DBMS Final Project

University Management System

***Mukan Yerbolat 210103036***

***Arystambek Zhannurkhan 210107073***

***Mukan Zhanbolat 210103219***

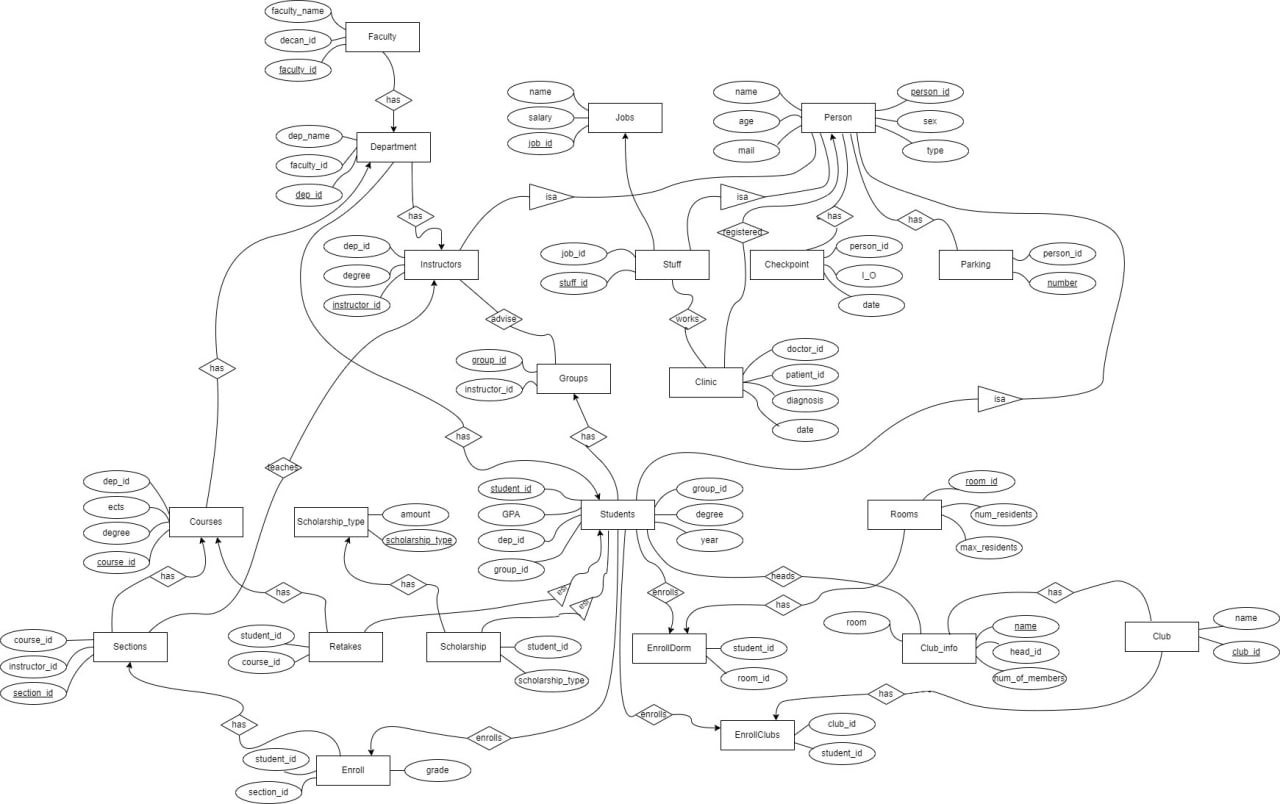
***Intoduction***

We get the idea of the project in Final Project Requirements. End users of our database is university administration, teachers and students.

**University Management System** allows you to store and manage data in a structured way, which helps to ensure that the data is accurate and consistent. This is especially important for a university, where data is often used for critical decision-making and research purposes.

Improved data accessibility. A database allows multiple users to access and update data simultaneously, which can help to improve the efficiency and productivity of the university. For example, professors can use a database to update grades, while students can use it to access their own grades and course schedules.

***Entity Relationship Diagram***



The Person entity has a one-to-one relationship with the Instructors, Stuff, and Students entities.

The Faculty entity has a one-to-many relationship with the Department entity.

The Department entity has a many-to-one relationship with the Faculty entity.

The Instructors entity has a one-to-one relationship with the Groups entity, as an instructor can have one groups.

The Students entity has a many-to-one relationship with the Groups entity.

The Jobs entity has a one-to-many relationship with the Stuff entity.

The Courses entity has a many-to-one relationship with the Department entity.

The Sections entity has a many-to-one relationship with the Courses entity.

