E-R Model

Hi! Here in this lecture we are going to discuss about the E-R Model.

What is Entity-Relationship Model?

The entity-relationship model is useful because, as we will soon see, it facilitates communication between the database designer and the end user during the requirements analysis. To facilitate such communication the model has to be simple and readable for the database designer as well as the end user. It enables an abstract global view (that is, the *enterprise conceptual schema*) of the enterprise to be developed without any consideration of efficiency of the ultimate database.

The entity-relationship model views an enterprise as being composed of *entities* that have *relationships* between them. To develop a database model based on the E-R technique involves identifying the entities and relationships of interest. These are often displayed in an E-R diagram. Before discussing these, we need to present some definitions.

Entities and Entity Sets

An *entity* is a person, place, or a thing or an object or an event which can be distinctly identified and is of interest. A specific student, for example, John Smith with student number 84002345 or a subject Database Management Systems with subject number CP3010 or an institution called James Cook University are examples of entities. Although entities are often concrete like a company, an entity may be abstract like an idea, concept or convenience, for example, a research project P1234 - Evaluation of Database Systems or a Sydney to Melbourne flight number TN123.

Entities are classified into different *entity sets* (or entity types). An entity set is a set of entity instances or entity occurrences of the same type. For example, all employees of a company may constitute an entity set *employee* and all departments may belong to an entity set *Dept*. An entity may belong to one or more entity sets. For example, some company employees may belong to *employee* as well as other entity sets, for example, *managers*. Entity sets therefore are not always disjoint.

We remind the reader that an entity set or entity type is a collection of entity instances of the same type and must be distinguished from an entity instance.

The real world does not just consist of entities. As noted in the last chapter, a database is a collection of interrelated information about an enterprise. A database therefore consists of a collection of entity sets; it also includes information about relationships between the entity sets. We discuss the relationships now.

Relationships, Roles and Relationship Sets

Relationships are associations or connections that exist between entities and may be looked at as mappings between two or more entity sets. A relation therefore is a subset of the cross products of the entity sets involved in the relationship. The associated entities may be of the same type or of different types. For example, *working - for* is a relationship between an employee and a company. *Supervising* is a relationship between two entities belonging to the same entity set (*employee*).

A *relationship set* R_i is a set of relations of the same type. It may be looked at as a mathematical relation among *n* entities each taken from an entity set, not necessarily distinct:

$$R_i = [e_1, e_2, \ldots, e_n] \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n$$

where each e_i is an entity instance in entity set E_i. Each tuple [e_1, e_2,..., e_n] is a relationship instance. Each entity set in a relationship has a *role* or a function that the entity plays in the relationship. For example, a person entity in a relationship *works-for* has a role of an *employee* while the entity set company has a role of an *employer*.

Often role names are verbs and entity names are nouns which enables one to read the entity and relationships for example as "employee works-for employer".

Although we allow relationships among any number of entity sets, the most common cases are binary relationships between two entity sets. Most relationships discussed in this chapter are binary. The *degree* of a relationship is the number of entities associated in the relationship. Unary, binary and ternary relationships therefore have degree 1, 2 and 3 respectively.

We consider a binary relation R between two entity types E_1 and E_2. The relationship may be considered as two mappings E_1 -> E_2 and E_2 -> E_1. It is important to consider the constraints on these two mappings. It may be that one object from E_1 may be associated with exactly one object in E_2 or any number of objects in E_1 may be associated with any number of objects in E_2. Often the relationships are classified as one-to-one, one-to-many, or many-to-many. If every entity in E_1 is associated with at most one entity in E_2 and every entity in E_2 is associated with no more than one entity in entity set E_1, the relationship is one-to-one. For example, marriage is a one-to-one relationship (in most countries!) between an entity set *person* to itself. If an entity in E_1 may be associated with any number of entities in E_2, but an entity in E_2 can be associated with at most one entity in E_1, the relationship is called one-to-many. In a many-to-many relationship, an entity instance from one entity set may be related with any number of entity instances in the other entity set. We consider an example of entity sets employees and offices.

1. If for each employee there is at most one office and for each office there is at most one employee, the relationship is one-to-one.
2. If an office may accommodate more than one employee but an employee has at most one office, the relationship between office and employees is now one-to-many.
3. If an office may accommodate more than one staff and a staff member may be assigned more than one office the relationship is many-to-many. For example, an engineer may have one office in the workshop and another in the design office. Also each design office may accommodate more than one staff. The relationship is therefore many-to-many.

These three relationships are shown in Figures 2.2a, 2.2b and 2.2c. Figure 2.2a shows a one-to-one relationship between employees and offices.

