

# Project:

## The Call for Cthulhu Projects

COMP40725: Introduction to Relational Databases and SQL Programming



## Table of Contents

Section 1: Introduction .....	4
1.1 Application Perspective/Vision.....	4
1.2 Application Components.....	5
1.3 Data Handling Assumptions.....	6
Section 2: Database Plan: Schematic View.....	7
2.1 Principal entities, attributes and relations.....	7
2.2 Entity relationship model .....	9
2.3 Design motivation .....	10
Section 3: Database Structure: Normalised View.....	11
3.1 Table description .....	11
a. Project allocation rules and assumptions.....	15
b. Satisfaction level calculation rules and assumptions.....	16
3.2 Database normalisation .....	18
3.3 EER Diagram .....	20
Section 4: Procedural Elements.....	21
Section 5: Database Views .....	26
Section 6: Example Queries .....	29
Section 7: Conclusion and Future Works.....	35
Acknowledgments .....	36
References .....	36

## List of Figures

Figure 1: High Level Application Architecture .....	6
Figure 2: Entity-Relationship Diagram .....	10
Figure 3.1: Table Student .....	12
Figure 3.2: Table Stream .....	12
Figure 3.3: Table Supervisor.....	13
Figure 3.4: Table Projects.....	14
Figure 3.5: Table Student Preferences.....	14
Figure 3.6: Project Allocation Rules Table.....	15
Figure 3.7: Table Student Project Mapping.....	16
Figure 3.8: Local Satisfaction Score Calculation Table.....	17
Figure 3.9: Table Student Satisfaction.....	18
Figure 3.10: Entity Extended Relationship Diagram.....	20
Figure 4.1: Procedure & Trigger 1 Results.....	22
Figure 4.2: Procedure & Trigger 2 Results.....	23
Figure 4.3: Procedure & Trigger 3 Results.....	24
Figure 4.4: Procedure & Trigger 4 Results.....	25
Figure 4.5: Procedure & Trigger 5 Results.....	26
Figure 5.1: View 1.....	27
Figure 5.2: View 2.....	27
Figure 5.3: View 3.....	28
Figure 5.4: View 4.....	28
Figure 5.5: View 5.....	29
Figure 6.1: Query Output1.....	30
Figure 6.2: Query Output2.....	30
Figure 6.3: Query Output3.....	31
Figure 6.4: Query Output4.....	31
Figure 6.5: Query Output5.....	32
Figure 6.6: Query Output6.....	33
Figure 6.7: Query Output7.....	33
Figure 6.8: Query Output8.....	35

## 1. Introduction

This report contains the implementation of a database system of an application: Call for Cthulhu Projects. This application is responsible for the systematic allocation of the projects to the final year students of a school: Cthulhu Studies, at Miskatonic University. This section gives detailed description on the vision of the Application and its key features:

### **Application Perspective/Vision:**

Call for Cthulhu Projects is an integrated solution for management of projects distribution among the students of the Cthulhu Studies. As per my assumption, it is a web-based application with primary purpose of optimal project allocation to the students according to their preferences with the maximum satisfaction achieved. Our role here is to develop a repository which captures accurate and consistent project and student information, which would further assist in the decision making of the choosing and allocation of the projects among the students and eventually storing the final results as well. We need to design a system that reduces data error, ensures that information is managed efficiently and is always up-to-date. Complete student history, can easily be searched, viewed and reported with the help of this application.

The ***main end users*** of the application are Students, Supervisors and the administrator.

#### **1. *Students:***

Each student belongs to a stream [CS Only or CS+DS]. Students are given a unique username and password to login. Each of them will have a different view (Application Logic). Each student can see the projects assigned to them their grades and they can also access or update their personal information. Each student will be able to view the projects available for their stream in the system and can also create their preference list from the available projects.

#### **2. *Supervisors:***

Staff members will also have a username and password to login. The different views for teachers will allow them to view their personal details as well some of the project related information. Staff members can also update their personal information but not all staff members will have the privilege to update the project related information.

### 3. *Administrators:*

The administrator here is a logical entity which will have access to all the student and supervisor related information in the different tables in the database.

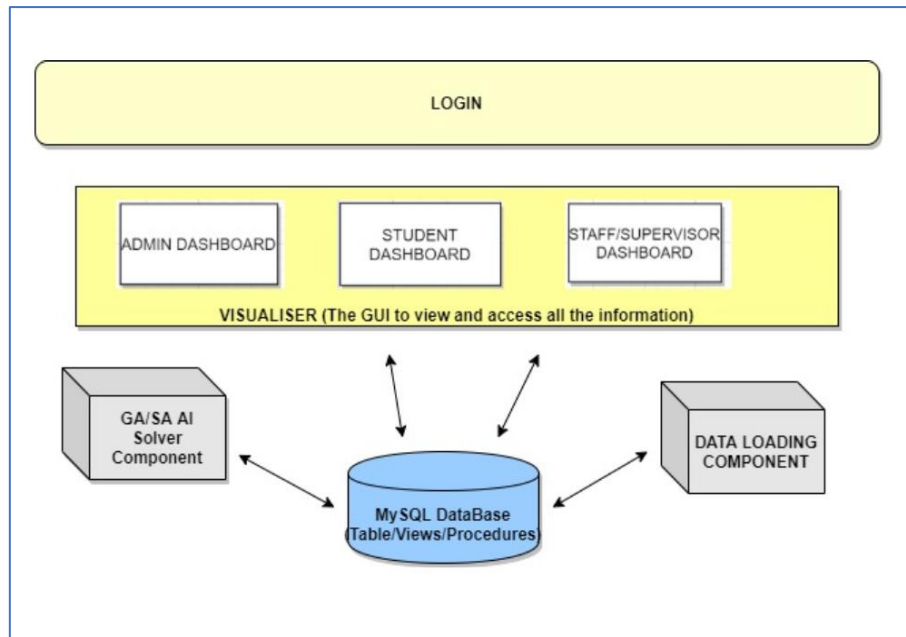
Administrator is responsible for managing the University related information, for instance data related to different Streams. Admins will also have the privileges to update this information. They will be provided with search and filter features so that they can access data efficiently.

[NOTE- Admin details and application login details are a logical concept and has not been handled as a part of this database system. It is assumed here that login details will also be stored as part of system]

### **Application Components:**

The Call of Cthulhu application consists of following main modules:

- ***MySQL Database Repository:*** which captures the details of Students personal and academic details, University Details like information related to Staff Members and finally the preferences expressed by the students and final mapping of the projects. Apart from the key entities Students, Supervisors and Projects Mapping the system also maintains inventory of the projects, different streams in the system, the satisfaction details of the students
- ***Front-end and a Visualiser:*** The Application will display all the above-mentioned captured information, with each user having restricted access to the information. For instance, a Student or a Supervisor and each individual can see the information related to him/her by logging in into individual accounts. The access can be restricted at both application level or by enforcing privileges at the database level.
- ***A Solver Component:*** The allocation of the projects to be done is by using AI solutions like Simulated Annealing or Genetic Algorithms, which are to be implemented as a part of application logic. (not a scope of this report)
- ***Data loading:*** Input Output mechanisms to load the dataset into the application.



*Figure 1: High Level Application Architecture*

#### **Data Handling Assumptions:**

- It is assumed in an ideal case all the initial data such as student records, staff records, university details all are fed into system via the Application, may be by a bulk sftp load, ETL Load or some other data ingestion mechanism handled at the application level as there could thousands of records to loaded at a time. For the sake of this project the records have been inserted manually into this DB and insert statements have been mentioned in the SQL file attached with this report. But there have been some constraints placed which would guide/check the format in which data is inserted into the tables. These constraints have been discussed further in this report.
- As mentioned in the white paper, another system is responsible for collecting preferences from the students, once done same could be fed into the student preference tables.
- Each student can login to the system and can view all information related to him or the other project related views accessible to the User student.
- Each staff member can login to the system and can view all information related to him or the other project related views accessible to the User staff.

## 2. Database Plan: A Schematic View

A conceptual design is an imperative step in the database design phase. We first create a structure based on the requirements of the application, before implementing them. The fundamental units of the conceptual model are Entities, Attributes and the Relationships.

Below is a high-level description for the principal entities of this model, their key attributes and the relationships among these entities. These relations could be 1 to 1 (One-to-One) or 1 to N (One-to-many) or N to N (Many-to-Many). Some of the relations are identifying relationships and others are non-identifying. When the primary key of the parent entity exists in the primary key of the child entity, then it is an identifying relationship. And when the primary key of parent exists in the child entity but not as the primary key then it is known as a Non-Identifying Relationship. In the latter, the child entity can stand on its own without the parent entity.

[NOTE: Detailed information about each table has been mentioned in the next section.]

### **PRINCIPAL ENTITIES, ATTRIBUTES AND RELATIONS:**

#### ***STUDENT***

The entity represents students. It holds the information that relate to details of a student. It has attributes Student ID, Student name, Stream id, grades, Date of Birth, gender and nationality. Student\_id is the key attribute for this entity because it uniquely identifies each student record. With the attribute stream\_id we can identify the student belongs to which stream. This field takes up two values 'CS01' and 'DS01' and the corresponding stream names for these id's can be retrieved from the Stream table.

#### ***STREAM***

The entity stream holds the information of the different course streams offered in the university. The entity has three attributes: key attribute stream\_id and unique stream title and the description for each stream. This entity has 1: N relationship with the Student Entity. *'A stream can have many students', but 'A student belongs to a Stream'.*

### **PROJECTS**

Entity projects stores project related details, each record contains information about a project. It has attributes unique project identifier, unique project title, stream designator i.e. whether the project can be done by CS only students or CS+DS only students or student belonging to any stream can do it. Project\_id is the key attribute of this entity as it uniquely identifies each record. The attribute supervisor\_id relates this to supervisor entity with N:1 relationship, as *'Many projects can be supervised by one supervisor'*.

