Centralized College Database

Index

Project Summary

This project aims at creating a centralised repository of data for a college with several departments. We are using a mix of the Indian and international model of division of departments and courses. As we are aware most international Universities broadly classify courses/degrees into generalized departments like 'Business', 'Law', 'Engineering', etc. each offering different types of graduate and undergraduate degrees, certificates and diplomas. Within these departments there is overlap of subjects sometimes where electives are offered across departments. The aim of this project is to create a relational schema such that there is less than 1% of data redundancy, although it may seem complicated to split entities when it may appear that the data could have been incorporated into an existing table, this has been done deliberately. The purpose of splitting the data into more tables is to ensure flexibility of the database, it also allows the user to change the value of an attribute in one place only thereby maintaining the integrity of the database. It allows the database to grow and add other entities by tying in relations to the existing tables. This allows for more derived attributes that may be dynamic, for example, change in fees, staff, nomenclature of programmes etc. We have aimed to keep the database dynamic. This allows the user to query the database for a myriad amount of information and also easily add newer data to it.

ER Diagram

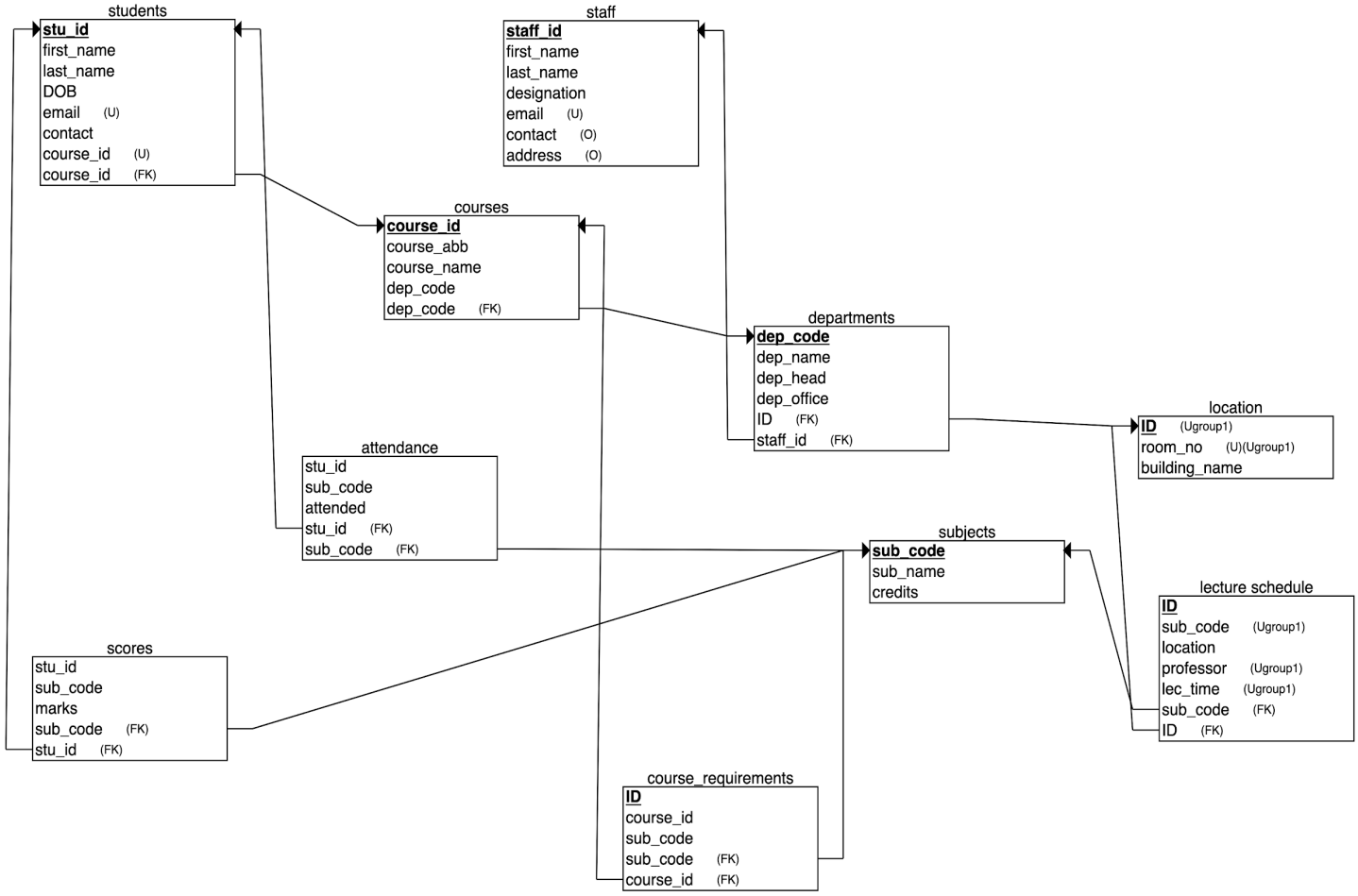
students
stu_id
first_name
last_name
DOB
email (U)
contact
course_id (U)
course_id (FK)
staff
staff_id
first_name
last_name
designation
email (U)
contact (O)
address (O)
courses
course_id
course_abb
course_name
dep_code
dep_code (FK)
departments
dep_code
dep_name
dep_head
dep_office
ID (FK)
staff_id (FK)
location
ID (Ugroup1)
room_no (U)(Ugroup1)
building_name
attendance
stu_id
sub_code
attended
stu_id (FK)
sub_code (FK)
subjects
sub_code
sub_name
credits
lecture schedule
ID
sub_code (Ugroup1)
location
professor (Ugroup1)
lec_time (Ugroup1)
sub_code (FK)
ID (FK)
scores
stu_id
sub_code
marks
sub_code (FK)
stu_id (FK)
course_requirements
ID
course_id
sub_code
sub_code (FK)
course_id (FK)

