4.5

**Answer:** We should not add credits for courses with a null grade; further to correct handling the case where a student has not completed any course, we should make sure we don’t divide by zero, and should instead return a null value.

We break the query into a subquery that finds sum of credits and sum of credit-grade-points, taking null grades into account. The outer query divides the above to get the average, taking care of divide by 0.

```sql
create view student_grades(ID, GPA) as
select ID, credit_points / decode(credit_sum, 0, NULL, credit_sum)
from ((select ID, sum(decode(grade, NULL, 0, credits)) as credit_sum,
       sum(decode(grade, NULL, 0, credits * points)) as credit_points
       from takes natural join course natural left outer join grade_points
       group by ID)
union
select ID, NULL
from student
where ID not in (select ID from takes))
```

The view defined above takes care of NULL grades by considering the credit points to be 0, and not adding the corresponding credits in credit sum.

The query above ensures that if the student has not taken any course with non-NULL credits, and has credit_sum = 0 gets a gpa of NULL. This avoids the division by 0, which would otherwise have resulted.

An alternative way of writing the above query would be to use student natural left outer join gpa, in order to consider students who have not taken any course.

4.6
create table student
  (ID varchar(5),
   name varchar(20) not null,
   dept_name varchar(20),
   tot_cred numeric(3,0) check (tot_cred >= 0),
   primary key (ID),
   foreign key (dept_name) references department
   on delete set null);

create table takes
  (ID varchar(5),
   course_id varchar(8),
   section_id varchar(8),
   semester varchar(6),
   year numeric(4,0),
   grade varchar(2),
   primary key (ID, course_id, section_id, semester, year),
   foreign key (course_id, section_id, semester, year) references section
   on delete cascade,
   foreign key (ID) references student
   on delete cascade);

create table advisor
  (i_id varchar(5),
   s_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);

create table advisor
  (i_id varchar(5),
   s_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);
4.8

a) (Courtesy to Swapnil Gawande)
select i.name, i.id
from (select * from section natural join teaches) as t,
(select * from section natural join teaches) as a natural join instructors as i,
where t.course_id = a.course_id
and t.year = a.year
and t.id = a.id
and t.semester = a.semester
and t.sec_id = a.sec_id
and ((t.room_no <> a.room_no and t.building_no = a.building_no)
or (t.room_no <> a.room_no and t.building_no <> a.building_no));

b)
create assertion A check
(not exists (select i.name, i.id
from (select * from section natural join teaches) as t,
(select * from section natural join teaches) as a natural join instructors as i,
where t.course_id = a.course_id
and t.year = a.year
and t.id = a.id
and t.semester = a.semester
and t.sec_id = a.sec_id
and ((t.room_no <> a.room_no and t.building_no = a.building_no)
or (t.room_no <> a.room_no and t.building_no <> a.building_no))
));

4.14
create view tot_credits(year, tot_credits) as
(select year, sum(credits)
from takes natural join course
group by year)

Note that this solution assumes that there is no year where students didn’t take any course, even though sections were offered.

4.18

Answer: If the authorizations are stored as a graph, a cycle will result in the authorization graph. As the public operator provides authorization to everyone, B is now authorized. And given the with grant option, B can authorize A and cause an edge from B to A in the authorization graph, leading to a cycle.

5.5

For section table, one sample trigger could look like:

create trigger T after update of section on (time_slot_id)
referencing new row as nrow
for each row
when (nrow.time_slot_id) not in
(select time_slot_id from time_slot)
begin
  rollback
end;

Note: you may also create other kind of triggers, such as:

● If the section.time_slot_id does exist in the time_slot table, do update on the responding attribute of time_slot table.
● If the update is done upon other attribute(s), the trigger might check if the time_slot_id’s existence, and then do update or rollback.

5.6

a) One possible trigger could be: (Courtesy to Ramyaswetha Chethireddy)
create trigger T after update of takes on (grade)
referencing new row as nrow
referencing old row as orow
for each row
when (nrow.grade = 'F' and nrow.grade IS NOT NULL
and (orow.grade <> 'F' or orow.grade IS NOT NULL))
or
(nrow.grade <> 'F' and nrow.grade IS NOT NULL
and (orow.grade = 'F' or orow.grade IS NULL))

begin atomic
  update student
    case (when nrow.grade = 'F' and nrow.grade is not null
         and (orow.grade <> 'F' or orow.grade is not null))
      set tot_cred = tot_cred -
        (select credits from course
         where course_id = nrow.course_id)
        where student.id = nrow.id;
    else when (nrow.grade <> 'F' and nrow.grade is not null
               and (orow.grade = 'F' or orow.grade is null))
      set tot_cred = tot_cred -
        (select credits from course
         where course_id = nrow.course_id)
        where student.id = nrow.id;
    end;

b)
create trigger T after insert on takes
refterring new row as nrwo
for each row
when nrow.grade <> 'F' and nrow.grade IS NOT NULL
begin atomic
  update student
    set tot_cred = tot_cred +
      (select credits from course
       where course_id = nrow.course_id)
      where student.id = nrow.id;
end;

c) (Note: the following is one possible assumption, other reasonable assumptions
may also make sense)
Assume there is other integrity referential constraint, such as delete on cascade, or
assertion statement enforced on this attribute.