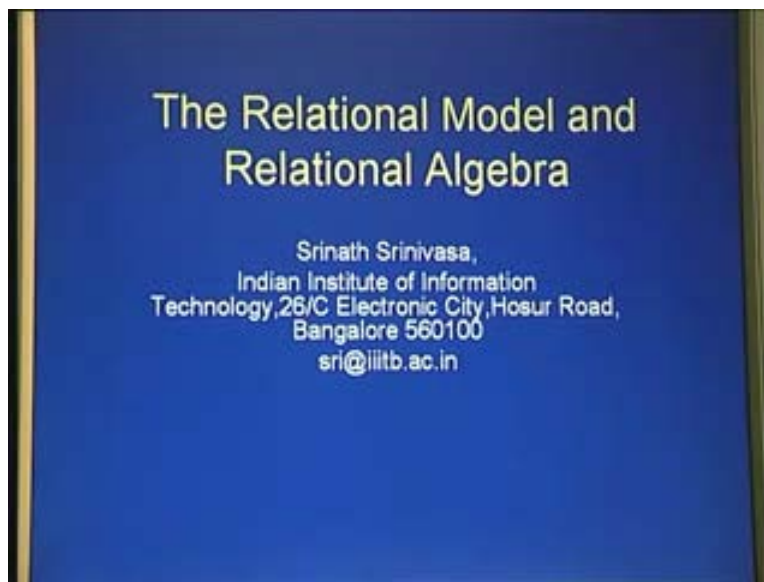


Database Management System
Dr. S. Srinath
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 3

Relational Model

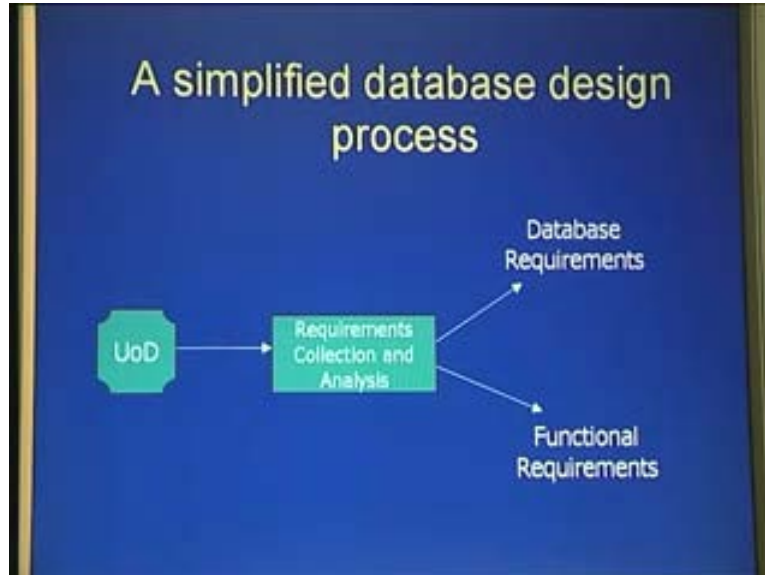
Hello everyone, we have been looking into the process of database design and let us continue with this in this session as well. As we saw in the previous sessions, a database design goes through several different phases and we have been mainly looking into the conceptual design of a database. A conceptual design essentially means a high-level design of the database or the database system which is mainly meant for targeting the end users that is trying to explain your database model to the end users. Today we are going to look at another model of data which is called the relational model.

(Refer Slide Time: 00:02:04)



And how do we place relational model with respect to the entity relationship model that we have been considering until now. In order to answer this question, let us revisit our typical database design process that we saw in one of the previous sessions.

(Refer Slide Time: 00:02:10)

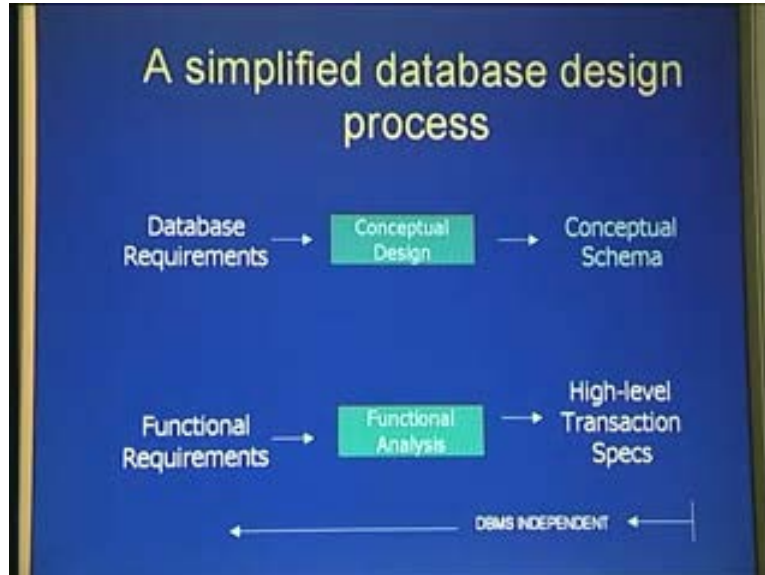


As we had seen that a database design process is contained within a universe of discourse that is a universe essentially is the information system context within which a database is designed whether it is a bank or whether it is railway reservation, whether it is even your mobile phones. In many of these different application context databases are usually embedded. So it is this application context that makes up the universe of discourse.

Now once we analyze the universe of discourse, we essentially get two kinds of requirements one was what was called as the database requirements as shown in the slide here and the other is what is called as the functional requirements. So the database requirements essentially meant, what are the data elements that make up the system and how are they interrelated and how should we make sense out of the data elements and functionality requirement or functional requirements or the application programming requirements which say what kinds of processes have to run on these databases and what are the semantics of these processes.

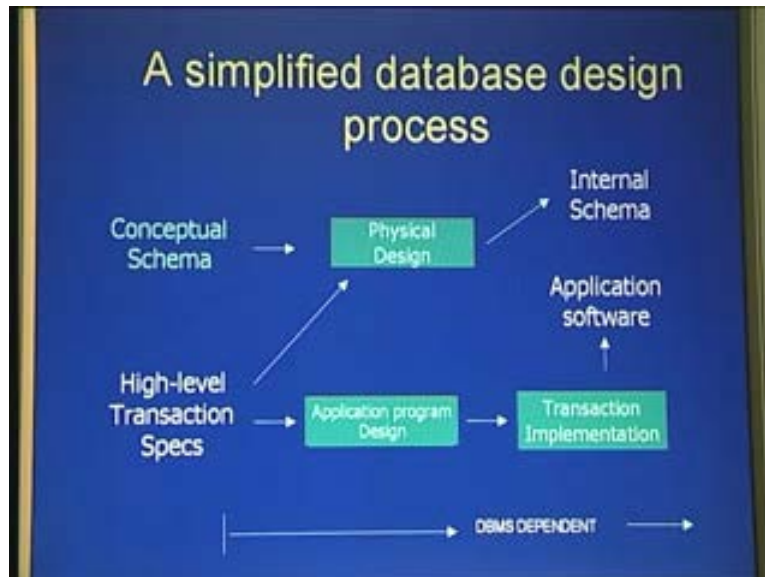
So these requirements in turn gave rise to two kinds of parallel processes. The database requirements gave rise to the conceptual design of the database and from the conceptual design came the conceptual schema and we saw in the previous class that the conceptual schema is usually built using the ER model that is an entity relationship diagram.

