# CS 432: Databases

**Assignment 1: SafeDocs**

GitHub Repository

Harshita Singh, Dhruv Goel, Nikhil Goyal, Chinmay Pendse, Vedant Sharan

February 15, 2026

## 1 Project Overview

Our system, SafeDocs, a Secure PDF Management System, focuses on enforcing data integrity, normalization, and systematic conceptual modeling. The design ensures that document storage is not only scalable but also adheres to strict role-based access control and audit requirements.

## 2 Core Technical Functionalities

The system implements various functionalities; the five core functionalities out of these are as follows:

1. **Identity & Access Management (IAM):** Utilizing the `User`, `Role`, and `Permission` entities to manage authenticated access.
2. **Dynamic Policy Enforcement:** The `Policy` entity classifies documents and restricts access based on `MaxAllowedRoleID`.
3. **Audit Logging & Tracking:** The `Log` entity records actions, timestamps, and IP addresses to maintain a transparent audit trail.
4. **Version Control:** Automatic tracking of document changes via the `Version` entity, allowing for historical rollbacks.
5. **Security & Encryption:** Management of document security through `Password` hashing and encryption method metadata.

A detailed overview of other functionalities can be viewed in the UML diagram.

## 3 Conceptual Modeling: UML Class Diagram

The UML diagram serves as our high-level conceptual blueprint. It identifies classes, their internal attributes, and functional methods.

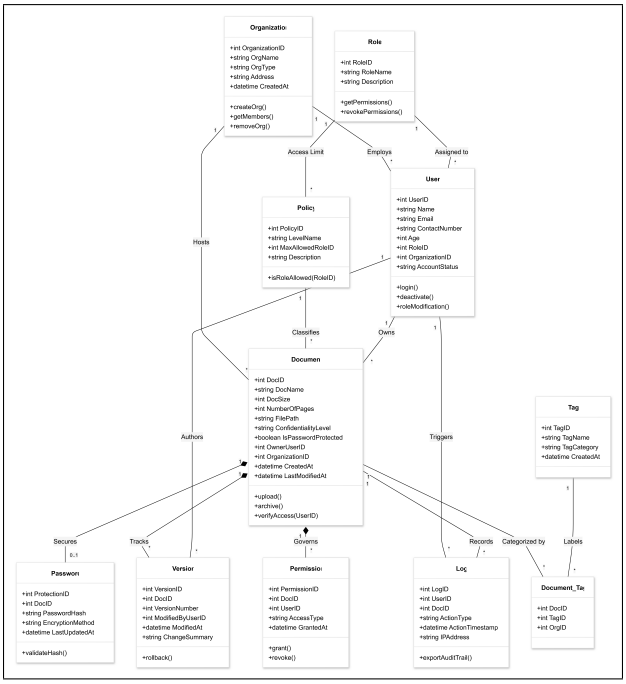


Figure 1: UML Diagram for database

## 3.1 Key Classes and Attributes

- **User (Member Table):** Includes required attributes such as `Name`, `Age`, `Email`, and `ContactNumber`.
- **Document:** The central entity containing metadata like `ConfidentialityLevel` and `IsPasswordProtected`.
- **Methods:** Behavior is defined through methods such as `verifyAccess(UserID)` in the Document class and `exportAuditTrait()` in the Log class.

## 3.2 Structural Relationships

The design utilizes composition and aggregation to show ownership:

- **Composition:** The `Permission` and `Version` entities are shown with a black diamond, indicating they are part of the `Document` lifecycle and cannot exist independently.
- **Multiplicity:** Standard UML conventions are followed (e.g., $1..*$ for Organization to User) to represent the hierarchy.

## 3.3 Entities present

- Document
- Document_Tag (Associative Entity)
- Log
- Organization

- Password
- Permission
- Policy
- Role
- Tag
- User
- Version

## 3.4 Attributes in each table

- **Document**: DocID, DocName, DocSize, NumberOfPages, FilePath, ConfidentialityLevel, IsPasswordProtected, OwnerUserID, OrganizationID, CreatedAt, LastModifiedAt
- **Document_Tag**: DocID, TagID, OrgID
- **Log**: LogID, UserID, DocID, ActionType, ActionTimestamp, IPAddress
- **Organization**: OrganizationID, OrgName, OrgType, Address, CreatedAt
- **Password**: ProtectionID, DocID, PasswordHash, EncryptionMethod, LastUpdatedAt
- **Permission**: PermissionID, DocID, UserID, AccessType, GrantedAt
- **Policy**: PolicyID, LevelName, MaxAllowedRoleID, Description
- **Role**: RoleID, RoleName, Description
- **Tag**: TagID, TagName, TagCategory, CreatedAt
- **User**: UserID, Name, Email, ContactNumber, Age, RoleID, OrganizationID, AccountStatus
- **Version**: VersionID, DocID, VersionNumber, ModifiedByUserID, ModifiedAt, ChangeSummary

## 3.5 How Entities are related(main)

- **Organization – User (Employs):** An Organization can employ multiple Users, establishing a one-to-many (1:N) relationship. Each User is associated with exactly one Organization through the OrganizationID attribute.
- **Role – User (Assigned to):** A single Role can be assigned to many Users (1:N). However, each User is assigned exactly one Role, which determines their level of access and responsibilities within the system.

