## ABSTARCT

The Apartment Visitor Management System is a comprehensive solution aimed at enhancing both security and convenience in residential complexes. In many societies today, visitor management is still handled with paper logs where guests scribble their names, which often lacks the necessary verification and accountability. This outdated method leaves the door open for potential security risks, as unauthorized individuals may gain access without sufficient oversight.In contrast, the Apartment Visitor Management System transforms this process by offering a modern, secure, and efficient approach to welcoming visitors. With this system, each visitor is accurately recorded and verified, ensuring that only authorized individuals can enter the premises. Regular visitors, such as family members or delivery personnel, can be issued access passes, making their visits seamless and hasslefree. The system also enables real-time tracking of all visitors, allowing security staff and residents to monitor who is on the premises at any given time.Beyond improving security, the system offers a more professionalized and organized way to manage guest entries. It eliminates the need for manual record-keeping, reducing human error and administrative workload. Detailed logs and reports are automatically generated, which can be accessed easily, offering transparency and peace of mind to residents and society management. The ability to view visitor history also aids in investigations should any security issues arise .

# INTRODUCTION

## 1.1 Introduction to Database Management System

A Database Management System (DBMS) is a software used to store and manipulate data in a database. DBMS Software provides an interface between the end-user and the database by simultaneously managing the data, the database schema, and the database engine which in turn facilitates data organization and manipulation. The main aim of DBMS is to provide convenience in storing and retrieving database information efficiently.

## 1.2 Background of the Project

The Apartment Visitor Management System (AVMS) is a web-based application designed to streamline the process of managing and tracking visitors in a society or apartment complex. This system offers a secure and efficient way for societies to handle visitor records and issue visitor passes. The application plays a crucial role in assisting security guards and administrators in monitoring visitor traffic, ensuring that all entries and exits are logged accurately. The system is developed using PHP and MySQL to store and manage the data. AVMS features a user-friendly interface where the admin can perform several tasks. These include viewing the total number of visitors within a specified period, managing visitor categories, adding new visitors, and managing visitor records, including their exit times. The admin can also create, view, and delete entry passes, search for visitors using specific details like name and phone number, and generate reports for visitor and visitor pass activities over selected date ranges. Additionally, the admin has the ability to update their profile, change their password, and recover it if needed.

## 1.3 Necessity of the Project

The purpose of developing the Apartment Visitor Management System is to replace traditional manual methods with a computerized solution that facilitates faster, more efficient record-keeping and reporting. This system helps manage the data related to apartment visitors and provides a centralized database for easy retrieval. It is designed to simplify the management of visitor information, improve security, and support gate guards in handling large volumes of visitor traffic. The system ensures that only the administrator has access to sensitive functions such as adding visitors and generating reports, making the data processing both secure and efficient. The Apartment Visitor Management System is a powerful and flexible tool, providing real, tangible benefits to apartment societies by enhancing visitor management and security.

## 1.4 Applications and advantages

The Apartment Visitor Management System (AVMS) finds applications in efficiently managing visitor records, issuing visitor passes, and maintaining a centralized database for quick retrieval of information. It enhances security by enabling real-time tracking of visitor entries and exits, assists gate guards in verifying credentials, and ensures that all sensitive data is securely handled by authorized administrators. The system simplifies the search for specific visitor details, generates customizable reports for audits or analysis, and supports seamless management of large volumes of visitor traffic. By automating these processes, AVMS offers significant advantages such as improved efficiency, accuracy, and reliability. It replaces traditional manual methods with an intuitive interface, reducing administrative overhead while enhancing the visitor experience. Moreover, its scalability and cost-effectiveness make it an ideal solution for apartment complexes of all sizes, providing a secure, centralized, and streamlined approach to visitor management.

