1. **ER Diagram (University Schema)**
2. **Pre-requisites(Detailed University Schema)**

**Schema of University Database:**

classroom (building, room\_number, capacity)

department (dept\_name, building, budget)

course (course\_id, title, dept\_name, credits)

instructor (ID, name, dept name, salary)

section (course\_id, sec\_id, semester, year, building, room number, time\_slot\_id,day)

teaches(ID, course\_id, sec\_id, semester, year)

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

takes(ID, course\_id, sec\_id, semester, year, grade)

advisor(s\_ID, i\_ID)

time slot(time\_slot\_id, day, start\_time, END\_time)

prereq(course\_id, prereq\_id)

Create database University;

1. **“Create Table” command for creating all tables**

1. classroom table

create table University.classroom

(building varchar (15),

room\_number varchar (7),

capacity decimal (4,0),

primary key (building, room\_number));

2. department table

create table University.department

(dept\_name varchar (20),

building varchar (15),

budget decimal (12,2) check (budget > 0),

primary key (dept\_name));

3. course table

create table University.course

(course\_id varchar (8),

title varchar (50),

dept\_name varchar (20),

credits decimal (2,0) check (credits > 0),

primary key (course\_id),

foreign key (dept\_name) references department(dept\_name) on delete set null);

4. instructor table

create table University.instructor

(ID varchar (5),

name varchar (20) not null,

dept\_name varchar (20),

salary decimal (8,2) check (salary > 29000),

primary key (ID),

foreign key (dept\_name) references department(dept\_name) on delete set null);

5. timeslot table

create table University.timeslot

(time\_slot\_id varchar (4),

day varchar (1) check (day in (’M’, ’T’, ’W’, ’R’, ’F’, ’S’, ’U’)),

start\_time time,

END\_time time,

primary key (time\_slot\_id, day));

6. section table

create table University.section

(course\_id varchar (8),

sec\_id varchar (8),

semester varchar (6) check (semester in (’Fall’,

’Winter’, ’Spring’, ’Summer’)),

year decimal (4,0) check (year > 1701 and

year < 2100),

building varchar (15),

room\_number varchar (7),

time\_slot\_id varchar (4),

day varchar (1) check (day in (’M’, ’T’, ’W’, ’R’, ’F’, ’S’, ’U’)),

primary key (course\_id, sec\_id, semester, year),

foreign key (course\_id) references course(course\_id) on delete cascade,

foreign key (time\_slot\_id,day) references timeslot (time\_slot\_id, day),

foreign key (building, room\_number) references classroom(building,room\_number) on delete set null);

7. teaches table

create table University.teaches

(ID varchar (5),

course\_id varchar (8),

sec\_id varchar (8),

semester varchar (6),

year decimal (4,0),

primary key (ID, course\_id, sec\_id, semester, year),

foreign key (course\_id, sec\_id, semester, year) references section(course\_id, sec\_id, semester, year) on delete cascade,

foreign key (ID) references instructor(ID) on delete cascade);

8. student table

create table University.student

(ID varchar (5),

name varchar (20) not null,

dept\_name varchar (20),

tot\_cred decimal (3,0) check (tot\_cred >= 0),

primary key (ID),

foreign key (dept\_name) references department(dept\_name) on delete set null);

9.takes table

create table University.takes

(ID varchar (5),

course\_id varchar (8),

sec\_id varchar (8),

semester varchar (6),

year decimal (4,0),

grade varchar (2),

primary key (ID, course\_id, sec\_id, semester, year),

foreign key (course\_id, sec\_id, semester, year) references section(course\_id, sec\_id, semester, year) on delete cascade,

foreign key (ID) references student(ID) on delete cascade);

10. advisor table

create table University.advisor

(s\_ID varchar (5),

i\_ID varchar (5),

primary key (s\_ID),

foreign key (i\_ID) references instructor (ID) on delete set null,

foreign key (s\_ID) references student (ID) on delete cascade);

11. prereq table

create table University.prereq

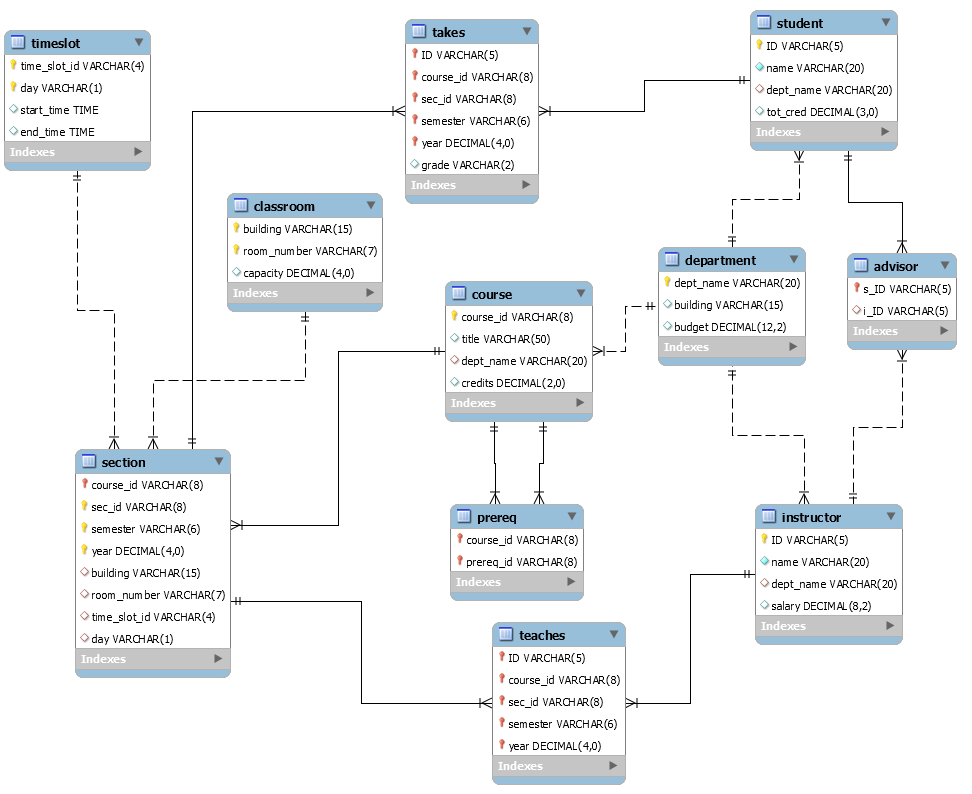
(course\_id varchar(8),

prereq\_id varchar(8),

primary key (course\_id, prereq\_id),

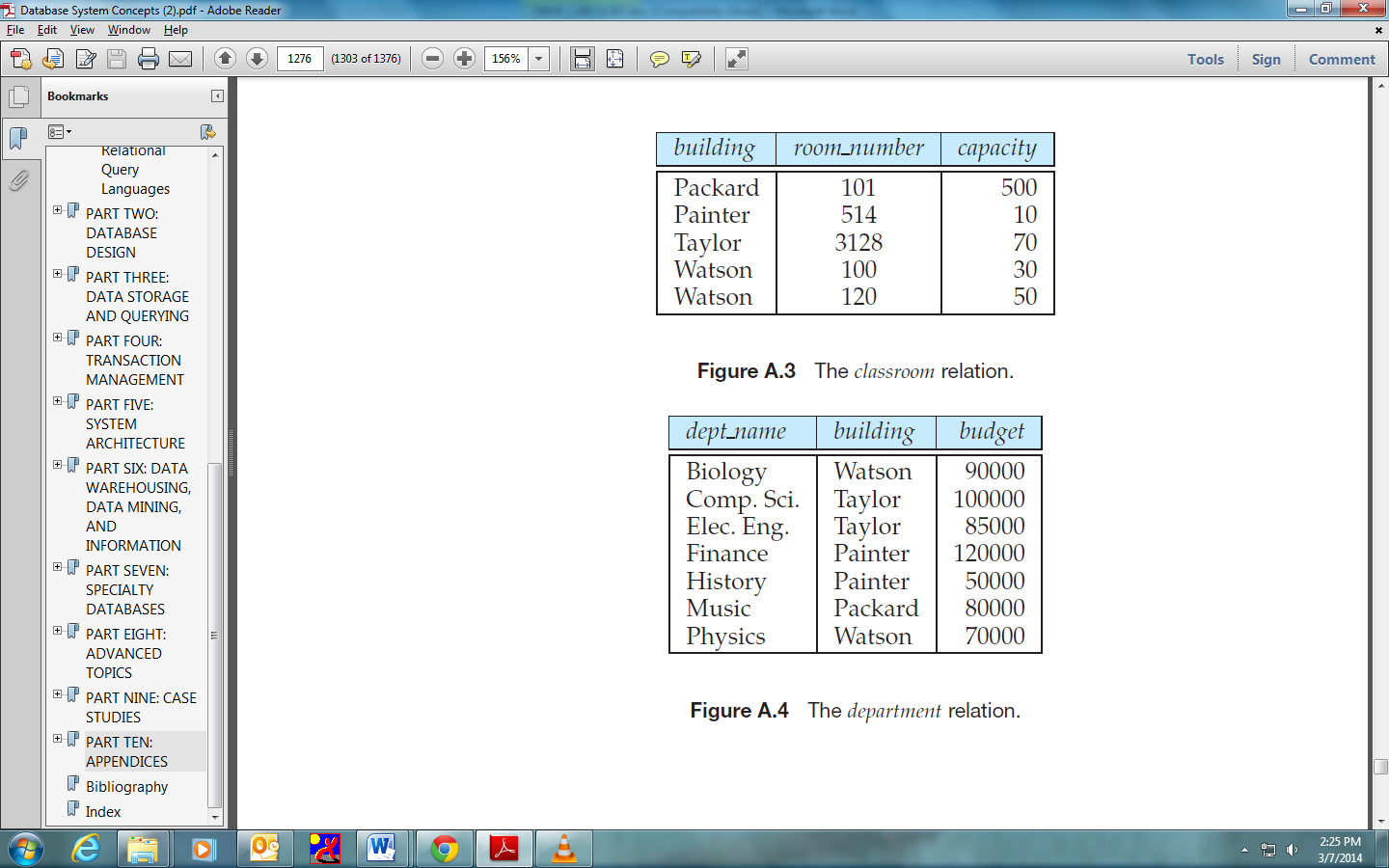
foreign key (course\_id) references course(course\_id) on delete cascade,

foreign key (prereq\_id) references course(course\_id));



