4: SQL –Basic Query

Database System Concepts

Abraham Silberschatz, Henry F. Korth, S. Sudarshan

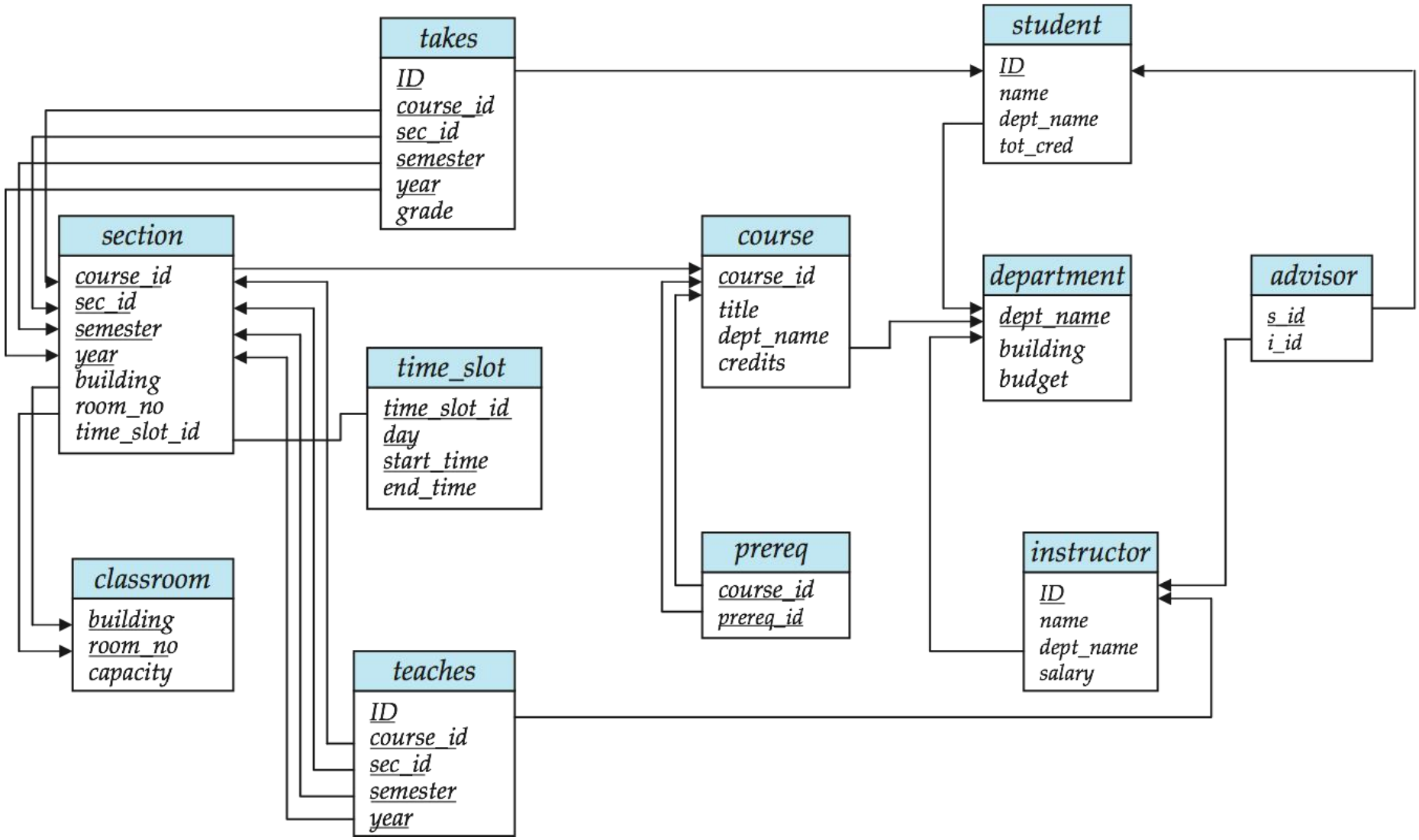
&

Oracle database SQL Reference

Basic SQL Query

- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

Schema



Schema referred in query examples.

Basic Query Structure

- The SQL **data-manipulation language (DML)** provides the ability to query information, and insert, delete and update tuples
- 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 - condition.
- The result of an SQL query is a relation.

The select Clause

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

select *name*

from *instructor*

- **NOTE**: SQL names are **not case-sensitive** (i.e., you may use upper- or lower-case letters.)

The select Clause (Cont.)

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

```
select distinct dept_name
from instructor ;
```

- The keyword **all** specifies that duplicates not be removed.

```
select all dept_name
from instructor ;
```

The select Clause (Cont.)

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

```
select *
from instructor ;
```

- 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.

The where Clause

- The **where** clause specifies conditions that the result must satisfy
- **To find all instructors in Comp. Sci. dept with salary > 80000**

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.' **and** *salary* **>** 80000
- Comparison results can be combined using the logical connectives **and, or,** and **not.**
- **Comparisons** can be **applied to results of arithmetic expressions.**
- **Find the instructors whose salary exceed 100000 after 10% increment**

  select *name*
  from *instructor* **where Salary*1.1 >100000;**

The from Clause

Cartesian Product

- The **from** clause lists the relations involved in the query
  - Corresponds to the **Cartesian product** operation.
- **Find the Cartesian product *instructor X teaches***

```
select *
from instructor, teaches ;
```

  - generates **every possible instructor – teaches pair,** with all attributes from both relations.
- Cartesian product **not very useful directly**, but **useful** combined **with where-clause** condition.

Cartesian Product: *instructor X teaches*

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

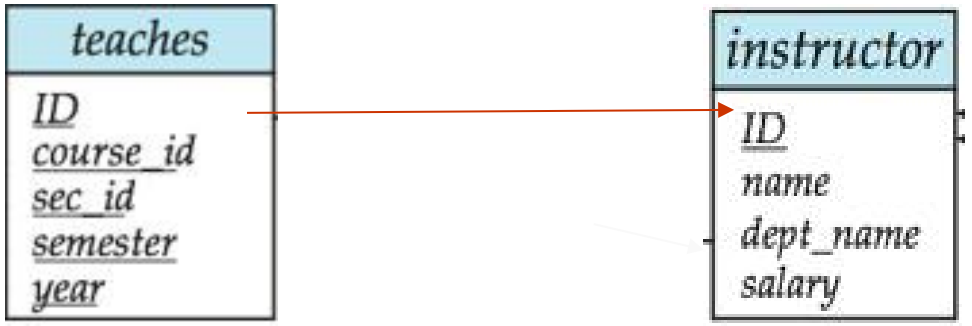
teaches

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

inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Joins

For all instructors who have taught some course, find their names and the course ID of the courses they taught.



select *name, course_id*

from *instructor, teaches*

where *instructor.ID = teaches.ID*;

