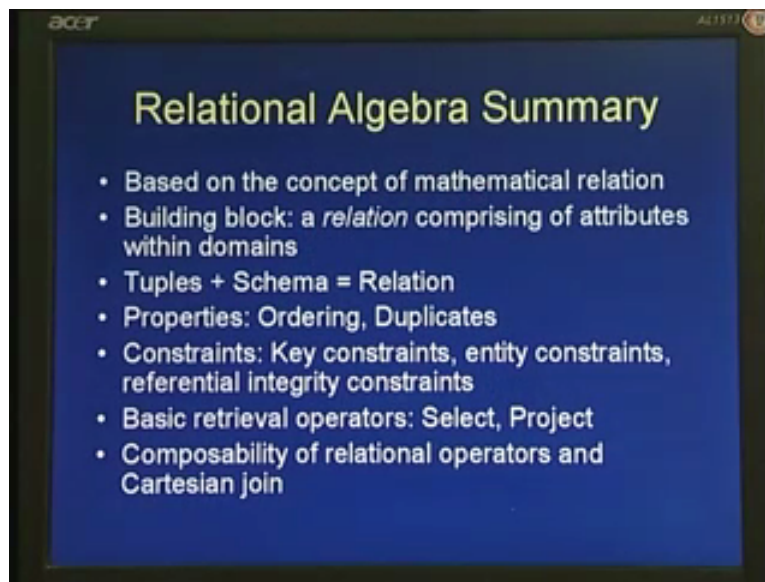


**Database Management System**  
**Dr. S. Srinath**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Madras**  
**Lecture No. # 4**

**Relational Model**

Hello and greetings. In today's session we shall be exploring relational algebra into some more depth. We started with relational algebra in the previous session with the definition that relational algebra is a data model that is based around the mathematical concept of a relation. Let us briefly review what we have learned about relational algebra before going further on into some more concepts in the relational algebra. So just to revise what we have already seen, the relational algebra is based up upon the notion of a mathematical relation.

(Refer Slide Time: 00:01:48)



A mathematical relation as we saw before represents a mapping between two or more domains. Essentially one relation represents a mapping between two domains that relates each element of particular domain to some other element of another domain. So for example I might have the set of all student names being related to the set of all roll numbers and a mathematical relation is a subset of this mapping that is the set of all valid assignments between students and roll numbers. So this forms underpinnings or mathematical underpinnings behind which relational model is based upon and as we also saw yesterday, the relational model is mainly meant for the design of the internal schema of a database. An internal schema is meant or is meant or optimized towards machine consumption that is its optimized towards efficient storage, retrieval and queries of the data rather than human consumption that is trying to look at the data model and trying to understand what the data modal does or what the schema is all about.

So the building block of a relational model as we saw yesterday and it's also shown in the slide here is a set of relations that contains a set of attributes and each attribute belongs in a certain domain. For example the set of all students belongs to the domain of the set of all valid student names. Similarly the set of all age attributes belongs to the set of all valid age for a given employee or student or whatever. So we also saw the notion of a relational schema which describes how this relation looks like that is what are all the attributes that makes up these relation and what are the domains of the attributes that make up these relation.

An each relation is dereferenced by a relation name and every instance of a relation that is every data element that conforms to this relation is called a tuple and each tuple in a relation is independent of every other tuple in the relation. Hence for example if I have one record of a student compressing of the roll number, name, date of registration, date of birth and so on that constitutes a tuple in the relation called the student relation and each tuple that is each record about student is independent of the other records of other students. And a relation by itself is a combination of tuples plus schema that is given a particular schema and a set of tuples that conform to this particular schema is what is called as a relation.

As you saw in the previous class, a relation is the input and the output for most of the relational algebra expressions like select and project which we had seen. And what are the properties of relations? The first property is about the ordering or the lack of it that is tuples in a relational schema or in a relation need not have any order, they need not be placed in any particular order. There relation is just a set of tuples that conform to a particular schema and the second property was about duplicates. Traditionally or the pure relational model does not allow for duplicates of a tuple which means to say that the each tuple is unique when you take a tuple in its entirety.

So we also made a statement that by default the entire tuples forms the super key for a tuple that is using the contents of a tuple, you can identify each tuple uniquely in the relation. We are going to actually generalized on this concept today to see how things would change, if you can allow for duplicates of the tuples and whether it is required or whether it is desirable to allow for duplicates to exist in a relation. We also saw that the relational model is defined by certain kinds of constraints and one of the first of which is the key constraint. So we defined a super key as something which can identify a tuple uniquely and we also defined a minimal super key or a key which says that no subset of which is also a super key in itself. And we also saw the notion of a candidate key where two or more sets of minimal super keys can be used as keys and the notion of the primary key and a foreign key when it comes to referential integrity constraints.

We also looked at entity constraints which says that the primary key of a relation may never be null. So you cannot have a tuple in which the primary key is null because null is not a valid value for an attribute. It basically says that the attribute is not applicable that there is no value associated with null. And we also saw basic retrieval operators select and project operators represented by the Greek letter sigma and pi and select is an operator which selects a subset of tuples from a given relation without changing the

schema of the relation that is the input relation and the output relation from a select operator have the same schema while a project operator selects in a sense, specific columns of the input relation that is it changes the schema of the relation without changing the data in the tuple. It does it or it may change the number of tuples, if we mandate the fact that the output of the project relation has to be a set that is it cannot be a bag or a multi-set that is each tuple from the output relation has to be unique. So, in which case the number of tuples that are returned would be less than the number of tuples that already exists in the relation.

We also saw that both select and project operators require specific relation that is exactly one relation has input and provide exactly one relation as output. Now whatever it do when we have more than one relations on which we have to answer a query. So we saw one possible solution to this that is to use the Cartesian join or the Cartesian product. Now how do we define the Cartesian product over relations? Remember what is meant by the Cartesian product over sets. If you have two sets A and B, a Cartesian product A times B or A cross B is the set of all mappings from all elements of A to all elements. This is the same definition for a Cartesian product for relations as well when we consider relations as simply set of tuples.

So a Cartesian join between two or more relations is the combination of set of all tuples from every relation to every other relation. So, a Cartesian join as we saw yesterday is unnecessarily expensive in the sense that if I have m tuples in the first relation and n tuples in the second relation it needs to first compute m times n or m n number of tuples to generate a table which in turn goes as input to the select and project operators and from where selection has to be made. Clearly this is very inefficient for most operations on involving two or more tables. So let us move on today to look at other forms of join operators which generate for lesser number of tuples than the canonical join that is represented by the Cartesian join.

The first join operator that we are going to see today is what is called as theta join operator. This as shown in the slide here, a theta join operator shows a join symbol which has essentially something like Cartesian product symbol with two parallel lines and which has a subscript called theta which is again shown in the title here.

(Refer Slide Time: 00:09:26)

**Theta Join Operator ( $\triangleright\triangleleft_{\theta}$ )**

Theta join is used to combine related tuples from two or more relations ( specified by the condition theta ), to form a single tuple.

Consider the earlier relation Student and Lab.

The relation :

$$\text{Student } \triangleright\triangleleft_{\text{Student.Lab = Lab.Name}} \text{ Lab}$$

returns the same result as :

$$\sigma_{\text{Student.Lab = Lab.Name}} (\text{Student} \times \text{Lab})$$

However, the number of tuples generated by Join is 4 rather than 12 as in the Cartesian join.