Table Details

A recurring theme that can be observed in this database is the abundant use of foreign keys. We use this method in order to avoid repetition of information, this also allows for easy cross-referencing. Since the database was designed by us we have not used any datasets as those would not have the required data fields. We are predominantly using foreign keys therefore there is very little data entry. This is done keeping in mind that the actual user will always have an interactive front end that will display the values of the keys and not the keys themselves therefore, we can make the most efficient use of the database.

1) **Departments Table**

```
CREATE TABLE departments(
        dep_code INT NOT NULL PRIMARY KEY,
        dep_name VARCHAR (255) NOT NULL,
        dep_head INT NOT NULL,
        dep_office INT NOT NULL,
        FOREIGN KEY(dep_office) REFERENCES location (ID),
        FOREIGN KEY(dep_head) REFERENCES staff (staff_id)
);
```

This table consists of the main departments. Each department has its own unique identification code. Every department is assigned one staff member as Dean and its own office. The column names are self explanatory and we use the foreign key constraints in order to identify the dean and office location. It's the simplest table.

2) **Staff table**

```
CREATE TABLE staff(

staff_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
first_name VARCHAR(255) NOT NULL,
last_name VARCHAR (255) NOT NULL,
designation VARCHAR (255) NOT NULL,
email VARCHAR (255) NOT NULL UNIQUE,
contact VARCHAR (255),
address VARCHAR (255) );
```

The staff table keeps record of all the personnel/employees of the university. The departments can share staff and teachers. Email in this table is a unique and mandatory constraint. This will prevent duplication of entries.

3) **Location Table**

```
CREATE TABLE location(
        ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
        room_no INT NOT NULL,
        building_name VARCHAR (255) NOT NULL,
        UNIQUE (room_no, building_name)) ;
```

This table identifies and stores all the classrooms, gyms, libraries, department offices, etc. All the campus locations will be recorded here. Since every building will have their own room numbers, there will be overlap, in order to prevent multiple entries of the same room number in the same building we create a unique constraint and therefore each building will not have repeated room numbers.

4) **Courses Table**

```
CREATE TABLE courses(
        course_id INT NOT NULL,
        course_abb VARCHAR (255) NOT NULL,
        course_name VARCHAR (255) NOT NULL,
        dep_code INT NOT NULL,
        PRIMARY KEY (course_id),
        FOREIGN KEY(dep_code) REFERENCES departments (dep_code)
);
```

This table stores all the courses offered by different departments, the only new values we enter here are the course name and its abbreviation. The department name is identified using the department code as a foreign key.

5) **Subjects Table**

```
CREATE TABLE subjects(
sub_code INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
sub_name VARCHAR (255) NOT NULL,
credits INT NOT NULL,
);
```

All the subjects across the university are shared here. Sometimes courses are offered across departments, or there is a course from one department that allows electives from another department. This table also reduces the need for multiple entries of the same course. The credit hours for each course are also listed here. These hours also double up as the number of lectures for that subject. The subject is identified by its own unique code.

6) **Course Requirements Table**

```
CREATE TABLE course_requirements(
        ID INT AUTO_INCREMENT KEY,
        course_id INT NOT NULL,
        sub_code INT NOT NULL,
        FOREIGN KEY(sub_code) REFERENCES subjects (sub_code),
        FOREIGN KEY(course_id) REFERENCES courses (course_id)
        );
```

This table ties the course and subjects table together, it stores all the core requirements that are necessary to complete the course, another table can later be added for the electives chosen by each student. This provides flexibility to the database. We can again see the use of foreign keys to prevent multiple repeated entries which also protects against errors.

