Chapter 2: Data Models

2.1 Conceptual, Logical and Physical Model:

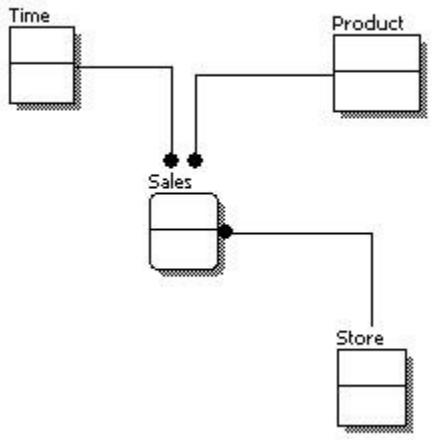
Conceptual Data Model:

A conceptual data model identifies the highest-level relationships between the different entities.
Features of conceptual data model include:

- Includes the important entities and the relationships among them.
- No attribute is specified.
- No primary key is specified.

The figure below is an example of a conceptual data model.

Conceptual Data Model



From the figure above, we can see that the only information shown via the conceptual data model is the entities that describe the data and the relationships between those entities. No other information is shown through the conceptual data model.

Logical Data Model:

A logical data model describes the data in as much detail as possible, without regard to how they will be physical implemented in the database. Features of a logical data model include:

- Includes all entities and relationships among them.
- All attributes for each entity are specified.
- The primary key for each entity is specified.
- Foreign keys (keys identifying the relationship between different entities) are specified.
- Normalization occurs at this level.

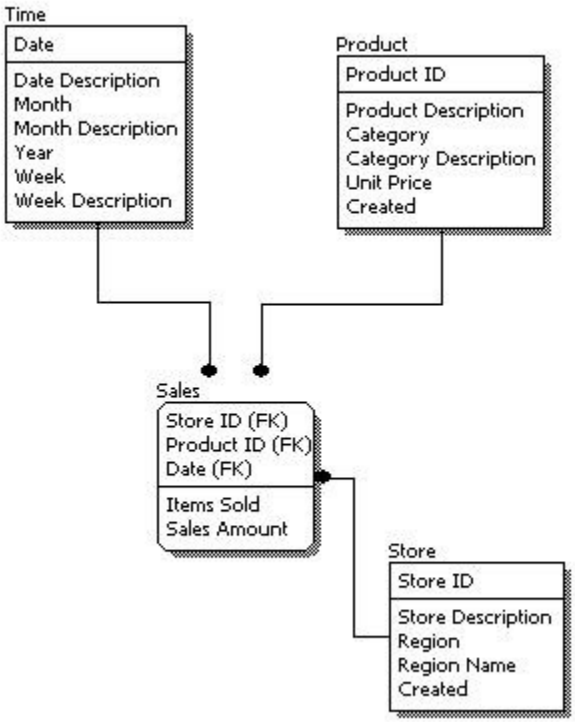
The steps for designing the logical data model are as follows:

1. Specify primary keys for all entities.

2. Find the relationships between different entities.
3. Find all attributes for each entity.
4. Resolve many-to-many relationships.
5. Normalization.

The figure below is an example of a logical data model.

Logical Data Model



Comparing the logical data model shown above with the conceptual data model diagram, we see the main differences between the two:

- In a logical data model, primary keys are present, whereas in a conceptual data model, no primary key is present.
- In a logical data model, all attributes are specified within an entity. No attributes are specified in a conceptual data model.
- Relationships between entities are specified using primary keys and foreign keys in a logical data model. In a conceptual data model, the relationships are simply stated, not specified, so we simply know that two entities are related, but we do not specify what attributes are used for this relationship.

Physical Data Model:

Physical data model represents how the model will be built in the database. A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables. Features of a physical data model include:

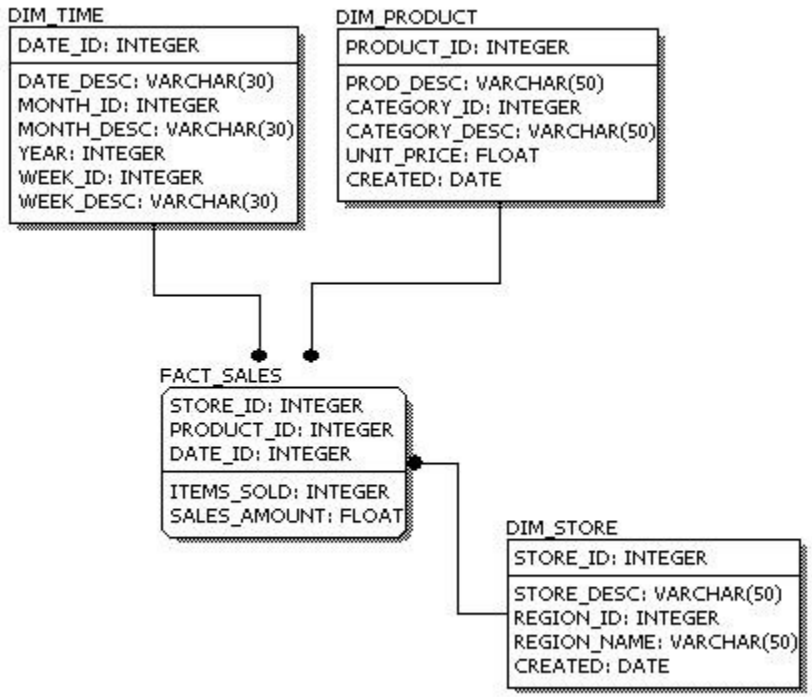
- Specification all tables and columns.
- Foreign keys are used to identify relationships between tables.
- Denormalization may occur based on user requirements.
- Physical considerations may cause the physical data model to be quite different from the logical data model.
- Physical data model will be different for different RDBMS. For example, data type for a column may be different between MySQL and SQL Server.

The steps for physical data model design are as follows:

1. Convert entities into tables.
2. Convert relationships into foreign keys.
3. Convert attributes into columns.
4. Modify the physical data model based on physical constraints / requirements.

The figure below is an example of a physical data model.

Physical Data Model



Comparing the physical data model shown above with the logical data model diagram, we see the main differences between the two:

- Entity names are now table names.
- Attributes are now column names.
- Data type for each column is specified. Data types can be different depending on the actual database being used.

The table below compares the different features:

Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	✓
Foreign Keys		✓	✓
Table Names			✓
Column Names			✓
Column Data Types			✓

2.2 E-R Model:

The entity-relationship(E-R) data model is based on a perception of a real world that consists of a collection of a basic objects, called entities, and of relationships among these objects. The E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema. It employs three basic notions: entity sets, relationship sets, and attributes.

- **Entities and Their Attributes.** The basic object that the ER model represents is an **entity**, which is a *thing* in the real world with an independent existence. An entity may be an object with a physical existence (for example, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for instance, a company, a job, or a university course). Each entity has **attributes**—the particular properties that describe it. For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job. A particular entity will have a value for each of its attributes. The attribute values that describe each entity become a major part of the data stored in the database.
- A **relationship** is an association among several entities. For example, a **depositor** relationship associates a customer with each account they have. The set of all entities of the same type and the set of all the relationships of the same type are termed as **entity set** and **relationship set** respectively.

Types Of Attributes:

• **Simple** and **composite** attributes. The attributes are not divided into subparts are called **Simple** attributes. **Composite** attributes, on the other hand, can be divided into subparts (that is, other attributes). For example, an attribute *name* could be structured as a composite attribute consisting of *first-name*, *middle-initial*, and *last-name*. Using composite attributes in a design schema is a good choice if a user will wish to refer to an entire attribute on some occasions, and to only a component of the attribute on other occasions. Figure 2.2 depicts these examples of composite attributes for the *customer* entity set.

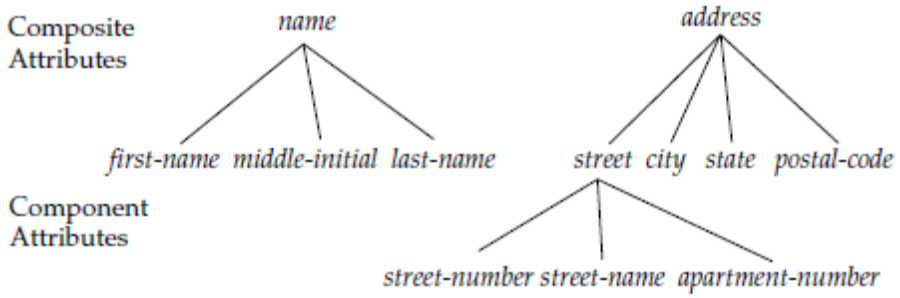


Figure 2.2 Composite attributes *customer-name* and *customer-address*.

• **Single-valued** and **multivalued** attributes. The attributes in our examples all have a single value for a particular entity. For instance, the *loan-number* attribute for a specific loan entity refers to only one loan number. Such attributes are said to be **single valued**. There may be instances where an attribute has a set of values for a specific entity. Consider an *employee* entity set with the attribute *phone-number*. An employee may have zero, one, or several phone numbers, and different employees may have different numbers of phones. This type of attribute is said to be **multivalued**.

