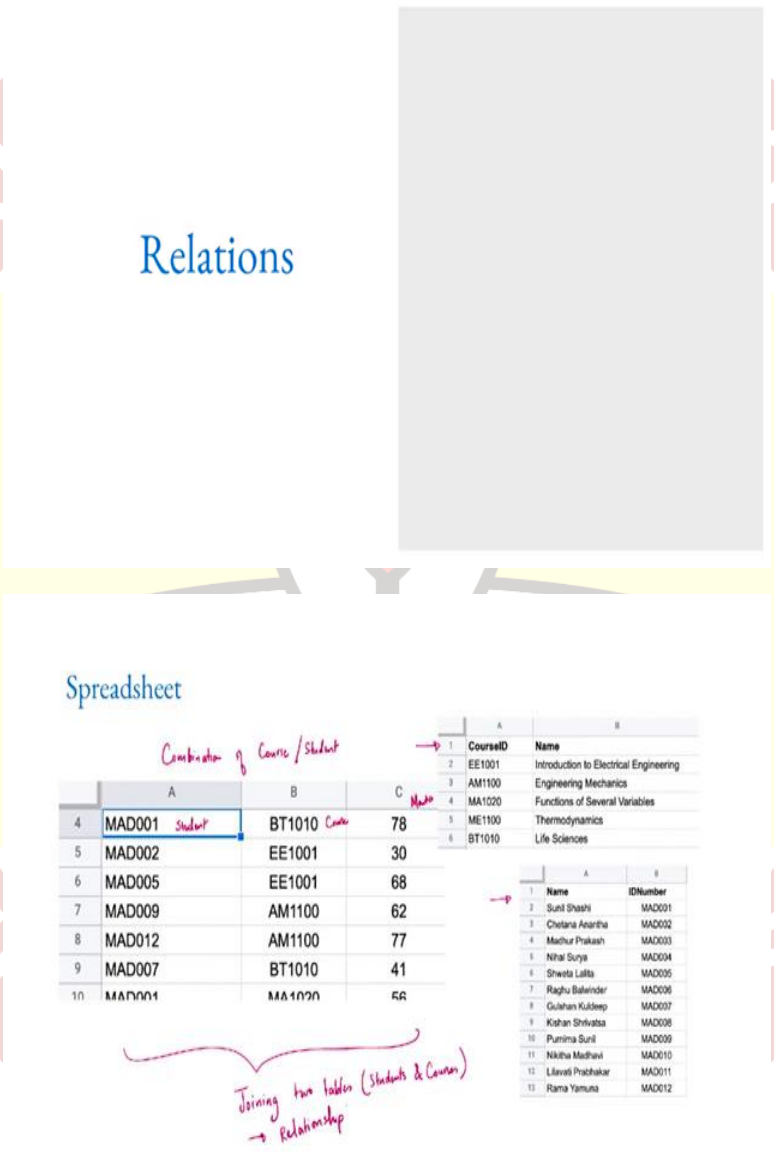
IIT Madras

ONLINE DEGREE

Modern Application Development - I
Professor Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Relations and ER Diagram

(Refer Slide Time: 00:18)



Hello, everyone, and welcome to this course on Modern Application Development. Okay, the main reason for using databases is, of course, to store data. But in addition to that, we want to be able to express relations between different kinds of data. And that is where the real power of a

database comes from the fact that you can connect different parts of data together and say that there is some relationship between different parts of them.

So, what exactly are these relationships, let us look at some examples to try and understand that better. Going back to our running example that we have been using a great book, we looked at it in the form of a spreadsheet. And the spreadsheet is primarily used because it helps us to display and enter tabular data. And what I mean by tabular data is data that can be expressed in the form of tables, basically, meaning that there are multiple columns, each column corresponds to some logical piece of information. And each row in the table corresponds to a separate entry.