(Refer Slide Time: 00:03:24)



Now let us follow this upper stream that you see in the slide here a little bit further and see what happens to the conceptual schema.

(Refer Slide Time: 00:3:57)

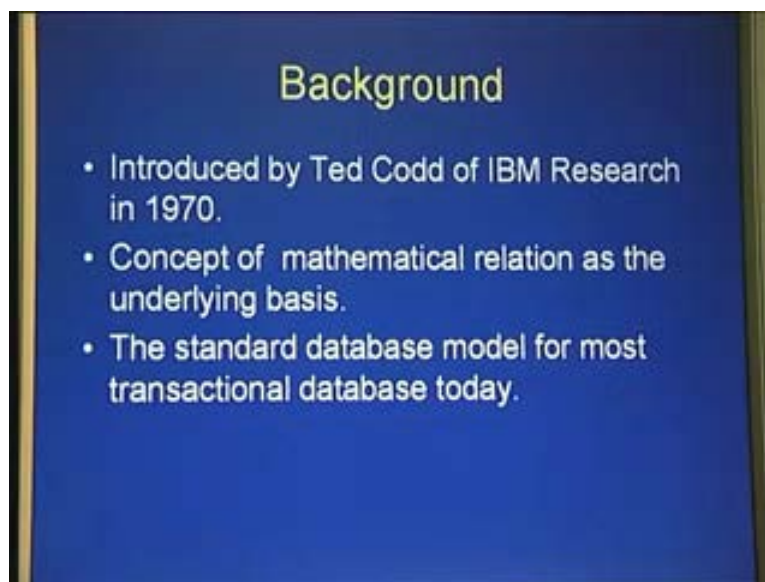


Now the conceptual schema which is typically meant for are essentially meant for communication with the end user is in turn going to give rise to the physical schema. What is a physical schema? The physical schema is essentially the schema that is actually build on the database system on the computer and as you might have imagined, while the conceptual schema is oriented towards human understanding that is communicating your schema or communicating your design with the end user, the physical schema is oriented

towards machine understanding or essentially efficiency in terms of storage and retrieval of data elements. So the physical schema is optimized towards quick updates, quick inserts, easy searches and so on.

So it is one of this physical schema or the building blocks of such a physical schema is what we are going to see today and the model that we are going to see today the relational data model is the most widely used data model in most databases today whether it is any kinds of application context, whether it is banks or railways or telephone exchanges or whatever. Any of these application context typically used relational model to store data as part of the internal schema structure.

(Refer Slide Time: 00:05:22)



So what is the background of the relational data model? The relational data model was introduced in the 1970's, the early 70's by Ted Codd from IBM research and in fact there is you can do a web search for Ted Codd today and many of his seminar papers that where the database or the relational model was proposed are actually available over the internet and you can actually have a look at them to see why it was so influential.

Before the relational model was proposed there where another models like hierarchical and network model and so on which were not very amenable to internal storage which are not very efficient in terms of storage and retrieval complexity and so on. And the relational model was extremely elegant in terms of storage and updates and retrieval searches and all these. The main concept behind the relational model is the notion of a mathematical relation. You might have studied in course on discrete mathematics and mathematical relation is just a mapping between two or more sets, so where each set constitutes a domain and a mapping between each of these domains forms a mathematical relation. In intuitive terms a mathematical relation is no different from what we understand as a relationship in normal English.

A relationship is similar, is essentially some kind of an association or some kind of linkage between two or more different data elements. An employee is associated with a department, an employee has a name associated with him, an employee has an id associated with him, an employee has a salary that's associated with him and so on and so forth. So it's this mathematical relation let us say the set of all employees versus a set of all salaries, you can establish a mathematical relation between these two sets. It is this set that forms the underlying basis for the relational model. So this is the standard database model for most transactional databases today.

So let us go step by step into the nuances of the relational model. So what exactly are the building blocks or the essential concepts that make up the relation? So what is a relation in turn? A relation intuitively represents a table as you can see in the slide here or it's also called a Flatfile of records. This slide shows here a small table which has three different columns. The first column is named as roll number, the second column is named as name and the last column is named as date of registration and there are several different rows and each row corresponds as you might have guessed by now each row corresponds to one particular record or one particular student in this case.

(Refer Slide Time: 00:07:29)

The slide is titled "Relational Model Concepts" in yellow text on a dark blue background. It contains three bullet points in white text:

- A relational database is a collection of relation.
- A relation resembles a "Table" or a "Flatfile" of records.
- Each row in such a table represents a collection of such related data values (an "instance" of the relation depicted by the table)

Below the text is a table with three columns: "Roll No.", "Name", and "Date of Registration". The table contains three rows of data:

Roll No.	Name	Date of Registration
2003-01	Adithya	12-08-2003
2003-02	Ananth Kumar	16-08-2003
2003-03	Barath Kumar	17-08-2003

Below the table is the caption: "Table 1: A example relation called STUDENT". At the bottom of the slide, there is another bullet point in white text:

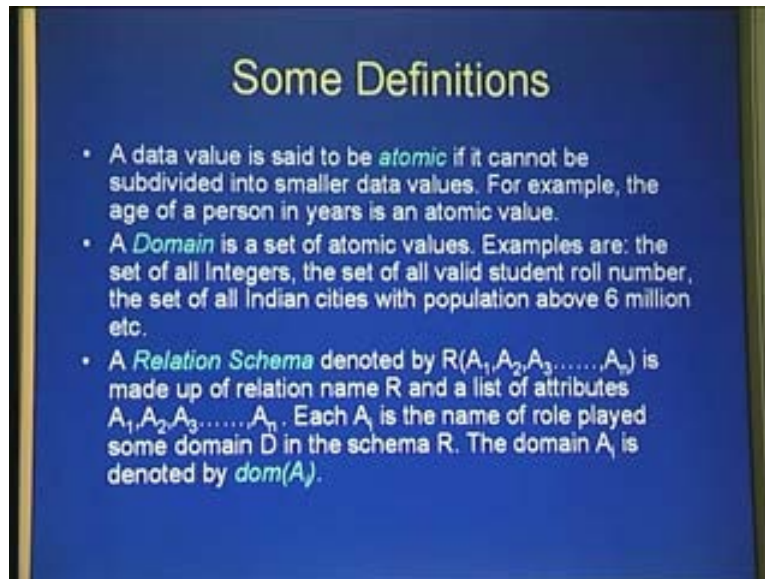
- Each row or a relation instance is called a *tuple* and each column is called *attribute* of the relation

So the first row has a student by name Adithya and his roll number is 2003-01 and there is a particular date of registration. Now this is one data element, one set of data elements that are interrelated, so one schematic data element. Now, this row is independent of the second row which talks about another student called Ananth Kumar and with his own roll number and with his own date of registration and this in turn is an independent data element that is independent of both the first row and the third row.