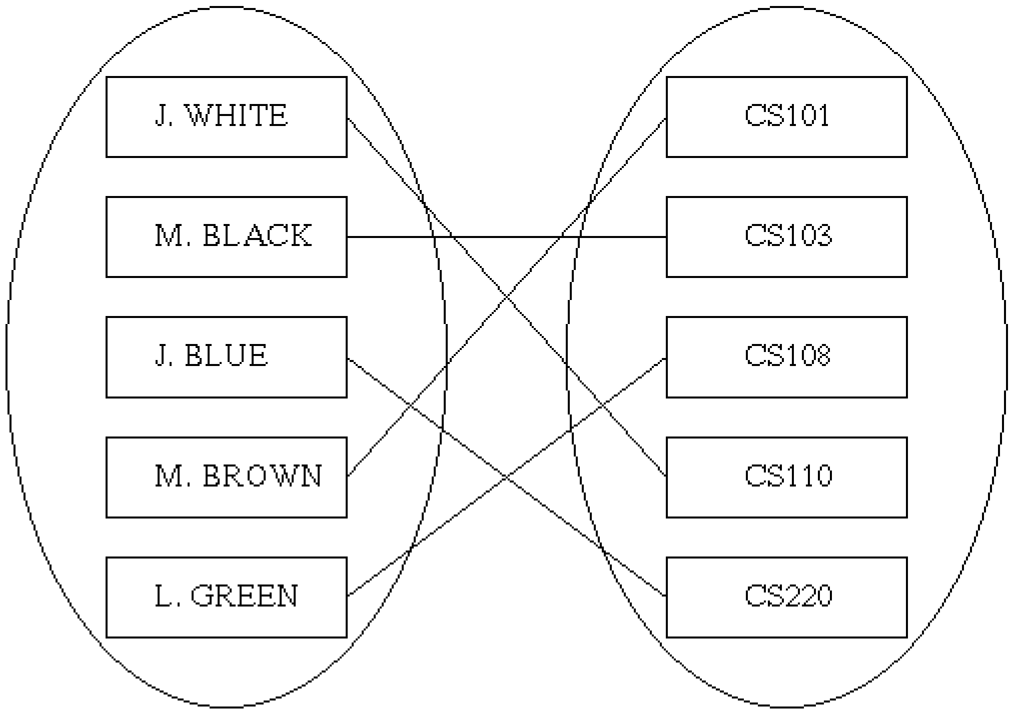

Figure 2.2a. One-to-one relationship Figure 2.2b shows a many-to-one relationship between employees and offices.

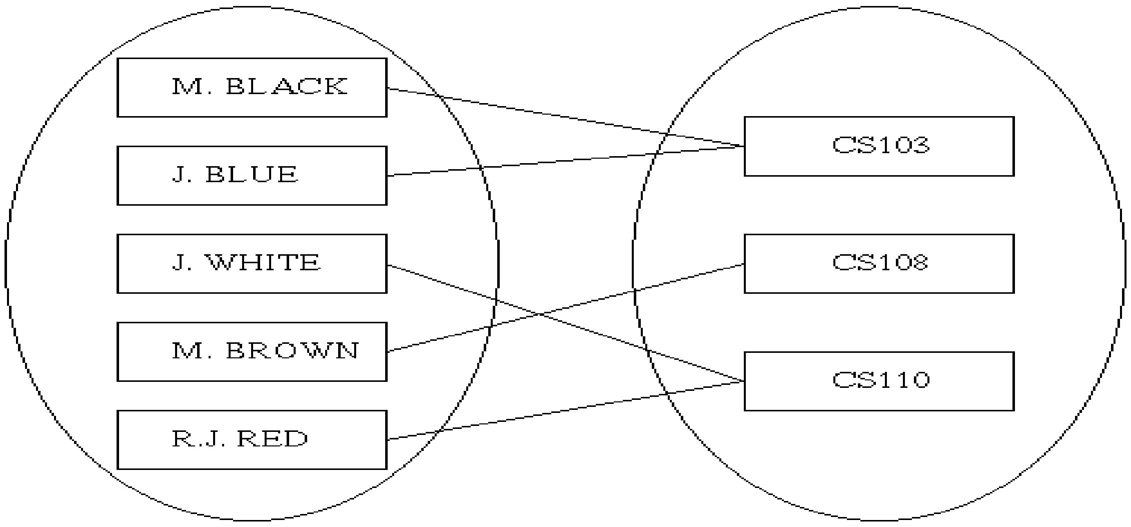

Figure 2.2b. Many-to-one relationship

Figure 2.2c shows a many-to-many relationship between employees and offices.