### **SUPERVISOR**

Entity Supervisor holds the staff related information. Supervisor\_id is the key attribute of this entity and other attributes include personal and the contact details of the staff members like unique Supervisor name, their specialisation stream, email address, DOB, gender and nationality. This entity has 1:N relationship with entity project as *'One Supervisor can supervise many projects.'* Supervisor also has N:N relationship with the entity stream as *'Many supervisor can specialise in more than one stream'*.

### **STUDENT\_PREFERENCES**

Student\_Preferences hold the list of preferences for projects of each student. It is a weak entity, as it depends on the entity student. A student's preference cannot be identified independently as it relies on the Student. Student\_id is the key attribute for this entity. The other attributes of the entity hold the value for the preferred projects by the student. This entity is in 1:1 HAS a relationship with the table student as *'One student has a list of twenty preferences'*.

### **STUDENT\_PROJECT\_MAPPING**

Student\_project\_mapping is the weak entity as it depends on the entity student. Each record contains details of a project assigned to a particular student. Project\_id is the key attribute of this entity, the other attributes include student\_id, prefAllocated (the nth preference allocated). This entity has 1:1 relationship with the Student and the project entity. *'Each student is allotted a project.'*



### ***STUDENT\_SATISFACTION***

Student satisfaction holds the student\_id and satisfaction level for each student. It is a weak entity as it depends on the entity student as no records in student\_satisfaction exist without a student record. Student\_satisfaction entity has 1:1 relationship with the primary key. *'Each student has a satisfaction level'*.


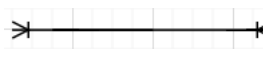


### **ENTITY RELATIONSHIP MODEL:**

Entity-Relationship diagram below depicts the conceptual model for the database of our application. It is a visual description of all the entities and how these entities are related to each other.

#### ***ER Diagram description:***

Each orange coloured rectangular box in below diagram is represents the above-mentioned entities in the system. A weak entity (for instance student preferences) is enclosed in a double rectangular box, different entity attributes have been represented ellipses. All the key attributes are underlined in the ellipses. The attribute with the dotted line beneath is a weak key attribute. Relationships among the entities have been represented by blue coloured diamonds, with weak relationships in double diamond box (for instance relationship between student preferences and student).

The links which connect these entities, with cardinality on these links signify below relationships:

-  : One to One relationship
-  : Many to Many Relationship
-  : Many to One Relationship
-  : One to Many Relationship

## E-ER DIAGRAM

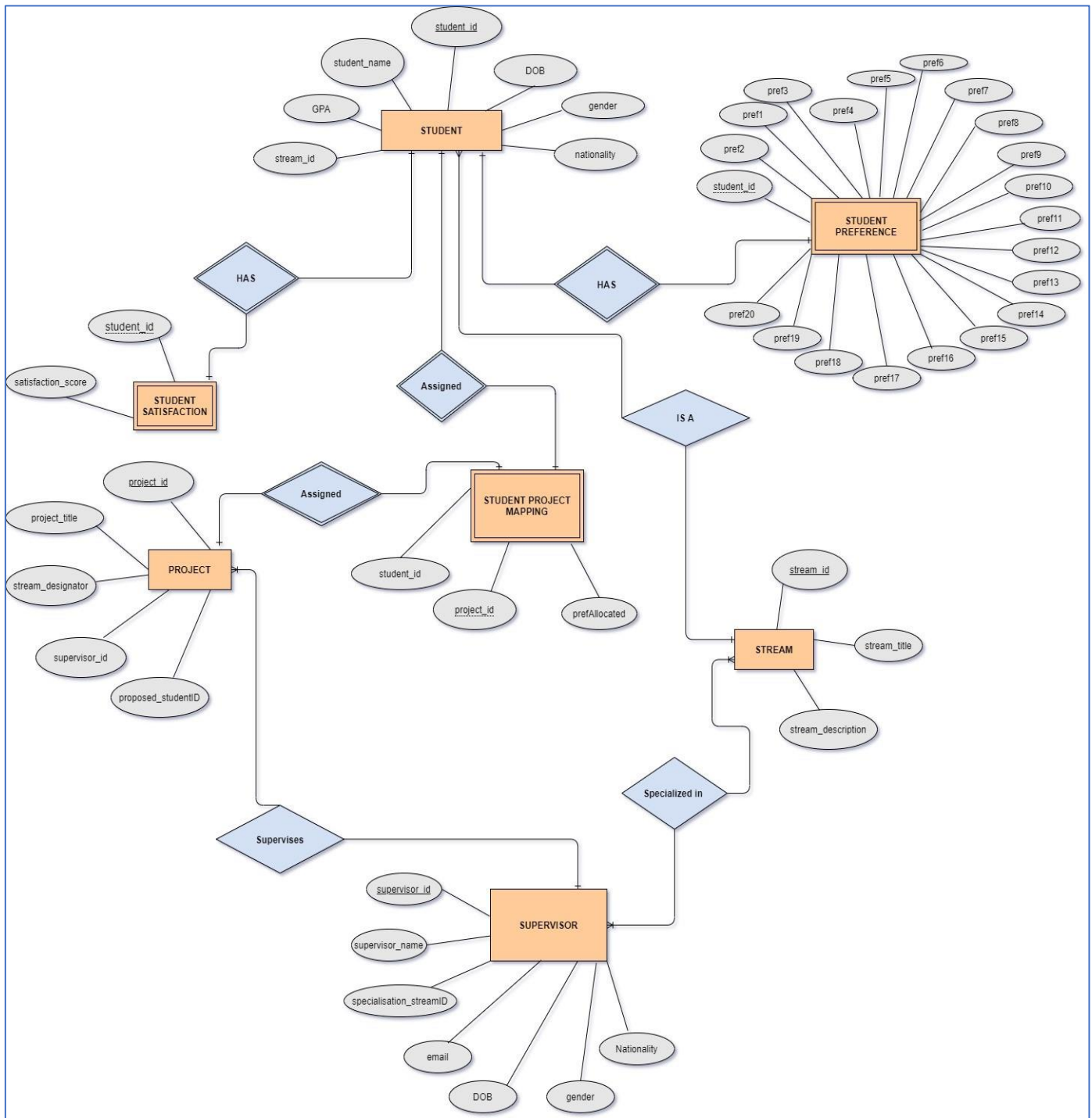


Figure 2: Entity-Relationship Diagram

### **DESIGN MOTIVATION:**

The above ER Diagram represents the logical structure of the underlying database system for The Call of Cthulhu Application, whose design implementation is the main focus of this project. As per the problem statement, a database is to be designed which acts as a repository for the following key elements/entities:

STUDENT, PROJECTS, SUPERVISORS (Staff) and STUDENT PREFERENCES.

STUDENT\_PROJECT\_MAPPING and the STUDENT\_SATISFACTION are the derived tables which would hold the final results of the allocation system. Also, an additional table has been introduced in-order to present the tables in the normalised form and keeping in mind the future work i.e. suppose if we want to add more streams to the system or if we want to change the title of the stream, rather than changing each and every records in all the tables which have stream information, we will require to make changes only to one table.

(NOTE: The problem-solving component used to achieve the final mappings is beyond the scope of this project, however a theoretical concept and how records have been inserted is discussed further in this report.)

### **3. Database Structure: A Normalized View**

This section gives a detailed description of all the tables and their significance, it also includes information on the constraints imposed on each of these table to maintain data integrity and how the normalisation has been achieved to minimize data redundancy and to avoid insert, update and delete anomalies.

#### **TABLE DESCRIPTION:**

##### **1. Student:**

Student table contains the unique student identifier (student\_id), student name(student\_name), Identifier of the Stream/department student belongs to (stream\_id), the grades of the student(GPA), Date-of-Birth (DOB) of the student in the format YYYY-MM-DD, gender and nationality of the student. Student\_id is the primary key for this table and is therefore by default NOT NULL. Stream\_id is the foreign key in this table which links to the primary key of the Stream table. Also, the student name is the full name of the student, some database designers do separate the name as FirstName and LastName, which makes the querying records by only firstname or lastname easy.

##### ***Constraints and design considerations:***

- Student\_id, student\_name, stream\_id and GPA are mandatory columns as will further assist in decision making process and therefore defined as NOT NULL.
- Stram\_id is the foreign key of this table, stream\_name has not been used here because it is a good database design practise to maintain different entities

individually, so that updates in one entity does not requires the changes in another entity.

- GPA should be a decimal number between 0-4.2, this constraint has been achieved through a Trigger discussed further in the report.
- This is an assumption that system should not hold records of the students which are less than 18 years old, this constraint has been again achieved through a Trigger discussed further in the report.

Field	Type	Null	Key	Default	Extra
student_id	varchar(6)	NO	PRI	NULL	
student_name	varchar(40)	NO		NULL	
stream_id	varchar(10)	NO	MUL	NULL	
GPA	decimal(3,2)	NO		NULL	
DOB	date	YES		NULL	
gender	varchar(6)	YES		NULL	
nationality	varchar(10)	YES		NULL	

Figure 3.1 Table Student

## 2. Stream

Table stream is the repository for maintaining data about different streams in the system. As of now, there are only two records and two columns in the table. The table currently has records for streams: 'CS' for CS Only stream and 'CS+DS'.