So each row in a table represents, a collection of such related data values are what is called as an instance of the relation. The relation in this case is or the schema in this case is a table comprising of three different columns roll number, name and registration. An

instance of this relation is one of these rows, one which says 2003-01 and name as Adithya and a date of registration as 12- 8- 2003. So each row in a relation is called a tuple and each column is called an attribute of the relation.

(Refer Slide Time: 00:09:32)



So let us dwell little bit deeper into the relational model and before we do that we need to define some certain crucial elements in the relational model. Now the first definition we are going to comeback to it again, the first definition in the slide talks about the notion of an atomic data type. Now have a look at the slide once again, a data type is said to be atomic if it cannot be subdivided into further values.

Remember in one of the previous sessions we talked about attributes in an ER schema being either a simple attribute or a composite attribute. A composite attribute is a non atomic attribute that is for example name if the name of a person in turn comprises of other attributes like first name, middle name, last name, title, initials and so on. This is not an atomic attribute.

On the other hand the age of a person is an atomic attribute because you cannot sub divide this attribute into further sub attributes. Now this is important because a relation is defined only over atomic attributes as we will see in the next slide and the next definition that we are going to consider is the notion of a domain. We have already seen the notion of a domain in the ER model and even here the notion of a domain is no different. A domain is basically a set of atomic values which defines the space within which an attribute might obtain a value for itself.

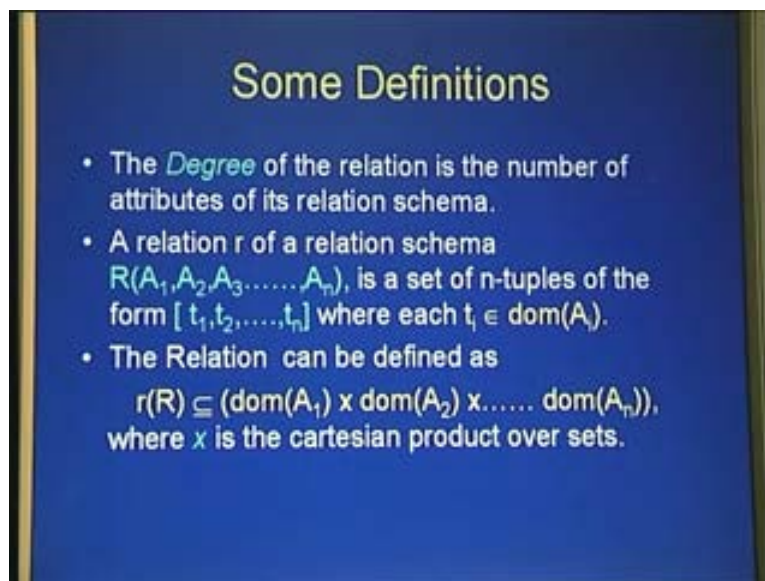
For example we had seen the examples of the age of an employee which may not be lesser than 18 years and greater than 65 years. So the age of an employee is a set of all numbers maybe even fractions between 18 and 65. So this constitutes the domain for this attribute called age. Similarly there are domains for names and dates and so on and so far.

This slide actually shows some examples like the set of all integers, the set of all valid student roll numbers, the set of all Indian cities with population above 6 million or whatever. Anything that defines a space of possible values is called a domain.

A relation schema or a relational schema, this is a crucial definition. The relational schema is as you can see in the slide has a very specific notation. A relational schema is defined by the name of the relation. Here it is shown as R and a list of attributes, here its shown as $A_1 A_2$ extra to until A_n . So this is the name of the relation R, in the previous example the name of the relation was called student record and the attributes are roll number, name and date of registration. So each A_i shown in this definition, here is the name of certain some attribute or as shown in the slide here is the name of the role played by some domain which is an other way of putting the same thing, name of an attribute or the name of a role played by a particular domain.

Now for example in the previous case their roll number, the domain of roll numbers was the set of all possible valid student roll numbers. Now what is the role that this domain plays in this relation is the attribute which is the part of the relational schema. So few more definitions. Just like we had that degree of relationship in the ER model, we define the degree of a relation in a pretty analogies fashion, it is simply the number of attributes in the relation schema.

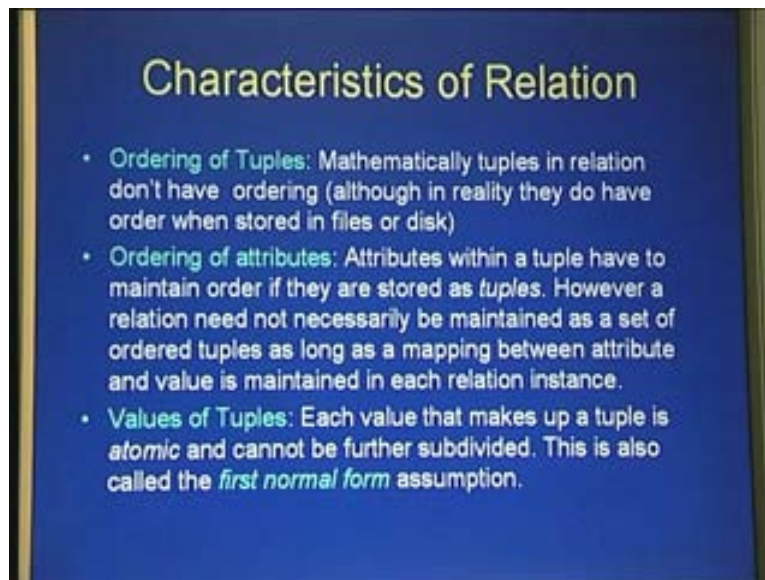
(Refer Slide Time: 00:13:08)



For example in the previous schema that we took, the degree of the relation was three, it had three attributes roll number, name and date of registration. So a relation R or what is even called a set of instances of relational schema, a relation this is different from a relational schema. The relation of a relational schema is a set of tuples which belong to the schema or which conforms to the schema along with the schema itself. So a set of tuples of the form $t_1 t_2$ extra t_m were each tuple is of the form of n different attributes or n different values one for each of the attribute.

So again in the example that we saw earlier, it had three different tuples. There were three different student records with three different roll numbers. So the entire set that is the schema plus the set of all tuples is called a relation. So a relation as you can say, it can also be defined as a subset of a cross product of all these domains. So the cross product of the set of all roll numbers times the set of all student names times the set of all registration dates that are possible. Now a subset of this cross product is what is going to form a relation. Now what are some of the characteristics of a relation in the relational model? Ordering of tuples, that is one of the first issues that we are going to look at. This slide shows here three different characteristics of tuples.

(Refer Slide Time: 00:15:02)



Ordering of tuples: Mathematically the relational model does not have any ordering that is specified over the tuples. Now it does not matter as far as the mathematical model of the relational schema is concerned whether the set of all tuples are ordered roll number wise or name wise or whatever. It is just a set of tuples that are in conformance with the relational schema. So but in reality of course they do have some kind of an order which is specifically the order in which they are stored on the disk whether they are stored in sorted order or not, it doesn't matter but they have some kind of an order in reality.

