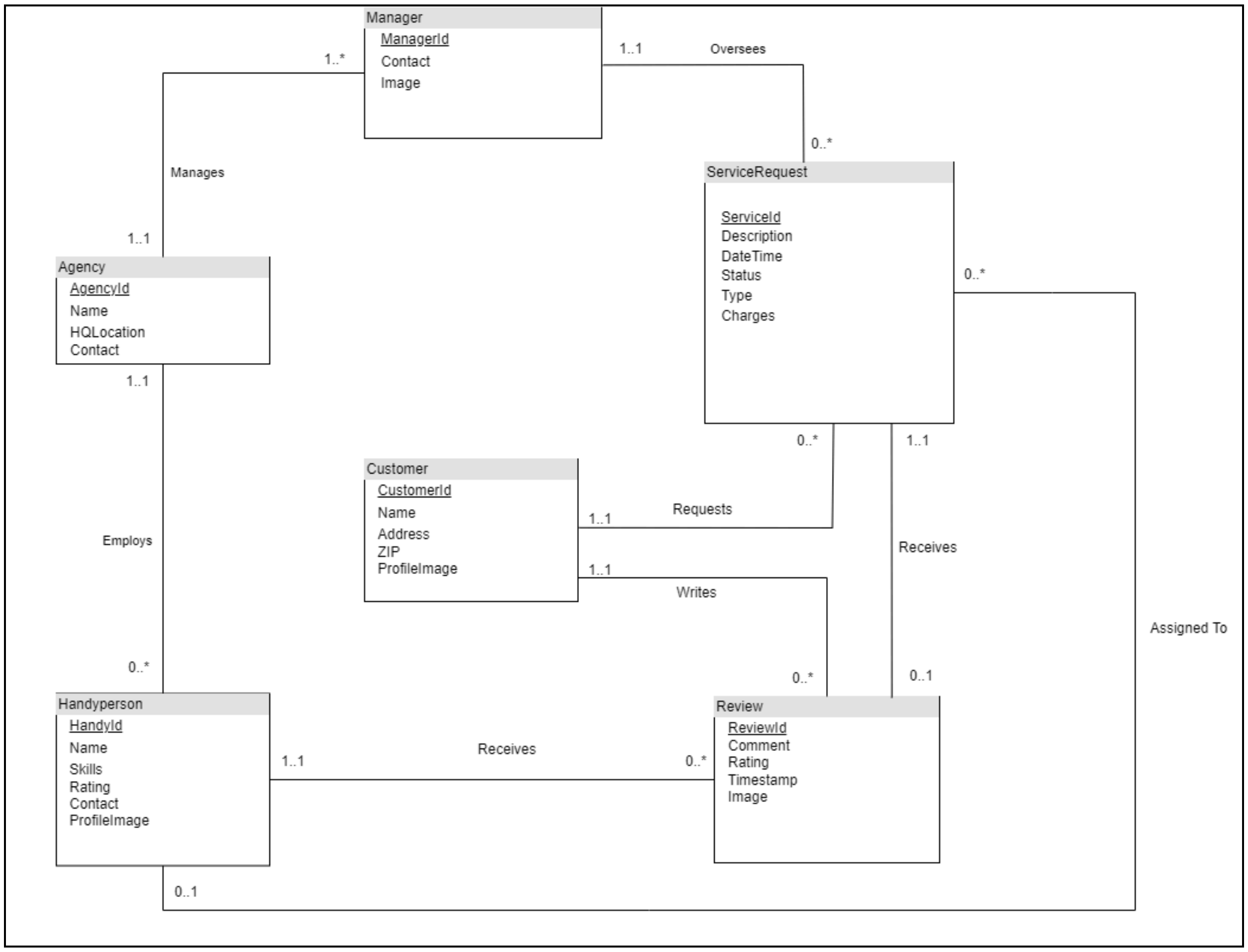
Project Track 1 Stage 2
Team 047 - Join_The_Party

Project Title: HandyIllinois - Connecting Homeowners with Reliable Handyperson Services

UML Diagram:



Explanation :

1. **Entities & Attributes**:
    a. Agency - Attributes: AgencyId, Name, HQ Location, Contact
    Represents the primary organization that manages various managers and handypersons. An agency is a company that connects customers with handypersons.
    Why it's an entity: The agency needs to manage multiple handypersons and service requests, and it has multiple attributes that are essential to the system,

which makes it a natural entity instead of just an attribute of another entity (like service requests).

