DELHI TECHNOLOGICAL UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE ENGINEERING

CO301
Software Engineering

SUBMITTED TO:

Lavindra Gautam

SUBMITTED BY:

Ayush Tandon
2K22/CO/133

INDEX

S.No.	Experiment	Date	Signature
1.	Overview of Software Engineering		
2.	Write the SRS (Software Requirement Specification) for Cab Booking System.		
3.	Draw data flow diagram, ER diagram, structure chart for given problem statement.		
4.	To classify the requirement into functional and non-functional requirements.		
5.	To perform the Requirement analysis of the specified problem and draw a flowchart.		
6.	To identify various elicitation techniques and their usage for the problem.		
7.	Draw use case diagram for different domains, like library system, banking systems, hospital management systems and hotel management system.		
8.	Draw use case diagram along with use case description for banking system.		
9.	Draw the ER Diagram for library management system and banking system.		
10.	Estimate the Project Metrics using all COCOMO models.		

Experiment-1

Aim: - Overview of Software Engineering

Introduction

Software Engineering is the application of a systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software. In today's technology-driven world, Software Engineering ensures that large-scale, complex systems are developed effectively and meet user needs.

---

a) Principles of Software Engineering

The principles of Software Engineering guide software developers to produce high-quality software. These principles aim to ensure reliability, maintainability, and efficient management of software projects.

1. **Modularity:**
    - Breaking down a large software system into smaller, manageable components (modules) that can be developed and tested independently.
2. **Abstraction:**
    - Hiding unnecessary details to focus on higher-level aspects of the software, promoting clarity and simplicity in design.
3. **Encapsulation:**
    - Encapsulating data and functions together helps in protecting data from unauthorized access and manipulation, ensuring the integrity of the system.
4. **Separation of Concerns:**
    - Dividing a software problem into distinct sections that address specific concerns, enabling easier debugging and maintenance.
5. **Maintainability:**
    - Designing software in a way that makes it easy to update and modify without extensive rework. This principle is key to long-term success in software development.
6. **Scalability:**
    - Ensuring that software can efficiently handle growing amounts of data, traffic, and user interactions without degrading performance.

7. **Portability:**
    - Designing software so that it can easily be transferred across different platforms, enhancing its usability in diverse environments.
8. **Continuous Integration and Testing:**
    - Integrating code changes frequently and testing them rigorously ensures early detection of issues and maintains the integrity of the codebase.
9. **User-Centered Design:**
    - Involving users in the design and development process helps create software that meets their needs, improving user satisfaction.

---

b) Components of Software Engineering

1. **Software Development Life Cycle (SDLC):**
    - **Requirement Analysis:** Understanding and documenting what the user needs.
    - **System Design:** Structuring the system architecture.
    - **Implementation:** Writing code and converting designs into executable software.
    - **Testing:** Verifying that the software functions as expected.
    - **Deployment:** Releasing the software for use.
    - **Maintenance:** Updating and improving the software post-deployment.
2. **Project Management:**
    - Planning, scheduling, and managing resources to ensure that software development is completed within budget and time constraints.
3. **Software Testing:**
    - Testing techniques ensure that the software is free of defects and performs according to the specifications. These include unit testing, integration testing, system testing, and acceptance testing.
4. **Software Quality Assurance (SQA):**
    - Monitoring the processes and methods used during software development to ensure the quality of the final product.

5. **Configuration Management:**
    - Managing changes in the software to maintain consistency across the system, ensuring stability throughout the development process.
6. **Documentation:**
    - Providing clear and comprehensive documentation for developers, testers, and users, which is essential for understanding the system and maintaining it over time.
7. **Software Metrics:**
    - Quantitative measures used to assess various aspects of software development such as code complexity, performance, and defect density.

---

c) Applications of Software Engineering

Software Engineering is used in various domains, including but not limited to:

1. **Business and Enterprise Systems:**
    - ERP (Enterprise Resource Planning) systems such as SAP, financial systems, CRM (Customer Relationship Management), and supply chain management applications.
2. **Web and Mobile Applications:**
    - Development of web platforms, social media applications, e-commerce systems, and mobile apps for Android and iOS.
3. **Embedded Systems:**
    - Used in devices like smart TVs, medical devices, automotive systems (such as self-driving cars), and household appliances.
4. **Artificial Intelligence and Machine Learning:**
    - Building AI applications, such as recommendation systems, voice recognition software, and predictive analysis platforms.
5. **Cloud Computing:**
    - Developing cloud infrastructure and services that allow scalable storage, computing, and data processing capabilities, such as AWS, Google Cloud, and Microsoft Azure.

6. **Gaming Industry:**
    - Designing and developing computer games, including their AI components, graphics rendering, and multiplayer systems.
7. **Cybersecurity Systems:**
    - Developing software to protect systems against cyber threats such as malware, phishing, and ransomware attacks.
8. **Health Informatics:**
    - Software engineering applications in the healthcare sector for managing patient records, medical imaging systems, and telemedicine platforms.

Conclusion

Software Engineering is critical to developing reliable, scalable, and maintainable systems. It integrates principles, methodologies, and tools to guide engineers throughout the development process. As technology advances, the role of Software Engineering becomes even more pivotal in ensuring that software is robust, efficient, and able to adapt to evolving requirements.

Experiment –2

Aim: - Write the SRS (Software Requirement Specification) for the Cab Booking System.

Problem Statement:

Software Requirement Specification (SRS) is a complete specification and description of requirements of the software that need to be fulfilled for the successful development of the software system. These requirements can be functional as well as non-functional depending upon the type of requirement.

The Cab Booking System is designed to provide a convenient and reliable transportation service for users. This system allows customers to book cabs for their transportation needs, track the cab's location in real-time, and manage their bookings. The system also includes features for cab drivers to receive and accept ride requests, navigate to pickup locations, and provide updates on the ride status.

The project provides the following facilities to the **customers:**

- It allows customers to book cabs on-demand or schedule bookings in advance.
- It provides real-time tracking of the cab's location and estimated time of arrival.
- It offers multiple payment options, including cash, digital wallets, and credit/debit cards.
- It allows customers to manage their booking history and preferences.

The project provides the following facilities to the **cab drivers:**

- It allows drivers to receive and accept ride requests.
- It provides navigation assistance to the pickup and drop-off locations.
- It enables drivers to update the ride status and provide feedback.

The project provides the following facilities to the **administrators:**

- It allows the management of cab fleet, including vehicle information and driver profiles.
- It provides operational insights and reports to optimize the cab service.
- It facilitates the management of customer accounts and transactions.

Experiment – 3

Aim: - Draw data flow diagram, ER diagram, structure chart for given problem statement.

Theory:

Data Flow Diagram (DFD)

A Data Flow Diagram (DFD) visually represents the flow of data within a system. It illustrates the processes, data stores, external entities, and data flows, emphasizing how data moves through the system. DFDs are used in system design to understand data sources and destinations, as well as to identify possible inefficiencies. They are typically divided into levels, starting with a high-level context diagram and moving into more detailed layers, showcasing specific system components.

Entity-Relationship Diagram (ER Diagram)

An ER Diagram visually represents entities, attributes, and relationships within a database. It defines entities as objects (like customers or orders) and establishes connections between them. ER Diagrams use symbols like rectangles for entities, ovals for attributes, and diamonds for relationships, providing a clear overview of database structure. They are valuable in database design, helping in normalization and defining primary and foreign keys, ultimately contributing to a well-organized and efficient database.

Data Dictionary

A data dictionary is a centralized repository of information about data within a system. It defines data elements, their types, descriptions, relationships, and usage, serving as a reference for developers, analysts, and users. The dictionary enhances consistency and helps avoid redundancy by clarifying data definitions across the project. It includes details such as field names, data types, lengths, and constraints, ensuring all stakeholders have a clear understanding of data structures.

Structure Chart

A structure chart breaks down a system's design into hierarchical modules, showing their relationships and functions. It depicts the system's architecture, highlighting module dependencies and flow of control. Structure charts are especially useful in top-down system design, helping developers understand module organization and data transfer among modules. This clarity aids in modular programming, making it easier to maintain, test, and debug parts of a system independently while ensuring all elements work cohesively.

Level 0 (Context Diagram)

Data Flow Diagram (DFD) of Cab Booking System.

