

# **Arkham University Project Allocation System**

**Vishal Padma and 19200174**  
(vishal.padma@ucdconnect.ie)



## Table of Contents

Section 1: Introduction .....	4
Section 2: Database plan .....	4
Section 3: Database structure .....	10
Section 4: Database Views .....	17
Section 5: Procedural Elements .....	19
Section 6: Triggers .....	20
Section 7: Example Queries .....	26
Section 8: Conclusions .....	36

## List of Figures

Figure 1: Physical Table ER Diagram .....	9
Figure 2: ER Diagram with entities .....	10
Figure 3: Procedure call 'Insert_into_Individual_details' .....	27
Figure 4: Procedure call 'Insert_into_Projects_Allocated' .....	27
Figure 5: Procedure call 'Projects_Related_to_Student_ID' for 'CS' Stream ...	28
Figure 6: Procedure call 'Projects_Related_to_Student_ID' for 'DS' Stream ...	28
Figure 7: Procedure call 'Projects_Related_to_Student_ID' for 'CS/DS' Stream	29
Figure 8: View - Student Details .....	29
Figure 9: View - Supervisor Details .....	30
Figure 10: View – Remaining Projects .....	30
Figure 11: Trigger to check Student GPA is not greater than 4.2 .....	31
Figure 12: Trigger to check Student GPA is not less than 0 .....	31
Figure 13: Trigger to check Student Age if less than 17 .....	32
Figure 14: Trigger to check Student Age as Future Date .....	32
Figure 15: Trigger to check Supervisor Age less than 18 .....	32
Figure 16: Trigger to check Supervisor Age as Future Date .....	33
Figure 17: Trigger to check Student ID Starting with 'Stu' .....	33
Figure 18: Trigger to check Supervisor ID Starting with 'Super' .....	34
Figure 19: Trigger to check if 1st preference is null .....	34
Figure 20: Trigger to check new project allocated is not a duplicate ...	34
Figure 21: Trigger to check student is assigned one project only .....	35
Figure 22: Trigger to check if the student belongs to final year .....	35
Figure 23: Trigger to check if student stream is different than project stream	36

## **1. Introduction**

The main task in this project is designing a database system for an application which is used by the University for allocating projects on which the students will be working throughout their last year. There will be many projects which will belong to three streams CS Stream, DS Stream and CS/DS Stream. So each student can work on projects from the stream he is in. But the students who are in CS/DS stream can work on projects from all the streams. Here there will be a clash in the project allocation since one project can attract many students. There will be projects which are most popular and multiple students want to work on that project and there will be projects which won't be preferred that much and some projects might be just left out with no one's preference. So this task will be done by the application as it will be considering many factors on why the particular project should be submitted to the selected person. So for the application to do the desired task, it first needs to have some data at hand which is very the database system will come into the picture. Firstly, we are working on university data that means it's going to be huge as there will be many students as well as quite some professor teaching them. Although we will be working on allocating the projects to final year students, we will be having data of students from all the years. So we will have many tables in this database for the university but the main function for the database system is to provide all the data whenever asked by the application for processing the allocation task. The crucial part would be the data availability when requested for. And also the time taken by the system to process the requested query by the application. Database should be efficient enough to return the requested data in as less as possible time. The database should also avoid having duplicate entries for any table being used by the application.

## **2. Database Plan: A Schematic View**

There are seven different entities displayed in the diagram which are a part of the database design for project allocation system of the university. All the entities with their attributes are explained in detail in the section below.

First is the Stream entity which is used to details about different streams taught at the university. There are three different streams which are taught at the university and these are 'CS', 'DS' and 'CS+DS' and students can belong either one of them when they are admitted to the university. This entity has four attributes and they are described below:

- Stream ID is the primary key for this entity and it's an auto increment field.
- The next attribute is the Stream title which will store the title of the stream.
- Stream Code is the next attribute which will store the Code for each stream.
- Stream Description is the last attribute in this entity which will store a short description about the stream.

Next entity is the Individual which will store all the details about an individual associated with the university. This table will contain all the details regarding to the individual like name, DOB and other which are listed below with their description:

- Individual ID is the primary key in this case and each individual in the table will be having unique ID for themselves and its an alphanumeric value.
- Individual Fname is the one which will store the First name of the individual and is has a length of 50 which is fixed.
- Individual Lname is the one which will store the Last name of the individual and is has a length of 50 which is fixed.
- Individual Stream ID is the one which will store stream ID with which that individual is associated and it's an integer.
- 5. Gender is the attributes which will store the gender of the individual.
- 6. Individual Role is the attributes which will store the role as Student or as Supervisor in the university.
- 7. DOB is the last attribute for this entity and it will store the Date of Birth for the individual in the table.

Contact and Address is the next entity in the database design and this is will store all the details regarding to the contact and address details of all the individual's in the individual table. The attributes related to this table are mentioned below:

- Contact\_ID is the primary key for this entity and it's an auto increment field.

- Individual ID will store individual ID's for all the individuals from the individual table and it's an alphanumeric value.
- Mail\_ID is the next attribute in the list and it will store the Mail ID for the given individual from the individual table.
- Phone\_Number is the attribute which will store the Phone number for the individual.
- Address is the last attribute which will store the address of the Individual.

Student entity is the next one and this is an important entity which will be storing all the student details which will be used in the project allocation. The attributes associated with this table are listed below:

- Student ID is the primary key in this case and each student will have a unique ID and this field is alphanumeric.
- Semester\_Code is the next attribute which stores the numeric value for the semester in which the student is currently studying.
- Year Enrolled will store the year the student took admission in the university.
- Stream is the attribute which will store the Stream in which the student is associated.
- GPA is the attribute which records GPA of the student and this GPA is of the last semester the student was in.

The second last entity in this database design is the Project table which is again an important table in this design as it is used to in the project allocation system by the application. This entity stores all the project related details and the attributes related to them are listed below:

- Project ID is the primary key for this entity which will be used to uniquely identify the projects and it's an auto increment field.
- Project Topic is the next attribute which will store the Topic of the project and its length is fixed at 50 with datatype as varchar.
- Project Description attribute will store detailed description about the project which will be used by student so that they can know more about the project.

- Project Supervisor ID attribute is used to store the Supervisor for the project who will guide the students who will be assigned the project after the allocation.
- Project Stream ID is the attribute which will store the Stream ID of the project.
- Project Owner ID will store the ID of the individual who proposed the project in the first place and this can be supervisor as well as the students.

Preferences entity is the entity which will store all the preferences given by each student and this table will also be used while project allocations. The attributes related to this project are listed below:

- Student ID is the primary key attribute which will be used to uniquely identify the students.
- Preference\_1 to Preference\_20 are the next 20 attributes which will store the project ID's given by the student.

The last entity used in this database design is the Allocated which will store the details regarding which project is assigned to which student and the attributes in this entity are listed below:

- Student ID is the primary key attribute which will be used to uniquely identify the students.
- Project ID is the attribute which will store the Project ID which is allocated to the student by the application.
- Satisfaction Score attribute stores the value calculated taking into consideration multiple factor by the application while project allocation.
- Selected Preference attributes stores an integer value which indicated which project preference was selected for the given student.

Relation between all the different streams are explained below:

1. Stream and Individual entities are related to each other using 'belong' relationship as displayed in the ER Diagram. There can be many individuals who belong to the same stream. This is one-to-many type of relationship between the

Individual and stream where there are multiple people related to a single stream. This is a weak relation between these two entities.

2. Stream and Project entities are related to each other using 'has' relationship as displayed in the ER Diagram. This is a one-to-one relationship between these entities as one project can only belong to one stream. This is a weak relation between these two entities.

3. Individual and Contact/Address entities are related to each other using 'has' relationship as displayed in the ER Diagram. There can be multiple contacts for a single individual whose is associated with the university. This is one-to-many type of relationship between the Individual and Contact/Address where there is a single person having multiple Contact and address. This is a weak relation between these two entities.

4. Individual and Student entity are related to each other using 'ISA' relationship as displayed in the ER Diagram. This states that the individual can be a student.

5. Individual and Projects entities are related to each other using 'Supervises' relationship as displayed in the ER Diagram. This relationship states that the individual who are supervisor can supervise the projects in the project entity. There can be a single supervisor who can supervise more than one project and this is one-to-many type of relationship between the Individual and project and this is a weak relation between these two entities.

6. Individual and Projects entities are related to each other using 'owns' as another relationship as displayed in the ER Diagram. This relationship states that the individual can own projects in the project entity. There can be a single individual who can own more than one project and this is one-to-many type of relationship between the Individual and project and this is a weak relation between these two entities.

7. Students and Preferences entities are related to each other using 'Has' relationship as displayed in the ER Diagram. This relationship states that the Student can give more than one project as a preference since the limit to project preferences is 20. There can be a single individual who can give more than one preference and this is one-to-many type of relationship between the Student and preference and this is a weak relation between these two entities.



8. Student and Allocated entities are related to each other using 'assigned' relationship as displayed in the ER Diagram. This is a one-to-one relationship between these entities as one project can only be assigned to one student. This is a strong relation between these two entities.