b. Manager - Attributes: ManagerId, Contact, Image
   Represents an individual managing handypersons within an agency. Each manager can be associated with only one agency.
   Why it's an entity: Manager is an entity because it has a relationship with both the Agency and ServiceRequest entities. It's not just an attribute since managers can have multiple interactions and may supervise multiple services.

c. Handyperson - Attributes: HandyId, Name, Skills, Rating, Contact, ProfileImage
   Represents the skilled individuals who perform services requested by customers. A handyperson may work under a single agency. The system tracks the handyperson's skills, making it easier to match them with the right service requests. Each handyperson has a unique ID and a skill associated with them.
   Why it's an entity: Modeled as a separate entity because it has a direct relationship with ServiceRequest and Agency. A handyperson might have attributes like Skills and Rating that cannot be part of other entities.

d. Service Request - Attributes: ServiceId, ManagerId (foreign key), HandyId (foreign key), Description, DateTime, Status, Type, Charges
   This represents a request for handyperson services. A customer makes service requests, and each service request is linked to an agency. Attributes include a unique identifier, date, status, type, and charges.
   Why it's an entity: Service requests are central to the system and have multiple associated attributes, which make it necessary to track them as their own entity. These attributes are essential for managing the state and type of services offered, making it much more complex than an attribute of a customer or handyperson. It also represents a transactional interaction and serves as a link between Customer, Handyperson, and Manager. This entity would also be used to track historical data for analysis.

e. Customer - Attributes: CustomerId, Name, Address, ZIP, ProfileImage
   A customer can make multiple service requests, and they have attributes such as a unique identifier, name, address, and contact information.
   Why it's an entity: Customers are key actors in the system, and modeling them as entities helps associate them with service requests, reviews, and other potential future interactions.

f. Review - Attributes: ReviewId, Comment, Rating, Timestamp, Image
   Customers leave reviews after a service request is completed. Reviews are tied to specific service requests and handypersons.
   Why it's an entity: Reviews have a relationship with both customers and handypersons, making them more than just attributes of one or the other. They are central to evaluating service quality, so they need to be tracked independently.

2. **Relationships and Cardinality:**
   a. Agency - Manager:
      Cardinality: 1 Agency : 1..* Manager
      Assumption: An agency may have one or more managers, but each manager is associated with only one agency.
   b. Agency - Handyperson:
      Cardinality: 1 Agency : 0..* Handyperson
      Assumption: An agency may have multiple handypersons working for it, but a handyperson can only be linked to a single agency.
   c. Customer - ServiceRequest:
      Cardinality: 1 Customer : 0..* ServiceRequest
      Assumption: A customer may have multiple service requests, but each service request is associated with exactly one customer.
   d. Manager - ServiceRequest:
      Cardinality: 1 Manager : 0..* ServiceRequest
      Assumption: A manager can handle multiple service requests, but each service request is supervised by a single manager.
   e. Handyperson - ServiceRequest:
      Cardinality: 1 Handyperson : 0..* ServiceRequest
      Assumption: A handyperson can be assigned to multiple service requests, but each service request involves only one handyperson.
   f. ServiceRequest - Review:
      Cardinality: 1 ServiceRequest : 1 Review
      Assumption: A service request can have only 0 or 1 review, and each review is linked to a single service request.
   g. Customer - Review:
      Cardinality: 1 Customer : 0..* Review

Assumption: A customer can create multiple reviews for different service requests, but each review is linked to a single customer.

Relational Schema:

1. Manager
- Manager(ManagerId: INT [PK], Contact: VARCHAR(255), Image: BLOB, AgencyId: INT [FK to Agency.AgencyId])

2. Agency
- Agency(AgencyId: INT [PK], Name: VARCHAR(255), HQLocation: VARCHAR(255), Contact: VARCHAR(255))

3. Customer
- Customer(CustomerId: INT [PK], Name: VARCHAR(255), Address: TEXT, ZIP: VARCHAR(10), ProfileImage: BLOB)

4. Handyperson
- Handyperson(HandyId: INT [PK], Name: VARCHAR(255), Skills: TEXT, Rating: DECIMAL(3,1), Contact: VARCHAR(255), ProfileImage: BLOB, AgencyId: INT [FK to Agency.AgencyId])

5. ServiceRequest
- ServiceRequest(ServiceId: INT [PK], Description: TEXT, DateTime: DATETIME, Status: VARCHAR(50), Type: VARCHAR(255), Charges: DECIMAL(10,2), CustomerId: INT [FK to Customer.CustomerId], ManagerId: INT [FK to Manager.ManagerId], HandyId: INT [FK to Handyperson.HandyId])

6. Review
- Review(ReviewId: INT [PK], Comment: TEXT, Rating: DECIMAL(3,1), Timestamp: DATETIME, Image: BLOB, CustomerId: INT [FK to Customer.CustomerId], HandyId: INT [FK to Handyperson.HandyId], ServiceID: INT [FK to ServiceRequest.ServiceId])

Normalization:
We evaluate the given relational schema step by step and determine if it is in **Boyce-Codd Normal Form (BCNF)**. If any of the tables do not meet BCNF, we will normalize them.

