

School of CS – Project allocation System

Tejas Karimanne Indushekar



Table of Contents

Section 1: Introduction	1
Section 2: Database Plan: A Schematic View	2
Section 3: Database Structure: A Normalized View	3
Section 4: Database Views	4
Section 5: Procedural Elements	5
Section 6: Example Queries	6
Section 7: Conclusions	7
Section 8: Acknowledgements	5
Section 9: References	6

List of Figures

Figure 1: ER-Diagram	3
Figure 2: User table	4
Figure 3: Login table	5
Figure 4: Contact table	6
Figure 5: Stream table	7
Figure 6: Student table	8
Figure 7: Project table	9
Figure 8: Preference table	10
Figure 9: Project Allocation table	11
Figure 10: Physical model	13
Figure 11: Staff view output	20
Figure 12: Student view output	21
Figure 13: Stream 1 view output	22
Figure 14: Stream 2 view output	22
Figure 15: Procedure output	24
Figure 16: User table trigger output	24
Figure 17: Login table trigger output	25
Figure 18: Project table trigger output	26
Figure 19: Contact table trigger output	26
Figure 20: Student table trigger output	27
Figure 21: Preference table trigger output	28
Figure 22: Project allocation table trigger output	29
Figure 23: Query 1 output	30
Figure 24: Query 2 output	30

Figure 25: Query 3 output	31
Figure 26: Query 4 output	31
Figure 27: Query 5 output	32

1. Introduction

The dean of the school of Cthulhu Studies (CS) at Miskatonic university is responsible for allocating projects to the final year students. Until now projects were allocated to interested students on an adhoc basis, but recently there has been a surge in the number of students in the course and hence it requires a robust application to handle the allocation system. These projects are undertaken by students in the last two semesters of their final year. Students and staff members belong to one of the two streams below:

- Cthulhu Studies(CS only)
- Cthulhu Studies with a specialization in Dagon Studies (CS+DS).

Prominently projects are proposed by staff members which generally reflects their own research. Staff generally tend to propose projects which belongs to the stream they are associated with, although they can propose projects that are suited to both the streams. In addition to this student can also propose their own projects and are generally directly assigned to them. Rest of the projects proposed by staff members are assigned based on the student's interest.

Domain of the project and the application:

The purpose of the application is to simulate an environment to ease the adhoc manual allocation of projects and automate the entire process of project allocations using the algorithms. Two approaches application makes use of are:

- Simulated Annealing
- Genetic Algorithms

The above methods help us in calculating the local and global satisfaction score which is the core process in project allocation. The aim of the application is to assign projects to students with maximum local and global satisfaction scores with taking into consideration few of the constraints.

Below are few hard constraints:

- Each student should be assigned to one of the projects preferred by them

- Each project can be assigned to only one student
- Each student is assigned a project of their stream

Below are few soft constraints:

- Chances of getting a preferred project increases with a higher GPA
- Projects are equally distributed among the supervisors.

Students can give up to 20 preference of projects with a compulsion of at least one. With the increase in popularity of the course, it has been difficult to allot projects manually and a robust application is required to facilitate the process.

Role of the database system:

Role of the database is to facilitate an application that will handle the entire process of project allocation. Thus, the database will facilitate the backend of the application which includes storing all the necessary data. It includes students profile data like name, gpa, semester details, contact information and other information with respect to staff details, project and student preferences.

2. Database Plan: A Schematic View

The schematic view of the database plays an important role in defining a high-level model of the database which is represented visually. It Consists of three main features:

- Entities: It can be described as a real-world object which is equivalent to tables in a database. Represented as a rectangle in the schema.
- Attributes: It is a characteristic/feature of an entity. Represented as an oval in the schema.
- Relationship: It describes the relationship/interaction between the entities. Represented as a diamond shape in the schema.

The below diagram represents the schematic view of the database in support of the application.

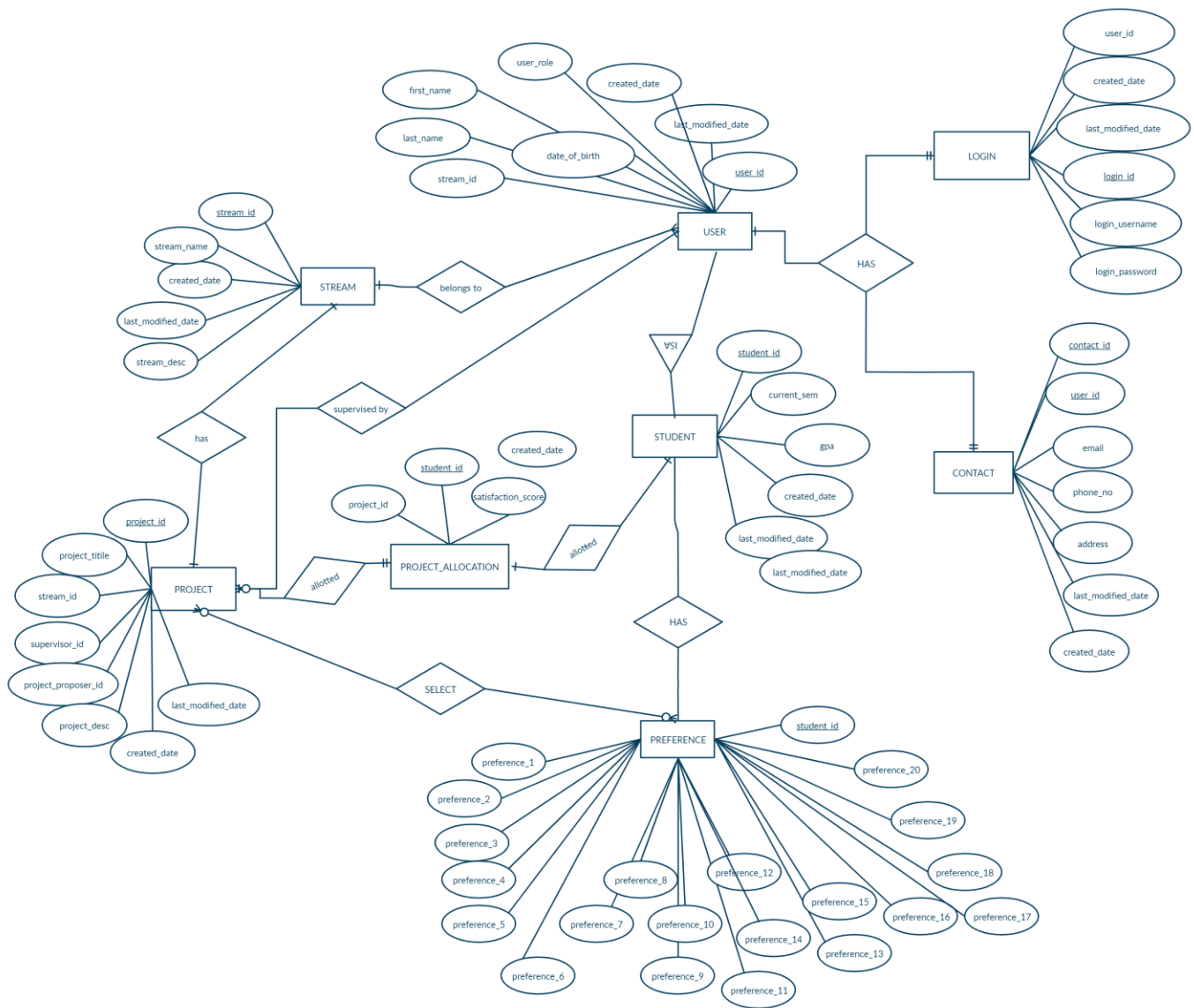


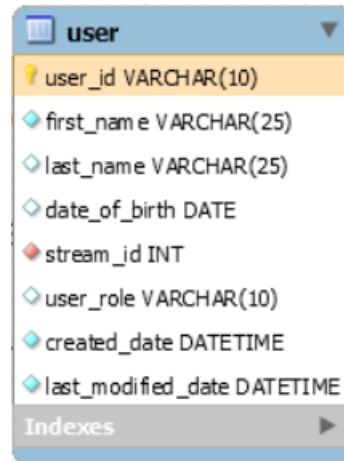
Figure 1: ER-Diagram

Entities and Attributes:

Entities of the database in support of the application are:

- User
- Login
- Contact
- Student
- Stream
- Preference
- Project
- Project_allocation

Entity User:



The image shows a screenshot of a database table definition for a table named 'user'. The table has the following attributes: 'user_id' (VARCHAR(10)) is the primary key, indicated by a yellow key icon; 'first_name' (VARCHAR(25)), 'last_name' (VARCHAR(25)), 'date_of_birth' (DATE), 'stream_id' (INT), 'user_role' (VARCHAR(10)), 'created_date' (DATETIME), and 'last_modified_date' (DATETIME) are other attributes, each indicated by a blue diamond icon. At the bottom, there is a tab labeled 'Indexes' with a right-pointing arrow.