1. **Sample Data for each relation(table):**

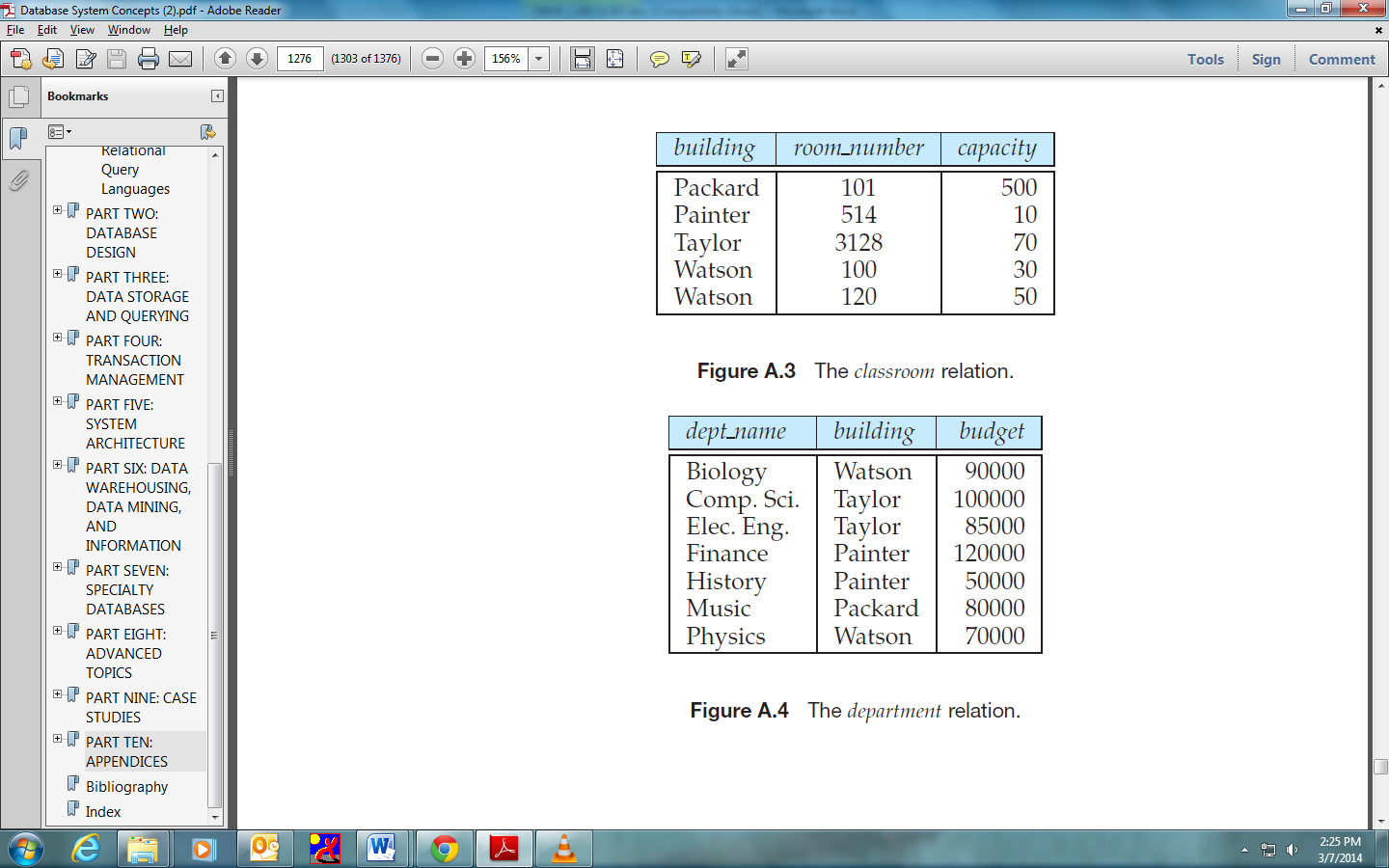
1. The classroom relation:



INSERT into classroom VALUES ('Packard','101',500);

INSERT into classroom VALUES ('Painter','514',10);

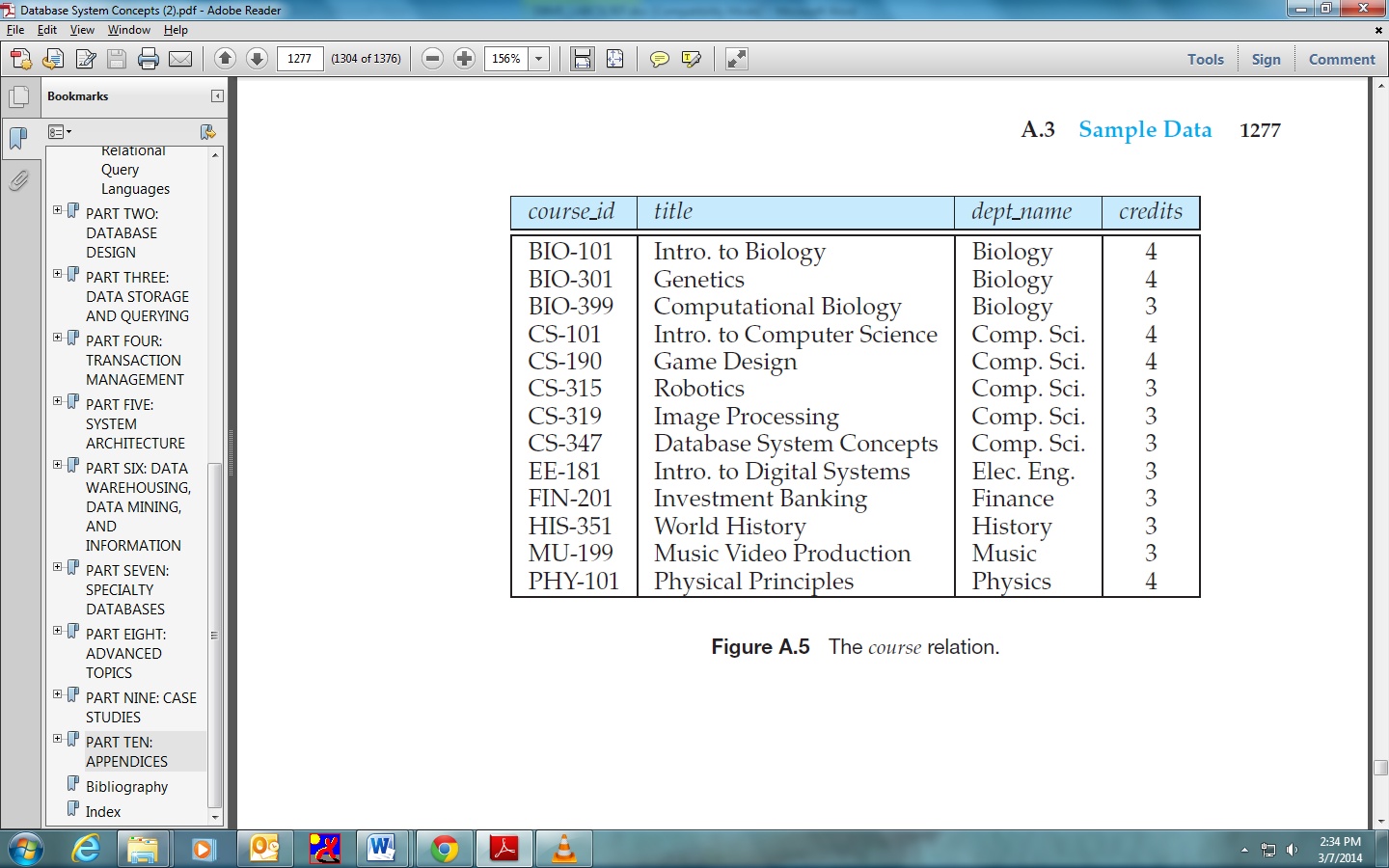
2. The department relation:



INSERT into department VALUES ('Biology','Watson',90000);

INSERT into department VALUES ('Comp.Sci','Taylor',100000);

