Basic SQL

Introduction

- The Structured Query Language (SQL) has several parts:
  1. **Data-definition language (DDL)** provides commands for defining relation schemas, deleting relations, and modifying relation schemas
  2. **Data-manipulation language (DML)** provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
  3. **Integrity** includes commands for specifying integrity constraints that the data stored in the database must satisfy.
     - Updates that violate integrity constraints are disallowed.
  4. **View definition** includes commands for defining views
  5. **Transaction control** includes commands for specifying the beginning and ending of transactions.

Introduction

6. **Embedded and dynamic SQL** define how SQL statements can be embedded within general-purpose programming languages
7. **Authorization** includes commands for specifying access rights to relations and views.

Basic Types

- **char(n):** A fixed-length character string with user-specified length n. The full form, character, can be used instead.
- **varchar(n):** A variable-length character string with user-specified maximum length n. The full form, character varying, is equivalent
- **int:** An integer. The full form, integer, is equivalent
- **smallint:** A small integer
- **numeric(p, d) or (number(p, d)):** A fixed-point number with user-specified precision

  - number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point.
- **float(n):** A floating-point number, with precision of at least n digits

Basic Schema Definition - CREATE TABLE construct

```
create table r(
A1 D1,
 A2 D2,
 . . . ,
 An Dn,
 <integrity-constraint1 >,
 . . . ,
 <integrity-constraintk >);
```

```
create table instructor(
        ID varchar(5),
        name varchar(20) not null,
        deptname varchar (20),
        primary key (ID),
        foreign key (deptname) references department );
```

Basic Schema Definition - CREATE TABLE construct

```
create table classroom (
        building varchar (15),
        roomnumber varchar (7),
        capacity numeric (4,0),
        primary key (building, roomnumber));

create table department (
        deptname varchar(20),
        building varchar(15),
        budget numeric(12, 2) check (budget > 0),
        primary key (deptname));
```

Basic Schema Definition - CREATE TABLE construct

create table course(

courseid **varchar**(8),

title **varchar** (50),
deptname **varchar** (20),
credits **numeric (2,0) check** (credits > 0),
primary key (courseid),
foreign key (deptname) **references** department **on delete set null;**

Basic Schema Definition - CREATE TABLE construct

```
create table section
(courseid varchar (8),secid varchar (8),
semester varchar (6) check (semester in
        ('Fall', 'Winter', 'Spring', 'Summer')),
year numeric(4,0) check(year >1701 and year <2100),
building varchar (15),
roomnumber varchar (7),
timeslotid varchar (4),
primary key (courseid,secid, semester, year),
foreign key (courseid)references course
on delete cascade,
foreign key (building,roomnumber) references
                classroom on delete set null);
```

Modification of DB - Insertion

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

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

- or equivalently

```
insert into course(courseid, title, deptname, credits) values ('CS-437', 'DBSystems', 'Comp. Sci.', 4);
```

- Add a new tuple to student with totcreds set to null

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

Modification of DB – Delete Construct/Drop Table

- Deleting all the contents of the table

  delete from student;

- Deleting a specific content from the table

  delete from student

  where P;

  example: delete from student

  where deptname = ' ICT ' ;

- Deleting table

  drop table student;

Modification of DB - Updation

- Annual salary increases are being made, and salaries of all instructors are to be increased by 5 percent

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

- If a salary increase is to be paid only to instructors with salary of less than ₹ 70000

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

- Increase salaries of instructors whose salary is over ₹100,000 by 3%, and all others receive a 5% raise

Modification of DB - Updation

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

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

Modification of DB - Updation

- Same query as before but with case statement

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

ALTER Table Construct

- Used to add or drop an attribute to/from a table

  alter table instructor **add** age **int**;

- Delete the attribute

  alter table instructor **drop** age ;

  alter table instructor **drop column** age ;

ALTER Table Construct

- Adding a foreign key

```
alter table B
        add foreign key ( name ) references A;

alter table B
        add constraint fkname
        foreign key ( name ) references A( name ) ;

alter table B
        drop constraint fkname ;

create table B(
        id number primary key,
        name varchar(10),
        constraint fkname foreign key ( name ) . . . ) ;
```

Basic Query Structure

- A typical SQL query has the following form:

  select A1, A2, … An

  from r1, r2, … rm

  where P;

- The result of an SQL query is a relation

SELECT Clause

- Select clause lists the attributes desired in the result of a query.

➢ Find the names of all instructors:

```
select name
from instructor;
```

- SQL names are case insensitive

➢ Find the department names of all instructors,

```
select deptname
from instructor;
```

- SQL allows duplicates in relations as well as in query results

SELECT Clause

- 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 name **from** instructor;

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

select *ID*, *name*, *salary/12* **from** *instructor*

WHERE Clause

- The where clause specifies conditions that the result must satisfy

➢ Find all instructors in Comp. Sci. dept with salary greater than ₹80000

Select name from instructor where dept='cse' 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

Queries on Multiple Relations

➢ Retrieve the names of all instructors, along with their department names and department building name

- The role of each clause is as follows:
  - ✓ The **select** clause is used to list the attributes desired in the result of a query.
  - ✓ The **from** clause is a list of the relations to be accessed in the evaluation of the query.
  - ✓ The **where** clause is a predicate involving attributes of the relation in the from clause
- Operational order: first **from,** then **where**, and then **select**
- The **from** clause by itself defines a Cartesian product of the relations listed in the clause.

Queries on Multiple Relations

select * from instructor, department;

Displays the Cartesian product of every tuple in the relations

- Instead, if we say

  select * from instructor, department

  where instructor . deptname = department . deptname;
- This displays the details of the instructors only once.