Ayush Tandon
2K22/CO/133

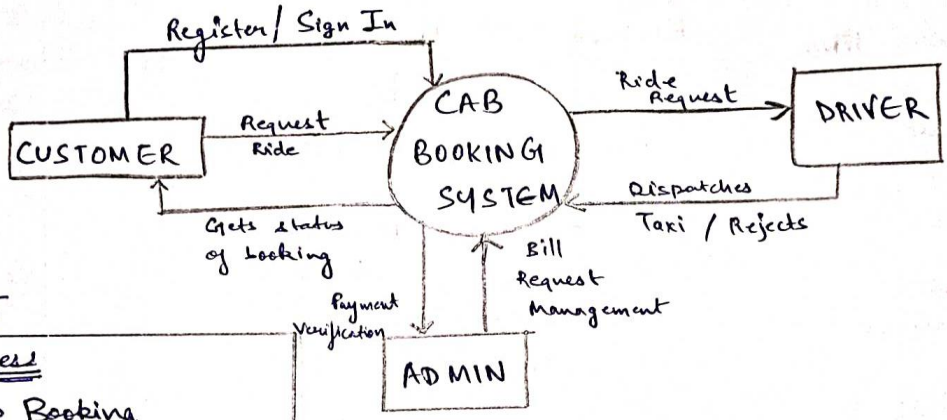
- Context Diagram — Level 0

*Entities

Driver, Passenger || Admin

Process

Booking, Payment, Bill Generation, Location selection



Process

- Cab Booking
- Vehicle Selection
- Payment Processing
- Driver Confirmation
- Bill Generation.

Level 1 DFD

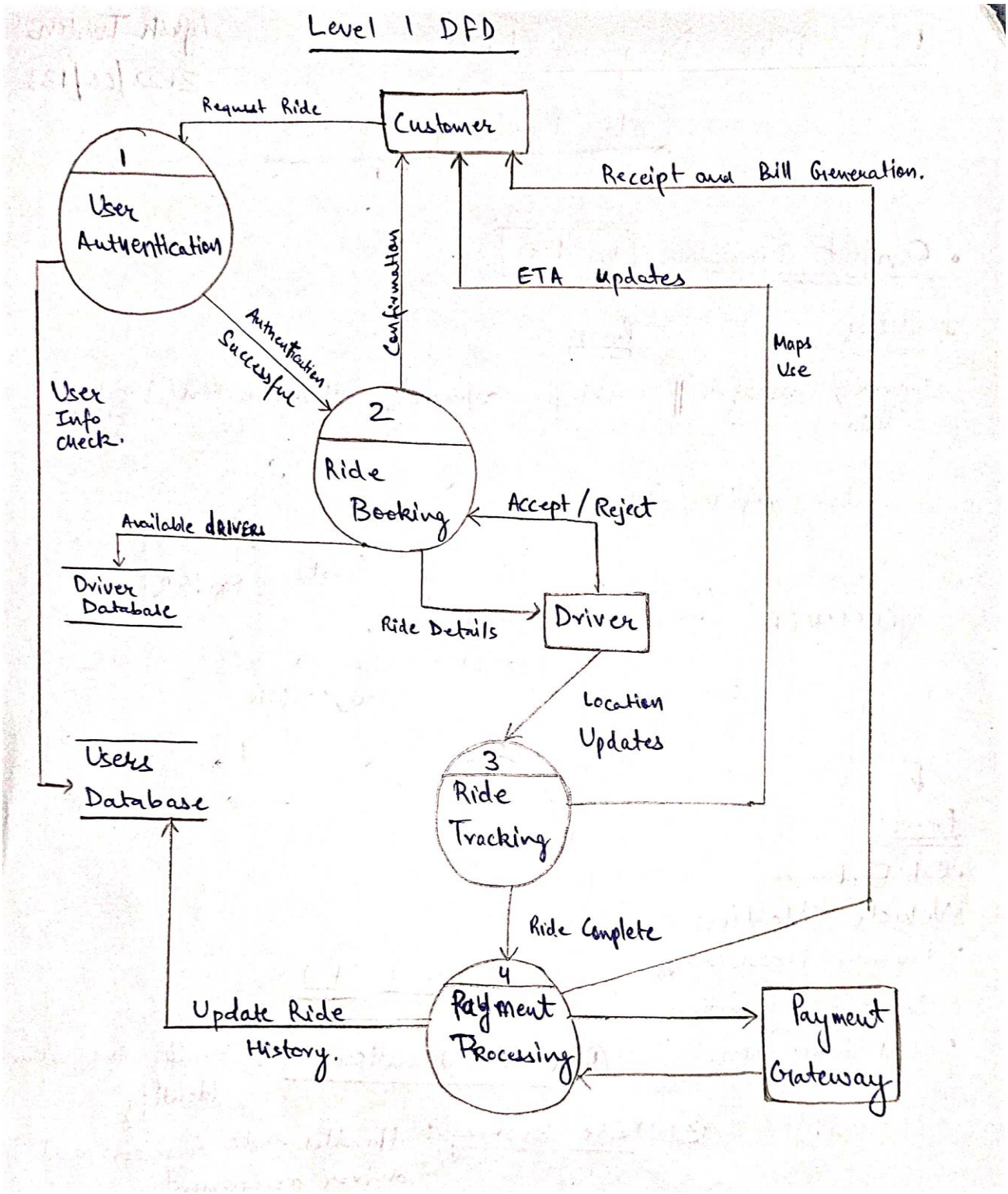
① User Authentication : Verify the customers identity

② Ride Booking : Handles ride requests and driver assignment.

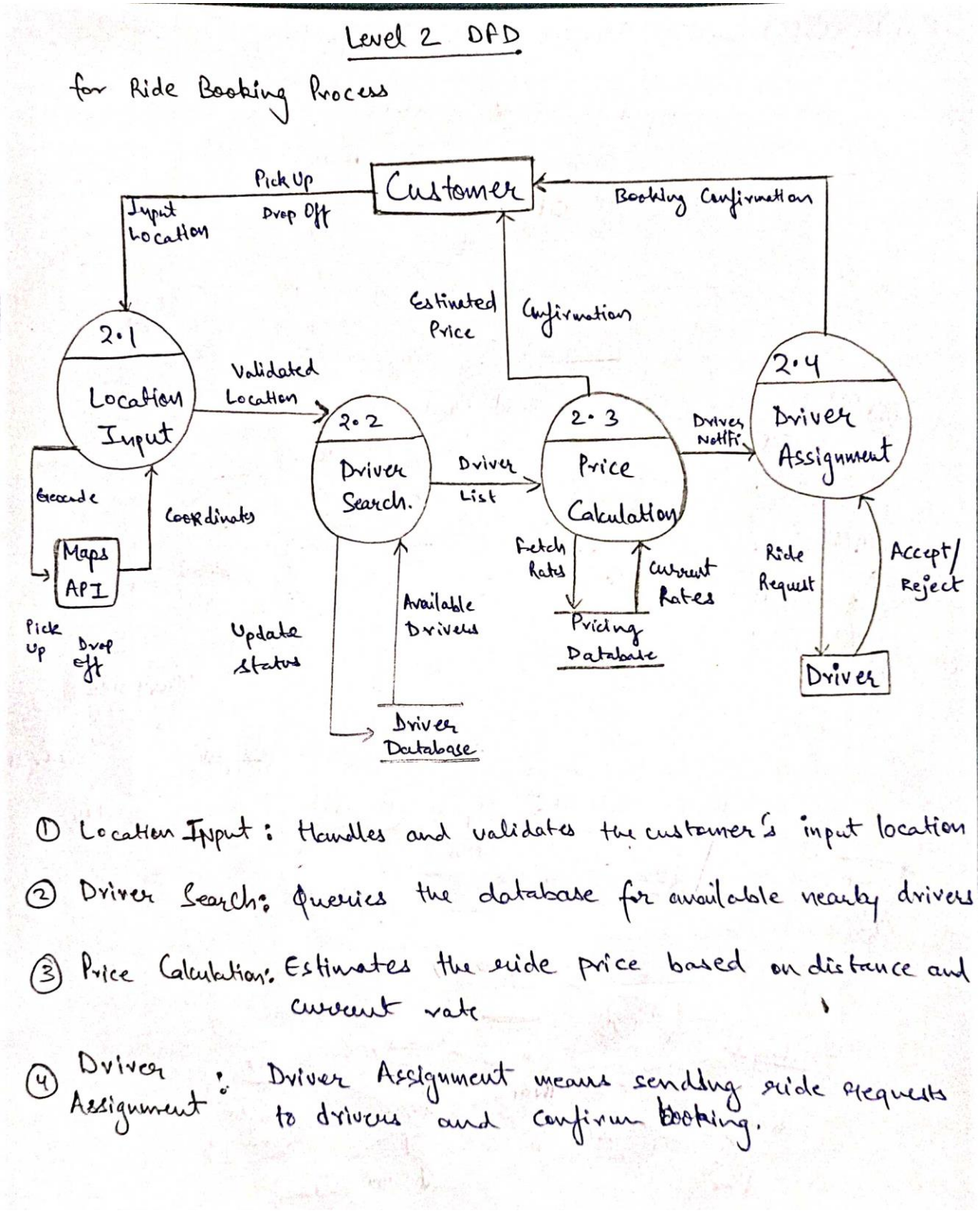
③ Ride Tracking : Monitores ride progress and provides updates