## 1.5 Implementation

The implementation of the Apartment Visitor Management System (AVMS) involves developing a dynamic backend using PHP to handle server-side operations such as user authentication, visitor data management, and the generation of reports. For the frontend, HTML, CSS, and JavaScript are utilized to create an intuitive and responsive user interface, ensuring seamless interaction between administrators and the system. This integration of a PHP-powered backend with a visually appealing and user-friendly frontend enables efficient and secure management of visitor records while providing an enhanced user experience.

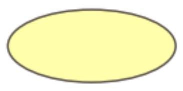
# DESIGN

Design is the first critical step in software development, translating requirements into a blueprint for the system's realization. It involves defining processes and systems in detail to guide coding, implementation, and testing. Decisions made during this phase impact the software's success, reliability, and maintainability. By transforming customer requirements into a clear representation, design ensures the system meets expectations. It fosters quality and provides a foundation for building robust and maintainable software. The design process typically includes a preliminary phase to conceptualize requirements and a detailed phase to refine them into actionable solutions.

## UML Diagrams:

**Actor:** An actor represents a coherent set of roles that users or external systems play when interacting with use cases. Actors can be individuals, other systems, or devices that interact with the system to achieve a goal. They are typically represented by stick figures in UML diagrams and serve as initiators or receivers of system functionalities.

**Use case:** A description of sequence of actions, including variants, that a system performs that yields an observable result of value of an actor.



UML stands for Unified Modeling Language. UML is a language for specifying, visualizing and documenting the system. This is the step while developing any product after analysis. The goal from this is to produce a model of the entities involved in the project which later need to be built. The representation of the entities that are to be used in the product being developed need to be designed.

## USECASE DIAGRAM:

Figure 1 illustrates a Use case is a description of a set of sequences of actions. Graphically, it is rendered as an ellipse with a solid line, including only its name. Use case diagrams are behavioral diagrams that show a set of use cases, actors, and their relationships. They visually represent the interactions between the system and external entities. It is an association between the use cases and actors, where each actor represents a real-world object interacting with the system. These diagrams help in understanding system requirements and user interactions more effectively. Primary Actor – Sender, Secondary Actor – Receiver.