Queries on Multiple Relations

department(deptname, building, budget)

course(courseid, title, deptname, credits)

instructor(ID, name, deptname, salary)

section(courseid, sectionid, sem, year, building, roomnumber, timeslotid)

teaches(ID, courseid, secid, semester, year)

- For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught
- Find instructor names and course identifiers for instructors in the Computer Science department
- Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

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

select *name, course id*
from *instructor, teaches*
where *instructor.ID= teaches.ID*;

- Find instructor names and course identifiers for instructors in the Computer Science department
- Select name, courseid from instructor, teaches where instructor.id=teaches.id and deptname=‘cse’;

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

course(courseid, title, deptname, credits)

section(courseid, sectionid, sem, year, building, roomnumber, timeslotid)

- Select courseid, seme, year, building, title from course, section where (course.courseid=section.courseid and deptname=‘cse’);

- Sql will display the duplicates. To eliminate it use the keyword Distinct
- select distinct attribute from relation;

  select distinct deptname from instructor;

- Keyword ALL keeps all duplicates

Natural Join

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column

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

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

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

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

select name, courseid

from instructor, teaches

where instructor.id = teaches.id;

select name, courseid

from instructor **natural join** teaches;

Natural Join

➢ List the names of instructors along with the titles of courses that they teach

select name, title

from instructor **natural join** teaches **natural join** course;

- Is the query correct? Will this work if there is an instructor who teaches a course that belongs to another department?

Natural Join (Cont.)

- Danger in natural join: beware of unrelated attributes with same name which get equated incorrectly
- course(course_id, title, dept_name, credits)
- teaches(ID, course_id, sec_id, semester, year)
- instructor(ID, name, dept_name, salary)
- List the names of instructors along with the titles of courses that they teach
  - **select** *name*, *title*
    from *instructor* **natural join** *teaches* **natural join** *course*;

the natural join of *instructor* and *teaches* contains the attributes (*ID*, *name*, *dept name*, *salary*, *course id*, *sec id, sem, year*),

while the *course* relation contains the attributes (*course id*, *title*, *dept name*, *credits*).

As a result, the natural join of these two would require that the *dept name* attribute values from the two inputs be the same, in addition to requiring that the *course id* values be the same. This query would then omit all (instructor name, course title) pairs where the instructor

teaches a course in a department other than the instructor's own department.

Instructor natural join teaches

- (*ID*, *name*, *dept name*, *salary*, *course id*, *sec id*, *sem*, *year*), ①

Course relation is as follows :

- (*course id*, *title*, *dept name*, *credits*). ②

If ① natural join ②

common attributes courseid, deptname

- This query would then omit all (instructor name, course title) pairs where the instructor teaches a course in a department other than the instructor's own department

instructor . dept-name = course.dept-name ✗

List the names of instructors along with the titles of courses that they teach

- correct version
  - **select** *name*, *title*
    from (*instructor* **natural join** *teaches*)
    join *course* **using**(*course_id*);

- The operation **join** . . . **using** requires a list of attribute names to be specified.
- No other common attribute even if present will be matched, other than the ones indicated in using clause..

- ***Display id of students who have taken the course offered by physics department***

student(**SID**, name, dept_name, tot_cred)

takes(**SID**, course_id, sec_id, semester, year, grade)

course(course_id, title, dept_name, credits)

Select student.SID from student, course, takes where student.SID= takes.SID and course.course_id =takes.course_id and course.dept_name=”physics” ;

Select student.SID from (student natural join takes) join course using (course_id) where course.dept_name=”physics”

- Display the student' details who scored A or A+ grade in "DBS" course.
- student(**SID**, name, dept_name, tot_cred)
- takes(**SID**, course_id, sec_id, semester, year, **grade**)
- course(course_id, **title,** dept_name, credits)

```
select * from (student natural join takes)
        join course using (course_id)
where   grade = "A" or grade = "A+"
and     title = "DBS";
```

Rename Operation – as clause

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

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

```
select name, courseid
from instructor as  i , teaches  as t
where i.id = t.id;
```

Another usage of rename operation is a case where we wish to compare tuples in the same relation.

q. Find the names of all instructors who have a higher salary than at least one instructor in 'Comp. Sci'.

Solution:

```
select distinct T.name
 from instructor as T, instructor as S
  where T.salary > S.salary and S.dept_name= 'Comp. Sci';
```

Also known as Correlation name, correlation variable, tuple variable, table alias

```
NAME
--------------------
Brandt
Einstein
Singh
Wu
Kim
Gold
Crick
Katz

8 rows selected.
```

String operation: like

- Strings are specified in single quotes. Ex: 'Computer'
- SQL includes a string-matching operator for comparisons on character strings. The operator "like" uses patterns that are described using two special characters
  - ✓ percent (%): The % character matches any substring
  - ✓ underscore (_): The _ character matches any character

- Like ‘_ _ _’ matches exactly string of length 3
- Like ‘_ _ _% ‘ matches atleast string of length 3
- Like ‘%comp%’ matches substring comp
- Like ‘intro% string starts with intro

➢Find the names of all instructors whose name includes the substring "dar"

```
select name
from instructor
where name like ‘%dar%’;
```

String Operations

- Match the string "100 %"

  like ‘100\% ‘ **escape character** ‘ \ '

- Match the string “ab%cdffffff”
- **like** ‘ab\%cd%'
- matches all strings beginning with "ab\cd".
- **like** ‘ab\\cd%'

Ordering the Display of Results

- The **order by** clause causes the tuples in the result of a query to appear in sorted order.

