The Call for Cthulhu Projects Database

Project Report

COMP40725-Introduction to RDBMS & SQL Programming

Farooq Shaikh
SN: 19200161
Email: farooq.shaikh@ucdconnect.ie
University College Dublin

Table of Contents

List of Figures

1. Introduction: Project Overview

1.1 Background

In, Miskatonic University, Philipp Howards took over the charge as the new Dean of the school of Cthulhu Studies (CS) and he wanted to supplant the current system of assigning Final year Projects to the student with a newer and more efficient system. A system that would keep satisfaction of to the students and the Supervisors at the forefront. The goal was to ensure that we allow students to provide preferences for the Projects that they are interested in by expressing up to 20 projects and build a system that would assign each student a unique project taking into consideration the preference expressed and his/her GPA. The Dean wanted a system that would be fair to every student and yet ensure that everyone is happy at the end of it. The entire system for allocation of the projects to student comprises of many components such a front end that a student interacts and where he/she expresses his preferences, a genetic algorithm that takes all the data and assigns each student a unique project ensuring that the student satisfaction is ensured and a Data base that supports the entire system by providing necessary tables to ensure efficient storage and retrieval of large amounts of Data.

In this report, we will be focusing only on the Database side of the project and understand how the underlying structure would be supporting the overall system that Dr Phillip had planned.

1.2 Goal

The primary objective of this project is to design and create a database to centrally store all the information about the students, the supervisors (teachers), and all the information that would be needed by the Project Allocation System for assigning of the projects to the students. The aim is to provide a supporting structure in the form of a database to store and provide access to all the above mentioned information with an ease that can be accessed by other components of the overall system including the web-based interface.

1.3 Data Storage Requirements and Scope

Data Storage Requirements for the system fall into three categories; the one that deals with storing the information about each student and the staff enrolled in the university, the streams offered at the university etc. This a broader global information that we need to store in the student and is common for various applications. The two categories are restricted to the current task of allocation of the project. One deals with the all the available projects, their origin (who proposed them), the storage of the Student Preferences etc. and the final category is the end result of the entire system which essentially is one-to-one mapping of students and the projects. In the final category we also meticulously store finer

details such the which preference each student was allocated and the satisfaction index of each student as well as the supervisor.

A high-level summary of the data we are interested in storing is summarized below:

1. Student Data: all the data related to each student, example their full name, GPA (grade point aggregate), their stream
2. Supervisor Data: All the details about the supervisors available at the University
3. Details of the Available Project: The title of project, Project ID, which stream does it belong to (one of the following: CS-only, CS+DSonly, or CS and/or CS+DS), and the details as to who proposed the Project; A project can either be proposed by the Miskatonic teaching staff or the student himself (after consulting with the supervisor)
4. Preferences provided by the Student: Project preference matrix that stores up to 20 ranked preferences provided by each student.
5. Allocation Details: The Details about the one-to-one mapping of each project to a student
6. Satisfaction Parameters: As the primary aim of the system is to maximize the satisfaction of each student as well as the supervisor while allocating projects to them, we need to store critical parameters that would enable quick and easy way to calculate these satisfaction index for each supervisors allocated as well as store it for future reference. Thus, we store the satisfaction index for student and supervisor and also store the preference number of the allotted project to each student.

The following table gives the outline of the tables created to store the above-mentioned details in a tabular structure:

Database Table	Description
Student	Contains all of the student's Information
Supervisor	Contains all of the Supervisor's Information
Stream	Contains al the information about the streams offered at Miskatonic University
Project	A global table containing all the projects that were proposed either by student or the student. It also serves as one of the main linking table for other tables via foreign keys
Student_proposed_project	Containing the projects that were proposed by students highlighting which student proposed which particular project and the supervisor they approached to finalize that project
Supervisor_proposed_project	Containing the projects that were proposed by supervisors
Student_preference	A project preference matrix containing 20 ranked preferences.
Student_project_map	A table containing the final one-to-one mapping of the projects with the student

Student_satisfaction	A table storing the satisfaction index for each student based on the project allocated
Supervisor_satisfaction	A table storing the satisfaction index for each supervisor based on the number of projects assigned (Workload)
Student_pref_alloc	Contains student Id and the preference of the project they were allotted

Scope:

It is to be noted that the entire system is made up of various components as stated in above sections. The Database forms a supporting frame for the storing the necessary data and hence is not complete by itself. It relies on the other system for various inputs. However, an important aspect worth mentioning over here is that while designing of the database, it was taken into consideration that even if the application fails to impose certain restriction on the type of input or the various constraints as defined in Software requirement specification, the database should not be inconsistent and every measure was taken to impose the constrains and checks at the database level wherever possible. In most cases, PLSQL is used for performing the validations on the data which might seem redundant given that the application will also be doing the same but the added layer of validation at the database level will ensure that the data entering the database abides by the constraints set and will not make it inconsistent.
Furthermore, there were certain constrains which would been difficult to impose at the database level because they could be imposed with ease at the application level without affecting the performance of the system, for example, to ensure that the student does not enter same preference 20 times. This if done at the data base level using the procedure, requires too many comparisons and greatly impacts the performance of the database system and hence it is best left to application to handle such constrains. We shall discuss more on the constrains dealt with at database level in depth in the subsequent sections.

2. Database Plan: A Schematic View

Database Structure with high level ER diagram

In this section offer we offer just high-level view of the database and its design. The ER diagram [1] drawn here is simply a conceptual roadmap of how our database will be modelled. The Details of the actual design of the Database can be found in the Section 3 of this report. Section 3 describes in detail the reason for finalizing the current schema of the database and the design choices that were made. This section serves as a building block for designing and modelling of the final schema.

Identifying the entities:

Given the present requirement, we can define the following Entities which are represented by their entity sets which can have multiple instances.

- Student
- Supervisor
- Stream
- Project
- Student Preference
- Student Proposed (Weak entity)
- Supervisor Proposed (Weak entity)

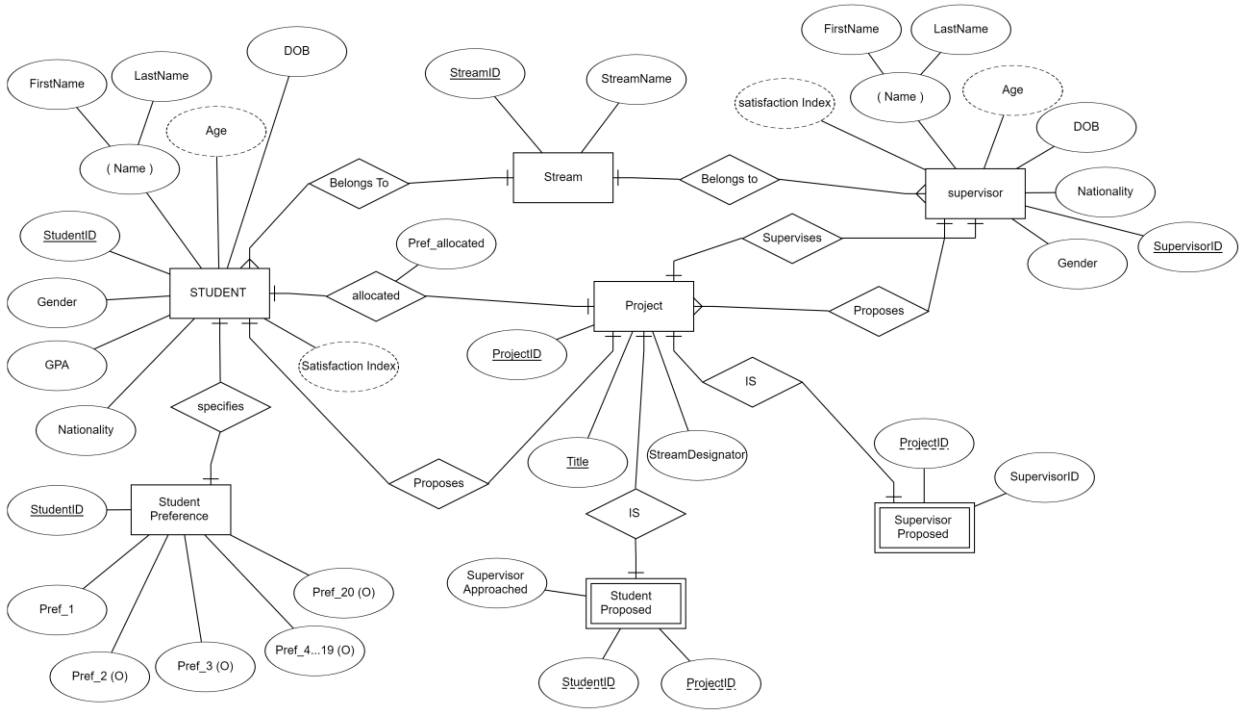

*Figure 1 ER diagram for the database**

**The attribute for student Preference- 'Pref4…19' represents all attributes from Pref_4 till Pref_19. It was not possible to include all of them here hence the format.*

The above listed entities only online a very high-level view of how our database would look like.
In the above ER diagram, the strong entities are represented by a rectangle such as student and weak entities are represented by a double rectangle as in case of Student proposed and Supervisor Proposed.

1. **Student:** Student is the main entity in the ER diagram. It has a number of attributes as outlined in the ER diagram. A few of which worth mentioning are Name which is composite attribute comprising of FirstName and LastName. While the attributes age and satisfaction index is a derived attribute. They are termed Derived attribute because Age can easily be calculated from the DOB and Student satisfaction index can be calculated from the preference allotted and the GPA

   Relationships: The Student Entity has many to one relationship with the Stream since many students may 'belong' to one stream. Student has two kinds of relationship with the Entity Project. When a student proposes a project, we model this activity using the 'proposes' relationship. This again is one to one relationship since one student can propose only one project. The same entities also share another one to one relationship 'allotted' which signifies that one student can only be allocated to one project. The final relationship associated with the entity student is 'specifies' which it shares with the entity student Preference. This again is one to one relationship since one student provides the input once although he provides 20 preferences the student Id occurs only once in the entire table.

