Project: The Call for Cthulhu Project SQL

Daniel Stein 19201398
Daniel.stein@ucdconnect.ie

Table of Contents

List of Figures

1. Introduction

The aim of this project is to create a structured database, where data about students, teachers, projects, and the student preference in projects will be stored. By storing all this information in a database, we aid the application team to find the best solutions to allocate projects to students. It is important to note that the support we are offering is merely on the access and storage of data, and we take no part in the allocation process itself.

The tables I have created for the database are:

- Student table: Here we store essential information of the students that will be used at the high-level application such as student ID, student, grades, names, and their field of studies.
- Preference table: This table displays student preferences on projects, as such the only fields needed are the student ID and a ranking system for the projects, I found that using twenty fields called Rank1 through Rank20 was the best way to structure the table, where each Rank field will display the name of a project, Rank1 being the most preferred and Rank20 the least. This table is the key to assigning project and measuring student satisfaction in the long run.
- Project table: Here all the information about each project is stored, those fields being project ID, title, teacher responsible for the project, teacher ID, and the field of studied the project falls under.
- Teacher table: Contains information about each teacher such as ID, name, field of studies, email, and phone number.
- Finally, we have the project assigned table, where we can display the final solution form the application team after projects have been allocated. This table will get student ID and project ID from the high-level application and fill the other fields such as names and IDs from neighboring tables.

There are some assumptions we are making that might not be mentioned in the white paper:

- Students are not obliged to fill on their preferred projects. Teachers will not always have the same number of projects to supervise.

- The final allocated project table will be stored in this database after the application team has completed allocation, then all fields will be filled by using data from separate tables.
- We are assuming the allocation is based mainly on GPA standing (students with higher GPA are more likely to get their preferred project), hence students are allowed to have repeats in their preferences or "cheat". This should not affect the allocation as if a better performing student selected the same project, they are still more likely to get that project, however, it might be useful to identify students who have tried to cheat the system.
- Students will not create their own projects, only professors.

The database stores data related to students, teachers, and projects, giving a unique numerical identifier for each. This is to avoid complications in the scenario that there are shared names between teachers or students, and as for the project, it is easier to select a specific project using an ID than by writing the whole name. All these steps were taking in consideration of the white paper as well as support and facilitate the process for the high-level application and the team.

2. Database Plan: A Schematic View

The main role of this database is to assist the high-level application team in order to determine the most suitable allocation of project for each student. As such it is imperative that the database is not cluttered with unusable data.

The main entities in this database are student, teachers, projects, and stream. Entities does not refer to the number of tables, as there are six tables: Student table, preference table, teacher table, project table, Stream table, and final assigned projects table. Entities are more conceptual while tables are physical construct in the database.

The main attribute for each entity I believe is the following:

Student: It requires a unique **Student ID** which can be used to identify each student and will be assigned a Primary Key. We also need **GPA** or grades, as it is from this that the application team can determine who is more deserving of their top picks. We also decided to add as standard a **Full_Name** for the students.

Another important attribute is the **Studies**, because students are enrolled in 3 different Streams and each student can only apply for projects which match with their own studies.

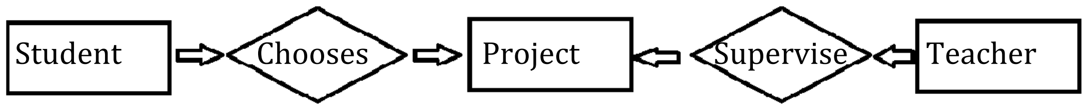
Project: Like with students, each project needs an identifier **ProjectID** which is the Primary Key. The only other attributes we need to describe this entity are a **project title** and a **Studies field**. The field attribute will tell us if a student is allowed to take certain studies or not, this is based on if there is a match between the student studies and the project studies.

Stream: This entity has only two attributes, **Stream ID** which is the primary key and a unique identifier, and **Study** which is the different studies offered in the university. This table is needed to follow normalization rules that will be explained later on.

Teachers: They have a unique **TeacherID** which is the primary key, as well as name, **field of studies** and contact information such as **Email** and **Phone**. We found that having contact information could be useful for the end result.

There are weak entities in this database such as Preference, which main purpose is to hold information belonging to the students, if students don't have any preference then this table would be empty, so it is completely dependent on Students.

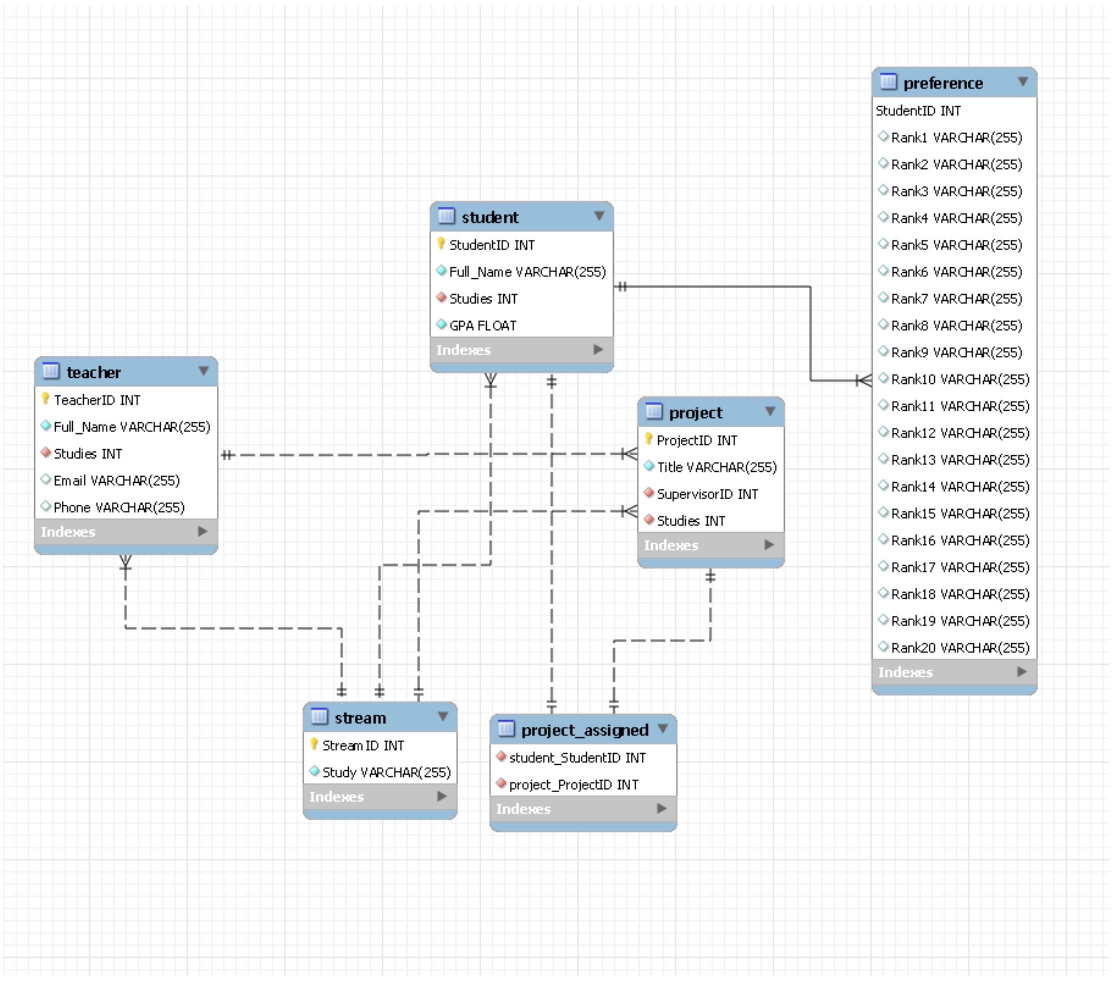
Figure 1.



