

# UNIVERSITY ORGANIZATION DATABASE DESIGN

A detailed design document for building a  
university organization database logic.

Dexter Astorga

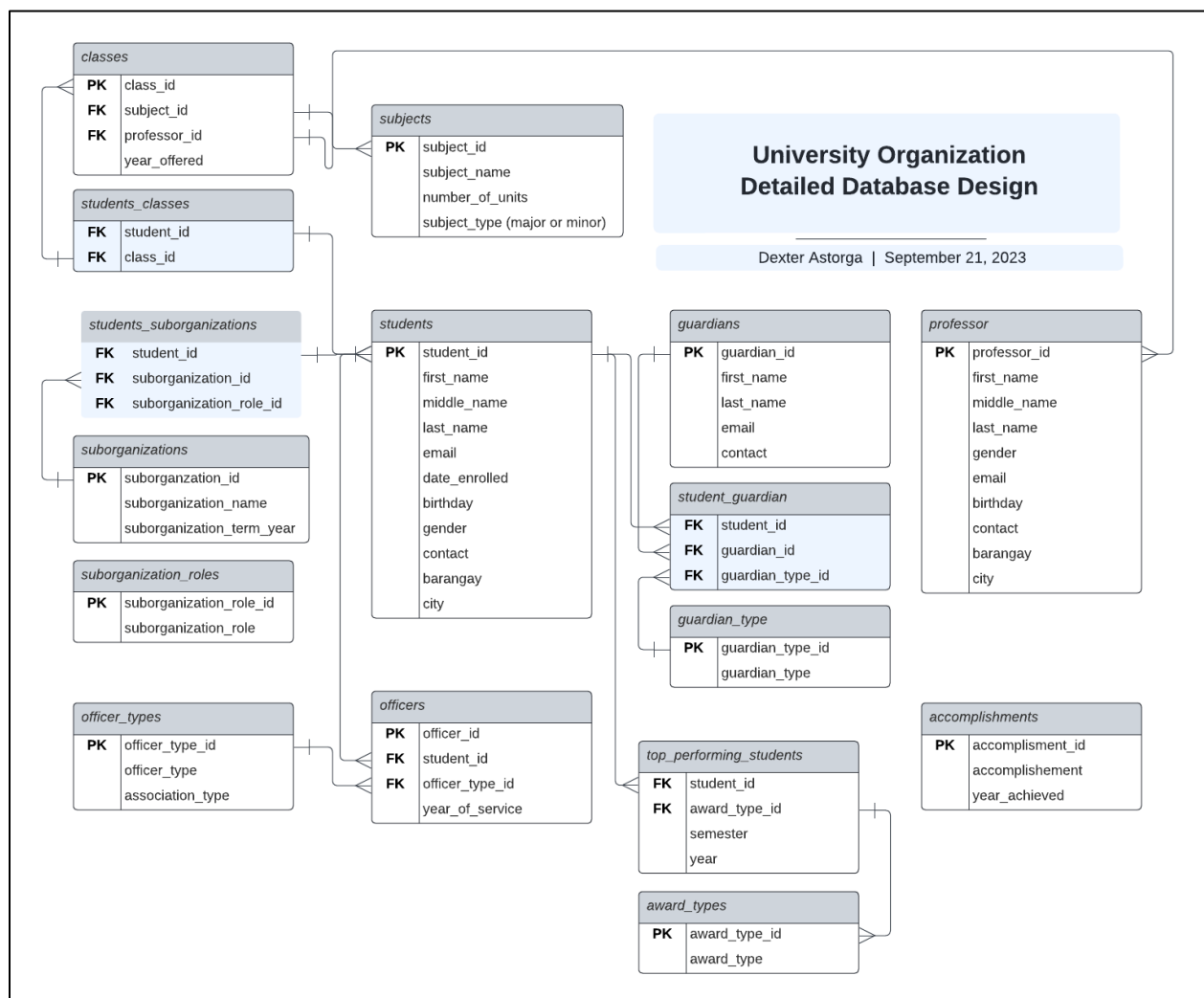
<dex>  
</dev>

## University Organization Database Design

This document contains:

- Entity-Relationship Diagram (ERD)
- Table Descriptions
- SQL Scripts (for creating the database tables)
- Example queries
- Conclusion

### ENTITY-RELATIONSHIP DIAGRAM



This diagram is available here: [https://github.com/dexdevtech/univ-org-db-design/blob/main/univ\\_org\\_ERD.png?raw=true](https://github.com/dexdevtech/univ-org-db-design/blob/main/univ_org_ERD.png?raw=true)

<https://github.com/dexdevtech/univ-org-db-design>



## TABLE DESCRIPTIONS

### students

Holds all students in the organization through the years.

column	description
student_id	Unique identifier for each student. This is assigned by the University.
first_name	First name of the student.
middle_name	Middle name of the student (Optional).
last_name	Last name of the student.
email	Email will be used to contact the student.
date_enrolled	When the student entered the organization.
birthday	Date of birth of the student.
gender	Gender.
contact_number	Will be used to contact the student (Secondary).
barangay	Barangay where the student lives.
city	City where the student lives. The database administrator can update this design by creating a separate table that will store the exact address of a student including the house number, street name, etc.

### guardians

Stores information about the student's guardian/s.

column	description
guardian_id	Unique identifier for each guardian. To make things easier, this id is also the student_id with -G at the end.
first_name	First name of the guardian.
last_name	Last name of the guardian.
email	Email will be used to contact the guardian (Secondary)..
contact_number	Will be used to contact the guardian.

### student\_guardian

It is a common convention in Relational Database Management Systems to create a joining tables for entities that have many-to-many relationships. In this case, a student can have multiple guardians and multiple students can have a single guardian.

column	description
student_id	Foreign key referencing the students table.
guardian_id	Foreign key referencing the guardians table.
guardian_type_id	Foreign key referencing the guardian_type table.



### guardian\_type

A look up table for all the guardian types.

column	description
guardian_type_id	Unique identifier for each guardian type.
guardian_type	Values of the guardian type.

### professors

Holds all professors in the organization through the years.

column	description
professor_id	Unique identifier for each professor.
first_name	First name of the professor.
middle_name	Middle name of the professor (Optional).
last_name	Last name of the professor.
gender	Gender.
email	Email will be used to contact the professor.
birthday	Date of birth of the student. Can be used to calculate age.
contact_number	Will be used to contact the professor (Secondary).
barangay	Barangay where the professor lives.
city	City where the professor lives. The database administrator can update this design by creating a separate table that will store the exact address for both students and professors.

### suborganizations

These are the small organizations within the organization. They could be a group of dancers, academic achievers, or athletes.

column	description
suborganization_id	Unique identifier for each suborganization.
suborganization_name	The name of the suborganizations.
suborganization_term_year	This is useful to track who were the members of each suborganization in each year passing.

### suborganization\_roles

This is useful to track who are the members and heads of each suborganization.

column	description
suborganization_role_id	Unique identifier for each suborganization role.
suborganization_role	The roles for the suborganization: chairman, co-chairman, members.



#### students\_suborganizations

This is a joining table. Students can be a part of at most 2 suborganizations (depending on your preference) and a suborganization can have as many members as possible.

column	description
student_id	Foreign key referencing the students table.
suborganization_id	Foreign key referencing the suborganizations table.
suborganization_role_id	Foreign key referencing the suborganization_roles table.

#### subjects

This table holds information about the subjects offered in the organization.

column	description
subject_id	Unique identifier for each subject.
subject_name	The name of the subject offered.
number_of_units	Subjects in a university usually has units.
subject_type	Major or minor.

