CMSC424: Database Design
Relational Model/SQL

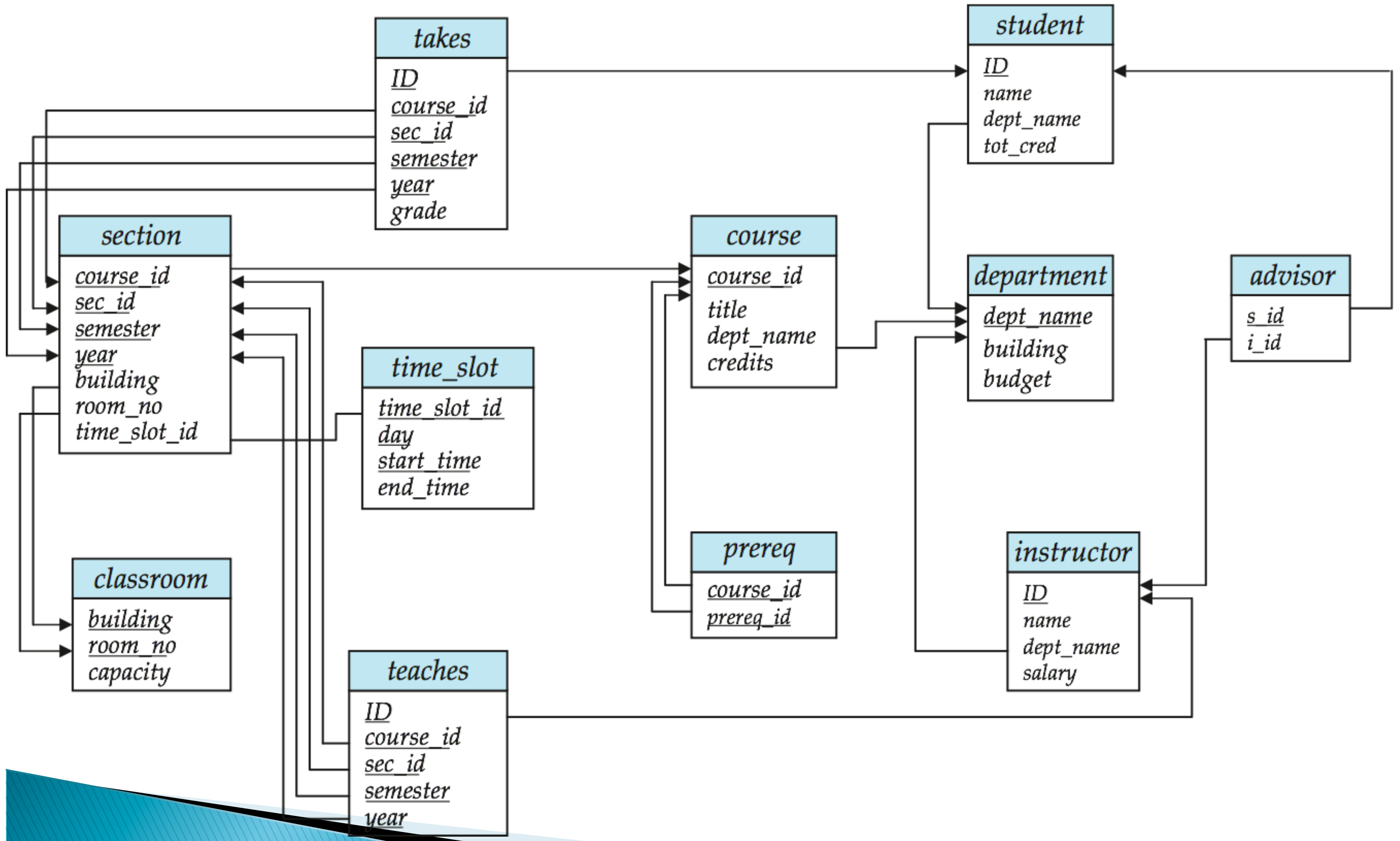
Instructor: Amol Deshpande
amol@cs.umd.edu

Topics covered so far

- Why Databases
  - Data Modeling
  - Importance of abstraction/independence layers
- Relational Model
  - Relations, Tuples
  - Primary Keys, Foreign Keys
  - Referential Integrity Constraints
- Relational Algebra Operations
- SQL
  - Data Definition Language: How to create relations, change schemas, etc.
  - Data Manipulation Language: Simple single-table queries

Outline

Schema Diagram for University Database



Basic Query Structure

select $A_1, A_2, \ldots, A_n$ — Attributes or expressions

from $r_1, r_2, \ldots, r_m$ — Relations (or queries returning tables)

where P — Predicates

Find the names of all instructors:
select *name*
from *instructor*

Remove duplicates:
select distinct *name*
from *instructor*

Order the output:
select distinct *name*
from *instructor*
order by *name* **asc**

Apply some filters (predicates):
select *name*
from *instructor*
where salary > 80000 **and** dept_name = 'Finance';

Basic Query Constructs

Find the names of all instructors:

```
select name
from instructor
```

Select all attributes:

```
select *
from instructor
```

Expressions in the select clause:

```
select name, salary < 100000
from instructor
```

More complex filters:

```
select name
from instructor
where (dept_name != ‘Finance’ and salary > 75000)
or (dept_name = ‘Finance’ and salary > 85000);
```

A filter with a subquery:

```
select name
from instructor
where dept_name in (select dept_name from
                    department where budget < 100000);
```

Basic Query Constructs

Find the names of all instructors:
select *name*
from *instructor*

Renaming tables or output column names:
select *i.name, i.salary * 2* **as** *double_salary*
from *instructor i*
where *i.salary < 80000* **and** *i.name like '%g_';*

More complex expressions:
select *concat(name, concat(', ', dept_name))*
from *instructor;*

Careful with NULLs:
select *name*
from *instructor*
where *salary < 100000* **or** *salary >= 100000;*

Wouldn't return the instructor with NULL salary (if any)

Multi-table Queries

Cartesian product:
select *
from *instructor, department*

Use predicates to only select "matching" pairs:
select *
from *instructor i, department d*
where *i.dept_name = d.dept_name;*

Identical (in this case) to using a natural join:
select *
from *instructor* **natural join** *department;*

Natural join does an equality on common attributes – doesn't work here:
select *
from *instructor* **natural join** *advisor;*

Instead can use "on" construct (or where clause as above):
select *
from *instructor* **join** *advisor* **on** *(i_id = id);*

Multi-table Queries

3-Table Query to get a list of instructor-teaches-course information:

select *i.name* **as** *instructor_name, c.title* **as** *course_name*
from *instructor i, course c, teaches*
where *i.ID = teaches.ID and c.course_id = teaches.course_id;*

Beware of unintended common names (happens often)
You may think the following query has the same result as above – it doesn't

select *name*, *title*
from *instructor* **natural join** *course* **natural join** *teaches;*

I prefer avoiding "natural joins" for that reason

Note: On the small dataset, the above two have the same answer, but not on the large dataset. Large dataset has cases where an instructor teaches a course from a different department.

Set operations

Find courses that ran in Fall 2009 or Spring 2010

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

In both:

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

In Fall 2009, but not in Spring 2010:

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

Set operations: Duplicates

Union/Intersection/Except eliminate duplicates in the answer (the other SQL commands don't) (e.g., try 'select dept_name from instructor').

Can use "union all" to retain duplicates.

NOTE: The duplicates are retained in a systematic fashion (for all SQL operations)

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*

Set operations: Duplicates

Union/Intersection/Except eliminate duplicates in the answer (the other SQL commands don't) (e.g., try 'select dept_name from instructor').

Can use "union all" to retain duplicates.

NOTE: The duplicates are retained in a systematic fashion (for all SQL operations)

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*

Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

SQL: Nulls

The "dirty little secret" of SQL

(major headache for query optimization)

Can be a value of any attribute

e.g: branch =

bname	**bcity**	**assets**
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

What does this mean?

(unknown) We don't know Waltham's assets?

(inapplicable) Waltham has a special kind of account without assets

(withheld) We are not allowed to know

SQL: Nulls

Arithmetic Operations with `Null`

`n + NULL = NULL` (similarly for all *arithmetic ops*: `+`, `-`, `*`, `/`, `mod`, …)

e.g: branch =

bname	**bcity**	**assets**
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

```
SELECT bname, assets * 2 as a2
FROM branch
```

=

bname	**a2**
Downtown	18M
Perry	3.4M
Mianus	.8M
Waltham	NULL

SQL: Nulls

Boolean Operations with `Null`

`n < NULL = UNKNOWN` (similarly for all *boolean ops*: `>, <=, >=, <>, =, …`)

e.g: branch =

bname	**bcity**	**assets**
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

```
SELECT *
FROM branch
WHERE assets = NULL
```

=

bname	**bcity**	**assets**

Counter-intuitive: NULL * 0 = NULL

Counter-intuitive: select * from movies
where length >= 120 or length <= 120

SQL: Nulls

Boolean Operations with `Null`

`n < NULL = UNKNOWN` (similarly for all *boolean ops*: `>, <=, >=, <>, =, …`)

e.g: branch =

bname	**bcity**	**assets**
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

```
SELECT *
FROM branch
WHERE assets IS NULL
```

=

bname	**bcity**	**assets**
Waltham	Boston	NULL

SQL: Unknown

Boolean Operations with `Unknown`

`n < NULL = UNKNOWN` (similarly for all *boolean ops*: `>, <=, >=, <>, =, …`)

`FALSE OR UNKNOWN = UNKNOWN`

`TRUE AND UNKNOWN = UNKNOWN`

Intuition: substitute each of TRUE, FALSE for unknown. If different answer results, results is unknown

`UNKNOWN OR UNKNOWN = UNKNOWN`

`UNKNOWN AND UNKNOWN = UNKNOWN`

`NOT (UNKNOWN) = UNKNOWN`

Can write:

```
SELECT …
FROM …
WHERE booleanexp IS UNKNOWN
```

UNKNOWN tuples are not included in final result

Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)

Aggregates

Find the average salary of instructors in the Computer Science

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

Other common aggregates:
max, min, sum, count, stdev, …

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

Can specify aggregates in any query.

Find max salary over instructors teaching in S'10

```
select max(salary)
from teaches natural join instructor
where semester = 'Spring' and year = 2010;
```

Aggregate result can be used as a scalar.
Find instructors with max salary:

```
select *
from instructor
where salary = (select max(salary) from instructor);
```

Aggregates

Aggregate result can be used as a scalar.
Find instructors with max salary:

select *
from *instructor*
where *salary* **= (select max***(salary)* **from** *instructor);*

Following doesn't work:

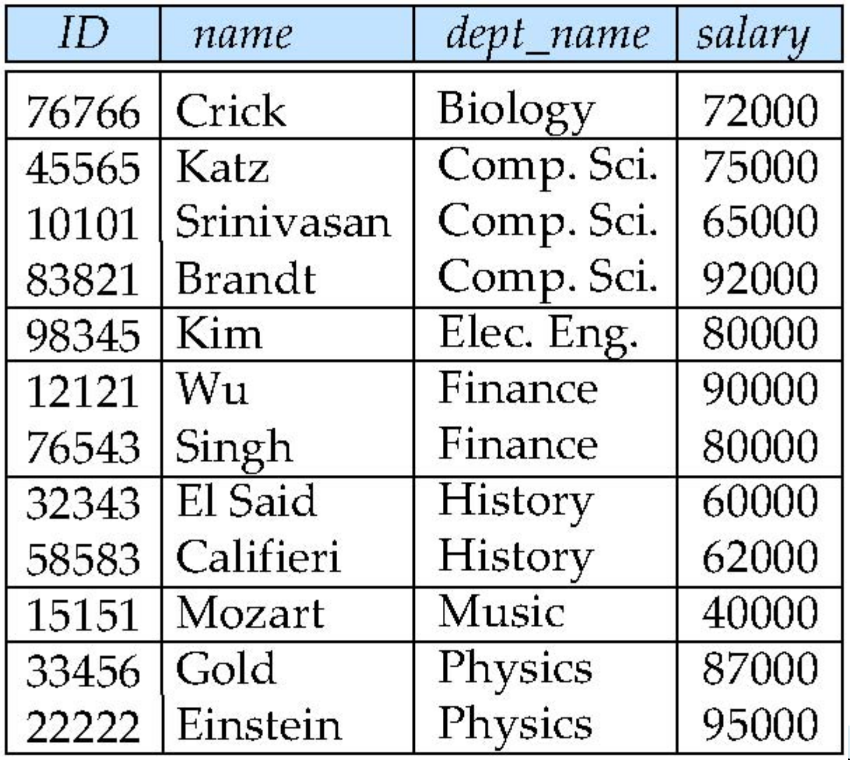
select *
from *instructor*
where *salary* **= max***(salary);*

select *name,* **max***(salary)*
from *instructor*
where *salary* **= max***(salary);*

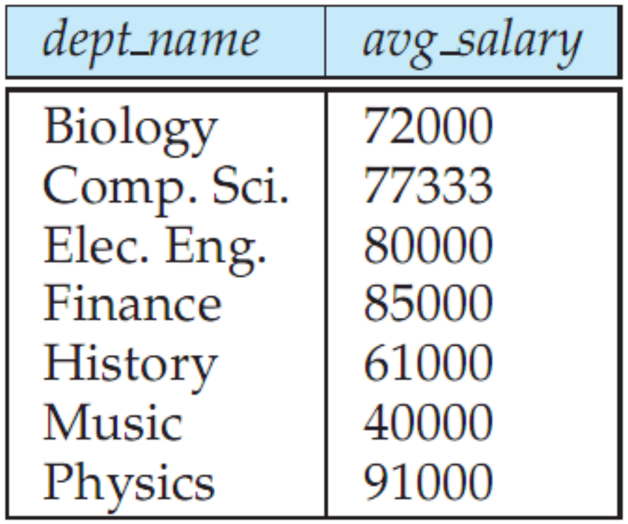
Aggregates: Group By

Split the tuples into groups, and computer the aggregate for each group

select *dept_name*, **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

Aggregates: Group By

Attributes in the select clause must be aggregates, or must appear in the group by clause. Following wouldn't work

select *dept_name*, ID, **avg** (*salary*)
from *instructor*
group by *dept_name*;

"having" can be used to select only some of the groups.

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

Aggregates and NULLs

Given

branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

Aggregate Operations

```
SELECT SUM (assets) =
FROM branch
```

SUM
11.1 M

`NULL` *is ignored for SUM*

Same for `AVG` *(3.7M),* `MIN` *(0.4M),* `MAX` *(9M)*

Also for COUNT(assets) -- returns 3

But `COUNT (*)` *returns*

COUNT
4

Aggregates and NULLs

Given

branch =

<u>bname</u>	<u>bcity</u>	<u>assets</u>

```
SELECT SUM (assets) =
FROM branch
```

<u>SUM</u>
NULL

- *Same as* `AVG, MIN, MAX`
- *But* `COUNT (assets)` *returns*

<u>COUNT</u>
0

Summary

- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)