### Constraints and design considerations:

- Stream\_id is the PRIMARY KEY for the table and by default is NOT NULL
- stream\_title marked UNIQUE to avoid redundancy and cannot be NULL.
- Stream\_description give the description about each stream is again marked as not null.

Field	Type	Null	Key	Default
stream_id	varchar(10)	NO	PRI	NULL
stream_title	varchar(20)	NO		NULL
stream_description	varchar(40)	NO		NULL

Figure 3.2 Table Stream

## 3. Supervisor:

Supervisor table maintains all the staff related information. There is a single record for each supervisor, hence supervisor\_name is the field which holds the full name of

the supervisor and is marked as UNIQUE. The table also contains the personal details of the supervisors: email, DOB(Date Of Birth) in the format YYYY-MM-DD, the gender and nationality of the supervisor.

**Constraints and design considerations:**

- unique identifier supervisor\_id is the PRIMARY key of this table.
- Specialisation\_streamID is the foreign key of this table (like in student table).
- Supervisor\_name is unique so that there exists only 1 record for each staff member and is also a mandatory attribute.

Field	Type	Null	Key	Default	Extra
supervisor_id	varchar(6)	NO	PRI	NULL	
supervisor_name	varchar(40)	NO		NULL	
specialisation_streamID	varchar(10)	NO	MUL	NULL	
email	varchar(40)	NO		NULL	
DOB	date	YES		NULL	
gender	varchar(6)	YES		NULL	
nationality	varchar(10)	YES		NULL	

Figure 3.3 Table Supervisor

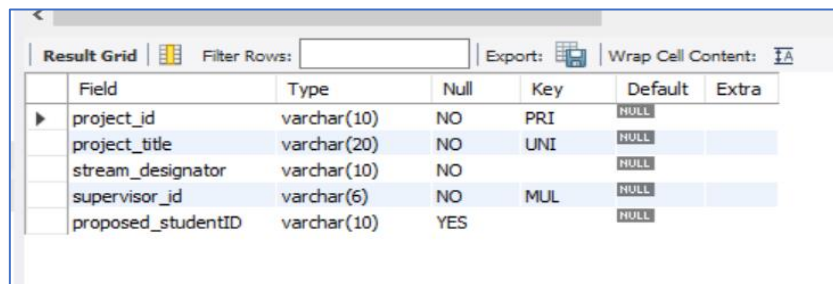
#### 4. Project:

Project table is the repository for all the project related information. There is a single record for each project is the table designated by unique project id and unique project title. Field stream\_designator defines that whether the project can be opted by 'CS Only' enrolled Student or 'CS+DS' student or 'ALL' signifies that students enrolled for any of the above two courses can opt for the projects. This field is also mandatory and hence cannot be NULL. Another field, supervisor\_id is present in the table which signifies the project is supervised by which supervisor. Lastly, most of the projects in the system are staff proposed, but some of the projects could also be student proposed, hence the records for the projects which are proposed by the students will have student\_id populated in this field. This is the student\_id of the student who proposed the project.

**Constraints and design considerations:**

- project\_id is the unique identifier therefore marked as PRIMARY KEY of the table and by default is NOT NULL.
- There should be only 1 record for each project therefore, project\_title is marked as UNIQUE and NOT NULL.

- Not all the projects are student proposed therefore, proposed\_studentID can contain null values.
- Supervisor\_id is the FOREIGN KEY which links each record to a supervisor .



Field	Type	Null	Key	Default	Extra
project_id	varchar(10)	NO	PRI	NULL	
project_title	varchar(20)	NO	UNI	NULL	
stream_designator	varchar(10)	NO		NULL	
supervisor_id	varchar(6)	NO	MUL	NULL	
proposed_studentID	varchar(10)	YES		NULL	

Figure 3.4 Table Projects

### 5. Student\_Preferences

Table Student\_Preferences maintains the list of the preferred projects by each student, first field is Student\_id and the other 20 fields contain the project\_id's of the project preferred by each student. The highly preferred project is stored in the second field and consequently each column has the lesser preferred project\_id. This field not marked as mandatory as some students might not enter all the twenty preferences.

#### Constraints and design considerations:

- Student\_id marked as PRIMARY KEY as it uniquely identifies each record.
- Student\_id is also marked as foreign key to link it to student table, as student preference cannot exist if the student doesn't exist in the system.
- Pref\_1 is mandatory. Hence marked as not null.
- To maintain the authenticity of the preferences entered we can make the 20 preference fields as foreign keys but this would complicate the system, hence an idea of how we can implement this through stored procedures has been discussed in the procedures section.

Field	Type	Null	Key	Default	Extra
student_id	varchar(6)	NO	PRI	NULL	
pref1	varchar(10)	NO		NULL	
pref2	varchar(10)	YES		NULL	
pref3	varchar(10)	YES		NULL	
pref4	varchar(10)	YES		NULL	
pref5	varchar(10)	YES		NULL	
pref6	varchar(10)	YES		NULL	
pref7	varchar(10)	YES		NULL	
pref8	varchar(10)	YES		NULL	
pref9	varchar(10)	YES		NULL	
pref10	varchar(10)	YES		NULL	
pref11	varchar(10)	YES		NULL	
pref12	varchar(10)	YES		NULL	

Figure 3.5 Table Student Preferences

## 6. Student\_Project\_Mapping

### PROJECT ALLOCATION RULES AND ASSUMPTIONS:

Before describing the table, below is a discussion on the rules or assumption on how the projects will be allotted to each student (It is assumed the implementation of these rules is done at application level):

1. Each student is going to state the list of at max 20 projects they would prefer to do.  
With Preference 1 being most desired project. Each student needs to mention at least 1 preference, some of the students might also try to cheat the system by entering same preference multiple times.
2. Each student should be allotted project aligned as per his/her stream. For Instance, CS Student can be allotted a 'CS-Only Project' or projects with stream designator 'ALL', as those projects can be allotted to 'CS Only or/and CS+DS' students.
3. Students can also self-propose a project, if self-proposed project aligns with his/her stream, then the project should be allotted to the student.
4. Each student should be allotted a project and each project should be allotted to only 1 student and all the projects must be equally distributed among the projects.
5. High GPA means high probability of getting most preferred project, as per the availability.
6. Supervisors have been allocated projects equally.

APPLICATION PROJECT RULES
Only those projects will be assigned to the student which align to their stream even though student expresses an interest in a project which does not aligns to his stream.
If two students have expressed same projects at same preference level, then the student whose stream is aligned with the project and with higher grade should be allotted the project.
If two students have same preferences and same grades, then one of them would be allotted on first come first serve basis (As of now we are not storing timestamp now but this possibly could be a future work)
If all the preferences of the student have been allotted to other students already, then system will assign from the left-over projects.
STUDENT PROPOSED PROJECT RULES
If a student proposes a project, the project should be assigned to him/her only.
If a student proposes a project but does not enters the project as the first preference in preference, then the student will be allotted the project as per the application rules above.

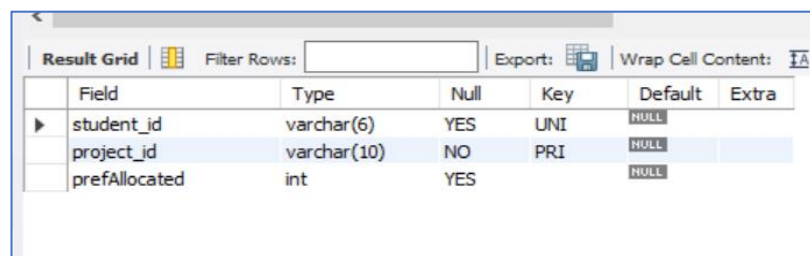
Figure 3.6 Project Allocation Rules Table

Student\_project\_mapping table stores the mapping for the project allotted to each student using problem solving AI component of the application. Last field in this table is pref\_allocated, it is the INT value which stores the preference number of the project allotted.

**[NOTE: Records have been inserted into the table keeping in mind the above rules]**

**Constraints and design considerations:**

- project\_id is the PRIMARY KEY for this table.
- A CONSTRAINT has been added on this table through stored procedure such that entry will be created in this table only if the project record exists in the project table.
- Student\_id is the student identifier and each student can have only one project assigned to him/her, therefore it is marked as UNIQUE and is a mandatory field as each student should be assigned a project. Also, student\_id is the foreign\_key of this table, as every student record should belong to a valid student\_id.
- Project\_id is also made foreign\_key in this table, as each project\_id should be of a valid project. But this check could also be done through a stored procedure. (Discussed in procedures section)



Field	Type	Null	Key	Default	Extra
student_id	varchar(6)	YES	UNI	NULL	
project_id	varchar(10)	NO	PRI	NULL	
prefAllocated	int	YES		NULL	

Figure 3.7 Table Student Project Mapping

## 7. Student\_Satisfaction

### SATISFACTION LEVEL CALCULATION RULES AND ASSUMPTIONS:

This section discusses about the rules or assumptions on how the satisfaction-level is associated with this project.

1. Local satisfaction of a student is subjective to the preference received by him/her, the student who received the most preferred subject will most satisfied and the one allotted the last preference will be least satisfied. This satisfaction will be stored into the student\_satisfaction table.



2. I have assumed that Global satisfaction i.e. satisfaction level of overall cohort is calculated at the application level, which is a sum of local satisfaction and the weightage of the other extenuating factors in the project. (not scope of this report but could be calculate like take average satisfaction level of overall stream)
3. Below table presents how the satisfaction scores will be calculated by the application or the same could also be done using a stored procedure. The perfAllocated field in the project table can be used to calculate the satisfaction score. If the student receives the preferred project local satisfaction score will 100, if he receives the 20<sup>th</sup> preference then his satisfaction score will be 5 and the one who receives a project for which he did not express any interest his satisfaction score will be 0.

