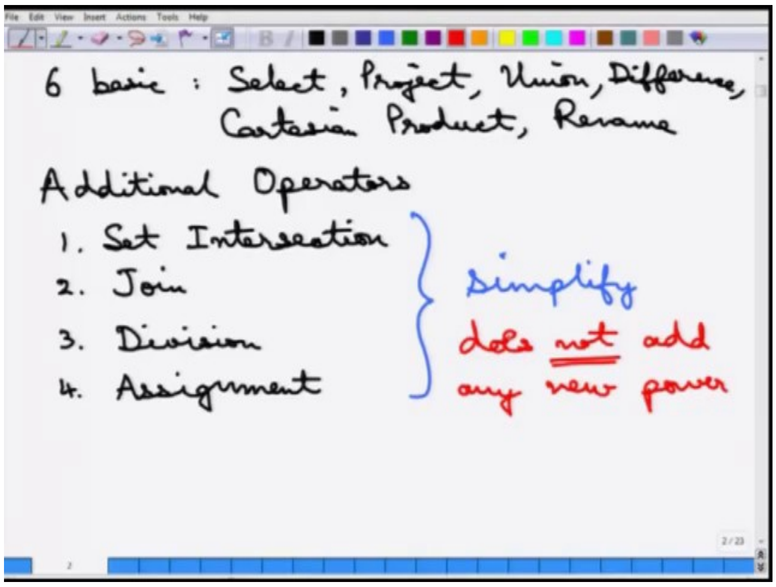
Fundamentals of Database Systems

Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 05
Relational Algebra: Additional Operators

Last time we saw six basic operators.

(Refer Slide Time: 00:12)



6 basic : Select, Project, Union, Difference, Cartesian Product, Rename

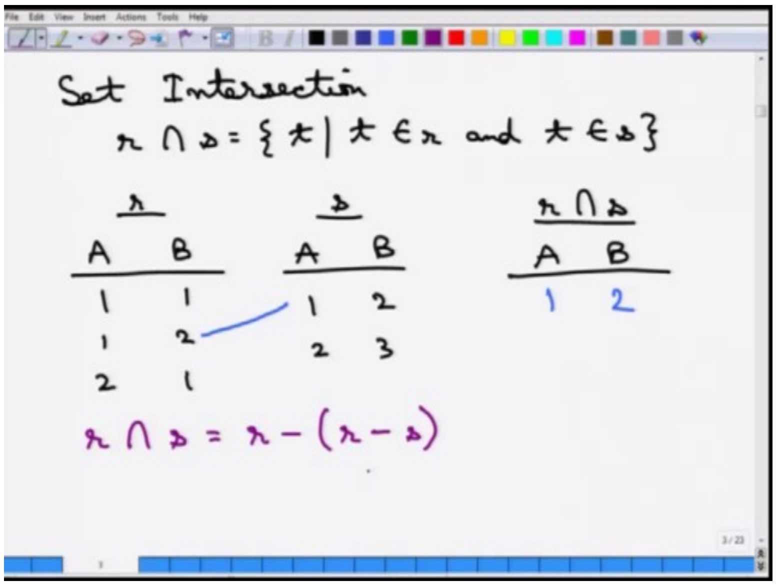
Additional Operators
1. Set Intersection
2. Join
3. Division
4. Assignment

simplify
does not add any new power

Now, we will introduce some additional operators. Let me first enumerate them and then I will go over what does it mean. So, first one is **intersection** or the set intersection, then the second one is **join**, third one is called **division** and the fourth one is **assignment**. So, essentially what these additional operators let you do this simplify the queries, but does not add any new power to the basic relational algebra.

So, any of the queries that can be solved using these new four additional operators can also be solved using the six basic operators. So, these four operators do not add any power to the type of queries that can be solved. However, this simplify writing the query very heavy and we will see examples of that later, but let me first go over each of these operators to see what they mean.