2. **Project:** Project is the most significant entity of the ER diagram. Everything in the above ER revolves around this entity. In the ER diagram it has been modeled to represent all the projects that are proposed by either the student or the supervisors as well as the final allocation of the projects to the student and the supervisor. Although it would be represented in separate tables in the final database design. This representation gives an overall view of how the 'Project' entity interacts with the other entities and is right at the heart of the system. There are two entities that are dependent directly on this entity namely, the Student proposed and Supervisor Proposed as a result of this these two entities are termed as weak entities in the system. They serve only one purpose who is categorize all the available projects into the two categories (either student proposed, or supervisor proposed).

   Relationships: The students and projects are linked with the relationship termed as 'allocation' which represents the one to one mapping of the students and the projects. The relationship is final outcome of the system proposed and hence hold great significance. The relationship will be modelled into a table to store the details and will be discussed in the subsequent sections. The relationship itself has an attribute 'preference_alloted' associated with it. Furthermore, the Project entity also shares a one to one relationship- 'Supervises' with the entity supervisor. Which represents the details as to which supervisor would be supervising which project. As mentioned earlier, the weak entities share one to one relationship with the project as one project can only belong to only and only one of the two categories.

3. **Supervisor**: Supervisor entity represents the details of the supervisors available in the university. The attributes for Supervisor are quite similar to those of student. Furthermore, the supervisor too has an interesting derived

attribute named 'Satisfaction Index' which is dynamically calculated and stored in the database based on the workload of each supervisor.

Relationships: Supervisor entity has a many to one relationship with Stream because many supervisors belong to one stream. The supervisor and the project entity are also linked with a relationship termed as 'supervises' which represents the mapping of the project and the supervisor. Again, this is a one to one mapping because one supervisor is supervising only one project (or one student to put in context). Furthermore, a Supervisor is the main source of projects as the bulk of the projects are proposed by the supervisors. This relationship is documented by one to many relationship 'proposes' with the project. It highlights an important aspect that one supervisor can propose many projects.

4. **Stream:** This Entity represents the details of the Streams that offered by the university.

   Relationships: Every Student enrolled in the university belong to one stream and hence the many to one relationship 'belongs' between the student entity and the stream entity (many students belong to one stream). Same is the case with the Supervisor and stream. Many supervisors belong to one stream and hence, many to one relationship between supervisor and stream.

5. **Student_preference**: This entity represents the Student's Preference matrix wherein the student enters the preference for 20 projects in the order of interest. Meaning that preference 1 is most desired and preference 20 is the least desired by the student. These 20 preferences are represented by the 20 attributes of the entity. Due to space constraints, the preferences from 4 to 19 have been clubbed into one oval but in reality, they are all distinct attributes. The '(O)' next to each Preference attribute except the 1st one signifies that they are optional.

   Relationship: Each student has one entry in the preference matrix with the row as StudentID and 20 Columns representing each of the preference. The 'specifies' relationship represents the activity wherein each student provides one set of preferences hence one to one relationship.

3. Database Structure and Design: A Normalized View

The previous section contained the ER Diagram which gave us the general overview of the database.
This section describes the following:

- Database Design process
- Description about why certain design choices were made
- Extended Entity Relationship diagram (EER)
- Table Description and Normalization

3.1 Database Design Process and EER Diagram

The design process for the table began with the understanding of the data storage requirements and various constraints that were defined in the Software Requirement Specification. As the Database system relied on the front-end application as well the genetic algorithm for Input/output and project allocation to students respectively, A clear demarcation was needed to understand the segregate the functionality of the three systems. In other words, what aspects of the requirement will each of the three systems handle needed to be clearly defined. The database then needed to be the design keeping in mind how is it going to support the storage of data.

In our case, the system flow would follow the following pattern:

The purpose of assigning of the projects for the final year students at the University began with the process of gathering all the projects that we are going to assign to various students for this particular year. The primary source of proposing the projects and deciding what project do the students get to work on this year is the supervisors. They are the ones who propose and approve the projects for a current Year. Such projects are termed as supervisor proposed project. However, a small number of projects are also proposed by the students who come up some interesting ideas and then discuss it with the supervisors. After the supervisor approves this project, the student can go ahead and nominate that project for himself such projects which are nominated by the students are known as student proposed project. Now our database needs to store a central repository of all the projects that were nominated this year irrespective of who nominated or proposed it and thus we create a table named 'Project' in our database. Now we also need to store the information as to who proposed the project, whether the supervisor project it or the student proposed it and if a supervisor proposed it then which supervisor proposed it or which student proposed it. All such minute details needed to be stored in the database keeping in mind the normalization of the tables involved (which we cover in the subsequent sections) It was in the best interest to create two separate tables to store student proposed and supervisor proposed projects as by doing which a thing the dependencies in the project table on student and supervisor ID was eliminated.
Once we have the list of projects ready with us now, we need to the students to view the list the list of available projects and specify the up to 20 preferences of project. The application must fetch the data from the Project table and display the relevant projects to each student. One of the main constrains involved over here is that the Project proposed by one student should not be visible to other students. It should only and only be visible to the students who proposed it. Another, constraint which the application must take care while displaying the project is that each student belongs to one of the two stream that is either CS or CS+DS and We have a Stream assigned to each Project as well. So the Projects belong to three category namely, CS only, CS+DS only, CS/CS+DS (general). Each student should be able to view only his/her category and the projects in the general category. To facilitate this, the database stores the information about the Student Stream and the Project Stream.

Furthermore, now once the appropriate projects are displayed to the students, the students must enter their preferences. Every student is required to enter at least one preference and a maximum of up to 20. We need to store these preferences in the table which is done by the table 'Student_preference'. It has 20 columns, one for each preference. The 1st column is not null, and the student must provide the input over here. An important consideration that must be taken care of at the application level is that a student should not be allowed to enter one preference multiple times. A Student can misuse these 20 preferences by entering a same project 20 times. This can be handled by providing a simple drop-down list at the time of selection. The data should be read from the project table and show all the eligible projects (stream constraint) for the 1st preference. Now once the student enters the 1st choice and moves to the next the dropdown should contain all the eligible projects except the one entered as the first choice. This way we can avoid the student from gaming the system. Handling such scenarios involved at the database level meant that we need to compare 1st preference with 19 other columns to check if they are similar then 2nd column needed to be compared with all other columns except the first column. This would have been very difficult to do at the database level with the limited functionality we have hence not considered in the design of the database. After we have the preferences of each student and the project list available with us. Now it is up to the genetic algorithm to take these as the input and assign each student a unique project. We database design also helps provide the performance history of each student in terms of their GPA so that the algorithm gives the students with high GPA a project that among top few preferences. The end result of the algorithm is the one to one mapping of project with each student and project with each supervisor. We need to store this information in the database. This task is facilitated by the table 'Student Project Map' which stores the project ID along with the supervisor ID and student ID mapped in one to one relationship. We also store the preference number of the project assigned to each student because it is this preference number that will help us determine how happy a student is after the allocation process. Note that, the algorithm can also directly provide the Preference number but since it was something within the scope and limitation of Database, the preference number of the allocated project is initialized with null at the time of table creation and then dynamically calculated using procedure in Database itself. We will discuss the same in section 5 of this report.

The final part of the system requires us to calculate and store the satisfaction score of the student and the supervisors. That was the primary criterion for allocating the projects between the students. Our aim was to ensure that the students get the projects that they had wished for and the supervisors are not overloaded with the work that they are assigned thus we need a way to capture their satisfaction in the database. These satisfaction scores are stored in two separate tables in our database one for student and one for the supervisors. In order to satisfy all the normalization constraints they were kept as a separate entity and moreover , in future if we need to add some extra parameters to capture the satisfaction of the students and supervisors then the current design provides the necessary support to enable the changes wherever needed .

The diagram below shows the EER Diagram of the designed database:

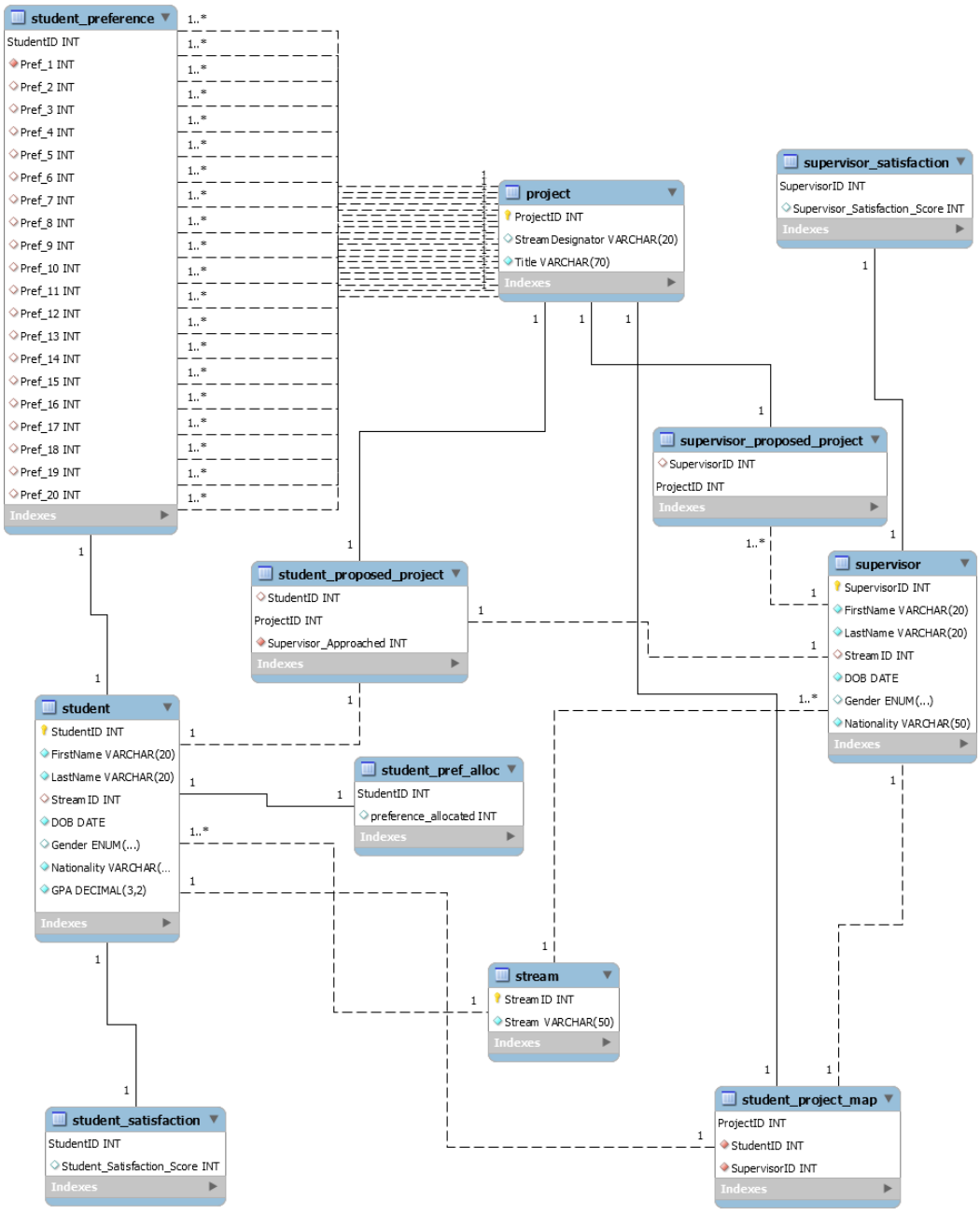

Figure 2 EER Diagram

The entire database consists of 10 Tables all of which have been normalized to Boyce-Codd Normal form or BCNF. The type of relationship between the two tables is shown in by the dotted lines and the numbers next to them. For example 1…1 Shows that the tables have one to one relationship while 1…1.* shows that that the tables share a one to many relationship.

One of the most important factors in database design is definition of the tables, throughout the design process of the above tables, I have ensured that maximum constraint are enforced at the schema level itself so that the data remains

consistent. With the schema defined above the aim was to design a database that minimizes redundancy without losing any data. That is, we aim to use the least amount of storage space for our database while still maintaining all links between data. Moreover, Normalization plays a critical role over here because normalization avoids numerous anomalies that may creep into the database while inserting, updating, or deleting data and, therefore, normalization is an essential step in the design process that helps to keep consistent data in the database. This is what prompted the current design wherein each of the table is in BCNF. Before we move on the Normalization of the tables, it is important to state and understand the concept of functional dependence.[4] [5] [6]

Functional Dependence: In a table, A column is functionally dependent on another column if a value 'A' determines the value for 'B' at any one time.

1NF or First Normal form: 1NF states that states that each attribute or column value must be atomic in nature. In other words, it stats that, each attribute must contain only a single value, not a set of values or multiple values.

2NF or Second Normal Form: A table is said to be in 2NF is the table is in 1NF and all the attributes that are not the part of the key are fully functionally dependent on the primary key. Let's take the example of Student table, It is in 2NF since all the columns be it firstname, lastname, DOB, nationality, gender are all fully functionally dependent on the StudentD which is the primary key in this case.

3NF or Third Normal Form: A table is said to be in 3NF if it is in 2NF and there is no transitive dependencies between the columns. For example, in the above EER diagram, satisfaction_score cannot be included in the Student _Project _Map table because satisfaction score depends on the studentID and studentID depends on the projectID which is the primary key over here hence there is a transitive dependency and this must be avoided and to avoid that a new table was created.

BCNF or Boyce Codd normal form: A table is said to be in BCNF if the table is in 3NF and all the functional dependencies in the table are only and only on the super key. In simpler terms we say a table is in BCNF if every functional dependency such that $X \rightarrow Y$, and here X is the super key of the table.

In the following section we will describe each of the available table and explain ow it follows satisfies the various normalization criteria.

3.2 Tables and Normalization

1. Student Table

Field	Type	Null	Key	Default	Extra
StudentID	int	NO	PRI	NULL	
FirstName	varchar(20)	NO		NULL	
LastName	varchar(20)	NO		NULL	
StreamID	int	YES	MUL	NULL	
DOB	date	NO		NULL	
Gender	enum('Male','Female')	YES		NULL	
Nationality	varchar(50)	NO		NULL	
GPA	decimal(3,2)	NO		NULL	

Figure 3 Structure: Student table

This table contains all the information associated with a student. The columns in the table are self-explanatory. While designing of the database, I have ensured that every table is complete it itself and that the tables serve only one purpose. Here the student table contains all the information associated only and only with the student and there is not other information available over here. The StudentID serves as the primary key for this table since it is unique. StreamId uniquely identifies the Stream of the Student and is a foreign key over here connecting the Stream table. I have restricted the input type for gender to just two categories to avoid inconsistencies in the entries for example some might enter 'M' for Male. The nationality column is given sufficiently large space to accommodate the country name. For the GPA I have used the decimal format (3,2) has 2 digits for the fractional part and 3-2 = 1 digits for integer part. The GPA should be between 0 and 4.2 this can be restricted using the check constraint but since the check constraint does not work in older versions of MYSQL and more over I needed ti write a procedure to check if the age of the student is above 18, I decided to use procedure over there.

The Student table does not contain any repetitive value for one studentID neither does it contain any redundant data. Hence, it can be said that the Student table is in 1NF form. The table is also in 2NF as all non-prime attributes are dependent on the StudentID which is the primary key. Furthermore, none of the columns have transitive dependency as there is no such column that derives other column Therefore, student table is in 3NF. Finally, since all the attributes, firstname, lastname, streamID, DOB, gender, nationality, GPA is dependent on studentID which is primary key and there is no multivalued dependency hence we can say that the table is in BCNF.

2. Supervisor Table

Field	Type	Null	Key	Default	Extra
SupervisorID	int	NO	PRI	NULL	
FirstName	varchar(20)	NO		NULL	
LastName	varchar(20)	NO		NULL	
StreamID	int	YES	MUL	NULL	
DOB	date	NO		NULL	
Gender	enum('Male','Female')	YES		NULL	
Nationality	varchar(50)	NO		NULL	

Figure 4 Structure: Supervisor Table

The structure of the supervisor table is exactly the same as that of the student table. It is also considered to a main table that contains supervisor information for various other purposes and not just the project allocation as a result the design of the data is generic in nature and the table is complete in itself. The Supervisor ID is the primary key for the take and the StreamId is the foreign key that connects this table to the streams table. The other attributes are exactly the same as in case of students. Even In this table the Gender input is again restricted to 2 specifiv values only.

The Supervisor table does not contain any repetitive value for one SupervisorID neither does it contain any redundant data. Hence, it can be said that the Supervisor table is in 1NF form. The table is also in 2NF as all non-prime attributes are dependent on the SupervisorID which is the primary key. Furthermore, none of the columns have transitive dependency as there is no such column that derives other column Therefore, Supervisor table is in 3NF. Finally, since all the attributes, firstname, lastname, streamID, DOB, gender, nationality are dependent on SupervisoID which is primary key and there is no multivalued dependency hence we can say that the table is in BCNF.

3. Stream table

Field	Type	Null	Key	Default	Extra
StreamID	int	NO	PRI	NULL	
Stream	varchar(50)	NO		NULL	

Figure 5 Structure: Stream Table

The table contains just the two columns namely, the StreamID and the Stream which essentially is the name of the stream. StreamId is the primary key. Currently it contains only two streams but in future, it may have more than the two given field. This table shares the relationship with student and supervisor and is used to identify the stream name based on the stream ID.

As there are only two columns in the table and one is primary key. The table satisfies all the conditions of 1NF, 2NF, 3NF and BCNF as stream is the only non prime attribute present and it is directly dependent on the StreamID.

4. Project table

Field	Type	Null	Key	Default	Extra
ProjectID	int	NO	PRI	NULL	
StreamDesignator	varchar(20)	YES		NULL	
Title	varchar(70)	NO	UNI	NULL	

Figure 6 Structure: Project Table

This table is the repository for all the available projects in the database. The ProjectId is the primary key over here. The streamDesignator field specifies the stream of the project. It is to be noted that the project stream is different from the student stream and supervisor stream and hence it was not included in the stream table. The student and supervisor stream can be CS or CS+DS while the project stream can take three values CS , CS+DS, CS/CS+DS (general) thus it was included here in the projects table itself. The Title of the project is kept Unique because we do not want two projects to have same titiles.

The Projects table does not contain any repetitive value for one ProjectID neither does it contain any redundant data. Hence, it can be said that the Project table is in 1NF form. The table is also in 2NF as all non-prime attributes, namely StreamDesignator and the title are dependent on the ProjectID which is the primary key. Furthermore, none of the columns have transitive dependency as there is no such column that derives other column Therefore, Project table is in 3NF. Finally, since all the attributes i.e. StreamDesignator, Title are functionally dependent only on the ProjectID and Project ID is the primary key and there is no multivalued dependency hence we can say that the table is in BCNF.

