Final Project/Exam: DBMS for Sustainable Development

Course Name: Database Management System 1
Program/Year Level: BSIT, 2nd Year
Semester/Term: 1st Semester

- **Introduction**

1.1.The student tracker database is a system that helps manage and records student information in an organized way. It makes tracking student details, grades, and students that are enrolled easier for school. This project supports UN SDG 4: Quality education, because it helps schools to monitor student progress and be able to support them. Overall, it aims to make school record keeping easier.

1.2. Many schools continue to struggle with ineffective and unorganized record keeping systems, resulting in slow data retrieval, difficulty in monitoring student progress, and challenges in making timely and informed administrative decisions. Despite the vital role that accurate and well managed records play in ensuring efficient school operations, some institutions still rely on outdated, inconsistent, or poorly maintained methods of documentation. This leads to problems such as data loss, limited accessibility, compromised transparency, and inefficient information flow. (Akinloye, Adu, & Ojo, 2017).

Reference: Record Keeping Management Practices and Legal Issues in the School System

II. Requirements and analysis

2.1. Functional Requirements:

- **FR1: Student Management: The system must be able to store and retrieve student personal information (Name, DOB, Gender, Email, Enrollment Status).**
- **FR2: Teacher Management: The system must be able to store and retrieve teacher information (Name, Expertise, Qualification, Years of Experience).**
- **FR3: Subject/Course Management: The system must manage subject details (Subject Code, Name, Credit Units, Prerequisite).**
- **FR4: Course Offering: The system must record instances of subjects being offered in a specific semester and academic year, linking them to a Teacher.**
- **FR5: Enrollment: The system must allow students to enroll in course offerings, tracking the enrollment date, status, and final grade.**
- **FR6: Assessment: The system must manage assessments for each course offering, including type, title, maximum score, weight, and due date.**

- FR7: Grading: The system must allow recording of student scores for specific assessments, including the achieved score, submission date, and remarks.

- FR8: Performance Summary: The system must generate a summary of student performance per course, including the calculated overall percentage score and final grade (View: `course_performance_summary`).

  Non Functional Requirements:

- NFR1: Integrity (Data Constraints): Data must adhere to integrity rules, such as ensuring non-negative years of experience for teachers, positive maximum assessment scores, and assessment weights between 0 and 100 (enforced via `CHECK` constraints).
- NFR2: Consistency (Keys and Relationships): Data relationships must be consistent, enforced by Primary Keys and Foreign Keys that maintain referential integrity (e.g., a grade must relate to an existing student and assessment).

- NFR3: Uniqueness: Certain key data points must be unique, such as a student's contact email or the combination of subject, teacher, semester, and year for a course offering.

- NFR4: Performance: The use of Indexes on primary and foreign keys suggests an emphasis on efficient data retrieval and relationship lookups.

2.2. Data Requirements

Table	Data Structure (Key Attributes)	Data Size (Number of Rows)
student	Student_ID (PK), First_Name, Last_Name, Contact_Email (Unique)	50 rows
teacher	Teacher_ID (PK), First_Name, Last_Name, Expertise_Area	5 rows

subject	Subject_Code (PK), Subject_Name (Unique), Credit_Units, Prerequisite_Subject (FK)	5 rows
course_offering	Offering_ID (PK), Subject_Code (FK), Teacher_ID (FK), Semester, Academic_Year	5 rows
enrollment	Enrollment_ID (PK), Student_ID (FK), Offering_ID (FK), Final_Grade	50 rows
assessment	Assessment_ID (PK), Offering_ID (FK), Assessment_Type, Title, Max_Score, Weight	50 rows
grade	Assessment_ID (PK, FK), Student_ID (PK, FK), Score_Achieved, Submission_Date	50 rows

