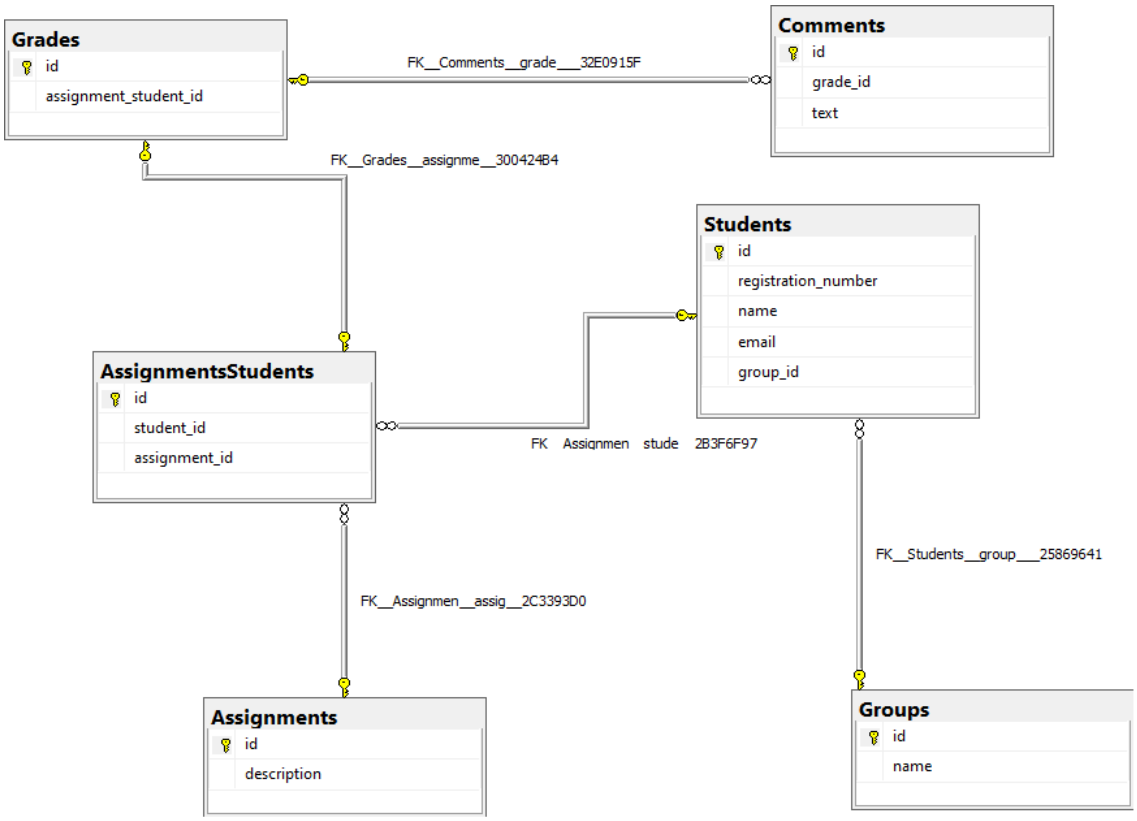
Database Management Systems – Practical Test
Computer Science in English, 2nd year

I.

Question Number	Answer
1	**150**
2	**150**
3	**c**

II.

a)




```
use testdb

create table Groups (
	id int primary key,
	name varchar(50)
)

create table Students (
	id int primary key,
	registration_number int,
	name varchar(50),
	email varchar(50),
	group_id int references Groups(id)
)
```

```
create table Assignments (
	id int primary key,
	description varchar(50)
)

create table AssignmentsStudents (
	id int primary key,
	student_id int references Students(id),
	assignment_id int references Assignments(id),
	unique(student_id, assignment_id)
)

create table Grades (
	id int primary key,
	assignment_student_id int references AssignmentsStudents(id),
	unique (assignment_student_id)
)

create table Comments (
	id int primary key,
	grade_id int references Grades(id),
	text varchar(100)
)
```

b)