(Refer Slide Time: 01:42)



Set Intersection

$r \cap s = \{ t \mid t \in r \text{ and } t \in s \}$

r	
A	B
1	1
1	2
2	1

s	
A	B
1	2
2	3

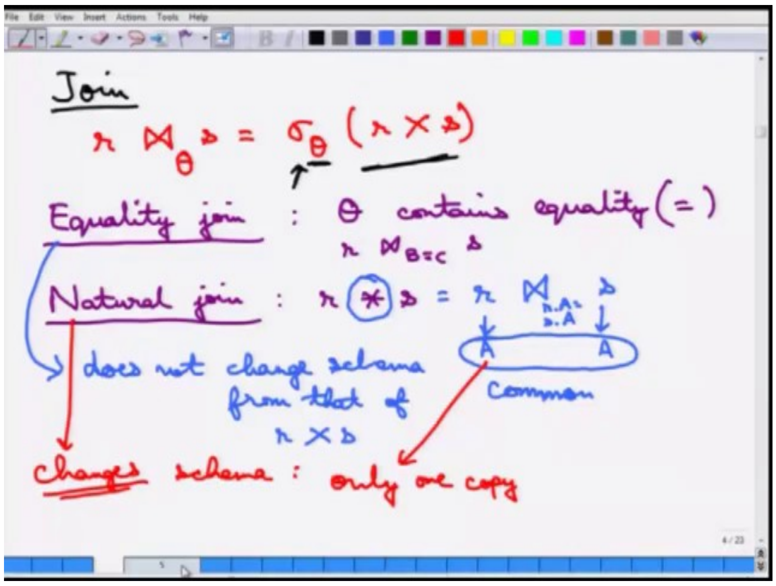
$r \cap s$	
A	B
1	2

$r \cap s = r - (r - s)$

So, the first one is set intersection. Set intersection is very simply the set intersection that we all understand for normal sets. So, essentially $r \cap s = \{t|t \in r \text{ and } t \in s\}$ so it has to be both the cases and just like the set union and the set difference, they have to have the same schema etcetera, etcetera and it is quite simple actually. So, just to complete let us have an example, so suppose this is (A, B) with (1, 1), (1, 2), (2, 1) and s is the same schema (A, B) with (1, 2), (2, 3) that is $r \cap s$ will produce again (A, B). So, the thing that is common in both the cases is simply this, so the only answer is (1, 2).

Now, the question is we are claiming that set intersection is not a new operator, it can be defined using the other operators and the answer to that how do we do that, because essentially you can see that set intersection, $r \cap s$, can be written in terms of the set difference operator. So, use the set difference operator, the set intersection can be written, so it does not really add any new power.

(Refer Slide Time: 03:08)



Join

$r \bowtie_\theta s = \sigma_\theta (r \times s)$

Equality join : θ contains equality (=)

$r \bowtie_{B=C} s$

Natural join : $r \circledast s = r \bowtie_{r.A = s.A} s$

A A

common

does not change schema from that of $r \times s$

changes schema : only one copy

So, let us go to the next operator, **join** and this is a very, very important operator as we will see, but again it does not really add any new power, but let us first define what join is. So, the symbol for join is this $\bowtie$, so this is like two triangles facing each other and we define there is a condition θ, so this is essentially $\sigma_\theta(r \times s)$. So, now, we can understand why I have been saying that this does not add any new power.

So, essentially what it does is it takes a Cartesian product and then select certain tuples based on the predicate theta. So, it is a selection using theta on the Cartesian product and we have already seen one example. So, the when we took a $borrower \times loan$, when we took that Cartesian product and then applied a theta which is $borrower.lno = loan.lno$, this is essentially a join using this condition.

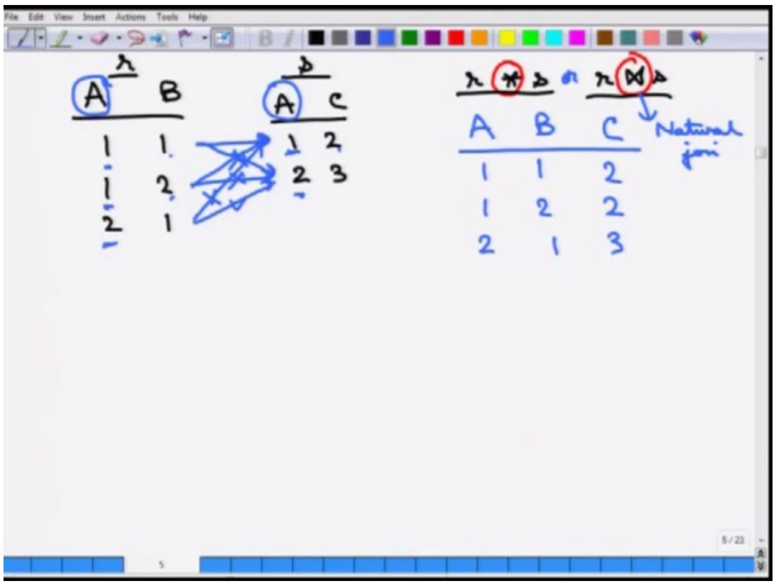
Anyway, so let us see an example, so join is actually very useful, so there are some versions of join that are defined, the first one is called an *equality join*. So, equality join essentially is that the θ condition contains equality (=), that is the equal to operator. So, for example, $r \bowtie_{B=C} s$. So, this equality operator is there, that is why it is an equality of join.