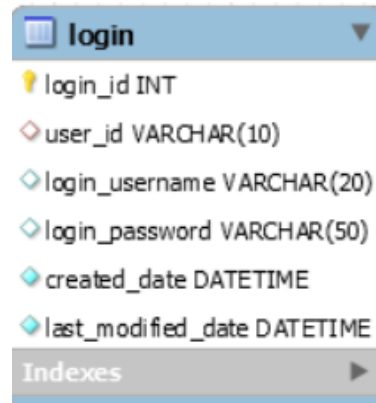
Attribute	Data Type
user_id	VARCHAR(10)
first_name	VARCHAR(25)
last_name	VARCHAR(25)
date_of_birth	DATE
stream_id	INT
user_role	VARCHAR(10)
created_date	DATETIME
last_modified_date	DATETIME

Figure 2: User table

This entity plays a very important role in the database. This entity stores all the information about the users of this application. Below are the attributes of this entity.

- user_id: This is the primary key of this entity and uniquely identifies every user of this application.
- first_name: This attribute captures the first name of the user.
- last_name: This attribute captures the last name of the user.
- date_of_birth: This attribute captures the birth date of the user in format 'yyyy-mm-dd'.
- stream_id: This attribute captures the stream id that the user belongs to. This is a foreign key.
- user_role: It captures the role of the user.
- created_date: This is used to capture the created date of the record. The default value has been set to current timestamp.
- last_modified_date: This is used to capture the last modified date of the record. The default value has been set to current timestamp.

Entity Login:



login	
login_id	INT
user_id	VARCHAR(10)
login_username	VARCHAR(20)
login_password	VARCHAR(50)
created_date	DATETIME
last_modified_date	DATETIME

Indexes

Figure 3: Login table

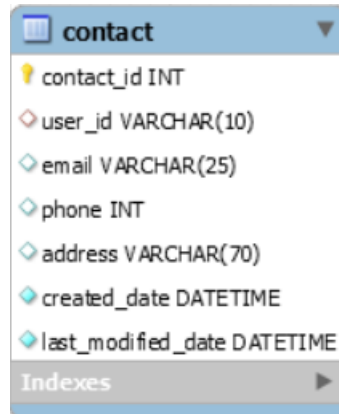
This entity captures the login details of all the users for the application. The attributes for the entity are:

- login_id: This is the primary key of the entity. It is defined as an auto-increment attribute, and hence values need not be inserted.
- user_id: This attribute contains the user id; the login is associated to and is a foreign key.
- login_username: Captures the username of the related user for login.
- login_password: Captures the credentials of the related user for login.
- created_date: This is used to capture the created date of the record. The default value has been set to current timestamp.
- last_modified_date: This is used to capture the last modified date of the record. The default value has been set to current timestamp.

Relations:

User and Login: One user can have only one login record in the login table. They are connected in the form of one-to-one with a relationship as 'has'. Hence, the relationship can be defined as 'Each user has only one login record'.

Entity Contact:



The image shows a screenshot of a database table definition window titled 'contact'. It lists the following attributes: 'contact_id' as an INT with a primary key icon (yellow lightning bolt), 'user_id' as a VARCHAR(10) with a foreign key icon (red diamond), 'email' as a VARCHAR(25) with a uniqueness icon (blue diamond), 'phone' as an INT with a uniqueness icon (blue diamond), 'address' as a VARCHAR(70) with a uniqueness icon (blue diamond), 'created_date' as a DATETIME with a uniqueness icon (blue diamond), and 'last_modified_date' as a DATETIME with a uniqueness icon (blue diamond). At the bottom, there is a tab labeled 'Indexes' with a right-pointing arrow.

Attribute	Data Type	Constraint
contact_id	INT	Primary Key
user_id	VARCHAR(10)	Foreign Key
email	VARCHAR(25)	Unique
phone	INT	Unique
address	VARCHAR(70)	Unique
created_date	DATETIME	Unique
last_modified_date	DATETIME	Unique

Figure 4: Contact table

This object stores the contacts details of the users. Contact has been made as a separate entity so that in case the users have multiple emails and phone number to be captured; it would be useful. For now, it has been assumed that the users have only one contact information to be stored. The attributes for the entity are:

- **contact_id:** This is the primary key of the entity. It is defined as an auto-increment attribute, and hence values need not be inserted.
- **user_id:** This attribute contains the user id to which the contact information is related to and is a foreign key.
- **email:** Captures the user's email id. A constraint is added to check integrity of the email id.
- **phone:** Captures the phone number of the user.
- **address:** This attribute stores the address details.
- **created_date:** This is used to capture the created date of the record. The default value has been set to current timestamp.
- **last_modified_date:** This is used to capture the last modified date of the record. The default value has been set to current timestamp.

Relations:

User and Contact: As we have assumed each user can have one distinct contact information, one user can have only one contact record in the contact table. They

are connected in the form of one-to-one with a relationship as 'has'. Hence, the relationship can be defined as 'Each user has one contact record'.

Entity Stream:

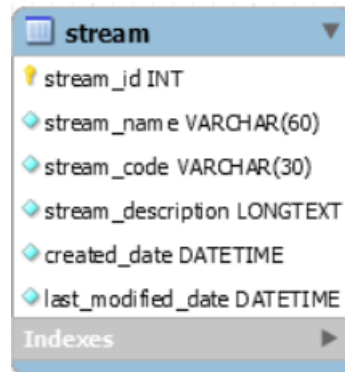


Figure 5: Stream table

This object has been created to capture the stream details of the staff, students and projects. There are mainly two streams, Cthulhu Studies (CS only) and Cthulhu Studies with a specialization in Dagon Studies (CS + DS). A third stream has been created (CS and/or CS + DS) since few of the projects can be associated to this stream, which can be taken up by students from both the streams. The attributes for the entity are:

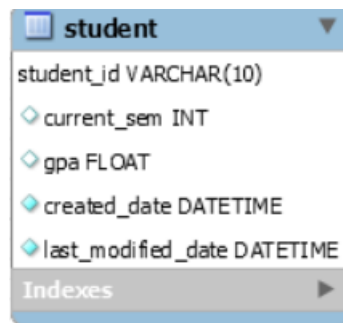
- **stream_id:** This is the primary key of the entity. It is defined as an auto-increment attribute, and hence values need not be inserted.
- **stream_name:** This attribute stores the stream name.
- **stream_code:** This attribute stores the stream code.
- **stream_description:** This attribute stores the description about the stream.
- **created_date:** This is used to capture the created date of the record. The default value has been set to current timestamp.
- **last_modified_date:** This is used to capture the last modified date of the record. The default value has been set to current timestamp.

Relations:

User and Stream: Each of the user (staff and student) belong to one of the two streams(CS or CS+DS). The third stream (CS and/or CS+DS) is defined only for

projects and users cannot be associated with this stream and check for the same has been defined in the stored procedure 'insert_users' which will be explained later as part of the report. User and Stream are connected in the form of many-to-one with a relationship as 'belongs to'. Hence, the relationship can be defined as 'One stream can be associated with many users and each user belongs to only one stream.

