separate table allows a member to have an infinite number of contact points without creating phone1 and phone2 columns in the main table.

e. **Contact_Details ↔ Data_Categories**
   i. Relation Name: classified_as
   ii. Cardinality: N: 1 (Many-to-One)
   iii. Justification: Every single phone number or email is individually tagged with a security level. This allows a member's "Office Email" to be Public (Category 1) while their "Personal Mobile" remains Private (Category 2).

f. **Members ↔ Locations**
   i. Relation Name: has_location
   ii. Cardinality: 1: N (One-to-Many)

iii. Justification: Members often operate from multiple physical spaces (e.g., a Student has a Hostel Room and a Lab assignment; a Professor has an Office and a Research Center desk).

g. Locations ↔ Data_Categories

i. Relation Name: secured_under
ii. Cardinality: N: 1 (Many-to-One)
iii. Justification: Similar to contacts, physical locations have different privacy needs. An "Office" is generally public information, whereas a "Hostel Room" is residential data protected by the Residential category.

h. Members ↔ Emergency_Contacts

i. Relation Name: has_emergency_contact
ii. Cardinality: 1: N (One-to-Many)
iii. Justification: Separating this from the main Member table ensures that sensitive family data is not loaded during routine searches.

i. Emergency_Contacts ↔ Data_Categories

i. Relation Name: protected_by
ii. Cardinality: N: 1 (Many-to-One)
iii. Justification: This relation enforces a hard constraint that all emergency data is linked to the "Sensitive" or "Admin-Only" category, preventing accidental exposure to standard users.

C. Auditing & History

j. Members ↔ Search_Logs

i. Relation Name: performs_search
ii. Cardinality: 1: N (One-to-Many)
iii. Justification: Each member in the system can perform multiple search operations over time. Every search action generates a separate record in the Search_Logs table. However, each search log entry is associated with only one member (if logged in). Therefore, the relationship between Members and Search_Logs is one-to-many.

k. Members ↔ Audit_Trail

i. Relation Name: performs_action
ii. Cardinality: 1: N (One-to-Many)
iii. Justification: For accountability, any modification (INSERT, UPDATE, DELETE) is linked to the specific actor (Member) who performed it. This relationship ensures transparency and responsibility within the system.

D. Additional Constraints & Rules

1. **Database Integrity Constraints**

These are strictly enforced by the SQL table definitions.

a. Unique Identifiers:

i. Departments: The dept_code must be unique (e.g., you cannot have two 'CSE' departments).
ii. Roles: The role_title must be unique (e.g., 'Warden' can only be defined once).
iii. Categories: The category_name must be unique to prevent duplicate security levels.

b. Data Validation:
i. Age Limit: A CHECK constraint enforces age >= 16, ensuring valid data for university members.
ii. Restricted Inputs: Columns like gender, contact_type, location_type, and action_type use ENUM types. The database will reject any value that is not in the predefined list (e.g., preventing 'Cell Phone' instead of 'Mobile').

c. Referential Integrity:
i. Cascading Clean-up: ON DELETE CASCADE is applied to Contact_Details, Locations, and Emergency_Contacts. If a member is physically removed from the system, their personal data is automatically wiped to prevent orphan records. This will only work if a Hard delete is performed, but we mostly use soft delete so that data is preserved.
ii. Log Preservation: Search_Logs uses ON DELETE SET NULL. If a member is deleted, their search history remains in the database (with the member_id field set to NULL) for analytics purposes.

2. Application & Logic Constraints (Soft Rules)

a. Soft Deletion Protocol:
i. Records in the Members table are not physically deleted during standard operations.
ii. The system sets is_deleted = 1 and records the timestamp in deleted_at.
iii. Note: The application layer must filter out rows with is_deleted = 1 during normal searches.

b. Default Security Posture:
i. Contacts: New contact entries default to category_id = 2 (likely 'Residential' or 'Internal'). This ensures that if a user forgets to tag a phone number, it does not accidentally become Public.
ii. Locations: New locations default to category_id = 1 (Public), assuming most location entries are Offices or Labs.
iii. Roles: New roles default to can_edit_others = 0 (Read-Only), ensuring that administrative power is explicitly granted, never assumed.

c. Active vs. Deleted:
i. The schema distinguishes between is_active (Default 1) and is_deleted (Default 0). This allows the system to temporarily suspend

a user (Active = 0) without considering them "deleted" from the records.

d. Implicit Deny (The Matrix Rule):
    i. Access to data is denied by default. A Role can only view a Category if a specific row exists in the Role_Permissions table. If the row is missing, the data remains hidden.

Team Members and Contributions

Name	Roll Number	Specific Contributions
Anuja Chaudhari	23110034	Discussed and verified the schemas, generated the entries for all tables, created the SQL dump file, and verified the ER diagram
Arpita Kumawat	23110045	Discussed and verified the schemas, made the ER diagram, and verified the final report and the integrity constraints
Chaudhari Meshvakumari Jiteshbhai	23110075	Discussed and verified the schemas, made the ER diagram, and verified the entries of the tables and referential integrity
Seemanshi Mall	23110292	Discussed and verified the schemas, made the final report and verified the relations of the ER diagram, verified the core functionalities
Sia Hetal Jariwala	23110309	Discussed and verified the schemas, generated the entries for all tables, created the SQL dump file, and verified usage analytics