Exercise 1.

part 1

```
Create table log_works_on
(
                    essn_now    CHAR (9) ,
  pno_now   INTEGER ,
  hours_now   INTEGER,
          ESSN_before char(9) ,
          PNO_before  integer ,
          hours_before integer,
          log_id serial ,
          log_time timestamp,
          Constraint log_pkey Primary key (log_id)
);

create or replace function worksOn() returns trigger as $BODY$
begin if (tg_op = 'INSERT') then insert into
log_works_on(essn_now, pno_now, hours_now,log_time)
values(new.essn, new.pno, new.hours,  now());
return new;
end if;
if (tg_op = 'UPDATE') then insert into
log_works_on(essn_now,essn_before, pno_now,pno_before, hours_now,hours_before,log_time)
values(new.essn,old.essn, new.pno, old.pno,new.hours,old.hours, now());
return new;
end if;
if (tg_op = 'DELETE')then insert into
log_works_on(essn_before, pno_before, hours_before,log_time)
```

```
values(old.essn, old.pno, old.hours, now()) ;
return new;
end if;
return null;
end;
$BODY$ language plpgsql;

select * from works_on;
select * from log_works_on;
select* from project;

create trigger log_insert
before insert on works_on for each row
execute procedure worksOn();

create trigger log_update
before update on works_on for each row
execute procedure worksOn();

create trigger log_delete
before delete on works_on for each row
execute procedure worksOn();


update works_on set pno = '3' where hours = '40';
select * from log_works_on;
```

part 2

```
select*from project;
create or replace function count_project() returns trigger as $BODY$
declare
        project_count integer;
begin
        if(tg_op = 'INSERT') then
 select count(*) into project_count from Project where dnum = new.dnum;
            if project_count >= 4
             then raise exception 'Max 5 projects for each branch  due to company rule';
              end if;
                          return new;
        end if;
        return null;
        end;
   $BODY$ language plpgsql;
create trigger log_insert
before insert on project for each row
execute procedure count_project();
select * from project;
INSERT INTO PROJECT (PNAME,
              PNUMBER,
              PLOCATION,
              DNUM)
    VALUES ('test1',
         4,
         'Stafford',
         5);

INSERT INTO PROJECT (PNAME,
```

```
            PNUMBER,
            PLOCATION,
            DNUM)
   VALUES ('test2',
        5,
        'Stafford',
        5);
INSERT INTO PROJECT (PNAME,
            PNUMBER,
            PLOCATION,
            DNUM)
   VALUES ('testPLZWORK',
        6,
        'Stafford',
        5);
```