- **Role – Policy (Defines Access Limit):** One Role may define multiple Policies (1:N). Policies specify confidentiality rules and access limits based on the maximum allowed role.
- **Policy – Document (Classifies):** Each Document is classified under one Policy, while a Policy can classify multiple Documents (1:N). This ensures that documents follow predefined confidentiality levels.
- **User – Document (Owns):** A User can own multiple Documents (1:N). Each Document has exactly one owner, identified using OwnerUserID, establishing accountability and control.
- **Document – Version (Tracks):** A Document can have multiple Versions (1:N), representing its revision history. Each Version is associated with exactly one Document.
- **Document – Password (Secures):** A Document may optionally have one Password for protection (1:0..1). A Password entity cannot exist independently and must be associated with a Document.
- **User – Log (Triggers):** A User can generate multiple Log entries (1:N). Each Log record captures an action performed by a specific User.
- **Document – Log (Records):** A Document can have multiple associated Log entries (1:N). Each Log entry corresponds to a specific Document and records activities performed on it.
- **Document – Tag (Categorization):** Document and Tag share a many-to-many (M:N) relationship, which is resolved through the Document_Tag associative entity. A Document can have multiple Tags, and each Tag can label multiple Documents for classification and search purposes.

## 3.6 Methods to be used

- **Organization**: createOrg(), getMembers(), removeOrg()
- **Role**: getPermissions(), revokePermissions()
- **Policy**: isRoleAllowed(RoleID)
- **User**: login(), deactivate(), roleModification()
- **Document**: upload(), archive(), verifyAccess(UserID)
- **Version**: rollback()
- **Password**: validateHash()
- **Permission**: grant(), revoke()
- **Log**: exportAuditTrail()

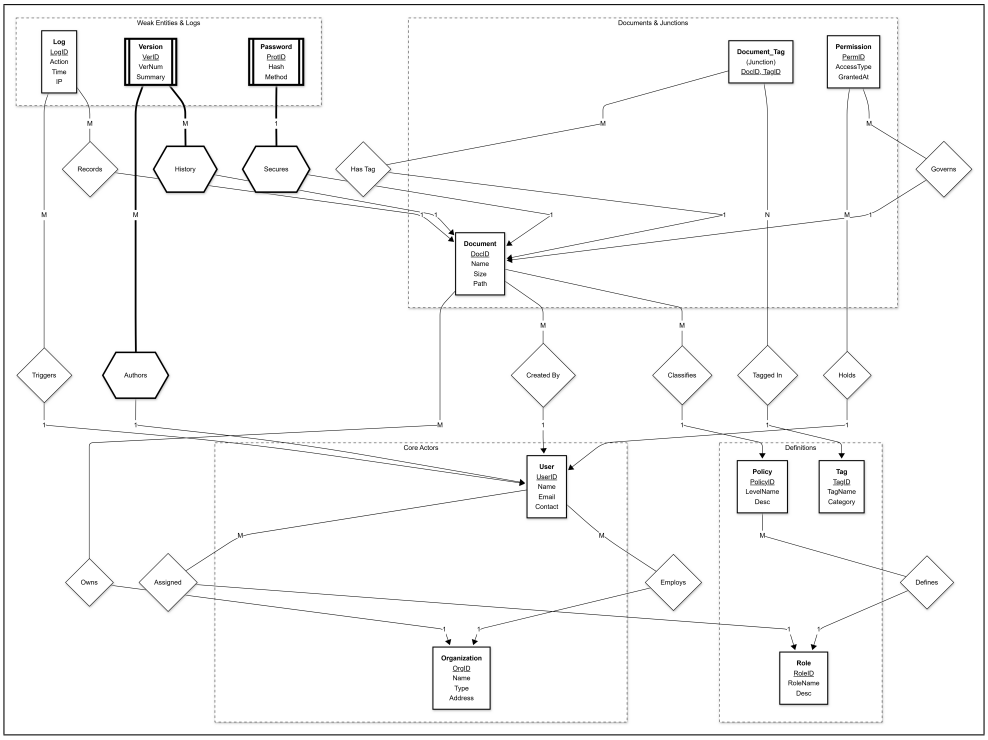


Figure 2: ER diagram for the database

# 4 Logical Modeling: ER Diagram

The ER diagram converts UML classes into database entities, focusing on keys and relational constraints.