The next very important part is called a *natural join*. So, this will come up many, many times when we are studying SQL, etcetera later on and by join generally people do mean only natural join. So, natural join by the way has its own operator, this r it can be just denoted by this star operator (*), this is essentially equal to $r \bowtie s$. Now, to define a natural join between r and s, r must have an attribute A and s must have the same attribute A. So, I mean, so r must have some attribute which has the same name and the same schematic meaning as s, so it is essentially saying $r.A = s.A$. So, when there is the same attribute between two relations r and s and an equality join is proposed between r and s on that attribute on only that attribute, then it is called a natural join. So, there has to be a common attribute. So that is the thing there has to be a common attribute and that is why it is defined.

Now, a couple of interesting points about this is that equality join for say does not change the schema, or any join does not change schema from that of $r \times s$; However, natural join changes schema.

So, how does it change the schema? It essentially has the common attribute only one copy of the common attribute. So, it does not store both $r.A$ and $s.A$ which is obvious, because $r.A$ must be equal to $s.A$, so it keeps only one copy, so it does change the schema little bit, it essentially removes the redundant attribute. So, that is one thing to remember and let us see some examples of join.

(Refer Slide Time: 06:56)

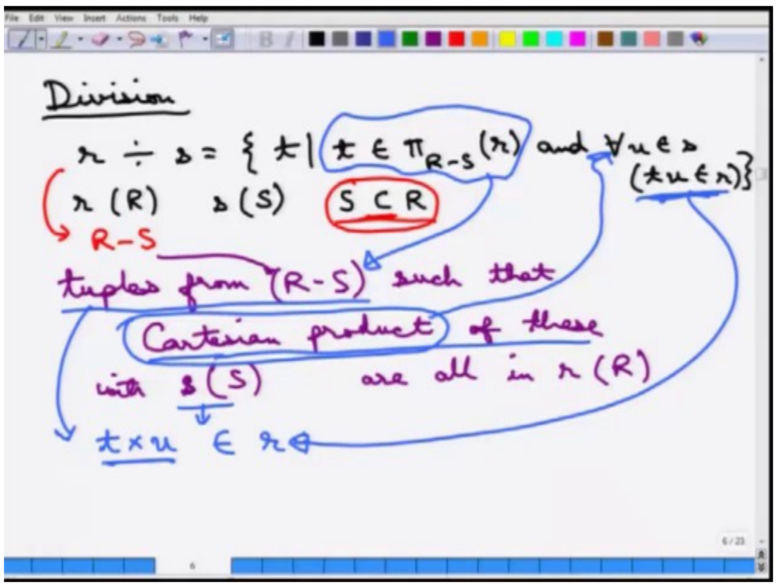


So, suppose this is *r* with the schema of (A, B) and these are the tuples there (1, 1), (1, 2), (2, 1) this is s with the schema of (A, C) and (1, 2), (2, 3). Now, if we do $r * s$ this can be also denoted as $r \bowtie s$, so this also stands for natural join, so this is natural join, so the answer for this is that. So, the common attribute is first identified which is A, so A is the common attribute, so only one copy of it is kept, then the rest are copied, so B and C and this is the schema of (A, B, C) then only those tuples are taken into account where A is same.

So, between this the A is the same, so this is copied, so only one copy of one is kept, B is 1 and C is 2. So, this is that between these two, these join does not happen, because the A is not the same, then we move on this joint does happen, because this 1 is equal to 1 and then the copy that is kept is 1, for B it is 2, for C it is 2 that is the thing. And similarly, this does not happen, because the joint conditions do not agree, this does not happen, because the join condition do not agree and this does happen, because 2 is the same and this produces (2, 1, 3).