Ordering of attributes: Note that a tuple, you might have studied in course on discrete mathematics a list is something where the order is important. A list is different from a set. So tuple is basically a list of n different elements which means that the order of these n different elements is actually important. So if the relational schema says that my relational scheme is called student record and I have these attributes roll number, name and date of registration and if I have a tuple having three different values the first value corresponds to roll number, the second value corresponds to name and the third value corresponds to the date of registration.

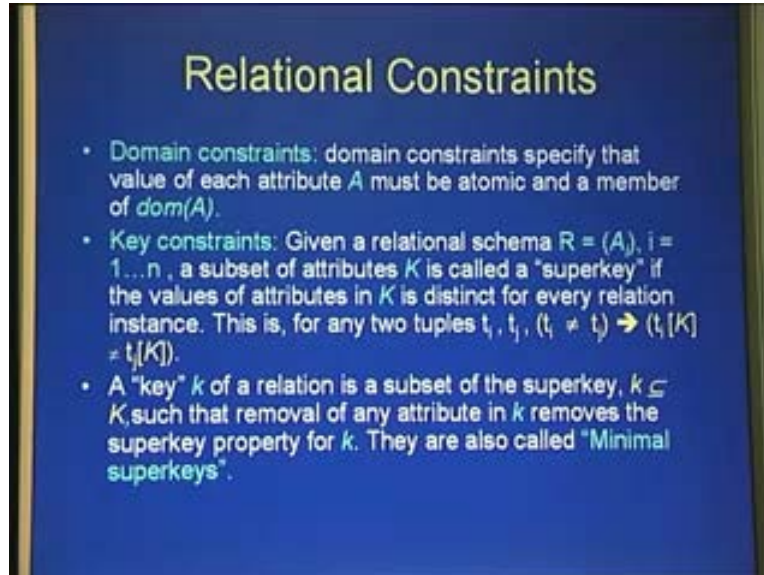
So the ordering is important but that is mathematically speaking. In fact, in reality though we can do away with ordering within a tuple as well. I can as well dereference a particular attributes by its name, I can as well say what is the roll number value of this relation, what is the name value of this relation, what is the date of registration value of this relation and so on. And values of tuples which is the third characteristic that we are going to look at today. So like I mentioned before each tuple or each value that makes up a tuple is assumed to be atomic in nature and this is what is called as a first normal form assumption.

In fact we are going to see in one of later sessions that the first normal form is just the first step in a series of different normal forms in which the database can be optimized for enhanced maintenance of the data. What do we mean by enhance maintenance? Easy addition of data elements, easy searching for data elements, easy updation of data elements and so on. And this is primarily the reason why it is stipulated that or why it is required that each data value that makes up a relation are to be atomic in nature and atomic as we had seen earlier is something that's not composite that is some value that cannot be subdivided into further semantic values. So just like entity relationship model, a relational model also are specified by certain kinds of constraints on the data model.

One of the first constraints that we are going to look at is pretty obvious which is what is called as the domain constraints. What is the domain constraints say? Each value of an attribute within a relation has to have a value which lies within the domain which is pretty obvious. So the roll number of a particular student has to be a valid roll number that is it has to belong to the set of all possible roll numbers If the set of all possible roll numbers range from 1 to 150, I obviously cannot have a roll number which is 200 or I obviously cannot have a roll number which says A B C and so on.

So the domain constraint specification basically states that each data value that makes up a relation has to be or has to belong to the domain in which it is the domain of the attribute in which it utilizes.

(Refer Slide Time: 00:18:16)



The second constraint in the relational model is what is called as the key constraint and this is quite similar to the key constraint that you saw in the ER model as well. In any relation there could be a subset of attributes that have a property that for every tuple in which those attributes appear, they have a unique value for those tuples. Such kinds of attributes are called super keys. Now what is the use of a superkey? Obviously to be able to uniquely identify every tuple in a relation.

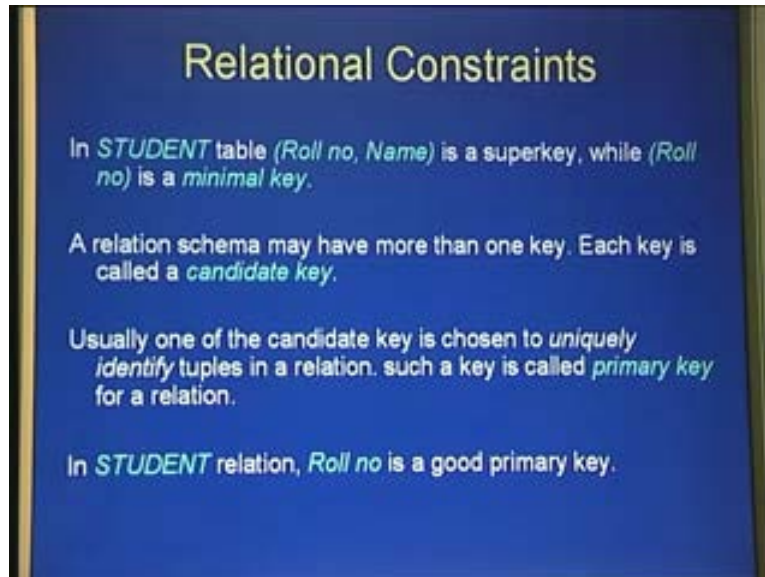
In the previous example where we took that a relation having three different attributes roll number, name and date of joining you can see that roll number forms a superkey because roll number is an attribute whose value is unique for every tuple in the relation. However you can also combine roll numbers along with the names and you can see that it's always going to be unique. If the roll number is unique roll number plus name is obviously going to be unique. So roll number plus name or in fact the entire record can be called the super key for each record which brings us to another property of the relational model.

In the relational model each tuple is distinct that means in the worst case the entire tuple is the superkey. The relation does not allow multiple tuples having the same value in the sense that it is a set of tuples and not a multi set of tuples or what is typically called a bag of tuples. So each tuple are to be different in value from the other. So, a key is a set of tuples in which is defined in a similar fashion as in the ER model a key of a relation is a subset of the superkey such that no subset of a key is a key in itself.

If you remember this was more or less very similar to the way we defined a key in the ER model itself. So roll number in the case of the student record is a key and if you take away the roll number, you cannot identify or you cannot distinguish one tuple from the other.

So however roll number plus name is not a key in itself. Why? Because you can still take out the name attribute from the key and you can still uniquely identify each tuple using just the roll number because the roll number is sufficient to identify each tuple uniquely in the relation. So there are also called minimal superkeys.