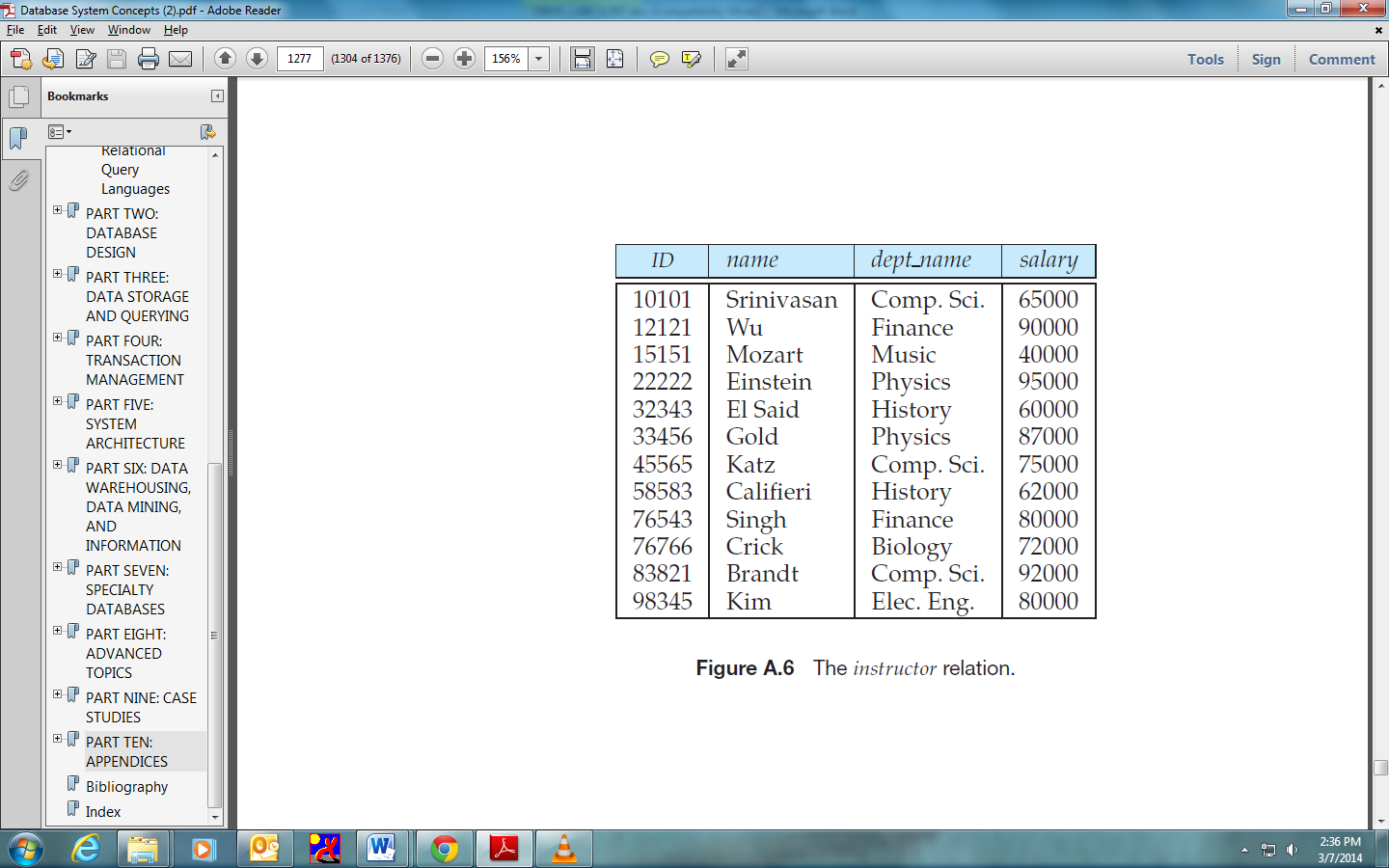
**3. The course relation:**



INSERT into course VALUES ('BIO-101','Intro. to Biology','Biology',4);

INSERT into course VALUES ('BIO-301','Genetics','Biology',4);

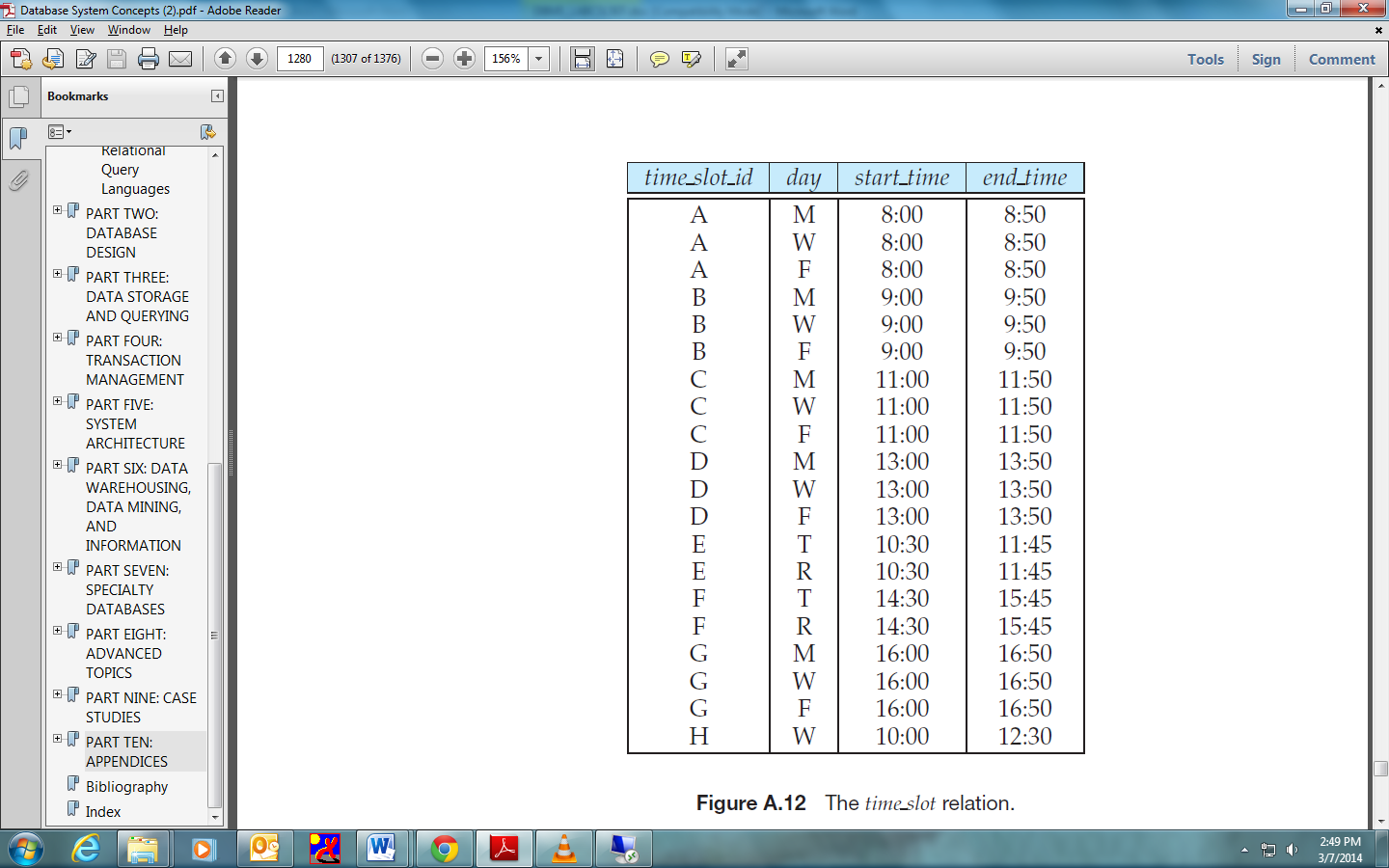
**4.The instructor relation:**



INSERT into instructor VALUES ('10101','Srinivasan','Comp.Sci',65000);

INSERT into instructor VALUES ('12121','Wu','Biology',90000);

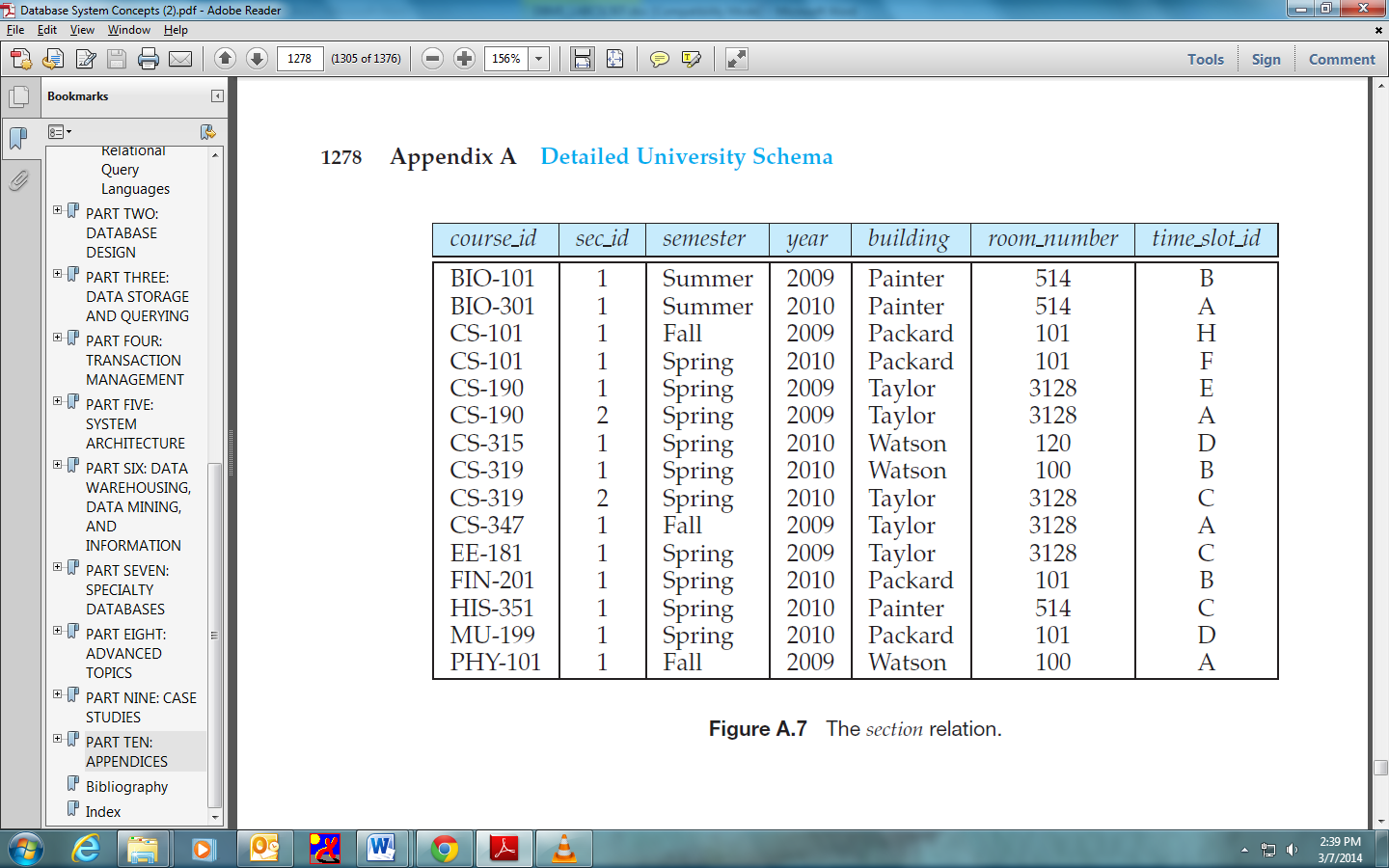
**5. The time\_slot relation:**



INSERT into timeslot VALUES ('A','M','08:00','08:50');