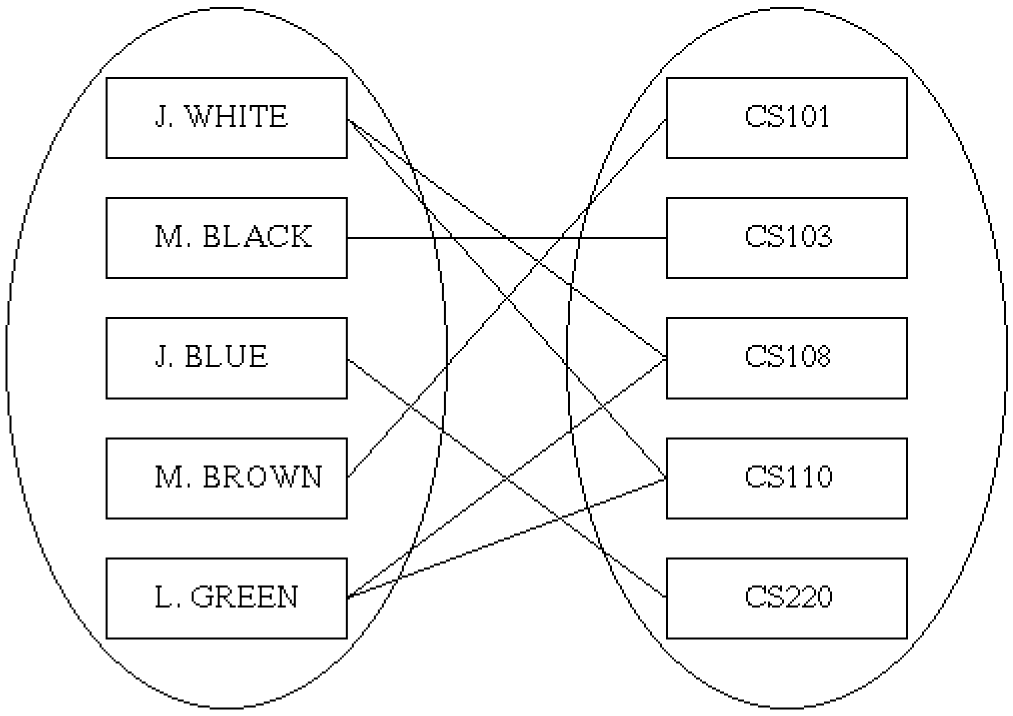

Figure 2.2c. Many-to-many relationship

As noted earlier, relationships are often binary but more than two entity types may participate in a relationship type. For example, a relationship may exist between entity type's *employee*, *project* and *company*.

<u>Attributes, Values and Value Sets</u>

As we have noted earlier, our database model consists of entities and relationships. Since these are the objects of interest, we must have some information about them that is of interest. The information of interest for each object is likely to be

Object name or identifier
object properties

values of the properties
time

Object name or identifier enables us to identify each object uniquely. Each object is described by properties and their values at some given time. Often however we are only interested in the current values of the properties and it is then convenient to drop time from the information being collected.

In some applications however, time is very important. For example, a personnel department would wish to have access to salary history and status history of all employees of the enterprise. A number of ways are available for storing such temporal data. Further details are beyond the scope of these notes.

Each entity instance in an entity set is described by a set of *attributes* that describe their qualities, characteristics or properties that are relevant and the values of the attributes. Relationships may also have relevant information about them. An employee in an *employee* entity set is likely to be described by its attributes like *employee number, name, date of birth*, etc. A relationship like *enrolment* might have attributes like the *date of enrolment*. A relationship *shipment* between suppliers and parts may have *date of shipment* and *quantity shipped* as attributes.

For each attribute there are a number of legal or permitted values. These set of legal values are sometimes called *value sets* or *domain* of that attribute. For example, employee number may have legal values only between 10000 to 99999 while the year value in date of birth may have admissible values only between year 1900 and 1975. A number of attributes may share a common domain of values. For example, employee number, salary and department number may all be five digit integers.

An attribute may be considered as a function that maps from an entity set or a relationship set into a value set. Sometime an attribute may map into a Cartesian product of value sets. For example, the attribute *name* may map into value sets *first name*, *middle initial* and *last name*. The set of (attribute, data value) pairs, one pair for each attribute, define each entity instance. For example, an entity instance of entity set Employee may be described by

1. (employee number, 101122)
2. (employee name, John Smith)
3. (employee address, 2 Main Street Townsville Q4812)
4. (telephone, 456789)
5. (salary, 44,567)

In this chapter, we will discuss the following entity sets:

1. *EMPLOYEE* - the set of all employees at a company. The attributes of Employees are: name, employee number, address, telephone number, salary.
2. *DEPT* - the set of all departments at the company. The attributes are: department number, department name, supervisor number.
3. *PROJECTS* - the set of all projects at the company. The attributes are: project number, project name, project manager number.

