

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

**Lecture-06**  
**Introduction to SQL/1**

Welcome to module 6 of database management systems, this is the starting of week 2. In week 1 we have done 5 modules, after the overview of the course, we have primarily introduced the basic notions of DBMS and we have discussed about the relational module, the fundamentals of it.

(Refer Slide Time: 00:28)

PPD

## Week 01 Recap

- **Module 01: Course Overview**
  - Why Databases?
  - KYC: Know Your Course
- **Module 02: Introduction to DBMS/1**
  - Levels of Abstraction
  - Schema & Instance
  - Data Models
  - DDL & DML
  - SQL
  - Database Design
- **Module 03: Introduction to DBMS/2**
  - Database Design
  - Database Engine, Users, Architecture
  - History of DBMS
- **Module 04: Introduction to Relational Model/1**
  - Attribute Types
  - Relation Schema and Instance
  - Keys
  - Relational Query Languages
- **Module 05: Introduction to Relational Model/2**
  - Relational Operations
  - Aggregate Operations

Database System Concepts - 6th Edition 96.2 CS/Bierschalt, Korth and Sudarshan

(Refer Slide Time: 00:54)

**Module Objectives**

- To understand relational query language
- To understand data definition and basic query structure

Database System Concepts - 6th Edition

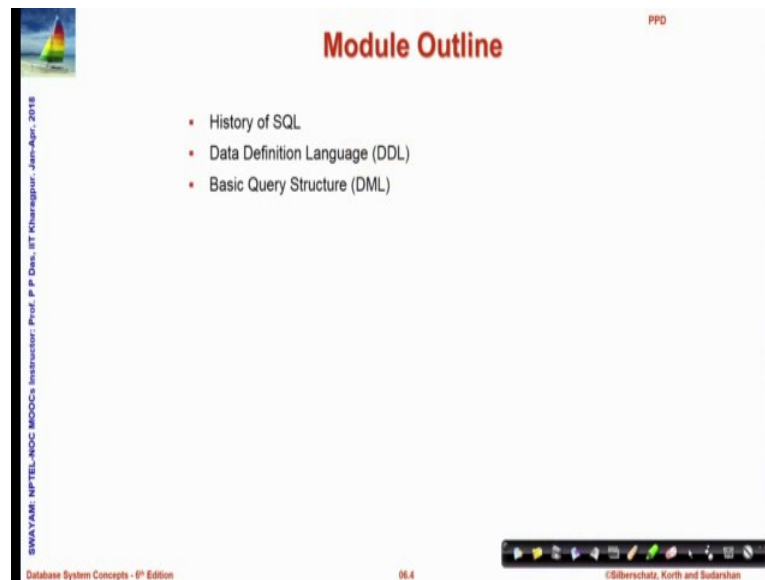
06.3

CS509: Introduction to Database Systems

In this background, in this week we are primarily focusing on the query language the structured query language SQL. So, all the 5 modules will relate to discussions on query language and this module 6 7 and 8, the 3 modules will introduce SQL at a first level and the last 2 modules 8 and 9, I am sorry 9 and 10 will discuss about intermediate, that is somewhat advanced level of features in the SQL.

The objective of the current module is to, understand the relational query language and particularly the data definition and the basic query structure, that will hold for all SQL queries, particularly this the modules this week would be important for writing any kind of database applications, squaring the database to find information from the existing data and to manipulate it. So, please put a lot of focus in the whole material of this week and practice them well, to understand the basic issues of database systems, in depth in a well oriented manner. In this module we will first talk about the history and then we will see how to define data and start manipulating them.

(Refer Slide Time: 02:25)



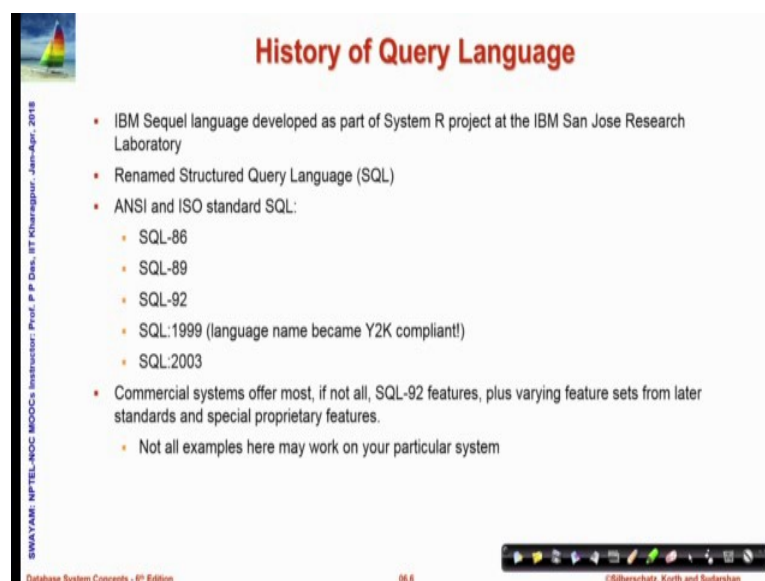
The slide is titled "Module Outline" in red text. It features a small sailboat icon in the top left corner. The main content is a bulleted list: History of SQL, Data Definition Language (DDL), and Basic Query Structure (DML). The slide is part of a presentation titled "Database System Concepts - 6th Edition" by C.J. Date, with a slide number of 96.4. The footer includes the names "Silberschatz, Korth and Sudarshan".