9. Project and Allocated entities are related to each other using 'allocate' relationship as displayed in the ER Diagram. This is a one-to-one relationship between these entities as one project can only be assigned to one student. Here there is no full participation between the entities and is a weak relationship.

10. Preference and Project entities are related to each other using 'Choose' relationship as displayed in the ER Diagram. This is a one-to-one relationship between these entities as one project can only have one preference. Here there is no full participation between the entities and is a weak relationship.

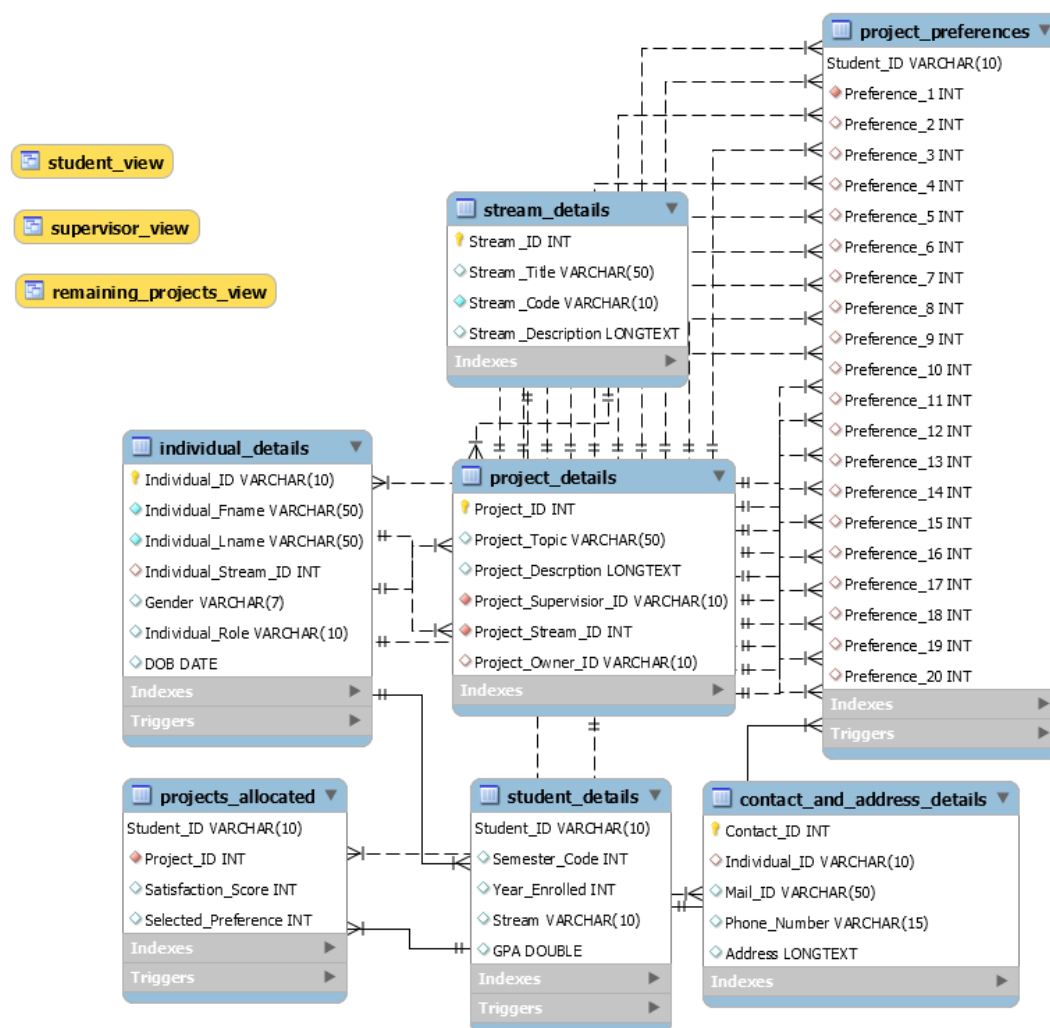


Figure 1: Physical Table ER Diagram

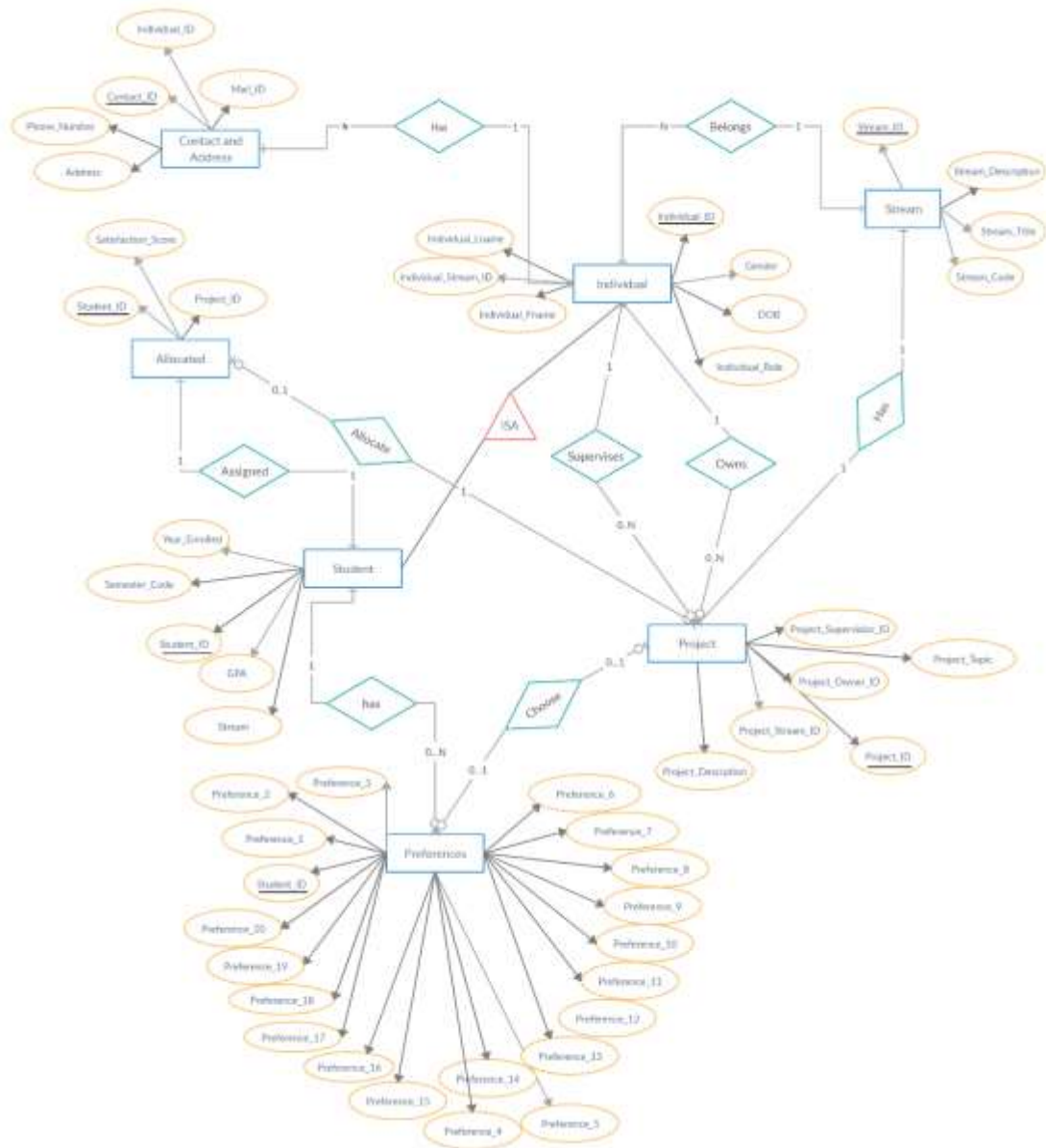


Figure 2: ER Diagram with entities

### 3. Database Structure: A Normalized View

These six different tables are listed below and below that are the description of each table.

- Streams\_Details
- Individual\_Details
- Contact and Address details
- Student\_Details
- Project\_Details
- Project\_Preferences

- Projects\_Allocated

Streams\_Details tables, the purpose of this table is to store the details about different streams available at the university. The attributes in this tables are Stream\_ID, Stream\_Title, Stream\_Code and Stream\_Description. These are the attributes which will be used by the application and also attributes can be added depending upon the requirement.

The purpose of the first attribute, Stream ID which has datatype integer and is an auto increment field where for each new entry ID is auto incremented. This serves two purpose, every Stream ID will be different and in order and secondly this is a field is primary key, and every cell will have a unique value. The next attribute is the title of the stream which has datatype of varchar(50), where its length is fixed to 50 maximum where the title of the stream will be stored. Next is the Stream Code with datatype varchar(10), which is again fixed at length 10 as Stream code is abbreviation to the title and it will be as small as possible. The next attribute is the stream description which has a datatype varchar(50) and this attribute is used to give some description about the stream and its length is fixed to 50 so that it doesn't take too long of description to avoid processing time while retrieving the data during query processing.

