DBS HAND IN 3

Exercise 1

1.

```
CREATE VIEW view1 AS SELECT * FROM works_on;
SELECT * FROM view1;
```

2.

```
CREATE VIEW vie2(PROJ_Name, PROJ_No, PROJ_Location, DEPART_No, hours)
        AS SELECT p.pname, p.pnumber, p.plocation, p.dnum, SUM(wo.hours)
        FROM Project p, Works_on wo
        WHERE p.pnumber = wo.pno
        GROUP BY p.pnumber, p.pname
        ORDER BY p.pnumber;
SELECT * FROM view2;
```

3.

```
CREATE VIEW view3(EMP_No, EMP_Name, PROJ_No, PROJ_Name, Hours, Cost)
        AS SELECT e.ssn, e.fname || ' ' || e.lname, p.pnumber, p.pname, wo.hours, wo.hours *
300
        FROM Employee e, Works_on wo, Project p
        WHERE e.ssn = wo.essn AND wo.pno = p.pnumber;
SELECT * FROM view3;
```

4.

```
CREATE VIEW view4(DEPART_Name, MANAGER_Name, MANAGER_Salary)
        AS SELECT d.dname, e.fname || ' ' || e.lname, e.salary
        FROM Department d, Employee e
```

```
        WHERE d.mgrssn = e.ssn;
SELECT * FROM view4;
```

5.

```
CREATE VIEW view5(EMP_Name, SUPERVISOR_Name, EMP_Salary)
        AS SELECT e1.fname || ' ' || e1.lname, e2.fname || ' ' || e2.lname, e1.salary
        FROM Employee e1, Employee e2, Department d
        WHERE e1.dno = d.dnumber AND d.dname = 'Research' AND e1.superssn = e2.ssn;
SELECT * FROM view5;
```

6.

```
CREATE VIEW view6(PROJ_Name, DEPART_Name, No_of_EMP, Hours_per_week)
        AS SELECT p.pname, d.dname, COUNT(*), SUM(wo.hours)
        FROM Project p, Department d, Works_on wo
        WHERE p.dnum = d.dnumber AND p.pnumber = wo.pno
        GROUP BY p.pname, d.dname;
SELECT * FROM view6;
```

7.

```
CREATE VIEW view7(PROJ_Name, DEPART_Name, No_of_EMP, Hours_per_week)
        AS SELECT *
        FROM view6 v6
        WHERE v6.No_of_EMP > 1;
SELECT * FROM view7;
```

8.

```
CREATE VIEW view8(EMP_Name)
```

```
        AS SELECT e1.fname || ' ' || e1.lname
        FROM Employee e1, Employee e2, Employee e3
        WHERE e1.superssn = e2.ssn AND e2.superssn = e3.ssn AND e3.ssn = '888665555';
SELECT * FROM view8;
```

9.

```
CREATE VIEW view9(DEPART_Name, No_of_EMP)
        AS SELECT d.dname, COUNT(e.dno)
        FROM Department d, Employee e
        WHERE e.dno = d.dnumber
        GROUP BY d.dname
        HAVING AVG(e.salary) > 30000;
SELECT * FROM view9;
```

10.

Create a view which contains the project name, project number, department name, department location for the projects with an average of worked hours per employee bigger than 20

```
CREATE VIEW view10(PROJ_Name, PROJ_No, DEPART_Name)
        AS SELECT p.pname, p.pnumber, d.dname
        FROM Project p, Department d, Works_on wo
        WHERE p.dnum = d.dnumber AND p.pnumber = wo.pno
        GROUP BY p.pname, p.pnumber, d.dname
        HAVING AVG(wo.hours) > 20;
SELECT * FROM view10;
```