Representing Entities and Relationships

It is essential that we be able to identify each entity instance in every entity set and each relationship instance in every relationship set. Since entities are represented by the values of their attribute set, it is necessary that the set of attribute values be different for each entity instance. Sometimes an artificial attribute may need to be included in the attribute set to simplify entity identification (for example, although each instance of the entity set *student* can be identified by the values of its attributes *student name*, *address* and *date of birth*, it is convenient to include an artificial attribute student number to make identification of each entity instance easier)

A group of attributes (possibly one) used for identifying entities in an entity set is called an *entity key*. For example, for the entity set *student*, *student number* is an entity key and so is (*student name, address, date of birth*). Of course, if k is a key then so must be each superset of k. To reduce the number of possible keys, it is usually required that a key be *minimal* in that no subset of the key should be key. For example, *student name* by itself could not be considered an entity key since two students could have the same name. When several minimal keys exist (such keys are often called *candidate keys*), any semantically meaningful key is chosen as the *entity primary key*.

Similarly each relationship instance in a relationship set needs to be identified. This identification is always based on the primary keys of the entity sets involved in the relationship set. The primary key of a relationship set is the combination of the attributes that form the primary keys of the entity sets involved. In addition to the entity identifiers, the relationship key also (perhaps implicitly) identifies the role that each entity plays in the relationship.

Let *employee number* be the primary key of an entity set EMPLOYEE and *company name* be the primary key of COMPANY. The primary key of relationship set WORKS-FOR is then (*employee number*, *company name*). The role of the two entities is implied by the order in which the two primary keys are specified.

In certain cases, the entities in an entity set cannot be uniquely identified by the values of their own attributes. For example, children of an employee in a company may have names that are not unique since a child of one employee is quite likely to have name that is the same as the name of a child of some other employee. One solution to this problem is to assign unique numbers to all children of employees in the company. Another, more natural, solution is to identify each child by his/her name *and* the primary key of the parent who is an employee of the company. We expect names of the children of an employee to be unique. Such attribute(s) that discriminates between all the entities that are dependent on the same parent entity is sometime called a *discriminator*; it cannot be called a key since it does not uniquely identify an entity without the use of the relationship with the employee. Similarly history of employment (Position, Department) would not have a primary key without the support of the employee primary key. Such entities that require a relationship to be used in identifying them are called *weak entities*. Entities that have primary keys are called *strong or regular entities*. Similarly, a relationship may be weak or strong (or regular). A *strong relationship* is between entities each of which is strong; otherwise the relationship is a *weak relationship*. For example, any relationship between the children of employees and the schools they attend would be a weak relationship. A relationship between employee entity set and the employer entity set is a strong relationship.

A weak entity is also called *subordinate* entity since its existence depends on another entity (called the *dominant* entity). This is called *existence dependence*. A weak entity may also be *ID dependent* on the dominant entity, although existence dependency does not imply ID dependency. If a weak entity is ID dependent, the primary key of the weak entity is the primary key of its dominant entity plus a set of attributes of the weak entity that can act as a discriminator within the weak entity set. This method of identifying entities by relationships with other entities can be applied recursively until a strong entity is reached although the relationship between dominant entity and a weak entity is usually one-to-one, in some situations the relationship may be many-to-many. For example, a company may employ both parents of some children. The dependence is then many-to-many.

We note several terms that are useful:

1. Weak entity relation - a relation that is used for identifying entities, for example, relationship between employees and their dependents.
2. Regular entity relation - a relation not used for identifying entities.
3. Similarly regular relationship relation and weak relationship relations.

<u>The ER Diagrams</u>

As noted earlier, one of the aims of building an entity-relationship diagram is to facilitate communication between the database designer and the end user during the requirements analysis. To facilitate such communication, the designer needs adequate communication tools. Entity-Relationship diagrams are such tools that enable the database designer to display the overall database view of the enterprise (the enterprise conceptual schema).

An E-R diagram naturally consists of a collection of entity sets and relationship sets and their associations. A diagram may also show the attributes and value sets that are needed to describe the entity sets and the relationship sets in the ERD. In an ERD, as shown in Figure 2.4, entity sets are represented by rectangular shaped boxes. Relationships are represented by diamond shaped boxes.