The stream table is a normalized view which properly follows all the forms that are 1NF, 2NF and 3NF. The attributes used in this table are the Stream ID which is the primary key, Stream Title, Stream code and Stream Description. As 1NF requires there is no redundant and repeating of groups this is handled by the primary key Stream\_ID where each stream can have only one ID. This table satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain <sup>[1]</sup>. 2NF requires 1NF along with no partial dependency between the columns and all the columns depend on the primary key. In stream table no columns are dependent on each other and they depend only on Stream\_ID. 3NF requires 1NF and 2NF and also all the columns should only depend on the primary key. Stream\_ID is used to retrieve all the data related to a stream, that means all the columns are dependent solely on Primary key. As all the columns are dependent on primary key or stream

id and no other column is related to each other, this also satisfies the BCNF form as well because there are no two columns which form a super key.

The next table is the Individual\_Details tables, the purpose of this tables is to store details about individuals associated with the university. The attributes in this table are Individual\_ID, Individual\_Fname varchar, Individual\_Lname varchar, Individual\_Stream\_ID, Gender, Individual\_Role, Date of Birth. If needed more attributes can be added depending upon the requirements.

The column Individual\_ID varchar(10) having datatype as varchar and this is the primary key in this table which stores ID for each individual's data in the table. The length is fixed at 10 as length of all the individual's will be the same. This column will be used to get data for different individuals' as no two rows will have the same ID's. Next is the Individual\_Fname varchar(50) column having datatype as varchar, which store the First name of the individual and this column cannot be null in the table. Same goes for the next column Individual\_Lname varchar(50) not null, having same datatype and length. These fields cannot be null as we can multiple common values in them and to avoid only having Individual\_ID with no name will be an inconvenient to handle such data or record where only Id is there with no name it in. Next column is the Individual\_Stream\_ID which tells us to which stream the individual belongs and its datatype is integer. This is a foreign key which refer to stream ID in stream details table to get the stream ID. Next Column is the Gender column with datatype as varchar and its length it set to 7. Next column is the Individual\_Role with datatype as varchar and its length is fixed at 10. The only two values expected in this column are 'Student' and 'Supervisor'. This feature can be expanded depending upon the requirements. The last column is the Date of Birth column with datatype as date and is used to record DOB of individuals. Here the columns Individual\_Stream\_ID, Gender, Individual\_Role, Date of Birth can have null values.

The individual table is also in a normalized view. The attributes of this table are Individual\_ID the primary key in this table which maintains the atomicity and no ID is repeated in the table, this satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain <sup>[1]</sup>. The other attributes like Individual\_Fname, Individual\_Lname, Individual\_Stream\_ID and remaining are not partial dependent

on each other which satisfies the requirement for 2NF. It also satisfies the 3NF form where 1NF and 2NF are already satisfied and all the remaining columns depend on the primary, where one can select all the other columns just by using the primary key. Since only individual ID is used to retrieve all the data related to an individual, this table satisfies BCNF form also as there are no two columns which can form a super key.

The Contact and Address Details table is used to store the Contact and the address of the individual's associated with the University. The attributes used in this table are Contact\_ID, Individual\_ID, Mail\_ID, Phone\_Number and Address. Upon extra requirements we can add multiple attributes which can be added later.

The first column in this table is Contact which has datatype as Integer and this column is set to auto increment, every time we insert a new row the Contact ID will be populated. This column is the Primary key so there won't be any problem in duplicate entries. The second column is the Individual\_ID with datatype Varchar with a fixed length of 10. This is basically the Individual ID which refers to the Individual\_Details tables. This ID tells us whom does the contact belong to. The next up is the Mail\_ID column which has varchar with length fixed at 50 and this column stores Mail ID for all the individual present in the individual table. Here there is a check which is used to check the format of the mail id. Next column is the Phone Number column with datatype as Varchar and the individual's Phone number is saved in this column. Last column is the Address with datatype as Long text as address can be long enough so can't specify a length to this column. Foreign key in this table is the Individual ID which refers the Individual Table for Individual ID.

Contact and address details table has five columns and the first column is the Contact ID which is an auto increment column and is the primary key which is used to retrieve the contact details for an individual member of the university and this satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain [1]. 2NF requires 1NF (already satisfied) and along with that it requires no partial dependency between the columns and there are no partial dependencies between the columns. 3NF requires 1NF and 2NF (Already satisfied) and along with that all the columns should be dependent on the primary key and in this table using the

individual ID we can select data from all the columns. BCNF is also satisfied in this as there are two columns dependent on each other the Contact ID and the Individual ID. Any given individual can have multiple contact details and these two will form a Super key. This satisfies the BCNF form.

The next table is about the student details, and it is used to store data about the students at the university. The attributes associated with this table are Student\_ID, Semester\_Code, Year\_Enrolled, Stream, GPA. More attributes can be added if there is any specification change.

Student\_ID is the first column and its datatype is varchar (10) and its length is fixed at 10 as all the ID will be having the same length. The next column is the Semester\_Code which has datatype as int and this is a field in which each student's semester code is stored. Here we have a check to store only numbers between 1 and 6 as there are only six semesters. Year\_Enrolled is the next column which has datatype of int where start year for each student is stored. Stream is the next column which is used to store the Stream data in it and there is a check to only include 'CS' and 'CS/DS' values in this field. It has a datatype of varchar with length fixed at 10. GPA is the column with datatype as double where we will store GPA of a student and this GPA will be of last semester completed by the student. Here there is a check associated to only include values between 0 to 4.2. This table refers to Individual\_Details tables for the details of the student using the Individual\_ID which is stored as Student\_ID.

Student details table with five different columns and the first column is the Student ID which is a primary key of this table and this will make sure that there are no duplicate entries in the table. As 1NF requires there is no redundant and repeating of groups this is handled by the primary key Student\_ID where each student can have only one ID. This table satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain [1]. Next is the 2NF form which requires 1NF and there should be no partial dependency on the other columns and all columns should be dependent on primary key only. As all the columns depend on primary key and no other column is related to each other it satisfies 2NF also. As for 3NF, 1NF and 2NF are required and all the other columns should also depend totally on the primary key column and no other column. To select students details we can

only use Student\_ID and no other detail from student details table, this also satisfies 3NF form. The BCNF form is also satisfied as there is no multivalued dependency.

Project\_Details table is the 4th table used in the database which is used to stored details about the projects which are to be allocated to the students. This tables contains Project\_ID, Project\_Topic, Project\_Description, Project\_Supervisor\_ID, Project\_Stream\_ID, Project\_Owner\_ID. More can be added to this table if needed.

Project\_ID is the primary key here and its datatype is int. This field is set to Auto Increment which will set new value for all the project during inserting the data into the table. Project\_Topic is the next column with the datatype varchar(50) where the Topic or domain of the project will be stored. Column Project\_Description is used to store a short description of the project and it has datatype as longtext. Next column is Project\_Supervisor\_ID which indicates the supervisor for the project and its datatype is varchar and this field cannot be null as each project should be associated with one supervisor. Next column is the Project\_Stream\_ID which has datatype as int and this field indicates the stream the particular project belongs to. Next column is the Project\_Owner\_ID with datatype as varchar and this field indicated the owner of the project where in most cases it is the supervisor, but some projects can be suggested by students as well. There are total three foreign key references in this table, Project\_Owner\_ID and Project\_Supervisor\_ID both refers to the Individual\_ID in the Individual\_Details table and Project\_Stream\_ID refers the Stream\_ID from the streams table.

Project details table has six different columns in it and the very first column is the Project\_ID which is the primary key of this table which maintains the atomicity and no duplicate ID are seen in the table, this satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain <sup>[1]</sup>. 2NF requires 1NF and it also needs the columns to have no partial dependency between them and it is not seen in this table which satisfies the 2NF form. 3NF form is the where we 1NF and 2NF forms are required along with one more requirement that all columns are solely dependent on primary key and in this table to select details about the project we need the project id, this satisfies the 3NF form. The BCNF form is also satisfied as there is no multivalued dependency.

Second last table used in this database is the Project\_Preferences table which is used to store preferences from each student. Each student is allowed to give 20 preferences for different projects and each student should provide at least one preference and maximum 20. Important thing is student cannot give multiple preferences for same project. The attributes of this table are Student\_ID, Preference\_1 to 20.

Student\_ID is the first column in this table which is also the primary key and datatype is varchar and this values is referred from the Student\_Details tables. The next Column is the Preference\_1 with datatype as int where Project\_ID will be stored and this field cannot be null as one preference is always required. The next 19 columns are same like the Preference\_1 column where 1 will be replaced with the respective column number and these can be null as some students can give not all 20 preferences. All the preferences column refers to the project table for the project ID.

The project preference table satisfies the 1NF form as there is a primary key column, Student ID which is used to get all the preferences given by each student and duplicate entries will not be allowed which satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain <sup>[1]</sup>. No columns in this tables are partially dependent on each other and this satisfies the 2NF form. The 3NF form is satisfied as all the columns in this table are totally dependent on the Student ID. BCNF form is also satisfied as there are no two columns which can make a super key in this table.