So, for example, we had that the on the right hand side, you have the course information, where essentially, we would have the CourseID and the name of the course, the CourseID could be some unique identifier, EE1001, AM1100, and so on. These are examples of some courses that are taught at IIT Madras, but they could be anything, I mean it is not necessary that you have to follow a specific pattern.

Ideally, you want to have some kind of a pattern simply so that the human being to see it can recognize it easily. But as far as the computer is concerned all that matters is that it is unique, and that it is fairly easy to index. Indexing is something that we will be looking at later. And of course, the columns out here are the CourseID, and the name.

Now, similarly, we had another table which corresponded to the students. And over here we had the name and the IDNumber, the IDNumber in the case of a college or a course would usually be the roll number of the student. Now, when you are actually storing this in a database, you might find that there is a separate internal ID that is automatically assigned by the database itself, which is used in order to keep track of the number of entries and ensure that you have a unique identifier for each row and also to make it easy to sort of index and search through.

But as far as we are concerned, the logical identifier is the roll number, or the IDNumber as I have marked in this place. So, you have several names out here, each student has their corresponding IDNumber. And what we have on the left hand side, this is the, it is sort of a combination of the course and the student.

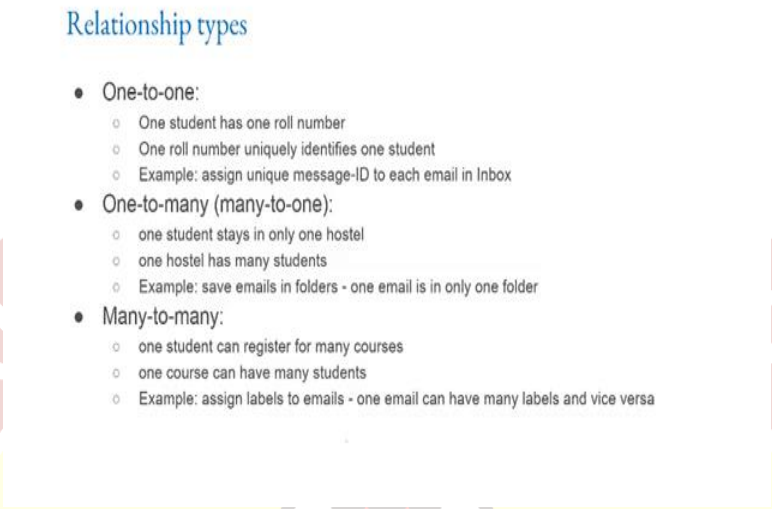
So effectively, what it is saying is that this first column A corresponds to the student ID. And the second column corresponds to the CourseID. And the third column could probably be something like the marks obtained in that course. It could be also just used in order to indicate that they have enrolled in the course. But it makes sense to sort of use the same column or writing in the same table to also store the marks that they finally obtained in the course.

So, what we have in this way is that this is so called joining two tables together. The students and the courses; this is not exactly the same as what we say is a join statement in the SQL or the Structured Query Language, over there the database itself takes care of sort of joining the two tables together in order to create one compound table on which you can perform certain queries. For here what we are doing us we are expressing a relationship between the two tables.

So, the relationship between the two tables in this case, basically say something like student with the ID MAD001, has registered for course, BT1010 and obtained 78 marks. So that is how you interpret this combined table that expresses the relationship. So please keep in mind that even though I am using the word joining, the word join in the context of SQL has a slightly more specific meaning. And that is something that you will be learning in detail, of course in the database management course.

But it is not to be confused with what they are trying to express over here. They are used for substantially the same thing because that ultimately a join is expressing some kind of a relationship between different sets of data. So what we can see over here is that even though we were using a spreadsheet, the spreadsheet is good at expressing tables or data in the form of tables. And by using one extra sheet in the spreadsheet, we can also express the relationship between different entities that were already expressed in other parts of the spreadsheet.

(Refer Slide Time: 05:32)



So, what are these kinds of relationships? Let us look at a couple of examples, there are; in general, a few different kinds that we can think of. The most basic in some ways is so called one-to-one relationship. And the simple way to think about this is, every student is assigned a roll number.