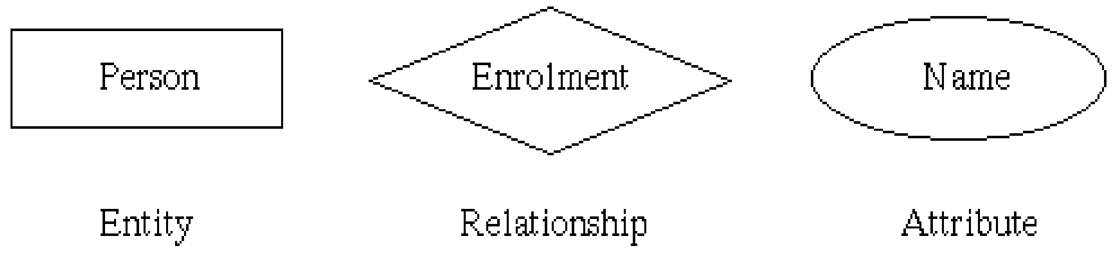

Figure 2.3

Ellipses are used to represent attributes and lines are used to link attributes to entity sets and entity sets to relationship sets.

Consider the following E-R diagram.

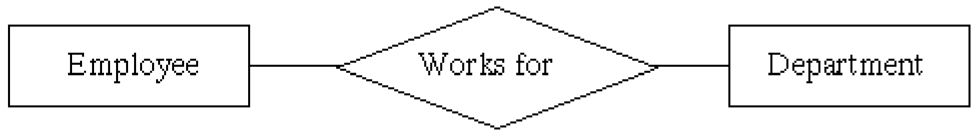

Figure 2.4a A simple E-R diagram. The following E-R diagram gives the attributes as well.

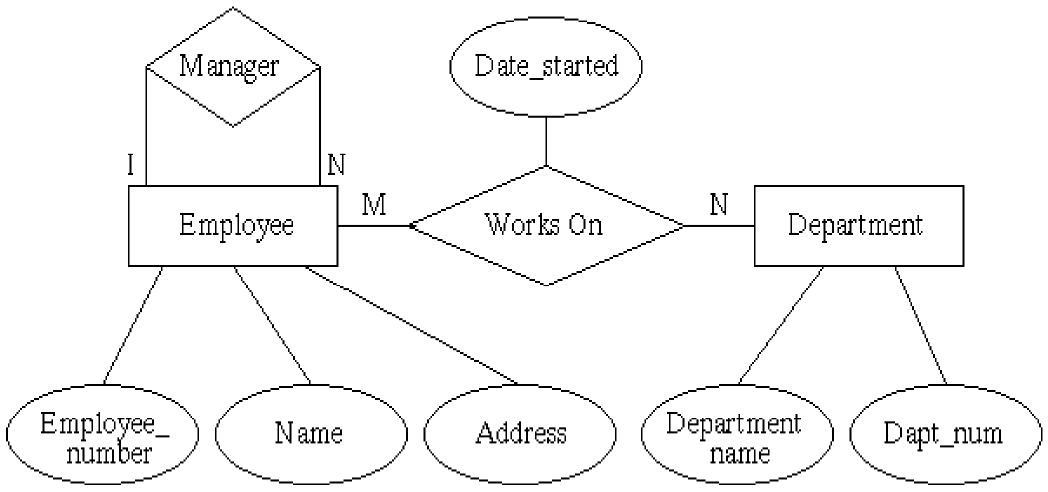

Figure 2.4b An E-R diagram with attributes.

The following E-R diagram represents a more complex model that includes a weak entity.