Note that this is most efficient when Student.Lab is a *foreign key* into Lab (and Lab Name is a primary key of Lab) and referential integrity is maintained.

So a theta join combines two or more relations or combines the tuples of two or more relations in a way that is specified by a join condition and the join condition is specified by the operator theta. So the slide here shows a specific example where the same relation that we took yesterday the student and lab relation is computed using a theta join operator. That is the slide shows student in a join operations with the lab condition that is the student relation theta join lab such that student . lab equal to lab . name.

So student . lab equal to lab . name is the joint condition and the theta join operator is the operator that is going to combine student and lab relations. So this is the relation that is shown here and as you can see the output of this relation is the same as the select operator that we saw in the previous session. That is select student . lab equal to lab . name from student time's lab that is the Cartesian join between student and lab. The only difference here is that the condition for this Cartesian join is specified as part of the join operator itself. Now how many tuples does this generate?

(Refer Slide Time: 00:11:20)

### 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.

Roll No	Name	Lab
2003 - 121	Arindam	Speech Lab
2003 - 122	Ravi	Open system Lab
2003 - 123	Hema	Distributed computing lab
2003 - 124	Vasu	Open system lab

Name	Faculty	Dept
Speech Lab	Prof. Fernandes	EE
Open system Lab	Prof. Baradwaj	CSE
Distributed computing lab	Prof. Ramanan	CSE

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

Let us have a look at the student and lab tables from the previous example. The student table has four different tuples and the lab table has three different tuples. Now the Cartesian join operator initially generated 4 times 3 that is 12 different tuples as input to the select operator.

(Refer Slide Time: 00:11:39)

### Cartesian Join (x)

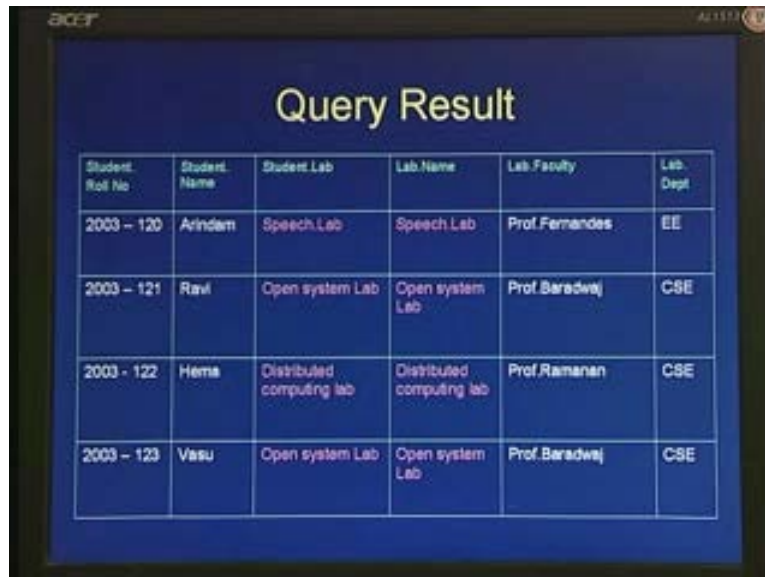
Student Roll No	Student Name	Student Lab	Lab Name	Lab Faculty	Lab Dept
2003 - 120	Arindam	Speech Lab	Speech Lab	Prof. Fernandes	EE
2003 - 120	Arindam	Speech Lab	Open System Lab	Prof. Baradwaj	CSE
2003 - 120	Arindam	Speech Lab	Distributed computing lab	Prof. Ramanan	CSE
2003 - 121	Ravi	Open system Lab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

On the other hand, the theta join operator starts with the condition that student . lab equal to lab . name is a prerequisite for computing the join between the two tables. So in this slide here such tuples where student . lab equal to lab . name is shown in a different color, are shown in pink. So, as you can see there are only 4 such tuples that match this

condition. Hence the number of tuples that are generated as input to this select condition is just 4 instead of 12.

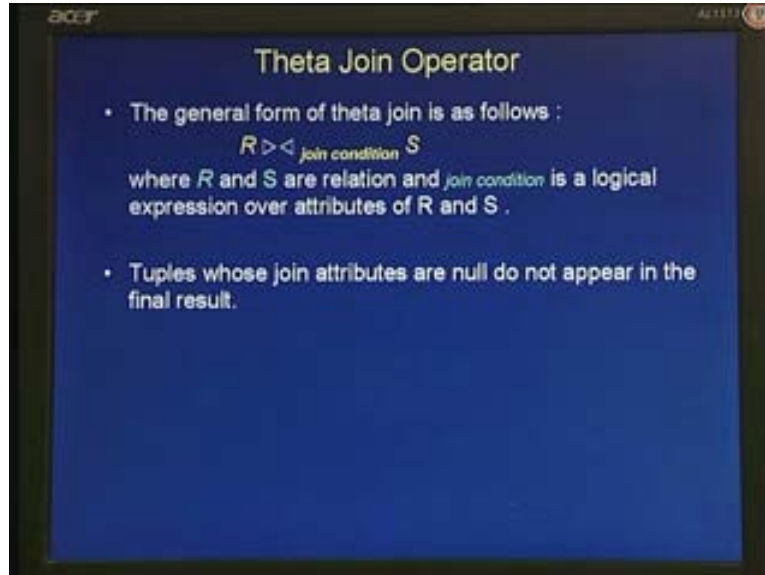
(Refer Slide Time: 00:12:13)



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	Vasu	Open system Lab	Open system Lab	Prof.Baradwaj	CSE

So how is the theta join computed? Well, the general answer is it depends. It would be note that in this case, it would be most efficient to compute this theta join operator. If lab dot name were to be a primary key of lab that is the lab is being referenced by the name and student . lab is a foreign key in to the lab relation and of course referential integrity is maintained. So because we have to compute the equality student . lab equal to lab . name, all I need to do is take up each student . lab attribute and search for the corresponding tuple in the lab relation because it is a foreign key and lab . name is a primary key this search can be uniquely done efficiently using several techniques which we are going to see later like indexing or hashing and then you compute the join between the two relations.

