DBMS ER Model Concept

> **ER Model:** ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system. It develops a conceptual design for the database. It also develops a very simple and easy to design view of data. In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram. There are several notation to represent ER diagram. They are:

→ Barker's Notation
→ Chen Notation
→ IDEF1X Notation
→ Arrow Notation
→ UML Notation
→ Crow's Foot Notation

Choosing which notation to use is typically left up to personal preference or conventions. For getting more idea about each notation you may visit: http://tiny.cc/upxe9y

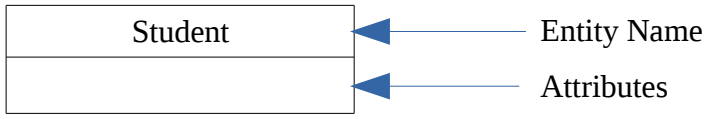
> **Components of ER model:**

- → Entity
  - → Strong Entity
  - → Weak Entity
- → Attributes
  - → Key Attribute
  - → Simple Attribute
  - → Composite Attribute
  - → Single Valued Attribute
  - → Multi Valued Attribute
  - → Descriptive Attribute
- → Relationship
  - → Degree of Relationship
  - → Mapping Cardinality

> **Entity and Entity Set:** An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Entity is represented by a single divided rectangle in UML. For example in a school database students entity can be represented as:



→ **Strong Entity:** The strong entity has a primary key. Weak entities are dependent on strong entity. Its existence is not dependent on any other entity.

→ **Weak Entity:** The weak entity in DBMS do not have a primary key and are dependent on the parent entity. It mainly depends on other entities.

Compiled By:
Mir Lutfur Rahman
Lecturer
Dept of CSE, NEUB

> **Attributes:** Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

→ **Key Attribute** − A key attribute is the unique characteristic of the entity. For example, a student id is the unique attribute for student entity.

→ **Simple Attribute** − Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.

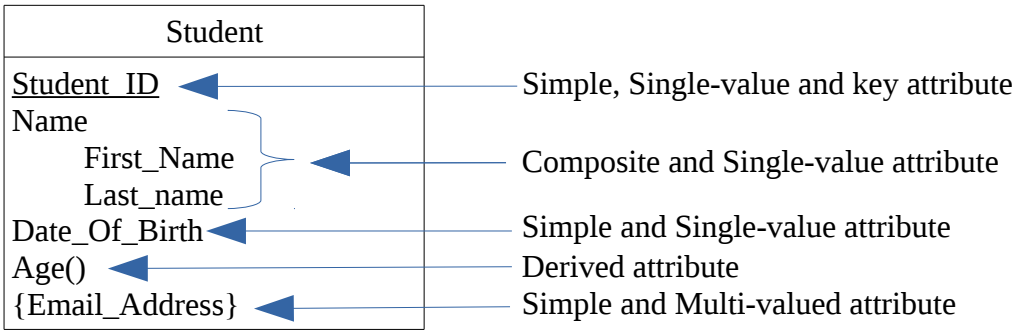
→ **Composite Attribute** − Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.

→ **Single-value Attribute** − Single-value attributes contain single value. For example, a student's id value is always a single value.

→ **Multi-value Attribute** − Multi-value attributes may contain more than one values. For example, a student can have more than one email_address, etc.

→ **Derived Attribute** − Derived attributes are the attributes that **do not exist in the physical database**, but their values are derived from other attributes present in the database. For example, a student age can be derived from his/her data_of_birth.

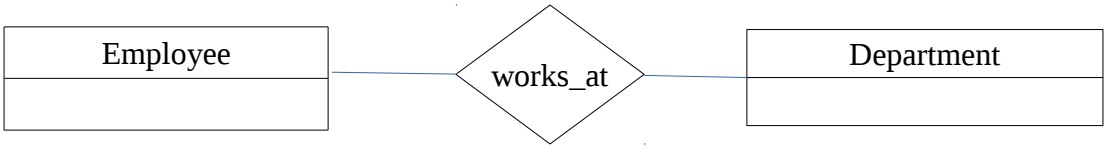
In UML notation:



