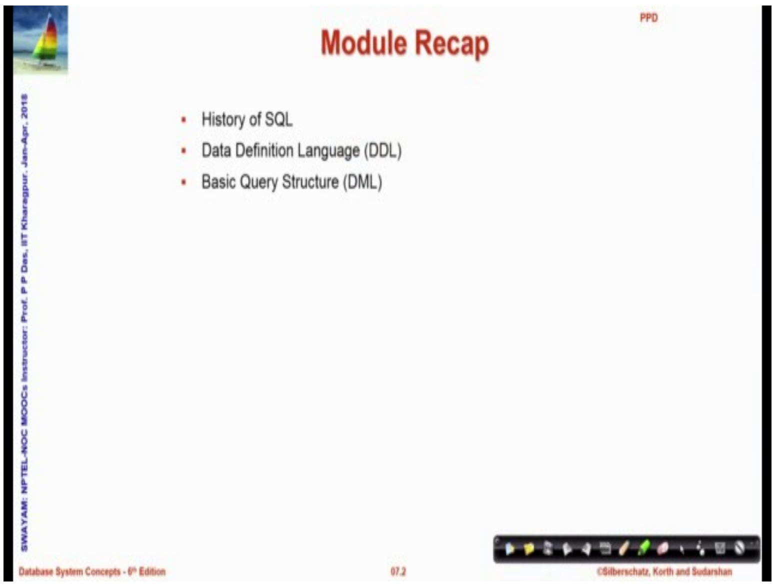
Database Management System
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 07
Introduction to SQL/2

Welcome to module 7 of database management systems, this is a second part out of total 3 of introduction to SQL.

(Refer Slide Time: 00:31)



Module Recap

- History of SQL
- Data Definition Language (DDL)
- Basic Query Structure (DML)

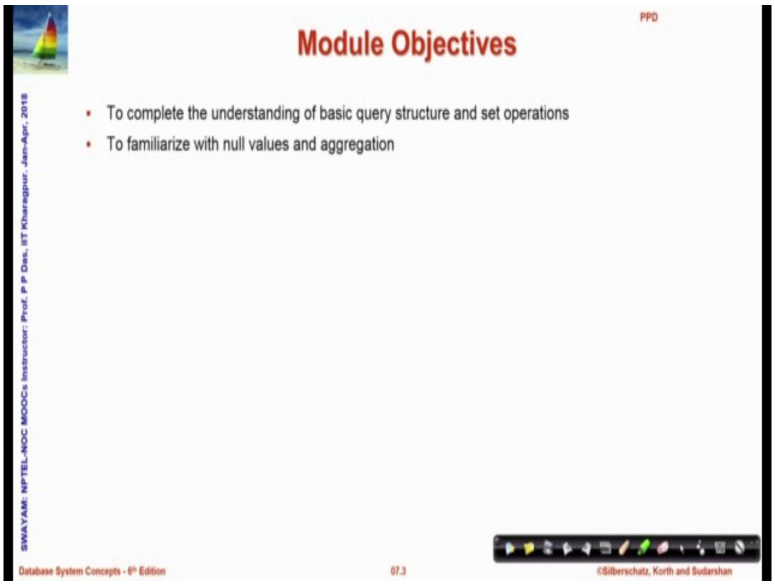
PPD

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

Database System Concepts - 6th Edition 07.2 ©Silberschatz, Korth and Sudarshan

In the last module, we have discussed about the evolution of SQL the data definition language part of it and the basic structure of queries.

(Refer Slide Time: 00:42)


Module Objectives

PPD

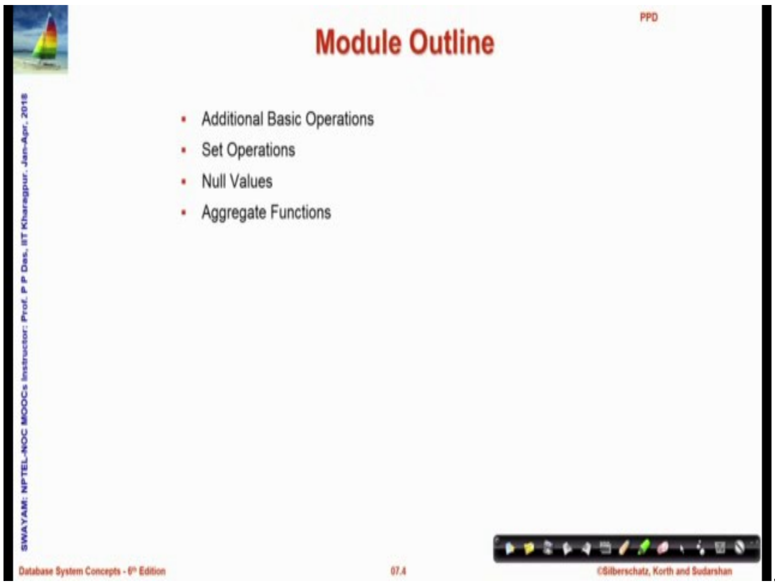
- To complete the understanding of basic query structure and set operations
- To familiarize with null values and aggregation

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

Database System Concepts - 6th Edition 07.3 ©Silberschatz, Korth and Sudarshan

In the current module we will complete the understanding of the basics queries structure and will see how, common set theoretic operations can be performed in terms of queries. We will familiarize ourselves with the handling of null values and aggregation operation that will be frequently required for forming queries.

(Refer Slide Time: 01:05)


Module Outline

PPD

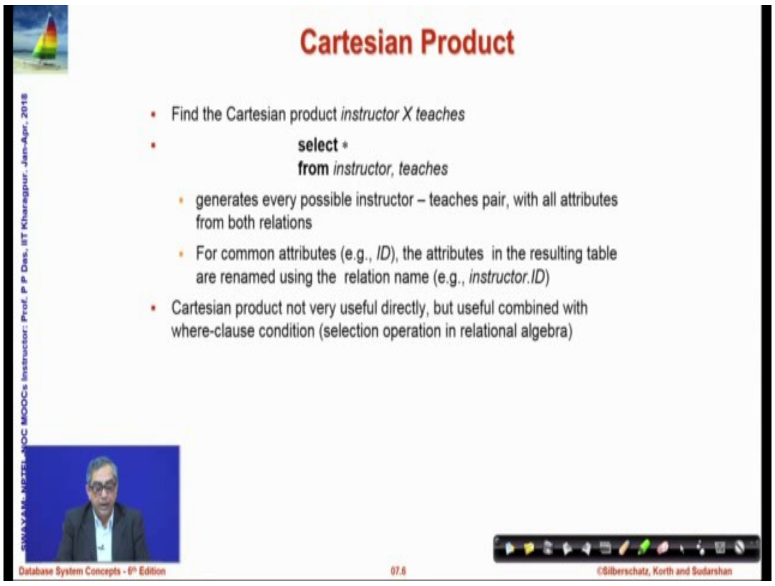
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions

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

Database System Concepts - 6th Edition 07.4 ©Silberschatz, Korth and Sudarshan

So, this is a module outline, these are the topics, that we will discuss. So, we started the discussion of some more basic operations in the query.

(Refer Slide Time: 01:15)



Cartesian Product

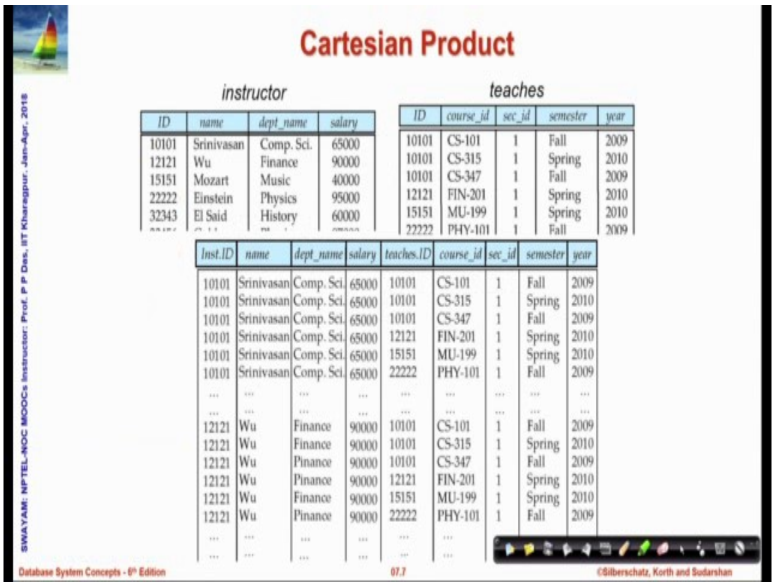
- Find the Cartesian product *instructor X teaches*
- **select** *
  from *instructor, teaches*
  - generates every possible instructor – teaches pair, with all attributes from both relations
  - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

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

Database System Concepts - 6th Edition 07.6 ©Silberschatz, Korth and Sudarshan

We have already discussed this that, if we do SELECT * FROM 2 tables then it results in a Cartesian product we have seen this result earlier..

