



900 San Marcelino Street, Ermita, Manila 1000

CPE419AL : DATABASE DESIGN AND DEVELOPMENT LAB

Database and Design Class Student Record Database

Group C

Group Members:

Herrera, Emmanuel Santillan

Maranan, Aron John M.

Pon-an, Jolyn D.

Reyes, Reinaliza P.

Engr. Baluyot, Raffaello Manalaysay

Instructor

Date of Submission: 10 December 2018

Title of Project: Database Design Class Student Record Database

I. Introduction

Recording data of a student in a class plays a key role in the management system in any school but the whole process of it adds heavy workload for the teachers. A database system of a class record is a highly desirable addition to the educational tool-kit particularly when it can provide less effort and a more effective and timely outcome.

The project titled “Database Design Class Record Database” is a class record management database system for monitoring and controlling informations in a class record. The project is developed in structured query language, which mainly focuses on basic operations like adding, reading, updating and removing students and their activities, attendance and merits. It is designed to help users maintain and organize a class record.

II. Objectives

- 2.1. To develop a Database Design Class Student Record applying Database Programming technique exemplified by MySQL application.
- 2.2. To create a structure and design to express the visualized concept with the help through software.
- 2.3. To create UML class diagram of the created database project to demonstrate its component and relationships.
- 2.4. To apply the lessons learned in this course, Database Design and Development.