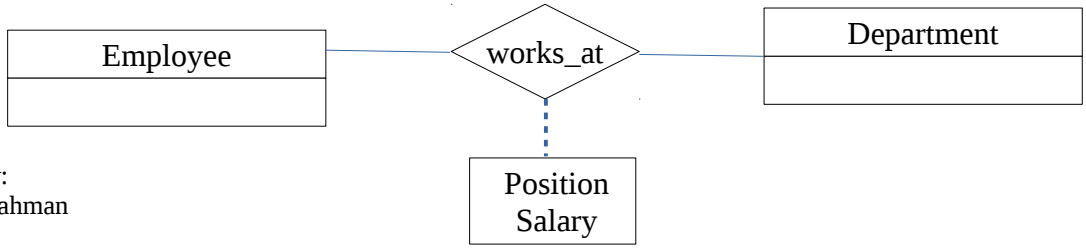
> **Relationship and Relationship Set:** The association among entities is called a relationship. For example, an employee works_at a department, a student enrolls in a course. Here, Works_at and Enrolls are called relationships.

A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

Relationship is represented by a diamond shape in UML. For example in a company an employee works at a department it can be represented as:



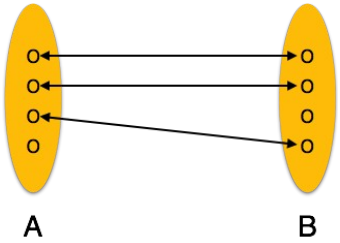
In UML descriptive attributes represented by an undivided rectangle which connected with the relationship by a dotted line. For example if employee works at a department in a specific position and salary then position and salary both will be descriptive attributes and can be represented as:



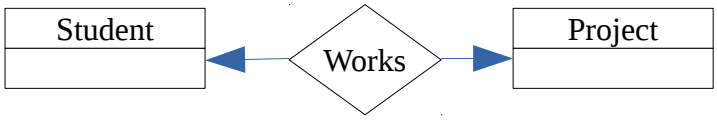
→ **Degree of Relationship:** The number of participating entities in a relationship defines the degree of the relationship. The degree of a relationship can be binary, ternary, n-ary. For example, in our previous figure employee and department entity has a relationship where employee works at any department. So in that case two entity creating the works at relationship hence degree of this relationship will be 2 or binary.

→ **Mapping Cardinality:** Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other entity set via relationship set. For a binary relationship there will be four possible cardinalities:

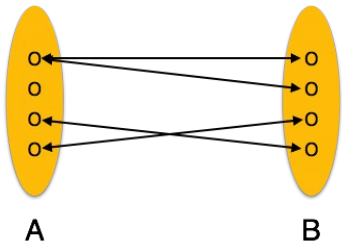
→ **One to One:** One entity from entity set A can be associated with at most one entity of entity set B and vice versa. For example, A student works in a specific project.



In UML notation:

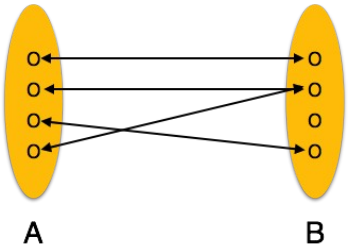


→ **One to Many:** One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity. For example, A department consists of many students.



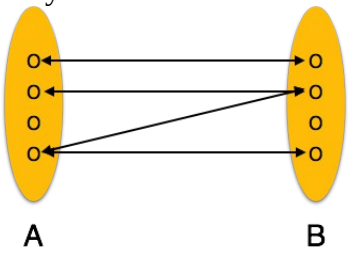
In UML notation:

→ **Many to One:** More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A. Consider the previous example in reverse form.



In UML notation:

→ **Many to Many:** One entity from A can be associated with more than one entity from B and vice versa. For example, relationship exists between customers and products: customers can purchase various products, and products can be purchased by many customers.



In UML notation:

> **Types of Relationship Constraints:** We can distinguish two main types of relationship constraints

1) Cardinality Constraints - An important feature of entity-relationship schemata is the possibility of specifying cardinality constraints. These are instructions of the form "every person has exactly one mother", or "every document must have at least an author (but possibly many)". There constraints are useful because they allow to maintain the *logical integrity* of the database: in a logically integral database, you can always trust a document to have an author, and you can avoid to check if it is missing.

Cardinality constraints restrain the possible multi-relation that an instance can assign to the relationship type. For each of the two entity types you can specify a pair of indices separated by two dots, as in `L..H`. The left character may be `0` or `1`, while the right character may be `1` or `*`.

