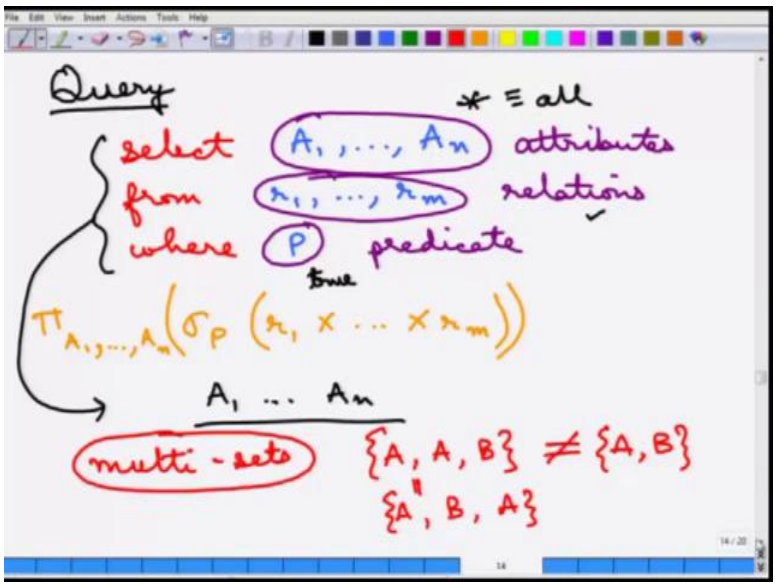
Fundamentals of Database Systems
Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 09
SQL: Basic Queries

The query in SQL is what it is useful for.

(Refer Slide Time: 00:11)



And the basic query and the typical basic query structure is of this very, very famous select, from, where. This is the very, very famous concept of select, from, where and what can you select, you can select a set of attributes and you can select from a set of tables and you can specify a set of and you can specify a predicate. Now, let us go over this one by one, these are attributes, these are relations and this is a predicate.

So, what does this mean, this means that we want to select from the Cartesian product of r 1, r 2 to r m which is we first do this r 1 to r m and only select those tuples, where this p is valid and we do not want to output all the attributes of them, we only want to output the attributes A 1 to A n, so that is the essential meaning of it. So, if we write down the corresponding relational algebra query, so this is the following thing, the first one that is done is that r 1.

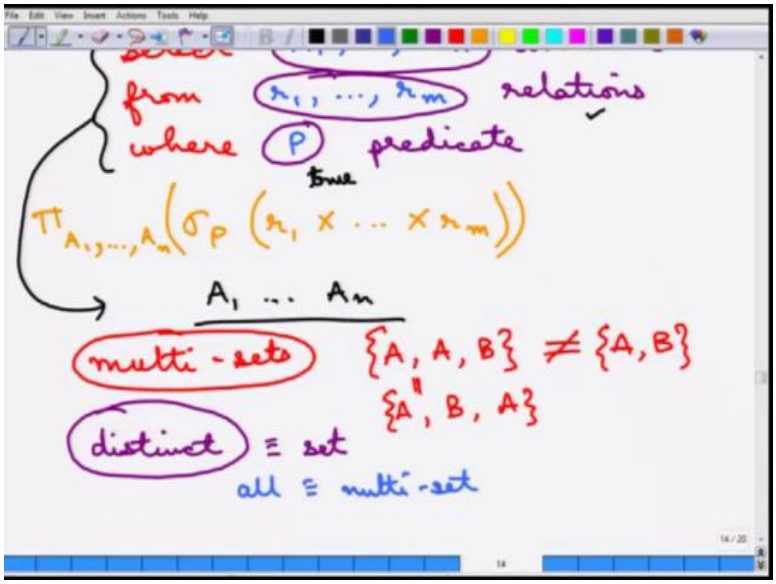
So, all the cross products is taken, then we apply the condition p on that and then we select out only these particular attributes, so that is the equivalent relational algebra query. So, now, a couple of things is that, so the result of schema of all of these is essentially A 1 A n. So, that is the schema it produces, so it produces a new table, where the names of those attributes are A 1 to A n number 1, number 2 if p is left out p is by default true.

