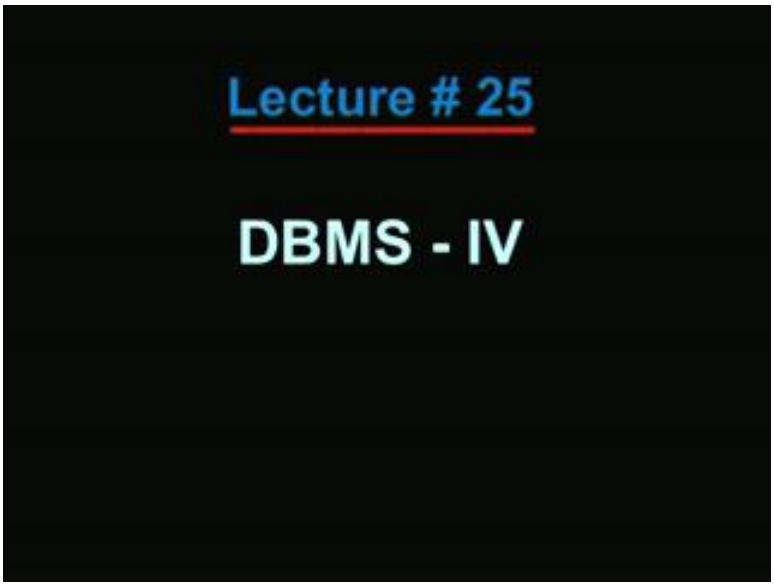
Management Information System
Prof. Biswajit Mahanty
Department of Industrial Engineering & Management
Indian Institute of Technology, Kharagpur

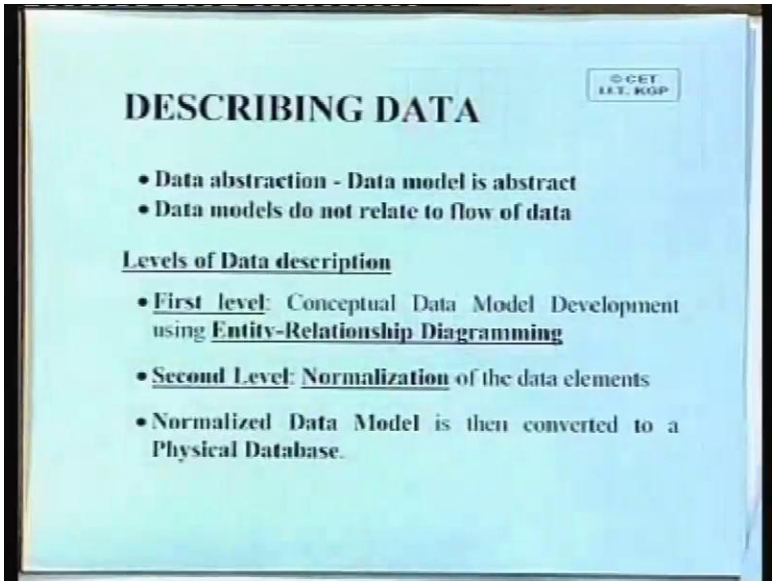
Lecture No. # 25
System Design – II
DBMS – IV

(Refer Slide Time: 00:47)



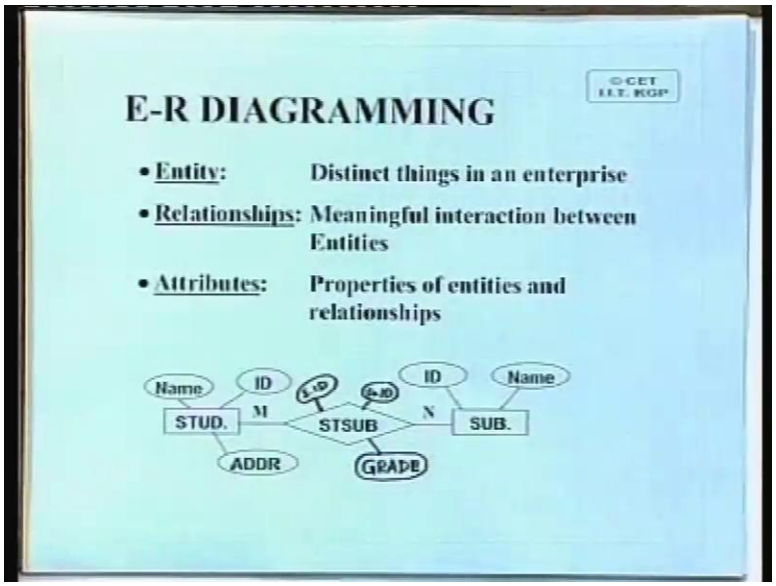
Data description. Essentially what I was telling you on the previous class that the data description is a kind of data abstraction.

(Refer Slide Time: 01:01)



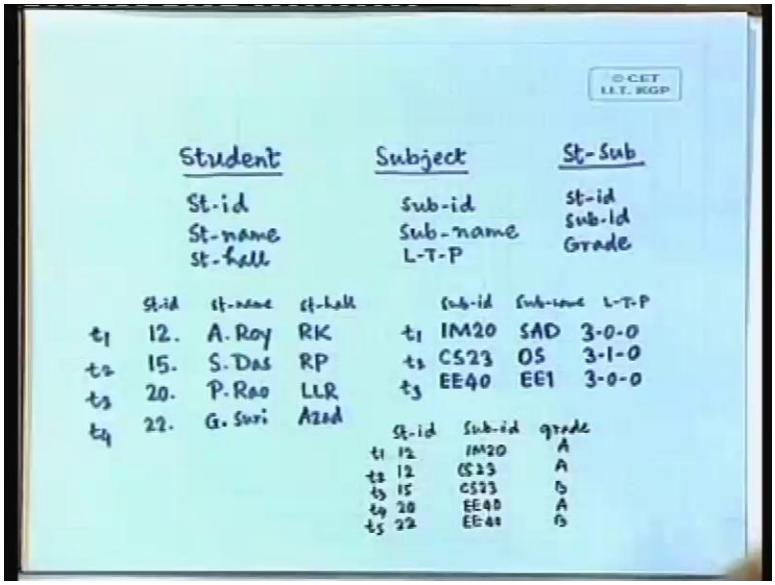
And it is independent of the data flows and there are 2 ways. It can be done 1 is entity relationship diagramming and the second is through the normalization.

(Refer Slide Time: 01:14)



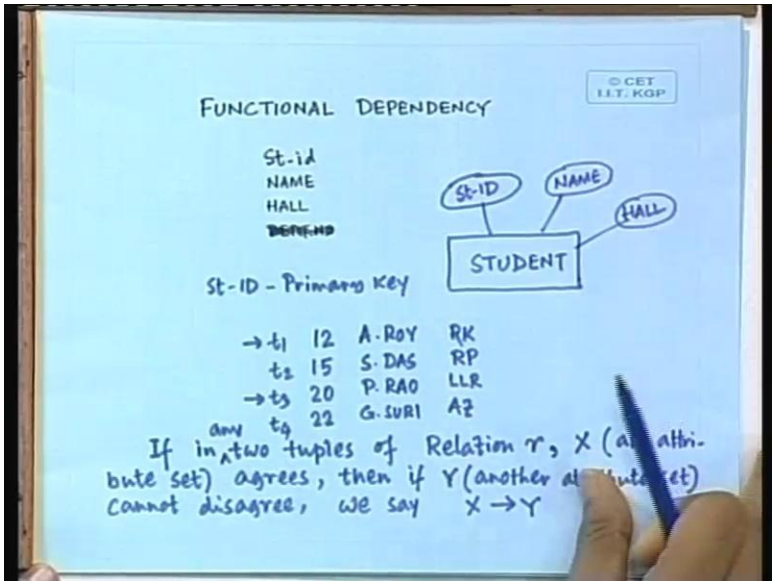
Then I was discussing about a very simple problem where we have the student and the subject. And these are the 2 entities with certain attributes and there was a relationship called STSUB and the relationship. See the relationship should also share the ID's of both the entities. So you can say the student ID and the subject ID. So both the student ID and the subject ID must also be shared by STBUB right.

(Refer Slide Time: 02:04)



So using this we can you can see here that we have the student id here, the subject we have subject id and in the STSUB we have the student id as well as the subject id right. So from a primary key point of view the student id is the primary key of the student. The subject id is the primary key of the subject, subject table and for the STSUB we have the student id and subject id together is the primary key right. Now in this context there is I have also told you about the foreign key concept with regard to STSUB, this ST id and subject id together is the primary key. However both ST id and subject id are foreign key because they are primary key to some other tables all right and this is important for keeping what is known as referential integrity. Now before we proceed, let us understand something very basic. See there is a concept called functional dependency. Will try to understand the functional dependency mathematically.

(Refer Slide Time: 03:40)

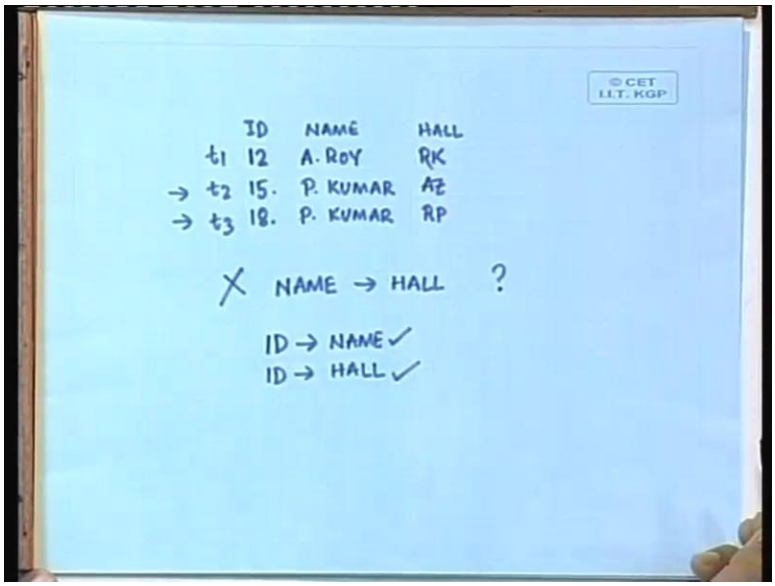


But before that let us understand in a very simple manner. See let us say we have the student id as the key and we have a number of say from an entity relationship diagram. It may look like this. Say we have the student as an entity. Now it has got ST id name hall okay. Now we have kept department number. But this would be more complicated problem. So for the time being let us not consider department number right. Because it will be little complicated to understand right in the beginning. So forget it for the time being we have the student id this name its hall so the student is like an entity and these are its attributes. Now we say here that ST id is the primary key. Why it is a primary key? Because, yes, is a primary key because it uniquely identifies uniquely identifies all students each student okay. So if we have a number of students like say some tuples, let us consider same thing I am writing RP. So you see we have 4 tuples the idea is if we take any 2 tuples. If we take any 2 tuples and we see the student id is same we take any 2 tuples. Say for example we take t1 and t3.

Now definitely the student id is not same is not same. So it does not matter but if the student id is same in 2 tuples. Due to whatever reason then the name let us say another field name if the name cannot disagree. So it is like this. Let me write if in 2 tuples of a of a relation r. So if in 2 tuples of relation r see we are using the small r usually you must be able to differentiate between relation and relation scheme the scheme is usually identified by a capital letter and the relation is

that means the data sets that is identified by small letter right. X is an attribute set already agrees then if y cannot disagree we say x functionally determines y all right. Is it clear? So this is the idea now due to whatever reason you have seen that in some classes. There is a same name same name of 2 students say Pradeep Kumar very common name. So we have 2 Pradeep Kumars may be in the same institute. But so what is happening suppose we think of name as the key think of there are 2 Pradeep Kumars. Let us take this example.

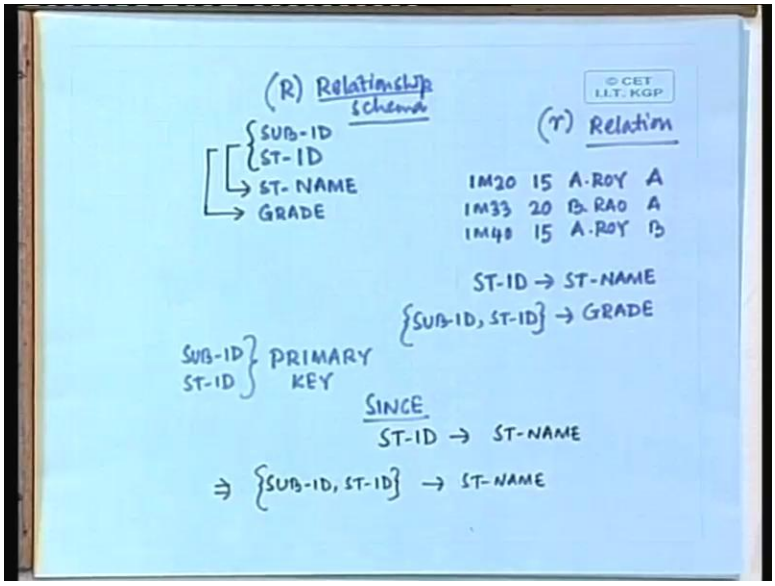
(Refer Slide Time: 10:24)



See, 12, A Roy RK, 15, P Kumar. So we have these 3 tuples taken here t1, t2 you see look at this 2 tuples t 2 and t 3. Now we want to find out whether this is true. Is it true you see what is happening the, what is a definition of functional dependency in any 2 tuples. See it is not that just you take tuples if in any 2 tuples. So I have not written any perhaps. So let us write that any that means if in any 2 tuples of relation r x agrees then y cannot disagree then if y cannot disagree. So in this case in these 2 tuples we have the name has agreed can you see that the name is same but what about hall they are different. So this is not true so this is the idea of functional dependency. That means if name is same in 2 tuples hall should also be same if we want name determine hall. So this is 1 disadvantage of name as an unique key as an unique key right. So we say that yes this is functionally dependent but are these 2 tuples true sorry these 2 functional dependence is true. Can you say that id determines name id determines hall we say that both are true okay. So these

2 functional dependencies are perfectly all right. Whereas this is not true okay. Let's take another example.

(Refer Slide Time: 13:40)



Take this example we have student name we have student id we have subject id we have student name we have grade. So let us take some example also. Let us say IM 20, 15 A Roy A IM 33, 20. Think about this. We have shown some tuples and this is another set of this is a relation schema R see this is R this is R. See we differentiated this way. This is a relationship schema and this is a relation. I think you are understanding. The relation see the, it is not that this data is a relation that relations includes the schema definitely without the schema how do you have a relation but also includes the data. So we have a relation R which includes the relationship schema capital R and it set of data and definitely its definition of functional dependencies a set of functional dependencies right so a relation includes schema that data and set of functional dependencies. Now if this is our schema tell me what is a primary key student id can it be primary key? Yes or No. See because if you look at it from a functional dependency point of view then the grade requires both the id's whereas ST name requires only student id.

So these are the 2 functional dependencies right so these are the 2 functional dependencies that are defined here. One is student id determines student name or student name is functionally dependent on student id and the grade is functionally dependent on subject id and student id clear. So we can say that by using these functional dependency sets that subject id and student id together is the primary key see this is the primary key. See for primary key that should be uniquely determining uniquely determining all the other attributes. In other words all the other attributes should be functionally dependent on them. But we can see that here the ST name is dependent on part of the key rather than on the complete key. But if the part of the key agrees can the whole key not disagree is it possible. See the question. Suppose ST id alone determines ST name that means in 2 tuples if ST id agrees ST name has to agree all right. So in 2 tuples if subject id and ST id both agrees can ST name be different not possible so from this fundamental calculations we can say that since we can write this.

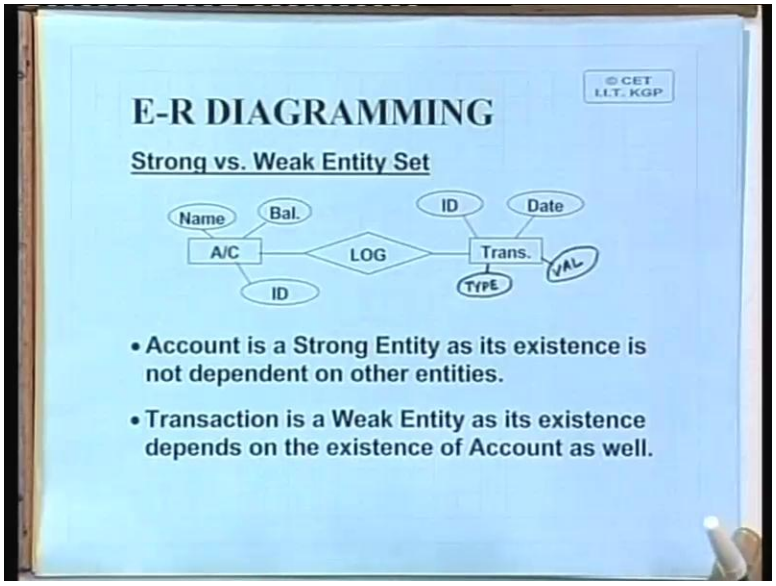
Since we have this can be derived all right. Now how to derive this mathematically there are something called Armstrong's axioms. So those things we shall discuss later, but at least this much you understand at this much of time. For something to be defined as a primary key all the other attributes. See this is we are trying to define as a primary key. So all the other attributes must be functionally dependent on the primary key all right. So basic definition of primary key are all the non key attributes should be a fact of the primary key. That means all the non key attributes should be functionally determined by the primary key is it clear so that is the thing. But there are problems in this relationship scheme right. These particular scheme the schema is not very good can you see why because look at these student name is repeated student name is repeated in 2 tuples. So if you keep if something is wrong in 1 of the 1 of the tuples then it will be difficult to know which 1 is the correct name all right.

Suppose you want to make the data base proper that means all the students name corresponding to the student id should be all the time unique. Since student id uniquely determines the student name all right. So due to whatever reason if 1 tuples instead of A Roy you mistakenly enter it A Ray instead of ROY mistakenly entered RAY. So which 1 is correct ROY or RAY that confusion will come and even if you decide that yes it is ROY then you have to go through all the records and correct this update anomaly. So as I was telling you there are 2 kinds of problems 1 is called update anomaly and the second 1 is the pure redundancy. Since you are keeping

student name in many such records you are having more space than is required right so that is called redundancy. Redundancy parse is not a big problem; but update anomaly is a bigger problem but there are advantages also. You know what is the advantage? Advantage is if you can somehow keep the data like this all the data is available in 1 one table.

So query answering could be easier query answering could be easier. This is why many people still sometimes think that a flat file may be better in some peculiar situations. (( )) (22:20) See it is not a question of whether you define a view or not define a view the basic idea is that if you keep the relationship schema like this this schema is prone to redundancy and prone to update anomaly all right. Now as long as these 2 things are present your data is not integrated. But what is you have told right in the beginning 1 of the primary objective is to have an highly integrated database data integrity 1 of the very basic objective of DBMS okay. Now coming back to will come back to the dependency and this is 1 of the fundamental understanding that we require. So we come back to functional redundancy later on. But let us first complete our discussion on entity relationship diagramming.

(Refer Slide Time: 23:20)



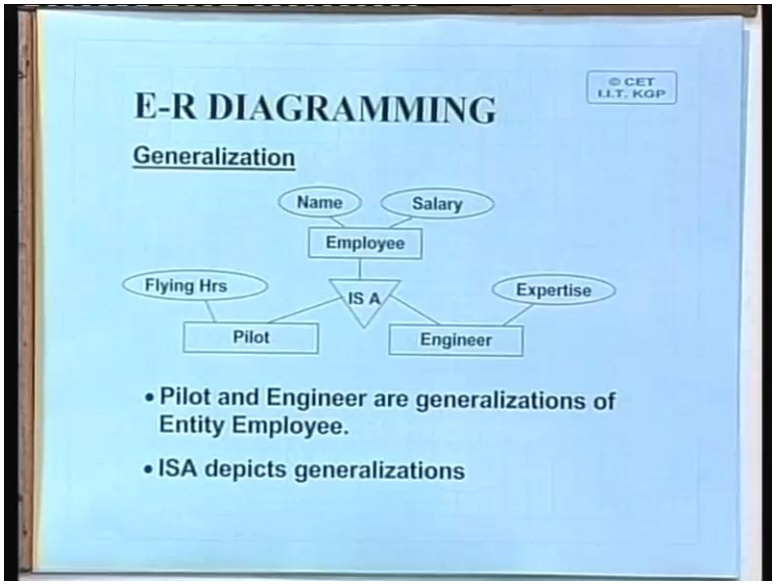
So this is another concept see we have understood 2, 3 things on our previous class. We have understood that what is an entity relationship diagram and what is relationship cardinality right. One is to m, m is to n and 1 is to 1 will give more examples. Let us see here that in entity

relationship diagramming we have the strong versus weak entity set is another important concept. That is when you take a student and a subject the student and a subject okay fine. The student is a separate entity subject is a separate entity both are strong entity sets. In 90 of the situations we have only strong entity sets. There are no difficulties but in some peculiar cases we have so called weak entity sets. What is an weak entity set an weak entity set has a definitely it can be given a primary key. Primary key for example an account and a transaction okay. Suppose you have an account that means you have some money in the bank and the account as an id it is like your account number and the name and the balance that you have in that account definitely many more things might be there.

But there is a minimal set. Then the transaction that you are doing suppose on your account you are depositing some money you are withdrawing some money. So all these are like transactions to your account. Now we can give each transaction a transaction number and date and definitely many other things like value. What is the type of transaction? So I have not given all those things here but definitely they can be included. For example value of these transaction what is being how much money and type what is the type of transaction that means is it a withdrawal or is it a deposit all right. So is it a deposit or is it a withdrawal or what is the value of transaction. Now definitely the transaction id is a primary key. There is no difficulty in that please understand. This that suppose each transaction is given a running serial number that running serial number along with the date or not, even not with the date uniquely should determine all the other attributes. So id will be a primary key no doubt about it but think from a practical point of view will it serve any purpose will it serve any purpose.

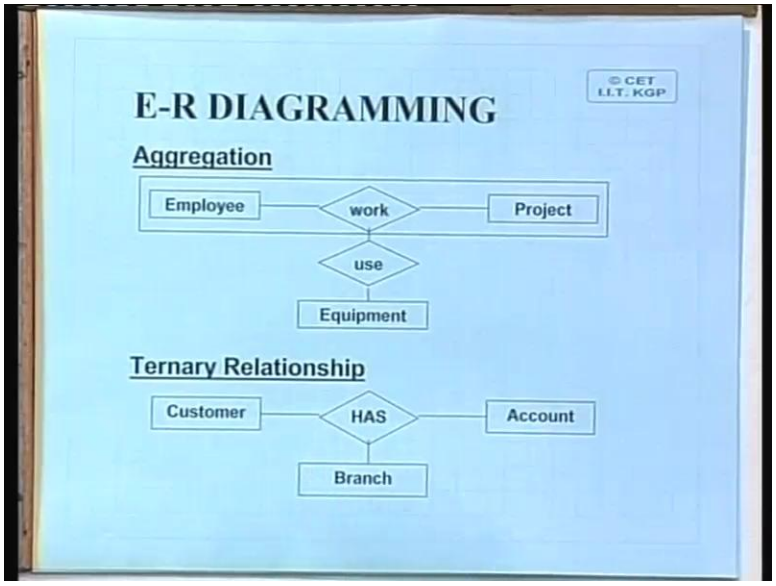
If we could cannot relate the transaction to the account because every transaction is tied with an account all right. So we should be very easily able to identify the transaction with the account otherwise we have to separately maintain to which account the transaction was there all right. So that is why the transaction should also share the id of the account that is the problem of the weak entity set. So its existence depends on the existence of the account as well basically here you have to read existence in terms of sharing the id. In other words whenever you are writing a transaction the transaction should have the account id as well as the transaction id all right. Only then the transaction can be properly maintained. So these must this concept you 1 should be clear about that there is a strong and there is a weak entity set.

(Refer Slide Time: 27:29)



Then the next concept is that of generalization. Generalization you will see later is an important concept because this is going to give us the concept of inheritance. Inheritance in the object oriented analysis and design. So suppose we have an employee, the employee could be a pilot employee could be an engineer right. So whatever is attributes of employee are definitely will also be attributes of pilot or engineer all right. So whenever we are thinking of the general things like name address id salary so on and so forth we can put it under employee. But specific things like flying hour's specific things like expertise specific expertise of an engineer we can put it under the generalizations. So in the sense that this is actually shown in the form of a IS A. IS A relationship I have shown in a triangular form and these IS A basically shows that pilot is an employee engineer is also employee the pilot is a employee engineer is an employee. So pilot is definitely sharing all the attributes of employee an engineer is also sharing all the attributes of the employee clear. That is known as generalizations.

(Refer Slide Time: 29:10)

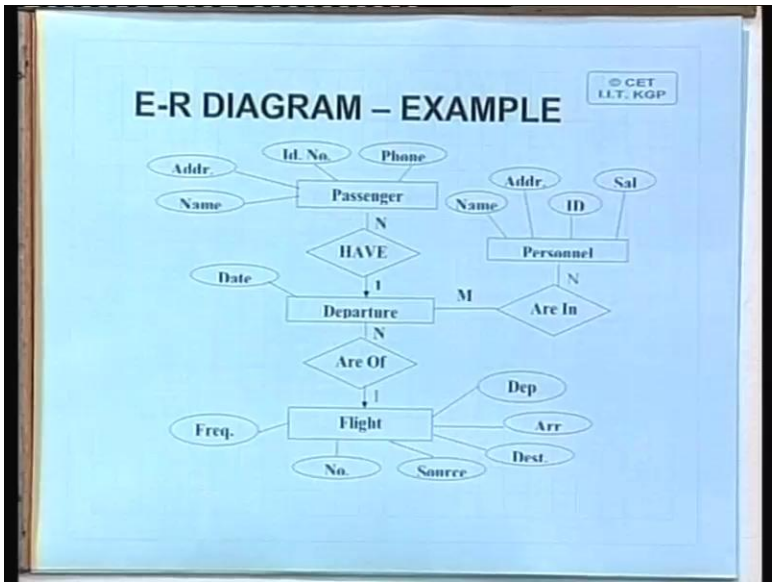


Then aggregation and ternary relationship let us see the ternary relationship first. See sometimes the relationship may not be between 2 between 2 entities it could be between 3 entities. So it will be a little more difficult to handle say for example customer, account and branch. So customer has an account in a branch right. So you must have an identifier for the customer, for the account and also for the branch all 3 together fine. Usually whenever you have a ternary relationship it is definitely difficult to handle. Otherwise what you probably can do in state of a ternary relationship you may have to have 3 way relationships. See either you the choice often becomes either you have a ternary relationship or you accomplish it through 3 way relationships. So you have to handle lot of additional information but ternary relationships are difficult to handle aggregation. See look at here that employee first you think of the use the basically employee project an equipment together as a ternary relationship. So there are some equipment which are being used for specific projects by a specific employee all right. So it is it is something like this. Suppose you have bought sophisticated equipment by pooling money from several projects several projects.

So what is happening these equipment is being shared by all employees in a project but are also shared by all employees in another project where the employees could be common may be the same employee is using but under different projects all right. So suppose you are an employee of

both project 1 and project 2. So you have use the equipment in total let us say for 100 hours, but out of that 50 hours in project 150 hours in project 2. So we have to keep it separately because of that we require all the 3. But there is another peculiar thing we also want to keep employee and project also want to keep employee work in a project where equipment has nothing to do all right. So employee and project share an binary relationship an employee project and equipment share an ternary relationship we need to show all the relationships. So this is how it can be aggregated we can show it in these manner all right. That is the basic use of aggregation.

(Refer Slide Time: 32:27)



Let us try to have a look at an E-R diagram an example of an E-R diagram in some detail. Think of an airline company where we have passengers and these passengers have departure see we have differentiated between departure and flight. What is the difference between flight and a departure the flight has a number all right. So let us say some number like I C something say 4 0 3 okay. So this particular number I C 4 0 3 is a flight. When is this flight may be in the evening may be in the evening. Everyday in the evening moving from city 1 to city 2 all right. So this particular flight has got a number its frequency. What is its frequency daily in the evening source and destination arrival time and the departure time right. So it has a number it has a source it has a destination it has an arrival time and it has a departure time and it has a frequency. The departure you may say is of every flight every flight has a departure in a sense that a particular

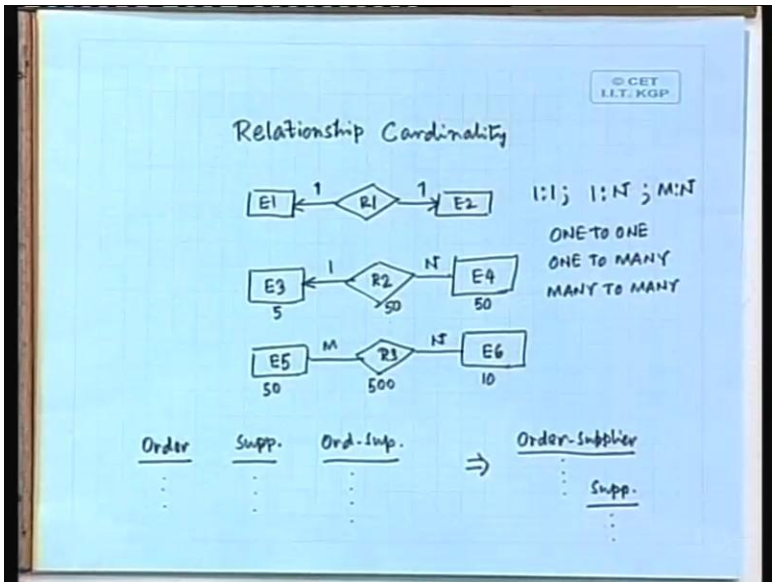
flight of a particular day okay I C 4 0 3. But I C 4 0 3 of today is different from I C 4 0 3 of tomorrow is it clear that is what is defined as a departure clear.

So that is a departure and the passengers are could be actually the passengers are assigned to a departure. So you can see you may you may ask why are we keeping flight and departure separately. Can you tell me why cannot we keep departure in 1 file itself? What is the disadvantage if you would have done that? Lot of repetitions of data would have become because all these general information about flight suppose your airline company has got 10 flight. So you have just 10 flight records. But over a year the 10 flights assuming each flight is daily once would have made 3650 departures. So if you are keeping 1 at least 1 year record then there are 10 records only 10 flights and 3 6 5 0 departures all right. Now if you want to know about a particular departure the general information you can always take it from the flight details. Obviously you may like to know some for a particular departure although the arrival time and departure time is fixed the actual arrival actual departure could be different that you may store separately in the departure all right. So this is a very skeleton model you can definitely add much more to it depending on your requirements.

So it is very clear that flight to departure is an 1 is to n relationship, 1 is to n relationship. What about departure to passenger? A passenger can be booked in many departures or 1 departure. See this is the fundamental difference in view. See you are a passenger you can you can book yourself in many departures. It is not necessary that you have to go by 1 plain only once and that is the end of it you can definitely be booked in many departures. On the other hand a departure has got many passengers all right. So we could have add an many to many relationship but the problem is for an large airline company or a large rail company to identify every passengers separately is difficult impossible almost impossible. Until unless some very specific schemes are created for a very specific schemes okay fine. But for running the general show it is better to even if the same person is coming again and again it's better to give him a new passenger number. Have you understood the basic idea? You may be booked by a train today you might be going with the same train tomorrow. But you should not be given the same passenger number as for as railways are concerned you are another passenger okay. This is for general scheme but definitely even I know that airline companies definitely have right.

They also have some past records they are trying to build up some kind of customer base so that they can identify frequent fliers have specific scheme for frequent fliers feedbacks giving some specific benefits to person who travels a lot all these things. That is specific schemes. But this is where we are discussing the very general schemes so we keep it as 1 is to n. Now each departure has got some personnel. Personnel, who are these personnel? The personnel could be the crew. The crew that are assign to a particular departure. So we have the departure we have the personnel for their id, name, address, salary, etcetera, etcetera and we have various m is to n relationships fine. So this is our entity relationship diagram and how do you design data base out of the entity relationship diagram? Very simple each entity will be a table each relationship will be a table but before doing it we have to do little bit of analysis. What is that little bit of analysis particularly in the case of 1 is to n, you will see many of the 1 is to n I was discussing yesterday that these particular example in the relationship cardinality particularly 1 is to n.

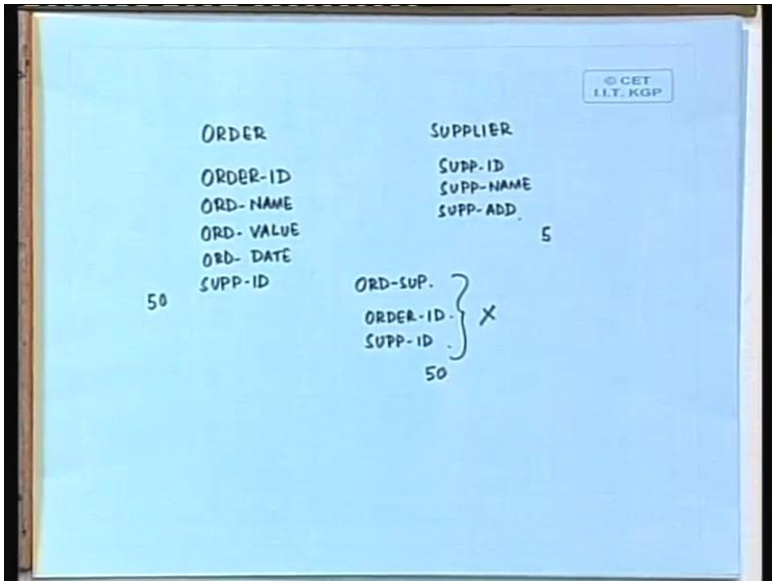
(Refer Slide Time: 39:32)



Say between order and supplier since the number of records would be exactly same. See look at these example 550 50 we were discussing yesterday order and supplier all right. We can have an order details we can have a supplier details and we can have an order supplier details. But since there is a 1 is to many relationship and that means every order has got only 1 supplier instead of separately keeping order supplier. We can add supplier number in the order details then it does

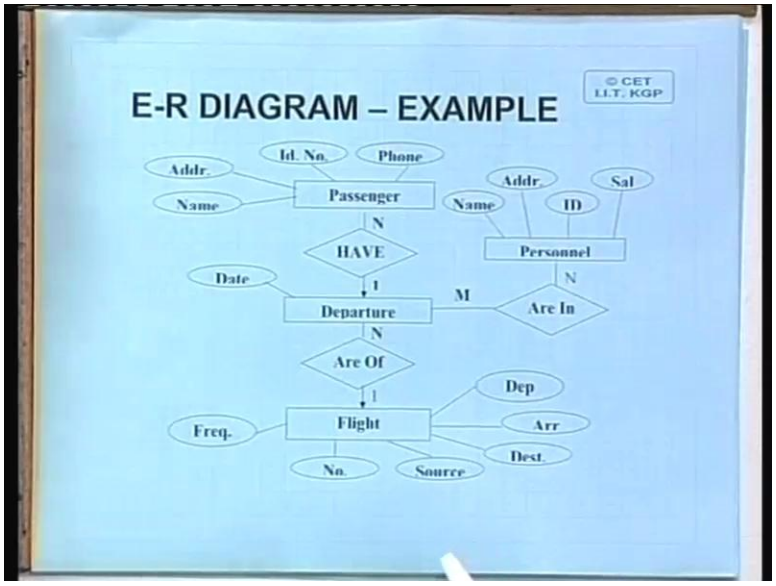
not lead to redundancy because the number of records are exactly same all right. So we merge the 2 files and we simply keep order supplier and supplier we may call it order, just order. So we have the order number let us look at it at some detail so that your understanding becomes clearer.

(Refer Slide Time: 41:59)



Right. So order file has got order id, order name, order value, order date. Supplier has got supplier id, supplier name, supplier address. Order supplier has got order id and supplier id right. Now what you may say here is that if we keep this 3 files and let us say there are 50 orders and let us say there are 5 suppliers. Since every order can have only 1 supplier the order supplier also will have 50 50 records. So between the order and the order supplier you look at this here also you have 50 records here also you have 50 records. So we can as well merge the 2 and we can keep the supplier id here and this we may decide to forget all right. This is what I said in the previous class. So when you are translating basically this discussion came up because how to translate the entity relationships diagram to database.

(Refer Slide Time: 43:17)



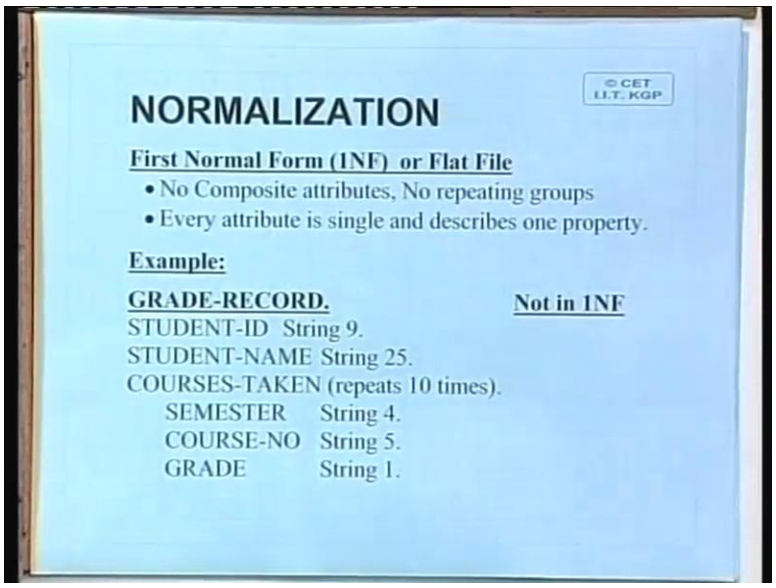
So what I said is you translate each entity to a table each relationship to a table. But before doing certain analysis what is that analysis? In analysis is that between flight and departure we can see the similar situation all right. So there is no need to keep this are of separately. We can simply keep flight we can simply keep departure because departure anyway is an weak entity can you see that can you see that departure is an weak entity without flight number the departure has got little meaning right. So when you translate this to database we have the flight as a table flight. That means a departure as another table now tell me do you require have or not. This is although it is 1 is to N. See we have the departure details we have the passenger details. Now separately do we keep the HAVE? That is which includes the departure hash and the passenger hash do we keep it or not. Why we have we have not kept are of?

One is to N every departure has got many passengers every passenger has got only 1 departure think about it. See the basic question is that whenever we have the departure each departure has got many passengers each departure has got many passengers. And usually what you may have is for backup purposes. You need to keep or you may like to keep lot of data about the old passengers please understand this all right. You need to keep lot of data about the old passengers fine. Now you can definitely add whenever you have some passengers the departure number in them departure number in them. But sometimes you may decide that since a passenger might

have taken lot of departures and more than that we are more interested in the kind of passengers that we are having and in the have relationship. We can have only the current people who are booked try to understand. What is of immediate attention to the airline company? The passengers who are immediately booked to you who have already completed your what you say service or has already gone. That is fine.

But a passenger who are booked and for the immediate departures in the next few days are going to be important to us. So if the company wants to keep those data only live, that is very important. Because if suppose a company is here for 10 years then there could be 36500 departures each with so many passengers for railways is it is mind boggling number right. So you cannot keep all these data online right. You may like to keep only the recent data online all right. To have this you can keep it in this relationship scheme. So that is why I said that you have to think about the current situation and accordingly decide. Similarly M is to N relationship you have to have all 3; you cannot cut of the relationship scheme. So accordingly you can design your database right. So that is about entity relationship diagram.

(Refer Slide Time: 47:37)

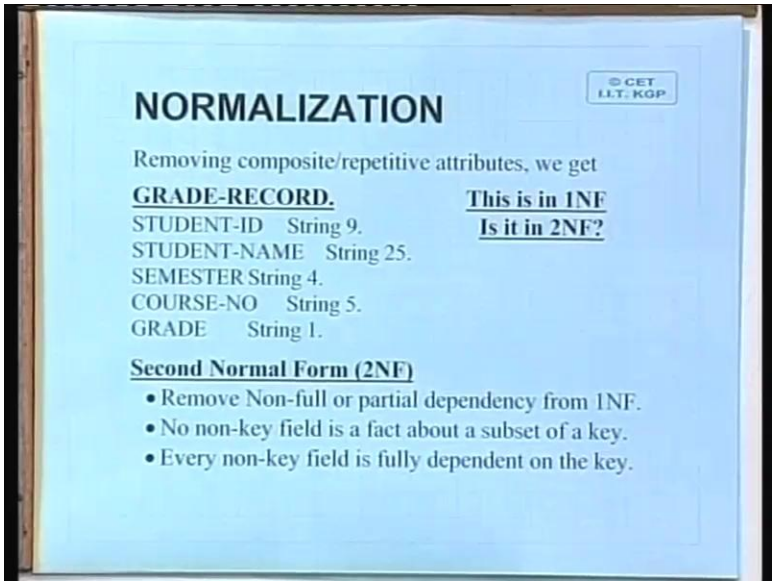


Now let us see how this translates to normalization. Now as I said the normalization we shall have basically 2 approaches 1 is a layman's approach by first we try to understand normalization from a layman's approach. After we have understood normalization from a layman's approach

then we try to understand it from a mathematical point of view. So in the layman's approach the first thing the there are various kinds of normal forms. See whenever you have a database table the database table you may say or a set of tables you can say in are various kinds of normal forms. What are those various normal forms first normal form, second normal form, third normal form, fourth number normal form, fifth normal form and voice cod normal form. So there are 6 normal forms they are immediately available. Out of this all these different normal forms the third normal form is the most important right. The fourth and fifth normal form cannot be understood until unless you understand another concept known as multi value dependency. We have understood functional dependency.

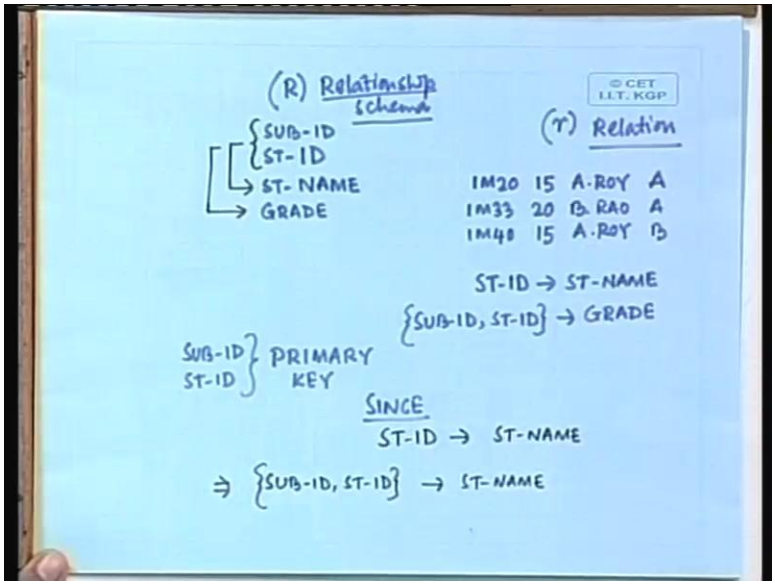
Similarly you can have what is known as multi valued dependency the concept of multi value dependency is required to understand fourth and fifth normal form right. The Boyce-Codd normal form is difficult to achieve many a time Boyce-Codd normal form is better than the third normal form definitely but it is difficult to achieve all right. In many situation Boyce-Codd normal form will be difficult to achieve therefore most of the time you know for most of the practical database design definitely from application point of view the third normal form we try to achieve clear. So we try to understand the third normal form first up to the third normal form first. And later on we shall see mathematically how this can be done. So let us today only do the first normal form in the first normal form. We have no composite attributes and no repeating groups. This you have already understood that is every attribute is single and describes 1 property. Look at this example in a grade record we have the student id, student name, course taken, semester course number, grade, 10 times it is repeated. This example we have already seen all right. This is not in first normal form. How to bring it to first normal form?

(Refer Slide Time: 51:49)



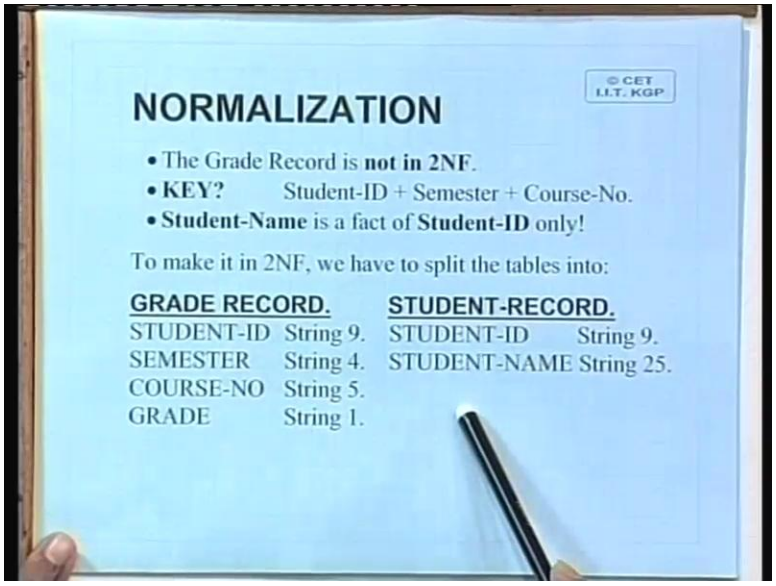
You have to remove, composite our repetitive attributes. So when you do that you have got student id student name semester course number grade this we have discussed in some detail, so this is in first normal form. Now question is it in second normal form. The second normal form requires removal of non-full or partial dependency from first normal form. That is no non-key field should be a fact about a subset of a key. That means every non-key field should be fully dependent on the key. Every non-key field should be fully dependent. When we say dependent we mean functional dependent functionally dependent. Every non-key field should be fully functionally dependent on the key element. Let us look at this example once again.

(Refer Slide Time: 51:54)



We have the subject id, we have the student id. Now in this example tell me is it true or not true? What are the non key fields? ST name and grade. What is the primary key? Subject id ST id together all right. So is it fully dependent? No, because ST name is a fact of ST id. That means there is a partial dependency this is known as partial or non-full dependency right. So that means ST name and grade this would have been in second normal form if ST name and grade would have been facts of sub id and ST id which it is no doubt. But sub ST name can be derived from a part of the primary key which means a non-full key. Non full dependence or a partial dependence this must be removed from the first normal form. Only then you get what is known as second normal form I just show the second normal form and then we stop here today.

(Refer Slide Time: 53:11)



So to make it in second normal form this is how it must be split all right. So the grade record is not in second normal form. Student name is a fact of student id only. This we have seen here. Because in this case you we have the student id, semester and course number. All 3 are basically in this particular example, you see student id semester course number that will be the key designate. Out of that grade depends on all of them. But student name depends on student id only because of this, we must remove the non-full dependency. This is possible by splitting it into 2 tables: grade record and student record. The grade record has got student id, semester, course number, which is a key and grade is dependent on all of them. And the student record student id and student name fine. So I stop here today and in our next class we continue from here will see how normalization can be carried out further.