The last table in the database is the Projects\_Allocated which is used to store data about students and project allocated to it. The attributes in this tables are Student\_ID, Project\_ID and Satisfaction\_Score. We can include more attributes when the requirements are increased.

Student\_ID is the first column here with datatype as varchar and it is also the primary key here which stores the student ID. This Student\_ID is referred from the student details table. The next column is the Project\_ID with datatype int which indicates the project ID which is referred from the Project table. Last column in this Satisfaction\_Score which with datatype int and this column stores the value calculated by the application.



Project allocation table has four columns and the primary key in this table is the student ID which tells us that which student has been allocated which project and there can be no duplicate entries of same student as this column is the primary key. This satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain [1]. The other three columns are not partially dependent on each other so this satisfies the 2NF form and all the other columns are totally dependent on the primary key column student id and this satisfies the 3NF form. BCNF form is also satisfied as there are no two columns which can make a super key in this table.

#### **4. Database Views**

There are total of six views designed for this database which can be used by the application for project allocation. View names are mentioned below:

- Student\_View
- Supervisor\_View
- Remaining\_Projects\_View

The first view is a join of two tables, namely the student table and individual table. The attributes used in this view are Student\_ID, GPA, Semester\_Code, Stream (these are taken from the student table), Individual\_Fname (Individual First Name), Individual\_Lname (Individual Last Name), DOB (Date of Birth) (From Individual table), Mail\_ID and Phone Number (From Contact and Address table). The Student table is joined to the individual table by mapping the student ID as the individual ID. The individual table is joined to the Contact and address table by mapping the Individual ID of individual as the individual ID of contact and address table. After selecting these attributes, there is a condition in the where clause which only selects the Students with Semester Code as 5. The reason to do so is that the project allocation is only done to the final year students and it not done for students in the initial and mid years. This view can be used by the application to see the number of students present in the final year. Also the view has attributes like GPA which will be considered as a valuable part while calculating the satisfaction score for each student. The stream from the view can be used to check the stream of the student and allocated the desired stream

project. The other attributes like F\_name, L\_name, DOB and ID are pretty much self-explanatory in the view. The Mail ID and Phone number are taken from the Contact and address table and this will be used to communicate with the student. The case of project being allocated student will be notified on mail id or phone about the project allocation. These can also be used to communicate if there are some issues with the preferences and other issues related to the allocation.

The next view is of the supervisor with attributes taken from Individual table as well as the contact table. This view consists of Supervisor\_ID, Supervisor\_Fname (First Name), Supervisor\_Lname (Last Name), Gender, DOB, Stream (These taken from individual table), Mail\_ID and Phone number (These taken from contact and Address table). The individual table is joined to the Contact and address table by mapping the Individual ID of individual as the individual ID of contact and address table. So select all the supervisor present in the individual table, there is a where clause in which the Individual Role is selected as the Supervisor. By doing so, all the professor or supervisor of the university will be stored in the view. This view can be used to check the supervisor details for each project while allocating the projects. It can be also used to check how many supervisors belong to each stream. The mail id and Phone can be used to communicate with the supervisor in case a project is selected, and in turn the supervisor will start to notify the student about the next process. It can also be used in times when an issue is seen while allocating where supervisor needs to give input while deciding on project allocation.

The last view in this database design is related to the Projects remaining after the project allocation starts and some projects have been assigned to the students after initial project allocation process. This view contains attributes Project ID, Project Topic, Project Supervisor, Project Owner ID, Project Stream (All taken from the Project table), Supervisor Fname (First Name) and Supervisor Lname (Last name) (last two attributes taken from the individual table). The Project table is joined to the individual table by mapping the Project Supervisor ID as the individual ID. There is a where clause which is used to separate all the project ID's which are not present in the Projects Allocated table. This way we can have an idea about all the projects which are not yet assigned to any student. In a case where the student project preferences are not available, the applications need to assign a project to that student. To do so this view can be use. It can also we used by the

application to check if the newly project being assigned is not already assigned to any student.

## 5. Procedural Elements

There a total of three Procedures used in this database design and they are used for data insertion and also to check conditions while inserting the data into the table. The procedure names are mentioned below:

- Insert\_into\_Individual\_details
- Insert\_into\_Projects\_Allocated
- Projects\_Related\_to\_Student\_ID

The first procedure purpose is to insert the individual data coming from the application into two different tables. The data from this procedure will be inserted into the Individual\_Details and Contact\_And\_Address\_Details table. This procedure has parameters Individual\_ID VARCHAR (10), Individual\_Fname VARCHAR (50), Individual\_Lname VARCHAR (50), Individual\_Stream\_ID INT, Gender VARCHAR (10), Individual\_Role VARCHAR (10), DOB DATE, Mail\_ID VARCHAR (50), Phone\_Number VARCHAR (20), Address LONGTEXT. Parameters Individual\_ID, Individual\_Fname, Individual\_Lname, Individual\_Stream\_ID, Gender, Individual\_Role and DOB belong to the Individual details. These parameters will be inserted into the Individual\_Details tables once the insert statement block is executed in the procedure. The remaining parameters Mail\_ID, Phone\_Number and Address will be inserted into the Contact\_And\_Address\_Details tables once the insert statement block is executed in the procedure. Since the application will be recording individual details along with their contact and address details it will be a hectic and time consuming task to write multiple insert statements for both the tables. Using this procedure, we can insert data of the individual into two tables at the same time and will be save a lot of time as the data will be coming at different time interval which makes it hard to write insert statement every time a new individual data is inserted into the application.

The next procedure is used to insert into the Project allocated table which holds details about which project was allocated to which student. The data from this

procedure will be inserted into the Projects\_Allocatedn table which has four columns. The parameters used for this procedure are Student\_ID varchar (10), Project\_ID int, Satisfaction\_Score int, selected\_Preference int and all these parameters belong to the projects allocated table. Once the application decides to allocate a project to a particular student we can use this procedure and insert the data into the project allocated table. Using this procedure, we can avoid writing insert queries again and again for each time a new project is allocated to the student.

The next procedure is used with the purpose to fetch details about the projects when we give an input to the procedure and this input will be a student ID. The procedure name is Projects\_Related\_to\_Student\_ID where there is only one input parameter the student ID as ID. Here I have declared a variable as STR with datatype as varchar. Used a select statement where we will use ID as the input parameter from the procedure call to match the student ID in the student details table and take the stream and store the same in str variable declared initially. After this there is an if-else condition to check if the Stream belongs to CS, DS or CS/DS and after it matches it the student stream, then there is a select statement. The attributes associated selected in this select statement are Project ID, Project Topic, Project Supervisor, Project Owner ID, Project Stream (All taken from the Project table), Supervisor Fname (First Name) and Supervisor Lname (Last name) (last two attributes taken from the individual table). The Project table is joined to the individual table by mapping the Project Supervisor ID as the individual ID. There are three different where clause which are used to select all the project streams which are under CS, DS and CS/DS stream, depending upon the match of the students stream all the projects will be displayed accordingly. This procedure can be used to get details about the projects related to the student stream while allocating the projects for an individual student selected. It can also be used to see how many projects are assigned to the single supervisor.

## **6. Triggers**

There are multiple triggers used in this database design which are used to display a basic error message when the criteria are not met while data entry or while updating the data in the tables. These triggers are explained one by one below:

The first Trigger is about checking the preferences given by the student for the projects. This data will be stored in the project preferences table where each student can give up to 20 preferences. Each project will have exactly one preference in the table. The student should can give up to 20 preferences but he has to provide one preference for sure. So this trigger will be checking if the student has given at least one preference. SO in the case where the student doesn't give a preference, trigger will be invoked where it will display a message "Preference\_1 should be entered.". In this way we can avoid inconsistent data enter about student's project preferences in the table.

The same trigger logic can also be used when we are trying to update the table. So as mentioned the student can give one or twenty preferences, so in the case where the student needs to edit or update this preferences, we can use this trigger. While updating the table with new preferences we can add to delete preferences between 1 to 20 preferences. So in case where an update query is used to update the preferences and zero preferences are sent in the update query, in such case the message "Preference\_1 should be entered." will be displayed. This way we can avoid inconsistent data update about student's project preferences in the table.

The next trigger we will be the one to check the GPA of the students in the student table. So GPA can be between the range of 0 to 4.2 for all the students associated with the university. SO this value of the GPA cannot be less than 0 or greater than 4.2. In case where the GPA is less than 0 or greater than 4.2 data inconsistency will be seen. So this trigger is to check if GPA entered is in the range or not. While entering the data if the GPA is less than zero that is a negative value (-1), error message "Enter a Valid GPA for the student" will be displayed. At the same time if the value of the GPA is greater than 4.2 (4.3), error message "Enter a Valid GPA for the student" will be displayed. The reason to include the value zero is to use in the cases where the student is new to the university and not yet been graded or hasn't finished any semester.

