Database Management Systems (CSE-251)

Presented by

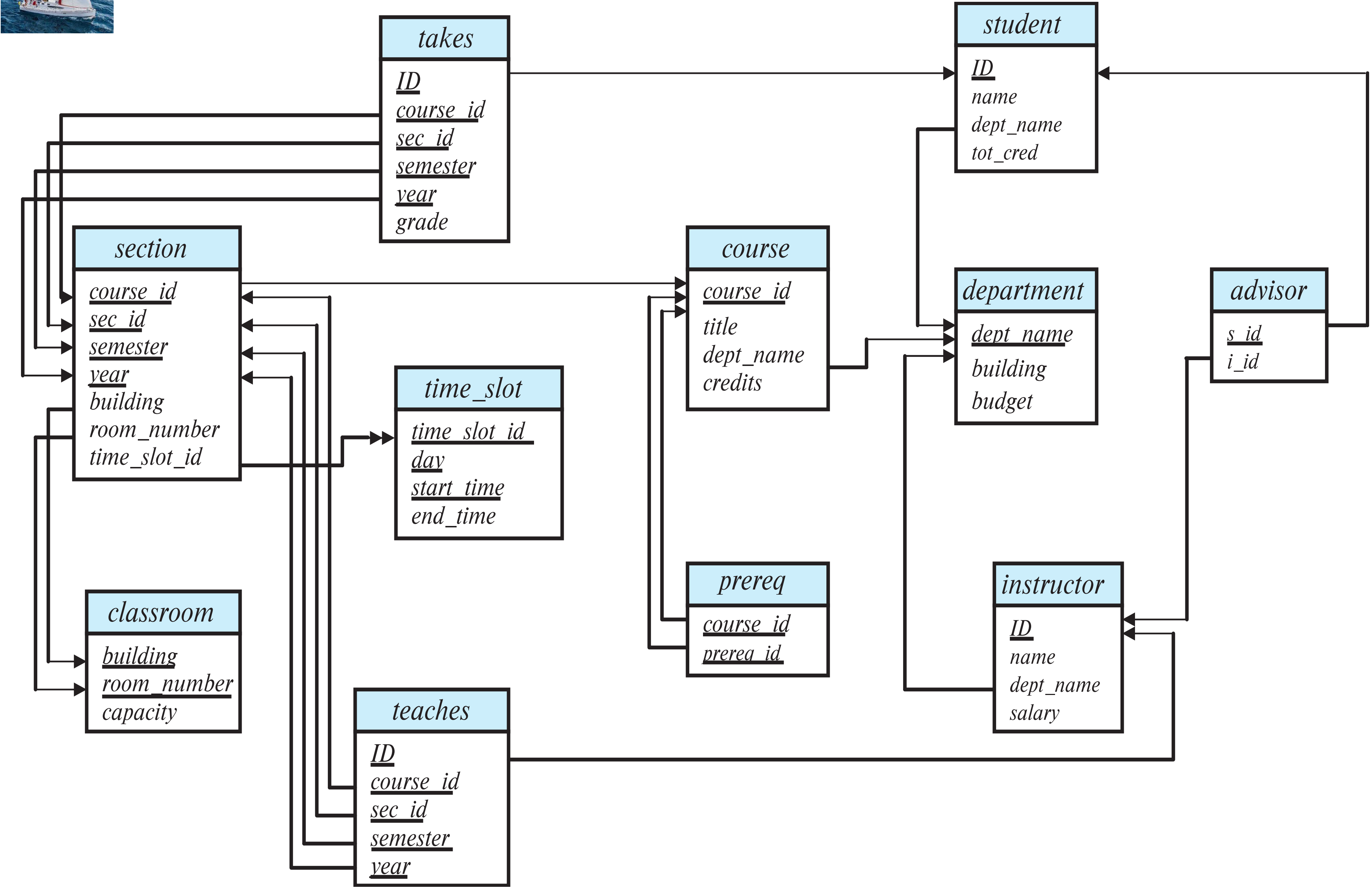
Md. Atiqul Islam Rizvi

Assistant Professor, Dept. of CSE, CUET

Chapter 3: Introduction to SQL

Database System Concepts, 7th Ed.

Schema Diagram for University Database



takes
ID
course_id
sec_id
semester
year
grade

student
ID
name
dept_name
tot_cred

section
course_id
sec_id
semester
year
building
room_number
time_slot_id

course
course_id
title
dept_name
credits

department
dept_name
building
budget

advisor
s_id
i_id

time_slot
time_slot_id
day
start_time
end_time

classroom
building
room_number
capacity

prereq
course_id
prereq_id

instructor
ID
name
dept_name
salary

teaches
ID
course_id
sec_id
semester
year

SQL Query Language

- SQL query language is nonprocedural. A query takes as input several tables (possibly only one) and always returns a single table.
- Example - **find all instructors in Comp. Sci. dept**

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.'

- SQL is **NOT** a Turing machine equivalent language
- SQL does not support actions such as input from users, output to displays, or communication over the network.
- Such computations and actions must be written in a **host language**, such as C/C++, Java or Python, that supports embedded SQL queries
- Application programs generally access databases through one of
  - Language extensions to allow embedded SQL
  - Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

SQL Parts

- **DDL** – provides commands for defining relation schemas, deleting relations and modifying relation schemas.
- **DML** – provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- **Integrity** – the DDL includes commands for specifying integrity constraints.
- **View definition** – The DDL includes commands for defining views.
- **Transaction control** – includes commands for specifying the beginning and ending of transactions.
- **Embedded SQL and dynamic SQL** – define how SQL statements can be embedded within general-purpose programming languages.
- **Authorization** – includes commands for specifying access rights to relations and views.

Data Definition Language (DDL)

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

- The schema for each relation.
- Domain constraints
  - domain of possible values (data type, date, time, etc.)
- The Integrity constraints
  - Primary key (ID uniquely identifies instructors)
- The set of indices to be maintained for each relation.
- Security and authorization information for each relation.
  - Who can access what
- The physical storage structure of each relation on disk.

- DDL compiler generates a set of table templates stored in a ***data dictionary***
- Data dictionary contains metadata (i.e., data about data)

Create Table Construct

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

  create table *r*

  (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (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

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 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.

Integrity Constraints in Create Table

- Each type may include a special value called the ‘NULL’ value. It indicates an absent value that may exist but unknown or may not exist at all.

- Types of integrity constraints
  - **primary key** (A_1, ..., A_n)
  - **foreign key** (A_m, ..., A_n) **references** *r*
  - **not null**

- SQL prevents any update to the database that violates an integrity constraint.

And a Few More Relation Definitions

- **Example – Create *student* relation**
- **create table** *student* (

```
        ID              varchar(5) not null,
        name            varchar(20) not null,
        dept_name       varchar(20),
        tot_cred        numeric(3,0),
        primary key (ID),
        foreign key (dept_name) references department);
```

- **Example – Create *takes* relation**
- **create table** *takes* (

```
        ID              varchar(5) not null,
        course_id       varchar(8) not null,
        sec_id          varchar(8) not null,
        semester        varchar(6) not null,
        year            numeric(4,0) not null,
        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);
```

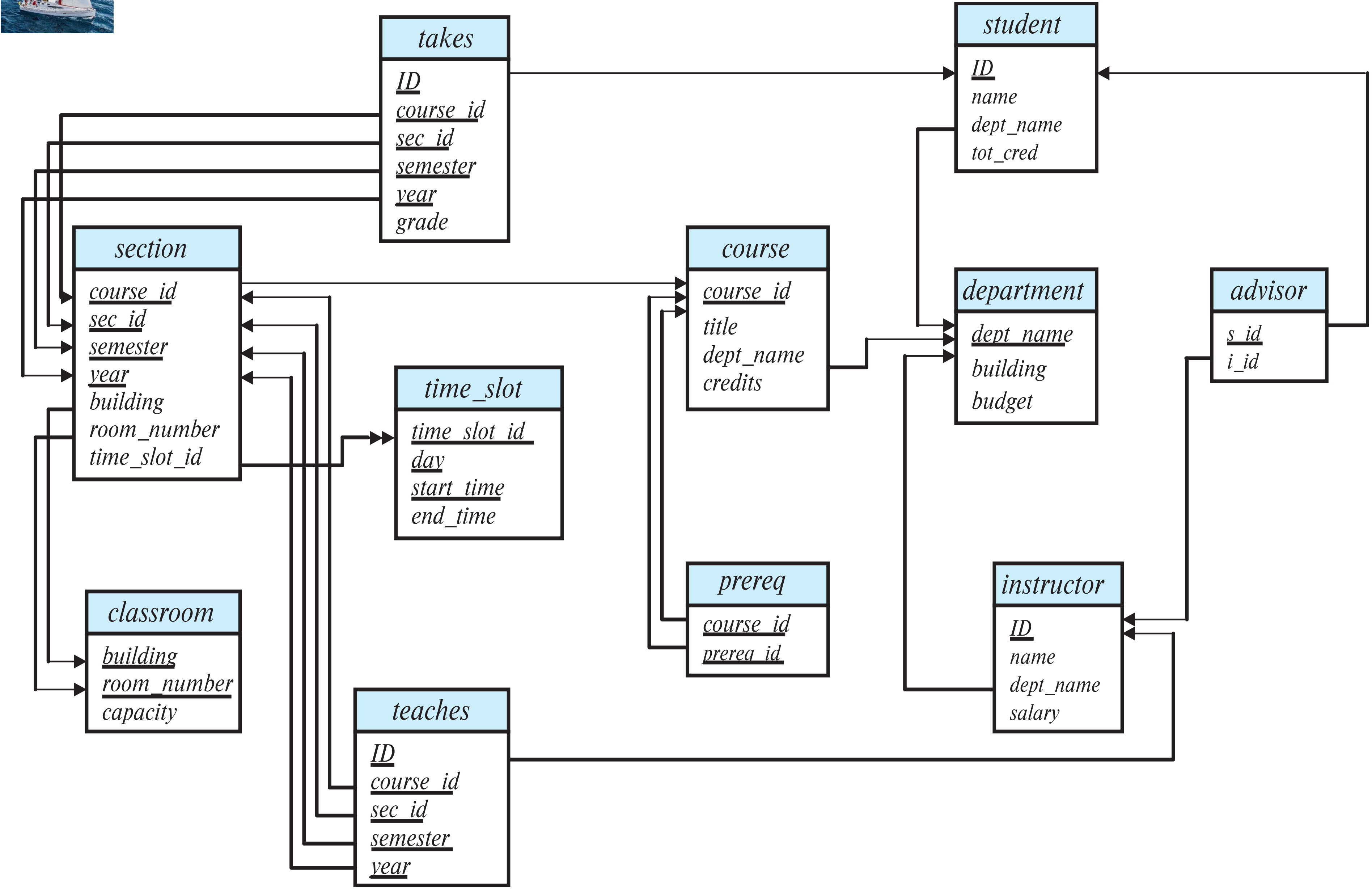
Database Management Systems (CSE-251)