5. Student Preferences table

Field	Type	Null	Key	Default	Extra
StudentID	int	NO	PRI	NULL	
Pref_1	int	NO	MUL	NULL	
Pref_2	int	YES	MUL	NULL	
Pref_3	int	YES	MUL	NULL	
Pref_4	int	YES	MUL	NULL	
Pref_5	int	YES	MUL	NULL	
Pref_6	int	YES	MUL	NULL	
Pref_7	int	YES	MUL	NULL	
Pref_8	int	YES	MUL	NULL	
Pref_9	int	YES	MUL	NULL	
Pref_10	int	YES	MUL	NULL	
Pref_11	int	YES	MUL	NULL	
Pref_12	int	YES	MUL	NULL	
Pref_13	int	YES	MUL	NULL	
Pref_14	int	YES	MUL	NULL	
Pref_15	int	YES	MUL	NULL	
Pref_16	int	YES	MUL	NULL	
Pref_17	int	YES	MUL	NULL	
Pref_18	int	YES	MUL	NULL	
Pref_19	int	YES	MUL	NULL	
Pref_20	int	YES	MUL	NULL	

Figure 7 Structure: Student Preference

This table is the student preference matrix. It used to accept the 20 preferences from each of the student. The StudentId here acts as the primary key. One important design consideration made over here is that the table contains 19 foreign keys. Each of the remaining 19 preferences acts as the foreign key and reference the project table so that we can go back to find the details regarding each and every preference. Another, reason for having such 19 foreign keys in one single table is that it eliminates the chances of anyone entering an erroneous entry into the preference table. Without the foreign key we would have to verify that each of the preference entered by the student actually exists in the project table. Writing a procedure to do this comparison is tedious task and will not efficient therefore, having 19 foreign keys solves the problem at the design level/ schema level itself and there is no need to write any procedure to verify anything in this table.

The Student_Preference table does not contain any repetitive value for one studentID. Hence, it can be said that the Student Preferencce table is in 1NF form. The table is also in 2NF as all non-prime attributes are the student's preferences which are dependent on the StudentID which is the primary key. Furthermore, none of the columns have transitive dependency as there is no such column that derives other column. Each preference is independent of each other and Therefore, Student_preference table is in 3NF. Finally, since all the 20 preferences are functionally dependent only on the StudentID and StudentID is the primary key and there is no multivalued dependency hence we can say that the table is in BCNF.

6. Student Project Map

	Field	Type	Null	Key	Default	Extra
▶	ProjectID	int	NO	PRI	NULL	
	StudentID	int	NO	UNI	NULL	
	SupervisorID	int	NO	MUL	NULL	

Figure 8 Structure: Student Project Map

This table represents the final allocation of the Projects to the Students and the supervisor. It has a one-to-one relationship with the Student table and Supervisor table using the foreign keys StudentID and SupervisorID respectively. The primary key here is the ProjectID since every project must be unique. Moreover, I have also kept the StudentID as unique because we want every student exactly once in the table, and this ensures that. The supervisorID need not be unique because it a supervisor can be assigned to two different projects and hence for two different ProjectID there can be same supervisorID. Furthermore, the ProjectID over here is also a foreign key that links it to the Project table. Since the relationship between the Project table and Student Project Map table is one to one, we can have a primary key which is also a foreign key as there will not be any duplicates.
The Student Project Map table does not contain any repetitive value for one ProjectID. Hence, it can be said that the Student Project table is in 1NF form. The table is also in 2NF as all non-prime attributes are dependent on the ProjectID which is the primary key. Furthermore, none of the columns have transitive dependency as there is no such column that derives other column. Each column requires the ProjectID in order to determine its value and Therefore, the table is

in 3NF. Finally, to prove that the table is in BCNF consider the following, ProjectId determines the studentID, for every projectID there is a different StudentID in the table. The ProjectID also determines the SupervisorID since for every ProjectId there is a unique supervisor. Hence it is also functionally dependent on ProjectID. Now, since all the columns are functionally dependent only on the ProjectID and ProjectID is the primary key and there is no multivalued dependency hence we can say that the table is in BCNF.

7. Supervisor Proposed

Field	Type	Null	Key	Default	Extra
SupervisorID	int	YES	MUL	NULL	
ProjectID	int	NO	PRI	NULL	

Figure 9 Structure: Supervisor Proposed Table

'Project' table in our database contains all the available projects for current year. Now we also need to store the information as to who proposed the project, whether the supervisor project it or the student proposed it and if a supervisor proposed it then which supervisor proposed it or which student proposed it. This table stores the Information about the supervisor who proposed a Project. I did not include the information about the Supervisor in the project itself a project can either be student Proposed or Supervisor proposed and can never be both at the same time. Which meant there would be null values in the table against the ProjectID where the project was proposed by a student and vice versa therefore, it would be better to keep the project table as the central repository of the projects alone without including any further details in it.
The structure of the table is simple, the ProjectID is the primary key since it determines which project a supervisor gets to supervise this year. The supervisorID is the foreign key and links this table to the Supervisor table in the database. Since the Project table and supervisor proposed table share a one to one relationship. We have the ProjectID also acting as a foreign key to link it with the Project table. Since the relationship is one to one this is allowed in the Database.

As there are only two columns in the table and one is primary key. The table satisfies all the conditions of 1NF, 2NF, 3NF and BCNF as SupervisorID is the only non-prime attribute present and it is directly dependent on the ProjectID.

8. Student Proposed

Field	Type	Null	Key	Default	Extra
StudentID	int	YES	UNI	NULL	
ProjectID	int	NO	PRI	NULL	
Supervisor_Approached	int	NO	MUL	NULL	

Figure 10 Structure: Student Proposed Table

A explained in the previous table, this table simply identifies which project is Student Proposed from the available projects in the Project table. The ProjectID is the primary key over here and also the foreign key. Since the relationship between the Student Proposed table and the Project table is one to one a primary key can

be a foreign key. The studentID is declared Unique because every student can propose only a single project. The supervisor approached column stores the Supervisor ID whom the student spoke to before finalizing and proposing the project. This is important to identify which supervisor approved the Student's Project and the same supervisor must then be allocated by the algorithm to the project present over here.

As there are only three columns in the table and both dependent on the primary key, ProjectID. The table satisfies all the conditions of 1NF, 2NF, 3NF and BCNF as StudentID and the Supervisor Approached depends are the only two non-prime attributes present and they are is directly functionally dependent on the ProjectID therefore it satisfies the condition for BCNF.

9. Supervisor Satisfaction

Field	Type	Null	Key	Default	Extra
SupervisorID	int	NO	PRI	NULL	
Supervisor_Satisfaction_Score	int	YES		0	

Figure 11 Structure: Supervisor Satisfaction

The system requires us to calculate and store the satisfaction score of the student and the supervisors. That was the primary criterion for allocating the projects between the students. Our aim was to ensure that the students get the projects that they had wished for and the supervisors are not overloaded with the work that they are assigned thus we need a way to capture their satisfaction in the database. These satisfaction scores are stored in two separate tables in our database one for Supervisor (Supervisor Satisfaction table) and one for the Student (Student Satisfaction table, the following table).

The SupervisorID is the primary key and the foreign key over here. Primary key since it allows us to uniquely identify the satisfaction score and foreign key because the table shares a one to one relationship with the supervisor Table as every supervisor will have only one satisfaction score associated with him therefore this is completely allowed in the database.

As there are only two columns in the table and one is primary key. The table satisfies all the conditions of 1NF, 2NF, 3NF and BCNF as Supervisor satisfaction score is the only non-prime attribute present and it is directly dependent on the SupervisorID.

10. Student Satisfaction

Field	Type	Null	Key	Default	Extra
StudentID	int	NO	PRI	NULL	
Student_Satisfaction_Score	int	YES		NULL	

Figure 12 Structure: Student Satisfaction Score

The table stores the information about the satisfaction score of the student as explained in the description of the previous table.

The StudentID is the primary key and the foreign key over here. Primary key since it allows us to uniquely identify the satisfaction score and foreign key because the table shares a one to one relationship with the student Table as every student will have only one satisfaction score associated with him therefore this is completely allowed in the database.
As there are only two columns in the table and one is primary key. The table satisfies all the conditions of 1NF, 2NF, 3NF and BCNF as Student satisfaction score is the only non-prime attribute present and it is directly dependent on the StudentID.

11. Student Pref alloc Table

Field	Type	Null	Key	Default	Extra
StudentID	int	NO	PRI	NULL	
preference_allocated	int	YES		NULL	

Figure13 Structure: Student pref alloc table

The table stores the information about the Preference of the project allotted. The StudentID is the primary Key as well the foreign key since it has one to one relationship with student table.
As there are only two columns in the table and one is primary key. The table satisfies all the conditions of 1NF, 2NF, 3NF and BCNF as the preference_allocated is the only non-prime attribute present and it is directly dependent on the StudentID.

4. Database Views

A view can be defined as "A view is a virtual table whose contents are defined by a query. Like a table, a view consists of a set of named columns and rows of data. Unless indexed, a view does not exist as a stored set of data values in a database. The rows and columns of data come from tables referenced in the query defining the view and are produced dynamically when the view is referenced." [2]

Simply put a view is just a virtual table that is formed by the result of a complex query. In order to hide the complex details such as large number of joins between the table and other details we may choose to instead store the result in the form of a view so that we can directly access it at a later stage in our code. The other major reason as to why we need view is the for the security mechanism. Using a view we can select only a few columns and rows from multiple tables and then display them to the specific user. The user should be able to see only what it is supposed to see as some of the information stored in the table may be confidential. Furthermore, we can set permissions on the view instead of the underlying tables thereby allowing a certain data to be shown to the user instead of all of them. In

the context of this project, I have used views precisely for the above stated reasons. We shall see the purpose of the view when we define them in the subsequent sections.

1. View for Projects available after the allocation

Code:

```
CREATE VIEW `Post_Allocation_Available_Projects` AS
 SELECT * FROM Project as p WHERE NOT EXISTS
 ( SELECT * FROM Student_Project_Map as spm
       WHERE spm.ProjectID = p.ProjectID);
```

Output:

ProjectID	StreamDesignator	Title
110	CS/CS+DS	Mysteries in Azuithlir
121	CS	Whispers from Rlyeh Monitoring application
553	CS	Adventures of Vhuidrratl Game
786	CS+DS	Mafia Yimh'ibrod
34213	CS+DS	The rising Evug'dri
42121	CS/CS+DS	Traiobb'dhrin

Figure 13 Post allocation project availability

Purpose:

The above view shows us all the available projects remaining after all the projects have been assigned to the students by the algorithm. This information needs to be stored as a view because it may happen due to some unforeseen circumstances the student did not enter the preferences for the table and hence the algorithm did not assign him a project. Now, we need to know what all projects are available to rerun the algorithm on these available projects and assign projects to all the remaining students.