#### classes

This table holds all the classes held in the organization. You can replace this table with a 'sections' table if the organization has block sections. In this case, it is taken into account the irregular students.

column	description
class_id	Unique identifier for each class.
subject_id	Foreign key referencing the subjects table.
professor_id	Foreign key referencing the professors table.
year_offered	Since the same subjects are taught every year, it is a great idea to distinguish them using years.

#### students\_classes

This table is a joining table that will tell us the students that are in a class or classes a student attends.

column	description
student_id	Foreign key referencing the students table.
class_id	Foreign key referencing the classes table.

<dex>  
</dev>

## University Organization Database Design



### officers

The officers of the organization, main org officers and class officers.

column	description
officer_id	Unique identifier for each officer.
student_id	Foreign key referencing the students table.
officer_type_id	Foreign key referencing the officer_types table.
year_of_service	This is to track each officer every school year.

### officer\_types

This is a look up table to list all officer types in the organization.

column	description
officer_type_id	Unique identifier for each officer type.
officer_type	President, vp...
association_type	This is added to distinguish main organization officers to class officers.

### top\_performing\_students

It is great to track student who excel in the organization.

column	description
student_id	Foreign key referencing the students table.
award_type_id	Foreign key referencing the award_types table.
semester	Every semester, there are students who receive awards so let's record all of them.
year	There are 2 semesters per year so it's advisable to add a year column for data management.

### award\_types

There are many categories for awards so a look up table is added.

column	description
award_type_id	Unique identifier for each officer type.
award_type	The type of award achieved.

### accomplishments

Holds the record of the achievements won by the organization.

column	description
accomplishment_id	Unique identifier for each accomplishment.
accomplishment	The accomplishment won.
year_achieved	This is useful to track which officers lead the achievement.



## SQL SCRIPTS (for creating the database tables)

```
CREATE DATABASE univ_org;
```

```
CREATE TABLE IF NOT EXISTS students(  
    student_id VARCHAR(100) NOT NULL PRIMARY KEY,  
    first_name VARCHAR(100) NOT NULL,  
    middle_name VARCHAR(100),  
    last_name VARCHAR(100) NOT NULL,  
    email VARCHAR(255),  
    date_enrolled DATE NOT NULL,  
    date_of_birth DATE NOT NULL,  
    gender VARCHAR(50) NOT NULL,  
    contact_number VARCHAR(50) NOT NULL,  
    barangay VARCHAR(100) NOT NULL,  
    city VARCHAR(100) NOT NULL  
);
```

```
CREATE TABLE IF NOT EXISTS guardians(  
    guardian_id VARCHAR(100) NOT NULL PRIMARY KEY,  
    first_name VARCHAR(100) NOT NULL,  
    last_name VARCHAR(100) NOT NULL,  
    email VARCHAR(255),  
    contact_number VARCHAR(50) NOT NULL  
);
```

```
CREATE TABLE IF NOT EXISTS guardian_type(  
    guardian_type_id BIGSERIAL NOT NULL PRIMARY KEY,
```

<dex>

</dev>



```
guardian_type VARCHAR(100)
```

```
);
```

```
CREATE TABLE IF NOT EXISTS student_guardian_relationship (  
    student_id VARCHAR(100) NOT NULL REFERENCES students (student_id),  
    guardian_id VARCHAR(100) NOT NULL REFERENCES guardians (guardian_id),  
    guardian_type_id BIGINT NOT NULL REFERENCES guardian_type (guardian_type_id)  
);
```

```
CREATE TABLE IF NOT EXISTS suborganizations(  
    suborganization_id BIGSERIAL PRIMARY KEY NOT NULL,  
    suborganization VARCHAR(50) NOT NULL,  
    suborganization_term_year DATE NOT NULL  
);
```

```
CREATE TABLE IF NOT EXISTS suborganization_roles(  
    suborganization_role_id BIGSERIAL PRIMARY KEY NOT NULL,  
    suborganization_role VARCHAR(50) NOT NULL  
);
```

```
CREATE TABLE IF NOT EXISTS students_suborganizations(  
    student_id VARCHAR(100) NOT NULL REFERENCES students(student_id),  
    suborganization_id BIGINT NOT NULL REFERENCES suborganizations(suborganization_id),  
    suborganization_role_id BIGINT NOT NULL REFERENCES  
suborganization_roles(suborganization_role_id)  
);
```

```
CREATE TABLE IF NOT EXISTS subjects(  