Preference Number	Satisfaction Percentage
0	0
1	100
2	95
3	90
4	85
5	80
6	75
7	70
8	65
9	60
10	55
11	50
12	45
13	40
14	35
15	30
16	25
17	20
18	15
19	10
20	5

Figure 3.8 Local Satisfaction Score Calculation Table

This table contains two fields: student\_id and statisfaction\_score which represents the percentage of local satisfaction level of each student according to above table. (Insert statements have been given for this project, however this could also be handled through stored procedure by calculating satisfaction score every time insert is made into this table.)

**Constraints and design considerations:**

- Student\_id uniquely identifies each record and therefore marked as the PRIMARY\_KEY of the table. It is also foreign key of this table as it links each student\_id to the student present in the student table.

Result Grid   Filter Rows:   Export:   Wrap Cell Content:						
	Field	Type	Null	Key	Default	Extra
▶	student_id	varchar(6)	NO	PRI	NULL	
	satisfaction_score	int	YES		NULL	

Figure 3.9 Table Student Satisfaction

## **DATABASE NORMALISATION**

Database normalisation is a systematic technique of decomposing tables for simplified querying, eliminating redundancy and to avoid Insertion, Update and Deletion Anomalies. Below is the explanation of how this database conforms to norms of normalisation:

### ***First normal form(1NF)***

This rule states that an attribute of a table cannot hold multiple values. It should hold only atomic values. By imposing a unique primary key, we can make ensure that there are no redundant, repeating groups of data.

All the tables in this application have a primary key. Also, all the fields in these table have atomic values. For Instance, instead of having separate student\_preference fields we could insert comma separated all the preferences of the students in the student table in one field, or each preference in a new row but this would lead to redundant student ids in the table. Hence, primary key would restrict this redundancy.

### ***Second normal form(2NF)***

A table is said to be in 2NF if:

- Table is in 1NF
- There is no partial dependency of any field on the primary key. By partial dependency, we mean that for instance if primary key is made up of more than one attribute then non-key attributes should be dependent on the whole key.

There is no partial dependency in these tables, each of them has a single primary key and all the records in the table depend on the primary key of the table as a whole.

### ***Third normal form(3NF)***

A table is said to be in the Third Normal Form when,

- It is in the 2NF.
- And, it doesn't have Transitive Dependency. Transitive dependency means changing any one non-key attribute might cause changes in the other non-key attributes. The non key attributes should be dependent on nothing but the entire primary key.

All the tables in the application are in third normal form. There exists no transitive dependency in any of the tables. For instance, consider in the table student\_project\_mapping instead of supervisor\_id if there were all the supervisor details present in the table. Now, if in future if we wanted to change the supervisor for a project, then it would require to change all the other details of the supervisor as well, this would break the 3NF rule. Instead as per the current normalised tables, if we want to change the supervisor of the project, we only have to change the supervisor\_id in the student\_project\_mapping Table.

### ***Boyce & Codd normal form (BCNF)***

A table is said to be in BCNF when,

- Table must be in 3rd Normal Form
- and, for each functional dependency ( $X \rightarrow Y$ ), X should be a super Key.

The tables described above for this application conform to the Boyce Code Normal form, there are no non-functional trivial dependencies in these tables. For instance in the Table STUDENT, student\_id is the super key and all the other attributes depend on this key. Similarly, with the project table, project\_id is the super key of all the relations. If it had been like {Project\_id  $\rightarrow$  Project\_title, stream\_designator, supervisor\_id} and {student\_id  $\rightarrow$  Student\_name}, all the project detail and the project mapping details in the same table, This would have result in violation BCNF rules, instead we have three separate tables for Student details, Project details, and for Student\_project\_mapping. Also, even though project title in projects title is unique and can be used as an identifier of the record but project\_id is the only primary key of the table and all the other attributes of the table rely on it.

Extended entity diagram represents the entire *final proposed structure of the database with the tables*. Each rectangular box in below diagram is represents the above-described entities/tables. Table name is enclosed in the blue heading of each box, below is the list of different entity attributes, primary key of each entity has a key before its name. The attributes with red diamond are the foreign keys of each entity. The links which connect these entities, with cardinality on these links signifying the type of relationship.

Also, the solid lines represent the identifying relationship, whereas the disconnected lines represent the non-identifying relationship, as explained above the child entity can stand on its own without the parent entity.

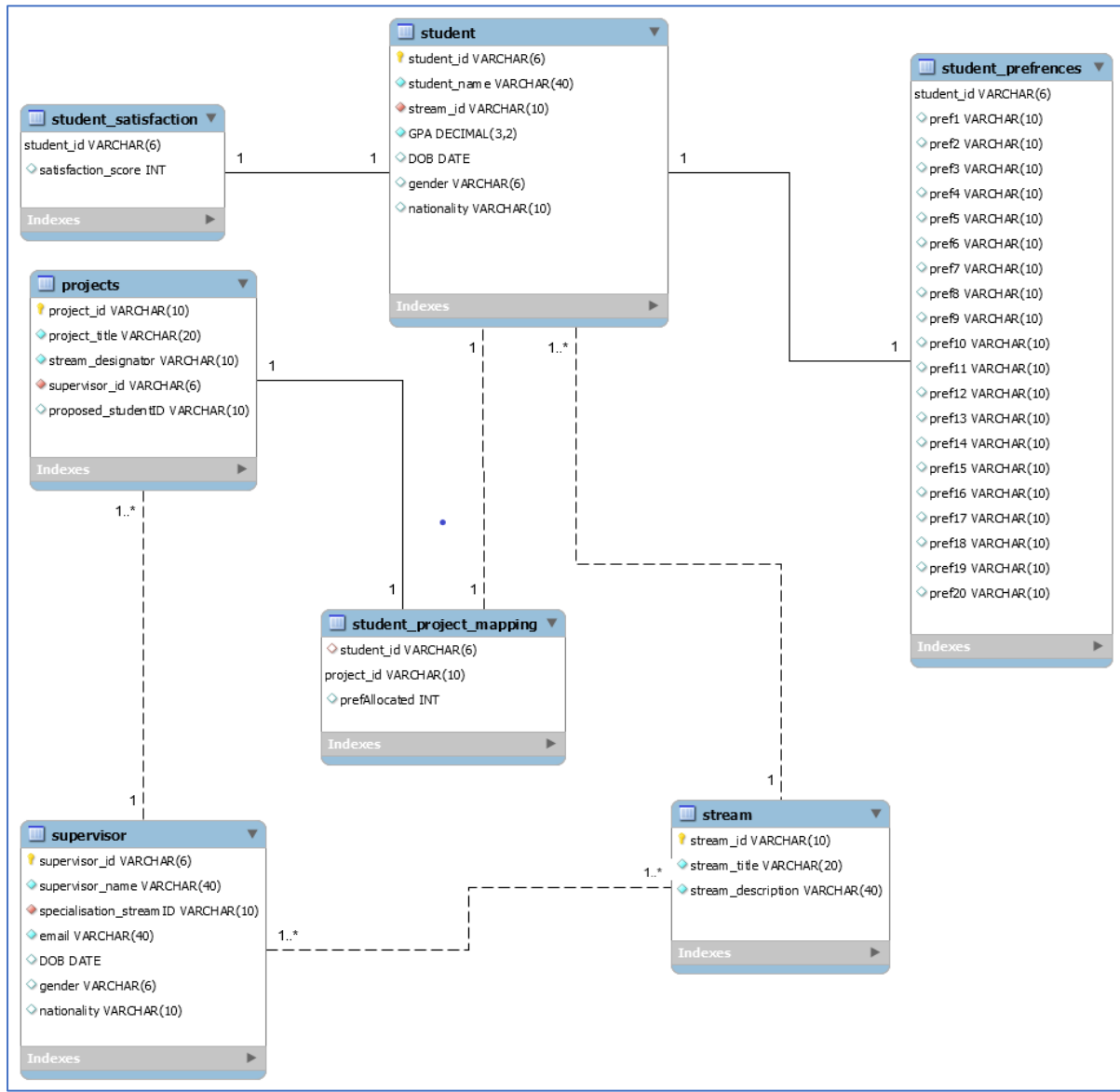


Figure 3.10: EER Extended Entity Relationship Diagram

## 4. Procedural Elements

Procedures are of SQL statements that are created to perform one or more DML operations. Procedures or functions are used when a same set of operations are to be performed multiple times. Stored procedures accept some input parameters and may or may not return a value.

We have created procedures in our system to enforce constraints on some of the tables in order to maintain the structural integrity. Below is the list of such procedures and triggers for them respectively.