7) **The Schedule Table**

```
CREATE TABLE lecture_schedule(
        ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
        sub_code INT NOT NULL,
        professor INT NOT NULL,
        location INT NOT NULL,
        lec_time DATETIME ,
        FOREIGN KEY (professor) REFERENCES staff (staff_id),
        FOREIGN KEY (location) REFERENCES location (ID),
        UNIQUE(sub_code, professor, lec_time)
);
```

This table keeps track of the schedule of classes. The only new information we enter here is the date and time of the lecture, all the other attributes are taken by way of foreign keys from other tables. We have also identified a unique constrain being the subject, professor and time combination. This ensures that the lectures don't clash. We store both the date and time of the lecture.

8) Students Table

```
CREATE TABLE students(

stu_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
first_name VARCHAR(255) NOT NULL,
last_name VARCHAR (255) NOT NULL,
DOB VARCHAR (255) NOT NULL,
email VARCHAR (255) NOT NULL UNIQUE,
contact VARCHAR(255),
course_id INT NOT NULL,
FOREIGN KEY(course_id) REFERENCES courses (course_id)
);
```

This table stores the students' details and has the student id as the primary key. Similar to the staff table, email is unique and mandatory in this table as well. This is aimed at preventing redundancy. The student's chosen course is linked through the course_id that eventually connects with the subjects.

9) Scores Table

```
CREATE TABLE scores(
        stu_id INT NOT NULL,
        sub_code INT NOT NULL,
        marks DECIMAL DEFAULT 0,
        FOREIGN KEY(sub_code) REFERENCES subjects (sub_code),
        FOREIGN KEY(stu_id) REFERENCES students (stu_id)
        );
```

This table stores the student's marks along with the subject code. Since there is a lot of repetition of the student and subject we now start to see the liberal use of foreign keys. Since new students do not have marks the default is set to zero.

10) Attendance Table

```
CREATE TABLE attendance(
        stu_id INT NOT NULL,
        sub_code INT NOT NULL,
        attended INT DEFAULT 0,
        FOREIGN KEY (stu_id) REFERENCES students (stu_id),
        FOREIGN KEY (sub_code) REFERENCES subjects (sub_code)
        );
```

As the name implies this table stores the students attendance. The credits from the subject table can be used to calculate the percentage as the credits double up as the number of lectures per subject.

SAMPLE QUERIES

 # getting student course details

```
SELECT concat(s.first_name, " ", s.last_name) as name, s.DOB, s.email, s.contact,
courses.course_abb, courses.course_name FROM students AS s
JOIN courses
ON s.course_id = courses.course_id;
```

 # getting student scores

```
SELECT concat(s.first_name, " ", s.last_name) as Name, s.stu_id, sub.sub_name, scores.marks,
sub.credits from students as s
JOIN scores ON s.stu_id = scores.stu_id
JOIN subjects AS sub ON scores.sub_code = sub.sub_code;
```

 # getting course requirements

```
SELECT c.course_abb, c.course_name, sub.sub_name,
sub.credits FROM courses as c  JOIN course_requirements as cr ON
cr.course_id = c.course_id JOIN subjects as sub ON cr.sub_code = sub.sub_code;
```

 # getting department information

```
SELECT d.dep_code, d.dep_name, concat(s.first_name, " ", s.last_name) AS DeptHead,
concat(l.room_no, " ", l.building_name) AS Location
FROM departments AS d JOIN staff AS s
ON d.dep_head = s.staff_id JOIN location as l
ON d.dep_office = l.ID;
```

Learnings - Riddhi Bhatt

As simple as it may look, a database is more than just a group of tables. For a database to make sense there needs to be the right relationship set between the different tables and the interactions between them need to be very much kept in mind when designing a database, making the design of the database schema one of the most important and complex tasks. To decide whether something is a strong or a weak entity etc are decisions that shape how the database functions and hence need quite some thought process. It should always be kept in mind that in the future we may have to add some components to the database that we may not have thought of now/ don't exist now. For example, in a college if the college starts to take up counselling, we would have to account for it in the database or a new fest which would require loads of additional logistics. In that way we should aim to keep our database buildable.

Learnings - Urvi Vaidya

I have learned that designing the database is the most important part. It is necessary to first map out the relations between blocks of information before we even begin to start writing queries. Unlike programming in which we break the problem into blocks of code and solve the smaller bits which we later tie in together, while creating a database it's the other way around. We must first have a larger picture of the requirements and then simultaneously define the most basic relationships and tie them in such a way that it is easier to add more relationships later. I have really learned to look at data differently and more acutely identify possible relationships between two bits of data.

Bibliography

1) https://dev.mysql.com/doc/refman/8.0/en/create-table.html
2) https://www.sqlshack.com/
3) stackexchange