(Refer Slide Time: 00:22:10)



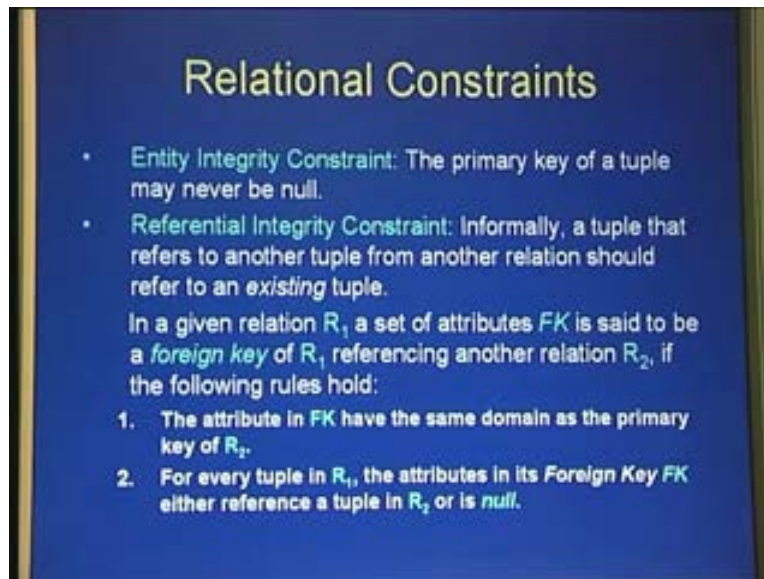
So this slide shows the talks about what we saw just now that in the student table roll number and name is a super key. However it is not a key or it's not a minimal super key because you can still take away name and still be left out with the key that is you can still identify each tuple uniquely in the relation or in the table. So, just roll number is a minimal or a minimal super key. Now, just like we had seen in the example earlier that a house can have more that one keys, the key for the front door and key for the back door. Any relational scheme or any relation can have more than one key or so on.

Take an employee record and usually employees are given employee identification number and they also have a pan number which is given by the government. Now using either of these two, you can identify an employee uniquely because each of them are unique for an employee. So each such key or each such set of subset of attributes which can uniquely identify tuples in a relation is called a candidate key. So either the employee number or the pan number is a candidate key in itself but usually one of those candidate keys are used for identifying tuples in a relation.

In a company contacts its usually the employee number, we do not usually identify people with that pan number when you are talking about their performance records or salary statements or anything of that sort, we usually talk about their employee number. Now whichever candidate key is used for the purpose of retrieval in quires and insertions and so on is called the primary key of a relation. So in the student relation that we saw earlier usually it's the roll number is what we use for students, so the roll number is a good primary key.

Now just like their entity constraints as part of the ER model, there are certain constraints that make up the relational model as well. The first constraint is what is called as the entity integrity constraint. So what is the entity integrity constraint? It essentially says that whichever entity the, now look at what we mean by an entity here. It's a slight change of nomenclature what we mean by an entity, here is a tuple in the relation.

(Refer Slide Time: 00:24:14)



So the primary key of a tuple can never be null obviously because we won't be able to identify each tuple uniquely. The second integrity constraint that's important in the relational model is what is called as a referential integrity constraint. Now what is a referential integrity constraint mean? Sometimes some set of attributes in a relation may point to certain other tuples in another relation. We had seen such an example in one of the previous sessions as well, when we said that a department is headed by an employee who is the manager or the head of the department. So that the headed by is usually contains an employee id of the person who is going to head this department.

So here what we are and ensure that we are going to get here is what is called as the referential integrity. Here there is a reference from the department entity to the employee entity. Now this referential integrity constraint basically says that whenever I make a reference from any tuple to any other tuple, it should make a reference to an existing tuple. That means I can appoint somebody, I can appoint some employee with some employee id as manager of the department as long as or only as long as such an employee already exist in the database. So such a reference is what is called as a foreign key.

So in any given relation set of attributes is set to be a foreign key if the following rules hold. The first rule basically says that the attribute in foreign key has to be the same as the primary key of the other relation, this is fairly obvious. If I am going to say that the department is headed by an employee and I write an employee id here but use employee name as the name of the person who heads, it obviously is not a foreign key. It has to be

either both the primary key in the employee record and the reference in the department record has to be names or they both have to be employee id's in some sense. And for every tuple in the referencing attribute or in the referencing relation like department, the attributes in its foreign key refer to existing tuples that is each department should have a manager who exists, who already exists in the database or there should be null that means there should not have a manager at all. So either of these two should hold for the referential integrity constraint to hold.

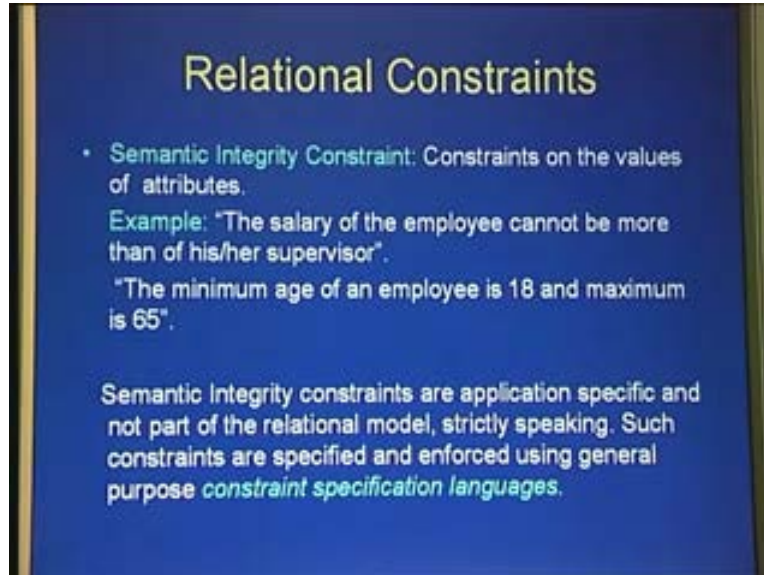
(Refer Slide Time: 00:27:36)



So referential integrity is usually depicted in a diagrammatic fashion as shown in this slide here. This slide shows two different schemes or relational schemes, one called the employee schema and the second called the department schema. So the employee schema has employee id, name, works_in and reports_to. So employee id would be the primary key here and the works_in actually is a foreign key which refers to the department id and reports_to is another foreign key which refers to another employee id within the same relation.

So note that foreign keys can be from a relationship to itself but the same referential integrity constraints hold. That is if an employee A is reporting to employee B, employee B should already exist in the database by the time employee A is being added to the database.

(Refer Slide Time: 00:28:35)



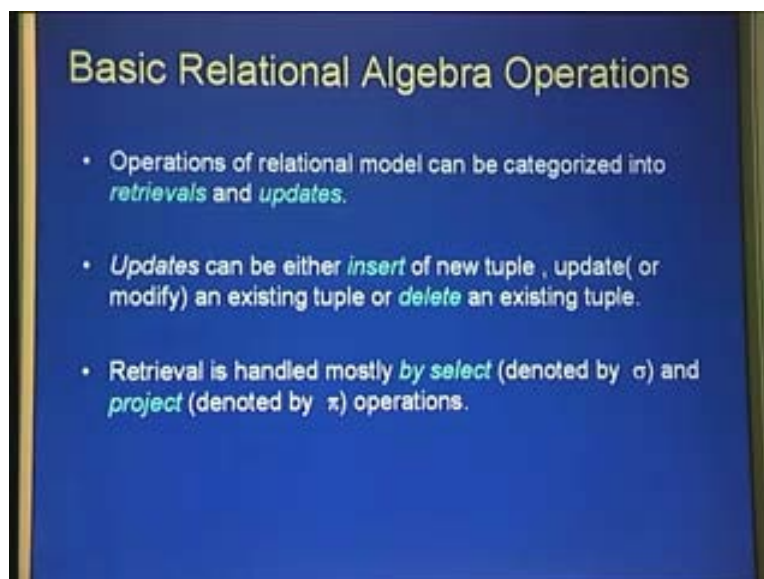
Relational Constraints

- **Semantic Integrity Constraint:** Constraints on the values of attributes.
Example: "The salary of the employee cannot be more than of his/her supervisor".
"The minimum age of an employee is 18 and maximum is 65".

