Hand-In 3

1.1.

```
CREATE VIEW WORKS_ON_VIEW AS
SELECT *
FROM works_on;
```

1.2.

```
CREATE VIEW sum_of_hours_per_project AS
SELECT sum(works_on.hours) AS sum, works_on.pno
 FROM works_on
 GROUP BY works_on.pno
 ORDER BY works_on.pno;
```

1.3.

```
CREATE VIEW employee_project_hours_cost AS
SELECT e.ssn AS EMP, CONCAT(e.fname, ' ', e.lname) AS EMP_Name, p.pnumber AS PROJ, p.pname AS Project_Name, SUM(w.hours) AS hours, SUM(w.hours)*300 AS Cost
FROM employee e, project p, works_on w
WHERE e.ssn=w.essn AND p.pnumber=w.pno
GROUP BY ssn, pnumber
ORDER BY ssn;
```

1.4.

```
CREATE VIEW department_view AS
SELECT dname AS dep_name, CONCAT(fname, ' ', lname) AS manager_name, salary
FROM department, employee
WHERE mgrssn=ssn;
```

1.5.

```
CREATE VIEW reasearch_employees_view AS
SELECT CONCAT(e.fname, ' ', e.lname) AS emp_name, CONCAT(s.fname, ' ', s.lname) AS supervisor_name, e.salary
FROM department, employee e, employee s
WHERE  dname='Research' AND e.superssn=s.ssn;
```

1.6.

```
CREATE VIEW project_and_department_view AS
SELECT pname, dname, COUNT(essn) AS number_of_employees, SUM(hours) AS number_of_hours
FROM works_on w, project p, department d
WHERE w.pno=p.pnumber AND p.dnum=d.dnumber
GROUP BY pnumber, dname;
```

1.7

```
CREATE VIEW project_and_department_more_than_one_emp_view AS
SELECT pname, dname, COUNT(essn) AS number_of_employees, SUM(hours) AS number_of_hours
FROM works_on w, project p, department d
WHERE w.pno=p.pnumber AND p.dnum=d.dnumber
GROUP BY pnumber, dname
HAVING COUNT(essn) > 1;
```

1.8

```
CREATE VIEW name_of_all_emoloyees AS
SELECT CONCAT(e.fname, ' ', e.lname) AS Name
FROM employee e, employee s
WHERE e.superssn=s.ssn AND s.superssn='888665555';
```

1.9

```
CREATE VIEW average_salary AS
SELECT dname, COUNT(ssn) AS number_of_employees
FROM department, employee
WHERE dno=dnumber
GROUP BY dname
HAVING AVG(salary) > 30000;
```

1.10

```
CREATE VIEW number_of_employees_in_location AS
SELECT dlocation, COUNT(essn) AS number_of_employees
FROM dept_locations d, works_on, department, project
WHERE department.dnumber=d.dnumber AND department.dnumber=project.dnum AND
project.pnumber=pno
GROUP BY dlocation;
```

2.1

```
CREATE OR REPLACE FUNCTION log_works_on() RETURNS TRIGGER AS $BODY$
DECLARE count_works_on INTEGER;
BEGIN
IF (TG_OP='INSERT') THEN
INSERT INTO log_works_on(essn, operation, date_stamp)
VALUES(new.essn, 'INSERT', NOW());
RETURN NEW;
END IF;
IF (TG_OP='UPDATE') THEN
INSERT INTO log_works_on(essn, operation, date_stamp)
VALUES(new.essn, 'UPDATE', NOW());
RETURN NEW;
END IF;
IF (TG_OP='DELETE') THEN
INSERT INTO log_works_on(essn, operation, date_stamp)
VALUES(old.essn, 'DELETE', NOW());
RETURN NEW;
END IF;

RETURN NULL;
END;
$BODY$ LANGUAGE plpgsql;


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

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

create trigger log_delete
after delete on works_on for each row
execute procedure log_works_on();
```