So, this is the answer for the natural join, so this is once more the natural join. So, this is one example of how to define natural join, etcetera.

(Refer Slide Time: 08:43)



Division

$r \div s = \{ t \mid t \in \pi_{R-S}(r) \text{ and } \forall u \in s \; (tu \in r) \}$

$r(R) \quad s(S) \quad \boxed{S \subset R}$

$R-S$

tuples from $(R-S)$ such that

Cartesian product of these

with $s(S)$ are all in $r(R)$

$t \times u \in r$

So, let us now move on to the next operator which is called **division**, now division is a slightly complicated operator. Essentially what division tries to do is, let me give you the first big definition, the formal definition of division,

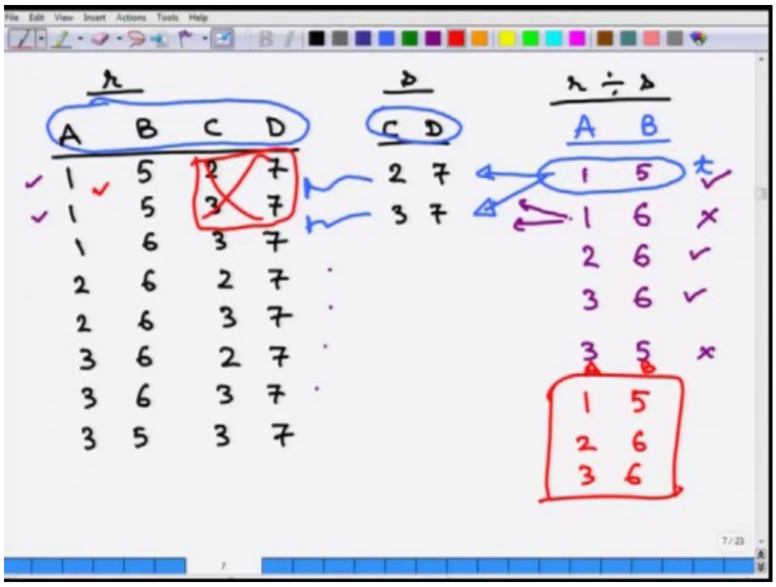
$$r \div s = \{t \mid t \in \pi_{R-S}(r) \text{ and } \forall u \in s, (tu \in r)\}$$

. So, this is really complicated as it sounds, but let me tell you what it does, essentially let me write it down this following way. First of all the schema of r. So, suppose r follows the schema R and s follows the schema S.

Now, for the division operator to be applied it must be that, the schema $S \subset R$, this is an important condition, this must be holding. So, this must be S is a subset of R, this must be happening; otherwise, this division operator cannot be applied. The number two is that the schema of r divided by s essentially becomes $R - S$ that is why this is important. But, more importantly what does $r \div s$ does is that it chooses tuples from this particular schema $R - S$ such that Cartesian product of these with s(S) are all in r(R).

So, suppose you take a tuple from $r - s$ suppose the tuple is *t*, so Cartesian product of this with *s* is suppose this tuple is *u*. Now, the Cartesian product... So, *t u* must be part of your r relation r, because this is what the Cartesian product you have taken, so this is what this condition says. Now, this is for all *u*'s, the Cartesian product, because it is a Cartesian product this captures all *u*, this is the all *u* part and this part is the *t* belonging to *r* and this is essentially saying what is the schema of this, this is from this part, so which is what it is doing. So, it is a little complicated example and let me go over an example to say what I have been saying.