The relationship between the entities are as shown above. However, to further explain the relationship between entities we must also take into account every table in the database. Shown below is the ER Diagram from the database, from there we can describe the relationship between all entities and tables: Students must have a one to one relationship with the final allocated project, meaning that the end result will display a unique student ID with a unique project ID. However, students can input their preference towards projects shown in the table

Preference, in here we can see that students have a one to many relationship, which could actually be a none to many relationship, as students can choose many projects as their preferred or disliked, or leave it empty. This step is needed as the application team determine the allocation based on the students' preference and GPA.

Figure 2.



There is a one to many relationship between teachers and projects, as each teacher supervises many projects as well as an indirect one to many relationship between teachers and students.

There are two different project tables, one which stores project information from the start (such as ID, title, studies) and one which is the final allocation of one unique project per students. These have a one to one relationship as each project will appear once in the final allocated table.

The Stream table has a one two many relationship with the other entities, as many projects, students, and teachers will have one field of study.

3. Database Structure: A Normalized View

The main tables are Student, Preference, Project, Assigned Project, Teacher, and Stream. I will give a description of the main tables below:

Student: This table contains vital information of the students that will help in the allocation process:

- This data is comprised of a unique Student ID that cannot be repeated int the table and it is the primary key. By doing so we are following the 1NF, which says there cannot be two repeated entries in the database. The only way the rule could be broken is if the same student information is inserted twice with a different student ID, which is very unlikely. To avoid this scenario a trigger or a single delete function could be put in place to detect whenever all the information exists in the table already.

- This table also contains the student name which is a sort of default field, as students are not just numbers.

- Studies: This field is needed to analyze whether the student may take a project or not, as both students and projects have this field, if they match, then the student can be allocated the project. This is a foreign key referring to the Stream table. I have decided to separate the field of study to conform with the 3NF and BCNF, as if a study field was to change then we could change it in the Stream table and it will be reflected across all other tables instead of changing the Studies field for each table.

- GPA: This field displays the overall grade for each student, and this field will help determine which students get their first choices based on how well they did.

Preference: This table's purpose is to assist the Student table and to help conform with the rules of normalization. This information together with GPA will give the application team the resources to determine whether a student gets a project or not.

- Student ID: This field is a foreign and primary key in this table, as each student will have one preference range. This means that no student should appear twice in this table, and why making Student ID into the primary key Conforms with 1NF, 2NF and 3NF.
- Rank1 to Rank20: these 20 fields are designed to show a student's preference across projects, they can choose all or none. The table will be updated to remove any project that does not match with a student.

Project: This table holds all information regarding projects available. By using these together with Stream, Preference, and GPA, the application team can design an efficient allocation process for projects.

- Project ID: a unique identifier and primary key that helps table conform with 1NF.
- Title: a name to the project that is used to match this table with student preference table. It is default to include this just like it is for names in students.
- Supervisor ID: This shows which teacher is responsible for a given project by having the ID be a foreign key.
- Studies: Is a foreign key that refers to the Stream, as mentioned before this is to conform with the 3NF and BCNF.

Teacher: This table holds the information for teachers, it is needed to assign a supervisor for a project.

- Teacher ID: a unique identifier and primary key that helps conform with 1NF.
- Name: is a default field.
- Studies: Is a foreign key that refers to the Stream and conforms with 3NF.
- Email/Phone: are contact information that might be useful to have for the application team. This conforms to the 2NF.

Stream: This table was created to conform with 3NF, as if a study name was to change it would mean changing across three different tables, however, if we have one table that all students, teachers, and projects refer to then a single change in this table is enough to update all tables.

- Stream ID: Unique identifier and primary key that conforms with the 1NF.
- Study: Name of the project that conforms with the 2NF as it is fully functional dependent on the primary key.

Project Assigned: This table is more of a final product where the application team has submitted the student project combination. This table serves as storage that of data that will be used for other applications.

- Student ID: Unique student ID which I chose to be the primary key of this table as well as being the primary key.
- Project ID: Unique project ID which is another foreign key.

This table is used to connect with other tables and create views or functions that further help the application team.

This database conforms with 1NF as each entity has no repeated entries and each table has a unique primary key.

It conforms with 2NF as there is partial dependencies between fields on the primary key.

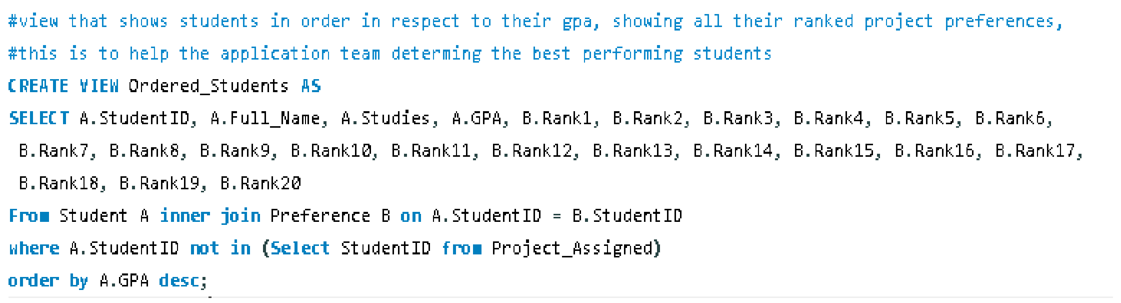
And it conforms with the 3NF as every field is dependent only on the primary key.