Entity Student:



student	
student_id	VARCHAR(10)
current_sem	INT
gpa	FLOAT
created_date	DATETIME
last_modified_date	DATETIME
Indexes	

Figure 6: Student table

This entity is an extension of the user entity. This entity is created separately to capture the details which belongs only to the student. The attributes for the entity are:

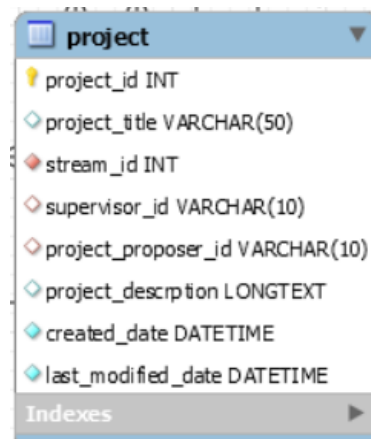
- **student_id:** This attribute captures the student id and uniquely identifies each records of the entity. This is also a foreign key which refers to the user_id in the user table and hence student_id being inserted has to be part of the user tables user_id.
- **current_sem:** Captures the current semester of the student. Assuming the student takes two years to graduate and involves 4 semesters, the student should belong to the 3rd semesters for the project allocation as the project takes place in the final two semesters.
- **gpa:** Captures the gpa of the student and the value can be between 1 to 4.2.
- **created_date:** This is used to capture the created date of the record. The default value has been set to current timestamp.

- `last_modified_date`: This is used to capture the last modified date of the record. The default value has been set to current timestamp.

Relations:

Student and User: Student table is a weak entity as it does not have a primary key of its own and `student_id` refers to `user_id` of the user table. As mentioned above student is an extension of the user table and is created to capture student specific details. User and Student have a 'ISA' relationship. Hence, the relationship can be defined as Student entity is a child of User entity. Students have to be part of the user table and user table contains both the staff and students.

Entity Project:



project	
project_id	INT
project_title	VARCHAR(50)
stream_id	INT
supervisor_id	VARCHAR(10)
project_proposer_id	VARCHAR(10)
project_description	LONGTEXT
created_date	DATETIME
last_modified_date	DATETIME
Indexes	

Figure 7: Project table

Project entity captures the entire project details. The attributes for the entity are:

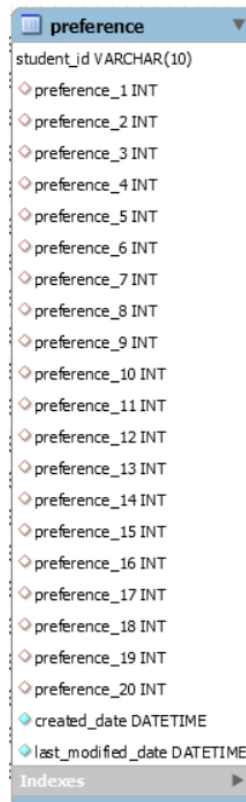
- `project_id`: This is the primary key of the entity. It is defined as an auto-increment attribute, and hence values need not be inserted.
- `project_title`: This attribute captures the project title and a constraint – UNIQUE has been added since the project titles have to be different from each other.

- `stream_id`: This attribute captures the stream id that the user belongs to. This is a foreign key.
- `supervisor_id`: This captures the staff id who will be supervising the project and only a staff can supervise a project. This is a foreign key and refers to `user_id` on the user table.
- `project_proposer_id`: This attribute captures the ID, who has proposed the project and it can be either a staff or student. This is a foreign key and refers to `user_id` on the user table.
- `project_description`: This attribute stores the description about the project.
- `created_date`: This is used to capture the created date of the record. The default value has been set to current timestamp.
- `last_modified_date`: This is used to capture the last modified date of the record. The default value has been set to current timestamp.

Relations:

Project and Stream: Unlike users who can belong to only two streams, here projects can be related to any of the three streams. Each project is related to one of the three streams (CS, CS+DS, CS and/or CS+DS) and each stream is related to multiple projects. Project and Stream have a 'has' relationship.

Entity Preference:



preference	
student_id	VARCHAR(10)
preference_1	INT
preference_2	INT
preference_3	INT
preference_4	INT
preference_5	INT
preference_6	INT
preference_7	INT
preference_8	INT
preference_9	INT
preference_10	INT
preference_11	INT
preference_12	INT
preference_13	INT
preference_14	INT
preference_15	INT
preference_16	INT
preference_17	INT
preference_18	INT
preference_19	INT
preference_20	INT
created_date	DATETIME
last_modified_date	DATETIME
Indexes	

Figure 8: Preference table

A student can give a maximum of 20 preferences for the project preferences with a compulsion of at least one. Student_id here is a foreign key that refers to the student_id on the student table and hence student_id being inserted has to be part of the student tables student_id. All the preference attributes are foreign keys which refer to project_id on the project table and hence the preference being entered has to be part of the project_id on the project table. created_date and last_modified_date is used to capture the created and last modified date of the record. The default value for both has been set to current timestamp. This is a weak entity since it does not have any attributes of its own and both the attributes are foreign keys.

Relations:

Project and Preferences: A student has to give a minimum of one project preference with a maximum of 20. Here each preference given by student is relates to one of the projects. The relationship between Project and Preferences is 'select' and the cardinality is each preference is related to one project and one project is related to may preferences.

Student and Preference: Here students can have a maximum of 20 preferences. Considering only 3rd semesters students are eligible for the projects, students belonging to other semesters will not be allowed to give their preference. The relationship between Student and Preference is 'has' and the cardinality is each preference is related to one student and only eligible students have Preference associated with them.

Entity *project_allocations*:

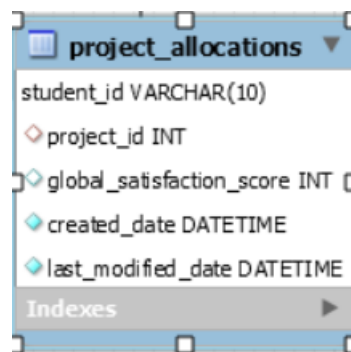


Figure 9: Project allocation table

This entity is used to capture the final projects allotted to students and each student can have only one project allotted. The attributes for the entity are:

- **student_id:** Captures the student id and acts as both primary key and foreign key.
- **project_id:** Captures the project id and acts as a foreign key.
- **created_date:** This is used to capture the created date of the record. The default value has been set to current timestamp.
- **last_modified_date:** This is used to capture the last modified date of the record. The default value has been set to current timestamp.

Created_date and last_modified_date have been used in all the entities of the database and are inspired from [\[2\]](#).

Relations:

Student and Project Allocation: Project is allotted to all student from 3rd semester. There is a one-to-ne relationship between student and project allocation. The relationship between Student and Project Allocation is 'allotted' and the

cardinality is each record present in the project allocation belongs to one unique student and vice-versa.

Project and Project allocation: There might be many projects proposed by the staff but not all of them need to be allotted to students. There is a one-to-many relationship between Project and project allocation. The relationship between Project and Project Allocation is 'allotted'. Cardinality is each of the record present in the project allocation is related to one of the projects in the project table but not all projects in the project table be related to records of the project allocation table.

Physical Model: Physical model plays an important role and acts as foundation in creation of the database. 'A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables' [<https://www.1keydata.com>][1].