- **Derived** attribute. The value for this type of attribute can be derived from the values of other related attributes or entities. For example, If the *customer* entity set also has an attribute *date-of-birth*, we can calculate *age* from *date-of-birth* and the current date. Thus, *age* is a derived attribute. In this case, *date-of-birth* may be referred to as a *base* attribute, or a *stored* attribute. The value of a derived attribute is not stored, but is computed when required.
- An attribute takes a **null** value when an entity does not have a value for it. The *null* value may indicate "not applicable"—that is, that the value does not exist for the entity. For example, one may have no middle name. *Null* can also designate that an attribute value is unknown. An unknown value may be either *missing* (the value does exist, but we do not have that information) or *not known* (we do not know whether or not the value actually exists).

Constraints:

An E-R enterprise schema may define certain constraints to which the contents of a database must conform. Here, we examine mapping cardinalities and participation constraints, which are two of the most important types of constraints.

Mapping Cardinalities:

Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.

For a binary relationship set *R* between entity sets *A* and *B*, the mapping cardinality must be one of the following:

• **One to one**. An entity in *A* is associated with *at most* one entity in *B*, and an entity in *B* is associated with *at most* one entity in *A*.

• **One to many**. An entity in *A* is associated with any number (zero or more) of entities in *B*. An entity in *B*, however, can be associated with *at most* one entity in *A*.

• **Many to one**. An entity in *A* is associated with *at most* one entity in *B*. An entity in *B*, however, can be associated with any number (zero or more) of entities in *A*.

• **Many to many**. An entity in *A* is associated with any number (zero or more) of entities in *B*, and an entity in *B* is associated with any number (zero or more) of entities in *A*.

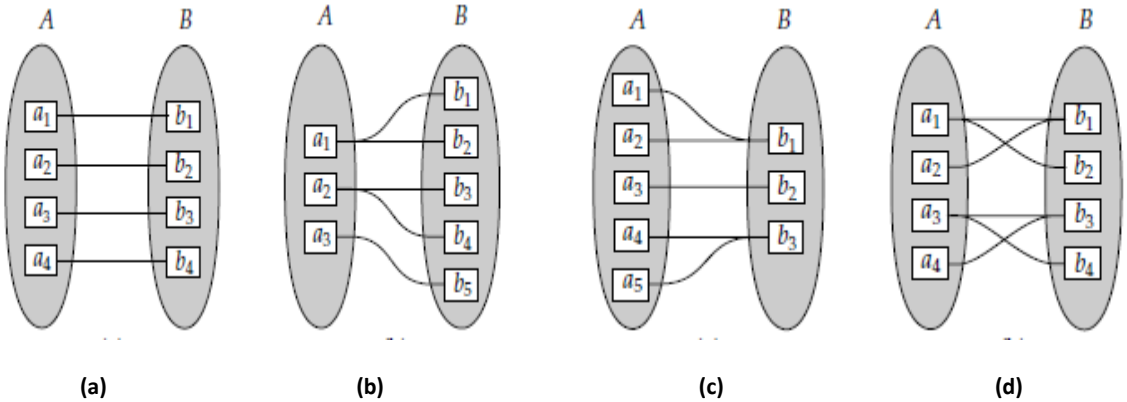


Fig: Mapping Cardinalities (a) One to one. (b)One to many. (c) Many to one. (d) Many to many

Participation Constraints:

The participation of an entity set *E* in a relationship set *R* is said to be **total** if every entity in *E* participates in at least one relationship in *R*. If only some entities in *E* participate in relationships in *R*, the participation of entity set *E* in relationship *R* is said to be **partial**. For example, we expect every loan entity to be related to at least one customer through the *borrower* relationship. Therefore the participation of *loan* in the relationship set *borrower* is total. In contrast, an individual can be a bank customer whether or not she has a loan with the bank. Hence, it is possible that only some of the *customer* entities are related to the *loan* entity set through the *borrower* relationship, and the participation of *customer* in the *borrower* relationship set is therefore partial.

Entity-Relationship Diagram:

An **E-R diagram** can express the overall logical structure of a database graphically. E-R diagrams are simple and clear—qualities that may well account in large part for the widespread use of the E-R model. Such a diagram consists of the following major components:

• **Rectangles**, which represent entity sets
• **Ellipses**, which represent attributes
• **Diamonds**, which represent relationship sets
• **Lines**, which link attributes to entity sets and entity sets to relationship sets
• **Double ellipses**, which represent multivalued attributes
• **Dashed ellipses**, which denote derived attributes
• **Double lines**, which indicate total participation of an entity in a relationship set
• **Double rectangles**, which represent weak entity sets

Consider the entity-relationship diagram in Figure below, which consists of two entity sets, *customer* and *loan*, related through a binary relationship set *borrower*. The attributes associated with *customer* are *customer-id*, *customer-name*, *customer-street*, and *customer-city*. The attributes associated with *loan* are *loan-number* and *amount*. In Figure, attributes of an entity set that are members of the primary key are underlined. The relationship set *borrower* may be many-to-many, one-to-many, many-to-one, or one-to-one. To distinguish among these types, we draw either a directed line (→) or an undirected line (—) between the relationship set and the entity set in question.