And once you are given a roll number that uniquely identifies the student, so there is a one-to-one relationship between the student and the role number, one student will have only one roll number assigned to them. And given a roll number, you can uniquely identify who was a student, who was assigned that number.

To take another example, let us say that you are using some kind of a desktop client Outlook or Thunderbird in order to read your emails. And when you download the emails, you will see them in an inbox. And what happens is internally though, it might not be visible to you to start with, if you look at the detailed header information corresponding to the emails, you will see that each email has some kind of a unique message ID associated with it.

Now, why is this message ID there because as far as the email client is concerned, it needs some way to uniquely refer to different emails. So, this unique ID might be assigned either by a client, or it might be assigned globally at the system level, by whatever was the email client that the sender used in order to send the email, so that is a one-to-one relationship. Every email has a

unique ID. And the unique ID obviously, by definition uniquely identifies the email address associated with a student and a roll number or a course and an ID number.

Now, that is not the only kind of relationship you could also have a one-to-many relationship. And when I say one-to-many, an example of that is any given student in IIT, for example, usually they are expected to stay in the hostels. And it is quite obvious that one student can only be in one hostel, you are assigned only one room. On the other hand, one hostel will have many students.

So, you have a one-to-many and a corresponding many-to-one relationship over there. So, every time there is a one-to-many relationship, there is also a converse many-to-one relationship associated with that. Over here the relationship is, one hostel has many students. And similarly, many students can belong to one hostel. So, both sides, you can express it in either direction. And it is completely equivalent in this case.

Similarly, with the other example that we were looking at with email, one common way of organizing emails is to store emails in folders. So, you might have one folder corresponding to work another folder corresponding to some courses that you are studying, and other folders are corresponding to your family members and another one corresponding to some games that you play. And all of those would be arranged in such a way that the only thing that you can guarantee is any one email is stored in only one folder.

And there again, you have a one-to-many relationship. One folder has many emails, and many emails can be in one folder. But any given email is only going to be in one folder. That is the constraints that we have. And the final type of relationship that we can think of over here is a so called many-to-many relationship. In many-to-many, an example of that is one student can register for many courses. When you are a student in college, you can typically register you are expected to register for multiple courses.

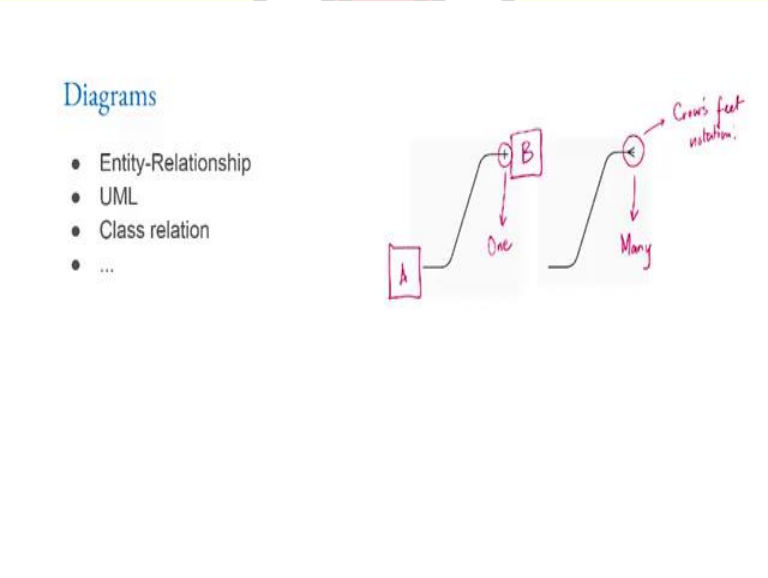
At the same time, each course will also have multiple students. There is never a course that is offered for only one student. So, you can have many students attending a single course. And you can also have one student taking multiple courses. So that is a many-to-many relationships. You could have a multiplicity on either side. Going back to the email example, rather than the folder

way, if you look at how Gmail organizers thinks, for example, they do not put things in folders, they use so called labels.