2) Participation Constraints - A participation constraint defines the number of times an entity In an entity set can participate in a connected relationship set. Every connection of a relationship set must have a participation constraint. However, participation constraints **do not apply to relationships.** There are two types of participation constraints:

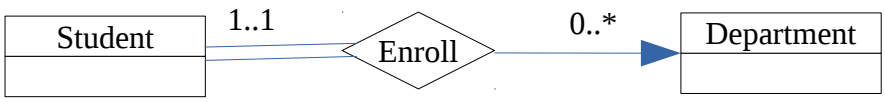
→ **Total Participation:** It specifies that each entity in the entity set must compulsorily participate in at least one relationship instance in that relationship set. That is why, it is also called as **mandatory participation.** Total participation is represented using a double line between the entity set and relationship set. For example, a department may have zero to many students but a student must be enrolled in a department. So every student entity will participate in the department relationship and it will be a total participation.

→ **Partial Participation:** It specifies that each entity in the entity set may or may not participate in the relationship instance in that relationship set. That is why, it is also called as optional participation. Partial participation is represented using a single line between the entity set and relationship set. In above example a department may have zero to many enrolled students so in that case it will be a partial participation. Because there is a possibility that a department have no student.

Minimum cardinality tells whether the participation is partial or total.

- If minimum cardinality = 0, then it signifies partial participation.
- If minimum cardinality = 1, then it signifies total participation.

Maximum cardinality tells the maximum number of entities that participates in a relationship set. In above example if we use cardinality constraint then it's UML notation will be:



Here one student must be enrolled in a department so minimum cardinality is 1 and that's why it is a total participation. And maximum cardinality 1 represent that a student at most enrolled in a single department. There is no such possibility that a student enrolled more than one department.

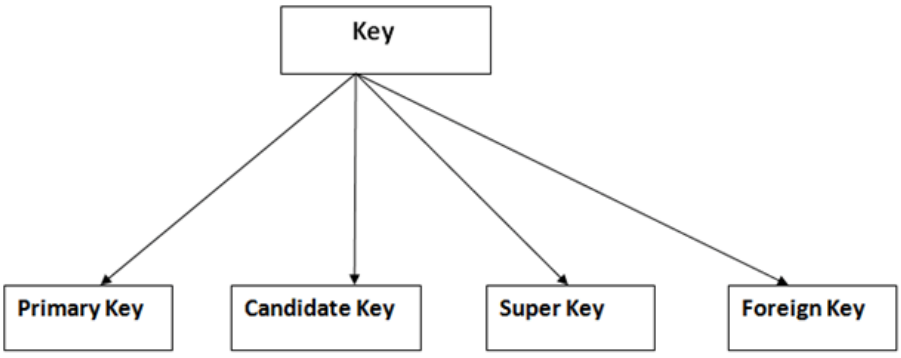
On the other hand department minimum cardinality 0 represent that a department may have zero student and maximum cardinality * represent that a department may have many students.

> **Identifying Relationship:** An Identifying Relationship is a relationship between a strong and a weak entity type, where the key of the strong entity type is required to uniquely identify instances of the weak entity type. The weak entity type will have a **partial key (discriminator) attribute** that, in conjunction with the key of the strong entity type, uniquely identifies weak entity instances.

Consider a university environment where many different courses are offered. Many universities assign numbers to their courses and so, for example, we could have courses such as 2914, 3902, 3913, etc. It is typical that a Course may be offered zero, one, or more times, in a single term. Hence, there may be several Sections of the same Course each term. Each Section needs to be identified distinctly from any other. The typical solution is use a section number such as 001, 002, 003, etc. The section number is appended to the course identifier to yield identifiers for the course offerings, such as 2914-001, 2914-002, 3902-050, etc. In this example, Section is modeled as a Weak Entity Type and its relationship to Course is an identifying relationship.

Note in the following diagram that the identifying relationship is drawn with a double line. Since Section is a weak entity it is shown with a double lined box, and since a Section cannot exist on its own but in relationship to a Course we shown a mandatory relationship too. Note that Section has a **partial key** sectionNo, Course has a key courseNo.