➢ List all the instructors who work in Physics department in ascending

```
SELECT  name
FROM instructor
WHERE   Dept_name='Physics'
ORDER BY  name
```

- Specify **desc** for descending order or **asc** for ascending order, for each attribute

```
SQL> select * from 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

ID    NAME                 DEPT_NAME              SALARY
----- -------------------- -------------------- ----------
98345 Kim                  Elec. Eng.              80000
```

SELECT name
FROM instructor
WHERE Dept_name='Physics'
ORDER BY name

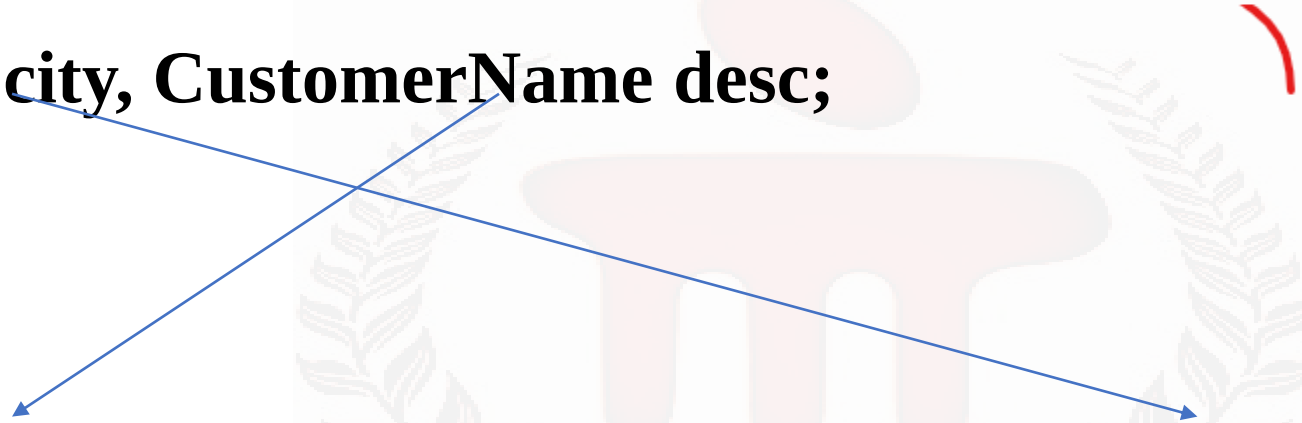
```
NAME
--------------------
Einstein
Gold
```

SELECT * FROM Customers
ORDER BY city, CustomerName ;

Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 5NT	UK
Consolidated Holdings	Elizabeth Brown	Berkeley Gardens 12 Brewery	London	WX1 6LT	UK
Eastern Connection	Ann Devon	35 King George	London	WX3 6FW	UK
North/South	Simon Crowther	South House 300 Queensbridge	London	SW7 1RZ	UK
Seven Seas Imports	Hari Kumar	90 Wadhurst Rd.	London	OX15 4NB	UK

SELECT * FROM Customers

ORDER BY city, CustomerName desc;



Seven Seas Imports	Hari Kumar	90 Wadhurst Rd.	London	OX15 4NB	UK
North/South	Simon Crowther	South House 300 Queensbridge	London	SW7 1RZ	UK
Eastern Connection	Ann Devon	35 King George	London	WX3 6FW	UK
Consolidated Holdings	Elizabeth Brown	Berkeley Gardens 12 Brewery	London	WX1 6LT	UK
B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 5NT	UK
Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

WHERE Clause Predicate: Between - And

- 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

```
NAME
--------------------

Wu
Einstein
Brandt
```

- Tuple comparison

```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
```

- All SQL platforms may not support this syntax

Aggregate Functions

- Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value
- SQL offers five built-in aggregate functions
  - ✓ Average: **avg**
  - ✓ Minimum: **min**
  - ✓ Maximum: **max**
  - ✓ Total: **sum**
  - ✓ Count: **count**
- Input to **sum** and **avg** must be a collection of numbers, but the other operators can operate on collections of nonnumeric data types, such as strings, as well.

Aggregate Functions

- Find the average salary of instructors in the Computer Science department
  - select avg (salary)
    from instructor
    where dept_name= 'Comp. Sci.';

```
AVG(SALARY)
-----------
 77333.3333
```

- Find the total number of instructors who teach a course in the Spring 2010 semester
  - select count (distinct ID) [Distinct: Since same teacher teaching more than one subject in Spring 10]
    from teaches
    where semester = 'Spring' and year = 2010

```
COUNT(DISTINCT(ID))
-------------------
                  6
```

- Find the number of tuples in the course relation
  - select count (*) from course;

- There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples **- group by** clause

“group by” clause

There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples; we specify this wish in SQL using the **group by** clause.

- The attribute or attributes given in the **group by** clause are used to form groups.
- Tuples with the same value on all attributes in the **group by** clause are placed in one group.

