# Lecture No. 07

## Reading Material

| | |
|---|---|
| "Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan. | Section: 5.2 5.3 |
| Hoffer | Page: 85 - 95 |

## Overview of Lecture

- Entity
- Different types of Entities
- Attribute and its different types
- In the previous lecture we discussed the importance and need of data models. From this lecture we are going to start detailed discussion on a data model, which is the entity relationship data model also known as E-R data model.

## Entity-Relationship Data Model

It is a semantic data model that is used for the graphical representation of the conceptual database design. We have discussed in the previous lecture that semantic data models provide more constructs that is why a database design in a semantic data model can contain/represent more details. With a semantic data model, it becomes easier to design the database, at the first place, and secondly it is easier to understand later. We also know that conceptual database is our first comprehensive design. It is independent of any particular implementation of the database, that is, the conceptual database design expressed in E-R data model can be implemented using any DBMS. For that we will have to transform the conceptual database design from E-R data model to the data model of the particular DBMS. There is no DBMS based on the E-R data model, so we have to transform the conceptual database design anyway.

A question arises from the discussion in the previous paragraph, can we avoid this transformation process by designing our database directly using the data model of our selected DBMS. The answer is, yes we can but we do not do it, because most commercial DBMS are based on the record-based data models, like Hierarchical, Network or Relational. These data models do not provide too much constructs, so a database design

in these data models is not so expressive. Conceptual database design acts as a reference for many different purposes. Developing it in a semantic data model makes it much more expressive and easier to understand, that is why we first develop our conceptual database design in E-R data model and then later transform it into the data model of our DBMS.

**Constructs in E-R Data Model**
The E-R data model supports following major constructs:

- Entity
- Attribute
- Relationship

We are going to discuss each one of them in detail.

## The Entity

Entity is basic building block of the E-R data model. The term entity is used in three different meanings or for three different terms and that are:

- Entity type
- Entity instance
- Entity set

In this course we will be using the precise term most of the time. However after knowing the meanings of these three terms it will not be difficult to judge from the context which particular meaning the term entity is being used in.

**Entity Type**
The entity type can be defined as a name/label assigned to items/objects that exist in an environment and that have similar properties. It could be person, place, event or even concept, that is, an entity type can be defined for physical as well as not-physical things. An entity type is distinguishable from other entity types on the basis of properties and the same thing provides the basis for the identification of an entity type. We analyze the things existing in any environment or place. We can identify or associate certain properties with each of the existing in that environment. Now the things that have common or similar properties are candidates of belonging to same group, if we assign a name to that group then we say that we have identified an entity type.

Generally, the entity types and their distinguishing properties are established by nature, by very existence of the things. For example, a bulb is an electric accessory, a cricket bat is a sports item, a computer is an electronic device, a shirt is a clothing item etc. So identification of entity types is guided by very nature of the things and then items having properties associated with an entity type are considered to be belonging to that entity type or instances of that entity type. However, many times the grouping of things in an environment is dictated by the specific interest of the organization or system that may supersede the natural classification of entity types. For example, in an organization, entity

types may be identified as donated items, purchased items, manufactured items; then the items of varying nature may belong to these entity types, like air conditioners, tables, frying pan, shoes, car; all these items are quite different from each other by their respective nature, still they may be considered the instances of the same entity type since they are all donated or purchased or manufactured.

What particular properties of an entity type should be considered or which particular properties jointly form an entity type? The answer to this question we have discussed in detail in our very first lecture, where we were discussing the definition of database. That is, the perspective or point of view of the organization and the system for which we are developing the database is going to guide us about the properties of interest for a particular group of things. For example, if you have a look around you in your bedroom, you might see tube light, a bulb, fan, air conditioner, carpet, bed, chair and other things. Now fan is an item that exists in your room, what properties of the fan we are interest in, because there could be so many different properties of the fan. If we are developing the database for a manufacturer, then we may be interested in type of material used for wings, then the thickness of the copper wire in the coil, is it locally manufactured or bought ready made, what individual item costs, what is the labor cost, what is the total cost, overhead, profit margin, net price etc. But if we are working for a shopkeeper he might be interested in the name of the company, dealer price, retail price, weight, color of fan etc. From the user perspective; company name, color, price, warranty, name of the dealer, purchase date and alike. So the perspective helps/guides the designer to associate or identify properties of things in an environment.

The process of identifying entity types, their properties and relationships between them is called abstraction. The abstraction process is also supported by the requirements gathered during initial study phase. For example, the external entities that we use in the DFDs provide us a platform to identify/locate the entity types from. Similarly, if we have created different cross reference matrices, they help us to identify different properties of the things that are of interest in this particular system and that we should the data about. Anyway, entity types are identified through abstraction process, then the items possessing the properties associated with a particular entity type are said to be belonging to that entity type or instances of that entity type.

While designing a system, you will find that most of the entity types are same as are the external entities that you identified for the DFDs. Sometimes they may be exactly the same. Technically, there is a minor difference between the two and that is evident from their definitions. Anything that receives or generates data from or to the system is an external entity, where as entity type is name assigned to a collection of properties of different things existing in an environment. Anything that receives or generates data is considered as external entity and is represented in the DFD, even if it is a single thing. On the other hand, things with a single instance are assumed to be on hand in the environment and they are not explicitly identified as entity type, so they are not represented in the E-R diagram. For example, a librarian is a single instance in a library system, (s)he plays certain role in the library system and at many places data is generated

from or to the librarian, so it will be represented at relevant places in the DFDs. But the librarian will not be explicitly represented in the E-R diagram of the library system and its existence or role is assumed to be there and generally it is hard-coded in the application programs.

**Entity Instance**
A particular object belonging to a particular entity type and how does an item becomes an instance of or belongs to an entity type? By possessing the defining properties associated with an entity type. For example, following table lists the entity types and their defining properties:

| Entity Types | Properties | Instances |
|---|---|---|
| EMPLOYEE | Human being, has name, has father name, has a registration number, has qualification, designation | M. Sharif, Sh. Akmal and many others |
| FURNITURE | Used to sit or work on, different material, having legs, cost, purchased | Chair, table etc. |
| ELECTRIC APPLIANCES | Need electricity to work, purchased | Bulb, fan, AC |
| OFFICE EQUIPMENT | Used for office work, consumable or non-consumable, | Papers, pencil, paper weight etc. |

Table 1: Entity types, their properties and instances

Each entity instance possesses certain values against the properties with the entity type to which it belongs. For example, in the above table we have identified that entity type EMPLOYEE has name, father name, registration number, qualification, designation. Now an instance of this entity type will have values against each of these properties, like (M. Sajjad, Abdul Rehman, EN-14289, BCS, and Programmer) may be one instance of entity type EMPLOYEE. There could be many others.

**Entity Set**
A group of entity instances of a particular entity type is called an entity set. For example, all employees of an organization form an entity set. Like all students, all courses, all of them form entity set of different entity types

As has been mentioned before that the term entity is used for all of the three terms mentioned above, and it is not wrong. Most of the time it is used to mention an entity type, next it is used for an entity instance and least times for entity set. We will be precise most of the time, but if otherwise you can judge the particular meaning from the context.

## Classification of entity types

Entity types (ETs) can be classified into regular ETs or weak ETs. Regular ETs are also called strong or independent ETs, whereas weak ETs are also called dependent ETs. In the following we discuss them in detail.

**Weak Entity Types**
An entity type whose instances cannot exist without being linked with instances of some other entity type, i.e., they cannot exist independently. For example, in an organization we want to maintain data about the vehicles owned by the employees. Now a particular vehicle can exist in this organization only if the owner already exists there as employee. Similarly, if employee leaves the job and the organization decides to delete the record of the employee then the record of the vehicle will also be deleted since it cannot exist without being linked to an instance of employee.

**Strong Entity Type**
An entity type whose instances can exist independently, that is, without being linked to the instances of any other entity type is called strong entity type. A major property of the strong entity types is that they have their own identification, which is not always the case with weak entity types. For example, employee in the previous example is an independent or strong entity type, since its instances can exist independently.

**Naming Entity Types**
Following are some recommendations for naming entity types. But they are just recommendations; practices considered good in general. If one, some or all of them are ignored in a design, the design will still be valid if it satisfies the requirements otherwise, but good designs usually follow these practices:

- Singular noun recommended, but still plurals can also be used
- Organization specific names, like customer, client, gahak anything will work
- Write in capitals, yes, this is something that is generally followed, otherwise will also work.
- Abbreviations can be used, be consistent. Avoid using confusing abbreviations, if they are confusing for others today, tomorrow they will confuse you too.

**Symbols for Entity Types**
A rectangle is used to represent an entity type in E-R data model. For strong entity types rectangle with a single line is used whereas double lined rectangle is drawn to represent a weak entity type as is shown below:

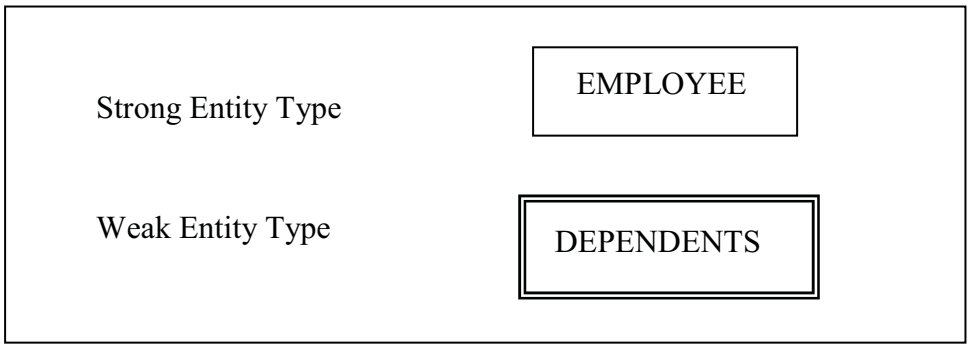



Figure 1: Symbols used for entity types

We have discussed different types of entity types; in the next section we are going to discuss another component or E-R data model, that is, the attribute.

## Attribute

An attribute of an entity type is a defining property or quality of the instances of that entity type. Entity instances of same entity type have the same attributes. (E.g. Student Identification, Student Name). However, values of these attributes may be same or different. For example, all instances of the entity type STUDENT may have the attributes name, father name, age; but the values against each of these attributes for each instance may be different. Like, one instance may have the values (M. Hafeez, Noor Muhammad, 37) other may have others. Remember one thing, that the values of the attributes may be same among different entity instances. The thing to remember at this stage is that attributes are associated with an entity type and those attributes then become applicable /valid for all the instances of that entity type and instances have values against these attributes.

An attribute is identified by a name allocated to it and that has to be unique with respect to that entity type. It means one entity type cannot have two attributes with the same name. However, different entity types may have attributes with the same name. The guidelines for naming an attribute are similar to those of entity types. However, one difference is regarding writing the names of attributes. The notation that has been adopted in this course is that attribute name generally consists of two parts. The name is started in lower case, and usually consists of abbreviation of the entity types to which the attribute belongs. Second part of the attribute name describes the purpose of attribute and only first letter is capitalized. For example empName means name attribute of entity type EMPLOYEE, stAdrs means address attribute of the entity type STUDENT and alike. Others follow other notations, there is no restriction as such, and you can follow anyone that you feel convenient with. BUT be consistent.

**Domain of an Attribute**

We have discussed in the previous section that every attribute has got a name. Next thing is that a domain is also associated with an attribute. These two things, name and the domain, are part of the definitions of an attribute and we must provide them. Domain is the set of possible values that an attribute can have, that is, we specify a set of values either in the form of a range or some discrete values, and then attribute can have value out of those values. Domain is a form of a check or a constraint on attribute that it cannot have a value outside this set.

Associating domain with an attribute helps in maintaining the integrity of the database, since only legal values could be assigned to an attribute. Legal values mean the values that an attribute can have in an environment or system. For example, if we define a salary attribute of EMPLOYEE entity type to hold the salary of employees, the value assigned to this attribute should be numeric, it should not be assigned a value like 'Reema', or '10/10/2004', why, because they are not legal salary values[1]. It should be numeric. Further, even if we have declared it as numeric it will have numeric values, but about a value like 10000000000. This is a numeric value, but is it a legal salary value within an organization? You have to ask them. It means not only you will specify that the value of salary will be numeric but also associate a range, a lower and upper limit. It reduces the chances of mistake.

Domain is normally defined in form of data type and some additional constraints like the range constraint. Data type is defined as a set of values along with the operations that can be performed on those values. Some common data types are Integer, Float, Varchar, Char, String, etc. So domain associates certain possible values with an attribute and certain operations that can be performed on the values of the attribute. Another important thing that needs to be mentioned here is that once we associate a domain to an attribute, all the attributes in all entity instances of that entity type will have the values from the same domain. For example, it is not possible that in one entity instance the attribute salary has a value 15325.45 and in another instance the same attribute has a value 'Reema'. No. All attribute will have values from same domain, values may be different or same, whatever, but the domain will be the same.

[1] Sometimes when some coding has been adopted, then such strange values may be legal but here we are discussing the general conditions

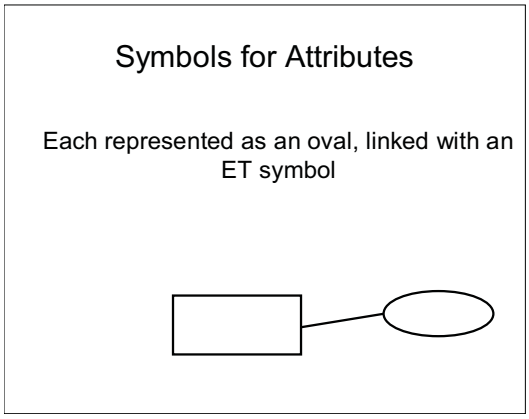



Figure 2: Symbol used for attribute in E-R diagram

## Types of Attributes

Attributes may be of different types. They may be:

- Simple or Composite
- Single valued or multi-valued
- Stored or Derived

**Simple or Composite Attributes:**
An attribute that is a single whole is a simple attribute. The value of a simple attribute is considered as a whole, not as comprising of other attributes or components. For example, attributes stName, stFatherName, stDateOfBorth of an entity type STUDENT are example of simple attributes. On the other hand if an attribute consists of collection of other simple or composite attributes then it is called a composite attributes. For example, stAdres attribute may comprise of houseNo, streetNo, areaCode, city etc. In this case stAdres will be a composite attribute.

**Single valued or multi-valued Attributes:**
Some attribute have single value at a time, whereas some others may have multiple values. For example, hobby attribute of STUDENT or skills attribute of EMPLOYEE, since a student may have multiple hobbies, likewise an employee may have multiple skills so they are multi-valued attributes. On the other hand, name, father name, designation are generally single valued attributes.

**Stored or Derived Attributes:**
Normally attributes are stored attributes, that is, their values are stored and accessed as such from the database. However, sometimes attributes' values are not stored as such, rather they are computed or derived based on some other value. This other value may be stored in the database or obtained some other way. For example, we may store the name, father name, address of employees, but age can be computed from date of birth. The

advantage of declaring age as derived attribute is that whenever we will access the age, we will get the accurate, current age of employee since it will be computed right at the time when it is being accessed.

How a particular attribute is stored or defined, it is decided first by the environment and then it has to be designer's decision; your decision. Because, the organization or system will not object rather they will not even know the form in which you have defined an attribute. You have to make sure that the system works properly, it fulfills the requirement; after that you do it as per your convenience and in an efficient way.

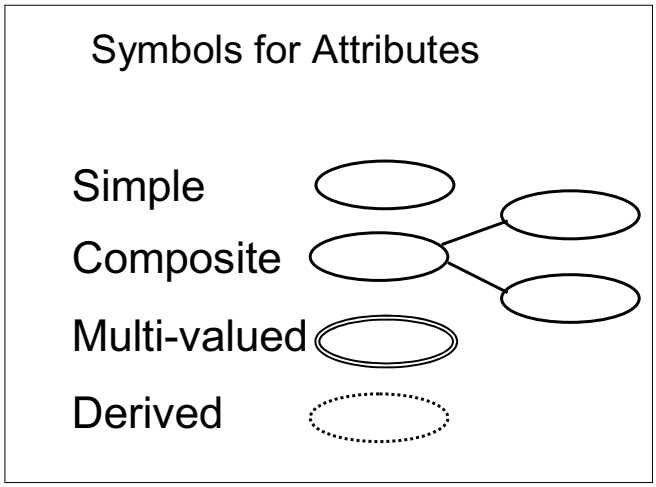



Figure 3: Symbol used for different types of attributes in E-R diagram

An example diagram representing all types of attributes is given below:

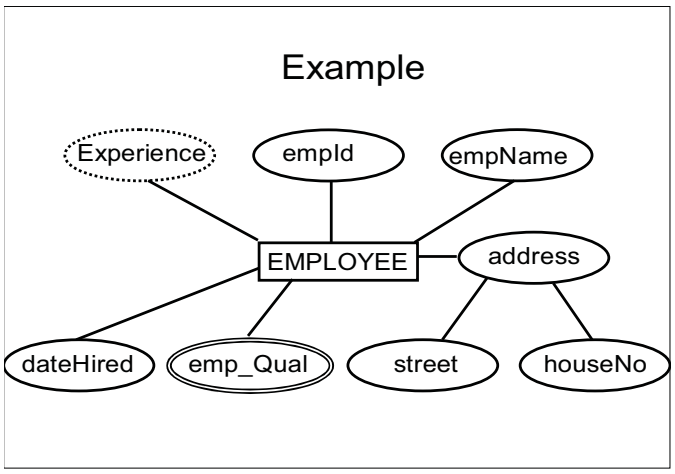



Figure 4: Example entity type with attributes of different types
This concludes this lecture.

## Summary:

In this lecture we have discussed entity and attribute. We discussed that there are three different notions for which the term entity is used and we looked into these three terms in detail. They are entity type, entity instance and entity set. An entity type is name or label assigned to items or objects existing in an environment and having same or similar property. An entity instance is a particular item or instance that belongs to a particular entity type and a collection of entity instances is called an entity set. We also discussed in this lecture the attribute component of the E-R data model and its different types. The third component the E-R data model, that is, the relationship will be discussed in the next lecture.

## Exercises:

- Take a look into the system where you work or study or live, identify different entity types in that environment. Associate different types of attributes with these entity types.
- Look at the same environment from different possible perspectives and realize the difference that the change of perspective makes in the abstraction process that results in establishing different entity types or/and their different properties.

# Lecture No. 08

## Reading Material

| | |
|---|---|
| "Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan. | Section 5.4 |

## Overview of Lecture

- Concept of Key and its importance
- Different types of keys

## Attributes

**Def 1:**

An attribute is any detail that serves to identify, qualify, classify, quantify, or otherwise express the state of an entity occurrence or a relationship.

**Def 2**:

Attributes are data objects that either identify or describe entities.

Identifying entity type and then assigning attributes or other way round; it's an "egg or hen" first problem. It works both ways; differently for different people. It is possible that we first identify an entity type, and then we describe it in real terms, or through its attributes keeping in view the requirements of different users' groups. Or, it could be other way round; we enlist the attribute included in different users' requirements and then group different attributes to establish entity types. Attributes are specific pieces of information, which need to be known or held. An attribute is either required or optional. When it's required, we must have a value for it, a value must be known for each entity occurrence. When it's optional, we could have a value for it, a value may be known for each entity occurrence.

## The Keys

Attributes act as differentiating agents among different entity types, that is, the differences between entity types must be expressed in terms of attributes. An entity type can have many instances; each instance has got a certain value against each attribute defined as part of that particular entity type. A *key* is a set of attributes that can be used to

identify or access a particular entity instance of an entity type (or out of an entity set). The concept of key is beautiful and very useful; why and how. An entity type may have many instances, from a few to several thousands and even more. Now out of many instances, when and if we want to pick a particular/single instance, and many times we do need it, then key is the solution. For example, think of whole population of Pakistan, the data of all Pakistanis lying at one place, say with NADRA people. Now if at sometime we need to identify a particular person out of all this data, how can we do that? Can we use name for that, well think of any name, like Mirza Zahir Iman Afroz, now we may find many people with this name in Pakistan. Another option is the combination of name and father name, then again, Amjad Malik s/o Mirza Zahir Iman Afroz, there could be so many such pairs. There could be many such examples. However, if you think about National ID Card number, then no matter whatever is the population of Pakistan, you will always be able to pick precisely a single person. That is the key. While defining an entity type we also generally define the key of that entity type. How do we select the key, from the study of the real-world system; key attribute(s) already exist there, sometimes they don't then the designer has to define one. A key can be simple, that is, consisting of single attribute, or it could be composite which consists of two or more attributes. Following are the major types of keys:

- Super Key
- Candidate Key
- Primary Key
- Alternate Key
- Secondary Key
- Foreign Key

The last one will be discussed later, remaining 5 are discussed in the following:

- **Super key**
  A **super key** is a set of one or more attributes which taken collectively, allow us to identify uniquely an entity instance in the entity set. This definition is same as of a key, it means that the super key is the most general type of key. For example, consider the entity type STUDENT with attributes registration number, name, father name, address, phone, class, admission date. Now which attribute can we use that can uniquely identify any instance of STUDENT entity type. Of course, none of the name, father name, address, phone number, class, admission date can be used for this purpose. Why? Because if we consider name as super key, and situation arises that we need to contact the parents of a particular student. Now if we say to our registration department that give us the phone number of the student whose name is Ilyas Hussain, the registration department conducts a search and comes up with 10 different Ilyas Hussain, could be anyone. So the value of the name attribute cannot be used to pick a particular instance. Same happens with other attributes. However, if we use the registration number, then it is 100% sure that with a particular value of registration number we will always find exactly a single unique entity instance. Once you identified the instance, you have all its attributes available, name, father name,

everything. The entity type STUDENT and its attributes are shown graphically in the figure 1 below, with its super key "regNo" underlined.

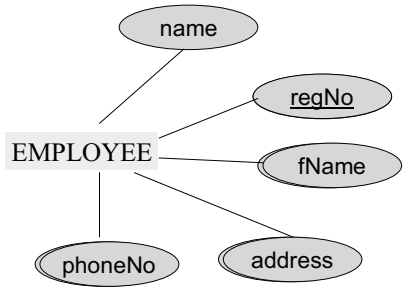



Fig. 1: An entity type, its defining attributes and super key (underlined)

Once specific characteristic with super key is that, as per its definition any combination of attributes with the super key is also a super key. Like, in the example just discussed where we have identified regNo as super key, now if we consider any combination of regNo with any other attribute of STUDENT entity type, the combination will also be a super key. For example, "regNo, name", "regNo, fName, address", "name, fName, regNo" and many others, all are super keys.

- **Candidate key**

  A super key for which no subset is a super key is called a candidate key, or the minimal super key is the candidate key. It means that there are two conditions for the candidate key, one, it identifies the entity instances uniquely, as is required in case of super key, second, it should be minimum, that is, no proper subset of candidate key is a key. So if we have a simple super key, that is, that consists of single attribute, it is definitely a candidate key, 100%. However, if we have a composite super key and if we take any attribute out of it and remaining part is not a super key anymore then that composite super key is also a candidate key since it is minimal super key. For example, one of the super keys that we identified from the entity type STUDENT of figure 1 is "regNo, name", this super key is not a candidate key, since if we remove the regNo attribute from this combination, name attribute alone is not able to identify the entity instances uniquely, so it does not satisfy the first condition of candidate key. On the other hand if we remove the attribute name from this composite key then the regNo alone is sufficient to identify the instances uniquely, so "regNo, name" does have a proper subset (regNo) that can act as a super key; violation of second condition. So the composite key "regNo, name" is a super key but it is not a candidate key. From here we can also establish a fact that every candidate key is a super key but not the other way round.

- **Primary Key**

A candidate key chosen by the database designer to act as key is the primary key. An entity type may have more than one candidate keys, in that case the database designer has

to designate one of them as primary key, since there is always only a single primary key in an entity type. If there is just one candidate key then obviously the same will be declared as primary key. The primary key can also be defined as the successful candidate key. Figure 2 below contains the entity type STUDENT of figure 1 but with an additional attribute nIdNumber (national ID card Number).

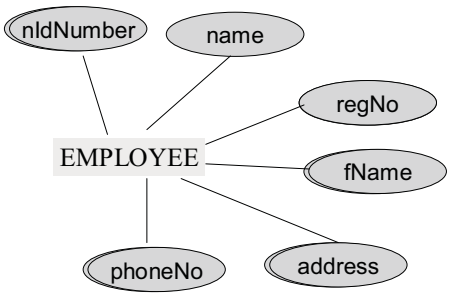



Fig. 2: An entity type, its defining attributes and two candidate keys

In figure 2, we can identify two different attributes that can individually identify the entity instances of STUDENT and they are regNo and nIdNumber, both are minimal super keys so both are candidate keys. Now in this situation we have got two candidate keys. The one that we choose will be declared as primary key, other will be the alternate key. Any of the candidate keys can be selected as primary key, it mainly depends on the database designer which choice he/she makes. There are certain things that are generally considered while making this decision, like the candidate key that is shorter, easier to remember, to type and is more meaningful is selected as primary key. These are general recommendations in this regard, but finally it is the decision of the designer and he/she may have his/her own reasons for a particular selection that may be entirely different from those mentioned above. The relation that holds between super and candidate keys also holds between candidate and primary keys, that is, every primary key (PK) is a candidate key and every candidate key is a super key.

A certain value that may be associated with any attribute is NULL, that means "not given" or "not defined". A major characteristic of the PK is that it cannot have the NULL value. If PK is a composite, then none of the attributes included in the PK can have the NULL, for example, if we are using "name, fName" as PK of entity type STUDENT, then none of the instances may have NULL value in either of the name or fName or both.

- **Alternate Keys**
  Candidate keys which are not chosen as the primary key are known as alternate keys. For example, we have two candidate keys of STUDENT in figure 2, *regNo* and *nIdNumber*, if we select *regNo* as PK then the *nIdNumber* will be alternate key.

- **Secondary Key**

Many times we need to access certain instances of an entity type using the value(s) of one or more attributes other than the PK. The difference in accessing instances using the

value of a key or non-key attribute is that the search on the value of PK will always return a single instance (if it exists), where as uniqueness is not guaranteed in case of non-key attribute. Such attributes on which we need to access the instances of an entity type that may not necessarily return unique instance is called the secondary key. For example, we want to see how many of our students belong to Multan, in that case we will access those instances of the STUDENT entity type that contain "Multan" in their address. In this case address will be called secondary key, since we are accessing instances on the basis of its value, and there is no compulsion that we will get a single instance. Keep one thing in mind here, that a particular access on the value of a secondary key MAY return a single instance, but that will be considered as chance or due to that particular state of entity set. There is not the compulsion or it is not necessary for secondary key to return unique instance, where as in case of super, candidate, primary and alternate keys it is compulsion that they will always return unique instance against a particular value.

**Summary**

Keys are fundamental to the concept almost any data model including the E-R data model because they enable the unique identity of an entity instance. There are different type of keys that may exist in an entity type.

**Exercises:**

- Define attributes of the entity types CAR, BOOK, MOVIE; draw them graphically
- Identify different types of keys in each one of them

# Lecture No. 09

## Reading Material

| | |
|---|---|
| "Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan. | |

## Overview of Lecture

- Relationships in E-R Data Model
- Types of Relationships

## Relationships

After two or more entities are identified and defined with attributes, the participants determine if a *relationship* exists between the entities. A relationship is any association, linkage, or connection between the entities of interest to the business; it is a two-directional, significant association between two entities, or between an entity and itself. Each relationship has a name, an optionality (*optional* or *mandatory*), and a degree (how many). A relationship is described in real terms.

Assigning a name, optionality, and a degree to a relationship helps confirm the validity of that relationship. If you cannot give a relationship all these things, then perhaps there really is no relationship at all.

Relationship represents an association between two or more entities. An example of a relationship would be:

- Employees are assigned to projects
- Projects have subtasks
- Departments manage one or more projects

Relationships are the *connections and interactions* between the entities instances e.g. DEPT_EMP associates Department and Employee.

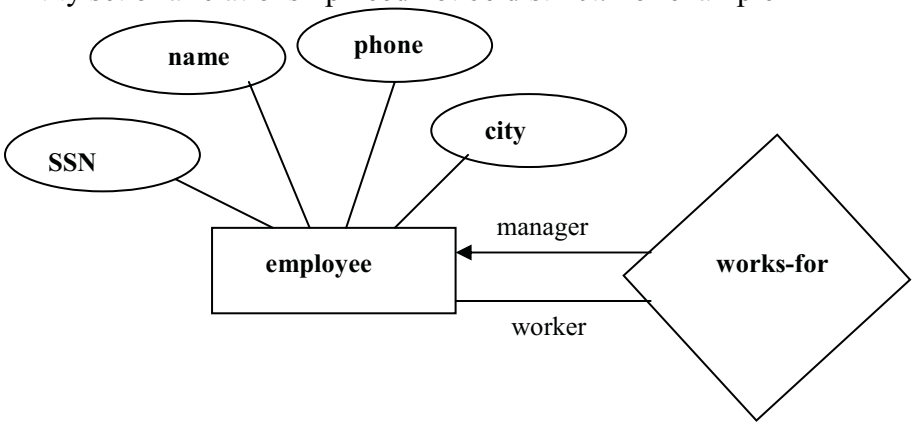
- A ***relationship type*** is an abstraction of a relationship i.e. a set of relationships instances sharing common attributes.

- Entities enrolled in a relationship are called its ***participants***.

The participation of an entity in a relationship is ***total*** when all entities of that set might be participant in the relationship otherwise it is ***partial*** e.g. if every ***Part*** is supplied by a ***Supplier*** then the SUPP_PART relationship is total. If certain parts are available without a supplier than it is partial.

**Naming Relationships:**
If there is no proper name of the association in the system then participants' names of abbreviations are used. STUDENT and CLASS have ENROLL relationship. However, it can also be named as STD_CLS.

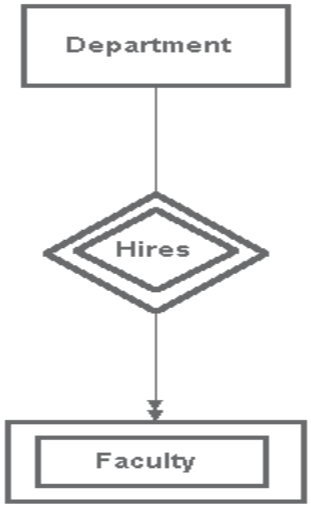
**Roles:**
Entity set of a relationship need not be distinct. For example




The labels "manager" and "worker" are called "roles". They specify how employee entities interact via the "works-for" relationship set. Roles are indicated in ER diagrams by labeling the lines that connect diamonds to rectangles. Roles are optional. They clarify semantics of a relationship.

**Symbol for Relationships:**

- Shown as a Diamond
- Diamond is doubled if one of the participant is dependent on the other
- Participants are connected by continuous lines, labeled to indicate cardinality.
- In partial relationships roles (if identifiable) are written on the line connecting the partially participating entity rectangle to the relationship diamond.
- Total participation is indicated by double lines

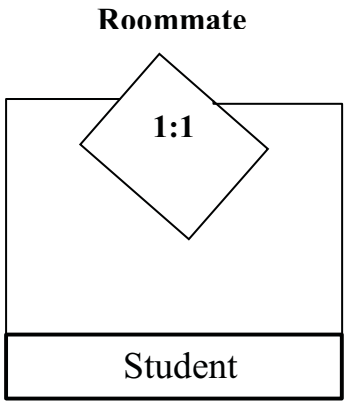



## Types of Relationships

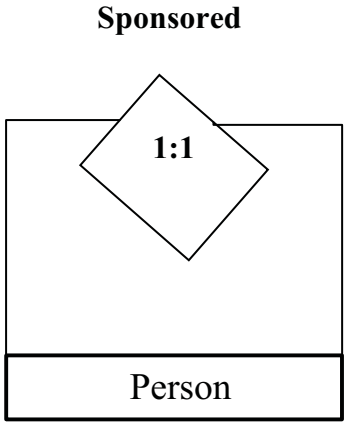
- **Unary Relationship**

  An ENTITY TYPE linked with itself, also called recursive relationship. Example Roommate, where STUDENT is linked with STUDENT

  **Example 1:**



**Example 2:**




- **Binary relationship**

  A **Binary** relationship is the one that links two entities sets e.g. **STUDENT-CLASS.** Relationships can be formally described in an ordered pair form.

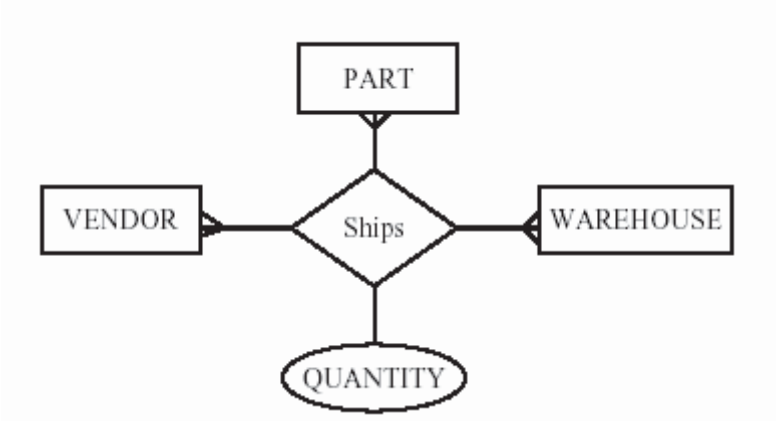
  Enroll = {(S1001, ART103A), (S1020, CS201A), (S1002, CSC201A)}

  Entire set is relationship set and each ordered pair is an instance of the relationship.


- **Ternary Relationship**

  A **Ternary** relationship is the one that involves three entities e.g.

  **STUDENT-CLASS-FACULTY**.



- **N-ary Relationship**

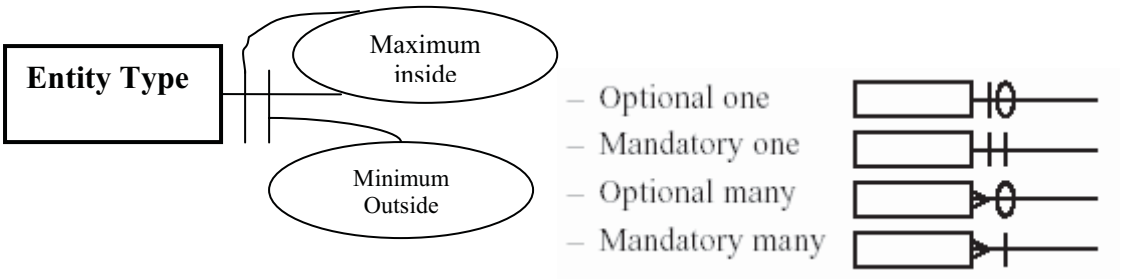
  Most relationships in data model are binary or at most ternary but we could define a relationship set linking any number of entity sets i.e. **n-ary** relationship

Entity sets involved in a relationship set need not be distinct. E.g.

Roommate = {(Student1, Student2) | Student1 ∈ Student Entity Set, Student2 ∈ Student Entity Set and Student 1 is the Roommate of Student2}

**Relationship Cardinalities**

The cardinality of a relationship is the number of entities to which another entity can map under that relationship. Symbols for maximum and minimum cardinalities are:

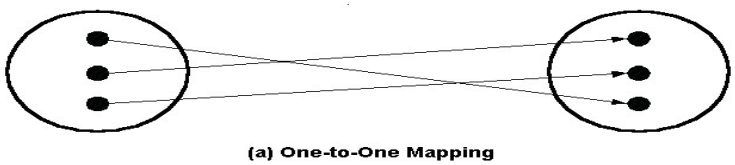



- **One-to-One mapping:**

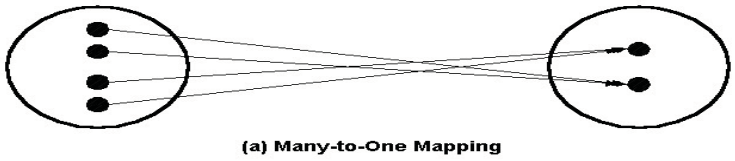
  A mapping R from X to Y is one-to-one if each entity in X is associated with at most one entity in Y and vice versa.

- **Many-to-One mapping:**

  A mapping R from X to Y is many-to-one if each entity in X is associated with at most one entity in Y but each entity in Y is associated with many entities in X.



(a) One-to-One Mapping



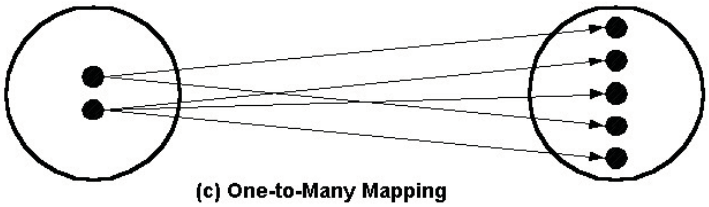
(a) Many-to-One Mapping

- **One-to-Many mapping:**

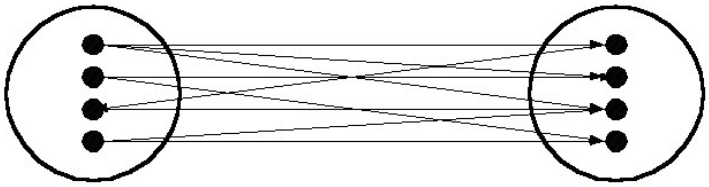
  A mapping R from X to Y is one-to-many if each entity in X is associated with many entities in Y but each entity in Y is associated with one entity in X.

- **Many-to-Many mapping:**

  A mapping R from X to Y is many-to-many if each entity from X is associated with many entities in Y and one entity in Y is associated with many entities in X.



(c) One-to-Many Mapping



(d) Many-to-Many Mapping

# Lecture No. 10

## Reading Material

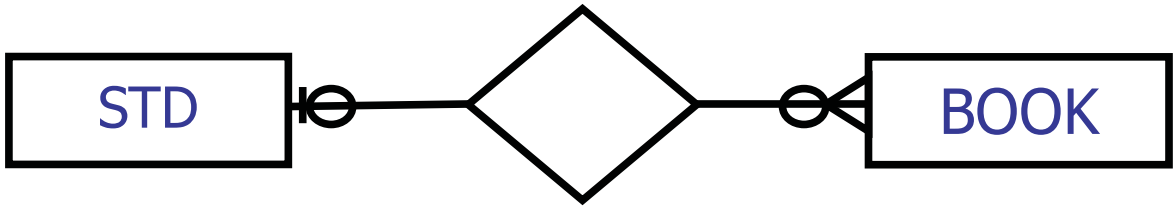
| | |
|---|---|
| "Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan. | Page: 155 – 160 |
| Hoffer | Page: 103 – 111 |

## Overview of Lecture

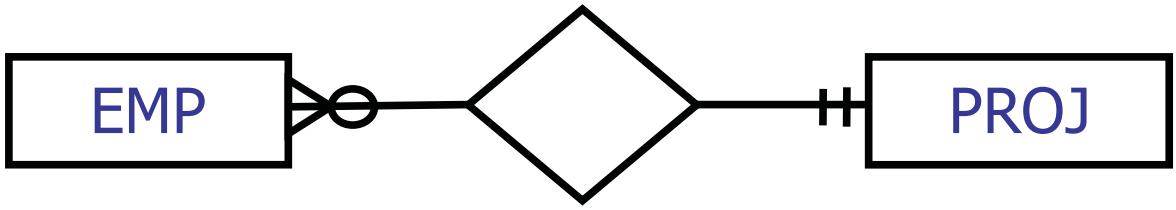
- Cardinality Types
- Roles in ER Data Model
- Expression of Relationship in ER Data Model
- Dependency
- Existence Dependency
- Referential Dependency
- Enhancements in the ER-Data Model
- Subtype and Supertype entities

Recalling from the previous lecture we can say that that cardinality is just an expression which tells us about the number of instances of one entity which can be present in the second relation. Maximum cardinality tells us that how many instance of an entity can be placed in the second relation at most. Now we move onto discuss that what the minimum cardinality is.

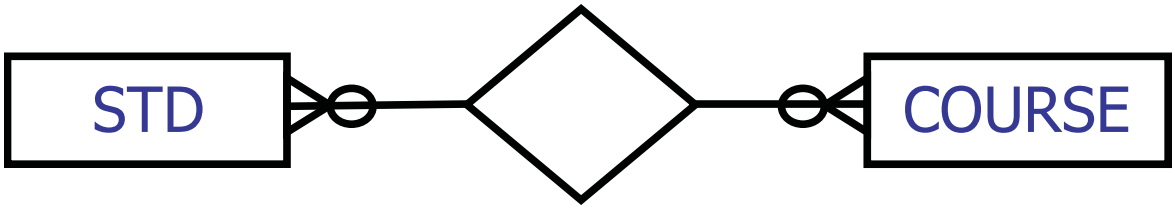
**Minimum Cardinality:**
As the name suggests that the minimum cardinality is the inverse of the maximum cardinality so we can say that the minimum cardinality show us that how many instance of one entity can be placed in another relation at least. In simple words it can be said that the minimum cardinality tells that whether the link between two relations is optional or compulsory. It is very important to determine the minimum cardinality when designing a database because it defines the way a database system will be implemented.



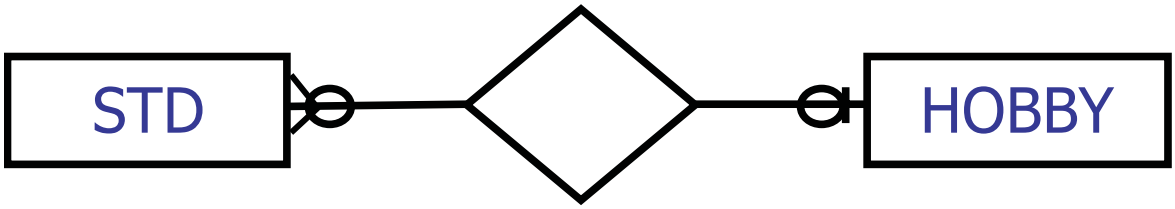

**One to Many (optional)**




**Many to One (Mandatory)**




**Many to Many (optional)**




**Many to One (optional)**

Fig 1: Different Cardinalities

In the figure-1 we have one to many cardinality between the entities. Maximum cardinalities are shown with the modifier that appears on the link and is adjacent to the entity rectangle. The other modifier which is next to the maximum cardinality modifier tells the minimum cardinality. The minimum cardinality modifier lies at more distance from the entity as compared to the maximum cardinality modifier.

Determination of the cardinalities is done by interviewing the users of the system and by the analysis of the organization.

The cardinality shown in First Part of the Figure-1 is shown using a relationship between a student and book; this can be a library scenario where students are borrowing books from the library. We can see in the diagram the shape  adjacent to the student entity it shows that the minimum cardinality for the student relationship is zero and maximum cardinality is one. Where as on the other side of the diagram the shape  adjacent to

the book entity show that at most there can be many instances of the book associated with a single instance of student entity, and that there can be at-least no instance associated with the student entity. In general library scenario we can say that one student can borrow at least no and at most many books. Hence the minimum and maximum cardinality is shown.

In the second part of the Figure-1 we see a relationship between the Employee and project entities, the relationship describes one to many association between the project and the employees, It shows that there can be one project having a number of employees, but for the existence of one employee at one project is necessary. So the minimum and maximum cardinality on the project side of the relationship is one, and employees associated with each project can be many at most and none at-least.

Third part of the Figure-1 shows the association between the student and the course entities. Here we can see that the relationship between the student and the course is zero at least and many at most on both the sides of the relationship. The minimum cardinality with zero minimum is also called the optional cardinality. It also shows that one student can have registered more that one subjects and one subject can also be taken by many students. Also it is not necessary for a student to get registered one subject.

In the fourth part of the Figure-1 we can see the one to many cardinality between the student and hobby entities the cardinality descriptors show that a student may have no or at most one hobby, but it is worthwhile to notice that the cardinality of the hobby with the student in many but optional, now we can say that one hobby can be associated to nay student but there is a chance that no hobby is associated to one student at a certain time.

**Other Notations:**

The notation mentioned above is known as crow's foot notation for the expression of ER-Diagrams, there can be other notation as well which can be used for creating ER-Diagrams; one of these notations is shown in the Figure-2. We can see that the one to many cardinality shown in the first part of the diagram is expresses with single and double arrows. The Single arrow in this case shows the one and double arrow show the many cardinality.

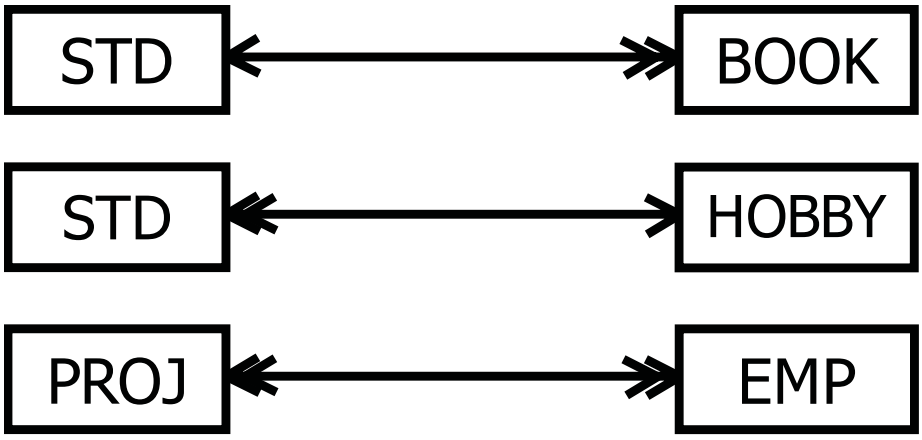



Fig. 2: Arrow-head notation

So the First part of the figure-2 show One to many cardinality, second part of the Figure shows many to one and the third part of the cardinality shows many to many cardinality between the entities involved.

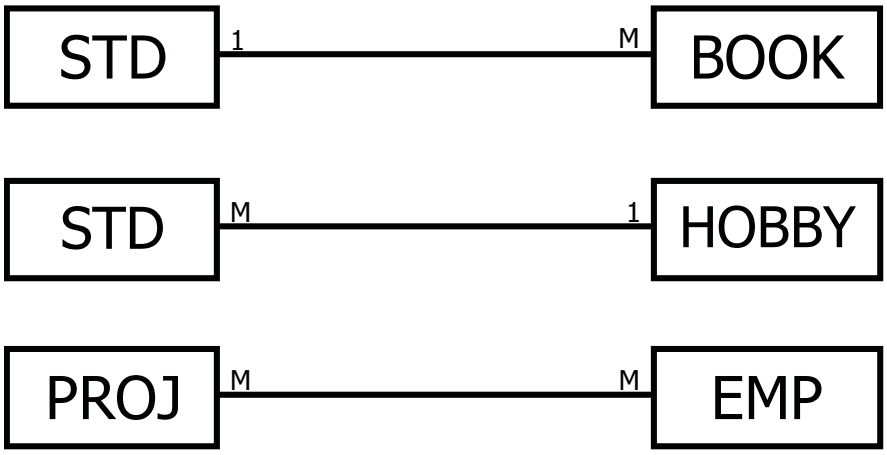



Fig. 3: Alphabetical notation

The above Figure shows another notation for creating ER-Diagrams which show that to show the one cardinality we have used 1 and for many cardinality M or N is used.

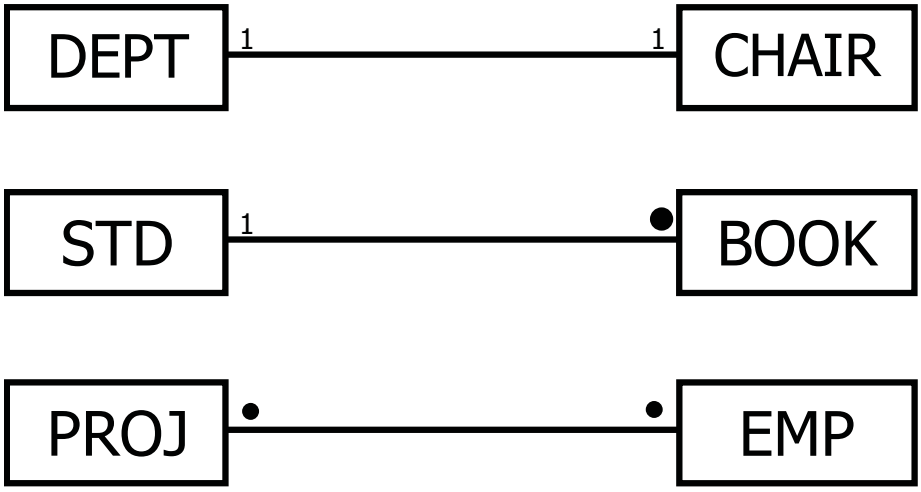



Fig. 4: Dot-based notation

Notations shown in the Figure-4 above as also used for creating ER-Diagrams where 1 is used for showing the single cardinality and the black filled Dot is used for showing many cardinality.

## Roles in Relationships

The way an entity is involved in a relationship is called the role of the entity in the relationship. These details provide more semantics of the database. The role is generally clear from the relationship, but in some cases it is necessary to mention the role explicitly.
**Two situations to mention the role explicitly**

**Recursive Relationship:**
This is the situation when any attribute of one entity is associated with another attribute of the same entity. Such a link initiates from one entity and terminates on the same entity.

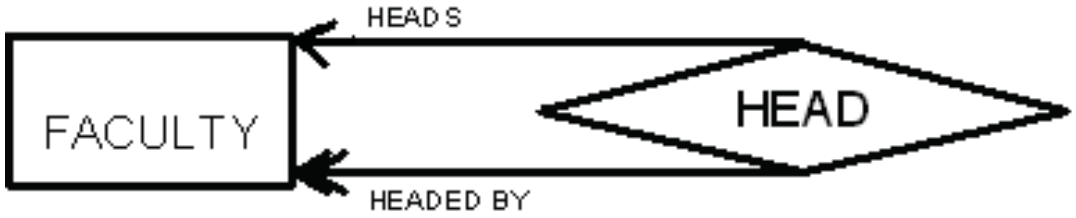



Fig-5: Roles in a unary relationship

Figure-5 above shows the recursive relationship which tells that in the faculty of a certain institute we can have one faculty member from among the same faculty as the head of the faculty. Now the role mentioned on the relationship tell that many Faculty instance are headed by one of the entity instance from the same faculty relation.

**Multiple Relationships:**
This is the second situation which needs the role to be mentioned on the relationship link when there is more than one relationship.

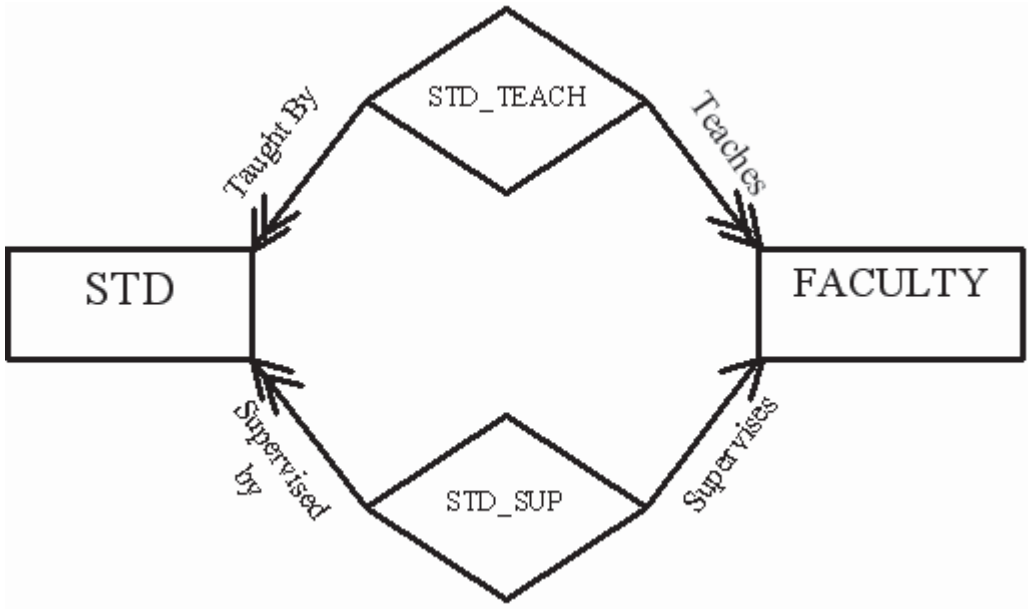



Fig. 6: Multiple relationships

As an example we can have a relationship of Faculty members and students as one faculty member may teach a number of students and at the same time one student may have been taught by a number of faculty members. This is one side of the picture. Now on the other side we can say that a faculty member may be supervising a number of students for their final projects. It shows two types of associations between the faculty and the students. So in this type of situation it is necessary to mention the role of the entities involved in the relationship.

## Dependencies

Dependency is a type of constraint, for example once we define the cardinality or relationship among two entities it also is a constraint or check that tells that cardinality should be followed while populating data in relations. Similarly the dependency is a constraint. There are a number of dependency types which are expressed below:

**The Existence dependency:**
This is the type of dependency which exists when one entity instance needs instance of another entity for its existence. As we have seen earlier in case of employee of and organization and the projects associated with the employees there we see that employees are dependent on projects, it means that if no project is assigned to an employee it can not exist. In other words we can say that at a certain time an employee must be working on at least one project.

**Identifier Dependency:**
It means that the dependent entity incase of existence dependency does not have its own identifier and any external identifier is used to pick data for that entity. And to define a key in this entity the key of the parent entity is to be used in the key for this entity may be used as composite keys.

**Referential Dependency:**
This is the situation when the dependent entity has it own key for unique identification but the key used to show the reference with the parent entity is shown with the help of an attribute of the parent entity. Means to show the link of the parent entity with this entity there will be an attribute and a record in this entity will not exist without having a record in the parent entity. Despite of having its own identifier attribute.
This type of identifier or attribute in the weak entity is known as foreign key.

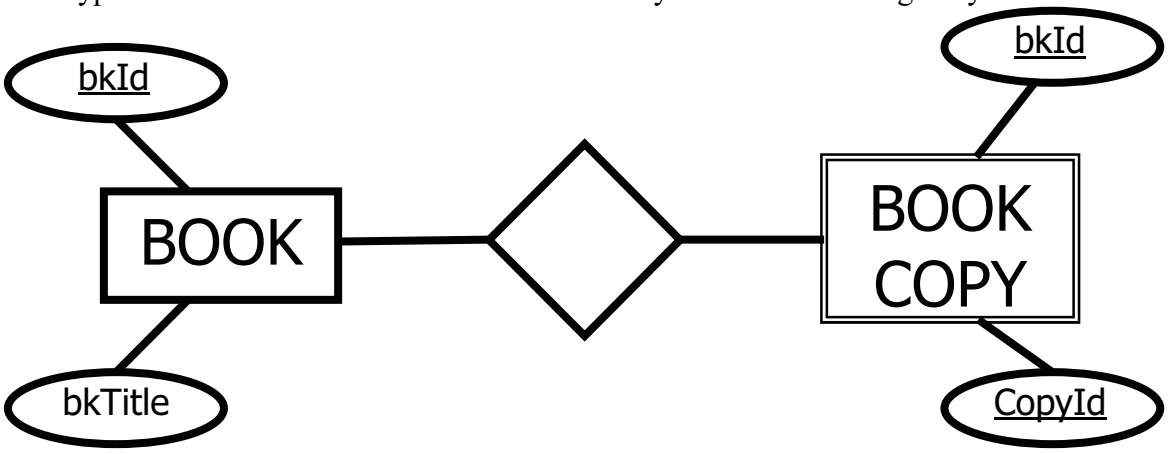



Fig-7

In the Figure-7 above the relation shown is expression the existence dependency where it is necessary for a book instance to exist if there exist the copies of the book with the same bkId.

## Enhancements in E-R Data Model:

The topics that we have discussed so for constitute the basics of ER-Model. The model is further extended and strengthened with addition of some new concepts and modeling constructs, which are discussed below

## Super-type and Subtypes

These are also relationships existing between entities, also referred to as generalized and specialized respectively let us examine the figure below to grasp the idea of super-type and subtype.

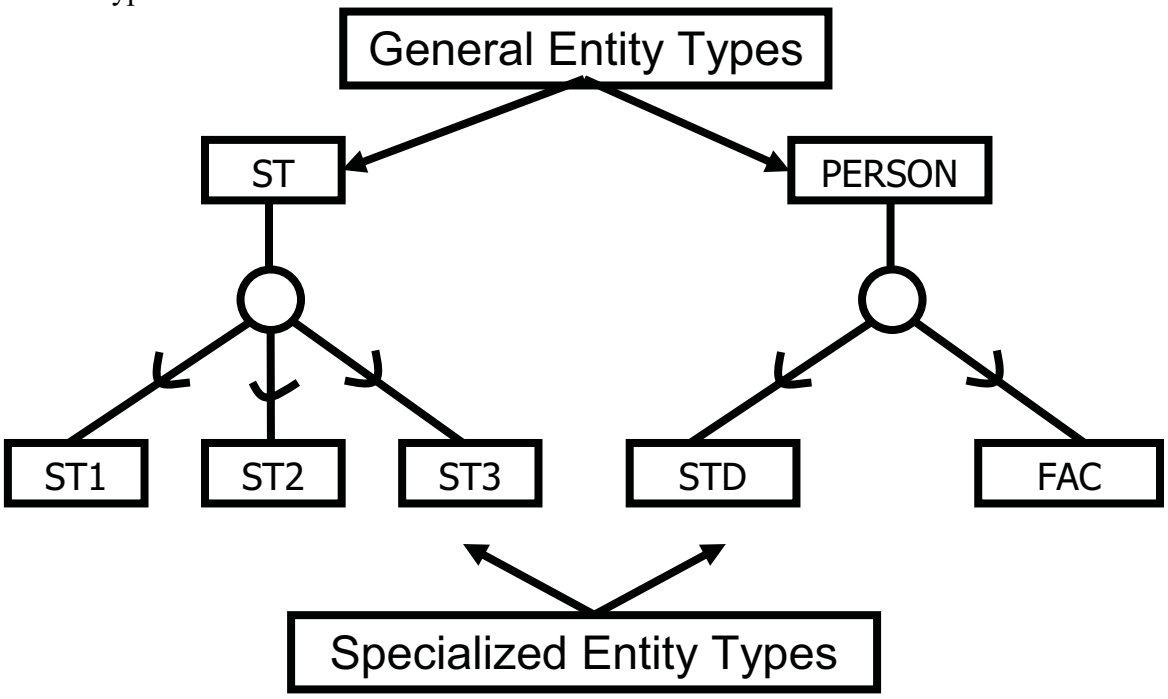



Fig-8 (Super-types and Subtypes)

In the Figure:8 show above there are different levels of existence of entities, at the top level we have general entity type, which are described as having a number of Subtype entities, these sub entities are in-turn acting as supertypes entities for a number of other entities. As we see in case of person supertype we can have further classify the person entity as Student (STD) and Teacher of Faculty member (FAC). Subtype entities are expressed with a link to the supertypes having an arc on the link—the arms of which

point to the supertype entity. As we move downward the distributed entities are known as specialized entities.

In the next Lecture the process of Generalization and Specialization will be discussed in detail.

## Summary:

In this lecture we have discussed an important topic of cardinalities and their representation in the E-R data model. For a correct design the correct identification of cardinalities is important.

# Lecture No. 11

## Reading Material

| | |
|---|---|
| "Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan. | |

## Overview of Lecture

- Inheritance
- Super type
- Subtypes
- Constraints
- Completeness
- Disjointness
- Subtype Discrimination

According to the Microsoft Dictionary of Computing

## Inheritance Is

The transfer of the characteristics of a class in object-oriented programming to other classes derived from it. For example, if "vegetable" is a class, the classes "legume" and "root" can be derived from it, and each will inherit the properties of the "vegetable" class: name, growing season, and so on2. Transfer of certain properties such as open files, from a parent program or process to another program or process that the parent causes to run. Inheritance in the paradigm of database systems we mean the transfer of properties of one entity to some derived entities, which have been derived from the same entities.

## Super types and Subtypes

Subtypes hold all the properties of their corresponding super-types. Means all those subtypes which are connected to a specific supertype will have all the properties of their supertype.

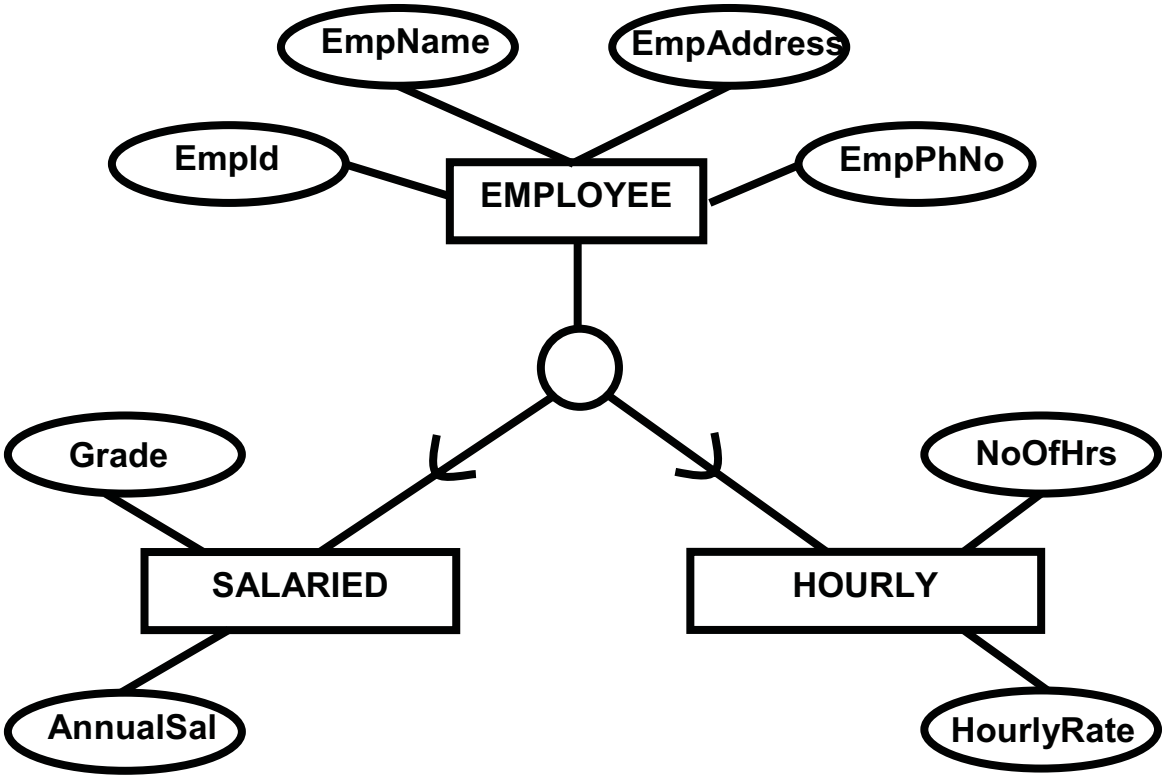



Fig-1 a

The Figure:1 above shows that the supertype and subtype relation between the SALARIED and HOURLY employees with the supertype entity EMPLOYEE, we can see that the attributes which are specific to the subtype entities are not shown with the supertype entity. Only those attributes are shown on the supertype entity which are to be inherited to the subtypes and are common to all the subtype entities associated with this supertype.

The example shows that there is a major entity or entity supertype name EMPLOYEE and has a number of attributes. Now that in a certain organization there can be a number of employees being paid on different payment criteria.

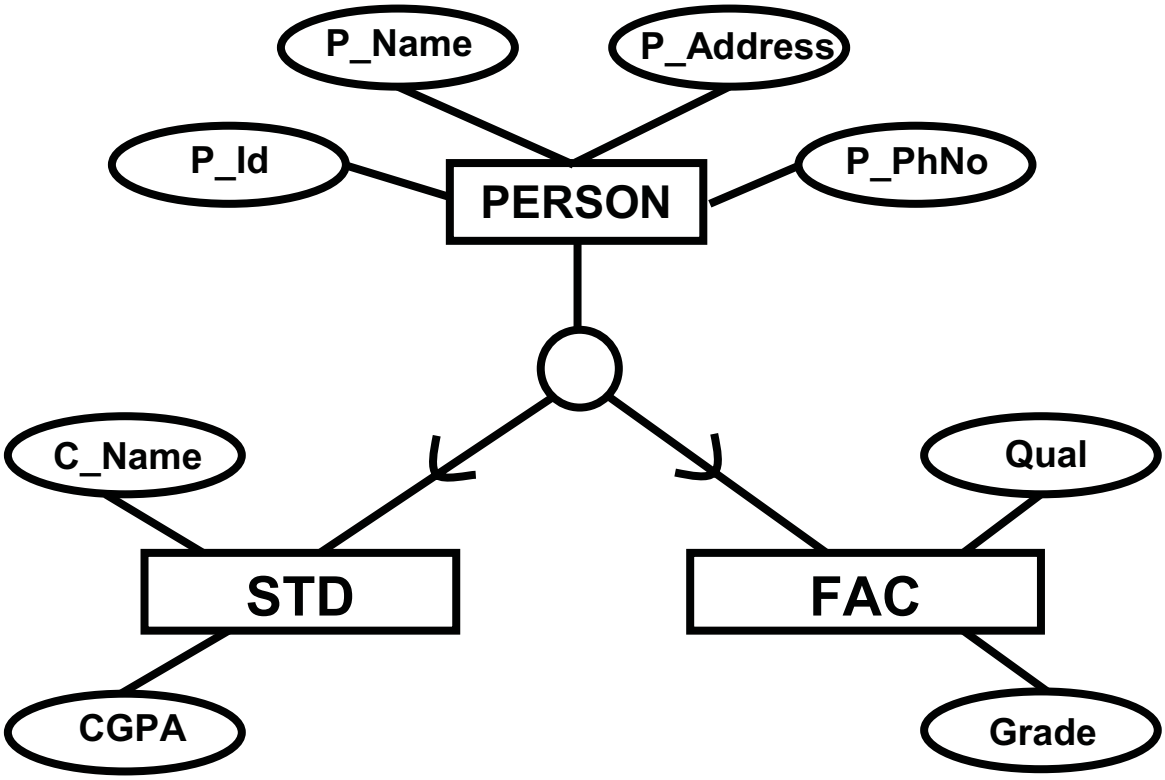



Fig – 1 b

The second example is that of student and the Faculty members who are at the super level same type of entities. Both the entities at the super level belong to the same entity of type Person. The distinct attributes of the student and faculty members are added later to he sub entities student and fac.

**Supertype / subtype Relationship:**

The use of supertype and subtype for the entities is very useful because it allows us to create hierarchy of the entities according to the attributes they have and we need not to write all the attributes again and again. We can group similar types of entities and the attributes associated with those entities at certain levels.

This also adds clarity to the definitions of the entities as it is not necessary to write the attribute again and again for all the entities.

Moreover it also eases the operation of removing or adding attributes from the entities, here it is worth noting that adding an attribute at the super entity level will add the

attribute to the below listed or derived sub entities and removing the attribute will remove the attribute from the entities at sublevels in the same way.

The process of identifying supertype and creating different type of sub entities is supported by the general knowledge of the designer about the organization and also based of the attributes of the entities which are entities existing in the system..

## Specifying Constraints

Once there has been established a super/sub entity relationship there are a number of constraints which can be specified for this relationship for specifying further restrictions on the relationship.

## Completeness Constraint

There are two types of completeness constraints, partial completeness constraints and total completeness constraints.

**Total Completeness:**

Total Completeness constraint exist only if we have a super type and some subtypes associated with that supertype, and the following situation exists between the super type and subtype.

All the instances of the supertype entity must be present in at one of the subtype entities, i.e.—there should be not instance of the supertype entity which does not belong to any of the subtype entity.

This is a specific situation when the supertype entities are very carefully analyzed for their associated subtype entities and no sub type entity is ignored when deriving sub entities from the supertype entity.

**Partial Completeness Constraint:**

This type of completeness constraint exists when it is not necessary for any supertype entity to have its entire instance set to be associated with any of the subtype entity.

This type of situation exists when we do not identify all subtype entities associated with a supertype entity, or ignore any subtype entity due to less importance of least usage in a specific scenario.

## Disjointness Constraint

This rule or constraint defines the existence of a supertype entity in a subtype entity. There exist type types of disjoint rules.

- Disjointness rule
- Overlap rule

**Disjoint constraint:**
This constraint restricts the existence of one instance of any supertype entity to exactly one instance of any of the subtype entities.
Considering the example given in Fig 1a it is seen that there can be two types of employees, one which are fixed salary employees and the others are hourly paid employees. Now the disjoint rule tells that at a certain type an employee will be either hourly paid employee or salaried employee, he can not be placed in both the categories in parallel.

**Overlap Rule:**
This rule is in contrast with the disjoint rule, and tells that for one instance of any supertype entity there can be multiple instances existences of the of the instance for more then one subtype entities. Again taking the same example of the employee in an organization we can say that one employee who is working in an organization can be allowed to work for the company at hourly rates also once he has completed his duty as a salaried employee. In such a situation the employee instance record for this employee will be stored in both the sub entity types.

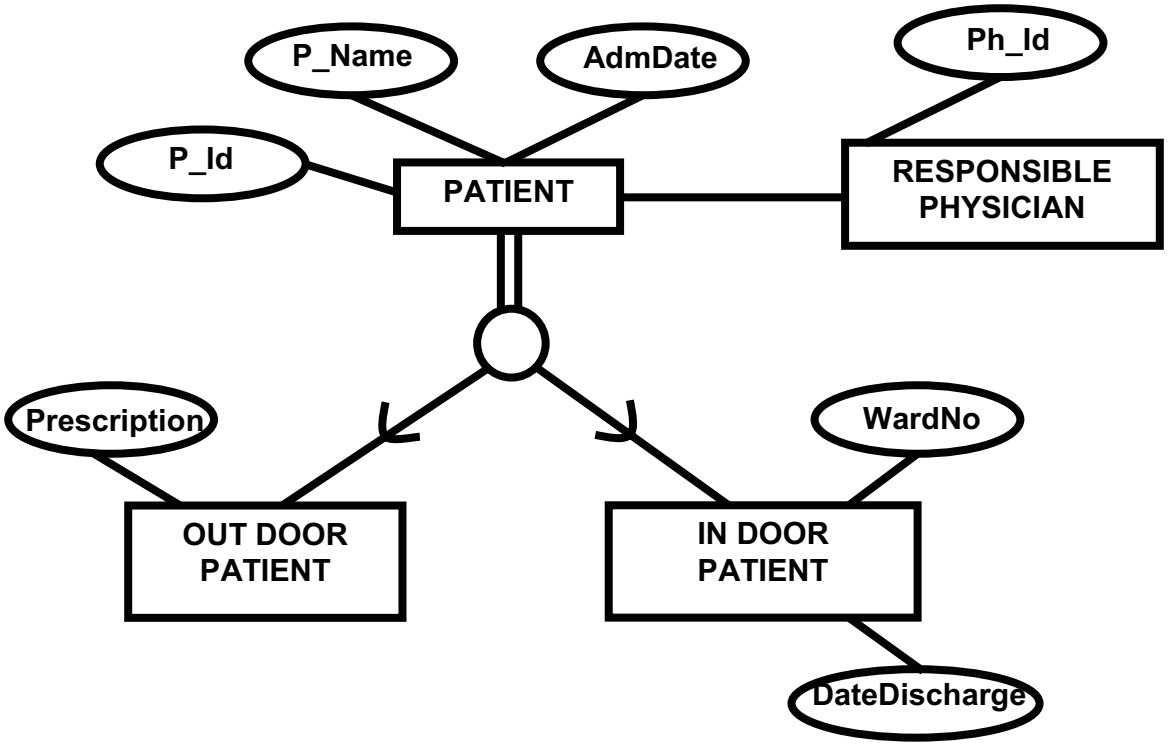



Fig 2-a

In the example the completeness of the relation is shown between the supertype entity and the subtype entity, it shows that for the data of patients we can have only two type of patients and one patient can be either an outdoor patient or indoor patient. In it we can see that we have identified all possible subtypes of the supertype patient. This implies a completeness constraint. One more thing to note here is the linked entity physician to the patient entity. And all the relationships associated with the supertype entity are inherited to subtype entities of the concerned supertype.

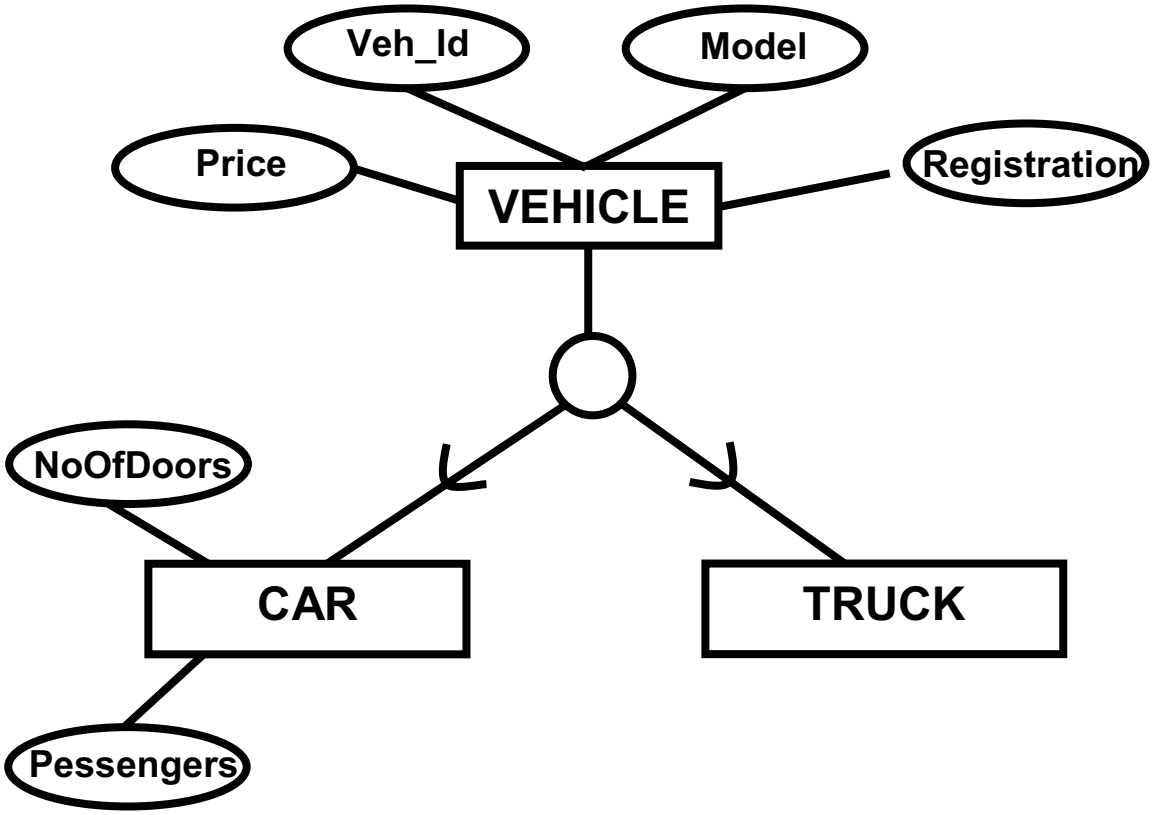



Fig 2-b

The Figure2b shows the supertype and subtype relationship among different type of vehicles. Here we can see that the Vehicle has only two subtypes, known as Truck and Car, As it is normal to have a number of other vehicles in the company of a certain type but when we have noted just a limited number of vehicles then it means that we are not interested in storing information for all the vehicles as separate entities. They may be stored in the vehicle entity type itself and distinct vehicle may be stored in the subtypes car and truck of the Vehicle.

This is a scenario where we have the freedom to store several entities and neglect others, and it is called as partial completeness constraint rule.

After the discussion of the Total Completeness and Partial completeness let us move to the next constraint that is disjointness and check for its examples.

Again in the Figure 2-a. we have the environment where patient entity type has two subtypes indoor and outdoor patient. To represent disjointness we place the letter “D” in the circle which is splitting the super entity type into two sub entity types. Suppose that the hospital has placed a restriction on the patient to be either a n indoor patient or

outdoor patient, in such a case there exists disjointness which specifies that the patients data can not be place in the database in both the subtype entities. It will be wither indoor or outdoor.

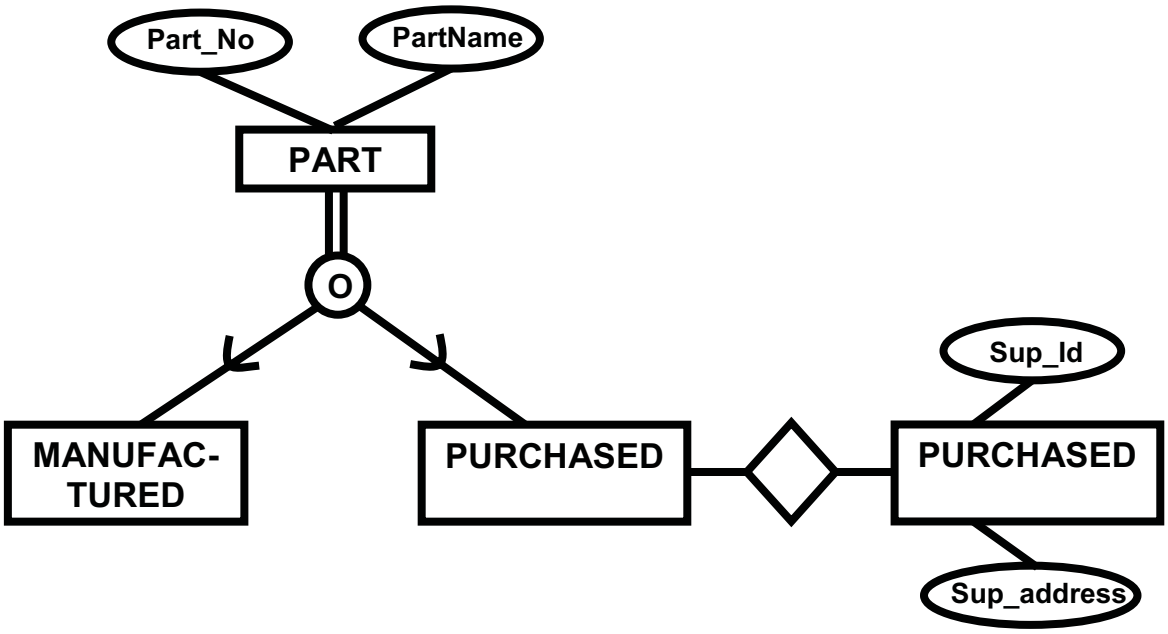



Fig- 3

The figure 3 above shows the second type of disjoint constraint which tells that the entity subtype instance can be repeated for any single entity supertype instance. We can see the relationship of a certain hardware company for the parts provided by the company to its clients. Now there may exist an overlapping situation for a certain part which is to be provided to a certain firm, but the manufactured quantity of that part is not enough to meet the specific order, In this situation the company purchases the remaining the deficient number of parts form the other suppliers. We can easily say that the data for that specific part is to be placed in both the entity subtypes. Because it belongs to both the subtype entities, this is an overlapping situation and expresses disjointness with overlapping. Another important thing which is to be noted here that the purchased part subtype entity has a relationship with another entity where the data for the suppliers is stored from whom the parts are bought. Now this relation does not have nay interaction with the manufactured parts relation as it is not connected with its supertype i.e.—parts supertype entity.

Considering the above discussed we can have four different types of combination existing for the supertype and subtype entities.

- Complete Disjoint
- Complete Overlapping
- Partial Disjoint
- Partial overlapping

## Subtype Discriminator

This is a tool or a technique which provides us a methodology to determine that to which subtype one instance of a supertype belongs.

To determine the relation we place an attribute in the entity supertype which can specify through its value, that to which entity subtype it belongs.

For example we consider the example

There can be two different situations which specify the placement or relationship of a supertype entity instance in a subtype entity instance. First situation is that of disjoint situation where one supertype entity instance can be placed only in one subtype of that supertype. Let us consider the example of vehicles above in Figure-2-b it show that there can be two different vehicles car and truck associated with the supertype vehicle now if we place an attribute named Vehicle_type in the supertype we can easily determine the type of the associated subtype by placing a C for car and a T for truck instance of the vehicle.

The other situation where the Subtype discriminator is required the overlapping constraint; it is the situation where one supertype attribute can be placed in more than one subtype entities.

Considering again the part example shown in Figure 3, which has an overlapping constraint; In this situation we can have many solution one common solution is to place two attribute in the supertype one for manufactured and other one for purchased. We can combine them as a composite attribute, when we place Y for manufacture and N for

Purchased then it means the part is manufactured by the company, and similarly the following situation will give us further information

| Attribute | | |
|---|---|---|
| Manufacture | Purchased | Result |
| Y | Y | Manufacture Purchased |
| Y | N | Manufactured |
| N | Y | Purchased. |

**Significance of Subtype Discriminator:**

Existence of subtype discriminator helps us a lot in finding the corresponding subtype entities, although we can find a subtype entity instance without having a subtype discriminator in the supertype but that involves lots of efforts and might consume a huge time in worst case situations.

This concludes out discussion of The ER Model in the course.

# Lecture No. 12

## Reading Material

| | |
|---|---|
| "Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan. | |

## Overview of Lecture

In today's lecture we will discuss the ER Data model for an existing system and will go through a practice session for the logical design of the system

The system discusses is an examination section of an educational institute with the implementation of semester system.

## Steps in the Study of system

**Preliminary study of the system**

- Students are enrolled in programs.
- The programs are based on courses
- Different courses are offered at the start of the semester
- Students enroll themselves for these courses at the start of semesters
- Enrolled courses by students and offered courses must not be same.
- The difference is due to the individual situation of every student, because if one student has not pass a certain course 'A' in the previous semester he will not be able to register for a course 'B' offered in this semester as the course 'A' is the prerequisite for course 'B'.
- After valid registration classes start.
- A Course which is offered is assigned to a teacher also
- There can be any mid term exams and in this system we have only one mid term
- All the students are given assignments and quizzes and are awarded marks against their performance.
- Result of the student is prepared on the basis of assignment marks, sessional and mid term marks and the final exam.
- GP (Grade point) for students is calculated in each subject.
- Average grade point is calculated on the basis of GPs in individual subjects

- And the Cumulative GPA is calculated for all the passed semesters.

**Outputs Required**

- Teachers and controller need class list or attendance sheet, class result; subject and overall
- Students need transcripts, semester result card, subject result

**Entities associated with the system**

- Students
- Teachers
- Controllers

Once the analysis of the system is done it is important to make a draft of the system using a standard tool which specifies the component and design of the system. This design is useful because anyone using the design can work on the existing system and clearly understand the working without working on the system from the scratch.

Tool used for such graphical design is Data Flow Diagram (DFD)
In the Figure -1 of the system we have a context diagram of the system which shows integration of different entities with the examination system, these include Registration system, controller, student and teacher entities.

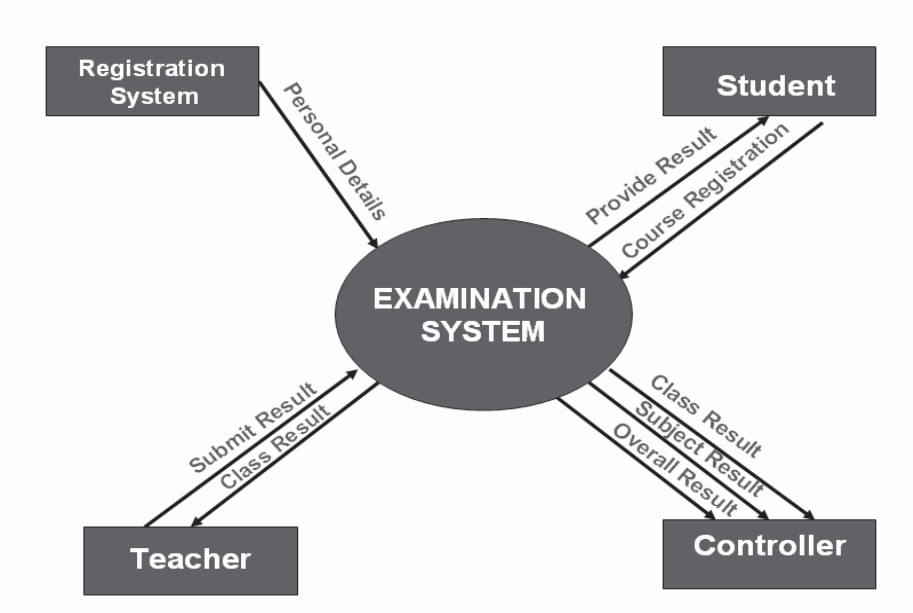



Fig-1

- From the diagram we can understand basic functionality of the system and can find how the data is flowing in the system and how different external entities are communicating or interacting with the system.

- First of all we have registration system, which provides the data of students to the systems once the registration process has been completed, this data is now free of errors in terms of validity of a certain student for a certain course or a semester.
- Second external entity interacting with the system is the teacher, a Teacher is given a list of students who are enrolled in a class and the registration system has declared them as valid students for that very course. Then the teacher allows those students in the class and continues the process of teaching the class, during this process the teacher takes test of the students and prepares papers for the students and also prepares quizzes to be submitted by students. All the data of students' attendance quizzes and assignments along-with different sessional results is then submitted by the teacher to the examination system which is responsible for preparation of results of the students
- Third interacting entity with the system is the controller's office it is provided with the semester overall result, subject results and also the result of each class fir performance evaluation and many other aspects.
- Fourth entity is student which externally interacts with the system for getting its result, the result is submitted to the student and may be in one of different forms such as, transcript and result card etc.

**Level 0 Diagram**

The three major modules which have been identified are given below our level 0 diagram will be based on these three modules and will elaborate and describe each of the modules in details.

- Subject registration
- Result submission
- Result calculation

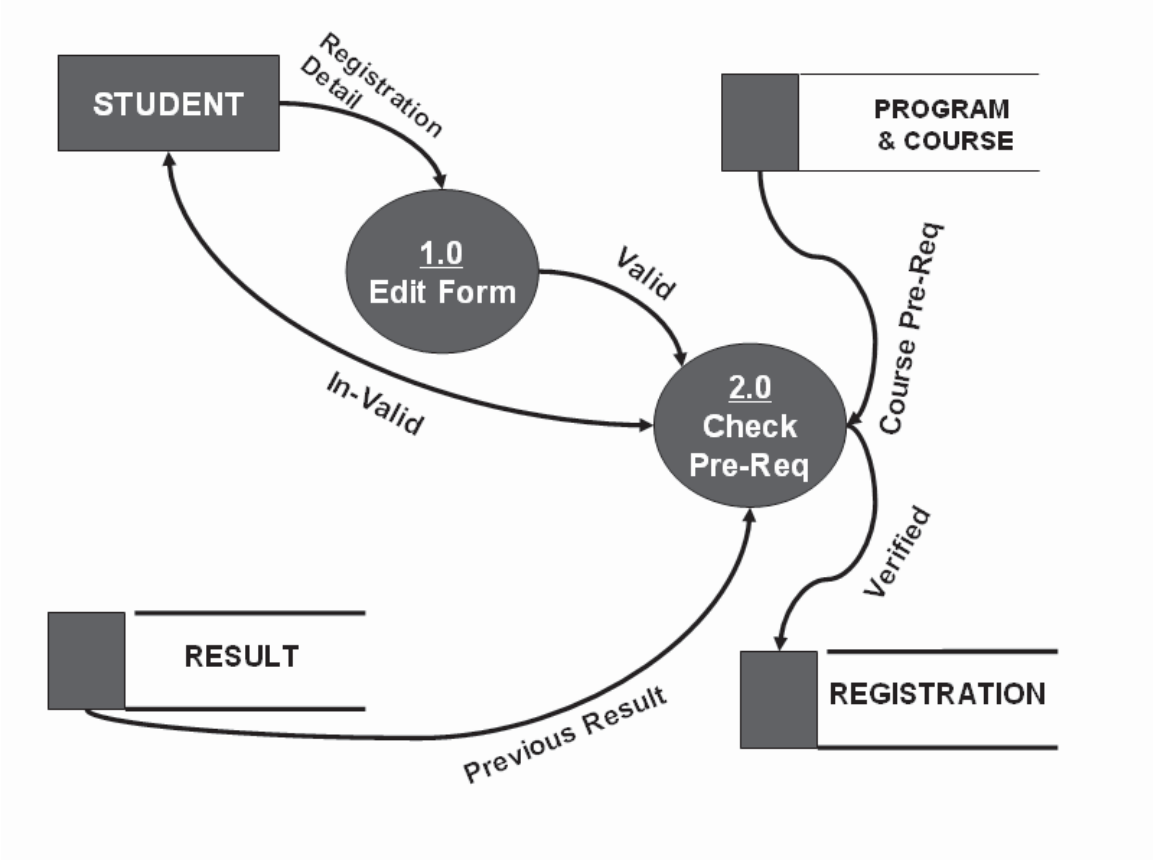



Fig 2

The first module identified in the system is the Registration of the students for the system As the DFD show a student applies for registration along-with certain registration information which is required by the system, Process 1.0 of the system checks the validity of information in the form if the Registration form is found to be valid the information in the form is passed onto the second process where the validity of registration is determined by checking certain prerequisites for the courses to which student wishes to be enrolled. After the prerequisite checking the data of the student is stored in a registration database for use by other processes in the system.

During this process the result of the students is also checked for the previous semester or previously studied subject to confirm whether the student has passed a certain pre-requisite subject before he can attempt to enroll for a second course which is based on that prerequisite.

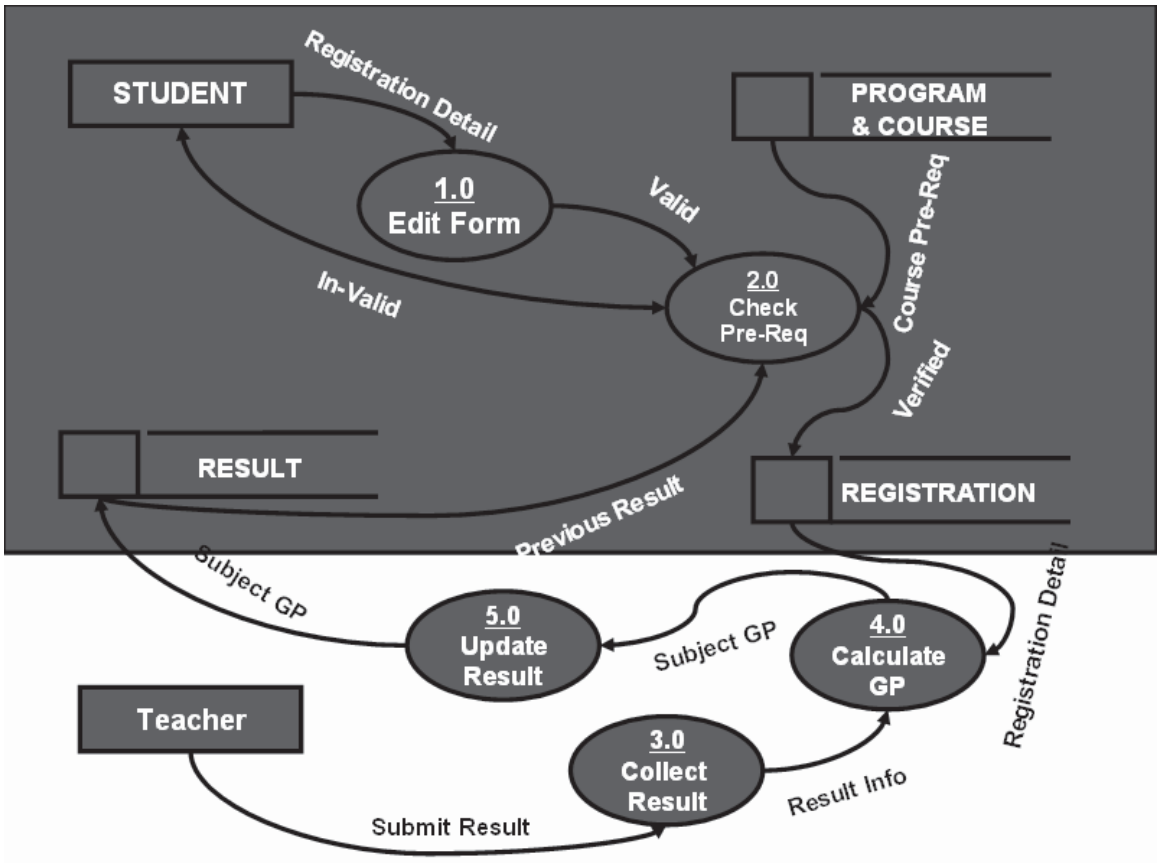



Fig-3

The Second DFD is in fact combination of the last diagram and some new details to the DFD this portion adds the result submission to the whole process of the system The teacher is the external entity here which is submitting the result, the result collection process is numbered 3.0, result is submitted by the teacher in parts, i.e. –separately for assignments, quizzes, tests, sessional and final result. The Collection process then forward the collected result to the Calculate GP Process, this process calculates the Grade point for the subject, the result with GP calculated is then moved forward to the update result process which then makes a change in the result data store by updating the result data for that specific student.

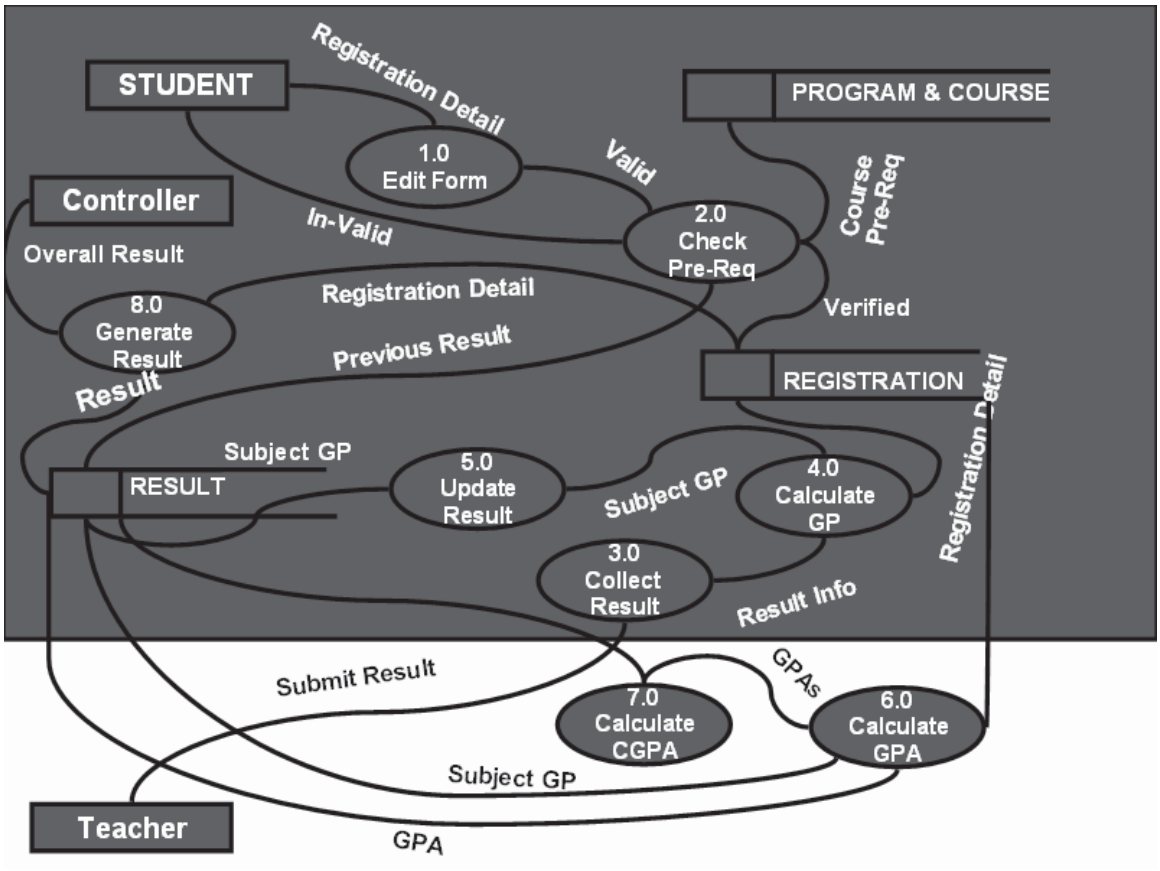



Fig-4

After the process of result submission the result for all the subjects is taken and the GPA is calculated, once the GPA is calculated the it is used for further calculation of CGPA and is forwarded to another process which is numbered 7.0 this process will calculate the CGPA by taking all the results of the current and previous semesters.

Further detailed diagram i.e.—Detailed DFD can be created using the given level 0 DFD and by expanding all the Processes further.

**Cross Reference Matrix: doth:**

This matrix is used to find out that what values or attributes will appear in which reports, for this purpose we write the major item names on a matrix in the row wise order and the reports which will be generated will be written on top or in column wise order.

Cross Reference Matrix

| | Transcript | Semester Result Card | Attendance Sheet | Class Result (Subject Wise) | Class Result |
|---|---|---|---|---|---|
| Course_Name | ✓ | ✓ | | ✓ | |
| CGPA | ✓ | ✓ | | | ✓ |
| Date | ✓ | ✓ | | ✓ | ✓ |
| F_Name | ✓ | ✓ | | | |
| NameOfStudent | | ✓ | | | |
| NameOfProgram | | ✓ | | | |
| Reg_No | ✓ | ✓ | ✓ | ✓ | ✓ |

This process infact is just cross link So the first Item transcript which may be or it will be needed by a specific student, second is Result card, next is attendance sheet then we have Class result (Subject wise) and finally the Class result as a whole, here by subject wise class result means that all the results of a specific class for a specific student considering each component, such as assignments, quizzes, sessional and terminal results.

Similarly all the mentioned items are marked with a tick which may needed by a certain output.

Let us see how the DFD and CRM are used in creating the ER-Diagram

The process of Creating ER-Diagram in fact lies in the Analysis phase and is started with identifying different entities which are present in the system. For this purpose we can use the DFD first of all.

Lets check our DFD, from there we can find the following entities.

| | |
|---|---|
| Student | Controller |
| Courses | Teachers |
| Courses Offered | Programs |
| Registration | Results |
| Semester | |

Here the point to be noted is that, we have picked the controller as the entity, although the controller is acting as an external entity for providing or getting information from the system, but in case of ER-Diagram the controller can not be represented as an entity because there is only one controller in any examination system and for such an entity instances a complete entity is not used.

So in this way we can exclude the controller entity, we will also take care of other entities before including them in our ED-Diagram. Another such example is results, which may not be as it is, added to the ER-Diagram, because there can be a number of result types at different stages of the Process, so there will be a number of different results.

We use our CRM in creating the ER-Diagram, because when we see the CRM, it has a number of item/attributes appearing on it, now from there we can see that whether these items belong to the same entity or more than one entity. And even if they belong to multiple entities we can find the relationship existing between those entities.

Considering our CRM we have transcript, it has a number of items appearing on it , as we know that there is to appear result for each semester on the transcript. So the attributes which belong to the personal information of the student shall be placed in the student entity and the data which belongs to the students' academic data will be placed in the courses or results entity for that student.

In the next phase we have to draw different entity type and the relationship which exist between those entities.

***These we will discuss in the next lecture that how we draw relationships between different entities.***

# Lecture No. 13

## Reading Material

| | |
|---|---|
| **Case Study** | |

## Overview of Lecture

- E – R Diagram of Examination System
- Conceptual Data Base Design
- Relationships and Cardinalities between the entities
- In the previous lecture we discussed the Preliminary phase of the Examination system. We discussed the outputs required from the system and then we drew the data flow diagrams DFDs. From this lecture we will start the conceptual model of the system through E-R Diagram.

## Identification of Entity Types of the Examination System

We had carried out a detailed preliminary study of the system, also drawn the data flow diagrams and then identified major entity types. Now we will identify the major attributes of the identities, then we will draw the relationships and cardinalities in between them and finally draw a complete E-R Diagram of the system.. So first of all we will see different attributes of the entities.

**Program:**
This entity means that what different courses are being offered by an institute, like MCS, BCS etc. Following are the major attributes of this entity:-

- **pr_Code.** It can be used as a primary key of the entity as it would always be unique for example MBA, MCS, etc.
- **max_Dur** It means that what is the maximum duration of any particular course , like 1 year , 2 years and so on.
- **no_of_Semesters** How many semesters this program has like four ,six and so on.
- **Pr_Lvl** This course is of undergraduate, graduate or post graduate level.

**Student:**

Following are the major attributes of this entity:-

- **Reg_No** This can be used as a primary key for this entity as it will be unique for every student.
- **st_Name** This would be the first name of all the students of an institute.
- **St_Father_name** This would represent the father's name of a student.
- **St_date_of_Birth.** The date of birth of all students including year , month and day.
- **st_Phone_no**
- **st_GPA** This is a very important attribute. Now to know the GPA of any student, we need to know the student reg no and the particular semester. So this is a multi valued attribute as to know the GPA, different attributes values are required. So this represented by a relation, which will be discussed in the relationships in between entities.
- **st_Subj_Detail** This is also a multi valued attribute ,as to know the marks in mid terms and final papers , student reg no and the particular subject are required

**Teacher:**
Following are the major attributes of this entity: -

- **teacher_Reg_No** This can be used as a primary key for this entity as it will be unique for every teacher.
- **teacher_Name** This would be the first name of all the teachers of an institute.
- **teacher_Father_name** This would represent the father's name of a teacher.
- **Qual**. The qualification of a teacher like Masters or Doctorate.
- **Experience** This can also be a multi-valued attribute or a single valued attribute. If only total experience of any teacher is required then it can be single valued, but if details are required as per the different appointments, then in that case it would be multi valued.
- **teacher_Sal** The total salary of the teacher.

There is one thing common in between teacher and student an entity that is the personal details of both, like name, father's name and addresses.

**Course:**
Following are the major attributes of this entity: -

- **course_Code** This can be used as a primary key for this entity as it will be unique for every course like CS-3207.
- **course_Name**
- **course_Prereq** This would also be a multi valued attribute as there can be a multiple requisites of any course . For example, Networking can have pre-requisites of Operating System and Data Structures. In this case this is a

recursive relation as pre-requisite of a course is a course. We will treat it as a recursive relation here.

- **Courses_Offered_in** This is also a multi valued attribute as to know the courses offered , program and semester both are required so this can also be represented by a relation

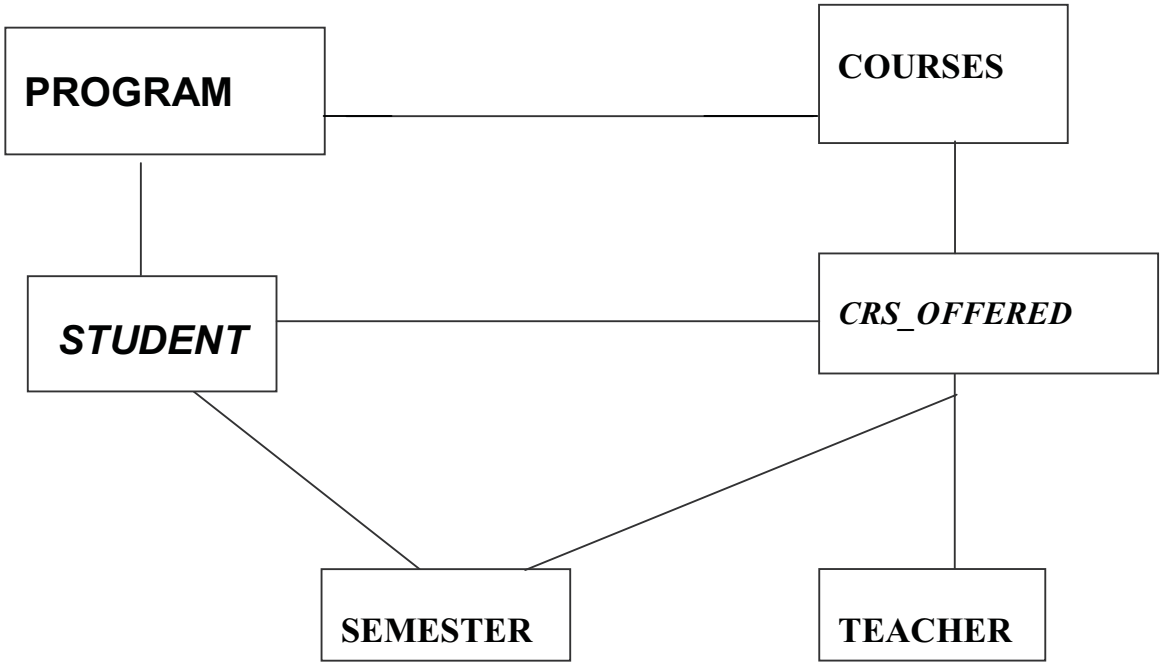
**Semester:**
Following are the major attributes of this entity: -

- **semester_Name** This can be used as a primary key for this entity as it will be unique for every semester like fall 2003 or spring 2004.
- **semester_Start_Date** The starting date of the semester
- **semester_End_Date** The ending date of the semester

**Derived Attributes**
There are certain attributes in the examination system which is derived like CGPA of a student can only be achieved from the semesters GPA. Similarly FPA of any particular semester can be achieved from subjects GPA of the semester. So this has to be kept in mind while drawing the E-R Diagram of the system.

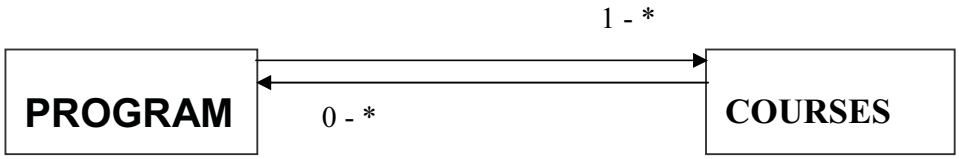
## Relationships and Cardinalities in between Entities

Relationships and cardinalities in between entities is very important. We will now see the relationship of different entities one by one. The block diagrams of different entities are as under: -



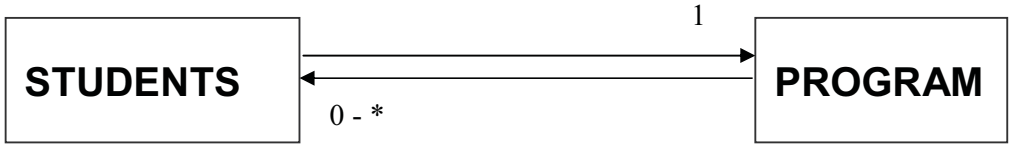
**Program and Courses**

The relationship between program and courses is that, if we want to know the courses in any particular program then the course codes and program codes are required. The cardinality between program and courses is of one to many (1 -*) , which means that any program will have minimum one course, and as many as required for that particular program. The cardinality of courses and program can be zero to many (0 - *). It means that if an institution wants, it can have a course without even any program as well. This course can be included in any program later on.




**Students and Programs**

The cardinality in between student and program is one, which means that every student can have minimum and maximum one program at any time. The cardinality in between programs and students can be zero to many (0 - *), which means that depending upon the requirements of any organization it can have a program which is presently not being offered to any students.




**Semester and Course**

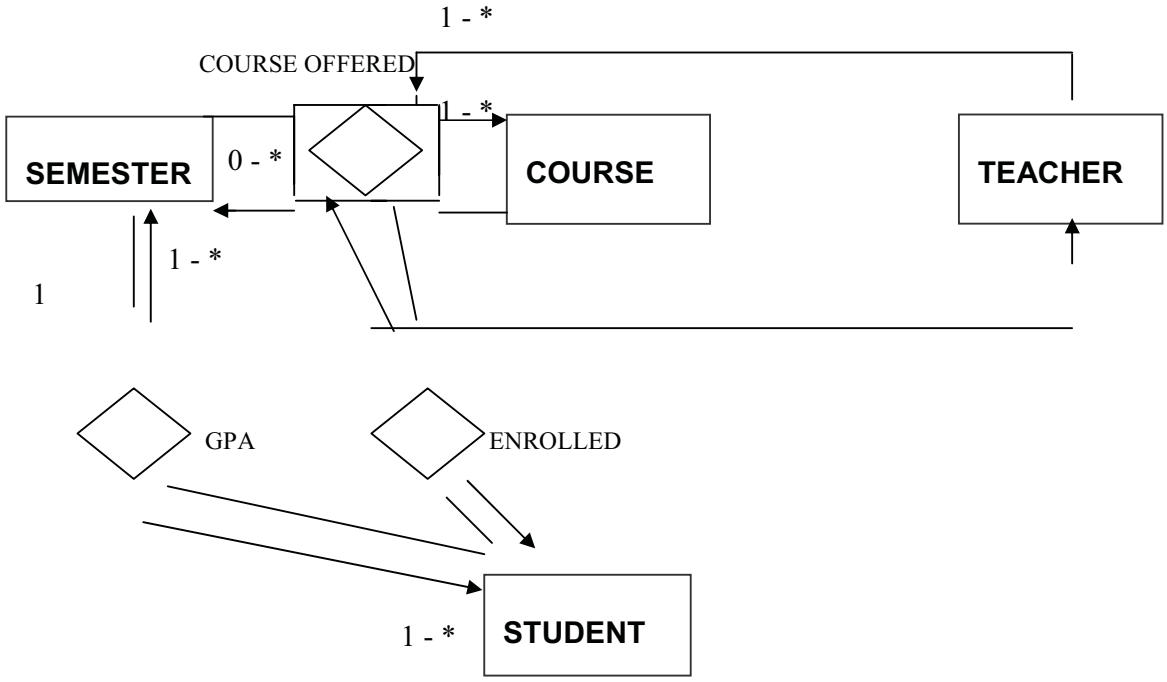
The relationship in between semester and course is many to many. But it is essential to know the course offered during any particular semester so there is a requirement of an attribute, which is of relationship and when it is many to many it, can also serve as entity which is represented by a diamond in a rectangle. So here this can be a courses offered attribute, which would also be an entity. The primary key of semester that is semester code and primary key of course that is course code, after combining it becomes composite key which would be used to identify any particular course.

**Course Offered and Teacher**

There is a relationship in between course offered and teacher. The cardinality of course offered and teacher is one that is a teacher can only have one course at a time. Similarly the cardinality in between teacher and course offered is one to many, which means that a teacher can teach many courses.

**Student and Course Offered**
The relationship in between student and course offered is many to many so this relationship is also through enrolled attribute, which can also serve as entity type. The primary keys of semester, course and student are used as composite key to identify, that which student has got which course.

**Semester and Student**
To find out GPA of any student the semester is also required to be known. So the relationship in between these two can be through result whose attribute GPA can be used. There is a many to many relationship in these two entities.




## Conceptual Database Design

The outcome of analysis phase is the conceptual database design, which is drawn through E-R model. This design is independent of any tool or data model. This design can be implemented in multiple data models like network, relational or hierarchal models.

## Logical Database Design

This is the next phase of database design, in which appropriate data model is chosen, and from here onwards it becomes tool dependent. We will select relational data model and our further lectures will be concerning relational data models

## Conclusion

The E – R Model of Examination system of an educational institute discussed above is just a guideline. There can certainly be changes in this model depending upon the requirements of the organization and the outputs required. After drawing an E-R model, all the outputs, which are required, must be matched with the system. If it does not fulfill all the requirements then whole process must be rehashed once again. All necessary modifications and changes must be made before going ahead. For Example, if in this system attendance sheet of the students is required then program code, semester and course codes are required, this composite key will give the desired attendance sheet of the students.