INSERT into timeslot VALUES ('A','W','08:00','08:50');

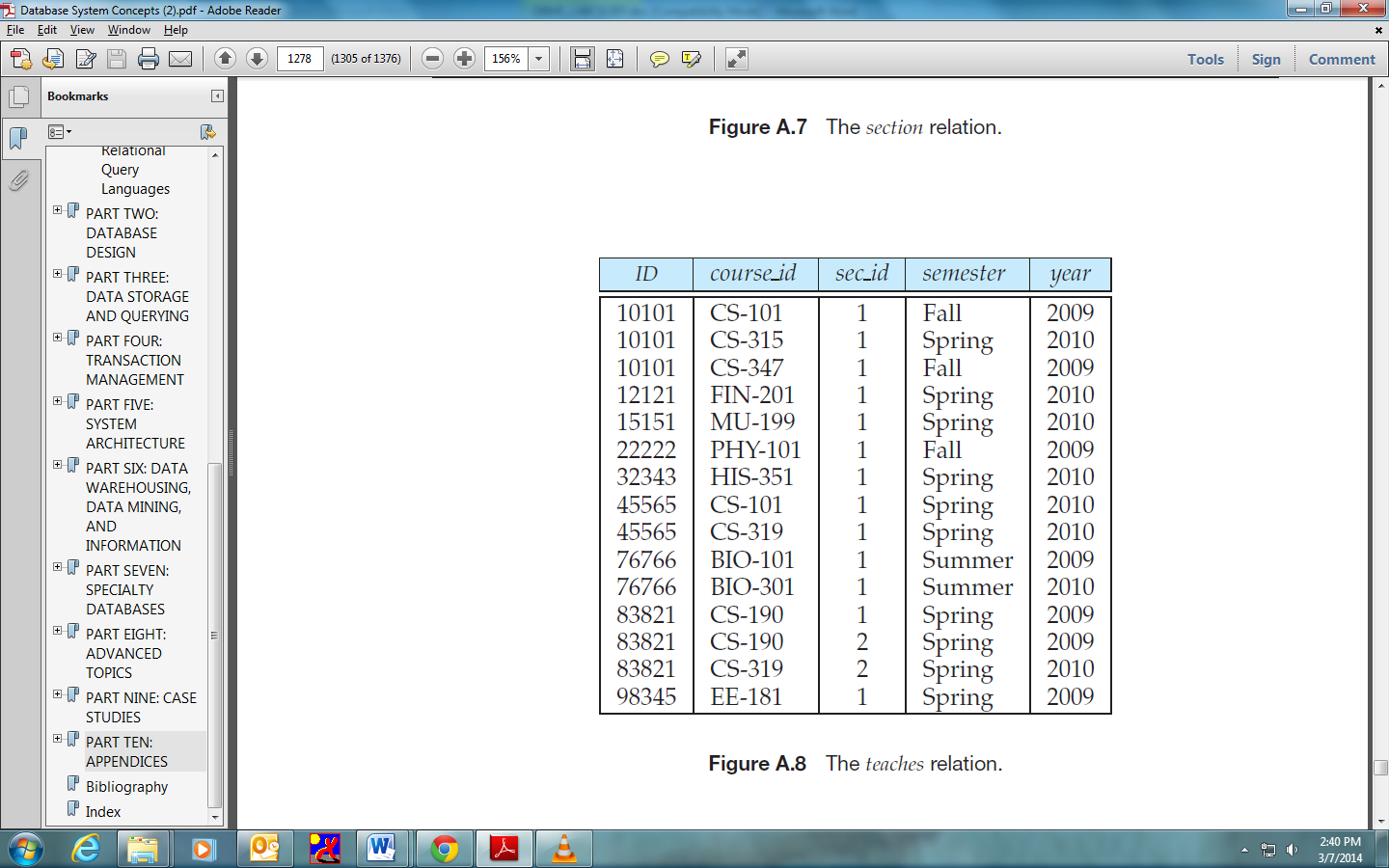
**6. The section relation:**



INSERT into section VALUES ('BIO-101','1','Summer',2009,'Painter','514','B','T');

INSERT into section VALUES ('BIO-301','1','Summer',2010,'Painter','514','A','M');

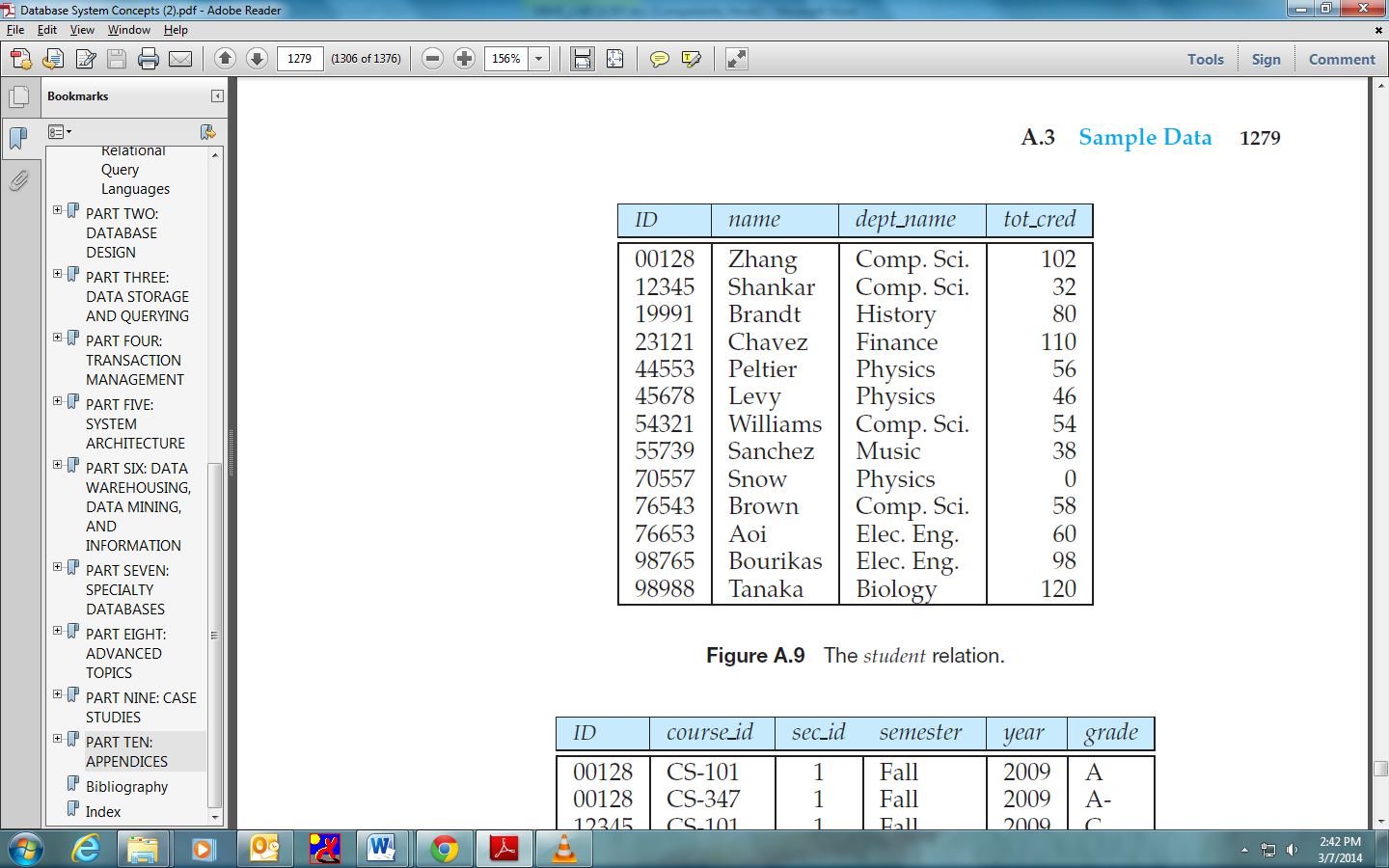
**7. The teaches relation:**



INSERT into teaches VALUES ('10101','CS-101','1','Fall',2009);

INSERT into teaches VALUES ('10101','CS-315','1','Spring',2010);