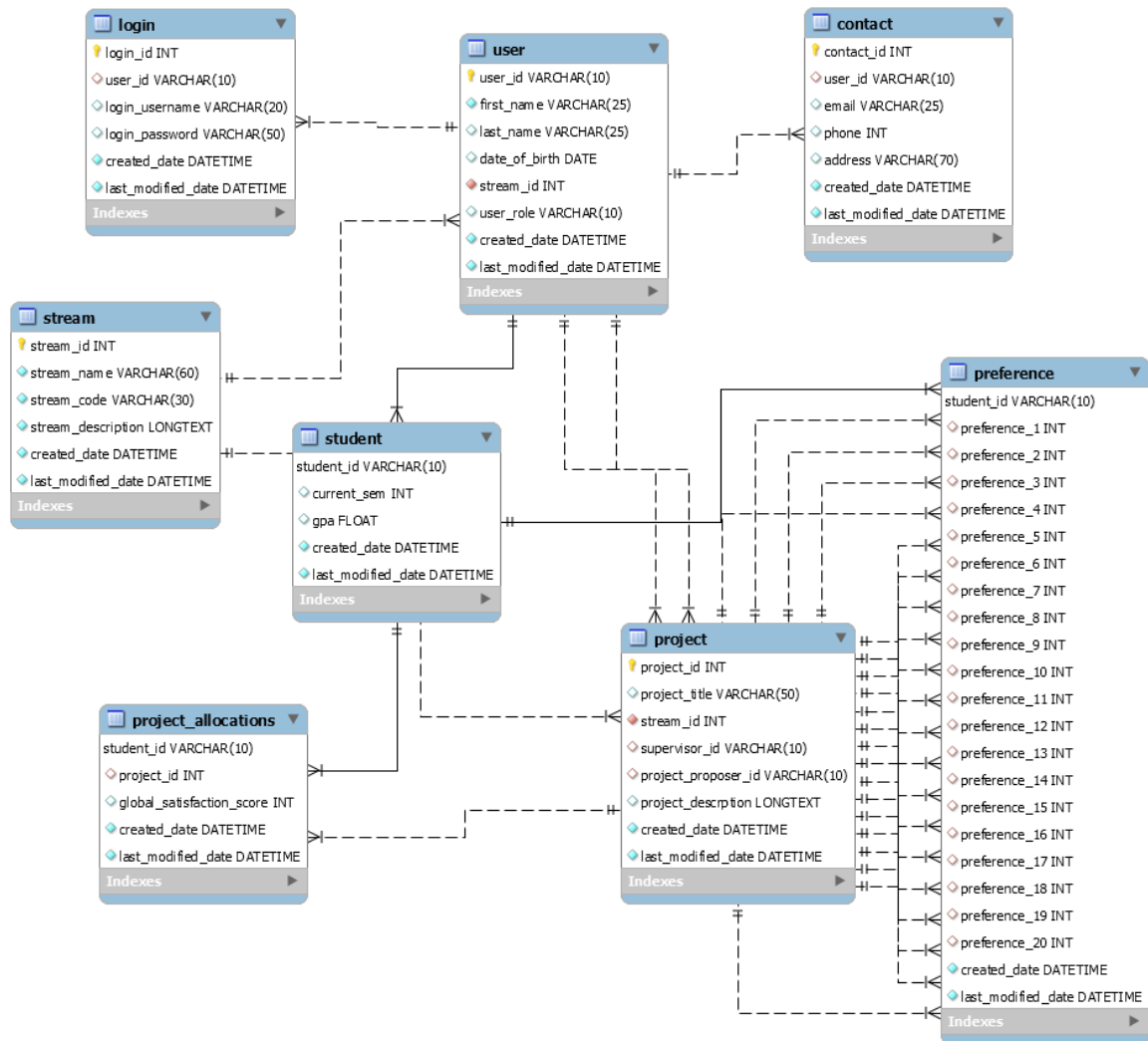


Figure 10: Physical model

The above design is proposed by taking into account some of the effective principles of a good database design. All tables are subject based, which helps us to avoid any redundant data. The above design provides access to information that is needed to join tables when required. All the tables are normalized which helps in support activities and also ensures the integrity and the accuracy of the data stored. Database are also used for a wide range of reporting and data processing needs; the above design suffices the same. Entities have been generalized keeping in mind further extensibility to the project. Taking into account the security aspect of the system, a login entity has been created to support the same. All the above consideration makes this design an effective one.

3. Database Structure: A Normalized View

Data stored illogically causes a wide variety of problems. An effective database has to be logical, efficient and robust. Two of the main things to be taken into consideration while designing a database would be data redundancy and integrity. Normalization helps in eliminating any redundant data and also makes sure data is stored logically. Different types of normalizations are:

1NF: A entity is in 1NF if it contains only atomic values and there are no repeating groups.

2NF: A entity is in 2NF if it satisfies 1NF and also non-key attributes are fully functional dependent on the primary key.

3NF: A entity is in 3NF if it satisfies 2NF and also there is no transitive functional dependency.

BCNF: A entity is in BCNF if it satisfies 3NF and also non-prime attributes shouldn't depend on part of prime attribute.

While designing this database have taken into account normalization so that data is stored logically and there is no redundant data. All the tables part of this database are normalized till BCNF.

User Table details:

There are 8 attributes as part of the [user table](#) namely user_id, first_name, last_name, date_of_birth, stream_id, user_role, created_date and last_modified_date. User_id has a data type of VARCHAR (10) which stores the id of the user and is a primary key which uniquely identifies all the records in the entity. First_name has a data type of VARCHAR (25) and stores the first name of the user. Last_name has a data type of VARCHAR (25) and stores the last name of the user. Date_of_birth is used to store the birth date of the user and has been defined as DATE. Stream_id is a foreign key to the stream table which helps in identifying the stream of the user and has been defined as an integer. User_role is defined as VARCHAR (10) and stores the role of the user which is either a staff or student and the same will be checked as part of the stored procedure which will be explained further in the report. created_date and last_modified_date are

defined as date with a default of current timestamp, these two fields are used to track the record creation and update.

User_id is the primary key and all the attributes are unique and are of atomic value hence the table is in 1NF. User_id is the only primary key and all other attributes are directly dependent on the primary key hence the table is in 2NF. The table also satisfies 3NF because all the columns first_name, last_name, date_of_birth, stream_id, user_role, created_date and last_modified_date are dependent on primary key and there is no transitive dependency. Table also satisfies BCNF since all not prime attributes depend on the primary key which is user_id.

Login Table details:

There are 6 attributes as part of the [login table](#) which are login_id, user_id, login_username, login_password, created_date and last_modified_date. Login_id is defined as auto increment and acts as a primary key. User_id is a foreign key that refers to the user table and is defined as VARCHAR (10). Login_username and login_password are used to capture username and password of the users respectively. Login_username and login_password is defined as VARCHAR (20) and VARCHAR (50) respectively. created_date and last_modified_date are defined as date with a default of current timestamp, these two fields are used to track the record creation and update.

login_id is the primary key and all the attributes are unique and are of atomic value hence the table is in 1NF. login_id is the only primary key and all other attributes are directly dependent on the primary key hence the table is in 2NF. The table also satisfies 3NF because all the columns user_id, login_username, login_password, created_date and last_modified_date are dependent on primary key and there is no transitive dependency. Table also satisfies BCNF since login_id and user_id are the two determinants which are also candidate keys.

Stream Table details:

There are 6 attributes in the [stream table](#) namely stream_id, stream_name, stream_code, stream_description, created_date and last_modified_date. stream_id

is defined as auto increment and acts as a primary key. Stream_name is defined as VARCHAR (60) and is used to store the name of the stream. Stream_code is defined as VARCHAR (30) and stores the associated stream code. Stream_description is defined as longtext which stores the information about the stream which could be lengthier and hence the longtext data type. created_date and last_modified_date are defined as date with a default of current timestamp, these two fields are used to track the record creation and update.

stream_id is the primary key and all the attributes are unique and are of atomic value hence the table is in 1NF. stream_id is the only primary key and all other attributes are directly dependent on the primary key and hence the table is in 2NF. The table also satisfies 3NF because all the columns stream_name, stream_code, stream_description, created_date and last_modified_date are dependent on primary key and there is no transitive dependency. Table also satisfies BCNF since there is only one primary key and all not prime attributes depend on the primary key which is stream_id.