Presented by

Md. Atiqul Islam Rizvi

Assistant Professor, Dept. of CSE, CUET

Schema Diagram for University Database



Modifications to tables

- **Drop Table** – used to remove a relation from SQL database. Drastic action than delete.
  - **drop table** *prereq*

- **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.

  - **alter table** *r* **change** *Previous_A New_A New_D New_Constraints*

Rename, Truncate

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

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

- **Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.**
  - **select distinct** *T.name*
    from *instructor* **as** *T*, *instructor* **as** *S*
    where *T.salary* > *S.salary* **and** *S.dept_name* = *'Comp. Sci.'*

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

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

- **Truncate Table** – remove all records from a table, including all spaces allocated for the records are removed.
  - **truncate table** *prereq*

Data Manipulation Language (DML)

- Language for accessing and updating the data organized by the appropriate data model
  - DML also known as query language
- There are basically two types of data-manipulation language
  - **Procedural DML** -- require a user to specify what data are needed and how to get those data.
  - **Declarative DML** -- require a user to specify what data are needed without specifying how to get those data.
- Declarative DMLs are usually easier to learn and use than are procedural DMLs.
- Declarative DMLs are also referred to as non-procedural DMLs

DML - Insertion

- **Add a new tuple to *course***

```
insert into course
      values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- or equivalently

```
insert into course (course_id, title, dept_name, credits)
      values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- **Add a new tuple to *student* with *tot_creds* set to null**

```
insert into student
      values ('3003', 'Green', 'Finance', null);
```

- **Insert**

```
insert into instructor values ('10211', 'Smith', 'Biology', 66000);
```

DML - Deletion

- **Delete**
  - Remove all tuples from the *student* relation, but retains the relation structure.

```
delete from student
```

- **Delete all instructors**

```
delete from instructor
```

- **Delete all instructors from the Finance department**

```
delete from instructor
where dept_name= 'Finance';
```

DML - Updates

- Give a 5% salary raise to all instructors

```
update instructor
   set salary = salary * 1.05
```

- Give a 5% salary raise to those instructors who earn less than 70000

```
update instructor
   set salary = salary * 1.05
   where salary < 70000;
```

Basic Query Structure

- A typical SQL query has the form:

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

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

- A query takes the input relations as list in the **from** clause, operates on them as specified in the **where** and **select** clause, then produces the result.

- The result of an SQL query is a relation.

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*

- An asterisk in the select clause denotes "all attributes"

  select *
  from *instructor*

- According to mathematical definition of relational model, a relation is a set. Thus, duplicate tuples should never appear in relations.

The select Clause (Cont.)

- But, 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*

dept_name
Comp. Sci.
Finance
Music
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

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*

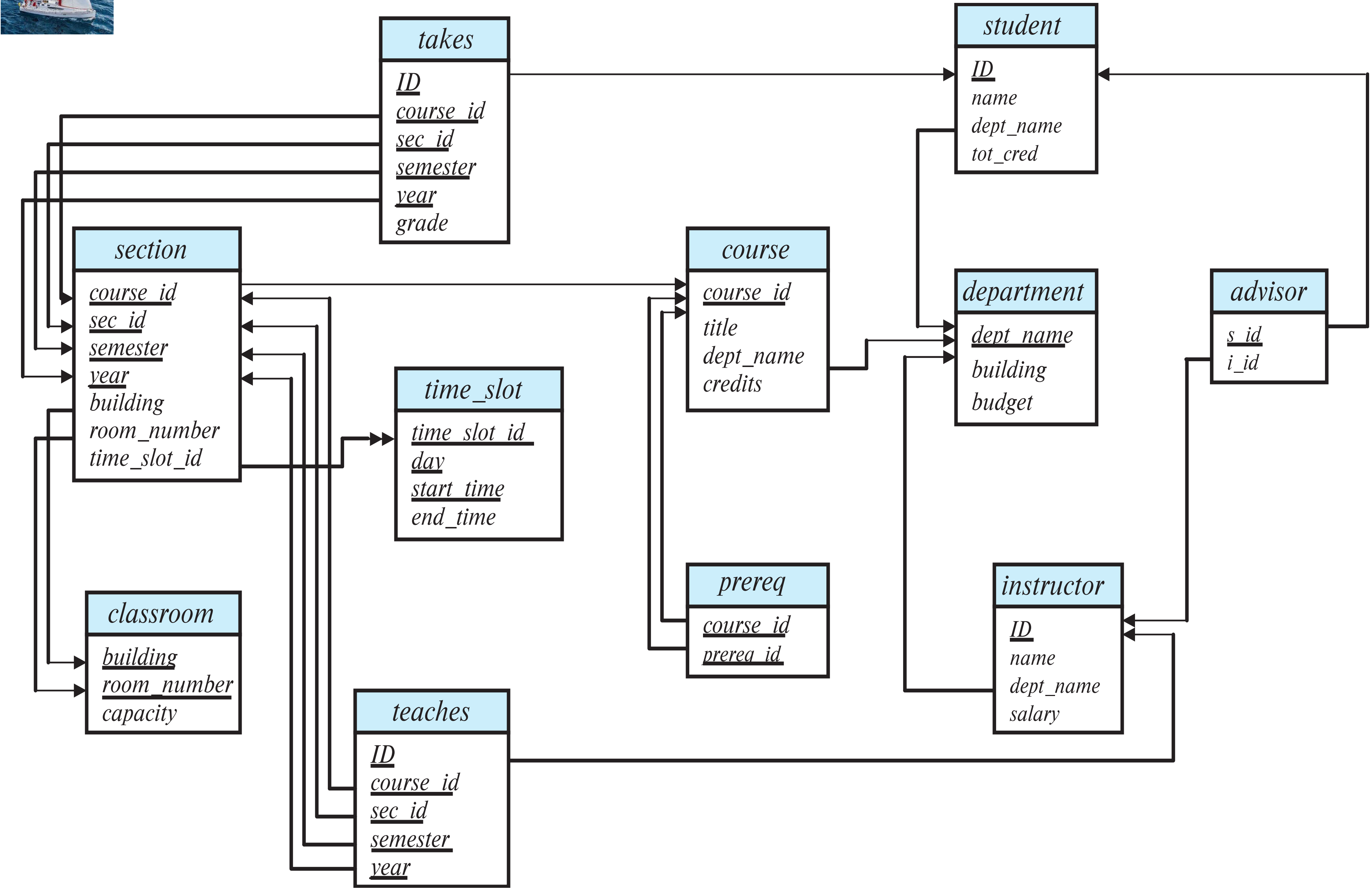
Database Management Systems (CSE-251)

Presented by

Md. Atiqul Islam Rizvi

Assistant Professor, Dept. of CSE, CUET

Schema Diagram for University Database



Referential Integrity

- Foreign *keys can be* specified as part of the SQL **create table** statement. By default, a foreign key references the primary-key attributes of the referenced table. SQL allows a list of attributes of the referenced relation to be specified explicitly.

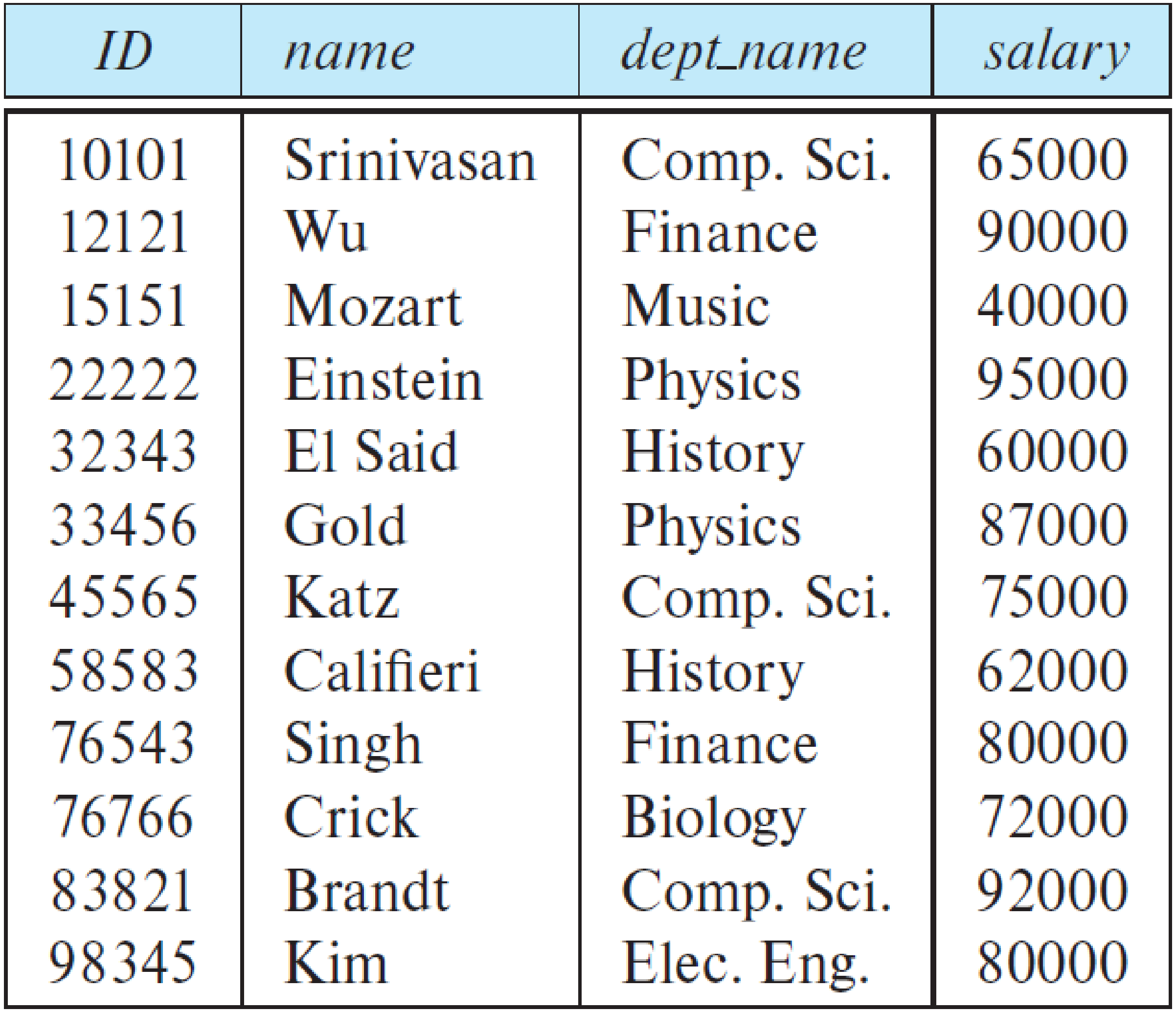
  foreign key (*dept_name*) **references** *department* (*dept_name*)

- When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation. An alternative, in case of delete or update is to cascade.

```
create table course (
      (…
       dept_name varchar(20),
       foreign key (dept_name) references department
            on delete cascade
            on update cascade,
        . . .)
```

- Instead of cascade we can use : **set null**, **set default, restrict, no action**

Instance of *instructor* Relation



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

Figure 2.1 The *instructor* relation.

The where Clause

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

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

- SQL allows the use of the logical connectives **and, or,** and **not**
- The operands of the logical connectives can be expressions involving the comparison operators <, <=, >, >=, =, and <>.
- Comparisons can be applied to results of arithmetic expressions
- **Find all instructors in Comp. Sci. dept with salary > 70000**

```
select name
from instructor
where dept_name = 'Comp. Sci.'  and salary > 70000
```

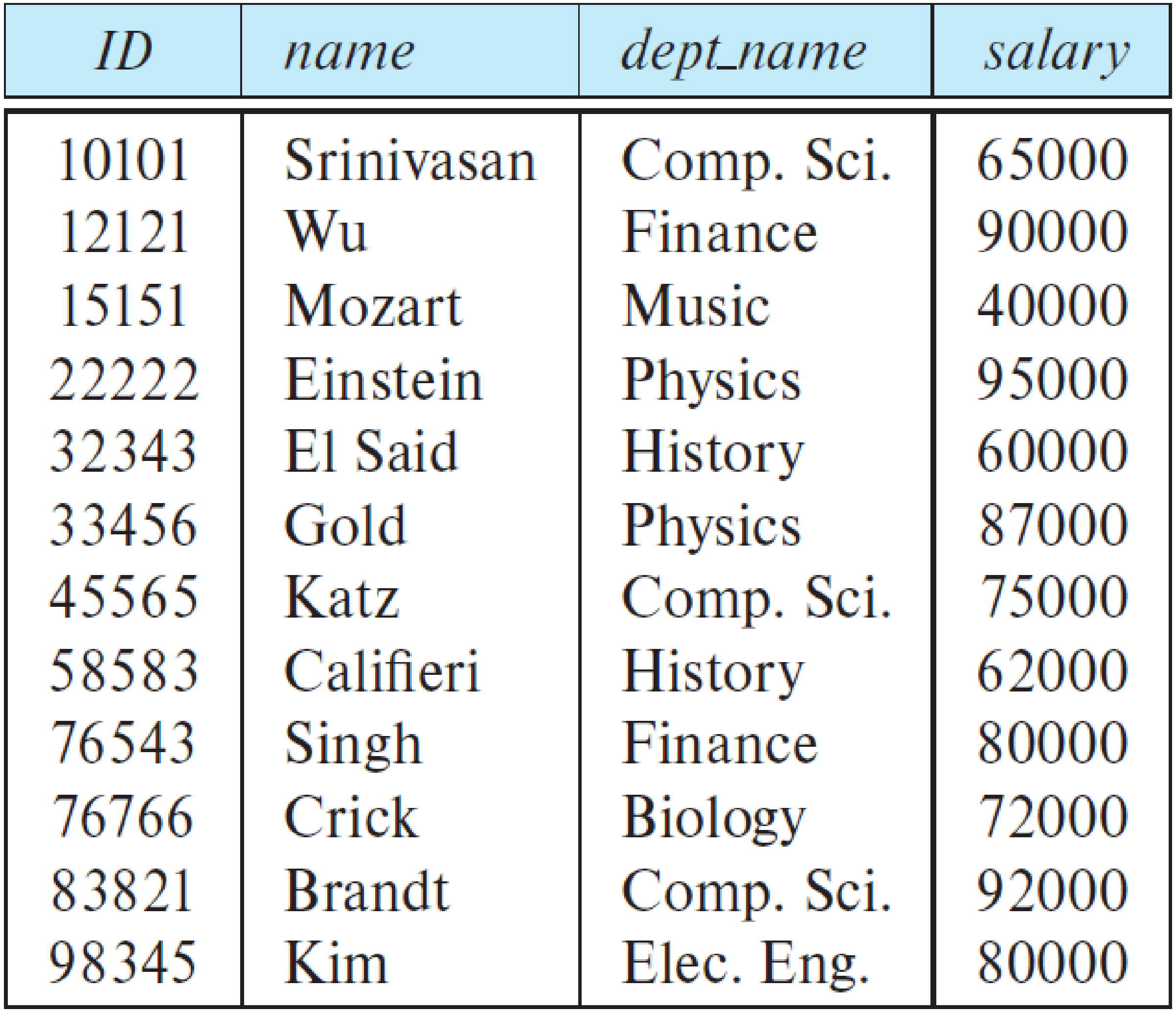
name
Katz
Brandt

Where Clause Predicates

- SQL includes a **between** comparison operator

- Example: **Find the names of all instructors with salary between $90,000 and $100,000 (that is, >= $90,000 and <= $100,000)**
  - **select** *name*
    from *instructor*
    where *salary* **between** 90000 **and** 100000

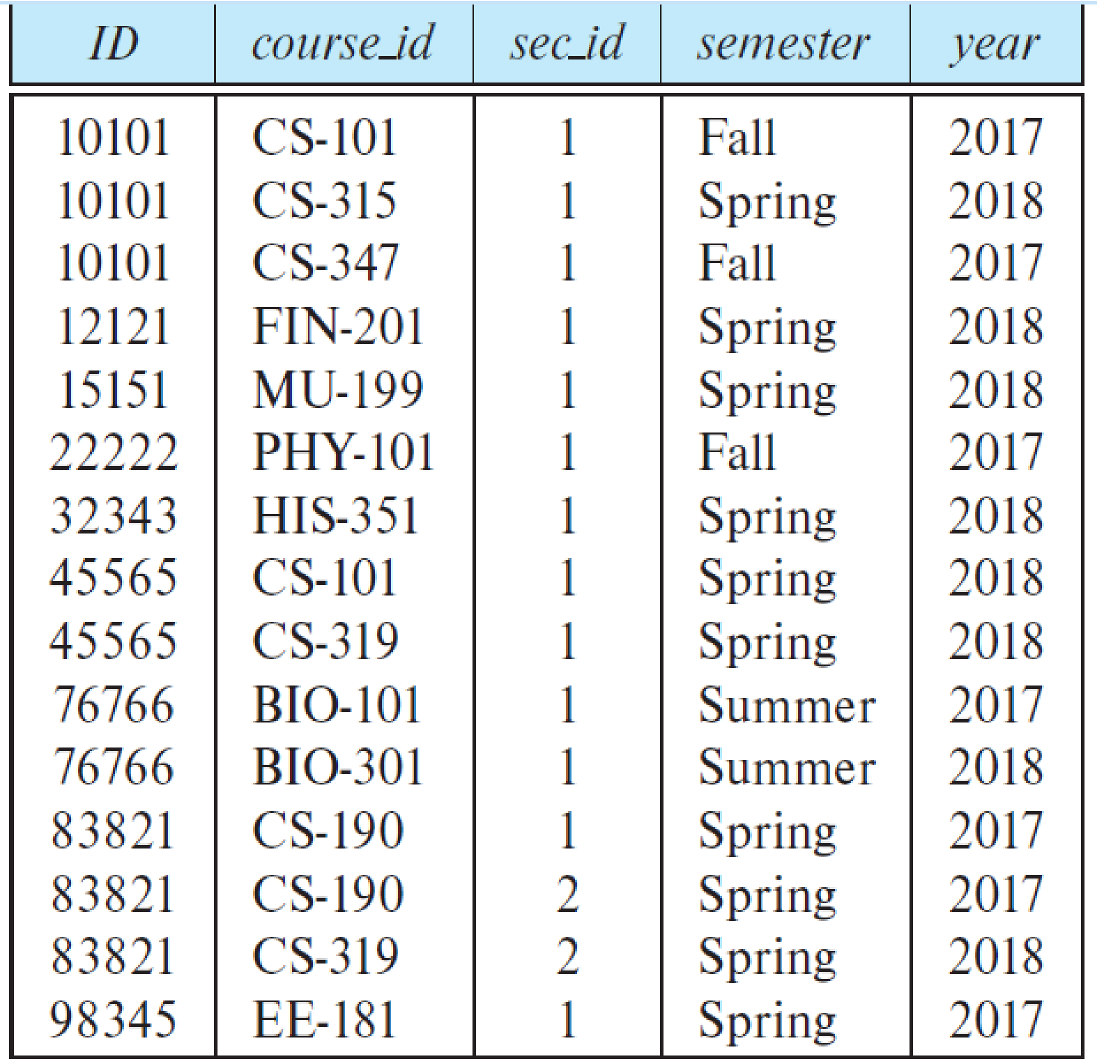
Instance of *instructor* Relation



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

Figure 2.1 The *instructor* relation.

Instance of *teaches* Relation



ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

Figure 2.7 The *teaches* relation.

The from Clause

- The **from** clause lists the relations involved in the query
- 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).

The *instructor* x *teaches* table

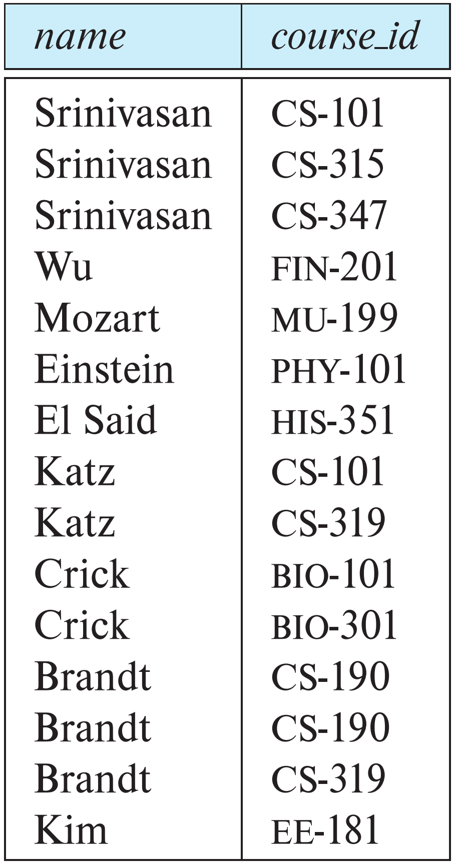
instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Examples

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

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



name	*course_id*
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

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* **asc**, *name* **desc**

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 (Cont.)

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

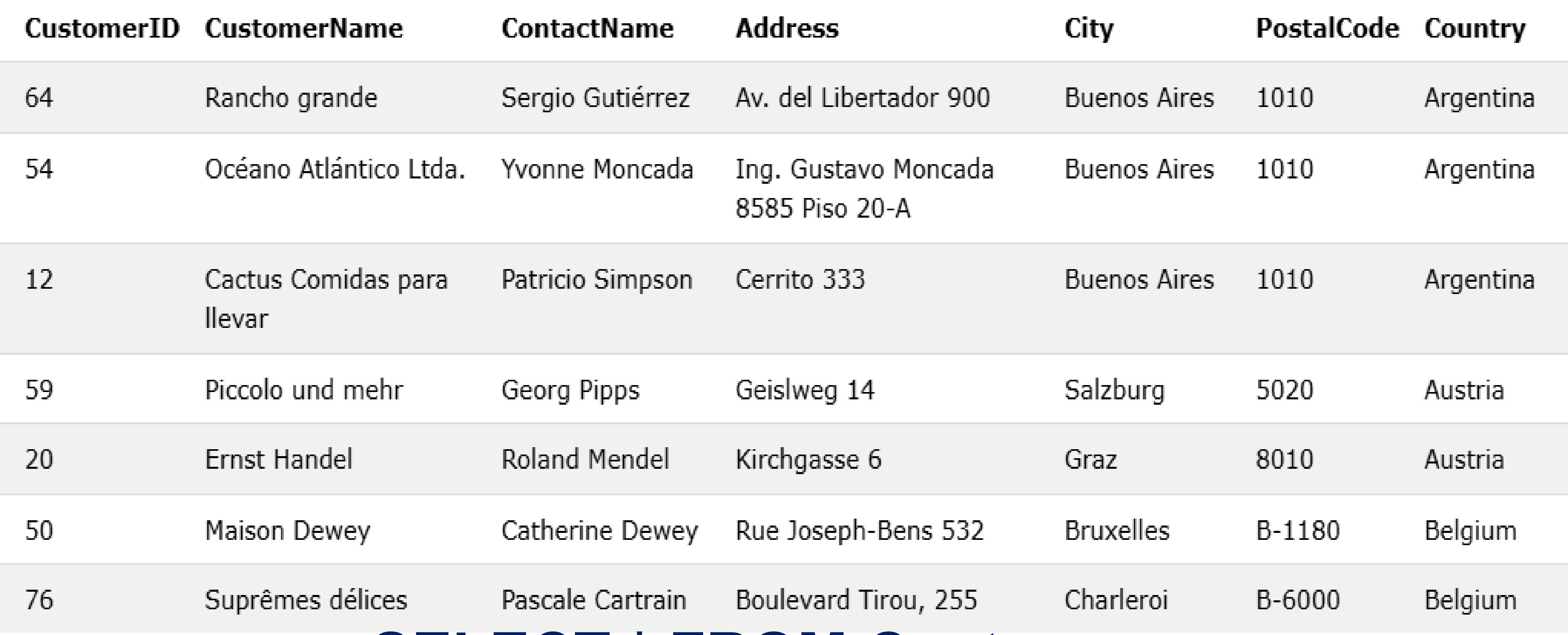
Database Management Systems (CSE-251)

Presented by

Md. Atiqul Islam Rizvi

Assistant Professor, Dept. of CSE, CUET

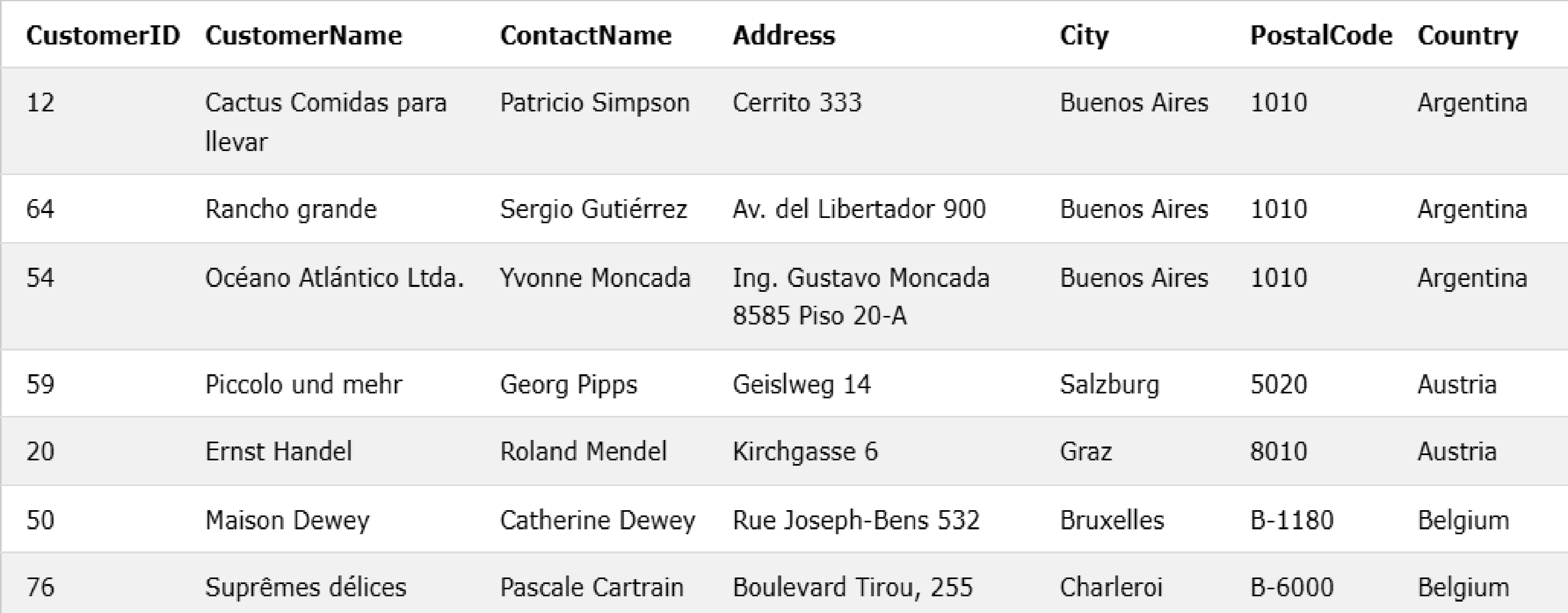
ORDER BY On Multiple Columns



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
64	Rancho grande	Sergio Gutiérrez	Av. del Libertador 900	Buenos Aires	1010	Argentina
54	Océano Atlántico Ltda.	Yvonne Moncada	Ing. Gustavo Moncada 8585 Piso 20-A	Buenos Aires	1010	Argentina
12	Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Aires	1010	Argentina
59	Piccolo und mehr	Georg Pipps	Geislweg 14	Salzburg	5020	Austria
20	Ernst Handel	Roland Mendel	Kirchgasse 6	Graz	8010	Austria
50	Maison Dewey	Catherine Dewey	Rue Joseph-Bens 532	Bruxelles	B-1180	Belgium
76	Suprêmes délices	Pascale Cartrain	Boulevard Tirou, 255	Charleroi	B-6000	Belgium

```
SELECT * FROM Customers
ORDER BY Country ASC;
```

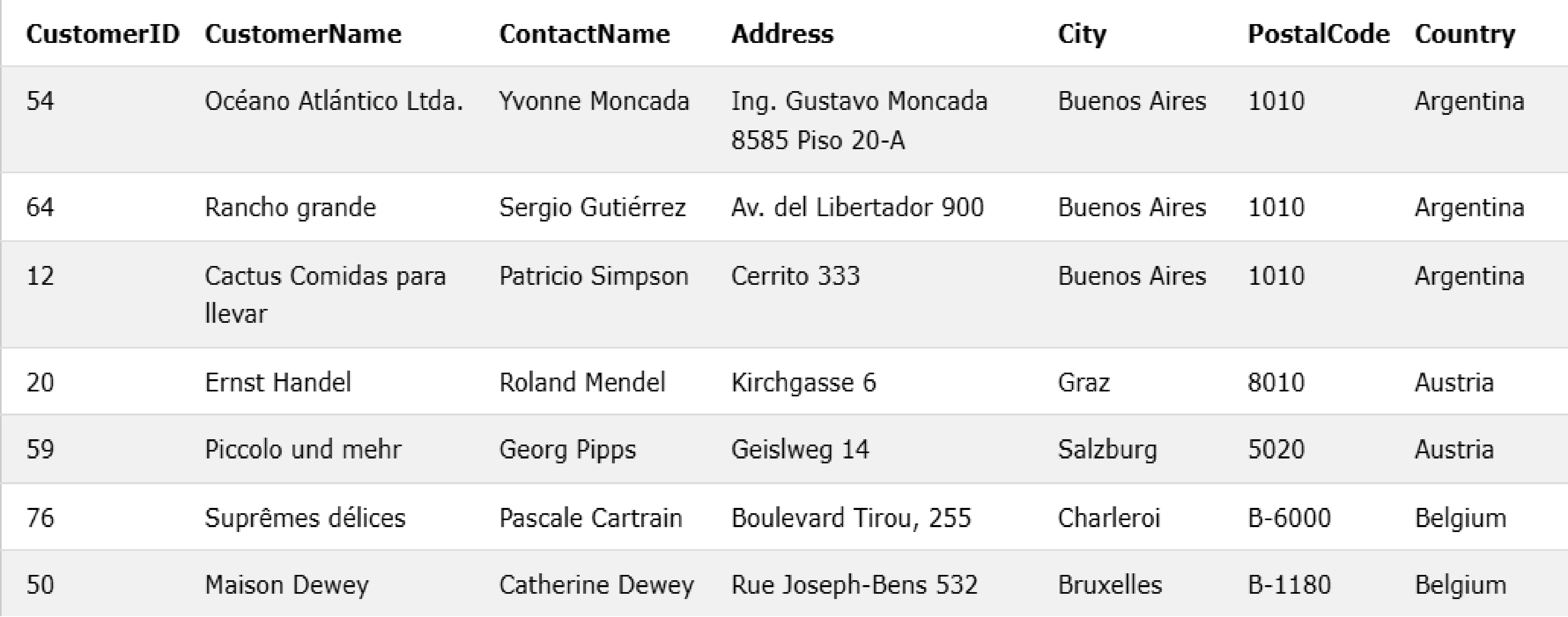
ORDER BY On Multiple Columns



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
12	Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Aires	1010	Argentina
64	Rancho grande	Sergio Gutiérrez	Av. del Libertador 900	Buenos Aires	1010	Argentina
54	Océano Atlántico Ltda.	Yvonne Moncada	Ing. Gustavo Moncada 8585 Piso 20-A	Buenos Aires	1010	Argentina
59	Piccolo und mehr	Georg Pipps	Geislweg 14	Salzburg	5020	Austria
20	Ernst Handel	Roland Mendel	Kirchgasse 6	Graz	8010	Austria
50	Maison Dewey	Catherine Dewey	Rue Joseph-Bens 532	Bruxelles	B-1180	Belgium
76	Suprêmes délices	Pascale Cartrain	Boulevard Tirou, 255	Charleroi	B-6000	Belgium

SELECT * FROM Customers
ORDER BY Country ASC, ContactName ASC;

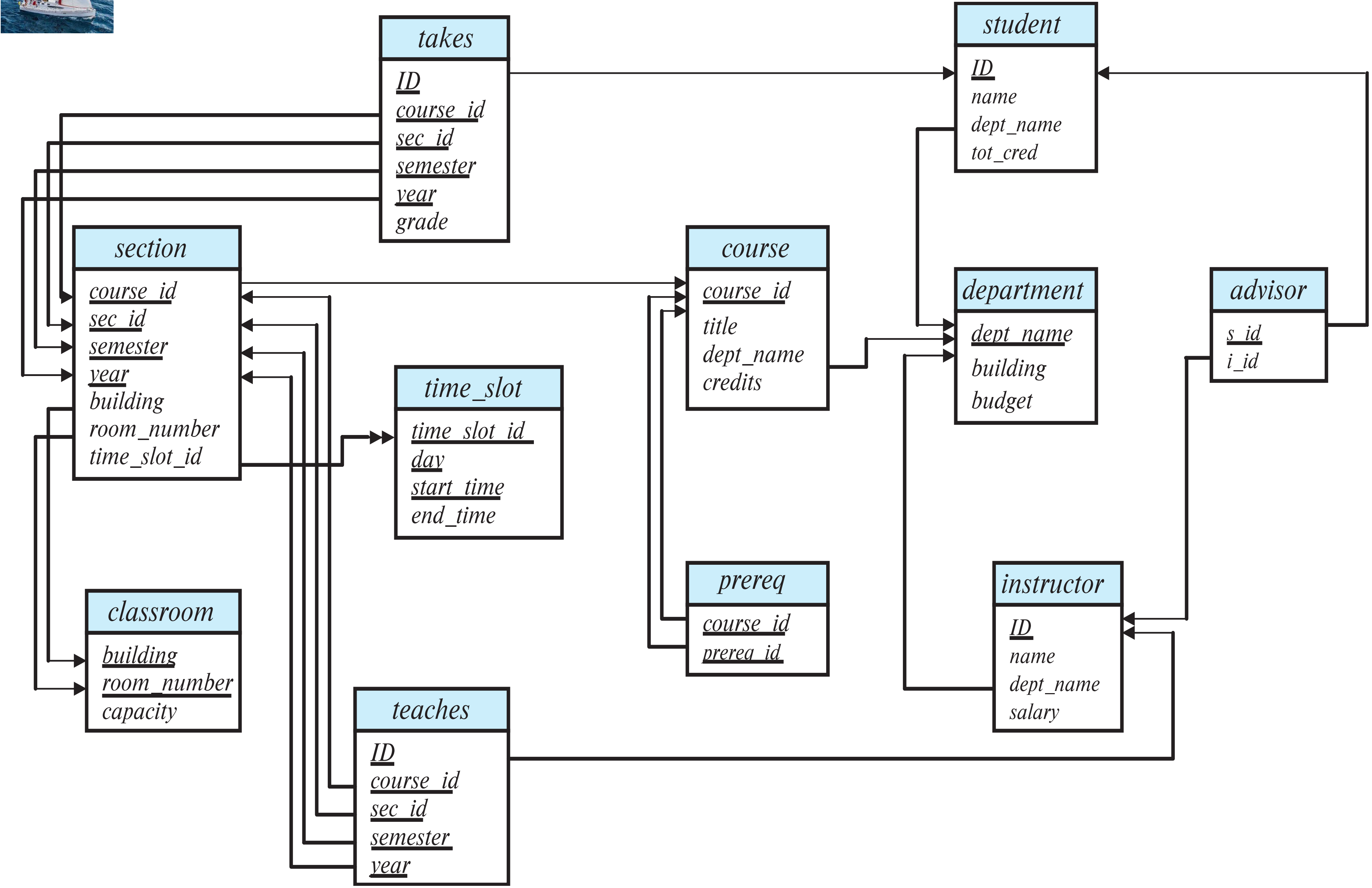
ORDER BY On Multiple Columns



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
54	Océano Atlántico Ltda.	Yvonne Moncada	Ing. Gustavo Moncada 8585 Piso 20-A	Buenos Aires	1010	Argentina
64	Rancho grande	Sergio Gutiérrez	Av. del Libertador 900	Buenos Aires	1010	Argentina
12	Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Aires	1010	Argentina
20	Ernst Handel	Roland Mendel	Kirchgasse 6	Graz	8010	Austria
59	Piccolo und mehr	Georg Pipps	Geislweg 14	Salzburg	5020	Austria
76	Suprêmes délices	Pascale Cartrain	Boulevard Tirou, 255	Charleroi	B-6000	Belgium
50	Maison Dewey	Catherine Dewey	Rue Joseph-Bens 532	Bruxelles	B-1180	Belgium

```
SELECT * FROM Customers
ORDER BY Country ASC, ContactName DESC;
```

Schema Diagram for University Database



Set Operations

- **Find courses that ran in Fall 2017 or in Spring 2018**

```
(select course_id  from section where semester = 'Fall' and year = 2017)
union
(select course_id  from section where semester = 'Spring' and year = 2018);
```

- **Find courses that ran in Fall 2017 and in Spring 2018**

```
(select course_id  from section where semester = 'Fall' and year = 2017)
intersect
(select course_id  from section where semester = 'Spring' and year = 2018);
```

- **Find courses that ran in Fall 2017 but not in Spring 2018**

```
(select course_id  from section where semester = 'Fall' and year = 2017)
except
(select course_id  from section where semester = 'Spring' and year = 2018);
```

- Set operations **union**, **intersect**, and **except**
  - Each of the above operations automatically eliminates duplicates
- To retain all duplicates use the
  - **union all**, **intersect all**, **except all**.

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 currently under recalculation.**

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

- The predicate **is not null** succeeds if the value on which it is applied is not null.

Null Values (Cont.)

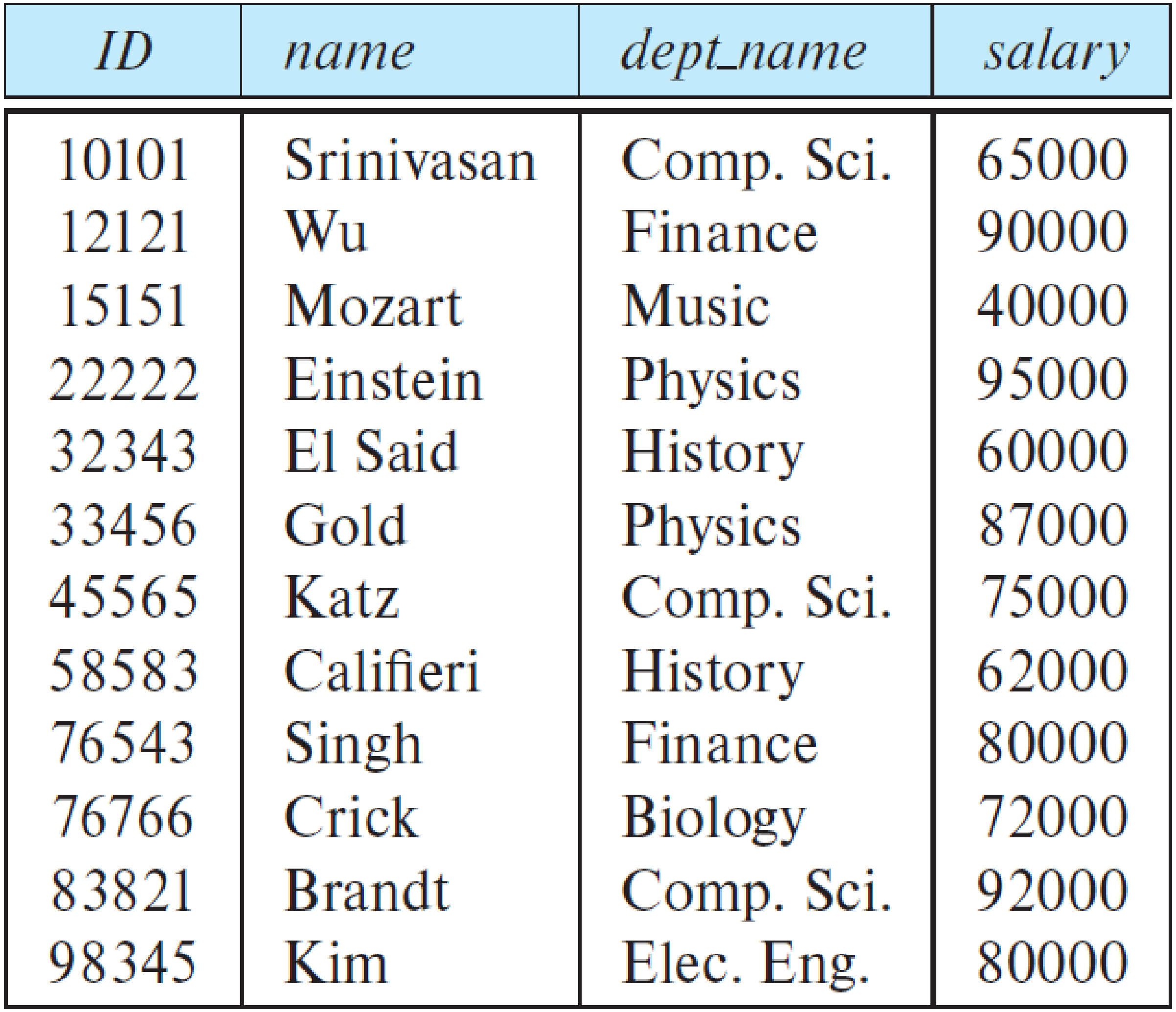
- SQL treats as **unknown** the result of any comparison involving a null value (other than predicates **is null** and **is not null**).
  - Example*: 5 <* **null** or **null** <> **null** or **null** = **null**
- The predicate in a **where** clause can involve Boolean operations (**and**, **or**, **not**); thus the definitions of the Boolean operations need to be extended to deal with the value **unknown**.
  - **and** : *(true* **and** *unknown) = unknown,*
    (false **and** *unknown) = false,*
    (unknown **and** *unknown) = unknown*
  - **or:** (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - **not:** (**not** unknown) = unknown
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

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

Instance of *instructor* Relation



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

Figure 2.1 The *instructor* relation.

Aggregate Functions Examples

- **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 2018 semester**

  -
    ```
    select count (distinct ID)
    from teaches
    where semester = 'Spring' and year = 2018;
    ```
- **Find the number of tuples in the *course* relation**

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

Aggregate Functions – Group By

- **Find the average salary of instructors in each department**
  - **select** *dept_name*, **avg** (*salary*) **as** *avg_salary*
from *instructor*
group by *dept_name*;

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

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

Aggregation (Cont.)

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

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

Aggregate Functions – Having Clause

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

```
select dept_name, avg (salary) as 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
- Thus, the sequence of operations are –
  - **from** clause is evaluated at first to get a relation
  - If **where** is present, it is applied on resulting relation
  - Tuples satisfying **where**, are placed into groups using **group by,** if present.
  - If **having** is present, then applied to each group.
  - Finally, **select** clause uses the resulting groups to generate tuples, applying the aggregate functions to get a single result tuple for each group

Database Management Systems (CSE-251)

Presented by

Md. Atiqul Islam Rizvi

Assistant Professor, Dept. of CSE, CUET

Nested Subqueries

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

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  where *P;*

  as follows:

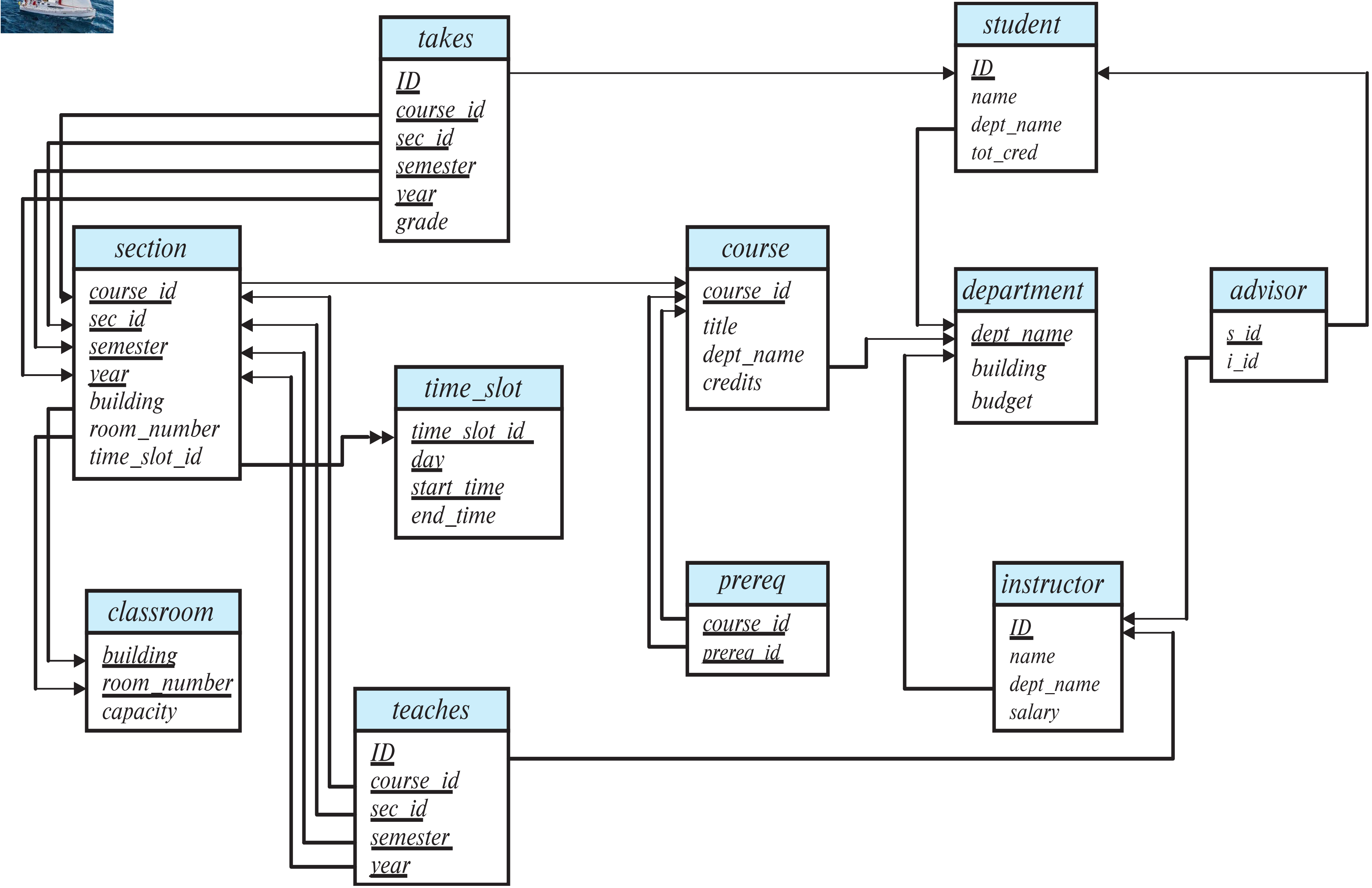
  - **Where clause:** *P* can be replaced with an expression of the form:

    B <operation> (subquery)

    B is an attribute and <operation> to be defined later.
  - **From clause:** r_i can be replaced by any valid subquery
  - **Select clause:**

    A_i can be replaced be a subquery that generates a single value.

Schema Diagram for University Database



takes
ID
course_id
sec_id
semester
year
grade

student
ID
name
dept_name
tot_cred

section
course_id
sec_id
semester
year
building
room_number
time_slot_id

course
course_id
title
dept_name
credits

department
dept_name
building
budget

advisor
s_id
i_id

time_slot
time_slot_id
day
start_time
end_time

classroom
building
room_number
capacity

prereq
course_id
prereq_id

instructor
ID
name
dept_name
salary

teaches
ID
course_id
sec_id
semester
year

Set Membership

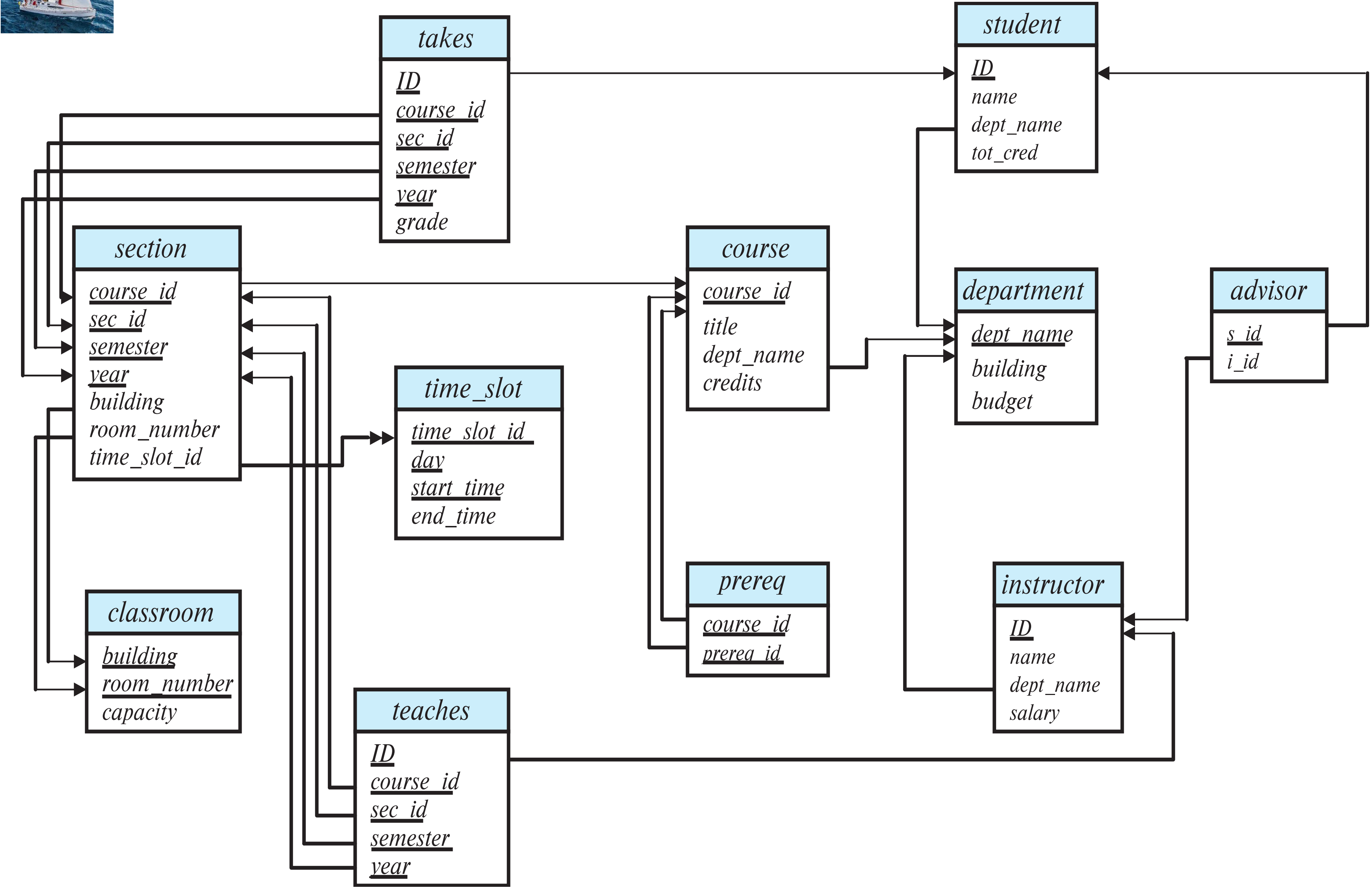
- Find courses offered in Fall 2017 and in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
          course_id in (select course_id
                        from section
                        where semester = 'Spring' and year= 2018);
```

- Find courses offered in Fall 2017 but not in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
          course_id  not in (select course_id
                             from section
                             where semester = 'Spring' and year= 2018);
```

Schema Diagram for University Database



takes
ID
course_id
sec_id
semester
year
grade

student
ID
name
dept_name
tot_cred

section
course_id
sec_id
semester
year
building
room_number
time_slot_id

course
course_id
title
dept_name
credits

department
dept_name
building
budget

advisor
s_id
i_id

time_slot
time_slot_id
day
start_time
end_time

classroom
building
room_number
capacity

prereq
course_id
prereq_id

instructor
ID
name
dept_name
salary

teaches
ID
course_id
sec_id
semester
year

Set Membership (Cont.)

- **Name all instructors whose name is neither “Mozart” nor “Einstein”**

```
select distinct name
from instructor
where name not in ('Mozart', 'Einstein’);
```

- **Find the total number of unique students who have taken course taught by the instructor with *ID* 10101**

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

Set Comparison – "some", "all" Clause

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

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

- Same query using > **some** clause

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

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

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

Definition of "some" Clause

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

$(5 < \textbf{some} \begin{array}{|c|}\hline 0\\\hline 5\\\hline 6\\\hline\end{array}) = \text{true}$ (read: 5 < some tuple in the relation)

$(5 < \textbf{some} \begin{array}{|c|}\hline 0\\\hline 5\\\hline\end{array}) = \text{false}$

$(5 = \textbf{some} \begin{array}{|c|}\hline 0\\\hline 5\\\hline\end{array}) = \text{true}$

$(5 \neq \textbf{some} \begin{array}{|c|}\hline 0\\\hline 5\\\hline\end{array}) = \text{true (since } 0 \neq 5)$

(= **some**) $\equiv$ **in**

However, ($\neq$ **some**) $\not\equiv$ **not in**

Definition of "all" Clause

- F <comp> **all** *r* ⇔ $\forall\, t \in r$ (F <comp> *t*)

$$(5 < \textbf{all} \begin{array}{|c|}\hline 0\\\hline 5\\\hline 6\\\hline\end{array}) = \text{false}$$

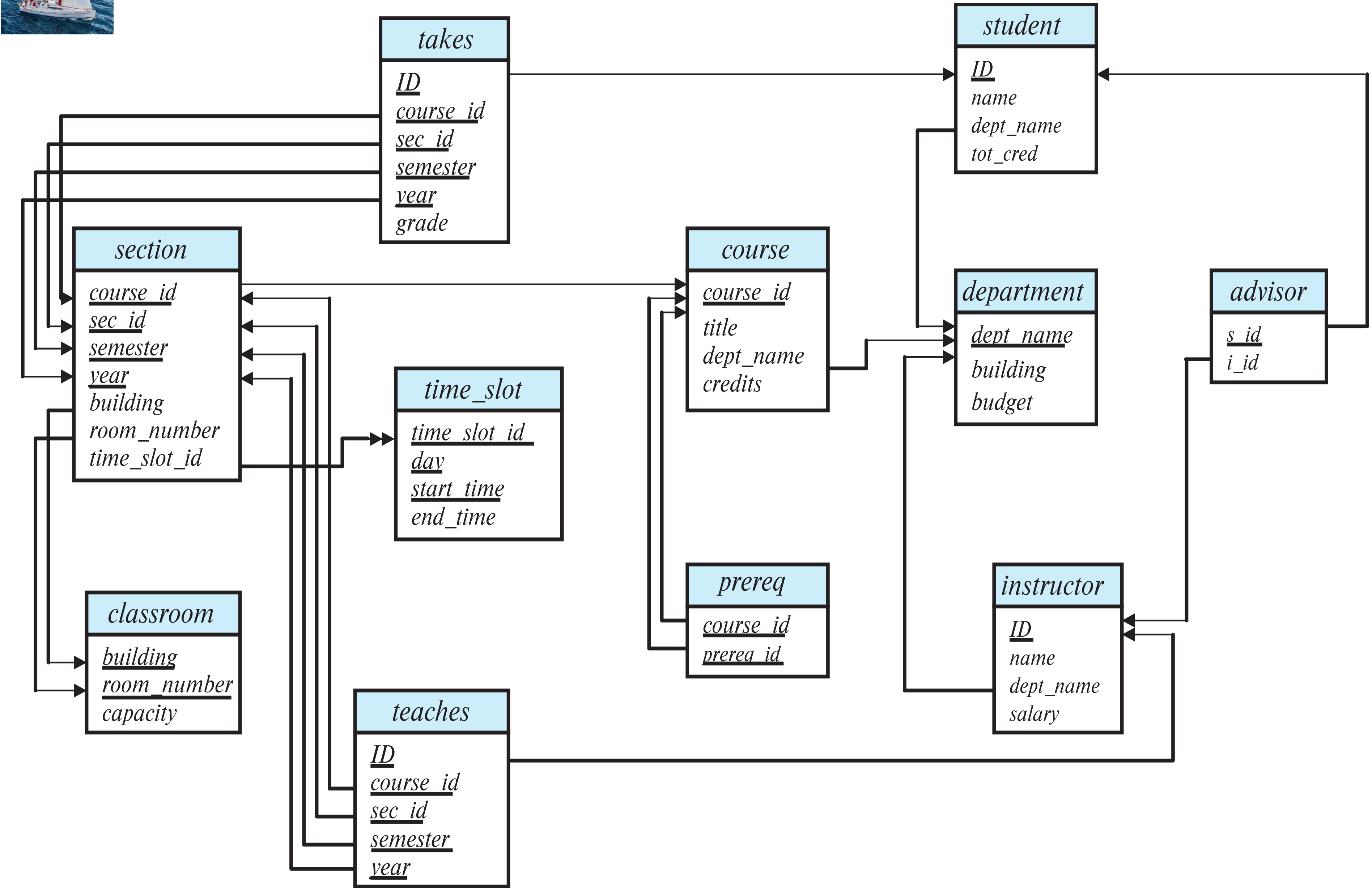
$$(5 < \textbf{all} \begin{array}{|c|}\hline 6\\\hline 10\\\hline\end{array}) = \text{true}$$

$$(5 = \textbf{all} \begin{array}{|c|}\hline 4\\\hline 5\\\hline\end{array}) = \text{false}$$

$$(5 \neq \textbf{all} \begin{array}{|c|}\hline 4\\\hline 6\\\hline\end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6)$$

($\neq$ **all**) $\equiv$ **not in**
However, (= **all**) $\not\equiv$ **in**

Schema Diagram for University Database



takes
ID
course_id
sec_id
semester
year
grade

student
ID
name
dept_name
tot_cred

section
course_id
sec_id
semester
year
building
room_number
time_slot_id

course
course_id
title
dept_name
credits

department
dept_name
building
budget

advisor
s_id
i_id

time_slot
time_slot_id
day
start_time
end_time

classroom
building
room_number
capacity

prereq
course_id
prereq_id

instructor
ID
name
dept_name
salary

teaches
ID
course_id
sec_id
semester
year

Use of “exists” Clause

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \varnothing$
- **not exists** $r \Leftrightarrow r = \varnothing$
- Yet another way of specifying the query “**Find all courses taught in both the Fall 2017 semester and in the Spring 2018 semester**”

```
select course_id
from section as S
where semester = 'Fall' and year = 2017 and
         exists (select *
                    from section as T
                    where semester = 'Spring' and year= 2018
                             and S.course_id = T.course_id);
```

- **Correlation name** – variable S in the outer query
- **Correlated subquery** – the inner query

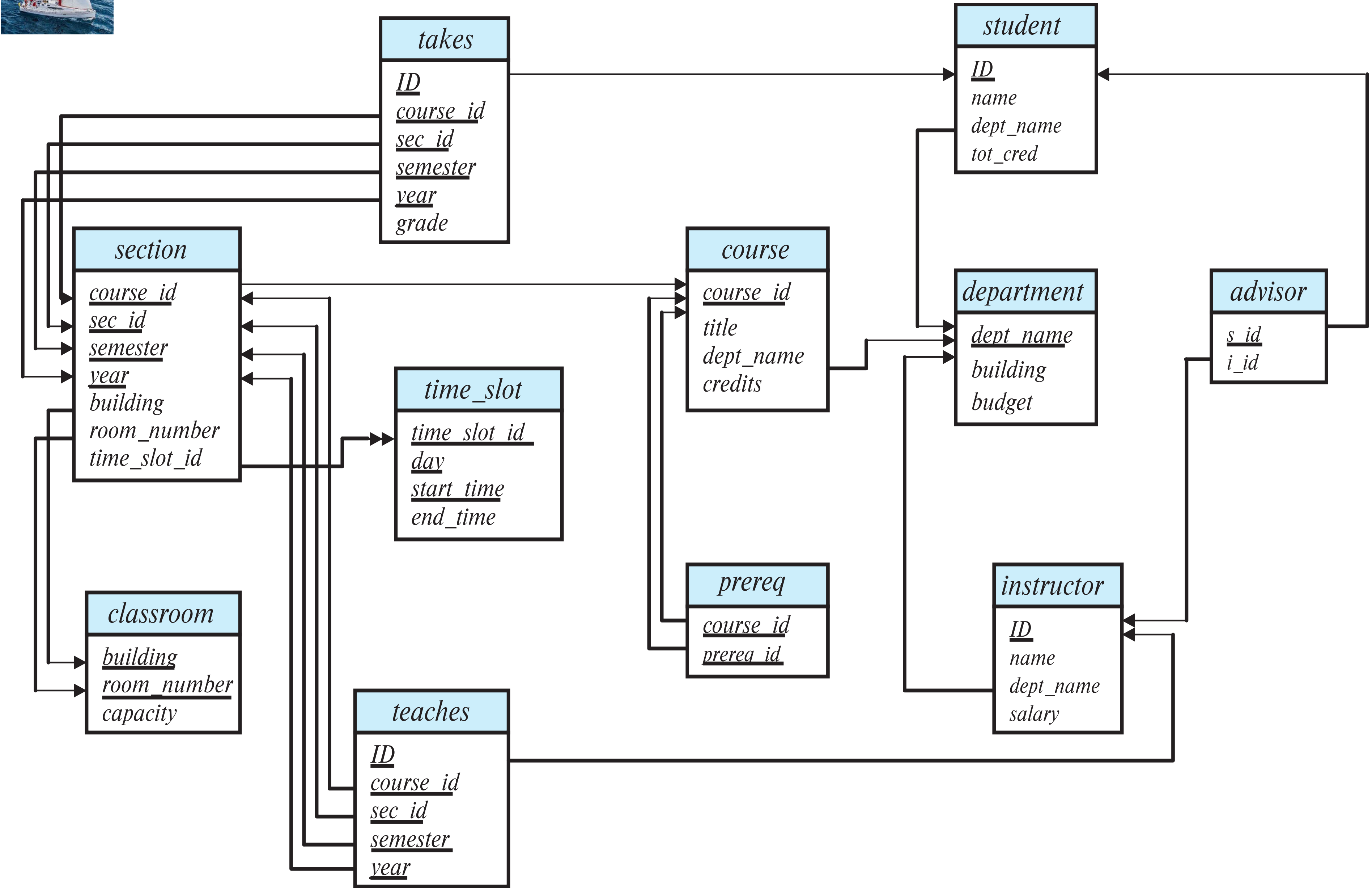
Database Management Systems (CSE-251)

Presented by

Md. Atiqul Islam Rizvi

Assistant Professor, Dept. of CSE, CUET

Schema Diagram for University Database



takes
- ID
- course_id
- sec_id
- semester
- year
- grade

student
- ID
- name
- dept_name
- tot_cred

section
- course_id
- sec_id
- semester
- year
- building
- room_number
- time_slot_id

course
- course_id
- title
- dept_name
- credits

department
- dept_name
- building
- budget

advisor
- s_id
- i_id

time_slot
- time_slot_id
- day
- start_time
- end_time

prereq
- course_id
- prereq_id

instructor
- ID
- name
- dept_name
- salary

classroom
- building
- room_number
- capacity

teaches
- ID
- course_id
- sec_id
- semester
- year

Use of "not exists" Clause

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

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

  - First nested query lists all courses offered in Biology
  - Second nested query lists all courses a particular student took
- Note that $X - Y = Ø \quad \Leftrightarrow \quad X \subseteq Y$
- Note: Cannot write this query using = all and its variants

Test for Absence of Duplicate Tuples

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
- It evaluates to “true” if a given subquery contains no duplicates .
- **Find all courses that were offered at most once in 2017**

```
select T.course_id
from course as T
where unique ( select R.course_id
               from section as R
               where T.course_id = R.course_id
                     and R.year = 2017);
```

Subqueries in the Form Clause

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

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

- Note that we do not need to use the **having** clause
- Another way to write above query

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

With Clause

- The **with** clause provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs.
- **Find all departments with the maximum budget**

```
with max_budget (value) as
        (select max(budget)
         from department)
select department.name
from department, max_budget
where department.budget = max_budget.value;
```

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

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

Scalar Subquery

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

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

- Runtime error if subquery returns more than one column

Modification - Insertion

- **Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of $18,000.**

```
insert into instructor
    select ID, name, dept_name, 18000
    from   student
    where  dept_name = 'Music' and total_cred > 144;
```

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

  Otherwise queries like

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

  would cause problem

Modification - Deletion

- **Delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building.**

```
delete from instructor
where dept name in (select dept name
                    from department
                    where building = 'Watson');
```

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

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

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

Modification - Updates

- Give a 5% salary raise to instructors whose salary is less than average

```
update instructor
set salary = salary * 1.05
where salary <  (select avg (salary)
                 from instructor);
```

Case Statement for Conditional Updates

- **Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%**
  - Write two **update** statements:

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

  - The order is important
  - Can be done better using the **case** statement
- Same query as before but with case statement

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

End of Chapter 3