(Refer Slide Time: 00:13:25)

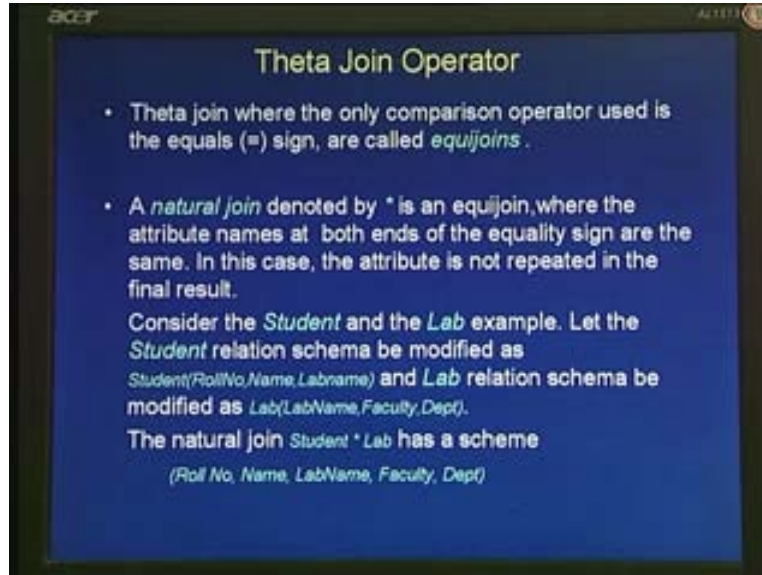


So the general form of the theta join relation is shown in this table or is shown in this slide. It's simply two relations with the join operator with a subscript join conditions, so the join condition is simply a logical expression over the attributes of  $R$  and  $S$ , which in the previous case we saw was the equality condition that is  $\text{student} . \text{lab} = \text{lab} . \text{name}$ . It need not necessarily be equality condition, it could actually be any other logical condition like less than or greater than or so on. Now it may so happen that in some cases the join attributes may be null which is also true even if there is referential integrity that is there may be a  $\text{lab} . \text{name}$  which is missing from a student record.

Now in such cases those tuples do not appear in the result that is whenever I cannot combine two or more relations or whenever a particular tuple cannot be combined with a corresponding tuple from the other relation, such tuples do not appear in the final result. So some more properties of the theta join operator. We saw a join condition now which sets  $\text{student} . \text{lab} = \text{lab} . \text{name}$ . Now the condition here or the logical operator here is the equality condition or the equal to condition.



(Refer Slide Time: 00:14:34)



Now theta join operators in which the only comparison operator that is used is equality condition is called an equijoin operator or an equijoin condition. So what we saw actually was an equijoin and theta join is more general in the sense that it could mean any other kind of attributes. Now a special kind of equijoin is of particular interest and this is what is called as natural join between two or more relations. So, natural join is denoted by star as shown in the slide here. It is an equijoin where some of the attribute names between the relations that are participating in this join are the same.

Now consider this example, consider again the student and lab example. Now let the student relation be modified like this that is the student relation has attributes name roll number, name and lab name instead of saying just lab. Similarly we have modified the lab relation as lab name faculty and department instead of just name. Now as you can see here, the lab name attribute or the name of this attribute is the same between the student and lab relations.

Now if I say student star lab which denotes a natural join between student and lab relations, it returns me a relation which is of this structure. That is the first three are the attributes of student and the last three are the attributes of lab and the middle attribute here lab name is the common attribute between student and lab. Now it is going to join only those tuples which where this common attributes match. So, this is a special kind of equijoin operator where not only equality condition is assumed but its also assumed which are the attributes on which the equality condition is operated upon. So the natural join simply takes attribute names which are the same in the two relations and then computes an equijoin over them.



(Refer Slide Time: 00:17:13)

**Renaming**

$$TA(id,LabName) \leftarrow \pi_{Roll\ No,Lab}(Student)$$

The above relation projects RollNo and Lab attributes from *Student* and renames them as id and LabName respectively in the output relation. It also names the output relation as "TA".

The above can also be achieved by "rename" ( $\rho$ ) operator as :

$$\rho_{TA(id,LabName)}(\pi_{Roll\ No,Lab}(Student))$$

So the next operator that we are going to be looking at is what is called as a renaming operator. We just saw here now that suppose we modify student as so and so or suppose we modify lab as so and so, we can just use natural join. Now can we formalize this notion of modification into the relational algebra itself? Can we introduce a notion into relational algebra by which we can say this relation is modified as this relation and then used in this expression and so on.

Now one way to achieve modification is by assignment. Yesterday we saw that the input and output of select and project operators are both relations therefore the output of these operators can be assigned to a new relation name and then this becomes a relation by itself. Now this assignment can be further generalized such that we not only assign to a new relation name, we also assign the attribute names of the new relation. So this slide shows the idea here, the first expression here projects the following statements that is it takes the student relation and then projects roll number and lab attributes of the relation.

Now once you get a relation as an output, it is in turn assign to another relation called TA or may be teaching assistant with the roll number attribute name replaced by id and lab replaced by lab name. So the output of this relation is another relation with its own name in this case TA and with its own attribute name that are different from the attributes names of the incoming relation.

Now this renaming operator or this kind of renaming can be implicitly achieved without an assignment statement by using the rename operator which is identified by the Greek symbol row which is shown below in the slide here. So the statement here, the relational expression here shows that the same thing that as the assignment statement above that is it projects roll number and lab from the student relation and then computes a rename or gives them as part of a rename expression that says rename it as TA id and lab name.

So how does the rename operator work in general? In general the rename operator contains or may be defined by the following properties. This is shown in this slide here. The first form of the rename operator shows that the entire relation, the input for the rename operator is of course a relation and of course the output is also a relation. So the entire relation is being renamed that is it is given a name S and the attributes are given names  $B_1 B_2$  extra until  $B_n$ .

The second kind of rename expression that is row subscript S operated upon relation R, we will rename only the name of the relation that is the output relation is called S in this case and in the third case where row has the subscript of  $B_1$  to  $B_n$  within braces or rather within parenthesis and the input is the same relation R, the renaming happens only on the attribute names and not on the table name itself.

(Refer Slide Time: 00:19:51)

**Renaming**

The general form of the rename operator is as follows :

$\rho_{S(B_1, B_2, \dots, B_n)}(R)$

Or

$\rho_S(R)$

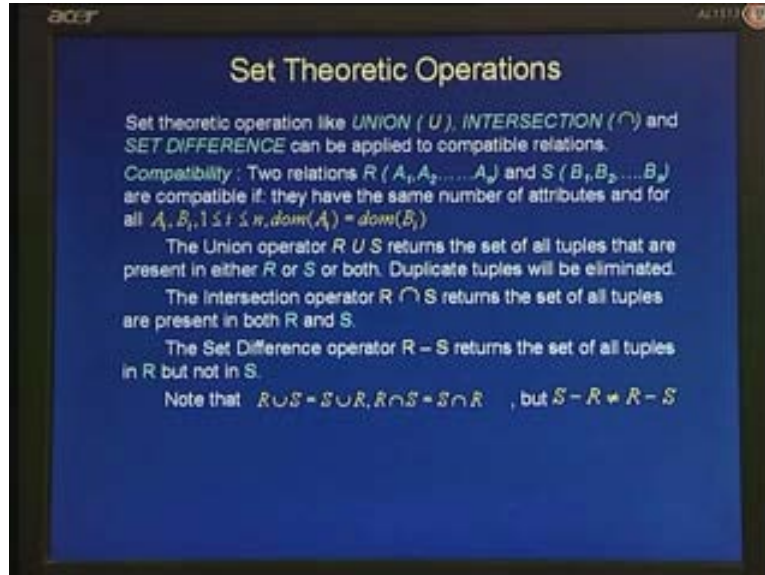
Or

$\rho_{(B_1, B_2, \dots, B_n)}(R)$

The first form renames both relation name and attribute names, the second form renames only the relation name and the third form renames only attribute names.

That is the name of the table or the name of the relation remains the same and the attributes are renamed to be called as  $B_1$  to  $B_n$  in this case.