Contact Table details:

There are 7 attributes in the [contact table](#) which are contact_id, user_id, email, phone, address, created_date and last_modified_date. stream_id is defined as auto increment and acts as a primary key. User_id is a foreign key that refers to the user table and is defined as VARCHAR (10). Email is defined as VARCHAR (25) and captures the email id of the user. Phone is defined as an integer and stores the phone number. Address is defined as VARCHAR (70) and stores the complete address details. created_date and last_modified_date are defined as date with a default of current timestamp, these two fields are used to track the record creation and update.

contact_id is the primary key and all the attributes are unique and are of atomic value hence the table is in 1NF. contact_id is the only primary key and all other attributes are directly dependent on the primary key and there is no partial dependency hence the table is in 2NF. The table also satisfies 3NF because all the columns user_id, email, phone, address, created_date and last_modified_date are dependent on primary key and there is no transitive dependency. Table also

satisfies BCNF since `contact_id` and `user_id` are the two determinants which are also candidate keys.

Student Table details:

There are 5 attributes in the [student table](#) namely `student_id`, `current_sem`, `gpa`, `created_date` and `last_modified_date`. `Student_id` is a primary key and also acts as a foreign key which refers to the `user_id` on user table. `Current_sem` is defined as an integer and captures the current semester which the student belongs to. `Gpa` is defined as float and captures the student gpa. `created_date` and `last_modified_date` are defined as date with a default of current timestamp, these two fields are used to track the record creation and update.

`student_id` is the primary key and all the attributes are unique and are of atomic value hence the table is in 1NF. `student_id` is the only primary key and all other attributes are directly dependent on the primary key and hence the table is in 2NF. The table also satisfies 3NF because all the columns `current_sem`, `gpa`, `created_date` and `last_modified_date` are dependent on primary key and there is no transitive dependency. Table also satisfies BCNF since there is only one primary key and all not prime attributes depend on the primary key which is `student_id`.

Project Table details:

[Project table](#) contains 8 attributes which are `project_id`, `project_title`, `stream_id`, `supervisor_id`, `project_proposer_id`, `project_description`, `created_date` and `last_modified_date`. `Project_id` is defined as an auto increment and acts as a primary key. `Project_title` has a data type of VARCHAR (50) and stores the title of the project, a constraint UNIQUE has been added to make sure each of the titles are distinct. `supervisor_id` is defined as VARCHAR (10) and is a foreign key which refers to the `user_id` on the user table. `Stream_id` is a foreign key to the stream table which helps in identifying the stream of the user and has been defined as an integer. `project_proposer_id` is defined as VARCHAR (10) and is a foreign key which refers to the `user_id` on the user table. `project_description` is defined as longtext which stores the information about the project which could be lengthier and hence the longtext data type. `created_date` and `last_modified_date` are

defined as date with a default of current timestamp, these two fields are used to track the record creation and update.

project_id is the primary key and all the attributes are unique and are of atomic value hence the table is in 1NF. project_id is the only primary key and all other attributes are directly dependent on the primary key and there is no partial dependency hence the table is in 2NF. The table also satisfies 3NF because all the columns project_title, stream_id, supervisor_id, project_proposer_id, project_description, created_date and last_modified_date are dependent on primary key and there is no transitive dependency. Table also satisfies BCNF since project_id and project_title are the two determinants which are also candidate keys.

Project Preference Table details:

[Project preference](#) table has 23 attributes. Student_id is the primary key and also a foreign key which refers to the student_id on the student table. All the 20 preference attribute are defined as integer which are foreign keys and refer to project_id on project table. created_date and last_modified_date are defined as date with a default of current timestamp, these two fields are used to track the record creation and update.

student_id is the primary key and all the attributes are unique and are of atomic value hence the table is in 1NF. student_id is the only primary key and all other attributes are directly dependent on the primary key and there is no partial dependency hence the table is in 2NF. The table also satisfies 3NF because all the attributes are dependent on the primary key and there is no transitive dependency. Table also satisfies BCNF since there is only one primary key and all not prime attributes depend on the primary key which is student_id.

Project Allocation Table details:

There are 5 attributes in the [project allocation](#) table namely student_id, project_id, global_satisfaction_score, created_date and last_modified_date. Student_id is the primary key and also acts as foreign key which refers to student_id on the student table. Project_id is a foreign key and refers to project table, global_satisfaction_score is defined as integer and stores the score

calculated by the application. `created_date` and `last_modified_date` are defined as date with a default of current timestamp, these two fields are used to track the record creation and update.

`student_id` is the primary key and all the attributes are unique and are of atomic value hence the table is in 1NF. `student_id` is the only primary key and all other attributes are directly dependent on the primary key and there is no partial dependency hence the table is in 2NF. The table also satisfies 3NF because all the columns `project_id`, `global_satisfaction_score`, `created_date` and `last_modified_date` are dependent on primary key and there is no transitive dependency. Table also satisfies BCNF since `student_id` and `project_id` are the two determinants which are also candidate keys.

4. Database Views

A view can be defined as a virtual table. It is similar to a table in structure, but data present in the view can be a combination of different tables in the database. It can contain all the rows or can be few based on the filtering conditions. There are many advantages of using views. With respect to security, views can help in providing access to only a set of users with a limited amount of data instead of the entire actual table. It helps in query simplification, since it can draw data from various tables and act as a single table and eliminates writing query for all the tables present in the view.

View for fetching all the details about the staff:

```
CREATE VIEW vw_staff_details AS
SELECT
  usr.user_id AS staff_id,
  concat(usr.first_name," ",usr.last_name) AS `name`,
  str.stream_name AS stream,
  cnt.email AS email_id,
  cnt.phone AS phone_number
from ((
  user usr
  join contact cnt on((cnt.user_id = usr.user_id)))
  join stream str on((str.stream_id = usr.stream_id)))
where (usr.user_role = "staff");
```

The above view is written to fetch all the details about the staff. The database has been normalized, hence a lot of information about staff are present in various

tables. This view helps in accessing all the important details about the staff whenever needed. Access to the above view can also be provided to students which will help them in understanding details about the professor and contact them if necessary. This view fetches data from three tables user, contact and stream. Staff_id, name is fetched from the user table, stream has been fetched from the stream table and email and phone is fetched from the contact table. User table has been joined with contact based on the user_id and stream has been joined with user table based on the stream_id. A filter has been on user_role attribute from user table which helps us fetch only staff records.

This view fetches data from three tables user, contact and stream. Staff_id, name is fetched from the user table, stream has been fetched from the stream table and email and phone is fetched from the contact table. User table has been joined with contact based on the user_id and stream has been joined with user table based on the stream_id. A filter has been used on user_role attribute from user table which helps us fetch only staff records.

	staff_id	name	stream	email_id	phone_number
▶	01STF001	Mike Lancaster	Cthulhu Studies	01STF001@cds.ie	855213252
	01STF002	Hendrix Huffington	Cthulhu Studies	01STF002@cds.ie	826565482
	01STF003	Maverick River	Cthulhu Studies with specialism in Dagon Studies	01STF003@cds.ie	833221452
	01STF004	Dakota Marigold	Cthulhu Studies	01STF004@cds.ie	866221456
	01STF005	Drake Booker	Cthulhu Studies with specialism in Dagon Studies	01STF005@cds.ie	833245567
	01STF006	Holden Huck	Cthulhu Studies	01STF006@cds.ie	833214568

Figure 11: Staff view output

View for fetching all the details about the student:

```
CREATE VIEW vw_student_details AS
select
sd.student_id AS student_id,
concat(usr.first_name," ",usr.last_name) AS `name`,
sd.current_sem AS current_sem,
str.stream_name AS stream,
sd.gpa AS GPA,
cnt.email AS email_id,
cnt.phone AS phone_number
from (((
student sd
join user usr on((usr.user_id = sd.student_id)))
join stream str on((str.stream_id = usr.stream_id)))
join contact cnt on((cnt.user_id = sd.student_id)));
```

The above view is written to fetch all the details about the students. The database has been normalized, hence a lot of information about students are present in various tables. This view helps in accessing all the important details about the students whenever needed. Access to this view can be given to staff to carry out administrative works whenever needed. This view can also help students and the class rep to know their classmates and to contact them for anything needed.