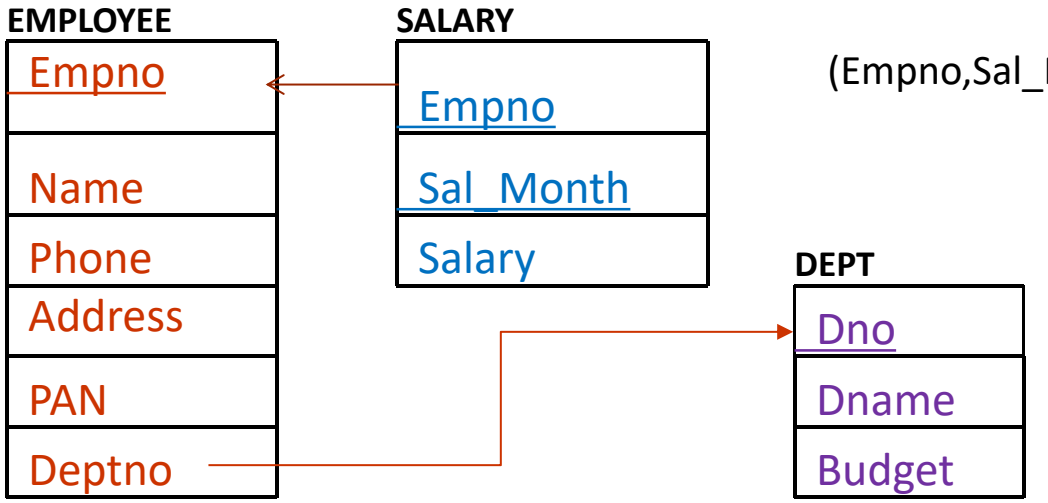
Joins

Find the course ID, semester, year and title of each course offered by the Comp. Sci. department.


```
select section.course_id, semester, year, title
  from section, course
  where   section.course_id = course.course_id  and
                dept_name = 'Comp. Sci. ' ;
```

Try Writing Some SQL Queries

- Suggest queries to be written…..



(Empno,Sal_Month) Primary key in Salary

1. Display the employee Names whose PAN number is **APQ4679Z5**
2. Display name of employee who is working in department number **D10** and having PAN number **APH69G4**.
3. Display the name of employees who got salary more than **90000** during month **August.**
4. Display the name of employees and their respective department names.
5. Display the name of employees who are working in the '**Accounts'** department
6. Display the name of employee, Salary and name of department in which he is working.

Natural Join

- Natural join matches tuples with the same values for all **common attributes**, and retains **only one copy** of each common column.
- **select** * **from** *instructor* **natural join** *teaches* ;

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

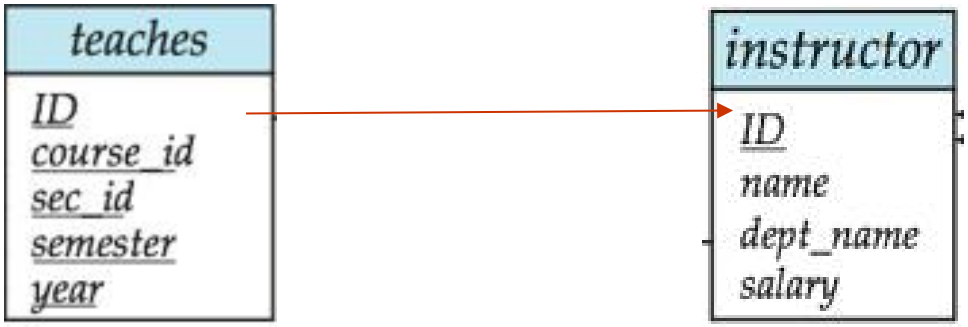
Natural Join Example

- **List the names of instructors along with the course ID of the courses that they taught.**

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

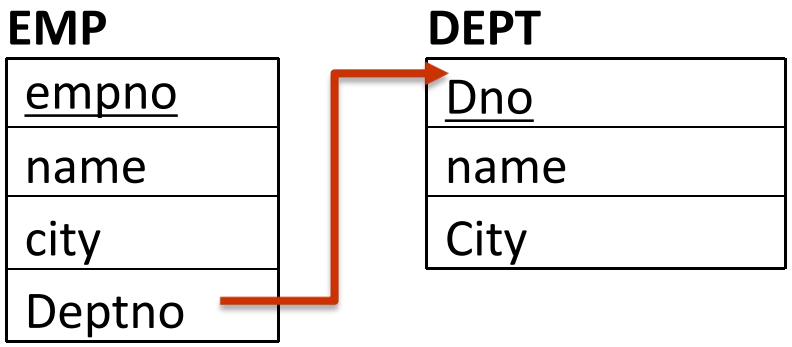
  OR

  - **select** *name*, *course_id*
    from *instructor* **natural join** *teaches*;



Natural Join (Cont.)

- **Danger in natural join:** beware of unrelated attributes with same name which get equated incorrectly.
- List Employee name, Department name of departments which they are working.



Select Name, Dname from Emp NATURAL JOIN Dept;

This is **incorrect.**

Three-Table Join

Consider the following table structure.

PROPERTYFORRENT (**pno**, street, area, city, pcode, type, rooms, rent, sno)

STAFF (**sno**, fname, lname, position, sex, DOB, salary, bno)

BRANCH (**bno**, street, city, postcode)

Example:

For each branch, list the staff who manage properties, including the city in which the branch is located and the properties they manage.

```
SELECT  b.bno, b.city, s.sno, fname, lname, pno
        FROM  propertyForRent  p, staff s, branch b
            WHERE  p.sno= s.sno AND s.bno= b.bno ;
```

Computing a Join with other clauses
DISTINCT,WHERE,ORDER BY

The procedure for generating the results of a SELECT with a join are as follows:

- Form the **Cartesian product** of the tables named in the **FROM** clause.
- If there is a **WHERE** clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition. In terms of the relational algebra, this operation yields a restriction of the Cartesian product.
- For each remaining row, determine the value of each item in the **SELECT list** to produce a single row in the result table.
- If **SELECT DISTINCT** has been specified, eliminate any duplicate rows from the result table.
- If there is an **ORDER BY** clause, sort the result table as required.

Order of Keywords- from > where > group by > having > order by

Outer Join

The **join** operation combines data from two tables by forming pairs of related rows where the matching columns in each table have the same value. If one row of a table is unmatched, the row is omitted from the result table.

Outer join include the **unmatched rows** in the result table.

Three types of outer join:

- Left
- Right
- Full

Join Example

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

SELECT b.*, p.*

FROM branch b, property p

WHERE b.bcity = p.pcity;

In the example Branch b , property p, **b** and **p** are said to be Alias names to the tables branch & property respectively.

Left Outer Join

Example:

List the branch offices and properties that are in the same city along with any unmatched branches.

```
SELECT  b.*, p.*
        FROM  Branch  b  LEFT  JOIN  Property  p ON
                b.bcity  =  p.pcity;
```

Result of Left Outer Join

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

```
SELECT  b.*, p.*
 FROM  branch  b
  LEFT  JOIN  property  p ON
            b.bcity  =  p.pcity;
```

Right Outer Join

Example:

List the branch offices and properties in the same city and any unmatched property.

```
SELECT  b.*, p.*
        FROM  Branch  b  RIGHT JOIN  Property  p ON
                b.bcity  =  p.pcity;
```

Result of Right Outer Join

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PL94	London
B002	London	PG4	Glasgow

```
SELECT  b.*, p.*
  FROM  Branch  b
     RIGHT  JOIN  Property  p ON
               b.bcity  =  p.pcity;
```

Full Outer Join

Example:

List the branch offices and properties that are in the same city and any unmatched branches or properties.

```
SELECT  b.*, p.*
        FROM  branch  b FULL JOIN  property  p ON
                b.bcity  =  p.pcity;
```

Result of FULL Outer Join

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

```
SELECT  b.*, p.*
  FROM  branch  b
     FULL  JOIN  property  p ON
                 b.bcity  =  p.pcity;
```