④ Payment processing : Manages payment transaction after the ride

Level 1 DFD


Level 1 DFD
Request Ride
Customer
1
User Authentication
Receipt and Bill Generation.
Confirmation
ETA updates
Maps Use
Authentication Successful
User Info check.
2
Ride Booking
Accept / Reject
Available DRIVERS
Driver Database
Ride Details
Driver
Location Updates
Users Database
3
Ride Tracking
Ride Complete
4
Payment Processing
Update Ride History.
Payment Gateway

Level 2 DFD



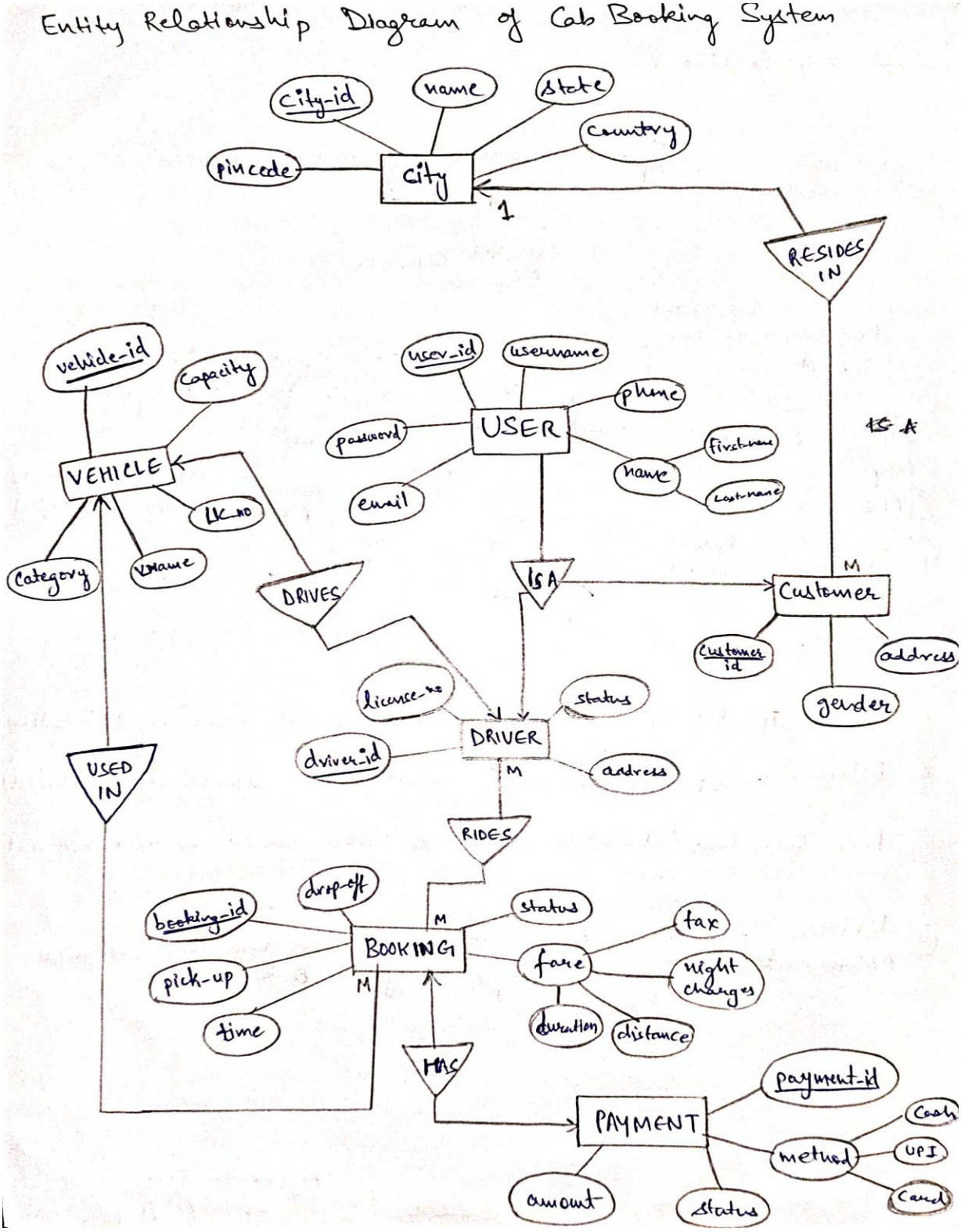
① Location Input : Handles and validates the customer's input location

② Driver Search: Queries the database for available nearby drivers

③ Price Calculation: Estimates the ride price based on distance and current rate

④ Driver Assignment : Driver Assignment means sending ride requests to drivers and confirm Booking.