The Enroll entity has a many-to-one relationship with the Sections entity.

The Club\_info entity has a one-to-one relationship with the Clubs entity.

The Scholarship\_type entity has a one-to-many relationship with the Scholarship entity, as multiple scholarships can be of the same type.

The Retakes entity has a many-to-one relationship with the Students and Courses entities, as multiple students can retake a single course, and a single student can retake multiple courses.

The Rooms entity has a one-to-many relationship with the EnrollDorm entity.

The EnrollDorm entity has a many-to-one relationship with the Students and Rooms entities.

The EnrollClubs entity has a one-to-many relationship with the Students and Clubs entities.

***Normalizations***

Normalization is the process of organizing a database in a way that reduces redundancy and dependency. It is a technique for designing relational database tables to minimize redundancy and dependency. There are several normal forms that can be used to normalize a database. These normal forms are:

1)First Normal Form (1NF): A table is in 1NF if it meets the following criteria:

It has a primary key, which is a unique identifier for each row in the table.

It does not have any repeating groups of data. This means that each column in the table should contain a single value, and not a list of values.

It does not have any null values.

2)Second Normal Form (2NF): A table is in 2NF if it is already in 1NF and meets the following additional criteria:

It does not have any partial dependencies. This means that every non-key column in the table must depend on the entire primary key, and not just a part of it.

3)Third Normal Form (3NF): A table is in 3NF if it is already in 2NF and meets the following additional criteria:

It does not have any transitive dependencies. This means that a non-key column in the table should not depend on another non-key column.

4)Boyce-Codd Normal Form (BCNF): A table is in BCNF if it is already in 3NF and meets the following additional criteria:It does not have any non-trivial functional dependencies on a proper subset of the primary key. This means that for every functional dependency X -> Y, where X is a proper subset of the primary key, Y must be a part of the primary key.

***All of our tables in BCNF or 1,2,3 Normal Forms. Because all of our tables satisfy rules of normalization forms.***

***Physical Design (physical is like table in database)***

All queries for creation table you can take from github [***https://github.com/yeboka/DBMS-University-Project***](https://github.com/yeboka/DBMS-University-Project)

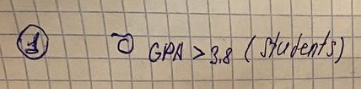
in main.sql file.

Data to fill this tables also in github

***Queries***

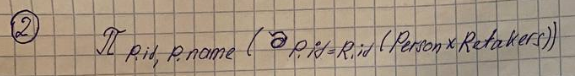
1. Select all high gpa students

select \* from students where gpa > 3.8

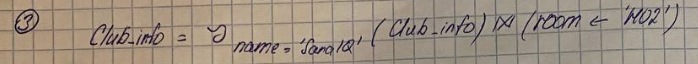


1. id and name of retakers

Select P.id, P.name from Person as P, Retakes as R where P.id = R.stud\_id

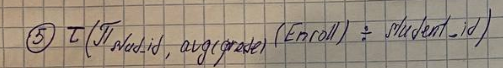


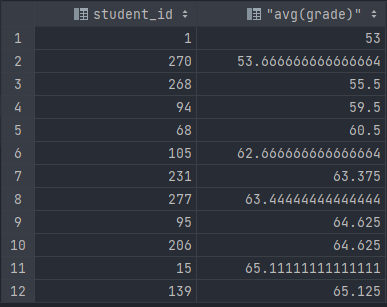
1. update Club\_info set room = 'H02' where name = 'Sana IQ'



1. Alter TABLE Student ADD COLUMN AMOUNT\_AW INT
2. Students with high avg grade on each group

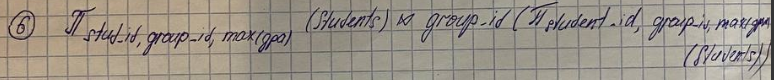
Select student\_id, avg(grade) from Enroll group by student\_id order by avg(grade)

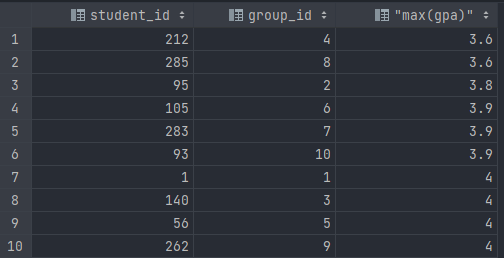




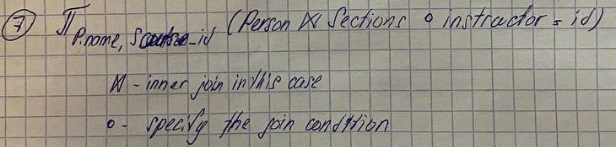
1. Best students of each groups

select student\_id,group\_id,max(gpa) from Students group by group\_id order by max(gpa)