(Refer Slide Time: 00:21:08)



The next set of operators that we will be looking here are the set theoretic operators. Again the set theoretic operators here operate on relations rather than specific sets as such. Now a relation is also a set but it could be multi-set or it could have certain differences when we are talking about joins and so on and so far. So the set theoretic operations like unions, intersection, set difference they can be applied in the relational model as well.

Now this set theoretic operation can be applied only to what are called compatible relations. Now what is meant by compatible relation between when we are considering two or more relations? That is can I compute a union operator between let us say a student relation and an employee relation or a student relation and a project relation. Now some of these union operators may make sense and some of these may not make sense.

Now that is why we formally define the notion of compatibility or union compatibility between two or more relations. Now what do we mean by compatibility? The formal definition is shown in this slide here. Suppose we consider two relations  $A$  and  $B$  or in this case  $R$  and  $S$  and suppose they have a set of attributes let us say  $A_1$  to  $A_n$  and  $B_1$  to  $B_n$ , they are compatible if and only if first of all you might have already noticed that the number of attributes are the same. That is the  $R$  has set of attributes  $A_1$  to  $A_n$  and  $S$  has a set of attributes  $B_1$  to  $B_n$ . So firstly the number of attributes are the same, if they have to be compatible and the domain of every corresponding attribute is also the same. That is the domain of  $A_1$  is the same as the domain of  $B_1$ . If  $A_1$  can span the set of all valid roll numbers,  $B_1$  should also span the set of all valid roll numbers and so on. Similarly for  $A_2$  to until  $A_n$ ,  $B_2$  until  $B_n$ .

So the set of all corresponding domains are the same. Note that there is nothing here about the names of each operators. Roll number could be called roll number in relation  $A$  and could be called id in relation  $B$ , it doesn't matter as long as the domains of each of

these attributes are the same, we should be able to compute the set theoretic operations like union intersection and set difference.

Now assuming that we have two or more relations which are compatible, how do we compute set theoretic expressions? The union operator  $R \cup S$  which is shown in the slide here it simply returns the set of all tuples that are present in either  $R$  or  $S$  or both and of course without any duplicates that is any tuple. Note here that entire tuples are compared between  $R$  and  $S$ , the entire tuples where the corresponding elements have to be same or compared. That is suppose I have one student relation or one record about a student in  $R$  and another record about a student in  $S$ , it just combines both of them that is the output of  $R \cup S$  is the set of tuples that lie either in  $R$  or in  $S$  or both and of course without duplicates.

Similarly the intersection operator returns the set of all tuples that are present in both  $R$  and  $S$  which is same as this intersection operator on sets. And similarly the set difference operator  $R - S$  returns a set of all tuples that are in  $R$  but not in  $S$ . So the set of all tuples that are unique to  $R$  but and not present in  $S$  would be the output of  $R - S$ . So we can note that this standard properties of set theoretic operations also apply here that is union and intersection are commutative  $R \cup S$  equal to  $S \cup R$  and  $R \cap S$  equal to  $S \cap R$ . However the set difference is not commutative,  $S - R$  is not the same as  $R - S$ .

So we shall be coming back to these set theoretic operators again when we relax the fact that a relation may not contain duplicates, now what happens if you allow for duplicates in the tuples. The next operator that we are going to be looking at is the division operator. The division is a slightly unintuitive operator in the sense that it needs a little bit of explanation to understand what or where a division is going to be used. A division operator is essentially used in cases where we may have to identify data elements that are associated with some other data element whenever the other data elements occur that is for all properties of the other data elements.

(Refer Slide Time: 00:25:43)

The Division Operator ( $\div$ ) is used to denote the conditions where a given relation  $R$  is to be split based on whether its association with every tuple in another relation  $S$ . Let the set of attributes of  $R$  be  $Z$  and set of attributes of  $S$  be  $X$ . Let  $Y = Z - X$ , that is the set of all attributes of  $R$  that are not in  $S$ . The result of the operation  $T(Y) = R(Z) \div S(X)$  is as follows. For every tuple  $t \in T$ ,  $\text{assoc}(t, S) \subseteq R$ .

A	B
a1	b1
a2	b1
a2	b1
a1	b2
a3	b2
a1	b3
a2	b3

A
a1
a2

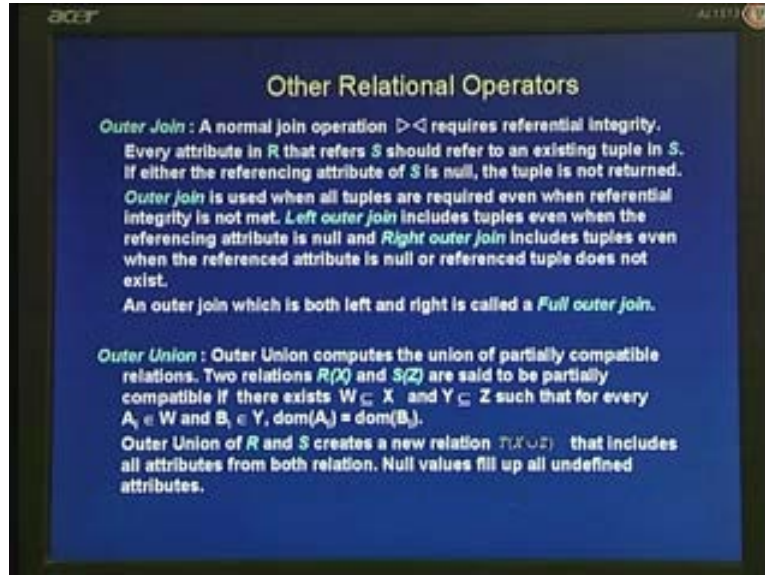
B
b1
b3

Have a look at this slide here which shows a particular example. Now firstly there is the definition here the division operator is used to denote conditions where a given relation  $R$  is to be split based on its association with every tuple in another relation  $S$ . Let me go straight to the example here and then go back to the explanation of division. The example shows two relations  $R$  and  $S$ . The  $R$  relation has two attributes  $A$  and  $B$  and the  $S$  relation has just one attribute  $A$ . Firstly the division of  $R$  and  $S$  is going to return the attribute  $B$  that is the  $R$  divided by  $S$  is the set of all attributes  $B$  such that there is some relation between the attributes in  $A$ . So what is that relation? The set of all attributes  $B$  contained in  $R$  that are associated with all values of attribute  $A$  of  $S$ . That is suppose let us take the example of  $b_1$ . Is  $b_1$  associated with  $a_1$  in  $R$ ? Yes it is,  $a_1 b_1$  is here. Is  $b_1$  associated with  $a_2$ ? Yes it is,  $a_2 b_1$  is also here and these are the only two values in  $S$ .

Hence  $b_1$  is a valid result in  $T$  equal to  $R$  divide by  $S$  that is every data element here that is associated with every other data element in the other relation is what is going to be returned as part of this relation. Consider an example something like which employee has worked with some other employee on all projects that he has worked. Let us say which employee has worked with some employee named Arun or something on all projects that Arun has worked. So suppose these were all the project that Arun has worked and there is this employee  $b_1$  who has also worked in all these projects, you are going to get  $b_1$  as an output.