I believe it complies with BCNF as all tables are broken in such a way that X -> X, Y.

4. Database Views

I have created a total of four views in my database, I will explain the purpose of each one below:

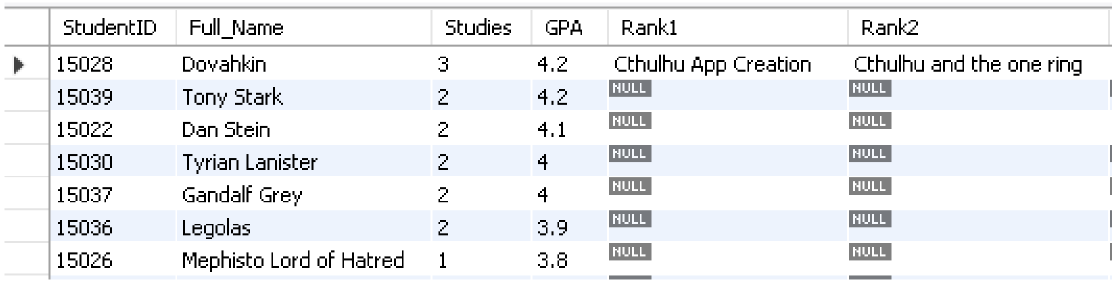
Figure 3.



```
#view that shows students in order in respect to their gpa, showing all their ranked project preferences,
#this is to help the application team determing the best performing students
CREATE VIEW Ordered_Students AS
SELECT A.StudentID, A.Full_Name, A.Studies, A.GPA, B.Rank1, B.Rank2, B.Rank3, B.Rank4, B.Rank5, B.Rank6,
 B.Rank7, B.Rank8, B.Rank9, B.Rank10, B.Rank11, B.Rank12, B.Rank13, B.Rank14, B.Rank15, B.Rank16, B.Rank17,
 B.Rank18, B.Rank19, B.Rank20
From Student A inner join Preference B on A.StudentID = B.StudentID
where A.StudentID not in (Select StudentID from Project_Assigned)
order by A.GPA desc;
```

This is an extract of what we would find inside the view:

Figure 4.



StudentID	Full_Name	Studies	GPA	Rank1	Rank2
15028	Dovahkin	3	4.2	Cthulhu App Creation	Cthulhu and the one ring
15039	Tony Stark	2	4.2	NULL	NULL
15022	Dan Stein	2	4.1	NULL	NULL
15030	Tyrian Lanister	2	4	NULL	NULL
15037	Gandalf Grey	2	4	NULL	NULL
15036	Legolas	2	3.9	NULL	NULL
15026	Mephisto Lord of Hatred	1	3.8	NULL	NULL

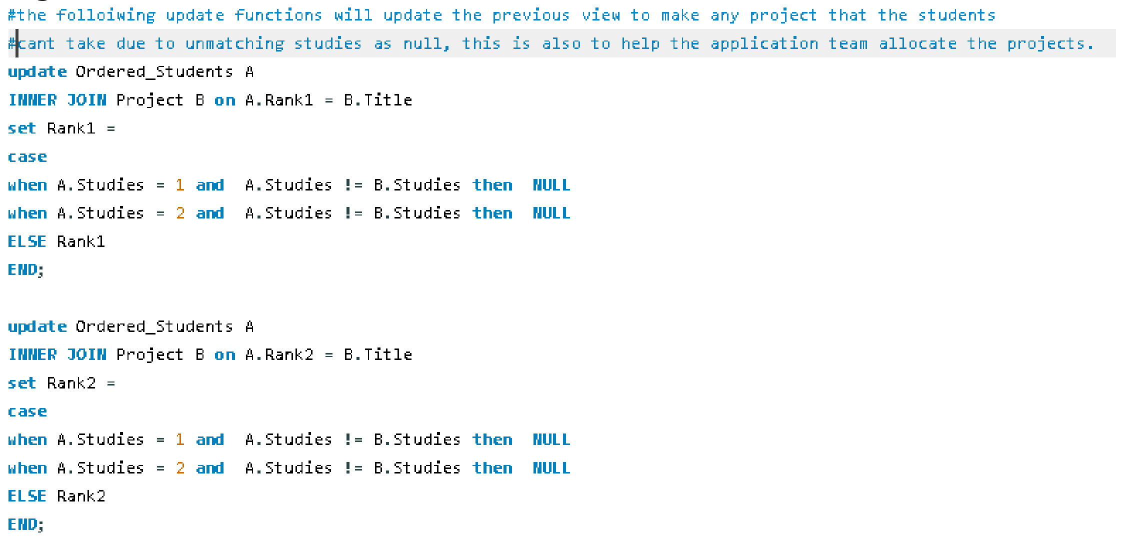
The first view I created was to put together students and the preferences together and ordering them by GPA. We can use this view to help the application team have almost all the information needed to allocate the project.

As we see above, the view will stay ordered according to GPA, which facilitates the allocation process further.

This view takes all the relevant information from the student and preference tables and combines them while ordering according to GPA. This required an inner join connecting the two tables on Student ID. This means that we can put together all the information belonging to one Student ID from two tables into one.

We also have a constraint where if a student already exists in the final assigned project table then it will not be included in this view. This is to avoid having old students who had their projects allocated put together with new students in the future. This way we can keep adding to all tables without assigning projects to the wrong students.