The above view has been fetched joining four tables student, user, contact and stream. Student_id, current_sem and gpa has been fetched from the student table, name is fetched from the user table, stream has been fetched from the stream table and email and phone is fetched from the contact table. A filter has been used on user_role attribute from user table which helps us fetch only student records.

	student_id	name	current_sem	stream	GPA	email_id	phone_number
▶	20STD001	Mike Pence	3	Cthulhu Studies	3.5	20STD001@cds.ie	874569321
	20STD002	James Robert	3	Cthulhu Studies	3.7	20STD002@cds.ie	858746986
	20STD005	Lily Olive	3	Cthulhu Studies	4.1	20STD005@cds.ie	832547862
	20STD007	Ella Garcia	3	Cthulhu Studies	3.8	20STD007@cds.ie	887445213
	20STD008	Kendall Sutton	3	Cthulhu Studies	3.5	20STD008@cds.ie	899654213
	20STD009	Madison Willow	3	Cthulhu Studies	3.3	20STD009@cds.ie	896541236
	20STD013	Abella Wren	3	Cthulhu Studies	3.1	20STD013@cds.ie	832546956
	20STD014	Araminta Birch	3	Cthulhu Studies	3.8	20STD014@cds.ie	863254566

Figure 12: Student view output

View for fetching all the Project belonging to stream 1 (CS) and stream 3(CS and/or CS+DS):

```
CREATE
VIEW `cs_projects` AS
select
project.project_id AS project_id,
project.project_title AS project_title,
project.stream_id AS stream_id,
str.stream_name AS stream_name,
project.project_description AS project_description,
concat(usr.first_name," ",usr.last_name) AS supervisor
from ((
project
join user usr on((usr.user_id = project.supervisor_id)))
join stream str on((str.stream_id = project.stream_id)))
where (project.stream_id = 1 or project.stream_id = 3);
```

	project_id	project_title	stream_id	stream_name	project_description	supervisor
▶	1	Sports and games in europe	1	Cthulhu Studies	Study about sports and games in europe	Mike Lancaster
	2	Sports and games in asia	1	Cthulhu Studies	Study about sports and games in asia	Hendrix Huffington
	3	Sports and games in africa	1	Cthulhu Studies	Study about sports and games in africa	Holden Huck
	4	Sports and games in north america	1	Cthulhu Studies	Study about sports and games in north america	Dakota Marigold
	5	Sports and games in south america	1	Cthulhu Studies	Study about sports and games in south america	Mike Lancaster
	6	Sports and games in antarctica	1	Cthulhu Studies	Study about sports and games in antarctica	Hendrix Huffington
	7	Sports and games in australia	1	Cthulhu Studies	Study about sports and games in australia	Holden Huck
	9	Sports and games in USA	1	Cthulhu Studies	Study about sports and games in USA	Hendrix Huffington

Figure 13: Stream 1 view output

View for fetching all the Project belonging to stream 2 (CS + DS) and stream 3(CS and/or CS+DS):

```
CREATE VIEW cs_ds_projects AS
select
project.project_id AS project_id,
project.project_title AS project_title,
project.stream_id AS stream_id,
str.stream_name AS stream_name,
project.project_description AS project_description,
concat(usr.first_name," ",usr.last_name) AS supervisor
from ((
project
join user usr on((usr.user_id = project.supervisor_id)))
join stream str on((str.stream_id = project.stream_id)))
where (project.stream_id = 2 or project.stream_id = 3);
```

	project_id	project_title	stream_id	stream_name	project_description	supervisor
▶	12	Football in europe	2	Cthulhu Studies with specialism in Dagon Studies	Study about football in europe	Maverick River
	13	Football in ireland	2	Cthulhu Studies with specialism in Dagon Studies	Study about football in ireland	Maverick River
	14	Football in africa	2	Cthulhu Studies with specialism in Dagon Studies	Study about football in africa	Drake Booker
	15	Football in antarctica	2	Cthulhu Studies with specialism in Dagon Studies	Study about football in antarctica	Drake Booker
	16	Football in asia	2	Cthulhu Studies with specialism in Dagon Studies	Study about football in asia	Drake Booker
	17	Football in sweden	2	Cthulhu Studies with specialism in Dagon Studies	Study about football in sweden	Mike Lancaster

Figure 14: Stream 2 view output

One of the important roles in this application is for the students to give preference for the projects. It is also important that students select projects belonging either to their own stream or projects that suit students from both the stream. Keeping this aspect in mind two views above have been created, one which fetches all the projects belonging to stream 1 (CS) and stream 3 (CS and/or CS+DS) and the other view which fetches all the projects belonging to stream 2 (CS+DS) and stream 3 (CS and/or CS+DS). Students belonging to the stream 1 (CS) can be given access to the first view while students belonging to stream 2 (CS+DS) can be given access to the other. This will make sure that students selection of projects will only belong to the stream that they are associated with.

Three tables (project, user and stream) have been joined to fetch the data. Supervisor is a concatenation of first_name and last_name which has been fetched from user table, stream has been fetched from stream table and all project related details have been fetched from project table. A filter has been used on stream_id to restrict project of the appropriate streams in both the views.

5. Procedural Elements

A stored procedure is a set of PLSQL statements which can be saved to be reused again and again. A trigger is a special type of stored procedure that automatically runs when an event occurs in the database server [3].

Procedures:

Procedure to insert users into user table and create their login records in login table:

```
DROP PROCEDURE IF EXISTS `insert_users`;
DELIMITER $$
CREATE PROCEDURE `insert_users`(
  IN user_id varchar(10),
  IN first_name varchar(25) ,
  IN last_name varchar(25),
  IN date_of_birth date,
  IN stream_id int,
  IN user_role varchar(10) )
BEGIN
  SET @pass = concat_ws('@',first_name,CAST(date_of_birth AS DATE));

  IF stream_id > 2
  THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Enter a valid stream id';
  END IF;

  IF (user_role != "staff" AND user_role != "student")
  THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Enter a valid user role';
  END IF;

  INSERT INTO user(user_id,first_name,last_name,date_of_birth,stream_id,user_role)
    values(user_id,first_name,last_name,date_of_birth,stream_id,user_role);

  INSERT INTO login(user_id,login_username,login_password)
    values(user_id,user_id,@pass) ;
END;
```

The above procedure is written to insert users into user table and automatically create login records for them in the login table. This procedure simplifies the process of creating login details separately. All the parameters defined are IN parameters. There are couple of checks that take place before inserting the details into the user table. Since users can belong to only stream 1 and 2, the stored procedure checks for the same in the first IF loop. Users are either staff or student hence the second IF loop checks the same for the attribute user_role. A variable has been declared to create a default password which is a combination of first_name and date of birth with a @ symbol in between.

```
mysql> CALL insert_users("20STD022","Jack","Micharney","2003-05-11",3,"student");
ERROR 1644 (45000): Enter a valid stream id
mysql> CALL insert_users("20STD022","Jack","Micharney","2003-05-11",2,"stnt");
ERROR 1644 (45000): Enter a valid user role
mysql> CALL insert_users("20STD022","Jack","Micharney","2003-05-11",2,"student");
Query OK, 1 row affected (0.02 sec)
```

Figure 15: Procedure Output

Triggers:

Trigger for user table:

```
DROP TRIGGER IF EXISTS user_bi_trig ;
DELIMITER $$
CREATE DEFINER=`root`@`localhost` TRIGGER `user_bi_trig` BEFORE INSERT ON `user`
FOR EACH ROW BEGIN
    IF EXISTS(select * from user where user_id=new.user_id)
        THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'User is already part of the system';
        END IF;
END$$
DELIMITER ;
```

The above trigger is written to prevent adding users who are already part of the system and gets invoked when any insert is made on the user table. This prevents data redundancy.

```
mysql> CALL insert_users("20STD001","Mike","Pence","2001-10-12",1,"student");
ERROR 1644 (45000): User is already part of the system
```

Figure 16: User table trigger output

Trigger for login table:

```
DROP TRIGGER IF EXISTS login_bi_trig ;
DELIMITER $$
CREATE TRIGGER `login_bi_trig` BEFORE INSERT ON `login` FOR EACH ROW BEGIN
    IF NOT EXISTS(select * from user where user_id=new.user_id)
        THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'User is not part of the system';
    END IF;
END$$
DELIMITER ;
```

The above trigger gets invoked if any inserts are made into login table. This trigger has been written to make sure the login records are allotted only if the user is already part of the system. The same trigger is also written for before update.

