Database Management Systems

Module 06: Introduction to Relational Model/1

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Week Recap

- The proliferation of DBMS in wide range of applications provide motivation to study the subject
- Know Your Course provided information about prerequisites, outline and text book
- The specific need for a DBMS discussed in contrast to a file system based application using a programming language like Python
- Basic notions of a DBMS are introduced

Module Objectives

- To understand attributes and their types
- To understand the mathematical structure of relational model
  - Schema
  - Instance
  - Keys
- To familiarize with different types of relational query languages

Module Outline

- Attribute Types
- Relation Schema and Instance
- Keys
- Relational Query Languages

Example of a Relation

attributes (or columns)

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

tuples (or rows)

Attributes

Attribute Types

- Consider
  $Students = Roll\#, First\ Name, Last\ Name, DoB, Passport\#, Aadhaar\#, Department$
  relation
- The set of allowed values for each attribute is called the **domain** of the attribute
  - **Roll #:** Alphanumeric string
  - **First Name, Last Name:** Alpha String
  - **DoB:** Date
  - **Passport #:** String (Letter followed by 7 digits) – nullable (optional)
  - **Aadhaar #:** 12-digit number
  - **Department:** Alpha String
- Attribute values are (normally) required to be **atomic**; that is, indivisible
- The special value **null** is a member of every domain. Indicates that the value is *unknown*
- The null value may cause complications in the definition of many operations

Attribute Types

- For
  $Students = Roll\#, First\ Name, Last\ Name, DoB, Passport\#, Aadhaar\#, Department$
- And **domain** of the attributes as:
  - **Roll #:** Alphanumeric string
  - **First Name, Last Name:** Alpha String
  - **DoB:** Date
  - **Passport #:** String (Letter followed by 7 digits) – nullable (optional)
  - **Aadhaar #:** 12-digit number
  - **Department:** Alpha String

Roll #	First Name	Last Name	DoB	Passport #	Aadhaar #	Department
15CS10026	Lalit	Dubey	27-Mar-1997	L4032464	1728-6174-9239	Computer
16EE30029	Jatin	Chopra	17-Nov-1996	null	3917-1836-3816	Electrical

Schema and Instance

Relation Schema and Instance

- $A_1, A_2, \cdots, A_n$ are *attributes*
- $R = (A_1, A_2, \cdots, A_n)$ is a *relation schema*
  Example: $instructor = (ID, name, dept\_name, salary)$
- Formally, given sets $D_1, D_2, \cdots, D_n$ a **relation** r is a subset of

$$D_1 \times D_2 \times \cdots \times D_n$$

  Thus, a relation is a set of n-tuples $(a_1, a_2, \cdots, a_n)$ where each $a_i \in D_i$
- The current values (**relation instance**) of a relation are specified by a table
- An element t of r is a tuple, represented by a row in a table
- Example:
  $instructor \equiv (String(5) \times String \times String \times Number+)$, where $ID \in String(5)$, $name \in String$, $dept\_name \in String$, and $salary \in Number+$

Relations are Unordered with Unique Tuples

- **Order of tuples / rows is irrelevant** (tuples may be stored in an arbitrary order)
- **No two tuples / rows may be identical**
- Example: *instructor* relation with unordered tuples

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

Keys

Keys

- Let $K \subseteq R$, where R is the set of attributes in the relation
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
  - Example: $\{ID\}$ and $\{ID, name\}$ are both superkeys of instructor
- Superkey K is a **candidate key** if K is minimal
  - Example: $\{ID\}$ is a candidate key for *instructor*
- One of the candidate keys is selected to be the **primary key**
  - Which one?
- A **surrogate key** (or synthetic key) in a database is a unique identifier for either an *entity* in the modeled world or an *object* in the database
  - The surrogate key is *not* derived from application data, unlike a *natural* (or *business*) key which is derived from application data

Keys

- *Students* = *Roll#*, *First Name*, *Last Name*, *DoB*, *Passport#*, *Aadhaar#*, *Department*
- **Super Key:** Roll #, {Roll #, DoB}
- **Candidate Keys:** Roll #, {First Name, Last Name}, Aadhaar#
  - Passport # cannot be a key. Why?
  - Null values are allowed for Passport # (a student may not have a passport)
- **Primary Key:** Roll #
  - Can Aadhaar# be a key?
  - It may suffice for unique identification. But Roll# may have additional useful information. For example: 14CS92P01
    - ▷ Read 14CS92P01 as 14-CS-92-P-01
    - ▷ 14: Admission in 2014
    - ▷ CS: Department = CS
    - ▷ 92: Category of Student
    - ▷ P: Type of admission: *Project*
    - ▷ 01: Serial Number

- **Secondary / Alternate Key:** {First Name, Last Name}, Aadhaar #
- **Simple Key:** Consists of a *single attribute*
- **Composite Key:** {First Name, Last Name}
  - Consists of more than one attribute to uniquely identify an entity occurrence
  - One or more of the attributes, which make up the key, are not simple keys in their own right

Roll #	First Name	Last Name	DoB	Passport #	Aadhaar #	Department
15CS10026	Lalit	Dubey	27-Mar-1997	L4032464	1728-6174-9239	Computer
16EE30029	Jatin	Chopra	17-Nov-1996	null	3917-1836-3816	Electrical
15EC10016	Smriti	Mongra	23-Dec-1996	G5432849	2045-9271-0914	Electronics
16CE10038	Dipti	Dutta	02-Feb-1997	null	5719-1948-2918	Civil
15CS30021	Ramdin	Minz	10-Jan-1997	X8811623	4928-4927-5924	Computer

Keys

- **Foreign key constraint:** Value in one relation must appear in another
  - **Referencing** relation
    - ▷ Enrolment: Foreign Keys – Roll #, Course #
  - **Referenced** relation
    - ▷ Students, Courses
- A **compound key** consists of *more than one attribute* to uniquely identify an entity occurrence
  - Each attribute, which makes up the key, is a simple key in its own right
  - {Roll #, Course #}

Students

Roll #	First Name	Last Name	DoB	Passport #	Aadhaar #	Department

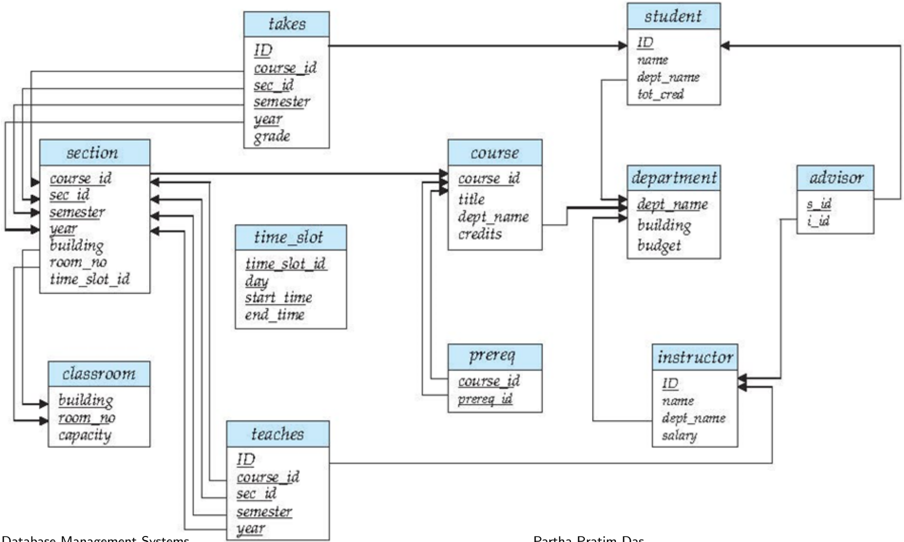
Courses

Course #	Course Name	Credits	L-T-P	Department

Enrolment

Roll #	Course #	Instructor ID

Schema Diagram for University Database



Relational Query Languages

Relational Query Languages

Procedural viz-a-viz Non-procedural or Declarative Paradigms

- **Procedural programming** requires that the programmer tell the computer what to do
  - That is, *how* to get the output for the range of required inputs
  - The programmer must know an appropriate algorithm
- **Declarative programming** requires a more descriptive style
  - The programmer must know *what* relationships hold between various entities

Relational Query Languages (Cont..)

Procedural vs. Non-procedural or Declarative Paradigms

- **Example: Square root of n**
  - Procedural
    - a) Guess x_0 (close to root of n)
    - b) $i \leftarrow 0$
    - c) $x_{i+1} \leftarrow (x_i + n/x_i)/2$
    - d) Repeat Step 2 if $|x_{i+1} - x_i| > delta$
  - Declarative
    - ▷ Root of n is m such that $m^2 = n$

Relational Query Languages

- "Pure" languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- The above 3 pure languages are equivalent in computing power
- We will concentrate on relational algebra
  - Not Turing-machine equivalent
    - Not all algorithms can be expressed in RA
  - Consists of 6 basic operations

Module Summary

- Introduced the notion of attributes and their types
- Taken an overview of the mathematical structure of relational model – schema and instance
- Introduced the notion of keys – primary as well as foreign

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".

Database Management Systems

Module 07: Introduction to Relational Model/2

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Basic notions of modeling introduced
  - Attributes and their Types
  - Schema and Instance
  - Keys and their Categorization
- Languages for Relation Model introduced

Module Objectives

- To understand relational algebra
- To familiarize with the operators of relational algebra

Module Outline

- Operations
  - Select
  - Project
  - Union
  - Difference
  - Intersection
  - Cartesian Product
  - Natural Join
- Aggregate Operations

Relational Operators

Basic Properties of Relations

- **A relation is set**. Hence,
- **Ordering of rows / tuples is inconsequential**

A	B
a1	b1
a1	b2
a2	b1
a2	b2

is same as:

A	B
a1	b1
a2	b1
a2	b2
a1	b2

- **All rows / tuples must be distinct**

A	B
a1	b1
a1	b2
a1	b2
a1	b1

is not valid

A	B
a1	b1
a1	b2

is

Select Operation – selection of rows (tuples)

- Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A=B \wedge D>5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Project Operation – selection of columns (Attributes)

- Relation r

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Union of two relations

- Relation r, s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cup s$

A	B
α	1
α	2
β	1
β	3

Set difference of two relations

- Relation r, s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r - s$

A	B
α	1
β	1

Set intersection of two relations

- Relation r, s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Note: $r \cap s = r - (r - s)$

Joining two relations – Cartesian-product

- Relation r, s

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- $r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Cartesian-product – naming issue

- Relation r, s

A	B
α	1
β	2

r

B	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- $r \times s$

A	$r.B$	$s.B$	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Renaming a Table

- Allows us to refer to a relation, (say E) by more than one name.

$$\rho_X(E)$$

returns the expression E under the name X

- Relations r

A	B
α	1
β	2

r

- $r \times \rho_s(r)$

$r.A$	$r.B$	$s.A$	$s.B$
α	1	α	1
α	1	β	2
β	2	α	1
β	2	β	2

Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$
- $r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- $\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

Joining two relations – Natural Join

- Let r and s be relations on schemas R and S respectively. Then, the "natural join" of relations R and S is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from r and t_s from s.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple t to the result, where
    - t has the same value as t_r on r
    - t has the same value as t_s on s

Natural Join Example

- Relations r, s :

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

- Natural Join
  - $r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

$$\pi_{A,r.B,C,r.D,E}(\sigma_{r.B=s.B \wedge r.D=s.D}(r \times s))$$

Aggregation Operators

Aggregate Operators

- Can we compute:
  - SUM
  - AVG
  - MAX
  - MIN

Notes about Relational Languages

- Each query input is a table (or set of tables)
- Each query output is a table
- All data in the output table appears in one of the input tables
- Relational Algebra is not Turing complete

Summary of Relational Algebra Operators

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary} >= 85000}(\textit{instructor})$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID,\ salary}(\textit{instructor})$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
x (Cartesian Product)	*instructor* x *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
∪ (Union)	$\Pi_{name}(\textit{instructor}) \cup \Pi_{name}(\textit{student})$
	Output the union of tuples from the *two* input relations.
- (Set Difference)	$\Pi_{name}(\textit{instructor}) - \Pi_{name}(\textit{student})$
	Output the set difference of tuples from the two input relations.
⋈ (Natural Join)	*instructor* ⋈ *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.

Module Summary

- Introduced relational algebra
- Familiarized with the operators of relational algebra

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".

Database Management Systems

Module 08: Introduction to SQL/1

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Introduced relational algebra
- Familiarized with the operators of relational algebra

Module Objectives

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

Module Outline

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

History of SQL

History of Query Language

- IBM developed *Structured English Query Language* (**SEQUEL**) as part of System R project. Renamed Structured Query Language (SQL: *pronounced still as* **SEQUEL**)
- ANSI and ISO standard SQL:

SQL-86	First formalized by **ANSI**
SQL-89	+ **Integrity Constraints**
SQL-92	Major revision (**ISO/IEC 9075 standard**), **De-facto Industry Standard**
SQL:1999	+ **Regular Expression Matching**, **Recursive Queries**, **Triggers**, **Support for Procedural and Control Flow Statements**, Nonscalar types (Arrays), and Some OO features (structured types), **Embedding SQL in Java (SQL/OLB)**, and **Embedding Java in SQL (SQL/JRT)**
SQL:2003	+ **XML features (SQL/XML)**, Window Functions, Standardized Sequences, and Columns with Auto-generated Values (identity columns)
SQL:2006	+ Ways of **importing and storing XML data** in an SQL database, **manipulating** it within the database, and **publishing both XML and conventional SQL-data in XML form**
SQL:2008	Legalizes ORDER BY outside Cursor Definitions + INSTEAD OF Triggers, TRUNCATE Statement, and FETCH Clause
SQL:2011	+ Temporal Data (PERIOD FOR) Enhancements for Window Functions and FETCH Clause
SQL:2016	+ Row Pattern Matching, Polymorphic Table Functions, and **JSON**
SQL:2019	+ Multidimensional Arrays (MDarray type and operators)

History of Query Language (2): Compliance

- SQL is the de facto industry standard today for relational or structred data systems
- Commercial systems as well as open systems may be fully or partially compliant to one or more standards from SQL-92 onward
- Not all examples here may work on your particular system. Check your system's SQL documentation

History of Query Language (3): Alternatives

- There aren't any alternatives to SQL for speaking to relational databases (that is, SQL as a protocol), but there are many alternatives to writing SQL in the applications
- These alternatives have been implemented in the form of frontends for working with relational databases. Some examples of a frontend include (for a section of languages):
  - SchemeQL and CLSQL, which are probably the most flexible, owing to their Lisp heritage, but they also look like a lot more like SQL than other frontends
  - LINQ (in .Net)
  - ScalaQL and ScalaQuery (in Scala)
  - SqlStatement, ActiveRecord and many others in Ruby
  - HaskellDB
  - ...the list goes on for many other languages.

Source: *What are good alternatives to SQL (the language)?*

History of Query Language (4): Derivatives

- There are several query languages that are derived from or inspired by SQL. Of these, the most popular and effective is SPARQL.
  - **SPARQL** (pronounced *sparkle*, a recursive acronym for *SPARQL Protocol and RDF Query Language*) is an RDF query language
    - ▷ A semantic query language for databases - able to retrieve and manipulate data stored in Resource Description Framework (RDF) format.
    - ▷ It has been standardized by the W3C Consortium as key technology of the semantic web
    - ▷ Versions:
      - SPARQL 1.0 (January 2008)
      - SPARQL 1.1 (March, 2013)
    - ▷ Used as the query languages for several NoSQL systems - particularly the Graph Databases that use RDF as store

Data Definition Language (DDL)

Data Definition Language (DDL)

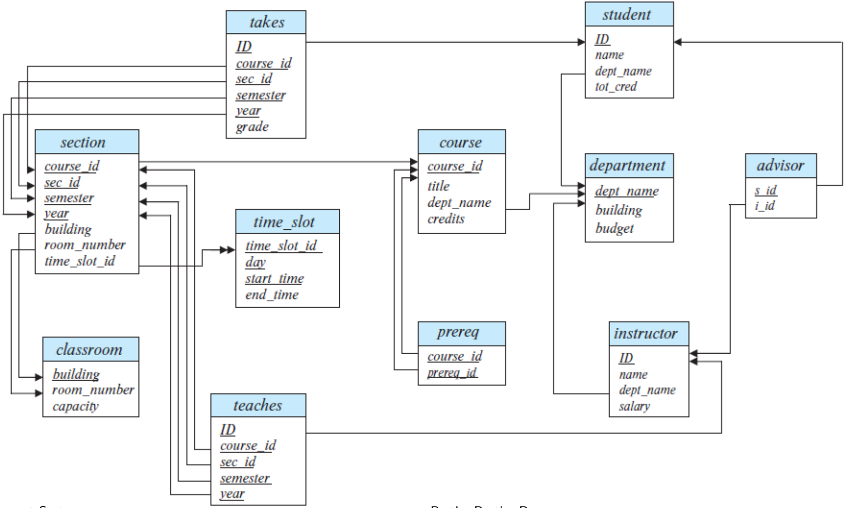
The SQL data-definition language (DDL) allows the specification of information about relations, including:

- 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

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(n)**. 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 stores 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

Schema Diagram for University Database



Create Table Construct

- An SQL relation is defined using the **create table** command:

$$\textbf{create table } r \; (A_1 D_1, A_2 D_2, \ldots, A_n D_n),\\ (\textit{integrity-constraint}_1),\\ \ldots\\ (\textit{integrity-constraint}_k));$$

  - r is the name of the relation
  - each A_i is an attribute name in the schema of relation r
  - D_i is the data type of values in the domain of attribute A_i

Create Table Construct (2)

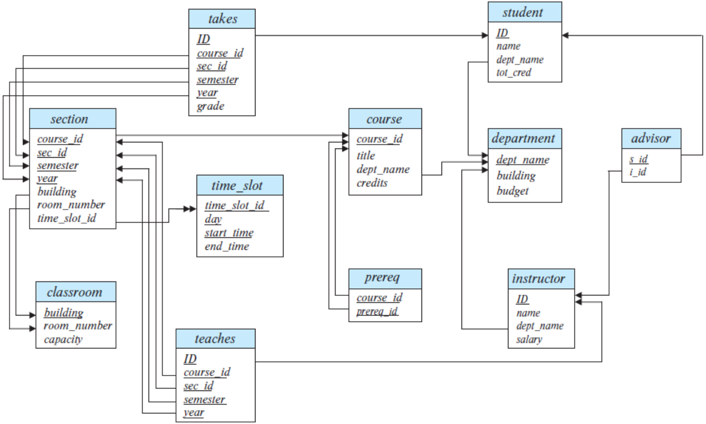
create table *instructor* (

ID **char**(5),

name **varchar**(20)

dept_name **varchar**(20)

salary **numeric**(8, 2));



Create Table Construct (3): Integrity Constraints

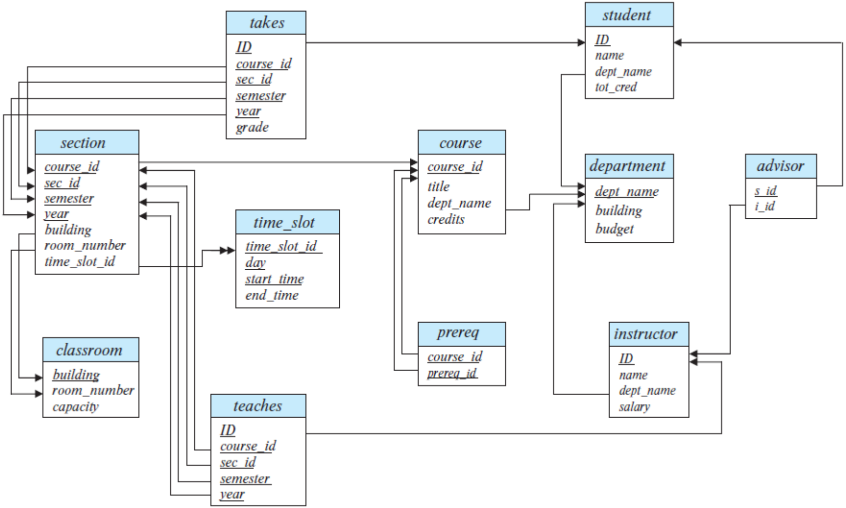
- **not null**
- **primary key** $(A_1, \ldots, A_n)$
- **foreign key** $(A_m, \ldots, A_n)$ **references** r

create table *instructor* (
  ID **char**(5),
  name **varchar**(20)
  dept_name **varchar**(20)
  salary **numeric**(8, 2));

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*));

primary key declaration on an attribute automatically ensures **not null**

University Schema



Create Table Construct (4): More Relations

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 *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*);

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

Update Tables

- **Insert (DML command)**
  - **insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000);
- **Delete (DML command)**
  - Remove all tuples from the *student* relation

    delete from *student*
- **Drop Table (DDL command)**
  - **drop table** *r*
- **Alter (DDL command)**
  - **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 existing 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

Data Manipulation Language (DML): Query Structure

Basic Query Structure

- A typical SQL query has the form:

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

  - A_i represents an attribute from r_i's
  - r_i represents a relation
  - P is a predicate
- The result of an SQL query is a relation

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 (that is, you may use upper-case or lower-case letters)
  - $Name \equiv NAME \equiv name$
  - Some people use upper case wherever we use bold font

Select Clause (2)

- **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*

Select Clause (3)

- 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'

Select Clause (4)

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*

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

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 (for example, *ID*), the attributes in the resulting table are renamed using the relation name (for example, *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

IIT Madras BSc Degree

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821			
98345			

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
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
				2010
				2009
				2009
				2010
				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	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Module Summary

- Introduced relational query language
- Familiarized with data definition and basic query structure

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".

Database Management Systems

Module 09: Introduction to SQL/2

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Introduced relational query language
- Familiarized with data definition and basic query structure

Module Objectives

- To complete the understanding of basic query structure

Module Outline

- Additional Basic Operations
  - Cartesian Product
  - Rename AS Operation
  - String Values
  - Order By
  - Select Top / Fetch
  - Where Clause Predicate
  - Duplicates

Additional Basic Operations

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 (for example, *ID*), the attributes in the resulting table are renamed using the relation name (for example, *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821			
98345			

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
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
				2010
				2009
				2009
				2010
				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	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

- 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

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821			
98345			

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
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
				2010
				2009
				2009
				2010
				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~~
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

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'

Rename AS 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* $\equiv$ *instructor T*

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"

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

String Operations (2)

- 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
- SQL supports a variety of string operations such as
  - concatenation (using "||")
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

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*

Selecting Number of Tuples in Output

- The **Select Top** clause is used to specify the number of records to return
- The **Select Top** clause is useful on large tables with thousands of records. Returning a large number of records can impact performance

  select top 10 **distinct** *name*
  from *instructor*

- Not all database systems support the SELECT TOP clause.
  - SQL Server & MS Access support **select top**
  - MySQL supports the **limit** clause
  - Oracle uses **fetch first** n **rows only** and **rownum**

  select distinct *name*
  from *instructor*
  order by *name*
  fetch first 10 **rows only**

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, $\geq \$90,000$ and $\leq \$100,000$)

  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')*;

In Operator

- The **in** operator allows you to specify multiple values in a **where** clause
- The **in** operator is a shorthand for multiple **or** conditions

  select *name*
  from *instructor*
  where *dept_name* **in** ('Comp. Sci.', 'Biology')

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:
  a) $\sigma_\theta(r_1)$: 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 $\sigma_\theta(r_1)$
  b) $\Pi_A(r)$: 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
  c) $r_1 \times r_2$: 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 $r_1 \times r_2$

Duplicates (2)

- Example: Suppose multiset relations $r_1(A, B)$ and $r_2(C)$ are as follows:
  $r_1 = \{(1, a)(2, a)\}$ $\quad r2 = \{(2), (3), (3)\}$
- Then $\Pi_B(r_1)$ would be $\{(a), (a)\}$, while $\Pi_B(r_1)$ x 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:

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

Module Summary

- Completed the understanding of basic query structure

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".

Database Management Systems

Module 10: Introduction to SQL/3

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Completed the understanding of basic query structure

Module Objectives

- To familiarize with set operations, null values and aggregation

Module Outline

- Set Operations: union, intersect, except
- Null Values
- Aggregate Functions: avg, min, max, sum, and count
    - Group By
    - Having
    - Null Values

Set Operations

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)

Set Operations (2)

- 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")

Set Operations (3)

- 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

Null Values

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**
- It is not possible to test for **null** values with comparison operators, such as =, <, or <>
  We need to use the **is null** and **is not null** operators instead

Null Values (2): 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*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

Aggregate Functions

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

Aggregate Functions (2)

- 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 *courses*;

Aggregate Functions (3): 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

Aggregate Functions (4): Group By

- 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*;

Aggregate Functions (5): Having Clause

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

  select *dept_name*, *ID*, **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

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

Module Summary

- Completed the understanding of set operations, null values, and aggregation

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with “PPD”.