Also, in certain rare situation, a student with special permission from the supervisor, may request for change of his/her allotted project therefore it is essential to have this information stored as a view and hide the query that generates this view.

2. View for knowing the number of Projects assigned to each Supervisor

Code:

```
CREATE VIEW `Supervisor_Project_Count` AS
```

```
select count(spm.ProjectID) as 'num', spm.SupervisorID, s.FirstName,
s.LastName from Student_Project_Map spm inner join Supervisor s on
s.SupervisorID = spm.SupervisorID group by SupervisorID;
```

Output:

	num	SupervisorID	FirstName	LastName
▶	1	1	Mack	Dennett
	1	2	Jenny	Hawket
	1	3	Lonni	Whitehall
	1	4	Catlin	Shawl
	2	5	Abelard	Haskett
	1	6	Bobby	Pretsell
	2	7	Jorie	Bridgwater
	2	8	Grazia	Pitbladdo

Figure 14 Snippet of the View to find number of assigned projects

This view generates the number of Projects that have been assigned to each supervisor. The output here is just a snippet of the entire output. The num field highlights the number of projects that were allocated to a particular Supervisor. For example: Supervisor Abelard with SupervisorID 7 was allotted 2 projects by the system.

Purpose:

The view may seem redundant as it can be generated using a simple inner join on the tables but the primary reason for storing this as a view is because the Satisfaction of the supervisor depends on the number of projects allocated to him/her. The projects should be more or less equally distributed among the supervisors but in case a supervisor get allotted to too many projects the satisfaction will drop due to increased workload. Therefore, it makes sense to have this information stored as a view so that we can access in at a later stage while computing the satisfaction score of the supervisor.

I have done just that; I have used this view as an input in a procedure that calculates the satisfaction score for each supervisor and then updates the supervisor table with the score. I would not be feasible to run the query again for each supervisor to get the number of projects assigned. We shall see this procedure in the 'Procedural Element' section of the report.

3. **A view for the dean of Miskatonic University, Philipp Howards showing supervisor and project details**

Code:

```
CREATE VIEW `Project_Supervisor_Association` AS
select combine.ProjectID, p.Title, combine.Supervisor, sp.FirstName,
sp.LastName from
(select Supervisor_Approached as Supervisor, ProjectID from
Student_Proposed_Project Union all (select SupervisorID, ProjectID
from Supervisor_Proposed_project)) as combine
 inner join Supervisor sp on combine.Supervisor=sp.SupervisorID
 inner join Project p on combine.ProjectID=p.ProjectID ;
```

Output:

ProjectID	Title	Supervisor	FirstName	LastName
873	Killing of D'othrex	5	Abelard	Haskett
1421	Great Old Ones	6	Bobby	Pretsell
334	The foreign Land of Cxylth'therc	7	Jorie	Bridgwater
89721	Pulp cathulu	7	Jorie	Bridgwater
120	The last of Avhaiognne Game application	12	Georgette	Gravenall
282	Killer Iaz'eggdi	15	Gannie	Ginnety
123	Cthulhu Crusades website	1	Mack	Dennett
553	Adventures of Vhuidrratl Game	1	Mack	Dennett
9841	Dangerous Aiuez'agos	2	Jenny	Hawket
42121	Traiobb'dhrin	2	Jenny	Hawket
786	Mafia Yimh'ibrod	3	Lonni	Whitehall
1023	The temple of dagons	3	Lonni	Whitehall
520	Cathulu owns Batman	4	Catlin	Shawl
110	Mysteries in Azuithlir	5	Abelard	Haskett
1221	Monster Mlagh'tho and its rampage	5	Abelard	Haskett

Figure 15 A view for Project-Supervisor Association

This view is meant primarily for the Dean of the university, As a person in charge of everything that happen in the university, he may want to get the updates on certain important projects that are being offered as the reputation of the university depends on the implementation of these projects. Therefore, this view allows him to have important information available so that he can directly contact the supervisor of the project he is interested in as and when needed. The view hides the complex query that is composed of one union and two inner joins.

4. View for all Eligible project for each student

Purpose: This view is just a sample for the sake of discussion as to how the security mechanism functions using the views. While displaying the projects to the students. It must be ensured that the student sees only the projects that he/ she is eligible for. By eligible I mean that a student who belongs to CS+DS stream is eligible for all projects from the CS+DS stream as well the general stream CS/CS+DS. The projects that have the stream Designator as 'CS' should be hidden from him. Furthermore, a student who has self-proposed a project should see all

the projects belonging to his stream, the general stream (CS/CS+DS) and only the project that he has self-proposed. He should not see the projects that have been self-proposed by other students. As the project table is a central repository of all the projects, it may happen that projects proposed by one student are displayed to other student. This view provides a sample view which a student with studentID 5 will see while entering his preferences.

Th student 5 has self-Proposed a project with Project ID 120 therefore it is visible to him and all the project belonging to his stream which is CS+DS is visible to him as well as the projects from general stream are visible to him while the project with projectId 1421 which was proposed by other student (studentID :20) but belong to the same stream CS+DS is not visible to this student.

Code:

```
CREATE VIEW `sample_Student_view` AS
select * from Project
WHERE ( ProjectID not in (select ProjectID from
Student_Proposed_Project)
OR ProjectID in (select ProjectID from Student_Proposed_Project where
StudentID=5))
and( StreamDesignator= (select sr.Stream from Stream sr inner join
Student s on s.StreamID=sr.StreamID where s.StudentID=5) OR
StreamDesignator='CS/CS+DS') ;
```

Output:

ProjectID	StreamDesignator	Title
110	CS/CS+DS	Mysteries in Azuithlir
120	CS+DS	The last of Avhaiognne Game application
123	CS+DS	Cthulhu Crusades website
153	CS+DS	The life of D'aioct'itroth Website
313	CS+DS	Love of Aiubralpeg
520	CS+DS	Cathulu owns Batman
786	CS+DS	Mafia Yimh'ibrod
1023	CS+DS	The temple of dagons
1232	CS/CS+DS	Secrets of Nol'thug
34213	CS+DS	The rising Evug'dri
42121	CS/CS+DS	Traiobb'dhrin

Figure 16 Sample view for eligible project for StudentID 5

5. Procedural Elements

"A stored Procedure is a computer program that is stored within, and executes within, the database server. The source code and sometimes any compiled version

of the stored program are almost always held within the database server's system. When the program is executed, it is executed within the memory address of a database server process or thread." [3]

The Procedural elements such as procedures, functions and triggers significantly add to the functionalities and the capabilities of MYSQL. They empower us to perform various validation on the table and if used appropriately, they can lead to greater database integrity as well as security thereby improving the application's overall performance and maintainability.

In the context of this project, most of the procedures and triggers have been used to provide an extra layer of security check to ensure that all the hard constrains as well as the soft constrains are satisfied. I have ensured that even if the application logic fails to enforce certain condition checks on the data, the database with its inbuild mechanism verifies the data before it can be entered into the system.

1. Procedure and Trigger to validate the student:

We begin with a simple procedure that validates the input data from the user at the database level. We do not want the user to enter a GPA that is greater than 4.2 or a negative GPA. While this could have been easily taken care off by a simple 'check' constraint at the schema level itself, I decided to use a procedure to do it mainly due to two reasons: a) The versions earlier than 5.5 do not support check constraint (Issue can be read here) b) I needed a procedure anyway to calculate the age of the student from the date of birth and then generate an error.
It may happen that certain students want to play around with the system and enter unrealistic date of birth for example a date of birth entered as the date of the previous day's date would mean the student is just 1 day old. Moreover, since the age is a dynamic it always changes so it is not worthwhile storing it. Hence, the following procedure trigger combination validates the entries before they can be entered in the student table.

Code:

```
###  STORED PROCEDURE

DROP PROCEDURE IF EXISTS validate_Student;
DELIMITER $$
CREATE PROCEDURE validate_Student(
    IN DOB date,
        IN GPA DECIMAL
)
DETERMINISTIC
BEGIN
        IF (SELECT FLOOR(GPA-0)) <= 0.00 THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'GPA must
be Greater than 0';
        END IF;
    IF GPA > 4.20 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'GPA must be lesser
than 4.2';
        END IF;
        IF (SELECT FLOOR(DATEDIFF(NOW(), DATE(DOB))/365)) < 18 THEN
```

```
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Age must
be Greater than 18';
        END IF;
END$$
DELIMITER ;

### TRIGGER to call that Relevant Procedure

DELIMITER $$
CREATE TRIGGER validate_Student_insert
before INSERT ON Student FOR EACH ROW
BEGIN
        CALL validate_Student(NEW.DOB, NEW.GPA);
END$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER validate_Student_update
Before UPDATE ON Student FOR EACH ROW
BEGIN
        CALL validate_Student(NEW.DOB, NEW.GPA);
END$$
DELIMITER ;
```

2. Procedure- Trigger Combination to Check a Project is both Student and supervisor Proposed.

The procedure is written to check if a project has been both proposed by the student as well the supervisor. This is soft constraint which we need to verify as a project cannot be both at the same time. The procedure checkProposal executes the query to count the records that are common in the supervisor proposed and student proposed by the means of a inner join and if the count returned is not 0 then then the procedure puts the system in an error state and displays the appropriate message at the time of insertion.
The trigger is executed after each insertion or after update in the student proposed table and calls then checkProposal procedure. After insert is used because we need the entry to be present in the table in order to perform join and check.