The same trigger logic can also be used in the scenario where we are trying to update any student's GPA in the student table. So after every semester the GPA for the student will be changed or updated after the semester results are declared. In this case we will need to update the value of the GPA in the student table between the range of 0 to 4.2. While updating the data if the GPA is less than zero that is a

negative value (-1), error message "Enter a Valid GPA for the student" will be displayed. At the same time if the value of the GPA is greater than 4.2 (4.3), error message "Enter a Valid GPA for the student" will be displayed. The same message can also be displayed when the GPA entered in the first data enter is incorrect, again while updating the GPA value has to be between the range 0 to 4.2 or the message "Enter a Valid GPA for the student".

There is a Date of Birth field in the Individual Details table which is used to store the date of birth for an individual associated with the university. When inserting the data SQL doesn't have a default check for the Date of birth column which has a datatype of date. SQL insert statement will insert blindly any date given in the insert query and this date can future date also. This will create an inconsistency in the data where any individual with a DOB as future date will make no sense. This trigger will be triggered in two scenarios, as there are two different roles in this table namely Student and Supervisor. Student cannot have a future date as their DOB, when we give a date in the insert query to insert the data in the table, the trigger will first declare a new variable of age. Later we calculate the age of the student by converting the DOB into days and at the same time we also calculate the number of days with respect to the Current date. The difference is taken between the Number of days with respect to Current date with number of days with respect to the Date of Birth. Later this difference is divided by 365.25 and stored into the Age variable declared initially. Before storing the value into the age it is rounded off so that we have an integer. Then this Age is used to check if the Date of the birth of the student less than 17 with the Role as Student. If the age is less than 17 error message "Age Should be greater than 17" is displayed. The same age variable is used for the roles of the supervisor. When the age is less than 18 and role is supervisor, error message "Age Should be greater than 18" is displayed. With this we can check if the age for any student details being inserted is less than 17 the above mentioned error message will be displayed. Same for the supervisor role, we can check if the age for any supervisor details being inserted is less than 18 cases the above mentioned error message will be displayed. Same to check if the future date for DOB, when the age is a negative value error message "Future Date is not allowed in DOB" will be displayed and this will work for both the student as well as the supervisor.

This same logic can also be used while updating the Date of Birth of all the individual's present in the individual details table. The scenarios where we need to update the DOB of the Student or Supervisor, we need to check again if the age of the student or the supervisor being update is correct in order to avoid the data inconsistency in the individual details table. The same logic is used so; Age will be the declared variable used which will be calculated as mentioned in the trigger above. Then while updating, Age is used to check if the Date of the birth of the student less than 17 with the Role as Student. If the age is less than 17 error message "Age Should be greater than 17" is displayed. The same age variable is used for the roles of the supervisor while updating the DOB column. When the age is less than 18 and role is supervisor, error message "Age Should be greater than 18" is displayed. With this we can check if the age being updated for any student details being is less than 17 the above mentioned error message will be displayed. Same for the supervisor role, we can check if the age for any supervisor details being inserted is less than 18 the above mentioned error message will be displayed. Same to check if the future date which is being updated for DOB, when the age being updated is a negative value error message "Future Date is not allowed in DOB" will be displayed and this will work for both the student as well as the supervisor.

Another trigger is used to check if the Student ID is correct which means the student ID should always start with characters 'Stu' for all the students associated with the university. In this trigger we have a condition to check when the new individual ID is given while calling the procedure it will check if the student ID starts with characters 'Stu', if not than the error message "Student ID should start with Stu" is displayed. The same is done when you try to update the Student\_ID where if the student ID doesn't start with characters 'Stu' the error message "Student ID should start with Stu" is displayed. This specific requirement for to check for the student ID where it starts with 'Stu' is to avoid the confusion of whether the individual is student or supervisor. Once we know that the ID starts with 'Stu' you can directly know that this individual is a student and this is a good way to differentiate between individual details in the individual table.

The same trigger is used to check if the Supervisor ID is correct which means the Supervisor ID should always start with characters 'Super' for all the Supervisor

associated with the university. In this trigger we have a condition to check when the new individual ID is given while calling the procedure it will check if the Supervisor ID starts with characters 'Super', if not then the error message "Supervisor ID should start with Super" is displayed. The same is done when you try to update the Supervisor\_ID where if the Supervisor ID doesn't start with characters 'Super' the error message "Supervisor ID should start with Super" is displayed. This specific requirement for to check for the Supervisor ID where it starts with 'Super' is to avoid the confusion of whether the individual is student or supervisor. Once we know that the ID starts with 'Super' you can directly know that this individual is a Supervisor and this is a good way to differentiate between individual details in the individual table.

The next trigger is used on the project allocated table where we store the data about projects allocated to the different students. As per the requirement, when a project is allocated to the student that project is no longer available for further allocation to any other student. This trigger will check if any project is being allocated which is already there in the project allocation table. Here there is a condition where it will check if the new project ID along with the student ID being passed is already present in the project allocation table and if the project ID is already present then there is an error message "Project is already allocated" is displayed. This way we can make sure that no same project is allocated to the two different students.

The same trigger above is used while updating the project ID on the project allocated table where we store the data about projects allocated to the different students. As per the requirement, when a project is allocated to the student that project is no longer available for further allocation to any other student. This trigger will check while updating the project ID if any project is being allocated which is already there in the project allocation table. Here there is a condition where it will check if the new project ID along with the student ID being updated is already present in the project allocation table and if the project ID is already present then there is an error message "Project is already allocated" is displayed. This way we can make sure that no same project is allocated to the two different students.



The next trigger uses above same logic is used on the project allocated table where we store the data about projects allocated to the different students. As per the requirement, when a project is allocated to the student that student should no longer need further allocation to any other project. This trigger will check if any student is being allocated to another, where the student is already allocated a project which is there in the project allocation table. Here there is a condition where it will check if the new student ID along with the project ID being passed is already present in the project allocation table and if the student ID is already present than there is an error message "Student has a project already allocated" is displayed. This way we can make sure that no same student is allocated to the two different projects.

The next trigger uses above same logic is used on the project allocated table where we store the data about projects allocated to the different students. As per the requirement, when a project is allocated to the student that student should no longer need further allocation to any other project. This trigger will be checked while updating if any student is being allocated to another, where the student is already allocated a project which is there in the project allocation table. Here there is a condition where it will check if the student ID being updated along with the project ID being passed is already present in the project allocation table and if the student ID is already present than there is an error message "Student has a project already allocated" is displayed. This way we can make sure that no same student is allocated to the two different projects while updating the student ID in the project allocation table.

There is a trigger to check the requirement which states that the Student should be allocated a project which belongs to their stream and no other stream projects will be allocated to them. When the call is made from the procedure to insert the into project allocation table, this trigger will store the stream id into a temporary variable declared at the start and will store the project stream id into another temporary variable declared at the start. Once these values are stored, there is an if condition which checks for two things, first it checks if the stream of student and project are different and along with that it will check if the student stream is not equal to 'CS/DS', if these two conditions are met then it will display an error message "Student ID stream doesn't match with project stream.". The second

condition where it checks the Stream which doesn't belong to CS/DS is because as per requirement CS/DS students can do projects from all the streams and hence this trigger will not work if the student stream belongs to CS/DS. This same trigger will work if we are updating the student and project ID.

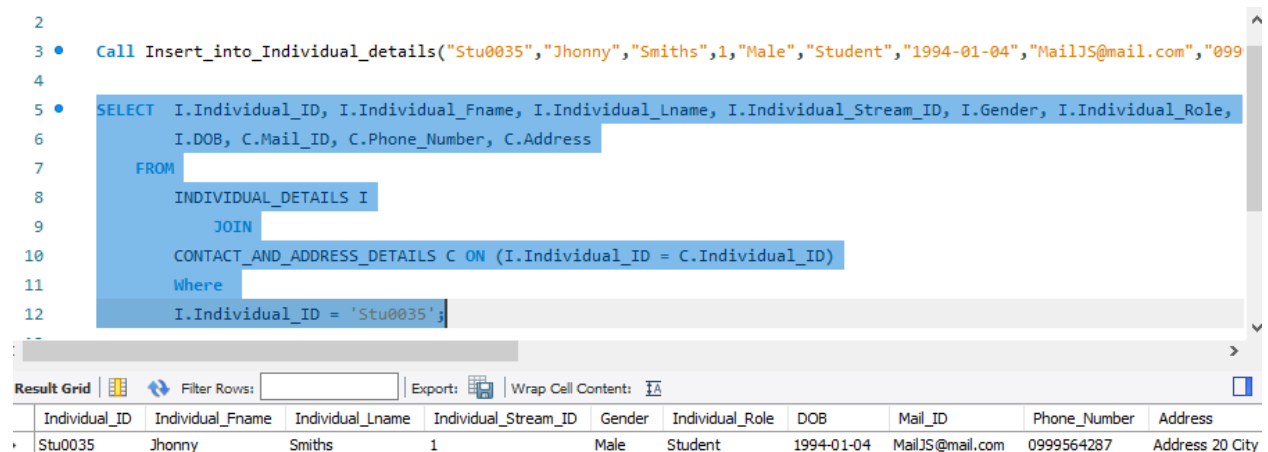
The last trigger for this project allocated table is to check if the student ID being stored here is a valid student ID or not. As per the requirement we need to allocate projects to the final year students and we cannot allocate to any student who is not in final year. In this case we can check if the student id being sent in the call procedure to insert data into the project allocated table is actually present in the student view created. If the ID is present we can insert the data into the table or else error message "Student ID doesn't exist in Student View" will be displayed. This message will also be displayed when updating the project allocation table with student ID which doesn't belong to Student view.