The Rename Operation

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

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

- **Renaming(Alias) to Columns**
- E.g.
  - **select** *ID, name, salary/12* **as** *monthly_salary*
    from *instructor;*

- **Renaming(Alias) to Tables**
  - **Select A.ID From** Instructor **A WHERE A.**Dept_Name='Comp Sc.';
- Keyword **as** is optional and may be omitted

  instructor **as** *T equivalent to instructor T*
  - Keyword **as** may be **omitted in Oracle**

Compare Tuples in the Same Relation

- Another reason to **rename a relation** is a case where we wish to **compare tuples in the same relation**. We then need to take the Cartesian product of a relation with itself.
- Since we need to join a tuple of relation with another tuple in the same relation, It becomes **impossible to distinguish one tuple from another without renaming.**
- **Example: Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.** ***Instructor(ID,Name,Salary)***
  - **select distinct** ***T.*** *name*

    from *instructor* ***T****, instructor* ***S***

    where ***T.****salary* **>** ***S.****salary* **and** ***S.****dept_name* = *'Comp. Sci.'*

An identifier, such as ***T*** and ***S***, that is used to rename a relation is referred to as **a correlation name** and is also commonly referred to as a **table alias**, or a **correlation variable**, or a **tuple variable**.

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 single character.
- **Find the names of all instructors whose name includes the substring "dhar".**

```
select name
    from instructor
    where name like '%dhar%'
```

String Operations (Cont.)

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

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

Ordering the Display of Tuples

- **ORDER BY column1 [ASC|DESC],Column2 [ASC|DESC],..**
- **List in alphabetic order the names of all instructors**.

  Example: 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: select distinct *name*
  from *instructor* **order by** *name* **desc**

- Can sort on **multiple attributes**

  Example: select distinct *name*
  from *instructor* **order by** *dept_name* **asc** , *name* **desc**

Order of Keywords- from **>** where **>** group by **>** having **>** order by

Where Clause Predicates

- SQL includes a **between , Not 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
- This is equivalent to
  - **select** *name*
    from *instructor*
    where *salary* **>=**90000 **and** **salry <=**100000
  - Similarly NOT BETWEEN can also be used
    - **Ex:** ….. WHERE salary **NOT BETWEEN** 90000 **and** 100000

Set Operations-UNION

- **Display Employee names who are working in deptno 10 , 20.**
- **Assume the schema EMP(empno,ename,sal,deptno)**

```
SELECT Ename
FROM Emp WHERE deptno=10
   UNION
SELECT Ename
FROM Emp WHERE deptno=20;
```

The query is equivalent to-

```
SELECT Ename
FROM Emp
WHERE deptno=10 OR deptno=20;
```

Set Operations-INTERESECT

- **Find the name of employees working in department –'RESEARCH' and drawing salary more than 2500.**
- **Assume Emp(**empno,ename,sal,deptno**) & Dept(**Deptno,Dname,City**)**
  - **Emp.deptno is F.key referencing Dept.Deptno(p.key)**

Select ename from emp,dept where **emp.deptno=dept.deptno and dname='RESEARCH'**

INTERSECT

Select ename from emp,dept where **emp.deptno=dept.deptno** and **sal>2500;**

Set Operations- MINUS

- **Find ename and deptno from emp who are not working in deptno 20.**

Select empno,ename,deptno from emp

MINUS

Select empno,ename,deptno from emp where deptno=20;

Set Operations

- **Find courses that ran in Fall 2009 or in Spring 2010**

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

- **Find courses that ran in Fall 2009 and in Spring 2010**

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

- **Find courses that ran in Fall 2009 but not in Spring 2010**

(**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
minus
(**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

Set Operations

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

Operator	Returns
UNION	All distinct rows selected by either query
UNION ALL	All rows selected by either query, including all duplicates
INTERSECT	All distinct rows selected by both queries
MINUS	All distinct rows selected by the first query but not the second

- Suppose a tuple occurs *m* times in *r* and *n* times in *s,* then, it occurs:
  - ***m*** **+** ***n*** times in ***r*** **union all** ***s***

Null Values- is null

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null*
  - **Example**: **5 + *null*** returns null
- The predicate **is null** can be used to **check for null values**.
  - **Example**: Find all instructors whose salary is null.

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

IS NULL/ IS NOT NULL

.

Example:

List the details of all staff who have only first name but not last name.

```
SELECT sno, fname, lname, salary
  FROM staff
    WHERE lname IS NULL AND fname IS NOT NULL;
```

Null Values and Three Valued Logic

- Any **comparison** **with** ***null*** **returns** ***unknown***
  - Example*: 5 < null or null <> null or null = null*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown)* = *unknown*
  - AND: *(true* **and** *unknown)* = *unknown,*
    (false **and** *unknown)* = *false,*
    (unknown **and** *unknown)* = *unknown*
  - NOT*: (***not** *unknown)* = *unknown*
- Result of **where** clause predicate is treated as ***false*** if it evaluates to *unknown*

Write SQL Queries for the followings

CUSTOMER(Custid, Name, Mid_Name, LastName, City, phone, email)

ACCOUNT(AccNo, CustId, Intr_CustId, AccType,City, Branch,Balance)

1. Find name of customers who are in Bangalore or Mangalore.
2. Find Name his AccNo in which balance is 200000/- to 500000/-.(**between**)
3. Find the customer name who hasn't given email id.
4. Sort the name of customers based on City in descending order and name in ascending order.
5. Question 1 using **set** operator.
6. Find the names who are from city Manipal and having at least one account in Manipal. (**set** operator)
7. Find the branches in all the cities except the branches in city Mangalore. (**set** operator)

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

Consider following relation to write sql queries(next slide)

Student(Rollo,Name,Mark1,Mark2,Attendance)

Student(RollNo,Name,Mark1,Mark2,Attendance)

- **AVG()**

  SELECT AVG(*column_name*) FROM *table_name* WHERE *condition*;

  SELECT AVG(Mark1) FROM Student;

- **SUM()**

  SELECT SUM(*column_name*) FROM *table_name* WHERE *condition*;

  SELECT SUM(Mark1) FROM Student;

- **COUNT()**

  SELECT COUNT(*column_name*) FROM *table_name* WHERE *condition*;

  SELECT COUNT(RollNo) FROM Student;

- **MAX()**

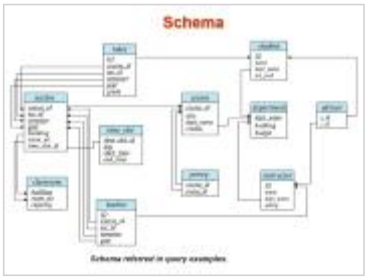
  SELECT MAX(*column_name*) FROM *table_name* WHERE *condition*;

  SELECT MAX(Mark1) FROM Student;

  SELECT MIN(Mark1) FROM Student;

Aggregate Functions (Cont.)

- **Find the average salary of instructors in the Computer Science department**



  - **select avg** (*salary*)
    from *instructor*
    where *dept_name*= ’Comp. Sci.’;

- **Find the total number of instructors who teach a course in the Spring 2010 semester**

  - **select count** (**distinct** *ID*)
    from *teaches*
    where *semester* = ’Spring’ **and** *year* = 2010;

- **Find the number of tuples in the *course* relation**

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

Aggregate Functions – Group By

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

INSTRUCTOR

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

Figure A.5 The *instructor* relation.

