Database Management System
Dr. S. Srinath
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 2

Conceptual Design

Greetings to you all. We have been talking about conceptual modeling of databases in previous session. So in this session let us continue with this subject into some more detail and before we continue or before we start with today's session, let us have a brief review of what we looked into conceptual modeling in the previous session. If you remember we first talked about what is a typical database process or design and development process looks like.

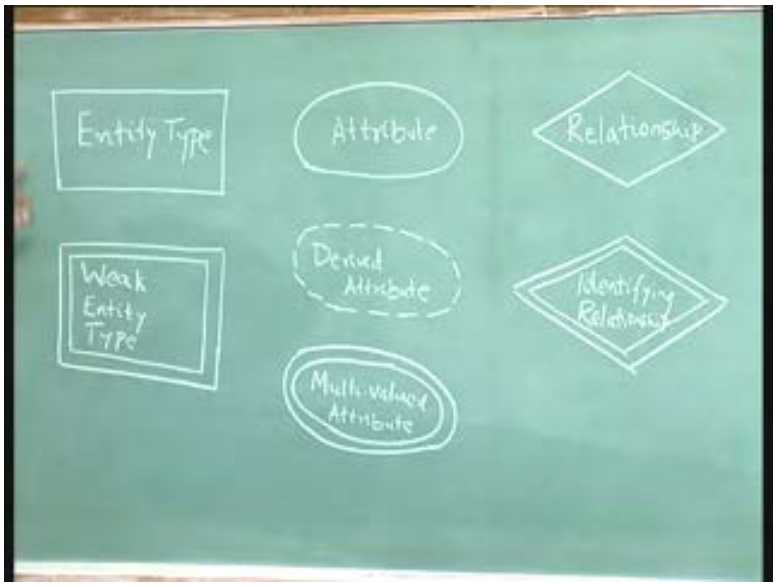
We first start with analyzing the UoD or the universe of discourse and the analysis of the UoD reveals us two different kinds of requirements. One is the data requirements and the second is the set of process or application requirements. So the database runs with in an application context like I gave the metaphor in the previous session, a database is like an engine and the application context is the overall body or the car or bus or whatever that you build on around the engine, so both are important when we are trying to analyze the UoD.

So coming to a database analysis or design of the database system, we first start with a high-level description of what the database should handle and this high-level description should not include any DBMS specific terms or DBMS specific issues. It is mainly meant for the end users, its mainly meant for us to show the end user saying this is what we have analyzed or this is what we have understood by analyzing your universe of discourse and it is mainly meant as a conceptualization of what are all the data requirement that exist in this domain.

So one of the most popular models for performing this conceptualization or building a conceptual schema is what is called as the ER diagram or the ER model, the entity relationship model. We saw that entity relationship modeling are made up of essentially two kinds of building blocks, entities and relationships. So we looked at different nuances of entities and relationships and several issues that affect them like constraints and attributes and so on.

So let us briefly describe some notation, this is not exhaustive review but review of some of the main points what we looked at in the entity relationship diagram.

(Refer Slide Time: 00:03:46)



An entity type is described by a rectangle like this, a simple rectangle and an entity type is something which represents a class of entities or objects that have an independent existence like a customer is an entity or a staff is an entity, account is an entity in a bank or in a company a department could be an entity, manager is an entity and so on. Any logical U-net that has an independent existence is called an entity and an entity type is an intension or some kind of a schema for a class of entities or for a set of entities. There could be a set of different managers but all of them share the same attributes or the same properties of the entity type called manager.

Similarly there could be several departments but all department share the same properties of the entity type called department. And then entities are associated with attributes which describe the characteristic of the entity and we saw that attributes are in turn defined by the domains. For example the age of an employee is defined by a domain that it should be greater than or equal to 18 years and less than or equal to 65 years typically. So a domain represents a space within which an attribute lies. And even within attributes we saw that there are several kinds of attributes, you could have a normal attribute, a simple attribute, an attributes which takes just one value or you could have a multi-valued attribute, we gave the example of the color of a peacock.

It doesn't have one color, it has many colors and so it could have different values for the same attribute. An attribute could be a composite attribute that is it could contain many sub attributes like first name, last name, middle name and so on. It could be a derived attribute like the age of a person which can be derived, if you know the date of birth of the person and the present date.

So there are several different of attributes, so all entities of a given type have the same set of attributes. Now we also saw what are called as key attributes or what are called as keys. I have not depicted this here in these diagrams but key is some set of attributes that

can uniquely identify an entity within an entity type. So if I have a set of employees and suppose employees are given an employee identification number, the employee identification number becomes the key. If they don't have an identification number, we usually have something like the income tax permanent account number called the pan number or something which forms the key, it uniquely identifies a person.

On the other hand we can't use something like name or age as the key. Two persons can have the same name and of course two or more people are very likely to have the same age. So it cannot uniquely identify a particular entity whereas something like the identification number identifies the entity uniquely. So key attributes identifies this entity types uniquely and there need not be just one key attribute, there could be more than one key attributes and like I gave the example of I need not have just one key to my house, I can have two keys, one for the front door and one for the back door. But we usually use just one of the keys which the default basis and sometimes use the other key frequent and then we talked about what is meant by a weak entity type.

A weak entity type is a kind of entity type which does not have a key attribute. We also saw an example yesterday, the example of an employee and his or her insurance record. So an employee has a key attribute, the employee number or pan number or whatever but the insurance record does not have any existence without a corresponding employee or without a corresponding person to be more general without being associated with a corresponding person.

Now once an insurance record is associated with a particular person whatever way, whatever we use to identify the person becomes the key to identify the insurance record as well. So for example identify an insurance record with a person having a particular pan number, I can as well identify the insurance record with the same pan number as I am identifying the person and we saw the second building block which are called relationships between attributes.

So a relationship basically ties in or brings in an association between two or more entity types. So even within relationship there are several different kinds of relationships and we saw certain constraints that identify some kinds of relationships. One of the first thing we saw was the cardinality constraints, a relationship which has a cardinality constraint says that, says how many entities of a particular type can participate in a relationship. For example we saw that department managed by manager. So it could be, there could be a constraint that a department may have just one manager or one head and one person may manage at most one department at a time.

So there is a cardinality constraint that exactly one, one department may be managed by exactly one manager. On the other hand if I have something like works in, an employee works in department. So it's rather than a 1:1 relationship, it is a 1: n relationship. One department may have n employees and there could be a constraint that an employee may be associated with just one department so that for n employees there are just one department so on.

And we also saw what is called as an identifying relationship. An identifying relationship is the one, is a relationship that identifies a weak entity type with a strong entity type. That is if I have an employee and says insurance has this insurance record using some kind of a relationship, it means that this relationship is giving an identification or an identity for this weak entity type called insurance record. So such relationships are called identifying relationships and we are also saw what is called as a total participation within a relationship.

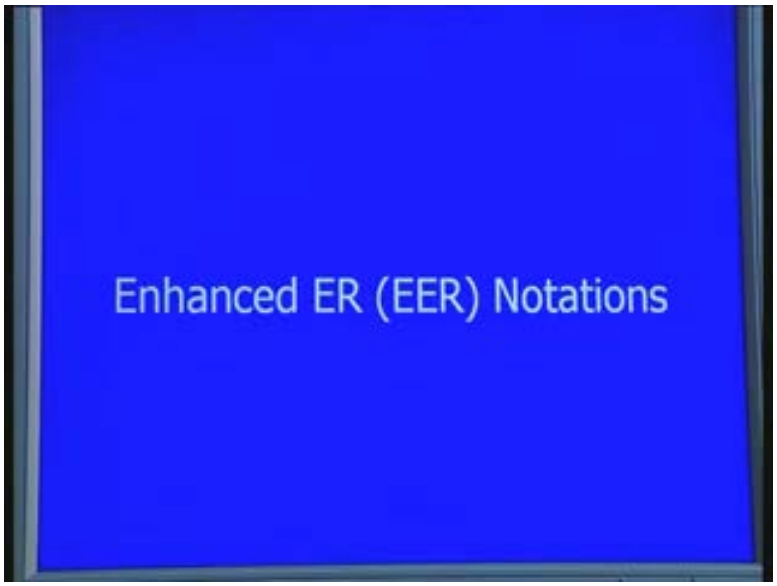
So the same thing here, an insurance record will have no existence without its corresponding employee or without a corresponding person. So this insurance record is set to totally participate in this identifying relationship. However we also saw that just because there is a total participation does not mean that the entity type is a weak entity type. How do we, what is an example for that? We saw an example of department managers project.

Let us say that a project has to be associated with a department otherwise well, it will not get funding, it won't get off the ground or something like that. So in this case this is a total participation that is the existence of project, the existence of projects will depend upon this relationship that is the existence of some department that is willing to manage this part.

However a project by itself need not be a weak entity type that means it may have a separate key by itself, a project will have its own separate project identification number which may have no relationship with the department number. Not every total participation implies identifying relationships but an identifying relationship implies total participation.

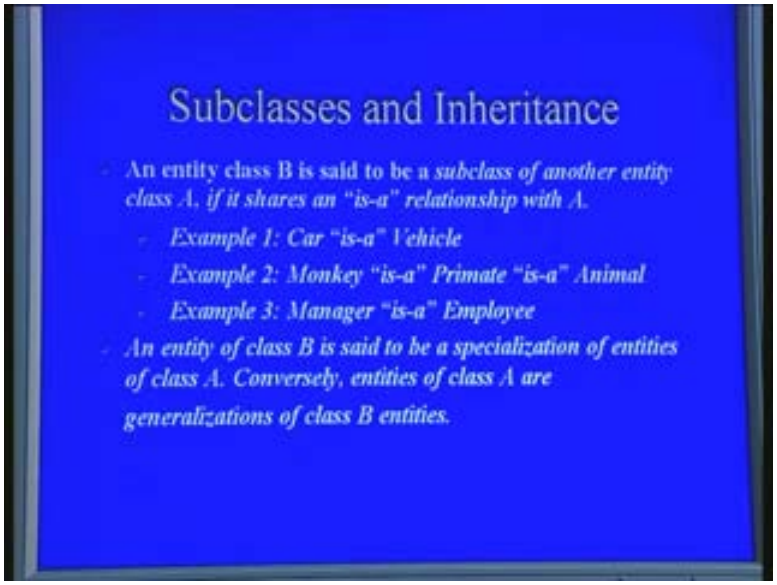
So we will be continuing further on today with some more notations and which can give us greater expressiveness to express what we perceive as relationships and associations between data elements in our UoD.

(Refer Slide Time: 00:11:47)



Now these kinds of, these sets of notations that we are going to see today are what are called as enhanced ER notations or sometime also called as a extended ER notations and abbreviated as EER notations.

(Refer Slide Time: 00:11:56)



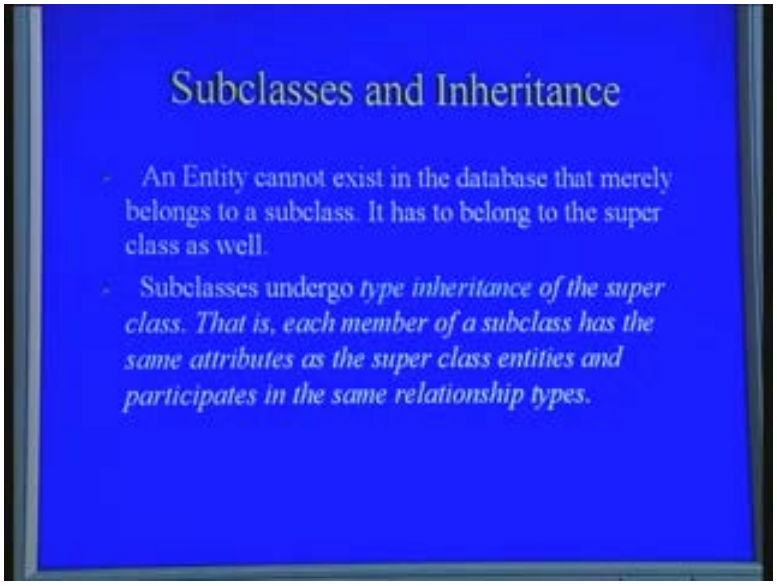
One of the first relationships or associations that we are going to see today is the notation of subclassing or inheritance. Subclassing essentially face what is called as an is-a relationship as you can see in the slide here. An entity class B is said to be a subclass of another entity class A if it shares an is-a relationship with A.

What is meant by an is-a relationship? Have a look at these examples. A car is-a vehicle, a monkey is-a primate and a primate is-a animal or is an animal whatever. A manager is-a employee. So if you have noticed here an is-a relationship identifies a specialization of some particular entity type. A car is-a vehicle but not all vehicles are cars, there could be trucks, there could be bicycles, there could be scooters and so on. So vehicle is a more general class of cars.

Similarly if you say a maruti 800 is a car but not all cars are maruti 800's. So car is a generalization of maruti 800 and maruti 800 is a specialization of the entity type called car. Similarly a monkey is-a primate and a primate is-a animal but not the other way. So the entity class B that is at the left hand side of the is a relationship is set to be a specialization of entity class A or on the other hand entities of class A are set to be generalizations of entities of class B.

So what are the properties of generalization and specialization? Have a look at this slide and there are some interesting properties when we are talking about a generalization and a specialization relationship. Now suppose I have an entity of type car. So let us say this entity with a particular registration number is a car that exists in the database. Now in the database I also have some entities of type vehicles. Now should this car entity type belong in the vehicle entity type? If you think about it carefully, you will say that the answer is yes because the simple reason is that a car is a vehicle.

(Refer Slide Time: 00:13:46)



If I have an entity that exists in the set of cars, the same entity should also exist in the set of vehicles. So an entity cannot exist in the database that merely belongs to a subclass, it also has to belong to the super classes and subclasses undergo type inheritance of the super class. What is meant by type inheritance of the super class? Again notice carefully here. Now let us say that we are going to describe some properties of vehicles. Now what kinds of properties can we think of vehicles? Vehicles will have wheels. Vehicles will

have I mean depending on what you call vehicle I mean you could also called a rocket as a vehicle.

So assuming that we are only looking at road vehicles. Suppose I say that a vehicle is represented by wheels and it should have some kind of controlling mechanism, it should have a driver seat or something of that sort and it should move. Now you see that all of these characteristics apply to all subclasses whether it is a car, a bicycle or a truck or a van or whatever all of these have to move, they have to have wheels, they have to have some kind of a control mechanism whether it's a handle or a steering wheel or anything like that and so on.

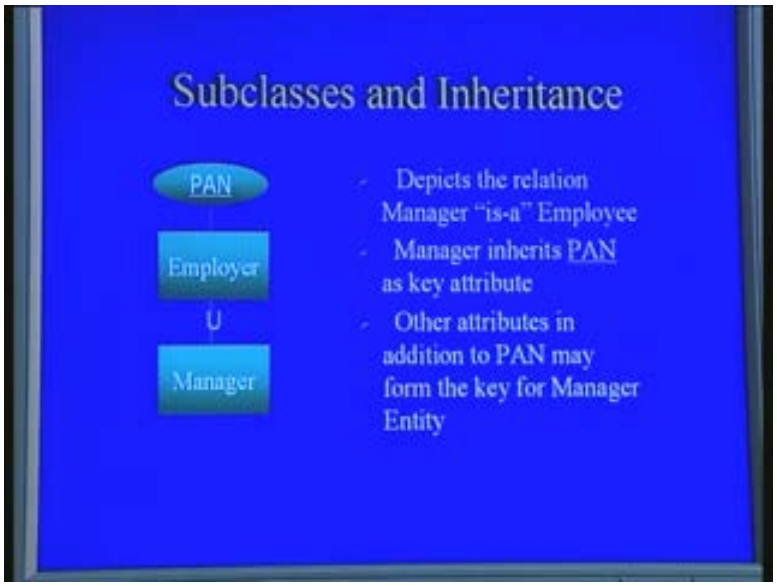
So, all attributes that describe a general class has to be inherited by the special classes, that the specialized class. Each member of a subclass has the same attributes as that of the super class entities and participates in the same relationship types. The second aspect is also important, if a general class entity participates in a particular relationship type, a specialized class entity should also be able to participate in the same relationship type. That is if I can use a vehicle to go from point A to point B, I should be able to use a car to go from point A to point B or I should be able to use a truck or a bicycle or whatever.

So if there is a relationship that exists between vehicles and let us say an employee using that vehicle, you should be able to replace this vehicle with any of the subclasses and the semantics should not change. The semantics of the entire database system should not become incorrect that's why said we are looking at road vehicles I mean it's a matter of naming the particular entity, a car is a road vehicle.

Obviously if I also consider rockets and airplanes as vehicles, this doesn't hold anymore. Using a rocket let us say I can go from here to the moon but I can't do that using a car. So you won't be able to replace a subclass, subclass entity wherever a super class entity arises. So when you are coming out with inheritances and special generalization and specializations in your database, you should be aware of the fact that this type of, this kind of replacements of general class entities with special class entities should be possible that is what establishes a correct inheritance relationship with a incorrect inheritance relationship.

So here is another example suppose I say that manager is an employee or a manager could also be an employer. So as you can see here the inheritance relationship is depicted by a U in the relationship type. That is there is a straight line with a U which represents an inheritance relationship or a specialization relationship.

(Refer Slide Time: 00:17:56)

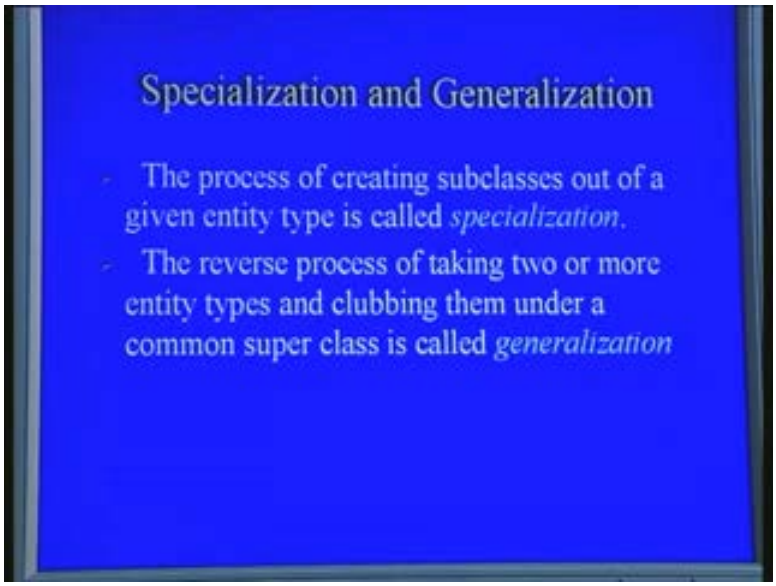


If you note that if this is an employee here and an employee is uniquely identified by the pan number, the same pan number can uniquely identify the manager as well. So what is that mean? It means that any key attributes that uniquely identifies, that uniquely identify entities of a super class or a more general class can also identify attributes of the special classes.

However special classes may have some more attributes in addition to the key attributes that the general classes have. For example a manager may also have one more identification which says for which department is a manager or what is a scale or whatever. Now in order to be able to identify one manager from another uniquely, you may have to combine that attribute with a pan attributes here. So there may be other attributes that form the key for the special classes but all the key attributes of the general class has to be retained in the special classes.

So another example of type inheritance, in this case it's more of a key inheritance the key has to be inherited directly. The process of creating subclasses out of a given entity type is called specialization.

(Refer Slide Time: 00:19:42)



That is suppose I have a particular entity type, suppose I have identified that the UoD here requires vehicles. Now out of these vehicles I identify that they require vans, they require cars, they require trucks and so on. And then I also identify that van is a vehicle and a truck is a vehicle and a car is a vehicle and so on. So I should be able to form what is called as an inheritance tree. So this process is called specialization.

On the other hand its also possible that we go in the reverse fashion. We first look at the UoD, we first go through the company, talk to people and see what is happening and then we identify different entities. We see that the company uses cars, the company uses buses, the company uses motorbikes, the company uses trucks and so on. The company uses vans and then once we have listed all of these we start seeing relationships among them, we say that all of these are vehicles and all of these share the same attributes as far as the company is concern. And then we put all of them in an inheritance tree and this is what is called as a generalization.

Now before we go to the next slide, let me inter check here to note that this is not such a straight forward process, which entity is a special class and which entity is a general class is not such a straight forward process. Sometimes depending on the usage context which becomes a general class and which becomes a special class, a specialized class may change from one context to the other.

Let me give a particular example. Take two entity types an airplane and a glider. Now which is correct? The first one which says a glider is an airplane without engines or whatever, a glider is a airplane or an airplane is a glider. Which is correct? So if you look at it carefully, let us go back to what are the properties of specialization and generalization classes.

The first property of specialization classes is that wherever I am planning to use the generalized class objects, I should be able to use a specialized class objects. So is that always true, wherever I use airplanes can I use gliders? Maybe or maybe not, I mean depends, it depends on the context and secondly whatever attributes that the generalized class has, has to be inherited by the specialized class.

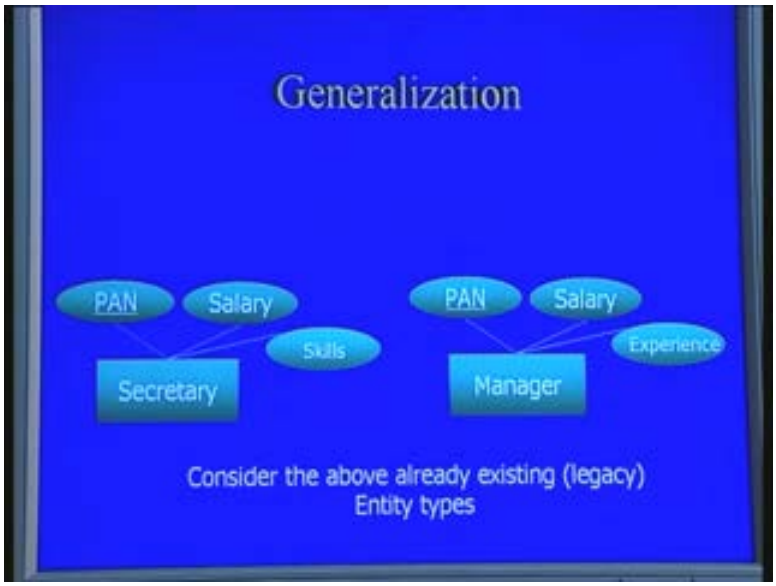
Again this seems to say that let us say an airplane has several different attributes, it has engines, it has wings, it has wheels, it has controls and so on. A glider also has all of them except that it doesn't have an engine. So it seems to suggest that an airplane is a glider is a correct one. So glider is more general and an airplane is more specific, so because a glider has a smaller number of attributes and an airplane has a larger number of attributes.

However look at it in the context of learning how to fly a glider or learning how to fly an airplane. Now if you see that let us say I have different paragraphs about or different kinds of skills that I have to learn for flying an airplane and for flying a glider. It could well be the case that depending on the sophistication of the airplane, there are some airplanes here where you don't have to do anything, you just have to go and plan your journey and push a button and it will take you there with all auto pilots and so on and so forth and so on.

So depending on the context, you may actually have to learn more to fly a glider than to learn than to fly an airplane. So if the number of attributes or the different kinds of skills I need to fly this, you see that the opposite inheritance tree is valid that is an airplane that is a glider is an airplane that is an airplane requires smaller number of skill sets to fly while a glider requires a larger number of skill sets to fly therefore a glider is a subclass of airplane.

So as you can see that it is not such a straight forward thing to identify is-a relationship. So it depends on the application context and we should not ignore the application context like we saw in the previous session, a database cannot ignore the information system context within which it is going to be run. Is the application context about building an airplane or a glider or is a application context about flying an airplane or a glider. Now that much change the inheritance semantics in our conceptual schema.

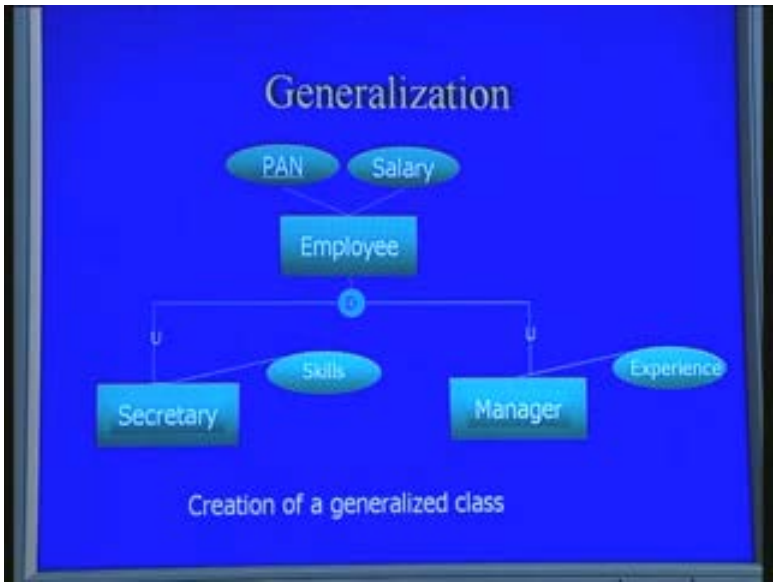
(Refer Slide Time: 00:24:57)



So coming back to specialization and generalization processes, let us take a small generalization example and see how we go about it. Let us say we have identified two entity types in our, back to our company database. So let us say we have identified an entity type called secretary and the secretary is identified by a pan and salary and the kinds of skills the secretary has typing short hand or whatever so on and so far.

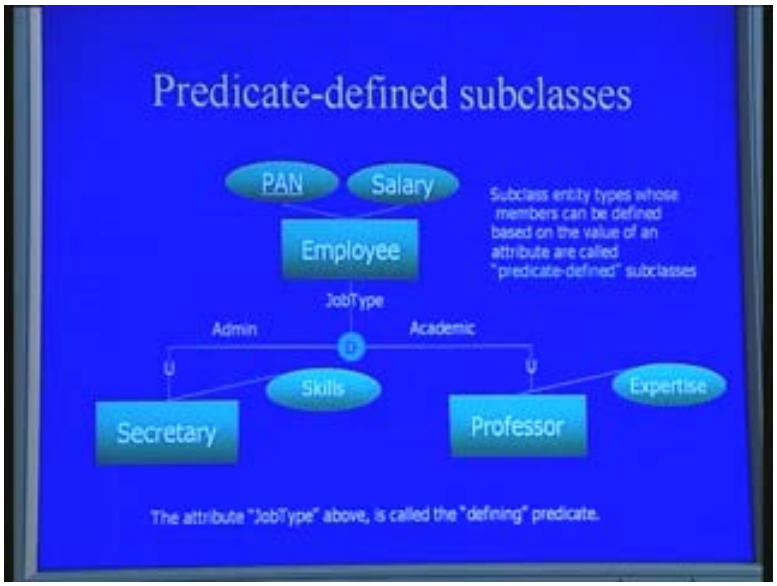
Similarly we have identified manager and we see that manager also has a pan number. A manager also has a salary and there is an experience field saying what kind of experience the manager has. Now when we see two or more entity types sharing the same kind of attributes for a large extent that is out of three attributes two are similar here. It gives us reason to believe that probably these two are special cases of the same general class. So we can generalize them something like that.

(Refer Slide Time: 00:25:59)



So we can create a more general class let us say called employee and then say secretary is a employee and manager is a employee. So note that the attributes that where first a part of secretary and manager have gone here that is only the common attributes between secretary and manager have moved up the hierarchy to go to the employee class and all those attributes which are specific to this specialized classes remain in the specialized classes that is skills remain here and experience remains here. Now in some cases it maybe able to, we maybe able to identify precisely how to distinguish one special class to specialized class to another specialized class, have a look at this slide here.

(Refer Slide Time: 00:26:41)

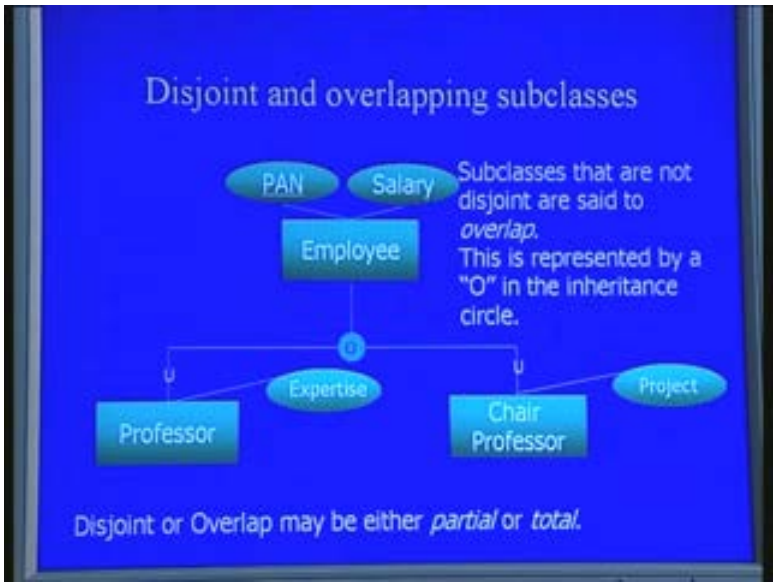


This slide show an entity type called employee which is defined by attributes called pan and salary and one more attribute called job type which is not actually shown here. Now there are two specialized classes secretary and professor. Now suppose we identify a property that every professor has a job type as academy and every secretary has a job type called admin.

So each professor belongs to a category of academic jobs and each secretary belongs to a category of administrative jobs. So we know exactly how entities of one specialized classes can be distinguished form entities of another specialized classes. So this is how we identify this here, we say job type and then we say admin is secretary and academic is professor. So such kinds of definitions are what are called as predicate-defined subclass. These subclasses are defined by the values of one or more predicates that exist in the ER schema.

Next we go to an example where we see that in some cases not all subclasses maybe unique. Now let us take back the example of secretary and professor. You see here that while denoting this subclasses, we have drawn a circle with a small D here. Now what is this D denote? This D denotes the fact that these two subclasses are disjoined. What is meant by disjoint here? That is they are mutually exclusive, no secretary is a professor and no professor is a secretary because all secretaries have to have a job type as admin and all professors have to have a job type of academic. So the set of all secretary is a disjoined set from the set of all professors.

(Refer Slide Time: 00:28:59)

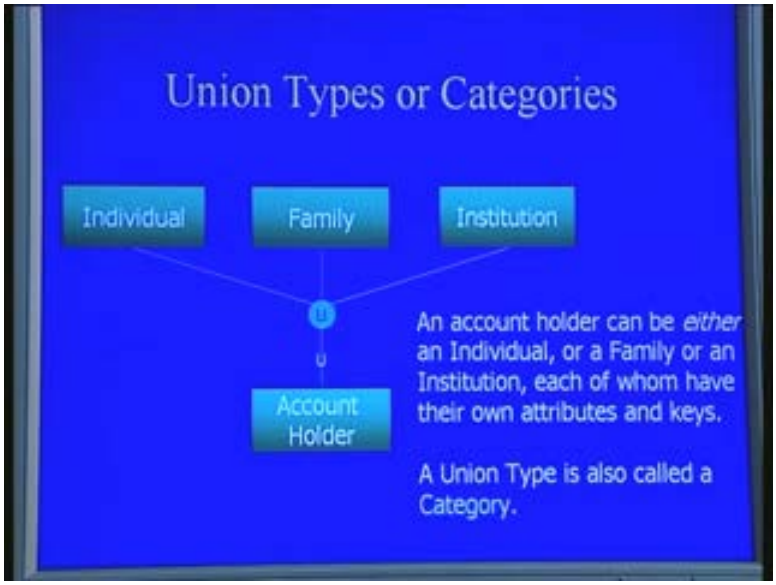


But this need not always be the case, sometimes two or more specialized classes or specialized entity types may actually overlap they need not be mutually exclusive from one another. For example suppose in some university there are notions of chair professors, chair professors are usually supported from external sources of, external funding sources but for all practical purposes they work as any other professors here.

Now it could well be the case that some professors are chair professors and some chair professors are normal professors that is they need not be supported by a project but they also work in other activities and so on. So such kinds of inheritance trees or subclasses are set to be overlapping subclasses. So these two subclasses need not be disjoined from one another and this overlap, this kind of overlapping maybe either partial or even total overlapping.

Now you might have, you might notice that every chair professor is a professor but not the other way around and so on. So it's a total overlap as far as chair professor is concerned. On the other hand if there are some chair professors who are not teaching, let us say who are not doing the normal activities of a professor here then the overlap is partial overlap.

(Refer Slide Time: 00:30:33)



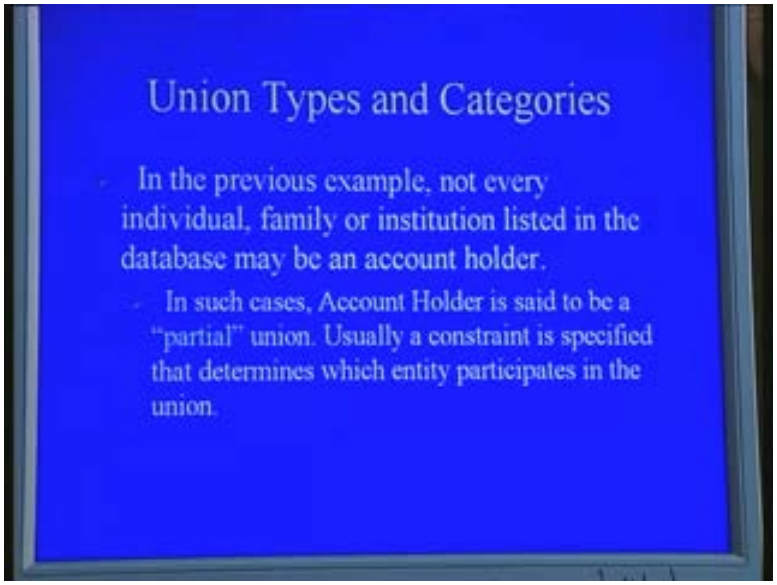
The next kind of generalization technique that we are going to do see is what is called as a union type or this is also called as a category. Now have a look at this slide a little more carefully. Now this slide shows entity type called account holder in a banking scenario. Now when you ask a banker who or what is an account holder, he will probably tell you that the account holder is just an abstraction, it is an entity, it does not necessarily represent a person because it may actually represent an institution. An institution maybe an account holder or an individual maybe an account holder or sometimes in some cases accounts maybe held by families or sometimes dynasties and so on.

So as far as the bank is concern, all of them are just account holders and they are just abstractions. But all of them share, all of them have their own set of attributes and have different sets of characteristics obviously an individual is different from an institution. Institution has characteristics like number of employees and so on which an individual which may not make sense for an individual entity type. So each of these entities here in

the top most in the top run here may have their own sets of attributes which may not be in common with one another but all of them are account holders here.

So each individual may have its own different key for example a pan number for an institution and address for a family or some kind of registration number for an institution but all of them are account holders as far as a bank is concerned. So such a kind of relationship is what is called as a union type. If you are familiar with programming, in C programming you have this notion of unions which has very similar co-notations. That is an account holder is either an individual or a family or an institution, a union type is also called a category.

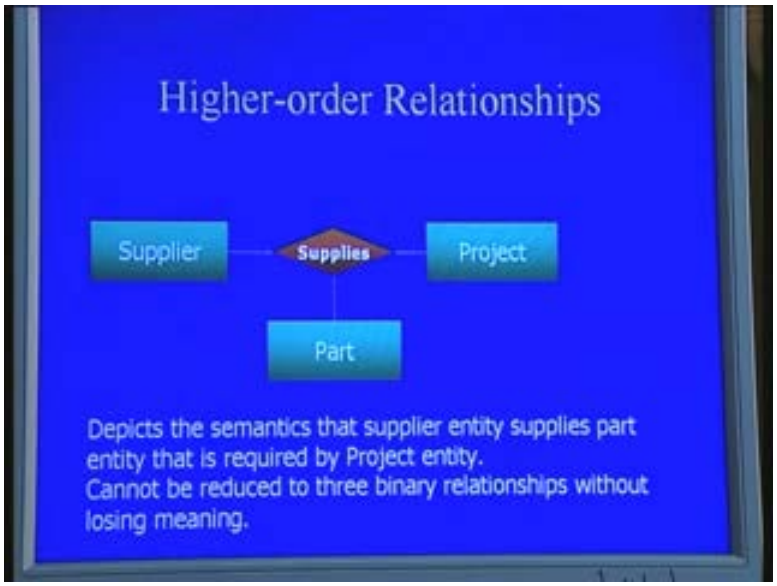
(Refer Slide Time: 00:32:51)



And just like we saw in the case of inheritance where in the case of subclassing where a subclassing, subclass could be either disjoined or overlapping we can have what is called as a partial union or a complete union. Now for example here, not every individual that exist in the database could be an account holder. An individual is an entity and an entity is something which has its own independent existence. So we may be keep in track of individuals for our own purposes but some individuals in the database could be account holders.

Similarly we maybe keep in track of institutions for some other purposes but some institutions in our database could be account holders. So when only a part of the entity set of individuals form or participate in this relationship, we call this as a partial union. On the other hand if every individual that we hold in our database is an account holder or participates in this union relationship then it's a full union relationship.

(Refer Slide Time: 00:34:04)



So that was briefly about generalizations and specializations and in fact this is a very crucial concept in being able to obtain the notion of abstraction that is to be able to abstract away unnecessary details from a special class and go to the general class. So if you are able say that we are going to use an entity type of a general class, it means that it has just enough details that is necessary for this relationship to exist.

That means if I say that I need a car for this particular activity, I don't need to worry about what kind of a car is that, what color of the car is that or what is the horse power of that car or whatever. It's all this attributes are specific to particular kinds of cars but for this particular activity any car would do. So we are essentially abstracting a way or covering up all the unnecessary details and looking at only the necessary detail what is required for a relationship to exist.

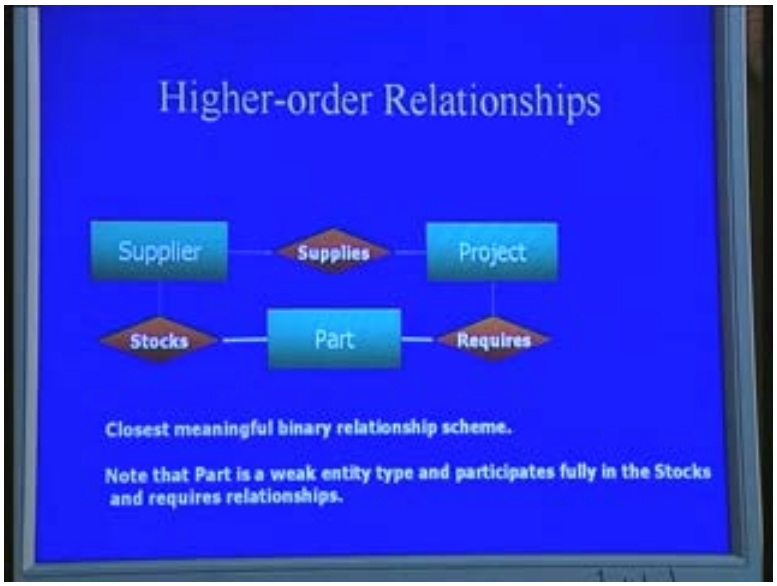
The next concept that we are going to be looking at here is the concept of higher order relationships. Until now we have been considering relationships with a degree of two. Recall that the degree of a relationship is the number of entity types that participate in this relationship. Now this slide shown here shows a relationship with a degree three. Have a look at this relationship carefully. It says that the relationship is called supplies and it relates three different entity types the supplier, part and project.

What is the relationship say or what is a semantics of this relationship again? There are three different entity types supplier, project and part. So basically it means that the supplier supplies this particular part for this particular project. Now if you think carefully it is not possible to reduce this ternary relationship or a relationship of degree 3 to any number of relationships of degree 2. A supplier may supply some parts but not all parts maybe designated for this particular project.

A supplier may supply for a project but you may not supply all parts that are required for the project. Now a project may use a certain parts but not all parts that are used by a project or may be supplied by just one supplier, there could be any number suppliers. So we cannot reduce it to three binary relationships without losing meaning. Let us try to do that and see what happens.

Now the closest possible binary relationship that tries to simulate this ternary relationship is something like this. A supplier supplies to a project, a supplier stocks some parts and a part is required by a project or a project requires certain kinds of parts.

(Refer Slide Time: 00:37:57)

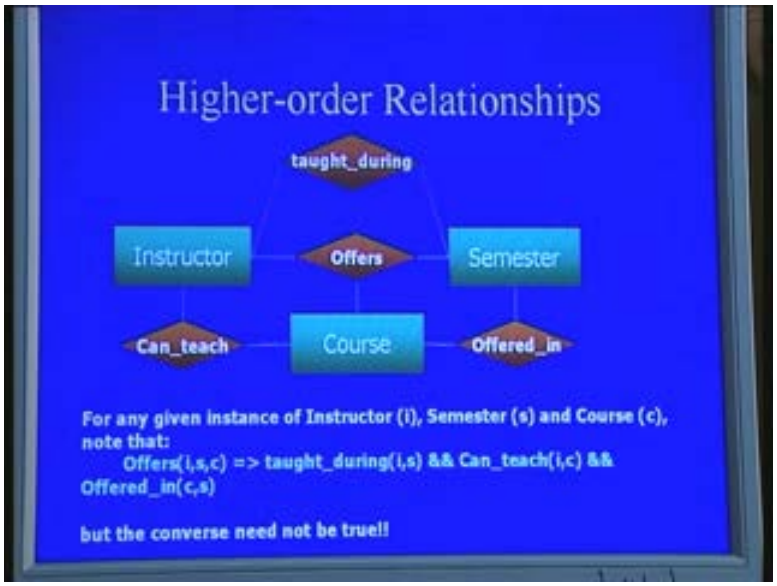


And to be fair, to be sure we also note that a part is a weak entity type, it has no existence by itself it has to be either associated with a supplier or with a project. So even when you do that this is probably the closest we can come to simulating the entity relationship but not quiet close. As you can see it can still, it just because suppler stocks certain parts doesn't mean that this part will be required by this project or vice versa.

Let us take another example of higher-order relationships and see whether we can reduce it to lower order relationships without losing meaning. So this slide here shows a relationship which is a ternary relationship called offers. So it says that instructor offers a course during a semester. Now there are also other relationships that we have identified in the database and which says that instructor taught_during certain semester or instructor can_teach a particular course or a courses offered_in a particular semester.

Now note that if I have an instance of this relationship that is offers I s c. What is i s c means? If there is an instance of this relationship called offers for a particular instructor in a particular semester for a particular course, this implies that the instructor has taught_during this semester and the instructor can_teach this course and the courses offered_in this semester.

(Refer Slide Time: 00:40:26)

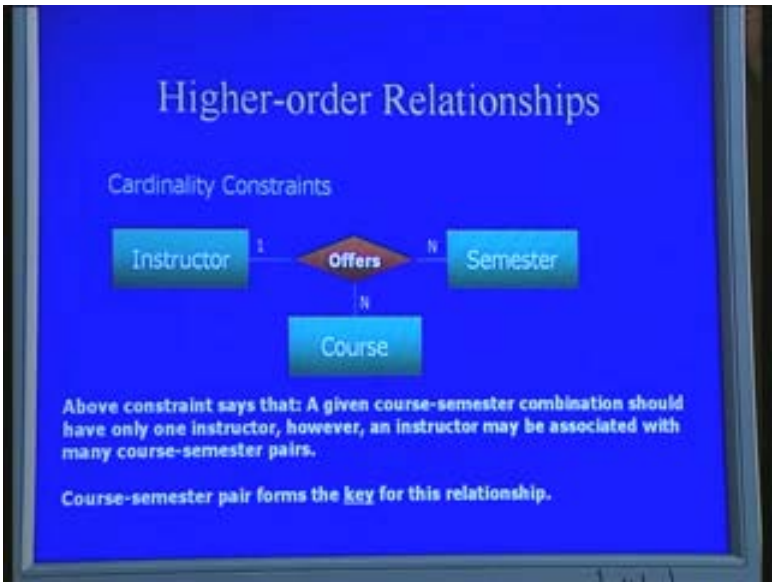


That is taught_during i s and can_teach i c and offered_in c s that is an existence of this ternary relationship implies the existence of all this binary relationships. However the converse need not be true, the converse that is suppose I have instructor can_teach, instructor I can_teach course c and instructor I taught_during semester s and course c was offered during semester s but that doesn't mean that the same instructor has offered this course during the semester is.

Instructor I can_teach this course but and the courses offered during a semester and the instructor taught_during that semester but that still doesn't say that the instructor taught the same course during the semester, he could have taught some other course and this course could have been taught by somebody else. So while this is true the converse is not true, reducing if instances of binary ration relationships exist, we cannot be sure that the instance of a ternary relationship also exists.

Cardinality constraints on higher higher-order relationships. So what is it mean when we say when we put cardinality constraints on higher-order relationship? Here is an example this examples shows again the instructor semester course example, so it has put a 1 here and a N here and a N here.

(Refer Slide Time: 40:50)

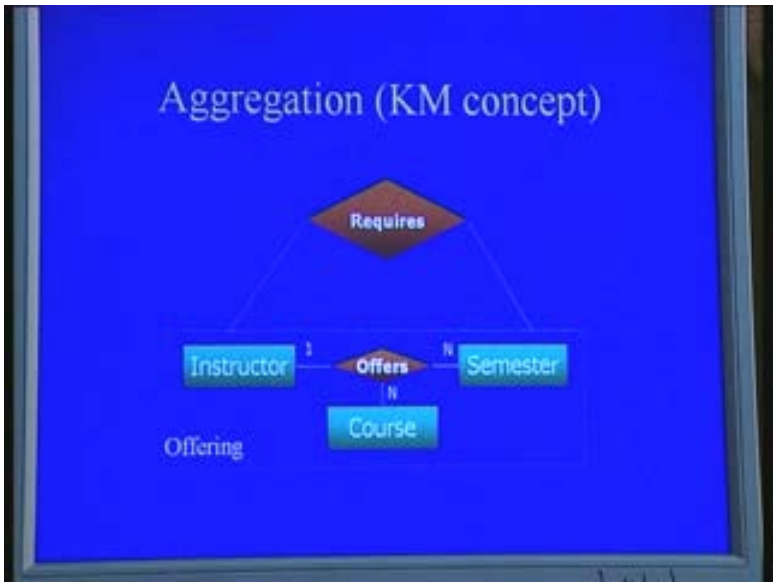


So that means that at given course-semester combination should have only one instructor that is in a particular course for a particular semester there has to be only one instructor. On the other hand a given instructor may have any number of course-semester relationships that is given instructor can teach in any number of semesters and any number of courses. So, again if you think about this carefully, if I have a set of all this relationship types instructor course-semester and so on.

How do I identify an instance of this relationship type uniquely? The key here is the course-semester pair. So if I take an instructor, an instructor may offer any number of courses in any number of semesters. However if I take particular semester and a course you see that it can uniquely identify an instructor that is because every course and semester pair should have just one instructor associated with it.

The last concept that we are going to look at in this session is the notion of aggregation. This concept is usually used in what is called as knowledge management or KM in the concept of ontology's and so on.

(Refer Slide Time: 00:41:51)



So an aggregation basically aggregates a particular ER schema and makes into an entity at a higher level of abstraction. Note the certain difference or and very important difference between the kind of abstraction introduced by aggregation and the kind of abstraction introduced by inheritance or specialization. Aggregation brings about the concept of composition or contains relationships.

So here this slide show an aggregated entity called offering which contains one or more instances of the relationship called instructor, course and semester. So an instructor offering a particular course in a particular semester is called a course offering or an entity type called offering. So the relationship between the offering entity type and this relationship called offers is that of contains offering contains offers.

On the other hand the relationship between generalized and specialized classes is that of is-a relationship or rather between specialized and generalized classes. A car is-a vehicle and a bus is-a vehicle or monkey is-a primate and so on but aggregation offers the concept of containment this contains this contains this and so on.

So even aggregation brings about a kind of abstraction that is you are covering up unnecessary details, if I am not really required to know what is the structure this offering. I don't need to really worry about that. So this slide here shows the relationship between offering and offering that is one course offering requires another course offering. So let us say course number A requires or has a prerequisite that some other course lets it said has to be taken up by the student.

So a course offering of A requires a course offering of Z, so without Z being in the database I cannot have a course offering of A. So here the abstraction basically throws away all details which that talks about what exactly an offering is about.

(Refer Slide Time: 00:44:44)

So with that we have covered the major parts in the enhanced ER notation or EER notation. So before we conclude the session let us take up a small example of a university database and see how or in which kinds of situations do we get these inheritance and generalization and specialization. How do we go about identifying that we might probably generalize here or even probably specialize here and so on. Now take up a small university database. Now this example here is by no means exhaustive I mean we cannot build a complete database in the course of a session like this but we just trying to see what kinds of typical problems or typical kinds of issues that can arise here.

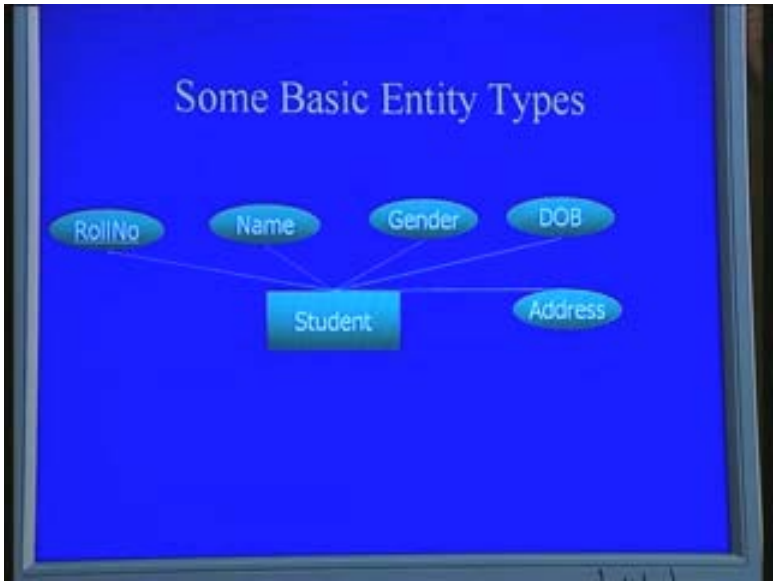
(Refer Slide Time: 00:45:38)

So some basic entity types, so each universities has a student and of course several other entity types. Again I am abstracting away unnecessarily details that is students, faculty members and staff and so on and so on. So let us say we have identified a basic as entity type called student.

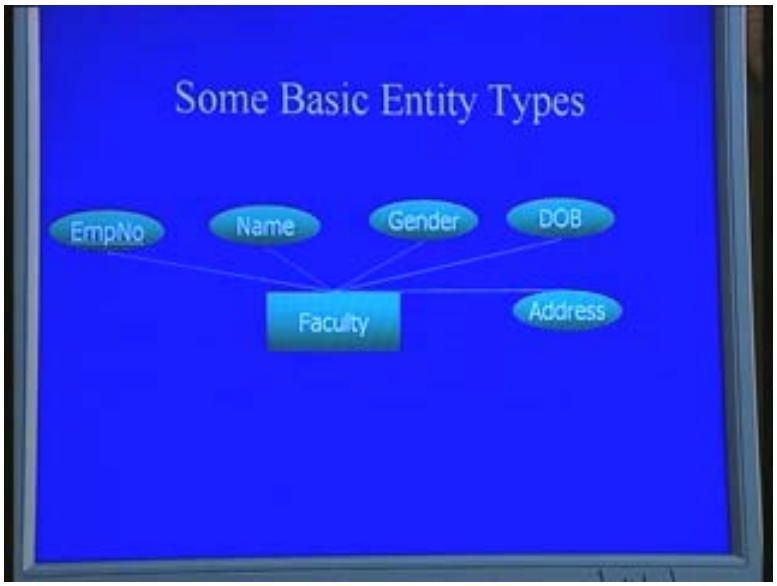
Now we then go about adding some attributes, for students. We note that each student is given a roll number which uniquely identifies the students as long as the student is in the university. Each student has a name, each student has a gender, a date of birth, address and so on and then we go about looking at other entity types.

Let us say we say that faculty is an entity type and then we talk about what are the attributes that characterizes a faculty member. Then we come out with some more attributes like this let's say each faculty has an employee number, each faculty has a name and a gender and date of birth and address and so on.

(Refer Slide Time: 00:45:57)

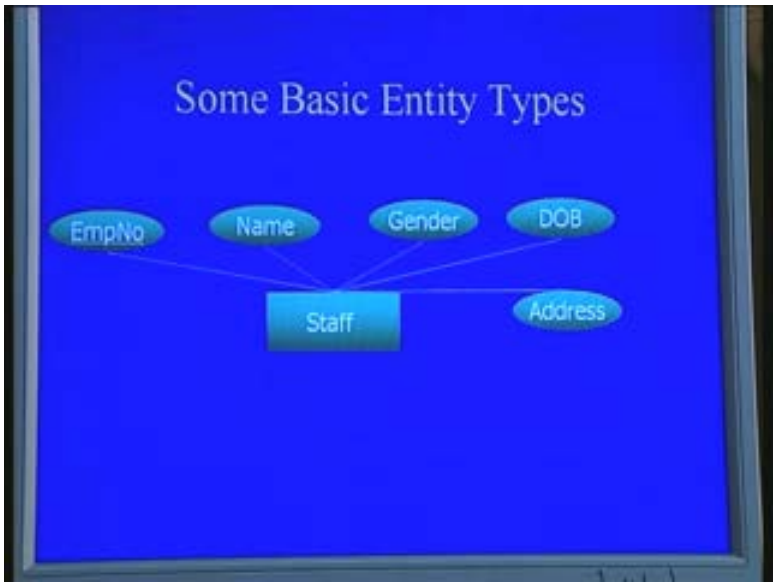


(Refer Slide Time: 00:46:16)



Now if you see student and faculty they look quiet similar all ready. Then we identify let us say some kind of non teaching staff and then we see that even they have the same kinds of attributes that is employee number, name, gender, date of birth, address and so on.

(Refer Slide Time: 00:46:51)

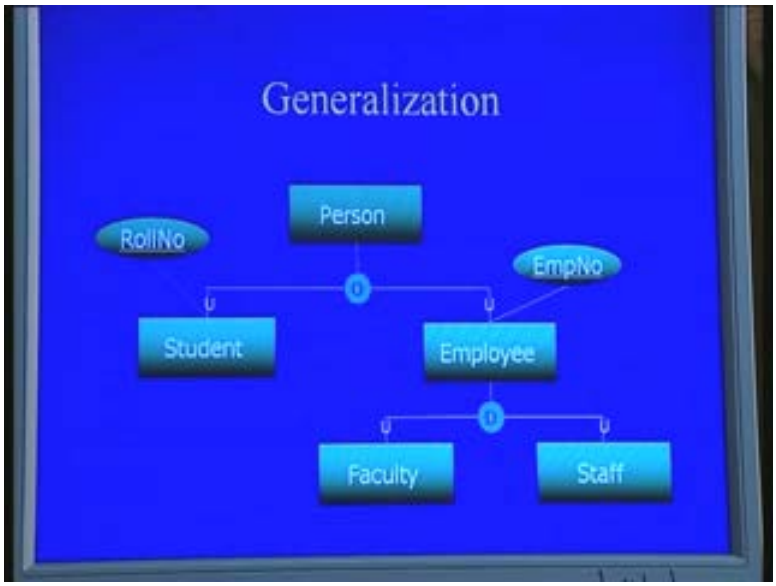


So which tells is that we are actually looking at different entities of the same generalized class. So what kinds of generalization can we make out of these three different classes? If you see staff and faculty, there is hardly any difference between the two entity types but between faculty, staff and student there are certain differences. So how do we identify

these differences here? This brings us to a generalization and specialization tree. We see that faculty and staff can both be categorized as employees and they have the same key called employee number.

On the other hand the same, the key called employee number cannot identify a student, a student is given a roll number. So employee and student do not belong to the same level as a faculty but they belong to the same entity type called person. Now what is the property of this person entity type everything else that was common between the three entity types.

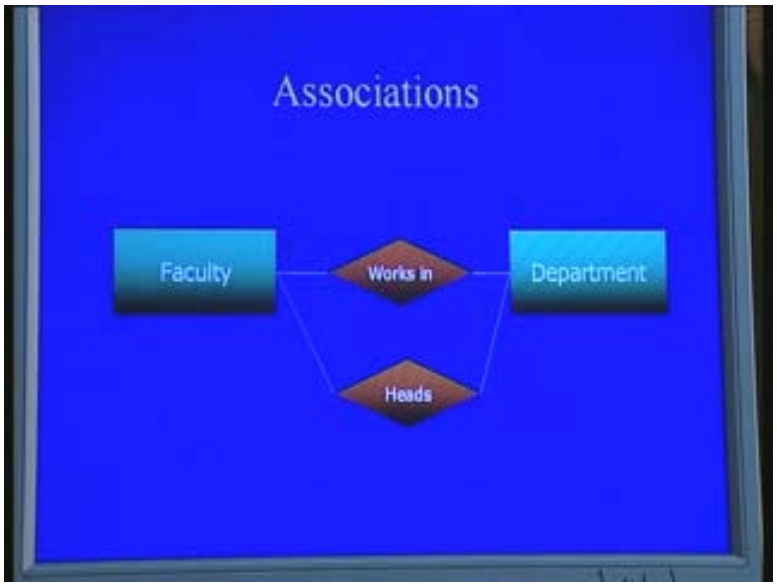
(Refer Slide Time: 00:47:41)



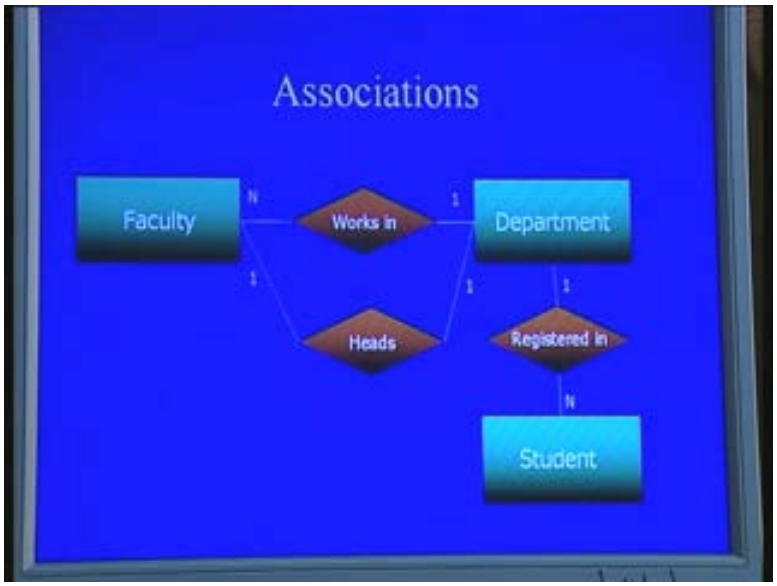
What are the common attributes between the three entity types name, gender, date of birth, address? So the attributes that going to person would be all of these attributes name, gender, date of birth, address and so on which all of them share whether it's a student or a faculty or staff all of them share. Similarly we start looking at certain association let us say a faculty works in department and a faculty heads department and we identify certain kinds of association constraints that says that n number of faculty member may work in a department while only one faculty member may head a department. And we also identify some more associations which says that n number of students maybe registered in a particular department. And we can also find some aggregations which says that a project involves particular department or project is headed by a faculty member and a faculty member belongs to a particular department and we see that this whole thing can be aggregated into an entity type called sponsored project.

So a sponsored project means that there has to be a project entity which is involving a particular department and is headed by a particular faculty member and so on. So how well schema is aggregated into the sponsored project entity type?

(Refer Slide Time: 00:48:40)

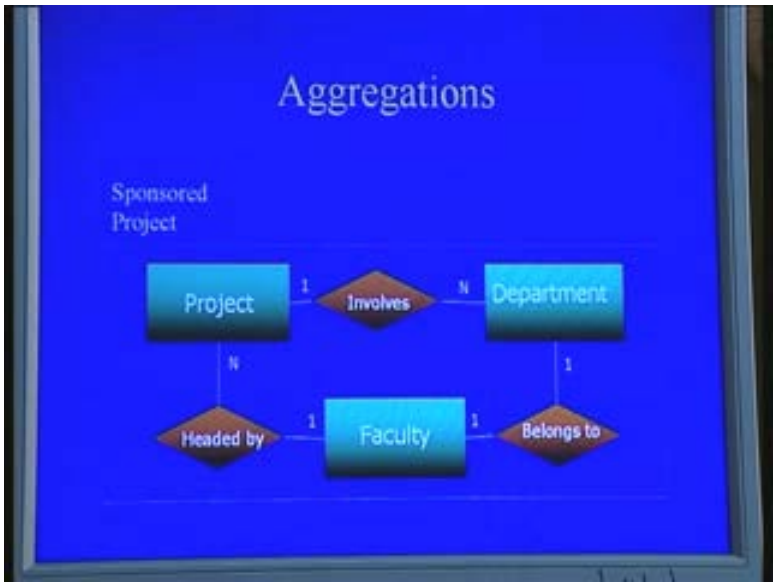


(Refer Slide Time: 00:49:02)

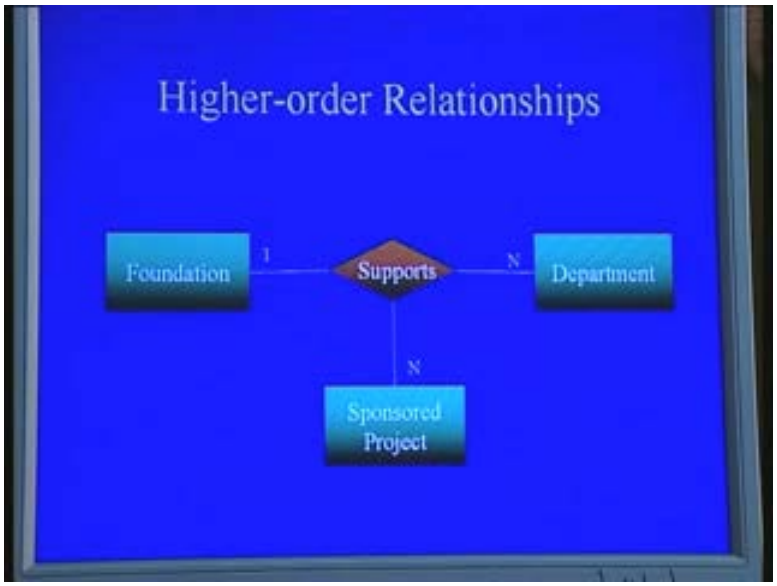


We can also see a certain higher-order relationships. For example let us say some foundation, some organization or non-governmental organization or whatever supports a particular project and a particular department. So again we see here that foundation supports department on this project and we can see that we cannot reduce it to binary relationships, foundations may support sponsored project and may support department but the ternary relationship says that for this project and for this department this foundation is supporting.

(Refer Slide Time: 00:49:02)

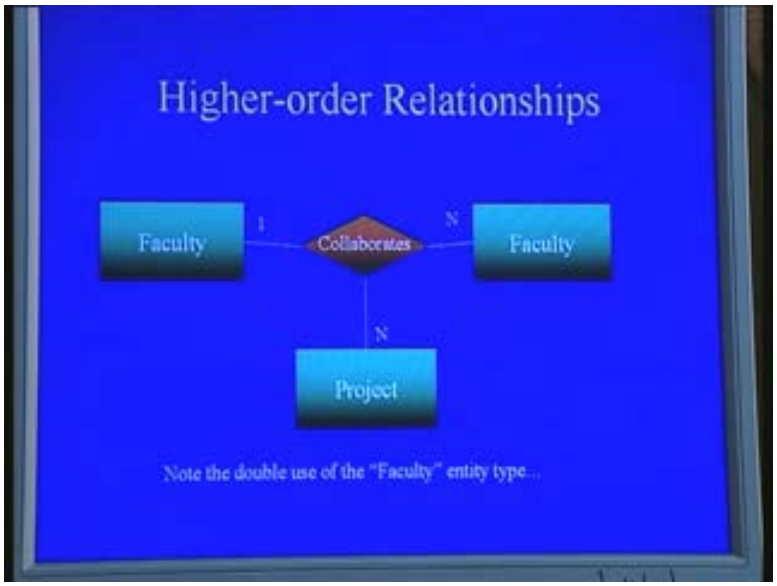


(Refer Slide Time: 00:49:53)

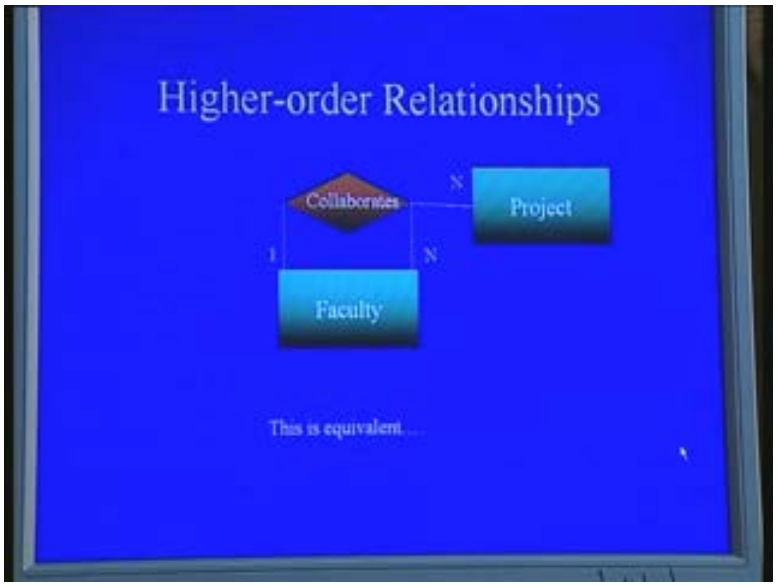


So have a look at this higher-order relation relationship here. Let us say I have a relationship that says a faculty member collaborates with some other faculty member on a particular project. So note the double use of this faculty entity type that is this faculty member collaborates with this faculty member on a particular project.

(Refer Slide Time: 00:50:30)

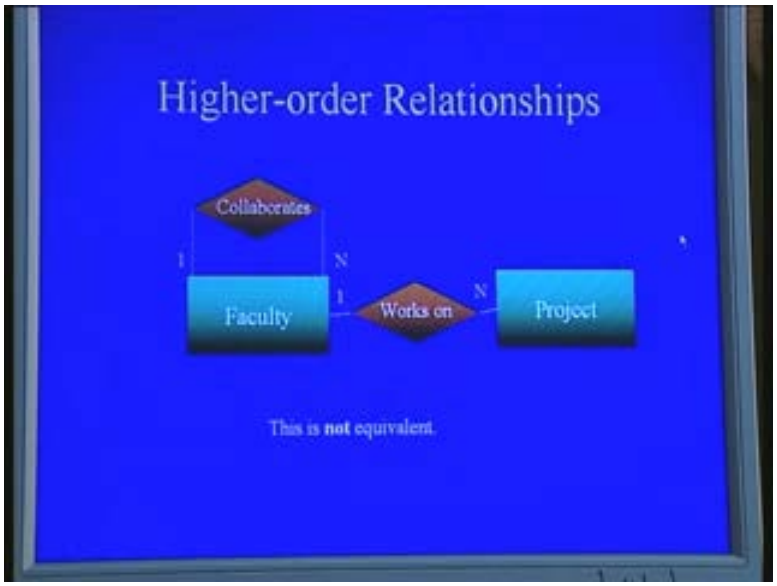


And so one faculty member probably, the head of the project may collaborate with n other faculty members on n other on n different projects.



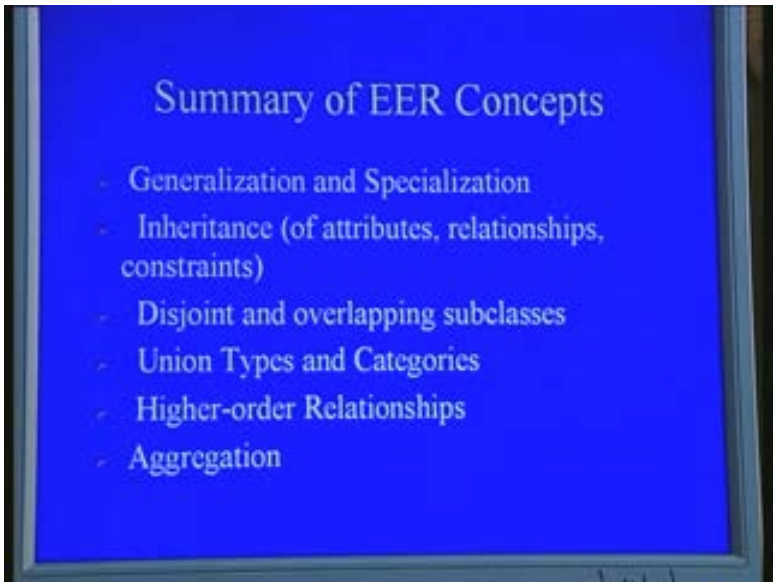
Now you can reduce it like this that is a faculty member collaborates with other faculty members and the same collaboration extends to project.

(Refer Slide Time: 00:51:09)



However you can't reduce it like this that is a faculty member collaborates with another faculty member and works on a project because we are going to lose semantics.

(Refer Slide Time: 00:51:27)



So that brings us to the end of the second session of enhanced entity relationship concepts. So before we conclude let us briefly go through the different concepts that we learn today. The first concept that we learned was about generalization and specialization where you achieve abstraction using an is-a relationship. So in a generalization and specialization relationship, for any entity of the general class can be replaced by any entity of the special class or the specialization without losing semantics only then will

you be able to say that my generalization is correct. It need not always be correct just because something looks like is-a would hold doesn't mean that the generalization is correct.

So we saw the notion of inheritance that is each specialized class or a subclass inherits all attributes including key attributes and constraints and relationships from the generalized class and we also saw the notion of overlapping subclasses and disjoined subclasses and how to build a entity type using union types or categories and we saw the notion of higher-order of relationships and how they cannot be reduced to lower order relationship without losing semantics. And the final concept that we saw today was the notion of aggregation which is again a kind of abstraction relationship but however which establishes the notion of containment rather than is-a that is abstracted by the specialization relationship. So that brings us to the end of this second session on enhanced entity relationship concepts.