So, if p is... So, you can if you leave out where then p is true, essentially everything is left and you can of course, select only one relation and then there is a special syntax for select the attribute, you can say select star, star means select all the attributes. So, this stands for all the attributes and p is one means it is true. Now, we have been saying that SQL is based on relational algebra etcetera, etcetera, but what is the big difference of SQL from relational algebra is that SQL is a multi set, SQL relations are multi sets this is a very important thing this is a multi sets.

So, what is a multi set, it is just like a set, but an element can repeat, so it is a bags of tuples. So, a particular tuple may come may show up more than one time in an SQL query, which is fine in relational algebra it is a set it is strictly a set. So, it does not let you select a tuple more than once, but in SQL by default it is a multi set, so it can select multiple tuples. So, a multi set example just to highlight the point again, so A, A B is a multi set, because although the element A is the same thing it is repeated and this is not equivalent to A, B, this is not the same as A, B.

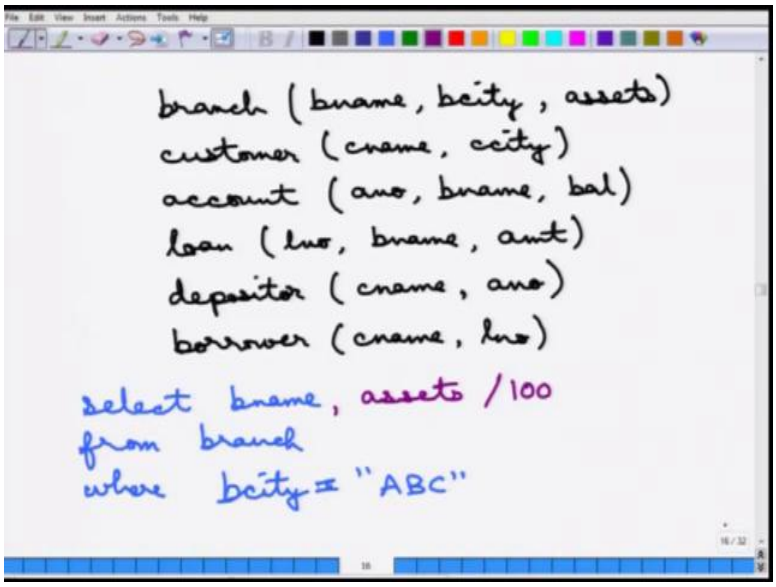
But of course, this is equivalent to A, B, A because the set order does not matter, so that is a multi set. So, if; however, the SQL can behave like a relational algebra by specifying this key word distinct.

(Refer Slide Time: 03:54)



So, if one says select distinct, this distinct essentially make it a set and not a multi set and the opposite of this distinct is all which makes it multi set. So, all is by default, so we will go over each of this constructs select from where in some detail next.

(Refer Slide Time: 04:23)



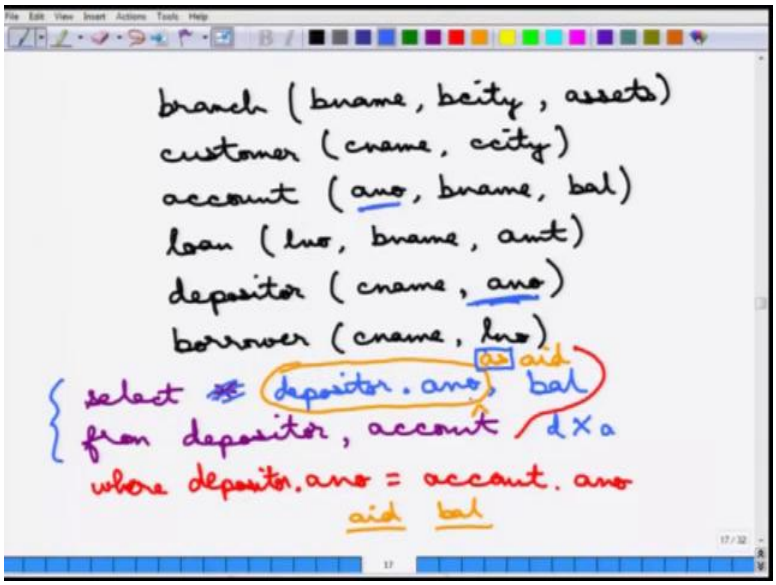
So, we will use the same banking example as we have been using earlier. So, a branch has a branch name, branch city and an additional attribute called assets, which is the total amount of all the assets in the branch. The customer has a name and a city, account has a account number which branch it is in and the corresponding balance and the loan has a