2.3. Schema Normalization Analysis

1. **enrollment Table**

- **Attributes: Enrollment_ID, Student_ID, Offering_ID, Enrollment_Date, Final_Grade, Enrollment_Status.**
- **Primary Key: Enrollment_ID.**
- **Candidate Key: The composite key (Student_ID, Offering_ID) is a unique identifier, making it a candidate key.**
- **Functional Dependencies (FDs):**
  - **$Enrollment\_ID \rightarrow Student\_ID, Offering\_ID, Enrollment\_Date, Final\_Grade, Enrollment\_Status$ (Trivial from PK)**
  - **$Student\_ID, Offering\_ID \rightarrow Enrollment\_ID, Enrollment\_Date, Final\_Grade, Enrollment\_Status$ (From Candidate Key)**
- **Normalization Justification:**
  - **1NF & 2NF: Passed, as there are no repeating groups or partial dependencies (all non-key attributes depend on the full composite candidate key).**

  - 3NF: Passed, as there are no transitive dependencies (i.e., no non-key attribute depends on another non-key attribute). For example, Final_Grade does not determine Enrollment_Status.
  - Conclusion: The enrollment table is in 3NF. Since every determinant is a candidate key, it is also in BCNF.

2.assessment Table

- Attributes: Assessment_ID, Offering_ID, Assessment_Type, Title, Max_Score, Weight, Due_Date.
- Primary Key: Assessment_ID.

- Functional Dependencies (FDs):

Attribute Name	Data Type	Description
Assessment_ID	int	Primary Key. Unique identifier for the assessment.
Offering_ID	int	Foreign Key to course_offering. The class this assessment belongs to.
Assessment_Type	varchar	Type of assessment (e.g., 'Exam').
Title	varchar	Name or description of the assessment (e.g., 'Midterm Exam').
Max_Score	decimal	The maximum possible score.

Weight	decimal	The percentage weight of this assessment in the final grade.
Due_Date	date	The date the assessment is due.

- **Normalization Justification:**
    - **1NF & 2NF: Passed.**
    - **3NF/BCNF: Passed. All attributes are fully dependent on the primary key, Assessment_ID. No non-key attributes determine other non-key attributes. For instance, Max_Score is for a specific assessment and doesn't determine the Assessment_Type.**
    - **Conclusion: The assessment table is in 3NF (and BCNF).**

3. GRADE TABLE

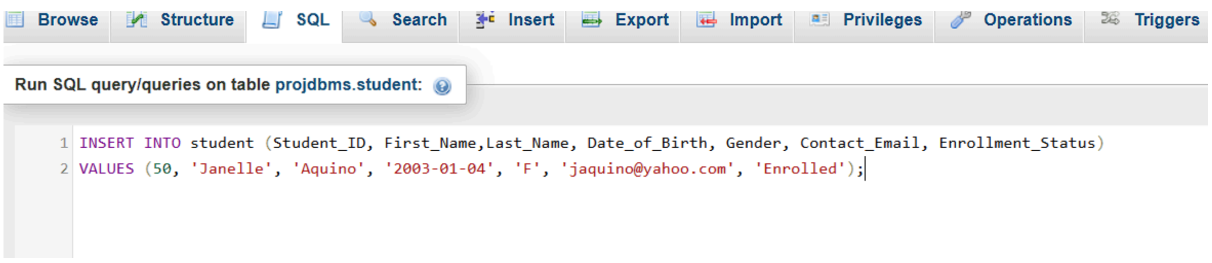
- **Attributes: Assessment_ID, Student_ID, Score_Achieved, Submission_Date, Remarks.**
- **Primary Key: The composite key (Assessment_ID, Student_ID).**
- **Functional Dependencies (FDs):**
    - **$Assessment\_ID, Student\_ID \rightarrow Score\_Achieved, Submission\_Date, Remarks$ (Trivial from PK)**
- **Normalization Justification:**
    - **1NF & 2NF: Passed. The non-key attributes (Score_Achieved, Submission_Date, Remarks) are fully dependent on the entire composite key (Assessment_ID, Student_ID). A student's score is meaningless without knowing which assessment it belongs to.**
    - **3NF/BCNF: Passed, as there are no transitive dependencies.**
    - **Conclusion: The GRADE table is in 3NF (and BCNF).**

III. Design Specification

3.1. Core DBMS Concepts Used

DML (Prelims)