Semantic Integrity constraints are application specific and not part of the relational model, strictly speaking. Such constraints are specified and enforced using general purpose *constraint specification languages*.

And the last kind of constraint over a relational model is what is called as the semantic integrity constraint. An semantic integrity constraints is usually more of an application specific constraints something like the age of an employee cannot be less than 18 years and greater than 65 years. So this is not per say part of the relational model but usually this is important to be implemented within a database context and automatically checked and verified that these integrity constraints are maintained. Now let us quickly look at what are some of the basic relational algebra operations that make up the relational model. Now essentially the operations of relational algebra can be categorized into one of two different kinds of operations namely retrieval of data or updates to the database.

(Refer Slide Time: 00:29:07)



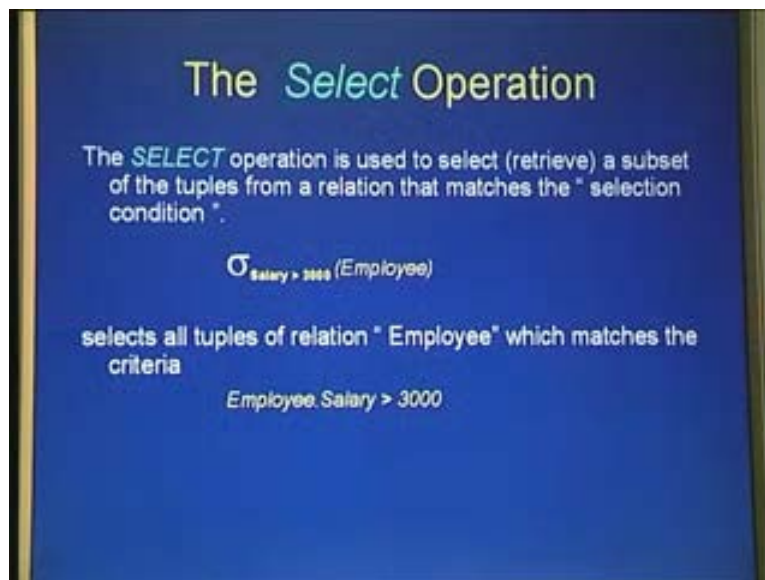
Basic Relational Algebra Operations

- Operations of relational model can be categorized into *retrievals* and *updates*.
- *Updates* can be either *insert* of new tuple , update(or modify) an existing tuple or *delete* an existing tuple.
- Retrieval is handled mostly by *select* (denoted by σ) and *project* (denoted by π) operations.

Now updates to the database are usually handled by what are called as insert and delete operations. We should not be looking into insert and delete operations in any detail in this session mainly because they don't have many, very many properties that we can explore at this moment. So right now we will be looking mainly at the retrieval operations and retrieval operations are handled by two basic operations called select. Select is denoted by a sigma as shown in the slide here and the project operation which is denoted by a pi which is also shown in the slide here.

So let us go to the select operation. The select operation is a very simple operation that is used to select a set of tuples from an existing relation. So remember a relation is basically a set of different data tuples along with the schema. Now given this set of tuples and schema, we can use the select operation to select a subset of those tuples. Now this slide here shows an example which says select salary greater than 3000 from employee. So as it shows here sigma salary greater than 3000 as a subscript and employee has the parameter this operation.

(Refer Slide Time: 00:30:11)



So as you can see there is a operation here, there is a condition and there is a domain or there is a relation over which the operation is going to be performed. So this is going to select the set of all records from the employee relation where the value of the attributes salary is greater than 3000.

(Refer Slide Time: 00:31:15)

The general syntax of **SELECT** is as follows:

$$\sigma_{\langle \text{Condition} \rangle} (\text{Relation})$$

where condition is of the form :

Condition \rightarrow (Expression) logicaloperator condition | Expression
Expression \rightarrow attributename operator constant | attributename operator attributename
Logicaloperator \rightarrow AND | OR
Operator \rightarrow < | > | = | \neq | \leq | \geq | ...

So the general form of select is shown in this slide, its simply as like this select condition relation. So where condition is a conditional expression, I have written a slightly formal grammar of how a condition looks like. Essentially condition is a logical expression over attributes names, something like select salary greater than 3000 and gender equal to male from employee. So which basically says give me all male employees in this relation whose salary is greater than 3000 and so on.

(Refer Slide Time: 00:31:48)

Properties of **SELECT operation**

- The **SELECT** operation is *unary*. It operates on only one relation.
- Selection conditions are applied to each tuple *individually* in the relation. Hence the condition cannot be span more than one tuple.
- The degree of the relation resulting from $\sigma_c(R)$ is the same as that of R .
- $|\sigma_c(R)| \leq |R|$
- The **SELECT** operation is commutative:
$$\sigma_{c_1}(\sigma_{c_2}(R)) \Leftrightarrow \sigma_{c_2}(\sigma_{c_1}(R))$$

Hence, **SELECT** operation can be cascaded in any fashion.

Now what is some of the properties of the select operation? The first property which you might have noticed here is that the select operation is unary in nature. What is a unary

operator? A unary operator is something which operates on just one operand. The select operator operates on just one relation even if my database has many different relations, the select operator operates on just one relation and we have to somehow make sure that when we are giving the select operator we have just one relation as the argument of this select relation. And, each selection criteria the condition basically that's specified is applied to each tuple separately.

The condition that we specified here was select salary greater than 3000 from employee. Now it's going to apply this condition separately to each tuple. Basically it also means again that each tuple in a relation is independent of the other and the degree of the relation that emerges out of a select operation. Note that the output of a select operation is a relation in itself because it has just taken a subset of the tuples from the given relation and return them along with the schema.

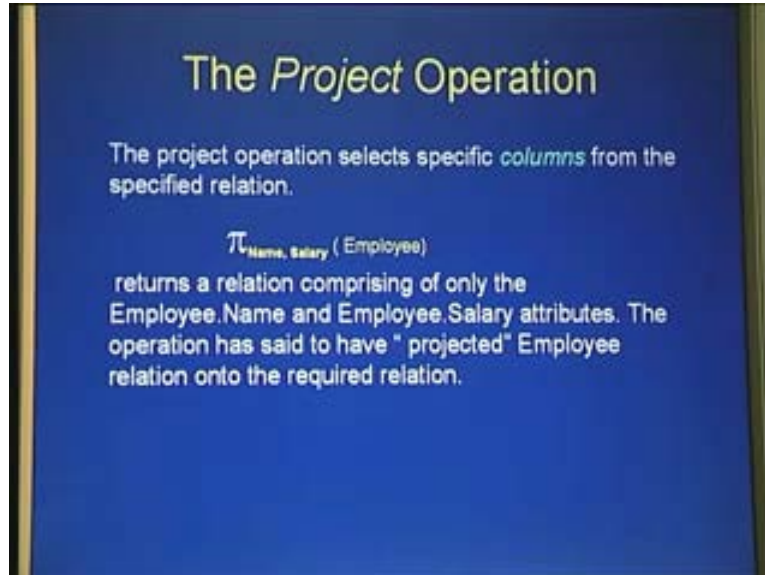
So the input to the select operator is a relation and the output is also a relation and the degree of the output relation is the same as the degree of the input relation. That is if the employee table in this example here had 4 different attributes, the output of the select operator also has 4 different attributes and in the same order as well. However the number of tuples returned by a select operator is bounded by the number tuples that already exist in the relation.