Aggregate Functions

- Find the average salary of instructors in each department

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
-------------------- ----------
Comp. Sci.           77333.3333
Biology                   72000
History                   61000
Finance                   85000
Elec. Eng.                80000
Music                     40000
Physics                   91000
```

- **select** *dept_name*, **avg** (*salary*) as avg_salary **from** *instructor* **group by** *dept_name*;

Aggregate Functions

- Any attribute that is **not present in the group by clause** must appear only inside an aggregate function if it appears in the select clause, otherwise the query is treated as erroneous
- Find the number of instructors in each department who teach a course in the Spring 2020 semester

/* *erroneous query* */

select dept_name, **name**, count (distinct **id**)

from instructor **natural join** teaches

where semester = ' Spring ' **and** year = 2020

group by dept_name;

Correct query:

select *dept _name*, **count** (**distinct** *ID*)
from *instructor* **natural join** *teaches*
where *semester* = ’Spring’ **and** *year* = 2010
group by *dept_name*;

Table: Subject_Selection

Subject	Semester	Attendee
ITB001	1	John
ITB001	1	Bob
ITB001	1	Mickey
ITB001	2	Jenny
ITB001	2	James
MKB114	1	John
MKB114	1	Erica

When you use a `group by` on the subject column only; say:

```
select Subject, Count(*)
from Subject_Selection
group by Subject
```

Subject	Count
ITB001	5
MKB114	2

```
select Subject, Semester, Count(*)
from Subject_Selection
group by Subject, Semester
```

we would get this:

```
+---------+----------+-------+
| Subject | Semester | Count |
+---------+----------+-------+
| ITB001  |        1 |     3 |
| ITB001  |        2 |     2 |
| MKB114  |        1 |     2 |
+---------+----------+-------+
```

having clause

- At times, it is useful to state a condition that applies to groups rather than to tuples - **having** clause

➢ Find departments where the average salary of the instructors is more than ₹42,000

- Any attribute that is present in the **having** clause without being aggregated must appear in the **group by** clause, otherwise the query is treated as erroneous.
- Order of execution: **from, where, group by, having, order by, select**

➢

Aggregate Functions – Having Clause

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

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

DEPT_NAME	AVG_SALARY
Comp. Sci.	77333.3333
Biology	72000
History	61000
Finance	85000
Elec. Eng.	80000
Physics	91000

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

➢For each course section offered in 2009, find the average total credits (totcred) of all students enrolled in the section, if the section had at least 2 students.

student(*ID*, *name*, *dept name*, *tot cred*)
Takes (*ID*, *course id*, *sec id*, *semester*, *year*, *grade*)
Query:

ID	course_id	sec_id	semester	year	grade
00128	CS-101	1	Fall	2009	A
00128	CS-347	1	Fall	2009	A-
12345	CS-101	1	Fall	2009	C
12345	CS-190	2	Spring	2009	A
12345	CS-315	1	Spring	2010	A
12345	CS-347	1	Fall	2009	A
19991	HIS-351	1	Spring	2010	B
23121	FIN-201	1	Spring	2010	C+
44553	PHY-101	1	Fall	2009	B-
45678	CS-101	1	Fall	2009	F
45678	CS-101	1	Spring	2010	B+
45678	CS-319	1	Spring	2010	B
54321	CS-101	1	Fall	2009	A-
54321	CS-190	2	Spring	2009	B+
55739	MU-199	1	Spring	2010	A-
76543	CS-101	1	Fall	2009	A
76543	CS-319	2	Spring	2010	A
76653	EE-181	1	Spring	2009	C
98765	CS-101	1	Fall	2009	C-
98765	CS-315	1	Spring	2010	B
98988	BIO-101	1	Summer	2009	A
98988	BIO-301	1	Summer	2010	*null*

Figure 4.2 The *takes* relation.

Aggregate Functions

For each course section offered in 2009, find the average total credits (totcred) of all students enrolled in the section, if the section had at least 2 students.

student(*ID*, *name*, *dept name*, *tot cred*)

takes(*ID*, *courseid*, *secid*, *semester*, *year*, *grade*)

Query:

select courseid, semester, year, secid, avg (totcred)

from takes **natural join** student

where year >= 2009

group by courseid, semester, year, secid

having count (ID) >= 2 ;

```
SQL> select * from student1;

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

ID    NAME                 DEPT_NAME            TOT_CRED
----- -------------------- -------------------- ----------
98765 Bourikas             Elec. Eng.                  98
98988 Tanaka               Biology                    120
```

output

```
COURSE_I SEMEST       YEAR SEC_ID   AVG(TOT_CRED)
-------- ------ ---------- -------- -------------
CS-315   Spring       2010 1                   65
CS-190   Spring       2009 2                   43
CS-347   Fall         2009 1                   67
CS-101   Fall         2009 1                   65
```

```
SQL> select * from takes;

ID    COURSE_I SEC_ID   SEMEST       YEAR GR
----- -------- -------- ------ ---------- --
00128 CS-101   1        Fall         2009 A
00128 CS-347   1        Fall         2009 A-
12345 CS-101   1        Fall         2009 C
12345 CS-190   2        Spring       2009 A
12345 CS-315   1        Spring       2010 A
12345 CS-347   1        Fall         2009 A
19991 HIS-351  1        Spring       2010 B
23121 FIN-201  1        Spring       2010 C+
44553 PHY-101  1        Fall         2009 B-
45678 CS-101   1        Fall         2009 F
45678 CS-101   1        Spring       2010 B+

ID    COURSE_I SEC_ID   SEMEST       YEAR GR
----- -------- -------- ------ ---------- --
45678 CS-319   1        Spring       2010 B
54321 CS-101   1        Fall         2009 A-
54321 CS-190   2        Spring       2009 B+
55739 MU-199   1        Spring       2010 A-
76543 CS-101   1        Fall         2009 A
76543 CS-319   2        Spring       2010 A
76653 EE-181   1        Spring       2009 C
98765 CS-101   1        Fall         2009 C-
98765 CS-315   1        Spring       2010 B
98988 BIO-101  1        Summer       2009 A
98988 BIO-301  1        Summer       2010
```

Aggregation with NULL and Boolean values

- Null values, when they exist, complicate the processing of aggregate operators

  select sum (salary)

  from instructor;

  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

Table1:

```
Field1    Field2    Field3
-------------------------
  1         1         1
NULL      NULL      NULL
  2         2         NULL
  1         3         1
```

Then