1. **INSERT: The command was used to add new records into the tables in our database. INSERT is essential to the database because it is responsible for populating the tables.**

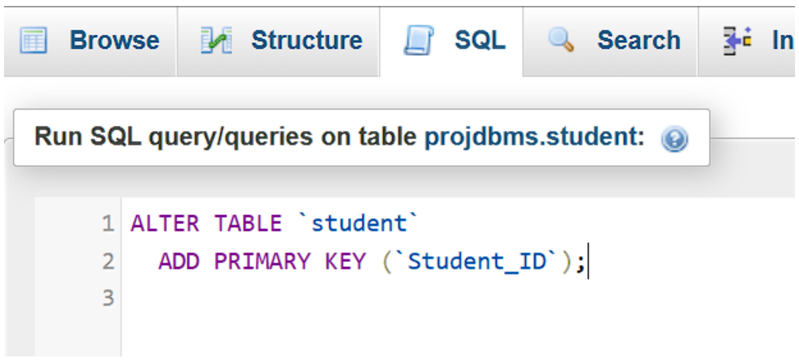


Run SQL query/queries on table projdbms.student:

```sql
INSERT INTO student (Student_ID, First_Name,Last_Name, Date_of_Birth, Gender, Contact_Email, Enrollment_Status)
VALUES (50, 'Janelle', 'Aquino', '2003-01-04', 'F', 'jaquino@yahoo.com', 'Enrolled');
```

CONSTRAINTS (Midterm)

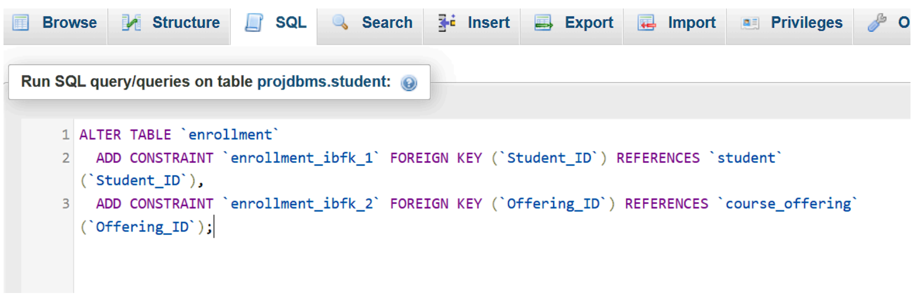
1. **Primary Key (PK): used to make sure that every time a data was recorded to a table, it is unique and will avoid repetition of PK. One example is the student, every student must only have a Student ID that's unique to avoid repetition of the Student ID. This will ensure that no Student ID will cause an error when recording**



Run SQL query/queries on table projdbms.student:

```sql
ALTER TABLE `student`
  ADD PRIMARY KEY (`Student_ID`);
```

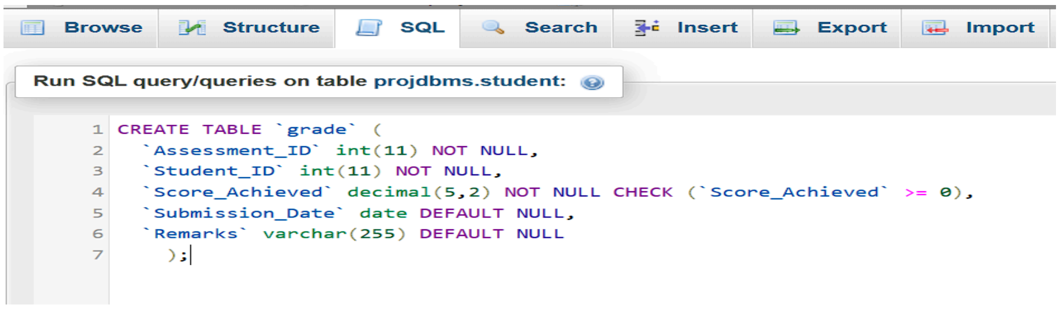
2. **Foreign Key (FK): Used to connect related tables and to maintain proper relationships between tables. For example, the enrollment table is connecting the students to the course offerings table, it is so that only existing students are the ones that are entitled for the course offerings.**