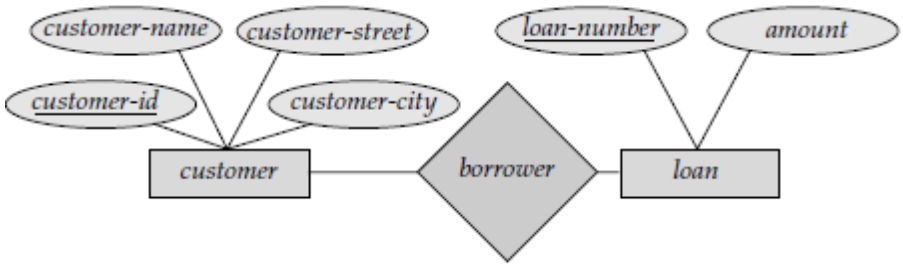


Fig: E-R Diagram corresponding to Customers and loans

• A directed line from the relationship set *borrower* to the entity set *loan* specifies that *borrower* is either a one-to-one or many-to-one relationship set, from *customer* to *loan*; *borrower* cannot be a many-to-many or a one-to-many relationship set from *customer* to *loan*.
• An undirected line from the relationship set *borrower* to the entity set *loan* specifies that *borrower* is either a many-to-many or one-to-many relationship set from *customer* to *loan*.
Returning to the E-R diagram of above Figure , we see that the relationship set *borrower* is many-to-many. If the relationship set *borrower* were one-to-many, from *customer* to *loan*, then the line from *borrower* to *customer* would be directed, with an arrow pointing to the *customer* entity set . Similarly, if the relationship set *borrower* were many-to-one from *customer* to *loan*, then the line from *borrower* to *loan* would have an arrow pointing to the *loan* entity set . Finally, if the relationship set *borrower* were one-to-one, then both lines from *borrower* would have arrows: one pointing to the *loan* entity set and one pointing to the *customer* entity set .

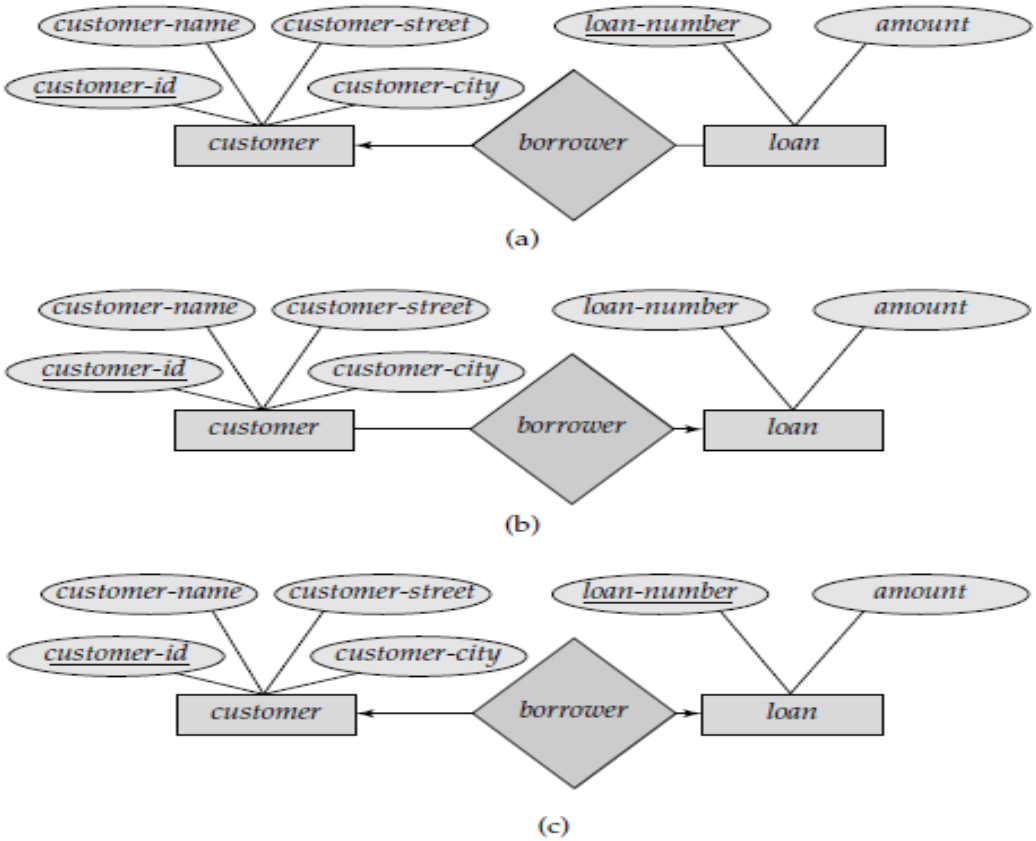


Fig: Relationships.(a) One to many.(b) Many to one.(c)One to one.

