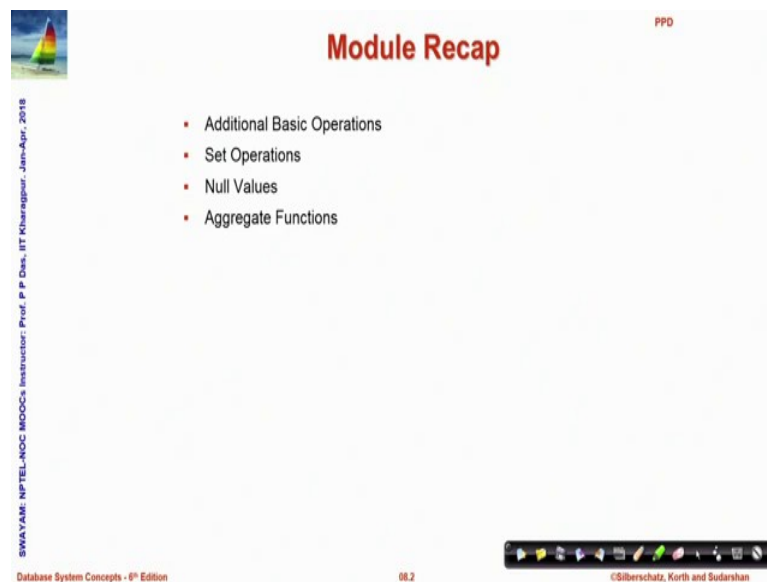


Database Management System
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 08
Introduction to SQL/3

Welcome to module 8 of Database Management Systems. We have been discussing basic SQL queries and this is the third and closing part of that introductory discussion that we started in the 6th module.

(Refer Slide Time: 00:34)



The slide is titled "Module Recap" in red text. It features a list of topics covered in the module: Additional Basic Operations, Set Operations, Null Values, and Aggregate Functions. The slide is part of a presentation by Prof. P. P. Das, IIT Kharagpur, Jan-Apr. 2018. The footer includes the text "Database System Concepts - 8th Edition" and "©Silberschatz, Korth and Sudarshan".

Module Recap

- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr. 2018

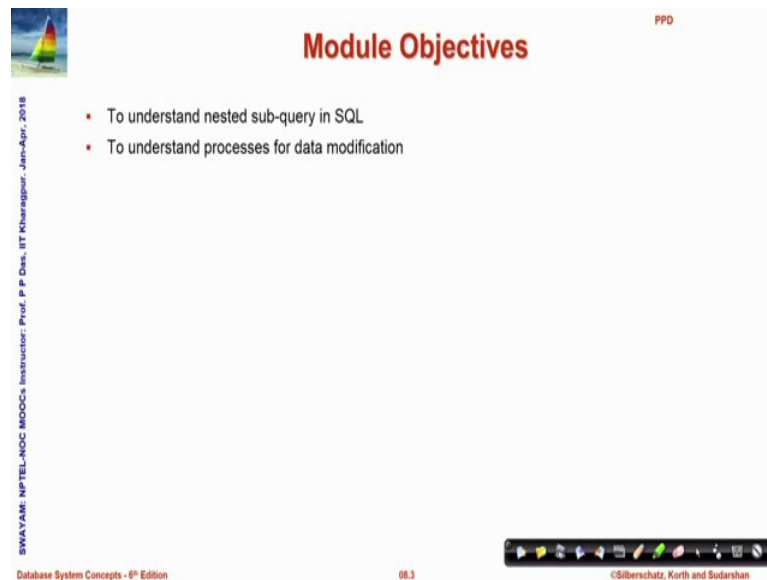
Database System Concepts - 8th Edition

08.2

©Silberschatz, Korth and Sudarshan

So, just to quickly recap, these are the things that we did in the last module completing the understanding of basic operations, the null values and aggregate functions.

(Refer Slide Time: 00:44)



Module Objectives

- To understand nested sub-query in SQL
- To understand processes for data modification

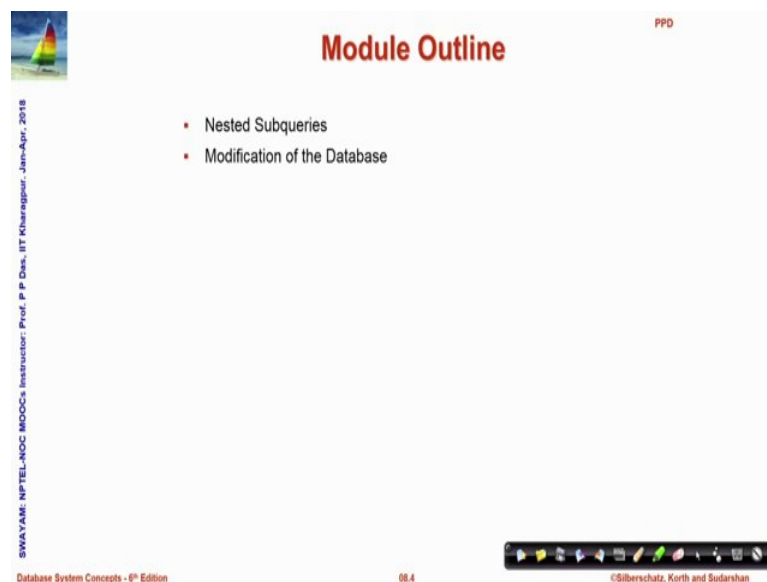
SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition 08.3 ©Silberschatz, Korth and Sudarshan

PPD

In the current module, we want to understand a feature which is very very important in SQL query forming it is called the nested query or more formally nested sub query. In SQL and we would like to understand the process of data modification.

(Refer Slide Time: 01:05)



Module Outline

- Nested Subqueries
- Modification of the Database

SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018


Database System Concepts - 9th Edition 08.4 ©Silberschatz, Korth and Sudarshan

PPD

And those are the two things that are outlined here.

So, let us start with nested sub queries.

(Refer Slide Time: 01:12)




Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries
- A **subquery** is a **select-from-where** expression that is nested within another query
- The nesting can be done in the following SQL query

```
select A1, A2, ..., An
from r1, r2, ..., rm
where P
```

as follows:

- A_i can be replaced by a subquery that generates a single value
- r_i can be replaced by any valid subquery
- P can be replaced with an expression of the form:
B <operation> (subquery)
where B is an attribute and <operation> to be defined later



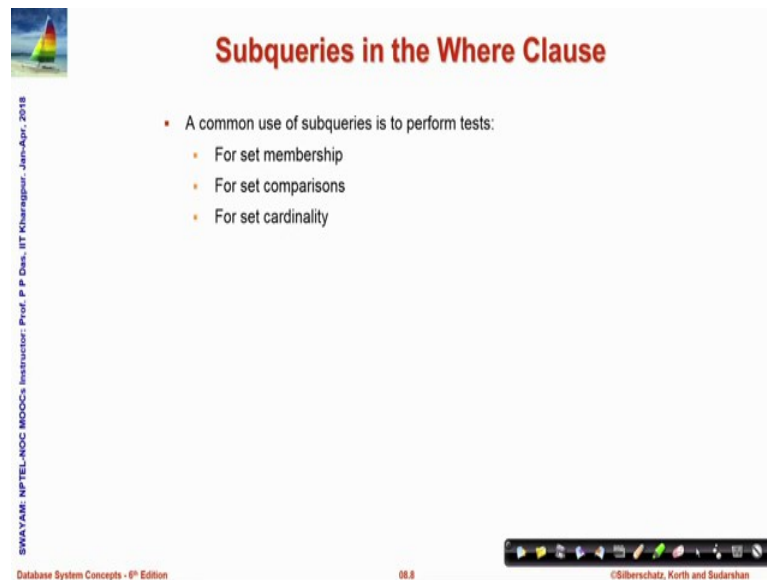
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition 08.6 ©Silberschatz, Korth and Sudarshan

A sub query is necessarily a SELECT FROM WHERE expression that is nested within another query, this is. So, these are the key things where expression. So, it is nothing new over; what we have already learned? But it is a part of another query it is nested within another query and that is the reason it is called a sub query. So, it by itself is not the result. this will be used, this is a SELECT FROM WHERE expression that will be used in a nested form in another some other queries to actually generate the result.