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace TestApp
{
    public partial class Form1 : Form
    {
        private readonly string GROUPS_TABLE = "Groups";
        private readonly string STUDENTS_TABLE = "Students";
        private readonly string FK_GROUPS_STUDENTS = "FK__Students__group___25869641";

        private DataSet dataSet = new DataSet();
        private SqlConnection dbConnection;

        private SqlDataAdapter dataAdapterGroups, dataAdapterStudents;
        private BindingSource bindingGroups = new BindingSource();
        private BindingSource bindingStudents = new BindingSource();

        private void Form1_Load(object sender, EventArgs e)
        {
        }

        // <sends changes operated through dgvStudents back to the database>
        private void update_Click(object sender, EventArgs e)
        {
            dataAdapterStudents.Update(dataSet, STUDENTS_TABLE);
        }
        // </sends changes operated through dgvStudents back to the database>

        public Form1()
        {
            // <code that connects to the database>
            InitializeComponent();
            dbConnection = new SqlConnection("Data Source = (LocalDb)\\MSSQLLocalDb; " +
                "Initial Catalog = testdb; Integrated Security = SSPI;");
```

```
            dataAdapterGroups = new SqlDataAdapter($"SELECT * FROM {GROUPS_TABLE}", dbConnection);
            dataAdapterStudents = new SqlDataAdapter($"SELECT * FROM {STUDENTS_TABLE}", dbConnection);

            // </code that connects to the database>

            // <fetches data into the application>

            new SqlCommandBuilder(dataAdapterStudents);

            dataAdapterGroups.Fill(dataSet, GROUPS_TABLE);
            dataAdapterStudents.Fill(dataSet, STUDENTS_TABLE);

            // <fetches data into the application>

            // <binds the DataGridViews such that whenever a different group is selected in dgvGroups,
dgvStudents displays all its students>

            var dataRelation = new DataRelation(
                FK_GROUPS_STUDENTS,
                dataSet.Tables[GROUPS_TABLE].Columns["id"],
                dataSet.Tables[STUDENTS_TABLE].Columns["id"]);
            dataSet.Relations.Add(dataRelation);
            bindingGroups.DataSource = dataSet;
            bindingGroups.DataMember = GROUPS_TABLE;

            bindingStudents.DataSource = bindingGroups;
            bindingStudents.DataMember = FK_GROUPS_STUDENTS;

            dgvGroups.DataSource = bindingGroups;
            dgvStudents.DataSource = bindingStudents;

            // </binds the DataGridViews such that whenever a different group is selected in dgvGroups,
dgvStudents displays all its students>
        }
    }
}
```

c)

Transaction 1	**Transaction 2**

`begin tran`
 	
`update Groups`
`set name = '932'`
`where id = 1`	

	`begin tran`
	
	`update Groups`
	`set name = '932'`
	`where id = 2`

`update Groups`
`set name = '933'`
`where id = 2`	

	`update Groups`
	`set name = '933'`
	`where id = 1`

`commit tran`	
	`commit tran`

The deadlock occurs because under any pessimistic isolation level such as the default read commited the update operation requires an exclusive lock on the row of the table Groups that is being updated. The lock is only released

when the transaction commits (or aborts, not our case), so what happens is that our query console 1 acquires a lock on the row with id=1, the query console 2 acquires a lock on the row with id=2. Thus when the query console 1 now requests a lock on id=2 it will have to wait for console 2, but query console 2 must also wait for the lock to be released by query console 1 on id=1. So a cycle on the dependency graph is detected and one of the transactions is chosen as victim to be aborted and the other is allowed to continue.

The way to prevent deadlocks is you have to choose an order to access your objects in your transaction, suppose you decide to use the rows of update in the order 1, 2. In this case transaction 2 will be modified to first access object 1 and then 2, thus cycles never form in the dependency graph and deadlocks are never encountered. But if you absolutely must use a different order to that you may also set the deadlock priority of a transaction by SET DEADLOCK_PRIORITY <VALUE> where <VALUE> can be HIGH, LOW or NORMAL. As such the transaction with the lower priority will usually be aborted.