part 3

```
create or replace function count_employee() returns trigger as $BODY$
declare
        employee_count integer;
begin
        if(tg_op = 'INSERT') then
 select count(*) into employee_count from works_on where pno = new.pno;
            if employee_count >= 3
             then raise exception 'Max 4 projects for each employee due to company rule';
             end if;
                        return new;
        end if;
        return null;
```

```
        end;
   $BODY$ language plpgsql;
create trigger log_insert_count_proj
before insert on works_on for each row
execute procedure count_employee();


INSERT INTO WORKS_ON (ESSN, PNO, HOURS)
    VALUES ('123456789', 10, 32.5);
          INSERT INTO WORKS_ON (ESSN, PNO, HOURS)
    VALUES ('123456789', 20, 32.5);
          INSERT INTO WORKS_ON (ESSN, PNO, HOURS)
    VALUES ('123456789', 30, 32.5);
                  select*from works_on;
part 4

Create table log_department
(
        dname_old       VARCHAR (20) UNIQUE,
  dnumber_old      INTEGER ,
  mgrssn_old      CHAR (9),
  mgrstartdate_old  DATE,
        dname_new       VARCHAR (20) UNIQUE,
  dnumber_new      INTEGER ,
  mgrssn_new       CHAR (9),
  mgrstartdate_new  DATE,
        log_id serial not null,
        log_time timestamp,
        Constraint log_pkey_dept Primary key (log_id)
```

```
);


create or replace function department() returns trigger as $BODY$

begin if (tg_op = 'INSERT') then insert into

log_department(dname_new,dnumber_new,mgrssn_new,mgrstartdate_new,log_time)

values('New department name: ' ||new.dname||', New department number: '||new.dnumber||',
New manager SSN: '||new.mgrssn||', New manager start date: ' ||new.mgrstartdate,  now());

return new;

end if;

if (tg_op = 'UPDATE') then insert into

log_department(dname_new,dnumber_new,
mgrssn_new,mgrstartdate_new,dname_old,dnumber_old,mgrssn_old,mgrstartdate_old,log_time)

values('Department updated. New department name: ' ||new.dname||', New department number:
'||new.dnumber||', New manager SSN: '||new.mgrssn||', New manager start date: '
||new.mgrstartdate ||', Old department name: '||old.dname||', Old department number:
'||old.dnumber||', Old manager SSN:'||old.mgrssn||', Old manager startdate: '||old.mgrstartdate,
now());

return new;

end if;

if (tg_op = 'DELETE')then insert into

log_department(dname_old,dnumber_old,mgrssn_old,mgrstartdate_old,log_time)

values('Department deleted. Old department name: '||old.dname||', Old department number:
'||old.dnumber||', Old manager SSN:'||old.mgrssn||', Old manager startdate: '||old.mgrstartdate,
now()) ;

return new;

end if;

return null;

end;

$BODY$ language plpgsql;
```

```
create trigger log_insert_dept
before insert on department for each row
execute procedure department();

create trigger log_update_dept
before update on department for each row
execute procedure department();

create trigger log_delete_dept
before delete on department for each row
execute procedure department();
select * from department;
update department set dname='Works' where dname='Research';
insert into department (dname,dnumber,mgrssn,mgrstartdate)
values('works',7,55587975,'10-5-1997');
```

Exercise 2.

5.

```
create view view_simple as
  select essn, pno, hours
  from works_on;
  select * from view_simple;
```

6.

```
 create view sum_hours as
  select pno, sum(hours)
  from works_on group by pno;

  select * from sum_hours;
```

7.

```
  create or replace view sum_empl as
```

```
select pno, essn, sum(hours*300) from works_on
group by pno, essn;

select * from sum_empl;
```

8.

```
create or replace view the_8 (
    dname, mgr_name, mgr_salary) as
select d.DNAME, e.FNAME, e.SALARY
from employee e, department d
where e.SSN = d.MGRSSN;

select * from the_8;
```

9.

```
create or replace view the_9 (
Fname, Sname, empSalary) as
select e.FNAME, c.FNAME, e.SALARY
from employee e, department d,
employee c where c.SSN = d.MGRSSN
and d.dname = 'Research';

select * from the_9;
```

We miss a few views..

Exercise 4.

UNF to 1NF

The ResearchNo act as a key for unnormalized tables.

Research (ResearchNo ,ResearchDate ,AnalysisNo, Description, Amount, Price, Total, TotalexclVAT, VAT, TotalinclVAT, Name, Street, Postcode, City)

1NF to 2NF

ResearchNo(PK), AnalysisNO(PK) are primary keys.

Research (ResearchNo ,ResearchDate)

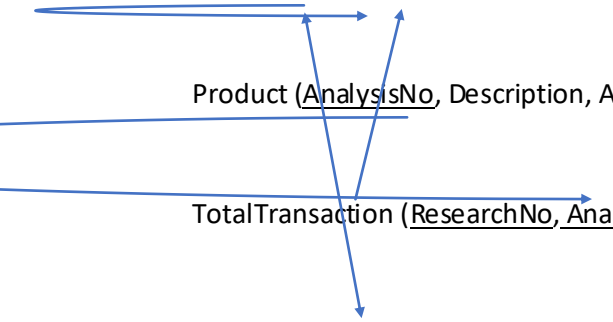
Product (AnalysisNo, Description, Amount, Price, Total, TotalexclVAT, VAT, TotalindVAT)

Costumer (RNo, Name, Street, Postcode, City)

2NF to 3NF

TotalTransaction have a transitive dependency, where TotalTransaction are dependent on both ResearchNo and AnalysisNo.