> **DBMS Keys:** Keys play an important role in the relational database. It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables. For example: In Student table, ID is used as a key because it is unique for each student. In PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.



→ **Primary Key:** It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys. The key which is most suitable from those lists become a primary key. For each entity, selection of the primary key is based on requirement and developers.

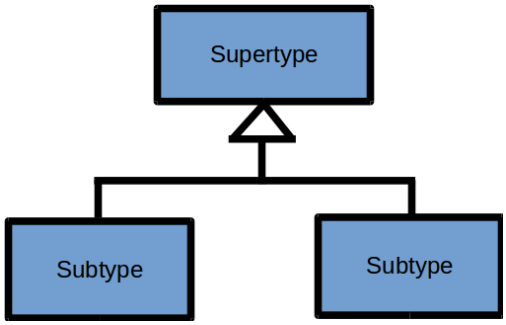
→ **Candidate Key:** A candidate key is an attribute or set of an attribute which can uniquely identify a tuple. The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key. For example: In the EMPLOYEE table,

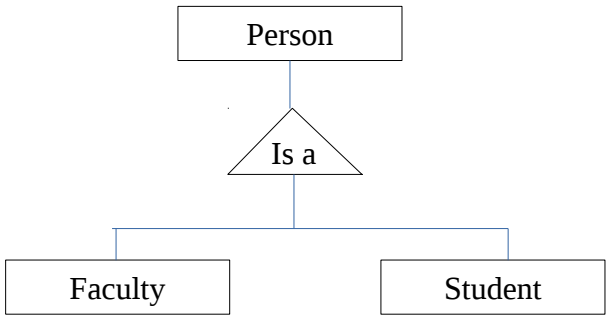
id is best suited for the primary key. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.

→ **Super Key:** Super key is a set of an attribute which can uniquely identify a tuple. Super key is a super set of a candidate key. For example: In EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key. The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

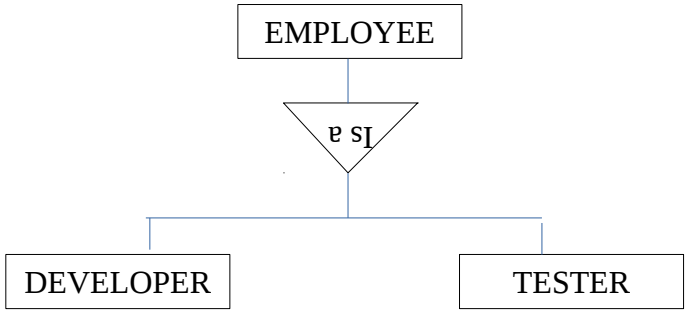
→ **Foreign Key:** Foreign keys are the column of the table which is used to point to the primary key of another table. In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table. We add the primary key of the DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table. Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.

> **DBMS Generalization:** Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common. In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity. Generalization is more like sub class and super class system, but the only difference is the approach. Generalization uses the bottom-up approach. In generalization, entities are combined to form a more generalized entity, i.e., sub classes are combined to make a super class. It is an **extended feature** of ERD. For example, Faculty and Student entities can be generalized and create a higher level entity Person.