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.)

- Apart from aggregate functions, attributes in **select** clause must be the attributes which appear in **group by** list only.
- **Instructor(ID, Name, Dept_Name, Salary)**
  - /* **erroneous** query- Not Group By field */

    select *dept_name*, *ID*, **avg** (*salary*)

    from *instructor*

    group by *dept_name*;

Aggregate Functions – Having Clause

Instructor(ID, Name, Dept_Name, Salary)

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

```
select dept_name, 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**

Display the name and average salary of **History** , **Comp Sc** departments if their average salary is greater than 42000

Null Values and Aggregates

- Total all salaries

  select sum (*salary*)
  from *instructor;*

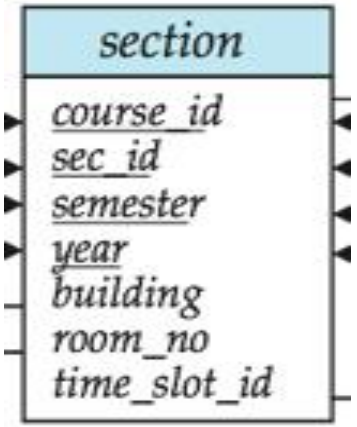
  - Above statement **ignores** null salary amounts
  - Result is *null* if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes.
- What if collection has only null values?
  - count returns **0**
  - all other aggregates return **null**

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.
- A common **use of subqueries** is to perform **tests** for **set membership**, **set comparisons**, and **set cardinality**.

Example-set membership using IN

- **Find courses offered in the both semester-'Fall' , year 2009 and in Spring, 2010**




```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
        course_id IN (select course_id
                      from section
                      where semester = 'Spring' and year= 2010);
```

tested membership in a **one-attribute** relation

Inner query gives **set** of course_id **= {CS-101, CS-315,CS-319,FIN-201,HIS-351,MU-199}**

Outer Query gives **set** of course_id**={CS-101,CS-347,PHY-101}**

IN operator will; check-

CS-101 ϵ {CS-101, CS-315,CS-319,FIN-201,HIS-351,MU-199} ? If YES , CS-101 is the Result
Similarly for CS-347,PHY-101

ANY OTER WAYS to WRITE the QUERY?

Section table data

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

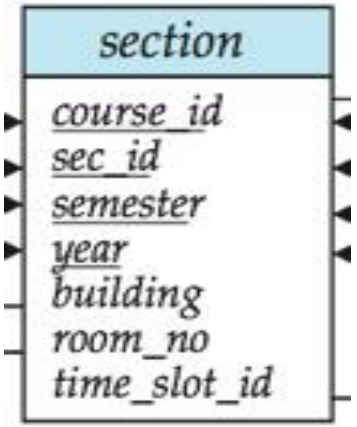
Fall' , year 2009 and in Spring, 2010

Inner query gives **set** of course_id **= {CS-101, CS-315,CS-319,FIN-201,HIS-351,MU-199}**

Outer Query gives **set** of course_id**={CS-101,CS-347,PHY-101}**

set membership Example Query

- **Find courses offered in Fall 2009 but not in Spring 2010**




```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
        course_id NOT IN (select course_id
                          from section
                          where semester = 'Spring' and year= 2010);
```

NOT IN operator will; check-

CS-101 ∉ **{ CS-101, CS-315, CS-319, FIN-201, HIS-351, MU-199 }** ?

If YES , CS-101 is the Result

Similarly for CS-347,PHY-101

Other way to write Query?

Example Query-IN ,order pairs

- **Find the total number of (distinct) students who have taken course sections taught by the instructor with *ID* 10101.**

select count (**distinct** *ID*)

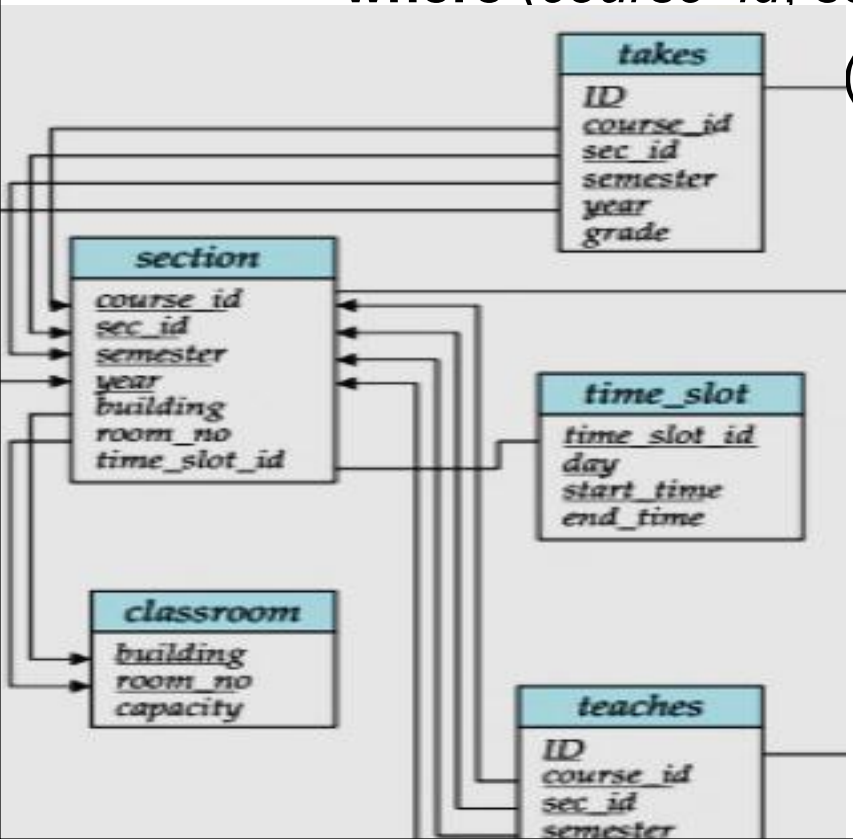
from *takes*

where (*course_id*, *sec_id*, *semester*, *year*) **IN**

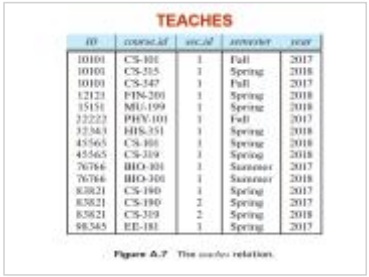
(**select** *course_id*, *sec_id*, *semester*, *year*

from *teaches*

where *teaches.ID*= 10101);



Test for membership to any relation (group of attributes)



Set Comparison

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

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 A.6 The *instructor* relation.

Set Comparison- some

- **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 T, instructor 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');
```

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 **< some** | 0 / 5 / 6 |) = true (read: **5 < some tuple in the relation**)

(5 **< some** | 0 / 5 |) = false

(5 **= some** | 0 / 5 |) = true

(5 **≠ some** | 0 / 5 |) = true (since 0 ≠ 5)

(**= some**) ≡ **in**

However, (**≠ some**) ≢ **not in**

SQL also allows **< some**, **<= some**, **>= some**, **= some**, and **<> some** comparisons.