```
SELECT COUNT(*), COUNT(Field1), COUNT(Field2), COUNT(DISTINCT Field3)
FROM Table1
```

Output Is:

```
COUNT(*) = 4; -- count all rows even null/duplicates

-- count only rows without null values on that field
COUNT(Field1) = COUNT(Field2) = 3

COUNT(Field3) = 2
COUNT(DISTINCT Field3) = 1 -- Ignore duplicates
```

Set Operations

- SQL operations union, intersect, and except operate on relations and correspond to the mathematical set-theory operations

➢ Find the set of all courses offered in the Fall 2021 semester

select courseid

from section

where semester = ' Fall ' **and** year = 2021;

➢ Find the set of all courses offered in the Spring 2022 semester

select courseid

from section

where semester = ' Spring ' **and** year = 2022;

Set Operations

➢ Find the set of all courses offered **either in** Fall 2021 **or in** Spring 2022 **or both**

```
select courseid
from section
where semester = 'Fall' and year = 2021
union
select courseid
from section
where semester = 'Spring' and year = 2022;
```

- The union operation automatically eliminates duplicates, unlike the select clause.
- If the duplicates are needed, then union all is to be used.

Set Operations

➢ Find courses that were offered in Fall 2021 as well as in Spring 2022 - **(Intersect)**

(**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 2021 but not in Spring 2022 **- (except or minus)**

(**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2021)
except
(**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2022)

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

Null values

- Null signifies an unknown value or that a value does not exist.
- Null values in Arithmetic operations, comparison operations, and set operations???
- The result of an arithmetic expression (involving, for example +, −, ∗, or /) is

null if any of the input values is null. For example, if a query has an expression *r.A*+ 5, and *r.A* is null for a particular tuple, then the expression result must also be null for that tuple. If salary is null

Ex: update instructor set salary=salary+100;

- SQL therefore treats as **unknown** the result of any comparison involving a *null* value.

Null Values

- **and:** The result of true **and** unknown is unknown

  false **and** unknown is false, while

  unknown **and** unknown is unknown

- **or:** The result of true **or** unknown is true

  false **or** unknown is unknown, while

  unknown **or** unknown is unknown.

- **not:** The result of not unknown is unknown.

A	B	A AND B
T	T	T
T	F	F
F	F	F
F	T	F
T	U	U
F	U	F
U	U	U

A	B	A OR B
T	T	T
T	F	T
F	F	F
F	T	T
T	U	T
F	U	U
U	U	U

IS NULL/ IS NOT NULL

ID	Student	Email1	Email2	Phone	Father
1	Stan Marsh	NULL	smarsh@gmail.com	740-097-0951	Randy Marsh
2	Butters Stotch	bstotch@spelementary.com	bstotch@gmail.com	NULL	Stephen Stotch
3	Kenny McCormick	NULL	NULL		Stuart McCormick
4	Kyle Broflovski	kbroflovski@spelementary.com	kbroflovski@gmail.com	619-722-2491	Gerald Broflovski
666	Eric Cartman	ecartman@spelementary.com	ecartman@gmail.com	740-6733-5671	NULL

- **SELECT** ID, Student,Email1,Email2
  FROM tblSouthPark **WHERE** Email1 **IS NULL AND** Email2 **IS NULL;**

ID	Student	Email1	Email2
3	Kenny McCormick	NULL	NULL

- Ex: Find all instructors who appear in the instructor relation with null values for salary.

IS NOT NULL

- Find all customers who have their address listed in the database
- SELECT CustomerName, Address
  FROM Customers
  WHERE Address IS NOT NULL;

Raj is a database programmer, has to write the query from EMPLOYEE table to search for the employee who are working in 'Sales' or 'IT' department, for this he has written the query as:
SELECT * FROM EMPLOYEE WHERE department='Sales' or 'IT';
But the query is not producing the correct output, help Raj and correct the query so that he gets the desired output.

Raj is a database programmer, He has to write the query from EMPLOYEE table to search for the employee who are not getting any commission, for this he has written the query as:

SELECT * FROM EMPLOYEE WHERE commission=null;

But the query is not producing the correct output, help Raj and correct the query so that he gets the desired output.

Nested Subquery

- 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
- Set membership is checked using **in** and **not in** constructs

Nested Subqueries

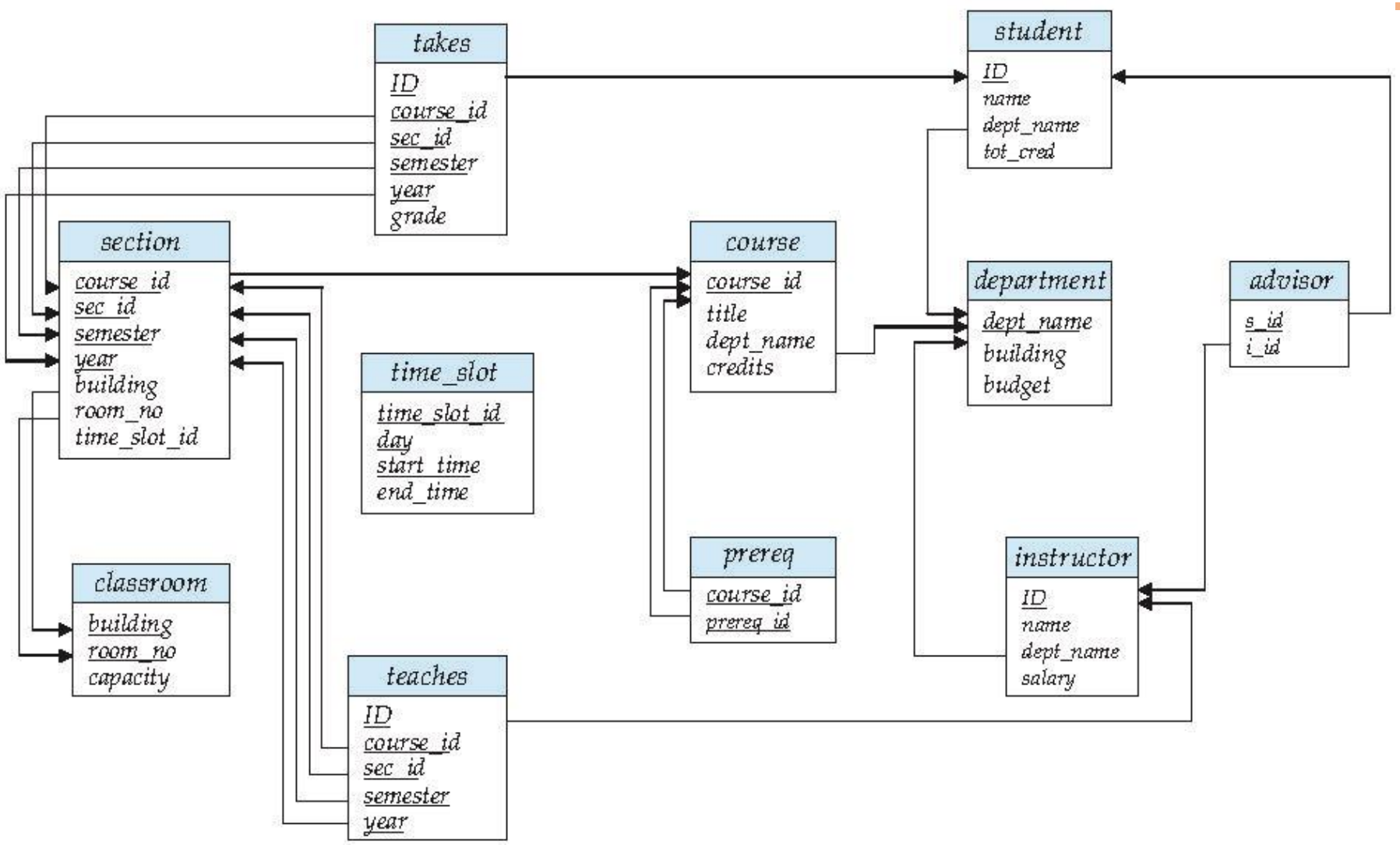
- SQL provides a mechanism for the nesting of subqueries.
- A **subquery** is a **select-from-where** expression that is nested within another query.
- A common use of subqueries is to perform tests for
  1. set membership,
  2. set comparisons,
  3. set cardinality.

- Set membership
- SQL allows testing tuples for membership in a relation. The **in** connective tests for set membership, where the set is a collection of values produced by a **select** clause.

- The **not in** connective tests for the absence of set membership.

- "Find all the courses taught in the both the Fall 2009 and Spring 2010 semesters.

- Question hint " by checking for membership….."


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

Set Membership (Operates: in, not in)

Find all the courses taught in the both the Fall 2009 and Spring 2010 semesters.