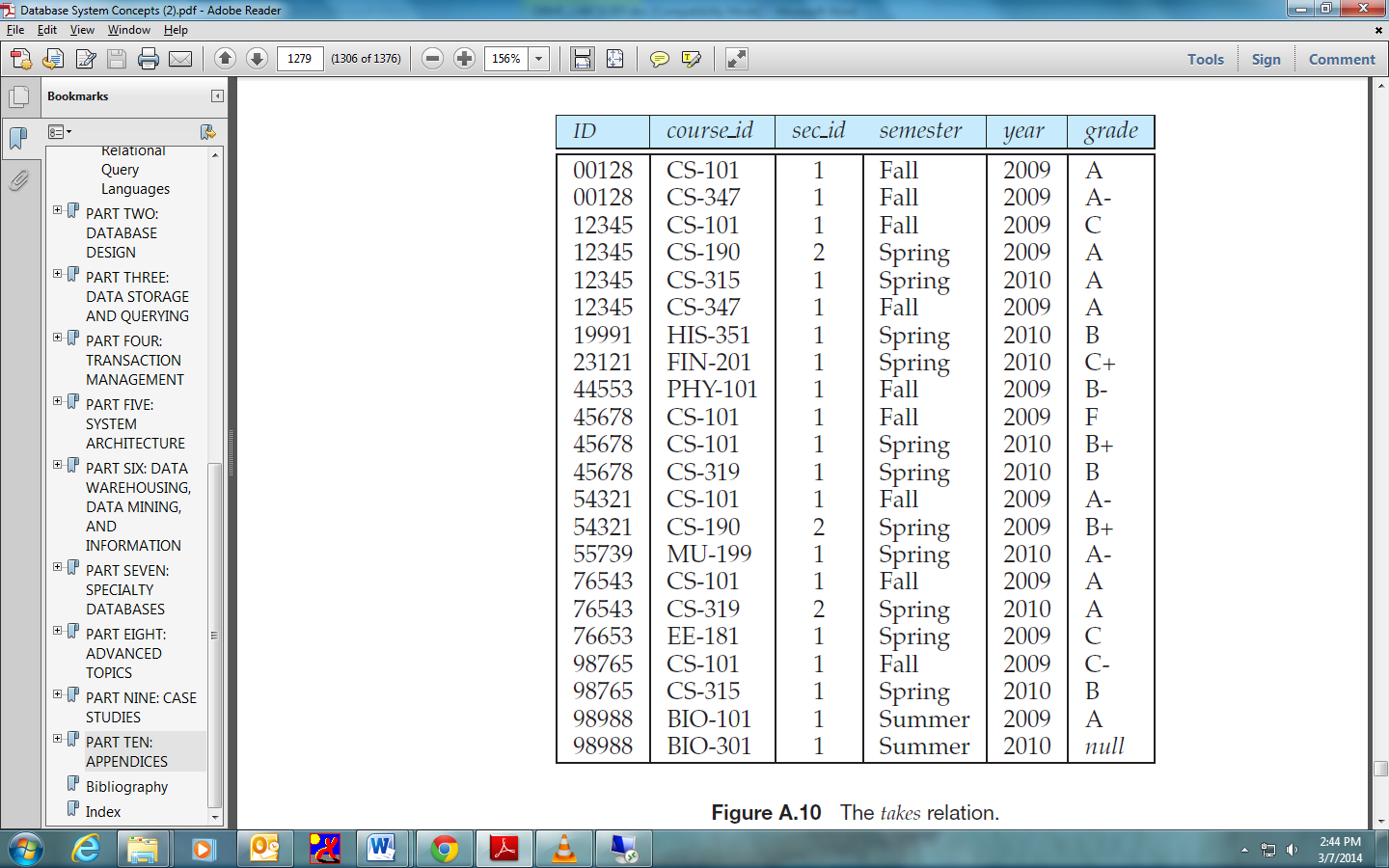
**8. The student relation:**



INSERT into student VALUES ('00128','Zhang','Comp.Sci',102);

INSERT into student VALUES ('12345','Shankar','Comp.Sci',32);

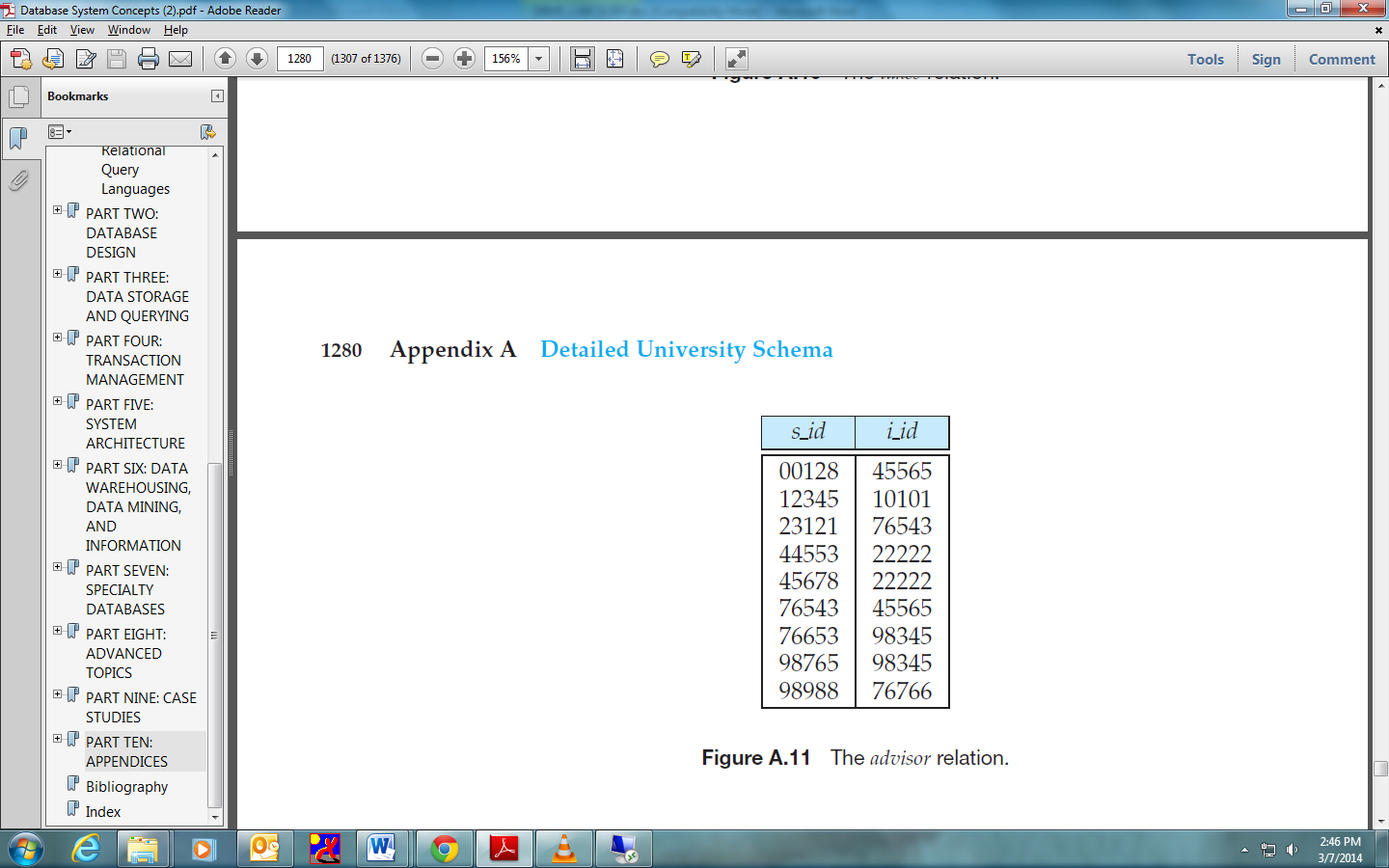
**9. The takes relation:**



INSERT into takes VALUES ('00128','CS-101','1','Fall',2009,'A');

INSERT into takes VALUES ('00128','CS-347','1','Fall',2009,'A-');

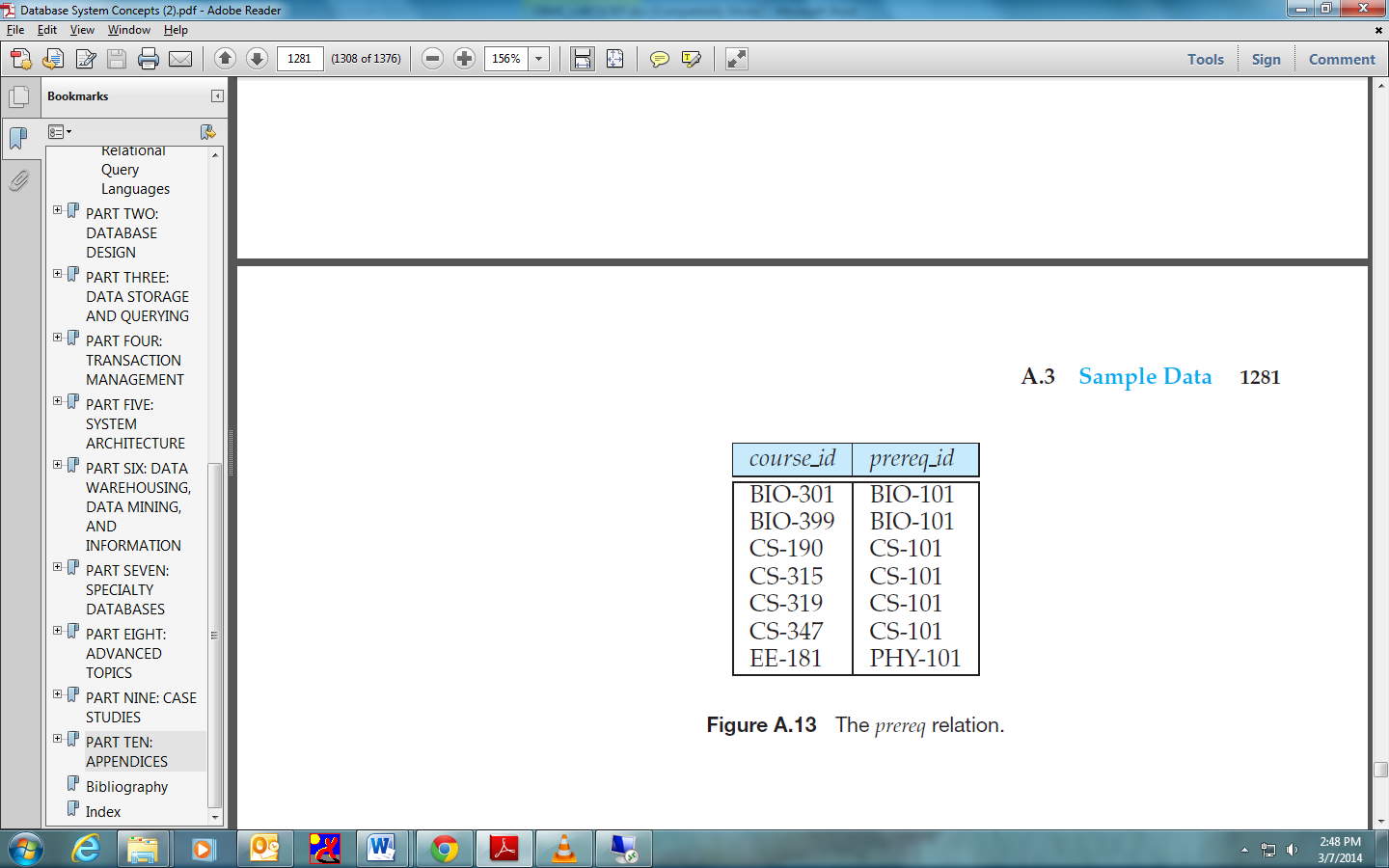
**10. The advisor relation:**



INSERT into advisor VALUES ('00128','45565');

INSERT into advisor VALUES ('12345','10101');

**11. The prereq relation:**



INSERT into prereq VALUES ('BIO-301','BIO-101');

INSERT into prereq VALUES ('BIO-399','BIO-101');

1. **Queries:**

**Operator : = , like , regexp**

1. Find the names of all the instructors FROM comp.sci. department

Solution:

SELECT name FROM instructor WHERE dept\_name = 'Comp.Sci.';



1. Print the name of instructors whose name starts with alphabet ‘E’

Solution:

SELECT name FROM instructor WHERE name like 'E%';

Or

SELECT name FROM instructor WHERE name regexp '^E';



1. Print the name of Students whose name has letters ‘an’ as a substring

Solution:

SELECT name FROM instructor WHERE name like '%an%';

Or

SELECT name FROM instructor WHERE name regexp 'an';



1. Print the name of instructors whose name ENDs with alphabet ‘h’

Solution:

SELECT name FROM instructor WHERE name like '%h';

Or

SELECT name FROM instructor WHERE name regexp 'h$';

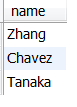


**Operator : >, <, <=,>=, != <>**

1. Find all the names of the students whose total credits are greater than 100.

Solution :

SELECT name FROM student WHERE tot\_cred > 100;

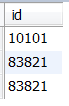


**Operator : AND, OR , IN, Between , NOT**

1. Find the list of instructors who teaches the course CS-101 or CS-190 and during fall or spring semester

Solution:

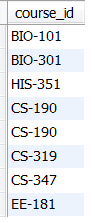
SELECT id FROM teaches WHERE (course\_id ='CS-101' or Course\_id='CS-190' ) and semester in ('Fall','Spring');



1. Print the course id WHERE the buildings are painter or Taylor and room numbers are either 514 or 3128.

Solution:

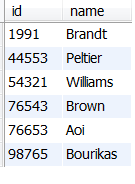
SELECT course\_id FROM section WHERE building in ('Painter','Taylor') and room\_number in (514,3128);



1. Print the course id, section id whose time slots are A,B,H

Solution:

SELECT id,name FROM student WHERE tot\_Cred between 50 and 100;

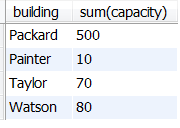


**Operator : Aggregate functions :sum ,max, min , avg, count**

1. Find the total capacity of each building in the university.

Solution:

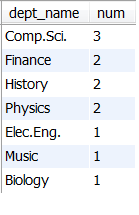
SELECT building, sum(capacity) FROM classroom GROUP BY building;



1. Find the departments that have at least one instructor, and list the names of the departments along with the number of instructors; order the result in descENDing order of number of instructors.

Solution:

SELECT dept\_name,count(\*) as num FROM instructor GROUP BY (dept\_name) ORDER BY num desc ;



1. Find the number of instructors who have never taught any course. IF the result of your query is empty, add appropriate data (and include corresponding INSERT statements) to ensure the result is not empty.

Solution:

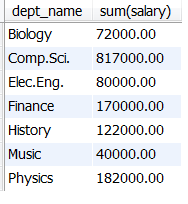
SELECT count(\*) FROM instructor WHERE ID not in (SELECT ID FROM teaches);



1. Print the dept\_name and total salary of each department

Solution:

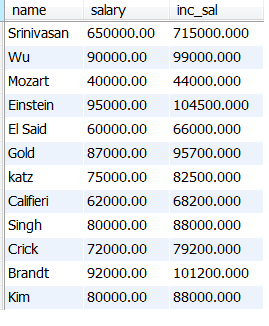
SELECT dept\_name , sum(salary) FROM instructor GROUP BY dept\_name;



1. Print the instructor name and total salary as salary + 10% hike.

Solution:

SELECT name, salary,(1.1\*salary) as inc\_sal FROM instructor;



**Operator : with Sub query**

1. Find the name of instructor who gets highest salary.

Solution:

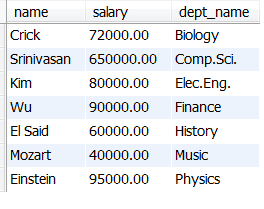
SELECT name FROM instructor WHERE salary = (SELECT max(salary) FROM instructor);



1. Find the names of the instructor who gets highest salary in each department

Solution:

SELECT name,salary,dept\_name FROM instructor WHERE (SELECT max(salary) FROM instructor) GROUP BY dept\_name;



1. Find the instructor who earn second highest salary

Solution:

SELECT name,salary

FROM university.instructor

GROUP BY salary

ORDER BY salary DESC

LIMIT 1, 1;

Or

SELECT MAX(salary) FROM instructor WHERE Salary NOT IN ( SELECT Max(Salary) FROM instructor);



1. Find the instructor who earn third highest salary

Solution:

SELECT name FROM instructor ORDER BY salary desc limit 3,1;

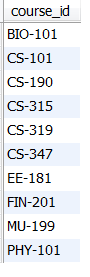


**Operator : Distinct**

1. Print the names of the courses which are taken by students.

Solution :

SELECT distinct(course\_id) FROM takes;

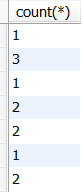


**Operator : GROUP BY**

1. Find the no of instructors in each department

Solution :

SELECT count(\*) FROM instructor GROUP BY dept\_name;



1. Find the sum of salaries of instructors , max salary and min salary of all the instructors

Solution:

SELECT sum(salary), max(salary), min(salary) FROM instructor;



**Operator : GROUP BY and HAVING**

1. Print the sum of salary in each department in which more than two instructors are working

Solution:

SELECT sum(salary) FROM instructor GROUP BY dept\_name HAVING count(\*) >2;



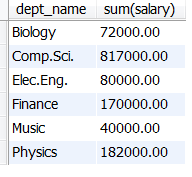
1. Print dept\_name , sum of salary of each department except History

Solution:

SELECT dept\_name, sum(salary) FROM instructor WHERE dept\_name not like 'History' GROUP BY dept\_name;

Or

SELECT dept\_name, sum(salary) FROM instructor GROUP BY dept\_name HAVING dept\_name not like 'History';



1. print the student id who has taken more than two courses in a year.

Solution:

SELECT id FROM takes GROUP BY course\_id HAVING count(\*)>2;

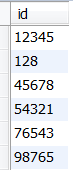


**Operator: UNION, UNION ALL, ALL**

1. Print the distinct student id IF who had taken course CS-101 or CS-190

Solution:

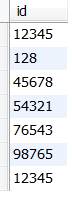
(SELECT id FROM takes WHERE course\_id='CS-101') UNION (SELECT id FROM takes WHERE course\_id='CS-190');



1. Print the student id IF who had taken course CS-101 or CS-190

Solution:

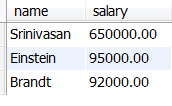
(SELECT id FROM takes WHERE course\_id='CS-101') UNION ALL (SELECT id FROM takes WHERE course\_id='CS-190');



1. Print the instructor name whose salary greater than all the instructors working in department History, Finance

Solution:

SELECT name ,salary FROM instructor WHERE salary > all (SELECT salary FROM instructor WHERE dept\_name in ('History', 'Finance'));



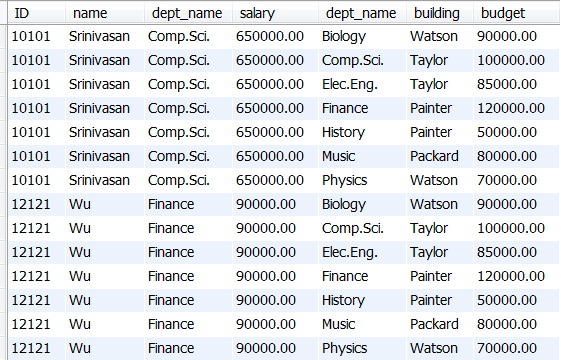
**Operators: JOINS**

**Cross join**

1. Print the list of instructors and departments in university schema

Solution:

SELECT \* FROM instructor cross join department;

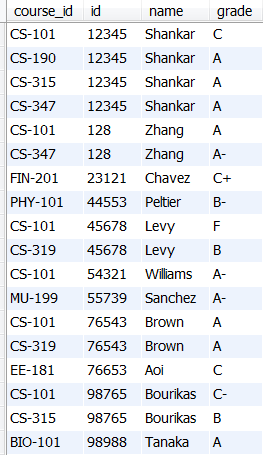


**Inner Join**

1. Find the course id and grade of all courses taken by the students

Solution:

SELECT c.course\_id, s.id,s.name, c.grade FROM takes as c, student as s WHERE c.id=s.id;



1. Find the course id and grade of all courses taken by the student name ‘tanaka’

Solution:

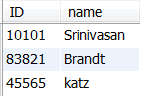
SELECT c.course\_id, c.id, c.grade FROM takes as c, student as s WHERE c.id=s.id and s.name='Tanaka';



1. Find the ID and name of instructors who have taught a course in the Comp. Sci. department, even IF they are themselves not FROM the Comp. Sci. department.

Solution:

SELECT distinct instructor.ID,name FROM teaches, instructor, course WHERE instructor.ID=teaches.ID and course.course\_id=teaches.course\_id and course.dept\_name='Comp.Sci.';



1. Find the courses which are offered in both 'Fall' and 'Spring' semester (not necessarily in the same year).

Solution:

SELECT course\_id, title FROM course WHERE exists (SELECT \* FROM takes as t WHERE t.semester='fall' and course.course\_id=t.course\_id and exists(SELECT \* FROM takes as s WHERE s.semester='spring' and course.course\_id=s.course\_id));



1. Find the name of instructors who have taught at least one course in 2009 Spring semester

Solution:

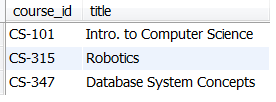
SELECT distinct name FROM instructor, teaches WHERE instructor.ID = teaches.ID and teaches.semester = 'Spring' and teaches.year = '2009';



1. Find the course id and titles of all courses taught by an instructor named 'Srinivasan'.

Solution:

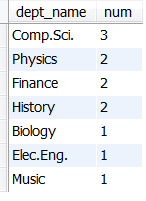
SELECT course.course\_id, title FROM teaches, instructor, course WHERE instructor.ID=teaches.ID and course.course\_id=teaches.course\_id and instructor.name='Srinivasan';



1. Find all departments that have at least one instructor, and list the names of the departments along with the number of instructors, include departments even IF they do not have any instructor, with the count as 0, order the result in descENDing order of number of instructors.

Solution:

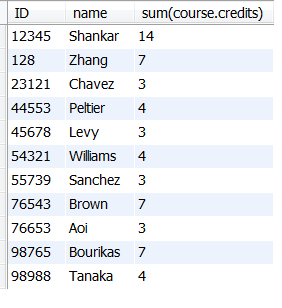
SELECT dept\_name, (SELECT count(\*)FROM instructor WHERE instructor.dept\_name = department.dept\_name) as num FROM department ORDER BY num desc ;



1. For each student, compute the total credits they have successfully completed, i.e. total credits of courses they have taken, for which they have a non-null grade other than 'F'. Do NOT use the tot\_creds attribute of student.

Solution:

SELECT takes.ID, name, sum(course.credits) FROM takes, student, course WHERE grade <> 'F' and grade is not null and takes.course\_id = course.course\_id and student.ID = takes.ID GROUP BY takes.ID, name;



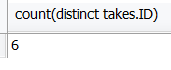
**Natural Join**

1. Find the number of students who have been taught (at any time) by an instructor named 'Srinivasan'. Make sure you count a student only once even IF the student has taken more than one course FROM Srinivasan.

Solution:

SELECT count(distinct takes.ID) FROM instructor, takes, teaches, section

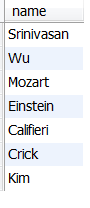
WHERE takes.course\_id = teaches.course\_id and takes.sec\_id = teaches.sec\_id and takes.semester = teaches.semester and takes.year = teaches.year and section.course\_id=teaches.course\_id and section.semester=teaches.semester and section.sec\_id=teaches.sec\_id and section.year=teaches.year and teaches.ID=instructor.ID and instructor.name='Srinivasan';



1. Find the name of all instructors who get the highest salary in their department.

Solution:

SELECT name FROM instructor I1 , (SELECT max(salary) as maxsal,dept\_name FROM instructor I2 GROUP BY dept\_name ) I2 WHERE (I2.maxsal = I1.salary and I1.dept\_name = I2.dept\_name );



1. Find all students who have taken all courses taken by instructor 'Srinivasan'. (This is the division operation of relational algebra.) You can implement it by counting the number of courses taught by Srinivasan, and for each student (i.e. GROUP BY student), find the number of courses taken by that student, which were taught by Srinivasan. Make sure to count each course ID only once.

Solution:

SELECT distinct S.ID,name FROM takes as S, student WHERE S.ID = student.ID and not exists (SELECT course\_id FROM teaches,instructor WHERE instructor.ID=teaches.ID and instructor.name='Srinivasan' and course\_id not in(SELECT course\_id FROM takes as T WHERE student.ID = T.ID));

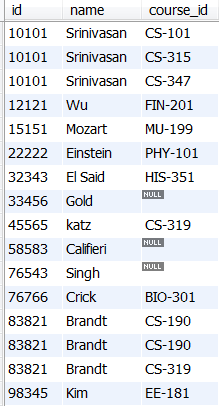


**Left Join**

1. List out the courses teaches by each instructor

Solution:

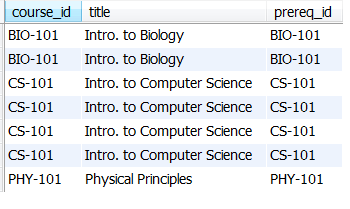
SELECT i.id,i.name, t.course\_id FROM instructor as I LEFT JOIN teaches as t on i.id=t.id;



1. List out the pre request courses of each course.

Solution:

SELECT c.course\_id,c.title,p.prereq\_id FROM course as c right join prereq as p on c.course\_id=p.prereq\_id;



**Full Outer Join**

1. List out the instructors and students details WHERE instructor is acting as a advisor.

Solution:

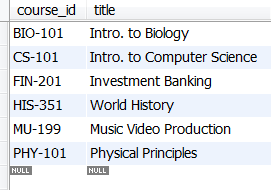
SELECT \* FROM instructor full outer JOIN advisor on instructor.id=advisor.s\_id;

**Nested Subqueries**

1. Find the id and title of all courses which do not require any prerequisites

Solution:

SELECT course\_id,title FROM course WHERE course\_id not in (SELECT course\_id FROM prereq);



1. Find the names of students who have not taken any biology dept courses.

Solution:

SELECT name FROM student WHERE not exists (SELECT \* FROM takes, course WHERE takes.course\_id=course.course\_id and course.dept\_name = 'Biology' and takes.ID = student.ID);



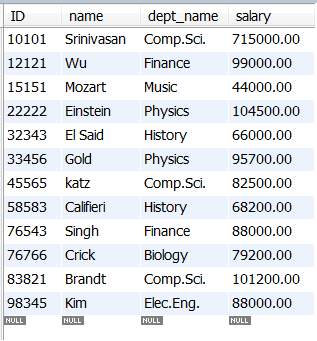
**Update Queries**

1. Give a 10% hike to all instructors

Solution:

UPDATE instructor set salary = (1.1 \* salary);

SELECT \* FROM instructor;

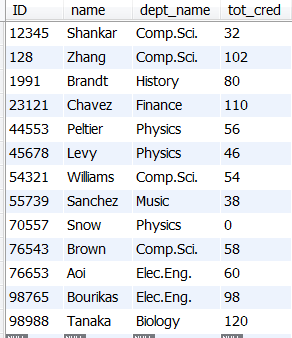


1. Increase the tot\_creds of all students who have taken the course titled “Genetics"by the number of credits associated with that course.

Solution:

UPDATE student set tot\_cred=tot\_cred + (SELECT course.credits FROM course WHERE course.title = 'Genetics') WHERE id in (SELECT id FROM takes, course WHERE takes.course\_id=course.course\_id and course.title = 'Genetics');

SELECT \* FROM student;



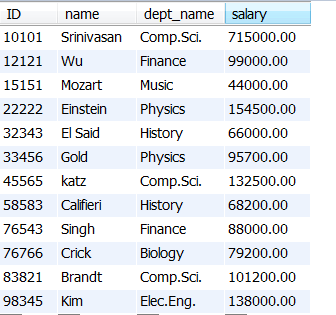
1. For all instructors who are advisors of at least 2 students, increase their salary by 50000.

Solution:

UPDATE instructor set salary = salary + 50000 WHERE instructor.ID in (SELECT i\_id FROM advisor

GROUP BY i\_id HAVING count(s\_id) >=2 );

SELECT \* FROM instructor;

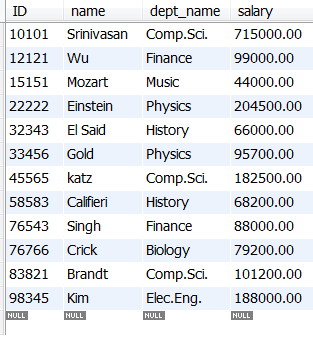


Alternate:

Solution:

UPDATE instructor set salary = salary + 50000 WHERE 2 <= (SELECT count(\*) FROM advisor WHERE i\_id = instructor.ID);

SELECT \* FROM instructor;

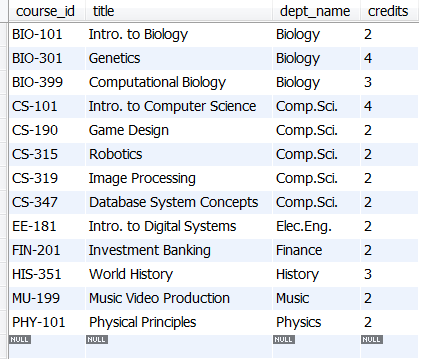


1. Set the  credits to 2 for all courses which have less than 5 students taking them (across all sections for the course, across all years/semesters).