(Refer Slide Time: 11:31)



r

A	B	C	D
1	5	2	7
1	5	3	7
1	6	3	7
2	6	2	7
2	6	3	7
3	6	2	7
3	6	3	7
3	5	3	7

s

C	D
2	7
3	7

$r \div s$

A	B
1	5
1	6
2	6
3	6
3	5

A	B
1	5
2	6
3	6

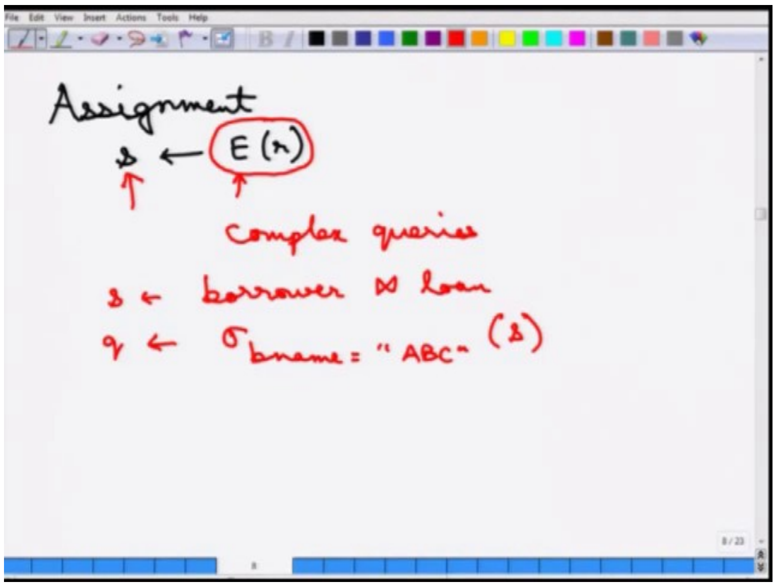
So, here suppose *r* is (A, B, C, D). This is the schema of *r* and this is your *s* which is just (C, D) which is just (2, 7) and (3, 7). So, first of all $r \div s$, so first of all we must decide what the schema is, now the schema is, as I told you, this is essentially you take this schema and this schema and it is the of difference. So, the schema is only (A, B) that part should be easier to understand, now suppose you have a particular tuple *t* here.

So, this is particular tuple we will fill it up later, but this *t* when joined with these two. So, that is taken the Cartesian product both of them must be in the original relation *r*, so both of them must be in this relation *r*. So, let us try and fit some bills. First of all, let us see if does (1, 5) is fit the bill, if (1, 5) fits the bill then (1, 5, 2, 7) must be part of that which is true, then (1, 5, 3, 7) will be part of that. So, (1, 5) is an answer set, so (1, 5) is correct.

Now, let us take (1, 6). Now (1, 6) again it must be joined with both of these, so (1, 6, 2, 7) is not part of the answer set, so this is gone, this is not part of this thing. So, then let us take (2, 6). Is (2, 6) part of it? well (2, 6, 2, 7), (2, 6, 3, 7) are there. So, this answer correct answer. Then is (3, 6) there? (3, 6, 2, 7) and (3, 6, 3, 7) are both there which is correct and (3, 5) no, because (3, 5, 2, 7) is not there. So, the answer essentially is then simply let me write it down, the answer is (1, 5), (2, 6), (3, 6), so this is the answer with A and B of course.

So, now, the question is how are this kind of division such a complicated operator useful. So, we will see some examples where the division operator will be used, but just a very quick basic idea to start making you thinking is that if this part of the division then it must happen. So, essentially you take the relation r and you divide by s, so you divide such that... So, if whatever 2 and 7 is here, so you select only those tuples where (2, 7) and (3, 7) are both present, so you get out that part and you just select. So, this is what the idea is, but we will see an actual example later.

(Refer Slide Time: 14:11)


