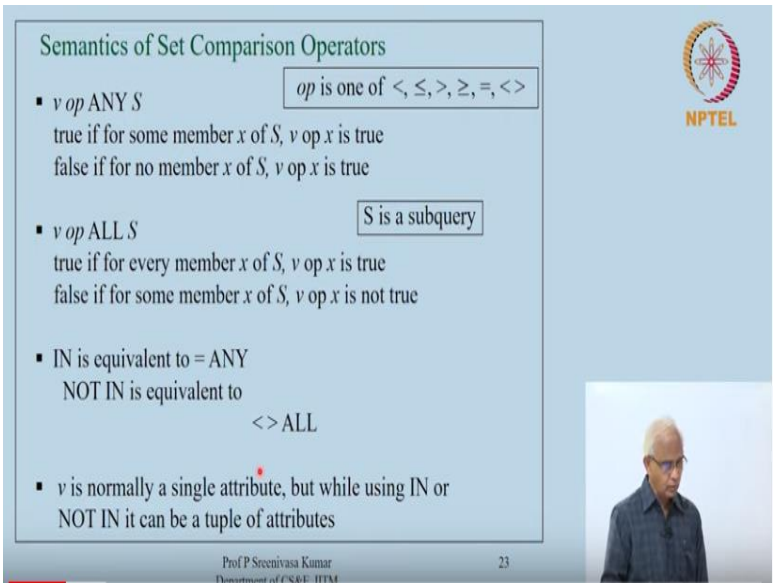
Database Systems
Dr. Sreenivasa Kumar
Department of Computer Science and Engineering
Indian Institute of Technology – Madras

Lecture – 15
Correlated Subqueries

(Refer Slide Time: 00:23)



Towards the end of last lecture, I was talking to you about how in an SQL query, we will be making use of what are called sub-queries or nested queries. Basically, this feature of having nested queries will allow us to break down the computation, break down the specification into logical units and then specify them using independent queries and then make use of the results of a sub-query in the outer query.

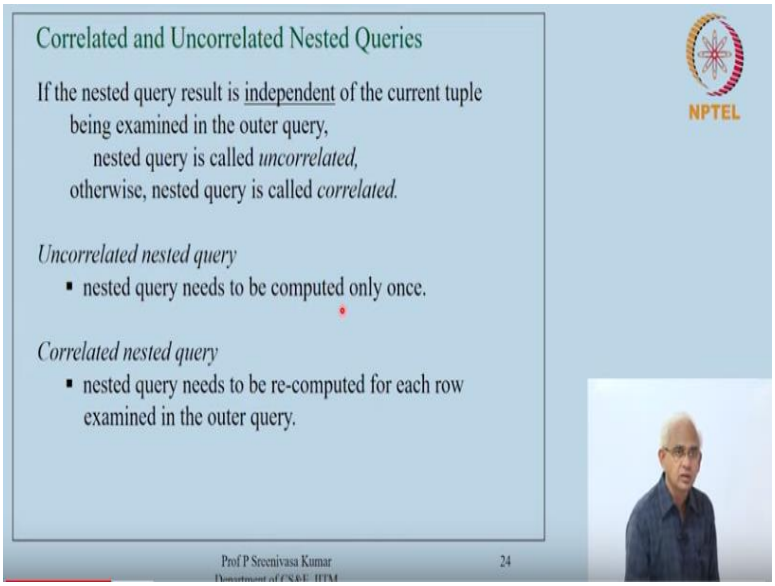
So, while doing that we needed operators for comparing or making use of the results of the sub-query. So, we have talked about this in the last class. These are the kind of operators that we make use of the results of a sub-query. If S is a sub-query, then we can put in this kind of an expression, v op ANY S, and we have seen what are the various operators that are allowed and what are the meanings of all these operators.

So, in a short form for =ANY, NOT IN is equal to <>ALL, and in general, we can also have the v, which is generated, which is kind of produced by the main query. It is possible that this particular v, also a tuple, in which case, the S is generating compatible tuples. So, v is

normally a single attribute, but while using this IN and NOT IN, it can be a tuple of attributes, in which case, the same kind of semantics will apply.

So, basically if there is a tuple and if we are using ANY, then we are basically there is exactly equal tuple in the result produced by S, similarly for Not Equal To. So, moving on in this lecture, I want to focus on a few more features of SQL, the query part of SQL.

(Refer Slide Time: 03:13)



There is a notion called correlated or uncorrelated nested queries. If the nested query result is independent of the current tuple being examined in the outer query. So, this requires some explanation. What do you mean by current tuple being examined in the outer query. Recall that, the meaning of the basic block is that at any point of time, you know, we are considering a particular combination of tuples of relations that are mentioned in the FROM class.