Research (ResearchNo ,ResearchDate)

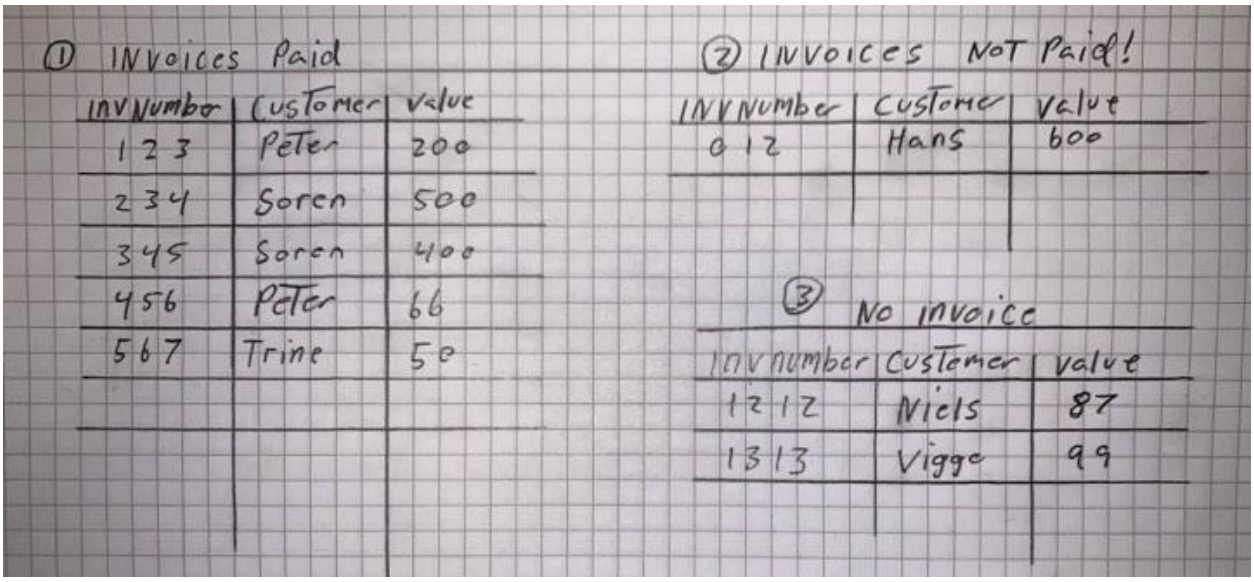


Product (AnalysisNo, Description, Amount, Price, Total)

TotalTransaction (ResearchNo, AnalysisNo, TotalexclVAT, VAT, TotalinclVAT)

Costumer (RNo, Name, Street, Postcode, City)

Exercise 5.



① Invoices Paid

INV Number	Customer	Value
123	Peter	200
234	Soren	500
345	Soren	400
456	Peter	66
567	Trine	50

② Invoices Not Paid!