(Refer Slide Time: 01:26)



Cartesian Product

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

teaches

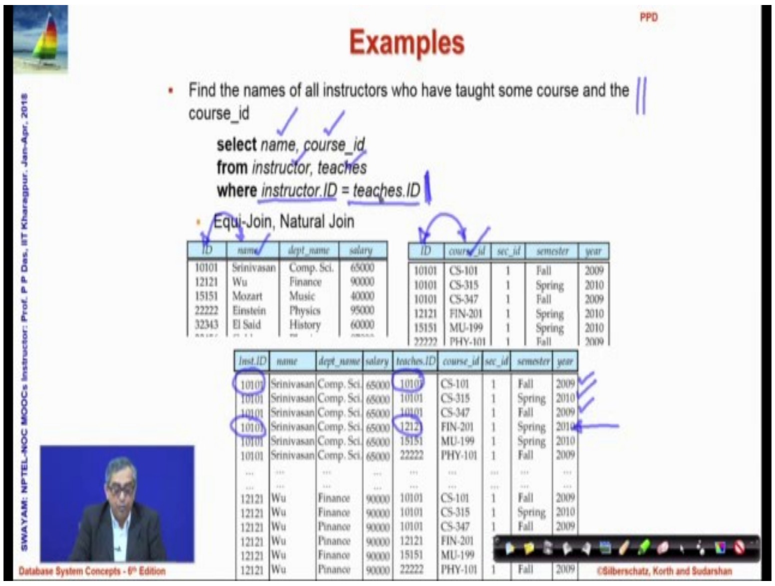
ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

Inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...			
...	...	...	...	...	...			

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

Database System Concepts - 6th Edition 07.7 ©Silberschatz, Korth and Sudarshan

(Refer Slide Time: 01:32)



PPD

Examples

- Find the names of all instructors who have taught some course and the course_id

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID
```

- Equi-Join, Natural Join

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

Inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	[illegible]	[illegible]	[illegible]
12121	Wu	Finance	90000	15151	MU-199	[illegible]	[illegible]	[illegible]
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009

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

Database System Concepts - 6th Edition

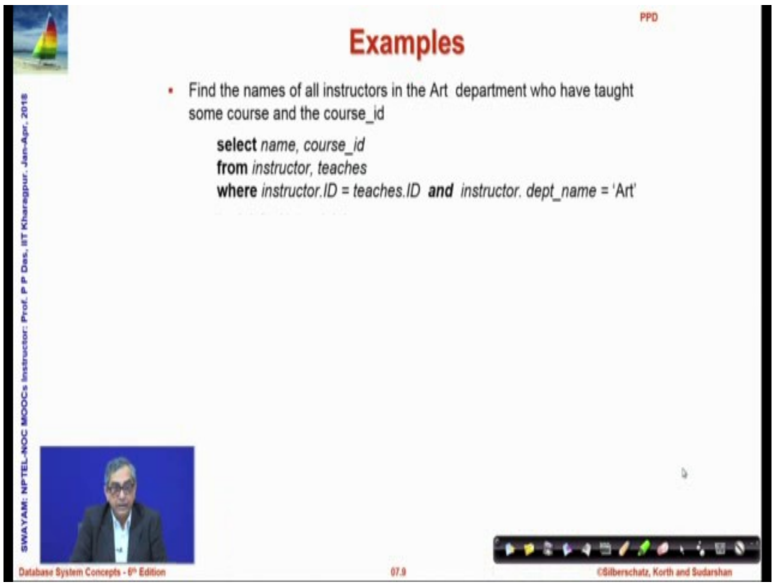
©Silberschatz, Korth and Sudarshan

Know by itself as we had said that, by itself the Cartesian product may not be very useful, but suppose we want to answer this kind of a query, that find all names of all instructors who have taught some course and for those you also write the course_id. So, what we are interested in is, the name of the instructor and course_id.

So, we put that on the select course these are the 2 tables that are required because, the name of the instructor is there in the instructor table relation and then the relationship between which instructors teach which course is in the teachers table. So, in the instructor table we have the name in the teachers table, we have the course_id and also this relationship as to which course is taught by which teacher and here, we have the relationship between which id of the instructor has which name. So, what we do is, when we do this Cartesian product will get something like this, as we have already seen, but we want to qualify it by with a predicate which say that, we will out of all these combinations will choose only those where, the instructor.ID equals the teaches.ID. So, if we look at say in the first row the instructor.ID is this, teachers.ID is this, which is id of the teachers table they are same. So, which says the this instructors Srinivasan actually teaches the course CS-101 whereas, if you look into this row, it says that instructor.ID is this and teaches.ID is this and we are really not interested in this combination, because this combination does not convey anything meaningful. So, by the use of this where clause, we will try to choose only those records where, these 2 ids are same which will tell us that, this particular instructor actually teaches that particular course.

So, if we do that, then you will find that majority of the records that, actually came about in the Cartesian product are eliminated from this result. So, I have struck them out. So, you can currently see in this part you can see only 4 records the 3 courses taught by Srinivasan. So, and the one course taught by you. So, in the output you will have this name, this name part and this course_id part because, you are projecting on these 2. So, this will be the output table which will get generated and just to remind you, this is the notion of natural join that, we had discussed in relational algebra in this case we will actually call it equi join because, we are using a equality condition after the Cartesian product to join these 2 relationship. So, this is a very critical operation in many of cases in our database query system.

(Refer Slide Time: 05:23)



Examples

- Find the names of all instructors in the Art department who have taught some course and the course_id

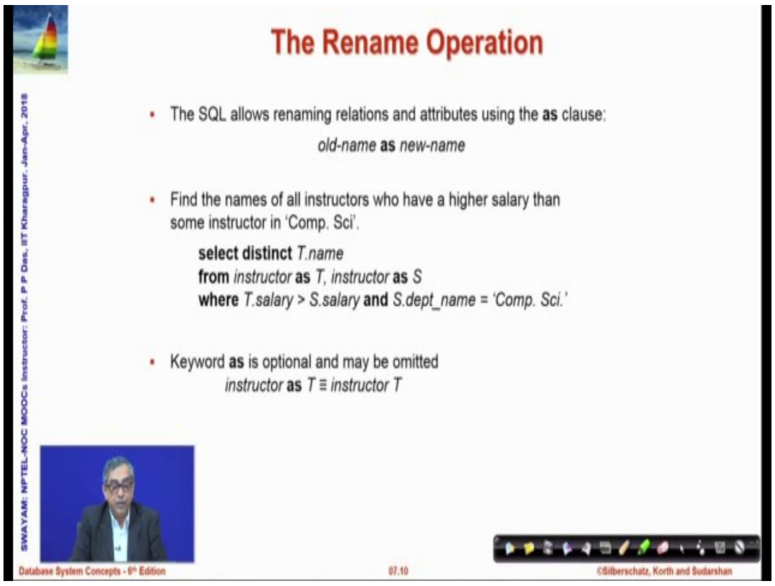
```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID and instructor. dept_name = 'Art'
```

PPD
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
07.9
©Silberschatz, Korth and Sudarshan

This is another extension of this similar example. So, here we have added another predicate in the where clause, specifying that instructed or department name is Art.

So, which means this will now, give the names of all instructors in the art department only who have taught some course and specify their course_id. So, in different such ways, you can manipulate and create queries.

(Refer Slide Time: 05:57)



The Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause:

  old-name **as** *new-name*

- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Comp. Sci.'
```