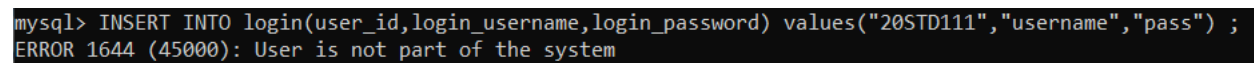
A terminal window with a black background and white text. It shows an SQL command being executed: 'mysql> INSERT INTO login(user_id,login_username,login_password) values("20STD111","username","pass") ;'. Below the command, an error message is displayed: 'ERROR 1644 (45000): User is not part of the system'.

Figure 17: Login table trigger output

Trigger for Project table:

```
DROP TRIGGER IF EXISTS project_bi_trig;
DELIMITER $$
CREATE DEFINER = CURRENT_USER TRIGGER `project_bi_trig` BEFORE INSERT ON
`project` FOR EACH ROW
BEGIN
    IF NOT EXISTS(SELECT * FROM user WHERE user_id=NEW.supervisor_id and user_role
    = "staff")
        THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Supervisor id entered is not a staff member';
    ELSEIF NOT EXISTS(SELECT * FROM user WHERE user_id=NEW.project_proposer_id)
        THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Invalid proposer ID';
    END IF;
END$$
DELIMITER ;
```

The above trigger is written on the project table and gets invoked with any inserts made. This trigger checks for two constraints, one to make sure the supervisor ID entered is only a member of staff since only staff can supervise a

project, the other one checks if the project proposer id is a valid id from the user table since both student and staff can propose a project. The same trigger is also written for before update.

```
mysql> INSERT INTO project VALUES(null,"Sports and games in europe",1,"20STD001","01STF001"," Study about and games in europe",curdate(),curdate());
ERROR 1644 (45000): Supervisor id entered is not a staff member
mysql> INSERT INTO project VALUES(null,"Sports and games in europe",1,"01STF001","05fff001"," Study about and games in europe",curdate(),curdate()); -- invalid project proposer id
ERROR 1644 (45000): Invalid project proposer ID
mysql>
```

Figure 18: Project table trigger output

Trigger for Contact table:

```
DROP TRIGGER IF EXISTS contact_bi_trig;
DELIMITER $$
CREATE TRIGGER `contact_bi_trig` BEFORE INSERT ON `contact` FOR EACH ROW
BEGIN
    IF NOT EXISTS(SELECT * FROM user WHERE user_id=NEW.user_id)
        THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'User is not part of the system';
        END IF;
END$$
DELIMITER ;
```

The above trigger is written on the contact table and gets invoked when any inserts are made. This trigger makes sure that the contact details being inserted is part of the system and hence avoiding any junk data being created in the system. The same trigger is also written for before update.

```
mysql> INSERT INTO contact(user_id,email,phone,address) VALUES("20STD111","20STD001@cds.ie",0874569321,"341,Abbey Street,Fingal,Dublin");
ERROR 1644 (45000): User is not part of the system
mysql>
```

Figure 19: Contact table trigger output

Trigger for Student table:

```
DROP TRIGGER IF EXISTS student_bi_trig;
DELIMITER $$
CREATE TRIGGER `student_bi_trig` BEFORE INSERT ON `student` FOR EACH ROW
BEGIN
    IF NOT EXISTS(SELECT * FROM user WHERE user_id=new.student_id AND
        user_role="student")
        THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Enter a valid student_id';
    ELSEIF new.current_sem > 4
        THEN
            SIGNAL SQLSTATE '45000'
```



```

        SET MESSAGE_TEXT = 'Enter a valid semester';
ELSEIF new.gpa < 0 or new.gpa > 4.2
    THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'GPA can only be between 0 to 4.2';
END IF;
END$$
DELIMITER ;

```

The above trigger gets invoked if any inserts are made on the student table. This trigger makes three checks before insertion. The first IF loop checks if the student id is valid and is part of the user table having user role defined as student. Second IF loop checks if the semester entered is valid or not, for this application I have assumed students have 2 years of study consisting of 4 semesters. The third IF loop checks if the entered gpa is valid, which should be between 0 and 4.2. The same trigger is also written for before update.

```

mysql> INSERT INTO student(student_id,current_sem,gpa) VALUES ("20STD111",3,3.5);
ERROR 1644 (45000): Enter a valid student_id
mysql> INSERT INTO student(student_id,current_sem,gpa) VALUES ("20STD022",5,3.5);
ERROR 1644 (45000): Enter a valid semester
mysql> INSERT INTO student(student_id,current_sem,gpa) VALUES ("20STD002",3,4.8);
ERROR 1644 (45000): GPA can only be between 0 to 4.2
mysql>

```

Figure 20: Student table trigger output

Trigger for Preference table:

```

DROP TRIGGER IF EXISTS `pref_bi_trig`;
DELIMITER $$
CREATE TRIGGER `pref_bi_trig` BEFORE INSERT ON `preference` FOR EACH ROW
BEGIN
    DECLARE std_stream INT;
    DECLARE std_sem int;
    SELECT stream_id INTO std_stream FROM user WHERE user_id=new.student_id;
    SELECT std.current_sem INTO std_sem FROM student std Where student_id =
new.student_id;

    DROP TEMPORARY TABLE IF EXISTS temp_pref;
    CREATE TEMPORARY TABLE temp_pref(Preference INT);

    INSERT INTO temp_pref VALUES
(new.preference_1),(new.preference_2),(new.preference_3),(new.preference_4),(new.preference_5 ),
(new.preference_6 ),(new.preference_7 ),(new.preference_8),(new.preference_9
),(new.preference_10 ),(new.preference_11 ),(new.preference_12),
(new.preference_13 ),(new.preference_14 ),(new.preference_15
),(new.preference_16),(new.preference_17 ),(new.preference_18 ),(new.preference_19
),(new.preference_20);

    IF NOT EXISTS(SELECT * FROM student WHERE student_id=NEW.student_id)
    THEN

```



```

        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Enter a valid student_id';
    ELSEIF (std_sem != 3)
        THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Only students from 3rd semester can give preference.';
    ELSEIF NEW.preference_1 IS NULL
        THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Preference 1 is a must';
    ELSEIF EXISTS(SELECT * FROM temp_pref tpf INNER JOIN project p ON tpf.Preference=p.project_id
    WHERE p.stream_id<>std_stream AND p.STREAM_ID <> 3)
        THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Entered project preference does not belong to students
stream';

    END IF;
END$$
DELIMITER ;

```

The above trigger gets invoked if any inserts are made on the preference table. This trigger makes four checks before any inserts. First IF loop checks if the student id is valid or not. Project is allocated in the students final year and it begins in the 3rd semester, second IF loop checks if the student belongs to 3rd semester or not. Since student is supposed to give a minimum of one preference, third IF loop checks if at least one preference is specified or not. Students are supposed to give preference for projects only if the selected project belongs to their stream or the project has to belong to CS+DS stream, fourth IF loop checks for the same. The same trigger is also written for before update.

```

mysql> INSERT INTO preference (student_id,preference_1,preference_2) values ("20STD112",3,6);
ERROR 1644 (45000): Enter a valid student_id
mysql> INSERT INTO preference (student_id,preference_1,preference_2) values ("20STD226",3,6);
ERROR 1644 (45000): Only students from 3rd semester can give preference.
mysql> INSERT INTO preference (student_id) values ("20STD225");
ERROR 1644 (45000): Preference 1 is a must
mysql> INSERT INTO preference (student_id,preference_1,preference_2) values ("20STD225",3,14);
ERROR 1644 (45000): Entered project preference does not belong to students stream
mysql>