## Use Case Diagrams:

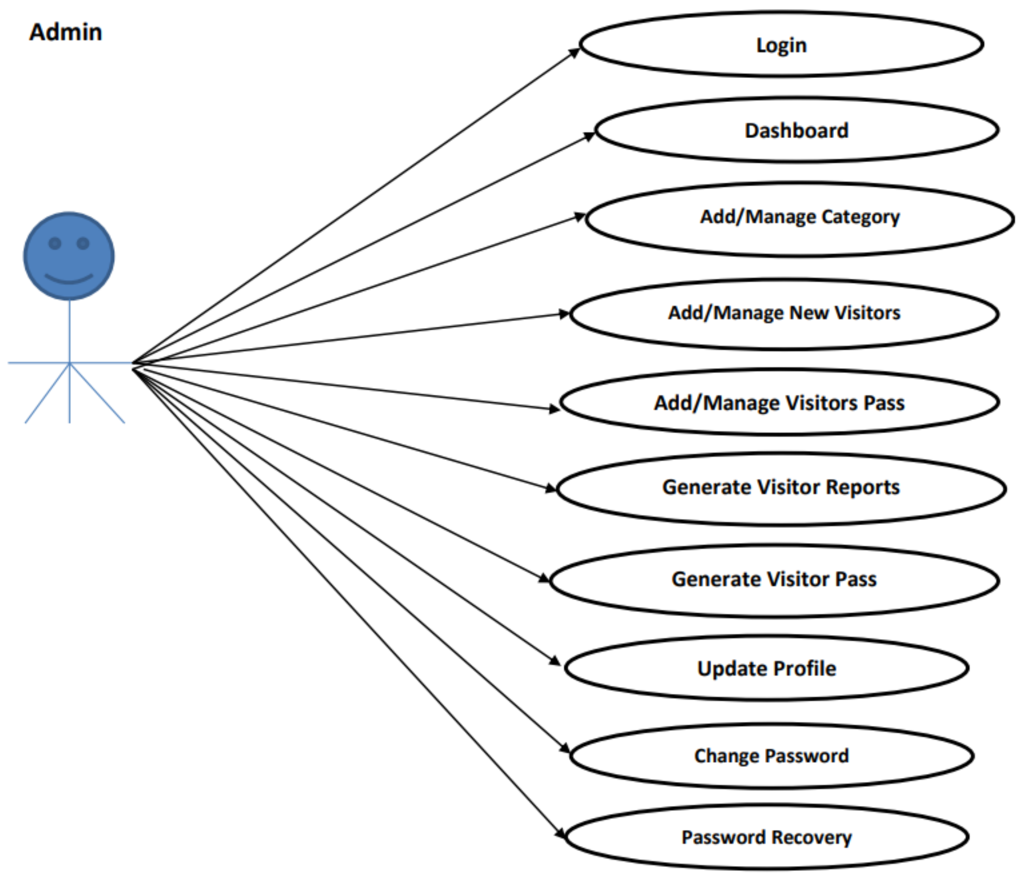


Fig 1:Use Case Diagram

## Class Diagram:

Figure 2 illustrates a class diagram is a description of a set of objects that share the same attributes, operations, relationships, and semantics. It visually represents the static structure of a system, showing how classes are related to each other. Each class typically contains attributes (data members) and operations (methods) that define its behavior. Relationships between classes, such as association, inheritance, and dependency, are depicted using specific notations. Class diagrams are essential for understanding and designing the blueprint of a system's structure and serve as a foundation for implementation.