- Keyword **as** is optional and may be omitted

  instructor **as** *T* ≡ *instructor T*

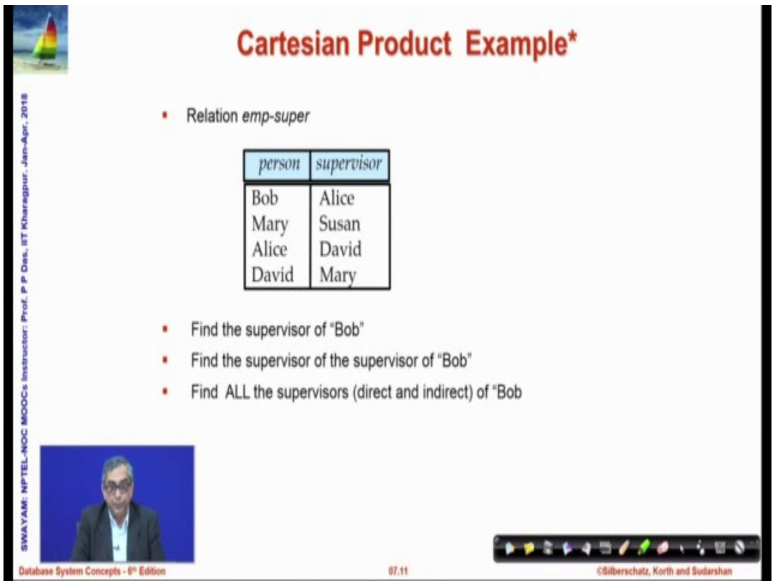
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 07.10 ©Silberschatz, Korth and Sudarshan

It is possible to rename, we have already seen examples you can rename a relation you can rename an attribute and the style is to use AS. So, here you can see that, in the SELECT query we have said from instructor as T. So, the name of this relation can be treated as T and we again see that as instructor as S. So, actually what we are doing, we are doing a join between the same relation; instructor and instructor.

And we are trying to find out all instructor who have higher salary than some instructor in computer science. So, the some instructor in computer science is specified by this condition, because department has to be computer science and the fact that salary is higher. So, as if you treat that, though it is actually a join between a product between instructor and instructor the same relation, but by renaming you treat them as if they are 2 different tables having name T and S, and then it becomes easier to write this kind of query. So, otherwise it is it is quite difficult to write this query to find out, because you need to actually create a product of one relation with itself.

(Refer Slide Time: 07:30)



Cartesian Product Example*

- Relation *emp-super*

person	*supervisor*
Bob	Alice
Mary	Susan
Alice	David
David	Mary

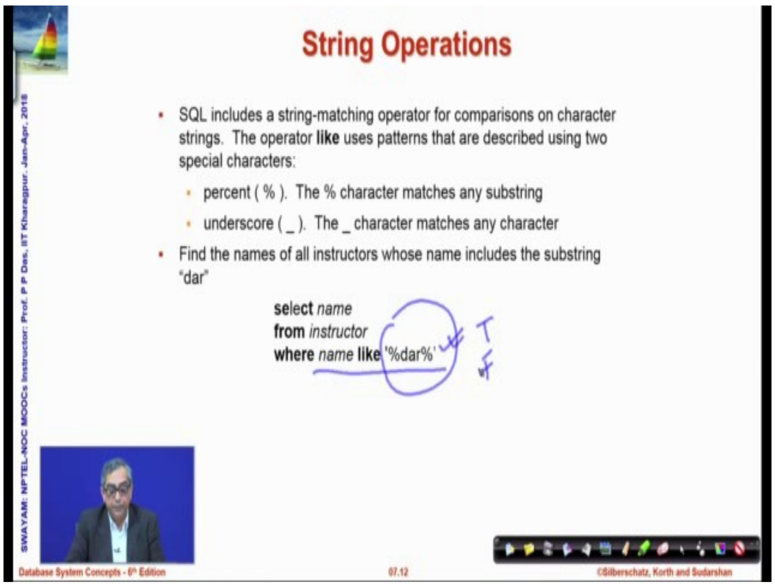
- Find the supervisor of "Bob"
- Find the supervisor of the supervisor of "Bob"
- Find ALL the supervisors (direct and indirect) of "Bob

Database System Concepts - 6th Edition 07.11 ©Silberschatz, Korth and Sudarshan

Keyword AS is optional you can just write instructor, and then the name the new name that you want to give and that itself will work here. Is another Cartesian product example, here given a relation which lists a person and his or her supervisor, we want to find out all supervisors direct or indirect of that person.

So, I leave this as an exercise to you, to think over as to how we can actually compute this query?

(Refer Slide Time: 07:56)



String Operations

- SQL includes a string-matching operator for comparisons on character strings. The operator **like** uses patterns that are described using two special characters:
  - percent (%). The % character matches any substring
  - underscore (_). The _ character matches any character
- Find the names of all instructors whose name includes the substring "dar"

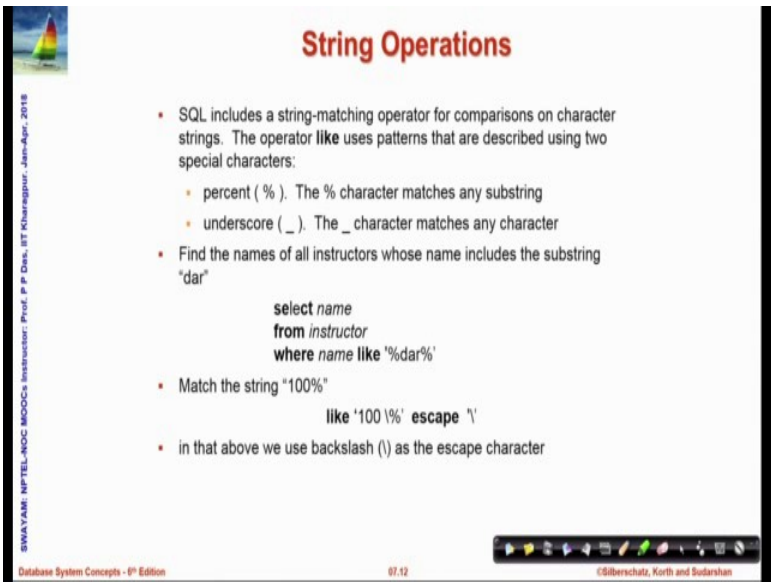
```
select name
from instructor
where name like '%dar%'
```

Database System Concepts - 6th Edition 07.12 ©Silberschatz, Korth and Sudarshan

SQL supports several string operations and of particular interest at 2 specific symbols, characters which allow us doing certain match, percentage is used to match any substring and underscore is used to match any particular character and we use a keyword LIKE, to find out different string patterns that can be matched. So, here we want to find the names of all instructors, whose name includes the substring "dar" and by writing this so saying the predicate is formed is named like this.

So, what is there is a percentage before there is a percentage after. So, anywhere "dar" will feature in the name, this predicate will turn out to be true otherwise if there is no "dar" in the name the predicate will turn out to be false and that particular record will not get selected. So, in this way we can using LIKE, we can actually do different kinds string operations as conditions in the weight loss.

(Refer Slide Time: 09:07)



String Operations

- SQL includes a string-matching operator for comparisons on character strings. The operator **like** uses patterns that are described using two special characters:
  - percent (%). The % character matches any substring
  - underscore (_). The _ character matches any character
- Find the names of all instructors whose name includes the substring "dar"

```
select name
from instructor
where name like '%dar%'
```

- Match the string "100%"

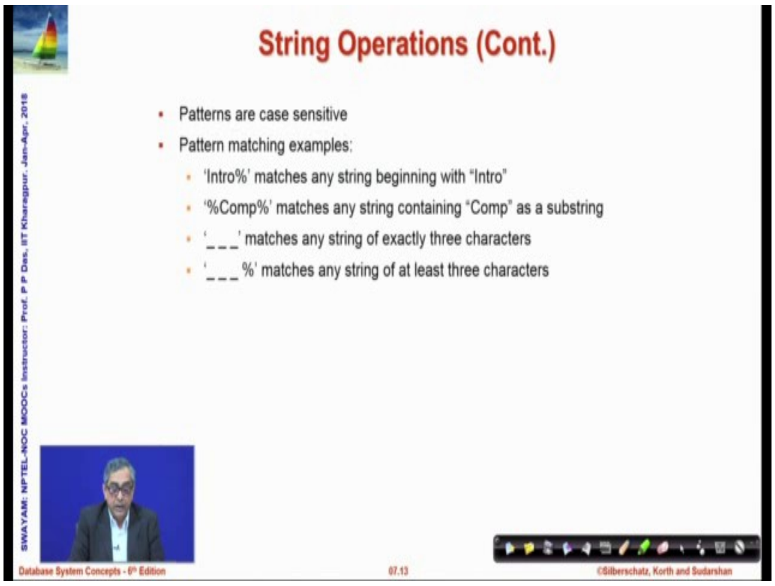
```
like '100 \%' escape '\'
```

- in that above we use backslash (\) as the escape character

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
07.12
©Silberschatz, Korth and Sudarshan

So now, naturally this brings in an issue of for what if my string itself has a percentage or an underscore character. So, the rule followed is you will need to escape that, with the escape character that you define. This is a style which you have seen in C programming as well.