Consider the case of  $b_2$  here, now  $b_2$  is associated with  $a_1$  but it is not associated with  $a_2$  there is some  $a_3$  with which it is associated. Hence  $b_2$  is not part of the result but  $b_3$  which is again associated with  $a_1$  and  $a_2$  it is part of the result. So the division operator in some sense divides the first relation based on which data elements in the second relation in some way completely divides that, that is associated with all data elements of the first relation. We are going to look at an example later on where division operator is going to be used and what is the power of the division operator.

(Refer Slide Time: 00:29:12)



There are also other relational operators which are also called additional relational operators which we shall briefly mention and have a look at their properties. We are going to mainly look at two such operators namely the outer join and the outer union operator. Now, going back to the notion of join, recall that join or a theta join operator takes a join condition as one of its input. Now suppose any of the attributes which match condition is null then such tuples are not further processed at all that is they are just thrown away from the relation. However in some cases it may be required to compute all possible joints even when the join attributes are null and this is what is called as outer join. So consider two relations R and S, now suppose let us say for the sake of simplicity we have defined a natural join between R and S.

Now every attribute in R which is participating in the joint should point to or should refer to an existing attribute in S that is it should not be null and the attribute that it refers to should exist in S, only then the natural join can be processed. However there can be two possible scenarios, the left outer join that is suppose the attributes that are participating in join the attributes of R that are participating in join are null. The left outer join includes such tuples even when the attributes that are or the referencing attributes are null. Similarly the right outer join includes tuples even when the referenced attributes does not exist.

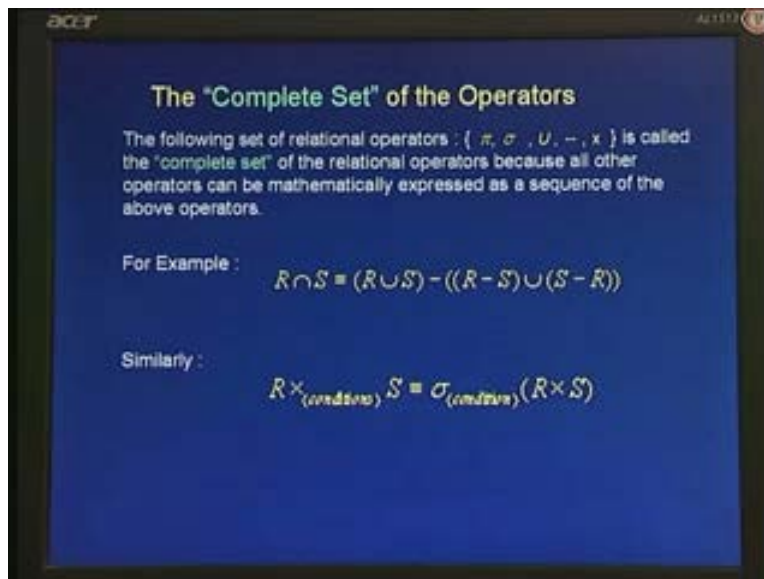
So it basically replaces them with nulls and then just includes that part of the tuples that is that part of the tuple from R which has data in it and the rest of the data elements would be null. So it is some kind of a union or a canonical, well I shouldn't say canonical but some kind of a union operator where you include every tuples any way, its some kind of an inclusive operator or inclusive join operator where it includes tuples any way even when the corresponding attributes are null. On the lines of outer join we can also define outer union operator.



Now we saw that the union operator that is union between or union or intersection or any set theoretic operators between two or more relations can be performed only when they are compatible. Now what did we define as compatibility as? The compatibility was that both of these relations should have the same number of attributes and the domains of the corresponding attributes should be the same in both these relations. Now the outer union is basically a relaxation of this constraint and the example shows two relations R X and S Z that is X is a set of all or the list of all attributes of R and Z is the list of all attributes of S.

Now suppose X and Z are not compatible, however a subset of X and Z are compatible that is W a subset of X and Y are subset of Z are compatible. Now a union or an outer union operator computes the union based on this compatible subset of these relations and then simply includes all other relations or all other attributes as they are in the relation. So similarly the outer union operator is simply some kind of an inclusive union operator that includes tuples or works on relations even when they are not union compatible or even when they are not perfectly compatible between them. So theoretically speaking there are, do we need all these operators or can we express one operators from other or has a basis of using other operators. This slide shows what is called as the complete set of relational operators.

(Refer Slide Time: 00:33:49)

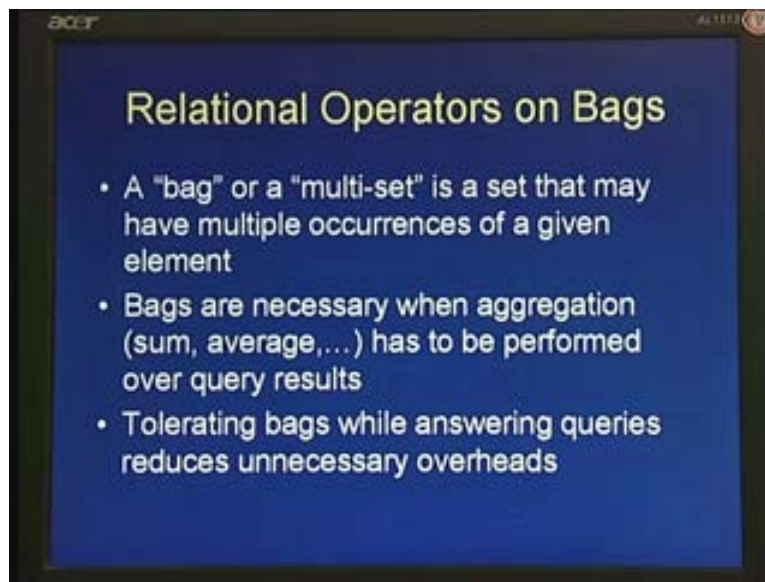


Now as you can see in the slide here, there are the following operators select, project, union, set difference and the Cartesian product. Now this set of operators is called the complete set of relational operators because every other kind of relational operator can be expressed as a sequence of the above operators. I am just giving two examples here but you can verify that for yourself taking each operator and trying to express it as a sequence of the other, one of the complete set of relational operators.



For example  $R \cap S$  can be or is equivalent to  $(R \cup S) - (R \setminus S \cup S \setminus R)$ . So let us not go into the set theoretic operation to prove this equality but I am sure this is quite obvious that you can express the intersection using union and set difference. Similarly joins can be expressed theoretically using a select condition over Cartesian products. We already saw that in the example there were student . lab equal to lab . name or in this case here  $R \bowtie S$  is the same as select condition over  $R \times S$ . The first one here should be the join symbol and  $R \bowtie S$  is equivalent to select condition over  $R \times S$ .

(Refer Slide Time: 00:35:25)



Now we are going to look at making one generalization over or trying to relax particular constraint on relational expression or on relations. Now until now we have been saying that relational operators should take relations that are sets and not multi-sets and return relations that are also sets and not multi-sets. However in some cases it might be necessary or even desirable to allow for the existence of duplicate or duplicate tuples in in relations both in the input and output relations.

Now first of all such relations or such sets where elements can occur more than once is called a bag or it's also called a multi-set. So bag or a multi-set is a set that may have multiple occurrences of a given element. As we can say it's a generalization over the present notion of a set that is every set is a multi-set in which each element occurs exactly once. However a multi-set is something where an element can occur more than once.