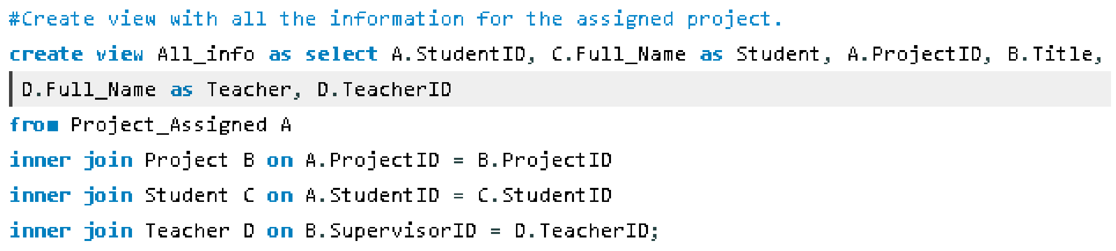
Figure 5.



```
#the following update functions will update the previous view to make any project that the students
#cant take due to unmatching studies as null, this is also to help the application team allocate the projects.
update Ordered_Students A
INNER JOIN Project B on A.Rank1 = B.Title
set Rank1 =
case
when A.Studies = 1 and  A.Studies != B.Studies then  NULL
when A.Studies = 2 and  A.Studies != B.Studies then  NULL
ELSE Rank1
END;

update Ordered_Students A
INNER JOIN Project B on A.Rank2 = B.Title
set Rank2 =
case
when A.Studies = 1 and  A.Studies != B.Studies then  NULL
when A.Studies = 2 and  A.Studies != B.Studies then  NULL
ELSE Rank2
END;
```

We must then update the view so that students who are doing certain studies can only take projects corresponding those studies. We achieve this with 20 update functions going from rank 1 to 20 and making any project with unmatching stream into a null value. This step is important for the application team as we do not want students taking incorrect projects.

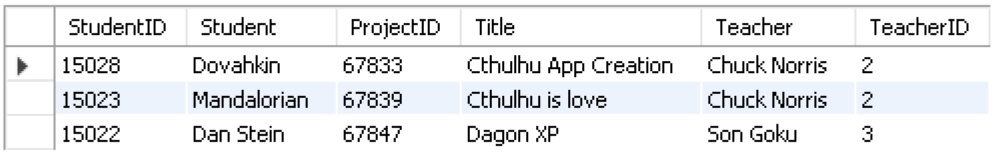
Figure 6.



```
#Create view with all the information for the assigned project.
create view All_info as select A.StudentID, C.Full_Name as Student, A.ProjectID, B.Title,
 D.Full_Name as Teacher, D.TeacherID
from Project_Assigned A
inner join Project B on A.ProjectID = B.ProjectID
inner join Student C on A.StudentID = C.StudentID
inner join Teacher D on B.SupervisorID = D.TeacherID;
```

This is an extract of what we would find inside the view:

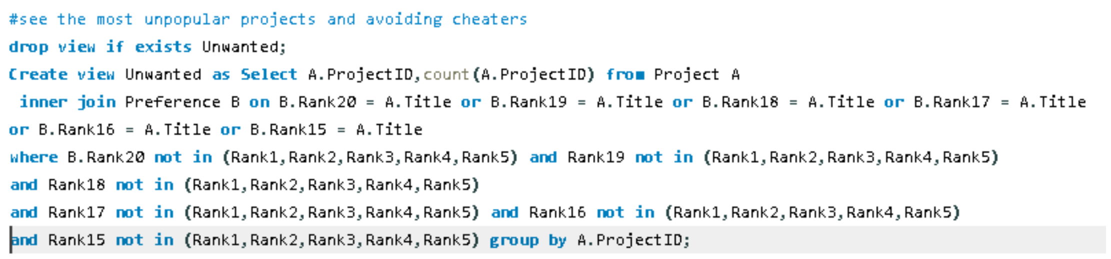
Figure 7.



	StudentID	Student	ProjectID	Title	Teacher	TeacherID
▶	15028	Dovahkin	67833	Cthulhu App Creation	Chuck Norris	2
	15023	Mandalorian	67839	Cthulhu is love	Chuck Norris	2
	15022	Dan Stein	67847	Dagon XP	Son Goku	3

I created this view in order to do other functions and queries easily later on. This view holds the IDs and names of student, teacher and projects. I later use this view to calculate satisfaction levels, and to do procedures. I chose to create this view because it holds all the information that we constantly use throughout this database.

Figure 8.

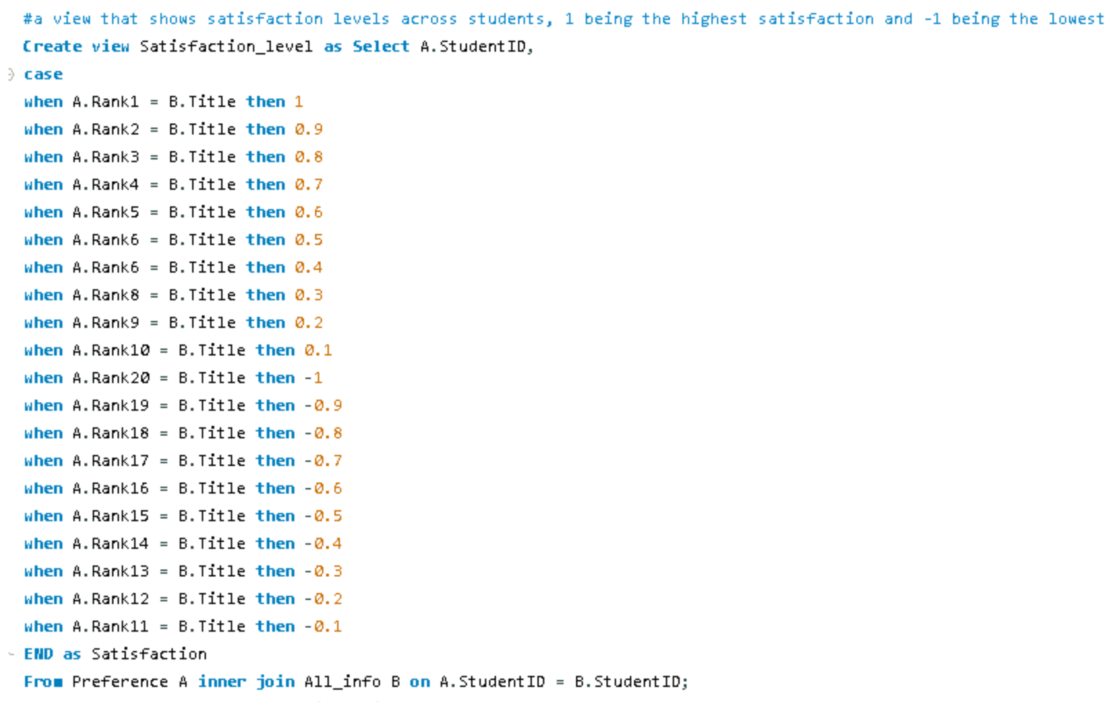


```
#see the most unpopular projects and avoiding cheaters
drop view if exists Unwanted;
Create view Unwanted as Select A.ProjectID,count(A.ProjectID) from Project A
 inner join Preference B on B.Rank20 = A.Title or B.Rank19 = A.Title or B.Rank18 = A.Title or B.Rank17 = A.Title
or B.Rank16 = A.Title or B.Rank15 = A.Title
where B.Rank20 not in (Rank1,Rank2,Rank3,Rank4,Rank5) and Rank19 not in (Rank1,Rank2,Rank3,Rank4,Rank5)
and Rank18 not in (Rank1,Rank2,Rank3,Rank4,Rank5)
and Rank17 not in (Rank1,Rank2,Rank3,Rank4,Rank5) and Rank16 not in (Rank1,Rank2,Rank3,Rank4,Rank5)
and Rank15 not in (Rank1,Rank2,Rank3,Rank4,Rank5) group by A.ProjectID;
```

