- 60 points Lab1-6 a mini DBMS system to implement.
- 40 points you can choose:
  i) implement System R query optimization algorithm in your mini DBMS
  ii) written test from query optimization. 10points theory, 30points to find the optimal execution strategy for a SELECT statement with join of 3 tables.
  iii) implement not specified extra features
  iv) quizzes in every course

Minimal criteria to pass the exam is to present first 4 labs and accumulate 50 points.

The labs have to be presented in the next lab. In this way you will accumulate the corresponding points. **For every 2 weeks delay you accumulate -2 points**.

Mini DBMS specification

The mini relational DBMS system project is for **2 students**.

You will have to implement your own Database Management System (DBMS). It will have a **server** and a **client** component, communicating by a communication protocol.

The client can be a visual interface like SQL Developer (of Oracle) or Management Studio of MS SQL Server. You can choose to implement the client in command line too.

The server will execute typical SQL statements: Create Database, Create Table, INSERT, DELETE, SELECT, etc. and send a message to the client about the success of failure of SQL statement execution. In case of the SELECT statement, the server will send the result rows of the statement to the client. In Students.sql you find an example of SQL statements your project will have to execute.

A powerful feature of a DBMS are the **index** files. B+ trees are the most used index types, but it is a highly complex task to implement them. The index files are implemented in form of B+ trees in key-value systems, usually. We will use a key-value system to store the data of our mini SGBD. Typical statements in a key-value system:

put(key, value)

value = get(key)

The key-value system creates automatically B+ tree index for the Key part. So if we use get(key) command, it will use the index automatically.

The tables of the system will be stored in a file of a key-value system, also the index files.
For each table T in the relational DB the following key-value files are produced:

- T data are stored in a master K-V file T.K-V
  - Key = T.PrimaryKey value
  - Value = concatenation of T non-primary key attributes (columns)
- for every unique index file UniqInd of T with key UniqKey a new UniqInd.K-V file is created
  - Key = UniqKey value
  - Value = T.Key (of master K-V file)
- for every non-unique index file NonUniqInd of T with key NonUniqKey a new NonUniqInd.K-V file is created
  - Key = NonUniqKey
  - Value = T.Key1#T.Key2#...#T.Key*p*

Where *p* is the number of rows with the same NonUniqKey value and T.Key*i* (*i*=1,..,*p*) are the primary keys of the rows, which are keys in the master file.

Example: Table: Students(StudID, GroupID, Name, Tel, email)
With one row (12345, 243,'Rusu Mihai', '0745123456','mrusu@gmail.com')

1. Master K-V file Students.kv
   Key = StudID
   Value = GroupID + Name + Tel + email
   This row above will be stored in K-V file as follows:
   Key: 12345
   Value:'243#Rusu Mihai#0745123456#mrusu@gmail.com'

2. Unique Index on email in K-V file email.ind
   Key = email
   Value = StudID
   For the row above will be one row in email.ind:
   Key = 'mrusu@gmail.com'
   Value = 12345
3. Non-unique index on GroupID in GroupID.ind
   Key = 243

Value = 12345

For other rows with GroupID = 243 and primary keys: 23456, 34567, 45678, 56789 the non-unique index will be:

Key = 243

Value = 12345#23456#34567#45678#56789

Lab1. 10p You have to implement the next facilities:

```
create database, create table, drop database,
 drop table, create index.
```

You can create one visual interface as in MS SQL Server Management Studio or you can execute the statements from command line.

The structure of the tables, primary key constraints, and foreign key constraints will be stored in an xml or json file, this will be the catalog of miniDBMS or metadata (Metadata is data about data).

See examples for system catalog in DataBases.xml and Catalog.xml. Implement only the structure of the tables. Inserts in tables will be implemented in Lab2.

Read about Key-value (K-V) systems. Choose one to implement facilities of a Database Management System. One relational table will be stored in a key-value file; one row of a table will be stored in a record of a key-value file (Lab2). In the Key part the primary key will be stored, in the Value part the rest of the record.

Lab2. 10p Implement a Visual Query Designer for INSERT, DELETE, or execute it from command line. Implement INSERT, DELETE in a relational table stored as a K-V file. The value of the primary key will be stored in the Key part, the other attributes in the Value part concatenated as strings.