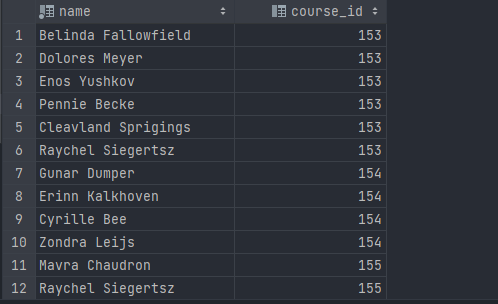




1. get name of instructors who teaching on some courses

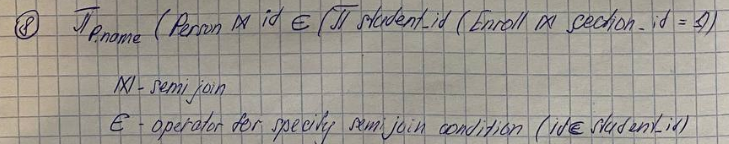


select name, Sections.course\_id from Person inner join Sections on Person.id = Sections.instructor\_id



1. Names of students from first section   
   select Person.name from Person where Person.id in

(select student\_id from Enroll where section\_id = 1)

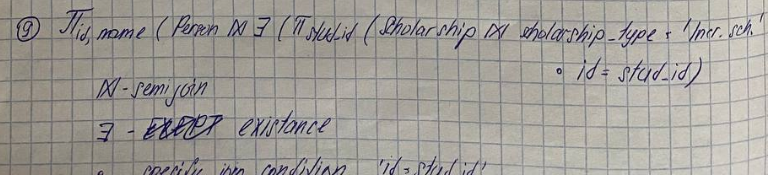


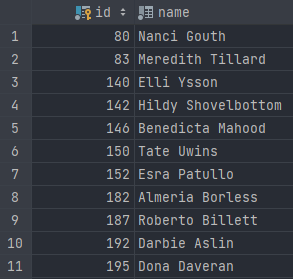


9) Select all students with increased scholarship if they exist

select id, name from Person where exists(

select stud\_id from Scholarship where Person.id = Scholarship.stud\_id and  scholarship\_type = 'Increased scholarships')

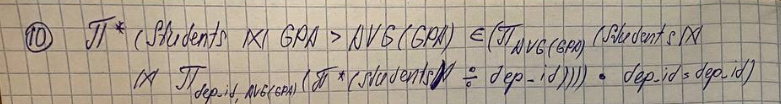




10) Find all students who have GPA greater than avg(GPA) of any department

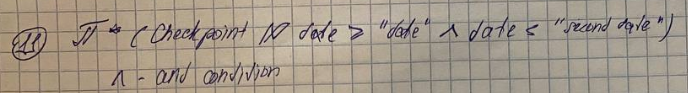
SELECT \*FROM students

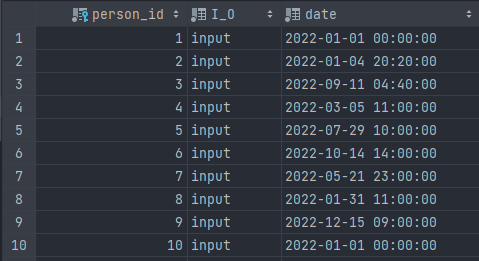
WHERE GPA> ANY (SELECT AVG(GPA) FROM StudentsGROUP BY dep\_id);



11) View all user who in to University

select \* from Checkpoint where date between '2022-01-01 00:00:00' AND '2023-02-01 23:59:59';



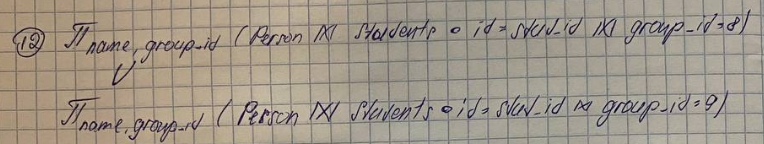


12)   Show all students who in 8 and 9 groups

select name, group\_id from Person , Students where id = Students.student\_id and group\_id = 8

union

select name, group\_id from Person , Students where id = Students.student\_id and group\_id = 9