```

Figure 21: Preference table trigger output

Trigger for Project allocation table:

```

DROP TRIGGER IF EXISTS project_allocations_bi_trig ;
DELIMITER $$
CREATE TRIGGER `project_allocations_bi_trig` BEFORE INSERT ON `project_allocations`
FOR EACH ROW BEGIN
    DECLARE std_stream int;
    DECLARE proj_stream int;

```

```
SELECT stream_id INTO proj_stream FROM project where project_id = new.project_id;
SELECT stream_id INTO std_stream FROM user where user_id=new.student_id;

IF EXISTS (select * from project_allocations WHERE project_id=new.project_id)
    THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Project has already been allotted to a different
student.';
    ELSEIF (proj_stream <> std_stream AND proj_stream <> 3)
        THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Students must be allocated a project of their
stream.';

END IF;
END$$
DELIMITER ;
```

The above trigger gets invoked when any inserts are made on the project allocation table. Each student should be allocated to one project and should be unique to each other. Each of the project allocated to the student must belong to their own stream or should belong to (CS+DS). The above trigger checks for both the cases. The same trigger is also written for before update.

```
mysql> INSERT INTO project_allocations(student_id,project_id,global_satisfaction_score) VALUES("20STD225",1,6); -- checks if project is already allotted or not
ERROR 1644 (45000): Project has already been allotted to a different student.
mysql> INSERT INTO project_allocations(student_id,project_id,global_satisfaction_score) VALUES("20STD225",20,6); -- checks if project allotted belongs to student stream
ERROR 1644 (45000): Students must be allocated a project of their stream.
mysql>
```

Figure 22: Project allocation table trigger output

Few of the triggers have been written though there might be a foreign key which might do the same job as trigger by not allowing insertion of irrelevant data, this has been done so that the error received from the trigger accurately points out the error made during insert.

6. Example Queries

In this database a number of tables, triggers, stored procedures and views are created which helps in maintaining the database redundancy and integrity. Below are few sample queries that provides few more insights.

Query 1:

```
SELECT
    pa.`student_id`,
    concat(usr.first_name," ",usr.last_name) AS `Student Name`,
```

```

    pa.project_id,
    'p`.`project_title`,
    'pa`.`global_satisfaction_score`,
    stf.name as supervisor,
    str.stream_name as 'project stream'
FROM project_allocations as pa
join project p on p.project_id = pa.project_id
inner join user usr on usr.user_id = pa.student_id
join stream str on str.stream_id = p.stream_id
join vw_staff_details stf on stf.staff_id = p.supervisor_id
order by student_id;

```

Project allocation table gives limited details about the final project allocations. The above query gives final projects allotted to students along with complete details about projects, output of this query can be used to publish the final results of the system.

student_id	Student Name	project_id	project_title	global_satisfaction_score	supervisor	project stream
20STD001	Mike Pence	1	Sports and games in europe	6	Mike Lancaster	Cthulhu Studies
20STD002	James Robert	3	Sports and games in africa	1	Holden Huck	Cthulhu Studies
20STD003	Audrey Jack	15	Football in antarctica	2	Drake Booker	Cthulhu Studies with specialisation in Dagon Stu...
20STD004	Georgia Juliet	17	Football in sweden	4	Mike Lancaster	Cthulhu Studies with specialisation in Dagon Stu...
20STD005	Lily Olive	2	Sports and games in asia	6	Hendrix Huffington	Cthulhu Studies
20STD006	Alfie Edward	14	Football in africa	3	Drake Booker	Cthulhu Studies with specialisation in Dagon Stu...
20STD007	Ella Garcia	9	Sports and games in USA	6	Hendrix Huffington	Cthulhu Studies
20STD008	Kendall Sutton	10	Sports and games in Germany	9	Dakota Marigold	Cthulhu Studies

Figure 23: Query 1 output

Query 2:

```

SELECT
    p.supervisor_id,
    concat(usr.first_name," ",usr.last_name) as supervisor_name,
    count(*) as total_no_of_projects
FROM Project as p
join user usr on usr.user_id = p.supervisor_id
GROUP BY p.supervisor_id;

```

The above query helps us to understand how many projects each of the staff are responsible for.

supervisor_id	supervisor_name	total_no_of_projects
01STF001	Mike Lancaster	4
01STF002	Hendrix Huffington	3
01STF003	Maverick River	2
01STF004	Dakota Marigold	2
01STF005	Drake Booker	6
01STF006	Holden Huck	3

Figure 24: Query 2 output

Query 3:

```

SELECT `project_id`,
       `project_title`,
       p.stream_id,
       `supervisor_id`,
       `project_proposer_id`,
       `project_description`
FROM `project` p
inner join user usr on usr.user_id = p.project_proposer_id
Where usr.user_role = "student";

```

The above query helps us understand how many projects were proposed by students. This will help us to allot those projects directly to students who proposed them.

project_id	project_title	stream_id	supervisor_id	project_proposer_id	project_description
15	Football in antarctica	2	01STF005	20STD003	Study about football in antarctica
16	Football in asia	2	01STF005	20STD012	Study about football in asia

Figure 25: Query 3 output

Query 4:

```

SELECT std.student_id,concat(usr.first_name," ",usr.last_name) as 'student
name',std.current_sem,stream_id as 'student stream' from student std
join user usr on usr.user_id=std.student_id
WHERE student_id NOT IN (
SELECT student_id from project_allocations);

```

The above query helps us understand how many students have not been allotted a project yet. The details will also help us understand why they were not allotted a project, one of them could be they might not belong to 3rd semester hence they were not allotted.

student_id	student name	current_sem	student stream
20STD225	Jack Micharney	3	1
20STD226	Jack Micharney	2	1

Figure 26: Query 4 output

Query 5:

```
SELECT project_id, project_title, stream_name AS 'project stream' from project p
inner join stream str on str.stream_id=p.stream_id
WHERE project_id NOT IN (
SELECT project_id from project_allocations);
```

The above query helps us to know how many projects were not allotted. It is possible that projects were proposed by staff exceeded the number of students. It could also help in allotting these projects to students who have not been allotted any projects because of various reasons.

project_id	project_title	project stream
5	Sports and games in south america	Cthulhu Studies
20	Football in asian games	Cthulhu Studies with specialisation in Dagon Stu...

Figure 27: Query 5 output

7. Conclusions

This database has been designed to support the application designed for project allocation to final year students belonging to School of CS. In future this could be extended to students of the entire university since we know that most of the schools of the university have project, internship or thesis as part of their final year. Taking this database as a base structure we could extend it to all the schools of the university. We can add another table for schools and the current stream table provides the flexibility to add stream belonging to all the schools. Another staff stream table would be required to add staff stream details. Since projects can also be proposed by the students, we can add another extension where these projects can undergo an approval from the staff who would want to supervise them. Considering this serves a limited number of users for now, in case this is extended to all the schools, indexing should be considered for the better performance of the system. Since most of the universities have an application to support the staff and students which includes registration, assignments tracking etc, the above thought of having this project allocation system can be integrated with the existing university base application instead of having two different applications.

Acknowledgements

The outcome of this project required a lot of effort and assistance. I would like to convey my gratitude to Prof. Veale Tony for all the guidance and assistance. Also, I would also like to thank all the teaching assistants who were always available for any guidance and help. I would also like to state that all the work in this project are completely done by me and wherever others work is referred, due references are given. This project was a great learning curve, thank you for providing me this opportunity.

References

1. <https://www.1keydata.com/datawarehousing/physical-data-model.html>
2. <https://www.marcus-povey.co.uk/2013/03/11/automatic-create-and-modified-timestamps-in-mysql/>
3. <https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver15>
4. <https://www.lucidchart.com/pages/database-diagram/database-schema>
5. <http://lazyheap.com/database-design-principles/>
6. <https://www.tutorialspoint.com/Types-of-DBMS-Entities-and-their-examples>
7. <https://www.guru99.com/er-diagram-tutorial-dbms.html>
8. <https://www.c-sharpcorner.com/blogs/advantages-and-disadvantages-of-views-in-sql-server1>
9. <https://www.1keydata.com/database-normalization/>