q1

(a) iii, (b) ii, (c) i, (d) ii, (e) iii,
(f) iv (g) iv, (h) i, (i) iv, (j) iii

q2a

See the solution for Lecture Exercises #2

q2b

Person(pid, name, street, city)
Employee(pid, eid, salary, phone, WorksForBranch)
Customer(pid, cid)
CustomerDate(pid, cid, date, reason)
Branch(bname, city, ManagerEmployee)
Account(acctNo, balance, HeldAtBranch)
CustomerAccount(cid, acctNo)
Transaction(cid, acctNo, type, access_date, amount)

q2c

```
create table Person (
        pid      integer,
        name     varchar(40), -- assume that 40 chars is long enough
        street   varchar(50), -- assume that 50 chars is enough
        city     varchar(20), -- assume that 20 chars is enough
        primary  key (pid)
);

create table Employee (
        pid      integer foreign key references Person(pid),
        eid      integer,
        salary   real check (salary > 0),
        phone    varchar(20), -- for flexibility
        worksIn  varchar(30) foreign key references Branch(bname),
        primary  key (pid)
);

create table Customer (
        pid      integer foreign key references Person(pid),
        cid      integer,
        primary  key (pid)
);

create table CustomerDates (
        pid      integer foreign key references Person(pid),
        when     date, -- date (ignoring year, since we assume recurrent)
        reason   varchar(40), -- why it's important
        primary  key (pid,when,reason)
);

create table Branch (
        bname    varchar(40), -- assume that 40 chars is long enough
        city     varchar(20), -- assume that 20 chars is enough
        manager  integer foreign key references Employee(pid),
        primary  key (bname)
);

create table Account
        acctNo   integer,
        balance  real, -- money value
        heldAt   varchar(40) foreign kley references Branch(bname)
);
```

```
create table CustomerAccount (
        customer integer foreign key references Customer(pid),
        account  integer foreign key references Account(acctNo),
        primary  key (customer,account)
);

create table Transaction (
        customer integer foreign key references Customer(pid),
        account  integer foreign key references Account(acctNo),
        type     varchar(10) check type in ('deposit',withdrawal'),
        accessed timestamp, -- maybe need more accuracy than date
        amount   real, -- money value
        primary  key (customer,account,type,accessed)
);


q3a

select e.ename, e.street, e.city
from   Employee e, Works w
where  e.ename = w.employee and w.company = 'First Bank Corp'

q3b

select e.ename
from   Employee e, Works w, Company c
where  e.ename = w.employee and e.city = c.city;

q3c

update Employee
set    salary = salary*1.10
where  ename in
        (select ename
         from   Manages m, Works w
         where  m.manager-name = w.employee and w.company = 'First Bank Corp')

q3d

select w.employee
from   Works w
where  w.salary in
        (select max(w.salary)
         from   Works w
         where  w.compnay = 'First Bank Corp')

q3e

select c.cname
from   Company c
where  not exists
        ((select city from Company where cname = 'Small Bank Corp')
         except
         (select city from Company where cname = c.cname))

q3f

create view CoEmp as
        select company,count(employee) as numEmps
        from   Works
        group  by company;

select company
from   CoEmp
```

```
where  numEmps >= (select max(numEMps) from CoEmp);

q3g

select company
from   Company
group  by company
having avg(salary) >
        (select avg(salary) from Work where company='First Bank Corp')


q4a

NOTE: solution in PLpgSQL, not PL/SQL

create function check_order(text,integer) returns text
as '
declare
        in_isbn alias for $1;
        ncopies alias for $2;
        in_stock integer;
begin
        select * from Editions where isbn=in_isbn;
        if (not found) then
                return ''The isbn does not exist'';
        end if;
        select into in_stock storeAmount
        from Store where isbn=in_isbn;
        if (not found) then
                in_stock = 0;
        end if;
        if (in_stock < ncopies) then
                return ''Not enough copies in stock. ''
                        ## ''The shortage is ''
                        ## (ncopies - in_stock)
                        ## ''.'';
        else
                return ''Completion of this order would leave ''
                        ## (in_stock - ncopies)
                        ## '' copies of '' ## in_isbn ## ''.'';
        end if;
end;
' language 'plpgsql;


q4b

NOTE: solution in PLpgSQL, not PL/SQL
The solution exploits PostgreSQL's method for determining
which operation invoked the trigger; not relevant in Oracle.

create function receive_supply() returns trigger
as '
declare
        recvid integer;
begin
        select * from Editions where isbn = NEW.isbn;
        if (not found) then
                raise ERROR ''Invalid ISBN ''##NEW.isbn;
                return;
        end if;
        select * from Publisher where pid = NEW.publisher;
        if (not found) then
                raise ERROR ''Invalid Publisher ID ''##NEW.publisher;
                return;
```

```
        end if;

        if (TG_OP = ''INSERT'') then
                update Store
                set    storeAmount = storeAmount + NEW.amount
                where  isbn = NEW.isbn;
                NEW.received = now();
        elsif (TG_OP = 'UPDATE'') then
                if (NEW.isbn <> OLD.isbn) then
                        -- changed book
                        update Store
                        set    storeAmount = storeAmount - NEW.amount
                        where  isbn = OLD.isbn;
                        update Store
                        set    storeAmount = storeAmount + NEW.amount
                        where  isbn = NEW.isbn;
                else
                        -- changed amount
                        update Store
                        set    storeAmount = storeAmount + (NEW.amount-OLD.amount)
                        where  isbn = NEW.isbn;
                end if;
        else
                raise ERROR ''Invalid operation on Received relation'';
                return;
        end if;
end;
' language 'plpgsql';
```