Set Comparison- ALL

Example Query

- **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 all Clause

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

(5 < **all** {0, 5, 6}) = false

(5 < **all** {6, 10}) = true

(5 = **all** {4, 5}) = false

(5 ≠ **all** {4, 6}) = true (since 5 ≠ 4 and 5 ≠ 6)

(≠ **all**) ≡ **not in**
However, (= **all**) ≢ **in**

SQL also allows < **all**, <= **all**, >= **all**, = **all**, and <> **all** comparisons

SQL> **select empno, ename,sal,deptno from emp order by deptno;**

EMPNO	ENAME	SAL	DEPTNO
7934	MILLER	10000	10
7782	CLARK	2450	10
7839	KING	5000	10
1000	DDD	3333	10
7566	JONES	2975	20
7369	SMITH	800	20
1001	EEEE	2005	20
7902	FORD	3000	20
7876	ADAMS	1100	20
7788	SCOTT	3000	20
7900	JAMES	950	30
7499	ALLEN	1600	30
7844	TURNER	1500	30
50	BBB	777	30
7521	WARD	1250	30
7698	BLAKE	2850	30
7654	MARTIN	1250	30

SQL> **select sal,deptno from emp where deptno=20;**

SAL	DEPTNO
2005	20
800	20
2975	20
3000	20
1100	20
3000	20

Find the Employee No, Sal and Deptno of employees who earn salary more than every employee in Dept 20 earn.

SQL> select empno,sal,deptno from emp where sal > ALL (select sal from emp where deptno=20);

EMPNO	SAL	DEPTNO
1000	3333	10
7839	5000	10
7934	10000	10

SET COMPARISION- using ALL & ALTERNATE WAY

- Write the previous query by comparing salary of employees with maximum salary of employees in the department 20;

```
Select empno,sal,deptno from emp where sal >
        Select max(sal) From Emp where deptno=20;
```

Test for Empty/Non-Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \emptyset$ *non-empty*
- **not exists** $r \Leftrightarrow r = \emptyset$ *empty*

"Find all courses taught during both the Fall 2009 semester and in the Spring 2010 semester"

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Figure A.7 The *section* relation.

slide 42

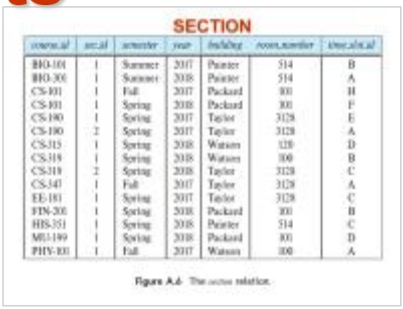
Correlation Variables-exists

- Yet another way of specifying the query "**Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester**"

```
select course_id
from section S
where semester = 'Fall' and year= 2009 and
        exists (select *
                from section T
                where semester = 'Spring' and year= 2010
                        and S.course_id= T.course_id);
```

- **Correlated subquery -**
- **Correlation name** or **correlation variable - S**

Correlation Variables-Exists


SECTION

- **select** *course_id*
  from *section* ***S***
  where *semester* = 'Fall' **and** *year*= 2009
- Result is –
- course_id**={CS-101,CS-347,PHY-101}**

- **select** *
  from *section* ***T***
  where *semester* = 'Spring' **and** *year*= 2010
- Result is-

{<CS-101,..>,< CS-315,…>,<CS-319,…>,<FIN-201,…>,<HIS-351,…>,<MU-199,…>}

and ***S***.*course_id*= ***T***.*course_id*);

CS-101=CS-101 **YES**, inner query return row(row exists satisfying)

CS-347=CS-101/CS-325/…./MU-199 **NO**, No row exists.

PHY-101= CS-101/CS-325/…./MU-199 **NO**, No row exists

General form of – Correlated Query

A correlated subquery is evaluated once for each row processed by the parent(outer) statement. The parent statement can be a **SELECT**, **UPDATE**, or **DELETE** statement.

```
SELECT column1, column2, ....
FROM table1 outer
WHERE column1 operator
        (SELECT column1, column2
         FROM table2
         WHERE expr1 = outer.expr2 );
```

Correlation Variables-Not Exists

- **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')
                   MINUS
                   (select T.course_id
                    from takes as T
                    where S.ID = T.ID));
```

All courses (ids) offered by Biology Department

Courses (Ids) taken by each student **S** referred in the outer query

- Note that $X - Y = \emptyset \Leftrightarrow X \subseteq Y$

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Courses

BIO-101,BIO-301,BIO-399 are the course offered by Biology department

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Figure 4.1 The *student* relation.

ID	course_id	sec_id	semester	year	grade
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	*null*

Figure 4.2 The *takes* relation.

BIO-101,BIO-301,BIO-399 are the course offered by Biology department

Find all the courses taken by each student in Student relation , e.g. 00128 has taken CS-101,CS-347
{BIO-101,BIO-301,BIO-399} MINUS {CS-101,CS-347}= NOT EMPTY set means Student has not taken not even one course Offred by the Biology department.

Previous Class

- NESTED SUB QUERIES
  - SET MEMBERSHIP
    - IN , NOT IN
  - SET COMPARISON
    - <comp> SOME, <comp> ALL
    - Where <comp> can be: $<, \leq, >, =, \neq$
  - SET CARDINALITY
    - EXISTS , NOT EXISTS

Find the customers who have placed order on the date 2016/04/18

Customer

customer_id	last_name	first_name
4000	Jackson	Joe
5000	Smith	Jane
6000	Ferguson	Samantha
7000	Reynolds	Allen
8000	Anderson	Paige
9000	Johnson	Derek

Order

order_id	customer_id	order_date
1	7000	2016/04/18
2	5000	2016/04/18
3	8000	2016/04/19
4	4000	2016/04/20
5	NULL	2016/05/01

using JOIN

```
Select First_Name FROM  Customer,Order where
Customer.Customer_id=Order.Customer_id and
order_date=to_date('2016/04/18','yyyy/mm/dd'));
```

Equivalently using IN

```
Select First_Name FROM  Customer  Where Customer_id IN ( select Customer_id
From Order where order_date=to_date('2016/04/18','yyyy/mm/dd'));
```

Equivalently using EXISTS

```
SELECT First_Name FROM customers  T WHERE EXISTS (SELECT * FROM
order_details WHERE T.customer_id = order_details.customer_id and
order_date=to_date('2016/04/18','yyyy/mm/dd'));
```

EXAMPLE

SQL> select *from emp5;

EMPNO	ENAME	MGR_ID	SAL	DEPTNO
1234	Ravi	1212	79999	111
1212	Raj	4567	89999	123
4567	Ram	1234	99000	124
4597	Ramesh	4567	99000	124
1297	Ravi		99000	123

SQL> select * from dept5;

DEPTNO	DNAME	LOC
111	research	MNG
123	Admin	MUB
124	Hr	BNG

SQL> select *from tax5;

EMPNO	TAX_YEAR	TAX_AMOUNT
1234	2019	45660
1234	2020	45790
4567	2020	95790
4567	2018	85790
1212	2018	85790
1212	2019	88990
1212	2020	86990
1297	2020	76990
1297	2019	76450
1297	2018	66450

EXAMPLE

EMP5(Empno, Ename, MGR_ID, Deptno)

DEPT(Deptno, Dname, Loc)

TAX5(Empno, Tax_Year,Tax_Amount)

Find the name of employees who are working in the department in which employee with employee number 1212 is working.

IN

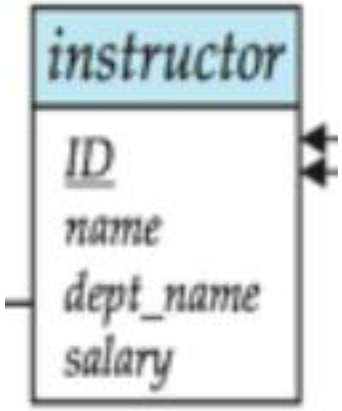
```
SQL> select ename from EMP5 where deptno
IN (Select deptno from EMP5 where
empno=1212);
```

Subqueries in the From 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) avg_salary
      from instructor
      group by dept_name) where avg_salary > 42000;
```