Module Outline

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

Database System Concepts - 6th Edition 96.4 Silberschatz, Korth and Sudarshan

(Refer Slide Time: 02:37)



The slide is titled "History of Query Language" in red text. It features a small sailboat icon in the top left corner. The main content is a bulleted list detailing the history of SQL, from IBM Sequel to various standards (SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003) and commercial implementations. The slide is part of a presentation titled "Database System Concepts - 6th Edition" by C.J. Date, with a slide number of 96.6. The footer includes the names "Silberschatz, Korth and Sudarshan".

History of Query Language

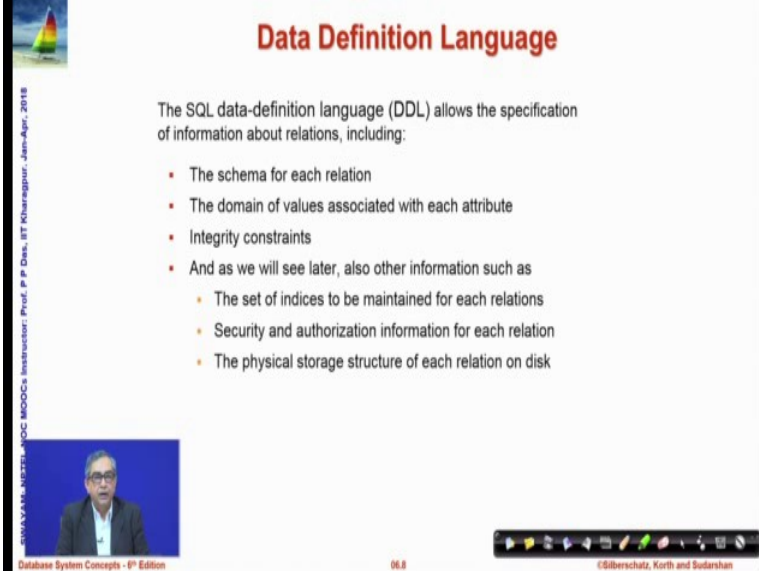
- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (language name became Y2K compliant!)
  - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system

Database System Concepts - 6th Edition 96.6 Silberschatz, Korth and Sudarshan

SQL was originally called IBM sequel language and was a part of system R, it was subsequently renamed as structured query language and like any other good programming language that we have, SQL also gets standardized by ANSI and ISO and there have been several standards of SQL, that has come up with SQL 92 being the most popular one, and the commercial system, most of them try to provide support for SQL 92 features, but they do vary between themselves. So, it is possible that the examples that we show here, may or may not all of them execute in the system that you are using. So, you will have to look at what standard your system is actually following.

So, the first what we will talk about is a DDL data definition language, as we had discussed earlier this is the features to create that schema, the tables in a database management system.

(Refer Slide Time: 03:59)



**Data Definition Language**

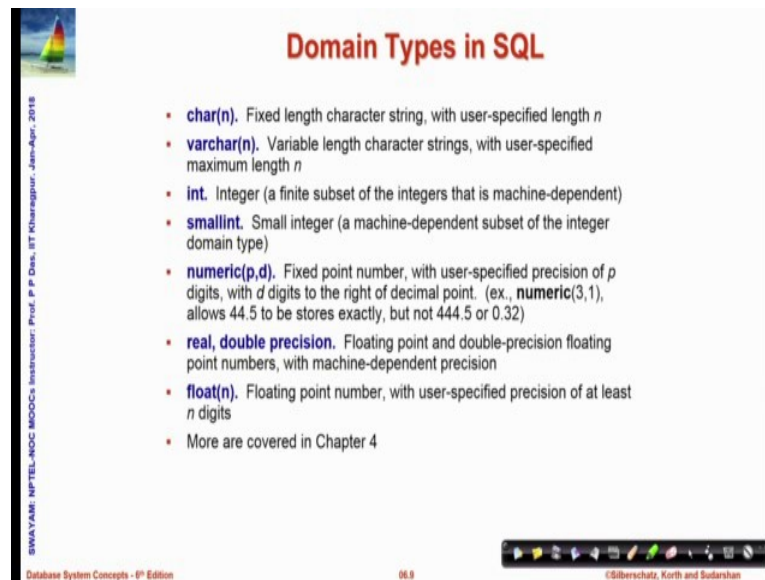
The SQL data-definition language (DDL) allows the specification of information about relations, including:

- The schema for each relation
- The domain of values associated with each attribute
- Integrity constraints
- And as we will see later, also other information such as
  - The set of indices to be maintained for each relations
  - Security and authorization information for each relation
  - The physical storage structure of each relation on disk

Database System Concepts - 6th Edition 96.8 ©Silberschatz, Korth and Sudarshan

So, it allows for the creation or definition of the schema, for each relation that we have in the database it specifies the domain of values associated with each attribute of the schema and it also defines a variety of integrity constraints, later in the course we will see that, it also has to specify other related information like indexing, security authorization, physical storage and so on.