## **7. Example Queries: Your Database In Action**

Your database will provide a structure for the data in the application, and means of accessing and viewing that data. In this section show us the database in action, by providing sample queries and their outputs (please do not provide large data sets as outputs; edit and summarize as appropriate). Provide specific queries to test on your database, and tell us what those queries provide to the application. Use the database you have defined as the basis for your queries and their results. If your query makes reference to specific tables that were not defined in class or in a practical assignment, then provide example rows of this table in section 3.

You may use screenshots here but do not overfill your report with screenshots. Ensure that there is a cohesive argument expressed in the text of the report and that it is not simply a bag of diagrams and queries and screenshots. When you include images, make sure they are readable and actually add to the discussion. Below are some sample queries which will show how the different procedures, view and trigger work and display the data required by the application from the database designed to support the project allocation for students of the university.

Firstly, we will see how the Procedure 'Insert\_into\_Individual\_details' works. As seen in the image below, we call the procedure and pass all the required data and execute the same later. Once the query is executed the data is inserted into two tables as mentioned in the procedure details. To check if the data was properly inserted into the requested tables, I have written a select query which joins two tables and displays the data inserted using the procedure 'Insert\_into\_Individual\_details'.



```

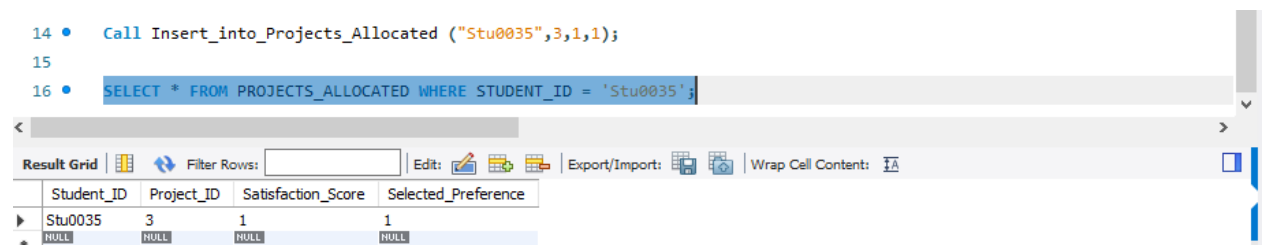
2
3 • Call Insert_into_Individual_details("Stu0035","Jhonny","Smiths",1,"Male","Student","1994-01-04","MailJS@mail.com","099
4
5 • SELECT I.Individual_ID, I.Individual_Fname, I.Individual_Lname, I.Individual_Stream_ID, I.Gender, I.Individual_Role,
6       I.DOB, C.Mail_ID, C.Phone_Number, C.Address
7     FROM
8       INDIVIDUAL_DETAILS I
9     JOIN
10    CONTACT_AND_ADDRESS_DETAILS C ON (I.Individual_ID = C.Individual_ID)
11    Where
12    I.Individual_ID = 'Stu0035';

```

Individual_ID	Individual_Fname	Individual_Lname	Individual_Stream_ID	Gender	Individual_Role	DOB	Mail_ID	Phone_Number	Address
Stu0035	Jhonny	Smiths	1	Male	Student	1994-01-04	MailJS@mail.com	0999564287	Address 20 City

Figure 3: Procedure call 'Insert\_into\_Individual\_details'

Second procedure is 'Insert\_into\_Projects\_Allocated' which is used to insert the data in the projects allocated table once the application decides on allocating a project to a particular student. We pass four parameters to this call which are student id, project id, satisfaction score and selected preference. Once we pass these parameters and execute the query, it inserts the data into projects allocated table as shown in the image below.



```

14 • Call Insert_into_Projects_Allocated ("Stu0035",3,1,1);
15
16 • SELECT * FROM PROJECTS_ALLOCATED WHERE STUDENT_ID = 'Stu0035';

```

Student_ID	Project_ID	Satisfaction_Score	Selected_Preference
Stu0035	3	1	1
NULL	NULL	NULL	NULL

Figure 4: Procedure call 'Insert\_into\_Projects\_Allocated'

The next procedure name is 'Projects\_Related\_to\_Student\_ID' and it is related to retrieving the projects which are related to the student ID or which a particular student can choose for giving preference so that we can do the project in his final year. The parameters passed to this procedure is just the student ID and we will get all the projects related to the student. For example, in the below image, all the projects related to CS are retrieved as the student ID entered belongs to a student whose stream is 'CS'.

18 • `CALL Projects_Related_to_Student_ID ('Stu0035');`

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [IA](#)

Project_ID	Project_Topic	Project_Supervisor_ID	Project_Owner_ID	Project_Stream_ID	Supervisor_Fname	Supervisor_Lname
1	CS Culture1	Super001	Super001	1	Jaden	Jones
2	CS Culture2	Super002	Super002	1	Mathew	Smith
3	CS Culture3	Super003	Super003	1	Katie	Chris
4	CS Culture4	Super005	Stu005	1	Ted	Math
5	CS Culture5	Super001	Super001	1	Jaden	Jones

Figure 5: Procedure call 'Projects\_Related\_to\_Student\_ID' for 'CS' Stream

The next image is where the procedure call when executed for a particular student who belongs to 'DS' stream is entered as a parameter in the call, all the projects related to the 'DS' Stream are retrieved as seen in the image below.

20 • `CALL Projects_Related_to_Student_ID ('Stu0012');`

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [IA](#)

Project_ID	Project_Topic	Project_Supervisor_ID	Project_Owner_ID	Project_Stream_ID	Supervisor_Fname	Supervisor_Lname
12	DS Dagon Culture1	Super004	Super004	2	Hen	James
13	DS Dagon Culture2	Super005	Super005	2	Ted	Math
14	DS Dagon Culture3	Super006	Super006	2	Karen	Smith
15	DS Dagon Culture4	Super002	Stu0012	2	Mathew	Smith
16	DS Dagon Culture5	Super004	Super004	2	Hen	James

Figure 6: Procedure call 'Projects\_Related\_to\_Student\_ID' for 'DS' Stream

The next image is where the procedure call when executed for a particular student who belongs to 'CS/DS' stream is entered as a parameter in the call, all the projects related to the 'CS/DS' Stream are retrieved as seen in the image below. Also, as the students of 'CS/DS' stream are allowed to do projects from all the streams, the projects from all the streams will be retrieved in this call.

22 • `CALL Projects_Related_to_Student_ID ('Stu0027');`

Project_ID	Project_Topic	Project_Supervisor_ID	Project_Owner_ID	Project_Stream_ID	Supervisor_Fname	Supervisor_Lname
21	DS Dagon Culture10	Super009	Super009	2	Nithin	Goud
22	DS Dagon Culture11	Super003	Stu0012	2	Katie	Chris
23	CS/DS Culture1	Super007	Super007	3	Ashok	Sharma
24	CS/DS Culture2	Super008	Super008	3	Rohit	Rathod
25	CS/DS Culture3	Super009	Super009	3	Nithin	Goud

Figure 7: Procedure call 'Projects\_Related\_to\_Student\_ID' for 'CS/DS' Stream

The way this above procedure call can be used is if the application needs to allocate a student who has not been allocated any project, this procedure can help in retrieve the projects which can be used later for allocation by the application system.

There are some views created in the database as mentioned above and all of them are displayed in the images below. The first view is the student view which I want to describe, as the name suggests this view stores all the details about the student. This view has three tables data joined together to get the student details. The tables joined in this view are the student details, individual details and contact and address details. The reason to join these tables is because student details are present in three different tables and to get them together we need to write a query which joins all the three tables. Writing joint queries again and again while project allocation can be time consuming. This way by making a view we can get a hold of all these data with a simple select statement as seen in the image below.

24 • `SELECT * FROM STUDENT_VIEW;`

Student_ID	GPA	Semester_Code	Stream	Student_Fname	Student_Lname	Gender	DOB	Mail_ID	Phone_Number
Stu0010	3	5	CS	Julie	Will	Female	1995-02-06	Mail11@mail.com	0982345678
Stu0012	3.1	5	DS	Sunny	Sharma	Male	1993-04-07	Mail13@mail.com	0987645678
Stu0015	3.9	5	DS	Ruhi	Pandit	Female	1996-08-01	Mail16@mail.com	0987654378
Stu0017	3.2	5	DS	Ron	Smith	Male	1997-01-09	Mail18@mail.com	0987654321
Stu0018	4	5	DS	Bob	Chris	Male	2000-05-05	Mail19@mail.com	0687654321

STUDENT\_VIEW 18 x Read Only

Figure 8: View - Student Details

The second view is the supervisor view which I want to describe, as the name suggests this view stores all the details about the supervisor. This view has two tables data joined together to get the supervisor details. The tables joined in this view are the individual details and contact and address details. The reason to

join these tables is because supervisor details are present in two different tables and to get them together we need to write a query which joins the two tables. Writing joint queries again and again while project allocation can be time consuming. This way by making a view we can get a hold of all these data with a simple select statement as seen in the image below.