- Note that we do not need to use the **having** clause



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 budget
from department, max_budget
where department.budget = max_budget.value;
```

Syntax: WITH query_name As (SQL query) SELECT * FROM query_name;

CREATE .. AS.. SELECT…

Create a table with the same schema as an existing table

Syntax:

CREATE TABLE <new_table_Name> AS SELECT

<column1>,<column2>,… FROM <existing_table_name>

WHERE <condition>;

Example:

- Create a table Emp_Copy1 with the same schema as an existing Emp table:

CREATE TABLE Emp_Copy1 AS SELECT * FROM EMP;

Example:

- Create a table Emp_Copy2 with columns Empno,Ename,Sal from an existing Emp table:

CREATE TABLE Emp_Copy2 AS SELECT Empno,Ename,Sal FROM EMP;

CREATE .. AS.. SELECT…

Create a table with the same schema as an existing table

Example:

- Create a table Emp_Copy3 with columns Eno, Name, Salary by taking the column structure information from an existing Emp table:

CREATE TABLE Emp_Copy3(Eno, Name, Salary) AS SELECT

Empno,Ename,Sal FROM EMP;

Create.. AS.. Select .. Command not only copies the structure of select copies but **also copies data from** those columns.

Example:

CREATE TABLE Emp_Copy3(Eno,Name,Salary) AS SELECT

Empno,Ename,Sal FROM EMP WHERE Deptno=10;

EXAMPLE

EMP5(Empno,Ename,MGR_ID,Deptno)

DEPT(Deptno,Dname,Loc)

TAX5(Empno,Tax_Year,Tax_Amount)

Find the employee numbers who work under the manager that employee number 1212 works

EXISTS

```
SQL> select empno from emp5 T where exists (select *
from emp5 S where T.mgr_id=S.mgr_id and
s.empno=1212);
```

IN

- SQL> select empno from emp5 where mgr_id IN (select mgr_id from emp5 where empno=1212);

Scalar Subquery

The value of the scalar subquery expression is the value of the select list item of the subquery. If the subquery returns **0** rows, then the value of the scalar subquery expression is **NULL** .

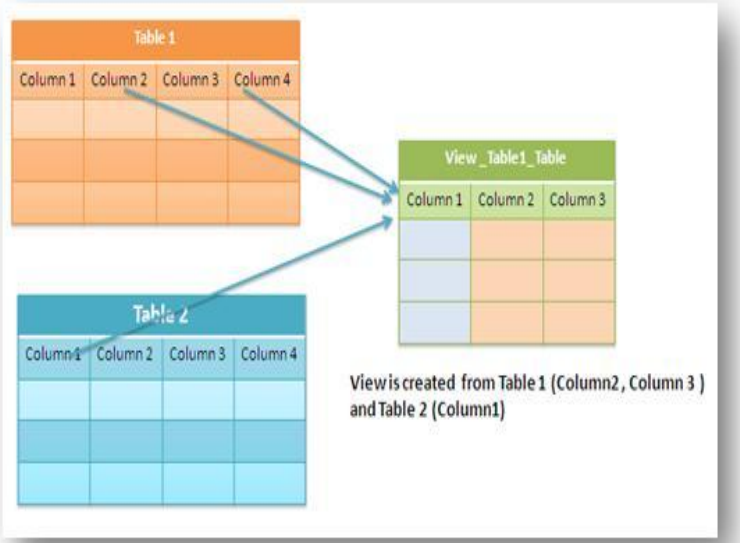
The subquery must **return only one tuple containing a single attribute**

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

VIEWS

- A **view** provides a mechanism to hide certain data from the view of certain users.
- Any relation that is not of the conceptual model but is made visible to a user as a "**virtual relation**" is called a **view**.

Some times, it is not desirable for all users to see the entire logical model . Thus Provides some kind of Security.



View Definition

- A view is defined using the **create view** statement which has the form

  CREATE VIEW *v* AS < query expression >

  where <query expression> is any legal SQL expression. The view name is represented by ***v*.**

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

Example:

- CREATE VIEW V1 AS SELECT * FROM Emp;

```
SQL> desc v1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------
 EMPNO                                     NOT NULL NUMBER(4)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(9)
 MGR                                                NUMBER(4)
 HIREDATE                                           DATE
 SAL                                                NUMBER(7,2)
 COMM                                               NUMBER(7,2)
 DEPTNO                                             NUMBER(2)
```

Example:
Renaming column names in the View

```
CREATE VIEW V2 (employee_number, Name, Date_of_Join) AS
SELECT empno, ename, hiredate FROM Emp;
```

```
SQL> desc v2
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 EMPLOYEE_NUMBER                           NOT NULL NUMBER(4)
 NAME                                               VARCHAR2(10)
 DATE_OF_JOIN                                       DATE
```

Example: View with aggregate function

CREATE VIEW V3 AS SELECT job, avg(sal) Avg_sal FROM Emp GROUP BY job;

```
SQL> desc v3;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 JOB                                                VARCHAR2(9)
 AVG_SAL                                            NUMBER
```

Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation v_1 is said to *depend directly* on a view relation v_2 if v_2 is used in the expression defining v_1

Example:

Creating a View from another view

- CREATE VIEW **v5** AS SELECT employee_number,name FROM **v2**

  WHERE date_of_join>'17-DEC-1980';

```
SQL> desc v5;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 EMPLOYEE_NUMBER                           NOT NULL NUMBER(4)
 NAME                                               VARCHAR2(10)
```

Querying a View

Views can be queried in the same way like Base Tables

SQL> SELECT * FROM V5;

EMPLOYEE_NUMBER	NAME
7499	ALLEN
7521	WARD
7566	JONES
7654	MARTIN
7698	BLAKE
7782	CLARK
7788	SCOTT
7839	KING
7844	TURNER
7876	ADAMS
7900	JAMES

Update of a View

- **Add a new tuple to *faculty* view which we defined earlier**

  INSERT INTO *faculty* **VALUES** ('30765', 'Green', 'Music');

  This insertion must be represented by the insertion of the tuple

  ('30765', 'Green', 'Music', null) into the *instructor* relation

insert into v2 values(7868,'Ravi','10-Oct-2012');

```
SQL> desc v2;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 EMPLOYEE_NUMBER                           NOT NULL NUMBER(4)
 NAME                                               VARCHAR2(10)
 DATE_OF_JOIN                                       DATE
```

select empno,ename,hiredate from scott.emp;

```
SQL> select empno,ename,hiredate from emp;