(Refer Slide Time: 04:30)



**Domain Types in SQL**

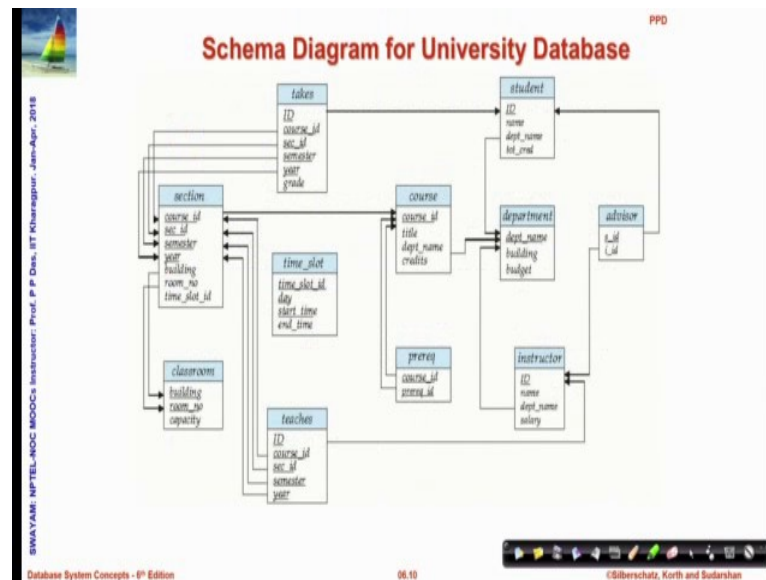
- **char(n).** Fixed length character string, with user-specified length  $n$
- **varchar(n).** Variable length character strings, with user-specified maximum length  $n$
- **int.** Integer (a finite subset of the integers that is machine-dependent)
- **smallint.** Small integer (a machine-dependent subset of the integer domain type)
- **numeric(p,d).** Fixed point number, with user-specified precision of  $p$  digits, with  $d$  digits to the right of decimal point. (ex., **numeric(3,1)**, allows 44.5 to be stored exactly, but not 444.5 or 0.32)
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision
- **float(n).** Floating point number, with user-specified precision of at least  $n$  digits
- More are covered in Chapter 4

Database System Concepts - 6th Edition 96.9 CS/Berschütz, Korth and Sudarshan

So, first the domain of possible values. So, we have already specified that, every domain in SQL is more of an atomic nature. So, they are more like the primitive or built in data types of languages like c, c++, java. So, the common domain types are character which are basically strings of character, having a certain length then you can have variable character string which means that the length here specifies that, the maximum length that the string can take, but a string could be shorter than that integer; obviously, then small integer, which in a system may give a smaller range of integer values.

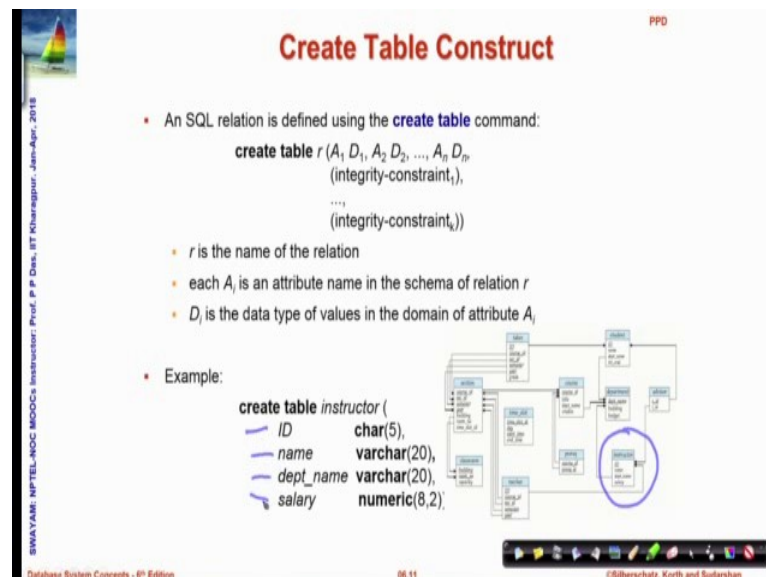
Then there is a numeric type which is often very important, which says what is the precision of the numbers that are to be written in this format stored in this format. So,  $d$  basically gives that precision value and  $p$  gives the size, then you can have real and double precision numbers you can have floating point numbers and so on, and there are some more data types, which we will discuss later in the course.

(Refer Slide Time: 05:52)



Given all these domain types; so, what we will try to do is, here is a schema for the university database, which has multiple different relations designed in that showing the attributes and marking out, what are the keys? And what are the foreign keys? So, we would take examples of some of these and try to code them in the SQL.

(Refer Slide Time: 06:19)



Now, to create a table this is how you go about, the SQL keywords CREATE TABLE is the basic command. So, with the CREATE TABLE you have to specify a name, here the name that is given is in terms of this name *r*, which is the name of the relation and then

you provide a series of attributes, separating by comma  $A_i$  are different attributes and for every attribute, there is a corresponding type domain type specified. So, it says that  $A_1$  is of domain type  $D_1$ ,  $A_2$  is of domain type  $D_2$  and so on. And all of these attribute descriptions, are then followed by a series of integrity constraints. It is possible that a CREATE TABLE may not provide any constraint, but often you will have a number of constraints to work with. So, here is one example.

