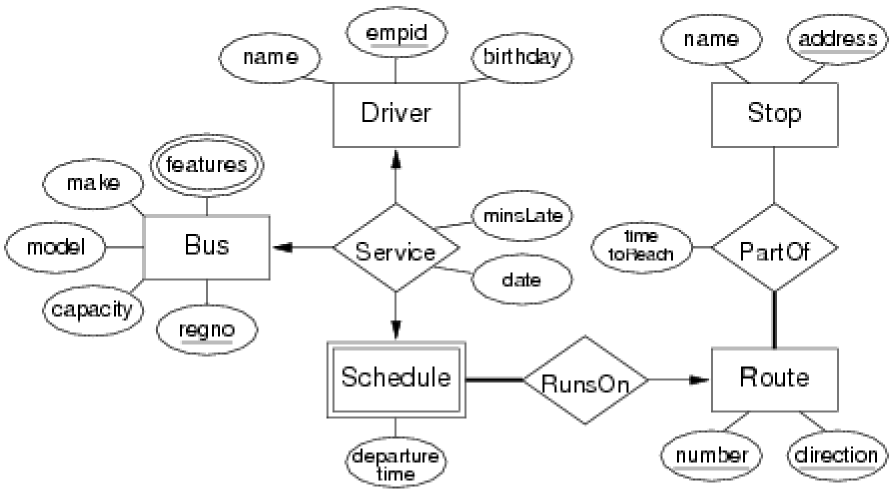
COMP9311 06s2 Final Exam
Sample Solutions

Question 1

An ER diagram for bus timetable system;

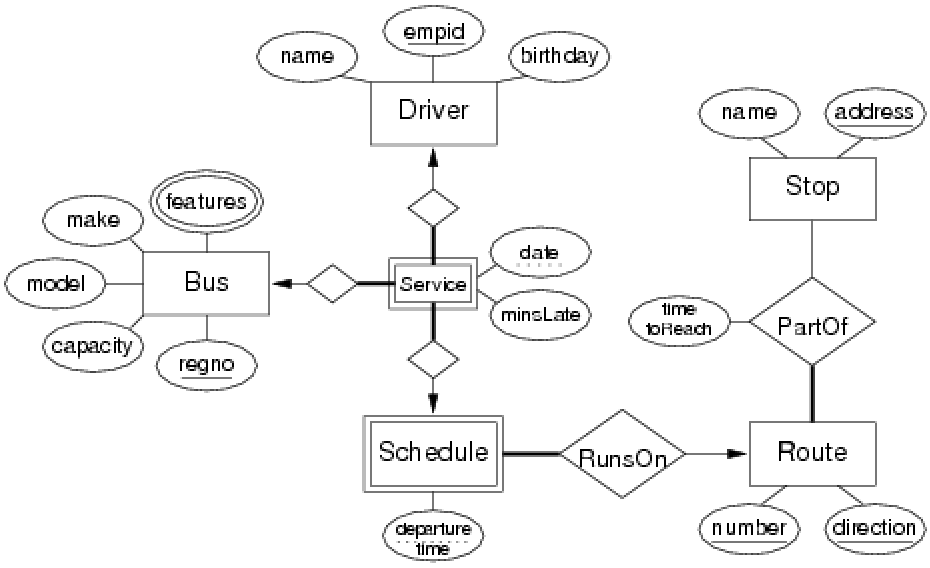
- Sample solution 1:

  The "Runs On" is an identifying relationship for weak entity "Schedule" and is to be represented by double diamond



- Sample solution 2:

  Three relationships connected to weak entity "Service" are all its identifying relations. The "Runs On" is also an identifying relationship for Schedule entity. If following the symbols used in lecture notes, all these relationships should be represented by a double diamond.