Solution:

UPDATE course set credits=2 WHERE course\_id in (SELECT course\_id FROM takes GROUP BY course\_id HAVING count(id)<=5);

SELECT \* FROM course;



1. **Stored routines / Functions:**
2. Write a user defined function to calculate the status of the student based on his total credits scored.

IF total credit greater than 100 status= completed

IF total credit between 90 and 100 status= almost completed

IF total credit between 90 and 50 status= half completed

IF total credit less than 50 status= just started

Solution:

DELIMITER $$

CREATE FUNCTION credit\_status(totalc int) RETURNS varchar(20)

BEGIN

declare status varchar(20);

IF totalc>100 then

set status='complete';

Elseif(totalc<=90 and totalc>=50) then

set status='half';

Elseif totalc<50 then

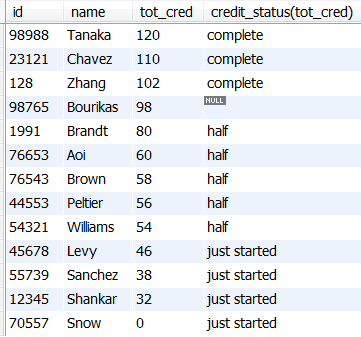
set status='just started';

END IF;

return (status);

END;

SELECT id,name,tot\_cred,credit\_status(tot\_cred) FROM student ORDER BY tot\_cred desc;



1. Write a user defined function to calculate the session time in mins for each time slot id.

Solution:

DELIMITER $$

CREATE FUNCTION session\_mins(startT varchar(8),ENDT varchar(8) ) RETURNS varchar(20)

BEGIN

declare hours, mins int ;

declare total\_mins int;

declare inMinutes varchar(20);

SELECT time\_format(timedIFf(ENDT,startT),"%i") into mins;

SELECT time\_format(timedIFf(ENDT,startT),"%H") into hours;

IF(hours>=1) then

set hours=hours\*60;

END IF;

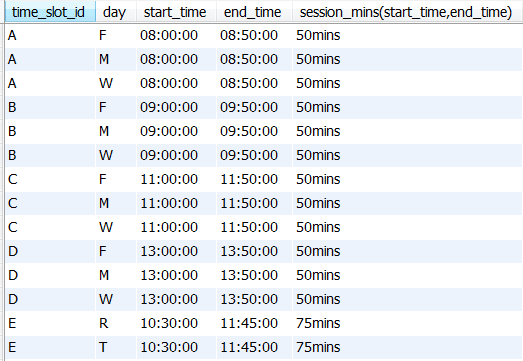
set total\_mins=hours+mins;

SELECT concat(total\_mins,'mins') into inMinutes;

return (inMinutes);

END;

SELECT time\_slot\_id, day, start\_time, END\_time, session\_mins(start\_time,END\_time) FROM timeslot;



1. **Stored Procedure**
2. Write a stored procedure to display all the data present in classroom table.

Solution:

Delimiter $$

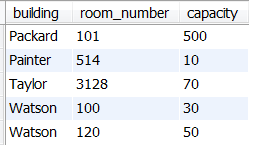
Create procedure University.getClassDetails ()

Begin

SELECT \* FROM classroom;

END

call university.getClassDetails();



1. Write a stored procedure to display a particular department (department name to be given as input parameter)

Solution:

DELIMITER //

CREATE PROCEDURE GetDepartmentDetails(IN DName VARCHAR(255))

BEGIN

SELECT \*

FROM Department

WHERE dept\_name = DName;

END //

DELIMITER ;

Call GetDepartmentDetails('Finance');



1. Write a Stored Procedure to INSERT VALUES into the table “classroom”

Solution:

DELIMITER //

CREATE PROCEDURE INSERTClassroomDetails(IN Bno VARCHAR(25), IN roomNo Varchar(25), IN cap INT)

BEGIN

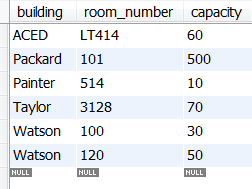
INSERT into classroom VALUES (Bno, roomNo,cap);

END //

DELIMITER ;

Call INSERTClassroomDetails('ACED','LT414',60);

SELECT \* FROM university.classroom;



1. Write a Stored Procedure to UPDATE a particular value in the table “classroom”

Solution:

DELIMITER //

CREATE PROCEDURE UPDATEClassroomDetails(IN Bno VARCHAR(25), IN roomNo Varchar(25), IN cap INT)

BEGIN

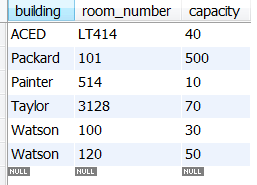
UPDATE classroom set capacity = cap WHERE building=Bno and room\_number=roomNo;

END //

DELIMITER ;

Call UPDATEClassroomDetails('ACED','LT414',40);

SELECT \* FROM university.classroom;



1. Write a Stored Procedure to Delete a particular record FROM the table “classroom”

Solution:

DELIMITER //

CREATE PROCEDURE DeleteClassroomDetails(IN Bno VARCHAR(25))

BEGIN

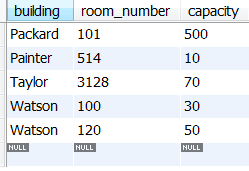
Delete FROM classroom WHERE building=Bno;

END //

DELIMITER ;

Call DeleteClassroomDetails('ACED');

SELECT \* FROM university.classroom;



1. Write a Stored procedure to display the count of number of students who took courses in 2009, spring semester and section 1

Solution:

DELIMITER //

CREATE PROCEDURE CountStudents( IN yr decimal(4,0), IN sem Varchar(10), IN sec varchar(8), OUT total INT)

BEGIN

SELECT count(course\_id)

INTO total

FROM takes

WHERE semester = sem and year=yr and sec\_id=sec;

END//

DELIMITER ;

CALL CountStudents(2009,'Spring',1,@total);

SELECT @total;



1. Write a stored procedure to validate users , if username and password are matching return valid user, if username is not matching return not a valid user or if password is not matching return not a valid password.

Solution:

DELIMITER //

CREATE PROCEDURE UpdateUser(IN gname VARCHAR(25), IN gpwd Varchar(25))

BEGIN

declare tname,tpwd varchar(25) default '';

declare temp int default 0;

select name into tname from user where name=gname;

if(tname!='') then

select count(\*) into temp from user where name=gname and password=gpwd;

if(temp!=0) then

select "valid user";

else

update user set no\_of\_attempts=no\_of\_attempts+1 where name=gname;

select "enter valid password";

end if;

else

select "enter valid user name";

end if;

END //

DELIMITER ;

1. **Views**
2. Create a view STUD\_VIEW as to Find the course id and grades of all courses taken by any student named 'Tanaka'.

Solution:

create view V\_stud as SELECT c.course\_id, c.id, c.grade FROM takes as c,student as s WHERE c.id=s.id and s.name='Tanaka';

SELECT \* FROM V\_stud;



1. **Trigger**
2. **After Insert department**

DELIMITER $$

CREATE TRIGGER university.department\_AFTER\_INSERT

AFTER INSERT ON department

FOR EACH ROW

BEGIN

INSERT INTO log\_dept (dept\_name,building,create\_date)

VALUES(NEW.dept\_name, NEW.building,NOW());

END$$

DELIMITER ;

1. **After Insert instructor**

DELIMITER $$

CREATE TRIGGER after\_instructor\_insert

AFTER INSERT ON instructor

FOR EACH ROW

BEGIN

INSERT INTO log\_instructor\_details

VALUES(NEW.ID, NEW.name, NOW());

END $$

DELIMITER ;

1. **BEFORE UPDATE student**

DELIMITER $$

CREATE TRIGGER before\_student\_update

BEFORE UPDATE

ON student FOR EACH ROW

BEGIN

Declare GRADE varchar(15);

IF NEW.tot\_cred>=90 THEN

SET GRADE = 'EXCELLENT';

ELSEIF NEW.tot\_cred>=75 AND NEW.tot\_cred<90 THEN

SET GRADE = 'VERY GOOD';

ELSEIF NEW.tot\_cred>=60 AND NEW.tot\_cred<75 THEN

SET GRADE = 'GOOD';

ELSEIF NEW.tot\_cred>=40 AND NEW.tot\_cred<60 THEN

SET GRADE = 'AVERAGE';

ELSE SET GRADE = 'NOT PROMOTED';

END IF;

END$$

DELIMITER ;

1. **AFTER UPDATE takes**

DELIMITER $$

CREATE TRIGGER after\_ takes \_update

AFTER UPDATE ON takes

FOR EACH ROW

BEGIN

Declare credit decimal(2,0);

Select credits into credit from course where course\_id=new.course\_id;

UPDATE student set tot\_cred=tot\_cred+credit where id=new.id;

END$$

DELIMITER ;

1. **BEFORE DELETE student**

DELIMITER $$

CREATE TRIGGER before\_student\_delete

BEFORE DELETE ON student

FOR EACH ROW

BEGIN

DECLARE vUser varchar(50);

-- Find username of person performing the DELETE into table

SELECT USER() INTO vUser;

-- Insert record into audit table

INSERT INTO student\_audit ( student\_id, deleted\_date, deleted\_by)

VALUES ( OLD.id, SYSDATE(), vUser );

END$$

DELIMITER ;

1. **Before insert department**

DELIMITER $$

CREATE TRIGGER before\_department\_insert

BEFORE INSERT ON departments

FOR EACH ROW

BEGIN

If new.budget> 0 then

SET new.budget=budget;

Else

Select 'enter budget > 0';

End if;

End;$$

DELIMITER ;

1. **AFTER DELETE takes**

DELIMITER $$

CREATE TRIGGER after\_takes\_delete

AFTER DELETE ON takes

FOR EACH ROW

BEGIN

Select credits from course where course\_id=new.course\_id;

UPDATE student set tot\_cred=old.tot\_cred-credit where id=new.id;

END$$

DELIMITER ;