### ***PROCEDURE 1: validate\_student\_gpa***

**Explanation:** validate\_student\_gpa is stored procedure used to impose constraint on the student table. As per the requirements, the GPA value for a student should be between 0 to 4.2. This procedure takes the GPA value as the input and check if the value is less than 0 or if greater than 4.2, then it throws an error message saying as an invalid input. This procedure has no return parameters.

```
DELIMITER //
CREATE PROCEDURE validate_Student_gpa(IN GPA DECIMAL)
DETERMINISTIC BEGIN IF (SELECT FLOOR(GPA-0)) <= 0.00 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'INVALID INSERT! GPA should
be greater than 0';
    END IF;
    IF GPA > 4.20 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'INVALID INSERT! GPA should be less than 4.2';
    END IF;
END // DELIMITER;
```

### ***TRIGGER1: validate\_gpa\_insert & validate\_gpa\_update***

**Explanation:** validate\_gpa\_insert and validate\_gpa\_update triggers below get triggered every time an insert or update statement is run on the student table. On every record inserted/updated, these triggers invoke call to stored procedure validate\_Student\_gpa by passing GPA from the record as the input.

```
DELIMITER //
CREATE TRIGGER validate_gpa_insert
```

```

BEFORE INSERT ON Student FOR EACH ROW
BEGIN CALL validate_Student_gpa (NEW.GPA);
END // DELIMITER ;

DELIMITER // CREATE TRIGGER validate_gpa_update
BEFORE UPDATE ON Student FOR EACH ROW
BEGIN CALL validate_Student_gpa ( NEW.GPA);
END // DELIMITER ;

```

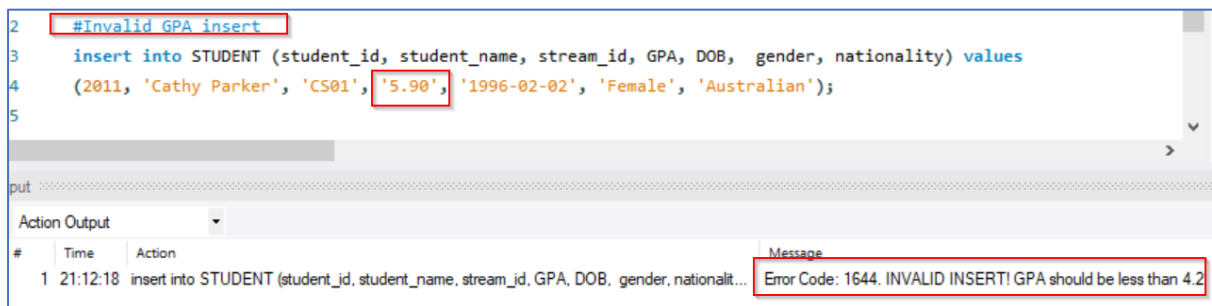


Figure 4.1 Procedure & Trigger 1 Results

## PROCEDURE 2: validate\_age

**Explanation:** validate\_age is stored procedure used to impose constraint on the student table. As per the assumptions, the age of a student should be greater than 18. This procedure takes the age value as the input and check if the value is less than 18, then it throws an error message saying as an invalid input. This procedure also has no return parameters.

```

DROP PROCEDURE IF EXISTS validate_age;
DELIMITER // CREATE PROCEDURE validate_age( IN DOB date)
DETERMINISTIC
BEGIN IF (SELECT FLOOR(DATEDIFF(NOW(), DATE(DOB))/365)) < 18 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'INVALID student age!!'; END IF;
END // DELIMITER ;

```

## TRIGGER2: validate\_age\_insert & validate\_age\_update

**Explanation:** validate\_age\_insert and validate\_age\_update triggers below get triggered every time an insert or update statement is run on the student table. On every record inserted/updated, these triggers invoke call to stored procedure validate\_age by passing age from the record as the input.

```

DELIMITER // CREATE TRIGGER validate_age_insert
BEFORE INSERT ON Student FOR EACH ROW

```

```

BEGIN CALL validate_age(NEW.DOB);
END // DELIMITER ;

DELIMITER // CREATE TRIGGER validate_age_update
BEFORE UPDATE ON Student FOR EACH ROW
BEGIN CALL validate_age(NEW.DOB);
END // DELIMITER ;

```

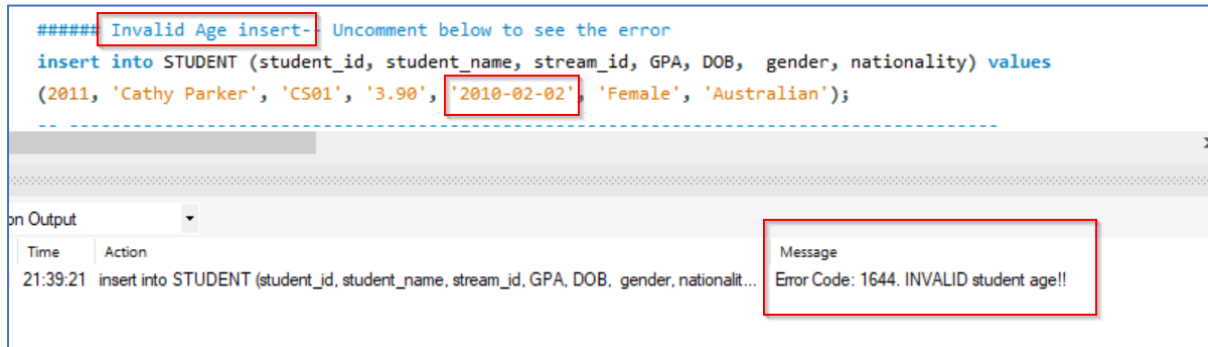


Figure 4.2 Procedure & Trigger 2 Results

### PROCEDURE 3: validate\_project\_exists

**Explanation:** validate\_project\_exists is stored procedure used to impose constraint on the student\_project\_mapping table. This procedure checks if the input project\_id is not present in the project inventory, then an error message will be thrown and the record with invalid project id will not be stored in the database.

```

DROP PROCEDURE IF EXISTS validate_project_exists;
DELIMITER // CREATE PROCEDURE validate_project_exists( IN projectID VARCHAR(6) )
DETERMINISTIC
BEGIN IF (select count(*)from projects p where p.project_id=projectID) =0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Project not found in the system';    END
IF; END // DELIMITER ;

```

### TRIGGER3: check\_project\_exists

**Explanation:** check\_project\_exists triggers below get triggered every time an insert statement is run on the student\_project mapping table. On every record inserted, these trigger invoke call to stored procedure validate\_project\_exists by passing project\_id from the record as the input.

```

DELIMITER //
CREATE TRIGGER check_project_exists
BEFORE INSERT ON Student_Project_Mapping FOR EACH ROW
BEGIN CALL validate_project_exists(NEW.project_id); END // DELIMITER ;

```

**[NOTE-Similar, Procedure-trigger can be applied to each of the 20 preference fields of Student\_preferences table, so that only valid preferences are entered into the system. This would**

be an alternate of marking preferences as foreign key in student\_preference\_table but would require 20 AND conditions.]

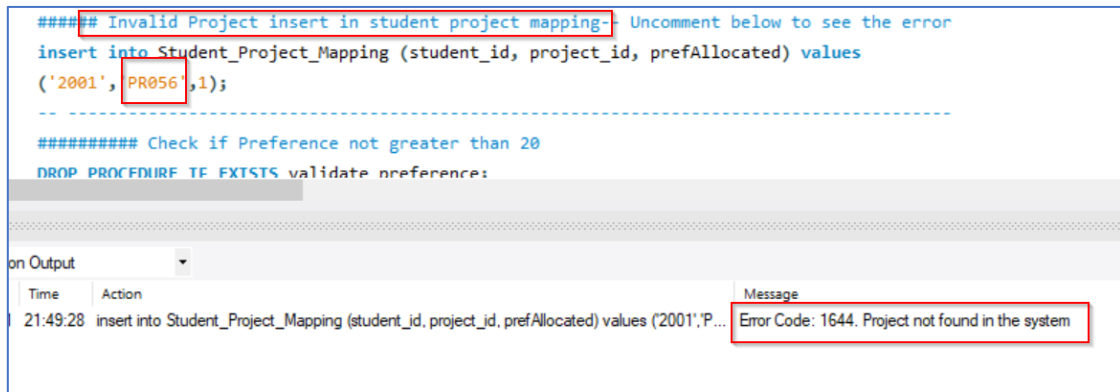


Figure 4.3 Procedure & Trigger 3 Results

#### **PROCEDURE 4: Validate each student is assigned a project aligned with the stream**

**Explanation:** validate\_project is stored procedure used to impose constraint on the student\_project\_mapping table. This procedure checks if the input project\_id is aligned with stream of the student\_id to whom the project is allotted. First it checks if the stream designator for the project is 'ALL' in the project table, then skip it. If not check with the stream of student and if it does not match then an error message will be thrown and the record with invalid data will not be stored in the database.

```

DROP PROCEDURE IF EXISTS validate_project;
DELIMITER // CREATE PROCEDURE validate_project( IN studentID VARCHAR(6), IN projectID
VARCHAR(6) ) DETERMINISTIC BEGIN
IF (select stream_designator from projects where project_id=projectID)='ALL' THEN
    SIGNAL SQLSTATE '01000';
ELSEIF (select count(*) from student where student_id=studentID and stream_id=(select (CASE
stream_designator when 'CS' then 'CS01' when 'CS+DS' then 'DS01' END) from projects where
project_id=projectID) ) =0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid Insert!
Possible Project and Student stream mismatch!'; END IF; END // DELIMITER ;

```

#### **TRIGGER4: check\_project**

**Explanation:** check\_project triggers below get triggered every time an insert statement is run on the student\_project mapping table. On every record inserted, these trigger invoke call to stored procedure validate\_project by passing project\_id and student\_id from the record as the input.

```

DELIMITER // CREATE TRIGGER check_project
BEFORE INSERT ON Student_Project_Mapping FOR EACH ROW
BEGIN CALL validate_project(NEW.student_id,NEW.project_id);
END // DELIMITER ;