III. Features

3.1. Seed Data

3.2. CRUD Functions:

- Student
- Attendance
- Activities
- Merits

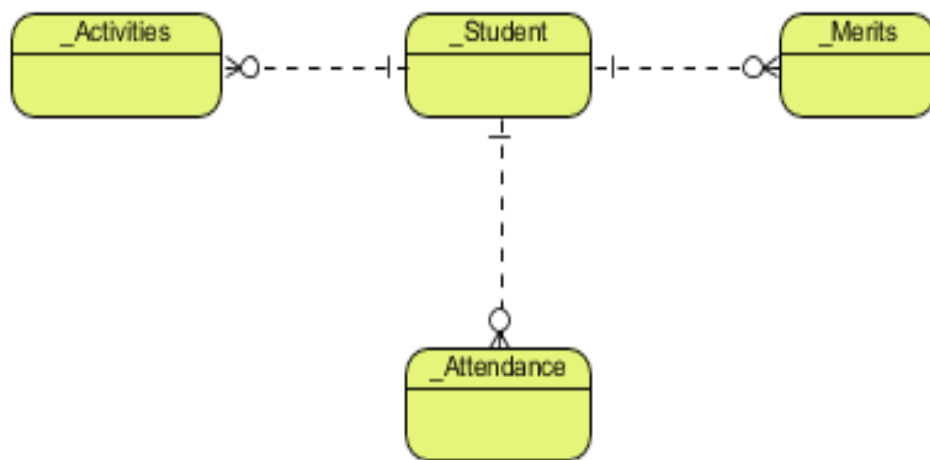
3.3. SQL Queries

- Tables
- Views
- Stored Procedures

IV. ERD Diagrams

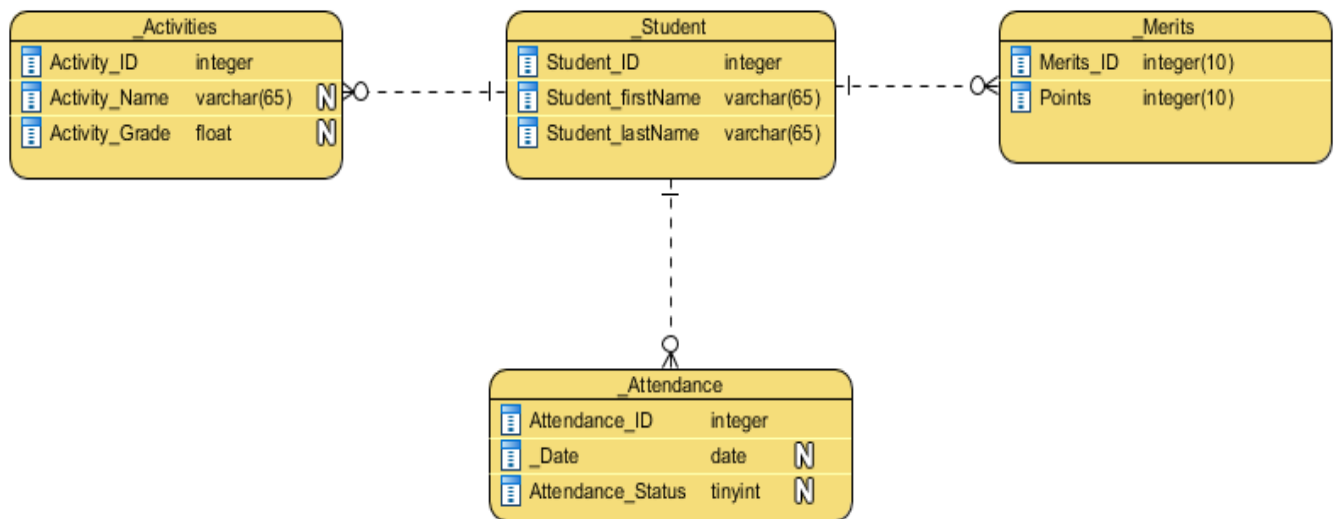
Conceptual ERD:

Conceptual ERD is the simplest type among the three ERDs. This model contains a kind of relationship between entities. In this Conceptual ERD, Student Table has one-to-many relationship with Activities Table, Merits Table, and Attendance Table.



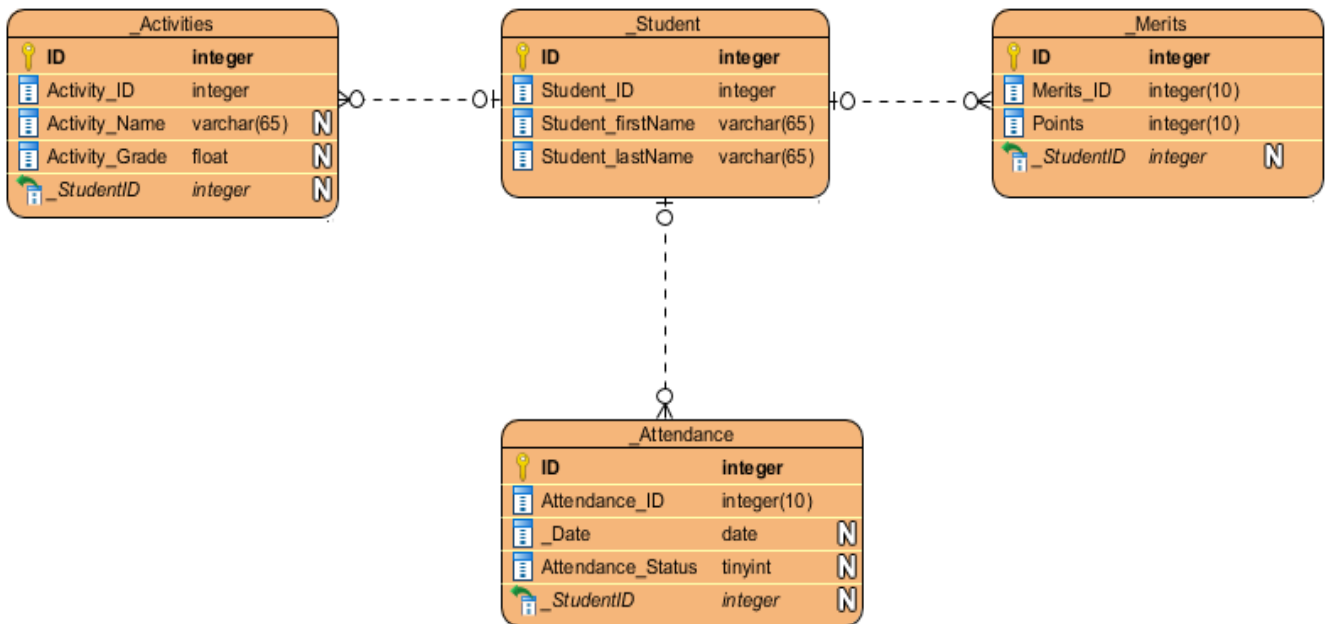
Logical ERD:

Logical ERD is more complex than Conceptual ERD since it contains the columns and its datatypes along with the relationships of the tables. Setting the datatypes for this model is optional. In this Logical ERD, each table contains its own columns with its respective datatypes.



Physical ERD:

Physical ERD represents the actual design blueprint of a relational database. Accurate datatypes, constraints, primary keys and foreign keys are added to the tables. It is important to add accurate features because it represents the data that should be structured and are related in a specific database management system. In this Physical ERD, the IDs represents the primary key while the foreign keys are _StudentID. The foreign key is a column for Merits, Activities and Attendance tables that references the primary key column in Student table.



V. Stored Procedures

1. Get the students id with the maximum and minimum absences

```
DELIMITER $$
• CREATE PROCEDURE complex01() ← Procedure Name
  BEGIN
    SELECT _Student.Student_ID, SUM(_Attendance.Attendance_Status) AS Present, COUNT(_Date) - SUM(_Attendance.Attendance_Status) AS Absent
    FROM _Student, _Attendance
    WHERE _Student.Student_ID = _Attendance.Student_ID
    GROUP BY Student_ID
    HAVING
      SUM(_Attendance.Attendance_Status) = (
        SELECT MAX(_SUM)
        FROM (
          SELECT _Student.Student_ID, SUM(_Attendance.Attendance_Status) AS _SUM
          FROM _Student, _Attendance
          WHERE _Student.Student_ID = _Attendance.Student_ID
          GROUP BY Student_ID) T, _Student) OR
      SUM(_Attendance.Attendance_Status) = (
        SELECT MIN(_SUM)
        FROM (
          SELECT _Student.Student_ID, SUM(_Attendance.Attendance_Status) AS _SUM
          FROM _Student, _Attendance
          WHERE _Student.Student_ID = _Attendance.Student_ID
          GROUP BY Student_ID) T, _Attendance);
  END $$
DELIMITER ;
• CALL complex01();
```

Query Returned

	Student_ID	Present	Absent
▶	1	19	0
	6	19	0
	7	0	19
	9	0	19
	29	19	0

2. Get the students whose total merit score is more than the average merit score of students.

```

DELIMITER $$
CREATE PROCEDURE complex02() ← Procedure Name
BEGIN
    SELECT _Merits.Student_ID, Student_firstname, Points
    FROM _Merits, _Student
    WHERE Points > (SELECT AVG(Points)
    FROM _Merits) AND _Merits.Student_ID = _Student.Student_ID;
END $$
DELIMITER ;
CALL complex02();

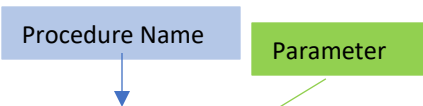
```

Query Returned

	Student_ID	Student_firstname	Points
►	51	Sofia	50
	52	Jaxon	50
	53	Josiah	50
	54	John	50
	55	Scarlett	50
	56	Luke	50
	57	Aria	50
	58	Ryan	50
	59	Elizabeth	50
	60	Camila	50
	61	Nathan	60
	62	Layla	60
	63	Isaac	60
	64	Owen	60
	65	Ella	60
	66	Henry	60
	67	Levi	60
	68	Aaron	60
	69	Caleb	60
	70	Chloe	60
	71	Zoey	70
	72	Jeremiah	70
	73	Lincoln	70