Exercise 2

1.

```
CREATE TABLE Log_works_on(
        ESSN    CHAR (9) NOT NULL,
        PNO     INTEGER NOT NULL,
        HOURS_now   INTEGER,
        HOURS_before   INTEGER,
        DAY_TIME TIMESTAMP);

CREATE FUNCTION Log_for_works_on() RETURNS TRIGGER AS $BODY$
BEGIN
        IF(tg_op = 'INSERT') THEN INSERT INTO Log_works_on(ESSN, PNO, HOURS_now,
DAY_TIME)
                VALUES(NEW.ESSN, NEW.PNO, NEW.HOURS, NOW());
        RETURN NEW;
        END IF;

        IF(tg_op = 'UPDATE') THEN INSERT INTO Log_works_on(ESSN, PNO, HOURS_now,
DAY_TIME)
                VALUES(NEW.ESSN, NEW.PNO, NEW.HOURS, OLD.HOURS, NOW());
        RETURN NEW;
        END IF;

   IF(tg_op = 'DELETE') THEN INSERT INTO Log_works_on(ESSN, HOURS_before, DAY_TIME)
                VALUES(OLD.ESSN, OLD.HOURS, 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_for_works_on();

CREATE TRIGGER Log_update BEFORE UPDATE ON Works_on FOR EACH ROW
EXECUTE PROCEDURE Log_for_works_on();

CREATE TRIGGER Log_delete AFTER DELETE ON Works_on FOR EACH ROW
EXECUTE PROCEDURE Log_for_works_on();
```

2.

```
CREATE FUNCTION prevent_insert() RETURNS TRIGGER AS $func$
DECLARE pcount integer;
BEGIN
        SELECT COUNT(*) into pcount
        FROM Project
        WHERE dnum = new.dnum;
        IF pcount >= 3 THEN
                RAISE EXCEPTION 'You cannot add more than 3 projects for a department';
    END IF;
RETURN NEW;
END;
$func$ LANGUAGE plpgsql;

CREATE TRIGGER trigger2 BEFORE INSERT ON Project
FOR EACH ROW
```

```
EXECUTE PROCEDURE prevent_insert();
```

3.

```
CREATE FUNCTION prevent_insert_wo() RETURNS TRIGGER AS $func$
DECLARE pnocount integer;
BEGIN
        SELECT COUNT(*) into pnocount
        FROM Works_on
        WHERE essn = new.essn;
        IF pnocount >= 4 THEN
                RAISE EXCEPTION 'You cannot add more than 4 projects for an employee';
    END IF;
RETURN NEW;
END;
$func$ LANGUAGE plpgsql;

CREATE TRIGGER trigger3 BEFORE INSERT ON Works_on
FOR EACH ROW
EXECUTE PROCEDURE prevent_insert_wo();
```

4.

```
CREATE TABLE Log_department(
        DNAME         VARCHAR (20) UNIQUE,
        DNUMBER       INTEGER NOT NULL,
        MGRSSN        CHAR (9),
        MGRSTARTDATE   DATE,
   DAY_TIME TIMESTAMP);
```

```
CREATE FUNCTION Log_for_department() RETURNS TRIGGER AS $BODY$

BEGIN

	IF(tg_op = 'INSERT') THEN INSERT INTO Log_department(DNAME, DNUMBER, MGRSSN_now, MGRSTARTDATE_now, DAY_TIME)

		VALUES(NEW.DNAME, NEW.DNUMBER, NEW.MGRSSN, NEW.MGRSTARTDATE, NOW());

	RETURN NEW;

	END IF;


	IF(tg_op = 'UPDATE') THEN INSERT INTO Log_department(DNAME, DNUMBER, MGRSSN_now, MGRSTARTDATE_now, DAY_TIME)

		VALUES(NEW.DNAME, NEW.DNUMBER, NEW.MGRSSN, OLD.MGRSSN, NEW.MGRSTARTDATE, OLD.MGRSTARTDATE, NOW());

	RETURN NEW;

	END IF;


   IF(tg_op = 'DELETE') THEN INSERT INTO Log_department(DNAME, DNUMBER, MGRSSN_before, MGRSTARTDATE_before, DAY_TIME)

		VALUES(OLD.DNAME, OLD.DNUMBER, OLD.MGRSSN, OLD.MGRSTARTDATE, NOW());

	RETURN NEW;

	END IF;


RETURN NULL;

END;

$BODY$ LANGUAGE plpgsql;


CREATE TRIGGER Log_insert BEFORE INSERT ON Department FOR EACH ROW
```

```
EXECUTE PROCEDURE Log_for_department();

CREATE TRIGGER Log_update BEFORE UPDATE ON Department FOR EACH ROW
EXECUTE PROCEDURE Log_for_department);

CREATE TRIGGER Log_delete AFTER DELETE ON Department FOR EACH ROW
EXECUTE PROCEDURE Log_for_department();
```