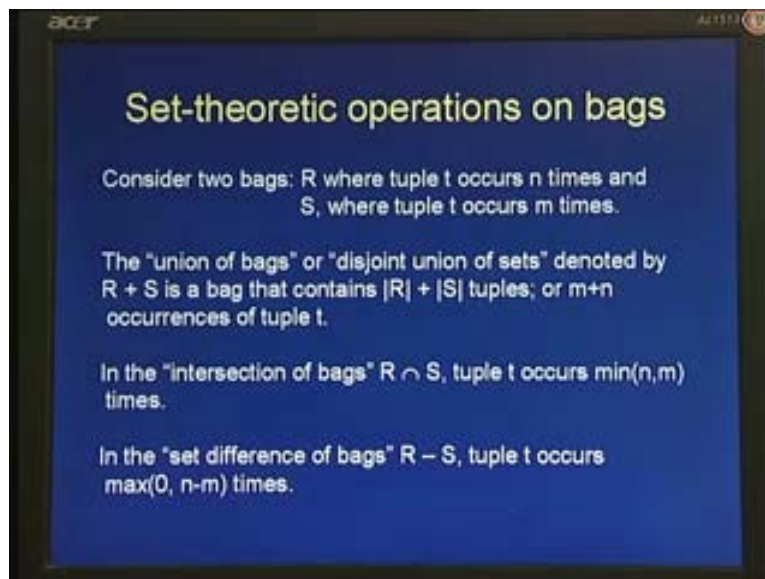
Now in some cases bags are actually necessary in the relational models not only desirable but also necessary. When are these conditions? Consider the case that we are querying the database to compute the average marks obtained by all students in a particular semester. So we have a student relation in which there is one of the attributes which is called as marks. Now we project this attributes saying project marks based on student.

Now based on the set of all marks that it has projected, we compute the average mark by computing the sum of all these marks and divided by the total number of entries that are there. Now in this case if duplicates were actually to be removed, we cannot compute the average in a correct fashion. We will actually be losing information when we change the multi-set or when we remove duplicates and make it into a normal set. So when we are computing aggregate relations like sum and average, we actually need multi-sets or duplicates in the relation, the duplicate should not be removed from the output relation.

Similarly if we can tolerate duplicates in the relations somewhere while evaluating relational expressions, it may make things much faster. For example every time I return the output of a project operations, I may have to spend considerable amounts of time trying to looking for each tuple and seeing whether there are duplicates of this tuple in the output relation. So especially when computing projects and unions duplicate removal may take a significant amount of time.

Now if I have a relational query that has several project operations and union operations and are embedded somewhere deep in the query, it may be terribly inefficient or to be computing or to be eliminating duplicates every time we compute project and union operator. So, sometimes it may be necessary or it may be desirable to tolerate bags or to tolerate multi-sets as part of a relation as part of an intermediate output in a relational expression. Now how does this generalization from sets to multi-sets affect relational operator?

(Refer Slide Time: 00:39:09)

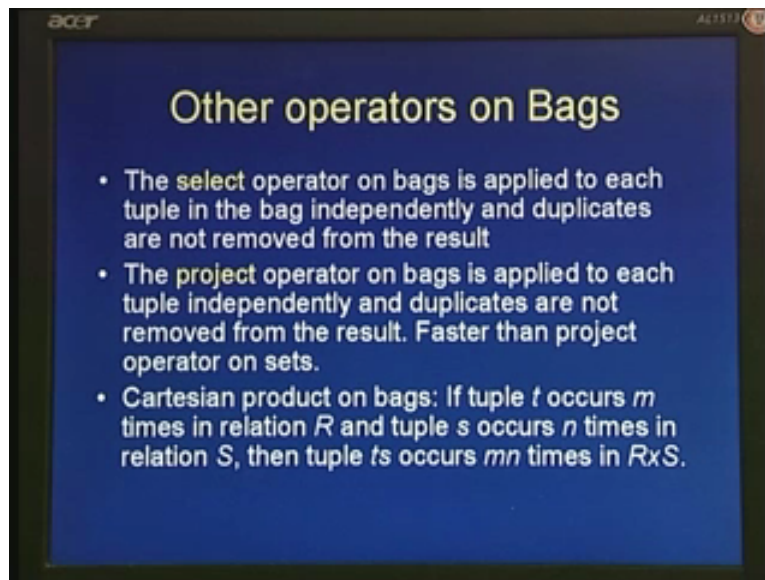


Now consider two bags R and S. Now in the first bag let a tuple t occur n number of times and the same tuple t occur m number of times **in R** in S. So how do we define set-theoretic operations based on these bags? The first operation that we define is called the union of bags or it's also called the disjointed union of sets and it is denoted by R plus S. You can also denote it by R union S when we are sure that R and S are bags and not sets.

So its simply denoted by  $R \cup S$  and its simply contains  $R \cup S$  that is the cardinality of  $R$  plus the cardinality of  $S$  number of tuples that is both bags are simply combined, every tuple in  $S$  is combined with every tuple in  $R$  and that's it. That is it has  $m$  plus  $n$  occurrences of this tuple  $t$  which has been repeating. Now when you are compute the intersection of bags,  $R \cap S$  what happens to tuples that occur multiple times? So tuple  $t$  which occurs  $n$  number of times in  $R$  and  $m$  number of times in  $S$  occurs only a minimum of  $m$  and  $n$  number of times in  $R \cap S$ . So as you can see the generalization here that is in sets, a tuple will appear in  $R \cap S$  only if it appears in both  $R$  and  $S$ . Here it will appear the minimum number of times it appears in both  $R$  and  $S$ .

Similarly the set difference between bags, the set difference between  $R$  and  $S$ ,  $R - S$  is where the tuple  $t$  occurs  $n - m$  times that is  $n$  number of times it had occurred in  $R$  and  $m$  number of times it had occurred in  $S$ . So the number of tuples that is going to occur in  $R - S$  is  $n - m$ , if and only if  $n$  is greater than or equal to  $m$ . If  $n$  is less than  $m$  then the number of times is going to appear is zero, of course it can't appear negative number of times. So tuple appears does not appear, if the number of tuples in  $R$  does not out number the number of tuples in  $S$ . You can think of it has something like canceling out tuples from  $R$  and  $S$ . So for every tuple in  $R$ , we cancel out ever tuple in  $S$  and then see how many tuple are remaining in  $R$  after we have cancelled out all tuples in  $S$  and that is the number of tuples that we are going to take in  $R - S$ .

(Refer Slide Time: 00:41:45)



Now what is the other operators on bags? The select operator does not change, the select operator operates the same whether it is on sets or bags. You just take a select condition and apply it to each tuple in the input relation regardless of whether the relation is a set or a bag. Similarly the project operator becomes simpler. It does change but it becomes simpler that is a project operator simply takes the requested columns or requested attributes and gives them out, there is no need to eliminate duplicates.

Similarly the Cartesian product of bags, it's also the same thing. It is if a tuple  $t$  occurs  $m$  times in  $R$  and  $n$  times in  $S$  as before then the tuple  $t$  in  $R \times S$  that is  $t$  combined when with  $S$  occurs  $mn$  number of times. That is every tuple in  $R$  is combined with every tuple in  $S$  regardless of how many times they appear in  $R$  or  $S$ , in each of these relations.