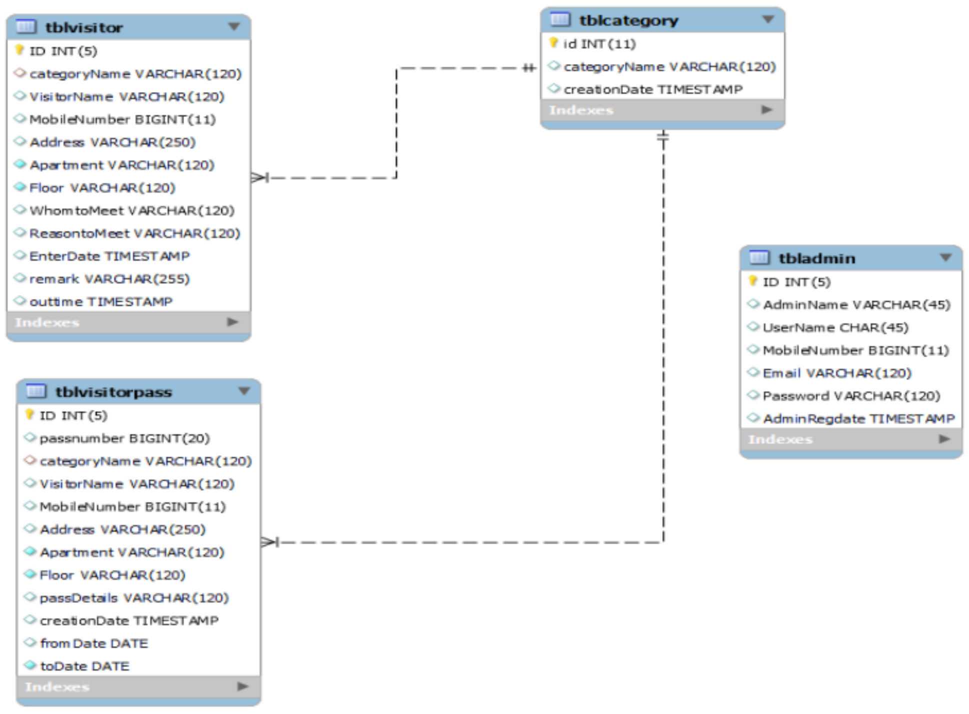


Fig 2:Class Diagram

## 2.1 ER Diagram :

An Entity-Relationship (ER) Diagram is a graphical representation of the data and relationships within a system. It serves as a blueprint for designing a database, visually depicting entities (such as objects or concepts) and their attributes, along with the relationships that connect them. Entities are represented as rectangles, attributes as ovals, and relationships as diamonds, often with lines connecting them to show associations. ER diagrams help in organizing data requirements, identifying primary and foreign keys, and ensuring the logical structure aligns with the system's objectives. They are an essential tool in database design, aiding developers in understanding and implementing a system that effectively models real-world scenarios.

## 2.1.1 ER Notation :

Figure 3 illustrates all notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. Each style uses a special set of symbols to represent the cardinality of a connection. The notation used in this document is from Martin. The symbols used for the basic ER constructs are:

• Entities are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.

• Relationships are represented by a solid line connecting two entities. The name of the relationship is written above the line. Relationship names should be verbs

• Attributes, when included, are listed inside the entity rectangle. Attributes which are identifiers a underlined. Attribute names should be singular nouns.

• Cardinality of many is represented by a line ending in a crow's foot. If the crow's foot is omitted, the cardinality is one.

**Existence** is represented by placing a circle or a perpendicular bar on the line. Mandatory existence is shown by the bar (looks like a 1) next to the entity for an instance is required. Optional existence is shown by placing a circle next to the entity that is optional.

Additionally, weak entities are represented using a double rectangle, and their relationships are depicted using a double diamond. Primary keys are highlighted by underlining attribute names, while foreign keys are often represented in italics or marked distinctly.