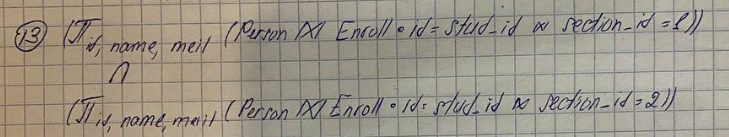


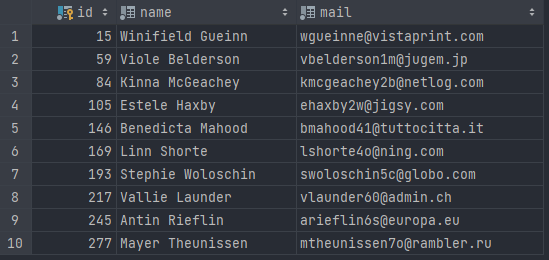
13) Find all students who have common sections 1 and 2

select id, name, mail from Person , Enroll where id = Enroll.student\_id and section\_id = 1

intersect

select id, name, mail from Person , Enroll where id = Enroll.student\_id and section\_id = 2



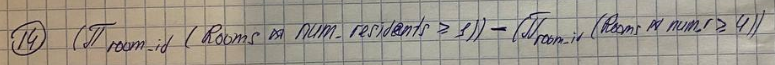


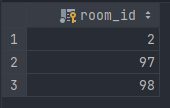
14) Find all rooms where now living more than 1 but less than 4 residents

select room\_id from Rooms where Rooms.num\_residents >= 1

except

select room\_id from Rooms where num\_residents >= 4





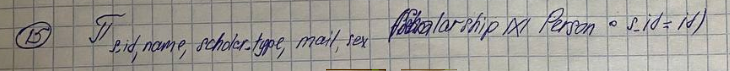
15) view which show info about students with scholarship  
create view info\_scholarship\_studs

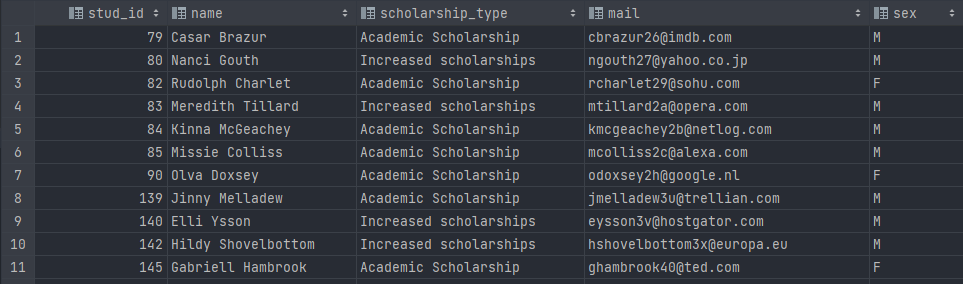
as

   select stud\_id, name, scholarship\_type, mail, sex from Scholarship, Person where id = stud\_id

to show this view

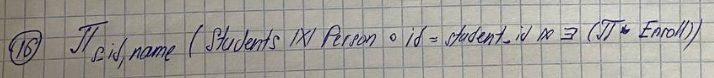
select \* from info\_scholarship\_studs

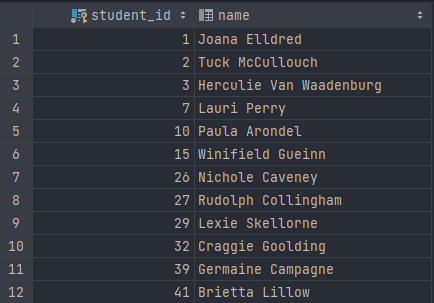




16) Show all students who has enroll at least one course

select student\_id, name from Students, Person where exists(select \* from Enroll) and Person.id = Students.student\_id





17) Trigger if the student is expelled from the university

create trigger Deduction  
 after delete on Students  
 begin  
 delete from Enroll where Enroll.student\_id = old.student\_id ;  
 delete from EnrollClubs where EnrollClubs.student\_id = old.student\_id;  
 delete from EnrollDorm where EnrollDorm.student\_id = old.student\_id;  
 end;

18) If the room is full then no one will move into that room

create trigger DeleteFromEnrollDorm  
 after insert on EnrollDorm  
 when(select num\_residents from Rooms where new.room\_id = Rooms.room\_id)  
 = (select max\_residents from Rooms where new.room\_id = Rooms.room\_id)  
 begin  
 delete from EnrollDorm where student\_id =new.student\_id;  
 end;

19) If student’s grade less than 70 then student lose his scholarship

create trigger DeleteScholarship  
 after update of grade on Enroll  
 when ((select count(student\_id) from Enroll where grade >= 70) <> (select count(student\_id) from Enroll) )  
 begin  
 delete from Scholarship where Scholarship.stud\_id = new.student\_id;  
 end;