ER Diagram of Cab Booking System

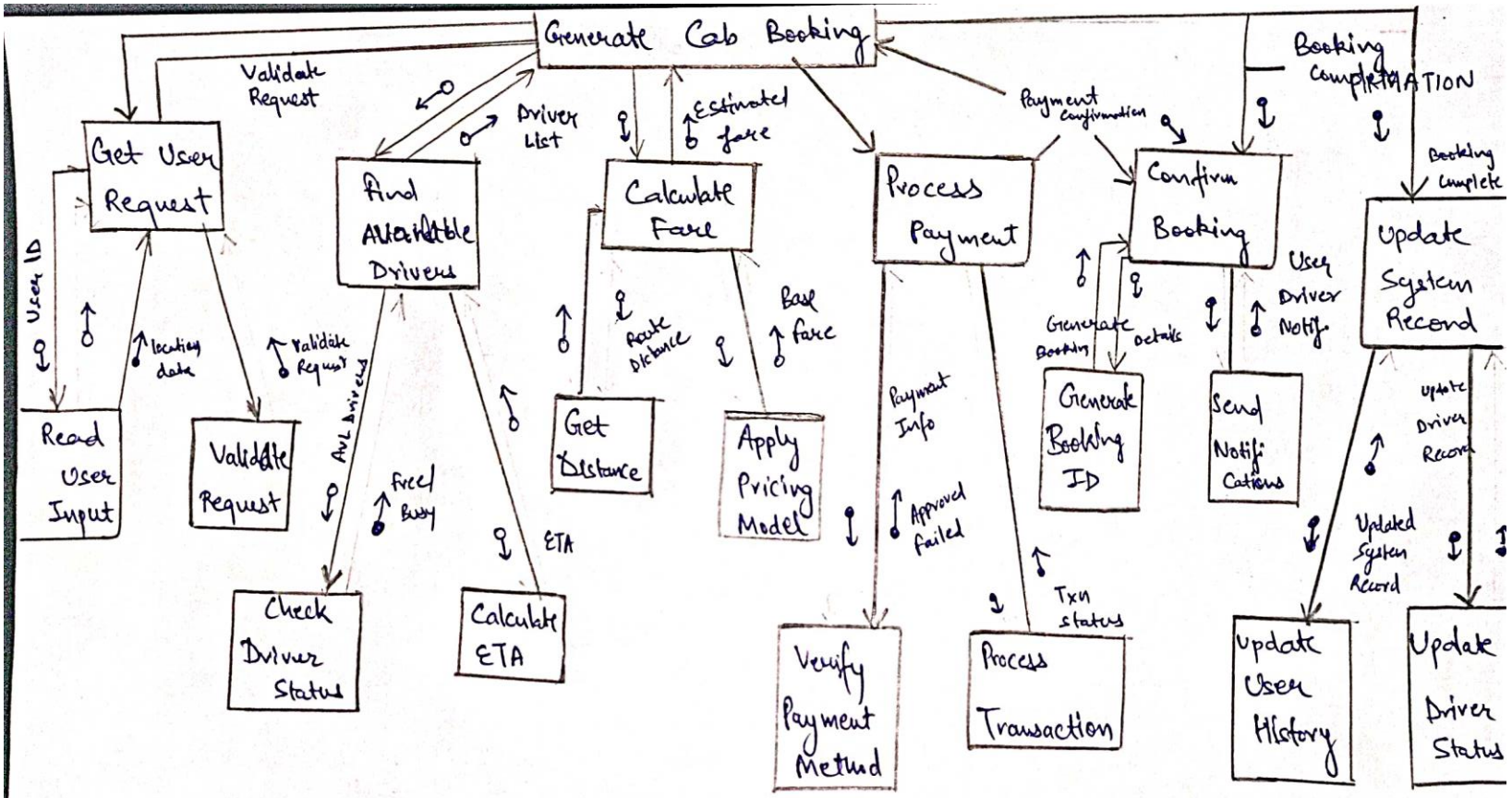


Data Dictionary

Data Dictionary for Cab Booking System

Field Name	Description	Data Type	Constraints
User_id	Unique identifier for each user	Integer	Primary Key Auto increment
Username	User name for account	varchar (50)	not null unique
password	encrypted password for user authentication	varchar (20)	Not null
Full Name	User's full name	Varchar (100)	Not null
Email	User's email address	Varchar (100)	not null
Phone Number	User's contact number	Varchar (15)	Not null
Driver ID	Unique identifier for each driver	Integer	Primary Key
Driver Name	Driver Full Name	Varchar (100)	not null
Driving License	Driver's License Number	Varchar (20)	notnull, unique
Vehicle ID	unique identifier for each vehic	Integer	PK.
Vehicle Type	Type of vehicle	Varchar (50)	Not null
License Plate	Vehicle license plate no.	Varchar (15)	not null unique
Booking ID	Unique identifier for each booking	Integer	PK
Picking Location	starting point of the ride	Varchar (255)	not null
Drop off Location	ending point of the ride	Varchar (255)	not null
Booking Time	Time when booking was made	Date Time	Not null
Fare	Cost of the ride	Decimal (10,2)	not null
Payment ID	Unique Identifier for payment	Integer	PK
Payment Method	method used for payment	Enum (Cash, Card UPI)	Not null
Payment Status	status of the payment	Enum (Pen, Suc, Fail)	Not null
City-ID	city in which booking is made	Integer (11)	PK

Structure Chart



STRUCTURE CHART FOR CAB BOOKING SYSTEM

AYUSH TANDON 2K22/CO/133

Experiment – 4

Aim: - To classify the requirement into functional and non-functional requirements.

Introduction: -

The requirements for the Cab Booking System can be classified into functional and non-functional requirements:

Functional Requirements:

● **User Registration:** The system should allow users (customers and drivers) to create accounts and manage their profiles.

● **Cab Booking:** The system should enable customers to book cabs, including on-demand and scheduled bookings, and provide options for pickup and drop-off locations.

● **Cab Tracking:** The system should provide real-time tracking of the cab's location and estimated time of arrival.

● **Payment Processing:** The system should support multiple payment options, including cash, digital wallets, and credit/debit cards.

● **Booking Management**: The system should allow customers to view their booking history, cancel or reschedule bookings, and provide feedback.

● **Driver Management:** The system should allow cab drivers to receive and accept ride requests, navigate to pickup locations, and update the ride status.

● **Admin Dashboard:** The system should provide an admin dashboard for managing the cab fleet, driver profiles, customer accounts, and operational data.

Non-Functional Requirements:

● **Usability:**

The system should have a user-friendly and intuitive interface, making it easy for customers and drivers to navigate and perform their tasks.

● **Reliability:**

The system should ensure reliable and consistent service, with minimal downtime or service interruptions.

● **Scalability:**

The system should be designed to handle a growing number of users and bookings without compromising performance.

● **Security:**

The system should maintain the privacy and security of user data, including personal information and payment details.

● **Responsiveness:**

The system should provide real-time updates and responses, ensuring a seamless user experience.

● **Availability:**

The system should be accessible 24/7, allowing users to book and track cabs at any time.

● **Accessibility:**

The system should be designed to be accessible to users with different devices and abilities.

● **Adaptability:**

The system should be flexible enough to adapt to changes in customer preferences, market trends, and regulatory requirements

Experiment – 5

Aim: - To perform the Requirement analysis of the specified problem and draw a flowchart.

Introduction: -

The requirement analysis for the Cab Booking System involves understanding the various user roles, their interactions with the system, and the key functionalities that need to be implemented.

User Roles:

● **Customer:** The primary user of the system, responsible for booking cabs, tracking their rides, and managing their bookings.

● **Driver:** The cab driver who receives and accepts ride requests, navigates to pickup locations, and updates the ride status.

● **Administrator:** The user responsible for managing the cab fleet, driver profiles, customer accounts, and operational data.

Key Functionalities:

● **User Registration and Authentication:** Customers and drivers should be able to create accounts, login, and manage their profiles.

● **Cab Booking:** Customers should be able to book cabs, including on-demand and scheduled bookings, and specify the pickup and drop-off locations.

● **Cab Tracking:** Customers should be able to track the cab's location in real-time and get updates on the estimated time of arrival.

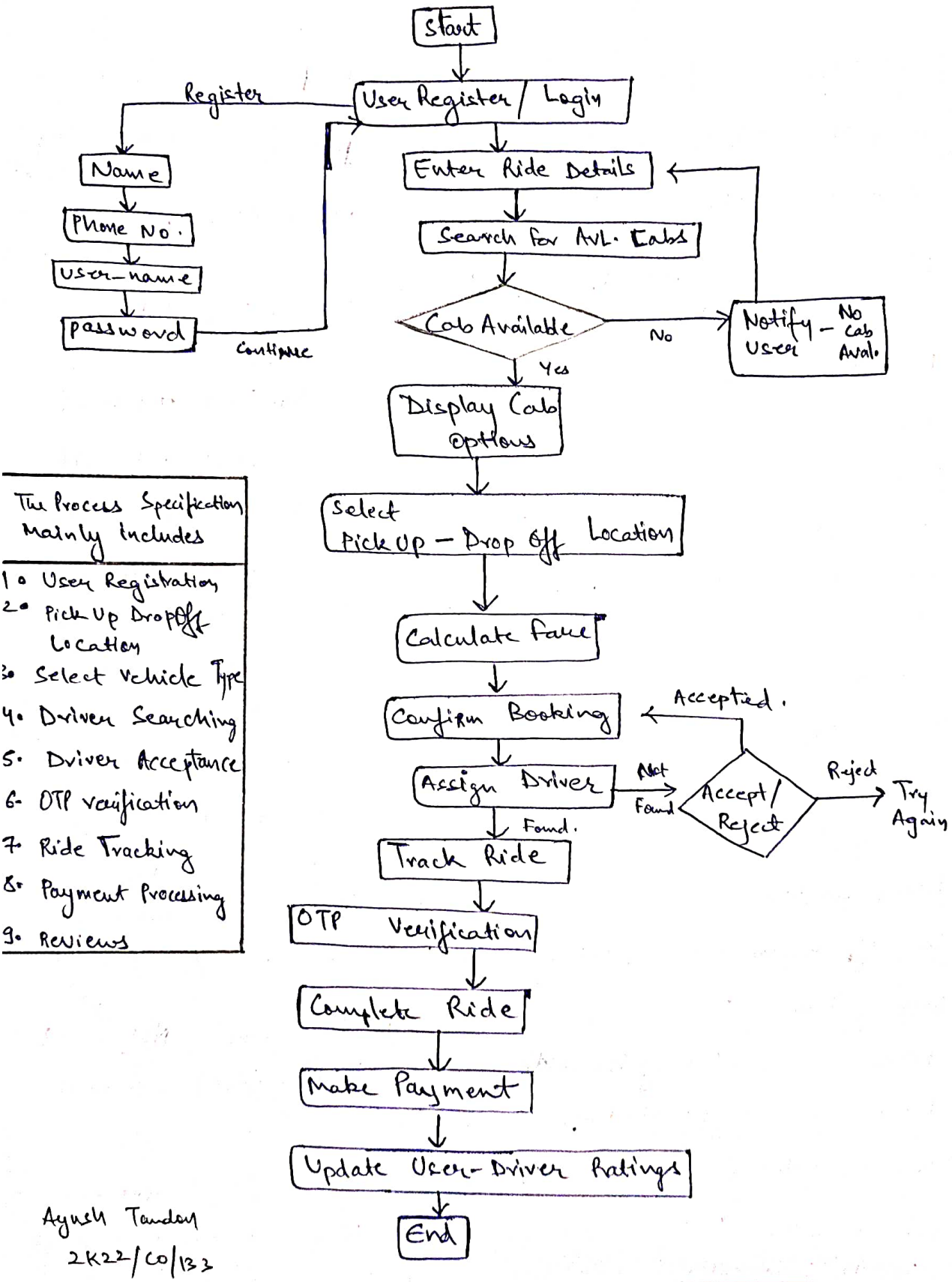
● **Payment Processing:** The system should offer multiple payment options, including cash, digital wallets, and credit/debit cards, and handle the payment transactions securely.