```
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);
```

- 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);
```

Example Query

- 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);
```

- The **in** and **not in** operators can also be used on **enumerated sets.** The following query selects the names of instructors whose names are neither “Mozart” nor “Einstein”.

select distinct *name*

from *instructor* **where** *name* **not in** (’Mozart’, ’Einstein’);

Nested Subquery – Set Comparison

➢ Find the names of all instructors whose salary is greater than at least one instructor in the Biology department.

- "**greater than at least one**" is represented in SQL by **>some**
- The **> some** comparison in the **where** clause of the outer **select** is true if the salary value of the tuple is greater than at least one member of the set of all salary values for instructors in Biology department.
- SQL also allows **< some, <= some, >= some, = some,** and **<> some** comparisons.

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 < at least 1 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**

Set Comparison (operators: some, all)

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

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

- Same query using > **some** clause

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

- The > **some** comparison in the **where** clause of the outer **select** is true if the *salary* value of the tuple is greater than at least one member of the set of all salary values for instructors in Biology.

- SQL also allows < **some**, <= **some**, >= **some**, = **some**, and <> **some** comparison
- = **some** is identical to **in**,
- whereas <> **some** is *not* the same as **not in**

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**

Nested Subquery – Set Comparison

- Find the names of all instructors who have a salary value greater than that of each instructor in the Biology department.

- Construct > **all** corresponds to the phrase "**greater than all**“
- SQL also allows < **all,** <= **all,** >= **all,** = **all,** and <> **all** comparisons
- <>**all** is identical to **not in**, whereas = **all** is not the same as **in**.

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');
```

As it does for **some**, SQL also allows < **all**, <= **all**, >= **all**, = **all**, and <> **all** comparisons.

As an exercise, verify that <> **all** is identical to **not in**, whereas = **all** is *not* the same as **in**.

➢Find the departments that have the highest average salary

- Ans:

select *dept name*

from *instructor*

group by *dept name*

having avg (*salary*) >= **all** (**select avg** (*salary*)

from *instructor*

group by *dept name*);

DEPT_NAME	AVG(SALARY)
Comp. Sci.	77333.3333
Biology	72000
History	61000
Finance	85000
Elec. Eng.	80000
Music	40000
Physics	91000

DEPT_NAME
Physics

Test for Empty Relation(exists/ not exists)

- The **exists** construct returns the value true if the argument subquery is nonempty
- **exists** r ⇔ r ≠ Φ and **not exists** r ⇔ r = Φ

➢ Find all courses offered in both the Fall 2009 semester and in the Spring 2010 semester

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

```
COURSE_I
--------
CS-101
```

Correlation Variables

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

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

- The above query also illustrates a feature of SQL where a correlation name from an outer query (*S* in the above query), can be used in a subquery in the **where** clause.

