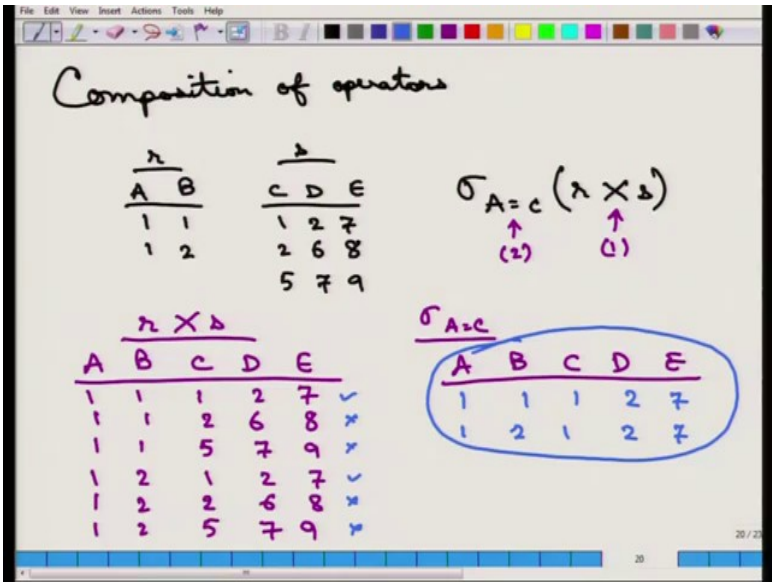
Fundamentals of Database Systems
Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 04
Relational Algebra: Composition of Operators

Welcome. Today we will talk about Composition of Operators. So, in the last time we saw some of the basic operators and today to understand how more than one operators can be used in the single query and we will also see certain queries for that.

(Refer Slide Time: 00:25)

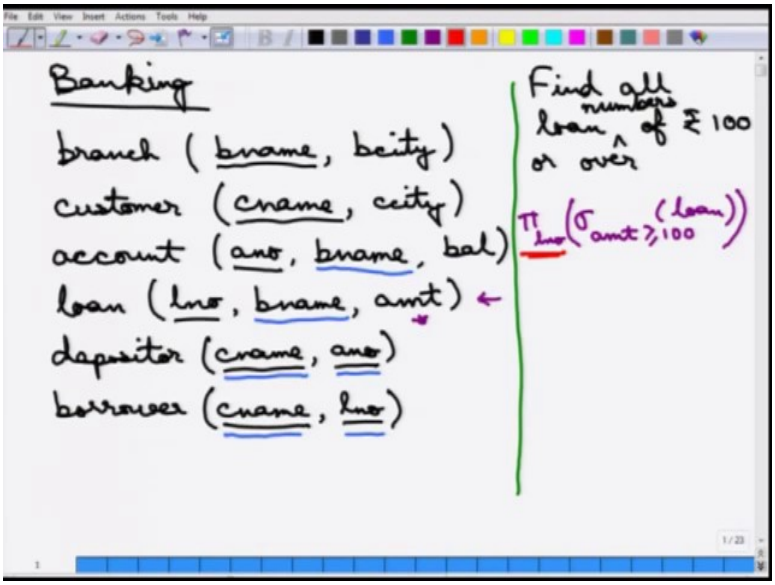


So, this is called a **Composition of Operators**, so the operators can be applied one after another and it has to be defined in what way the composition can take place. So, it essentially uses multiple operations. So, for example, suppose *r* is (A, B) and (1, 1), (1, 2). This is the same example that we saw earlier. And *s* is (C, D, E) with (1, 2, 7), (2, 6, 8) and (5, 7, 9). Now, suppose the operation that we are working on is this $\sigma_{A=C}r \times s$. Now this has to be understood in which way it is happening. So, this happens first this one is being done, this is number 1 operation then after that this is done, so this is number 2 operation. So, essentially what is being done is, initially $r \times s$ is produced. So, (A, B, C, D, E) that is being produced and that is the entire (1, 1 .. let me just write it down to make it complete .. 2, 6, 8), (1, 1, 5, 7, 9), (1, 2, 1, 2, 7), (1, 2, 2, 6, 8), (1, 2, 5, 7, 9). So, this is the $r \times s$ that is produced then the $\sigma_{A=C}$ is produced on the same thing.

So, the first one, the first $r \times s$ changes the schema, the second one does not change the schema and the second one produces the answer which is on $A = C$. So the first one is correct, this one is right, this one is wrong, this one is wrong, this one is again right. So, (1, 2, 1, 2, 7) this is wrong, this is wrong that is it, this is the final answer for the query of *r*. So, well that completes the set of basic operators, so we will go over the examples next.

So, let us work on with some of the examples for the six basic operators that we just saw.

(Refer Slide Time: 02:57)



So, here is a banking example that we will use many, many times, so let me just explain the example. So, there are six different relations in this, the first one is the *branch*, so the *branch* has the attributes, branch name, branch city. So, where is the branch name and which city it is in, the second one is the *customer* relationship, so all the customer. So, the customer name and let us say the customer city, so which city the customer is in.

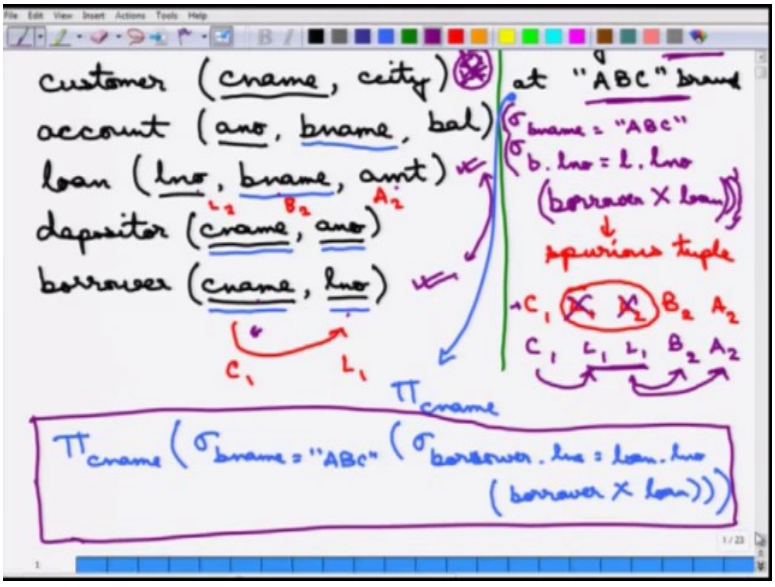
And by the way, I am underlining primary keys, so branch name is the primary key and customer name is the primary key. Then there is an *account*. So, the *account* is about all the accounts that are there, so the account name is the primary key and which branch it is in. So, the branch name, so branch name as you can see is a foreign key, and balance. So the foreign key let me highlight it using the blue color and then let us go to the next thing which is *loan*. Now, loan is about a same kind of thing like an account, but it is a loan number and a branch name and the amount of the loan. Once more, this is the primary key and this is the foreign key. Finally, there are two things which are, *depositor*, depositor has who is the customer that

took this customer name and which account number that is it. So, both are primary keys together both are primary keys and both are actually also foreign keys. We will come to nuances of this later. But, suppose and the last one is the *borrower*. borrower has a customer name, so, who borrowed what and which loan that is borrowed, So once more these are both primary keys as well as foreign keys.

So, with these things let us try to solve certain types of queries, so let me use this part to use this thing. So, the first query that we will try to solve is, find all loans of Rupees 100 or over. So, how do we solve it? So, this is first of all we need to understand that this is a query about a loan, so this is the loan table that we need to look into.

And essentially there is a amount attribute that we need to use and using that we can simply do a this thing. So, select all tuples from loan, where this amount is greater than equal to 100. That is it. So that solves this query, so find all loans of Rupees 100 or over. So, this returns everything loan numbers, so find all loan numbers of loans which has got rupees 100 or over. So, it is kind of the same thing, but except now what needs to be done, so it has to be the same kind of thing. So, you first find out all loans that are greater than 100 or over, but we only need the loan number. So, you project it on the loan number, that is it. This is the important part we just project it on the loan number. So, this is the important part that is the change from the previous query.

(Refer Slide Time: 06:47)



So, let us now move on to a little complicated query, so the query is the following. Find

names of all customers, having a loan at let us say ABC branch, so the branch name is ABC. So, now, the important part is to do that we need to find the names of all customers, so the first thing that we need to use is this table, the customer table. But, unfortunately the customer table by itself does not contain any information about the loan. So, for the loan we need to use this table, where the branch name is there. But again what happens is that, the loan table will contain the branch name, etc and we only need to find out the names of all the customers. So, instead of this table we may use this table, the borrower, because that contains the customer name. So, we can simply use instead of this table, we can use this table and we need to do whatever borrower and loan, so this is the Cartesian product that we need to take. So, if we take the borrower Cartesian product of loan we get the information of customer name, loan number, loan number, branch name and amount for all the possible loans.

But, this will generate what is called a *spurious tuple*, if we just do this Cartesian product this will generate what is called a spurious tuple, because what will happen is that. So, there will be some customer name corresponding to some loan number, let us say customer name C1, corresponding to loan number L1 and then there is a loan number L2 here with some let us say B2 and A2 this will generate tuples of the form (C1, L1, L2, B2, A2) which is not useful, because the loan numbers are different.

So, what we need to do is to ensure that this is the same loan number. So, for that what we will do is, instead of just doing this, what we will do is that we will select from this table everything, where the borrower dot loan number. So, let me just write it short hand $\sigma_{b.lno=l.lno}$, so this we need to select it on this table. So, this will then get rid of tuples like this and it will essentially only select tuples of the following form. So, this is what we require and this essentially means the customer C1 has the loan L1 and corresponding to the loan L1 this is in this branch B2 and has the amount A2. Now, we only need to find out the loans, where this is at ABC branch, so what we need to do is to do another selection on top of this. So, this is one part, this is another part, so then we need to do sigma over, everything where the branch name is ABC, so this you have to select on branch name is equal to ABC and that is applied over this entire thing, so this is what it is done, so this is the entire answer to this query.

So, if we now want to find names of all customers, then this is still not complete then we need to put in some more level here and let me just try to do it here. So, this is another. There are three brackets here, so this will be somewhere here, this we use to do a π_{cname}, because

we only need to do the customer name. So, essentially if we go down a little bit, this is a way to write down the entire query, this is

$$\pi_{cname}(\sigma_{bname="ABC"}(\sigma_{borrower.lno=loan.lno}(borrower \times loan)))$$

and let us complete that, borrower cross loan. So, this is the answer to the entire query that we were doing.

So, this is an example that shows you how quite a complicated query, like find names of all customers having a loans there are many, many things here can be broken down into small, small parts and then solved one at a time and we can do many more such queries. So, for example, we can use this to say well instead of having a loan at ABC branch find me all customers having a loan at ABC branch, but not any account.

So, what you then need to do is, you need to ensure that the customer does not have an account. So, you need to take the set difference from these depositors, so the depositor needs to be set difference done with that. So, we will leave this as an exercise to you, but the basic idea is, how to use only this six of these operators, the basic operators to solve different queries. So, if that is all done, so in the next module we will move into the additional operators.