This view shows the least preferred projects in the database. This is based on which projects consistently landed on the lowest ranks for the greatest number of students, while making sure that they did not appear in the top picks for the same students (cheaters).

This information is useful for the application team, as it could add to the allocation process to know which projects are most disliked, as well as allowing the university to change the disliked projects for different ones in the future.

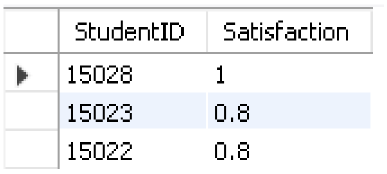
Figure 9.



```
#a view that shows satisfaction levels across students, 1 being the highest satisfaction and -1 being the lowest
Create view Satisfaction_level as Select A.StudentID,
case
when A.Rank1 = B.Title then 1
when A.Rank2 = B.Title then 0.9
when A.Rank3 = B.Title then 0.8
when A.Rank4 = B.Title then 0.7
when A.Rank5 = B.Title then 0.6
when A.Rank6 = B.Title then 0.5
when A.Rank6 = B.Title then 0.4
when A.Rank8 = B.Title then 0.3
when A.Rank9 = B.Title then 0.2
when A.Rank10 = B.Title then 0.1
when A.Rank20 = B.Title then -1
when A.Rank19 = B.Title then -0.9
when A.Rank18 = B.Title then -0.8
when A.Rank17 = B.Title then -0.7
when A.Rank16 = B.Title then -0.6
when A.Rank15 = B.Title then -0.5
when A.Rank14 = B.Title then -0.4
when A.Rank13 = B.Title then -0.3
when A.Rank12 = B.Title then -0.2
when A.Rank11 = B.Title then -0.1
END as Satisfaction
From Preference A inner join All_info B on A.StudentID = B.StudentID;
```

This is an extract of what we would find inside the view:

Figure 10.



StudentID	Satisfaction
15028	1
15023	0.8
15022	0.8

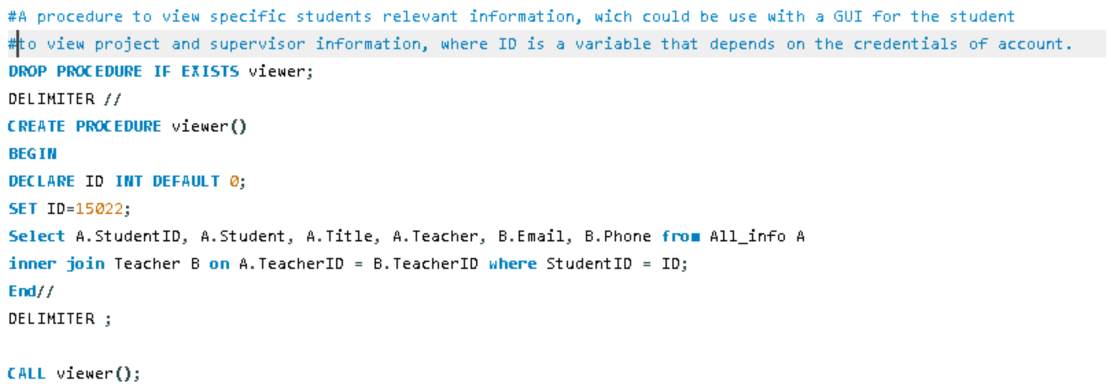
Next, we have a very useful view that will show us student satisfaction from my point of view. This means how did the student's actual assigned project matched with their preference. Not all students will appear in this view as it is not mandatory for students to fill in the preference table.

Satisfaction levels go from -1 being completely unsatisfied to 1 which is perfect satisfaction. A student will get a -1 if the project the had at ranked 20 is the project assigned to them, while they will have a satisfaction level of 1 if their number one choice is the one assigned.

5. Procedural Elements

I have two working procedural elements and one extra one that I found very useful as a concept but was unable to finish it:

Figure 11.



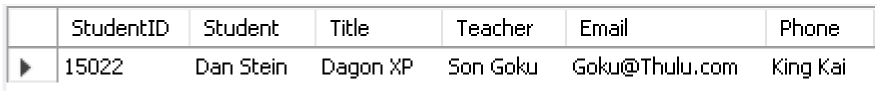
```
#A procedure to view specific students relevant information, wich could be use with a GUI for the student
#to view project and supervisor information, where ID is a variable that depends on the credentials of account.
DROP PROCEDURE IF EXISTS viewer;
DELIMITER //
CREATE PROCEDURE viewer()
BEGIN
DECLARE ID INT DEFAULT 0;
SET ID=15022;
Select A.StudentID, A.Student, A.Title, A.Teacher, B.Email, B.Phone from All_info A
inner join Teacher B on A.TeacherID = B.TeacherID where StudentID = ID;
End//
DELIMITER ;

CALL viewer();
```

I think this procedure would be useful in the case where a student was to be able to access the database to see their project and supervisor information. In here we have a variable ID which ideally would be set as the student ID of the current user, and it will display all the relevant information of the assigned project and supervisor. This procedure does not have any direct benefit to the application team, but it would give further use to the database.

The reason this had to be made into a procedure was because of the ID variable. Since it is the best way to store a variable in MySQL.