And I can assign many labels to a given email. I could assign work and I could also assign a specific project name, I could also perhaps assign it a label such as home, which says that this is something which I need to take care of when I get home, okay and one email can therefore have multiple labels, in the same way, each label will probably be associated with multiple emails that is how it becomes useful.

So, it is clear that these are different kinds of relationships that can exist between data. And the power of a database system comes in how it is able to capture such relationships. So, unless our database is able to compactly and efficiently capture these relationships and allows to express that it is going to be of limited use.

(Refer Slide Time: 10:46)



So, now that we know the different kinds of relationships, there are different ways that these can be represented. And some of them are graphical. And in particular, there is this concept of using certain kinds of diagrams in order to represent them, there is something called the entity relationship or ER diagram, and we will look at a couple of more examples of that moving forward.

But there are other methods as well, there is something called the Unified Modeling Language UML there are class relation diagrams, there are in fact even in so called ER diagrams that are there are different notations, that are used in order to express relationships. Now, of course, in a database management course, you will probably come across one technique, and use that systematically. But as I said, there are multiple different ways and in this particular course, we will just be looking, at least in the few examples that we do consider, we will look at one particular notation that is sometimes called the Crow's feet notation, as I will explain in a moment.

Now, keep in mind that like I said, this is not the only way of expressing relationships, it is just one compact and efficient way of expressing relationships which it is good for you to know. And it is always you know, even you are welcome to sort of look at and understand why the other notation fixes and how to sort of make use of them.

So, in an ER diagram, we usually have entities, okay, so there will be an entity of some Type A and it is related to some other entity of some Type B, as shown in this diagram. So, what have we done, we have got one entity of Type A and one entity of Type B, and they are connected by a link or a line.

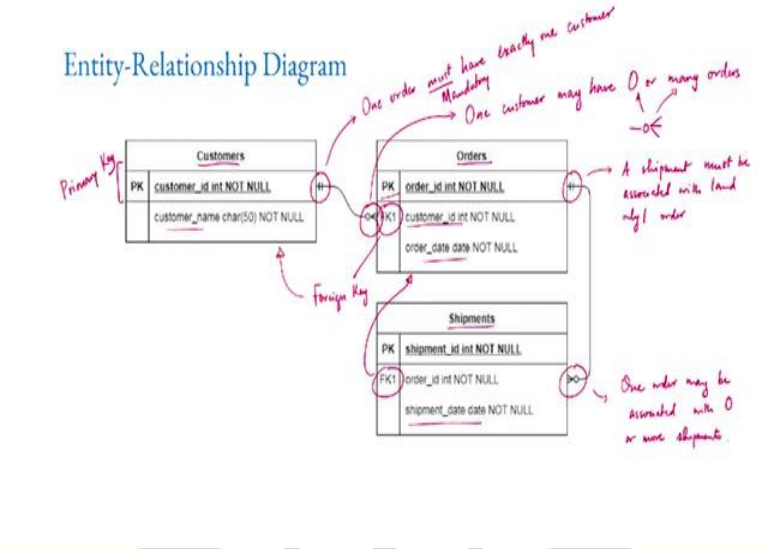
And what we have out here is on the line, we have certain kinds of notation markings. In this case, I have this marking, which basically indicates, one; there is a single line sort of line crossing the main line linking the two entities together? And that cross line indicates a one, what exactly do we mean by a one? We will get to that in a moment.

Another possible thing that we could have is this kind of notation, the 3 lines out there, which essentially indicates a many relationship. And this particular thing is why it basically looks like sort of the three doors of a word. So, it is sometimes called the Crow's feet notation. So, you might come across this term, Crow's feet notation when referring to an ER diagram. This is basically what it is referring to.