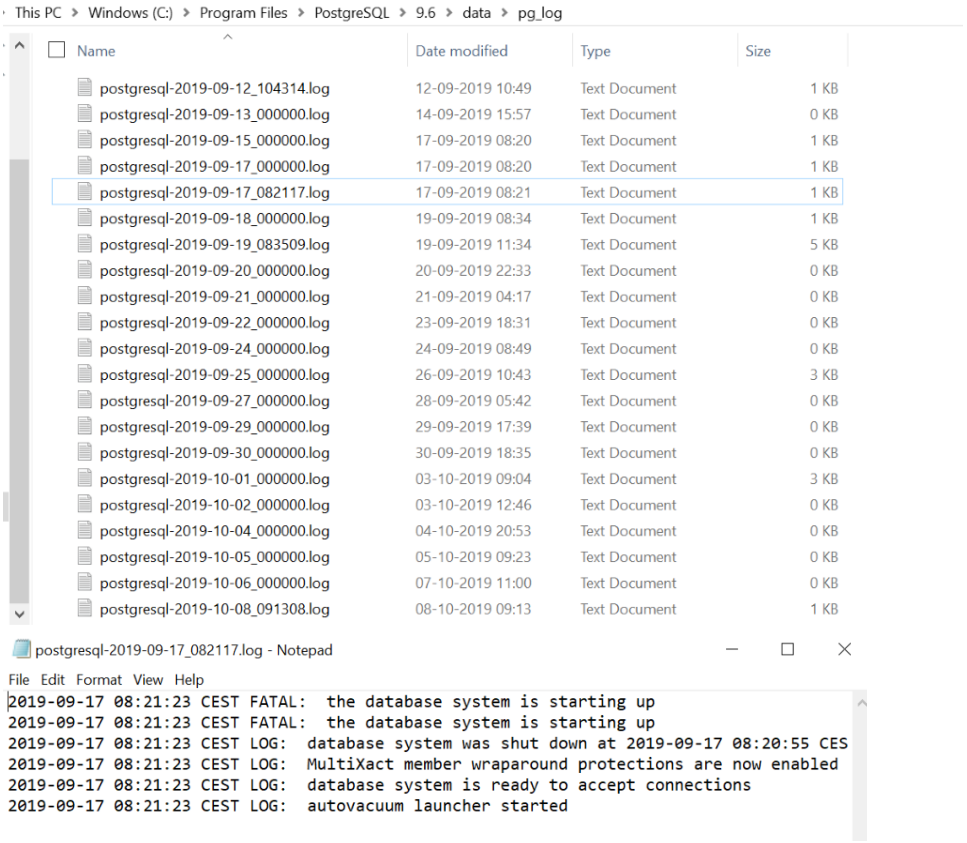
INV Number	Customer	Value
012	Hans	600

③ No invoice

INV number	Customer	Value
1212	Niels	87
1313	Vigge	99

Exercise 6.

1. Log file example



2. Dirty read problem

The dirty read problem happens when one transaction is able to read data that is currently being modified by another transaction which is running in parallel but has not committed itself yet.

The dirty read problem concerning data from the first poster occurs because users *Bruce* and *Sheila* are working on the same data at the same time. While *Bruce* is working on transaction for client 3, *Sheila* is working on all the customer data.

3. Non-repeatable read
   The non-repeatable read problem occurs when one transaction reads the same data twice, while at the same time it is modified by another transaction between the reads.
   Bruce and *Sheila* are again working at the same time (time T1). When *Sheila* executes the procedure for the first time, she can see that the balance is equal to 15 dollars(time T2), but when she checks again the balance is different (time T5), as it was changed by *Bruce* (time T3 and T4).

4. Phantom read

User: bruce	**Time**	**User:** sheila
BEGIN TRANSACTION;	T1	BEGIN TRANSACTION;
	T2	SELECT * FROM customers;
INSERT INTO customers VALUES	T3	
(		
6,		
'Neville, Robert',		
'555-9999',		
'1971-03-20',		
0.00		
);		

Hello my name is carl I have a fish in my pocket. This will have no effect on my role as your first man...

To the left we see an example of the phantom read problem.. Sheila is calling to get information from Customers 2 times. But in between Bruce inserts a new row of information. This means that Sheila will find a new row the second time she calls for information.

COMMIT TRANSACTION;	T4	
	T5	SELECT * FROM customers;
	T6	COMMIT TRANSACTION;