Output after calling procedure:

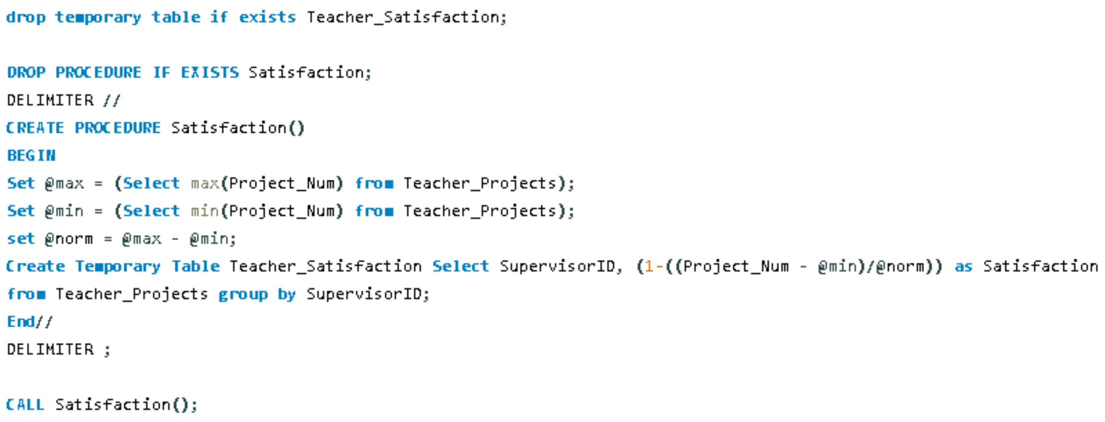
Figure 12.



	StudentID	Student	Title	Teacher	Email	Phone
▶	15022	Dan Stein	Dagon XP	Son Goku	Goku@Thulu.com	King Kai

The motivation behind this procedure was to give further use to the database, plus it is an item that could be used by the application team if they were in need for any of these data for a specific student.

Figure 13.



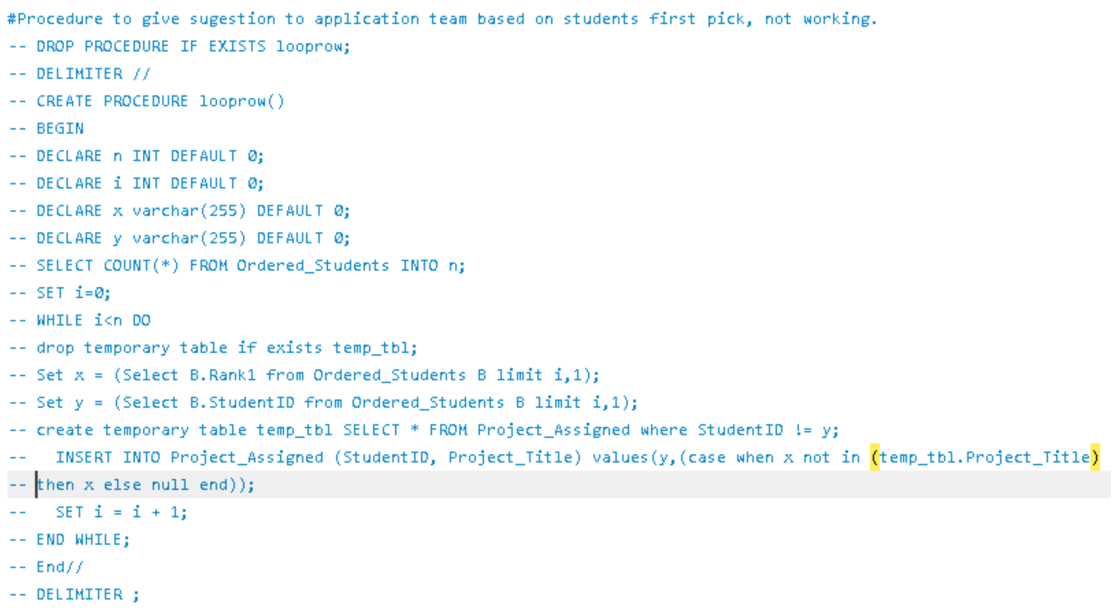
```
drop temporary table if exists Teacher_Satisfaction;

DROP PROCEDURE IF EXISTS Satisfaction;
DELIMITER //
CREATE PROCEDURE Satisfaction()
BEGIN
Set @max = (Select max(Project_Num) from Teacher_Projects);
Set @min = (Select min(Project_Num) from Teacher_Projects);
set @norm = @max - @min;
Create Temporary Table Teacher_Satisfaction Select SupervisorID, (1-((Project_Num - @min)/@norm)) as Satisfaction
from Teacher_Projects group by SupervisorID;
End//
DELIMITER ;

CALL Satisfaction();
```

Here we have a procedure that calculates teacher satisfaction. The reason I made it into a procedure was because selects did not work with max plus counts, and I would have to have two to three views to make this work, however, with a procedure I can set all the variables for the min max normalization within and create a single temporary table (views cannot hold variables so it has to be a temporary table.

I achieve this by doing a min max normalization on the count of projects. If a teacher has the lowest number of projects then he will have a 0, if they have the greatest count for projects then it is a 1. I inversed this number by doing 1 – X , which then makes it so the teacher with the lowest number of projects would be the happiest.

This satisfaction view only works because I assume teachers will be responsible for the same number of projects. If all the teachers have the same number of projects the equation would not work.

Figure 14.



```
#Procedure to give sugestion to application team based on students first pick, not working.
-- DROP PROCEDURE IF EXISTS looprow;
-- DELIMITER //
-- CREATE PROCEDURE looprow()
-- BEGIN
-- DECLARE n INT DEFAULT 0;
-- DECLARE i INT DEFAULT 0;
-- DECLARE x varchar(255) DEFAULT 0;
-- DECLARE y varchar(255) DEFAULT 0;
-- SELECT COUNT(*) FROM Ordered_Students INTO n;
-- SET i=0;
-- WHILE i<n DO
-- drop temporary table if exists temp_tbl;
-- Set x = (Select B.Rank1 from Ordered_Students B limit i,1);
-- Set y = (Select B.StudentID from Ordered_Students B limit i,1);
-- create temporary table temp_tbl SELECT * FROM Project_Assigned where StudentID != y;
--   INSERT INTO Project_Assigned (StudentID, Project_Title) values(y,(case when x not in (temp_tbl.Project_Title)
-- then x else null end));
--   SET i = i + 1;
-- END WHILE;
-- End//
-- DELIMITER ;
```

Unfortunately, I did not manage to make this procedure work. The idea behind this was to give a suggestion project for some students to the application team, which would have simplified the allocation process.