```



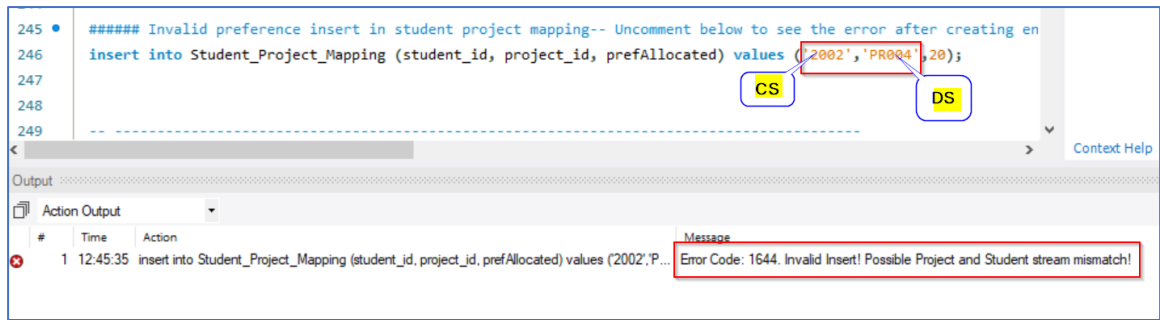


Figure 4.4 Procedure & Trigger 4 Results

### PROCEDURE 5: validate\_preference

**Explanation:** validate\_preference is stored procedure used to impose constraint on the student\_project\_mapping table. As mentioned above, this table stores the preference number allotted, which further is used to calculate the satisfaction score. This procedure takes the preference number as the input and check if the value is greater than 20 then it throws an error message saying as an invalid input, as a student can give maximum 20 preferences and will be allotted only one project out of all the preferences. However, this field can contain 0. 0 is assigned when the student gets a project which he has not expressed interest in. This procedure has no return parameters.

```
DROP PROCEDURE IF EXISTS validate_preference;
DELIMITER // CREATE PROCEDURE validate_preference( IN prefAllocated INT)
DETERMINISTIC
BEGIN IF (SELECT prefAllocated) > 20 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
'INVALID Preference number';
END IF; END // DELIMITER ;
```

### TRIGGER5: validate\_preference\_insert & validate\_preference\_update

**Explanation:** validate\_preference\_insert and validate\_preference\_update triggers below get triggered every time an insert or update statement is run on the student\_projec\_mapping table. On every record inserted/updated, these triggers invoke call to stored procedure validate\_preference by passing prefAllocated number from the record as the input.

```
DELIMITER //
CREATE TRIGGER validate_preference_insert
BEFORE INSERT ON Student_Project_Mapping FOR EACH ROW
BEGIN CALL validate_preference(NEW.prefAllocated);
END // DELIMITER ;
DELIMITER //
```

```
CREATE TRIGGER validate_preference_update
BEFORE UPDATE ON Student_Project_Mapping FOR EACH ROW
BEGIN CALL validate_preference(NEW.prefAllocated);
END // DELIMITER ;
```

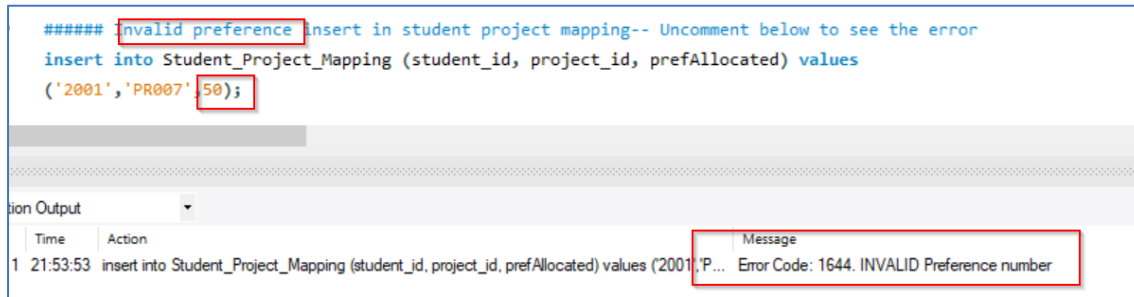


Figure 4.5 Procedure & Trigger 5 Results

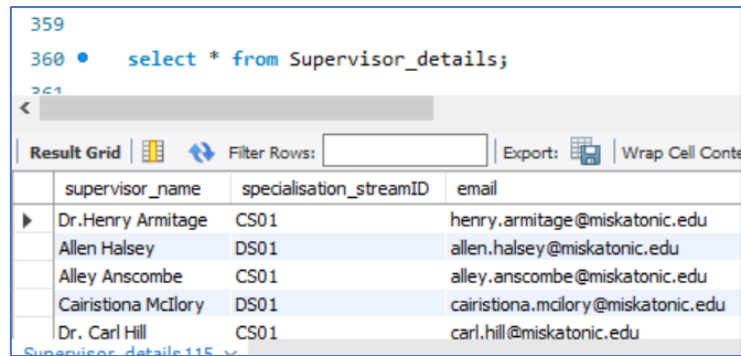
## 5. Database Views

Views in SQL are virtual tables, which are created by selecting attributes from one or more tables in the database. These views may or may not contain the entire records of the tables on which they are created. View are generated in order to restrict access from complete information of the table. Just like tables views can be created, updated or deleted. Views created on only one tables are known as simple views and views created using two or more tables are known complex views. Below are few simple and complex views created for the project allocation system, keeping in mind the purpose each view would serve to a particular user.

**VIEW1: Create a view of professors in case students want to know the supervisors of the project**

```
CREATE VIEW Supervisor_details
AS SELECT supervisor_name, specialisation_streamID, email
FROM Supervisor;
```

**Explanation:** This is a simple view created on the table Supervisors; this view would be one-stop solution to fetch the supervisor contact details. This view could be very useful to the students who want to contact the supervisors in case if any queries or if they want to gather information about the specialisation field of the supervisor and accordingly choose the project. Students, staff and admin all can access this view.



359  
360 • select \* from Supervisor\_details;  
361

Result Grid | Filter Rows: | Export: | Wrap Cell Conte

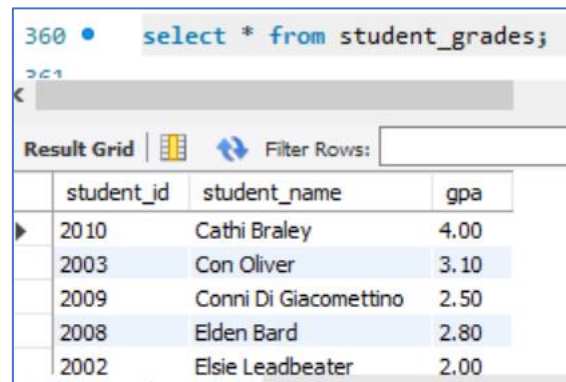
supervisor_name	specialisation_streamID	email
Dr. Henry Armitage	CS01	henry.armitage@miskatonic.edu
Allen Halsey	DS01	allen.halsey@miskatonic.edu
Alley Anscombe	CS01	alley.anscombe@miskatonic.edu
Cairistiona McIlory	DS01	cairistiona.mclory@miskatonic.edu
Dr. Carl Hill	CS01	carl.hill@miskatonic.edu

Figure 5.1: View 1

**VIEW2: Create a view to access students grades with student's name arranged in alphabetical order**

```
CREATE VIEW student_grades
AS SELECT student_id, student_name, gpa
FROM student ORDER BY student_name;
```

**Explanation:** Student\_grades is again a simple view, which displays a list of student\_id, student\_name and the student\_grades, arranged in the alphabetical order. This view could be used by the supervisors to know the grades of the students or by the admins if they want to get a report of students' performance. This view should be restricted to Supervisors and admins, so that one student cannot view the grades of another student.



360 • select \* from student\_grades;  
361

Result Grid | Filter Rows: |

student_id	student_name	gpa
2010	Cathi Braley	4.00
2003	Con Oliver	3.10
2009	Conni Di Giacomettino	2.50
2008	Elden Bard	2.80
2002	Elsie Leadbeater	2.00

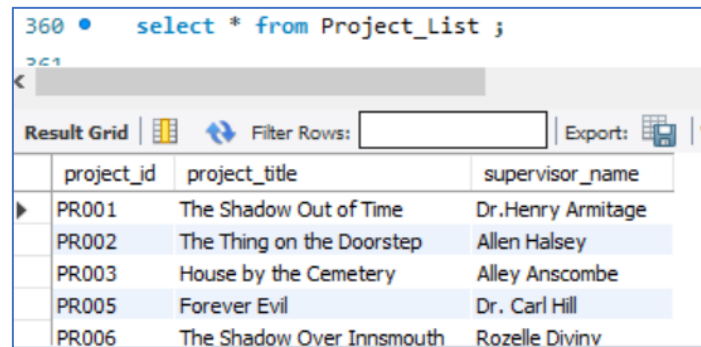
Figure 5.2: View 2

**VIEW3: Create a view to display all the Staff proposed projects along with the supervisor name**

```
CREATE VIEW Project_List AS
SELECT p.project_id, p.project_title, s.supervisor_name
FROM projects p INNER JOIN supervisor s ON s.supervisor_id = p.supervisor_id
WHERE p.proposed_studentID IS NULL;
```

**Explanation:** Project\_List view is a complex view that displays a list of projects that have been proposed by the Staff members. This view will fetch the list of project id, project title along with the supervisor name. This view is generated by joining the

supervisor table and project table, a join to supervisor table has been done in order to fetch the supervisor name along with the project titles. This view is very useful to students to a list of projects so that they can fill their preferences accordingly. Even the staff can be granted access to this view. Both the Staff members and students can have read only access to this view, update access to these views should not be granted to them as not every staff member should have the permission to updates these tables.



```
360 • select * from Project_List ;
361
```