That is this slide shows this as in a very compact fashion, the cardinality of the select output relation is less than or equal to the cardinality of the select input relation and the last properties shows that select is commutative which is again quite interesting and important. This slide shows here that if I select based on condition c_1 , an output of a select based on condition c_2 for R, I can as well replace c_1 by c_2 and c_2 by c_1 and it doesn't matter. So you can verify that for yourself that the select operator is commutative whether it doesn't matter in which order I am going to apply the conditions. The second operator that we are going to be looking at is what is called as a project operator.

Now select operator if you have absorbed carefully is going to return entire tuples, it is not going to modify the schema of the relation which is given as an input. For example if the employee relation is given as an input and the employee relation has four attributes in some particular order, the same set of attributes in the same order is what is going to be returned by the select operator.

On the other hand what if we can select over columns rather than select over rows and return a different relation with possibly a different scheme. In order to do that we are going to use the projector operator.

(Refer Slide Time: 00:34:33)



The *Project Operation*

The project operation selects specific *columns* from the specified relation.

$\pi_{\text{Name, Salary}} (\text{Employee})$

returns a relation comprising of only the Employee.Name and Employee.Salary attributes. The operation has said to have "projected" Employee relation onto the required relation.

The project operator as shown in the slide here is quiet similar to the select operator in the sense that it has first the command called project which is denoted by pi and a list of attributes, here it shows name, salary from employee. So the output of this relation is again another relation. However with a different structure from that of employee that means it is going to return just the name and salary attributes or the name and salary columns of the employee table as part of its output. So we can also say that it has projected the employee relation on to the selected set of list of operation.

So the general forms of the project operation is simply of the form project, attribute list and relation. So I can just give a list of attributes and the project operator returns relation in the same order that is being presented in the attribute list. Hence if I gave salary before name, it would also return salary before name in the tuple that is returned as part of the project operation. So what are the properties of the project operation? The first property is that which basically is one of the main properties of the relational model that is a relation may not have many duplicates.

(Refer Slide Time: 00:36:13)

The general form of *PROJECT* is as follows :

$$\pi_{\langle \text{attributelist} \rangle}(\text{Relation})$$

where *attributelist* is a comma-separated list of attributes belonging to the specified relation.

The result of the *PROJECT* is in the *same order* as the specified list of attributes.

(Refer Slide Time: 00:36:53)

Properties of *Project* operation

- The *PROJECT* operation removes duplicate in its results.
- The number of tuples returned by *PROJECT* is less than or equal to number of tuples in the specified relation.
- When attribute list of *PROJECT* includes the superkey then the number of tuples returned is equal to the number of tuples in the specified relation.
- *PROJECT* is not commutative.

$\pi_{I_1} \pi_{I_2}(R) = \pi_{I_1}(R)$ if I_1 is a substring of I_2 . Else the operation is an incorrect expression.

That means when I am projecting let us say I am just projecting name and salary attributes from the employee relation, it may so happen that there maybe two employees with the same name and the same salary but with of course different employee identification or employee numbers. So the project operator actually would start forming duplicates in the relation that emerges out of this however duplicates are not alone. So the project operations remove duplicates from its results when it returns results. And the number of tuples returned by project is less than or equal to the number of tuples in the specified relation. How can you verify that the number of tuples is less than or equal to, why not equal to because I am just asking for certain columns in the relation.

The answer to this lies in the first point that is there are no duplicates. So when I selected just the name and salary attributes from the employee table it may so happen that there may be certain duplicates that exist in the output relation. Now because the duplicates are removed, the number of tuples in the output relation is actually less than the number of tuples that forms the database, that was in the relation in the first place. So when the attribute list of project includes the superkey then the number of tuples is same as the number of tuples as in the database which again follow from the first two points and the last point is again important, the project is not commutative.

So have a look at the slide once more, it gives an example project l_1 and project l_2 out of R . If this were the case then this would become, this would be equivalent to just saying project l_1 from R if and only if l_1 is a substring of l_2 . So for example if l_1 is just name and l_2 is name, salary then I could just say project name from R instead of saying project name, salary from R . On the other hand if I try to do it the other way around then it becomes an incorrect expression. So I can't project name, salary after projecting just name from the relation

Composition: Now this is another property of the relational model. If you notice again carefully, we have mentioned this point in passing when we looked at both select and project but this is going to be very important now. The input for the relational operator select as well as project is a relation and the output is also a relation. The select operators return a relation containing only a subset of the tuples of the input relation. Similarly the project operator returns a relation which contains only a subset of the attributes of the input relation however both select and project returns a relation.

Now this brings us to a very important property that a relation can be dynamically defined, it need not actually statically exist in the database. That means to say that I can put a project operator to the output of a select operator and it would still make sense because the output is a relation and the project operator also expects a relation. So this is what is called as composability of the relational operators. So relational operators can be composed as shown in the slide here that which shows project name, salary as the outermost operator and there is an innermost operator called select.

(Refer Slide Time: 00:41:01)

Composition and Assignment

- Since the input and output of the relational operator is a single relation, they can be composed in any fashion with the output of one operation being the input of the other.

$$\pi_{\text{Name, Salary}} (\sigma_{\text{Salary} > 3000} (\text{Employee}))$$

returns Name and Salary field of all records of Employee, where salary is greater than 3000.

- Results of a query can also be assigned to a name to form a relation by that name.

$$\text{SalaryStatement} \leftarrow \pi_{\text{Name, Salary}} (\sigma_{\text{Salary} > 3000} (\text{Employee}))$$

create a new relation SalaryStatement out of the specified query.

So if you can look at the slide again, there is a project operator here. The project operator is operating obviously on a relation. Now what is this relation? The relation doesn't exist when the project operator is performed, in fact it exist only when the select operator is performed. That is once the select operator finishes, it basically brings out a relation which is the set of all tuples where the salary field is greater than 3000 from the employee record or the employee relation and this set of tuples which is dynamically created forms the input for the project operator which is going to be another relation.

Now because this is going to be another relation, it can be very well assigned to a relation called salary statements for example here. So which says salary statement equal to project name, salary from where select salary greater than 3000 from the set of all employees. Now before we conclude today, we will just look at one major question which I am sure you would be asking yourself. Now the question is both project and select operators expect only one relation. Now does it mean to say that I cannot ask any queries that span more than one relation, should I ask every query over just the employee record, over just the department record I mean the department relation or employee relation or so on. Can I not ask any question that spans employee and department relations together and so on.