If a relationship set has also some attributes associated with it, then we link these attributes to that relationship set. For example, in Figure 2.10, we have the *access-date* descriptive attribute attached to the relationship set *depositor* to specify the most recent date on which a customer accessed that account.

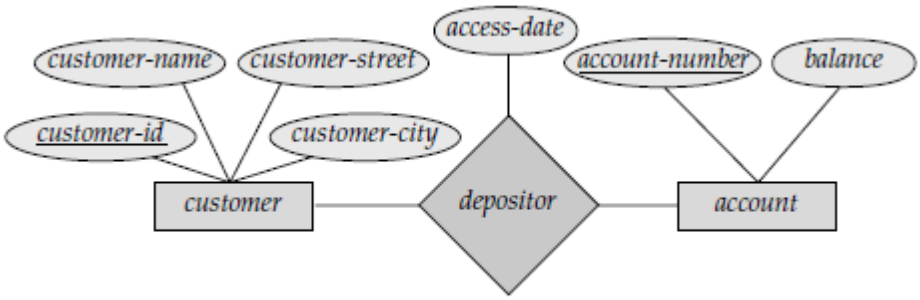


Figure 2.10 E-R diagram with an attribute attached to a relationship set.

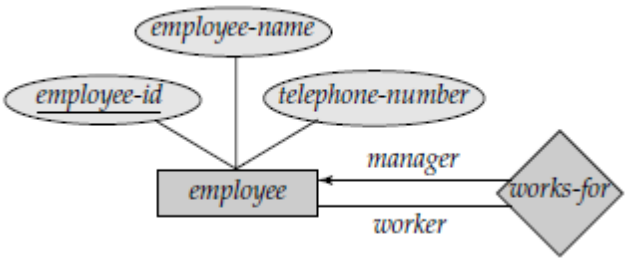


Fig: E-R diagram with role indicators.

Example: Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.

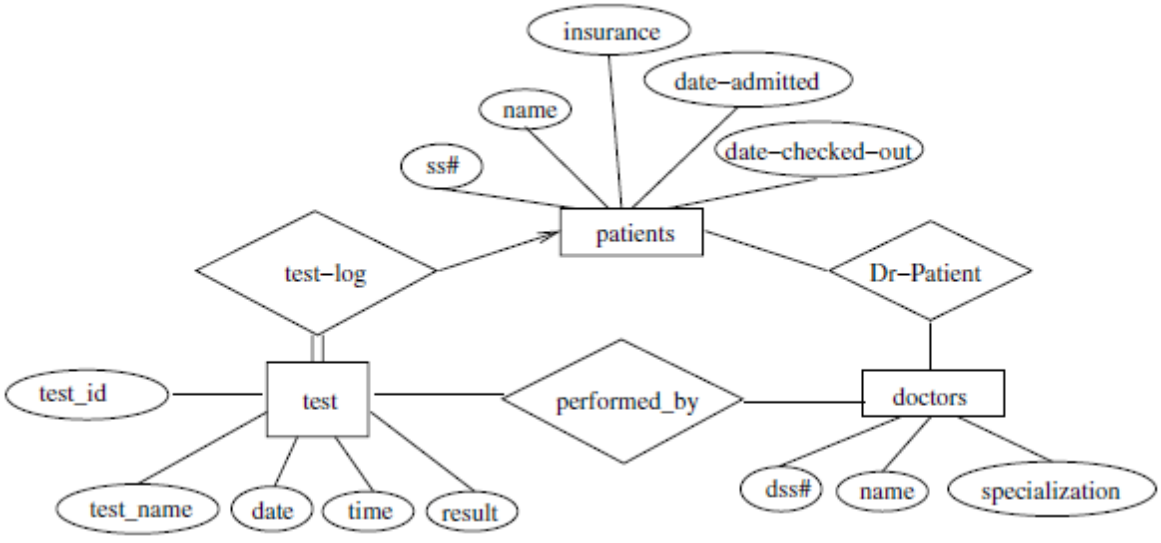


Fig: E-R diagram for a hospital

Example: Consider a database to record the marks that students get in different exams of different course offerings. Construct an E-R diagram that shows relationship, for the above database.