- What is **correlated subquery?**
- A subquery that uses a correlation name from an outer query is
- called a **correlated subquery**

- In queries that contain subqueries, a scoping rule applies for correlation

names. In a subquery, according to the rule, it is legal to use only correlation names defined in the subquery itself or in any query that contains the subquery.

- If a correlation name is defined both locally in a subquery and globally in a containing query, the local definition applies

```
mysql> SELECT * FROM EMPLOYEE;
+----------+------+----------+----------+-------+-----------+------+--------+
| SSN      | DNO  | SUPERSSN | FNAME    | LNAME | ADDRESS   | SEX  | SALARY |
+----------+------+----------+----------+-------+-----------+------+--------+
| RNSACC01 | 1    | RNSACC02 | AHANA    | K     | MANGALORE | F    | 350000 |
| RNSACC02 | 1    | NULL     | SANTHOSH | KUMAR | MANGALORE | M    | 300000 |
| RNSCSE01 | 5    | RNSCSE02 | JAMES    | SMITH | BANGALORE | M    | 500000 |
| RNSCSE02 | 5    | RNSCSE03 | HEARN    | BAKER | BANGALORE | M    | 700000 |
| RNSCSE03 | 5    | RNSCSE04 | EDWARD   | SCOTT | MYSORE    | M    | 500000 |
| RNSCSE04 | 5    | RNSCSE05 | PAVAN    | HEGDE | MANGALORE | M    | 650000 |
| RNSCSE05 | 5    | RNSCSE06 | GIRISH   | MALYA | MYSORE    | M    | 450000 |
| RNSCSE06 | 5    | NULL     | NEHA     | SN    | BANGALORE | F    | 800000 |
| RNSECE01 | 3    | NULL     | JOHN     | SCOTT | BANGALORE | M    | 450000 |
| RNSISE01 | 4    | NULL     | VEENA    | M     | MYSORE    | M    | 600000 |
| RNSIT01  | 2    | NULL     | NAGESH   | HR    | BANGALORE | M    | 500000 |
+----------+------+----------+----------+-------+-----------+------+--------+
11 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM PROJECT;
+-----+--------------------+------+------------+
| PNO | PNAME              | DNO  | PLOCATION  |
+-----+--------------------+------+------------+
| 100 | IOT                | 5    | BANGALORE  |
| 101 | CLOUD              | 5    | BANGALORE  |
| 102 | BIGDATA            | 5    | BANGALORE  |
| 103 | SENSORS            | 3    | BANGALORE  |
| 104 | BANK MANAGEMENT    | 1    | BANGALORE  |
| 105 | SALARY MANAGEMENT  | 1    | BANGALORE  |
| 106 | OPENSTACK          | 4    | BANGALORE  |
| 107 | SMART CITY         | 2    | BANGALORE  |
+-----+--------------------+------+------------+
8 rows in set (0.00 sec)
```

```
mysql> SELECT PNO FROM PROJECT WHERE DNO='8';
Empty set (0.00 sec)
```

```
ysql> SELECT E.FNAME, E.LNAME FROM EMPLOYEE E WHERE EXISTS(SELECT PNO FROM PROJECT WHERE DNO='8');
mpty set (0.00 sec)
```

```
mysql> SELECT PNO FROM PROJECT WHERE DNO='5';
+-----+
| PNO |
+-----+
| 100 |
| 101 |
| 102 |
+-----+
3 rows in set (0.00 sec)
```

```
mysql> SELECT E.FNAME, E.LNAME FROM EMPLOYEE E WHERE EXISTS(SELECT PNO FROM PROJECT WHERE DNO='5');
+----------+-------+
| FNAME    | LNAME |
+----------+-------+
| AHANA    | K     |
| SANTHOSH | KUMAR |
| JAMES    | SMITH |
| HEARN    | BAKER |
| EDWARD   | SCOTT |
| PAVAN    | HEGDE |
| GIRISH   | MALYA |
| NEHA     | SN    |
| JOHN     | SCOTT |
| VEENA    | M     |
| NAGESH   | HR    |
+----------+-------+
11 rows in set (0.00 sec)
```

Not Exists

We can test for the nonexistence of tuples in a subquery by using the **not exists** construct

```
mysql> SELECT E.FNAME, E.LNAME FROM EMPLOYEE E WHERE NOT EXISTS(SELECT PNO FROM PROJECT WHERE DNO='8');
+----------+-------+
| FNAME    | LNAME |
+----------+-------+
| AHANA    | K     |
| SANTHOSH | KUMAR |
| JAMES    | SMITH |
| HEARN    | BAKER |
| EDWARD   | SCOTT |
| PAVAN    | HEGDE |
| GIRISH   | MALYA |
| NEHA     | SN    |
| JOHN     | SCOTT |
| VEENA    | M     |
| NAGESH   | HR    |
+----------+-------+
11 rows in set (0.00 sec)
```

```
mysql> SELECT E.FNAME, E.LNAME FROM EMPLOYEE E WHERE NOT EXISTS(SELECT PNO FROM PROJECT WHERE DNO='5');
Empty set (0.00 sec)
```

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

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

o/p: NO rows selected . Since one course offered by bio is not taken by any student.