## 4.1 Entity-Relationship Mappings

- **Primary Keys (PK):** All entities have underlined unique identifiers (e.g., `OrgID`, `UserID`, `DocID`).
- **Mapping Tables:** The M:N relationship between `Document` and `Tag` is resolved through the `Document_Tag` intersection table.
- **Foreign Keys (FK):** Directed arrows in the ER diagram indicate the flow of foreign keys, such as `RoleID` being assigned to `User`.

## 4.2 How Entities are related (main)

- **Organization – User (Employs):** An Organization employs multiple Users. Each User belongs to exactly one Organization.
- **Role – User (Assigned):** A Role is assigned to Users. One Role can be assigned to many Users.
- **Role – Policy (Defines):** A Role defines access constraints through Policies. One Role may define multiple Policies.
- **Policy – Document (Classifies):** Each Document is classified under a Policy, while a Policy can classify multiple Documents.

- **User – Document (Created By / Owns):** A User creates and owns Documents. One User can create multiple Documents.
- **Document – Version (History):** A Document maintains multiple Versions representing its revision history.
- **Document – Password (Secures):** A Document may be secured by a Password entity.
- **User – Document – Permission (Governs):** Permission acts as an associative entity between User and Document to manage access rights.
- **User – Log (Triggers):** A User triggers Log entries when performing actions.
- **Document – Log (Records):** A Document has multiple Log records associated with it.
- **Document – Tag (Has Tag / Tagged In):** Documents are associated with Tags through a many-to-many relationship resolved by Document_Tag.

## 4.3 Cardinality Constraints

- **Organization – Employs – User:** Each User belongs to exactly one Organization (N:1).
- **Role – Assigned – User:** One Role can be assigned to many Users (1:N).
- **Role – Defines – Policy:** One Role defines many Policies (1:N).
- **Policy – Classifies – Document:** One Policy can classify many Documents (1:N).
- **User – Created By – Document:** One User can create many Documents (1:N).
- **Document – History – Version:** One Document can have many Versions (1:N).
- **Document – Secures – Password:** One Document may have at most one Password (1:0..1). Each Password is associated with exactly one Document (1:1).
- **User – Triggers – Log:** One User can trigger many Logs (1:N).
- **Document – Records – Log:** One Document can have many Logs (1:N).
- **Document – Has Tag – Document_Tag:** One Document can appear in many Document_Tag entries (1:N).
- **Tag – Tagged In – Document_Tag:** One Tag can appear in many Document_Tag entries (1:N).
- **User – Holds – Permission:** One User can hold many Permissions (1:N).
- **Document – Governs – Permission:** One Document can have many Permissions (1:N).

### 4.4 Participation Constraints

Here we are representing double line by thick line telling about total participation and weak relationship is represented by a hexagon.

- **Version** has **total participation** in the History relationship (double line), meaning a Version cannot exist without a Document.
- **Password** has **total participation** in the Secures relationship (double line), indicating it is a weak entity dependent on Document.
- **Document** has total participation in the identifying relationships for Version and Password.
- **Log** has total participation in both Records (with Document) and Triggers (with User), meaning each Log must be linked to a Document and a User.
- Other entities such as Organization, Role, Tag, and Policy show **partial participation**, since they may exist without necessarily being linked in all relationships.

## 5 Transition and Normalization Logic

### 5.1 UML to ER Transition

The transition involved several critical steps:

- **Method Stripping:** While UML classes include methods like `login()`, these were abstracted into functional requirements for the application layer, while the ER diagram focused solely on the resulting data state.
- **Cardinality Adjustment:** UML associations were translated into FK constraints in the ER model to ensure referential integrity.

### 5.2 Integrity Constraints

To satisfy Module A requirements:

- **NOT NULL Constraints:** At least three columns in every table (e.g., `Name`, `Email`, and `Status` in `User`) are marked NOT NULL.
- **Referential Integrity:** All relationships utilize FKs to ensure that updates or deletes do not leave orphaned records.
- **Logical Constraints:** Business rules, such as `LastModifiedAt` must be greater than or equal to `CreatedAt`, are enforced at the schema level.

## 6 Team Contributions

In accordance with the project guidelines:

- **Harshita Singh:** Documentation

- **Dhruv Goel:** Formal Verification and Documentation
- **Nikhil Goyal:** Python Scripting and Diagram Generation
- **Chinmay Pendse:** Integration with SQL and Diagram Editing
- **Vedant Sharan:** Code/ Constraints Verification