5.

```
CREATE FUNCTION prevent_insert_emp() RETURNS TRIGGER AS $func$
BEGIN
    IF new.salary <10000 THEN
            RAISE EXCEPTION 'You cannot add an employee having a salary less than 10000';
    END IF;
RETURN NEW;
END;
$func$ LANGUAGE plpgsql;

CREATE TRIGGER trigger5 BEFORE INSERT ON Employee
FOR EACH ROW
EXECUTE PROCEDURE prevent_insert_emp();
```

Exercise 3

1.

```
public static void createTable() throws Exception
{
      try
      {
            Connection conn = getConnection();
            PreparedStatement create = conn.prepareStatement(
```

```
"CREATE TABLE IF NOT EXISTS Book(id INTEGER NOT NULL, name VARCHAR(50), author
VARCHAR(50), PRIMARY KEY(id));");
                create.executeUpdate();
        }
        catch(Exception e)
        {
                System.out.println(e);
        }
        finally
        {
                System.out.println("Function complete.");
        }
}
```

2.

```
public static void insertIntoTable() throws Exception
{
        try
        {
                Connection conn = getConnection();
                PreparedStatement insert = conn.prepareStatement("INSERT INTO Book(id,
name, author) VALUES (1,'Ulysses', 'James Joyce),"
                + "(2, 'Pride and Prejudice', 'Jane Austen'),"
                + "(3, 'The Immortals', null);");
                insert.executeUpdate();
        }
        catch(Exception e)
        {
                System.out.println(e);
        }
        finally
        {
                System.out.println("Insert completed");
        }
}
```

3.

```
public static void updateTable() throws Exception
{
        try
        {
                Connection conn = getConnection();
                PreparedStatement update = conn.prepareStatement("UPDATE Book SET author
= 'Tamora Pierce' WHERE id = 3;");
                update.executeUpdate();
        }
        catch(Exception e)
        {
                System.out.println(e);
```

```
	}
	finally
	{
		System.out.println("Update completed");
	}
}
```

4.

```
public static void deleteContent() throws Exception
{
	try
	{
		Connection conn = getConnection();
		PreparedStatement delete = conn.prepareStatement("TRUNCATE Book;");
		delete.executeUpdate();
	}
	catch(Exception e)
	{
		System.out.println(e);
	}
	finally
	{
		System.out.println("Delete completed");
	}
}
```

5.

```
public static void dropTable() throws Exception
{
	try
	{
		Connection conn = getConnection();
		PreparedStatement drop = conn.prepareStatement("DROP TABLE Book;");
		drop.executeUpdate();
	}
	catch(Exception e)
	{
		System.out.println(e);
	}
	finally
	{
		System.out.println("Delete completed");
	}
}
```