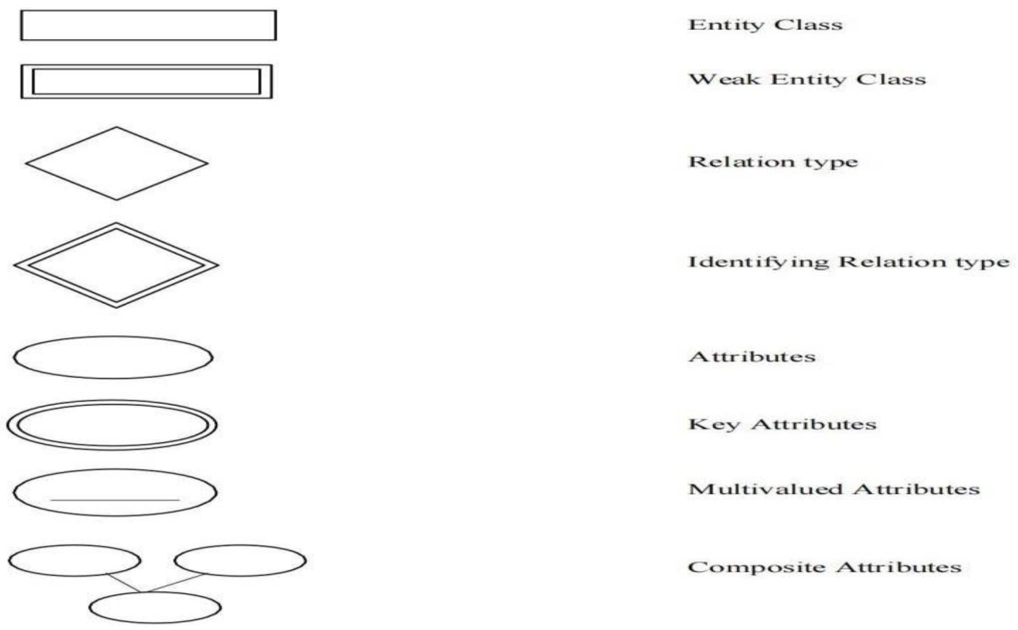


Fig 3:ER Notation

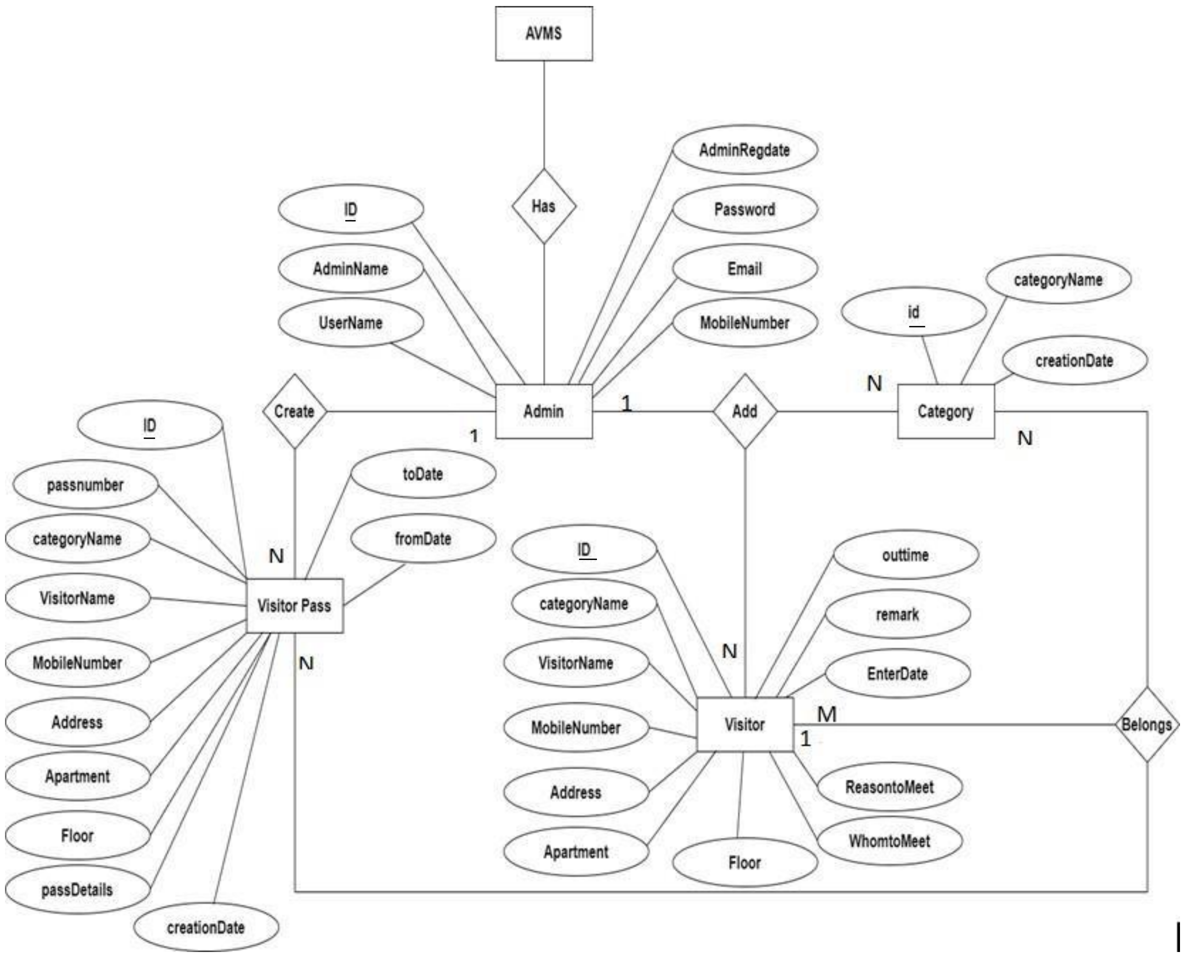


Fig 4:  ER Diagram for Apartment Visitors Management System

## 2.2 RELATIONAL SCHEMA

### Step 1:Mapping Regular Entities

For each regular entity, we create a table with its attributes. The primary key is explicitly defined.