```



<dev>

</dev>



```
subject_id BIGSERIAL NOT NULL PRIMARY KEY,  
subject_name VARCHAR(255) NOT NULL,  
number_of_units INT NOT NULL,  
subject_type VARCHAR(50)  
);
```

```
CREATE TABLE IF NOT EXISTS professors(  
  professor_id VARCHAR(100) NOT NULL PRIMARY KEY,  
  first_name VARCHAR(100) NOT NULL,  
  middle_name VARCHAR(100),  
  last_name VARCHAR(100) NOT NULL,  
  email VARCHAR(255),  
  date_enrolled DATE NOT NULL,  
  date_of_birth DATE NOT NULL,  
  gender VARCHAR(50) NOT NULL,  
  contact_number VARCHAR(50) NOT NULL,  
  barangay VARCHAR(100) NOT NULL,  
  city VARCHAR(100) NOT NULL  
);
```


```
CREATE TABLE IF NOT EXISTS classes(  
  class_id BIGSERIAL NOT NULL PRIMARY KEY,  
  subject_id BIGINT NOT NULL REFERENCES subjects(subject_id),  
  professor_id VARCHAR(100) NOT NULL REFERENCES professors(professor_id),  
  year_offered DATE NOT NULL  
);
```

```
CREATE TABLE IF NOT EXISTS students_classes(  

```

<dev>

</dev>



```
student_id VARCHAR(100) NOT NULL REFERENCES students(student_id),  
class_id BIGINT NOT NULL REFERENCES classes(class_id)  
);
```

```
CREATE TABLE IF NOT EXISTS officer_types(  
    officer_type_id BIGSERIAL NOT NULL PRIMARY KEY,  
    officer_type VARCHAR(100) NOT NULL,  
    association_type VARCHAR(100) NOT NULL  
);
```

```
CREATE TABLE IF NOT EXISTS officers(  
    officer_id BIGSERIAL NOT NULL PRIMARY KEY,  
    student_id VARCHAR(100) NOT NULL REFERENCES students(student_id),  
    officer_type_id BIGINT NOT NULL REFERENCES officer_types(officer_type_id),  
    year_of_service DATE NOT NULL  
);
```

```
CREATE TABLE IF NOT EXISTS award_types(  
    award_type_id BIGSERIAL NOT NULL PRIMARY KEY,  
    award_type VARCHAR(100) NOT NULL  
);
```

```
CREATE TABLE IF NOT EXISTS top_performing_students(  
    student_id VARCHAR(100) NOT NULL REFERENCES students(student_id),  
    award_type_id BIGINT NOT NULL REFERENCES award_types(award_type_id),  
    semester INT NOT NULL,  
    year_received DATE NOT NULL  
);
```

<dev>

</dev>

University Organization Database Design



```
CREATE TABLE IF NOT EXISTS accomplishments(  
  accomplishment_id BIGSERIAL NOT NULL PRIMARY KEY,  
  accomplishment VARCHAR(255),  
  year_achieved DATE NOT NULL  
);
```

<dev>  
</dev>

## University Organization Database Design



### EXAMPLE QUERIES

Refer to this link to view the sample queries file.

[https://github.com/dexdevtech/univ-org-db-design/tree/main/sample\\_queries](https://github.com/dexdevtech/univ-org-db-design/tree/main/sample_queries)

### CONCLUSION

This database design is for university organizations. Every organization can leverage this project with just minor manipulations to fit their organization setting. I hope this project makes a great reference. If you have any questions or feature requests, you can contact me at [dexdevtech@gmail.com](mailto:dexdevtech@gmail.com).

Thank you!

Dexter Astorga