● **Booking Management**: Customers should be able to view their booking history, cancel or reschedule bookings, and provide feedback on the cab service.

● **Driver Management:** Drivers should be able to receive and accept ride requests, navigate to pickup locations, and update the ride status.

● **Admin Dashboard:** The administrators should have access to a centralized dashboard to manage the cab fleet, driver profiles, customer accounts, and operational data.

Process Specification Diagram for Cab Booking System



The Process Specification Mainly Includes

1. User Registration
2. Pick Up Drop Off Location
3. Select Vehicle Type
4. Driver Searching
5. Driver Acceptance
6. OTP verification
7. Ride Tracking
8. Payment Processing
9. Reviews

Ayush Tandon
2K22/CO/133

Experiment – 6

Aim: - To identify various elicitation techniques and their usage for the problem.

Introduction: -

Eliciting requirements is a crucial step in the software development process, as it helps to understand the needs and expectations of the stakeholders.

The following elicitation techniques can be used for the Cab Booking System problem statement:

➢ Interviews:

Interviews can be conducted with both customers and cab drivers to understand their pain points, preferences, and expectations from the cab booking system. The interviews can be open-ended to gather a wide range of inputs or structured to focus on specific aspects of the system.

➢ Surveys:

Online surveys can be used to gather feedback from a larger pool of customers and drivers. The surveys can cover topics such as service quality, app usability, payment preferences, and overall satisfaction.

➢ Focus Groups:

Focus group discussions can be organized to gain in-depth insights from selected customers and drivers. These sessions can be used to explore specific features, user scenarios, and pain points in a collaborative setting.

➢ Observation:

Observing customers and drivers during their interactions with the existing cab booking services can provide valuable understanding of their behavior, pain points, and unmet needs.

➢ Prototyping:

Developing low-fidelity prototypes of the cab booking system and presenting them to customers and drivers can help elicit feedback on the user experience and desired features.

➢ User Personas:

Creating user personas based on customer and driver profiles can help to better understand their motivations, goals, and pain points, which can then inform the system requirements.

➢ Use Case Analysis:

Analyzing use cases for different user scenarios, such as booking a cab, tracking a ride, or managing driver profiles, can help identify the key functionalities and interactions required in the cab booking system.

➢ Domain Analysis:

Studying the cab booking industry, including competitors, regulations, and best practices, can provide insights into the technological and operational requirements for the system.

Experiment - 7

Aim: Draw use case diagram for different domains, like library system, banking systems, hospital management systems and hotel management system.

Introduction: Use case diagrams provide a high-level overview of the main functionalities and user interactions in each of the systems you mentioned: library, banking, hospital, and hotel management systems, helping to clarify requirements and identify key actors.

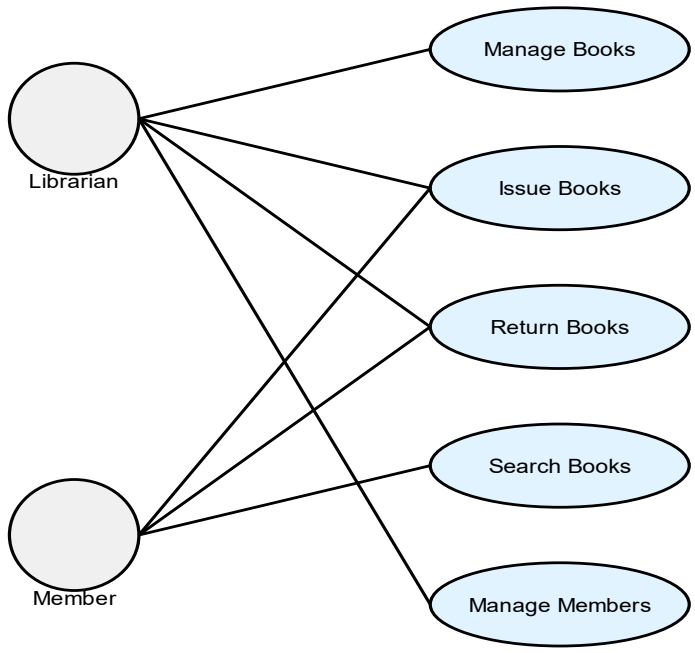
Library System Use Case Diagram

This diagram shows the main actors and use cases in a library system. The actors are:

1. Librarian
2. Member

The use cases include:

1. Manage Books: Add, remove, and update book inventory
2. Issue Books: Lend books to library members
3. Return Books: Process book returns from members
4. Search Books: Find books by title, author, or genre
5. Manage Members: Register new members and update accounts



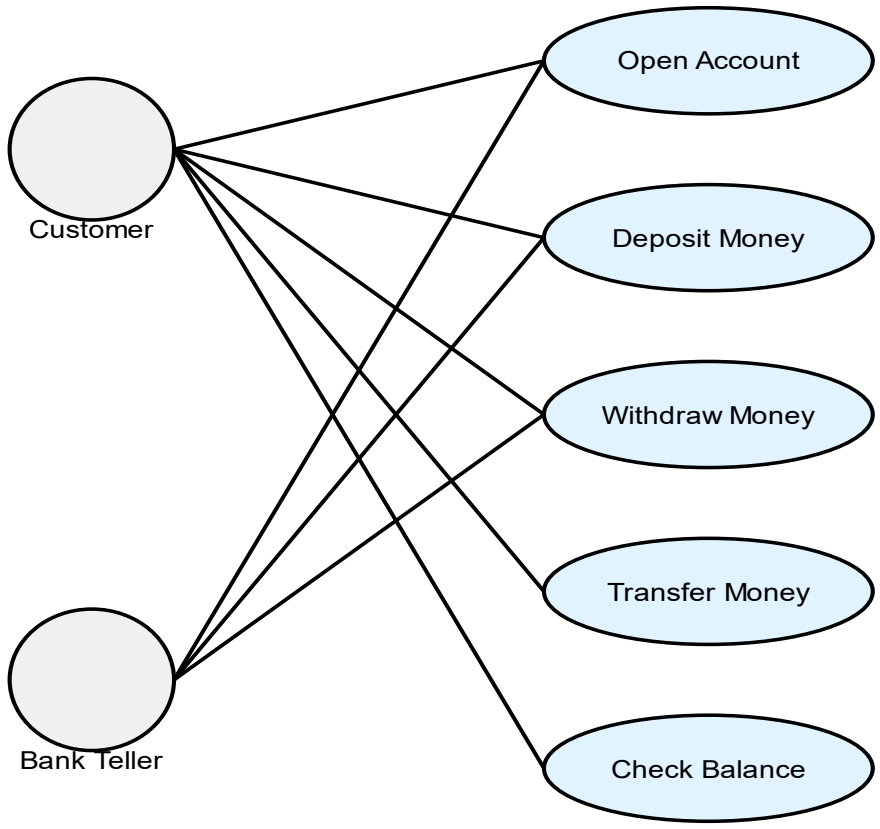
Banking System Use Case Diagram

This diagram illustrates the main actors and use cases in a banking system. The actors are:

1. Customer
2. Bank Teller

The use cases include:

1. Open Account: Create new customer accounts
2. Deposit Money: Add funds to customer accounts
3. Withdraw Money: Remove funds from customer accounts
4. Transfer Money: Move funds between accounts or banks
5. Check Balance: View current account balance