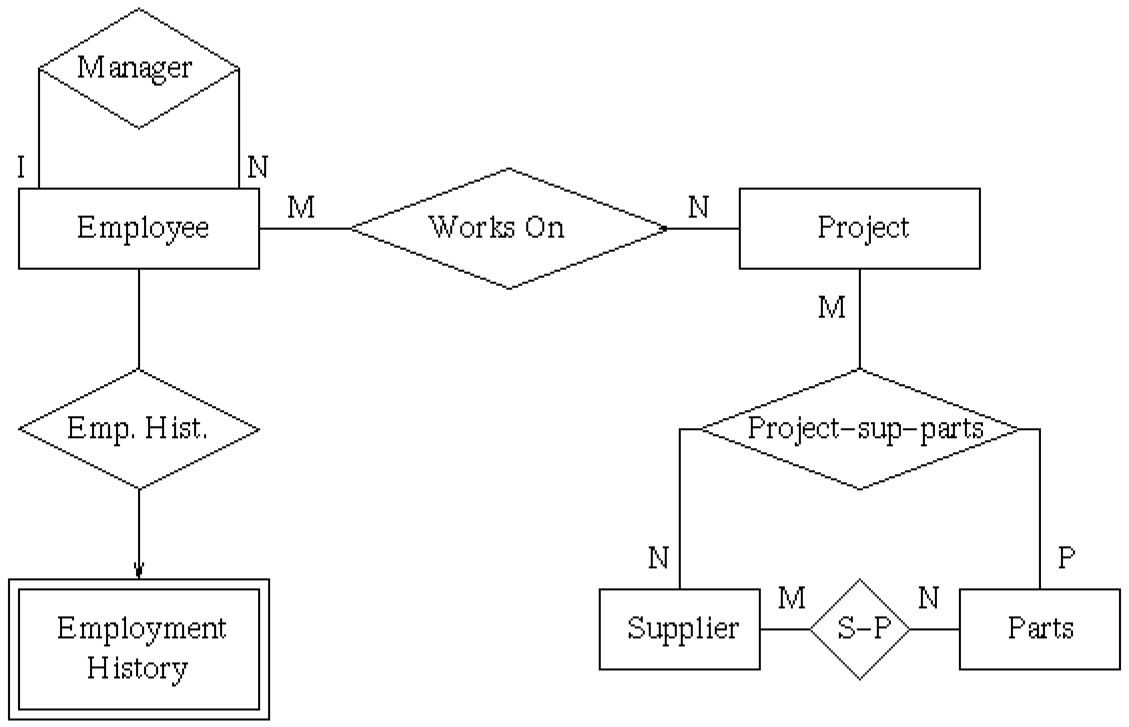

Figure 2.4c An E-R diagram showing a weak entity. The following E-R diagram represents more than one relationship between the same two entities.

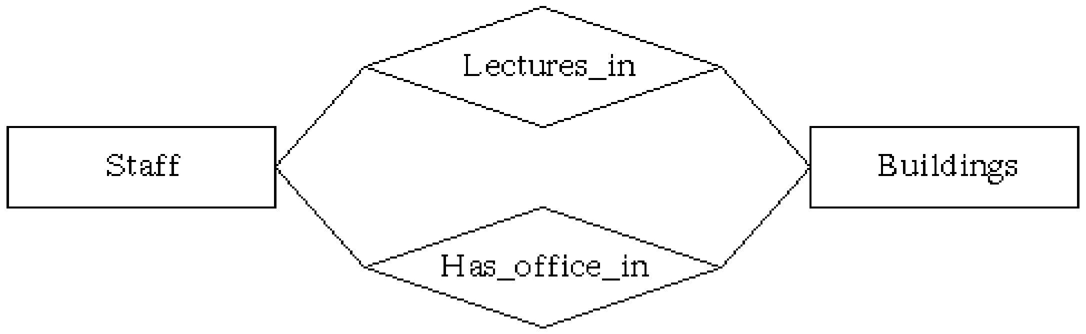

Figure 2.4d More than one relationship between two entities. The diagram below shows unary relationships.

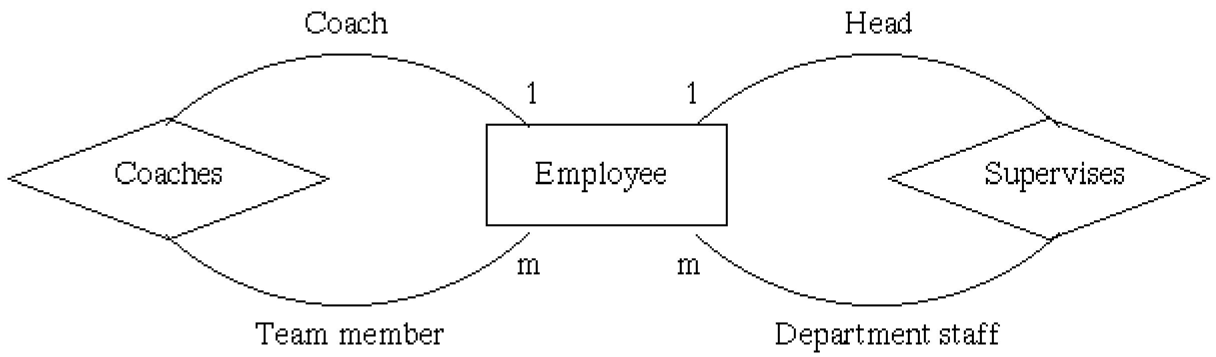

Figure 2.4e Two unary relationships.

Figures 2.4a to 2.4e show a variety of E-R diagrams. Figure 2.4a is an E-R diagram that only shows two entities and one relationship and not the attributes. Figure 2.4b shows the same two entities but shows an additional relationship between employees and their manager. This additional relationship is between the entity set *Employee* to itself. In addition, the E-R diagram in Figure 2.4b shows the attributes as well as the type of relationships (1:1, m:n or 1:m). The type of relationship is shown by placing a label of 1, m or n on each end of every arc. The label 1 indicates that only one of those entity instances may participate in a given relationship. A letter label (often m or n) indicates that a number of these entity instances may participate in a given relationship. The E-R diagrams may also show roles as labels on the arcs.