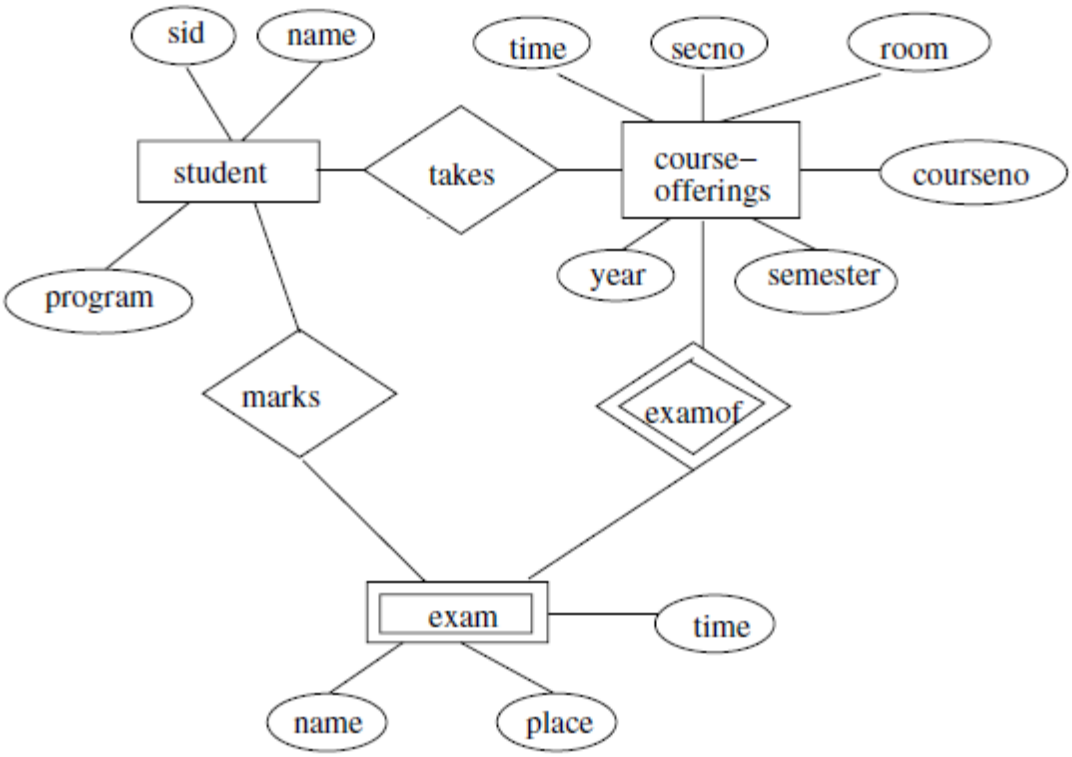


Fig: E-R diagram for marks database

Extended E-R Features:

Although the basic E-R concepts can model most database features, some aspects of a database may be more aptly expressed by certain extensions to the basic E-R model. Here, we discuss the extended E-R features of specialization and generalization.

Specialization:

An entity set may include subgroupings of entities that are distinct in some way from other entities in the set. For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set. The E-R model provides a means for representing these distinctive entity groupings. Consider an entity set *person*, with attributes *name*, *street*, and *city*. A person may be further classified as one of the following:

- *customer*
- *employee*

Each of these person types is described by a set of attributes that includes all the attributes of entity set *person* plus possibly additional attributes. For example, *customer* entities may be described further by the attribute *customer-id*, whereas *employee* entities may be described further by the attributes *employee-id* and *salary*. The process of designating subgroupings within an entity set is called **specialization**. The specialization of *person* allows us to distinguish among persons according to whether they are employees or customers.

Generalization:

The refinement from an initial entity set into successive levels of entity subgroupings represents a **top-down** design process in which distinctions are made explicit. The design process may also proceed in a **bottom-up** manner, in which multiple entity sets are synthesized into a higher-level entity set on the

basis of common features. The database designer may have first identified a *customer* entity set with the attributes *name*, *street*, *city*, and *customer-id*, and an *employee* entity set with the attributes *name*, *street*, *city*, *employee-id*, and *salary*. There are similarities between the *customer* entity set and the *employee* entity set in the sense that they have several attributes in common. This commonality can be expressed by **generalization**, which is a containment relationship that exists between a *higher-level* entity set and one or more *lower-level* entity sets. In our example, *person* is the higher-level entity set and *customer* and *employee* are lower-level entity sets. Higher- and lower-level entity sets also may be designated by the terms **superclass** and **subclass**, respectively. The *person* entity set is the superclass of the *customer* and *employee* subclasses. For all practical purposes, generalization is a simple inversion of specialization.