	Student_ID	Student_firstname	Points
	74	Landon	70
	75	Adrian	70
	76	Hunter	70
	77	Eli	70
	78	Penelope	70
	79	Skylar	70
	80	Jonathan	70
	81	Thomas	80
	82	Jack	80
	83	Jordan	80
	84	Connor	80
	85	Brayden	80
	86	Cameron	80
	87	Grace	80
	88	Bryson	80
	89	Mila	80
	90	Lillian	80
	91	Aaliyah	90
	92	Jose	90
	93	Lily	90
	94	Paisley	90
	95	Xavier	90
	96	Dominic	90
	97	Bella	90
	98	Nicholas	90
	99	Brooklyn	90
	100	Savannah	90

3. Get the students whose percentage activities grade is greater than params.



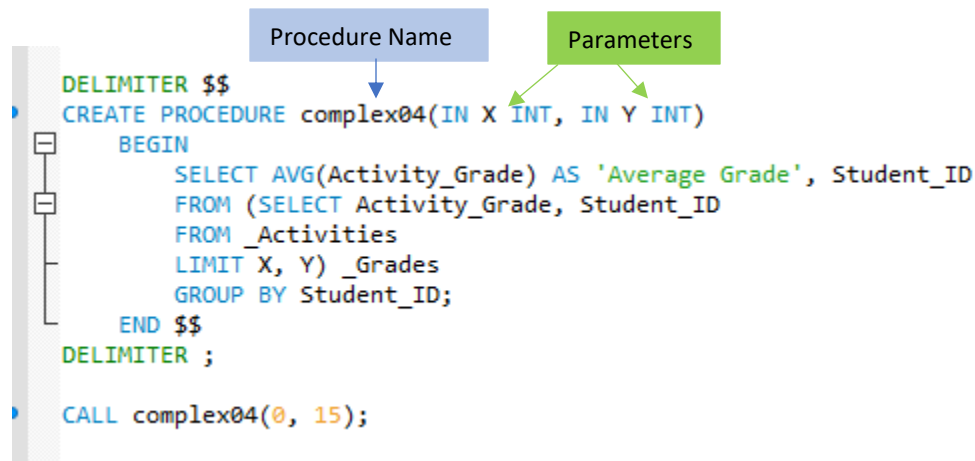
```
DELIMITER $$
CREATE PROCEDURE complex03(IN X INT)
BEGIN
    SELECT Student_ID, Activity_Grade
    FROM _Activities
    HAVING Activity_Grade > X;
END $$
DELIMITER ;

CALL complex03(0);
```

Query Returned

	Student_ID	Activity_Grade
▶	1	100
	1	90
	1	40
	1	30
	1	20
	2	80
	2	70
	2	60
	2	40
	2	30
	2	20
	3	60
	3	50
	3	40
	3	30
	3	20

4. Get the average score of the students in their activities given on start_param and end_param



```
DELIMITER $$
CREATE PROCEDURE complex04(IN X INT, IN Y INT)
BEGIN
    SELECT AVG(Activity_Grade) AS 'Average Grade', Student_ID
    FROM (SELECT Activity_Grade, Student_ID
    FROM _Activities
    LIMIT X, Y) _Grades
    GROUP BY Student_ID;
END $$
DELIMITER ;

CALL complex04(0, 15);
```

Query Returned:

	Average Grade	Student_ID
▶	56	1
	50	2
	45	3

5. Give the student and activity name whose percentage score is less than the average activity percentage score.

```
SELECT Activity_Name, Activity_Grade, Student_ID
From _Activities
WHERE Activity_Grade < (SELECT AVG(Activity_Grade)
FROM _Activities);

SELECT AVG(Activity_Grade)
FROM _Activities;
```

Query Returned:

	Activity_Name	Activity_Grade	Student_ID
▶	Activity 03	40	1
	Activity 04	30	1
	Activity 05	20	1
	Activity 03	40	2
	Activity 04	30	2
	Activity 05	20	2
	Activity 03	40	3
	Activity 04	30	3
	Activity 05	20	3

	AVG(Activity_Grade)
▶	48.75