This procedure would go row by row in the in the ordered student view and check the rank 1 project for each student (Starting with the highest GPA). As the procedure went down through the rows counting them (i) it will check if any higher GPA student have already selected the same rank 1 project, if so, the lower GPA student will not get the project and it will be set as null. Then we could apply the same logic through all ranks up to rank 10(after ranked 10 it is disliked projects). This could potentially give all students with a preferred project a chance to get that project, and leave null for students who did not have a preference.

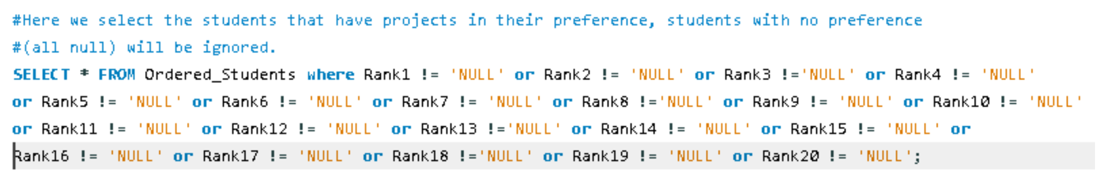
6. Example Queries: Your Database In Action

Throughout this database, it was always a debate whether I make a query or create a view. For data I found was important to the application team I made it into a view. From there, it would be easy to make queries from said views by using

SELECT * FROM (view).

There are other queries that could give relevant information, but did not need to be made into views:

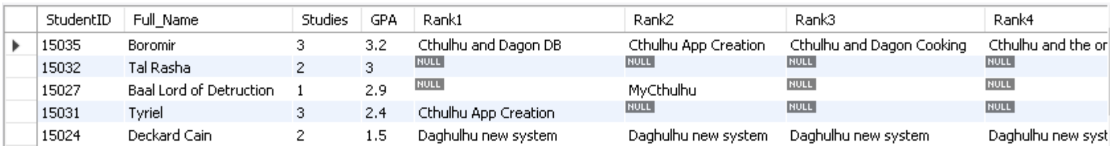
Figure 15.



```
#Here we select the students that have projects in their preference, students with no preference
#(all null) will be ignored.
SELECT * FROM Ordered_Students where Rank1 != 'NULL' or Rank2 != 'NULL' or Rank3 !='NULL' or Rank4 != 'NULL'
or Rank5 != 'NULL' or Rank6 != 'NULL' or Rank7 != 'NULL' or Rank8 !='NULL' or Rank9 != 'NULL' or Rank10 != 'NULL'
or Rank11 != 'NULL' or Rank12 != 'NULL' or Rank13 !='NULL' or Rank14 != 'NULL' or Rank15 != 'NULL' or
Rank16 != 'NULL' or Rank17 != 'NULL' or Rank18 !='NULL' or Rank19 != 'NULL' or Rank20 != 'NULL';
```

This query utilizes the first view in our database holding all students and preferred projects where they don't exist in the assigned table and where they have matching streams. With this query we can further filter the data for the application team by only showing students with a non-null value as a preferred project (excluding students with no preferences). This make it easier for the application team to read the data since it is not a cluttered with null values.

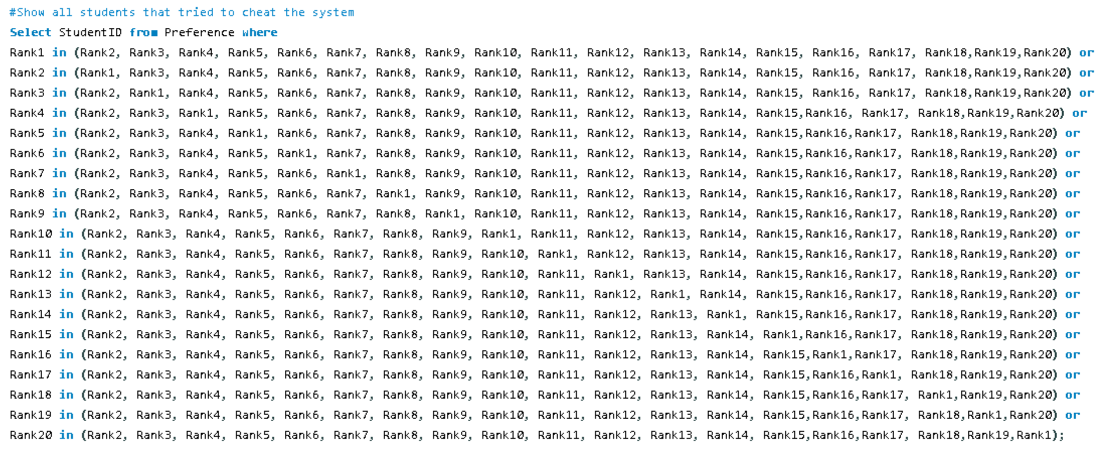
Figure 16.



	StudentID	Full_Name	Studies	GPA	Rank1	Rank2	Rank3	Rank4
▶	15035	Boromir	3	3.2	Cthulhu and Dagon DB	Cthulhu App Creation	Cthulhu and Dagon Cooking	Cthulhu and the or
	15032	Tal Rasha	2	3	NULL	NULL	NULL	NULL
	15027	Baal Lord of Detruction	1	2.9	NULL	MyCthulhu	NULL	NULL
	15031	Tyriel	3	2.4	Cthulhu App Creation	NULL	NULL	NULL
	15024	Deckard Cain	2	1.5	Daghulhu new system	Daghulhu new system	Daghulhu new system	Daghulhu new syst

Students may only indicate their disliked projects by inputting only in rank 11 -20 and leaving the rest null. It is important to note that this is after updating the view and deleting projects that students don't match with.