The definition of BCNF states that a relation R is in BCNF if and only if:

Whenever there is a nontrivial FD: A1A2…AnB, then A1A2…An is a superkey for R.

1. Table: Manager

- **Schema:**
    - Manager(ManagerId [PK], Contact, Image, AgencyId [FK])
- **Functional Dependencies (FDs):**
    - ManagerId ⟶ Contact, Image, AgencyId
      (The primary key ManagerId determines all other attributes.)
- **Candidate Keys:**
    - ManagerId (the primary key)

BCNF Check:

- ManagerId ⟶ Contact, Image, AgencyId is a valid functional dependency because ManagerId is the primary key, hence a superkey.
- **This table is in BCNF.**

2. Table: Agency

- **Schema:**
    - Agency(AgencyId [PK], Name, HQLocation, Contact)
- **Functional Dependencies (FDs):**
    - AgencyId ⟶ Name, HQLocation, Contact
      (The primary key AgencyId determines all other attributes.)
- **Candidate Keys:**
    - AgencyId (the primary key)

BCNF Check:

- AgencyId → Name, HQLocation, Contact is valid because AgencyId is the primary key, hence a superkey.
- **This table is in BCNF.**

3. Table: Customer

- **Schema:**
    - Customer(CustomerId [PK], Name, Address, ZIP, ProfileImage)
- **Functional Dependencies (FDs):**

    - CustomerId → Name, Address, ZIP, ProfileImage
      (The primary key CustomerId determines all other attributes.)
- **Candidate Keys:**
    - CustomerId (the primary key)

<u>BCNF Check</u>:

- CustomerId → Name, Address, ZIP, ProfileImage is valid because CustomerId is the primary key, hence a superkey.
- **This table is in BCNF.**

4. Table: Handyperson

- **Schema:**
    - Handyperson(HandyId [PK], Name, Skills, Rating, Contact, ProfileImage, AgencyId [FK])
- **Functional Dependencies (FDs):**
    - HandyId → Name, Skills, Rating, Contact, ProfileImage, AgencyId
      (The primary key HandyId determines all other attributes.)
- **Candidate Keys:**
    - HandyId (the primary key)

<u>BCNF Check</u>:

- HandyId → Name, Skills, Rating, Contact, ProfileImage, AgencyId is valid because HandyId is the primary key, hence a superkey.
- **This table is in BCNF.**

5. Table: ServiceRequest

- **Schema:**
    - ServiceRequest(ServiceId [PK], Description, DateTime, Status, Type, Charges, CustomerId [FK], ManagerId [FK], HandyId [FK])
- **Functional Dependencies (FDs):**
    - ServiceId → Description, DateTime, Status, Type, Charges, CustomerId, ManagerId, HandyId
      (The primary key ServiceId determines all other attributes.)
- **Candidate Keys:**
    - ServiceId (the primary key)

BCNF Check:

- ServiceId → Description, DateTime, Status, Type, Charges, CustomerId, ManagerId, HandyId is valid because ServiceId is the primary key, hence a superkey.
- **This table is in BCNF.**

6. Table: Review

- **Schema:**
    - Review(ReviewId [PK], Comment, Rating, Timestamp, Image, CustomerId [FK], HandyId [FK], ServiceId [FK])
- **Functional Dependencies (FDs):**
    - ReviewId → Comment, Rating, Timestamp, Image, CustomerId, HandyId, ServiceId
      (The primary key ReviewId determines all other attributes.)
- **Candidate Keys:**
    - ReviewId (the primary key)

<u>BCNF Check</u>:

- ReviewId → Comment, Rating, Timestamp, Image, CustomerId, HandyId, ServiceId is valid because ReviewId is the primary key, hence a superkey.
- **This table is in BCNF.**

In Summary,

- **Manager**: In BCNF
- **Agency**: In BCNF
- **Customer**: In BCNF
- **Handyperson**: In BCNF
- **ServiceRequest**: In BCNF
- **Review**: In BCNF

The schema for all tables is already in **BCNF**. Each table's functional dependencies involve attributes fully determined by the primary key (or candidate key), and there are no partial or transitive dependencies. Thus, no further normalization is required and the schema remains unchanged.