The E-R diagram in Figure 2.4c shows a number of entities and a number of relationships. Also note that the entity employment history that is shown as double box. A double box indicates that *Employment History* is a weak entity and that its existence depends on existence of corresponding employee entity. The existence dependency of employment history entity on employee entity is indicated by an arrow. A relationship between three entities is shown. Note that each entity may participate in several relationships.

The E-R diagram in Figure 2.4d shows two entities and two different relationships between the same two entities. Figure 2.4e shows two relationships also but these both relationships are between the entity *Employee* to itself.

Entity Type Hierarchy

Although entity instances of one entity type are supposed to be objects of the same type, it often happens that objects of one entity type do have some differences. For example, a company might have vehicles that could be considered an entity type. The vehicles could however include cars, trucks and buses and it may then be necessary to include capacity of the bus for buses and the load capacity of the trucks for trucks, information that is not relevant for cars. In such situations, it may be necessary to deal with the subsets separately while maintaining the view that all cars, trucks and buses are vehicles and share a lot of information.

Entity hierarchies are known by a number of different names. At least the following names are used in the literature:

1. Supertypes and subtypes
2. Generalization hierarchies
3. ISA hierarchies

We will not discuss entity hierarchies any further although use of hierarchies is now recognized to be important in conceptual modeling.

Guidelines for Building an ERM

We have so far discussed the basics of the E-R models and the representation of the model as an E-R diagram. Although the E-R model approach is often simple, a number of problems can arise. We discuss some of these problems and provide guidelines that should assist in the modeling process.

Choosing Entities

As noted earlier, an entity is an object that is of interest. It is however not always easy to decide when a given thing should be considered an entity. For example, in a supplier-part database, one may have the following information about each supplier

Supplier number
supplier name
supplier rating
supplier location (i.e. city)

It is clear that supplier is an entity but one must now make a decision whether city is an entity or an attribute of entity supplier. The rule of thumb that should be used in such situations is to ask the question "Is city as an object of interest to us?". If the answer is yes, we must have some more information about each city than just the city name and then city should be considered an entity. If however we are only interested in the city name, city should be considered an attribute of supplier.

As a guideline therefore, each entity should contain information about its properties. If an object has no information other than its identifier that interests us, the object should be an attribute.

Multivalued Attributes

If an attribute of an entity can have more than one value, the attribute should be considered an entity. Although conceptually multi-value attributes create no difficulties, problems arise when the E-R model is translated for implementation using a DBMS. Although we have indicated above that an entity should normally contain information about its properties, a multi-value attribute that has no properties other than its value should be considered an entity.

Database Design Process

When using the E-R model approach to database design, one possible approach is to follow the major steps that are listed below:

1. Study the description of the application.
2. Identify entity sets that are of interest to the application.
3. Identify relationship sets that are of interest to the application. Determine whether relationships are 1:1, 1: n or m: n.
4. Draw an entity-relationship diagram for the application.
5. Identify value sets and attributes for the entity sets and the relationship sets.
6. Identify primary keys for entity sets.
7. Check that ERD conforms to the description of the application.
8. Translate the ERD to the database model used by the DBMS.

It should be noted that database design is an iterative process.

Rigorous definition Of entity

Although we have discussed the concept of an entity, we have not presented a rigorous definition of an entity. A simple rigorous definition is not possible since there is no absolute distinction between entity types and attributes. Usually an attribute exists only as related to an entity type but in some contexts, an attribute can be viewed as an entity.

Further complications arise because an entity must finally be translated to relation. Since relations have a somewhat inflexible structure, an entity itself must satisfy several artificial constraints. For example, problems arise when things that we consider entities have attributes that are mutli-valued. The attribute then needs to be declared an entity to avoid problems that will appear in mapping an entity with multi-values attribute to the relational database.

Consider for example, an entity called *vendor*. The vendor has several attributes including vendor number, vendor name, location, telephone number etc. Normally the location will be considered an attribute but should we need to cater for a vendor with several branches, the location would need to be made an entity and a new relationship *located-in* would be needed.

To overcome some of the above problems, rule of thumbs like the following should be followed:

1. Entities have descriptive information; identifying attributes do not.
2. Multivalued attributes should be classed as entities.
3. Make an attribute that has a many-to-one relationship with an entity.
4. Attach attributes to entities that they describe most directly.
5. Avoid composite identifiers as much as possible.

Also there is no absolute distinction between an entity type and a relationship type although a relationship is usually regarded as unable to exist on its own. For example, an enrolment cannot be expected to exist on its own without the entities students and subjects.