Now, this one or many, by itself is not sufficient, we would also like to provide some additional information out there. And the sort of main minimum piece of information that we need is to sort of say, when I say one, do I mean exactly one? Do I mean zero or one? Do I mean one over many? What exactly does that refers to. So, there is usually also one additional indicator that is

applied next to the symbol that we have to say whether it is exactly one, whether it is a mandatory one. That is to say you have to have exactly one. Or it is an optional one, meaning that either 0 or 1 is okay. So based on that you can have these notations and they can be used in order to express the relationship between entities. Let us take an example.

(Refer Slide Time: 14:30)



Now, this is sort of maybe what you might come up with if you were trying to bring in some kind of E-commerce application. So, I have customers who can place orders and those orders need to be shipped out to the customers. So, let us say I am trying to create the next Amazon clone. This is the kind of structure I am likely to come up with. Okay.

So, let us look at this and understand the relationships between these different kinds of entities. So, first things first, the notion of entities should be clear, right, entity basically refers to some kind of an object or some kind of piece of information that captures something related to my application. And that I need to make sure is represented properly, what are all of the details.

So, in the case of the customer entity, the main thing that I need to have is the customer_name, okay, I would most likely also have an address or maybe even multiple addresses. But right now I am leaving that out because especially if you have multiple addresses, then maybe that has to be the address has to be a separate entity in itself.

Now, if I say that, you know, a customer is only allowed to have a single address then I could probably make it a part of the same entity block.

But essentially, what is happening is this rectangle out here right, the big box is referring to a customer entity. And of course, the customer_name is what needs to be sold. So, we have some details about it, its character set of characters, not more than 50 of them, and it cannot be empty.

And you also have one more piece of information, which is the so called primary key, which is why it is referred to as a PK. In this case, the customer_id is a PK, a primary key is declared as an integer, it cannot be null. Now, why is that necessary? Do I really need to associate a numeric ID with each customer?

Maybe there are circumstances in which we do not. But in general, it is good practice to have this kind of a numeric integer ID associated with each entity, each table with each row of your table. For the simple reason that it guarantees that there is at least one particular entry over there that you can index on, okay, and it makes it sort of easy to identify a particular entry and pull out all the data corresponding to the table.

So, this is more a question of good practice and not necessarily compulsory thing. But in general, it is a good thing to have when you are constructing the data model. Now, the next thing is that I have orders. So, what can I say about an order? Each order would be placed by one customer. So, one thing I can be sure of when I think about it is to say that an order can have only one customer associated with it.

I cannot think of one order which needs to go to different where should I shift it. So once again, the order_id becomes the primary key that is like I said, it is sort of identity. And you can think about it every table in your database typically will have a primary key associated with it. But in the case of the order, there is some more information. And then of course, you will need to know what exactly is the item been ordered and so on.

But in this case, you know, in the diagram that I have shown here, I have not even got that information. All that I am doing is storing the order_date, which is of type date. But the other thing that I do is I store a customer_id. And what is the customer_id is basically is says it is a

foreign key. So, a foreign key in the context of a database essentially says that it refers to another table in the database.

So, customer_id in this case is a foreign key that refers to the customer's table. That is not specified right here. But you know, the fact that it is an empty or foreign key can be used inside the database to see that it actually refers to this table. Alright, so we have this much information, we know that customers just have customer_id associated with them. So, the customer table, in other words, does not say anything about orders.

But if I look at the Orders table, it has a foreign key linking back into customer's ID. So how do I interpret that? Let us look at the Crow's feet notation that we have here, okay. This part of it essentially says one order must have exactly one customer. In other words, if I have an entry in the Orders table, it must link back to exactly one of the customers. It is mandatory right and it is only one customer permitted.

On the other hand, I also have the other side of the relationship where I now have one customer may have 0 or many orders associated with them. So here are many meanings 0, 1 or many okay, so you could have any number, any non-negative number, associated over them. So circle indicates the 0. And this crossword indicates the many.