**Admin Table**

| <u>ID</u> | Aname | Uname | Mno | Email | Password | Adrdate |
|---|---|---|---|---|---|---|

**Category Table**

| <u>ID</u> | Cname | Cdate |
|---|---|---|

**Visitor Pass  Table**

| <u>ID</u> | Pnum | Cname | Vname | Mno | Address | Apt | Wmeet | Pdtl | Cdate | Todate | FromDate |
|---|---|---|---|---|---|---|---|---|---|---|---|

**Visitor Table**

| <u>ID</u> | Cname | Vname | Mno | Address | Apt | Floor | Rmeet | Wmeet | Edate | Rmk | Outime |
|---|---|---|---|---|---|---|---|---|---|---|---|

## Step 2:Mapping of Weak Entity Types

The ERD of our project does not contain any Weak Entity.

## Step 3: Mapping of Binary 1:1 Relationships

The ERD of our project does not contain any 1:1 Relationships.

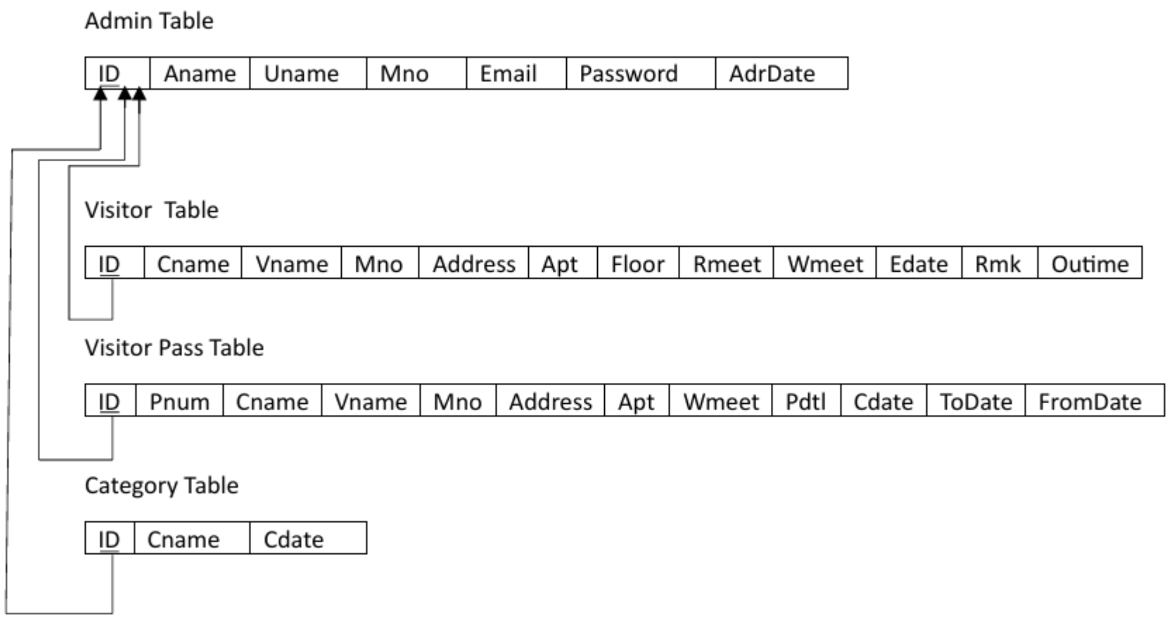
## Step 4:Mapping of Binary 1:N Relationships

1: N Refers to one-to-many relationship (1 object on one side is related to many on the other) In our ERD we have  1: N relation types.

The **1:N** relationship between Admin and VisitorPass is mapped by placing the Admin ID as a foreign key in VisitorPass.

The **1:N** relationship between Admin and Category is mapped by placing Admin ID in the Category table.