A careful reader would have noticed our reluctance to discuss time in our data models. We have assumed that only current attribute values are of interest. For example, when an employee's salary changes, the last salary value disappears for ever. In many applications this situation would not be satisfactory. One way to overcome such problem would be to have an entity for *salary history* but this is not always the most satisfactory solution. Temporal data models deal with the problem of modeling time dependent data.

The Facts-based View (of W.Kent) !!

If we study what we usually store in records we find that fields are character strings that usually represent facts. Any field by itself conveys little useful information. For example, a field may convey a name or a fact like department number but most useful information is conveyed only when the interconnections between fields are also conveyed. Information therefore is expressed not only by facts represented in fields but more importantly by relations among fields. Records tie information together; they represent aggregation of facts. Records may be considered to have three components:

1. What is each field about i.e. the entity type.
2. How each field refers to the entity representation

3. What information each field conveys i.e. a relationship or a fact about the entity.

The facts based view is based on the premise that rather than choosing entities as clustering points and aggregating facts around them we use a design based on aggregating single-valued related facts together.

Kent suggests the following as basis for facts based view of database design:

1. Single-valued facts about things are maintained in records having that thing's identifier.
2. Several single-valued facts about the same thing can be maintained in the same record.
3. All other kinds of facts (e.g. multi-valued) are maintained in separate records, one record for each fact.

The following methodology may be followed for fact-based database design:

1. Identify the facts to be maintained. Identify which are single-valued.
2. Generate a pseudo-record for each fact.
3. Identify "pseudo-keys" for each pseudo-record, based in single-valuedness of facts.
4. Merge pseudo-records having compatible keys
5. Assign representations
6. Consider alternative designs
7. Name the fields and record types

Extended E-R Model

When the E-R model is used as a conceptual schema representation, difficulties may arise because of certain inadequacies of the initial E-R model constructs. For example, view integration often requires abstraction concepts such as generalization. Also data integrity involving null attribute values requires specification of structural constraints on relationships. To deal with these problems the extended E-R model includes the following extensions:

(1) A concept of "Category" is introduced to represent generalization hierarchies and subclasses

(2) Structural constraints on relationships are used to specify how entities may participate in relationships.

A category is a subset of entities from an entity set. For example, a manager, secretary and technician may be categories of entity set Employee.

Employee

Manager Secretary Technician

The categories share most attributes of the entity type whose subset they are, some attributes may not be shared. For example, department may apply to Manager and Secretary and not to technician.

Further extensions have been suggested. For example, Chen has suggested that a concept of "composite entity" be added to the E-R model. A composite entity is an entity formed by other entities like a relationship. It differs from relationship in that a relationship cannot have attributes (descriptors) while a composite entity can.

Summary

The entity-relationship model is useful because, as we will soon see, it facilitates communication between the database designer and the end user during the requirements analysis. An E-R diagram naturally consists of a collection of entity sets and relationship sets and their associations. **An *entity* is a person, place, or a thing or an object or an event which can be distinctly identified and is of interest. *Relationships* are associations or connections that exist between entities and may be looked at as mappings between two or more entity sets.**

Questions

1. What do you mean by entity relationship model?
2. Explain Entity set and relationship set?
3. Explain entity type?
4. Explain extended E-R Model?

Activities

An Example

Consider building a data model for a University. The information available includes:

1. Students with associated information (student number, name, address, telephone number)
2. Academic staff with associated information (staff number, name, address, telephone number, salary, department, office number)
3. Courses with associated information (course number, course name, department name, semester offered, lecture times, tutorial times, room numbers, etc.)
4. Students take courses and are awarded marks in courses completed.
5. Academic staffs are associated with courses. It changes from year to year.

References

1. P. P. Chen (1976), "The entity-relationship model: Towards a unified view of data", ACM Trans. Database Systems, Vol. 1, pp. 9-36.
2. P. P. Chen (1977), "Entity Relationship Approach to Logical Database Design", Monograph Series, QED Information Sciences Inc., Wellesley, Ma.
3. P. P. Chen (1980), Ed., "Entity-Relationship to Systems Analysis and Design", North-Holland, New York/Amsterdam.
4. W. Kent (1986), "The Realities of Data: Basic Properties of Data Reconsidered", in *Database Semantics (DS-1)*, T.B.Steel and R.Meersman, Eds., North-Holland, pp. 175-188.

5. Ng, P. (1981), "Further Analysis of the Entity-Relationship Approach to Database Design", IEEE Trans. on Soft. Engg., Vol. SE-7, No. 1, 1981, pp. 85-98.
6. T. J. Theorey, D. Yang and J. P. Fry (1986), "Logical Design Methodology for Relational Databases", ACM Computing Surveys, June 1986, pp. 197-222.
7. Yao, S. B. (1985), "Principles of Database Design, Vol. 1, Logical Organizations", Prentice-Hall.