

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

Lecture-17
Views

Ok so we will start off with views, we have discussed this views, the concept of views long back when we were talking about the introduction to databases in general.

(Refer Slide Time: 00:34)

Views (or Virtual Tables)

- Views provide virtual relations which may contain data spread across different tables. Used by applications.
 - simplified query formulations
 - data hiding
 - a view of frequently used data – efficient query answering
- Once created, a view is always kept *up-to-date* by the RDBMS
- View is not part of conceptual schema
 - created to give a user group, concerned with a certain aspect of the information system, their *view* of the system
- View implementation
 - Views need not be stored as permanent tables
 - They can be created on-the-fly whenever needed or
 - They can also be *materialized* and kept up-to-date
- Tables involved in the view definition – base tables

NPTEL

Prof P Sreenivasa Kumar
Department of CSE, IITM

51

And we looked at the 3 schema architecture for a relational database management system. And you know we said that there is this the physical schema, then the conceptual schema, and then the view level schema. So the view level schema basically contained views and views are kind of virtual relations. And they are as we have seen earlier that they are actually provided to give a specific user group a particular view of the database without giving the full access to the database.

We give them a restricted view of the database for various reasons for especially for the reason of hiding unnecessary data to be you know hiding the data and then exposing only the data that is necessary to be seen by that group of individuals. So there we have only seen it as a concept and now that we know SQL and then query specification. We can exactly see how exactly how this concept of views is actually realized ok.

So in general these are they can the existence of views can also simplify the query formulations and of course it achieves the objective of data hiding. That means we do not want to expose the data that is not necessarily to be seen and by one group of people. Then supposing there is some particular kind of data that is frequently accessed. Then we might and it is not one of the actual tables in the database ok.

Then we may be able to create a view consisting of this data that is frequently accessed and then actually improve the efficiency of query answering. So these are various possibilities when we have views. So once we create this views we will see how exactly we will create this views, it is the view is kept up to date by the RDBMS system. So we give the responsibility of keeping this information that is there in the view to the up to date to the RDBMS system.

So notice that the actual data is there in the base tables we call them as the base tables the tables that have been created as part of the create table commands that we have given. And these are the base tables that are data is available in them and then views will be created by making use of this information that is there in the base table as virtual tables. And we will see how exactly we will declare a view.

So in these obviously when data changes in the base tables, the views will have to be changed you know. The data in the views we will go through those changes that are appropriate changes. So this propagation of the changes that is done now in the base tables to the views and then keeping them up to date is the responsibility of the RDBMS system.

So the once we declare something as a view, we can always assume that it is accurate because the RDBMS takes responsibility of keeping it up to date while the changes are actually happening on the base tables. And ok this is I have already mentioned this that views are not really part of the conceptual schema. But they are kind of created to give a particular user group or a particular concerned with certain aspect of the entire enterprise, their view of the system.

Now as far as the implementation of this idea of use is concerned, there are multiple approaches. It is possible that you know we do not actually keep the view as a permanent table in the database. But just keep the definition of the view alone in the database, and then kind of create that view as and when it is required. If it is being used in a particular query, then create that view and then make use of that ok.

So to give you a heads up basically we will now be looking at how to create a view but the view will be actually created as a result of a particular query ok. So that query will constitute the definition of the view ok. So we have the definition of the view and that will always be stored in the database management system. And so we now then have option of whether to keep the view already created, so in such case it is called materialized.

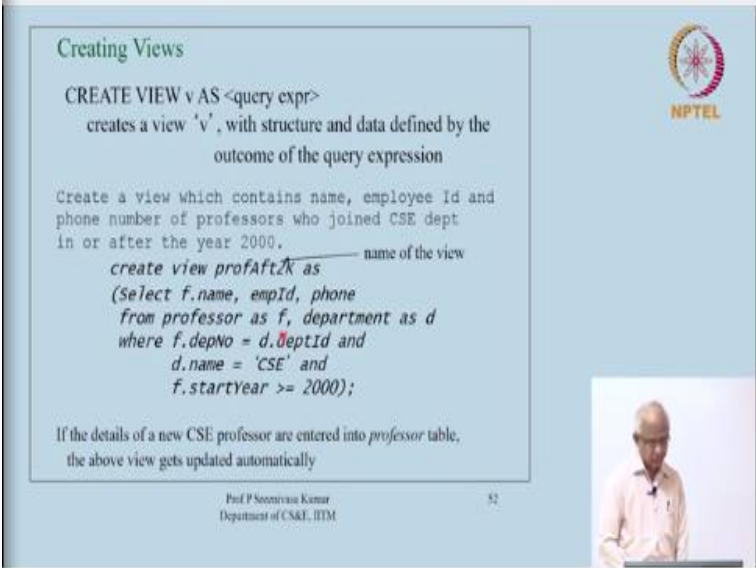
So we materialize the view and then keep it as an additional table as an additional relation in the apart from the base tables. And then of course incrementally keep updating it, so multiple ways of implementing this idea of view. So there are 2 possibilities one is to create this view as and when is required and used it and then kind of discard it actually. And then whenever again it is required it can be created.