The **1:N** relationship between Visitor and VisitorPass is mapped by placing Admin ID in the Visitor table.



Admin Table

| ID | Aname | Uname | Mno | Email | Password | AdrDate |
|---|---|---|---|---|---|---|

Visitor  Table

| ID | Cname | Vname | Mno | Address | Apt | Floor | Rmeet | Wmeet | Edate | Rmk | Outime |
|---|---|---|---|---|---|---|---|---|---|---|---|

Visitor Pass Table

| ID | Pnum | Cname | Vname | Mno | Address | Apt | Wmeet | Pdtl | Cdate | ToDate | FromDate |
|---|---|---|---|---|---|---|---|---|---|---|---|

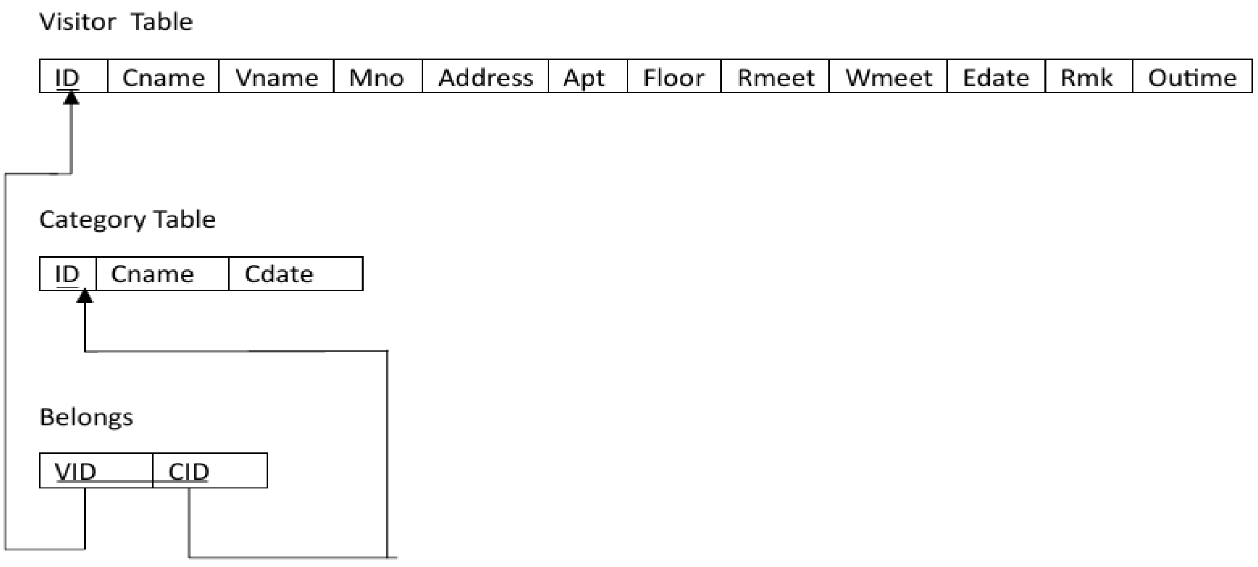
Category Table

| ID | Cname | Cdate |
|---|---|---|

## Step 5:Mapping of Binary M:N Relations

M: N refers to many-to-many relationship (many objects on one side is related to many on the other).

The **M:N relationship** between Visitor and Category is represented using a junction table (Belongs)**.**