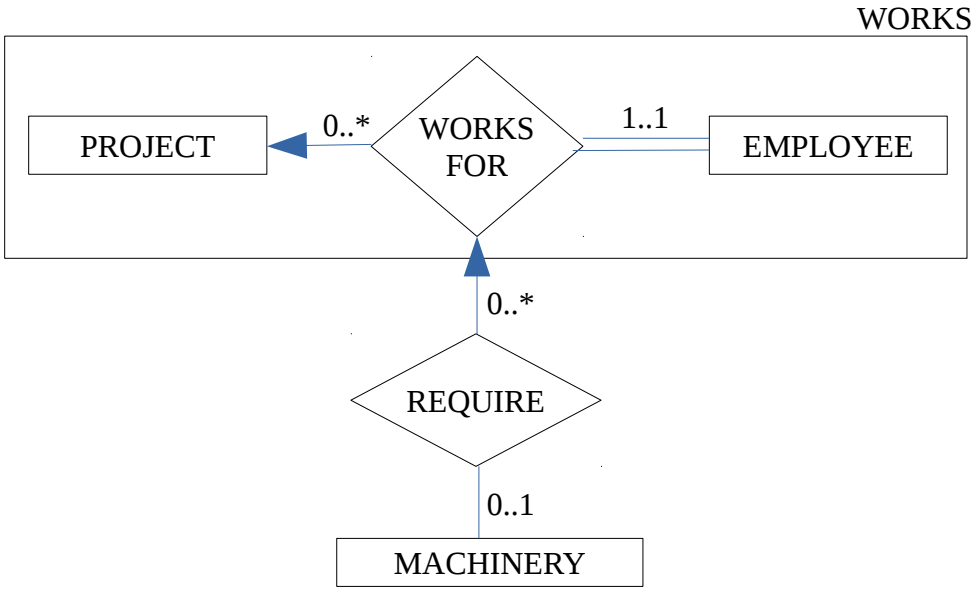




> **DBMS Specialization:** In specialization, an entity is divided into sub-entities based on their characteristics. It is a top-down approach and reverse of generalization where higher level entity is specialized into two or more lower level entities. It is an **extended feature** of ERD. For Example, EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER etc.



> **DBMS Aggregation:** An ER diagram is not capable of representing relationship between an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher level entity. For Example, Employee working for a project may require some machinery. So, REQUIRE relationship is needed between relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into single entity and relationship REQUIRE is created between aggregated entity and MACHINERY.



> **How to Draw ER diagrams:** Below points show how to go about creating an ER diagram:

1. **Identify all the entities** in the system. An entity should appear only once in a particular diagram. Create rectangles for all entities and name them properly.
2. **Identify relationships** between entities. Connect them using a line and add a diamond in the middle describing the relationship.
3. **Add attributes** for entities. Give meaningful attribute names so they can be understood easily.

Sounds simple right? In a complex system, it can be a nightmare to identify relationships. This is something you'll perfect only with practice.

> **ER Diagram Best Practices:**

1. Provide a precise and appropriate name for each entity, attribute, and relationship in the diagram. Terms that are simple and familiar always beats vague, technical-sounding words. In naming entities, remember to use singular nouns. However, adjectives may be used to distinguish entities belonging to the same class (part-time employee and full-time employee, for example). Meanwhile attribute names must be meaningful, unique, system-independent, and easily understandable.
2. Remove vague, redundant or unnecessary relationships between entities.
3. Never connect a relationship to another relationship.
4. Make effective use of colors. You can use colors to classify similar entities or to highlight key areas in your diagrams.

> **Benefits of ER diagrams:** ER diagrams constitute a very useful framework for creating and manipulating databases. First, ER diagrams are easy to understand and do not require a person to undergo extensive training to be able to work with it efficiently and accurately. This means that designers can use ER diagrams

to easily communicate with developers, customers, and end users, regardless of their IT proficiency. Second, ER diagrams are readily translatable into relational tables which can be used to quickly build databases. In addition, ER diagrams can directly be used by database developers as the blueprint for implementing data in specific software applications. Lastly, ER diagrams may be applied in other contexts such as describing the different relationships and operations within an organization.

> A SIMPLE EXAMPLE

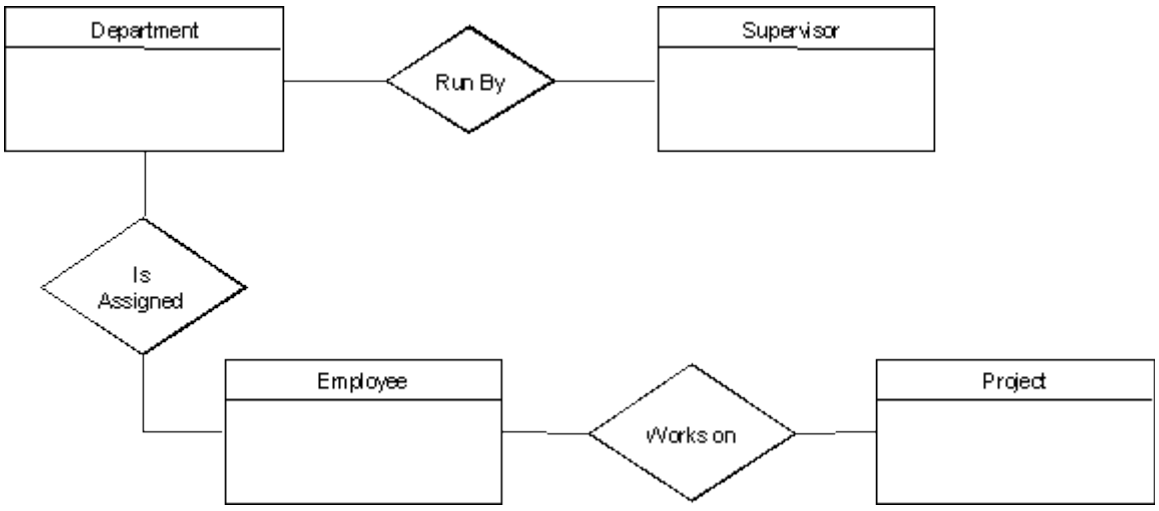
A company has several departments. Each department has a supervisor and at least one employee. Employees must be assigned to at least one, but possibly more departments. At least one employee is assigned to a project, but an employee may be on vacation and not assigned to any projects. The important data fields are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee number and a unique project number.