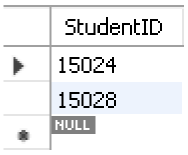
Figure 17.



```
#Show all students that tried to cheat the system
Select StudentID from Preference where
Rank1 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15, Rank16, Rank17, Rank18,Rank19,Rank20) or
Rank2 in (Rank1, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15, Rank16, Rank17, Rank18,Rank19,Rank20) or
Rank3 in (Rank2, Rank1, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15, Rank16, Rank17, Rank18,Rank19,Rank20) or
Rank4 in (Rank2, Rank3, Rank1, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15,Rank16, Rank17, Rank18,Rank19,Rank20) or
Rank5 in (Rank2, Rank3, Rank4, Rank1, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15,Rank16,Rank17, Rank18,Rank19,Rank20) or
Rank6 in (Rank2, Rank3, Rank4, Rank5, Rank1, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15,Rank16,Rank17, Rank18,Rank19,Rank20) or
Rank7 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank1, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15,Rank16,Rank17, Rank18,Rank19,Rank20) or
Rank8 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank1, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15,Rank16,Rank17, Rank18,Rank19,Rank20) or
Rank9 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank1, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15,Rank16,Rank17, Rank18,Rank19,Rank20) or
Rank10 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank1, Rank11, Rank12, Rank13, Rank14, Rank15,Rank16,Rank17, Rank18,Rank19,Rank20) or
Rank11 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank1, Rank12, Rank13, Rank14, Rank15,Rank16,Rank17, Rank18,Rank19,Rank20) or
Rank12 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank1, Rank13, Rank14, Rank15,Rank16,Rank17, Rank18,Rank19,Rank20) or
Rank13 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank1, Rank14, Rank15,Rank16,Rank17, Rank18,Rank19,Rank20) or
Rank14 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank1, Rank15,Rank16,Rank17, Rank18,Rank19,Rank20) or
Rank15 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank1,Rank16,Rank17, Rank18,Rank19,Rank20) or
Rank16 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15,Rank1,Rank17, Rank18,Rank19,Rank20) or
Rank17 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15,Rank16,Rank1, Rank18,Rank19,Rank20) or
Rank18 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15,Rank16,Rank17, Rank1,Rank19,Rank20) or
Rank19 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15,Rank16,Rank17, Rank18,Rank1,Rank20) or
Rank20 in (Rank2, Rank3, Rank4, Rank5, Rank6, Rank7, Rank8, Rank9, Rank10, Rank11, Rank12, Rank13, Rank14, Rank15,Rank16,Rank17, Rank18,Rank19,Rank1);
```

Above we can do a query to find which students tried to cheat the system. If any of the projects appear more than once in the preference table then their ID will be displayed here:

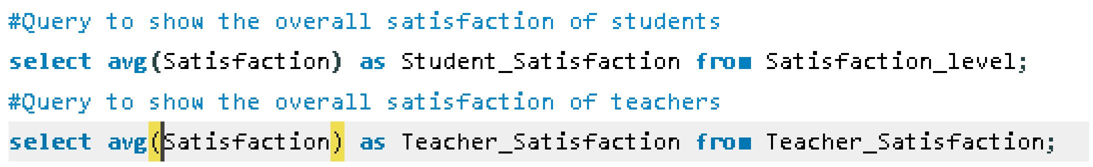
Figure 18.



StudentID
15024
15028
NULL

As we can see, these are indeed the only two students who had one project input more than once in their preference. This information could be useful if the application team decided to punish any attempts of cheating.

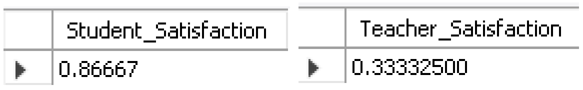
Figure 19.



```
#Query to show the overall satisfaction of students
select avg(Satisfaction) as Student_Satisfaction from Satisfaction_level;
#Query to show the overall satisfaction of teachers
select avg(Satisfaction) as Teacher_Satisfaction from Teacher_Satisfaction;
```

We can use queries to display the average or overall satisfaction for students and teachers from the satisfaction views.

Figure 20.

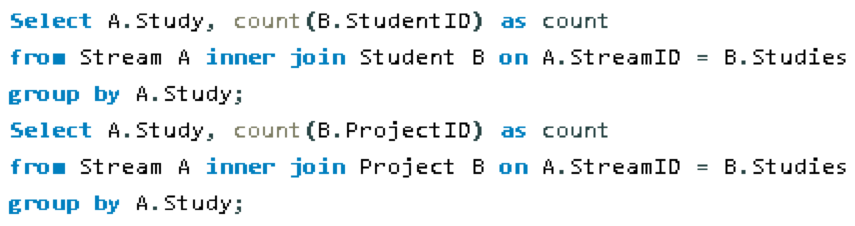


Student_Satisfaction
0.86667

Teacher_Satisfaction
0.33332500

This can indicate: 1. If the allocation process was satisfactory to students preference. 2. If there is a big disparity in number of projects supervised by teachers.

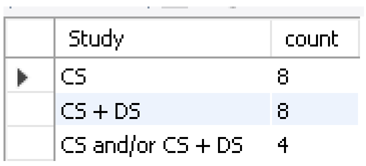
Figure 21.



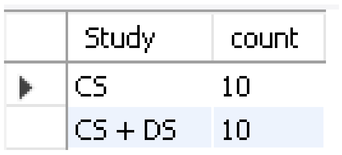
```
Select A.Study, count(B.StudentID) as count
from Stream A inner join Student B on A.StreamID = B.Studies
group by A.Study;
Select A.Study, count(B.ProjectID) as count
from Stream A inner join Project B on A.StreamID = B.Studies
group by A.Study;
```

We can also use queries to find how many students are in each stream:

Figure 22.



Study	count
CS	8
CS + DS	8
CS and/or CS + DS	4



Study	count
CS	10
CS + DS	10

With this we can determine if the number of student streams match the number of projects. As we can see as long as Student streams "CS" and "CS + DS" does not exeec the project stream with the same name and the remainder fall under "CS and/or CS + DS" there will be no issues. These simple queries could be use to check if there is balance in the database.

7. Conclusions

In conclusion I have shown how my tables conforms with 1NF, 2NF, 3NF and CBNF to the best of my ability with a detailed explanation of each table and attribute and how such data does indeed help the application team develop an allocation process for projects. We have also seen useful views, updates and procedures that further aid the application team in their quest to find the best allocation method. Together with useful queries that would see this database in action.

This database could be used for more than assisting the application team. As shown already, we could implement a procedure that allows Students to view their assigned projects as well as all the supervisor information. This could be easily done with the procedure provided in section 5.

This database could also be used by the university to determine which are the most unwanted projects. By taking the count of project ranked 15 or higher, the university could change the project format for the next semesters, which in time should increase satisfaction level across students.

This database also makes it very easy to expand the number of streams, which would be a great benefit with a growing university.

Acknowledgements

I acknowledge that this project is entirely my own work and it was written uniquely by me. I researched to improve my code and queries using websites: Stack Overflow to learn how to set variables in procedures:

https://stackoverflow.com/questions/11754781/how-to-declare-a-variable-in-mysql

MySQL documentation website for procedure syntax:

https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

References

There are no quotes in my written project. All code research websites were mentioned in the acknowledgements section.