Code:

```
# A project should either be student Proposed or Supervisor proposed
and not both

DROP PROCEDURE IF EXISTS CheckProposal;
DELIMITER $$
CREATE PROCEDURE CheckProposal()
BEGIN
IF(
select count(*) from supervisor_proposed_project supproj inner join
student_proposed_project stuproj on supproj.ProjectID =
Stuproj.ProjectID
)!=0 THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'The Project has already
been Proposed by the supervisor';
```

```
END IF;
END$$
DELIMITER ;

# Relevent Trigger to call the procedure

DELIMITER $$
CREATE TRIGGER validate_Proposal
after INSERT ON Student_Proposed_Project FOR EACH ROW
BEGIN
        CALL CheckProposal();
END$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER validate_Proposal_update
after UPDATE ON Student_Proposed_Project FOR EACH ROW
BEGIN
        CALL CheckProsal();
END$$
DELIMITER ;
```

3. Validation on Student Project Map table to see if the hard constraint; Every student must be allocated a project from his/her own Stream is satisfied

The following Procedure-Trigger combination is used to ensure that the algorithm does not make any mistake while allocating Projects to the students in terms of their Streams. Although the chances such a thing happening in are extremely rare, this procedure ensures there is two level of scrutiny done: one at the application level itself and two at the database level. The procedureTestStream will not allow any entry in the Student Project Map table such that the student belongs to one particular stream, say CS and he/she has been allocated projects from CS+DS stream. However, the projects from the common stream CS/CS+DS are allowed by the procedure. This is accomplished by again counting the number of students where the project allocated in the Student Project Map does not belong to their own stream or the general stream using the inner join of four tables project, student project map, student and stream. If the count is not 0 then then an appropriate error message is generated. The trigger calls the prcedureTestStream after every insert on the Student Project map table.

Code:

```
DROP PROCEDURE IF EXISTS procedureTestStream;
DELIMITER $$
CREATE PROCEDURE procedureTestStream()
BEGIN
IF(select count(s.StudentID) from Student_Project_Map s inner join
Project p on s.ProjectID=p.ProjectID
inner join Student st on s.StudentID=st.StudentID
inner join Stream sr on st.StreamID =sr.StreamID where ((sr.Stream !=
p.StreamDesignator) AND (p.streamDesignator!='CS/CS+DS')))!=0 THEN
```

```
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'The Students have been
Allocated to wrong Project Stream by the Algorithm';
END IF;
END$$
DELIMITER ;

# Trigger for relevant Procedure

DELIMITER $$
CREATE TRIGGER validate_allocation
after INSERT ON Student_Project_Map FOR EACH ROW
BEGIN
        CALL procedureTestStream();
END$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER validate_allocation_update
after UPDATE ON Student_Project_Map FOR EACH ROW
BEGIN
        CALL procedureTestStream();
END$$
DELIMITER ;
```

4. Procedure to fetch the preference number of the allocated project and Insert it into the Student Pref alloc

The preference number allocated to each student is holds great degree of significance because the satisfaction of the student after the allocation of the projects depends greatly on the which preference number' s project was finally allocated to the student.
At the time of creation of the tables this table was kept empty. After this the algorithm allocates the Projects to the student. After this process is over, I created the following procedure to fetch the preference number of the allocated project and insert the preference number and student ID in the Student Pref alloc table .
The procedure fetch_prefnum uses the cursor to run through all the studentID present in the Student_Project_map table and for each studentID it calls another procedure named updater and passes the current studentID as the parameter.

After this now it is the job of the updater procedure to find the preference number from the allocated project. It stores the projectID allocated in a temporary variable named as @pref and then uses union operation to find the column name where the preference is @pref for the given StudentID as the input.

After it has found the column name, it extracts just the number from the column name which is in this format: Pref_2. using substring it stores the portion from the 6th character in the string (the fifth character is the underscore). Now once that is done then it is stored in the @pref_num temporary variable. Now we always want that if the project was student proposed project then the preference number should always be one even if the student provides it as the 9th preference or 20th Preference. Therefore we check if the StudentID exists in the Student proposed

table, if yes then that mean he has already proposed the project and therefore the @pref_num is not updated to 1 signifying that he got the project he had proposed. If student is not found in the student proposed table then the value of @pref_num remains the same which is extracted from column name and this value is updated in the student project map by the procedure itself.

Code:

```
DROP PROCEDURE IF EXISTS fetch_prefnum;
DELIMITER $$
CREATE PROCEDURE fetch_prefnum()
BEGIN
 DECLARE done BOOLEAN DEFAULT FALSE;
  DECLARE _id BIGINT UNSIGNED;
  DECLARE cur CURSOR FOR SELECT StudentID from Student_Project_Map;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done := TRUE;

  OPEN cur;

  testLoop: LOOP
    FETCH cur INTO _id;
    IF done THEN
      LEAVE testLoop;
    END IF;
        call updater(_id);
  END LOOP testLoop;

  CLOSE cur;
END$$
DELIMITER ;


################################################
DROP PROCEDURE IF EXISTS updater;
DELIMITER $$
CREATE PROCEDURE updater(
    IN ID INTEGER
)
DETERMINISTIC
BEGIN
                          set @pref =( select ProjectID from
Student_Project_Map where StudentID=ID limit 1);

  set@pref_alloted= (SELECT col FROM (

   SELECT "Pref_1" AS col, Pref_1, StudentID AS value FROM
Student_Preference where Pref_1 =@pref
   UNION ALL SELECT "Pref_2", Pref_2, StudentID FROM
Student_Preference where Pref_2 =@pref
   UNION ALL SELECT "Pref_3", Pref_3,StudentID FROM
Student_Preference where Pref_3 =@pref
   UNION ALL SELECT "Pref_4",Pref_4, StudentID FROM
Student_Preference where Pref_4 =@pref
   UNION ALL SELECT "Pref_5", Pref_5,StudentID FROM
Student_Preference where Pref_5 =@pref
   UNION ALL SELECT "Pref_6",Pref_6, StudentID FROM
Student_Preference where Pref_6 =@pref
```

```
   UNION ALL SELECT "Pref_7",Pref_7, StudentID FROM
Student_Preference where Pref_7 =@pref
   UNION ALL SELECT "Pref_8", Pref_8,StudentID FROM
Student_Preference where Pref_8 =@pref
   UNION ALL SELECT "Pref_9",Pref_9, StudentID FROM
Student_Preference where Pref_9 =@pref
   UNION ALL SELECT "Pref_10", Pref_10,StudentID FROM
Student_Preference where Pref_10 =@pref
   UNION ALL SELECT "Pref_11",Pref_11, StudentID FROM
Student_Preference where Pref_11 =@pref
   UNION ALL SELECT "Pref_12", Pref_12,StudentID FROM
Student_Preference where Pref_12 =@pref
   UNION ALL SELECT "Pref_13",Pref_13, StudentID FROM
Student_Preference where Pref_13 =@pref
   UNION ALL SELECT "Pref_14",Pref_14, StudentID FROM
Student_Preference where Pref_14 =@pref
   UNION ALL SELECT "Pref_15",Pref_15, StudentID FROM
Student_Preference where Pref_15 =@pref
   UNION ALL SELECT "Pref_16", Pref_16,StudentID FROM
Student_Preference where Pref_16 =@pref
   UNION ALL SELECT "Pref_17",Pref_17, StudentID FROM
Student_Preference where Pref_17 =@pref
   UNION ALL SELECT "Pref_18",Pref_18, StudentID FROM
Student_Preference where Pref_18 =@pref
   UNION ALL SELECT "Pref_19",Pref_19, StudentID FROM
Student_Preference where Pref_19=@pref
   UNION ALL SELECT "Pref_20",Pref_20, StudentID FROM
Student_Preference where Pref_20 =@pref
) allValues WHERE value = ID limit 1) ;


set @pref_num =(select substring(@pref_alloted, 6)limit 1);
IF (SELECT EXISTS(SELECT StudentID from Student_Proposed_Project
WHERE StudentID=ID limit 1))=1 THEN
                set @pref_num =1;
        END IF;

INSERT into Student_Pref_alloc values
(ID, @pref_num);
END$$
DELIMITER ; END$$
DELIMITER ;
```

Output:

StudentID	preference_allocated
1	1
2	2
3	1
4	3
5	1
6	1
7	5
8	6
9	4
10	1

Figure 17 Snippet of the output of Student Pref alloc table after inserting

5. Procedure to Update the Student Satisfaction table

The procedure is used to compute the student satisfaction score using a simple formula and then populate the Student satisfaction table

Code:

```
DROP PROCEDURE IF EXISTS insertsat;
DELIMITER $$
CREATE PROCEDURE insertsat()
BEGIN
 DECLARE done BOOLEAN DEFAULT FALSE;
  DECLARE _id BIGINT UNSIGNED;
  DECLARE cur CURSOR FOR SELECT StudentID from Student_Project_Map;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done := TRUE;

  OPEN cur;

  testLoop: LOOP
    FETCH cur INTO _id;
    IF done THEN
      LEAVE testLoop;
    END IF;
        call satisfaction_insert(_id);
  END LOOP testLoop;

  CLOSE cur;
END$$
DELIMITER ;

DROP PROCEDURE IF EXISTS satisfaction_insert;
DELIMITER $$
CREATE PROCEDURE satisfaction_insert(
    IN ID INTEGER
)
DETERMINISTIC
BEGIN
        set @pref =( select preference_allocated from
Student_Pref_alloc where StudentID=ID limit 1);
    set @sat_score = 105-(@pref * 5);

Insert into Student_satisfaction values
(ID, @sat_score);
END$$
DELIMITER ;
```

In the code above the concept of cursor is used again. The procedure insersat initializes a cursor that selects each studentID from the Student Project Map makes a call to the satisfaction_insert procedure with the parameter as the studentID this process is done for each studentId that exists in the Student Project Map. Now the satisfaction_insert procedure takes in the parameter and fetches the preference_allocated value from the student Pref alloc table.

Then the satisfaction score is calculated using a simple formula:
105- (Pref_number *5)

This gives us the score of 100 for everyone who got their preference 1 and a score of 95 to anyone who got their second preference. The same thing is inserted against the respective StudentID (input parameter) in the satisfaction table.