     EMPNO ENAME      HIREDATE
---------- ---------- ---------
        50 BBB
      1212 XXX
      1213
      7868 Ravi       10-OCT-12
      1000 DDD
      1001 EEEE
      7369 SMITH      17-DEC-80
```

Some Updates cannot be Translated Uniquely

- Assume the following view is created-
- **create view** *instructor_info* **as**
  select *ID*, *name*, *building* **from** *instructor*, *department*
  where *instructor.dept_name= department.dept_name*;
- **insert into** *instructor_info* **values** ('69987', 'White', 'Taylor');
  - which department, if multiple departments in Taylor?
  - what if no department is in Taylor?
- Most SQL implementations **allow updates only on simple views**
  - The **from** clause has only one database relation.
  - The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification.
  - Any attribute not listed in the **select** clause can be set to null, such column should not set with NOT NULL or PRIMARY KEY
  - The query does not have a **group by** or **having** clause.

Example: VIEWS- Base tables- EMP & DEPT

```
SQL> select * from emp;

EMPNO DEP  ENAME          SAL DOB       JOB
----- --- --------- ------- --------- ----------
  100 D1  RAJ           75000 10-FEB-19
  101 D2  MAHESH        80000 15-FEB-20
  102 D3  RAVI          89000 12-DEC-19
  103 D1  TOM           78000 12-JAN-19
  104 D2  JAMES        100000
```

```
SQL> select * from dept;

DNO DNAME      LOCATION
--- ---------- ---------
D1  RESEARCH   MNG
D2  SALES      BNG
D3  ACCOUNTS   HYD
```

Creating- Updatable View

Create emp_view with fields ENO,NAME,DATE_OF_BIRTH by borrowing empno, ename, dob column definitions from EMP

CREATE VIEW emp_view(ENO,NAME,DATE_OF_BIRTH) AS SELECT empno, ename, dob FROM emp;

```
SQL> create view emp_view(ENO,NAME,DATE_OF_BIRTH) as select empno,ename,dob from emp;

View created.

SQL> select * from emp_view;

       ENO NAME       DATE_OF_B
---------- ---------- ---------
       100 RAJ        10-FEB-19
       101 MAHESH     15-FEB-20
       102 RAVI       12-DEC-19
       103 TOM        12-JAN-19
       104 JAMES
```

UPDATEABLE VIEW

```
SQL> select * from emp;

EMPNO DEP ENAME          SAL DOB       JOB
----- --- --------- ------- --------- ----------
  100 D1  RAJ         75000 10-FEB-19
  101 D2  MAHESH      80000 15-FEB-20
  102 D3  RAVI        89000 12-DEC-19
  103 D1  TOM         78000 12-JAN-19
  104 D2  JAMES      100000
```

Records in Base Table EMP Before updating View Emp_View

```
SQL> INSERT INTO emp_view VALUES(105,'VIJAY','01-OCT-2021');

1 row created.
```

Updating Emp_View by adding one new record

Records in Base Table EMP After updating View Emp_View.

i.e. Updates on View also affects underlying Base table

```
SQL> select * from emp;

EMPNO DEP ENAME          SAL DOB       JOB
----- --- --------- ------- --------- ----
  100 D1  RAJ         75000 10-FEB-19
  101 D2  MAHESH      80000 15-FEB-20
  102 D3  RAVI        89000 12-DEC-19
  103 D1  TOM         78000 12-JAN-19
  104 D2  JAMES      100000
  105     VIJAY              01-OCT-21

6 rows selected.
```

Example- Non Updateable View

Create a Emp_Dept_View(Name, Salary, Dept_Name) by borrowing column definitions- Ename,Sal from EMP and Dname from DEPT.

CREATE VIEW Emp_Dept_View (Name,Salary,Dept_Name) AS SELECT Ename,Sal,Dname FROM EMP,DEPT WHERE Deptno=Dno;

```
SQL> CREATE VIEW Emp_Dept_View (Name,Salary,Dept_Name) AS SELECT Ename,Sal,Dname FROM EMP,DEPT WHERE Deptno=Dno;
```

```
SQL> select * from emp_dept_view;

NAME           SALARY DEPT_NAME
---------- ---------- ----------
RAJ             75000 RESEARCH
TOM             78000 RESEARCH
MAHESH          80000 SALES
JAMES          100000 SALES
RAVI            89000 ACCOUNTS
```

..Example- Non Updateable View

CREATE VIEW Emp_Dept_View (Name,Salary,Dept_Name) AS SELECT Ename,Sal,Dname FROM EMP,DEPT WHERE Deptno=Dno;

```
SQL> desc emp_dept_view;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 NAME                                               VARCHAR2(10)
 SALARY                                             NUMBER(10)
 DEPT_NAME                                          VARCHAR2(10)