So, in other words, what I am saying is that it is not mandatory for a customer to have an order, I might have just registered on Amazon and not place any orders, that is perfectly fine, it just store my information over there. But if there is an order, then it must be associated with a customer. And conversely, the customer is allowed to have either 0, or 1 or 2 or many orders associated with them. So that customer ID can be present in multiple rows of this orders columns.

Now, what is an order, an order might be, let us say, for 10 books, and maybe for whatever reason, all the 10 books are not available at the same time. So, 2 of them get shipped on one day, 3 of them get shipped another day and the last 5 get shipped in a third consignment. So, because of that, I could also potentially think of shipments each associated with a shipment_id and a foreign key which is the order_id.

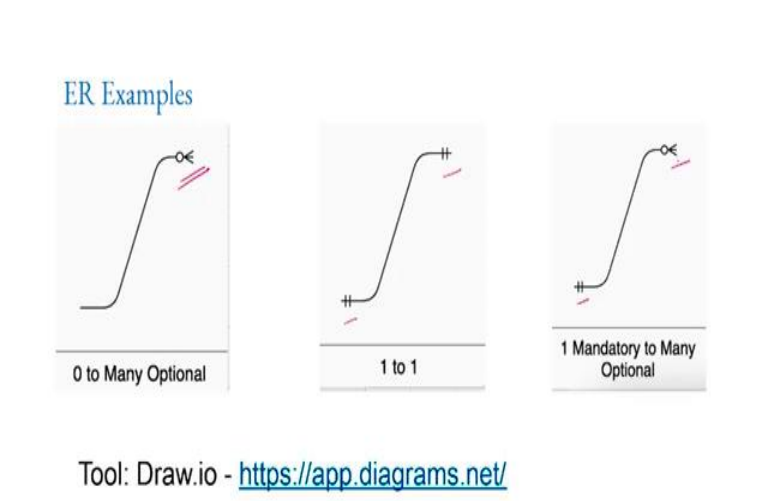
So, again, you have the same kind of relationship, any given shipment must be associated with one and only one order. On the other hand, one order may be associated with zero or more

shipments, zero shipments in the trivial case where nothing has shipped yet. Okay, so I have not, no, I have got the order from the customer, but I have not actually sent anything out yet. So, there are zero shipments associated with this order.

Or it could be one shipment, you know, I got all the books together at once, put them into one box and send them out. Or I could have multiple shipments, I break it up and send it out. So, these kinds of relationships can be captured with these diagrams. The good thing is, in a lot of ways, you know this, so this diagram is capturing a lot of information very compact, it has the full table structured that you need for customers for orders and for shipments.

It also captures the relationships between them, it is quite easy to take one look at it and sort of understand the relationships exactly. And in addition, there are also in some cases, some tools that will allow you to take this kind of a diagram and convert it into a template, a schema for a database, or even some kind of code that could allow you to structure what you want. Conversely, if you have written the code to start with, you can also generate diagrams from that insert into this.

(Refer Slide Time: 23:26)



But having said that, feel free to use whatever you are comfortable with. So, like I said, there are many different options out here, this would be a zero to many option. We already saw examples of that. Customer could have zero or many orders. One-to-One would be the example where you have for example, a student and the roll number.

Now, usually you might not even have a separate table corresponding to a one-to-one relationship. If it is only a student and the roll number, why would I create a separate table for the roll numbers, I might just make it part of the student. But so that is sort of a contrived example, but there might be other cases where I actually have explicit one-to-one relationship. I have two entities, each of which has their own store of information.

But there has to be a one-to-one relationship between two columns out there. Usually when you see such kind of one-to-one relationships, you need to sort of investigate further and see could they have been combined into a single table, but there could be reasons why you might want to split them into two and of course, you know the one mandatory too many optional this we already saw example.

Again, the customer shipment that kind, the point is all of these are easy to draw when you are using a tool like diagrams or even for that matter you know PowerPoint or anything else could potentially be useful in drawing. The important thing is to be able to look at it and understand the implications of the picture.