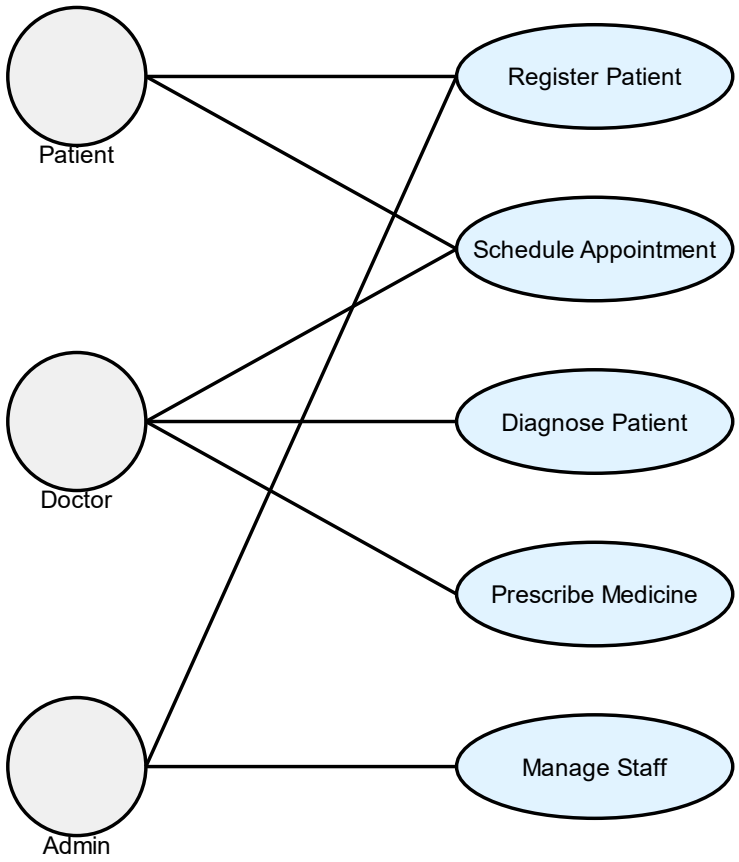
Hospital Management System Use Case Diagram

This diagram shows the main actors and use cases in a hospital management system. The actors are:

1. Patient
2. Doctor
3. Admin

The use cases include:

1. Register Patient: Add new patients to the system
2. Schedule Appointment: Book patient visits with doctors
3. Diagnose Patient: Record medical examinations and findings
4. Prescribe Medicine: Issue and track patient prescriptions
5. Manage Staff: Oversee hospital personnel and schedules



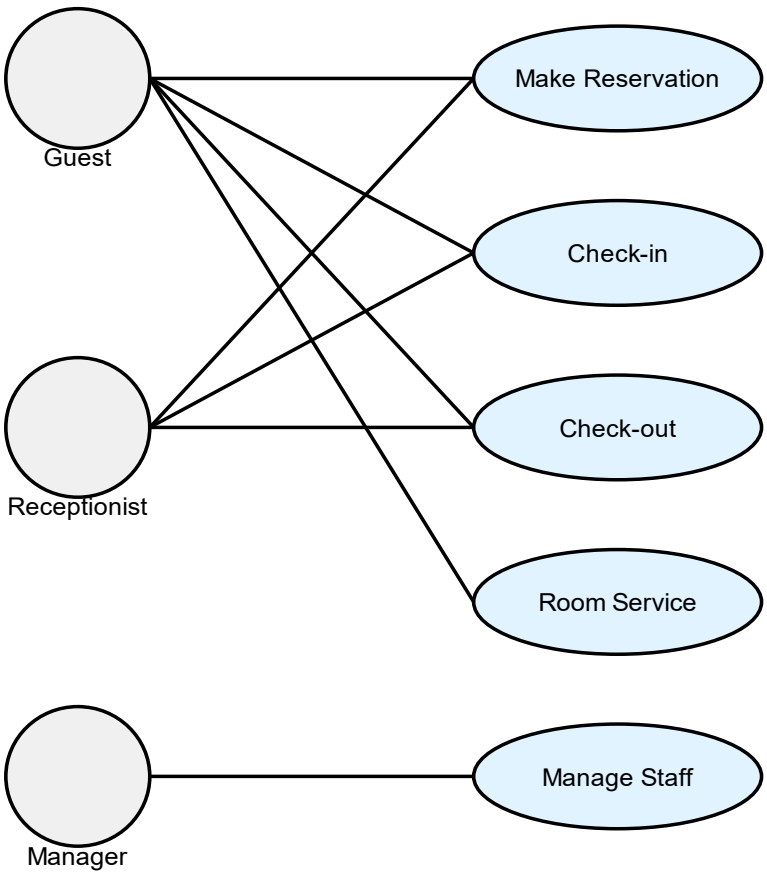
Hotel Management System Use Case Diagram

This diagram illustrates the main actors and use cases in a hotel management system. The actors are:

1. Guest
2. Receptionist
3. Manager

The use cases include:

1. Make Reservation: Book rooms for future stays
2. Check-in: Process guest arrivals and room assignments
3. Check-out: Finalize guest stays and process payments
4. Room Service: Manage in-room dining and amenities
5. Manage Staff: Oversee hotel personnel and schedules



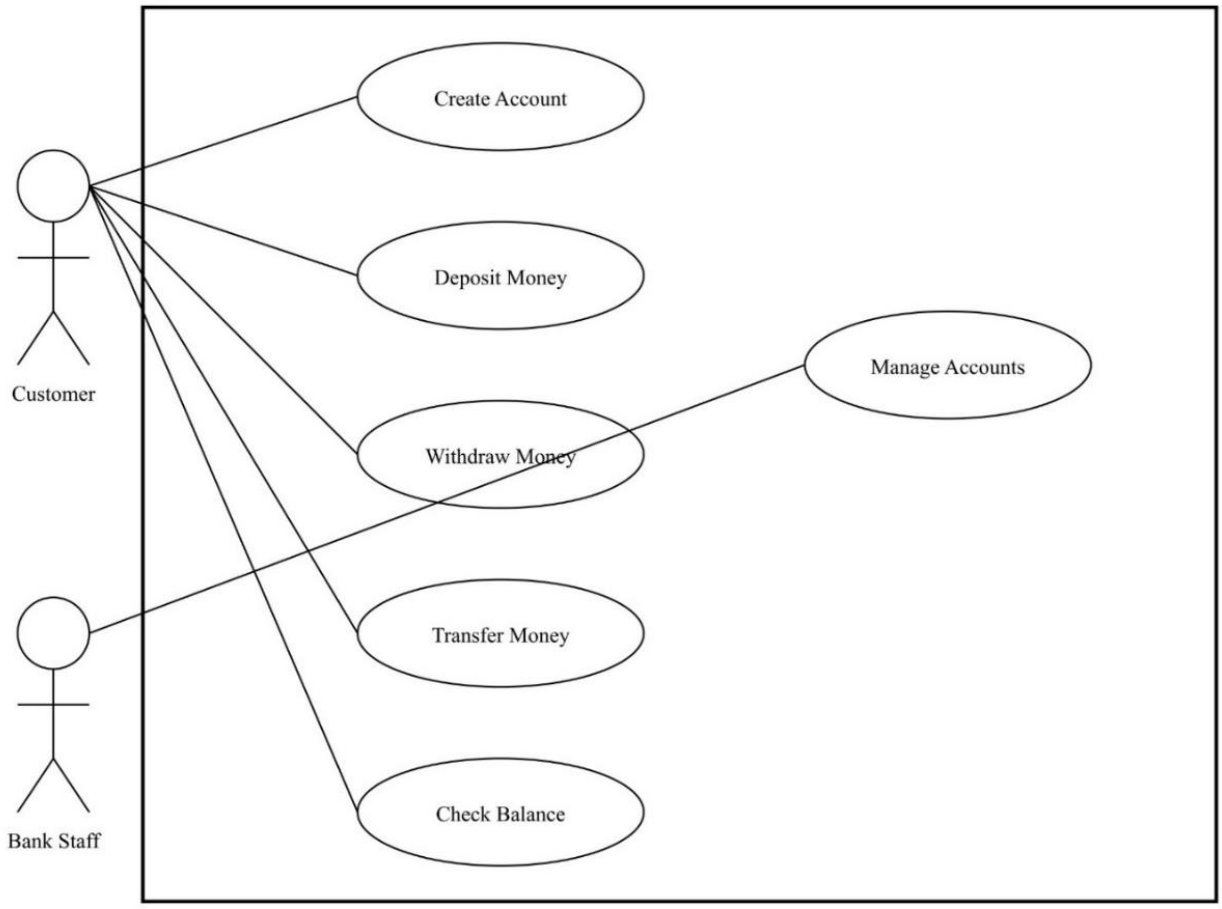
Experiment – 8

Aim: Draw use case diagram along with use case description for banking system.

Introduction:

The banking system is designed to provide secure, efficient, and convenient banking services to customers. The system enables customers to manage their accounts, perform transactions, and access various banking services online or through mobile devices. The system also provides administrative functionality for bank employees to manage customer accounts, process transactions, and generate reports.

Banking System Use Case Descriptions



1. Create Account

Actor: Customer

Description: This use case allows a customer to create a new bank account.