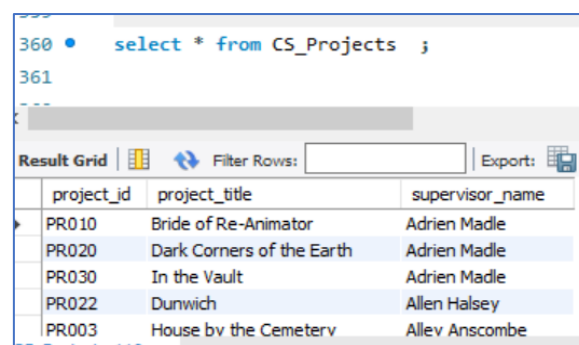
project_id	project_title	supervisor_name
PR001	The Shadow Out of Time	Dr.Henry Armitage
PR002	The Thing on the Doorstep	Allen Halsey
PR003	House by the Cemetery	Alley Anscombe
PR005	Forever Evil	Dr. Carl Hill
PR006	The Shadow Over Innsmouth	Rozelle Divin

Figure 5.3: View 3

**VIEW4: Create a view to display Projects for CS Students Only along with the supervisor name**

```
CREATE VIEW CS_Projects AS SELECT project_id, project_title, s.supervisor_name
FROM projects p INNER JOIN supervisor s ON s.supervisor_id = p.supervisor_id
WHERE p.proposed_studentID IS NULL AND p.stream_designator IN ('CS', 'ALL');
```

**Explanation:** CS\_Projects view is a complex view that displays a list of projects that should be made available to CS Students Only. This view will fetch the list of project id, project title and supervisor name of all the projects that are suitable for 'CS Only' or 'CS and/or CS+DS' students. This view is generated by joining the supervisor table and project table, a join to supervisor table has been done in order to fetch the supervisor name along with the project titles. This view could be useful to students to get a filtered list of projects rather than a complete set of projects in the system. Even the staff can be granted access to this view.



```
360 • select * from CS_Projects ;
361
```

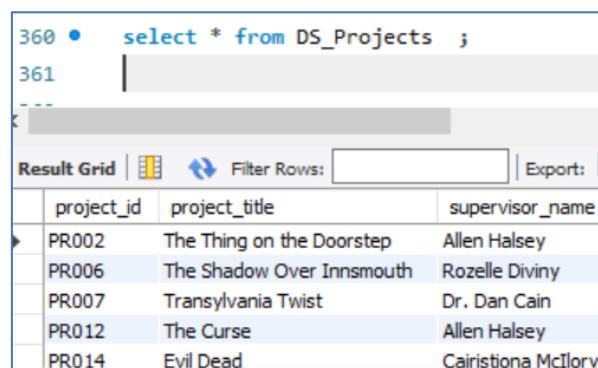
project_id	project_title	supervisor_name
PR010	Bride of Re-Animator	Adrien Madle
PR020	Dark Corners of the Earth	Adrien Madle
PR030	In the Vault	Adrien Madle
PR022	Dunwich	Allen Halsey
PR003	House by the Cemetery	Alley Anscombe

Figure 5.4: View 4

**VIEW5: Create a view to display Projects for CS+DS Students Only along with the supervisor name**

```
CREATE VIEW DS_Projects AS
SELECT project_id, project_title, s.supervisor_name FROM
projects p INNER JOIN supervisor s ON s.supervisor_id = p.supervisor_id WHERE
p.proposed_studentID IS NULL AND p.stream_designator IN ('CS+DS', 'ALL');
```

**Explanation:** DS\_Projects view is a complex view similar to the previous that displays a list of projects that should be made available to CS+DS Students Only. This view will fetch the list of project id, project title and supervisor name of all the projects that are suitable for 'CS+DS Only' or 'CS and/or CS+DS' students. This view is generated by joining the supervisor table and project table, a join to supervisor table has been done in order to fetch the supervisor name along with the project titles. This view could be useful to students to get a filtered list of projects rather than a complete set of projects in the system. Even the staff can be granted access to this view.



project_id	project_title	supervisor_name
PR002	The Thing on the Doorstep	Allen Halsey
PR006	The Shadow Over Innsmouth	Rozelle Diviny
PR007	Transylvania Twist	Dr. Dan Cain
PR012	The Curse	Allen Halsey
PR014	Evil Dead	Cairistiona McIlroy

Figure 5.5: View 5

## 6. Example Queries

This section contains a list of sample queries that could be run by the students, staff members or the admin in order to fetch the relevant data.

**QUERY 1: Find the number of students in each stream**

**Explanation:** The below query runs on the table student and table stream to fetch the number of students in each stream. This query utilizes aggregate function COUNT to find the total number of students reduced using GROUP BY operation on the stream\_id. This query could be used by the staff members, admins or by other University for the purpose of reporting.

```
SELECT s2.stream description, COUNT(*) as total_students
FROM student s1, stream s2
where s1.stream_id =s2.stream_id
```

*GROUP BY s1.stream\_id;*

	stream_description	total_students
▶	Cthulhu Studies	4
	Dagon Studies	6

Figure 6.1: Query Output1

**QUERY 2: Find top three students can be done through Union also**

**Explanation:** The below query could be used by the staff members, admins in order to find out the name of top 3 students among both the streams. This query runs on the view student\_grades, but could also be run on the tables student directly. This query makes use of inner query to fill perform the comparison of grades and the outer query filters the top three results and display the records in the decreasing order of GPA using ORDER BY clause.

```
SELECT student_name, gpa FROM student_grades a
WHERE 3 >= (SELECT COUNT(gpa) FROM student_grades b
WHERE a.gpa <= b.gpa) ORDER BY gpa DESC;
```

	student_name	gpa
▶	Gar Iacovelli	4.19
	Cathi Braley	4.00
	Samuele Arber	3.97

Figure 6.2: Query Output2

**QUERY 3: List all the students in the system which got their first preferences**

**Explanation:** Below query again uses an inner query to find the student\_id of all the students, who received their first preference from student\_project\_mapping table, then checks these ids in student table to fetch the names and grades of the respective students. The final result is displayed in the decreasing order of grades. This query could be done to carry out the analysis that students with higher grades were given their first preferred choices.

```
SELECT student_name, gpa AS grades
FROM student WHERE student_id IN (SELECT student_id
FROM student_project_mapping WHERE prefAllocated = 1) order by grades desc;
```

	student_name	grades
►	Gar Iacovelli	4.19
	Cathi Braley	4.00
	Samuele Arber	3.97
	Urbain Lyhane	3.90
	Mandy Ilive	3.57
	Morty McCobb	2.90
	Elsie Leadbeater	2.00

Figure 6.3: Query Output3

This query can also be altered further to retrieve the list of such students got projects say among their top 5 preferences. This could be done by changing condition to  $\text{prefAllocated} < 6$ .

**QUERY 4: Write a query to check if the student who got the project, they proposed**

**Explanation:** Below query can be utilised for the reporting tasks to analyse which all students got the projects they desired for. The data is retrieved by performing an inner join of student\_project\_mapping table with project and the student table. First column retrieved is the student id , second column is the student name, third column displays the project allotted to the respective student from student\_mapping table, fourth column again displays the student id from project table, but this is done to display the project proposed by that student in the fifth column.

```
select s.student_id, s.student_name , sp.project_id as alloted_project,
p.proposed_studentID, p.project_id as proposed_project
from Student_Project_Mapping sp
inner join student s on s.student_id =sp.student_id
inner join projects p on p.project_id =sp.project_id
where p.proposed_studentID is not null;
```

	student_id	student_name	alloted_project	proposed_studentID	proposed_project
►	2001	Urbain Lyhane	PR004	2001	PR004
	2004	Gar Iacovelli	PR012	2004	PR012
	2005	Mandy Ilive	PR018	2005	PR018
	2010	Cathi Braley	PR016	2010	PR016

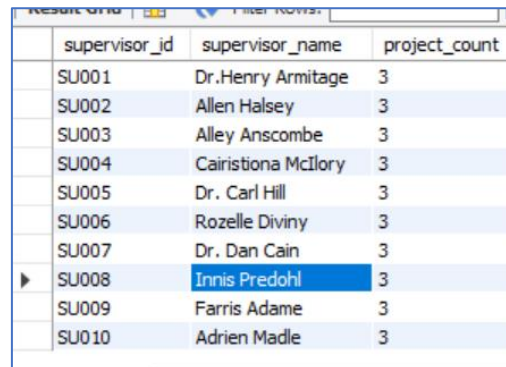
Figure 6.4: Query Output4

**QUERY 5: Display the number of projects assigned to each of the supervisor**

**Explanation:** Below is the query which displays the number of projects assigned to each of the staff member. This is a complex query, joins and group by clause. The inner query counts the number of projects assigned to each of the supervisor in project table

by grouping on the basis of supervisor\_id. These results are then joined with the supervisor table to retrieve the name of each of the staff member.

```
select s.supervisor_id, s.supervisor_name, p.project_count
from supervisor s join
(select count(1) as project_count, supervisor_id from projects
group by supervisor_id
) p on s.supervisor_id=p.supervisor_id;
```



supervisor_id	supervisor_name	project_count
SU001	Dr. Henry Armitage	3
SU002	Allen Halsey	3
SU003	Alley Anscombe	3
SU004	Cairistiona McClory	3
SU005	Dr. Carl Hill	3
SU006	Rozelle Diviny	3
SU007	Dr. Dan Cain	3
SU008	Innis Predohl	3
SU009	Farris Adame	3
SU010	Adrien Madle	3

Figure 6.5: Query Output5

**QUERY 6: List all the students which were assigned projects not aligned with their stream**

**Explanation:** Below complex query is to retrieve the list of the students who have actually received projects that are not aligned with their stream. The data is retrieved by performing an inner join of student\_project\_mapping table with project and the student table. CASE statement has been used here to display the stream\_title for which student belongs, projects having stream\_designator are not included in this query, as those were the projects that were intended for all type of students.