So, that is a nested sub query. So, the nesting can be done in one or more of the three clauses that a SELECT FROM WHERE SQL query has. An attribute can be replaced a relation can be any valid sub query or a sub query could form the part of a predicate in the where clause all of these are possible. So, we will discuss them one by one. So, first we will start by discussing how sub queries work in the where clause.

(Refer Slide Time: 02:44)



Subqueries in the Where Clause

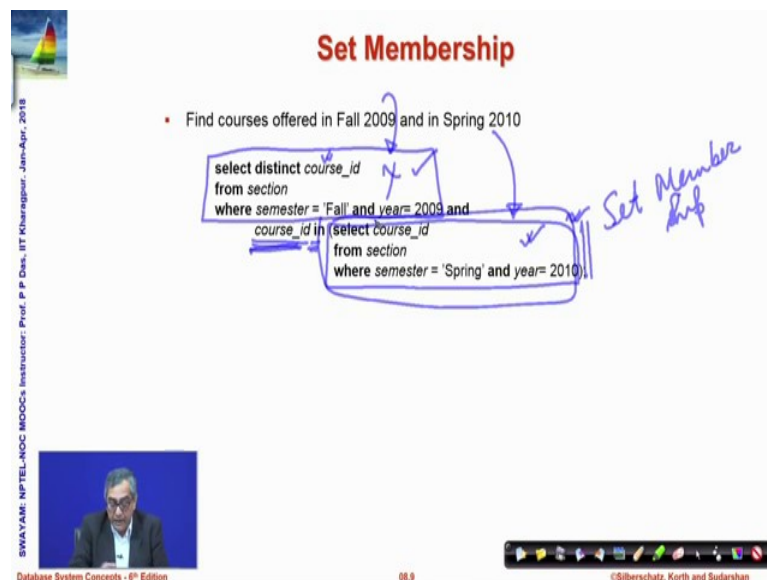
- A common use of subqueries is to perform tests:
 - For set membership
 - For set comparisons
 - For set cardinality

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition 08.9 ©Silberschatz, Korth and Sudarshan

The common use for you having sub queries in the where clause is to perform different kind of tests for membership comparison cardinality and so, on.

(Refer Slide Time: 02:58)



Set Membership

- Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
course_id in (select course_id
from section
where semester = 'Spring' and year= 2010);
```

Set Membership

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition 08.9 ©Silberschatz, Korth and Sudarshan

So, let us look at this; you have already seen this query before find courses offered in fall 2009, and in fall in spring 2010. Earlier we have shown different ways of coding this, now we are showing yet another way to be able to code this in SQL.

So, first from the beginning certainly we need the courses. So, the SELECT clause is trivial it has to be the distinct course id is certainly the information will come from

section which has information about offering of courses as we have also seen already. So, those two are no brainer. Now let us see what how do I find courses that are offered in fall 2009 and in spring 2010. So, again the first part, the courses offered in fall 2009 is coded in this part; in part of that where clause predicate when he says semester has to be fall and here is 2009. So, this part is also done.


Now, we need to ensure that whatever I mean, if I assume that after this this part were not there, then this will only give me courses which are offered in fall 2009, but we want the courses that are in fall 2009 and in spring 2010. So, we do something interesting what I do is, we write a separate query here which is courses that are offered in spring 2010. `SELECT course_Id` just in the same way, `SELECT course_Id, section, semester year`. So, this particular query will give me the courses offered in spring 2009. So, what do we have in one part? I have so, if you look at this part this courses that happened in fall 2009; if we look at this part courses that happened in spring 2010 good.

Now, what I want, I need that the; it I am interested in the courses that happened in both. So, for a course that exists here I want to specify that that course Id must be present here. So, what I am checking for? I am checking for a set membership; this is a set right. So, I am trying to check, whether that course Id which is being selected in the first part exists in, in is another keyword. In this particular, this particular relation that is specified by the second part of this query which is courses offered in spring 2010.

If it is, if the course Id is present, then that course must have been offered in both the semesters if it is not present, then it is offered only in fall 2009, and not in spring 2010 and certainly the courses that were not offered in fall 2009, and only offered in spring 2010 will feature here, but they do not exist here. So, they will never come up for tests. So, as a result what I get finally, is the effect of computing courses that are offered in fall 2009 and in spring 2010. this part of the query which I used as a part of the where clause is my nested sub query.

And in this case as we have seen it is used for set membership and this is a basic idea of using nested sub queries; that is a nested sub query will always give you a relation. So, you try to put that relation in the right context of the where clause from clause or `SELECT` clause, and then make use of it in building up your logical.

(Refer Slide Time: 07:14)



Set Membership

- Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
course_id in (select course_id
from section
where semester = 'Spring' and year= 2010);
```

- Find courses offered in Fall 2009 but not in Spring 2010


```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
course_id not in (select course_id
from section
where semester = 'Spring' and year= 2010);
```

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition 08.9 ©Silberschatz, Korth and Sudarshan

So, let us run through some examples this is, what you are saying is, earlier one was the courses offered in both here we are trying to do kind of the difference saying the courses offered in fall 2009, but not in spring 2010 certainly we easily get that by changing the membership to negation of the membership earlier it was in now you do not in you will simply get that it is up to you to take some examples and convince yourself that this kind of a nesting will work.

(Refer Slide Time: 07:43)




Set Membership (Cont.)

- Find the total number of (distinct) students who have taken course sections taught by the instructor with ID 10101

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
(select course_id, sec_id, semester, year
from teaches
where teaches.ID= 10101);
```

- Note: Above query can be written in a much simpler manner.
The formulation above is simply to illustrate SQL features.

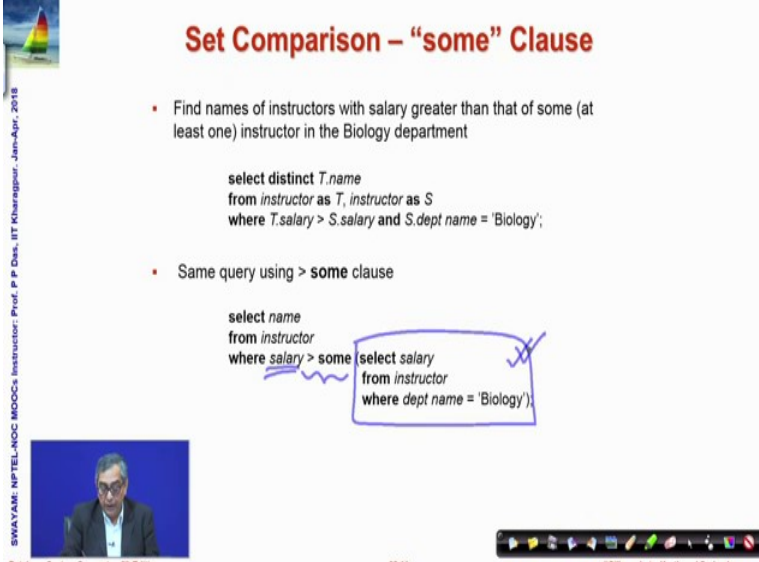


SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition 08.10 ©Silberschatz, Korth and Sudarshan

We find the set of the find the total number of distinct students, we have taken the course section taught by the instructor Id, some Id is given. So, again we form a nested query here is a nested query, which tells me the courses taught by this particular teacher 10101, and then we check set membership in terms of this course Id, section Id that is fields of the takes relation to see that, whether that particular tuple can exist in the course offered by the specific teacher; if it does then take out that I d which is which will turn out to be the student Id. In this case because that is the text relation has the student I d take out the student I d and count it as distinct. So, this can simply give you the answer to this squared; obviously, we agree that this can be written in a simpler form also, but we are including it here just for the sake of illustrating the feature.

(Refer Slide Time: 09:03)



Set Comparison – “some” Clause

- Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept name = 'Biology';
```