1) Identify Entities: The entities in this system are **Department**, **Employee**, **Supervisor** and **Project**. One is tempted to make Company an entity, but it is a false entity because it has only one instance in this problem. True entities must have more than one instance.

2) Find Relationships: We construct the following Entity Relationship Matrix:

	Department	**Employee**	**Supervisor**	**Project**
Department		is assigned	run by	
Employee	belongs to			works on
Supervisor	runs			
Project		uses		

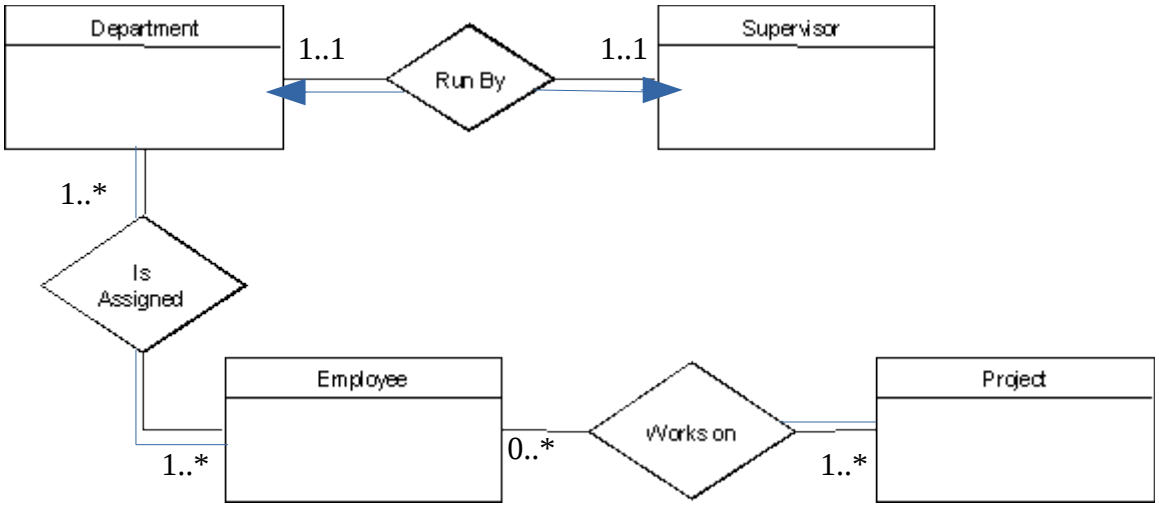
3) Draw Rough ERD: We connect the entities whenever a relationship is shown in the entity Relationship Matrix.



4) Fill in Cardinality: From the description of the problem we see that:

- ✔ Each department has exactly one supervisor.
- ✔ A supervisor is in charge of one and only one department.
- ✔ Each department is assigned at least one employee.

- ✔ Each employee works for at least one department.
- ✔ Each project has at least one employee working on it.
- ✔ An employee is assigned to 0 or more projects.



5) Define Primary Keys: The primary keys are Department Name, Supervisor Number, Employee Number, Project Number.

6) Identify Attributes: The only attributes indicated are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee NUMBER and a unique project number.

So the fully attributed ERD in UML notation: