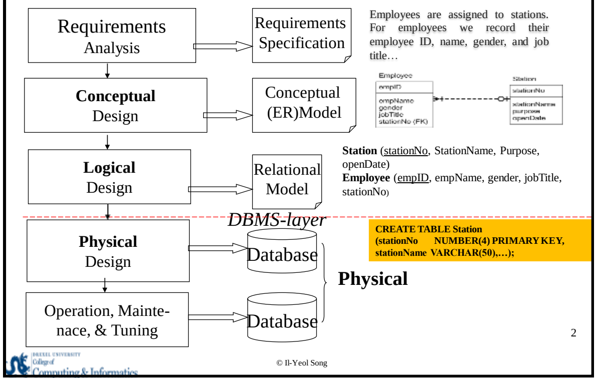
BIG DATA:
TECHNOLOGIES AND APPLICATIONS

3. SQL

Il-Yeol Song, Ph.D.
College of Computing & Informatics
Drexel University
Philadelphia, PA 19104

Drexel University College of Computing & Informatics

Typical Database Development Process



Requirements Analysis → Requirements Specification

Employees are assigned to stations. For employees we record their employee ID, name, gender, and job title…

Conceptual Design → Conceptual (ER)Model

Logical Design → Relational Model

Station (stationNo, StationName, Purpose, openDate)
Employee (empID, empName, gender, jobTitle, stationNo)

DBMS-layer

Physical Design → Database

CREATE TABLE Station (stationNo NUMBER(4) PRIMARY KEY, stationName VARCHAR(50),…);

Operation, Maintenace, & Tuning → Database

Physical

2

© Il-Yeol Song

Introduction to SQL

- Pronounced ‘see-quel’
- Standard database language for defining/managing databases
- American National Standards Institute (ANSI) and ISO (International Organization for Standard) standard
- Developed by IBM in 1974, System R project
- First commercial implementation by Oracle

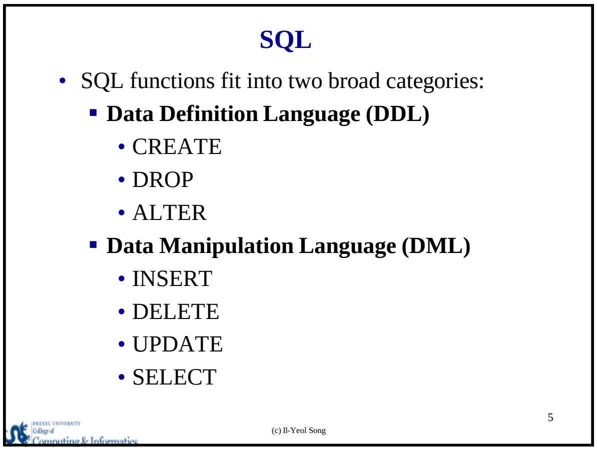
3

© Il-Yeol Song

Why SQL?

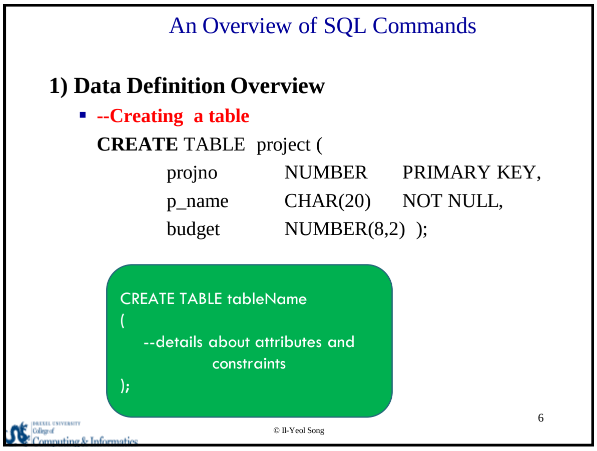
- SQL is relatively easy to learn:
  - Commands
  - It is non-procedural - you specify ***what*** information you require, rather than *how* to get it;
    - Called *Declarative* language
  - It is essentially a free-format.
    - The order of clauses are important
    - Within each clause, the order is not important
- ***The de facto* standard language for relational databases.**

© Il-Yeol Song



SQL

- SQL functions fit into two broad categories:
  - **Data Definition Language (DDL)**
    - CREATE
    - DROP
    - ALTER
  - **Data Manipulation Language (DML)**
    - INSERT
    - DELETE
    - UPDATE
    - SELECT



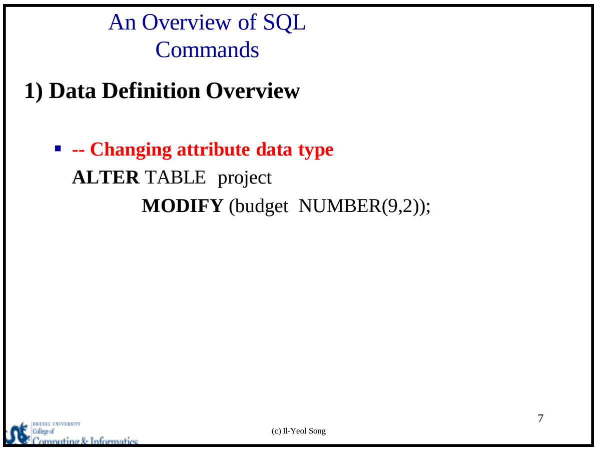
An Overview of SQL Commands

1) Data Definition Overview

- **--Creating a table**

```
CREATE TABLE  project (
        projno          NUMBER        PRIMARY KEY,
        p_name          CHAR(20)      NOT NULL,
        budget          NUMBER(8,2)  );
```

```
CREATE TABLE tableName
(
   --details about attributes and
          constraints
);
```

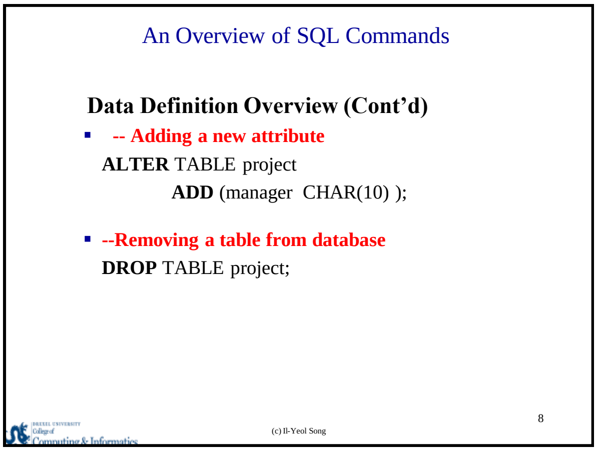


An Overview of SQL Commands

1) Data Definition Overview

- **-- Changing attribute data type**

```
ALTER TABLE  project
        MODIFY (budget  NUMBER(9,2));
```



An Overview of SQL Commands

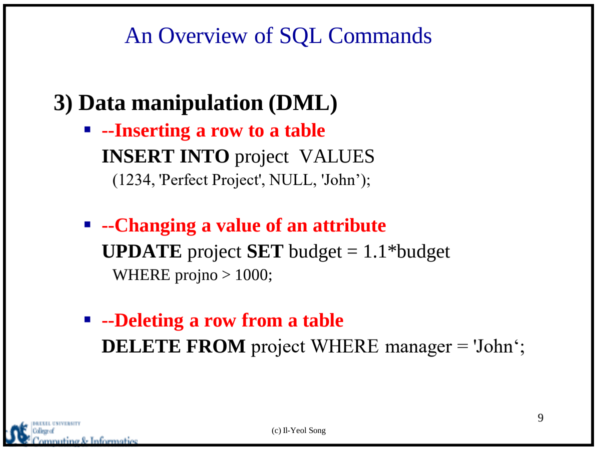
Data Definition Overview (Cont'd)

- **-- Adding a new attribute**

```
ALTER TABLE project
        ADD (manager  CHAR(10) );
```

- **--Removing a table from database**

```
DROP TABLE project;
```



An Overview of SQL Commands

3) Data manipulation (DML)

- **--Inserting a row to a table**

  INSERT INTO project VALUES
  (1234, 'Perfect Project', NULL, 'John');

- **--Changing a value of an attribute**

  UPDATE project **SET** budget = 1.1*budget
  WHERE projno > 1000;

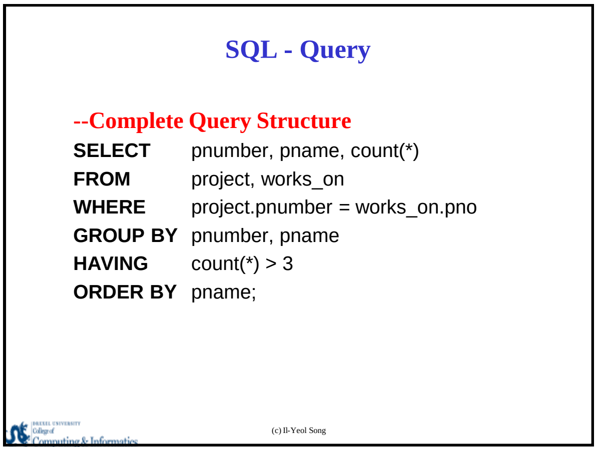
- **--Deleting a row from a table**

  DELETE FROM project WHERE manager = 'John';

DREXEL UNIVERSITY College of Computing & Informatics

(c) Il-Yeol Song

9



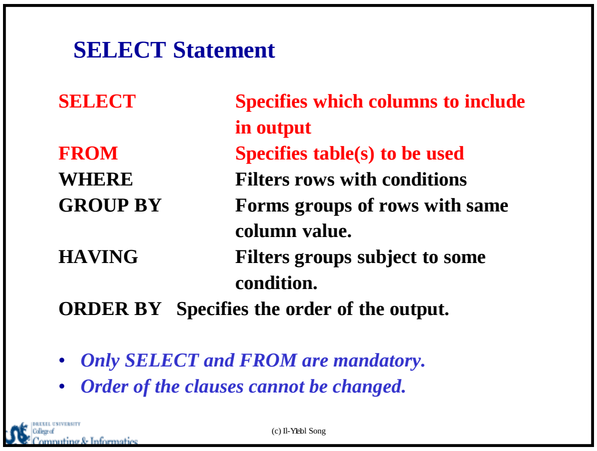
SQL - Query

--Complete Query Structure

```
SELECT      pnumber, pname, count(*)
FROM        project, works_on
WHERE       project.pnumber = works_on.pno
GROUP BY    pnumber, pname
HAVING      count(*) > 3
ORDER BY    pname;
```

DREXEL UNIVERSITY College of Computing & Informatics

(c) Il-Yeol Song



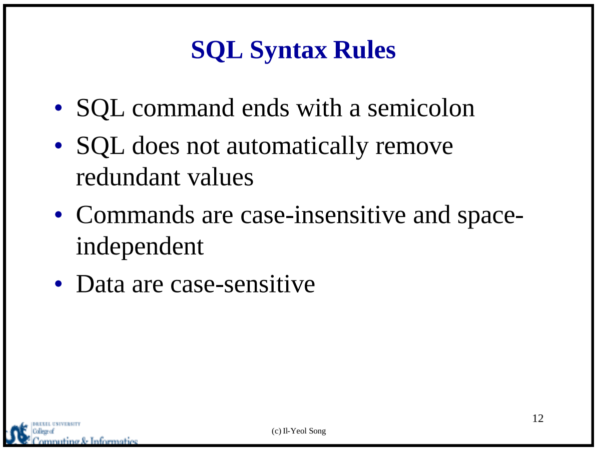
SELECT Statement

SELECT	**Specifies which columns to include in output**
FROM	**Specifies table(s) to be used**
WHERE	**Filters rows with conditions**
GROUP BY	**Forms groups of rows with same column value.**
HAVING	**Filters groups subject to some condition.**
ORDER BY	**Specifies the order of the output.**

- ***Only SELECT and FROM are mandatory.***
- ***Order of the clauses cannot be changed.***

DREXEL UNIVERSITY College of Computing & Informatics

(c) Il-Yeol Song



SQL Syntax Rules

- SQL command ends with a semicolon
- SQL does not automatically remove redundant values
- Commands are case-insensitive and space-independent
- Data are case-sensitive

DREXEL UNIVERSITY College of Computing & Informatics

(c) Il-Yeol Song

12

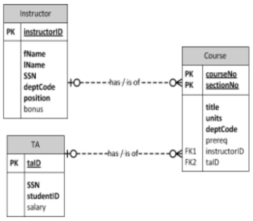
Comparison Operators

- Comparison operators available in SQL:

=	equals		
<>	is not equal to (ISO standard)		
!=	is not equal to (allowed in some dialects including Oracle)		
<	is less than	<=	is less than or equal to
>	is greater than	>=	is greater than or equal to

- More complex conditions can be generated using logical operators **AND**, **OR**, and **NOT**, with parentheses to show the order of evaluation

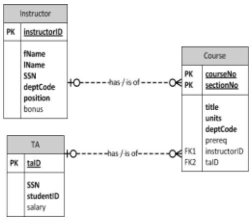
Example ERD and its meanings



Meaning of the ERD

- An Instructor teaches zero or many courses
- A course has zero or one Instructor.
- A TA is helping zero or one Course.
- A course has zero or one TA.

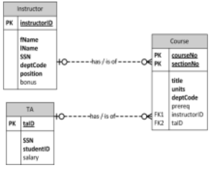
Example ERD and RDB Schema



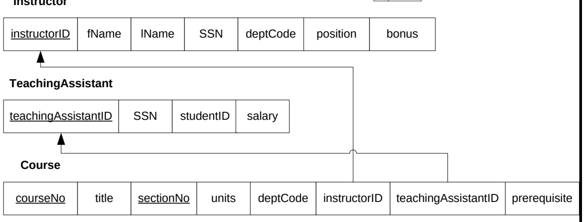
Relational Schema

Instructor (instructorID, fName, SSN, deptCode, position, bonus)
TA (taID, SSN, studentID, salary)
Course (courseNo, sectionNo, title, units, deptCode, prereq, instructorID, taID
FOREIGN KEY (instructorID) REFERENCES Instructor(instructorID)
FOREIGN KEY (taID) REFERENCES TA (taID)

Example: Course DB Relational Schema



Referential Integrity Diagram



Instructor

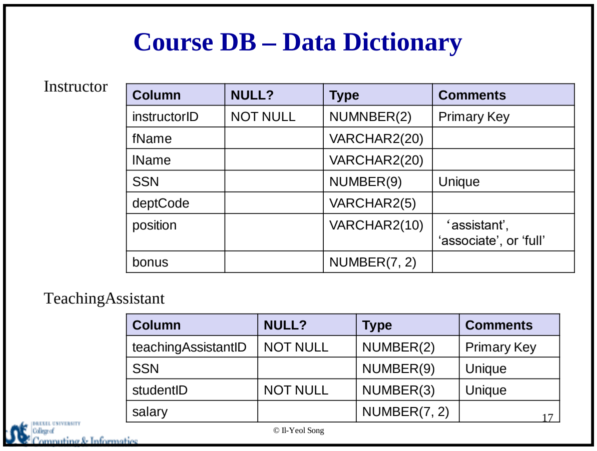
instructorID	fName	lName	SSN	deptCode	position	bonus

TeachingAssistant

teachingAssistantID	SSN	studentID	salary

Course

courseNo	title	sectionNo	units	deptCode	instructorID	teachingAssistantID	prerequisite



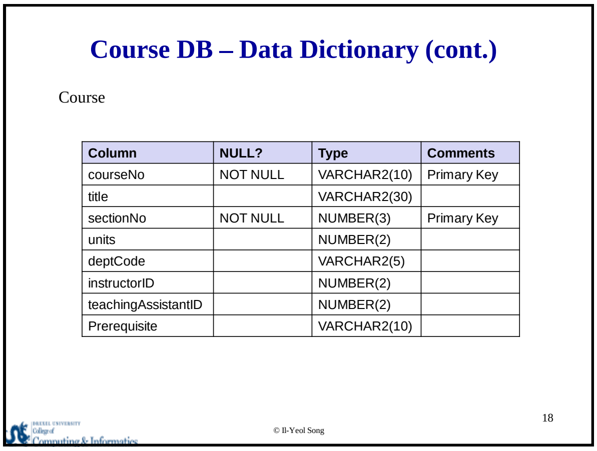
Course DB – Data Dictionary

Instructor

Column	NULL?	Type	Comments
instructorID	NOT NULL	NUMNBER(2)	Primary Key
fName		VARCHAR2(20)	
lName		VARCHAR2(20)	
SSN		NUMBER(9)	Unique
deptCode		VARCHAR2(5)	
position		VARCHAR2(10)	‘assistant’, ‘associate’, or ‘full’
bonus		NUMBER(7, 2)	

TeachingAssistant

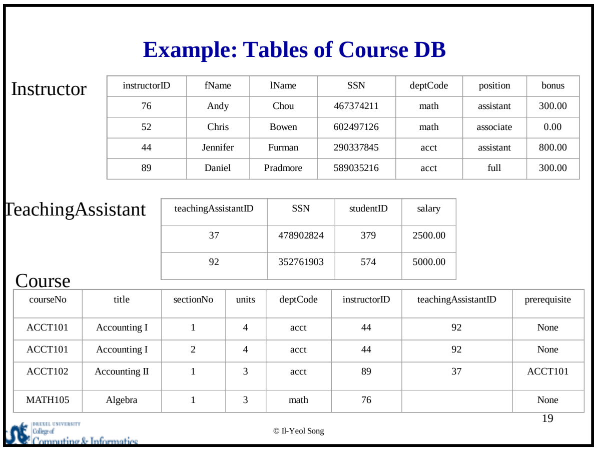
Column	NULL?	Type	Comments
teachingAssistantID	NOT NULL	NUMBER(2)	Primary Key
SSN		NUMBER(9)	Unique
studentID	NOT NULL	NUMBER(3)	Unique
salary		NUMBER(7, 2)	



Course DB – Data Dictionary (cont.)

Course

Column	NULL?	Type	Comments
courseNo	NOT NULL	VARCHAR2(10)	Primary Key
title		VARCHAR2(30)	
sectionNo	NOT NULL	NUMBER(3)	Primary Key
units		NUMBER(2)	
deptCode		VARCHAR2(5)	
instructorID		NUMBER(2)	
teachingAssistantID		NUMBER(2)	
Prerequisite		VARCHAR2(10)	



Example: Tables of Course DB

Instructor

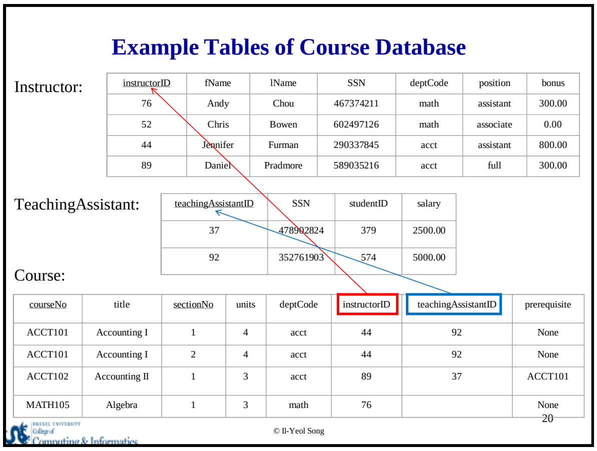
instructorID	fName	lName	SSN	deptCode	position	bonus
76	Andy	Chou	467374211	math	assistant	300.00
52	Chris	Bowen	602497126	math	associate	0.00
44	Jennifer	Furman	290337845	acct	assistant	800.00
89	Daniel	Pradmore	589035216	acct	full	300.00

TeachingAssistant

teachingAssistantID	SSN	studentID	salary
37	478902824	379	2500.00
92	352761903	574	5000.00

Course

courseNo	title	sectionNo	units	deptCode	instructorID	teachingAssistantID	prerequisite
ACCT101	Accounting I	1	4	acct	44	92	None
ACCT101	Accounting I	2	4	acct	44	92	None
ACCT102	Accounting II	1	3	acct	89	37	ACCT101
MATH105	Algebra	1	3	math	76		None



Example Tables of Course Database

Instructor:

instructorID	fName	lName	SSN	deptCode	position	bonus
76	Andy	Chou	467374211	math	assistant	300.00
52	Chris	Bowen	602497126	math	associate	0.00
44	Jennifer	Furman	290337845	acct	assistant	800.00
89	Daniel	Pradmore	589035216	acct	full	300.00

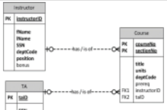
TeachingAssistant:

teachingAssistantID	SSN	studentID	salary
37	478902824	379	2500.00
92	352761903	574	5000.00

Course:

courseNo	title	sectionNo	units	deptCode	instructorID	teachingAssistantID	prerequisite
ACCT101	Accounting I	1	4	acct	44	92	None
ACCT101	Accounting I	2	4	acct	44	92	None
ACCT102	Accounting II	1	3	acct	89	37	ACCT101
MATH105	Algebra	1	3	math	76		None

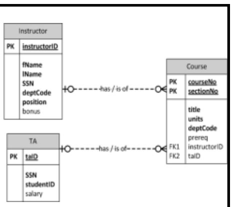
DDL for Course Database



```
CREATE TABLE TeachingAssistant
    ( teachingAssistantID  NUMBER(2) NOT NULL,
     SSN                              NUMBER(9) UNIQUE,
     studentID                NUMBER(3) NOT NULL UNIQUE,
     salary                   NUMBER (7, 2) CHECK (salary > 100.00),
    CONSTRAINT TeachingAssistant_PK  PRIMARY KEY (teachingAssistantID)
    );

CREATE TABLE Instructor
   ( instructorID  NUMBER(2),
   fName           VARCHAR2(20),
   lName           VARCHAR2(20),
   SSN             CHAR(9) UNIQUE NOT NULL,
   deptCode        VARCHAR2(5),
   position       VARCHAR2(10) CHECK (position IN ('assistant', 'associate', 'full')),
   bonus           NUMBER (7, 2),
   CONSTRAINT Instructor_PK PRIMARY KEY (instructorID) );
```

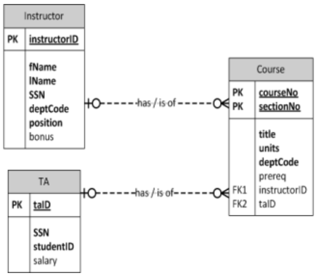
DDL for Course Database



```
CREATE TABLE Course
   ( courseNo      VARCHAR2(10) NOT NULL,
   title           VARCHAR2(30),
   sectionNo       NUMBER(3) NOT NULL,
   units           NUMBER(2),
   deptCode        VARCHAR2(5),
   instructorID    NUMBER(2),
   teachingAssistantID       NUMBER(2),
   prerequisite    VARCHAR2(10),
    CONSTRAINT Course_PK PRIMARY KEY (courseNo,
   sectionNo),
   CONSTRAINT Course_FK1 FOREIGN KEY (instructorID)
   REFERENCES Instructor(instructorID),
   CONSTRAINT Course_FK2 FOREIGN KEY (teachingAssistantID)
   REFERENCES TeachingAssistant(teachingAssistantID)
   );
```

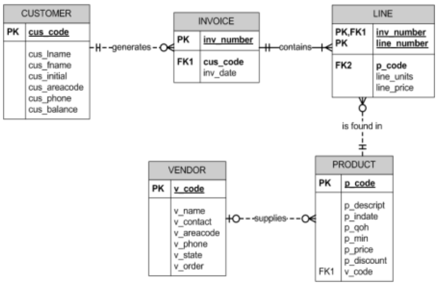
Which table to create first? ?

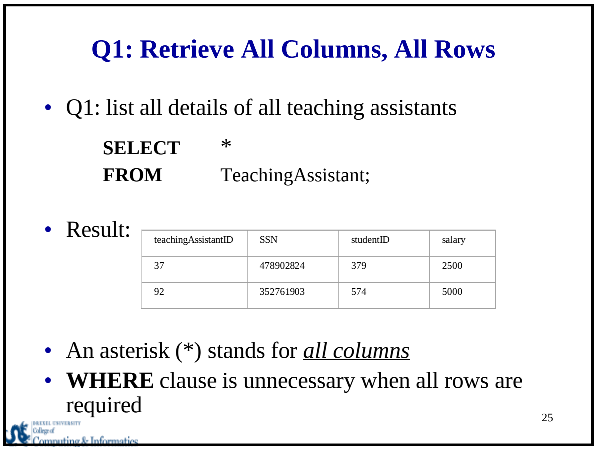
-



Which table to create first? ?

-





Q1: Retrieve All Columns, All Rows

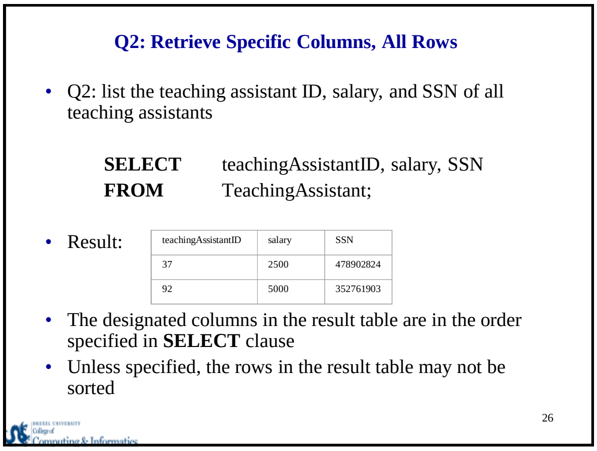
- Q1: list all details of all teaching assistants

```
SELECT    *
FROM      TeachingAssistant;
```

- Result:

teachingAssistantID	SSN	studentID	salary
37	478902824	379	2500
92	352761903	574	5000

- An asterisk (*) stands for *all columns*
- **WHERE** clause is unnecessary when all rows are required



Q2: Retrieve Specific Columns, All Rows

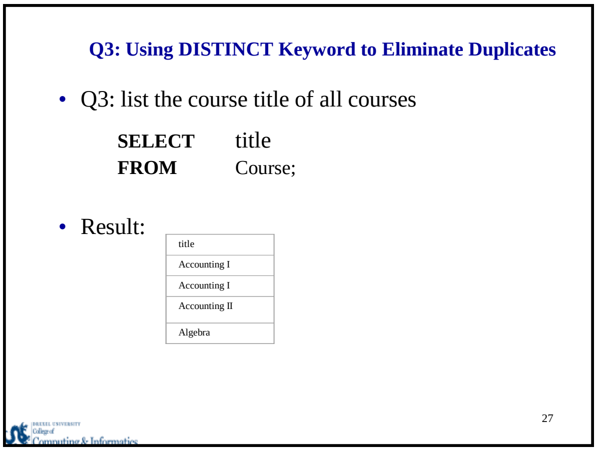
- Q2: list the teaching assistant ID, salary, and SSN of all teaching assistants

```
SELECT    teachingAssistantID, salary, SSN
FROM      TeachingAssistant;
```

- Result:

teachingAssistantID	salary	SSN
37	2500	478902824
92	5000	352761903

- The designated columns in the result table are in the order specified in **SELECT** clause
- Unless specified, the rows in the result table may not be sorted



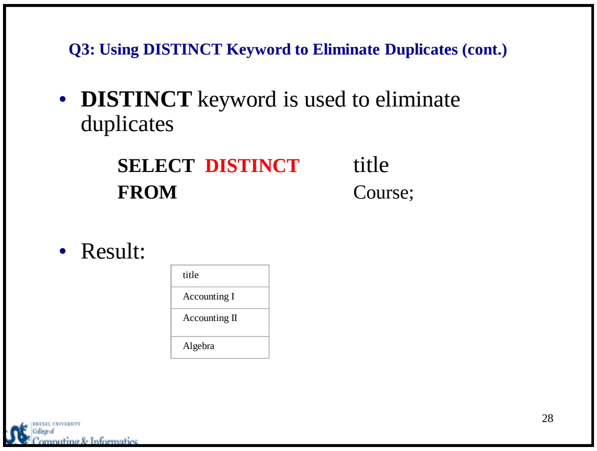
Q3: Using DISTINCT Keyword to Eliminate Duplicates

- Q3: list the course title of all courses

```
SELECT    title
FROM      Course;
```

- Result:

title
Accounting I
Accounting I
Accounting II
Algebra



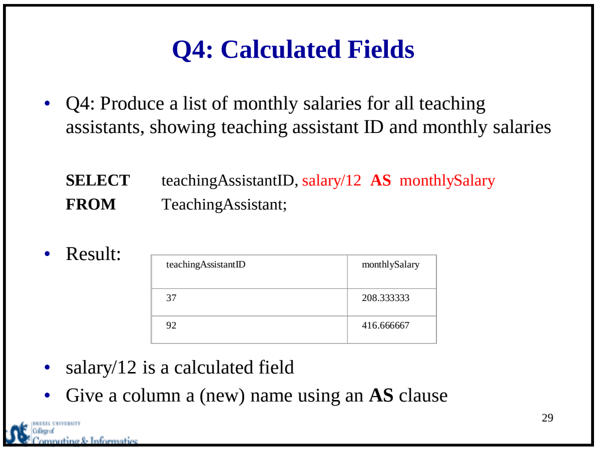
Q3: Using DISTINCT Keyword to Eliminate Duplicates (cont.)

- **DISTINCT** keyword is used to eliminate duplicates

```
SELECT  DISTINCT      title
FROM                  Course;
```

- Result:

title
Accounting I
Accounting II
Algebra



Q4: Calculated Fields

- Q4: Produce a list of monthly salaries for all teaching assistants, showing teaching assistant ID and monthly salaries

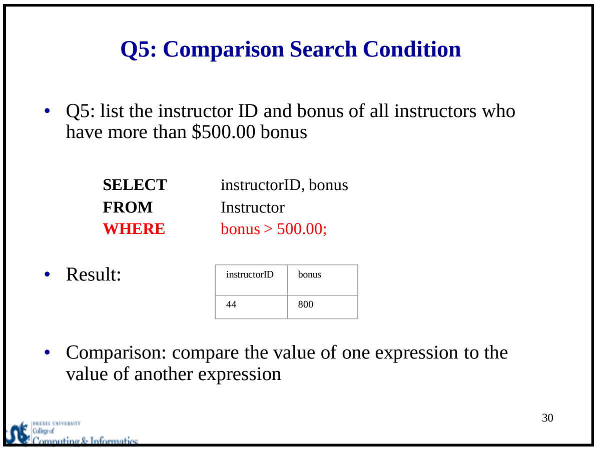
```
SELECT    teachingAssistantID, salary/12  AS  monthlySalary
FROM      TeachingAssistant;
```

- Result:

teachingAssistantID	monthlySalary
37	208.333333
92	416.666667

- salary/12 is a calculated field
- Give a column a (new) name using an **AS** clause

29



Q5: Comparison Search Condition

- Q5: list the instructor ID and bonus of all instructors who have more than $500.00 bonus

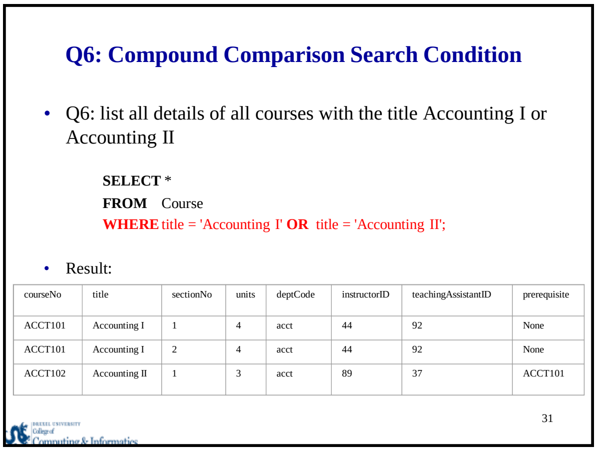
```
SELECT      instructorID, bonus
FROM        Instructor
WHERE       bonus > 500.00;
```

- Result:

instructorID	bonus
44	800

- Comparison: compare the value of one expression to the value of another expression

30



Q6: Compound Comparison Search Condition

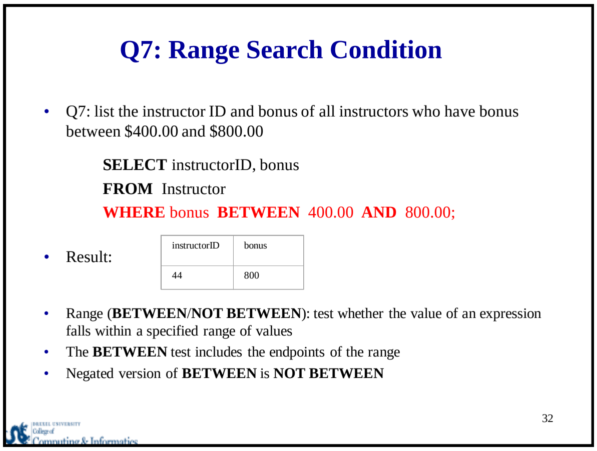
- Q6: list all details of all courses with the title Accounting I or Accounting II

```
SELECT *
FROM   Course
WHERE title = 'Accounting  I' OR  title = 'Accounting  II';
```

- Result:

courseNo	title	sectionNo	units	deptCode	instructorID	teachingAssistantID	prerequisite
ACCT101	Accounting I	1	4	acct	44	92	None
ACCT101	Accounting I	2	4	acct	44	92	None
ACCT102	Accounting II	1	3	acct	89	37	ACCT101

31



Q7: Range Search Condition

- Q7: list the instructor ID and bonus of all instructors who have bonus between $400.00 and $800.00

```
SELECT instructorID, bonus
FROM  Instructor
WHERE bonus  BETWEEN  400.00  AND  800.00;
```

- Result:

instructorID	bonus
44	800

- Range (**BETWEEN**/**NOT BETWEEN**): test whether the value of an expression falls within a specified range of values
- The **BETWEEN** test includes the endpoints of the range
- Negated version of **BETWEEN** is **NOT BETWEEN**

32

Q7: Range Search Condition (Cont.)

- **BETWEEN** test can be equally expressed using two comparison tests

- We can use

```
SELECT  instructorID, bonus
FROM  Instructor
WHERE bonus>=400.00  AND  bonus<=800.00;
```

to get the same result table in the previous slide

Q8: Set Membership Search Condition

- Q8: list the instructor ID and position of instructors who are associate professors or full professors

```
SELECT        instructorID, position
FROM          Instructor
WHERE         position IN (‘associate’, ‘full’);
```

- Result:

instructorID	position
52	associate
89	full

- Set membership (**IN**/**NOT IN**): test whether the value of an expression equals one of a set of values
- Negated version (**NOT IN**) can be used to check for data values that are not in a specific list of values

Q8: Set Membership Search Condition (cont.)

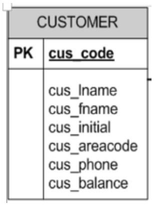
- **IN**/**NOT IN** test can be equally expressed using multiple comparison tests
- We can use

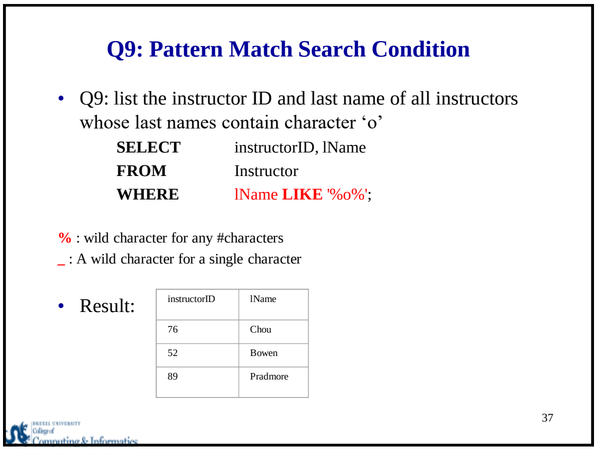
```
SELECT        instructorID, position
FROM          Instructor
WHERE         position = 'associate' OR position = 'full';
```

to get the same result table in the previous slide

Exercises:

Find customer first name, last name, and phone number whose area code is 215 and and whose balances are greater than 1000.





Q9: Pattern Match Search Condition

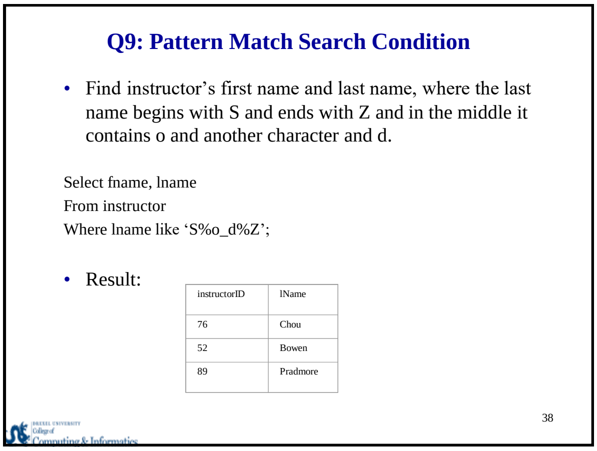
- Q9: list the instructor ID and last name of all instructors whose last names contain character ‘o’

```
SELECT      instructorID, lName
FROM        Instructor
WHERE       lName LIKE '%o%';
```

% : wild character for any #characters
_ : A wild character for a single character

- Result:

instructorID	lName
76	Chou
52	Bowen
89	Pradmore



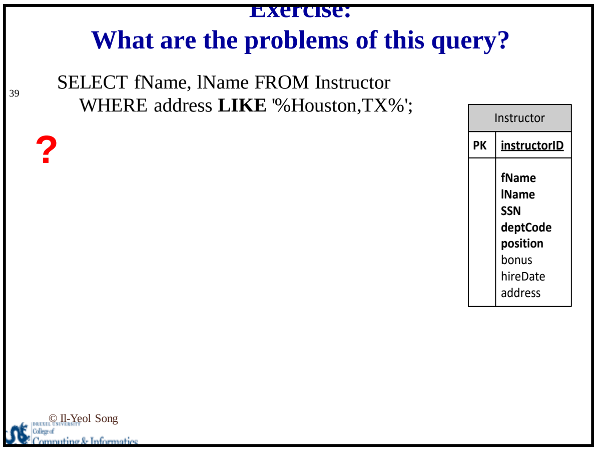
Q9: Pattern Match Search Condition

- Find instructor’s first name and last name, where the last name begins with S and ends with Z and in the middle it contains o and another character and d.

```
Select fname, lname
From instructor
Where lname like ‘S%o_d%Z’;
```

- Result:

instructorID	lName
76	Chou
52	Bowen
89	Pradmore

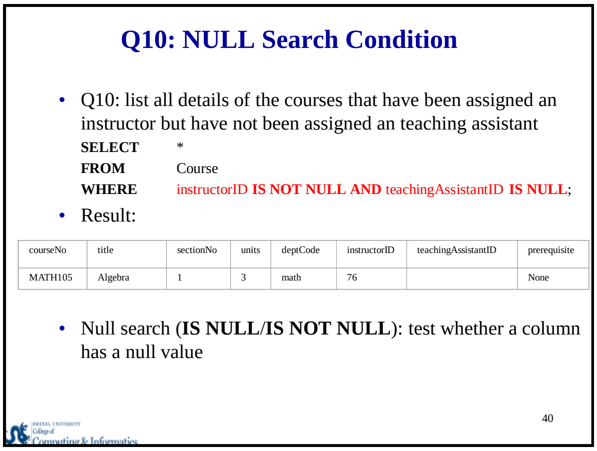


Exercise:
What are the problems of this query?

```
SELECT fName, lName FROM Instructor
   WHERE address LIKE '%Houston,TX%';
```

?

Instructor	
PK	instructorID
	fName **lName** **SSN** **deptCode** **position** bonus hireDate address



Q10: NULL Search Condition

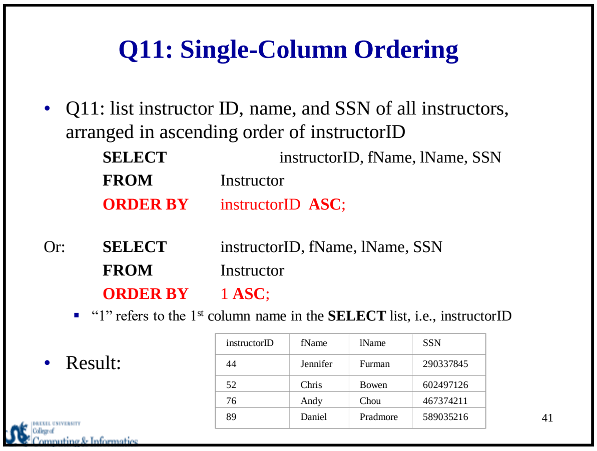
- Q10: list all details of the courses that have been assigned an instructor but have not been assigned an teaching assistant

```
SELECT      *
FROM        Course
WHERE       instructorID IS NOT NULL AND teachingAssistantID IS NULL;
```

- Result:

courseNo	title	sectionNo	units	deptCode	instructorID	teachingAssistantID	prerequisite
MATH105	Algebra	1	3	math	76		None

- Null search (**IS NULL**/**IS NOT NULL**): test whether a column has a null value



Q11: Single-Column Ordering

- Q11: list instructor ID, name, and SSN of all instructors, arranged in ascending order of instructorID

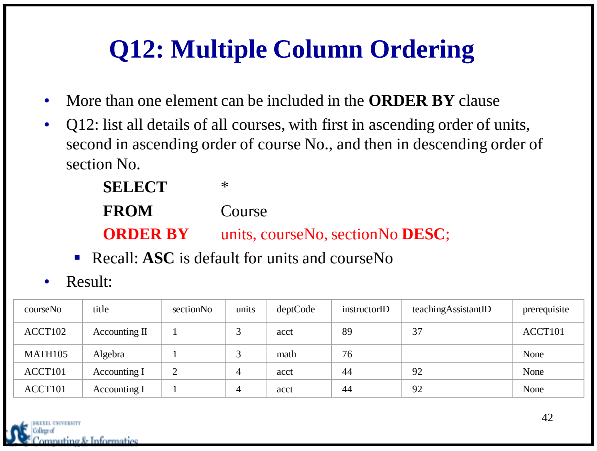
  SELECT instructorID, fName, lName, SSN
  FROM Instructor
  ORDER BY instructorID **ASC;**

Or: **SELECT** instructorID, fName, lName, SSN
FROM Instructor
ORDER BY 1 **ASC;**

- "1" refers to the 1st column name in the **SELECT** list, i.e., instructorID

- Result:

instructorID	fName	lName	SSN
44	Jennifer	Furman	290337845
52	Chris	Bowen	602497126
76	Andy	Chou	467374211
89	Daniel	Pradmore	589035216



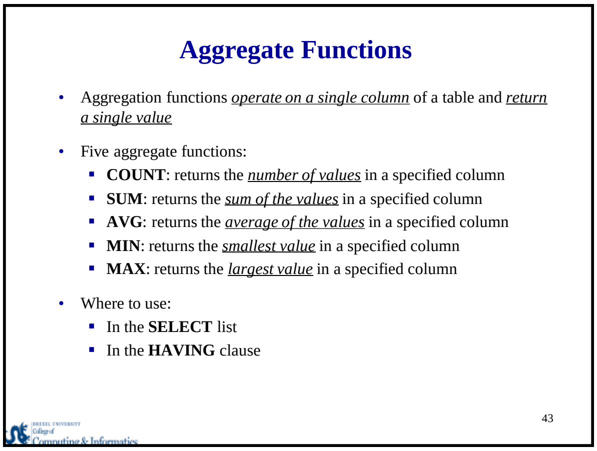
Q12: Multiple Column Ordering

- More than one element can be included in the **ORDER BY** clause
- Q12: list all details of all courses, with first in ascending order of units, second in ascending order of course No., and then in descending order of section No.

  SELECT *
  FROM Course
  ORDER BY units, courseNo, sectionNo **DESC;**

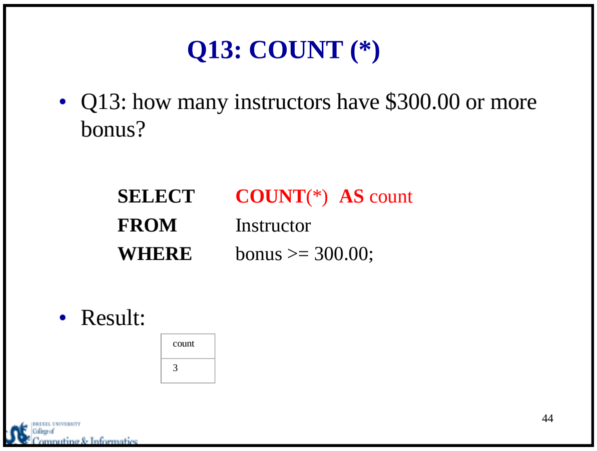
  - Recall: **ASC** is default for units and courseNo
- Result:

courseNo	title	sectionNo	units	deptCode	instructorID	teachingAssistantID	prerequisite
ACCT102	Accounting II	1	3	acct	89	37	ACCT101
MATH105	Algebra	1	3	math	76		None
ACCT101	Accounting I	2	4	acct	44	92	None
ACCT101	Accounting I	1	4	acct	44	92	None



Aggregate Functions

- Aggregation functions *operate on a single column* of a table and *return a single value*
- Five aggregate functions:
  - **COUNT**: returns the *number of values* in a specified column
  - **SUM**: returns the *sum of the values* in a specified column
  - **AVG**: returns the *average of the values* in a specified column
  - **MIN**: returns the *smallest value* in a specified column
  - **MAX**: returns the *largest value* in a specified column
- Where to use:
  - In the **SELECT** list
  - In the **HAVING** clause



Q13: COUNT (*)

- Q13: how many instructors have $300.00 or more bonus?

  SELECT COUNT(*) **AS** count
  FROM Instructor
  WHERE bonus >= 300.00;

- Result:

count
3

Q14: COUNT (DISTINCT)

- Q14: how many different course titles are there?

```
SELECT      COUNT(DISTINCT title)  AS  count
FROM        Course;
```

- Result:

count
3

- In this example, **DISTINCT** keyword is used to eliminate duplicate course titles

Q15: COUNT And SUM

- Q15: find the total number of assistant professors and the sum of their bonus

```
SELECT      COUNT(instructorID)  AS  count, SUM(bonus)  AS  sum
FROM        Instructor
WHERE       position = 'assistant';
```

- Result:

count	sum
2	1100

- In this example, the processing order of clauses: **FROM → WHERE → SELECT**

Q16: MIN, MAX, AVG

- Q16: find the minimum, maximum, and average bonus by instructors

```
SELECT     MIN(bonus)  AS  min,
           MAX(bonus)  AS  max, AVG(bonus)  AS avg
FROM       Instructor;
```

- Result:

min	max	avg
0	800	350

- In this example, all instructors are considered and therefore **WHERE** clause is not needed

Grouping Results Using GROUP BY Clause

- **GROUP BY** clause ***Produces a single summary row for each group*** (e.g., ‘acct’ group or ‘math’ group)
- GROUP BY always comes **after** **WHERE** clause
- When **GROUP BY** clause is used
  - The **SELECT** clause must include a combination of column names and aggregate functions.
  - ***GROUP BY*** *must include* ***all non-aggregate function column names*** *in the* ***SELECT*** *list.*

    Select, x, y, z, AVG(P)
    From r
    Group by (   );

Q17: GROUP BY

- Q17: find the total number of instructors in each department and the sum of their bonus, respectively

```
SELECT      deptCode, COUNT(instructorID)  AS count,
            SUM(bonus) AS sum
FROM        Instructor
GROUP BY    deptCode
ORDER BY    deptCode;
```

- Result:

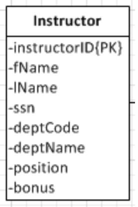
deptCode	count	sum
acct	2	1100
math	2	300

- In this example, the processing order of clauses: **FROM →GROUP BY →SELECT → ORDER BY**

- Recall: all column names in the **SELECT** list must appear in the **GROUP BY** clause unless the name is used only in an aggregate function

What's wrong with the following query?

Suppose we have the Instructor table as follows.

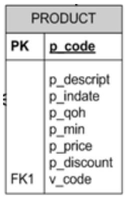


•Find the deptCode and deptName, and the total number of instructors in each department and the sum of their bonus, respectively

```
SELECT      deptCode, deptName,
            COUNT(instructorID)  AS count,
            SUM(bonus) AS sum
FROM        Instructor
GROUP BY    deptCode
ORDER BY    deptCode;
```

Exercise

Find the average price of all the products per vendor. Order the output by vender code and product description

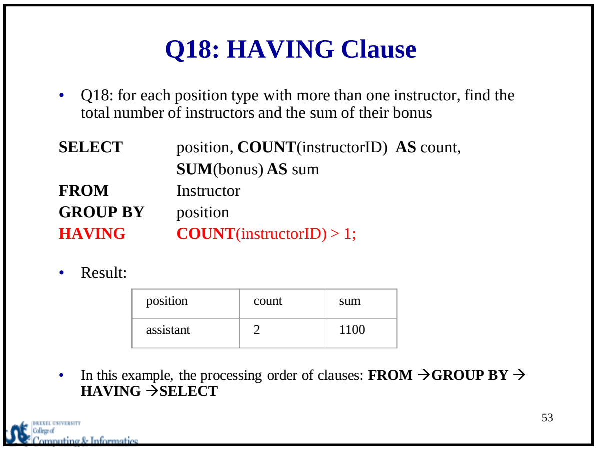


Restricting Groupings Using HAVING Clause

- **HAVING** clause is used with **GROUP BY** clause to restrict the groups that appear in the final result table

- **HAVING** clause vs. **WHERE** clause:
  - Serve different purposes:
    - **WHERE** clause filters *individual rows* going into the final result table
    - **HAVING** clause filters *groups* going into the final result table

- **HAVING** clause cannot be used without **GROUP BY** clause in a **SELECT** statement



Q18: HAVING Clause

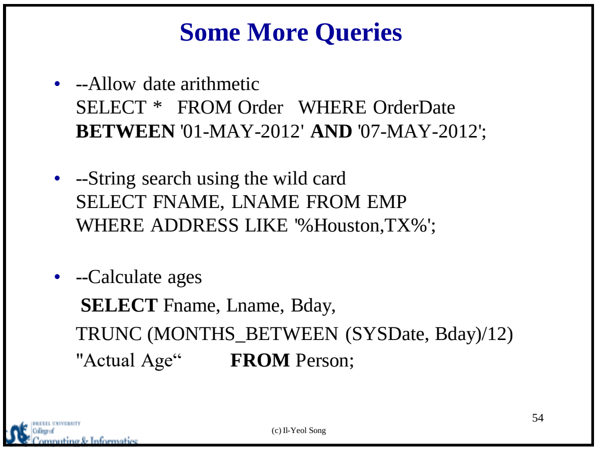
- Q18: for each position type with more than one instructor, find the total number of instructors and the sum of their bonus

```
SELECT      position, COUNT(instructorID)  AS count,
            SUM(bonus) AS sum
FROM        Instructor
GROUP BY    position
HAVING      COUNT(instructorID) > 1;
```

- Result:

position	count	sum
assistant	2	1100

- In this example, the processing order of clauses: **FROM →GROUP BY → HAVING →SELECT**



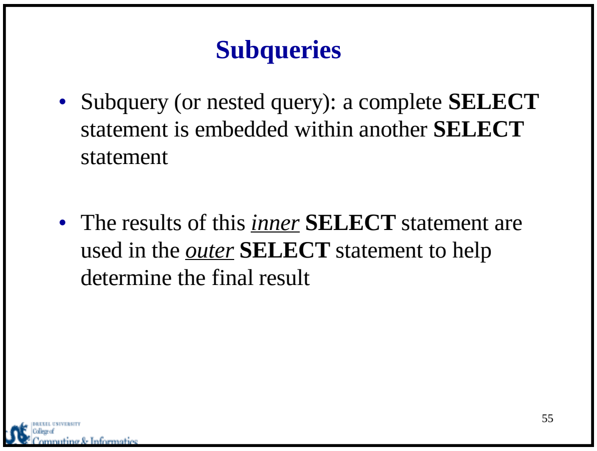
Some More Queries

- --Allow date arithmetic
  SELECT * FROM Order WHERE OrderDate **BETWEEN** '01-MAY-2012' **AND** '07-MAY-2012';

- --String search using the wild card
  SELECT FNAME, LNAME FROM EMP
  WHERE ADDRESS LIKE '%Houston,TX%';

- --Calculate ages
  SELECT Fname, Lname, Bday,
  TRUNC (MONTHS_BETWEEN (SYSDate, Bday)/12)
  "Actual Age“ **FROM** Person;

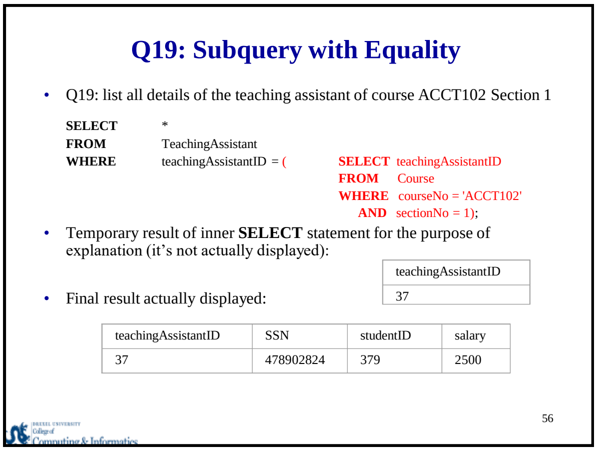
(c) Il-Yeol Song



Subqueries

- Subquery (or nested query): a complete **SELECT** statement is embedded within another **SELECT** statement

- The results of this *inner* **SELECT** statement are used in the *outer* **SELECT** statement to help determine the final result



Q19: Subquery with Equality

- Q19: list all details of the teaching assistant of course ACCT102 Section 1

```
SELECT      *
FROM        TeachingAssistant
WHERE       teachingAssistantID = (   SELECT  teachingAssistantID
                                      FROM    Course
                                      WHERE   courseNo = 'ACCT102'
                                         AND  sectionNo = 1);
```

- Temporary result of inner **SELECT** statement for the purpose of explanation (it’s not actually displayed):

teachingAssistantID
37

- Final result actually displayed:

teachingAssistantID	SSN	studentID	salary
37	478902824	379	2500

Q20: Subquery With An Aggregate Function

- Q20: list the instructor IDs of instructors whose bonus are greater than the average bonus, and show their bonus

```
SELECT  instructorID, bonus
FROM    Instructor
WHERE   bonus > (SELECT  AVG(bonus)  FROM  Instructor);
```

- Result:

instructorID	bonus
44	800

- Recall: aggregation functions cannot be used in **WHERE** clause, thus we cannot write "**WHERE**   bonus > **AVG**(bonus)"
- When a subquery is one of the two operands involved in a comparison, the subquery must appear on the right-hand side of the comparison

Q21: Nested Subqueries: Use of IN

- Q21: list all details of the instructors who teach courses that have teaching assistants with salary of at least $2500.00

```
SELECT   *
FROM     Instructor
WHERE    instructorID  IN  (SELECT   DISTINCT instructorID
                            FROM     Course
                            WHERE    teachingAssistantID  IN  (SELECT  teachingAssistantID
                                                               FROM    TeachingAssistant
                                                               WHERE   salary >= 2500.00));
```

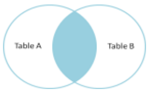
- Result:

instructorID	fName	lName	SSN	deptCode	position	bonus
44	Jennifer	Furman	290337845	acct	assistant	800
89	Daniel	Pradmore	589035216	acct	full	300

- For nested subqueries, work from the innermost query outwards
- When more than one value is found from subquery, use keyword **IN** rather than = to evaluate the equality

JOIN

- JOIN is the relational operation that combines data stored in two tables, creating a new result table



```
SELECT    <attributes_to_be_displayed>
FROM      A, B
WHERE     A.key = B.key;
```

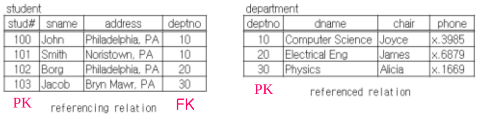
Two tables to be joined

JOIN Conditions in the form of A.PK = B.FK

JOIN

- JOIN follows **the PK-FK chain** of the two related tables.

```
SELECT *
FROM student, department
WHERE student.deptNO = department.deptNO;
```



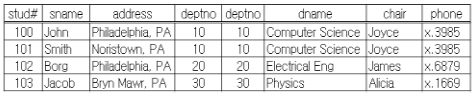
student

stud#	sname	address	deptno
100	John	Philadelphia, PA	10
101	Smith	Noristown, PA	10
102	Borg	Philadelphia, PA	20
103	Jacob	Bryn Mawr, PA	30

department

deptno	dname	chair	phone
10	Computer Science	Joyce	x.3985
20	Electrical Eng	James	x.6879
30	Physics	Alicia	x.1669

PK
referencing relation
FK
PK
referenced relation



stud#	sname	address	deptno	deptno	dname	chair	phone
100	John	Philadelphia, PA	10	10	Computer Science	Joyce	x.3985
101	Smith	Noristown, PA	10	10	Computer Science	Joyce	x.3985
102	Borg	Philadelphia, PA	20	20	Electrical Eng	James	x.6879
103	Jacob	Bryn Mawr, PA	30	30	Physics	Alicia	x.1669

How to Write a Join?

- How to combine information from two tables by join?
  - Form pairs of *related rows* from two tables where the value of PK of a table A matches with a value of FK of another table.

    A.PK = B.FK

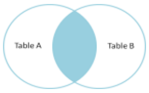
  - To write a join:

  (a) List more than one table name in the **FROM** clause, using a comma as a separator, and
  - e.g., **FROM** student, department

  (b) Typically use a **WHERE** clause to specify the join condition(s)
  - e.g., **WHERE student.deptNO = department.deptNO**

Visualization of JOIN

join produces only the set of records that match in both Table A and Table B.



Q24: For each department, find manager's SSN and name.

SELECT Dnumber, EmpSSN, Name
FROM Employee, Department
WHERE **Employee.EmpSSN= Department.Mgrssn;**

TABLE A (EMPLOYEE)

EMPSSN	NAME
888665555	Bill
333445555	Mary
123456789	John
999887777	Kate

TABLE B (DEPARTMENT)

DNUMBER	MGRSSN
5	333445555
4	999887777
1	888665555

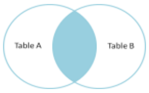
- **Steps of performing JOIN**
- Compare each EmpSSN with each of MgrSSN.
- Select only those rows their values match

DNUMBER	EMPSSN	NAME
5	333445555	Mary
4	999887777	Kate
1	888665555	Bill

© Il-Yeol Song

Visualization of JOIN

How JOIN is executed?



SELECT Dnumber, EmpSSN, Name
FROM Employee, Department
WHERE Employee.EmpSSN= Department.Mgrssn;

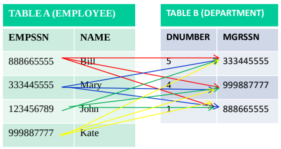


TABLE A (EMPLOYEE)

EMPSSN	NAME
888665555	Bill
333445555	Mary
123456789	John
999887777	Kate

TABLE B (DEPARTMENT)

DNUMBER	MGRSSN
5	333445555
4	999887777
1	888665555

Processing steps:
- For each row of Employee table, compare **each** EmpSSN with **each** value of MgrSSN.
- Select only rows their values match

Question: How many total number of comparisons do we need for this data sets?

DNUMBER	EMPSSN	NAME
5	333445555	Mary
4	999887777	Kate
1	888665555	Bill

Visualization of JOIN

SELECT Dnumber, EmpSSN, Name
FROM Employee, Department
WHERE Employee.EmpSSN= Department.Mgrssn;

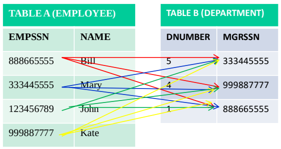


TABLE A (EMPLOYEE)

EMPSSN	NAME
888665555	Bill
333445555	Mary
123456789	John
999887777	Kate

TABLE B (DEPARTMENT)

DNUMBER	MGRSSN
5	333445555
4	999887777
1	888665555

JOIN algorithm:

```
FOR each row in Employee table
    LOOP
        IF EmpSSN = MGRSSN
            THEN select the row
```

DNUMBER	EMPSSN	NAME
5	333445555	Mary
4	999887777	Kate
1	888665555	Bill

Question: How many total number of comparisons do we need for this data sets?

Answer: Since EMPLOYEE table has 4 rows and Department table has 3 rows, the total number of comparisons we need for the above join is 4*3= 12 comparisons

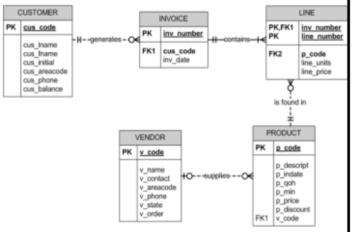
#comparisons needed for the join between tables A and B is:
|A| * |B|
Where |A| = cardinality (the number of rows) of table A

Join Operation

- JOIN is performed between two tables, which has a PK-FK relationship.
  - If your FROM clause contains **2 tables**, you need **one join**.
  - If your FROM clause contains **3 tables**, you need **two joins**.
  - If your FROM clause contains ***N*** **tables**, you need ***N-1*** **joins**.

Exercise: Join with aggregate functions

For each vendor name, find min price, max price, and average price of all the products supplied by each vendor. Also display vendor code. Sort output by vendor name.



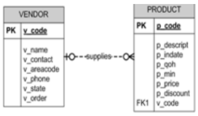
JOIN

- JOIN is the power of RDBMS
- JOIN is the most time-consuming operation in the queries
- Fortunately, modern DBMSs automatically optimize queries
  - **CBO (Cost-based optimization)**
    - **Use data statistics to optimize queries**

Alias

- An alias can be used for a table named in the **FROM** clause, where the alias is separated from the table name with a space
  - e.g., **FROM** Instructor **i**, Course **c**
  - Can two different tables have the same alias?
- When can an alias be used?
  - Used to qualify a column name whenever there is ambiguity regarding the *source of the column name*
  - e.g., **WHERE** **i**.instructorID = **c**.instructorID
  - Also used as a shorthand notation for the table name
- An alias can be used anywhere in place of the table name

Using Table Alias



- The following commands are the same:
- SELECT P_Code, Vendor.V_Code, V_Name
  FROM Product, Vendor
  WHERE Product.V_Code = Vendor.V_code;
- SELECT P_Code, **V**.V_Code, V_Name
  FROM Product **P**, Vendor **V**
  WHERE **P**.V_Code = **V**.V_code;
- SELECT **P** .P_Code, **V**.V_Code, **V**.V_Name
  FROM Product **P**, Vendor **V**
  WHERE **P**.V_Code = **V**.V_code;
- In Access
  SELECT P_Code, **V**.V_Code, V_Name
  FROM Product AS **P**, Vendor AS **V**
  WHERE **P**.V_Code = **V**.V_code;

Sorting a Simple Join

- list the instructor IDs, names, and course numbers, titles, and section numbers that they teach, and order results by instructor IDs (ASC), course numbers (ASC), & section numbers (DESC)

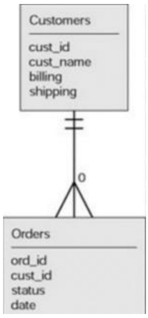
SELECT i.instructorID, fName, lName, courseNo, title, sectionNo
FROM Instructor i, Course c
WHERE i.instructorID = c.instructorID
ORDER BY i.instructorID, courseNo **ASC**, sectionNo **DESC**;

- Result:

instructorID	fName	lName	courseNo	title	sectionNo
44	Jennifer	Furman	ACCT101	Accounting I	2
44	Jennifer	Furman	ACCT101	Accounting I	1
76	Andy	Chou	MATH105	Algebra	1
89	Daniel	Pradmore	ACCT102	Accounting II	1

- In the **SELECT** list, i.instructorID qualifies that instructorID is chosen from Instructor table. Such qualification is achieved by prefixing the column name with the proper table name (or its alias)
- Note: instructorID exists in the PK in **Instructor** and an FK in **Course**

Exercise



- Find customer names who order status is back_ordered. Assume Orders.status has one value from {shipped, unshipped, back_ordered}

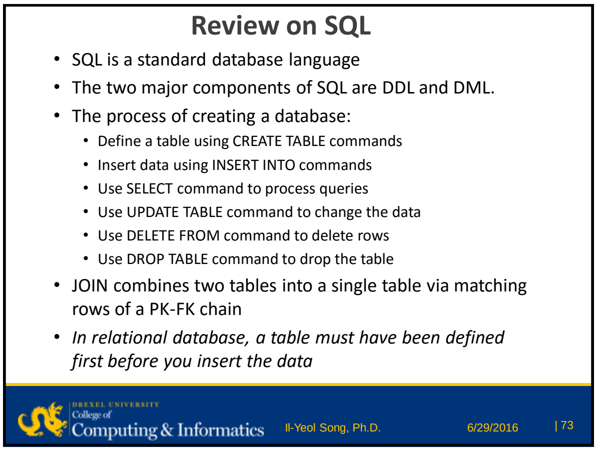
Q23: Three-table Joining

- For each course that *has an instructor and a teaching assistant*, list its courseNo, title, sectionNo, instructor ID & name, teachingAssistantID & studentID

SELECT courseNo, title, sectionNo, i.instructorID, fName, lName, t.teachAssistantID, studentID
FROM Instructor i, Course c, TeachingAssistant t
WHERE i.instructorID = c.instructorID **AND** t.teachingAssistantID = c.teachingAssistantID;

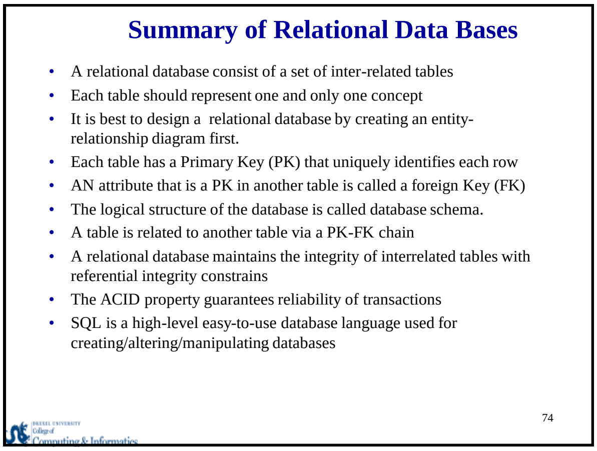
- Result:

courseNo	title	sectionNo	instructorID	fName	lName	teachingAssistantID	studentID
ACCT101	Accounting I	1	44	Jennifer	Furman	92	574
ACCT101	Accounting I	2	44	Jennifer	Furman	92	574
ACCT102	Accounting II	1	89	Daniel	Pradmore	37	379



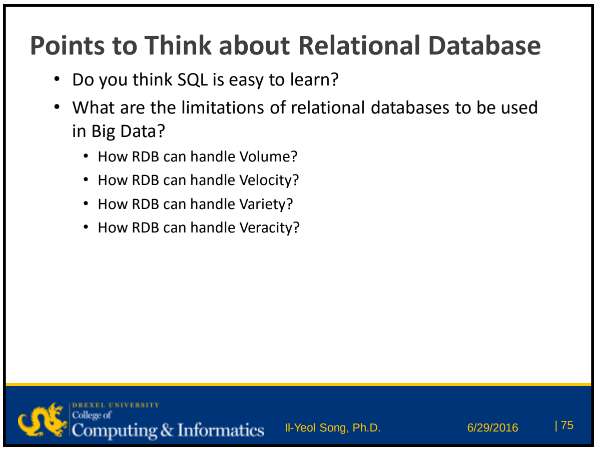
Review on SQL

- SQL is a standard database language
- The two major components of SQL are DDL and DML.
- The process of creating a database:
  - Define a table using CREATE TABLE commands
  - Insert data using INSERT INTO commands
  - Use SELECT command to process queries
  - Use UPDATE TABLE command to change the data
  - Use DELETE FROM command to delete rows
  - Use DROP TABLE command to drop the table
- JOIN combines two tables into a single table via matching rows of a PK-FK chain
- *In relational database, a table must have been defined first before you insert the data*



Summary of Relational Data Bases

- A relational database consist of a set of inter-related tables
- Each table should represent one and only one concept
- It is best to design a relational database by creating an entity-relationship diagram first.
- Each table has a Primary Key (PK) that uniquely identifies each row
- AN attribute that is a PK in another table is called a foreign Key (FK)
- The logical structure of the database is called database schema.
- A table is related to another table via a PK-FK chain
- A relational database maintains the integrity of interrelated tables with referential integrity constrains
- The ACID property guarantees reliability of transactions
- SQL is a high-level easy-to-use database language used for creating/altering/manipulating databases



Points to Think about Relational Database

- Do you think SQL is easy to learn?
- What are the limitations of relational databases to be used in Big Data?
  - How RDB can handle Volume?
  - How RDB can handle Velocity?
  - How RDB can handle Variety?
  - How RDB can handle Veracity?