Inner query: if all the bio courses are taken by student, not exists will be true

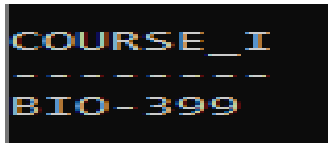
(**select** *course_id* **from** *course*
where *dept_name* = ’Biology’)

- BIO101
- BIO102

select *T.course_id* **from** *takes* **as** *T* **where T**.*ID* = S.*ID*));

- CS101
- CS102
- BIO101

- EXCEPT: BIO102(there is a biology course not taken by student)



Test for Empty Relation

- **Correlated Subquery:** A subquery that uses a correlation name from an outer query
- Result of **exists** is a Boolean value TRUE or FALSE
- The **exists** operator **terminates the processing of the subquery once the subquery returns the first row**

```
select *
from customers
where exists ( select *
               from orderdetails
               where customers.customerid = orderdetails.customerid );
```

- Will return all records from the customers table where there is at least one record in the order details table with the matching customer id

Test for Empty Relation

IN	Exists
Scans all rows returned by the sub query to conclude the result	Terminates the processing of the subquery once the sub query returns the first row
Return all rows where the attribute value is present in the subquery	Returns true of the subquery returns any rows, otherwise, it return false
select * **from** tablename **where** id **in (subquery);** **select *** **from** tablename **where** id **=** 1 OR id = 2 OR id = 3 OR id = null;	**Exists** or **Not exists** solely check the existences of rows in the sub query

Test for Absence of Duplicate Tuples

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
  - (Evaluates to “true” on an empty set)

- Find all courses that were offered at most once in 2009

```
select T.course_id
from course as T
where unique (select R.course_id
              from section as R
              where T.course_id= R.course_id
                    and R.year = 2009);
```

Q1: Get the students list who have taken at most one subject.

Without unique keyword

```
select T.course id
from course as T
where 1 <= (select count(R.course id)
from section as R
where T.course id= R.course id and
R.year = 2009);
```

"Find all courses that were offered at least twice in 2009" as follows:

select *T.course id*
from *course* **as** *T*
where not unique (**select** *R.course id*
from *section* **as** *R*
where *T.course id*= *R.course id* **and**
R.year = 2009);

Sub Queries in 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) as avg_salary
        from instructor
        group by dept_name)
where avg_salary > 42000;
```

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

Subqueries in the From Clause (Cont.)

- To display instructor, his salary along with his department's average salary.

  And yet another way to write it: **lateral** clause

```
select name, salary, avg_salary
from instructor I1,
        lateral (select avg(salary) as avg_salary
                 from instructor I2
                 where I2.dept_name= I1.dept_name);
```

- Lateral clause permits later part of the **from** clause (after the lateral keyword) to access correlation variables from the earlier part.

- Note: lateral is part of the SQL standard, but is not supported on many database systems; some databases such as SQL Server offer alternative syntax

With Clause

- The **with** clause provides a way of defining a temporary view/relation/table 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;
```

```
BUDGET
----------
120000
```

-

Complex Queries using With Clause

- With clause is very useful for writing complex queries
- Supported by most database systems, with minor syntax variations
- Find all departments where the total salary is greater than the average of the total salary at all departments

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

o/p: dept_total
Cs 50000
IT   60000
EC   70000

o/p: dept_total_avg
60000

o/p: EC department

Scalar Subquery

- Scalar subquery is one which is used where a single value is expected

- List out the number of instructors in each department.
- E.g.

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

- Runtime error if subquery returns more than one result tuple

-

- MODIFICATIONS TO THE DATABASE/ RELATIONS

Modification of database - Deletion

- Delete all tuples in the instructor relation pertaining to instructors in the Finance department
- Delete all instructors with a salary between ₹13000 and ₹15000
- Delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building
- Delete the records of all instructors with salary below the average at the university

- Performing all the tests before performing any deletion is important

- Delete from r where p;
- Delete all tuples in the instructor relation pertaining to instructors in the Finance department

  delete from instructor where deptname=‘Finance’;

Modification of database – Insertion and Updation

- Make each student in the Music department who has earned more than 144 credit hours, an instructor in the Music department, with a salary of ₹18000.

```
insert into instructor
            select ID, name, dept name, 18000
            from student
            where deptname = 'Music' and tot_cred > 144;
```

- Update totcred attribute of each student tuple to the sum of the credits of courses successfully completed by the student. Assume that a course is successfully completed if the student has a grade that is not 'F' or null

Modification of database – Insertion and Updation

```
update student S
set totcred = (select sum(credits)
        from takes natural join course
        where S.ID = takes.ID
        and takes.grade <> 'F ' and
        takes.grade is not null ) ;
```

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

Questions

1. Find the titles of courses in Comp. Sci. department that have 3 credits
2. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result
3. Find the highest salary of any instructor
4. Find all instructor earning the highest salary (there may be more than one with the same salary)
5. Find the enrollment of each section that was offered in Autumn 2021
6. Find the maximum enrollment across all sections in Autumn 2021
7. Find the sections that had the maximum enrollment in Autumn 2021
8. Increase the salary of each instructor in the Comp. Sci. department by 10%

Questions

9. Delete all courses that have never been offered (that is, do not occur in the section relation)
10. Insert every student whose total credit attribute is greater than 100 as a instructor in the same department, with a salary of ₹10000
11. Find the names of all students who have taken ateast one Comp. Sci. course; make sure there are no duplicate names in the result
12. Find the IDs and names of all students who have not taken any course offering before Soring 2021
13. For each department, find the maximum salary of the instructors in that department. You may assume that every department has atleast one instructor
14. Find the lowest, across all departments of the per-department maximum salary computed by the preceding query