6.

```
import java.sql.Connection;
import java.sql.DriverManager;
```

```
import java.sql.PreparedStatement;

public class Main {

	public static void main(String[] args) throws Exception
	{
		createTable();
		insertIntoTable();
		updateTable();
		deleteContent();
		dropTable();
	}


	public static void createTable() throws Exception {…}

	public static void insertIntoTable() throws Exception {…}

	public static void updateTable() throws Exception {…}

	public static void deleteTable() throws Exception {…}

	public static void dropTable() throws Exception {…}

	public static Connection getConnection() throws Exception

	{

		try
		{
			String driver = "org.postgresql.Driver";
			String url = "jdbc:postgresql://localhost:5432/postgres";
			String username = "username";
			String password = "password";
			Class.forName(driver);

			Connection con  = DriverManager.getConnection(url, username,
			password);
			System.out.println("Connected");
			return con;
		}
		catch(Exception e)
		{
			System.out.println(e);
		}
		return null;

	}

}
```

Exercise 4

Unnormalized form

RECEIPT												
ReceiptNo	Shop	Contact	ShopDesc	Item	Cat.	ItemPrice	Amount	Total	VAT	PaymentMethod	DateTime	Message
35954716	Føtex Vejle	www.foetex.dk, 76438000	ALLE DAGE 08.00 – 21.00, BAGEREN ÅBNER 07.00 MANDAG-SØNDAG	BABY KARTOFLER, BERNAISE 325G	FRUGT & GRØNT, SLAGTER	18,00, 15,95	2, 2	67,90	13,58	Dankort	25.04.2018, 15:40	Tak fordi du benyttede selvscanning I føtex

1NF

RECEIPT												
ReceiptNo	Shop	Contact	ShopInfo	Item	Cat.	ItemPrice	Amount	Total	VAT	PaymentMethod	DateTime	Message
35954716	Føtex Vejle	www.foetex.dk, 76438000	ALLE DAGE 08.00 – 21.00, BAGEREN ÅBNER 07.00 MANDAG-SØNDAG	BABY KARTOFLER	FRUGT & GRØNT	18,00	2	36,00	7,20	Dankort	25.04.2018, 15:40	Tak fordi du benyttede selvscanning I føtex
35954716	Føtex Vejle	www.foetex.dk, 76438000	ALLE DAGE 08.00 – 21.00, BAGEREN ÅBNER 07.00 MANDAG-SØNDAG	BERNAISE 325G	SLAGTER	15,95	2	31,90	6,38	Dankort	25.04.2018, 15:40	Tak fordi du benyttede selvscanning I føtex

2NF

SHOP					
Name	City	Website	Telephone	OpeningHours	ShopInfo
Føtex	Vejle	www.foetex.dk	76438000	ALLE DAGE 08.00 – 21.00	BAGEREN ÅBNER 07.00 MANDAG - SØNDAG

ITEM		
Name	Category	Price
BABY KARTOFLER	FRUGT & GRØNT	18,00
BERNAISE 325G	SLAGTER	15,95

PAYMENT					
ItemName	Price	Amount	TotalPrice	VAT	PaymentMethod
BABY KARTOFLER	18,00	2	36,00	7,20	Dankort
BERNAISE 325G	15,95	2	31,90	6,38	Dankort

RECEIPT						
ReceiptNo	Shop	TotalPrice	VAT	Date	Time	Message
35954716	Føtex	67,90	13,58	25.04.2018	15:40	Tak fordi du benyttede selvscanning I føtex

3NF

SHOP					
Name	City	Website	Telephone	OpeningHours	ShopInfo
Føtex	Vejle	www.foetex.dk	76438000	ALLE DAGE 08.00 – 21.00	BAGEREN ÅBNER 07.00 MANDAG - SØNDAG

ITEM		
Name	Category	Price
BABY KARTOFLER	FRUGT & GRØNT	18,00
BERNAISE 325G	SLAGTER	15,95

PAYMENT					
ItemName	Price	Amount	TotalPrice	VAT	PaymentMethod
BABY KARTOFLER	18,00	2	36,00	7,20	Dankort
BERNAISE 325G	15,95	2	31,90	6,38	Dankort

TOTAL	
TotalPrice	VAT
67,90	13,58

RECEIPT				
ReceiptNo	Shop	Date	Time	Message
35954716	Føtex	25.04.2018	15:40	Tak fordi du benyttede selvscanning I føtex