(Refer Slide Time: 09:29)

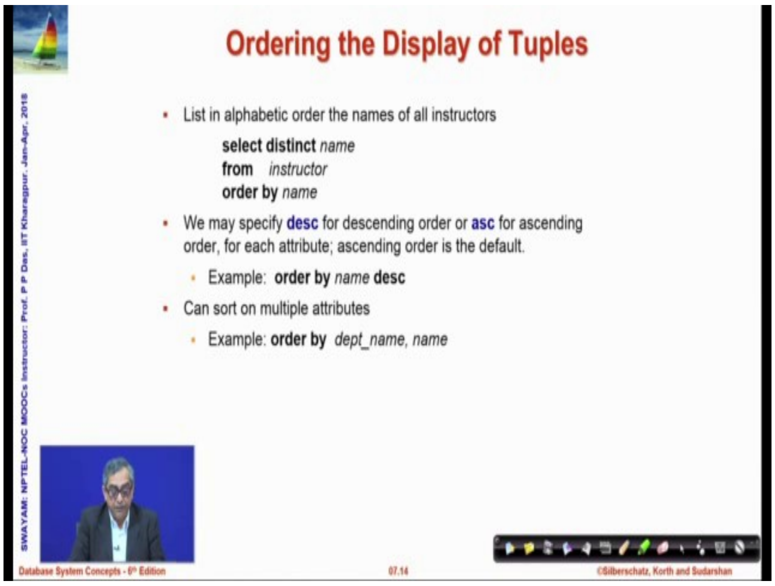


String Operations (Cont.)

- Patterns are case sensitive
- Pattern matching examples:
  - 'Intro%' matches any string beginning with "Intro"
  - '%Comp%' matches any string containing "Comp" as a substring
  - '_ _ _' matches any string of exactly three characters
  - '_ _ _%' matches any string of at least three characters

Patterns are certainly case sensitive. So, it depends it will distinguish between upper case as well as lower case, and these are different examples of string matching that you can do where, you can match at this beginning of a string, end of a string, anywhere in the string, specific number of characters in a string and so on. SQL supports concatenation, conversion of lower to upper case and vice versa and different other common string operations, those are available as functions in SQL and can be used for convenience.

(Refer Slide Time: 10:09)



Ordering the Display of Tuples

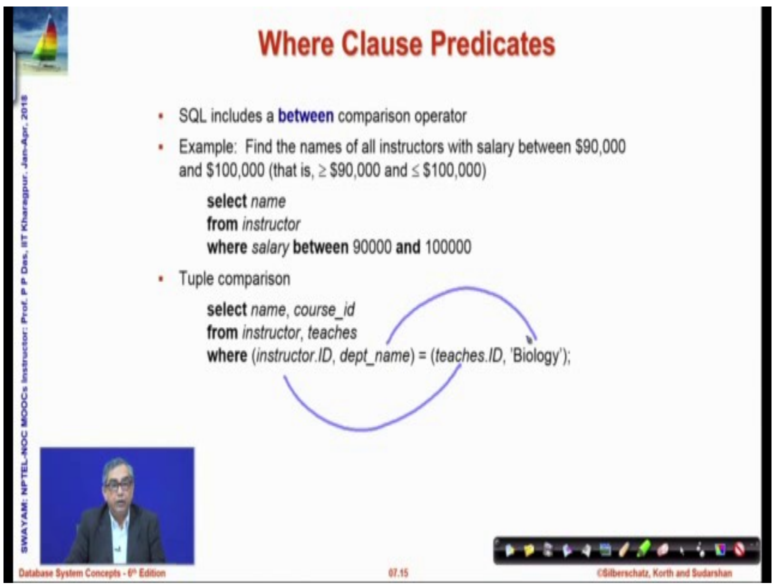
- List in alphabetic order the names of all instructors

  select distinct *name*
  from *instructor*
  order by *name*
- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - Example: **order by** *name* **desc**
- Can sort on multiple attributes
  - Example: **order by** *dept_name, name*

Now, let us address a different question. let us say, we have computed a query and then often we would want, that the result be ordered in according to certain order particularly the value of certain field if we want the result to be ordered, then SQL allows you to do that by another clause that you add to the query, which is called order by. So, what this will do, we have already seen this query this will find out the names of all the instructors and the names will occur in a distinct manner because, distinct is specified, but then the output will be in terms ordered by the name.

And the ordering can be descending or ascending, you can control that, by specifying whether you want descending or ascending by default the ordering is ascending. So, that makes the presentation of the result of and very easy and you can certainly sort on multiple fields as well, so it can be ordered based on combination of fields.

(Refer Slide Time: 11:15)



Where Clause Predicates

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between $90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)

```
select name
from instructor
where salary between 90000 and 100000
```

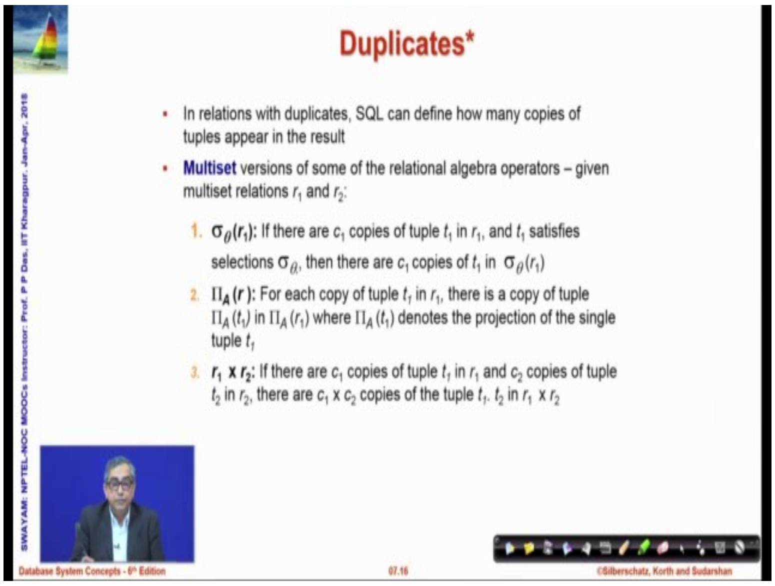
- Tuple comparison

```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
```

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
07.15
©Silberschatz, Korth and Sudarshan

SQL where clause also allows between as a comparison parameter. So, between can specify 2 values. So that, whenever the field value will be between these 2 given values the condition will be predicated will be taken to be true otherwise is taken to be false. You can compare based on people as well. So, in this case you could have written, you could have checked for equality of instructor.ID with teachers.ID and department name with the literal biology, but you can compact it by writing a tuple notation as is shown here. So, these are common convenient ways of writing different where clause.

(Refer Slide Time: 12:02)



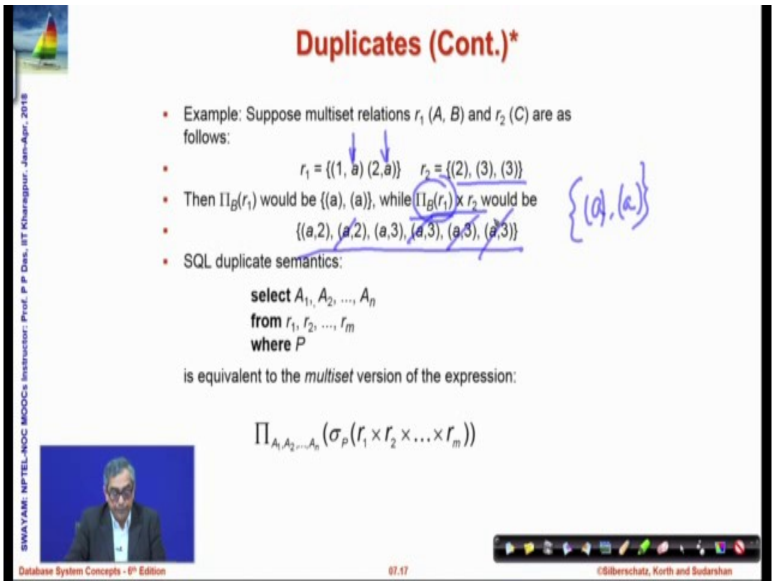
Duplicates*

- In relations with duplicates, SQL can define how many copies of tuples appear in the result
- **Multiset** versions of some of the relational algebra operators – given multiset relations r_1 and r_2:
  1. $\sigma_\theta(r_1)$: If there are c_1 copies of tuple t_1 in r_1, and t_1 satisfies selections σ_θ, then there are c_1 copies of t_1 in $\sigma_\theta(r_1)$
  2. $\Pi_A(r)$: For each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$ where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1
  3. $r_1 \times r_2$: If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2, there are $c_1 \times c_2$ copies of the tuple t_1. t_2 in $r_1 \times r_2$

Now, we have specified that SQL does carry duplicates. So, unlike relational algebra which set theoretically specify that, their duplicates should not be there if in SQL there could be duplicate entries in the same relation. So, there is a; this is called when duplicates are allowed in set theory, then such sets where duplicates are allowed and known as multi sets. So, there are multi set versions of the SQL queries or so to say the relational algebra operations.

So, you have a selection, which can be multi set selection, which means that, if there are certain c1 number of copies of a tuple in the relation, which satisfy the condition theta then all of them will feature in the result. And all those copies can be seen simultaneously, because it is a multi-set condition, similar definitions are hold for projection, as well as for Cartesian product. So, I will leave it to you to go through the details and convince yourself that, these multi set relations really extend the traditional single set distinct definition of the relational algebra.

(Refer Slide Time: 13:27)



Duplicates (Cont.)*

- Example: Suppose multiset relations $r_1(A, B)$ and $r_2(C)$ are as follows:
- $r_1 = \{(1, a)\ (2,a)\}$ $r_2 = \{(2), (3), (3)\}$
- Then $\Pi_B(r_1)$ would be $\{(a), (a)\}$, while $\Pi_B(r_1) \times r_2$ would be
- $\{(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)\}$
- SQL duplicate semantics:

select $A_1, A_2, \ldots, A_n$
from $r_1, r_2, \ldots, r_m$
where P

is equivalent to the *multiset* version of the expression:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

$\{(a), (a)\}$

Database System Concepts - 6th Edition 07.17 ©Silberschatz, Korth and Sudarshan

So, here is an example where, there are 2 multi set relations as you can see particularly this one, which has identical duplicate entries. So, using that you can define a projection of r1 on B, which will certainly give you it is you are doing projection on B. So, it will give you A only. So, you will have this result itself will be a multi set because you will get 2 as. So, this result will be like a a, and then you have r 2 with which you are doing the Cartesian product. So, you will have all possible combinations all these 6 are the result in the SQL whereas, in set theory typically the result should have been only these 2 tuples.

(Refer Slide Time: 14:38)

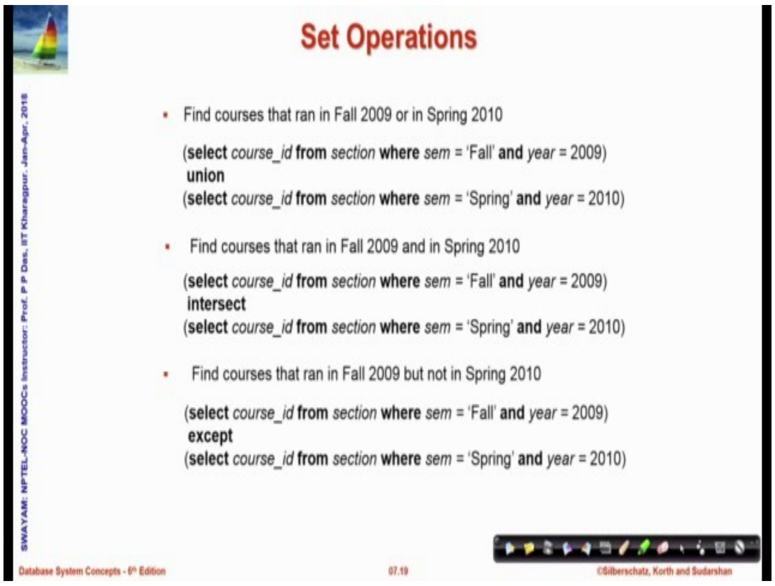
PPD

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

SET OPERATIONS

Database System Concepts - 6th Edition 07.18 ©Silberschatz, Korth and Sudarshan

(Refer Slide Time: 14:43)



Set Operations

- Find courses that ran in Fall 2009 or in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  union
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- Find courses that ran in Fall 2009 and in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  intersect
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- Find courses that ran in Fall 2009 but not in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  except
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

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

Database System Concepts - 6th Edition 07.19 ©Silberschatz, Korth and Sudarshan

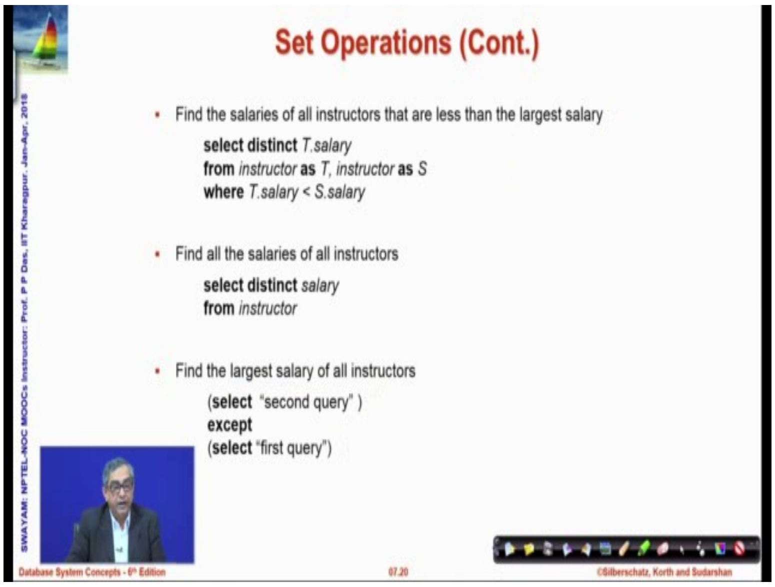
Now, we take a quick look into the common set operations. So, it is possible to do union, intersection, difference kind of operations very easily with SQL queries. So, suppose we want to find all courses, that ran in fall 2009 or in spring 2010. So, certainly the first part of the query is simple. This will give you all courses that run. So, you are taking out the course_id from section is where, the course running information is provided and you part putting 2 conditions, which say that they actually this courses ran in fall 2009.

So, this is the first query the second query says the courses, that run in spring 2010 and you are you have an or condition in the statement of what you are looking for. So, you do a union, union is another keyword. So, this will simply give you a relation of the course_id attribute as the only attribute, which has records from the first as well as the second query. Similarly, you can find out the courses that run, both in fall 2009 and spring 2010 by using intersect, which basically give you the intersection of the result of the first and the second query.

You could also do difference set difference by doing find courses, that ran in fall 2009, but not in 2010. So, what will that mean, that will mean that the result of the first query from the results of the second query be subtracted be done a difference from. So, those that, had run in the fall 2009 and then was again run in spring 2010 will get removed. So, that is done through the accept keyword.

So, in this way you can very easily do set operations, whenever that is easy to conceive; obviously, you can write these queries in several other different forms, but this is just to show you how set theoretic operations can be easily written.

(Refer Slide Time: 17:00)



Set Operations (Cont.)

- Find the salaries of all instructors that are less than the largest salary

```
select distinct T.salary
from instructor as T, instructor as S
where T.salary < S.salary
```

- Find all the salaries of all instructors

```
select distinct salary
from instructor
```

- Find the largest salary of all instructors

```
(select "second query" )
except
(select "first query")
```

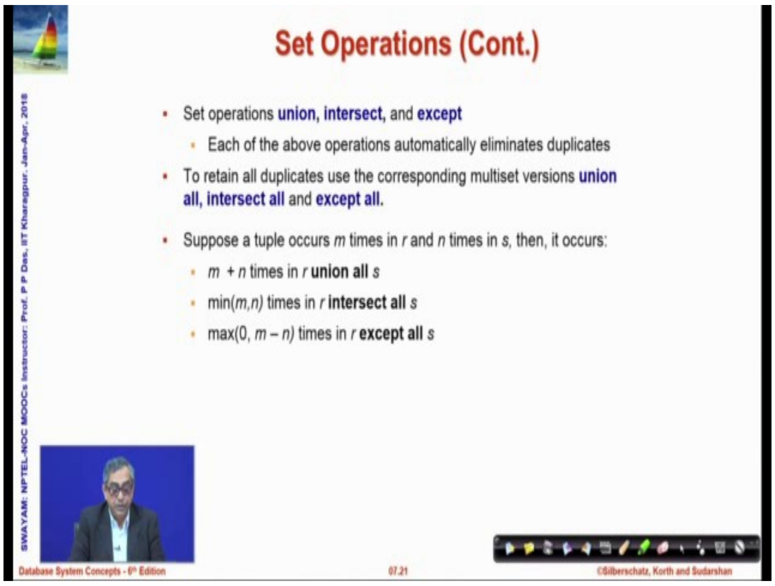
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 07.20 ©Silberschatz, Korth and Sudarshan

You can do set operations like this. in terms of fine salaries of all instructions that are less than a largest salary. So, again we are using renaming to think of the same relation as 2, and then as if from the relation T you are trying to look at relation S and finding out what are the salaries, which are smaller than that and certainly whatever comes out is the one which is not the largest because, certainly the largest will not satisfy this particular condition, because it will get compared with itself. You can find salaries of all instructors, and then you can find the largest salary.

So, this is all salaries which are less than largest, this is all salaries including the largest. So, what happens, if you subtract that is from this if you subtract this from all salaries, if you remove the salaries, that are not largest naturally what you get is a largest salary. So, this is a interesting way to find the largest salary we will see later on that there could be several other ways particularly the use of aggregate function, which make these computations easier to perform, but these are the typical ways to use set theoretic operations.

(Refer Slide Time: 18:29)



Set Operations (Cont.)

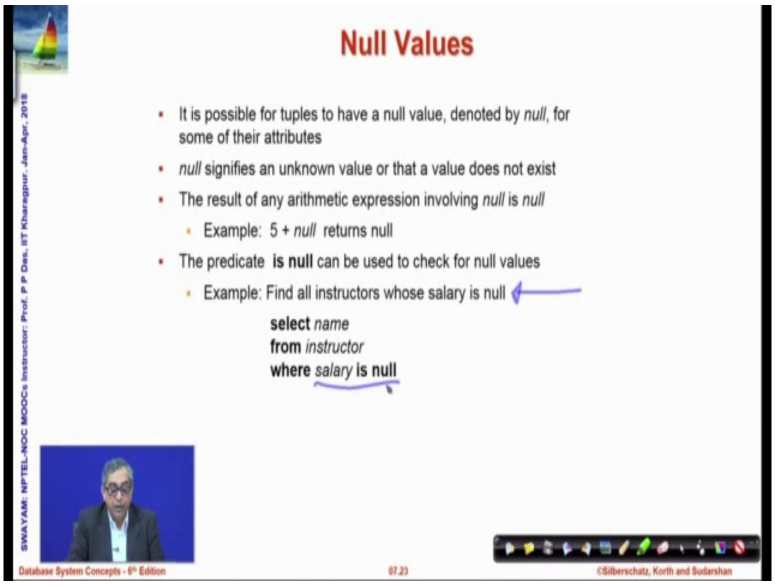
- Set operations **union**, **intersect**, and **except**
  - Each of the above operations automatically eliminates duplicates
- To retain all duplicates use the corresponding multiset versions **union all**, **intersect all** and **except all**.
- Suppose a tuple occurs *m* times in *r* and *n* times in *s*, then, it occurs:
  - *m* + *n* times in *r* **union all** *s*
  - min(*m*,*n*) times in *r* **intersect all** *s*
  - max(0, *m* – *n*) times in *r* **except all** *s*

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
07.21
©Silberschatz, Korth and Sudarshan

The set operations, so we have seen 3 of them union, intersect and except they automatically these operations are set theoretic. So, each of them automatically, eliminate the duplicate, unlike what SQL by default, SQL by default does what, allows duplicates, but set operations will eliminate duplicates, because they are set operations. So, if you want the SQL type of behaviour, if you want the duplicates to be preserved, then you can have a multi set version of these set operations, which are known as union all, intersect all, except all like that.

And naturally if you do these operations then, here is the simple formula of the number of tuples, that will get computed in different cases. you can study and convince yourself that these are the correct numbers. Let us, go to the treatment of we talked about null values, that we said that it is possible that certain records in a relation may have one or more attributes where, the value is not known and to represent, that the value is not known we are putting a placeholder called null.

(Refer Slide Time: 19:52)



Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist
- The result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- The predicate **is null** can be used to check for null values
  - Example: Find all instructors whose salary is null

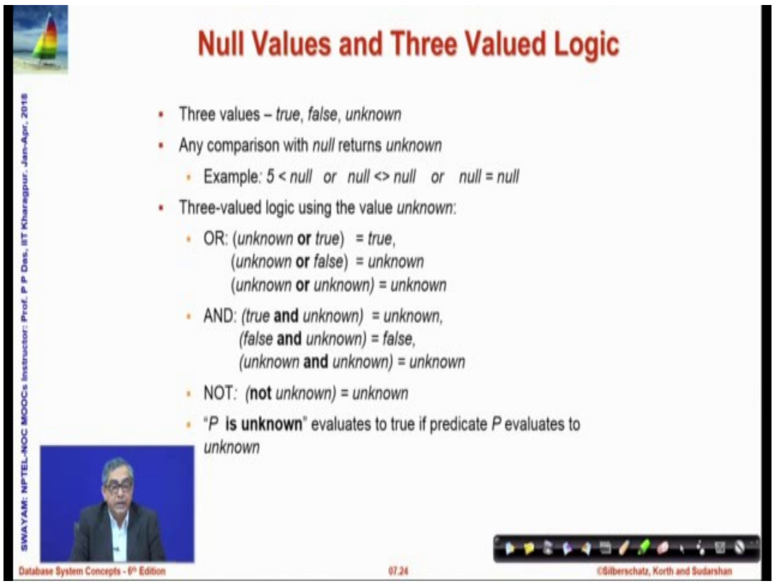
```
select name
from instructor
where salary is null
```

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
07.23
©Silberschatz, Korth and Sudarshan

So, let us see what is the consequence of that null value in terms of doing these query operations. So, the null signifies an unknown value. So, if I do 5 plus null then naturally the result is null. So, what you are saying that I am adding an unknown quantity to 5. So, then what would you say is the result is unknown. So, that is the basic semantics of adding null to a number. So, it is possible to check, if particularly a field is a null for a record, and that is done by a predicate "is null".

So, in this particular query we are trying to find all instructors, whose salary is null that is not known. So, this is a predicate. So, for a particular record for which salaries null, this will become true and that will get included in the result, but for all records for which, there is some value for the salary. So, salary is known it is not null those will not get included in the result.

(Refer Slide Time: 20:58)



Null Values and Three Valued Logic

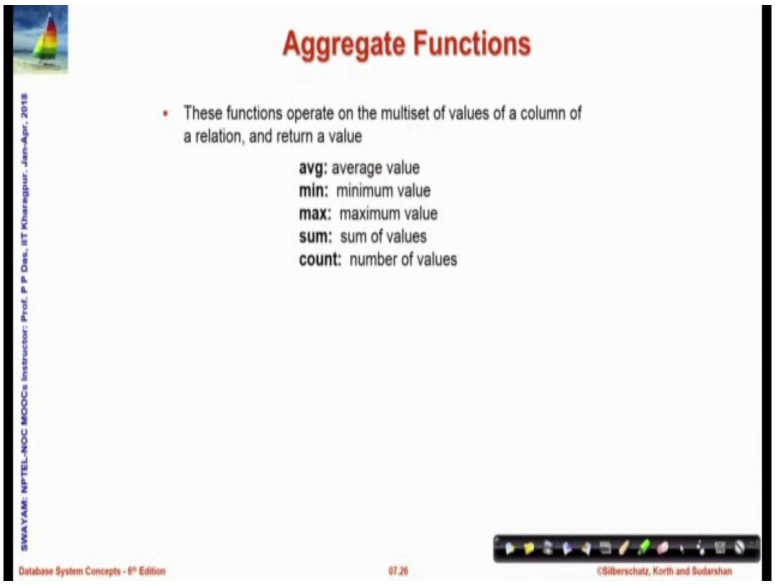
- Three values – *true*, *false*, *unknown*
- Any comparison with *null* returns *unknown*
  - Example: *5 < null* or *null <> null* or *null = null*
- Three-valued logic using the value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: (*true* **and** *unknown*) = *unknown*,
    (*false* **and** *unknown*) = *false*,
    (*unknown* **and** *unknown*) = *unknown*
  - NOT: (**not** *unknown*) = *unknown*
  - "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
07.24
©Silberschatz, Korth and Sudarshan

So, the basic semantics of null is then, combined with the truth values because, we know our basic predicate logic is 2 values true and false. But now, you have a third value unknown that is, you may not know the value of a predicate. So, how does it play around with the true and false values, you can reason through that quite easily if you are comparing with a null in whatever way, then naturally the result is unknown. So, it returns a null, if you are doing any connectives for example, if you are doing or of null or true, then the result should be true because, in or we say that if any of the components is true then the result is true.

So, here you do not need to know what is that unknown well you can say it is true, but if you do or with false or of unknown with false the second row or of unknown with false if you do this, then naturally this is unknown because, since this is false the result would be true only if unknown value is true and the result would be false if the unknown value is false, you do not know what that unknown value is. So, you have to say that you have result is unknown. So, using that same logic you could see verify, I would ask you to verify offline at home you please verify, that all these combinations of true false with unknown are valid. So, if P is unknown as a predicate will evaluate to true if P is not known..

(Refer Slide Time: 23:05)



Aggregate Functions

- These functions operate on the multiset of values of a column of a relation, and return a value

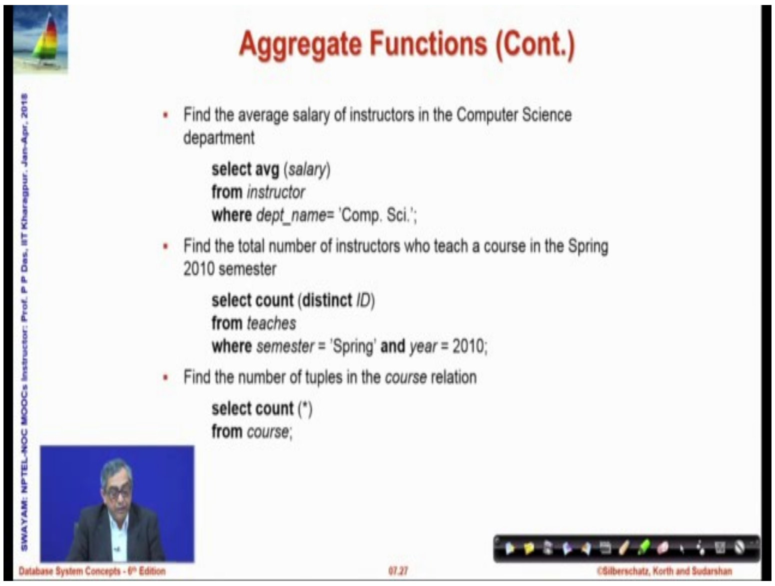
avg:	average value
min:	minimum value
max:	maximum value
sum:	sum of values
count:	number of values

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

Database System Concepts - 6th Edition 07.26 ©Silberschatz, Korth and Sudarshan

Now, we come to the aggregate functions, there are several aggregate functions they can be used for convenience and these are the common months, that operate on the multi set values naturally aggregate functions operate on a particular column they try to aggregate in a particular column and return a single value for example, average would be meaning, that you are trying to find average of the values of a particular column.

(Refer Slide Time: 23:28)



Aggregate Functions (Cont.)

- Find the average salary of instructors in the Computer Science department

```
select avg (salary)
from instructor
where dept_name= 'Comp. Sci.';
```

- Find the total number of instructors who teach a course in the Spring 2010 semester

```
select count (distinct ID)
from teaches
where semester = 'Spring' and year = 2010;
```

- Find the number of tuples in the *course* relation

```
select count (*)
from course;
```

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

Database System Concepts - 6th Edition 07.27 ©Silberschatz, Korth and Sudarshan

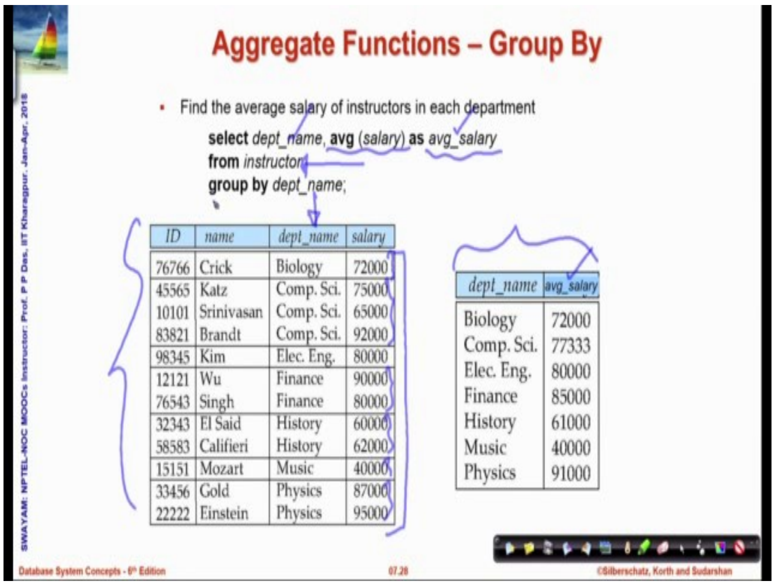
So, here is an example. So, we are trying to find the average salary of instructors in computer science department. So, naturally what you output is average salary. So, mind you this will output relation will have one attribute, which is average salary and since, average salary is a single quantity it will only have one, and here I have made use of this aggregate function average. So, it says you do average on the attribute salary and where do you get that attribute, from you get it from the table instructor and then we are saying that we are not interested to find average of salary of all instructors. We are interested to find the average salary of those instructors who work for computer science. So, you put this back clause. So, this will ensure, that you find the average salary of instructors in computer science department.

So, in similar way you can use other aggregate functions LIKE, if you want to know the total number of instructors who teach a course in this semester. So, you first put the where clause, naturally you where will you find this information you will find this information in teachers, teachers is the relation which tells you which instructor is teaching what course. So, that comes in from, then you have to specify that teaching a course in spring 2010 semester. So, the where clause specifies, that the semester is spring and the year is 2010. So, this will give you all records, which show that some instructor is teaching the course in spring 2010 semester.

Now, naturally there could be multiple the same instructor could happen multiple times, because an instructor may be teaching more than one course. So, you make the instructor.ID, instructor.ID that you have here, you make that distinct. So, that you get only those instructors, every instructor who is teaching one course or more than one course will feature only once in this total list, and then you simply count it. use aggregate function count on that.

So, that will tell you how many instructors are teaching some course in spring 2010 mind you, this is critical to use this key word distinct, because unless you use that, then all that you will eventually find out is not the number of instructors who are teaching the course you will find out the number of courses, that are being offered in spring 2010 because, there could be the same instructor teaching more than one course. If you just want to count the number of tuples you can do count on star. because, what is star is all the attributes. So, from if we want to find out the number of course, you suggest to count * on course.

(Refer Slide Time: 27:03)



Aggregate Functions – Group By

- Find the average salary of instructors in each department

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

ID	name	dept_name	salary
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	avg_salary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
07.28
©Silberschatz, Korth and Sudarshan

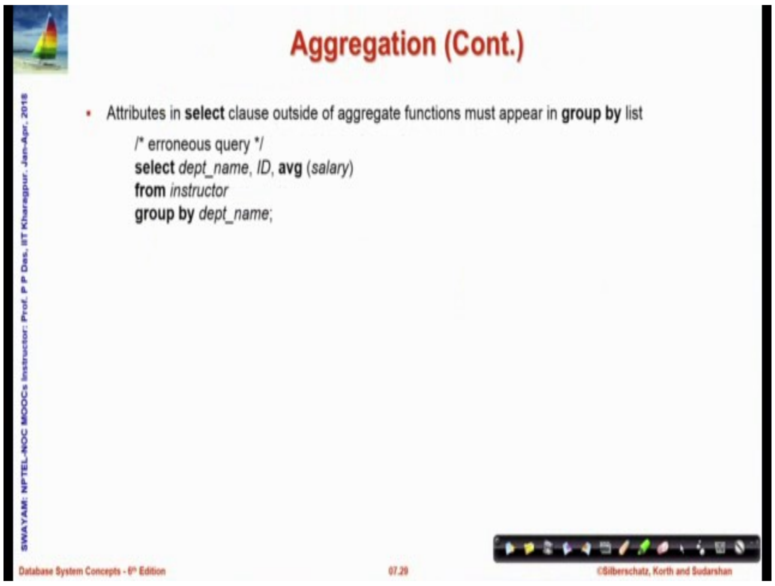
So, this is showing you the computation of average salary of instructors in each department.

So now, what do you want to do is earlier you try to find out the average salary in one department. Now, you want that for all the departments for each department I want. So, my result now, is not a single row, it is not a single value it is a pair where, I show the department and the average salary in that department. So, this is what I want visible and this is what I have.

So, naturally the information comes from instructor, that is from what I want is a department name and the average salary and I want to give it a nice name avg_salary. So, I have done a rename. So, I get avg_salary here, but then what I want is I do not want an average d1 over this whole set of fields. I want separate average to be done here, to be done here, to be done on this to be on this. So, these are these are basically groupings by the department as you can see, that this particular relation has been sorted according to the department name.

So, when I want to do apply an aggregate function on certain subgroups of records, I use this particular clause GROUP BY, and use a name of a field. So, what it does is if the values in the GROUP BY field in this case department name are identical those records are put together and over those records and average is computed. So, the average, that is computed over these records are put in here, average that is computed in terms of these records are put in here, there is only one record, so average that is computed in terms of that is put in here. So, GROUP BY is a very useful feature along with the aggregation functions and it allows you to club information based on certain attribute and then compute the aggregation on some other field.

(Refer Slide Time: 29:27)



Aggregation (Cont.)

- Attributes in **select** clause outside of aggregate functions must appear in **group by** list

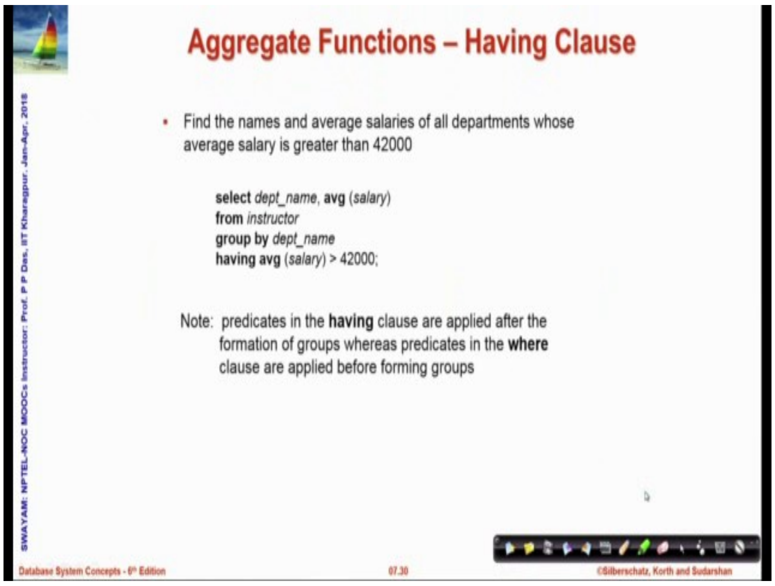
```
/* erroneous query */
select dept_name, ID, avg (salary)
from instructor
group by dept_name;
```

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

Database System Concepts - 6th Edition 07.29 ©Silberschatz, Korth and Sudarshan

Mind you, you will have to do GROUP BY and create the result id result table you will have to make sure that, all your result table attribute are used in the GROUP BY, which is not an aggregate function. So, here ID is not used. So, this is not a query, that SQL would support.

(Refer Slide Time: 29:53)



Aggregate Functions – Having Clause

- Find the names and average salaries of all departments whose average salary is greater than 42000

```
select dept_name, avg (salary)
from instructor
group by dept_name
having avg (salary) > 42000;
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

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

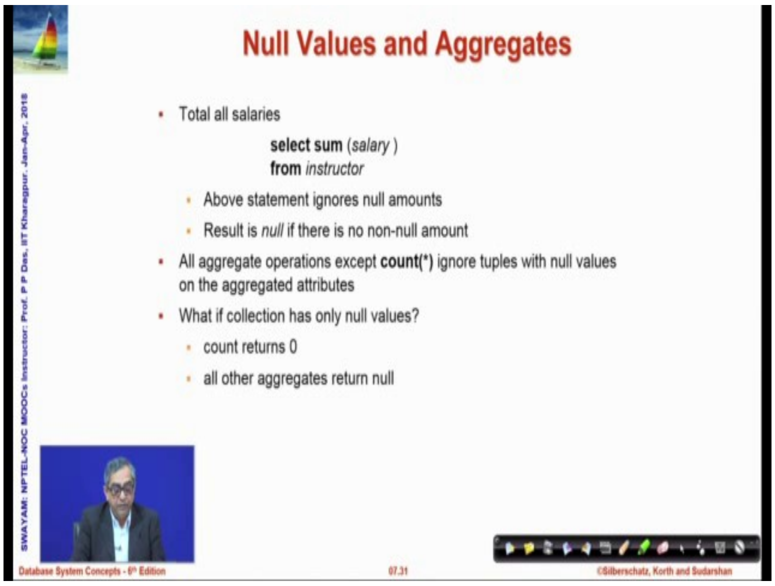
Database System Concepts - 6th Edition 07.30 ©Silberschatz, Korth and Sudarshan

You can further refine your result is saying that, fine names and average salary of all departments; this much you have already done.

Now, you are qualifying that, whose average salary is greater than 42000. So, of all that we have for example, if we look in here for example, in this music department average salary is less than 42000. So, you do not want that in the result you want only those where, the average salary is greater than 42000 and the way to do that is to add another clause called HAVING they say that, the average salary is greater than 42000. So, you are adding another predicate for actually qualifying the aggregated value. Now, the HAVING clause actually applies after along with the GROUP BY because, naturally the HAVING relates to the grouping.

So, once the grouping has happened groups have been formed then the HAVING clause will be evaluated on that, in contrast where clause also has a predicate, but the where clause is applied before forming the groups. So, this point this note has to be understood carefully because, if you have a where clause to choose the records they it will first apply, then out of those records chosen the grouping will happen and once the grouping has happened, then the aggregate function will evaluate and the HAVING clause will get evaluated, the predicate of HAVING clause will get evaluated.

(Refer Slide Time: 31:45)



Null Values and Aggregates

- Total all salaries

  select sum (*salary*)
  from *instructor*

  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

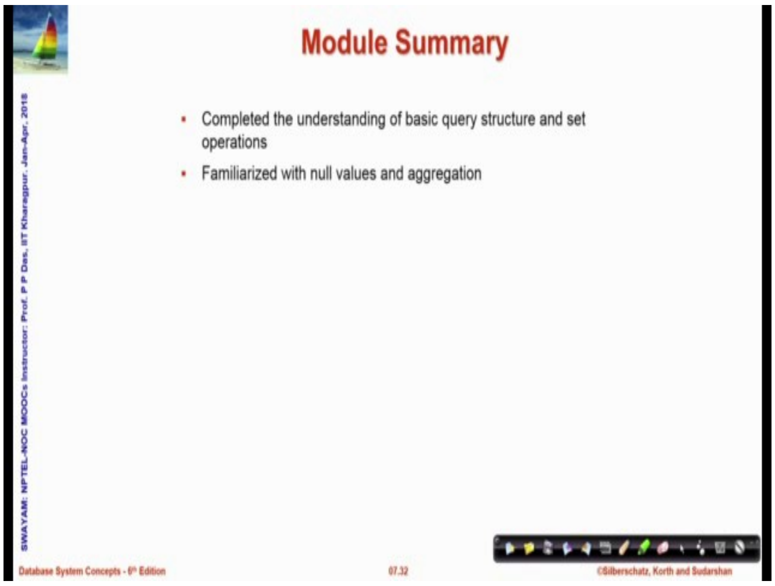
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 07.31 ©Silberschatz, Korth and Sudarshan

Certainly, if there are null values in terms of aggregates, then there is a question of what will happen. So, the general strategy is that, whenever you perform aggregation then the null values are all ignored. So, if on that field there is no value which is not null, that is if all values are null then the result is null otherwise the result is computed by ignoring the null values.

So, these are what do you have of course, if you count then, if the collection has only null values the count will return you 0, but all other aggregates will return you simply null.

(Refer Slide Time: 32:29)



Module Summary

- Completed the understanding of basic query structure and set operations
- Familiarized with null values and aggregation

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

Database System Concepts - 6th Edition 07.32 ©Silberschatz, Korth and Sudarshan

So, to summarize we have started the basic understanding of the basics query structure in the last module. Now, we have completed that with some more additional operations, we have understood the set theoretic operations and very importantly we have familiarized with the treatment of null values and aggregation functions particularly the GROUP BY and HAVING clauses and how do null values and aggregation interact in terms of an SQL query.