6. Procedure to update the Supervisor Satisfaction Table

The procedure is used to compute the supervisor satisfaction score using a simple formula and then populate the Supervisor satisfaction table

Code:

```
DROP PROCEDURE IF EXISTS satisfaction_supervisor;
DELIMITER $$
CREATE PROCEDURE satisfaction_supervisor()
BEGIN
 DECLARE done BOOLEAN DEFAULT FALSE;
  DECLARE _id BIGINT UNSIGNED;
  DECLARE cur CURSOR FOR SELECT SupervisorID from Supervisor;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done := TRUE;

  OPEN cur;

  testLoop: LOOP
    FETCH cur INTO _id;
    IF done THEN
      LEAVE testLoop;
    END IF;
        call supervisor_Satisfaction(_id);
  END LOOP testLoop;

  CLOSE cur;
END$$
DELIMITER ;

DROP PROCEDURE IF EXISTS supervisor_Satisfaction;
DELIMITER $$
CREATE PROCEDURE supervisor_Satisfaction(
    IN SupID INTEGER
)
DETERMINISTIC
BEGIN

    set @count = (select num from Supervisor_Project_count where
SupervisorID=supID limit 1 );
     set @sup_sat = (1/@count) * 100;
     IF (SELECT EXISTS(SELECT SupervisorID from
Supervisor_Project_count WHERE SupervisorID=supID limit 1))=0 THEN
                set @sup_sat =0;
       End If;
Insert into Supervisor_satisfaction values
(supID, @sup_sat);
END$$
DELIMITER ;
```

In the code above the concept of cursor is used again. The procedure satisfaction_supervisor initializes a cursor that selects each supervisorID from the Supervisor table makes a call to the Supervisor_satisfaction procedure with the

parameter as the SupervisorID this process is done for each SupervisorId that exists in the Supervisor table. Now the supervisor_satisfaction procedure takes in the parameter and fetches the num field from the view that we had defined earlier. This num is nothing but the number of projects that a supervisor has been assigned to each supervisor. We had created a view of this in the earlier section and now we will be using it to calculate the satisfaction score of the supervisor.
The supervisor_satisfaction function uses a simple mathematical formula:
@sup_sat = (1/number of projects allocated)* 100

This gives the inverse relationship between the projects allocated and the satisfaction score. It may happen that some supervisors are not present in the view as they may not be allotted any projects by the system foe such cases a simple query is used to check if the supervisor does not exist then the @sup_sat score is set to 0 in such cases.

7. Sample Queries: To test the database

1. Finding the top 5 most popular projects where popularity is defined as the number of times a project appears in preference columns 1,2,3.

```
select a.Title, a.ProjectID, b.Popularity
from Project as a
inner join
(
  select project , count(*) as Popularity
from (
(select Pref_1 as project from Student_Preference ) union all
(select Pref_2 as project from Student_Preference) union all (select
Pref_3 as project from Student_Preference)
    ) p
group by project
) as b
on a.ProjectID=b.project order by b.popularity DESC limit 5;
```

Output:

	Title	ProjectID	Popularity
▶	Hulal, the tracking app	146	8
	Whispers from Rlyeh Monitoring application	121	7
	The life of D'aioct'itroth Website	153	6
	Cthulhu Crusades website	123	5
	The great Haozhorh App	182	5

Figure 18 Top 5 most popular Projects

The inner query combines all the three columns into one temporary column project grouped by project and applies the aggregate function count on it. The inner join is done to get project title associated with the projectID. This is important piece of information since we that the Hulal, the tracking app is the very high in demand as a lot of students have expressed their interest in the project. It will be interesting to see who is awarded this project by the genetic algorithm.

2. Find the number of times each project has occurred in the preference matrix

Code:

```
select distinct a.Title, a.ProjectID, b.Num_of_Occurence
from Project as a
inner join
(
  select project , count(*) as Num_of_Occurence
from (

(select Pref_1 as project from Student_Preference ) union all
(select Pref_2 as project from Student_Preference) union all
(select Pref_3 as project from Student_Preference) union all
(select Pref_4 as project from Student_Preference) union all
(select Pref_5 as project from Student_Preference) union all
(select Pref_6 as project from Student_Preference) union all
(select Pref_7 as project from Student_Preference) union all
(select Pref_8 as project from Student_Preference) union all
(select Pref_9 as project from Student_Preference) union all
(select Pref_10 as project from Student_Preference) union all
(select Pref_11 as project from Student_Preference) union all
(select Pref_12 as project from Student_Preference) union all
(select Pref_13 as project from Student_Preference) union all
(select Pref_14 as project from Student_Preference) union all
(select Pref_15 as project from Student_Preference) union all
(select Pref_16 as project from Student_Preference) union all
(select Pref_17 as project from Student_Preference) union all
(select Pref_18 as project from Student_Preference) union all
(select Pref_19 as project from Student_Preference) union all
(select Pref_20 as project from Student_Preference)
     ) p
group by project
) as b
on a.ProjectID=b.project order by b.Num_of_Occurence ;
```

Output:

Title	ProjectID	Num_of_Occurence
Killer Iaz'eggdi	282	1
Great Old Ones	1421	1
Killing of D'othrex	873	1
Pulp cathulu	89721	1
The foreign Land of Cxylth'therc	334	1
The last of Avhaiognne Game application	120	1
Invincible Izaiombr'dre	93621	2
The temple of dagons	1023	2
Daredevil Vothrhorc Skiing app	128	3
Dangerous Aiuez'agos	9841	4
Secrets of Nol'thug	1232	5
Adventures of Vhuidrratl Game	553	6
Cthulhu Crusades website	123	6

Figure 19 Snippet of Number of occurrences of each Project

The above code does the job of providing the number of times each project occurred in the Student Preference Matrix. We can see that some of the projects occurred only once in the table that means they are not the ones which are sort after by the students. Such information is useful in identifying what type of projects students want.
The code is just the extension of the query done in previous part. We combine all the columns into the project column and then group by project and count the number of occurrences of each project.

3. Query for counting the number of Preferences entered by Each student and calculating the average number of preferences provided by all students

Query:

```
SELECT
  (
  IF(Pref_1 IS NOT NULL, 1, 0)
  + IF(Pref_2 IS NOT NULL, 1, 0)
  + IF(Pref_3 IS NOT NULL, 1, 0)
  + IF(Pref_4 IS NOT NULL, 1, 0)
  + IF(Pref_5 IS NOT NULL, 1, 0)
  + IF(Pref_6 IS NOT NULL, 1, 0)
  + IF(Pref_7 IS NOT NULL, 1, 0)
  + IF(Pref_8 IS NOT NULL, 1, 0)
  + IF(Pref_9 IS NOT NULL, 1, 0)
  + IF(Pref_10 IS NOT NULL, 1, 0)
  + IF(Pref_11 IS NOT NULL, 1, 0)
  + IF(Pref_12 IS NOT NULL, 1, 0)
  + IF(Pref_13 IS NOT NULL, 1, 0)
  + IF(Pref_14 IS NOT NULL, 1, 0)
  + IF(Pref_15 IS NOT NULL, 1, 0)
  + IF(Pref_16 IS NOT NULL, 1, 0)
  + IF(Pref_17 IS NOT NULL, 1, 0)
  + IF(Pref_18 IS NOT NULL, 1, 0)
  + IF(Pref_19 IS NOT NULL, 1, 0)
  + IF(Pref_20 IS NOT NULL, 1, 0)
    )
  AS 'Number of pref entered', StudentID
FROM  Student_Preference;
```

Output:

	Number of pref entered	StudentID
▶	4	1
	8	2
	10	3
	9	4
	7	5
	7	6
	10	7
	9	8
	4	9
	3	10

Figure 20 number of preferences entered by each Student

Since only the first preference in mandatory for the student to enter, the rest of the columns can be left blank. Finding out how many preferences where entered by the user is important before depending ton this information, we can calculate the average number of preferences provided by the user in the following query. The average number of preferences will help us in deciding the maximum number of preferences that we want to keep in future. We see that the average number of preferences here is just 7 so in future maybe we can allow only 10 preferences.

Query for average:

```
SELECT
  Avg(
  IF(Pref_1 IS NOT NULL, 1, 0)
  + IF(Pref_2 IS NOT NULL, 1, 0)
  + IF(Pref_3 IS NOT NULL, 1, 0)
  + IF(Pref_4 IS NOT NULL, 1, 0)
  + IF(Pref_5 IS NOT NULL, 1, 0)
  + IF(Pref_6 IS NOT NULL, 1, 0)
  + IF(Pref_7 IS NOT NULL, 1, 0)
  + IF(Pref_8 IS NOT NULL, 1, 0)
  + IF(Pref_9 IS NOT NULL, 1, 0)
  + IF(Pref_10 IS NOT NULL, 1, 0)
  + IF(Pref_11 IS NOT NULL, 1, 0)
  + IF(Pref_12 IS NOT NULL, 1, 0)
  + IF(Pref_13 IS NOT NULL, 1, 0)
  + IF(Pref_14 IS NOT NULL, 1, 0)
  + IF(Pref_15 IS NOT NULL, 1, 0)
  + IF(Pref_16 IS NOT NULL, 1, 0)
  + IF(Pref_17 IS NOT NULL, 1, 0)
  + IF(Pref_18 IS NOT NULL, 1, 0)
  + IF(Pref_19 IS NOT NULL, 1, 0)
  + IF(Pref_20 IS NOT NULL, 1, 0)
    )
  AS 'Average number of Preferences Entered'
FROM  Student_Preference;
```

	Average number of Preferences Entered
▶	7.3000

Figure 21 Average number of Preferences entered by the students

The logic behind the query is that we calculate the avg value of the sum of number of not nulls in each row of the table. The column pref_1 is checked to find if the value is not null, if so then 1 is added if it is null then 0 is added. This way we are able to calculate the average number of preferences.

4. Find the number of self-proposed projects in each stream

The self-proposed projects are important because the students take the initiative to mee the supervisors and decide which project they want to work on. This query along with other similar queries can form a reporting mechanism that provide a dashboard to the dean of the University. We can see which stream has the most

self-proposed projects. And also, we can come to know how many projects were self-proposed overall.

Query:

```
select count(std.StudentID) as 'Number of self Proposed Projects',
sr.Stream  from Student_Proposed_Project spm inner join Student std
on spm.StudentID = std.StudentID inner join Stream sr on
std.StreamID=sr.StreamID  group by sr.Stream;
```

Output:

Number of self Proposed Projects	Stream
4	CS
2	CS+DS

Figure 22 Stream wise breakdown of self-Proposed Projects

5. Given a Student ID find all the preferences which do not belong to his/her stream

The application will ensure that the students get the projects from their own stream or the projects from the general CS/CS+DS stream. In this the application may want to find out how many of the preference belonged to the ineligible stream (not in general category and not in own stream) these preferences should be discarded directly, and they are not useful in any way to the system.

Query:

```
set @x= 13; # Student ID is taken input from the User

select * from Project  where ProjectID in (
        ( select project from ((select Pref_1 as project from
Student_Preference where StudentID=@x ) union all
      (select Pref_2 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_3 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_4 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_5 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_6 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_7 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_8 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_9 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_10 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_11 as project from Student_Preference where
StudentID=@x) union all
```

```
      (select Pref_12 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_13 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_14 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_15 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_16 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_17 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_18 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_19 as project from Student_Preference where
StudentID=@x) union all
      (select Pref_20 as project from Student_Preference where
StudentID=@x)
     ) p
     group by project )
     ) and StreaMDesignator <> ( select sr.Stream from Student s
inner join Stream sr on s.StreamID=sr.StreamID where s.StudentID= @x)
and StreamDesignator <> 'CS/CS+DS';
```

Output:

	ProjectID	StreamDesignator	Title
▶	34213	CS+DS	The rising Evug'dri
*	NULL	NULL	NULL

Figure 23 Expressed Preference for ineligible project

Here we can see that the student expressed preference for a project he/she cannot be granted. The system can either use this in the reporting mechanism or use this query to eliminate all the ineligible entries from the table.

6. Find the supervisors who proposed projects from streams other than their own Stream

This query will generate all the details of the supervisor who proposed projects other than their own Stream, which project did the propose, and stream of that project. This can be used by the application to for the analytics part to see how the projects were proposed by the Supervisors.

Query:

```
select spm.supervisorID, spm.ProjectID,p.Title, p.StreamDesignator as
'Project Stream', s.FirstName, s.LastName, sr.Stream as 'Supervisor
Stream'  from Supervisor_Proposed_Project spm
 inner join Project p on spm.ProjectID = p.ProjectID
 inner join Supervisor s on spm.SupervisorID = s.SupervisorID
 inner join Stream sr on s.StreamID=sr.StreamID where
p.StreamDesignator<>sr.Stream ;
```

Output:

	supervisorID	ProjectID	Title	Project Stream	FirstName	LastName	Supervisor Stream
▶	2	42121	Traiobb'dhrin	CS/CS+DS	Jenny	Hawket	CS
	5	110	Mysteries in Azuithlir	CS/CS+DS	Abelard	Haskett	CS
	1	553	Adventures of Vhuidrratl Game	CS	Mack	Dennett	CS+DS
	8	1232	Secrets of Nol'thug	CS/CS+DS	Grazia	Pitbladdo	CS+DS

Figure 24 Supervisors who proposed projects from streams other than their own stream

We can see here that apart from these 4 four supervisors all the supervisors proposed projects from their own stream. Out of the 4, in three cases the supervisors proposed a general category project which is nothing but a project belonging to CS/CS+DS stream.

7. Find out all the students with the maximum satisfaction score after the project allocation.

Query:

```
select s.StudentID, s.FirstName, s.LastName, s.GPA,
sf.Student_satisfaction_score from Student_satisfaction sf inner join
student s on sf.studentId=s.StudentID where
sf.Student_satisfaction_score =(select
max(Student_satisfaction_score) from Student_satisfaction) ;
```

Output:

	StudentID	FirstName	LastName	GPA	Student_satisfaction_score
▶	1	Mano	Rentoll	3.56	100
	3	Courtnay	Basili	3.31	100
	5	Ravish	Kumar	3.59	100
	6	Deepika	Yadav	3.77	100
	10	Donaugh	Irnys	2.76	100
	11	Mellisa	Challiner	3.72	100
	15	Jeannette	Dorber	2.97	100
	17	Egor	Gabbitas	2.73	100
	18	Andriette	McFater	4.16	100
	20	Lisetta	Denzey	3.64	100

Figure 25 Students with the maximum satisfaction

Given that the primary aim of the system is maximize the student satisfaction, this query allows us to see how well the genetic algorithm performed in allocating projects to the students. The more the entries in this table, the better the performance of the algorithm. Furthermore, the application can also use this query again for the reporting mechanism wherein there is dashboard to show how many people were 100% satisfied with the project that the got.

8. Find out all the details of the students who specified same project at the preference 1.

```
Select sf.StudentID, sf.Pref_1 as '1st Preference' ,p.Title,
s.FirstName, s.LastName, s.GPA from Student_Preference sf inner join
Student s on sf.StudentID = s.StudentID inner join Project p on
sf.Pref_1= p.ProjectID where Pref_1 IN (SELECT Pref_1 FROM
Student_Preference GROUP BY Pref_1 HAVING COUNT(*) > 1) Order by
Pref_1, s.GPA;
```

Output:

StudentID	1st Preference	Title	FirstName	LastName	GPA
13	146	Hulal, the tracking app	Randolph	Schulz	2.70
15	146	Hulal, the tracking app	Jeannette	Dorber	2.97
19	153	The life of D'aioct'itroth Website	Onida	Valti	2.15
2	153	The life of D'aioct'itroth Website	Elias	Burge	2.50
14	153	The life of D'aioct'itroth Website	Bobbi	Clever	3.55
16	182	The great Haozhorh App	Hollyanne	Hurdle	3.30
3	182	The great Haozhorh App	Courtnay	Basili	3.31
9	803	The Great Ybharvurc App	Melany	Frodsam	2.61
7	803	The Great Ybharvurc App	Esta	Alner	3.92

Figure 26 Student Details who had same 1st Preference

The above query is used to find out the GPA and other details of the students who specified same project as their 1st Preference. All of this information is essential considering the fact that many students will provide the same preference for the popular projects, in the end the deciding factor would be the GPA of the students.

9. Find out the number of unassigned Supervisors after the allocation

Query:

```
select SupervisorID, FirstName, LastName from Supervisor where
SupervisorID not in (select SupervisorID from Student_Project_Map);
```

Output:

SupervisorID	FirstName	LastName
9	Freemon	Guilaem
11	Chrisse	Mateos

Figure 27 Number of Unallocated Supervisors

It may happen that the algorithm is run in multiple batches so after the first round of allocation we may want to find out how many of the supervisors were not assigned any project by the system. Moreover, it is also possible that these Supervisors were busy and did not want to supervise any students this year such a query can aid the application in displaying the details of such supervisors.

8. Conclusions

The database is currently fully functional and provides MySQL back-end to the application. It currents meets all the demands that were set aside in the software requirement specification and does the job of providing a supporting framework to the application for storing and retrieving all the data efficiently. However, there is always scope for improving existing system. The current system does not provide any mechanism for altering or changing the allocated projects. If the algorithm assigns a project that the student is unhappy with these is no proper mechanism that would allow a student to raise the grievance and ask for change of project based on the GPA. We may incorporate a system that allows a student to raise a grievance as to why he /she feels that the assigned project does not appeal to him/her and then send the grieve to the supervisor who can either approve it or reject it . If the appeal is approved, then the system will allow the student to pick a different project from the list of unallocated projects after all the allocation is done.

Another area with we can add to the existing system is allow the students to work in groups. Currently all the students are allocated projects on an individual basis this is fine if the number of students in the university is small, but if the number of students are large then it will be difficult to go ahead with the current one to one mapping of the projects as it will be difficult to find unique projects for large number of students. Furthermore, adding the option of working in groups will help in achieving better satisfaction scores for the students as currently if say 4 people specify a project XYZ as their first preference then only one amongst them will be getting it. If we can allow the students to work in group, then we can form a group of students who specified the same project and then allocate them that project.

Overall, I believe the project was a great learning experience for me and allowed mee to improve upon my SQL skills. Furthermore, I enjoyed designing the database and then constructing it all b myself and to be able to see the end results of the queries was satisfying. One area in particular that tis project can greatly helped me in is the PL/SQL. I had never gotten at opportunity to learn and

implement Procedural elements in the database. This project gave me the opportunity to do so and learn the fundamentals of how PL/SQL can bind the programming aspects to SQL. Finally, the project also allowed me to improve upon my writing skills and I believe being able to document your project is an important skills going forward in my career and this project helped me in improving that skills as well.

Acknowledgements

I would like to express my deep and sincere gratitude to our Professor Tony Veale, for giving me this opportunity to work on an interesting project that greatly helped me improve upon my knowledge of the Databases. I would also like to thank him for his invaluable guidance throughout this module and during the project.

I would also like to express my gratitude to the teaching assistants and the demonstrators who have helped me by solving all my doubts throughout the module.

Furthermore, my thanks go out to all the authors whose work I have referred to either directly or indirectly during the making of this report.

Lastly, I would like to take this opportunity to certify that this Project/report is my own work, based on my personal study and/or research.

References

1. Malay K. Pakhira. *Database Management System* PHI Learning 2012. Print.
2. Microsoft SQL Documentation (link)
3. MySQL Stored Procedure Programming: Building High-Performance Web Applications in MySQL By Guy Harrison, Steven Feuerstein
4. Principles of Database Systems by J D Ullman
5. Database Systems: A Practical Approach to Design, Implementation and Management" by Connolly
6. Prabhjot, Prabhjot and Neha V. Sharma. Overview of the Database Management System." *International Journal of Advanced Research in Computer Science* 8 (2017)