(Refer Slide Time: 00:42:45)

**Algebraic expressions on bags**

The following algebraic expressions are valid when  $R$  and  $S$  are sets, but invalid when they are bags:

$$(R \cup S) - T = (R - T) \cup (S - T)$$

[1-1-1 principle]

$$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$$

[2-1-2 principle]

$$\sigma_{C \text{ OR } D}(R) = \sigma_C(R) \cup \sigma_D(R)$$

However there are certain algebraic expressions on bags that do not hold when the relations are tuples. Have a look at these algebraic expressions that are shown in the slide here. The first algebraic expression is the distributivity over the set difference operator that is  $R \cup S$  minus of  $T$  is the same as and is equivalent to  $R$  minus  $T$  union  $S$  minus  $T$ . You can easily verify that this expression is true when  $R$ ,  $S$  and  $T$  are sets. However this expression is not true or does not hold when  $R$ ,  $S$  and  $T$  are bags and not sets, when they are multi-sets.

Now why is it not true? Let us take an example, this is called the 1-1-1 principle. Now consider that a tuple  $t$  occurs exactly once in  $R$ ,  $S$  and  $T$ . There are some particular tuple or some particular data element that occurs exactly once in  $R$ ,  $S$  and  $T$ . Now when we compute  $R \cup S$ , in the resulting set this tuple occurs twice if it is a multi-set or if it is a bag. Now once we consider and then once we compute  $R \cup S$  minus  $T$ , this tuple which we are considering now would have occurred once that is 2 minus 1 number of times, once it could have occurred.

On the other hand look at the right hand side here. When we compute  $R$  minus  $T$ , this common tuple which had occurred exactly once is not going to occur in  $R$  minus  $T$ , it's going to occur zero number of times. Similarly in  $S$  minus  $T$  this tuple vanishes, it occurs zero number of times. So in the union between  $R$  minus  $T$  and  $S$  minus  $T$ , this tuple does not exist at all whereas there is one occurrence of this tuple in the left hand side of this expression.

So this expression does not work when R S and T are bags and not tuples and not sets. Take this second expression. The distributivity over intersection and union that is  $R \cap (S \cup T)$  is equal to  $(R \cap S) \cup (R \cap T)$ . This is of course easily verified when R S and T are sets. However this does not hold when R S and T are bags and this can again be easily verified by considering a specific counter example which is called the 2-1-2 principle.

I will not be going into detail into the 2-1-2 principle, you can use the same argument as we have used in the first case where we took the 1-1-1 principle that is consider a tuple that occurs exactly once in R S and T. Here consider a tuple that occurs exactly twice in R, once in S and twice in T and see what happens and see if the left hand side of this expression is equal to the right hand side of the expression as far as this tuple is concerned. And you can see why this relation does not exist, does not hold or this equality does not hold.

The third expression is also significant when we are considering bags and not sets. Expression gives a select operator that is select C or D, C and D are some conditions over attributes of over R, so I am selecting C or D over R. That is select any tuple where either C or D or both holds and give me all those tuples. Now if it were a set that is if R were to be a set, I can rewrite this as select C union select D that is select C over R and union it with select D over R but the same thing does not work when R is a bag and not tuples.

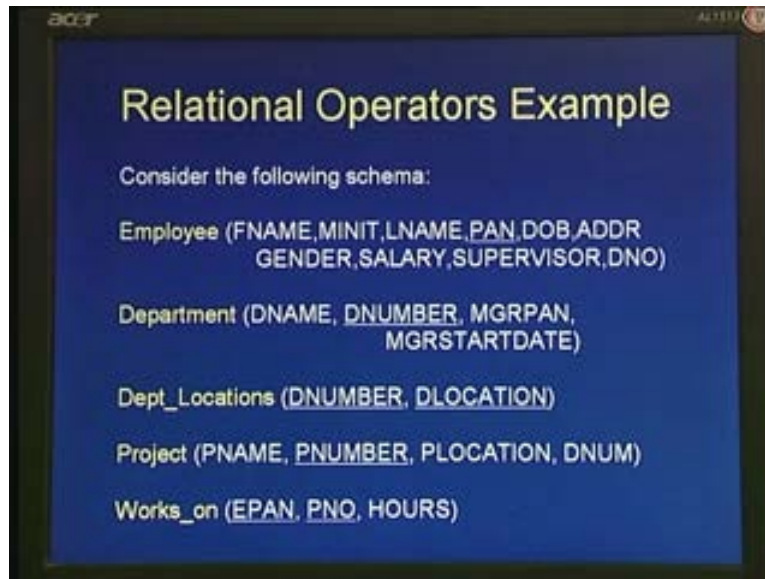
Now again you can take a very simple counter example to show that this is the case. Now consider a particular tuple where both C and D are true. Now that tuple is going to be returned only once in the left hand side of this relation but this tuple is going to occur twice in the right hand side of this relation because its going to be returned once from C and once from D and when we are taking a union or disjoint union, we are going to just add up both of them and its going occur twice in this relation. Hence this does not work when R is a bag

So tolerating bags is not only desirable but sometimes also necessary however bags pose their own unique problems, unique issues when we are considering set theoretic operations and algebraic expressions over bags and which we have to keep in mind when we say that when we either decide to tolerate bags or not tolerate bags. In a nut shell we have covered quite a few of quite a significant part of what constitutes relational algebra expressions and what constitutes or how to write queries in relational algebra.

In the next three slides let me give a small example of relational algebra queries and how queries can be composed from one another. So this slides shows a very small database schema comprising of 5 different relations employee, department, department locations, projects and works\_on. So employee is a relation that talks about details of an employee, it has the first name, middle, initials, last name, the employees pan number, date of birth, address, gender, salary, the supervisor of that employee and the department number where the employee works.

And the department contains department name, department number which is the key here, all primary keys are shown underlined and the pan number of the manager and the start date of the manager.

(Refer Slide Time: 00:48:18)



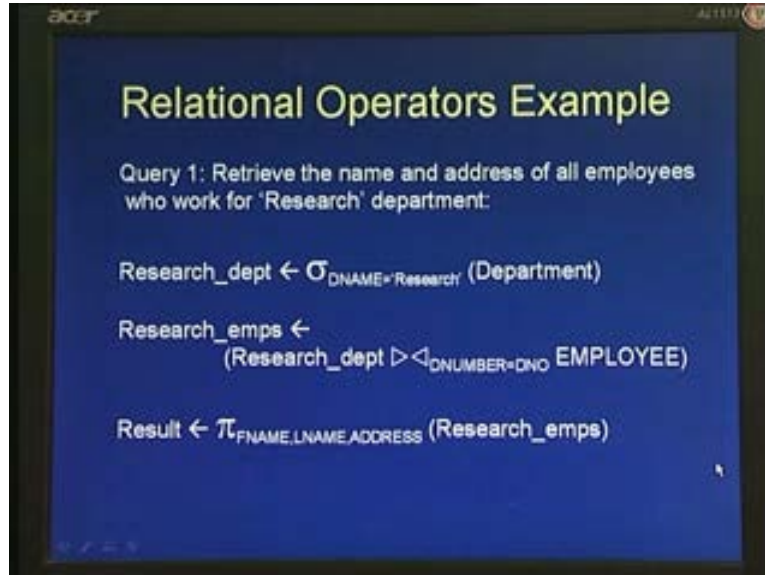
Department location only shows each department number and the location where it is located and project shows project name, project number, location and the department number where the project is working. And similarly works\_on talks about this employee works on this project for this number of hours and so on. So let us take some typical queries very quickly and go through how we can answer these queries.