Preconditions: The customer is not already registered in the system.

Main Flow:

1. Customer requests to create a new account.
2. System prompts for customer information (name, address, ID).
3. Customer provides required information.
4. System validates the information.
5. System creates a new account and generates an account number.
6. System confirms account creation and provides account details to the customer.

Alternative Flow:

- If the customer information is invalid, the system notifies the customer and requests correct information. **Postconditions**: A new account is created in the system for the customer.

2. Deposit Money

Actor: Customer
Description: This use case allows a customer to deposit money into their account.
Preconditions: The customer has an existing account.

Main Flow:

1. Customer selects the deposit option.
2. System prompts for account number and deposit amount.
3. Customer provides the required information.
4. System verifies the account and processes the deposit.
5. System updates the account balance and provides a confirmation to the customer.

Alternative Flow:

- If the account number is invalid, the system notifies the customer and requests a valid account number.

Postconditions: The account balance is updated with the deposited amount.

3. Withdraw Money

Actor: Customer

Description: This use case allows a customer to withdraw money from their account.

Preconditions: The customer has an existing account with sufficient balance.

Main Flow:

1. Customer selects the withdraw option.
2. System prompts for account number and withdrawal amount.
3. Customer provides the required information.
4. System verifies the account and checks for sufficient balance.
5. System processes the withdrawal and dispenses cash.
6. System updates the account balance and provides a confirmation to the customer.

Alternative Flow:

- If the account has insufficient balance, the system notifies the customer and cancels the transaction.

Postconditions: The account balance is updated with the withdrawn amount deducted.

4. Transfer Money

Actor: Customer

Description: This use case allows a customer to transfer money from their account to another account.

Preconditions: The customer has an existing account with sufficient balance.

Main Flow:

1. Customer selects the transfer option.
2. System prompts for source account number, destination account number, and transfer amount.
3. Customer provides the required information.
4. System verifies both accounts and checks for sufficient balance in the source account.
5. System processes the transfer.
6. System updates both account balances and provides a confirmation to the customer.

Alternative Flow:

- If either account is invalid or there's insufficient balance, the system notifies the customer and cancels the transaction.

Postconditions: Both account balances are updated to reflect the transfer.

5. Check Balance

Actor: Customer

Description: This use case allows a customer to check their account balance.

Preconditions: The customer has an existing account.

Main Flow:

1. Customer selects the check balance option.
2. System prompts for account number.
3. Customer provides the account number.
4. System verifies the account and retrieves the current balance.
5. System displays the current balance to the customer.

Alternative Flow:

- If the account number is invalid, the system notifies the customer and requests a valid account number.

Postconditions: The current account balance is displayed to the customer.

6. Manage Accounts

Actor: Bank Staff

Description: This use case allows bank staff to manage customer accounts.

Preconditions: The bank staff member is authenticated in the system.

Main Flow:

1. Bank staff selects the account management option.
2. System displays a list of management functions (e.g., view accounts, modify account details, close accounts).
3. Bank staff selects a specific function.
4. System prompts for necessary information based on the selected function.
5. Bank staff provides the required information.
6. System processes the request and updates the account information as necessary.
7. System confirms the action to the bank staff.

Alternative Flow:

- If the requested action cannot be completed, the system notifies the bank staff with the reason.

Postconditions: Account information is updated based on the management action performed.

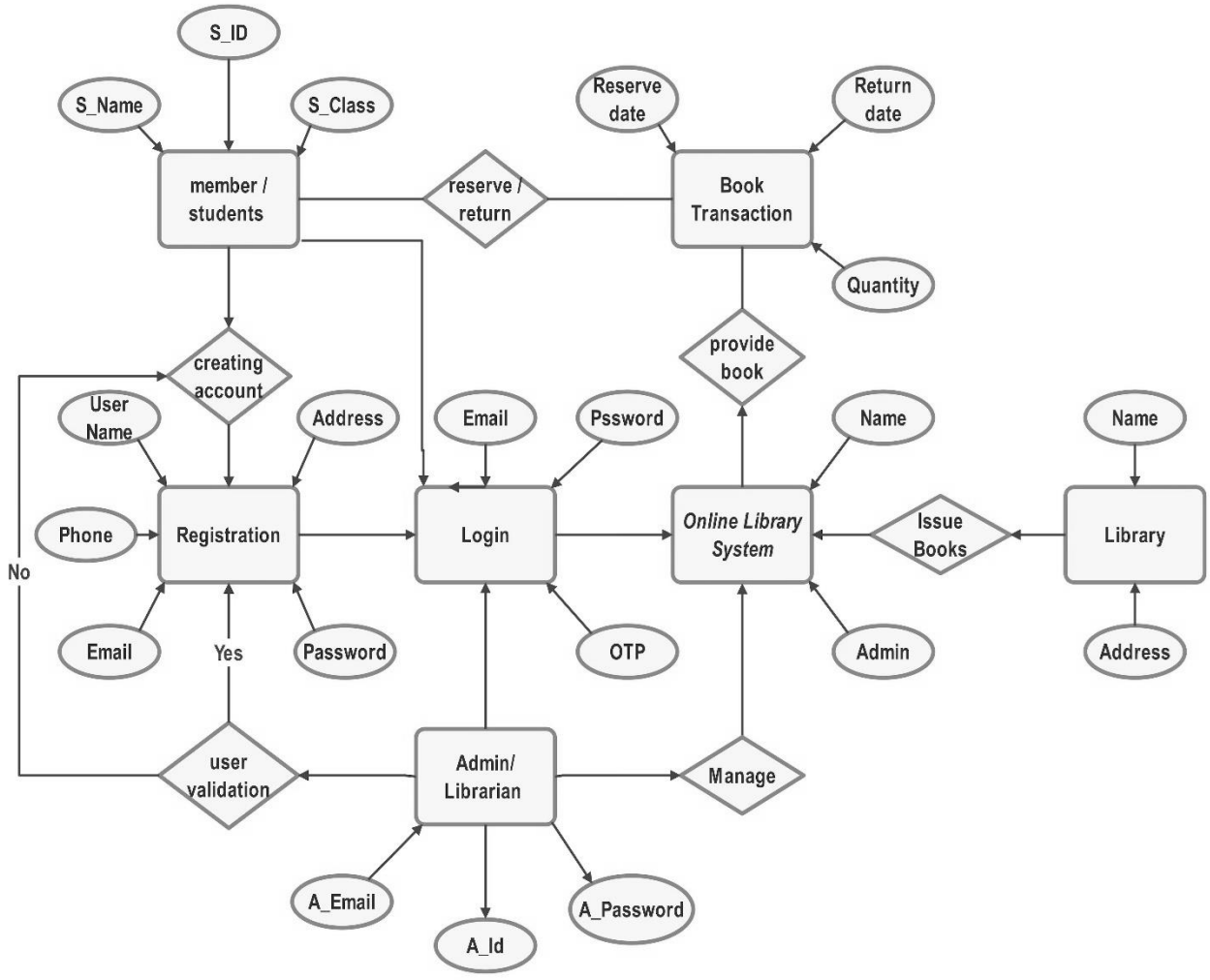
Experiment - 9

Aim: Draw the ER Diagram for library management system and banking system.

Introduction:

An ER Diagram (Entity-Relationship Diagram) represents the structure of a database, illustrating relationships between entities and their attributes. It is a tool for understanding and communicating database design.