This query would also help the admins to analyse the results of the problem-solving component.

**[NOTE-This query will return 0 records, as all students have been assigned projects that align with their stream, the second SS below is the final output, the first SS is the one which was taken with wrong dataset, that is just for displaying the outcome of the query]**

```
select s.student_name, (CASE s.stream_id when 'CS01' then 'CS' when 'DS01' then 'CS+DS' END) AS
student_stream, p.project_title, p.stream_designator as project_stream
from student_project_mapping sp
inner join student s on sp.student_id = s.student_id
inner join projects p on sp.project_id = p.project_id
where (s.stream_id != (CASE p.stream_designator when 'CS' then 'CS01' when 'CS+DS' then 'DS01'
else 'ALL' END)) AND p.stream_designator != 'ALL';
```





Result Grid






Filter Rows:

Export:

Wrap Cell Content:

	student_name	student_stream	project_title	project_stream

Figure 6.6: Query Output6

**QUERY 7: List the Top 3 Most preferred Projects by the students**

**Explanation:** Below complex query is to retrieve the list of top three most demanded projects in the system. Only pref1, pref2 and pref3 have been checked in the this as the students are like to mention their top preferred projects first, checking all the columns might also give wrong result as some students might have entered same project multiple times. Below query uses Joins, Union operator, aggregate function count() using group by clause, first we retrieve the projects mentioned in pref1,pref2 and pref3 combine them using UNION and then a count is done on all the projects grouped by project id, the results from this inner query is joined with the project table, so that project title can also be retrieved.

```

SELECT p.project_id, p.project_title, p2.preference_count
FROM projects p inner join
(SELECT projectID, COUNT(*) as preference_count
FROM ((SELECT pref1 as projectID FROM Student_Preferences ) UNION ALL
(SELECT pref2 as projectID FROM Student_Preferences) UNION ALL
(SELECT pref3 as projectID FROM Student_Preferences)) p1 GROUP BY projectID
) as p2 on p.project_id=p2. projectID order by p2.preference_count DESC limit 3;

```

	project_id	project_title	preference_count
▶	PR005	Forever Evil	4
	PR008	The Final Descendant	4
	PR009	Dark Heritage	4

Figure 6.7: Query Output7

**QUERY 8:** *List the names of students who have tried to cheat the system*

**Explanation:** Below is the query that could be used to identify the potential cheats in the system. Every student should express preference for a project only once, but some student might enter the same projects more than once in the preference matrix and

such students should be identified, before the project mapping runs. This query uses a self-join on student preference table, this query uses a lot of or conditions but the query can be optimised using PIVOT.

```
SELECT a.student_id,a.pref1,a.pref2, a.pref3,a.pref4,a.pref5,a.pref6,a.pref7,a.pref8,
      a.pref9,a.pref10,a.pref11,a.pref12,a.pref13,a.pref14,a.pref15,a.pref16,
      a.pref17,a.pref18,a.pref19,a.pref20
FROM   Student_Prefrences a, Student_Prefrences b
WHERE  a.student_id = b.student_id
AND    ((a.pref1 IN (b.pref2 , b.pref3, b.pref4, b.pref5, b.pref6, b.pref7, b.pref8, b.pref9, b.pref10,
b.pref11, b.pref12, b.pref13,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18,    b.pref19, b.pref20))
OR (a.pref2 IN (b.pref1 , b.pref3, b.pref4, b.pref5, b.pref6, b.pref7, b.pref8, b.pref9, b.pref10, b.pref11,
b.pref12, b.pref13,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18,b.pref19, b.pref20))
OR (a.pref3 IN (b.pref1 , b.pref2, b.pref4, b.pref5, b.pref6, b.pref7, b.pref8, b.pref9, b.pref10, b.pref11,
b.pref12, b.pref13,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
OR(a.pref4 IN (b.pref1 , b.pref2, b.pref3, b.pref5, b.pref6, b.pref7, b.pref8, b.pref9, b.pref10, b.pref11,
b.pref12, b.pref13,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
OR (a.pref5 IN (b.pref1 , b.pref2, b.pref3, b.pref4, b.pref6, b.pref7, b.pref8, b.pref9, b.pref10, b.pref11,
b.pref12, b.pref13,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
OR (a.pref6 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5, b.pref7, b.pref8, b.pref9, b.pref10, b.pref11,
b.pref12, b.pref13,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
OR (a.pref7 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref8, b.pref9, b.pref10, b.pref11,
b.pref12, b.pref13,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
OR (a.pref8 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref9, b.pref10, b.pref11,
b.pref12, b.pref13,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
OR (a.pref9 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref8, b.pref10, b.pref11,
b.pref12, b.pref13,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
OR (a.pref10 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref8,b.pref9, b.pref11,
b.pref12, b.pref13,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
OR (a.pref11 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref8,b.pref9, b.pref10,
b.pref12, b.pref13,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
OR (a.pref12 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref8,b.pref9, b.pref10,
b.pref11, b.pref13,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
OR (a.pref13 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref8,b.pref9, b.pref10,
b.pref11, b.pref12,b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
OR (a.pref14 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref8,b.pref9, b.pref10,
b.pref11, b.pref12,b.pref13, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
OR (a.pref15 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref8,b.pref9, b.pref10,
b.pref11, b.pref12,b.pref13, b.pref14, b.pref16, b.pref17, b.pref18, b.pref19, b.pref20))
```

OR (a.pref16 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref8,b.pref9, b.pref10, b.pref11, b.pref12,b.pref13, b.pref14, b.pref15, b.pref17, b.pref18, b.pref19, b.pref20))

OR (a.pref17 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref8,b.pref9, b.pref10, b.pref11, b.pref12,b.pref13, b.pref14, b.pref15, b.pref16, b.pref18, b.pref19, b.pref20))

OR (a.pref18 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref8,b.pref9, b.pref10, b.pref11, b.pref12,b.pref13, b.pref14, b.pref15, b.pref16, b.pref17, b.pref19, b.pref20))

OR (a.pref19 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref8,b.pref9, b.pref10, b.pref11, b.pref12,b.pref13, b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref20))

OR (a.pref20 IN (b.pref1 , b.pref2, b.pref3,b.pref4,b.pref5,b.pref6, b.pref7, b.pref8,b.pref9, b.pref10, b.pref11, b.pref12,b.pref13, b.pref14, b.pref15, b.pref16, b.pref17, b.pref18, b.pref19));

student_id	pref1	pref2	pref3	pref4	pref5	pref6	pref7	pref8	pref9	pref10	pref11	pref12	pref13	pref14	pref15	pref16
2002	PR005	PR005	PR005	PR005	PR005	PR005	PR005	PR005	PR005	PR005	PR005	PR005	PR005	PR005	PR005	PR005
2004	PR012	PR003	PR005	PR006	PR007	PR008	PR009	PR011	PR012	PR013	PR014	PR015	PR002	PR017	PR001	PR019
2009	PR008	PR024	PR023	PR022	PR021	PR020	PR019	PR012	PR017	PR002	PR015	PR014	PR013	PR005	PR011	PR010
2010	PR016	PR011	PR027	PR017	PR021	PR016	PR019	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 6.8: Query Output8

## 7. Conclusions and future work

Aim of this project was to design a database management system from scratch for an application that would be responsible for allocating the projects among the students in a manner that maximum global satisfaction is achieved. With the increasing admissions in Cthulhu School, this application and consistent database will provide an automated solution that would guide the project allocation rather than handling it manually.

Working on this project not only helped in learning the core concepts of SQL but also helped in attaining knowledge how databases systems are implemented in the real time projects. How we initiate a project by gathering the project requirements, building a logical model and lastly, proceeding to the final solution. From scratch, first database was created, tables and data were added to the database by conforming to the norms of normalisation up to BCNF, then constraints were added to the database using basic SQL structural and referential constraints. Also, imposed some integrity constraints using stored procedures and Triggers. **Thus, all the hard constraints as mentioned in the white paper were implemented on the database level.** Finally, a set of views and queries were written that could be useful for the University and students to view relevant information. Thus, achieved an understanding of how to build durable and consistent storage systems.

As a future work, we can add timestamp field in student preference table, this will help in breaking tie between students with same grades and same preference. This timestamp will be the time at which preference was recorded. There could also be addition of more streams as we currently have only two streams but it would be very easy to insert the data as we have already created a separate table Stream. So, only 1 table is to be updated. Currently, we have not placed any constraints on the projects that would be entered in the 20 fields of student\_preferences table, we can create a stored procedure, somewhat similar to validate\_projeject\_exists(but for all 20 columns), which would check if the preference entered exists in the project inventory or not. If present then only create entry for that record. We can also put constraints on DB or handle at application level that cheat records are not inserted into student preference table.

Thus, we can say that the project is completed with all the modules and constraints of the project allocation system successfully implemented, as per the requirements mentioned.

## **Acknowledgements**

I hereby acknowledge, all the work in this report is done by me, with the help of the information drawn from the references mentioned below.

## **References**

### ***Book***

<https://www.dartmouth.edu/~bknauff/dwebd/2004-02/DB-intro.pdf>

### ***Websites***

<https://beginnersbook.com/2015/05/normalization-in-dbms/>

<https://www.w3schools.in/dbms/data-schemas/>

<https://dev.mysql.com/doc/refman/8.0/en/stored-objects.html>

<https://www.tutorialspoint.com/sql/index.htm>

<https://hackr.io/blog/dbms-normalization>