Exercise 5

Initial form

Invoice No.	Contractor	Customer	Research number	Research Date	Analysis No.	Description	Amount	Price	VAT	Total Price
12	Holmsen Aps	ØKOLOGIC	200207	02/01/2016	10,15	Pesticide Test, Bacteria Analysis	2,3	2400, 1500	975	4875

1NF

Primary key

Invoice No.	Contractor	Customer	Research number	Research Date	Analysis No.	Description	Amount	Price	VAT	Total Price
12	Holmsen Aps	ØKOLOGIC	200207	02/01/2016	10	Pesticide Test	2	2400	600	3000
12	Holmsen Aps	ØKOLOGIC	200207	02/01/2016	15	Bacteria Analysis	3	1500	375	1875

2NF

Customer table

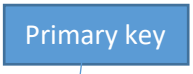
Primary key

CustomerID	Name	ZIP Code	City	Street	House
C001	ØKOLOGIC	8700	Horsens	Chr.Østegårds Vej	10

Analysis table

AnalysisID	Description	Price
10	Pesticide Test	1200
15	Bacteria analysis	500

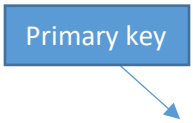
Invoice table



Invoice No.	Contractor	CustomerID	Research number	Research date	AnalysisID	Total price (no VAT)	VAT	Total price (with VAT)
12	Holmsen Aps	C001	200207	02/01/2016	10	2400	600	3000
12	Holmsen Aps	C001	200207	02/01/2016	15	1500	375	1875

3NF

Customer table



CustomerID	Name	ZIP Code	City	Street	House
C001	ØKOLOGIC	8700	Horsens	Chr.Østegårds Vej	10

Contractor table

Primary key

ContractorID	Name	ZIP Code	City	Street	House
CR001	Holmsen Aps	9990	Fnatting	Vesterhavsvej	25

Analysis table

Primary key

AnalysisID	Description	Price
10	Pesticide Test	1200
15	Bacteria analysis	500

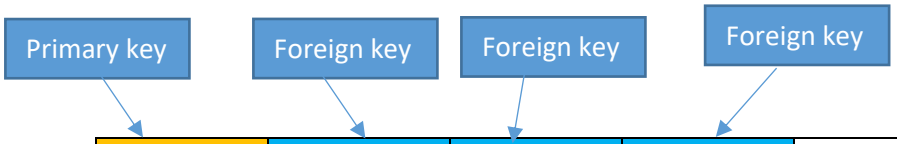
Research table

Primary key

Foreign key

Research No.	Research date	AnalysisID	Amount	Total price
200207	02/01/2016	10	2	2400
200207	02/01/2016	15	3	1500

Invoice table



Invoice No.	ContractorID	CustomerID	Research No.	Total price (no VAT)	VAT	Total price (with VAT)
12	CR001	C001	200207	3900	975	4875

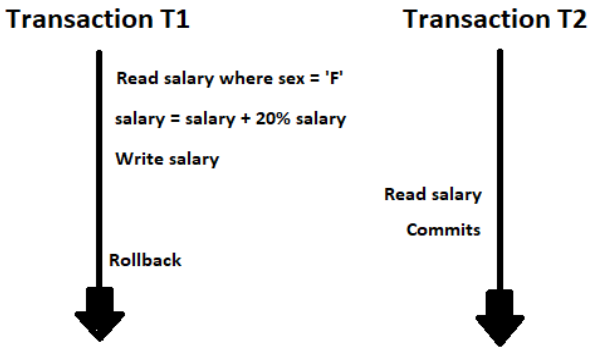
Exercise 6

2. DIRTY READ

Dirty read problem occurs when a transaction reads data that has been written by another uncommitted transaction running concurrently. If the former transaction executes a rollback, the latter uses data that does not actually exist.