- Same query using > some clause

```
select name
from instructor
where salary > some (select salary
                      from instructor
                      where dept name = 'Biology');
```

SWAYAM: NPTEL-NOC MOOCs Instructor Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition

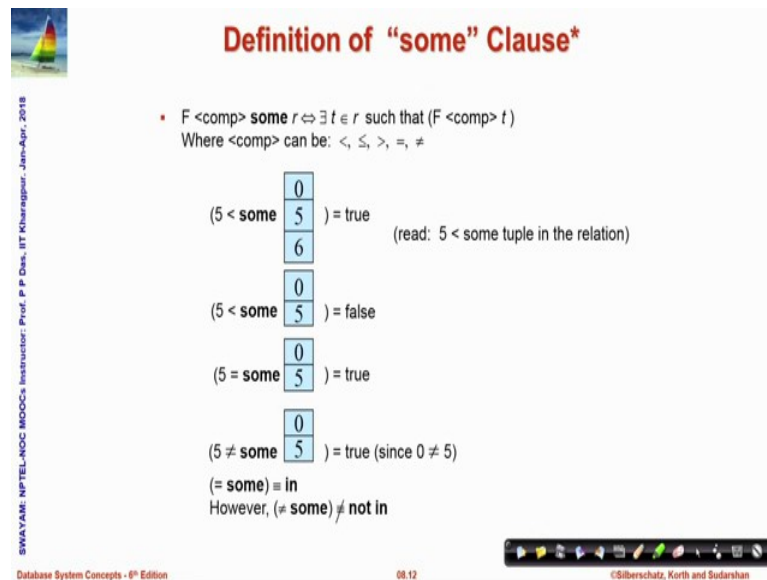
08.11

©Silberschatz, Korth and Sudarshan

There is another clause called the some clause look at this fine names of instructor; salary is greater than that of some which means at least one instructor in biology department and we have already seen this coding before.

Now, we can do this in terms of the nested query by using, again this is certainly the salary of instructors in biology department and we are doing greater than sum; that means, that the salary here being checked must find at least one record here. So that it is greater than that salary value. So, it is greater than some is a nice way to find existential records using the nested sub query.

(Refer Slide Time: 10:02)



Definition of "some" Clause*

- $F <comp> \text{some } r \Leftrightarrow \exists t \in r \text{ such that } (F <comp> t)$
Where $<comp>$ can be: $<, \leq, >, =, \neq$

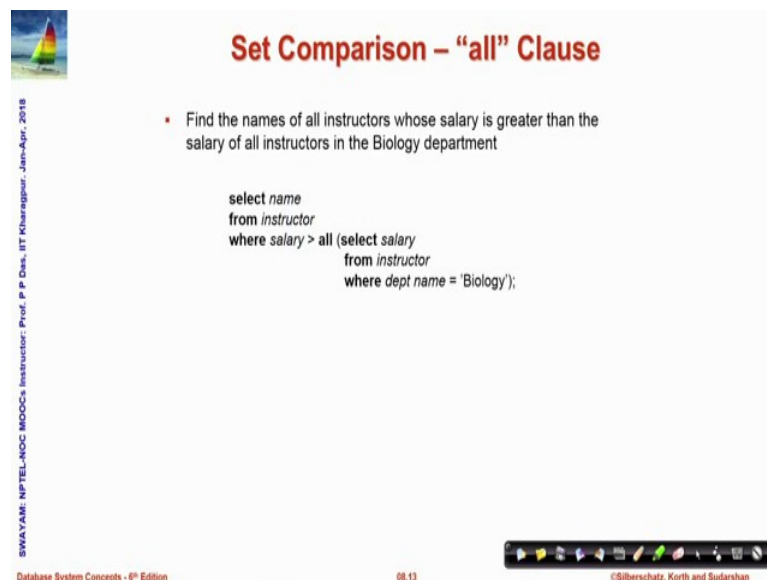
Examples:

- $(5 < \text{some } \begin{smallmatrix} 0 \\ 5 \\ 6 \end{smallmatrix}) = \text{true}$ (read: 5 < some tuple in the relation)
- $(5 < \text{some } \begin{smallmatrix} 0 \\ 5 \end{smallmatrix}) = \text{false}$
- $(5 = \text{some } \begin{smallmatrix} 0 \\ 5 \end{smallmatrix}) = \text{true}$
- $(5 \neq \text{some } \begin{smallmatrix} 0 \\ 5 \end{smallmatrix}) = \text{true (since } 0 \neq 5)$
- $(= \text{some}) \equiv \text{in}$
However, $(\neq \text{some}) \neq \text{not in}$

Database System Concepts - 8th Edition 08.12 ©Silberschatz, Korth and Sudarshan

The logic of some clause, I have detailed out here. So, we will not go through each one of them in this discussion.

(Refer Slide Time: 10:09)



Set Comparison – "all" Clause

- Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department

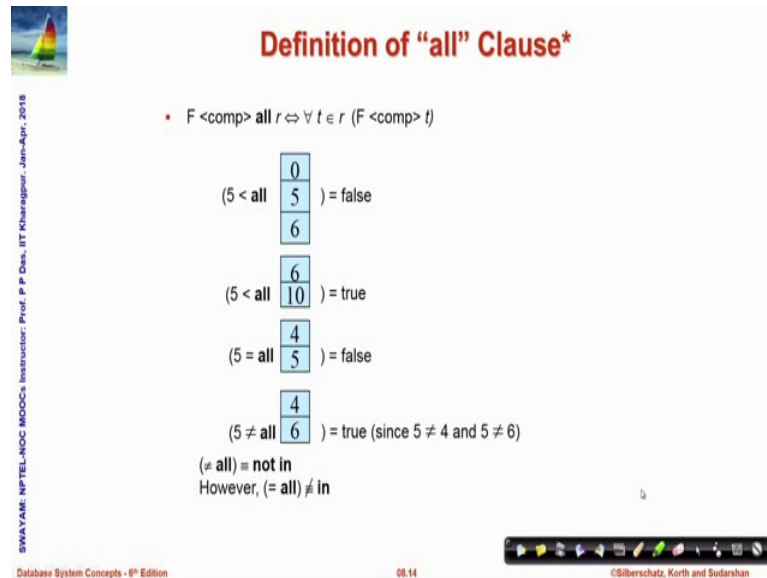
```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept name = 'Biology');
```

Database System Concepts - 8th Edition 08.13 ©Silberschatz, Korth and Sudarshan

I leave it unto you to study and convince yourself, that you understand the semantics of some. So, similarly we have an all clause which say that if we want to say, they find the names of all instructors; whose salary is greater than the salary of all instructors in the biology department in case of sum we can write or we will write all and it will check every salary will check with the whole set of salaries in this sub query. And only if that is

greater than that particular record, that particular name will be included in the result. Otherwise it will be excluded from the result.