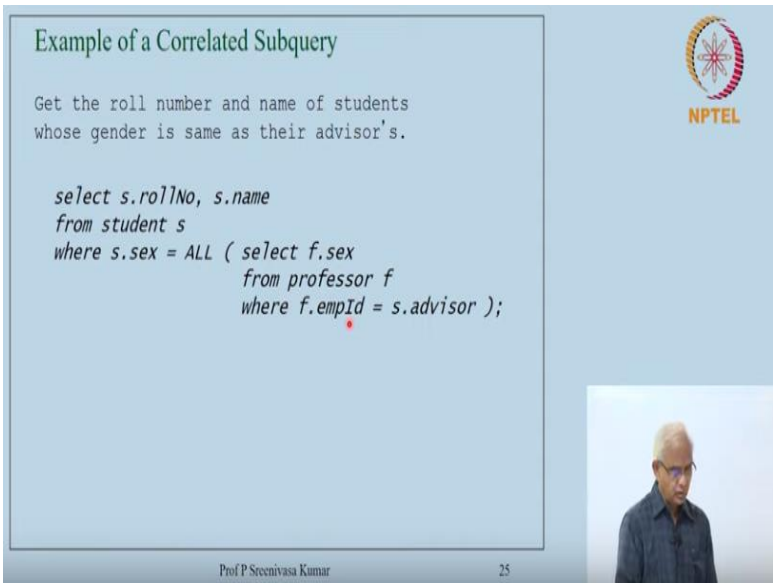
So that is the current tuple under consideration. So if the result is a nested query is independent, you know, it does not care about the current tuple being examined in the outer query, then such a query is what is called an uncorrelated query, whereas if this is not the case, we call such a nested query as a correlated query. Its result is in some sense correlated with the current tuple being considered in the main query.

I will give examples of these things and then you can try to understand. If it is an uncorrelated query, then the advantage we have is that the nested query can be computed only once and its result can be made use of in the main query. Whereas, if it is a correlated nested

query, then the nested query needs to be recomputed for each of the tuple combinations. You can also call it a row being examined in the outer query.

So, this difference is of great value when we are actually looking at the efficiency of running of the queries. So, if you detect that, the nested query is an uncorrelated query, then we can actually keep the computation, run it, take its result, store it somewhere and reuse it. So, lets now start looking at the example of this.

(Refer Slide Time: 05:55)



So, I will keep the examples simple, you know, kind of illustrate the main point we are looking at. So, get the roll number and name of students whose gender is same as their advisor's gender. You may wonder, why you want this information, but just for illustration purpose. So basically, I am now looking at whether, you can see that as I am kind of looking at various student tuples, I must then make use of the value of the sex attribute and then check with the advisor's attributes appropriately.

So that is where the correlation comes into picture. So, this can be specified easily like this, select the roll number name from the student. Here you can see that, we are using s. advisor. So this sub-query, which is checking whether the gender of the professor is same as the gender of the student whom he is advising or she is advising. He is making use of the table alias that is there in the outer query. That is the only way you can check that condition.

So this also brings into the focus that this table alias that is there in the outer query. That is the only way you can check that condition. So, this also brings into the focus that this table

alias that we introduced in the FROM class are accessible to the sub-queries. They are accessible and they can be made use of in the sub-queries. So, in the sub-query now, I am doing select f. sex from professor f, where f. empID is the s. advisor.

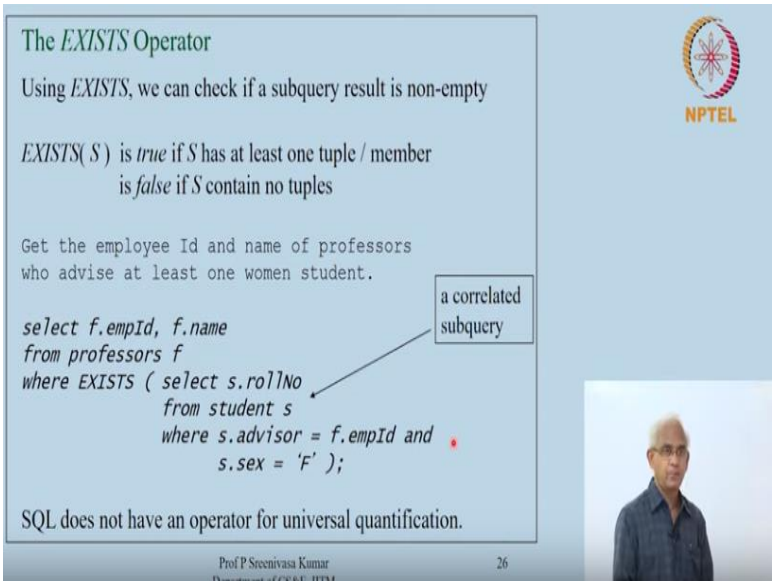
That means, this particular professor is the advisor for the current student under consideration. So, your question is if this nested query, what does it generate. It generates 1 tuple, so in which case is = ALL is required are not. So, it does not matter right even if you = ALL or = ANY, it would be the same actually. The effect will be the same. So, you could actually replace it by ANY or you could also just use IN. So, this is basically producing the one value and then we are checking whether this particular thing.

The main point I am trying to illustrate here is that, this sub-query is correlated because its value or whatever it produces depends on the current student under consideration. So, compare this with the query that we had where we said, get whole of all student advisors who are female professors, and then we used a sub-query to figure out who are all the female professors and then checking whether the advisor or the student, the employee ID is present in the employee IDs of the female professors.

So that sub-query, you know, kind of independently run without bothering about what is the current student I am considering, because that sub-query was basically just generating the empID of all female professors. So that is an example of uncorrelated query whereas this one is formulated and in this kind of formulation, this sub-query result will depend on the current tuple being considered for student, so it is an example of the correlated query.

So, that is the difference between the correlated query. We will see some more examples of this correlated query as we go along. Now, let me take you to a very interesting operator called the EXISTS operator.

(Refer Slide Time: 11:12)



So, in a SQL, there exists an operator called EXISTS, and so what is this operator. It basically allows you to check whether a particular sub-query is producing non-empty results or producing empty results. So, that is all it does, but it is very useful and in some sense plays the role of existential quantification, but it is very useful in some sense, you know, plays the role of a existential quantification in SQL.

So, the EXISTS operator is expressed like this EXISTS(S) where S is a sub-query, but usually it has to be used inside. This now becomes a new predicate thing. So, you can use it in a WHERE class of a query, and so S becomes a sub-query of that particular query. So, this EXISTS(S) is true if S has at least one tuple and is false if S contains no tuples.

It is very simple and we will start using it now. Get the employee Id and name of professors who advise at least one women student. So, we will try to get a sub-query of all the women advisees of this current professor under consideration and then check whether that particular set is empty or not. That is the idea of trying to use an EXISTS operator. So, select f. empId, f. name from professor f and now here comes the sub-query.

What is this sub-query doing, go through this select s. rollNo from student S where the student advisor is the current professor under consideration (f.empId) and this student is a woman. So, essentially what it is doing, it is producing the set of roll number of woman advisees of the current professor and then we are checking whether is this set empty or not.

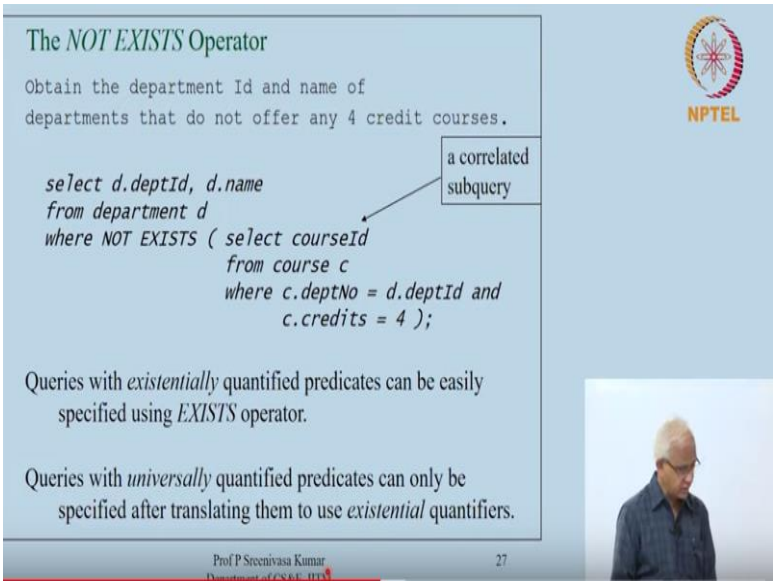
If this set is empty, then this particular empId is not advising at least one woman student and so should not be produced in the output and so if this thing is null set, then EXISTS null set would be false and so for that professor this predicate will be false and you will not be producing that professor in the output and if there is at least one member here in the result of this sub-query, then EXISTS for this s is true and that professor will be listed as a professor who is advising at least one woman student.

In all these examples, you must remember one thing that, I am using them to illustrate one of these features or operators that are there in SQL, you can also express the same queries in other ways. Every query can be actually be expressed in many ways and I am choosing the way of expression using the features that I am introducing and it should not be assumed that this is the only way to express the query.

So, you can express the same query in multiple ways. Remember that. Then, let's move on, this happens to be under correlated query because the result of the sub-query will depend on the current professor under consideration. Unfortunately, while this EXISTS is something like an existential quantifier, SQL does not have an operator for universal quantification. It is really unfortunate that we do not have a universal quantification operation in SQL.

There are other query languages like object query language and x- query and little bit, you know, query languages for more advanced data models, in which at the design stage itself, they have a kind of incorporated the universal quantification because it is a really useful quantification. So, as a result of this, while dealing with these queries that do involve universal quantification, we will be kind of forced to convert that into existential quantification, and then use EXISTS operator to check the conditions that we are looking for. So, let me illustrate that point.

(Refer Slide Time: 17:08)



Before we go further, here is a NOT EXISTS operator, so if EXISTS operator exists, NOT EXISTS will not be existing. Obtain the department Id and name of departments that do not offer any 4 credit courses. So, you can basically do this query by asking whether the department are producing the set of departments that are offering 4 credit courses and then checking.

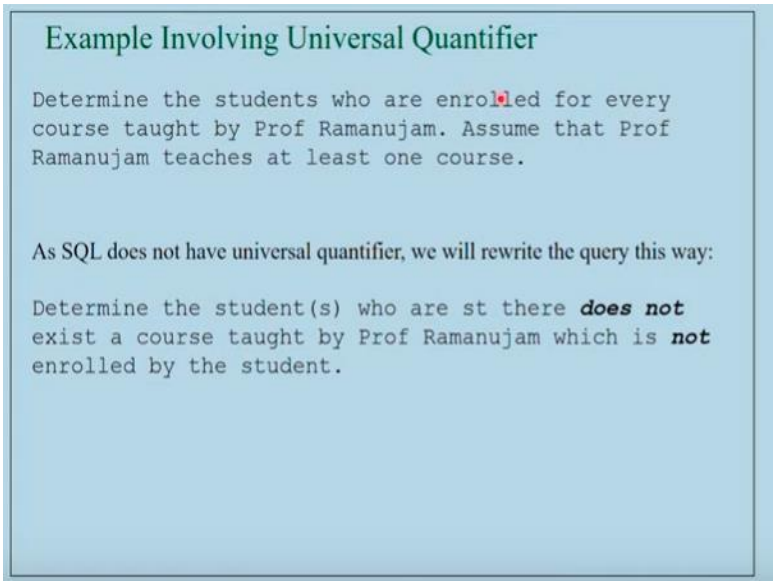
Let me show you the query. Department Id and name of departments that do not offer any 4 credit courses. So, here is a sub-query, what it does, is produce the 4 credit courses that are being offered by the department under consideration. So, how does it do it. Select courseId from course c where c. deptNo equals d. deptId. This will kind of ensure that the course is being offered by the current department under consideration and further check that the credit is 4.

So, this produce the set of courses, there are 4 credits and are offered by the department under consideration. What is the department under consideration? It is this d. We will consider all possible values for d right. So for each of these d, we are kind of computing what is the set of 4 credit courses offered by that particular department. If this is a set, which is non-empty set, then that department does not qualify for us because we are asking for the departments that do not offer any 4 credit courses.

So, that is why we put a NOT here and use a NOT EXISTS to get this condition that they do not offer any 4 credit courses. So, this also happens to be a correlated sub-query. So, both EXISTS and NOT EXISTS are very useful operators and most of the queries involving existentially quantified predicates can be kind of easily specified using this EXISTS operator,

but queries with universally quantified predicates can only be specified after we actually convert that predicate into something which uses existential quantifiers. So let me illustrate that. This is a very interesting part of the queries.

(Refer Slide Time: 20:42)

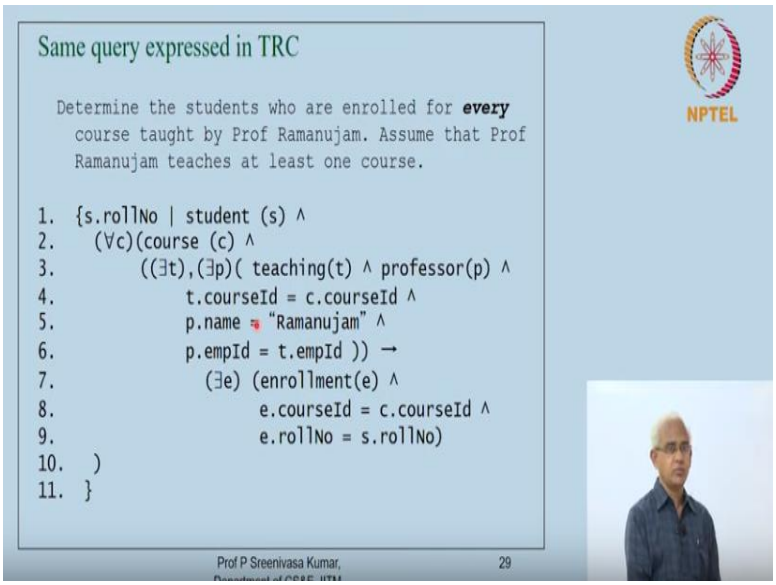


We have looked at this query earlier also when we were talking about tuple relational calculus. You have not looked at tuple relational calculus. This is the example I was discussing when the power went off. So, determine the students who are enrolled for every course taught by Prof. Ramanujam and assume that Prof. Ramanujam teaches at least one course.

So, why do you think that it involves a universal quantifier, because we are specifying that these students whom we are interested in are enrolled for every course taught by Prof. Ramanujam. So, one way of handling this, I mean, the way to handle this since SQL does not have the universal quantifier, we will rewrite this into something like this. Determine the students who are such that there does not exist a course taught by Prof. Ramanujam which is not enrolled by the student.

So, which semantically means the same thing that the student is enrolled for all courses taught by Prof. Ramanujam. Determine the student who are such that there does not exist any course that is taught by Prof. Ramanujam which this particular student has not enrolled into. So you can see that this is equivalent to the condition asked to check in this query and now since it uses EXISTS, we can express this in SQL. Before I move on to SQL, I will show you how we did this in double relational calculus. Just recall that slide here.

(Refer Slide Time: 23:42)



Same query in TRC, we did it like this. I want to go through this, I explained this thing earlier. So essentially what we are doing here is use this implication and universal quantifier and express this condition what are looking for is that for all courses taught by prof. Ramanujam, the student under consideration has done that. So, this translation is somewhat easier and direct translation of the condition.

So, every course taught by Prof. Ramanujam has been done that. So this translation is somewhat easier and direct translation of the condition that. So every course taught by Prof. Ramanujam has been done by. The way we express this, if the antecedent of this implication checks this is the course taught by Ramanujam and the consequent checks whether the student has enrolled for it.

And this is the universal quantifier for all courses, we are checking that if the course is taught by Ramanujam, then the student has enrolled in that. So, this you can is a more direct way of translating the query. Fortunately, we could do that because we have universal quantification in our hand, and of course, implication also is present and so we could do that. Now, take it as an exercise and then write down the

Where clause part of this predicate logic and put two negation symbols in front of it. If you put two negation symbols in front of it, it is logically equivalent to the same thing of that expression and then push one of the negations inside then see that you will get an equivalent expression that uses only existential quantifiers. So, let me now show you how you can deal

with this in SQL. Determine the students who are enrolled for every course taught by Prof. Ramanujam.

So, pass this slowly. There is a double negation, so it will be a little bit confusing, but that is inevitable. The logic that we are encoding here in the condition is that, there does not exist any course that is taught by Prof. Ramanujam, which is not enrolled by the student under consideration. So, select roll number, name from student s and something does not exist, what is that. Course taught by Prof. Ramanujam, which is such that the student is not enrolled for it.

Such a course does not exist. So that is easy to express, so select t.*. * is basically saying that get me all attributes. From teaching t, professor p, recall that this teaching information, who is teaching, what is available in the teaching relation and so we get the appropriate combination of teaching tuple and the professor tuple by doing the employee IDs is same in both the tuples, make sure that, and the name of the professor is Ramanujam.

So this will kind of fix the combination of the teaching and professor tuples, appropriate combination of teaching tuples such that the course is being taught by Prof. Ramanujam and then we are checking that in the enrollment tuples, e. courseId is same as t. courseId , that this is the course that is being taught by Ramanujam and the student has enrollment. There is NOT EXISTS here, remember that.

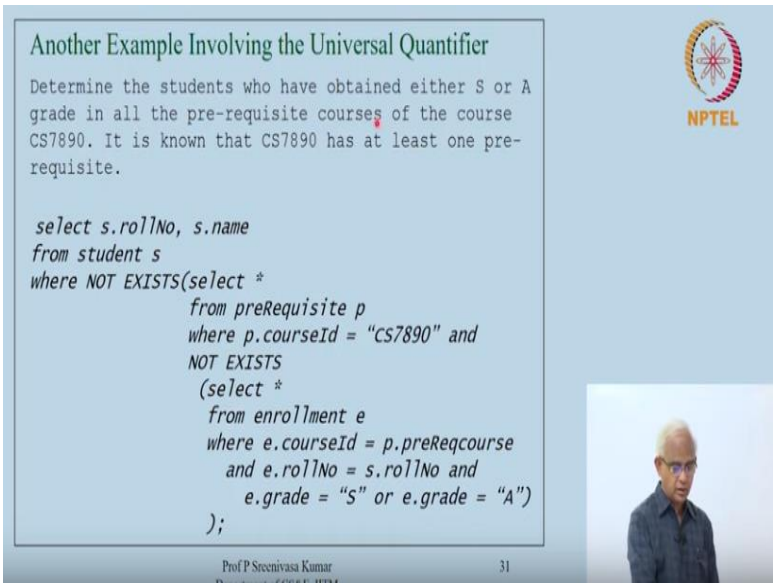
In such an enrollment tuple where the combination of roll number, s. roll number is the roll number of this particular student under consideration. Remember that, there is no other S anywhere here. The course of this the entire sub-query. Remember again, what is that we are checking, so under this sub-query, what we should get, a course taught by Ramanujam, which is not enrolled by the student. There does not exist such a course, that is the job of this outer NOT EXISTS.

So this sub-query is checking for the set of all courses taught by Ramanujam, which are not enrolled by the student. That is exactly what we are doing here. So, we are checking that enrollment tuple corresponding to the student under consideration, with the corresponding courseId does not exist. Okay any question here, because this is what we have to pass on. What I advise you to do is to kind of, you know, take some sample values of courses taught

by Prof. Ramanujam, some two or three of them and then run through this using a paper-pencil simulation and then figure out that.

So you can take the combinations of student doing some subset of the courses and student doing all of the courses. Then you figure out that this indeed will ensure that the student is actually enrolled for every course that is taught by Prof. Ramanujam. Now, let me show you a similar kind of a query, which again involves universal quantification.

(Refer Slide Time: 31:15)



Determine the students who have obtained either S or A grade in all the pre-requisite courses of the course CS7890. It is known that CS7890 has at least one pre-requisite. Again, this also involves checking some kind of condition for all the pre-requisites of a particular course. I will leave it this as an exercise for you to figure out that the same kind of a translation into existential quantification is required here and then you can express it like this.

We are asserting that there does not exist any pre-requisite of CS7890 such that the student under consideration has not got either S or A grade in such a course. So that is what this part of the query is doing. This query is ensuring that the course under consideration p is a prerequisite of CS7890 and this part of the query is checking whether the student has got S or A and there is a NOT EXISTS here.

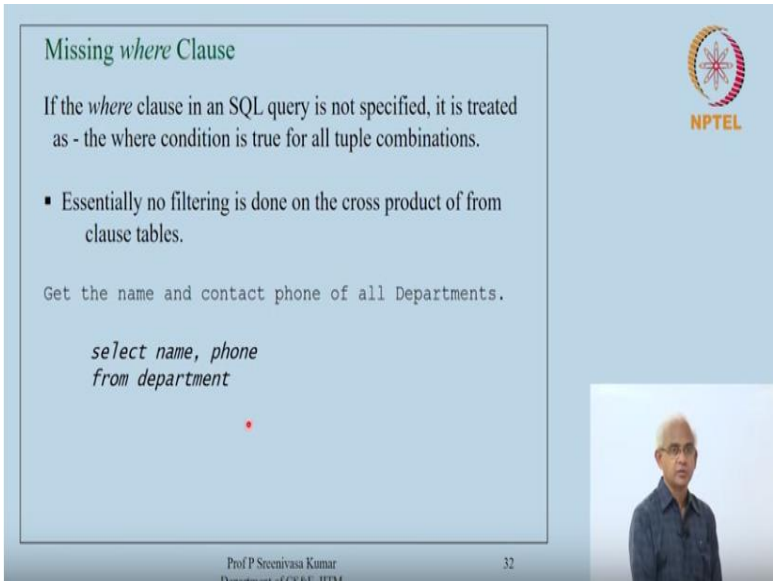
So, the way you can convince yourself about is to assume that say the course has some three pre-requisites, C1, C2, C3 and then assume that some student has done two of them, has got S or A, two of them and then what happens to the student. You can run through this and then

you will be able to figure out that student will not qualify. So the point I am trying to illustrate is basically that for those kind of queries that do involve checking conditions, kind of involve the universal quantifier, you re-write the X condition essentially using double negation.

And then use the EXISTS operator to express that opposite condition and I have given two of the queries, so that you can get this point across and I urge you to practice kind of such queries. So, you can also study this query and then come up with any questions that you may have in the next class so that we can clarify it. It is important to understand how to express these kind of condition SQL.

Of course, my advice is for you to try and express this in tuple relational calculus first, so that you kind of ready with the condition you want to express and convert into SQL. So that is about this aspect of the EXISTS operator and the way it helps us expressing in both expressing conditions that are involving existential as well as universal quantifiers.

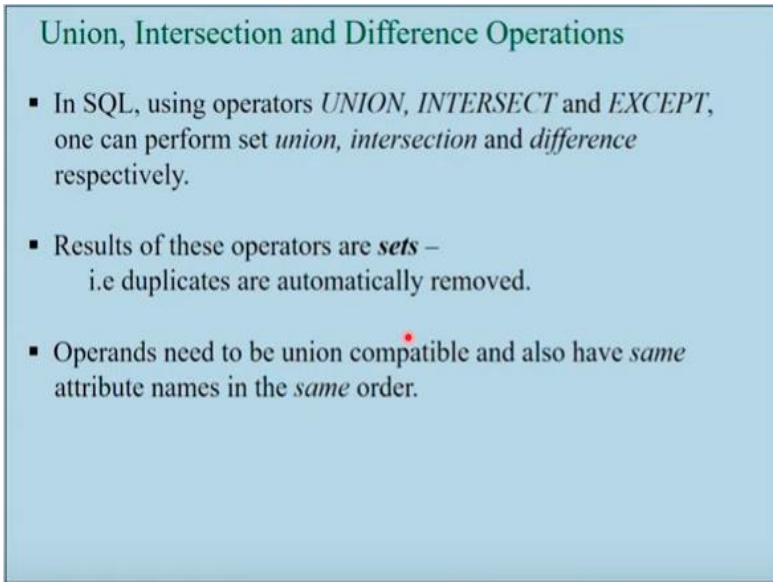
(Refer Slide Time: 35:45)



So moving on, the WHERE clause can sometimes be missing in the SQL query. If that is the case, essentially you will take it as a condition that is always true. So, in the interpretation in the basic block of SQL, the WHERE is a kind of a filter. For all those combinations, in which the WHERE condition is true, we are going to use that to produce the result. So if the WHERE clause is missing, then what we will do I, assume that all combinations satisfy the condition. So, it is a kind of true condition.

Essentially, no filtering is done on the cross product of FROM clause relations. So, a simple example of that could be get the name and contact phone numbers of all departments. So select name from department. You do not have to put any condition here.

(Refer Slide Time: 36:45)

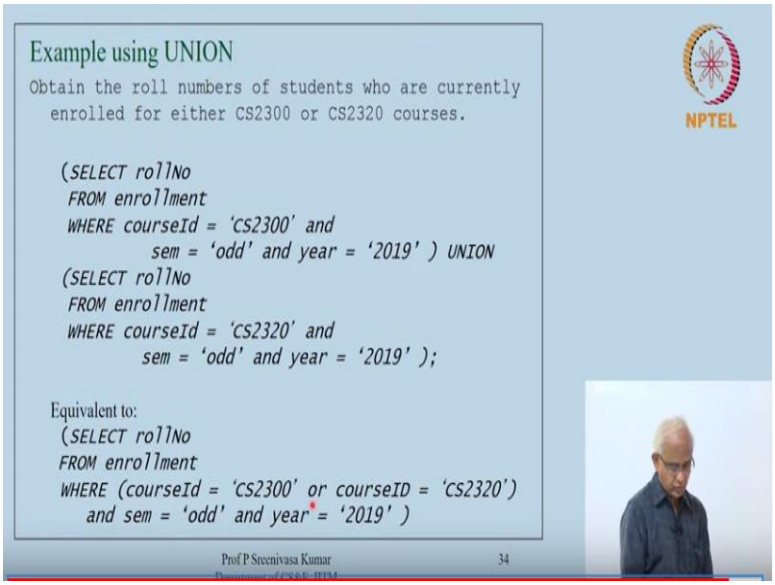


The next set of operators actually have to do with this Union, Intersection, and Difference operations. So, these operators can be made use of in SQL as well. Essentially, what we need here is that the two operands for these have to be Union compatible. Recall this notion of UNION compatibility, we have used it in relational algebra.

So essentially, the results of two sub-queries can be UNION, or you can use them for INTERSECT, and things like that, but then, both the sub-queries should produce results that are UNION compatible. Not only that, here we make even stronger assumption, that there should have the same attribute names in the same order in both the operands. This is one aspect. The other aspect is that the results of these operators are sets actually.

It may be slightly confusing, but these are sets, that means, duplicates will not be there. So, in some tuple exists in A and also exists in B, then A UNION B will have only one copy of that tuple. It will not have duplicate copies. So, it produces sets. Let me quickly show you some simple examples for illustrating this and then we will close the lecture.

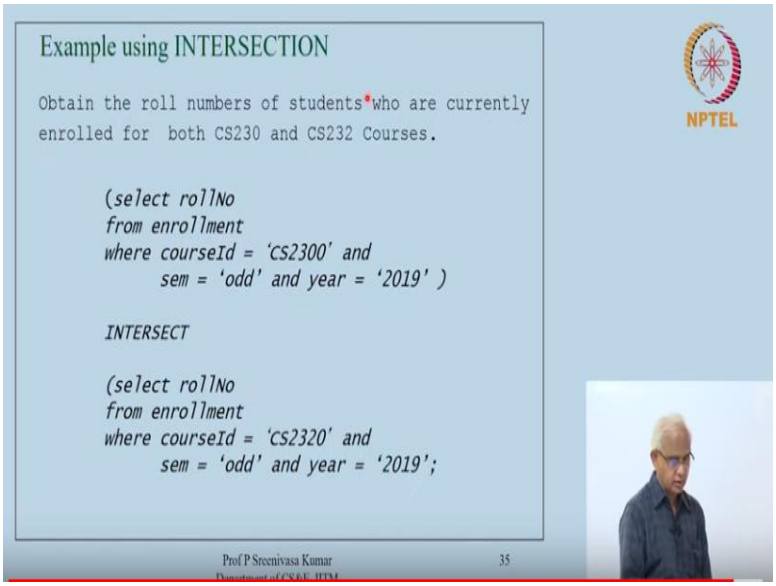
(Refer Slide Time: 38:47)



Obtain the roll numbers of students who are currently enrolled for either CS2300 or CS2320 courses. Obviously, you can just do it by putting these two condition in the WHERE clause, we kind of illustrate how to use UNION. Current semester of 2019 and SELECT rollNo from the enrollment where the course Id is this, it is the keyword UNION and the other thing is exactly the same.

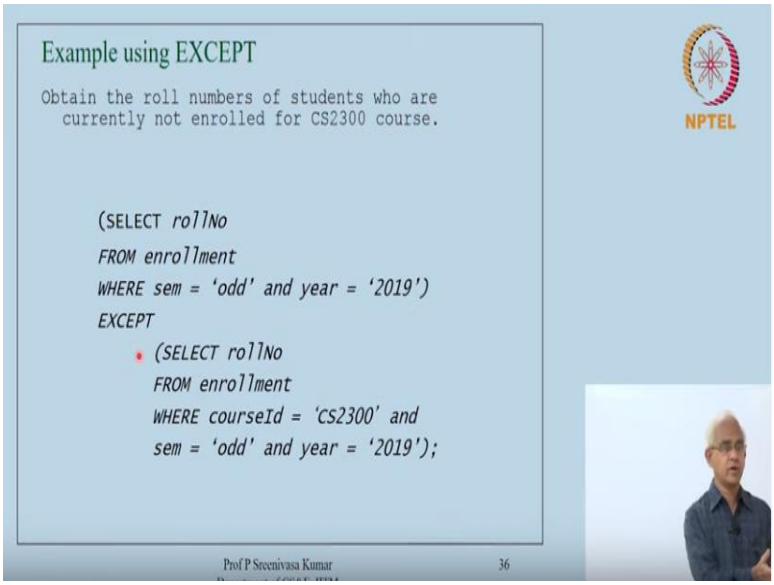
But this course is replaced by CS2320. Of course, is kind of equivalent to, you can put this or here in the condition. So the same example I have used in order to illustrate INTERSECT and also their difference.

(Refer Slide Time: 39:40)



So, I will skip this.

(Refer Slide Time: 39:48)



EXCEPT. So the keyword used for difference is EXCEPT. So this EXCEPT this. So, here is query. Obtain the roll numbers of students who are currently not enrolled for CS2300 course. So, you can get all students who are currently enrolled and then subtract the students who are currently enrolled for CS2300. So, with that we will close for today, and I will continue this discussion on SQL by bringing in what are called aggregation operators in the next class.

It is going to be very interesting, because they will be used for data analysis and other things, so we will see how we can express aggregations, but remember one thing, please keep practicing this stuff as we talk and look at more examples and express. You already have some set of queries with you and you can take queries from exercises, express them and clarify it.