SQL> insert into EMP_Dept_View values('Rocky',78909,'ADMINSTRATE');
insert into EMP_Dept_View values('Rocky',78909,'ADMINSTRATE')
*
ERROR at line 1:
ORA-01776: cannot modify more than one base table through a join view
```

END

ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Figure 4.1 The *student* relation.

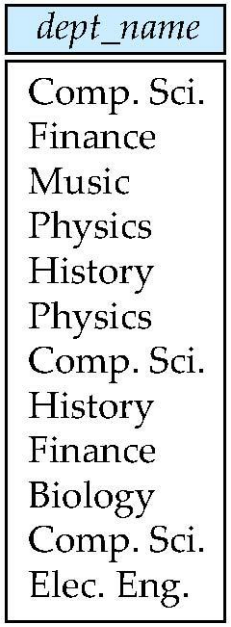
ID	*course_id*	*sec_id*	*semester*	*year*	*gra*
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	*nul*

Figure 4.2 The *takes* relation.

Figure 3.02

name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

Figure 3.03



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

Figure 3.04

Figure 3.05

name	*dept_name*	*building*
Srinivasan	Comp. Sci.	Taylor
Wu	Finance	Painter
Mozart	Music	Packard
Einstein	Physics	Watson
El Said	History	Painter
Gold	Physics	Watson
Katz	Comp. Sci.	Taylor
Califieri	History	Painter
Singh	Finance	Painter
Crick	Biology	Watson
Brandt	Comp. Sci.	Taylor
Kim	Elec. Eng.	Taylor

Figure 3.07

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

Figure 3.08

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Figure 3.09

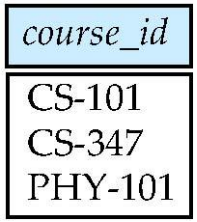
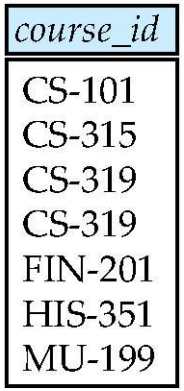
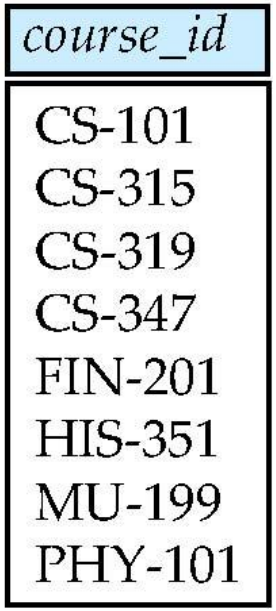


Figure 3.10



course_id
CS-101
CS-315
CS-319
CS-319
FIN-201
HIS-351
MU-199

Figure 3.11



course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Figure 3.12

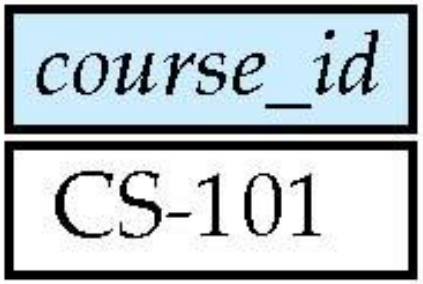


Figure 3.13

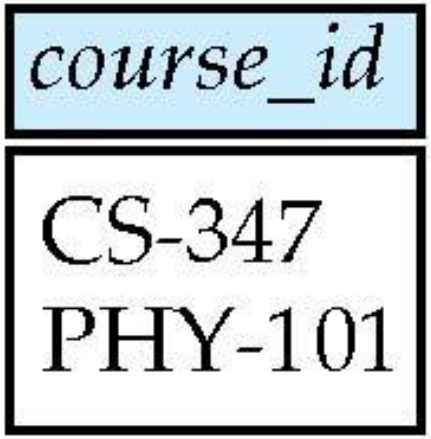


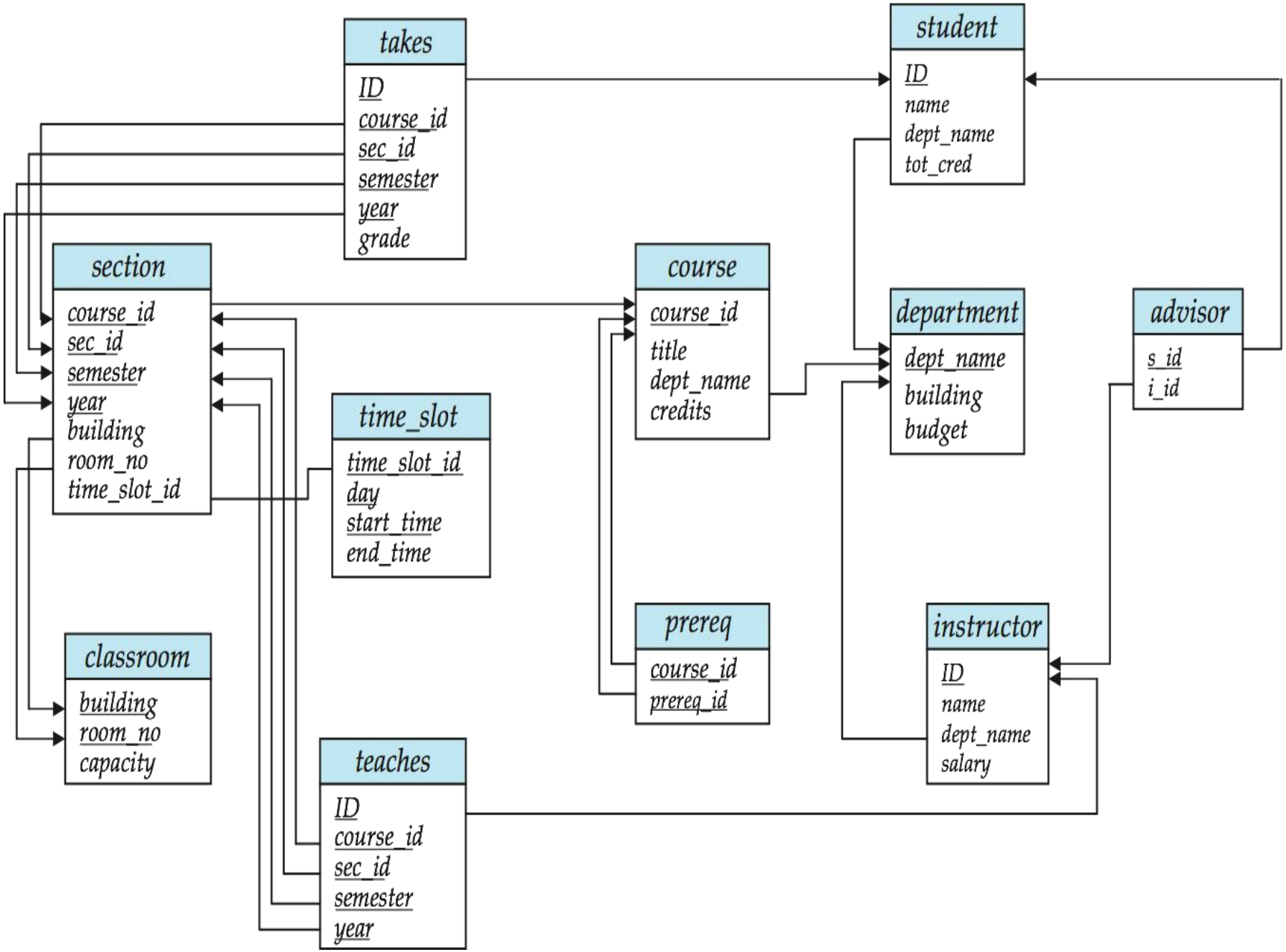
Figure 3.16

dept_name	*count*
Comp. Sci.	3
Finance	1
History	1
Music	1

Figure 3.17

dept_name	*avg(salary)*
Physics	91000
Elec. Eng.	80000
Finance	85000
Comp. Sci.	77333
Biology	72000
History	61000

Schema & Sample Data


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_no
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_no
capacity
teaches
ID
course_id
sec_id
semester
year

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

Figure A.2 The *classroom* relation

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Figure A.3 The *department* relation.

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Figure A.4 The *course* relation.

INSTRUCTOR

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

Figure A.5 The *instructor* relation.

SECTION

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

Figure A.6 The *section* relation.

TEACHES

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 A.7 The *teaches* relation.

STUDENT

ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Figure A.8 The *student* relation.

Takes & Advisor

ID	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	*null*

Figure A.9 The *takes* relation.

s_id	*i_id*
00128	45565
12345	10101
23121	76543
44553	22222
45678	22222
76543	45565
76653	98345
98765	98345
98988	76766

igure A.10 The *advisor* relation.

Time_Slot & PreReq

time_slot_id	*day*	*start_time*	*end_time*
A	M	8:00	8:50
A	W	8:00	8:50
A	F	8:00	8:50
B	M	9:00	9:50
B	W	9:00	9:50
B	F	9:00	9:50
C	M	11:00	11:50
C	W	11:00	11:50
C	F	11:00	11:50
D	M	13:00	13:50
D	W	13:00	13:50
D	F	13:00	13:50
E	T	10:30	11:45
E	R	10:30	11:45
F	T	14:30	15:45
F	R	14:30	15:45
G	M	16:00	16:50
G	W	16:00	16:50
G	F	16:00	16:50
H	W	10:00	12:30

Figure A.11 The *time_slot* relation.

course_id	*prereq_id*
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

igure A.12 The *prereq* relation.

time_slot_id	*day*	*start_hr*	*start_min*	*end_hr*	*end_min*
A	M	8	0	8	50
A	W	8	0	8	50
A	F	8	0	8	50
B	M	9	0	9	50
B	W	9	0	9	50
B	F	9	0	9	50
C	M	11	0	11	50
C	W	11	0	11	50
C	F	11	0	11	50
D	M	13	0	13	50
D	W	13	0	13	50
D	F	13	0	13	50
E	T	10	30	11	45
E	R	10	30	11	45
F	T	14	30	15	45
F	R	14	30	15	45
G	M	16	0	16	50
G	W	16	0	16	50
G	F	16	0	16	50
H	W	10	0	12	30

e A.13 The *time_slot* relation with start and end times separated into hour and minute.

Group By on 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

What be ID column values may be possible in the result given below ?

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

Note that , not possible to pick single ID value from the each group of dept_name. Therefore it **gives error**.