The other one is to materialize it, that means keep it computed but then if you materialize it then you have to keep it up to date. So you have to recourse take recourse to the incremental updates as and when data changes in the base tables keep on propagating those updates to this materialized views. There is also another approach actually that since any of the user going to be used in you know in query.

So supposing a query has been you know stated on a view, can we transform that query. So that you know it is translated into an equivalent query on the base tables, that is the another approach. But usually it is a little bit more complicated to transform a query SQL query which is using a certain virtual table into the one that does not use that virtual table but uses only the base tables. So that approach of query transformation is a little complicated and normally it is not used.

But these are the other 2 approaches where you can keep the view materialized or if it is not so frequently used then you compute it as and when it is required and then make use of it. So these are the 2 options of implementing this idea of a view ok. So let us now start looking at exactly how the view definitions are made. So the tables involved in the view definitions are obviously I have been using the term base tables. So these are the tables that have been created as part of the conceptual schema and they are all available for us ok.

(Refer Slide Time: 09:00)



Creating Views

CREATE VIEW v AS <query expr>
creates a view 'v', with structure and data defined by the outcome of the query expression

Create a view which contains name, employee Id and phone number of professors who joined CSE dept in or after the year 2000.

create view profAft2K as — name of the view
(select f.name, empId, phone
from professor as f, department as d
where f.depNo = d.deptId and
d.name = 'CSE' and
f.startYear >= 2000);

If the details of a new CSE professor are entered into *professor* table, the above view gets updated automatically

Prof P Sreenivas Kumar
Department of CS&E, IITM

NPTEL

52

So SQL gives this command create view, name of the view AS keyword and then a query expression ok. So ok, what this does is to create a view named v with the structure and the data as defined by this query expression basically ok. Let us look at an example, so create a view which contains name, employee Id and phone numbers of professors who joined the CSE department in or after the year 2000.


For some reason we want to have the details of these people. So we can now given name for that profs after 2K and then basically supply the corresponding query. So this query is now seen as a view definition but what it actually is doing is to compute this required information. So the professors who joined the CSE department in or after 2000 is actually realized by this executing this particular query ok.

And then we want this proof after 2K is now going to be a relation name that is going to be available for us because we created as a view. So we can freely now use this in other queries ok. And it is up to the RDBMS to make use of this definition and then compute this query when compute the view data and then use it wherever this definition with this particular relation or table name is actually used ok.

If in a query if you use this then the RDBMS will compute the information using this particular query and then make use of in that query ok. So the query itself is very simple, so I do not want to spend time on that. So you can just see that where putting some conditions on the start here greater than equal to 2000 and then picking up the appropriate faculty data ok.

Now so this particular relation will be kept up to date by the RDBMS system as and when changes happen in the professor table. So if new professors join the computer science department then this particular thing will automatically get updated ok.

(Refer Slide Time: 12:21)



Queries on Views

Once created a view can be used in queries just like any other table.


e.g. Obtain names of professors in CSE dept, who joined after 2000 and whose name starts with 'Ram'

```
select name
from profAft2K
where name like 'Ram%';
```

The definition of the view is stored in DBMS, and executed to create the temporary table (view), when encountered in query

Prof P Suresh Kumar
Department of CSE, IITM

53

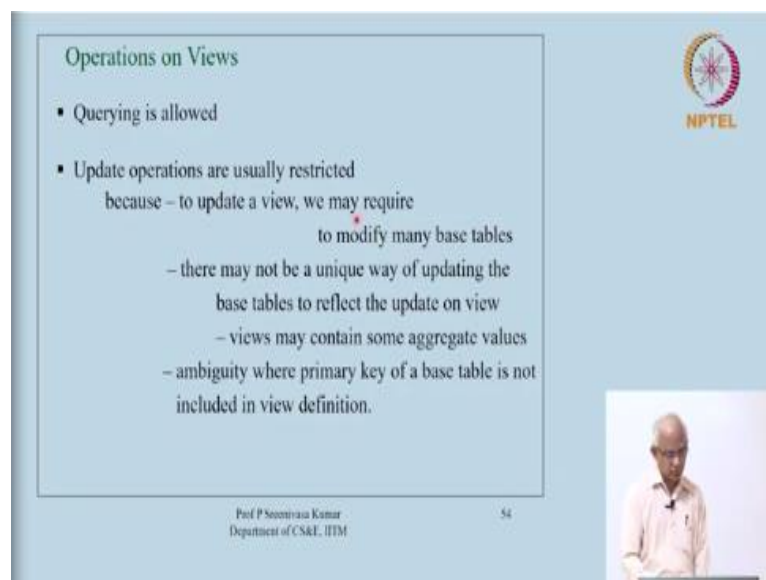


So once we create it, it can be used as I said you can always use it in queries just like any other table is being used, so I will give you some illustrations of that. So names of professors in whose name starts with Ravi, Ram something so some query, so now we are using. So far in all our queries, we have been using the tables that or part of the relation scheme. Now we started using

the views view names. Of course in order to write queries of this kind we should also whoever is writing this query should know what are the all the views that are available, right.

So generally this notion of use is going to be used in order to you know create a set of tables, a set of temporary virtual tables to give a certain restricted information for a user group as I said earlier study. So the thus whatever the definition we have to we have seen in the earlier slide is stored in the DBMS. And then when it encounters this query it will create either create the temporary table and use it or if it had already materialized it, it would use that anyway.

(Refer Slide Time: 14:09)



The slide is titled "Operations on Views" and contains a bulleted list of operations. The first bullet point is "Querying is allowed". The second bullet point is "Update operations are usually restricted", followed by a list of reasons: "because – to update a view, we may require to modify many base tables", "– there may not be a unique way of updating the base tables to reflect the update on view", "– views may contain some aggregate values", and "– ambiguity where primary key of a base table is not included in view definition." In the bottom right corner, there is a small video inset showing a man in a light-colored shirt speaking. The NPTEL logo is in the top right corner. At the bottom, it says "Prof P Sreenivasa Kumar, Department of CS&E, IITM" and the slide number "54".

- Querying is allowed
- Update operations are usually restricted
 - because – to update a view, we may require to modify many base tables
 - there may not be a unique way of updating the base tables to reflect the update on view
 - views may contain some aggregate values
 - ambiguity where primary key of a base table is not included in view definition.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

54

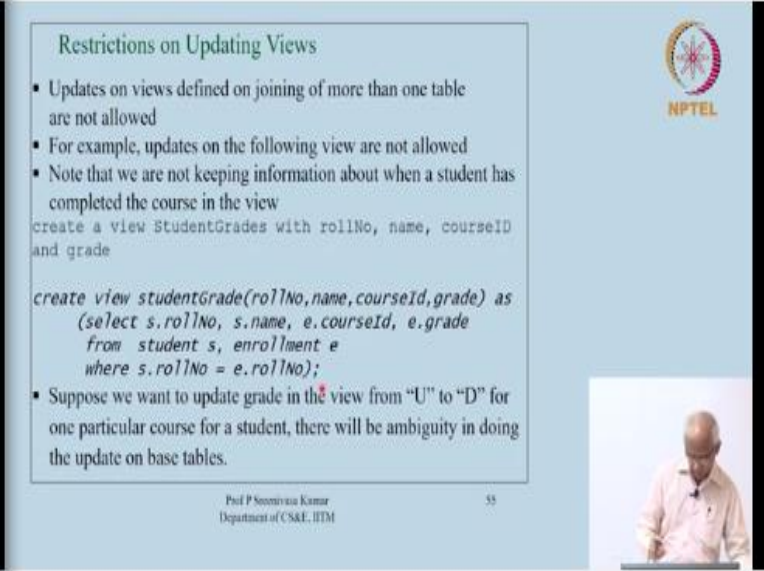
Now while querying is allowed what about any update operations, do they make sense. So we want to discuss this issue of updating though the actual SQL commands that are going to be used for update operation, I will talk about them a little later. But let us say we know how to update a table right. So, updating a table, so that means updating information in a particular attribute of a particular table.

We will see how exactly it can be done in SQL. But then ok when you look at updates on views itself usually you know that does not would not make sense. And I will tell you why it does not make sense. So and because of that the updates on views are going to be heavily restricted. And the main reason for that is to kind of update the view, we may actually require to modify the many base tables.

Because it is possible that the view makes use of many base tables you know so in the view definition many base tables might be involved. Also there may not be a unique way of updating the base tables to reflect the update on the view. And sometimes use may actually also contain some aggregate function values in which case it does not make sense. So I will give you examples for all these things now.

If the primary key of the base table is not included in the view definition, there could be even ambiguity as to what is the update that has to be performed. So we will look at examples for all of these things ok.

(Refer Slide Time: 16:20)



The slide is titled "Restrictions on Updating Views" and features the NPTEL logo in the top right corner. It contains the following content:

- Updates on views defined on joining of more than one table are not allowed
- For example, updates on the following view are not allowed
- Note that we are not keeping information about when a student has completed the course in the view

```
create a view StudentGrades with rollNo, name, courseId and grade

create view studentGrade(rollNo,name,courseId,grade) as
(select s.rollNo, s.name, e.courseId, e.grade
 from student s, enrollment e
 where s.rollNo = e.rollNo);
```

- Suppose we want to update grade in the view from "U" to "D" for one particular course for a student, there will be ambiguity in doing the update on base tables.

At the bottom of the slide, it says "Prof P Sreenivas Kumar, Department of CSE, IITM" and "55". There is also a small video inset of the professor in the bottom right corner.

So let us say we have a view which is created like this, I want to illustrate that if a view is created by joining more than one table ok. Then it is not you know good to encourage any kind of updates on that view at all. So in order to illustrate that, let us look at this the query view what is this view doing, create a view student grades with roll number, name, course Id and grade ok. So how do we create that, create view, student grade, roll number, name, course Id and grade. Where is that information is coming from you have the schema with you.

So from student and enrollment, we can get this information. Basically we are one more thing you see here that we are not keeping track of when that particular student has completed that

course, we are dropping off the semester year information. The enrollment table has other details as to when that particular student has done the course and obtain the grade. So we are dropping off all that information here to get us to kind of get the transcript of this student kind of thing.

So roll number, name, course Id, grade is there. So of course it is easy to get this because all that we have to do is to join student and enrollment with this and then impose this condition saying that roll number should be equal to the e dot roll number and pick up name from the student ok. Now supposing if I take this view now and then suddenly decide that you know let me update the grade value for some student in some course.

For a particular student in a particular course, let me upgrade his grade from U to some D something like that ok. Suppose a student has not cleared a course and he got U grade, then he kind of repeating the course. Then what will happen in this particular result of the view, the same the student will appear with that course Id multiple times right. If you have cleared the course, then you would apply it only once, if not cleared it and it is a core course and it has to be cleared.

So the student might appear actually multiple times, suppose he has taken it 2 times and then unfortunately both times is not cleared the course let us say. Then in fact there will be only view on record here because semester and year are dropped out from this particular thing. There will be one record saying that this particular student has done that and got a u grade even though he has done it 2 times.


Now let us say you want to do this upgrade saying that the grade value for a particular course for a particular student has to be upgraded from U to D. There is obviously ambiguity here because we do not know which of those of course we can which of those underlying I mean in the enrollment table for what semester and what year will you upgrade it from U to D. Because he might have done it, he has done it 2 or 3 times, so whatever.

So of course you may take a policy here saying that the last time he has done of course might be upgraded. But in general you can see the ambiguity because it is result the view is a result of joining multiple tables. And there are multiple more than one way in which the base tables can

actually be upgraded to give this joint which is the result of this particular query. So since there is going to be ambiguity when multiple tables are involved in the view definition.

And a upgrade a change in one of the attribute values might actually be realized by multiple ways of changing the base tables. Because of all these reasons, we basically do not allow any updates to be done on any view that is defined by joining more than one table ok. So that is a there is a reason why we restrict it like that.

(Refer Slide Time: 22:00)



Restrictions on Updating Views


- Updates on views defined with 'group by' clause and aggregate functions is not permitted, as a tuple in view will not have a corresponding tuple in base relation.
- For example, updates on the following view are not allowed

Create a view deptAvgCredits which contains the average credits of courses offered by a dept.

```
create view deptAvgCredits(deptNo, avgCredits)
as select deptNo, avg(credits)
from course
group by deptNo;
```

Prof P Sreenivas Kumar
Department of CS&E, IITM

56

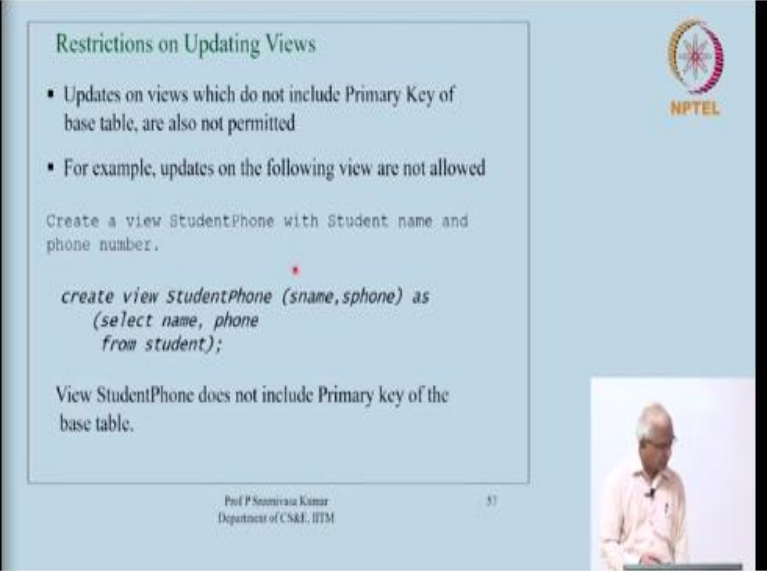


In a similar way supposing the view is created by making use of a group by and an aggregate function. Let us look at this view, what this view is doing is to compute what is the average number of credits of the courses that are offered by all departments ok. Across all departments, we want to now find out what is for all the courses that they are offering you look at the credits.

And then take the average of the credits and then take the average of the credits and then report that, how do you do that. Select department number and average of credits which is a aggregate function and do a group by department number ok. If you recall what we basically doing here is that the course tuples will be grouped in department wise partition department wise. And for each department, we will be computing the average and then that is the table will now have department number and average credits.

It is obviously does not make sense to kind of go and update this average credits attribute. Because it is not it does not make sense at all because there are multiple ways in which the credits could have changed to create that average right the updated average. So updates on this kind of views which involve using the aggregates will not be allowed at all. Again you can see that the main reason is that the update on this particular attribute for example average credits cannot be uniquely mapped to any update on the base tables, that is the reason ok.

(Refer Slide Time: 24:26)



The slide is titled "Restrictions on Updating Views" and features the NPTEL logo in the top right corner. It contains two bullet points: "Updates on views which do not include Primary Key of base table, are also not permitted" and "For example, updates on the following view are not allowed". Below the bullet points, it says "Create a view StudentPhone with Student name and phone number." followed by a red asterisk and the SQL code: `create view StudentPhone (sname,sphone) as (select name, phone from student);`. It then states "View StudentPhone does not include Primary key of the base table." At the bottom left, it identifies the speaker as "Prof P Sreenivasa Kumar, Department of CS&E, IITM". At the bottom right, there is a small video inset showing the professor speaking. The slide number "57" is located at the bottom center.

Restrictions on Updating Views

- Updates on views which do not include Primary Key of base table, are also not permitted
- For example, updates on the following view are not allowed

Create a view StudentPhone with Student name and phone number.

```
create view StudentPhone (sname,sphone) as
(select name, phone
from student);
```

View StudentPhone does not include Primary key of the base table.


Prof P Sreenivasa Kumar
Department of CS&E, IITM

57

Moving on in a similar spirit if the primary key, so is not included as part of the definition. So use in which do not include the primary key of the base tables will also be not permitted. Again the reason is that the update on the view can be mapped in multiple ways to the updates on the base table. So let us say here is a create a view student phone with student name and phone number, student name is not a key.

So you are just projecting name and phone from student and so there will be so many Kumar's with so many phones are there. If you try and update some Kumar's phone number, then you will be in trouble which Kumar is this.

(Refer Slide Time: 25:27)




Allowed Updates on Views

Updates to views are allowed only if

- defined on single base table
- not defined using 'group by' clause and aggregate functions
- view includes Primary Key of base table

Prof P Senthil Kumar
Department of CS&E, IITM

58



So basically with to kind of summarize what we are doing is allow updates on views only if the view is defined on a single base table. And it is not defined using any group by clause or aggregate functions and the view definition includes primary key of the base table. If the view definition includes the primary key of the base table. Then there is a possibility for uniquely propagating the update to the base table.

And so that is the restrictions on upgrading views ok. So basically to kind of summarize this concept of views, it is possible for us to create a bunch of virtual relations which will give a restricted view of the information system for use of a group of individuals. And these virtual relations can then be used by those group of people for querying information ok. And by and large updates on these things will not most of the time makes sense.


And if they make sense only they will be allowed, so that is the kind of summary on the views but they are of course very useful. Because they will create they will give us that particular group of individuals access to the part of the information system that they are allowed to make use of ok. And in general the RDBMS takes responsibility of keeping all these virtual relations up to date by it is own mechanism.

We do not care about how exactly does it, it might do it through materialization or through you know creating these views as and when it is required to be used, good. So that is the and in

general this feature of views itself is a little advanced feature that and some RDBMS might not even provide this facility ok. So it is part of the standard of course if you have used what should be done and how they should be created is all part of SQL standard.

So a particular RDBMS might not even implement this feature, it might just give you a basic version of SQL. And most enterprises whose operations do not read these things might be able to make use of such as RDBMS here ok.

(Refer Slide Time: 28:32)




Inserting data into a table

- Specify a tuple(or tuples) to be inserted
`INSERT INTO student VALUES
('CS05D014', 'Mohan', 'PhD', 2005, 'M', 3, 'FCS008'),
('CS05S031', 'Madhav', 'MS', 2005, 'M', 4, 'FCE009');`
- Specify the result of query to be inserted
`INSERT INTO ri SELECT ... FROM ... WHERE ...`
- Specify that a sub-tuple be inserted
`INSERT INTO student(rollNo, name, sex)
VALUES (CS05M022, 'Rajasri', 'F'),
(CS05B033, 'Kalyan', 'M');`
- the attributes that can be NULL or have declared default values can be left-out to be updated later

Prof P Sreenivas Kumar
Department of CS&E, IITM

59



So now let us go on to looking at a few other features of SQL which are basically to do with updating data. So now that we have been actually the reason why I am discussing updating inserting data towards the end. Instead of doing it at the beginning is that while updating also it is possible for us to kind of use query. That is the reason why we are discussing updates and inserting data towards the end actually ok.


So let us see how to insert a data into the tables, so there are SQL constructs for doing that. So insert into and then the relation name, values is a keyword and then you can supply a comma separated tuples ending with the semicolon. So that all this data goes into that particular thing. Of course, you should put values in quotes and then they should all be comma separated. So of course the RDBMS might give you a loading you know data loading functionality in which it might you know take data from multiple formats and then actually store it in to the tables.

But this is the SQL commands that I am talking about. There may be other ways of bulk loading data into the database in which case it might take data from say spreadsheets or you know CSV files and things. It is also possible for us to insert data by giving a query this is the reason why we are discussing this into at this stage. So insert into some relation and then give a query a select from where.

So the entire result of that particular query will be inserted into this possibly new relation. We can also specify that some sub tuple be inserted, so select insert into student roll number, name, sex alone. So you are giving what is the schema of the sub tuple here and then values of course these values the tuples that here should obviously match this number and then the data types we can give like this.

So the assumption here is that the rest of the attributes will be updated later or will be you know inserted as default values or something like that. So typically, so if you do not declare if you do not insert data then either the attributes if it is allowed to be null then null will be used. And if you have access to the default values then default values to be inserted ok. So inserting data into the tables can be done like this.

(Refer Slide Time: 32:12)



Deleting rows from a table

- Deletion of tuples is possible ; deleting only part of a tuple is not possible
- Deletion of tuples can be done *only from one* relation at a time
- Deleting a tuple might trigger further deletions due to *referentially triggered actions* specified as part of RIC's
- Generic form: *delete from r where <predicate>;*


Delete tuples from professor relation with start year as 1982.

```
delete from professor
where startyear = 1982;
```

- If 'where' clause is not specified, then all the tuples of that relation are deleted (Be careful !)

Prof P Sreenivas Kumar
Department of CS&E, IITM

60



Then deleting stuff from the deleting rows from tables. So deleting of tuples is obviously possible, but deleting any part of the tuple is not possible you have to drop the entire row ok. Now deletion of tuples can only be done from one relation at a time, you cannot delete couples from multiple relationships at the same time. So notice one thing that while you are deleting rows, it might trigger our referential integrity constraint violations you know it might trigger some actions.

So because one tuple goes from one relation then you know it is being referred tuple from another relation. And then might be you know referentially triggered actions defined in that case. In which case the deletion might cascade and all that, so one has to be aware of those issues when you are doing deletions. So the generic form for deleting a row is delete from the table name where predicates.



So all those rows that satisfy these predicates will be deleted from the thing ok. So here is an example, people have retired, so delete tuples from professor relation with start year as 1982. So delete from professor where start year equals 1982, simple but one has to be careful about one aspect about deletion . Let me tell you that ok before we going there, if the where clause is not specified then the whole of the data will be deleted.

So we have to be careful about it because you know that where clause if it is not specified it is taken as true right always true, so one has to be careful about that.

(Refer Slide Time: 34:39)

A Remark on Deletion

- The where predicate is evaluated for each of the tuples in the relation to mark them as qualified for deletion *before* any tuple is actually deleted from the relation
- Note that the result may be different if tuples are deleted as and when we find that they satisfy the where condition!
- An example:
Delete all tuples of students that scored the least marks in the CS branch:
DELETE
FROM gateMarks
WHERE branch = "CS" and
marks = ANY (SELECT MIN(marks)
FROM gateMarks
WHERE branch = "CS")



Prof P Sreenivas Kumar
Department of CS&E, IITM

61


See here is another interesting aspect of deletion that we have to be aware of. The where predicate actually is evaluated for each of the tuples in the relation and then we mark them as being qualified for deletion, and only then after we mark all of them we start deleting ok. So before deleting any tuples we are going to mark all those tuples that qualify the where condition that is specified.

This is very important as you will see this is very important actually. The result of you know deleting these rows as and when they have satisfying the where condition is a little dangerous. Let me illustrate that with a query here, you recall that we had a get marks thing. So delete all tuples of students that scored the least marks in the CS branch let us say ok. So delete from gate marks where branch equals computer science and marks is equal to you have a sub query to compute the minimum marks in the branch computer science.

So if the marks equal to the minimum marks then you delete that row. Now here you can see the danger right you can see the danger here very clearly, that if you start deleting as and when a tuple satisfies this condition. Then actually what will happen one by one all the students of computer science branch will get deleted because at some point of time everybody will become minimum among the remaining students and so it is the dangerous thing right.

So that is why it is very important for us to mark all the rows that satisfies the where condition and then go about deleting them after marking ok, so this is one aspect you have to be careful about.

(Refer Slide Time: 37:09)



Updating tuples in a relation

```
update r
set <<attr = newValue> list>
where <predicates>;
```


Change phone number of all professors working in CSE dept to "94445 22605"

```
update professors
set phone = '9444422605'
where deptno = (select deptId
                from department
                where name = 'CSE');
```

If 'where' clause is not specified, values for the specified attributes in all tuples is changed.

Prof P Sreenivas Kumar
Department of CS&E, IITM

62




Now let us come to the actual update, so we have been talking about updates, so what is the constructs for doing update. So, update relation here is a key word set and then attribute what is the attribute and what is the new value. So that kind of a list we can give, a list of attribute value pairs with the equality in between you can give. Again we can give a predicate here saying that for all those rows that satisfy this particular condition, you update the value to this new value.

So change the phone numbers of all professors working in CSE department to something it is a dangerous thing but anyway we will let us say how to do that. So update professors that is the relation name, set phone equals so and so where so department number equals department. So you do not know the department number of computer science, so you are selecting this could be equal to or in also it should be equal to any.

So this sub query will pursue the department Id of CSE department. So for all these people we are now setting the phone number to something is this a valid phone number at all ok, looks like that. Of course if where clause is not specified, then it is dangerous because all tuples will be

changed, so one has to be careful about it. So this is how updates happen, so here I gave one attribute and one value, you can give a list actually, you can also give a list of such pairs.

(Refer Slide Time: 39:05)



Miscellaneous features in SQL (1/3)


- Ordering of result tuples can be done using 'order by' clause
e.g., List the names of professors who joined after 1980, in alphabetic order.

```
select name  
from professor  
where startYear > 1980  
order by name;
```
- Use of 'null' to test for a null value, if the attribute can take null
e.g., Obtain roll numbers of students who don't have phone numbers

```
select rollNo  
from student  
where phoneNumber is null;
```

Prof P Sreenivas Kumar
Department of CS&E, IITM

63



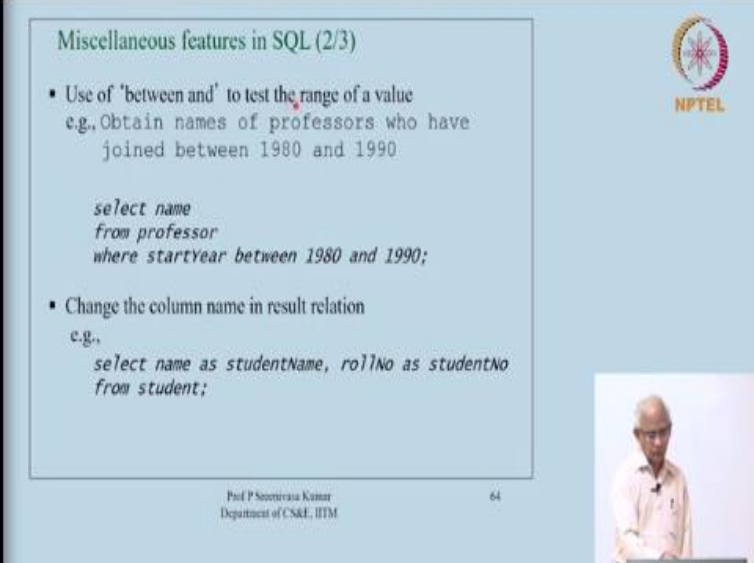
Then other miscellaneous features of SQL which we have seen, there is an interesting clause called order by clause that is new here. Order by is basically used after the query result is computed, you can order the tuples of the result by making use of the attributes that have been specified here. So in this case, select name professor where start year equals greater than 1980 order by name.

So the professor's names will be listed in alphabetical order, because that is how we order. So the default thing is ascending, then you can control whether it should be descending by giving a key word called descending in this case. If you do not give anything it is ascending value, so it would be arranged in alphabetically increasing order. So order by clause is available, so please pick up the information about.

So you can also order by multiple attributes, so again lexicographic ordering will be used. You can also check whether something is actually null, the value is null or not ok, so is null is a predicate that can be used in where clauses. The null values in SQL tables could be present for various reasons because you do not know the exact value or the value is not applicable etc.

So for multiple reasons there could be a null value and you can check whether the actual value is null. And it is recommended to use these things then, so the equal to null does not work actually it is this is null that works ok.

(Refer Slide Time: 41:11)



Miscellaneous features in SQL (2/3)

- Use of 'between and' to test the range of a value
e.g., Obtain names of professors who have joined between 1980 and 1990

```
select name  
from professor  
where startYear between 1980 and 1990;
```
- Change the column name in result relation
e.g.,

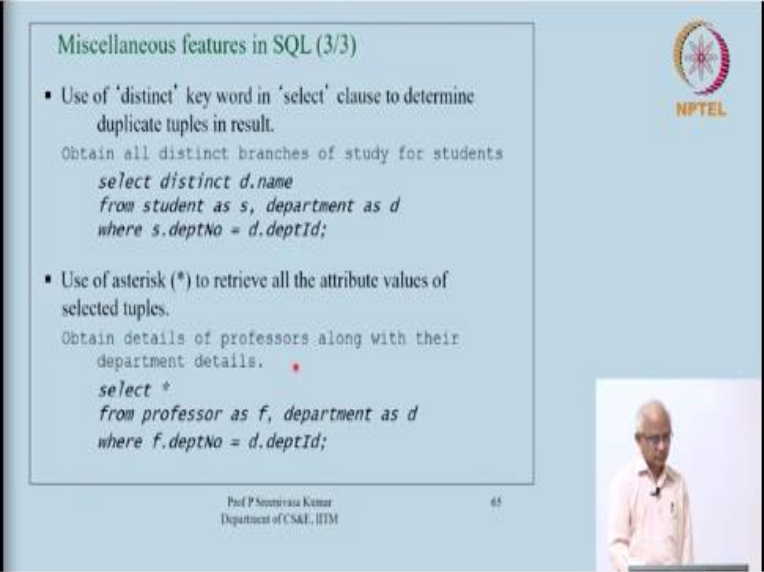
```
select name as studentName, rollNo as studentNo  
from student;
```

Prof P Sreenivasa Kumar
Department of CSE, IITM

64

You can also use between and to give a range of values start year between so and so. And we have already seen how result columns can be renamed using the AS keyword name as student name, roll number as student number etc. So this attribute renaming we have already seen.

(Refer Slide Time: 41:37)



Miscellaneous features in SQL (3/3)

- Use of 'distinct' key word in 'select' clause to determine duplicate tuples in result.
Obtain all distinct branches of study for students

```
select distinct d.name  
from student as s, department as d  
where s.deptNo = d.deptId;
```
- Use of asterisk (*) to retrieve all the attribute values of selected tuples.
Obtain details of professors along with their department details.

```
select *  
from professor as f, department as d  
where f.deptNo = d.deptId;
```

Prof P Sreenivasa Kumar
Department of CSE, IITM

65

And then use of distinct keyword we can see, if you want, the select clause you can use distinct keyword in order to eliminate duplicates. And then use of this asterisk to retrieve all attribute

values, so select star is called select star. So obtain details of professors along with their department details. So whatever all the available details you want obtain along with their department details.

So this will be a wide table in where all the details of the professor's concatenate with all the details of the department will be produced. So there are a lot of them and so instead of writing all of them, you will just simply put a star there. So that is how we can make use of this ok, good. So in this lecture, basically we have the main thing that we have seen is views and then there are a lot of other miscellaneous features of SQL that we have looked at.

In the next class I will be talking about how we can programmatically access data in a database. And then with that we will probably wind up the discussion on SQL ok.