Assignment

$s \leftarrow \boxed{E(r)}$

complex queries

$s \leftarrow \text{borrower} \bowtie \text{loan}$

$q \leftarrow \sigma_{\text{bname} = \text{"ABC"}}(s)$

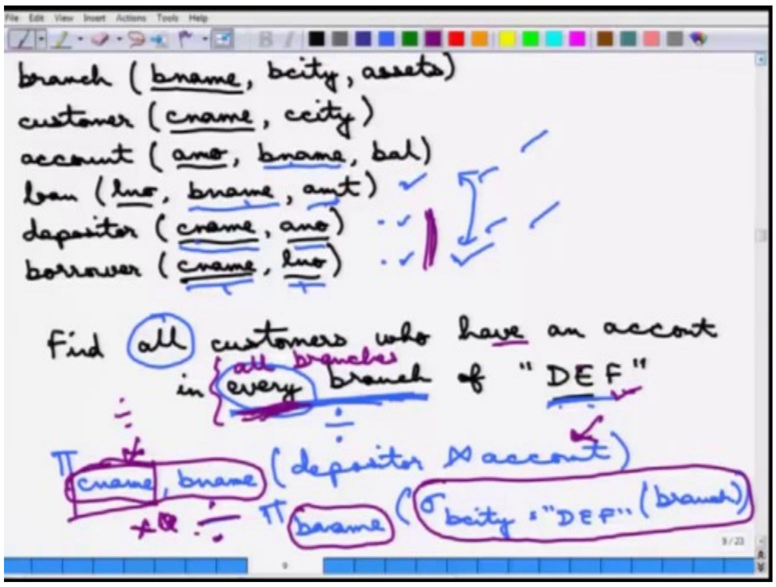
So, we will move on to the last general operator which is the **assignment**. This is denoted simply by this $s \leftarrow E(r)$. This is almost similar to the renaming operator, but what it does is that, so, this is an expression, the expression (*E*) has been applied on *r* and you temporarily assign it to a variable s. So, this is very useful in complex queries, where instead of just writing things on top of each other, you assign to s and then you write it down.

So, for example, in the previous query what some queries back, so you can write s to be that whatever you say. So, that you can say this is $borrower \bowtie loan$ and then you can say the answer to my query is $\sigma_{bname=ABC}(s)$. So, you can see that this does not add any power to this, but it essentially simplifies writing the query.

So, that finishes the basic definitions of what the additional operators are. We will go over some examples next.

Now, let us go over some examples that, uses the additional operators that we learnt.

(Refer Slide Time: 15:40)


branch (bname, bcity, assets)
customer (cname, ccity)
account (ano, bname, bal)
loan (lno, bname, amt)
depositor (cname, ano)
borrower (cname, lno)

Find all customers who have an account in every branch of "DEF"

all branches

$\Pi_{cname, bname}(depositor \bowtie account) \div \Pi_{bname}(\sigma_{bcity="DEF"}(branch))$

So, coming back here this is the same banking example that we have been following. So let us start solving some queries. So here the operative word is both. So, which means that this is a set intersection query and the way to solve this is essentially well we are just doing this let us say borrower intersection with depositor. Now, this is interesting, because although we say find all customers essentially we meant to say find names of all customers. So, that is the, because customer name is the identifying information for customer. So, we can just do this.

Now more importantly the borrower, so every customer which has a loan must be in this borrower table and every customer who have an account must be in depositor table. So, we can just use these two relations and solve this query. More importantly this projection of customer name must be done before the intersection is done. Because, borrower cannot be directly intersected with the depositor, the reason is they do not have the same schema. So, we must first ensure that the schema is the same and then the intersection can be taken. So, if this is done.

Let us move on to the some other queries. So this kind of queries wherever there is a find all customers, who have an account in every branch. So, the operative word is every branch, wherever there is a every branch kind of a query, this points out to the division operator, because we need to find out, So, the every branch that is what the division queries try to solve. So, essentially the way to solve this is that we need to first find out this customer name and branch name from the depositor natural join account. So, what does the depositor natural join account do? This depositor natural joint account gives the information. So, depositor this with account gives the information about all the customers and everything. Now, we are selecting only the customer name and the branch name, now this branch name must be in the every.. So, the branch name city is DEF, so that is what we need to do. So, this when we divide it with the branch name of everything where the branch city is DEF from of course, the branch table.

So let me do it, a little bit one at a time. So, what does this mean? So, this branch city is equal to this, from the branch this, selects all the branches in every, so essentially it is every branch in the city DEF. Now, we are selecting only the branch name out from this, that is what we need and here it selects a customer name and the branch name from every possible depositor. And then, if we do a division, then it essentially selects out this customer names, such that this has an account in every branch. Because, this division operator makes you sure that this, so this customer name will be part of it only when this entire join, entire Cartesian product is present here. So, this is how to solve a query where there is this every branch, so whenever there is a every branch or you can say all branches of … So, whenever this kind of query then the natural way to think of it is using the division operator. We will have many more examples etcetera later for your practice, but this is what the way to think.

So, this finishes part of this normal relational algebra, where we have seen six basic operators plus four additional operators. Just to remind you that the additional operators do not increase the power, but it makes the solving some of the queries easier. Next we will go over an extended relational algebra, so we will define some more operators, where it does actually increase the power of relational algebra.