Variations:

* add an id (key) to Service and Schedule and make them strong entities

Question 2

Write create domain statements;

```
-- (a)

create domain SmallInt as integer check (value between 1 and 100);
--or
create domain SmallInt as integer check (value>=1 and value<=100);

-- (b)

create domain PosNum as numeric check (value > 0);
--or
create domain PosNum as real check (value > 0);

-- (c)

create domain Name as varchar(50);
--or
create domain Name as text check (length(value) <= 50);

-- (d)

create domain Code as char(3) check (value ~ '[A-Z][A-Z][A-Z]');
--or
create domain Code as char(3) check (value ~ '[A-Z]{3}');
--or
create domain Code as text check (value ~ '^[A-Z]{3}$');
--or ...


-- Notes:
-- * other variations are possible; allow anything reasonable
-- * if they used the domain name rather than keyword "value"
--   in the check expressions, give them B
```

Question 3

Write PostgreSQL create table statements for ER diagrams;

```
-- (a)

create table U (
        id serial primary key,
        a  SmallInt,
        f  String
);

create table UM (
        u  integer foreign key references U(id),
        m  Code,
        primary key (u,m)
);

-- (b)

create table W (
        id serial primary key,
        e  PosNum
);

create table V (
        r  integer not null,
        h  String,
        a  SmallInt,
        primary key (r,h),
        foreign key (r) references W(id)
);

-- (d)
```

```
create table P (
	id serial primary key,
	f  Name,
);

create table C1 (
	id integer primary key references P(id),
	m  Code
);
create table C2 (
	id integer primary key references P(id)
);
create table C3 (
	id integer primary key references P(id),
	g  Name
);

-- Cannot represent the disjoint subclass condition
```

Question 4

a. What is the most recent music CD made by the group 'Rolling Stones'?

```
create view StonesCDs
as
select c.* from MusicCD c, MusicGroup g
where c.madeBy = g.id and g.name = 'Rolling Stones'
;

create view Q4a
as
select title from StonesCDs
where year = (select max(year) from StonesCDs);
;

--or

create view Q4a
as
select title
from   MusicCD c, MusicGroup g
where  c.madeBy = g.id and g.name = 'Rolling Stones'
        and year = (select max(year)
                    from   MusicCD c, MusicGroup g
                    where  c.madeBy = g.id
                           and g.name = 'Rolling Stones'
                   )
;

--or

create view StonesCDs
as
select c.* from MusicCD c, MusicGroup g
where c.madeBy = g.id and g.name = 'Rolling Stones'
;

create view mostRecentStonesCDYear
as
select max(year) as year from StonesCDs
;

create view Q4a
as
select title
from   StonesCDs cd, mostRecentStonesCDYear y
```

```
where  cd.year = y.year
;
```

b. Give the name of each group and the number of CDs that the group has made.

```
create or replace view Q4b
as
select g.name, count(c.id) as numOfCDsMade
from   MusicGroup g, MusicCD c
where  g.id = c.madeBy
group by g.name
;

-- renaming the second attribute is optional
```

c. What is the title and length of the longest music CD ever made?

```
create view CDlength
as
select cd.id, cd.title, sum(s.length) as length
from   MusicCD cd, Song s
where  s.onCD = cd.id
group by cd.id, cd.title
;

create view Q4c as
select title, length
from   CDlength
where  length = (select max(length) from CDlength)
;

--or

create view CDlength
as
select onCD, sum(length) as length
from   Song
group by onCD
;

create view Q4c as
select title, length
from   CDlength cl, MusicCD cd
where  cl.onCD = cd.id and
        cl.length = (select max(length) from CDlength)
;
```

d. Which musician(s) played the most number of different instruments?

```
create view nPlayed
as
select m.id, m.name, count(p.instrument) as ninst
from   Musician m, PlaysOn p
where  m.id = p.musician
group by m.id, m.name
;

create view Q4d as
select name
from   nPlayed
where  ninst = (select max(ninst) from nPlayed)
;
```

e. Which musicians are song-writers only? (i.e. compose but don’t perform)

```
create view ComposersOnly
as
(select musician from Composer)
except
```

```
(select musician from PlaysOn);

create view q4e
as
select m.name
from   Musician m, ComposersOnly c
where  m.id = c.musician
;

-- or

create view q4e
as
select name from Musician
where id in (select * from ComposersOnly)
;

-- or

create view Musicians
as
select m.id, m.name from Musician
;

create view Composers
as
select m.id, m.name
from   Musician m, Composer c
where  m.id = c.musician
;

create view ComposersOnly
as
(select * from Composers)
except
(select * from Musicians)
;

create view Q4e as
select name from ComposersOnly
;
```

f. Which musicians have been members of more than one group during their careers?

```
create view Q4f as
select m.name
from   Musician m, Member r
where  m.id = r.musician
group by m.id
having count(distinct r.musicGroup) > 1
;
```

g. Which member(s) of the group 'White Stripes' have played drums?

```
create view Q4g as
select distinct m.name
from   Musician m, MusicGroup g, Member r, PlaysOn p
where  g.name = 'White Stripes' and r.musicGroup = g.id
        and r.musician = m.id and m.id = p.musician
        and p.instrument = 'drums'
;
```

h. Which musicians played on all songs contained on the CDs made by 'The Cure' ?

```
create view SongsOnCureCDs
as
select s.id
from   Song s, MusicGroup g, MusicCD cd
```

```
where  s.onCD = cd.id and cd.madeBy = g.id and g.name = 'The Cure'
;

create view Q4h
as
select m.name
from   Musicians m
where  not exists (
        (select id from SongsOnCureCDs)
        except
        (select s.id from SongsOnCureCDs s, PlaysOn p
         where  s.id = p.song and p.musician = m.id)
        )
;

-- some people may assume

create view SongsOnCureCDs
as
select s.id
from   Song s, MusicGroup g
where  s.performedBy = g.id and g.name = 'The Cure'
;

-- the above does not strictly answer the question => grade:B

--or

create view SongsOnCureCDs
as
select s.id
from   Song s, MusicGroup g, MusicCD cd
where  s.onCD = cd.id and cd.madeBy = g.id and g.name = 'The Cure'
;

create view Q4h
as
select m.name
from   Musicians m
where  (select count(id) from SongsOnCureCDs)
        =
       (select count(s.id) from SongsOnCureCDs s, PlaysOn p
        where  s.id = p.song and p.musician = m.id)
;
```

Question 5

Write SQL function to check whether the assertion holds or not;

```
-- Overall strategy:
-- * produce a set of CDs with more than one band
-- * check that this set is empty

create function oneGroupPerCD() returns boolean
as $$
select count(*) = 0
from   MusicCD cd, Song s
where  s.onCD = cd.id
group by cd.id
having count(distinct s.performedBy) > 1
$$ language sql;
```

Question 6

PlpgSQL function

```
create function playsOn(integer) returns setof Musician
as $$
select distinct m.*
from   Song s, PlaysOn p, Musician m
where  s.onCD = $1 and p.song = s.id and p.musician = m.id
$$ language sql;

--memberOf(Musician.id,MusicGroup.id,Year)
create function memberOf(integer, integer, integer) returns boolean
as $$
declare
        _joined integer; _departed integer;
begin
        select extract(year from m.joined),extract(year from m.departed)
                into _joined, _departed
        from   Member m
        where  m.musician = $1 and musicGroup = $2;
        if (_joined is null) then
                return false;
        elsif (_joined <= $3 and _departed is null)
                return true;
        elsif (_joined <= $3 and $3 < _departed)
                return true;
        else
                return false;
        end if;
end;
$$ language plpgsql;

create function discography(groupName text) returns text
as $$
declare
        _gid integer;
        _out text := '';  _mem text;  _non text;
        _cd record;  _mus record;
begin
        select id into _gid from MusicGroup where name = groupName;
        if (not found) then
                raise exception 'No such group'
        end if;
        for _cd in select * from MusicCD where madeBy=_gid order by year
        loop
                _out := _out||'CD: '||_cd.title||' ('||_cd.year||')\n';
                _mem := ''; _non := '';
                for _mus in select * from playsOn(_cd.id)
                loop
                        if (memberOf(_mus.id, _cd.madeBy, _cd.year)
                        then
                                _mem := ', '||_mus.name;
                        else
                                _non := ', '||_mus.name;
                        end if;
                end loop;
                _out := _out||'Group members: '
                        ||substr(_mem,2,length(_mem))||'\n';
                _out := _out||'Other musicians: '
                        ||substr(_non,2,length(_non))||'\n';
        end loop;
        return _out;
end;
$$ language plpgsql;
```

Question 7

a.
```
create trigger DisbandGroup
after update on GroupMember
for each row execute procedure disbandGroup();
```

```
create function disbandGroup() returns trigger
as $$
declare
        _nremaining integer;
begin
        if (old.departed is null and new.departed is not null)
        then
                -- this is a departing band member
                select count(*) into _nremaining
                from   GroupMember
                where  musicGroup = old.musicGroup
                        and departed is not null;
                if (_nremaining = 0)
                then
                        update MusicGroup
                        set    disbanded = new.departed
                        where  id = old.musicGroup;
                end if;
        end if;
        return new;  -- return value ignored for after-trigger
end;
$$ language plpgsql;
```

b.
```
create trigger RenameGroup
before update on MusicGroup
for each row execute procedure renameGroup();

create function renameGroup() returns trigger
as $$
declare
        _gid integer;  _mid integer;
        _today date := CURRENT_DATE;
begin
        if (old.name <> new.name)
        then
                select max(id) into _gid from MusicGroup;
                _gid := _gid + 1;
                insert into MusicGroup values
                (_gid, new.name, _today, null, old.id);
                new.name := old.name;
                new.disbanded = _today;
                for _mid in select musician from Member
                            where musicGroup = old.id
                                  and departed is null
                loop
                        update Member
                        set    departed = _today
                        where  musicGroup = old.id
                                and musician = _mid;
                        insert into Member values
                        (_gid, _mid, _today, null);
                end loop;
        end if;
        return new;  -- required
end;
$$ language plpgsql;
```

Question 8

Given a relation {LESCTR) and functional dependecies F;

F = SCE →; L, LE →; C , CET →; L, LET → CR , LECT → R , LET → C

Convert the relation into 3NF.

Solution

First we find the minimal cover as follows;

1. Decompose all dependecies so that the right hand sides contain only one attribute. We get

   F = SCE → L, LE → C , CET → L, LET → C, LET → R , LECT → R , LET → C

   Removing the duplicate dependecies (e.g; LET → C);

   F = SCE → L, LE → C , CET → L, LET → C, LET → R , LECT → R

2. Now we remove the redundant attributes.

   We don't have any redundant attribute in SCE → L. (reminder: to check whether S is redundant or not, we find the attribute closure of CE on set of functional dependencies in F. If the closure contains L then S is redundant attribute). Similar checks for C and E infer that there is no redundant attribute in this dependency.

   LE → C and CET → L also don't have any redundant attribute.

   T is redundant in LET → C because the attribute closure of LE on F contains C. We get rid of T in this dependency. Now

   F = SCE → L, LE → C , CET → L, LE → C, LET → R , LECT → R
   After removing duplicate dependecies
   F = SCE → L, LE → C , CET → L, LET → R , LECT → R

   There is no redundant attribute in LET → R.

   L and C are redundant in LECT → R because attribute closures of both ECT and LET contain R. We can delete anyone of these two attributes (note: you cannot delete both together). Let's delete L and the set of functional dependecies F is now

   F = SCE → L, LE → C , CET →; L, LET → R , ECT → R

   Now there is no redundant attribute in ECT → R or any other dependency.
3. The final step is to remove redundant dependencies;

   SCE → L is not redundant because the closure of SCE on {F - SCE → L} doesn't contain L.

   LE → C is not redundant because the closure of LE on {F - LE → C} doesn't contain C.

   CET → L is not redundant because the closure of CET on {F - CET → L} doesn't contain L.

   LET → R is redundant because the closure of LET on {F - LET → R} contains R. We delete this dependency and get

   F = SCE → L, LE → C , CET → L, ECT → R

   ECT → R is not redundant because the closure of ECT on {F - ECT → R} doesn't contain R. So the minimal cover is

   F = SCE → L, LE → C , CET → L, ECT → R

   We use additivity to join the right hand sides of the dependencies that have same left hand sides and get Fc;

$Fc = SCE \rightarrow L, LE \rightarrow C , CET \rightarrow LR$

To decompose into 3NF, we create three relations as follows;

R1 = LSCE, R2= LEC, R3 = LECTR

Since R2 is a subset of R1, we can delete this table and are left with two relations;

R1 = LSCE, R3 = LECTR

Now we need to check whether some relation contains the primary key of the large relation (containing all attributes LESCTR).
First we check relation R1; the closure of SCE (the primary key of R1) doesn't contain R and T so R1 doesn't have a primary key for the large relation.
Now we check relation R2; the closure of ECT (the primary key of R2) doesn't contain S so it also doesn't have primary key. This means, we need to add a new relation that contains any primary key of the large relation.

One of the primary keys of the relation is LEST. So we add another relation R4 that contains LEST. So the final decomposition is

R1 = LSCE, R3 = LECTR, R4= LEST