loan number, branch name and amount, the depositor is the customer name and account number and the borrower has a customer name and loan number.

So, let us go over some examples, so first of all suppose we want to select all branches, where we want to select all branches from the city A B C. So, how do we do that? So, very simply we want to select all branches, so we want to select branch name from the table branch, where the branch city is you can simply say A B C. So, this selects all the branches from the city A B C just to correct it, so this is branch city and this is an equality operator, which says that within a street what is a city.

Now, suppose we not do not just want the branch name we also want the assets, but we do not want the assets in terms of by saw or whatever suppose and we just want it in terms of some other arithmetical operation. So, we can simply say that we can say simply say assets by 100 or whatever it does not matter any arithmetic operation can be done on the select thing as well. So, this is one way of doing the select, but and the from is essentially from branch etcetera, etcetera and let us now go over to someone or more examples on how to select something from two tables.

(Refer Slide Time: 06:12)



So, suppose if we do this, if we simply write this select star from depositor comma account then what happens by the way this is one interesting point to note that the select and from are mandatory. But, the where can be omitted we have if the where is omitted it essentially means where true; that means, all the tuples are select. So, if we just say

select star from depositor account, this is essentially a Cartesian product of these two. So, this is a Cartesian product of depositor and account.

And essentially every tuple that is in depositor account, then all the values are selected. Now, what happens is that depositor and account both have this attribute name a number. So, how is it finally, shown how is it actually being represented is that, so the number what the point is that a number this is a clash of the name and this is ambiguous. So, if something needs to be done on one of the account numbers that needs to be specified.

So, for example, if we want only the depositor account number, so what we may need to do is instead of this what we will need to do is, you say select depositor dot. So, this is the dot operator dot a number and suppose, we also want to select the, whatever balance etcetera. So, depositor dot a number; that means, now we are only selecting the a number from the depositor column and by the way this if we just write this query, as you can see that this query this is the Cartesian product query, this is not very useful we would rather want the join that we saw earlier.

So, how do we specify the join, again the join is an equality join, so we have to simply say a number of account is equal to a number of depositor. So, we have to make you can simply say a number is equal to a number. Now, you can see that this does not make any sense, this is ambiguous, so what we need to say is we need to qualify this whenever there is an attribute that is repeated you need to qualify it with from which table it is coming.

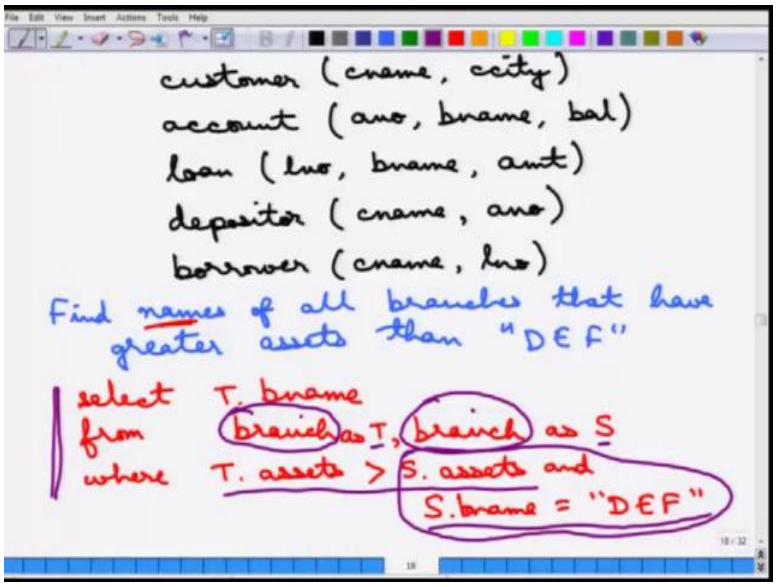
So, we need to say depositor dot a number is equal to account dot a number and that is the way to select this. So, that is the way things work, this is important this essentially is the way of doing the join and we will see the example of join later. And now what also happens is that, so couple of more things is that, this depositor dot a number this is a very clumsy name to use further. So, the SQL let us one rename operator, so you can say instead of depositor dot a number, you can add in depositor dot a number as let us say aid.

So, what the finally, what will be selected is aid and a balance table, so this is the table with aid and balance, because depositor dot a number is renamed. So, the s operator essentially is for renaming, this is the renaming operator as aid and even depositor can be renamed. So, we can write the above query in the following manner, a relation is

renamed it can be used directly here. So, again you can of course, have renamed depositor as d and just say d dot a number and d dot a number here and ((Refer Time: 09:50)).

So, that can be also done, so this is renaming, renaming is not just a way to reduce the name space, clash or etcetera is not just a way of convenience sometimes it is necessary and then here is an example, where it is necessary.

(Refer Slide Time: 10:05)



Now, these request to find branches from branches with name DEF, this is a little complicated query, but this can be broken up into the following manner. So, let me write down the answer to this query and then it will be clearer as to how this can be done. So, I am saying select you want to find the names of all branches. So, there is some b name that we require from of course, there is the branch that we require.

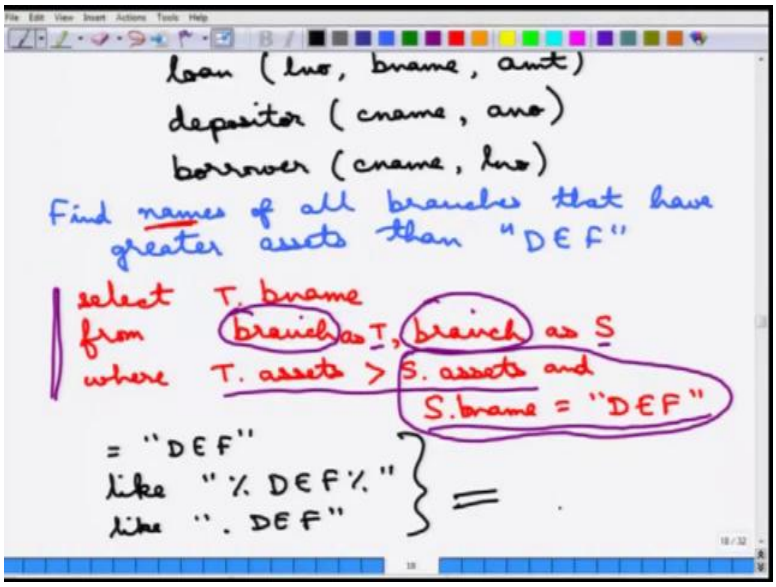
Now, the where part, so what do we want, we want the branch the b name of the branch whose assets is greater than the branch whose name is DEF. So, what do we do we require actually two branches here, because we need branches from the branch relation twice and now this is a problem. Now, both are branches, so how do we do that, so we cannot use both as branches we must rename.

So, let us say we rename the first branch as T and the second brand as S we want branch from T, where T dot assets this must be greater than S dot assets and S dot branch name

is equal to DEF. So, let us analyze this query first of all, so there are we are selecting the branch twice here and here, one is renamed as T, one is renamed as S and this is necessary if without renaming it cannot all.

Because, there are both are branches and we are saying that we want to select everything from this relation branch, such that it assets is greater than S, but which kind of S only those S whose branch name is DEF. So, essentially this to select the assets of the branch whose name is DEF and we want everything else, where that is greater and only selecting that this is an example of a little complicated query and where renaming is necessary and this also highlights the fact that the same name branch is used twice. That is why the renaming is necessary, just to now that we are inside this, this branch name is equal to there are certain kinds of string operations that we can define.

(Refer Slide Time: 12:36)



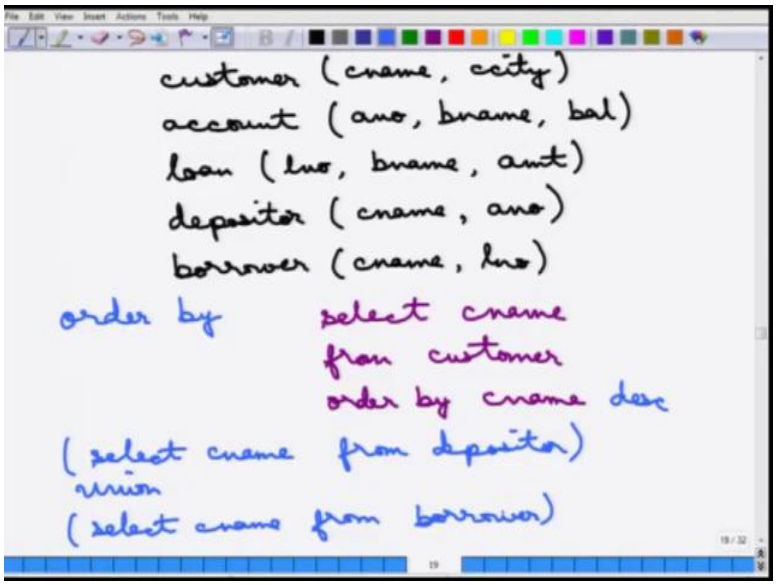
So, we have been seeing this equality operator is equal to DEF, this means this matches exactly DEF, you can also say like DEF. So, wherever this is like a percentage, so the percentage essentially is substring. So, everywhere where the name DEF is there it will be selected and there are many such things and so you can say a dot, so you can say like dot DEF. So, dot stands for a particular character and there is a regular expression thing that can be done here, so this can be checked up.

So, one important thing that we have been studying about relational algebra and SQL is that these are sets or multi sets. So, let us stick with SQL these are multi sets, so; that

means, by default there is no ordering of the tuples. Now, which is true, but sometimes the output needs to be in a particular order, now once more this is an important point to remember is that we the SQL let us you run a query and let us you see the output of this. So, now, how to actually visualize the output, the output can be in any order that the database engine wants,.

But, suppose you want the output to be in a particular order, so you can actually sort, but do remember the sorting is done after all the tuples have been selected and this is just a sorting is done only for the output of this, this is got no actual meaning in the relational algebra model or SQL model. So, how do we do over the ordering?

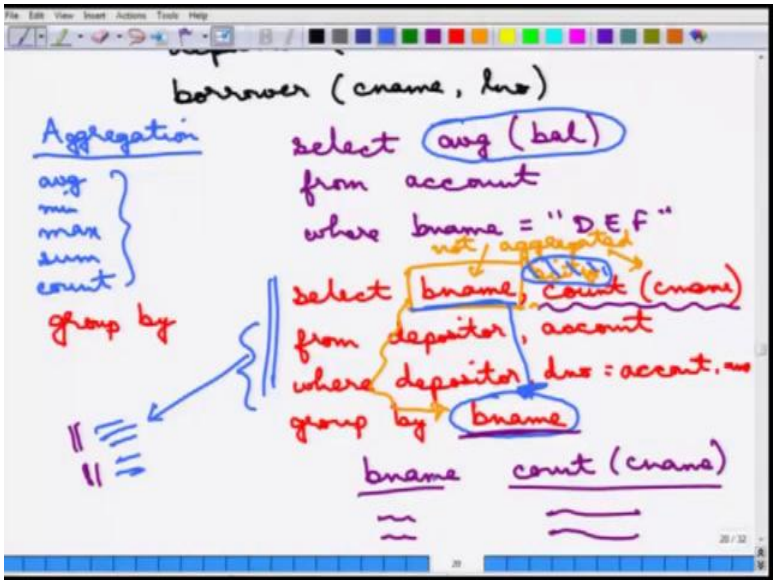
(Refer Slide Time: 14:15)



So, you simply use there is a construct called order by, so there is a construct order by which let us you order the tuples. So, you can say select c name from customer order by c name whatever. So, this is the cname will be now in order in ascending order by default, if you want to do something else, you can say descending. So, by default this is ascending you can also add descending to make it descending order.

So, this is one kind of operation, the other kind of operations is set union etcetera can be done and you can also say distinct, etcetera all of these can be done, so this is the set operation. The next important thing that what we want to do is aggregation, so the aggregation can be done.

(Refer Slide Time: 15:12)



So, again the aggregation is being done, so the aggregation, so five operations are allowed to be used there are average min, max, sum and count. So, these are the five aggregation operations that can be done and a query something like this can be specified. So, select average balance from let us say, so we would select the average balance from account, where let us say branch name is equal to DEF. So, essentially it selects the average balance of all the accounts in the branch DEF, so this is an aggregation operation that is being done on the balance.

Now, as we saw earlier it is not always useful to apply the aggregation on the complete set of tuples. But, on certain groups of tuples, so how does the grouping done, the SQL offers a way of grouping the tuples which is essentially using the construct group by. So, there is a construct group by this let us you group by. So, again we can do the following thing, so we can say account this thing or also let us write down another query. So, what are we doing here, this is important to understand is that.

So, what are we trying to do here is the following is that we are doing certain query, but we are saying group by a branch name. So, whatever is the output up to this point from depositor account etcetera. So, we are essentially selecting all the accounts and on the depositor information around it, we are grouping by b name. So, all of those tuples that are result as this are then grouped into different kinds of things according to how whether they have the same branch name or not.

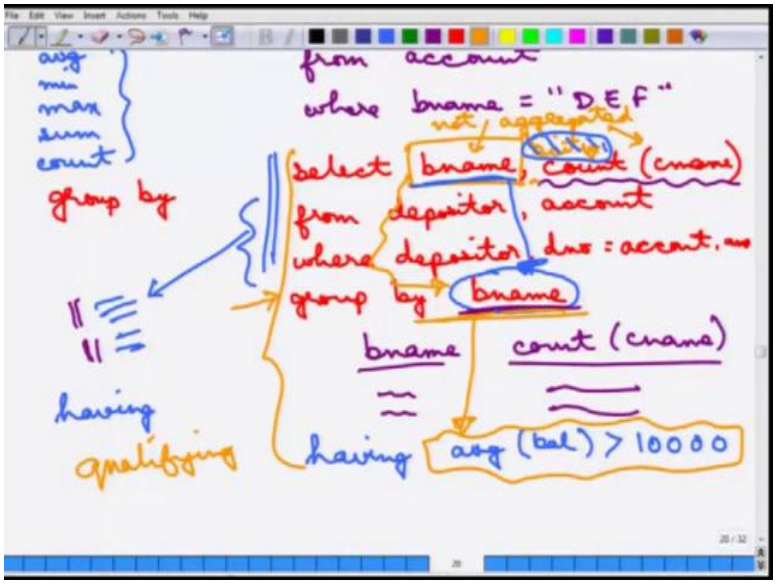
Now, once they have the same branch name for each of this branch name we are applying some aggregate operation, which is a count of c name. So, the answer to this query is looks like is of the following form is of the branch name and count of c name. So, for each of the branch names we will have a count of how many customers are there that is the thing. So, this is the same as the relational algebra that we talked about the relational algebra grouping and aggregation that we talk about.

Now, one important thing is that the attributes in the select clause that are not aggregated. So, this is our attribute in the select clause that are not aggregated, so this is not aggregated and this is aggregated, so this is not aggregated they must appear they must be grouped. So, they must appear in the group by; otherwise, which has got no meaning. So, for example, we cannot select branch name and let us see we cannot select branch name and branch city on the thing.

So, this branch city would have been wrong, because there is no way to get the branch city. Once you have selected by branch name there is no way to get to the branch city or once you have selected... So, it does not make any sense, so whatever is not aggregated in the select clause must be present in the group by. So, this is mandatory; otherwise, if you just say select branch name, branch city this thing then it is wrong. So, this is wrong you cannot do this, this is one important thing to remember.

So, then the group by now you can say this is group by, but you can also qualify the groups and how do you qualify the groups, you can qualify the groups. So, you can say I want to select particular groups.

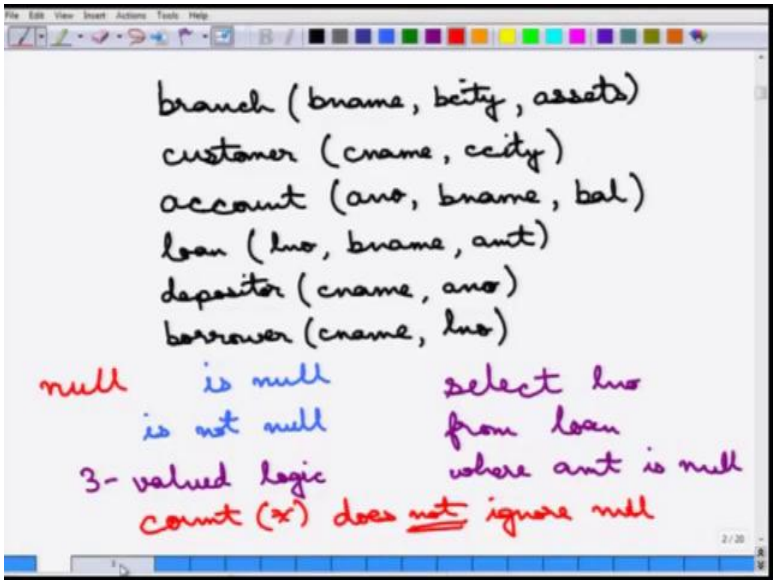
(Refer Slide Time: 18:59)



So, the qualifying is done by the having clause, so instead of saying select branch name and count c name from everything etcetera, etcetera you can say I want to select the branches, where the average balance or whatever, the average balance is at least some 1000 rupees or something like that. So, what you will do is, you will write down the same query group by etcetera and in the end you will add having average balance greater than whatever 10,000 rupees or something like that.

So, this four things let me highlight this four things to this five things together is the query. So, select branch name count c name from depositor account, where depositor dot d name is equal to account dot a number, group by branch name having average balance greater than 10,000. So, you are not going to do the group by on every branch name, you are going only going to do the group by on those branch names, where the average balance is greater than 1000. So, this is like qualifying a particular group, so this is useful, because sometimes you want to analyze only certain groups which are useful to your analysis.

(Refer Slide Time: 20:20)



So, we will next consider the issue of null in SQL a value can be checked whether it is null or not by this following construct is null or is not null an example query is of the following form select l number from loan, where amount is null. So; that means, it will try to find out the loans, where the amount there is some problem with the amount it has not been updated correctly etcetera. Now, for null again the same issue as the relational algebra happens, so this does follow that same three valued logic and evaluation.

So, result of the expressions involving null evaluate to null and comparison with null returns unknown plus the aggregate functions ignore null. So, for example, average etcetera will ignore null, but the only difference is the count star does not ignore null, every other aggregate operation ignores null. So, we will next move on to a very important part of SQL, which is called a nested sub query.