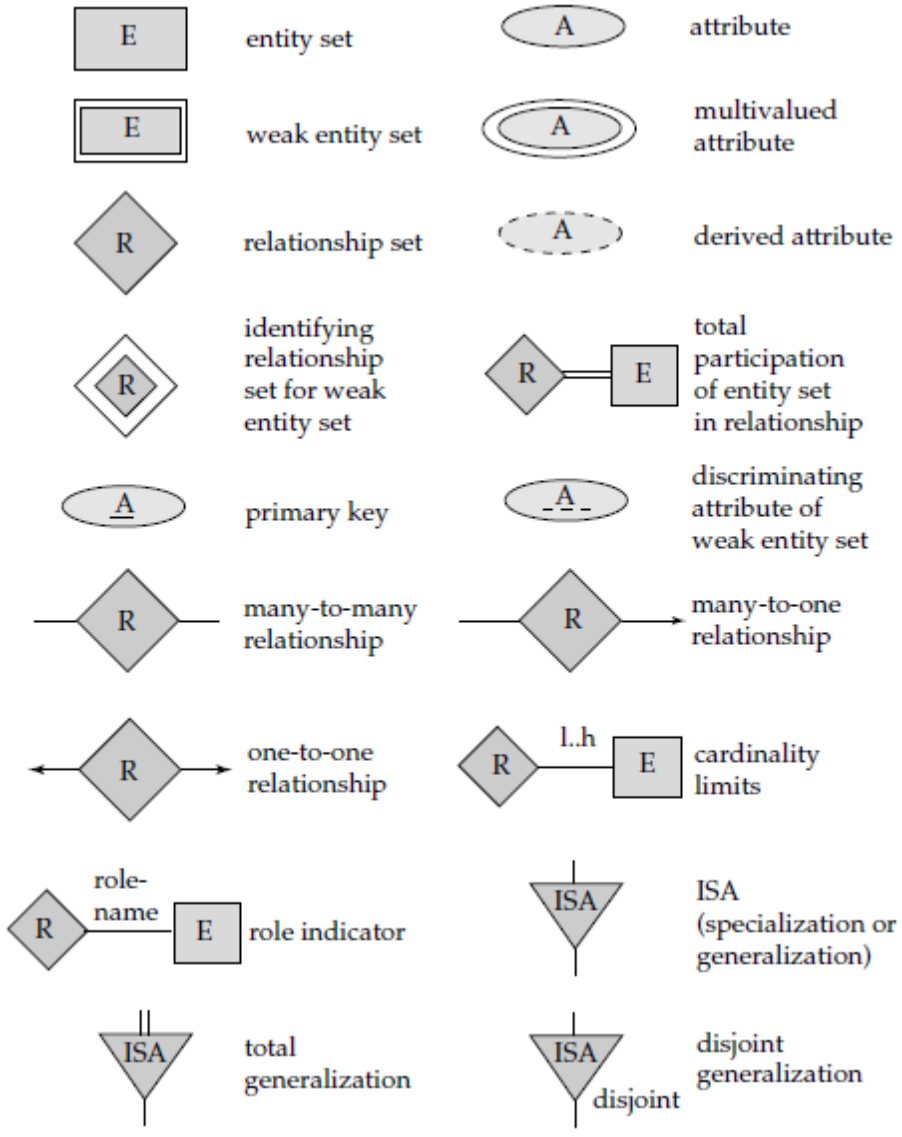


Fig: Symbols used in E-R notations

2.3 Relation with UML class diagram:

Entity-relationship diagrams help model the data representation component of a software system. Data representation, however, forms only one part of an overall system design. Other components include

models of user interactions with the system, specification of functional modules of the system and their interaction, etc. The **Unified Modeling Language** (UML), is a proposed standard for creating specifications of various components of a software system. Some of the parts of UML are:

• **Class diagram**. A class diagram is similar to an E-R diagram. Later we illustrate a few features of class diagrams and how they relate to E-R diagrams.

• **Use case diagram**. Use case diagrams show the interaction between users and the system, in particular the steps of tasks that users perform (such as withdrawing money or registering for a course).

• **Activity diagram**. Activity diagrams depict the flow of tasks between various components of a system.

• **Implementation diagram**. Implementation diagrams show the system components and their interconnections, both at the software component level and the hardware component level.