26 • `SELECT * FROM SUPERVISOR_VIEW;`

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

Supervisor_ID	Supervisor_Fname	Supervisor_Lname	Gender	DOB	Stream	Mail_ID	Phone_Number
Super001	Jaden	Jones	Male	1984-01-04	1	Prof1@mail.com	0707334339
Super0010	Nithin	Singh	Male	2022-04-01	3	Prof10@mail.com	0407131380
Super0011	Nithin	Rathod	Male	2023-04-01	3	Prof11@mail.com	0807131380
Super002	Mathew	Smith	Male	1983-01-07	1	Prof2@mail.com	0607134339
Super003	Katie	Chris	Female	1984-05-05	1	Prof3@mail.com	0607134389

Figure 9: View - Supervisor Details

The view which was created in this database design is the Remaining project view. This view is all about returning the projects which are not yet allocated to any other student. In general, this is a subquery which takes a lot time to write and process and a person cannot keep checking by writing query after assigning projects to students. The main aim behind this view is to make the projects which are not yet allocated available at instant and this view makes it easy as all you need is to just write a select statement and you have a list of all the projects remaining which are yet to be allocated.

28 • `SELECT * FROM REMAINING_PROJECTS_VIEW;`

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

Project_ID	Project_Topic	Project_Supervisor_ID	Project_Owner_ID	Project_Stream_ID	Supervisor_Fname	Supervisor_Lname
6	CS Culture6	Super002	Super002	1	Mathew	Smith
7	CS Culture7	Super003	Super003	1	Katie	Chris
9	CS Culture9	Super006	Super006	1	Karen	Smith
13	DS Dagon Culture2	Super005	Super005	2	Ted	Math

Figure 10: View – Remaining Projects

The next section will explain about all the different triggers which are used in this database design. As mentioned in the paper, we are expecting to hold of the details regarding the student, supervisor, projects and some other data, but just to maintain the data integrity in the database I have written some triggers to check the details of the student as well as the supervisor. To start with the

triggers, the trigger is on the student details table where I am checking if the GPA entered is correct or not. GPA is the marks obtained at the end of each semester to every individual student and this has a range of 0 to 4.2. The trigger is triggered when there is a GPA entered in the GPA column while data enter into the student table which is less than 0 or greater than 4.2. For example, in the below image when we are trying to enter the GPA has 5 which is more than, trigger is triggered and the error message is displayed as seen in the image.

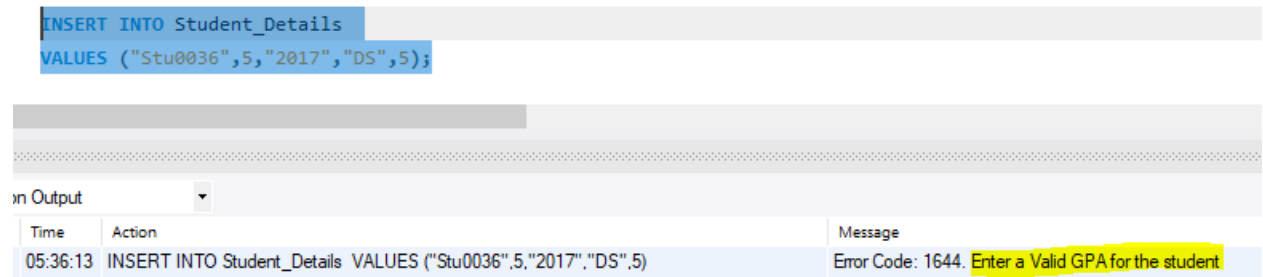


Figure 11: Trigger to check Student GPA is not greater than 4.2

When the value of the GPA values is less than zero the same error message is displayed as seen in the image below.



Figure 12: Trigger to check Student GPA is not less than 0

The next couple of triggers are written for the individual details table where we are checking if the data for all the individual is in good manner to main the data integrity and to make sure that when fetched the data makes sense. The first check is for the age of the student who is being admitted to the university where he needs to be of age 17, any student entered whose age is below 17 will be thrown an error message. In general, any student joining university will be of age 17 and this can be changed depending upon the requirement of the project. For example, as seen in the image below, when the DOB was entered as 2005, there was an error message displayed.

30 • `Call Insert_into_Individual_details("Stu0036","Jade","Smiths",2,"Male","Student","2005-02-04","MailJJS@mail.com","05`  
31

#	Time	Action	Message
46	04:38:30	SELECT * FROM INDIVIDUAL_DETAILS WHERE INDIVIDUAL_ID = 'Stu0036' LIMIT 0, 10...	0 row(s) returned
47	04:43:43	Call Insert_into_Individual_details("Stu0036","Jade","Smiths",2,"Male","Student","2005-02-...	Error Code: 1644: Age Should be greater than 17

Figure 13: Trigger to check Student Age if less than 17

This trigger can also be used when we get date of birth as a future date, where we will check if the date being entered is a future date, in this case the age calculated will be a negative value and if the age is negative value, below error message is displayed as shown.

• `Call Insert_into_Individual_details("Stu0036","Jade","Smiths",2,"Male","Student","2021-02-04","MailJJS@mail.com","05`

#	Time	Action	Message
66	05:03:11	Call Insert_into_Individual_details("Stu0036","Jade","Smiths",2,"Male","Student","2021-02-...	Error Code: 1644: Future Date is not allowed in DOB

Figure 14: Trigger to check Student Age as Future Date

The next trigger is to check is for the age of the supervisor who is being admitted to the university where he needs to be of age 18, any supervisor entered whose age is below 18 will be thrown an error message. In general, any supervisor joining university will be of age 18 and this can be changed depending upon the requirement of the project. For example, as seen in the image below, when the DOB was entered as 2004, there was an error message displayed.

3 • `Call Insert_into_Individual_details("Super0036","Jade","Smiths",2,"Male","Supervisor","2004-02-04","MailJJS@mail.com",`  
1

#	Time	Action	Message
67	05:03:40	Apply changes to individual_details	Changes applied
68	05:13:29	Call Insert_into_Individual_details("Super0036","Jade","Smiths",2,"Male","Supervisor","200...	Error Code: 1644: Age Should be greater than 18

Figure 15: Trigger to check Supervisor Age less than 18



This trigger can also be used when we get date of birth as a future date, where we will check if the date being entered is a future date, in this case the age calculated will be a negative value and if the age is negative value, below error message is displayed as shown.

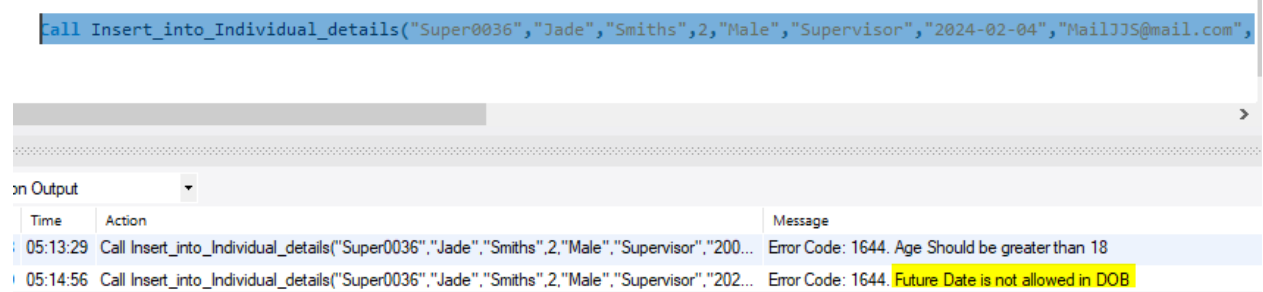


Figure 16: Trigger to check Supervisor Age as Future Date

The next trigger is to check if the individual ID for the Student start with specific letter which are 'Stu' and the reason to implement this is to avoid the confusion where individuals will have same 10-digit numeric ID and it would be hard to distinguish where selected individual is a student or a supervisor. So this trigger is triggered if we give Student ID which doesn't start with specific characters 'Stu'. Error message will be displayed as seen in the image below.