(Refer Slide Time: 10:52)



Definition of "all" Clause*

- $$F < \text{comp} > \text{all } r \Leftrightarrow \forall t \in r (F < \text{comp} > t)$$

Examples:

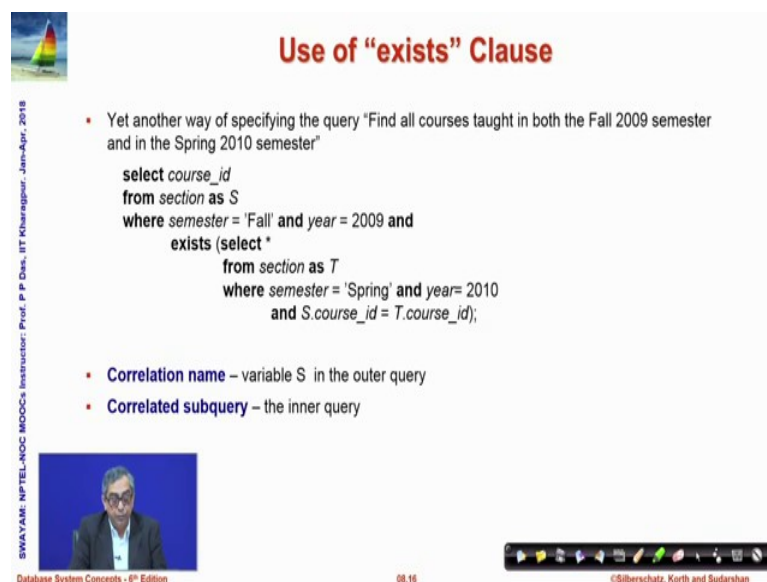
- $$(5 < \text{all } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{false}$$
- $$(5 < \text{all } \begin{array}{|c|} \hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{true}$$
- $$(5 = \text{all } \begin{array}{|c|} \hline 4 \\ \hline 5 \\ \hline \end{array}) = \text{false}$$
- $$(5 \neq \text{all } \begin{array}{|c|} \hline 4 \\ \hline 6 \\ \hline \end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6)$$

$(= \text{all}) = \text{not in}$
 However, $(\neq \text{all}) \neq \text{in}$

Database System Concepts - 9th Edition 08.14 ©Silberschatz, Korth and Sudarshan

Similar to some there is a basic semantics of all which is also worked out here and I leave that to your study at home.

(Refer Slide Time: 11:03)



Use of "exists" Clause

- Yet another way of specifying the query "Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester"


```

select course_id
from section as S
where semester = 'Fall' and year = 2009 and
exists (select *
from section as T
where semester = 'Spring' and year = 2010
and S.course_id = T.course_id);
            
```
- Correlation name** – variable S in the outer query
- Correlated subquery** – the inner query

Database System Concepts - 9th Edition 08.16 ©Silberschatz, Korth and Sudarshan

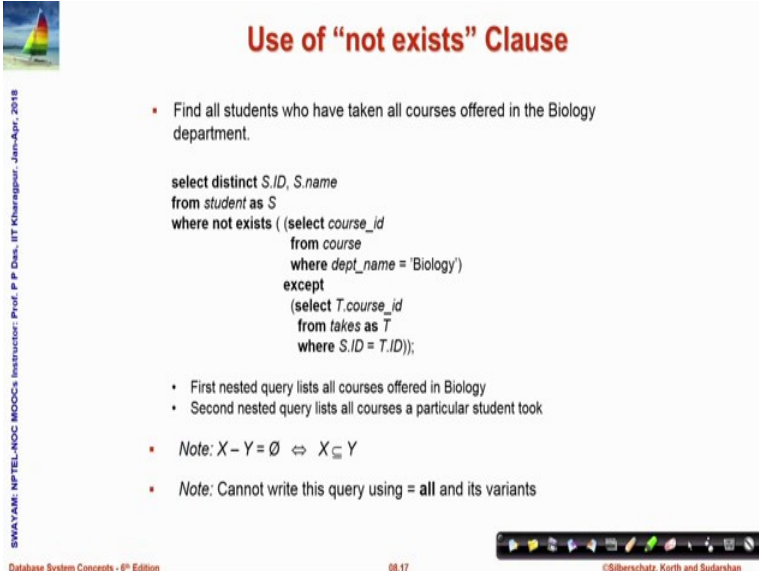
You can test for empty relations by using the exists construct. So, if you say exists r, then that is a predicate which means that r is not empty; if r is empty then exist is false and not exist is the negation of exist. So, it can be used to specify the query like find all courses

taught both in fall 2009 and spring 2010. So, all that you have to do earlier you did it by set membership.

So, now you are trying to do this by this exists. So, you are saying that this is again the same query which gives you the courses that are in spring 2010 and also in this fall 2009 and you check whether this relation, whether this particular nested query is an empty one or not. if it is an empty one, then exist will fail and the whole where clause will fail, if it was not an empty one then you have found such an entry it was offered and therefore, it will get included.

So, these are just different ways of expressing similar things, but what you should note is the nested sub query is a very convenient way to frame up the logic in multiple different ways that you would like to do. So, these are the different names given the correlation name and the correlated sub query. incidentally; the nested query is often referred to as the inner query and the query in which the nesting has happened, is known as the outer query.

(Refer Slide Time: 13:04)



Use of "not exists" Clause

- Find all students who have taken all courses offered in the Biology department.

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                   from course
                   where dept_name = 'Biology')
except
(select T.course_id
 from takes as T
 where S.ID = T.ID));
```

- First nested query lists all courses offered in Biology
- Second nested query lists all courses a particular student took


- Note: $X - Y = \emptyset \Leftrightarrow X \subseteq Y$
- Note: Cannot write this query using = all and its variants

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 08.17 ©Silberschatz, Korth and Sudarshan

Here is another example which illustrate the use of not exist; so which I leave it for your own study.

(Refer Slide Time: 13:15)



SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Test for Absence of Duplicate Tuples

- The **unique** construct tests whether a subquery has any duplicate tuples in its result
- The **unique** construct evaluates to "true" if a given subquery contains no duplicates
- Find all courses that were offered at most once in 2009


```
select T.course_id
from course as T
where unique (select R.course_id
              from section as R
              where T.course_id= R.course_id
              and R.year= 2009);
```

Database System Concepts - 9th Edition 08.18 ©Silberschatz, Korth and Sudarshan

We can check for uniqueness that is; test for absence of duplicate tuples by using the unique keyword. So, we can see here that there is a nested query and using unique to find out all courses that were offered at most once in 2009. So, if a course was offered more than once, then naturally multiple records will feature and the result the unique will fail. unique will be true only if; there is only one entry which shows that it is offered at most once in that semester.

Now, I move on. So, we have been discussing about sub queries in the where clause; now we move on to sub queries in the from clause.

(Refer Slide Time: 14:09)




Subqueries in the From Clause

- SQL allows a subquery expression to be used in the **from** clause
- Find the average instructors' salaries of those departments where the average salary is greater than \$42,000

```
select dept_name, avg_salary
from (select dept_name, avg (salary) as avg_salary
      from instructor
      group by dept_name)
where avg_salary > 42000;
```

(dept_name, avg_salary)

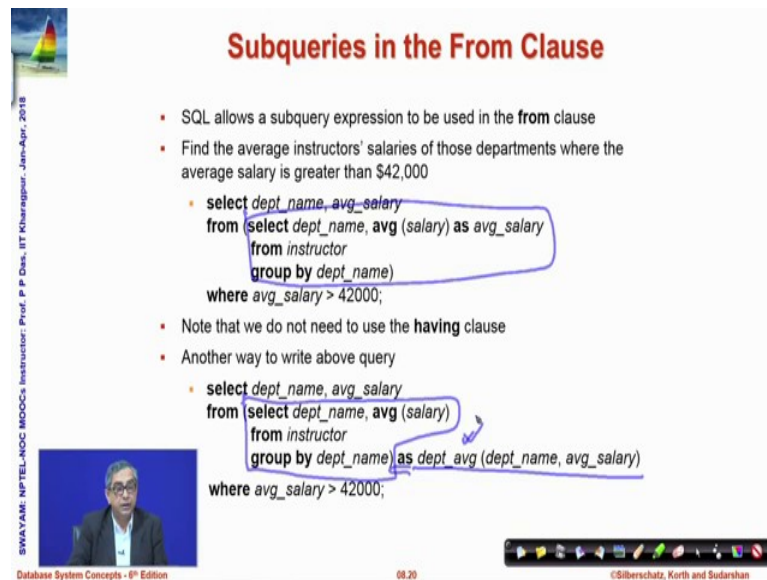


Database System Concepts - 9th Edition 08.29 ©Silberschatz, Korth and Sudarshan

So, as we have already seen a nested sub query is a relation. So, it can naturally be used in the place of any relation that we have in the from clause. So, we are trying to find out average instructor salaries of those departments where the average salary is greater than 42,000. So, look at. this is a nested sub query. So, where what is been found here this will compute the average salary department wise average salary which we have already seen group by department name and then you do average on the salary and you give it that field a new name.

So which means that; this is equivalent to having a relation which has two attributes dept name and avg_salary. So, from that you are now trying to do the selection and what is the condition that the average salary has to be written. So, you already have that as a part of the field the average salary. So, you just need to put that in the where clause and you have only those coming out of this particular relation where average salary is greater than 42,000 to be selected in this SELECT query. So, these will feature in the output of this selection.

(Refer Slide Time: 15:56)



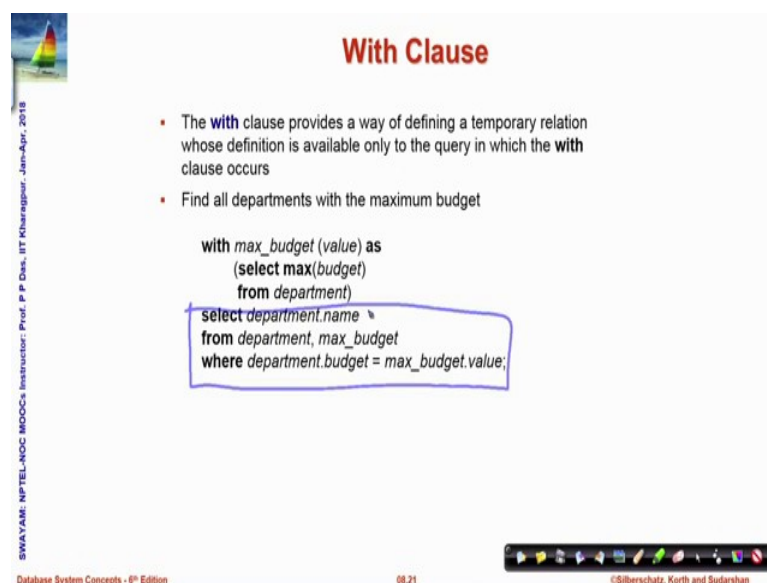
Subqueries in the From Clause

- SQL allows a subquery expression to be used in the **from** clause
- Find the average instructors' salaries of those departments where the average salary is greater than \$42,000
 - ```
select dept_name, avg_salary
from (select dept_name, avg (salary) as avg_salary
 from instructor
 group by dept_name)
where avg_salary > 42000;
```
- Note that we do not need to use the **having** clause
- Another way to write above query
  - ```
select dept_name, avg_salary
from (select dept_name, avg (salary)
      from instructor
      group by dept_name) as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

Database System Concepts - 9th Edition 08.20 ©Silberschatz, Korth and Sudarshan

So, that is how you can very easily use a nested sub query in the; from clause and for this we did earlier. We solve this problem using the having clause, but they here we will not need we did not need the having clause to do this there is this is another way. So, here what we have done is the same; if we if we look into the nested sub query. This is actually the same. all that we have done we have given it a new name by the renaming feature, and then this as if becomes a relation and on that the computation is done rest of it is similar.

(Refer Slide Time: 16:50)



With Clause

- The **with** clause provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs
- Find all departments with the maximum budget
 - ```
with max_budget (value) as
(select max(budget)
 from department)
select department.name
from department, max_budget
where department.budget = max_budget.value;
```

Database System Concepts - 9th Edition 08.21 ©Silberschatz, Korth and Sudarshan

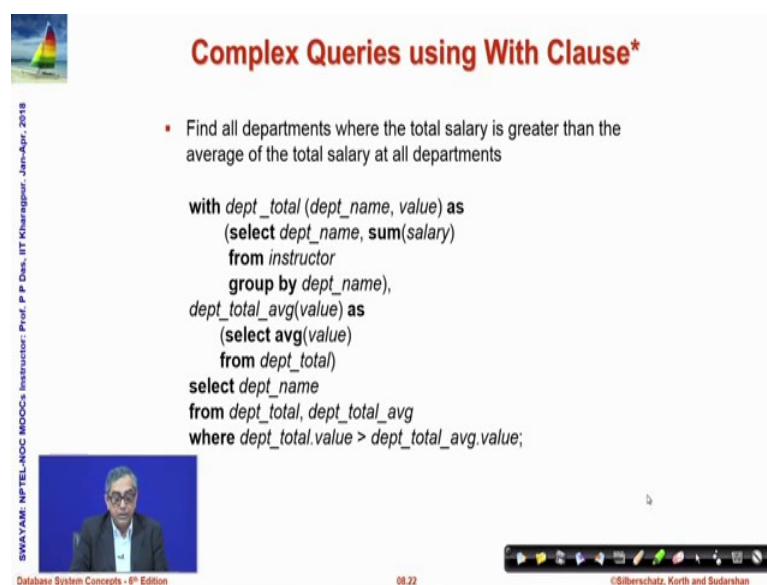
There is a with clause that provides a way of computing a temporary relation and that can be subsequently used.

So, let us look at example. we are trying to find all departments with maximum budget. So, this is my basic query, we want to find department name, department dot name, that is; a departments name from the department table and the budget must be same as maximum budget. So, for that I need to know the maximum budget that exists across the department. So, look at what has been done, here we have a nested query which aggregates the maximum budget from the departments.

So, this gives you the value of the maximum budget. We make that into a temporary table max budget with an attribute value. So, this is renaming. So, you cannot see the renaming is being used in very interesting ways. So, this is my nested query that gives me a relation and this is my definition of the relation. So, max budget now is a temporary relation a relation that I used subsequently in my from clause and with allows me to do that this relation will not be available.

Otherwise after this query this relation will not exist this is just a temporary one computed for this query. So, this gives me the budget value, this gives me the department and department specific budget, and this condition tells me that I can choose all the departments which has the maximum budget very nice way of using this.

(Refer Slide Time: 19:06)



**Complex Queries using With Clause\***

- Find all departments where the total salary is greater than the average of the total salary at all departments

```
with dept_total (dept_name, value) as
 (select dept_name, sum(salary)
 from instructor
 group by dept_name),
 dept_total_avg(value) as
 (select avg(value)
 from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.value;
```

SWAYAM: NPTEL-MOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr. 2018

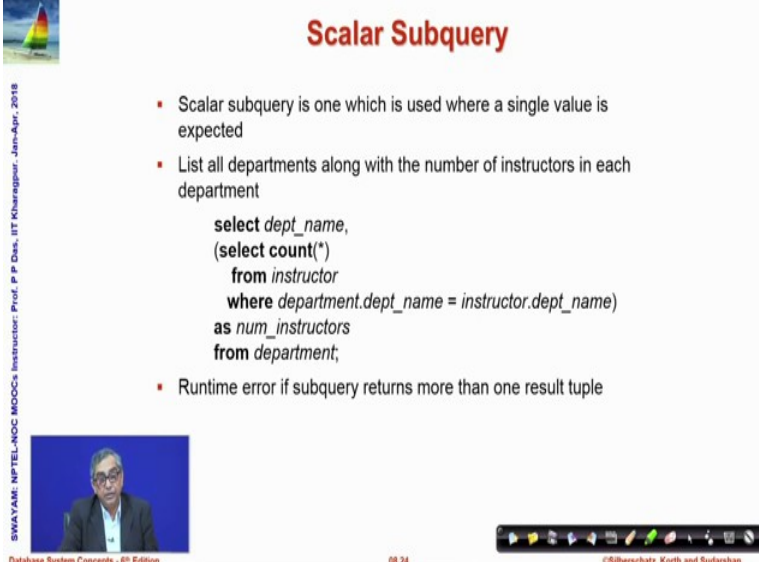
Database System Concepts - 9th Edition

08.22

©Silberschatz, Korth and Sudarshan

So, with clause can be used in even more involved way again this is an example which is more complex use and I leave it to you to practice study and understand. we move on to sub queries in the SELECT clause finally.

(Refer Slide Time: 19:26)



**Scalar Subquery**

- Scalar subquery is one which is used where a single value is expected
- List all departments along with the number of instructors in each department

```
select dept_name,
 (select count(*)
 from instructor
 where department.dept_name = instructor.dept_name)
 as num_instructors
from department;
```

- Runtime error if subquery returns more than one result tuple

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition

09.24

©Silberschatz, Korth and Sudarshan

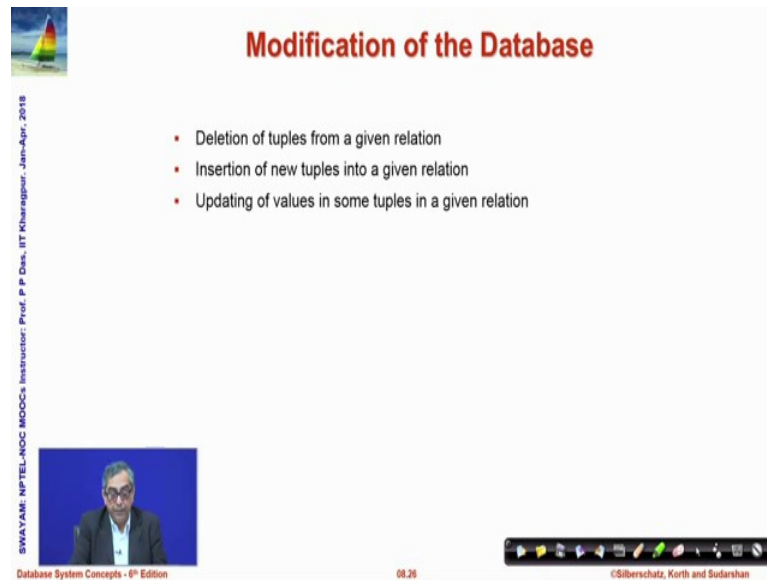
A scalar sub query is one; where there is a single value is expected. So, we can very easily use that in the SELECT. So, what if you look at this part which is the sub query? So, you are saying list all departments along with number of instructors each department has. So, this condition tells that, the from the instructor; we are taking out those that department name where the instructor works to count them and then you form that as a new attribute mind you in while we were using this in the from clause.

We were treating that as a relation because nested query will give a relation, but here in the SELECT clause the entities are attributes. So, this as is renaming of attribute which means, but this is a relation that is why this notion of scalar sub query is required, that is though this is a relation what does the relation compute it computes a single value. So, that value we are putting as an attribute named num instructors.

So, we have department name and the number of instructor, then for each and every department; that we actually have from the department list. So, it is a very interesting way of using this nested sub query in terms of the SELECT clause. naturally; since in the SELECT clause, I cannot have, I mean every entry in the SELECT clause has to be an attribute pure relations are not possible.

So, if the sub query returns more than one table which cannot be conceived as one or more attributes, then it will be runtime error that will not be allowed; because we do not know how to handle multiple tuples in terms of a select clause. ok. Next we move on to discussing the modifications to the database, how do we modify the database?

(Refer Slide Time: 22:12)



**Modification of the Database**

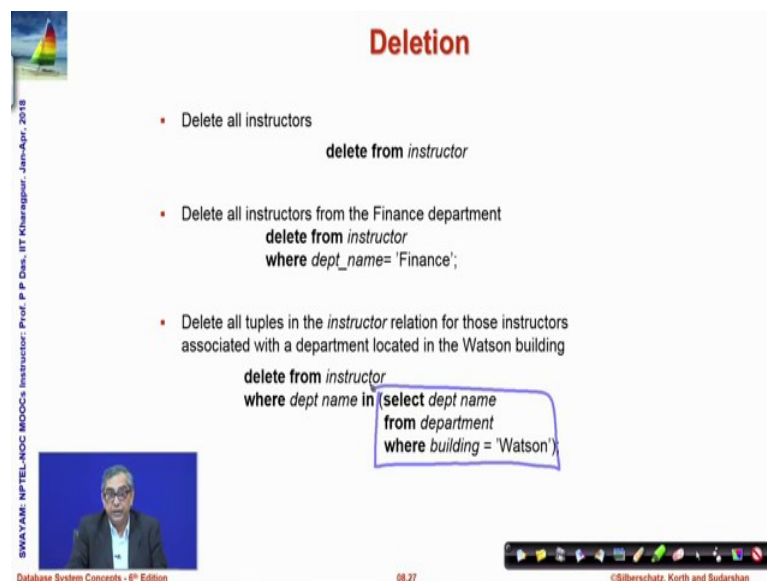
- Deletion of tuples from a given relation
- Insertion of new tuples into a given relation
- Updating of values in some tuples in a given relation

SWAYAM NPTEL-NOC MOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 08.26 ©Silberschatz, Korth and Sudarshan

So, we will look into some of the ways of changing the records or removing records from that earlier. We saw a delete of record which removed all records from a relation, but now we will see selective deletion insertion and update of values.

(Refer Slide Time: 22:33)



**Deletion**

- Delete all instructors  
`delete from instructor`
- Delete all instructors from the Finance department  
`delete from instructor  
where dept_name = 'Finance';`
- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building  
`delete from instructor  
where dept_name in (select dept_name  
from department  
where building = 'Watson');`

SWAYAM NPTEL-NOC MOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 08.27 ©Silberschatz, Korth and Sudarshan



Now, deleting all instructors are easy `DELETE FROM instructor` all records sorry this and this becomes an empty table, but suppose we want to delete all instructors from the finance department, then like we do in the `SELECT FROM WHERE` we again use the where clause as a predicate and say that `DELETE FROM instructor`, but you do the deletion provided this condition is satisfied that is; department name is same as finance. So, it is very similar to the `SELECT FROM WHERE`, but the effect is unlike `SELECT FROM WHERE`, where no tables change in the database here.

The table is actually changing; because these instructor records are deleted whose department name was finance. The third example shows delete all tuples in the instructor relation for those instructors, associated with the department located in the Watson building. So, Watson building may have multiple departments. So, all instructors who worked on those departments which are located in the Watson building that will go. So, you do, this is again you are using nested query.

Now, you know how to use a nested query. So, you use nested query which will give you the names of departments, which gives you a relation with a single attribute with names of departments housed in the Watson building, then you use the set membership to check whether a particular department belongs to that set, if it does then it is in Watson building; otherwise, it is not in Watson building if it does belong to the Watson building then this where clause becomes true and the corresponding instructor record is deleted and that is of this whole different kinds of selective deletion can happen.

(Refer Slide Time: 24:34)

**Deletion (Cont.)**

- Delete all instructors whose salary is less than the average salary of instructors

```
delete from instructor
where salary < (select avg (salary)
from instructor);
```

- **Problem:** as we delete tuples from deposit, the average salary changes
- **Solution used in SQL:**
  1. First, compute **avg** (salary) and find all tuples to delete
  2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

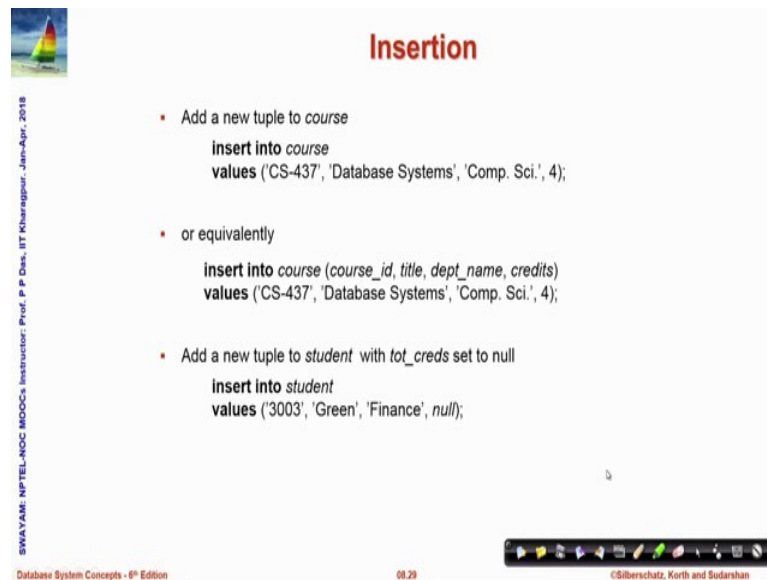
Database System Concepts - 9th Edition 08.28 ©Silberschatz, Korth and Sudarshan

Delete all instructors whose salary is less than the average salary of instructor. Again this is; so, you compute the selection sub query which computes the average salary and check if the salary is less than the average salary and delete that. sounds straightforward, but just wait, just wait, I mean did we do it do a right thing an average salary is computed by taking the sum of all salaries in the relation. And then dividing it by the number of relations this has to be the average salary certainly if I remove a record then the average itself will change.

So, if I write the query in this manner then what I am saying on the face of it looks correct, but then actually can it be correct because the moment a condition is satisfied and that record is deleted this average value itself has changed. So, that is not. So, that will depend then the result will depend on the order in which the deletion is happening, but that is not what was meant what was meant is take all the records for the present find out the average find out all records which have a salary less than that average and remove them.

So, this initially you know easy trivial looking solution is not actually correct. So, you will have to do the solution in two stages: first compute the average salary, find all tuples to delete? Next delete all tuples found above without re-computing. The average in the present solution the average is recomputed, which is the wrong thing.

(Refer Slide Time: 26:32)



**Insertion**

- Add a new tuple to *course*  
`insert into course`  
`values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);`
- or equivalently  
`insert into course (course_id, title, dept_name, credits)`  
`values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);`
- Add a new tuple to *student* with *tot\_creds* set to null  
`insert into student`  
`values ('3003', 'Green', 'Finance', null);`

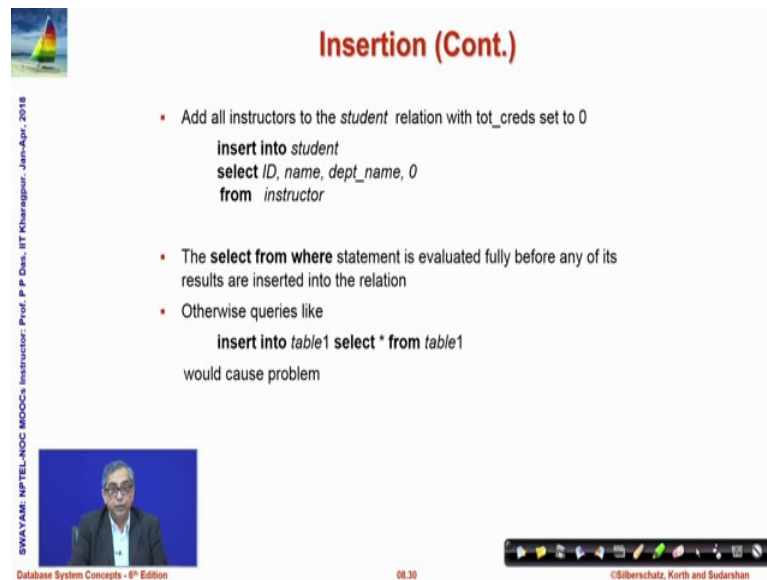
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 8th Edition 08.29 ©Silberschatz, Korth and Sudarshan

So, again I will leave that for you to solve. we move on to looking at modifications in terms of insertion. So, we had seen this earlier we can add a new tuple by inserting to then the relation names, and then you save values and the tuple of values. We can specify the; if we if we do not remember the order of attributes in the relation then we can also specify the order in which we are spaced actually giving the information.

So, you are saying INSERT INTO course and what we have done here is we have actually specified the order in which that tributes occurred and that order and the order of values must be the same. In the first case, this order of values is decided by the order of the attributes that exist in terms of the create table. Add a new tuple to student with total credit set to null that is I do not know; if we were adding a student initially does not have a credit right. The credit is a nullable field the credit will be earned after the student has gone through the courses and all that. So, if I do not know the value of a field then I can write null which is a special value designating unknown for at the time of insertion.

(Refer Slide Time: 28:00)



**Insertion (Cont.)**

- Add all instructors to the *student* relation with *tot\_creds* set to 0

```
insert into student
select ID, name, dept_name, 0
from instructor
```

- The **select from where** statement is evaluated fully before any of its results are inserted into the relation
- Otherwise queries like

```
insert into table1 select * from table1
```

would cause problem

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

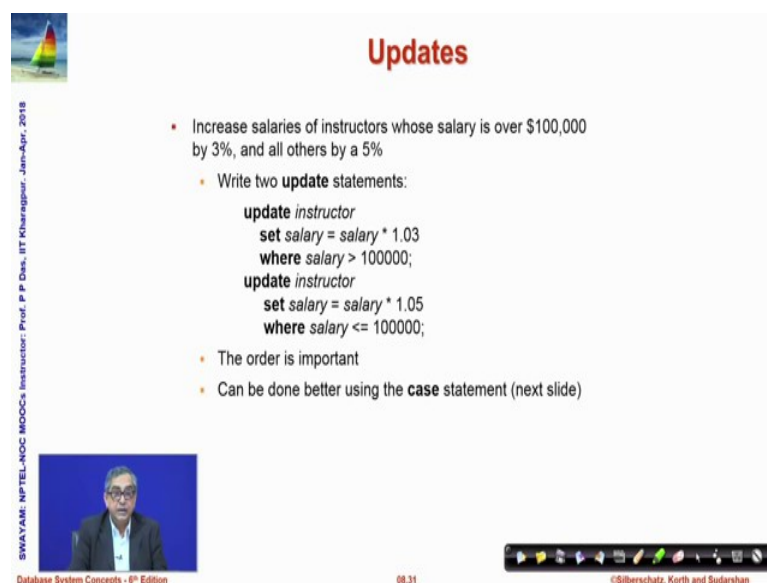
Database System Concepts - 9th Edition

08:30

©Silberschatz, Korth and Sudarshan

And all instructors to the student relation with total credit set to 0. So, I can also combine insert with select. So, we are taking the first part this part is selection which generates a whole lot of records having ID, name, department name and the total credit set to 0. From the instructor and insert them into the students. So, these will get inserted into the students SELECT FROM WHERE statement is evaluated fully. So, this first select from will be done before any of its results are inserted in the relation. So, that is the basic condition that SQL guarantees; because, if that were not the case then such situations will become circular and will cause problem.

(Refer Slide Time: 29:03)



**Updates**

- Increase salaries of instructors whose salary is over \$100,000 by 3%, and all others by a 5%

- Write two **update** statements:

```
update instructor
set salary = salary * 1.03
where salary > 100000;
update instructor
set salary = salary * 1.05
where salary <= 100000;
```

- The order is important
- Can be done better using the **case** statement (next slide)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition

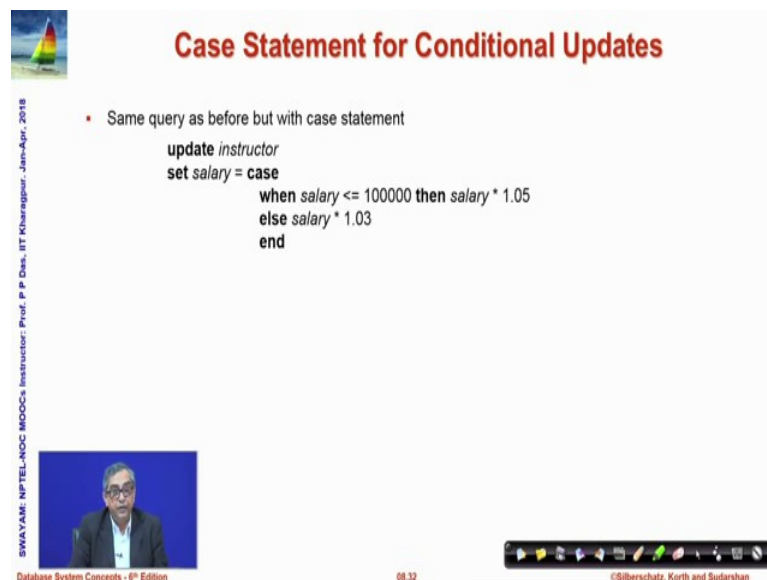
08:31

©Silberschatz, Korth and Sudarshan

Updates can be done based on particular values. So, you can update a table and what it means that? It you could update the values of specific fields.

So, here in the in the first case; we are giving trying to give a 3 percent salary raise for salaries which are more than 100,000 and some 5 percent raise for salaries which are less than equal to 100,000 and mind you this order in which you do the update is important, because if you do the later update first then someone who was what qualified in the later part was less than 100,000 with the increase will become more than 100,000 and will also qualify for the second one. So, that will become wrong. So, update often is dependent on the order.

(Refer Slide Time: 29:58)



**Case Statement for Conditional Updates**

- Same query as before but with case statement

```
update instructor
set salary = case
 when salary <= 100000 then salary * 1.05
 else salary * 1.03
end
```

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Khargpur, Jan-Apr, 2018

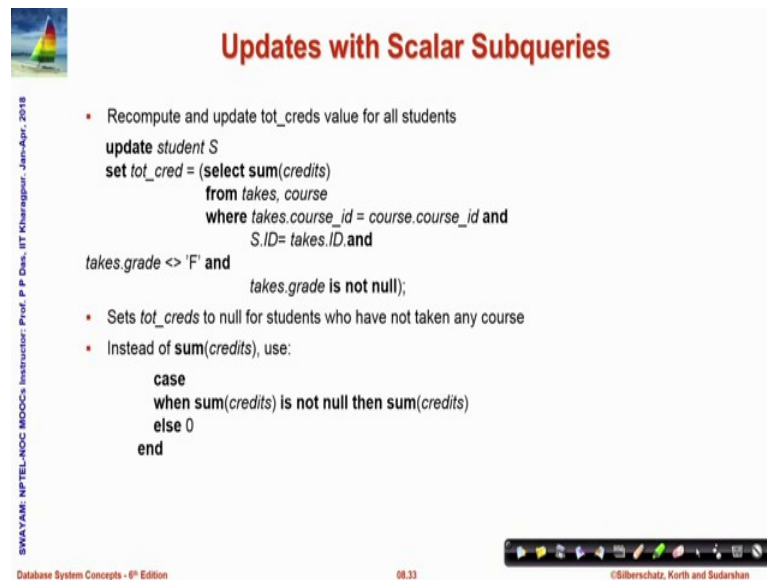
Database System Concepts - 9th Edition

08.32

©Silberschatz, Korth and Sudarshan

And therefore, you have yet another ah feature to take care of this when you have a specific order to do things it is called the case. So, you say when salary case is a new keyword, when is a key word when salary is less than equal to 100,000, then this is how you hike otherwise this is how you hike. So

(Refer Slide Time: 30:28)



### Updates with Scalar Subqueries

- Recompute and update tot\_creds value for all students

```
update student S
set tot_cred = (select sum(credits)
 from takes, course
 where takes.course_id = course.course_id and
 S.ID= takes.ID and
 takes.grade <> 'F' and
 takes.grade is not null);
```

- Sets tot\_creds to null for students who have not taken any course
- Instead of sum(credits), use:

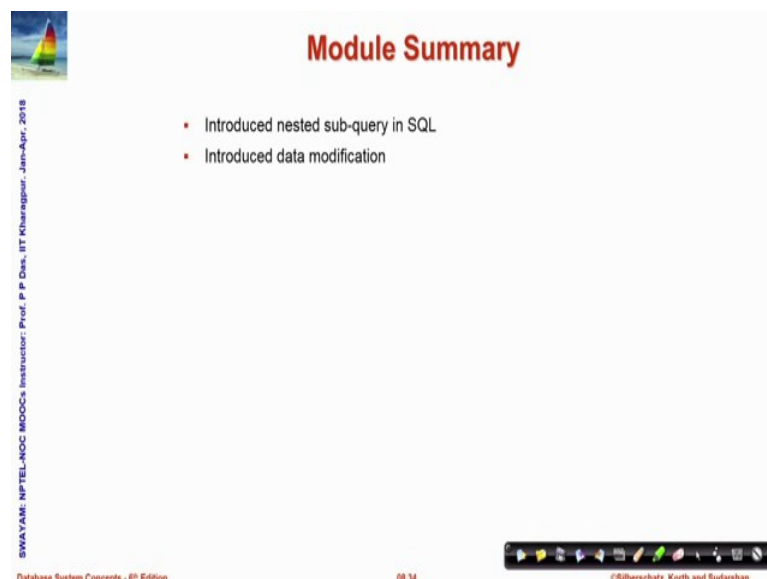
```
case
 when sum(credits) is not null then sum(credits)
 else 0
end
```

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition 08.33 ©Silberschatz, Korth and Sudarshan

It can it looks more like the; if statement of C, C++ you can do updates with scalar sub queries. We have seen scalar sub queries already. So, you can use a scalar sub query, again I would not go through that details please study and you will be able to understand.

(Refer Slide Time: 30:50)



### Module Summary

- Introduced nested sub-query in SQL
- Introduced data modification

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition 08.34 ©Silberschatz, Korth and Sudarshan

So, these are different examples. So to summarize, we have introduced a very powerful feature in SQL query known as nested sub query, where we can write a SELECT FROM WHERE expression as a part of the where predicate or as a relation in the from clause or as one or more collection of attributes in the select clause and it can be used in several

other places also. We have seen the ways to perform data modification in terms of deleting inserting and updating records. And we have also seen how nested sub query often may be very useful not only in terms of performing a query, but also in terms of performing certain data modifications.