Query one, the first query which we are going to consider says that retrieve the name and address of all employees who work for the research department. So how do we answer this query. First of all we take the set of all tuples that form the research department that is select DNAME equal to research from department, the set of all tuples which are the research department.

Now compute which are the set of all employees who works in the research department? How do we compute that? Compute a join between research department and employee where the department number is the Dnumber. Recall that in the employee record there was a Dnumber here which showed which is the department number where the employee work.



(Refer Slide Time: 00:49:24)



**Relational Operators Example**

Query 1: Retrieve the name and address of all employees who work for 'Research' department:

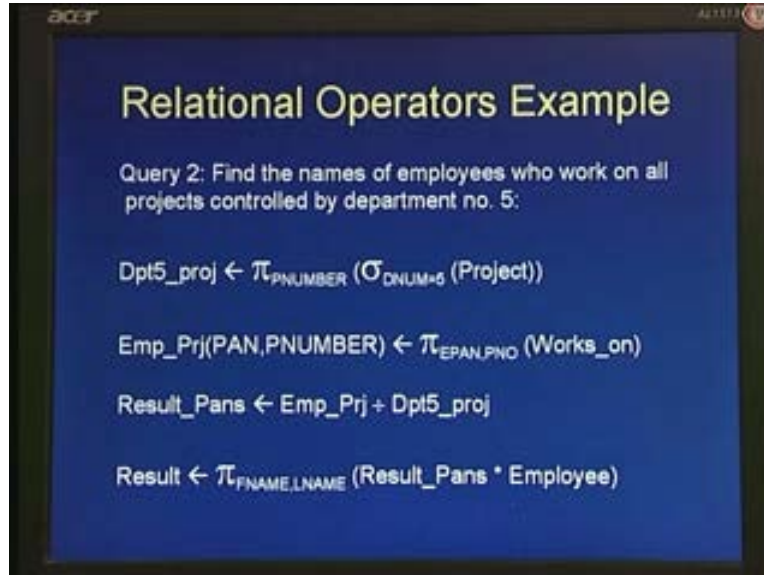
$$\text{Research\_dept} \leftarrow \sigma_{\text{DNAME}='Research'} (\text{Department})$$
$$\text{Research\_emps} \leftarrow (\text{Research\_dept} \bowtie_{\text{DNUMBER}=\text{DNO}} \text{EMPLOYEE})$$
$$\text{Result} \leftarrow \pi_{\text{FNAME,LNAME,ADDRESS}} (\text{Research\_emps})$$

So compute a join, an equijoin where this is this. Now from this we have got all details of employees who work in the research department, from that we need only the first name last name and address because that's what the query asked that is the name and address for employees, so project as a last query. Query two: Find the names of all employees who work on all projects controlled by department number 5. So have a look at this query again. We want the names of employees who work on all projects that are handled by this department. So how do we go about answering this? First of all let us find out what are all the projects that are being handled by department number 5.

So department 5 project is the name of the relation which says project, the project number and select from project where department number equal to 5 and project only the project number. Then which are all projects that employees work on? Take the pan number and the project number and then work on this. Now what we have to do is that we just have to compute a division between employee project and department 5 project which basically gives us the set of all pan numbers of employees who work on all projects or who are associated with all projects of department 5 which is the result.



(Refer Slide Time: 00:50:39)



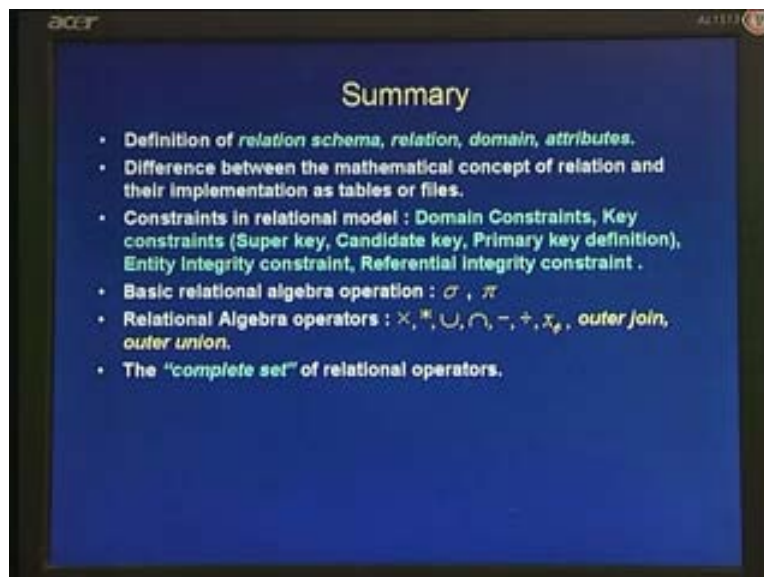
**Relational Operators Example**

Query 2: Find the names of employees who work on all projects controlled by department no. 5:

$$\text{Dpt5\_proj} \leftarrow \pi_{\text{PNUMBER}} (\sigma_{\text{DNUM}=5} (\text{Project}))$$
$$\text{Emp\_Prj}(\text{PAN}, \text{PNUMBER}) \leftarrow \pi_{\text{EPAN}, \text{PNO}} (\text{Works\_on})$$
$$\text{Result\_Pans} \leftarrow \text{Emp\_Prj} \div \text{Dpt5\_proj}$$
$$\text{Result} \leftarrow \pi_{\text{FNAME}, \text{LNAME}} (\text{Result\_Pans} * \text{Employee})$$

That is we in turn use that to and combine it with the employee record to return the first name and last name of employees. So in this way we can, as you can see here for any given query we usually need to perform a series of operation, series of relational algebra operations before we get to the final result.

(Refer Slide Time: 00:52:13)



**Summary**

- Definition of *relation schema, relation, domain, attributes*.
- Difference between the mathematical concept of relation and their implementation as tables or files.
- Constraints in relational model : Domain Constraints, Key constraints (Super key, Candidate key, Primary key definition), Entity integrity constraint, Referential integrity constraint .
- Basic relational algebra operation :  $\sigma$  ,  $\pi$
- Relational Algebra operators :  $\times$ ,  $*$ ,  $\cup$ ,  $\cap$ ,  $-$ ,  $\div$ ,  $\bowtie$ , *outer join*, *outer union*.
- The "complete set" of relational operators.

So let us summarize what we have learnt today in a brief fashion. So we saw the definition of relational schema, the notion of a relation, domains and attributes and the characteristics of relations especially with considering duplicates and ordering of tuples and so on.

We also saw the basic relational algebra retrieval operations that is select and project and so on and set theoretic operations and relations and also how this set theoretic operations change, when we relax the notion of the relation from being a set of tuples to a bag of tuples when we can allow for duplicates. We also saw why in some cases, it's not only desirable but also necessary to use bags. We also saw how we can, given a particular user requirement how we can go about formulating a relational algebra query in a step by step fashion. So that brings us to the end of this session on relational algebra. Thank you.