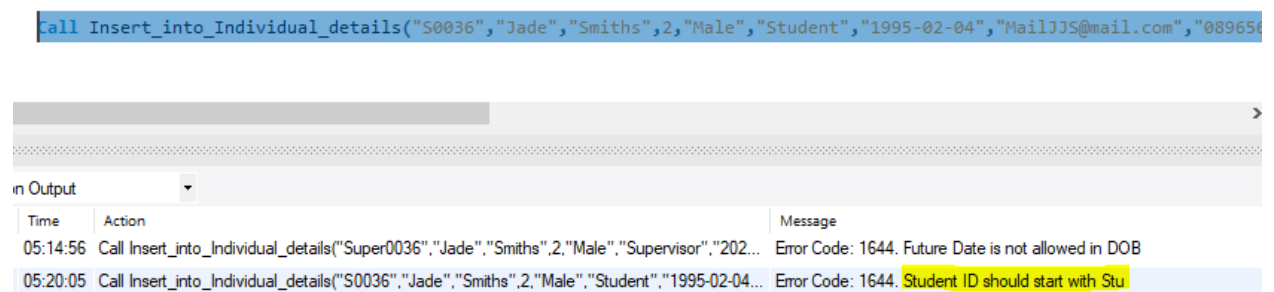


Figure 17: Trigger to check Student ID Starting with 'Stu'

The same trigger can be used to check if the individual ID for the Supervisor start with specific letter which are 'Super'. This trigger is triggered if we give Supervisor ID which doesn't start with specific characters 'Super'. Error message will be displayed as seen in the image below.

```
Call Insert_into_Individual_details("S0036","Jade","Smiths",2,"Male","Supervisor","1995-02-04","MailJJ5@mail.com","085
```

Time	Action	Message
15:20:05	Call Insert_into_Individual_details("S0036","Jade","Smiths",2,"Male","Student","1995-02-04...	Error Code: 1644. Student ID should start with Stu
15:23:16	Call Insert_into_Individual_details("S0036","Jade","Smiths",2,"Male","Supervisor","1995-02...	Error Code: 1644. Supervisor ID should start with Super

Figure 18: Trigger to check Supervisor ID Starting with 'Super'

The next trigger is set on the project preference table where we will be checking if any student has given zero preferences, because as per the requirement each student needs to give one preference and it is mandatory for all the final year students. The trigger will check if only student ID is sent or it will check if the 1<sup>st</sup> preference is null, error message shown in the below image will be displayed.

```
INSERT INTO Project_Preferences (Student_ID)
Values ("Stu0036");
```

#	Time	Action	Message
90	05:39:19	INSERT INTO Project_Preferences (Student_ID) Values ("Stu0036")	Error Code: 1644. Preference_1 should be entered

Figure 19: Trigger to check if 1<sup>st</sup> preference is null

The next trigger is on the project allocated table where we will be checking if the project allocated to each student is not repeated, this way we can make sure that one project is only allocated to one student. Error message is displayed as shown in the image below.

```
Call Insert_into_Projects_Allocated ("Stu0036",20,8,6);
```

Time	Action	Message
05:51:28	Select * from Projects_allocated LIMIT 0, 1000	18 row(s) returned
05:53:12	Call Insert_into_Projects_Allocated ("Stu0036",20,8,6)	Error Code: 1644. Project is already allocated

Figure 20: Trigger to check new project allocated is not a duplicate

The same table has another trigger to check if the same student is being assigned to different project, one of the requirement of the project is to allocate one

student one project and if this is not met there is an error message displayed which is shown in the image below.

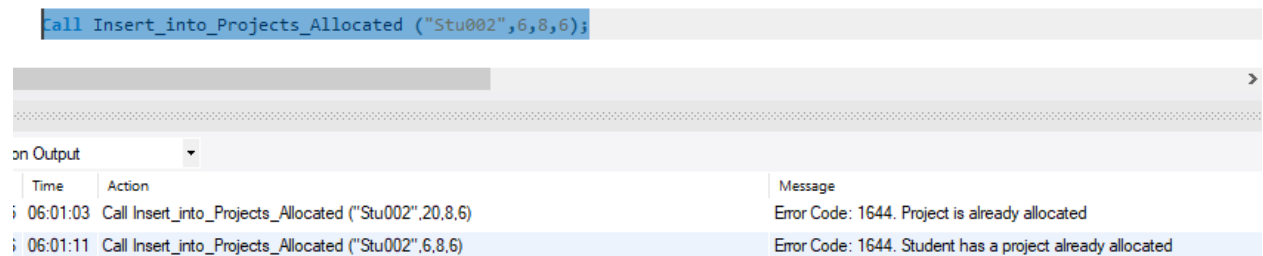


Figure 21: Trigger to check student is assigned one project only

The last trigger on this table is to check if the student being allocated the project is present in the student details table. There can be cases when the student ID which is not even present in the student view which was created to store all the details about the Final year students, in such case this trigger will be triggered and error message will be displayed as seen in the image below.

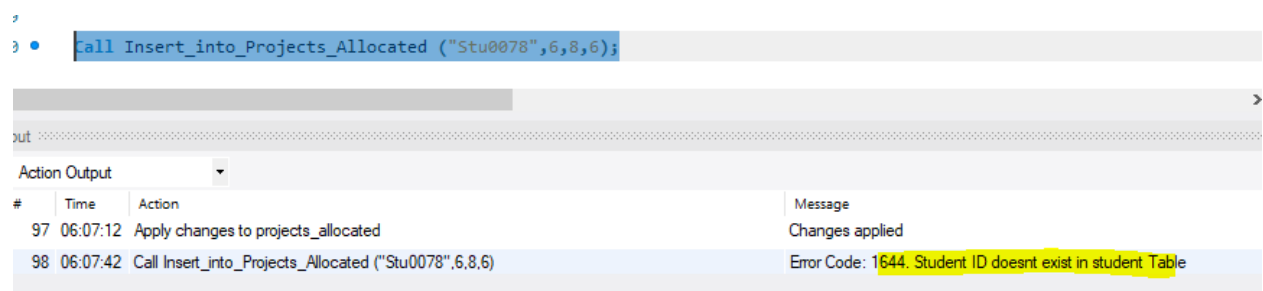


Figure 22: Trigger to check if the student belongs to final year

This trigger which is associated with the project allocation table is to check the requirement which states that, Student stream should match to the project Stream. This requirement is true for two streams that is 'CS' and 'DS' but this is not true for the 'CS/DS' stream. As CS stream students should only be assigned the CS projects and DS Stream student should only be assigned projects with DS stream, on the other hand 'CS/DS' stream students can take project from all the three streams that is CS, DS and CS/DS. This trigger will be triggered if CS student is being assigned a project with stream other than CS and it will trigger if DS student is being assigned a project with stream other than DS, below shown error message will be displayed as seen in the image. If the student belongs to CS/DS project, he can be assigned projects from all three streams and no error will be displayed.

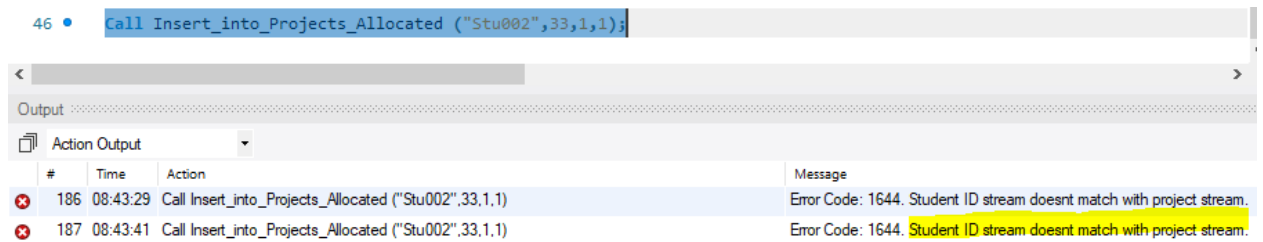


Figure 23: Trigger to check if the student stream is different than project stream

All the same triggers are set to work while updating all the above mentioned tables. Sam triggers will be used to check the above conditions when we are updating specific details in the specific tables.

## 8. Conclusions

The above database design is for the project allocation for the final year students and it consist of many different schemas which will be used by the application for project allocation. The design is made simple and easy where we make makes changes in the schemas and scale as the requirement changes. There are a lot of changes that can be made for example we can accommodate more than three streams in the stream table if any new stream comes up. This won't even require a new table just an entry in the stream table will be enough. Another change which can be made is the GPA column which is included in the student details table, we can make a separate table which will show distributed marks of each student and this can be done easily by adding a new table and mapping it to the respected tables. This way the application can consider the marks in each subject for calculating the satisfaction score and allocate the projects accordingly. Lastly, we can also make a separate table which will can store only staff details where we can show that one staff supervisor can teach two subjects and this multiple mapping can be handled between the staff table and stream. This is database design is decent enough for the project requirements given and it is also flexible enough to add new schemas to the design.

## **Acknowledgements**

This RDBMS project was not a simple task which can be done with hours, it required a lot of time and efforts at the same time. Initially it seemed difficult with the vague requirements but later on after reading carefully and taking notes while the online Q&A sessions many doubts were cleared and I would like to thank my prof. Tony and all the teaching assistance who helped me from the day one and made me capable of doing a project like this. It was their guidance and help which was important made it possible for me to get this project done in the stipulated time. All the online notes and others study material provided was of great help. I would like to acknowledge that the work done in this project is solely mine and each and every word and line written in this project is mine and one referred is cited inline. Thank you everyone who have helped me during the project.

## **References**

1. Boyce-Codd Normal Form (BCNF) of Database Normalization | Studytonight, 2020. <https://www.studytonight.com/dbms/boyce-codd-normal-form.php> [Accessed 30 April 2020].