20) If the room is not full then student can located in this room

create trigger EnrollToDorm  
 after insert on EnrollDorm  
 when(select num\_residents from Rooms where new.room\_id = Rooms.room\_id)  
 <> (select max\_residents from Rooms where new.room\_id = Rooms.room\_id)  
 begin  
 update Rooms set num\_residents = (select num\_residents from Rooms  
 where new.room\_id = Rooms.room\_id) + 1 where new.room\_id = Rooms.room\_id;  
 end;

21) Update gpa each time when students grade has updated

create trigger gpaEvaluate  
 after update of grade on Enroll  
 begin  
 update Students set GPA = (((select sum(grade) from Enroll where Enroll.student\_id = new.student\_id)  
 / (select count(grade) from Enroll where Enroll.student\_id = new.student\_id)) \* 4 / 100 )  
 where Students.student\_id = new.student\_id;  
 end;

22) If the student take retake then delete this student from scholarship table

create trigger retakers  
 after insert on Retakes  
 when(select stud\_id from Scholarship) not null  
 begin  
 delete from Scholarship where Scholarship.stud\_id = new.stud\_id ;  
 end;

***Transactions and indexes***

BEGIN TRANSACTION;  
  
INSERT INTO EnrollClubs (student\_id, club\_id)  
VALUES (123, 222);  
  
UPDATE Club\_info  
SET num\_of\_members = num\_of\_members + 1  
WHERE name = (SELECT club\_name FROM Clubs WHERE club\_id = 222);  
  
COMMIT;

CREATE INDEX my\_index\_name ON Clinic(doctor\_id, patient\_id, diagnosis, date);  
CREATE INDEX idx\_persons on Person (id, name, mail, type, sex);

***Functional dependencies:***

Key is a minimum super key in table

Person :

 {id} → {id,name,age,mail,sex,type}

Key → {id}

Super key → {name,mail} → {id,name,age,mail,sex,type}

Student:

{Stud\_id} → {stud\_id,dep\_id,group\_id, year, degree, gpa}

key - {Stud\_id}

Instructor:

{Instructor\_id} → {instructor-id,dep\_id,degree}

key - {Instructor\_id}

Staff:

{Stuff\_id} → {stuff\_id,Job\_id}

key - {Stuff\_id}

Faculty:

{Faculty\_id} → {faculty\_id,Faculty\_name, decan}

Primary key - {Faculty\_id}

super keys – {Faculty\_name,decan}

Department:

{dep\_id} → {dep\_id,dep\_name, faculty\_id}

key - {dep\_id}

super key – { dep\_name }

Courses:

{course\_id} → {course\_id,dep\_id,course\_name,ects,degree}

key - {course\_id}

super key - {dep\_id,course\_name} →{course\_id, dep\_id,course\_name,ects,degree}

Sections:

{section\_id} → {course\_id,instructor\_id}

key - {section\_id}

Enroll:

{section\_id,stud\_id} → {section\_id, stud\_id, grade}

key - {section\_id,stud\_id}

Rooms:

{room\_id} →{room\_id,num\_residents, max\_residents}

key - {room\_id}

EnrollDorm:

{Student\_id} → {Student\_id, room\_id}

key - {Student\_id}

Club\_info:

{name} → {name,head\_id, room,num\_of\_members}

{head\_id} → {name,head\_id,room,num\_of\_members}

keys - {name}, {head\_id}

Clubs:

{club\_id} → {club\_id, name}

name → {club\_id,}

keys - {club\_id},{name}

EnrollClubs:

{stud\_id} → {stud\_id, club\_id}

key - {stud\_id}

Groups:

{group\_id} → {group\_id, instructor\_id}

{instructor\_id} – {group\_id, instructor\_id}

keys - {group\_id},{instructor\_id}

Jobs:

{job\_id} → {job\_id, name, salary}

{name} → {job\_id, name, salary}

keys - {job\_id},{name}

Parking:

{number} → {number,person\_id}

{person\_id} → {number, person\_id}

Primary key → {number}, super key → {person\_id}

Checkpoint:

{person\_id} → {person\_id, I\_O, date}

key - {person\_id}

Clinic:

{id} → {id,doc\_id, patient\_id, diagnosis, date}

{doc\_id,patient\_id} – {id, doc\_id, patient\_id, diagnosis,date}

key - {id}

super key → {doc\_id,patient\_id}

Retakes:

{stud\_id} → {stud\_id,course\_id}

key - {stud\_id}

Scholarship\_type:

{scholarship\_type} → {scholarship\_type, amount}

key → {scholarship\_type}

Scholarship:

{stud\_id} → {stud\_id,scholarship\_type}

key - {stud\_id}