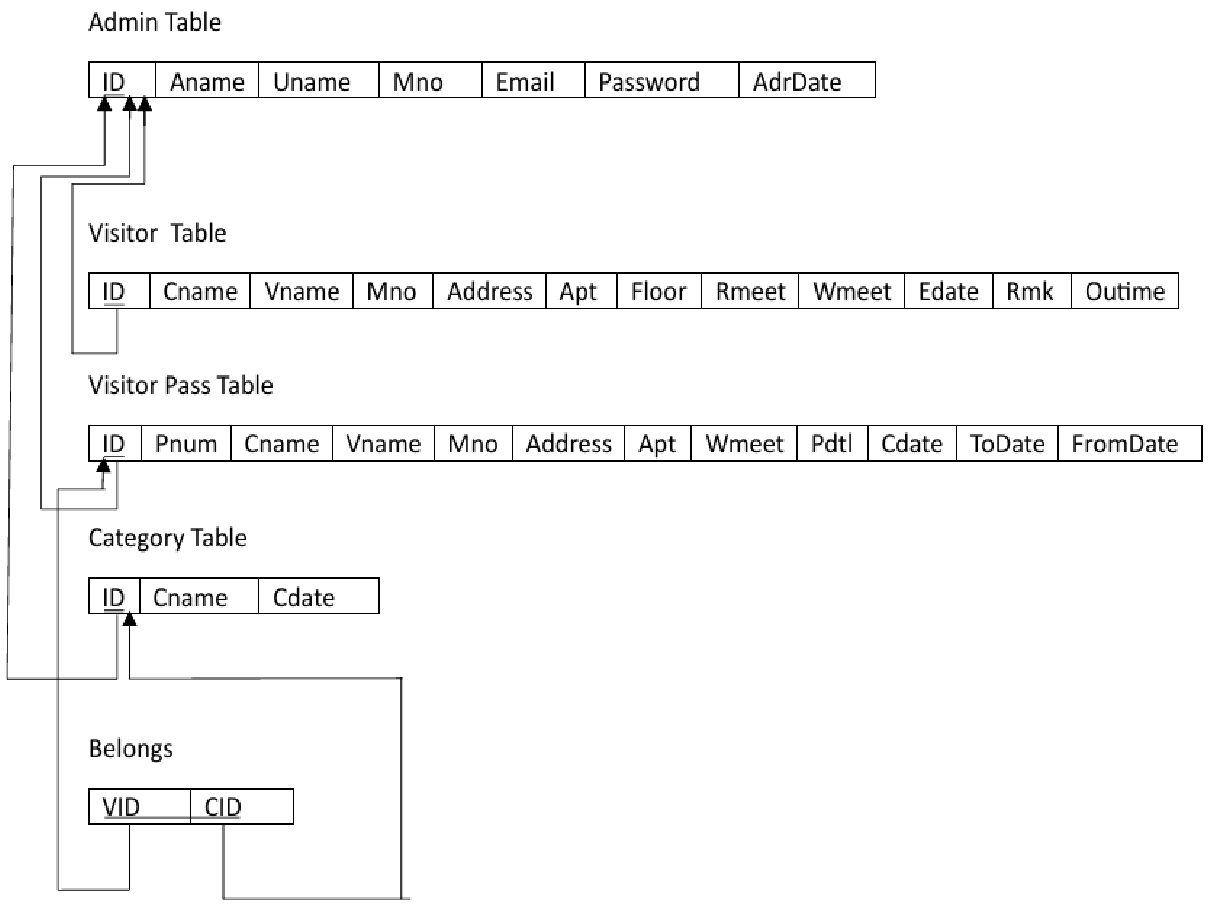
## Step 6:Mapping of Multivalued Attributes

The ERD of our project does not contain any Multivalued  Attributes.

## Step 7:Mapping of N-ary Relationships Types

The ERD of our project does not contain any N-Array relation types.

## 2.3 Schema Diagarm



Admin Table
ID
Aname
Uname
Mno
Email
Password
AdrDate
Visitor Table
ID
Cname
Vname
Mno
Address
Apt
Floor
Rmeet
Wmeet
Edate
Rmk
Outime
Visitor Pass Table
ID
Pnum
Cname
Vname
Mno
Address
Apt
Wmeet
Pdtl
Cdate
ToDate
FromDate
Category Table
ID
Cname
Cdate
Belongs
VID
CID