So, in this we are trying to code the creation of this instructor table, as you can see it has 4 different fields ID, name, department name and salary and for each one we have specified the domain type. So, ID is char 5, this means that the identity of this table instructor will be strings of length 5 whereas, the name or the department name are strings, but they have a maximum length 20, but they could have be of variable length where a salary is of numeric type having specification (8, 2). So, it can have 2 decimal places and be of size 8 maximum. So, this is the basic form of definition that we have for creating a table, defining a table or defining a schema.

(Refer Slide Time: 08:28)

**Integrity Constraints in Create Table**

- not null
- primary key ( $A_1, \dots, A_n$ )
- foreign key ( $A_m, \dots, A_n$ ) references  $r$

Example:

```
create table instructor (  
  ID          char(5),  
  name        varchar(20) not null,  
  dept_name   varchar(20),  
  salary       numeric(8,2),  
  primary key (ID),  
  foreign key (dept_name) references department);
```

The slide includes a diagram of a database schema with tables like instructor, department, and section. Handwritten blue annotations highlight the 'not null' constraint on the 'name' field and the 'primary key' and 'foreign key' constraints in the SQL code. A small video inset shows a person speaking.

Now, we can add a number of integrity constraints to the CREATE TABLE, 3 integrity constraints we will discuss here, one is not null, one is primary key and other third is foreign key. So, not null we specify whether a field can be null or not, primary key as we have seen will specify the attributes which form the primary key, and the foreign key will specify the attributes which reference some other table and our key in that table. So,

here is an example, in the instructor relation here, we had seen this part, the attribute what we have added here is, this not null. So, we say the name is not null which means that, in the instructor table it is not possible to insert a record, where the name of the instructor is null that is unknown, but it is possible. if the same thing is not said about department name same thing is not said about salary. So, it is possible that these could be null.

Now, we additionally say that primary key is ID. So, this field ID is a primary key and it is a property of SQL CREATE TABLE command that, if an attribute is referred as a primary key, then it cannot be not null. So, you do not need to specify that is here you do not need to write not null, because it is a primary key it will be known to be not null, because certainly we have discussed that key is the distinguishing attribute in a database table. So, it cannot be null.

So, it will not be able to distinguish similarly, we have finally we have the third integrity constant which is foreign key which says that, it is referencing this table department and the foreign key of this is here, the dep\_name this particular field is a foreign key, which is a key of the department table. And so, we will be able to refer this, from this table as a foreign key and we know that it will be key in the department table. So, these are the ways to specify the integrity constraint.

(Refer Slide Time: 11:24)

FPD

## And a Few More Relation Definitions

- create table student (  
    ID                 varchar(5),  
    name            varchar(20) not null,  
    dept\_name        varchar(20),  
    tot\_cred          numeric(3,0),  
    primary key (ID),  
    foreign key (dept\_name)  
                  references department);
- create table takes (  
    ID                 varchar(5),  
    course\_id         varchar(8),  
    sec\_id            varchar(8),  
    semester          varchar(6),  
    year              numeric(4,0),  
    grade             varchar(2),  
    primary key (ID, course\_id, sec\_id, semester, year) ,  
    foreign key (ID) references student,  
    foreign key (course\_id, sec\_id, semester, year) references section);
- Note: sec\_id can be dropped from primary key above, to ensure a student cannot be registered for two sections of the same course in the same semester

The ER diagram illustrates the relationships between several tables in a database. The tables are represented by rectangles: **advisor**, **section**, **prereq**, **prereq\_courses**, **prereq\_students**, **prereq\_takes**, and **prereq\_sections**. Lines connect the tables to their respective attributes (ovals). The **advisor** table has attributes ID, name, dept\_name, tot\_cred, and grade. The **section** table has attributes course\_id, sec\_id, semester, year, and grade. The **prereq** table has attributes course\_id, sec\_id, semester, year, and grade. The **prereq\_courses** table has attributes course\_id, sec\_id, semester, year, and grade. The **prereq\_students** table has attributes student\_id, section\_id, semester, year, and grade. The **prereq\_takes** table has attributes student\_id, section\_id, semester, year, and grade. The **prereq\_sections** table has attributes section\_id, sec\_id, semester, year, and grade. Relationships are indicated by lines connecting the tables to their respective attributes.

Srinivasan, NPTEL-UGC MOOCs Instruction Prof. P. Datta, IIT Kharagpur, Jan-Apr., 2018

Database System Concepts - 8th Edition

06.11

©Silberschatz, Korth and Sudarshan



So, here are couple of more examples. So, I will not go through them in detail, I will request you to take time and carefully understand them, again these are about different relations that exist here, about the student and about the courses that the students take, and in every case we have specified the set of fields, that you have in the table in the design of the schema are listed, in the CREATE TABLE the ID information about the primary key is provided and also the information about the foreign key.

Here department name is the foreign key which is mapping to this point, similar things can be observed about the takes relationship which show how students are actually taking courses. So, it relates different it has a set of fields but it has 2 kinds of foreign keys, one that relate to the student through the ID and this combination of attributes which refer to the section.

So, this is how different tables can be created using the data definition language, here is a note that you should observe that if you consider this section ID, the section ID is a part of the primary key which means that 2 records cannot be same, if they are different in the section ID, then such records are allowed. So which means that it is possible that, a student can attend or take a course in the same semester, in the same year with 2 different section IDs because they are primary keys.

So, they can be different. So, if we drop this from the primary key then we will enforce the condition that, no student will be able to take a course, in 2 sections in the same semester and the same year. So, these are the different design choices that we have and we will move on, here is one more example trying to show you the CREATE TABLE command for the course relation, that we have in the university database.

(Refer Slide Time: 13:59)

PPD

## And more still

- **create table** *course* (  
    *course\_id*    *varchar*(8),  
    *title*        *varchar*(50),  
    *dept\_name*   *varchar*(20),  
    *credits*       *numeric*(2,0),  
    **primary key** (*course\_id*),  
    **foreign key** (*dept\_name*) **references** *department*);

Database System Concepts - 6th Edition 06.14 C.S. Gifferschutz, Korth and Sudarshan

(Refer Slide Time: 14:11)

## Updates to tables

- **Insert**
  - **insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000);
- **Delete**
  - Remove all tuples from the *student* relation
  - **delete from** *student*
- **Drop Table**
  - **drop table** *r*
- **Alter**
  - **alter table** *r* **add** *A* *D*
    - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*
    - All exiting tuples in the relation are assigned *null* as the value for the new attribute
  - **alter table** *r* **drop** *A*
    - where *A* is the name of an attribute of relation *r*
    - Dropping of attributes not supported by many databases

Database System Concepts - 6th Edition 06.15 C.S. Gifferschutz, Korth and Sudarshan

Moving on, let us look at how to update or actually put in different records, in a table which has already been created, the basic command is insert, and the keywords for that is INSERT INTO and values, in between you write the name of the relation, where the record will have to be inserted and then the values, will have to be put as a tuple in the same order in which you have defined the attributes of that relation, and certainly each of the values like this is ID value, next is the name value, the department, the salary, each one of them should be from the same domain type, as has been specified during the CREATE TABLE come on.

So, these things will have to remember. And so, every record will get inserted through one insert command, similarly a deletion can be done by DELETE FROM students, if you do DELETE FROM students without specifying which record you want to delete, basically all records will get deleted. We will see how selective deletion will happen, that will come on later. DROP TABLE is a command to remove a table, a table that has been created can be removed from the database altogether, by doing DROP TABLE and the relation name you can also change the schema of a table by using ALTER TABLE.

So, the form is alter table is a key words, you can add a new attribute to relation are by writing the name of the attribute and the domain of the attribute one after the other. Similarly, it is possible also to drop an attribute that already exists and the syntax for that will be, ALTER TABLE the relation name drop is a keyword and the name of the attribute, mind you all database systems may not allow you to drop an attribute to ALTER TABLE to remove attributes. And so, it works in some and it does not work in the rest.

(Refer Slide Time: 16:54)

**Basic Query Structure**

- A typical SQL query has the form:

```

select  $A_1, A_2, \dots, A_n$ 
from  $R_1, R_2, \dots, R_m$ 
where  $P$ 
  
```

- $A_i$  represents an attribute
- $R_i$  represents a relation
- $P$  is a predicate

The slide also features a hand-drawn diagram of a table with three columns labeled  $A_1$ ,  $A_2$ , and  $A_n$ . The diagram shows a grid with three columns and several rows, with the first row containing the attribute labels.

Database System Concepts - 8<sup>th</sup> Edition 06.17 ©Silberschatz, Korth and Sudarshan

Now, that was about the definition of the table and the basic definition of the data. So, now we will get into the basics query structure which is with tables with existing data, how do I query and find out different information.

So, the structure of an SQL query and this you should observe very carefully is normally said to be, SELECT FROM WHERE colloquially will often say, let us have a SELECT

FROM WHERE. So, it has 3 keywords select which is followed by a set of; this is a set of attributes. So, this specifies that, when a select query runs it will finally give us a new relation and in that relation, the attributes that will be available are the attributes that feature in the select list. The next clause or the next keyword in this is from, which specifies a set of existing relations.

So,  $r_1, r_2, r_m$  represent different relations and these are the relations, which will be used to actually find the information extract the information. Finally, the where clause has a predicate as a condition, which specify that what condition has to be satisfied. So, that certain peoples from the relations  $r_1$  to  $r_m$ , will be chosen and put in this new selected result, table in terms of the attributes  $A_1$  to  $A_n$ .

(Refer Slide Time: 18:37)

**The select Clause**

- The **select** clause lists the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:  

```
select name  
from instructor
```
- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g.,  $Name \equiv NAME \equiv name$
  - Some people use upper case wherever we use bold font

Database System Concepts - 6th Edition 06.18 ©Silberschatz, Korth and Sudarshan

So, this is the basic understanding of the structure of the SQL query and naturally as I mentioned that will result in a relation. Now, we will go over each and every clause carefully, the select clause as I said will list all the attributes. So, it is like a projection (II) in terms of the relational algebra that we have done. So, if we write select name from instructor then this will result in finding the names of all instructors from the instructor table, because this is, this you know is a relation, because it is happening it is a featuring in the from clause and in select, we are saying that the attribute that we want to select is the attribute name.

So, it will the instructor table has 4 attributes ID, name, dept. name and salary from that it will simply take the name of the instructor and list that in the output table. So, the basic form of selection that happens and this point you may also note, that in SQL everything is case insensitive, it does not matter whether you write in upper case or lower case. So, you can choose the style that you prefer to use.

(Refer Slide Time: 19:51)

**The select Clause (Cont.)**

- SQL allows duplicates in relations as well as in query results
- To force the elimination of duplicates, insert the keyword **distinct** after select
- Find the department names of all instructors, and remove duplicates  

```
select distinct dept_name  
from instructor
```
- The keyword **all** specifies that duplicates should not be removed  

```
select all dept_name  
from instructor
```

Database System Concepts - 8th Edition 06.19 ©Silberschatz, Korth and Sudarshan

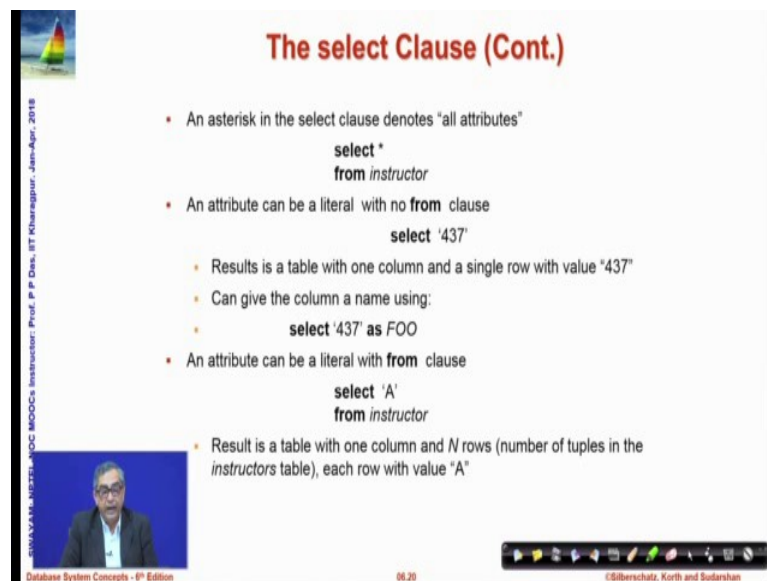
This is a very important factor, that you should keep in mind that we said, while introducing relational algebra, that in the relational algebra every relation is a set and which means that according to set theory. We cannot have 2 tuples in the same relation, which are identical in all its values, because set theory does not allow that, but please keep in mind that SQL actually allows duplicates in relations. So, it is possible that in the same relation in the same table, I may have more than one record which are identical in all the fields, in all the attributes of that table and this will have a lot of consequences and we will see how often, this property will have to be used.

So, if you want a typical set theoretic kind of output, that is if you want the result to be distinct, all records to be distinct, then you have to explicitly say that you want distinct values to be selected. So, all that you are doing is, you are after select and before the attribute name, you introduce another keyword distinct.

So, select distinct dept name from instructor will actually, select the departments of all instructors and quite why if it just selects department name of all instructor, then it is

quite possible that the same department name will appear number of times, because every department has multiple instructors. But when we use distinct, then every name will feature only once in that selection, then you can also specify another keyword all, which ensures that the duplicates are not removed. So, if you do select all depth name, then all the names will feature with duplicates, if some department had 3 instructors the name of that department will feature thrice.

(Refer Slide Time: 22:05)



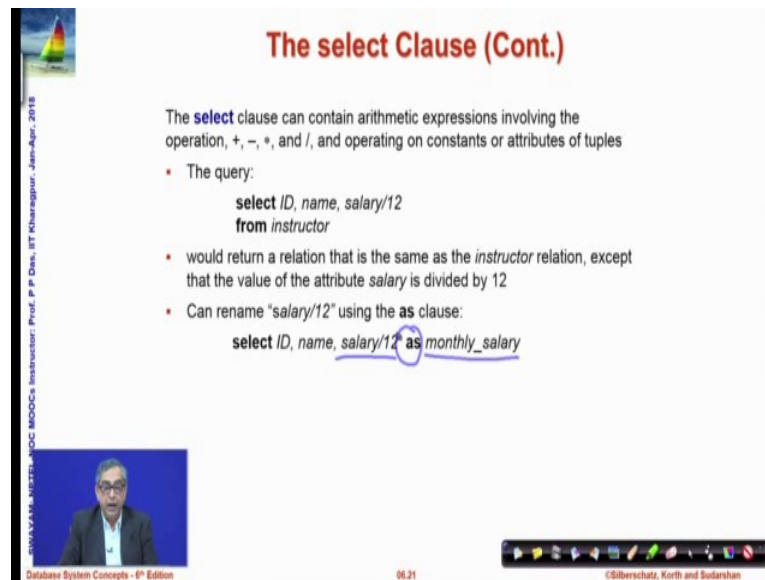
**The select Clause (Cont.)**

- An asterisk in the select clause denotes "all attributes"  
`select *`  
`from instructor`
- An attribute can be a literal with no **from** clause  
`select '437'`
  - Results is a table with one column and a single row with value "437"
  - Can give the column a name using:  
`select '437' as FOO`
- An attribute can be a literal with **from** clause  
`select 'A'`  
`from instructor`
  - Result is a table with one column and *N* rows (number of tuples in the *instructors* table), each row with value "A"

Database System Concepts - 6th Edition 06.20 ©Silberschatz, Korth and Sudarshan

You can use an asterisk after select, to specify that you are interested in all the attributes that the relation or the collection of relations in the from clause has. You can also specify a select with a literal and without a from clause, if you do that then it will simply return you a table with a single row having that literal value and you can also rename that table, using what is known as clause as a command. So, this will give you a table Foo, where there is only one row and that row has an entry 437. You can use that, for other purposes also you can do a select of a literal, from a table with using a from clause where in, you will get a single column table, where as many is as there are records in the instructor will be produced.

(Refer Slide Time: 23:16)



### The select Clause (Cont.)

The **select** clause can contain arithmetic expressions involving the operation, +, -, \*, and /, and operating on constants or attributes of tuples

- The query:  

```
select ID, name, salary/12  
from instructor
```
- would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12
- Can rename "salary/12" using the **as** clause:  

```
select ID, name, salary/12 as monthly_salary
```

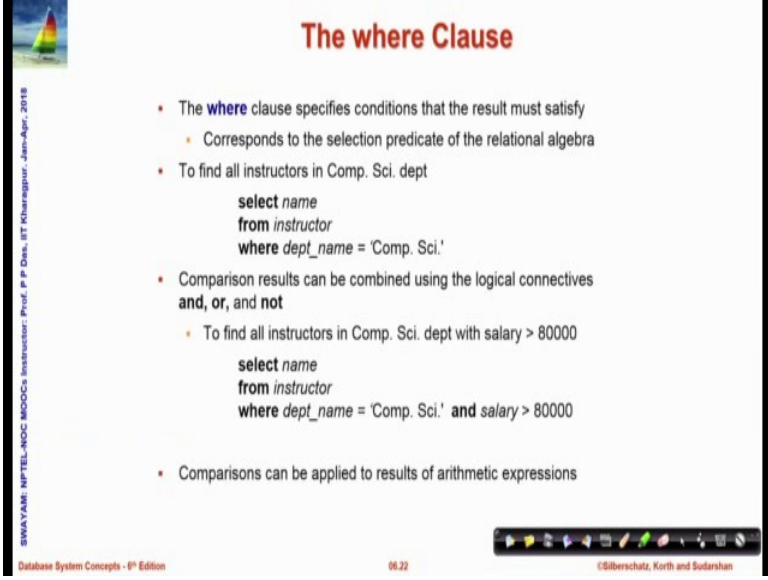
Database System Concepts - 6th Edition

08.21

CS509: Schatz, Korth and Sudarshan

Select clause can also use basic arithmetic operations for example, here we are showing a select where the third attribute as you can see, the third attribute is salary by 12, assuming that the instructor table has a salary number which is annual salary by 12 naturally you give you the monthly salary. So, those such arithmetic choices can also be made. you can also rename that field that particular salary by 12 field by a new name, as I said as can be used to rename. So, if we if you use that then, when you get the output you will get the column names are ID, name and monthly salary and in monthly salary you will actually have a computation, which is salary by 12 and in the same way, you can use multiple different kinds of arithmetic operators.

(Refer Slide Time: 24:16)



**The where Clause**

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra
- To find all instructors in Comp. Sci. dept

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```
- Comparison results can be combined using the logical connectives **and**, **or**, and **not**
  - To find all instructors in Comp. Sci. dept with salary > 80000

```
select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 80000
```
- Comparisons can be applied to results of arithmetic expressions

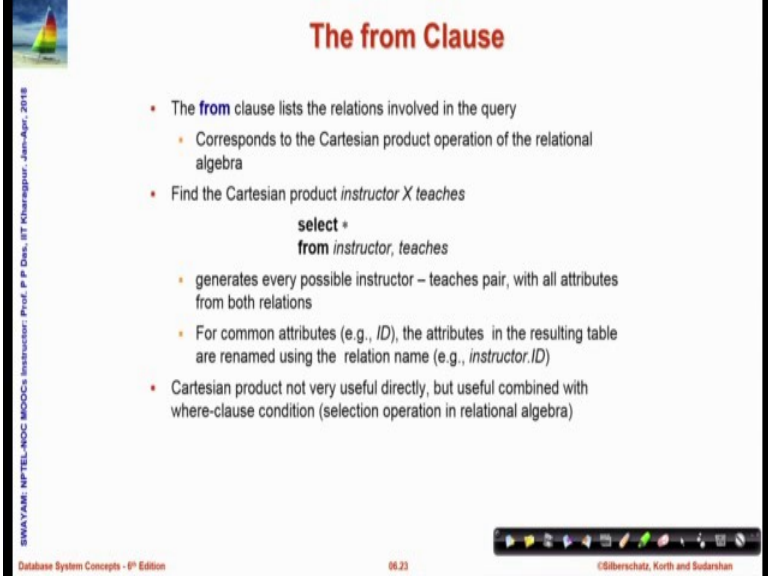
Database System Concepts - 6th Edition 09.22 ©Silberschatz, Korth and Sudarshan

Now, we come to the where clause, where clause specifies the condition is a predicate which corresponds to the selection predicate of relational algebra. So, it will specify some condition, here is an example if we want to find all instructors from the instructor table, who are associated with computer science department, then you can say select name from instructor and to specify that they are from the computer science department, you will specify department name is equal to computer science.

So, this will ensure that you select the records only when this condition is satisfied. So, all records for which department name is different from computer science will not be included here. You can also write predicates using the different logical connectives and or not and so on. So, here is an example, where you are finding all instructors in computer science with salary greater than 80000. So, here we have used and clause. So, only records where the department name is computer science and salary is greater than 80000 will be chosen in the result. So, and then the projection (**II**) will be done on the name of those instructors, you can apply comparisons of arithmetic expression. So, where clause can really write different kind of things.



(Refer Slide Time: 26:04)



**The from Clause**

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra
- 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)

Database System Concepts - 6th Edition 08.23 ©Silberschatz, Korth and Sudarshan

Finally, the from clause is sets all the different relations from where you are actually looking for the records. So, it kind of corresponds to the Cartesian product of the relational algebra. So, if we want to say compute instructor Cartesian product teachers, then you can say `SELECT * instructor one table comma teachers`. So, this will choose records from instructed relation, as well as from teacher's relation and in all possible combined way, it will put them in the output we have used a star.

So, all fields of in instructor and all fields of teaches will be there in the output, and since some fields may have identical name like ID in instructor and there is ID in teachers, they will be qualified by the name of the relation, from can have 1 relation, 2 relation any number of relations as you require. So, this will cause the Cartesian product to be computed which may not be very useful as an independent feature, but we will see in the next module how it can give very important computations, in terms of computing joints and so on.

(Refer Slide Time: 27:37)

### Cartesian Product

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

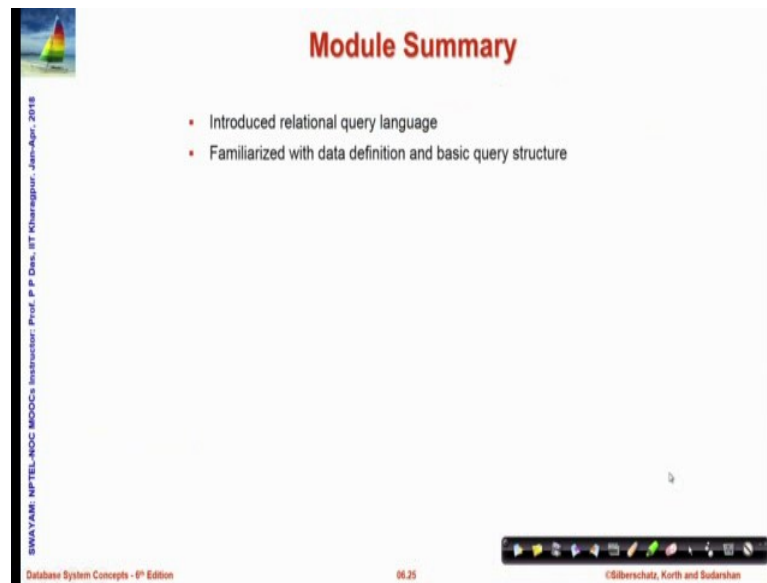
ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

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	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

So, here is an example of the Cartesian product that we talked of. So, here is a instructor relation, the teacher's relation and as you can see when we have done this cartesian product, that is `SELECT * from instructor comma teachers`, then all fields this is an ID of the instructor, there is an ID in teachers.

So, that is also specified here qualified by the name of the relation whereas, name comes in directly because there is no attribute called name in teachers, the department name comes in directly, salary comes in, directly course ID comes in, section ID comes in, semester comes in, year comes in and so on. And the combination of all tuples in their instructor relation, against all tuples of the teacher's relation all possible combinations have come in this result, which eventually is a cartesian product of the relational algebra.

(Refer Slide Time: 28:50)



The slide is titled "Module Summary" in red text. It contains two bullet points: "Introduced relational query language" and "Familiarized with data definition and basic query structure". The slide is part of a presentation, as evidenced by the navigation bar at the bottom and the vertical text on the left side.

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

Database System Concepts - 6th Edition

08.25

©Silberschatz, Korth and Sudarshan

So, this is what we have to summarize, we have introduced the relational query language and particularly familiarize ourselves with the data definition that is creation of the table creation, of the schema with the attribute names, domain types and constraints. And the updates to the table in terms of insertion and deletion of values or addition or deletion of attributes or removing a table altogether and then, we have given the basic structure of the SELECT FROM WHERE query of SQL, which will be the key language feature of a query language, that we will continue to discuss all through this course.