(Refer Slide Time: 00:42:56)

Cartesian Join (x)

PROJECT and SELECT operators expect a single relation.
When PROJECT and SELECT need to be run on a multiple relation, a single relation can be generated using the cartesian join (x) operator.

TABLE : Student			TABLE : Lab		
Roll No	Name	Lab	Name	Faculty	Dept
2003 - 121	Arindam	Speech Lab	Speech Lab	Prof. Fernandes	EE
2003 - 122	Ravi	Open system Lab	Open system Lab	Prof. Baradwaj	CSE
2003 - 123	Hema	Distributed computing lab	Distributed computing lab	Prof. Ramanan	CSE
2003 - 124	Vasu	Open system lab			

Consider the Relational Query : $\sigma_{\text{Student.Lab} = \text{Lab.Name}}$ (Student x Lab)

So the answer to this is the Cartesian join obviously. So essentially what we have to do here is because both select and project operators require just one relation, we have to somehow ensure that even if you have more than one relation they all fall back or they all combine to form just one relation. So we are going to look at one such very rudimentary operator to make just one relation which is namely the Cartesian join. In fact there are other much more efficient ways of combining two or more relations which we are going to explore in the next session.

So the Cartesian join as you might have imagined is very similar to the Cartesian product between two sets. What is a Cartesian product between two sets? You just take each element of one set and combine it with each other elements of the other set. So in the case of relations, you just take each tuple of one relation and combine it with every possible tuple of the other relation. So the slide here shows such an example. There are two tables here, one table is called student and the other table is called lab. Now student table has three different attributes roll number, name and lab. And lab table itself has three different attributes that is name faculty and department. That is the name of the lab, the faculty heading the lab and the department in which the lab belongs to.

Now consider the relational query here shown below the tables, select student . lab equal to lab . name. So that means to say that lab attribute from the student table equal to name attribute from the lab table from a Cartesian product of student and lab that is combine student and lab to get it. So what is the output of this relation or how is this relation or how is this query evaluated? Let us first straight away compute the Cartesian join or the Cartesian product of the two relations student and lab. Now what I have done here is I have take the student relation that is the student relation had 4 different students. So for each student I have combined it with each possible lab tuple to form one big relation.

(Refer Slide Time: 00:45:16)

Student Roll No	Student Name	Student Lab	Lab Name	Lab Faculty	Lab Dept
2003 - 120	Arindam	Speech Lab	Speech Lab	Prof.Fernandes	EE
2003 - 120	Arindam	Speech Lab	Open system Lab	Prof.Baradwaj	CSE
2003 - 120	Arindam	Speech Lab	Distributed computing lab	Prof.Ramanan	CSE
2003 - 121	Ravi	Open system Lab	Speech Lab	Prof.Fernandes	EE
2003 - 121	Ravi	Open system Lab	Open system Lab	Prof.Baradwaj	CSE
2003 - 121	Ravi	Open system Lab	Distributed Computing Lab	Prof.Ramanan	CSE
2003 - 122	Hema	Distributed Computing Lab	Speech Lab	Prof.Fernandes	EE
2003 - 122	Hema	Distributed computing Lab	Open system Lab	Prof.Baradwaj	CSE
2003 - 122	Hema	Distributed computing lab	Distributed computing Lab	Prof.Ramanan	CSE
2003 - 123	Vasu	Open system Lab	Speech Lab	Prof.Fernandes	EE
2003 - 123	Vasu	Open system Lab	Open system Lab	Prof.Baradwaj	CSE
2003 - 123	Vasu	Open system Lab	Distributed Computing Lab	Prof.Ramanan	CSE

Columns matching the query are shown in PINK.

So here it says note how the attribute names are changed that is it becomes student dot roll number, student dot name, student dot lab, lab dot name, lab dot faculty and lab dot department. Now the query that we require to match was student dot lab equal to lab dot name. Now all these quires that match are shown here in pink. Now it is these tuples that are going to be returned. So the result of the query would be something like this. That is where the student working in a particular lab is same as the name of the lab for whose record that we are maintaining.

So essentially what we have done here is that from two different attributes or two different relations student and lab, we have made just one relation by their Cartesian product and then given it as just any other input to a select operator. So it has become just one relation as seen in this slide here. The last slide that should be looking here today would be what exactly are the properties of this Cartesian join operator, just like we saw the properties of project and select.

(Refer Slide Time: 00:46:14)

Query Result

Student Roll No	Student Name	Student Lab	Lab Name	Lab Faculty	Lab Dept
2003 - 120	Arindam	Speech Lab	Speech Lab	Prof.Fernandes	EE
2003 - 121	Ravi	Open system Lab	Open system Lab	Prof.Baradwaj	CSE
2003 - 122	Hema	Distributed computing lab	Distributed computing lab	Prof.Ramanan	CSE
2003 - 123	Visu	Open system Lab	Open system Lab	Prof.Baradwaj	CSE

(Refer Slide Time: 00:46:59)

Properties of Cartesian Join (\times)

- The Cartesian Join represents a *Canonical Join* of two or more relations.
- If relation R having $|R|$ tuples and relation S having $|S|$ tuples are joined using \times , then $|R \times S| = |R| \cdot |S|$
- Cartesian join is too inefficient for most queries involving multiple relations.

Now the Cartesian join represents what might be termed as a canonical join between two relations that is it just joins every relation from the first tuple to every other relation in the second tuple. In the example here it just joined every student record with every lab record whether it made sense or not.

It was if a human being read the two tables, he would have noted that many of these joints do not make sense that is even though it says that the student works in a particular lab that record is joined with some other lab which has no relationship with what the student is doing.

So essentially if the number of tuples that come out of a Cartesian join is actually the product of the number of tuples that exist in each of the relations that make up the join. So hence Cartesian join as you might have imagined is actually two inefficient for joining tables, especially if you note that one table has 10000 records and the other table has 1 million records and the Cartesian product would be 1 million times 10000 records. And probably the output would be something which maybe 10 records or so which is not clearly worth it. So Cartesian join is mainly of theoretical interest in the sense that this is a canonical form of join operator by which we can join two or more relation to form just one relation because each operator requires just one relation as its input.

So in the next session we are going to be looking at several other join operators especially what is called as the theta join operator and see how it is actually computed and how its going to be more efficient than the Cartesian join operator. We are also going to look at certain more, certain other relational algebra constructs and see how we can express the several different queries using the relational algebra.

So to summarize what we saw today the relational model is a data model for the internal schema of a database and the internal schema is something which is oriented towards optimized performance on a computer rather than human consumption that is something that is meant for human beings to see and understand and so on.

And the relational algebra, an algebra is some kind of formalism over which we can build sound software that is something that's based on a mathematical formalism can be used to build sound software. So comprises of operators that very elegantly take relations as input and produce relations as output and then you can start combining relations from one another and so on. And we also just started to see how we can combine relations so that we can give just one relation as an input that is required by each of the relational operators.

In the next session we are going to be looking at certain more sophisticated forms of combining relations which are much more efficient both in terms of space and time required to compute these joins. So this brings us to the end of today's session.