Lab3. 10p Implement index files for unique keys (UniqIndex) and non-unique keys (NonUniqIndex).

Insert in UniqIndex with Key:uniqKey and PrimaryKey:primKey:

```
- find uniqKey in file UniqIndex,
    if uniqKey exists {reject the insert, unique key already exists}
      else {insert in UniqIndex Key:uniqKey and Value:primKey}
```

Insert in NonUniqIndex with Key:nonUniqKey and PrimaryKey:primKey

```
- find nonUniqKey in file NonUniqIndex,
    if nonUniqKey **not** exists
```

```
            {insert in NonUniqIndex Key:nonUniqKey and Value:primKey}
    else {for the nonUniqKey found let be the Value=foundValue
                update Value for nonUniqKey with = foundValue#primKey}
```

For composite keys, concatenate the values of attributes separated by $

Example: Grades(GDate,StudID,DiscID, Grade)

with primary key GDate+StudID+DiscID

Client creates one index on StudID,DiscID with name GradesIndStudDisc.

Insert into Grades Values ('2022.02.02',345,'DBImplem',8)

i) in data file will be Key: '2022.02.02$345$DBImplem', Value:'8'

ii) in GradesIndStudDisc index file:
Key:'345$DBImplem', Value:'2022.0202$345$DBImplem' for first insert

For second insert: Insert into Grades Values ('2022.02.20',345,'DBImplem',10)

i) in data file will be Key: '2022.02.10$345$DBImplem', Value:'10'

ii) in GradesIndStudDisc index file:
Key:'345$DBImplem',
Value:'2022.02.02$345$DBImplem#2022.02.20$345$DBImplem'

Create by default for Unique keys and Foreign keys index files. Validate the Primary, Unique Key and Foreign Key Constraints in a relational table stored in a Key-Value file using the implemented index files, see Key-valueSystemsIndex.pdf.

Foreign Key Constraint has to be checked for INSERT in child table and DELETE from parent table. For **Example**:

Groups (GroupID, Specialization, Language)

Students (StudID, GroupID, Name, Tel, email)

For INSERT statement:

Check if GroupID inserted in Students table exists in table Groups using get(GroupID) on Groups, (the get command will use automatically the B+ tree index PK : GroupID on Groups table). If GroupID does not exists in Groups table reject the INSERT statement.

For DELETE statement:

Check if GroupID deleted from Groups table exists in table Students. Use the created index on foreign key (GroupID) on table Students. If the deleted GroupID exists in table Students, reject the DELETE statement.

Check also for DROP TABLE, do not allow for user to drop a (parent) table, if it is referenced from another (child) table.

In the above example do not allow to delete table Groups.

Lab4. 10p Implement a Visual Query Designer for SELECT statement (possible join of *n* tables) or execute it from command line. Implement the **selection** and **projection**

operator with DISTINCT in SELECT, use the existing index files. You will implement WHERE conditions with AND only.

a) If there are index files on more than one attribute from WHERE use every index file.

Ex. Students(StudID, GroupID, Name, Tel, email, mark)

There is an index on GroupID and one index on mark, used both in implementing:

GroupID = 243 AND mark = 10

b) If the condition is a complex one, use index files for predicates with attributes as keys in index file, the other conditions will be processed in memory.
EX. GroupID = 243 AND mark = 10 AND email LIKE '*gmail.com'
There exists index on GroupID only, use first the index file, read the records by primary keys and put the mark = 10 AND email LIKE '*gmail.com'
Condition in memory.

c) Use index files for range queries also.

There is an index on mark, use it in implementing:

mark > 9

d) If there are composite index files, use them on prefix.

There is an index on GroupID+mark it can be used in:

GroupID = 243

It can be used implementing:

GroupID = 243 AND mark = 10

GroupID = 243 AND mark > 8

Insert 1.000.000 rows in tables, to see the difference in execution time using index file versus sequential scan.

Lab5. 10p Implement 2 **join** algorithms of

- indexed nested loop
- hash join
- sort merge join

Use the existing index files where it is possible.

Extend the execution of the SELECT statement for the join of more than too tables.

Lab6. 10p Implement group by and having, use the existing index files.

For a complex SELECT statement execute first selections, then the joins.