ER diagram for Library Management System



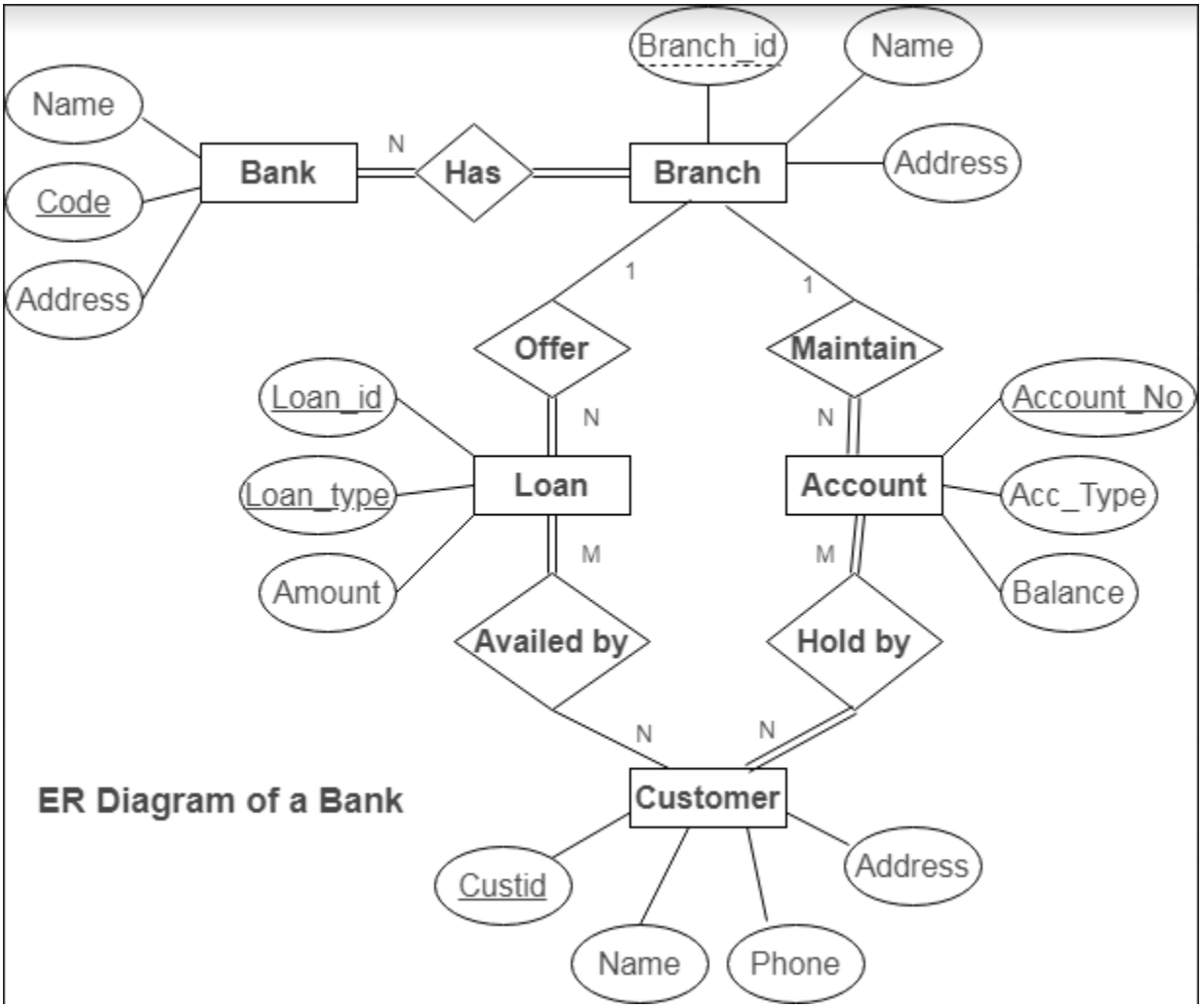
Entities and Attributes:

1. **Students**
    - **Attributes**: S_ID, S_Name, S_Class
    - **Description**: Represents the students who are members of the library.
2. **Registration**
    - **Attributes**: Username, Phone, Email, Password
    - **Description**: Used for creating a new account with information like phone number, email, and password.
3. **Login**
    - **Attributes**: Email, Password
    - **Description**: Used for authenticating users and allowing access to the library system.
4. **Admin / Librarian**
    - **Attributes**: A_Email, A_ID, A_Password
    - **Description**: Represents the administrative users who manage the library system.
5. **Book Transaction**
    - **Attributes**: Reserve date, Return date, Quantity
    - **Description**: Manages the reservation and return of books by students.
6. **Library**
    - **Attributes**: Name, Address
    - **Description**: Represents the library details including name and address.
7. **Online Library System**
    - **Attributes**: OTP
    - **Description**: The main system that integrates the functionality for book transactions, member management, and library management.

Relationships:

1. **Students - Registration**
    - **Relationship**: creating account
    - **Description**: Students need to register to create an account in the system. This includes providing their Username, Phone, Email, and Password.
2. **Registration - Login**
    - **Relationship**: user validation
    - **Description**: After registration, users can log in by validating their credentials, including email and password.
3. **Login - Online Library System**
    - **Relationship**: Login
    - **Description**: Users log into the main online library system after successful validation.
4. **Online Library System - Admin / Librarian**
    - **Relationship**: Manage
    - **Description**: Admin or Librarian manages the library system, overseeing book issues, returns, and library inventory.
5. **Book Transaction - Students**
    - **Relationship**: reserve/return
    - **Description**: Students can reserve and return books, which is tracked by the system using Reserve date, Return date, and Quantity.
6. **Book Transaction - Online Library System**
    - **Relationship**: provide book
    - **Description**: The library system manages the provision of books for transactions, handling reserves and returns.
7. **Online Library System - Library**
    - **Relationship**: Issue Books
    - **Description**: The system issues books from the library to students as managed by the library staff or system.

ER Diagram of Banking System



Entities and Attributes

1. **Bank Entity**
    - **Attributes**: Bank Name, Code, Address (main office)
    - **Primary Key**: Code
2. **Branch Entity**
    - **Attributes**: Branch ID, Branch Name, Address
    - **Primary Key**: Branch ID

3. **Customer Entity**
    - **Attributes**: Customer ID, Name, Phone Number, Address
    - **Primary Key**: Customer ID
4. **Account Entity**
    - **Attributes**: Account Number, Account Type, Balance
    - **Primary Key**: Account Number
5. **Loan Entity**
    - **Attributes**: Loan ID, Loan Type, Amount
    - **Primary Key**: Loan ID

Relationships

- **Bank to Branch** (1 : N)
    - A bank can have multiple branches, but each branch is associated with only one bank.
- **Branch to Account** (1 : N)
    - A branch can maintain multiple accounts, but each account is tied to a single branch.
- **Branch to Loan** (1 : N)
    - A branch can offer multiple loans, but each loan is managed by one branch.
- **Customer to Account** (M : N)
    - A customer can hold multiple accounts, and an account can be jointly held by multiple customers, establishing a many-to-many relationship.
- **Customer to Loan** (M : N)
    - A customer can avail multiple loans, and a loan can be jointly held by multiple customers, forming a many-to-many relationship.

Experiment – 10

Aim: - Estimate the Project Metrics using all COCOMO models for a Cab Booking System.

Introduction:

COCOMO is one of the most widely used software estimation models. COCOMO predicts the effort and schedule of a software product based on the size of the software.

In COCOMO, projects are categorized as:

1. **Organic**: Small, well-understood projects with experienced teams, like simple business or inventory systems.
2. **Semidetached**: Projects with mixed-experience teams and partial familiarity, like OS or DBMS development.
3. **Embedded**: Projects with tight hardware integration or strict regulations, like ATMs and air traffic control systems.

The necessary steps in this model are:

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors.
4. The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KLOC as the measure of the size. To determine the initial effort Ei in person-months, the equation used is:

$$E_i = a * (KLOC)^b$$

The values of the constants a and b depend on the project type.

Code:

```
#include <iostream>
#include <cmath>
using namespace std;

struct CocomomParams {
   double a, b, c, d;
};
CocomomParams cocomo_params[] = {
   { 2.4, 1.05, 2.5, 0.38 }, // Organic
   { 3.0, 1.12, 2.5, 0.35 }, // Semidetached
   { 3.6, 1.20, 2.5, 0.32 }  // Embedded
};

void cocomo_estimate(double kloc, int project_type, double& effort, double&
development_time, double& avg_staff) {
   CocomomParams params = cocomo_params[project_type];

   effort = params.a * pow(kloc, params.b);
   development_time = params.c * pow(effort, params.d);
   avg_staff = effort / development_time;
}

int main() {
   double project_size = 25.0;
   int project_type = 1;
```

```
    double effort, development_time, avg_staff;
    cocomo_estimate(project_size, project_type, effort, development_time, avg_staff);

    cout << "For a ";
    switch (project_type) {
        case 0: cout << "Organic"; break;
        case 1: cout << "Semidetached"; break;
        case 2: cout << "Embedded"; break;
    }
    cout << " project of size " << project_size << " KLOC:" << endl;
    cout << "Estimated Effort: " << effort << " person-months" << endl;
    cout << "Estimated Development Time: " << development_time << " months" << endl;
    cout << "Estimated Average Staff: " << avg_staff<< " persons" << endl;
    return 0;
}
```

Output

```
For a Semidetached project of size 25 KLOC:
Estimated Effort: 137.5 person-months
Estimated Development Time: 31.25 months
Estimated Average Staff: 4 persons
```