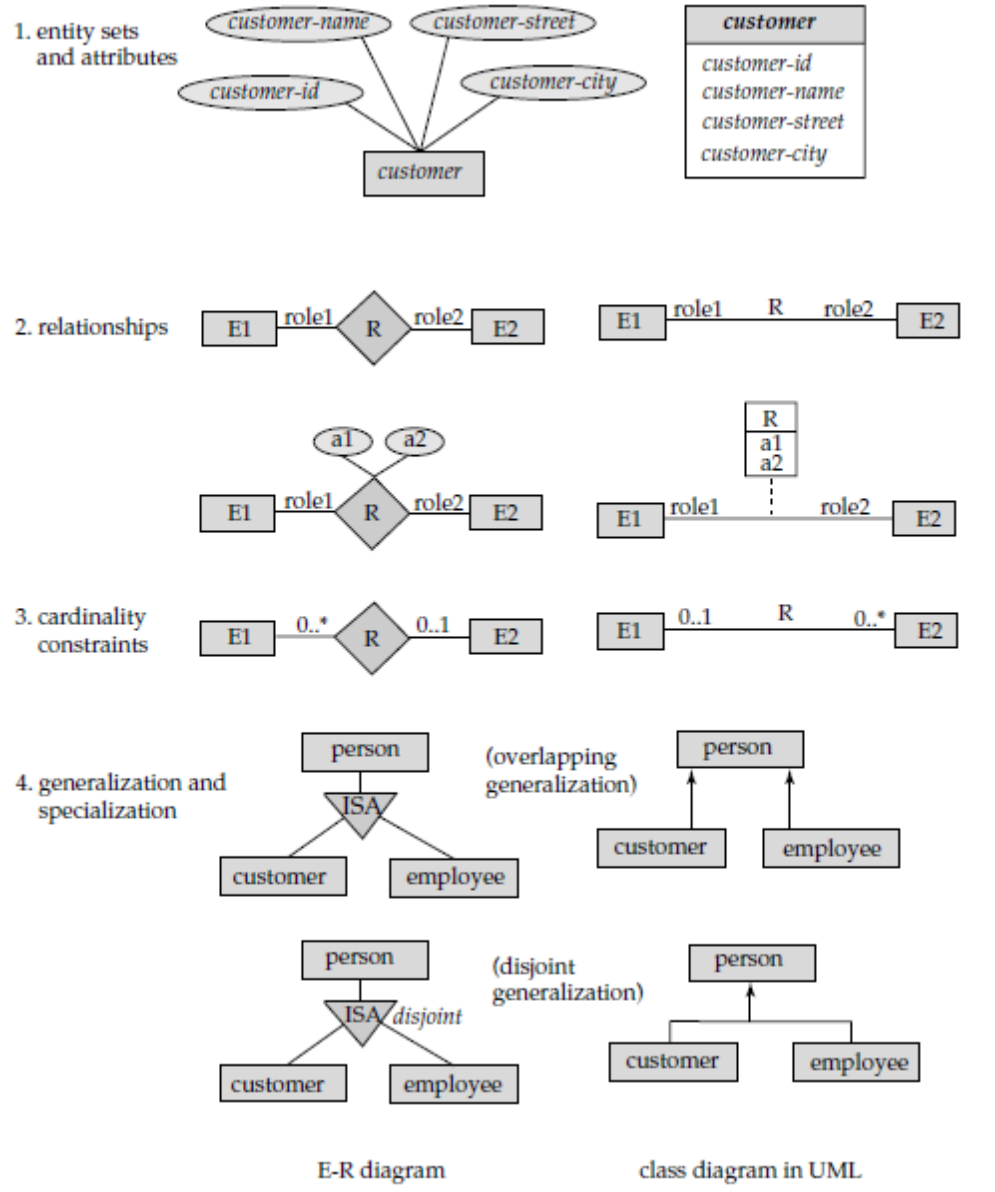
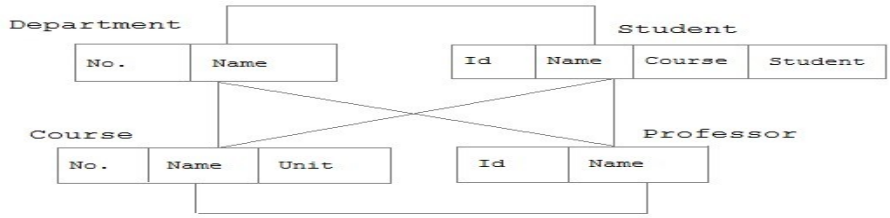


Fig: Symbols used in the UML class diagram notation

2.4 Alternative data models(Network Data Model, Heirarchical Data Model):

Network Data Model:

The **network model** is a database model conceived as a flexible way of representing objects and their relationships. Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs, is not restricted to being a hierarchy or lattice. The **network model** replaces the hierarchical model with a graph thus allowing more general connections among the nodes.The main difference of the **network model** from the hierarchical model is its ability to handle many to many relationships.In other words it allow a record to have more than one parent.
Example:



Advantages of a Network Database Model

- Because it has the many-many relationship, network database model can easily be accessed in any table record in the database
- For more complex data, it is easier to use because of the multiple relationship founded among its data
- Easier to navigate and search for information because of its flexibility

Disadvantage of a Network Database Model

- Difficult for first time users
- Difficulties with alterations of the database because when information entered can alter the entire database

Hierarchical Database Model:

The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. This structure implies that a record can have repeating information, generally in the child data segments. Data in a series of records, which have a set of field values attached to it. It collects all the instances of a specific record together as a record type. These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows. To create links between these record types, the hierarchical model uses Parent Child Relationships. These are a 1:N mapping between record types. This is done by using trees, like set theory used in the relational model, "borrowed" from maths. At the top of hierarchy there is only one entity which is called **Root**.
Example: