Database Management System
Dr. S. Srinath
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 42
Case Study - Part 2 Database Design

Hello everyone. Welcome to this session in DBMS. So in this session today what we would be doing is we would be continuing from the previous session were we talked about a case study of a database design. So again let me put forth a kind of assumptions or the prerequisites that I am expecting for this session. I am assuming that you know or you have gone through the first few sets of lectures on the life cycle of a DBMS and you also know what is conceptual modeling, what are the building blocks of conceptual modeling, what are the different kinds of nuances in conceptual modeling like a week entity type, like a multi valued attribute, like a composite attribute and so on.

You also know the relational schema or the relational model and the characteristics of relational model like, the models like the functional dependencies and normalization and so on. And you also have an idea of how to convert what kinds of rules that you can use to convert a given ER schema to a relational schema. So I am assuming that so will not be going into the rules of how to convert a ER schema into a relational schema except that we are going to take a running example of a conference management system that we started in the previous class. And start by explaining which kinds of, we just take up specific aspects of the ER diagram of the conference management system and just convert them to the corresponding relational model.

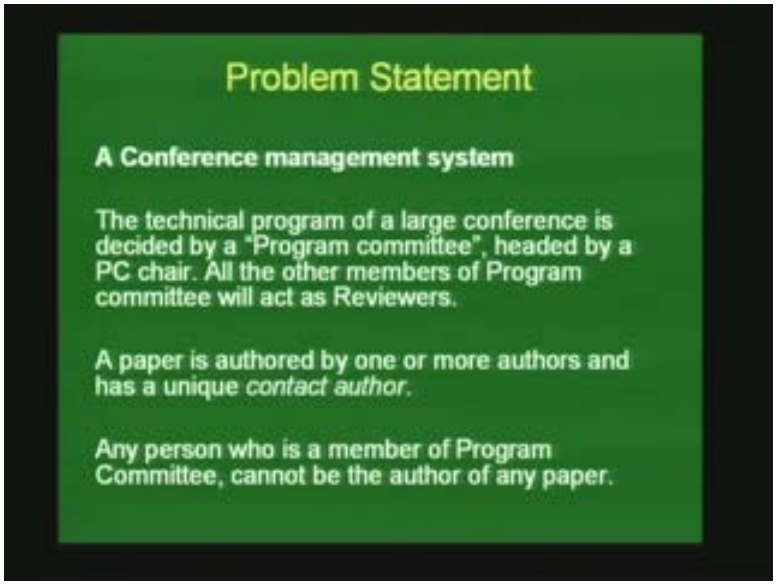
And of course we will also look at few kinds of transactions on top of this relational model. I know that we have not really looked into transactions as part of this series of lectures as such but I will also define transactions as we go along. And in an informal sense of course there we shall be looking at transactions and transaction processing in much more detail in later sets of classes. But at least I will be defining transactions in an informal sense so that we can see what kind of transactions at least we can think of on top of these sets of requirements.

So the main idea behind this being that to try to cover the large or the first few aspects of a systems development life cycle that the high level design, the architecting, the conceptual modeling, the high level transaction design and just have a look at how does the entire system look like. And then you can over a period of time, when we address several different questions throughout this course, you can try to target each one of these different architectural aspects of the system like the relational model, the conceptual schema, the business logic and so on and transactions, transaction manager and so on and then try to see how you can make a detail design into each of these different aspects of, each of these different building blocks of the overall system.

(Refer Slide Time: 04:28)

So let us come back to the case study for a database design that we are looking at. So today we look at the relational schema design and the transaction design for the case study that we are going to look in.

(Refer Slide Time: 04:52)



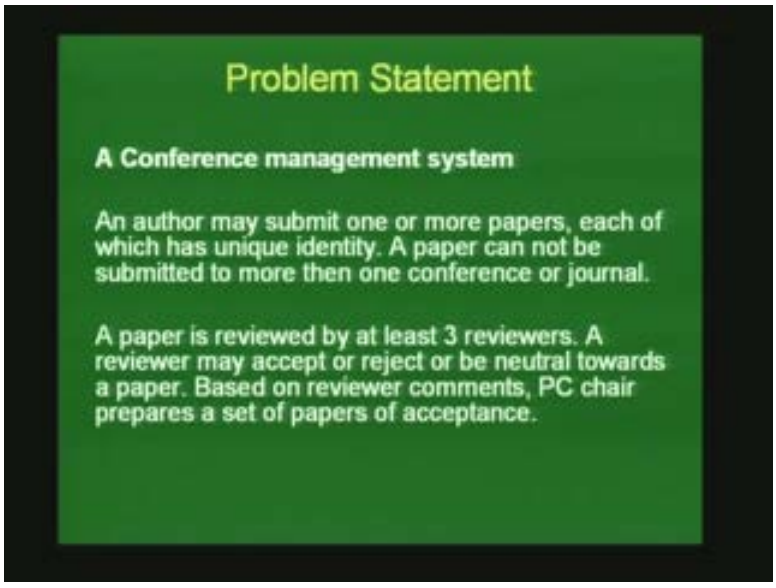
So what was the case study that we have been looking at? Let us briefly review the case study once more and before going into today's work. So the case study was about a conference management system and we went through the set of requirements in the previous class. So let me briefly go through them once again to try to see what exactly is required by this conference management system. Like I said in the last class, this is a

fairly comprehensive example in the sense that it's fairly representative of a real life conference management system although not as detailed as one. A real life conference management system would be far more detail than this but at least it kind of captures the main essence of a conference management system. So what are the requirements? The technical program of a large conference is headed by a program committee. So a conference is, a large conference the adjective called large actually we haven't really made use of here because there is no other further set of requirements as to how to handle small conference. So we are just talking about a conference. So as far as we could understand as a systems designers or systems analyst, the technical program of a conference is headed by a program committee and is determined by a program committee.

And the program committee in turn is headed by a PC chair, a program committee chair where we saw a program committee chair is a one person who heads the program committee and all other members of the program committee will actually act as reviewers. And again we saw in the previous class that the reviewer and the program committee chair should not be the same person. So a person who is the PC chair will not be a reviewer and vice versa. And so that's about the program committee. The next chunk of requirements for about the paper itself, the papers which are going to be presented with the conference.

So a paper is an entity again which is authored by one or more authors and as a unique contact author. It basically says that it should have a contact author and it's a unique contact, there is only one contact author for a paper. And any person who is a member of a program committee cannot be the author of any paper, like I said in the previous class also this is not a realistic requirement were because in most real life conferences this is not exactly true. But just to make things simpler, we are imposing this constraint in order to see the repercussions on our database design.

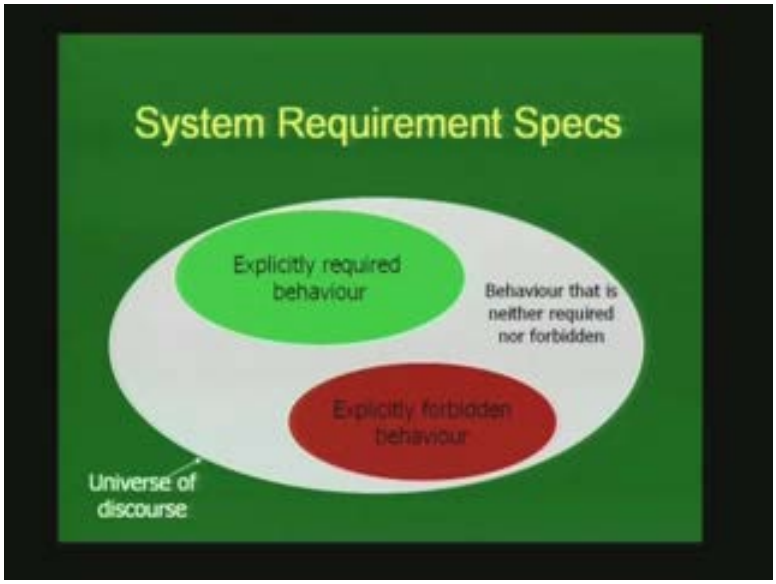
(Refer Slide Time: 07:56)



And an author may submit one or more papers each of which has a unique identity. So a paper has to have a unique identity which in turn means that a paper ought to be treated as a separate entity in its own. So an author may submit one or more papers, there is no constraint on that. However a paper cannot be submitted to more than one conference or a journal at the same time. So, a given paper cannot not be submitted to more than one conference, so given a particular paper it has to go to a particular conference and so on.

And a paper is reviewed by at least 3 reviewers, so is reviewed. So it basically says that a paper shall be reviewed or should be reviewed by at least 3 reviewers. So even if a paper is reviewed by 2 reviewers, it's not sufficient it has to be reviewed by 3 reviewers and a reviewer should give a decision. And the decision should be either accept or reject or be neutral towards the paper. So a reviewer has to give one of the 3 decisions and the reviewer can't say, I am neither going to say accept nor am I going to say reject nor am I going to say that I am neutral about this paper. I have an opinion but I am not going to state it and so on.

(Refer Slide Time: 09:09)



So anyway based on the reviewer comments, the PC chair either accepts or rejects papers and prepares a set of papers for acceptance. Now again we also saw this in the previous class were the characteristics of a given requirement specifications. So it's again good to review them once more before we go in to relational model design as well. So that will help us understand the suitability of our model much better.

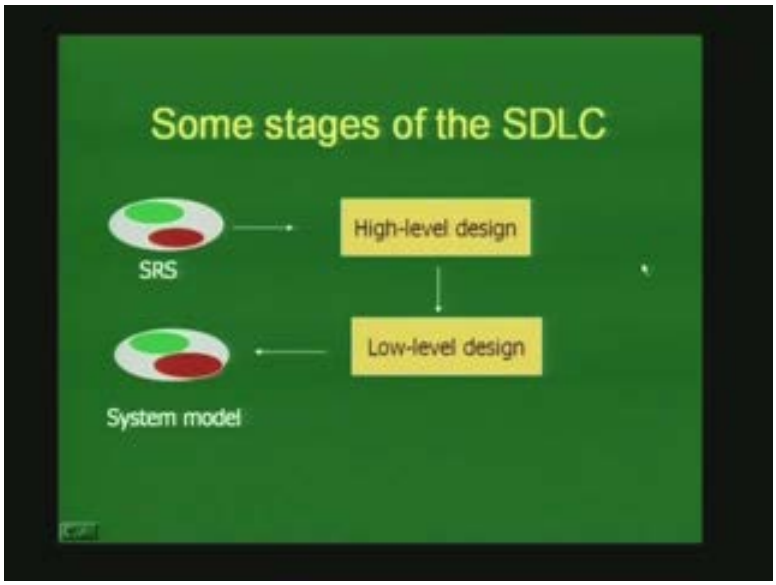
So, any given requirement specification has a set of explicitly required behavior. So the requirements or the specification say that this and this has to be there, this functionality has to be there and then there are a set of explicitly forbidden behavior which says this and this functionality should not be there or shall not be there and so on. So we also noted that the set of forbidden behavior constitutes what are called as safety constraints which usually you forbid something in the interest of safety.

So if you violate that constraint, it means that you are compromising the integrity or the safety of the system. Similarly the set of required behavior constitute the liveness of the system. So a completely safe system is one which does not work at all. So if you don't go out into the battle field then there is no danger of you being hit by a bullet. But the thing is that does not make a soldier, you have to go into the battle field. So there is a set of required conditions were which form the liveness properties of the system, so that are specified by the explicitly required behavior.

And in addition to liveness and safety usually in almost all cases, the UOD will contain a number of behaviors that are neither required nor forbidden. So the requirement specifications neither says yes, it is required nor does it say no it's forbidden. So one might call that, call them as permitted behavior or whatever. So depending on how you treat this kind of mid way area, your conference management tool can offer an edge over the others in the sense that my conference management tools manages birth dates of and sends greeting cards and so on. I mean they are not, they are neither required nor forbidden and so on.

And coming to the first few steps of a systems development life cycle or an SDLC, we see that what we want to do or what we are going to do here is given a set of requirements specs, requirement specifications we come out with the high level design. We kind of address this in the previous class in the sense that given SRS, we created an ER diagram for the given SRS.

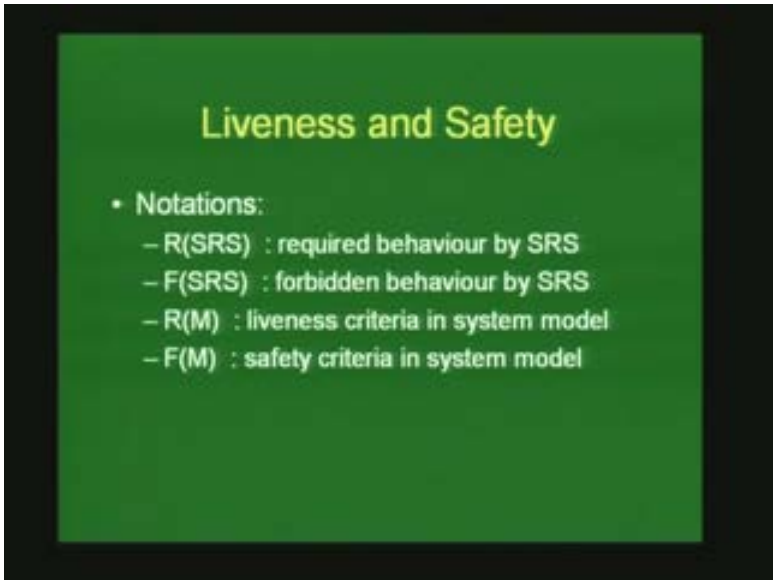
(Refer Slide Time: 11:35)



So today we are going to be doing this one and of course with a little bit of transaction design will also be doing this one. So from the high level design, we go to the low level design and from the low level design add dynamics to that is high level design low level design were purely in the static. So add dynamics to that by adding the transactions and so on, so you get into a system model.

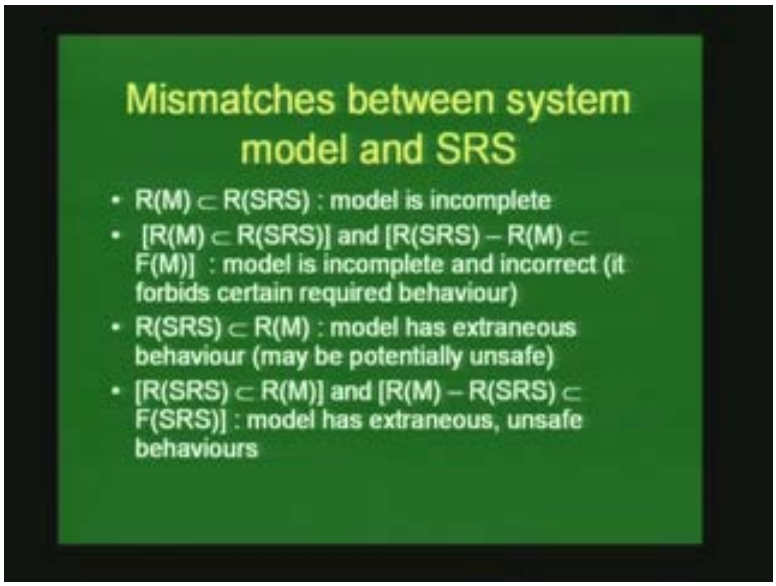
So the system model should ideally reflect the systems requirement spec or the SRS document and of course there are, usually there are mismatches between the systems model and the requirements spec and rather than going about wildly looking for features or looking for characteristic features of the systems model, it is good to classify this deviations between the systems model and the requirement spec into different kinds which helps us understand in what way is the systems model deviating from the set of requirements.

(Refer Slide Time: 12:48)



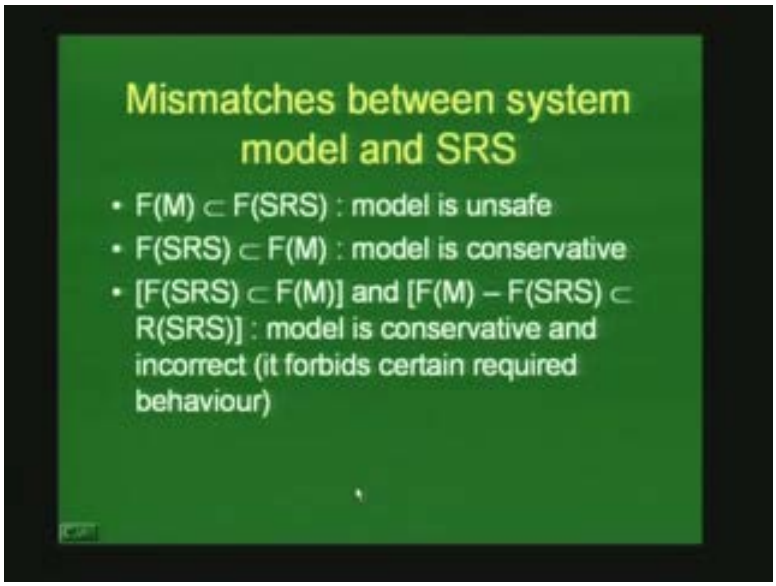
So we also saw this in the last class (Refer Slide Time: 12:47) were we saw how to quantify in some sense or the kind of deviations that system model has from the set of required requirement specifications. So let us briefly again revise through all these so that it helps understand how to evaluate a given system model, two or two or more systems model against one another. So one can say that a given systems model is incomplete, if the set of required behaviors that the model performs is a proper subset of the set of behaviors of the requirements itself, so it is incomplete. And being incomplete itself doesn't, may not be all that serious but sometimes it may be quite serious, it could be incomplete and incorrect.

(Refer Slide Time: 13:07)



So when is the systems model incomplete and incorrect? When not only does it not, only does it not satisfy all requirements but it forbids certain requirement that is it explicitly does not satisfy certain required behavior. So it forbids certain required behavior that is that's when you say that the model is actually incorrect and it's possibly unusable. And if the set of requirements, if the set of required behavior by the model is a super set, proper super set of requirements spec then the model has extraneous behavior, it does something extra. And again providing an extra feature doesn't always necessarily mean a good thing, it could potentially be unsafe as well. When will it be potentially unsafe? When this extraneous behavior actually trespasses on the forbidden conditions of the requirement spec. So R M minus R of SRS actually belongs to F of SRS that is where you trespass on the forbidden requirements.

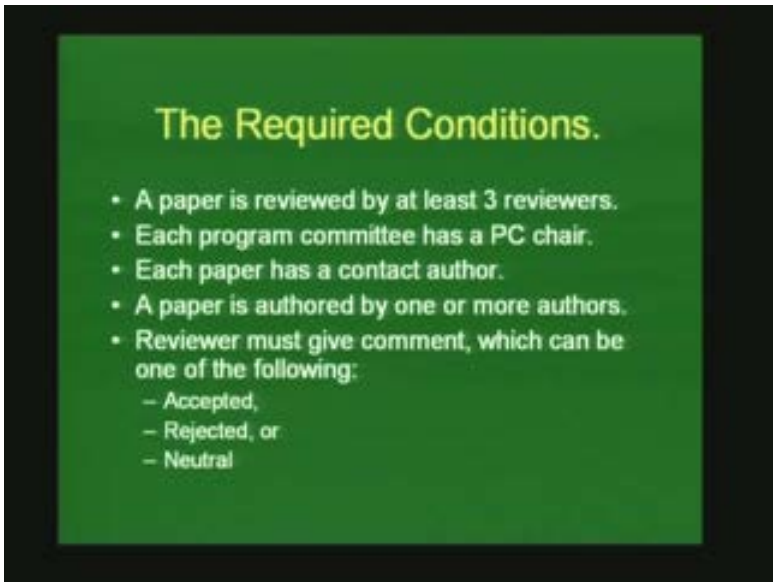
(Refer Slide Time: 14:57)



Similarly if I have a set of forbidden conditions by the model being a subset of the forbidden conditions by the set of SRS then the model is unsafe that is the model does not forbid everything that the SRS is asking you to forbid. And if it is the other way around that is if the model forbids more than what the SRS asks you to forbid then you say that the model is conservative. But again conservative doesn't necessarily mean that it's a safe system. Why? Because in trying to be conservative you might be blocking certain liveness behaviors, you might be actually stopping certain behaviors that are required functionality for the model.

So if F M minus F SRS is actually a part of R of SRS that is part of the required behavior then the model is not only conservative, its actually incorrect. So it forbids certain required behavior. And what are the required conditions, what are some of the required conditions in this case study? We saw certain required conditions in the case study that is a paper should be reviewed by at least 3 reviewers for example and each program committee should have a PC chair.

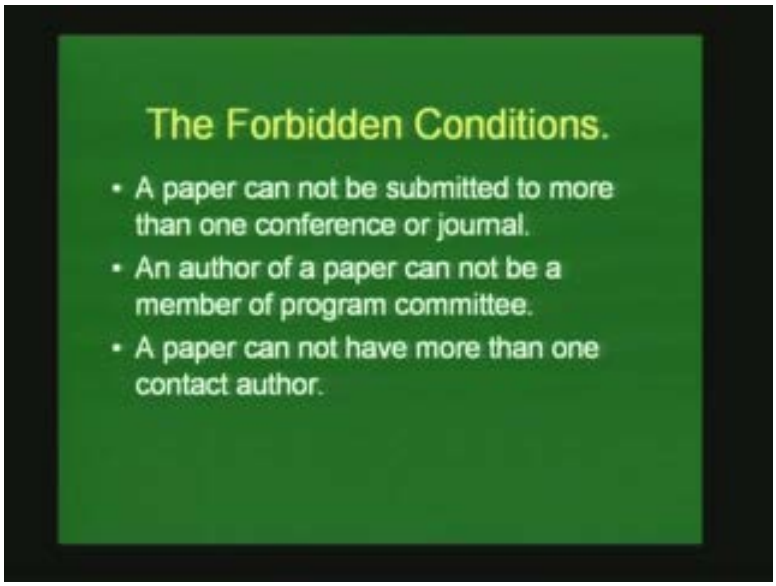
(Refer Slide Time: 16:10)



And each paper should have a contact author and a paper is authored by one or more authors, I mean we cannot accept papers with no authors in them and so on. And a reviewer must give a comment or must give a decision either accepted or rejected or neutral. So reviewer cannot afford to keep silent on a paper. And paper and what are some of the forbidden conditions in the model? A paper cannot be submitted or may not be submitted to more than one conference or a journal which is shown in the slide here.

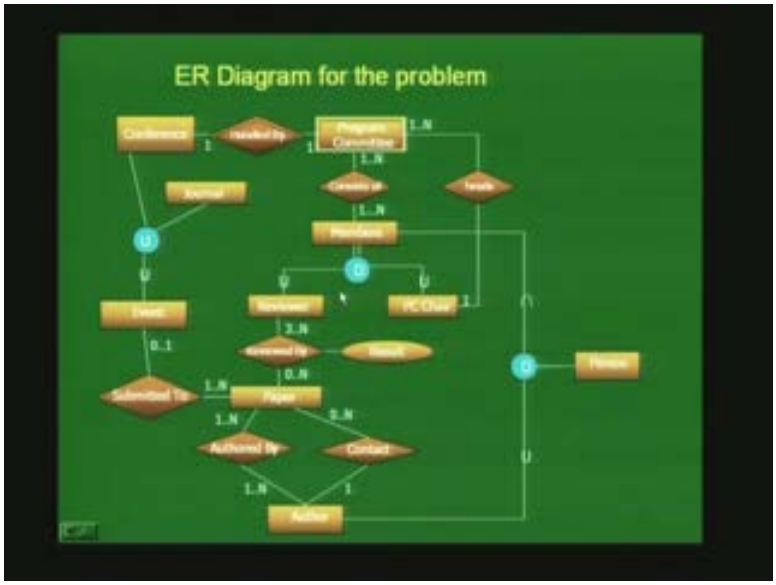
And an author of a paper cannot be a member of the program committee like we said even though this is not realistic, we have chosen that for the sake of simplicity here. That is I cannot submit a paper as well be in the program committee to review the paper and so on. And a paper may not have more than one contact author, each paper should have one and only one contact author.

(Refer Slide Time: 17:11)



So there is only may email sent out for each decision and it has to go to the contact author and so on. So having said that in the previous class, we went into each requirement sentence by sentence and identified several kinds of entities and relationships among them and also attributes for these entities.

(Refer Slide Time: 17:40)



And finally put all of these entities and relationships together to form one big ER diagram for the problem. So let us spend a few minutes to revise the ER schema for this problem. This is important because now we are going to cut this ER schema bit by bit and then convert it into a relational schema. Now what are the first thing that we did? We started

by a conference. A conference is handled by a program committee and so on. So a conference is an entity and it is handled by a program committee. And we also saw that the program committee is a week entity type it's not, if you can see, if you can notice carefully it is not really specified by the requirements and that the program committee are to be a week entity type. But we have to infer it, I mean that is why the high level schema design that the conceptual schema design is a very human centric process, you have to use your common sense in order to identify or in order to say which kinds of, what should be the characteristics of each of this entity type and so on.

So program committee is a week entity type and this is a identifying relationship which says that a program committee that handles a conference defines the existence of the program committee in the first place. And a program committee consists of different members and the set of members are divided into a disjoint set of specialized members which are called reviewers and PC chair. So given a member a member could either be a reviewer or a PC chair and there are to be only one PC chairs, so there is a cardinality constraint here. And these are disjoints, so therefore a reviewer may not be a PC chair and vice versa.

So, a program committee consists of members and the PC chair heads the program committee. So one PC chair heads any number of program committees or given a program committee there is just one PC chair for that program committee. And similarly a reviewer reviews a paper or a paper is reviewed by a reviewer. So a given paper, a given reviewer may review 0 to N paper that is a reviewer may not be assigned any papers at all. There is no, there is neither a requirement nor a forbidden condition as to how many paper should be assign to a reviewer. However there is a condition on how many reviewers should be assigned to a paper. That is a paper should be assigned at least to 3 reviewers, so 3 to N. so there can be any number of reviewers but there should be at least 3 reviewers.

Now once a reviewer reviews a paper, the reviewer gives a result. Now the reviewer has to give a result, it's a required condition again that is a reviewer has to say either accept, reject or neutral. So now this result is an attribute of neither the paper nor the reviewer itself. Why because a paper is reviewed by more than one reviewer and a reviewer may review more than one paper.

So you cannot assign result to any one of them. In fact the only way place you can assign result is to the relationship itself. That is that the process of being the process of reviewing a paper by a reviewer gives out an attribute called result. So in addition the paper itself is related to authors and a paper is authored by one or more authors. Note that here it is one or more, it can't be zero or more. So we also had a condition, forbidden condition that authorless papers are not accepted, are not acceptable. And a paper has to have exactly one contact author. So one of them is a contact author and it is authored by many of these other papers and so on.

And in addition authors and members form a disjoint specialization for person that is we store certain personal details of people in the system their address, name, date of birth,

phone numbers and so on and all of them have to be stored for both members and authors. And similarly a paper can be submitted to exactly one event or zero or one event, maybe submitted to exactly one event that is it need not be submitted. So but if I choose to submit, I can submit it to exactly one event. And what is an event? Event is a conference or a journal. So if I am submitting a paper to a conference then I should see that it's neither submitted to any other conference or nor to any journal as such. So this forms the overall ER schema as part of the problem that we saw in the previous class.

now let us go to the next step now that is from the ER schema we should reduce it down to a relational schema and which is actually managed by the DBMS. Now the parlance of a relational schema is quite different I mean you have, you don't have entities and relationships and attributes and so on. Although, you do have something called attributes but in a different sense. You have tables and columns and rows and attributes which is a column and there are key relationships then foreign keys and decomposition of tables and functional dependencies and so on and so forth. So let us take things step by step and take up ER to relational mapping.

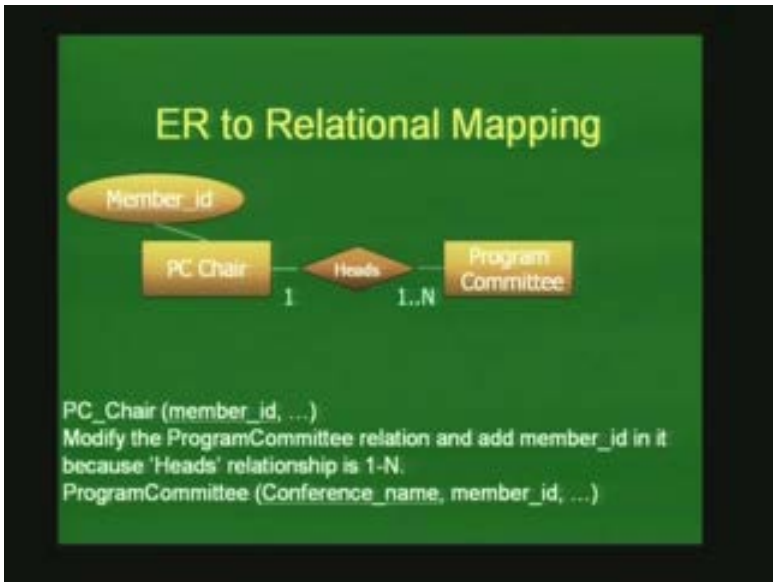
So the first set of ER to relational mapping that we are doing is shown in the slide here. So we started out with conference. Now conference is handled by a program committee and we also saw that one conference, one program committee. And program committee is a week entity type that is its existence is defined by a conference. And the conference is defined by several attributes like conference name, place, date, topic and so on.

(Refer Slide Time: 23:34)

And the conference name is used as the conference key or the primary key for the conference. So how do we reduce this to an ER model? First thing we notice that the entity conference can become a table called conference. And all the attributes that the conference has will form the attributes of table of which conference name is the primary key.

Now here if you look at program committee, program committee may have a key but that is not sufficient. Why because, the program committee has no existence without a conference. So essentially while we can create a table called program committee which corresponds to the entity called program committee, it is not sufficient to just add the attributes of program committee here and leave it at that. Why because we need the conference name as well so, because it is defined by the conference name, program committee for which conference. So take the primary key of conference and make it into the primary key of the program committee itself. So in fact this would be a foreign key relationship into the conference table.

(Refer Slide Time: 25:17)



So the next set of conversions. Now a PC chair heads a program committee and a PC chair and every other member. Note that again for the sake of clarity, I have not shown the entire ER diagram here but if you remember how it was a PC chair here (Refer Slide Time: 25:41) is a member. This kind of relationship here defines a is a relationship.

So a PC chair here is a member and every member is defined by a member id. The every member is uniquely identified by a key attribute called member_id and whenever a general entity class or an entity set has a key attribute every subclass of it or every specialization of that entity type also inherits the key attribute. Therefore the PC chair also inherits the member_id as its key attribute. So now the member_id is the key attribute of this PC chair and what is the key attribute of program committee. It is the conference name, so that is a key attribute of the program committee.

Now a PC chair heads program committee and look at this relationship. It's a 1 to N relationship. So 1 PC chair heads, may head several program committees and so on. So how do we convert this? First of all converting a PC chair to a table is trivial. so make a table called PC chair and put all the attributes of PC chair in addition to all the attributes of member into this table and make the key of the super of the general entity type that is a

super class if one might call it that, so of the general entity type called member plus any key attributes in PC chair make into composite key in the PC chair table.

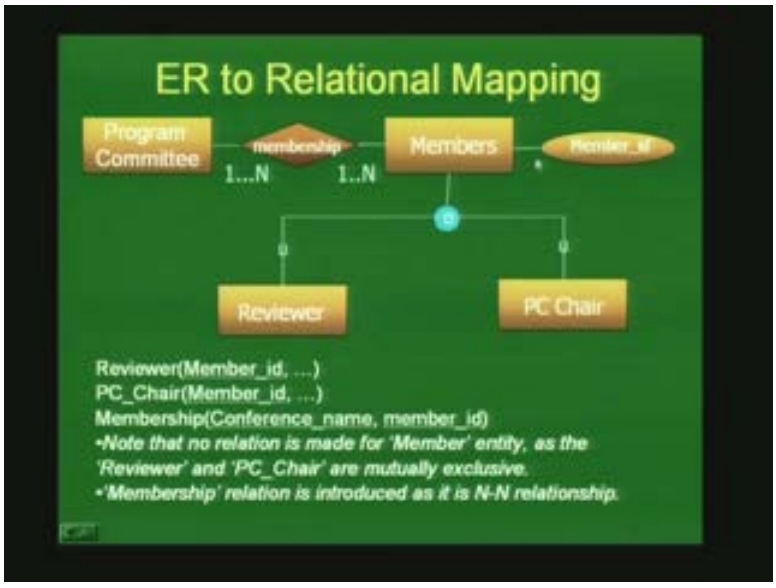
Now for this relationship itself, how do we go about doing that? Take program committee as the key, now as the table and its key attribute was conference name. Now in addition to the key attribute called conference name just put the member_id of the PC chair here as one more attribute. So there are two foreign keys here, so one forming the conference name attribute and that is one is the conference name and the other is the member_id.

However it is sufficient if we just keep the conference name as the primary key even though this is a foreign key here. Why is it sufficient to keep just the conference name as the primary key? Because let us look back here (Refer Slide Time: 28:02). Program committee is a week entity type and its existence is defined by the conference. And what is the primary key?

A primary key usually of course is something which a key attribute which defines the existence of a particular tuple in a table. So it's the conference name which defines the existence of the program committee. Therefore while we add member_id to the program committee table, it is sufficient if we just keep conference name as the key for this program committee table.

Now the next set of attributes ER to relational mapping. So have a look at this thing, so we already saw PC chair here. PC chair is a member and a member could be a either a reviewer or a PC chair. And a program committee can consist of several members and each member is uniquely defined by a member _id. Now because this is a super class or a generalized entity type, this member_id is inherited by both reviewer and PC chair. So it forms a part of the primary key for both reviewer and PC chair and they may have other key attributes as well.

(Refer Slide Time: 29:20)



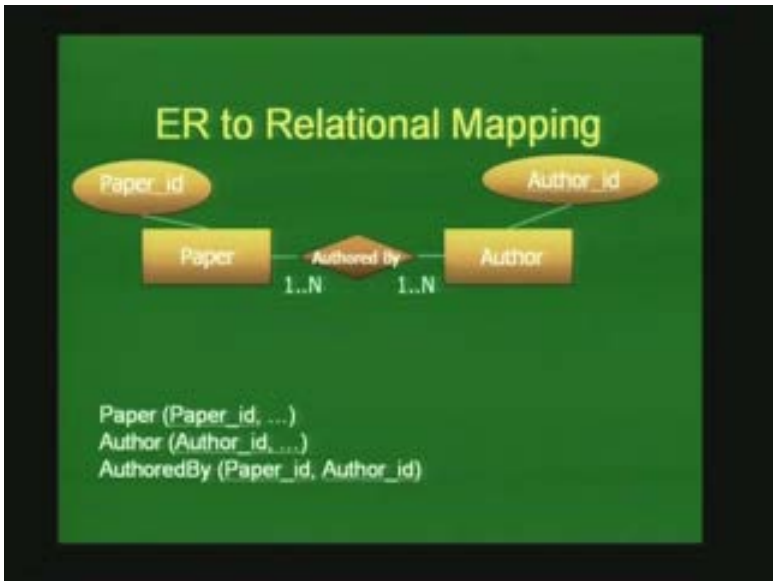
So one way to reduce this is we already made a table for PC chair with member_id as the, with member_id as the key, we can just make one more table for reviewer and again say member_id as the key and all other attributes. Now you might ask a question, why not make table for members itself. Members is an entity and why not make a table for members itself. Note that here it is not really necessary. Why because, a member is either a reviewer or a PC chair but not both because it's a mutually exclusive relationship between reviewer and PC chair.

So there is no special identity assigned to a member other than a reviewer or a PC chair. And we have seen that we need this PC chair table in some other context, in the previous context here. So we need PC chair as a separate table in some other context and therefore we have also made one more table for reviewer and that's it and both of them are collectively exhaustive in terms of this thing. So not only are they mutually exclusive, there are also collectively exhaustive and they collectively define the complete set of members.

Now the bigger problem here is the reduction of this larger relationship. What is that relationship? A program committee may comprise of several members. However a member can also be a member of several program committees, so it's a N: N relationship not a 1:1 relationship. So how do you reduce a N to N relationship into a relational schema? The best way to reduce that is make a separate table. So we are calling this table as membership here where just take the primary keys of both of these.

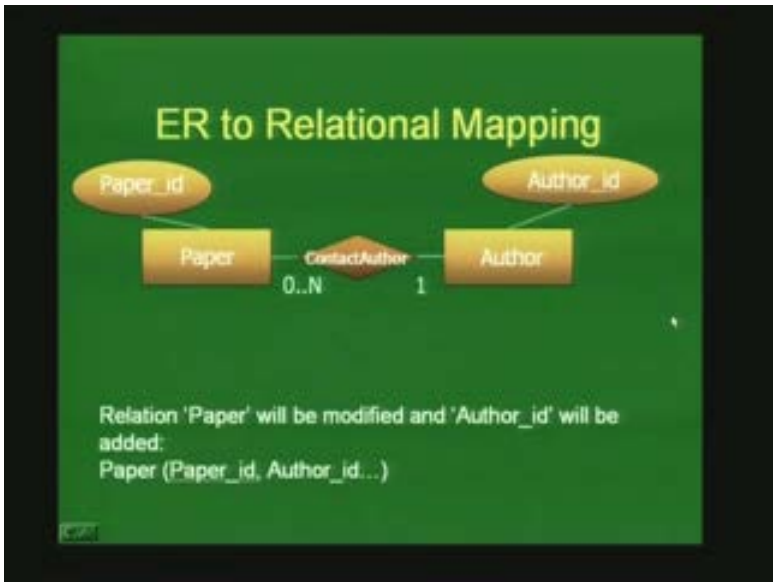
What is the primary key here? The conference name. So just take the conference name and member_id, so of a given member. So given a member I know of which conference is that member of, PC member of and so on. So, N to N relationships can be reduced by creating a separate table and combining both of the primary keys into that table.

(Refer Slide Time: 32:21)



The next set of reduction here. So we have a paper that is authored by an author. However a paper can be authored by more than one authors and an author can submit any number of papers. And of course a paper has a key attribute called paper_id whereas author has a key attribute called author_id. so again because this is an N to N relationship, you need one entity called paper with paper _id as a primary key rather one table called paper with paper_id as the primary key, one table called author with author_id as the primary key and a separate table called authored by which relates this and this. So it's an N to N relationship, so which contains both paper_id and author_id as part of its tables. So, two foreign keys into both these earlier tables.

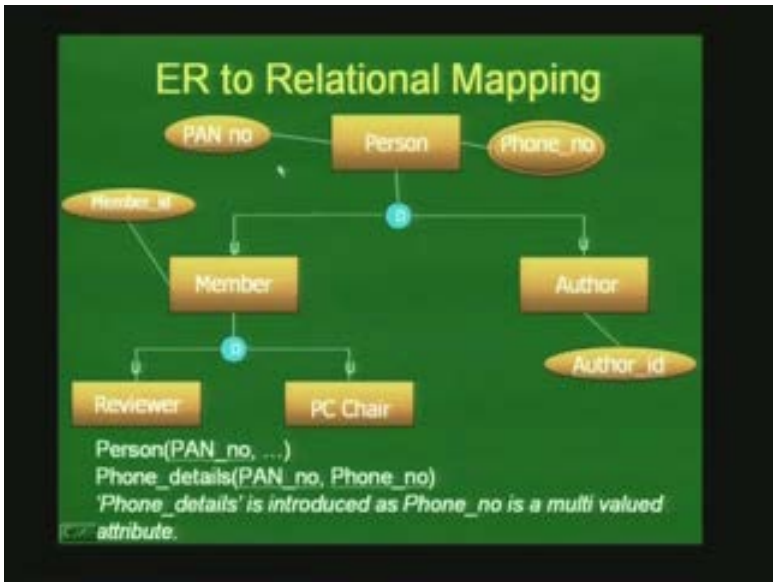
(Refer Slide Time: 33:01)



Now on the other hand for the contact author each paper ought to have exactly one contact author and it ought to have a contact author. So as you can see here this forms a 1 to N relationship. So in such case in a 1 to N relationship, there is no need to create a separate table here except that for a given paper, you just add the author_id here which is the id of the author who is the contact author. So the primary key of the single entity type here can just go as an attribute to the table of the multiple entity type in a 1 to N relationship.

Now again another part of the schema. As you can see here we are taking parts of the schema, breaking it down and bringing them down to the actual relational schema. Now here we saw that both members and authors are persons obviously of course. And why do we need that person relationship here? Because there were certain other requirements that we needed to capture. That is for every person in the system, we need to have some personal details like the pan number or the email address or whatever that we use to uniquely identify a person, the phone numbers, the address name, last name, first name and so on and so forth and whatever else that is required for maintaining records about a person.

(Refer Slide Time: 34:10)



Now we said that members and authors are persons, that is member is a person and author is a person. In addition they are disjoint that is the members and authors are disjoint. Why? There is again a specific safety condition that is explicitly specified in the requirements that no PC member ought to be or is allowed to be an author of any paper.

So you only submit papers, you only consider papers by authors who are not members of your program committee. And members have member_ids and so on and authors have author_ids as their words and a member can be a reviewer and a PC chair. So among here we already have tables for reviewer and PC chair and we saw that we don't need a table for member and we already have a table for author. So, all that we need is a table for person that is a person and pan number and so on and so forth. And when we say person, we also have to add all of these details but we see here that phone number is a multi valued attribute that is a person can have more than one phone numbers.

So when you have a multi valued attribute, you cannot, you don't know how many number of phone numbers a person is going to have. One person may have just one phone number, another person may have 5 phone numbers, another person may have 3 phone numbers and so on. So in order to deal with multi valued attributes, you have to create a separate table here called phone details where you put the primary key for person plus the phone number of the person.

So the primary key may repeat while the phone number may change, so you can have all possible phone numbers associated with a person. And of course you might have got a doubt that why are we creating a separate table called person here and not club them into members and authors. Because members and authors are mutually exclusive as well as they are collectively exhaustive as far as this system is concerned. That is in this system we are only talking about either a PC member or an author and nobody else as far as the requirements is concerned.

The answer is yes, you can do that is you can put, if you want to do that then what you have to do is take this pan number and bring it down to the member table or an author table so that it forms part of the primary key of each of them and then take all other details of person and bring it down to either member or author. And then for phone number, you need to take just this one the pan number of this one and then create the different phone numbers here and so that would form a good, I mean that would form a valid derivation schema.

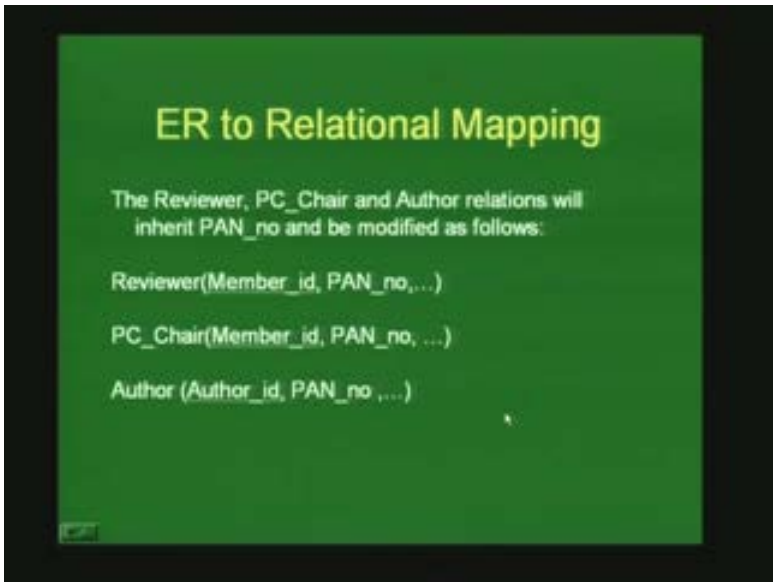
But as you can see the problem there is that when you bring pan number down to member, you suddenly have two key attributes there is a pan number and member_id. So which attribute do you associate with phone number? Now it is not incorrect, it is not in correct if I associated member_id with phone number as well in phone details it is not in correct if I say member_id plus phone number or it is not in correct if I say author_id and phone number but it is bad design. Why is it bad design? Because when you perform any kind of systems design, we should always be concerned about what kinds of changes will the system expect in the future.

So in the future we may want to add more categories of people into this system for example. So right now the only two categories of people we have in the system are either member or author. and they are mutually exclusive and collectively exhaustive but then they need not be collectively exhaustive in the future, we might add one more kind of a person maybe say delegate or a conference registrant or conference volunteer or something like that who is neither a member nor an author.

And in that case the conference delegate or a volunteer is also a person who has a phone number and pan number and so on and so forth. So the safest bet to associate phone number is with the pan number. So that is the primary key of the person attribute is the safest bet for associating the phone number attribute. So as far as this particular schema is concerned here, it is not in correct if I say that oh I won't have separate table called person but in the interest of or in the interest of anticipating what might be required in the future, this might not to be a good idea in future.

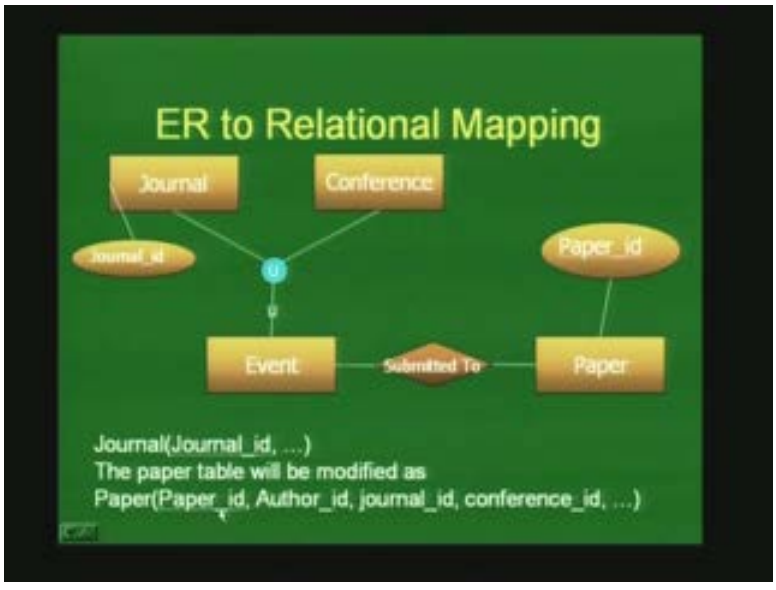
And the software design or any kind of information system design is full of such implicit requirements in the sense that if you write a software that works here and now, it is not usually sufficient. Your software should be extensible and extensible with minimal changes and with minimal impact and that is what makes a good design.

(Refer Slide Time: 40:11)



So basically this is what I said earlier that is one way to take this up is either bring all of them down that is from person bring it down to member. but then we see that member, there is no table called member as at all so bring it in turn down to reviewer or PC chair. So you can either bring down everything to either reviewer or PC chair and author who are all persons and then put pan number and every other attribute there. But in the interest of robustness of your system, it might probably be a good idea not do that and create a separate table called person itself.

(Refer Slide Time: 41:23)



Now coming to some more parts of the ER schema. So a paper maybe submitted, a paper may not be submitted to more than one conference or journal at the same time. That is if I am submitting a paper to this conference then one of the requirements is that, it is not submitted to any other conference neither is it submitted to any other journal. Note that even though we have not using journal at all in this system that is we are not, the journal is not playing an active role in the system but it is still required because we need to check whether a paper is submitted to a journal or not. And here we have, how did we model that using a conceptual model? We used a separate entity called event were an event could be either a conference or a journal and conference has its conference name has its key and journal has journal_id has its primary key and a paper can be submitted to 0 or 1 event that was what we had made here.

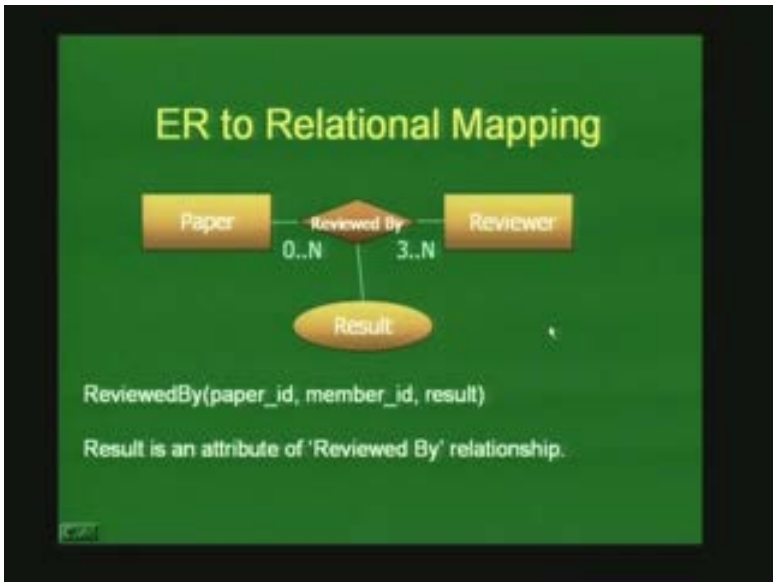
So how do you reduce this to a relational model? Conference, we already have a table called conference. So just create a table called journal as well because journal is an entity and it has its own key and attributes and so on. Now because a paper is submitted to one of these two, journal or a conference just take these two primary keys and add them as foreign keys here. So put both journal id and conference id or conference name rather as part of the paper. So given a paper, you know exactly to which journal or conference was it submitted.

Now note that these are added as foreign keys and not primary keys of course. that is they don't define the existence of the paper and a characteristic of foreign keys is that foreign keys can be null that is because a paper can be submitted at most to one of these two, that is one of either journal or a conference one of these is guaranteed to be null in any given tuple. In fact if they are, if both of them are non null then we have an integrity violation, we have a constraint violation in the system already.

And the last one is the attribute which is assigned to a relation itself that is a paper should be reviewed by at least three reviewers so 0 to N papers reviewed by 3 to N reviewers and the reviewer should assign some result accept, reject or neutral. And the result as we see here belongs neither to the paper nor to the reviewer. In fact the result belongs to the process of reviewing itself.

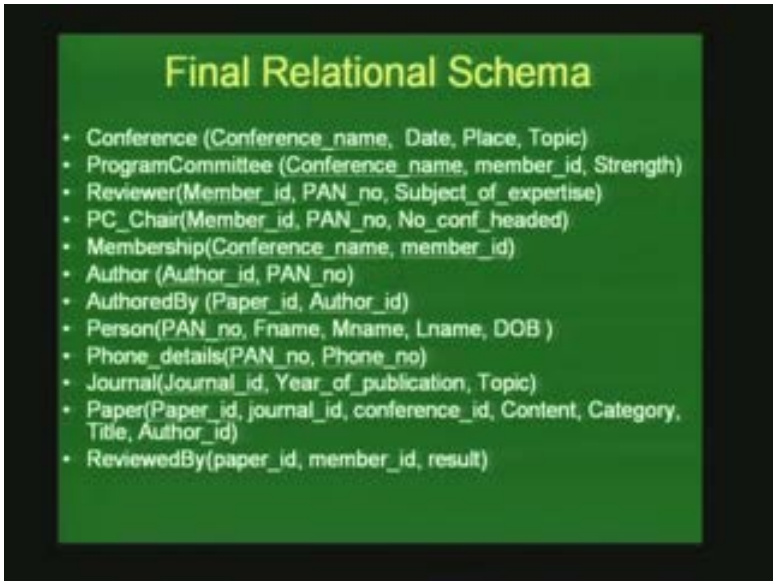
So the result belongs to the relationship called paper is reviewed by a reviewer. So how do we manage this? In order to capture that the result belongs to a relationship make a separate table in the name of the relationship. So reviewed by becomes a table and how do you recognize or how do you define tuples in this table. Take the paper id or take the primary key here, make part of the primary key here, take the primary key here member_id that becomes a primary key here and the result.

(Refer Slide Time: 43:38)



So for this paper_id, this member_id gave this result and so on. So whenever we have a relationship that is part of the, that is part of the rather whenever we have an attribute that is part of the relationship then we have to create a separate table in order to maintain this, in order to reduce this to the relational model.

(Refer Slide Time: 46:01)



So that brings us to the relational schema itself, so the final relational schema. So how does the relational schema look like? There are so many tables that that we have already found. There is the conference table which we created to begin with where conference

name is the primary key and then there is the date of the conference place, topic and so on and so forth.

And then there is the pro program committee table which actually represents the week entity type called program committee. Therefore it takes the primary key of the conference table namely conference name as its own primary key. And then there are other attributes like strength which is the attribute of the program committee and then there is member_id which is the foreign key into the PC chair table which says who is the PC chair of this program committee and so on.

And then we have the reviewer table, after the program committee table we have the reviewer table where member_id is the is the primary key and of course pan number, we inherited from the person attribute because we may require it for other purposes and then some other attributes like subject_of_expertise and so on. And then there is the PC chair table where member_id is again the primary key and maybe some other attributes like number of conferences that is headed and so on and whatever. And in addition we have inherited any other attributes from members or persons and so on, because PC chair is a member which is in turn a person and so on.

And then there is the membership table, remember were the membership table came from. The membership table came because a member or PC member can be a member of more than one conference committee and a conference committee would have more than one PC members and so on. So in order to maintain this N to N relationships, we created a separate table called membership where both conference name and member_id form the primary key and which are both foreign keys into their respective tables that is the conference and either program committee or either reviewer or PC chair and so on.

And it's actually the reviewer not the PC chair. And then we have the author table where author_id is the primary key then pan number is inherited from the person entity and everything else I have not specified any other attributes that an author might have. Then authored by is another table in order to denote the N to N relationships between an author and a paper. That is an author may submit any number of papers and a paper may have any number of authors, more than one authors in there.

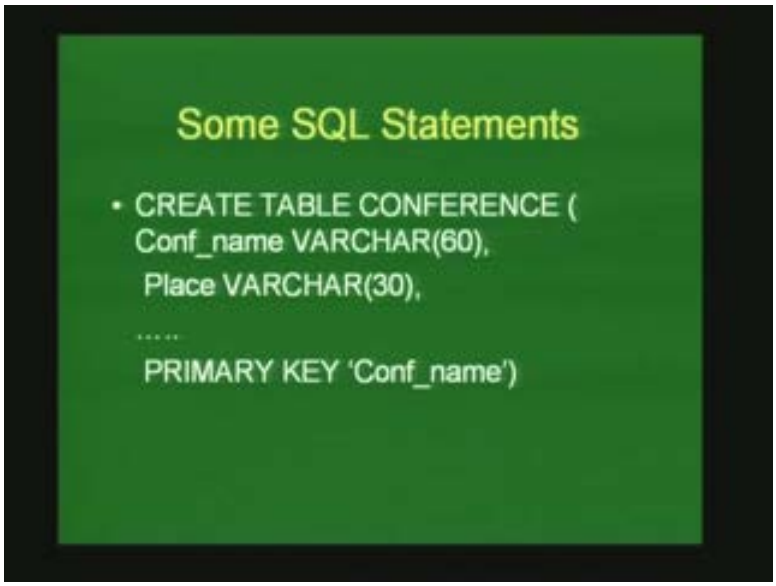
Similarly person is a separate entity. Now we created the separate entity called person like I said previously in the interest of extensibility of the design rather than even though it is not in correct to just take all this person details like first name, middle name, last name, date of birth and put them straight to reviewers or authors and so on. But in the interest of extensibility and the interest of the robustness of the system, we have made person into a separate table. And of course as you can see, there is no mathematical rule that says that you have to do that and its again a common sense decision that we expect that more kind of persons may be added to the program, may be added to the system at a later point in time and so on.

Then a phone details is another table where which is required because of the multi valued attribute in the person table were a person can have any number of phone numbers. So

you have a pan number and phone number combination which details any number of phone numbers that a person may have. And then there is the journal which is required just to keep track of the fact that a paper may not be submitted to more than one conference or journals at the same time. And then the journal_id, in fact for our purposes here the only requirement of journal is the journal_id or the primary key of the journal in this thing.

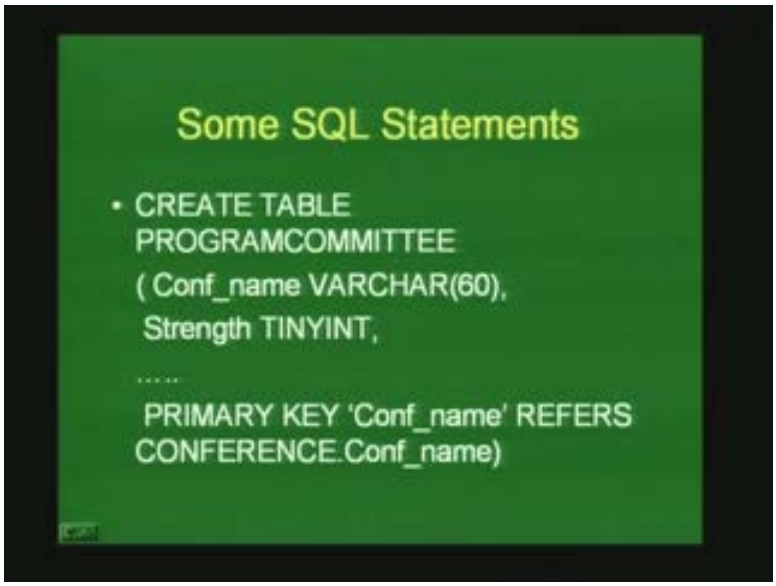
So the journal_id comes into the paper as well as conference name or not the conference_id and we can add an integrity constraint that no tuple shall have both journal_id and conference_id as non-null attributes or not null attributes and so on. And reviewed by is another table which comes out because of the attribute called result which is a member of the relationship reviewed by rather than a member of either paper or reviewer, so we get a separate table. So once I create table tables like this then you can essentially for each table, you can write the corresponding SQL statements.

(Refer Slide Time: 50:35)



So create table conference conf_name place and so on and say primary key conf conf_name and so on or something like create table program committee and conference name and strength and so on.

(Refer Slide Time: 50:57)

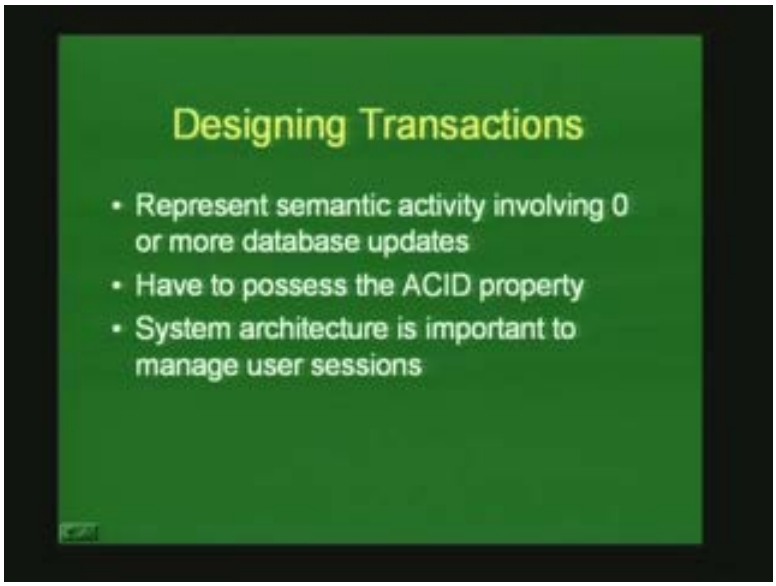


And say primary key is conference name and but conference name is the same conference name as the conference name of the other table. Therefore you have a foreign key relationship saying that it refers to that attribute in that other table and so on.

So, basically that I won't go in to the creation of those SQL statements again I am assuming that you know SQL and given a set of relational, I mean given a set of table specifications I am sure that you know how to write corresponding SQL statements and give it to a DBMS client which in turn can create a tables in a DBMS at the back end.

Now let us briefly come to the last aspect of today's lecture as well namely that of designing transactions or designing the dynamics of the system. We have not really talked about transactions as yet in the course of this series of lectures but nevertheless for the sake of completeness, let us just look at one or two transactions and we leave it at that and we won't go in to lot of details into transactions itself. So what's a transaction? A transaction is a set of activities or a set of tasks which represents one semantic activity or a set of database tasks like read, write, update and so on which represents zero or more database updates which represent one semantic activity. This is a very simplistic definition of transactions but we will just have it at that.

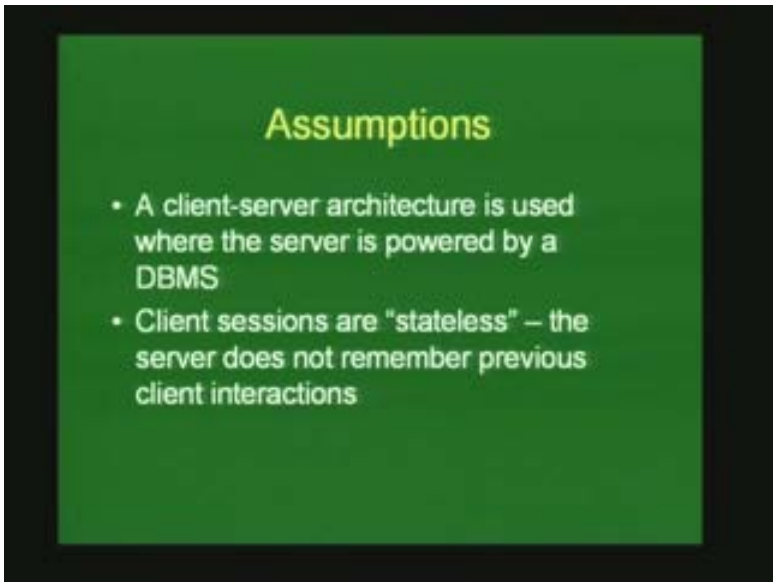
(Refer Slide Time: 52:29)



And transactions have to follow, have to possess what is called as the ACID property. Again I shall not be going into more details into this ACID property but you will be looking into ACID properties in a later lecture in much more detail. ACID basically means stands for atomicity consistency isolation and durability. So essentially what it means is that the entire set of activities or tasks that form a transaction should be performed as one atomic whole either you do the entire thing or none of them.

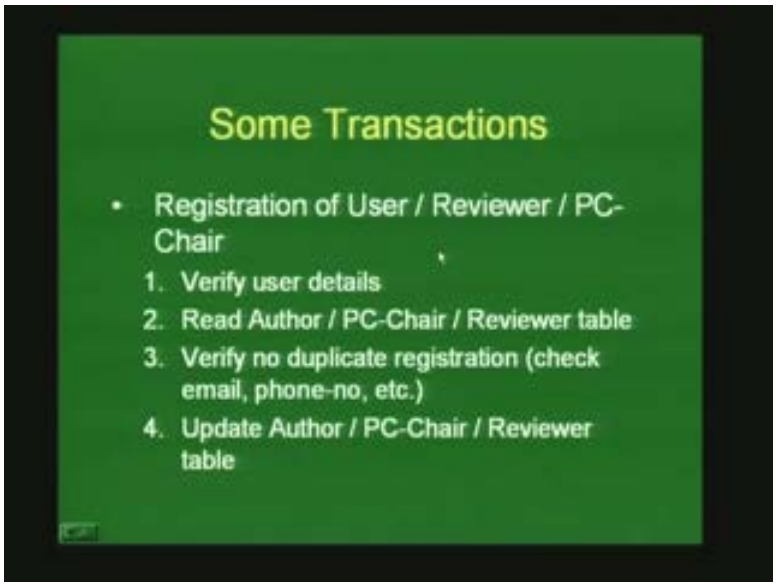
And they should be performed in isolation of other transactions. So no two activities of, no activities of two or more transactions should interfere with one another logically and so on. And in order to manage a design transaction, we should also know what kind of system architecture we use in this. So let us make some assumptions about the system architecture. So we are using a client server architecture where the databases serve or is powered by a DBMS or a database management systems and the client sessions are stateless. Meaning what? The server does not remember previous client interaction. So every time you need the server to do something, you need to give every possible requirement or every possible data to it in order to perform this activities.

(Refer Slide Time: 53:42)



So let us look at some transactions some simple transactions that give us an idea as to what kind of activities would go on. So let us say registration, let us say I want to register a new author or user or whatever reviewer or PC chair into the system.

(Refer Slide Time: 54:37)



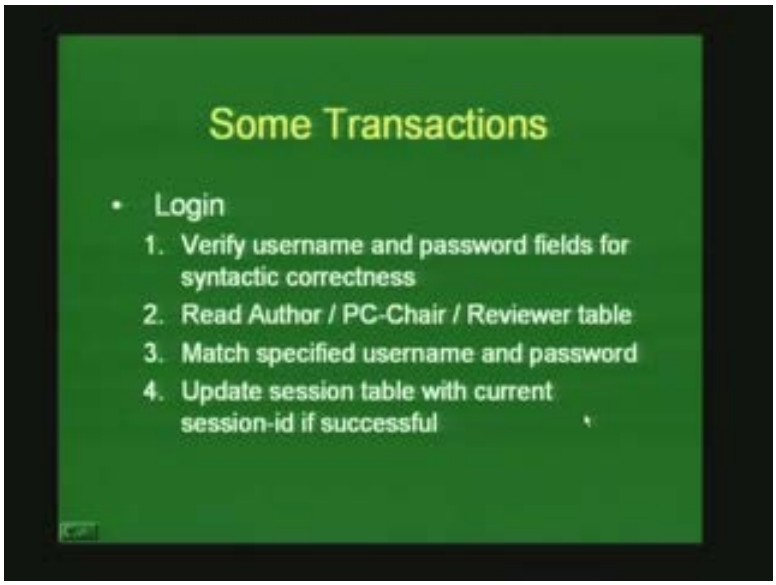
So how would you register? Typically you would have a form in front of you where you would have all details. What are the details? look at the person or author or reviewer tables and just make all those details there, the name, first name, last name, pan number and so on and so forth and phone numbers and so on.

So verify those details for any syntactic errors then read the corresponding tables, verify that there are no duplicate registration that is one person is not trying to register multiple times. Once this is not there, once this is not the case, once we verify that this is not the case then you can update the table.

Now what this means is that all this four activities have to be performed as one semantic whole. As you can see how many different activities are happening here, there are N number of reads happening here and one updates happening here either all of them should happen or none of them should happen. So, all of these form one semantic activity called the transaction.

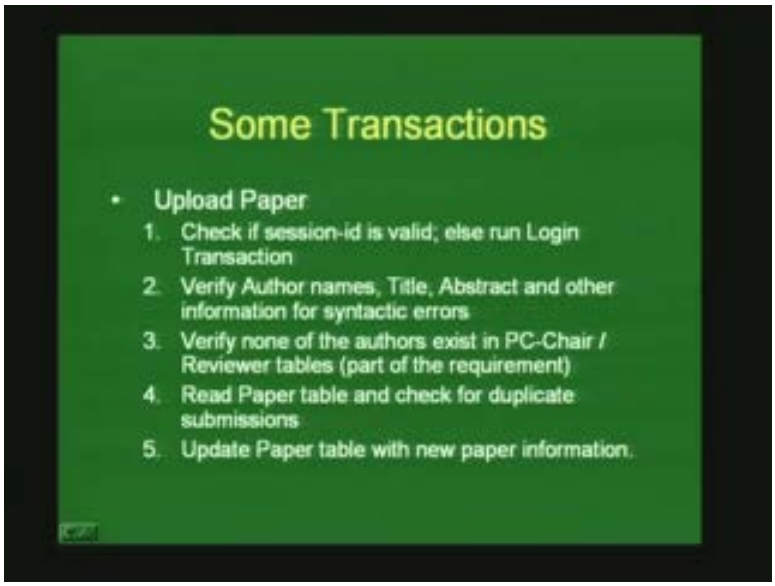
Similarly login. So let us say that I have created user id for myself, I have to login. So verify user name and password field for syntactic correctness that is I provide my user name and password for login. Then read the corresponding table author, PC chair, reviewer table, match specified user name and password, we have not really taken care of password then.

(Refer Slide Time: 55:35)



So another idea here is that once you start designing transactions, you will see that you need these two attributes also that is user name and password which we had not factored in at all when we are looking at the ER or the relational model. So you have to go back and make those changes and this thing. And then you have to maintain sessions and so on and let me not go into sessions, session management and so on. And one more transaction which I am not going to go into detail here, upload paper transaction or whatever.

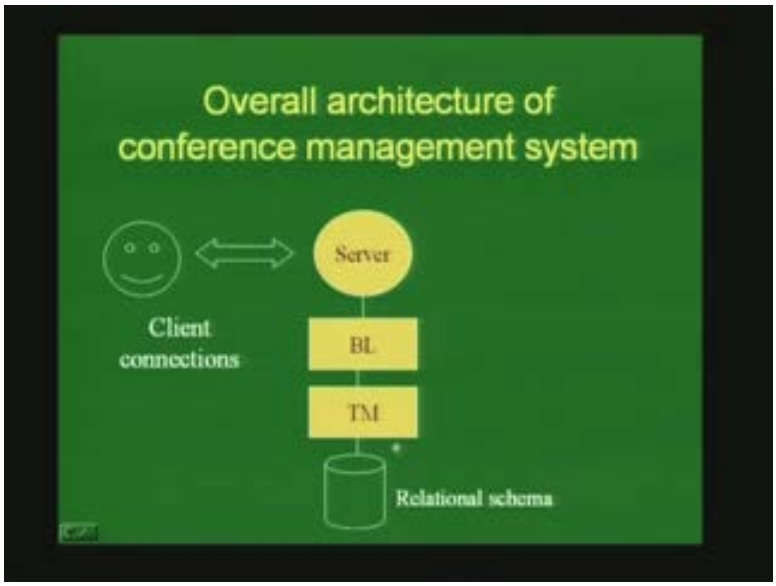
(Refer Slide Time: 56:19)



(Refer Slide Time: 56:27)

And you can think of several more transactions like this assign reviewers, submit review, arbitrate so all of these are semantic activities which perform zero or more updates of the entire system. So how would the entire system look like? See why did I just take up transactions for a brief while because just to show you how does the entire system look like.

(Refer Slide Time: 56:49)



So here we had the ER model and from the ER model we develop the relational schema and this is the DBMS here, actually this whole thing would be the DBMS, so this is the database here. And on top of the relational schema there would be a transaction manager which would manage transactions, on top of this would be what is called as a business logic manager which handles multiple transactions and so on which in turn would be served by some kind of a DBMS server through which the client connections are handled.

So this would, basically this would conclude the overall system design of one particular case study of the entire system. Of course firstly, we are not comprehensive I mean this not comprehensive enough to take up a real life system. But mainly the whole idea here was to give an idea, the whole idea behind this is to give you a perspective into what it takes into designing a particular DBMS. So with that let us come to the end of the session.