2.2

```
CREATE OR REPLACE FUNCTION project_check()
RETURNS TRIGGER AS $$
DECLARE project_no INTEGER;
BEGIN
SELECT COUNT(pnumber) INTO project_no
FROM project
GROUP BY dnum;
IF(project_no > 3) THEN
RAISE EXCEPTION 'Too many projects for this department!';
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER max3
AFTER INSERT ON project
FOR EACH ROW
EXECUTE PROCEDURE project_check();
```

2.3

```
CREATE OR REPLACE FUNCTION employee_check()
RETURNS TRIGGER AS $$
DECLARE employee_no INTEGER;
BEGIN
SELECT COUNT(pno) INTO employee_no
FROM works_on
GROUP BY essn;
IF(employee_no > 4) THEN
RAISE EXCEPTION 'Too many projects for this employee!';
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER max4
AFTER INSERT ON works_on
FOR EACH ROW
EXECUTE PROCEDURE employee_check();
```

2.4

```
CREATE OR REPLACE FUNCTION log_department() RETURNS TRIGGER AS $$
BEGIN
IF(TG_OP='INSERT') THEN
INSERT INTO log_department(mgrssn, operation, date_stamp)
VALUES(new.mgrssn, 'INSERT', NOW());
RETURN NEW;
ELSIF (TG_OP='UPDATE') THEN
INSERT INTO log_department(mgrssn, operation, date_stamp)
VALUES(new. mgrssn, 'UPDATE', NOW());
RETURN NEW;
ELSIF (TG_OP='DELETE') THEN
INSERT INTO log_department(mgrssn, operation, date_stamp)
VALUES(old. mgrssn, 'DELETE', NOW());
RETURN NEW;
END IF;
END;
$$ LANGUAGE plpgsql;

create trigger log_insert
before insert on department for each row
execute procedure log_department();

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

create trigger log_delete
```

```
after delete on department for each row

execute procedure log_department();
```

2.5

```
CREATE OR REPLACE FUNCTION hours_salary_check()

RETURNS TRIGGER AS $$

BEGIN

IF(employee.salary > 80000 OR employee.salary < 25000) THEN

RAISE EXCEPTION 'That salary is not compatible with the companys policy';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;
```

3.2

An example of the “dirty read problem” might be a situation when a user inserts an update in the table works_on and right after that another user is viewing the log_works_on table. Then if the first user rolls back the update, the second user has already seen that an update has been made and is left with incorrect information.

3.3

An example of the “non-repeatable read” might be a situation when one user views the log_works_on table and later another user makes changes in the works_on table (either inserts, updates or deletes a row). Then when the first user views the log_works_on table again, he will only be able to see the table with new rows added.

3.4

An example of the “phantom read” might be a situation when a user views all tuples from log_works_on with a specific essn and another user adds another operation on this essn later. Then when the first user views the log_works_on table with this essn again, there will be an extra row added. The extra row is the “phantom row”.

4.

Analysis no -> description, price
Research numer -> research date
Company name -> company address
clientName -> clientAddress

First normal form

clientName, clientAddress, companyName, companyAddress, invoiceNo, researchNo, researchDate, analysisNo, description, amount, price

Second normal form

analysisNo, description, price

researchNo, researchDate

invoiceNo, researchNo, analysisNo, clientName, clientAddress, companyName, companyAddress , amount

Third normal form

clientName, clientAddress

companyName, companyAddress

analysisNo, description, price

researchNo, researchDate

invoiceNo, researchNo, analysisNo, clientName, companyName, amount

5.1

SELECT *

FROM payments EXCEPT debtors;

5.2

SELECT *

FROM payments INTERSECTION debtors;

SELECT *

FROM payments RIGHT JOIN debtors;

SELECT *

FROM debtors LEFT JOIN payments;

5.3

SELECT Customer

FROM payments EXCEPT debtors;