Example on COMPANY database:

Supposing a transaction T1 increases the salary for all females with 20% but does not commit the change and another concurrent transaction T2 gets the salary for all employees and commits, then if T1 executes a rollback changing the data to its initial state, T2 will have an erroneous answer.

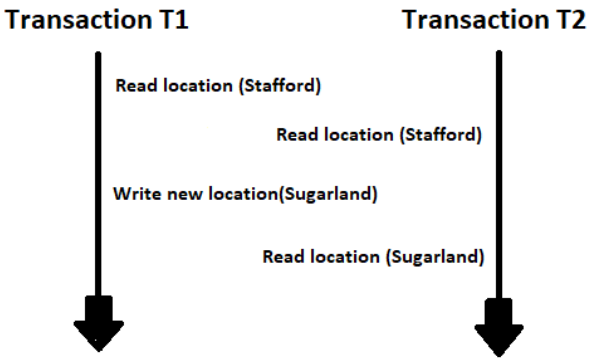


3. NON-REPEATABLE READ

Non-repeatable read problem occurs when during the course of a transaction, a row is retrieved twice and the values within the row differ between reads.

Example on COMPANY database:

Supposing two transactions, T1 and T2 are interested in getting the location for department number 4, then if T1 will update the name of the location from 'Stafford' to 'Sugarland' for instance, a second attempt for T2 of reading the location will return 'Sugarland'. In the end, there are two locations for the same department.

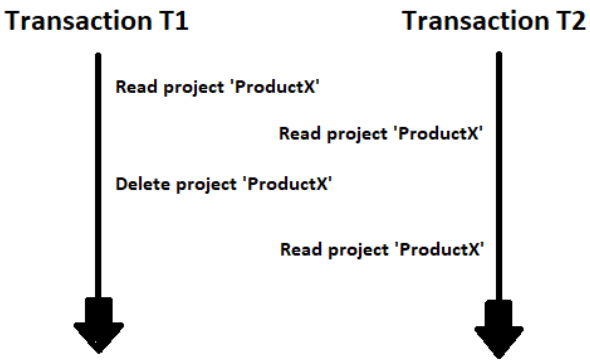


4. PHANTOM READ

Phantom read problem occurs when, in the course of a transaction, two identical queries are executed, and the collection of rows returned by the second query is different from the first.

Example on COMPANY database:

Supposing two transactions, T1 and T2 are interested in getting all the information about a project named 'ProductX', then if T1 will delete the project, a second attempt for T2 of getting the information about 'ProductX' will fail, due to the fact that 'ProductX' does not exist anymore.



Exercise 7

The first table has been referred to as T1, while the second as T2.

- Make a list with invoices which have been paid.

  Using EXCEPT:

```
SELECT invoiceNo, customer, value
FROM T1
EXCEPT
SELECT invoiceNo, customer, value
FROM T2;
```

  Using JOINS:

```
SELECT invoiceNo, customer, value
FROM T1
LEFT JOIN T2 USING(invoiceNo, customer, value)
WHERE invoiceNo IS NULL;
```

(left join and is null clause)

```
SELECT invoiceNo, customer, value
FROM T1 t1
WHERE NOT EXISTS (
```

```
    SELECT *
    FROM T2 t2
    WHERE ( t1.invoiceNo, t1.customer, t1.value) = ( t2.invoiceNo, t2.customer, t2.value)
    );
```

(anti-join)

- Make a list with invoices which have not been paid.

    Using INTERSECT:

```
SELECT invoiceNo, customer, value
FROM T2
INTERSECT
SELECT invoiceNo, customer, value
FROM T1;
```

    Using JOINS:

```
SELECT invoiceNo, customer, value
FROM T2
INNER JOIN T1 USING(invoiceNo, customer, value);
```

(inner join)

```
SELECT invoiceNo, customer, value
FROM T2
WHERE (invoiceNo, customer, value) IN (
    SELECT invoiceNo, customer, value FROM T1);
```

(semi-join)