Run SQL query/queries on table projdbms.student:

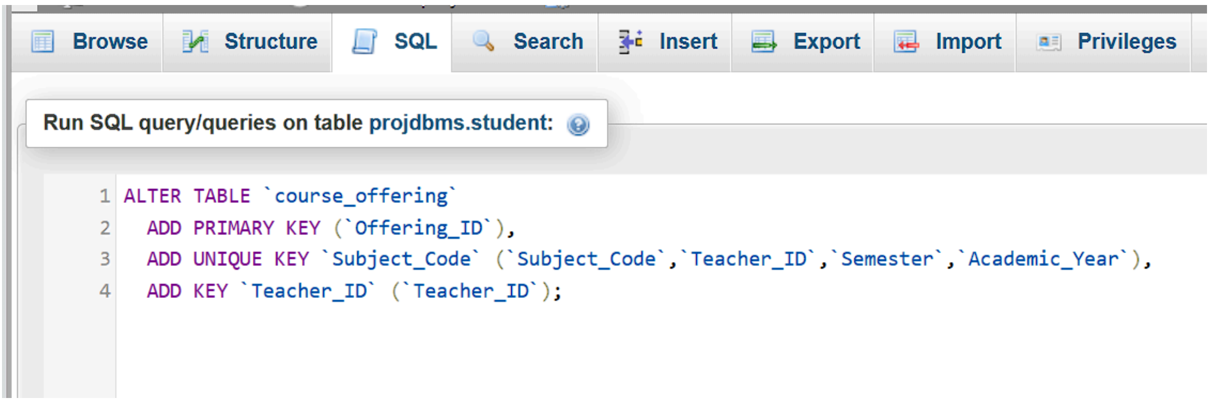
```sql
ALTER TABLE `enrollment`
  ADD CONSTRAINT `enrollment_ibfk_1` FOREIGN KEY (`Student_ID`) REFERENCES `student`
(`Student_ID`),
  ADD CONSTRAINT `enrollment_ibfk_2` FOREIGN KEY (`Offering_ID`) REFERENCES `course_offering`
(`Offering_ID`);
```

3. **CHECK: it is used to make sure that only valid values will be stored inside the database. This will prevent any invalid values like negative score or credit units.**



```
CREATE TABLE `grade` (
  `Assessment_ID` int(11) NOT NULL,
  `Student_ID` int(11) NOT NULL,
  `Score_Achieved` decimal(5,2) NOT NULL CHECK (`Score_Achieved` >= 0),
  `Submission_Date` date DEFAULT NULL,
  `Remarks` varchar(255) DEFAULT NULL
    );
```

4. **UNIQUE: We used UNIQUE to ensure that some values like**



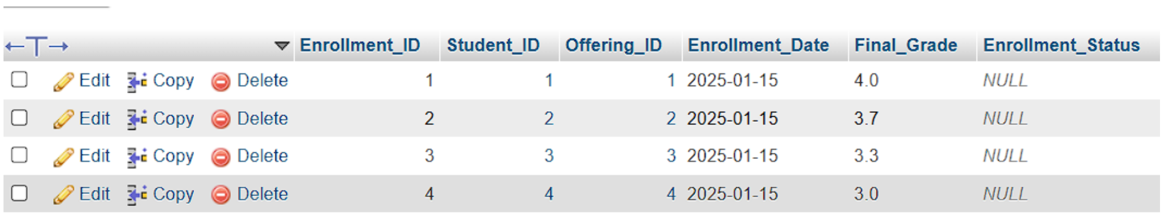
```
ALTER TABLE `course_offering`
  ADD PRIMARY KEY (`Offering_ID`),
  ADD UNIQUE KEY `Subject_Code` (`Subject_Code`,`Teacher_ID`,`Semester`,`Academic_Year`),
  ADD KEY `Teacher_ID` (`Teacher_ID`);
```

Subject_code do not repeat inside the database. This will avoid duplication and inconsistency in the records.

NORMALIZATION (Finals)

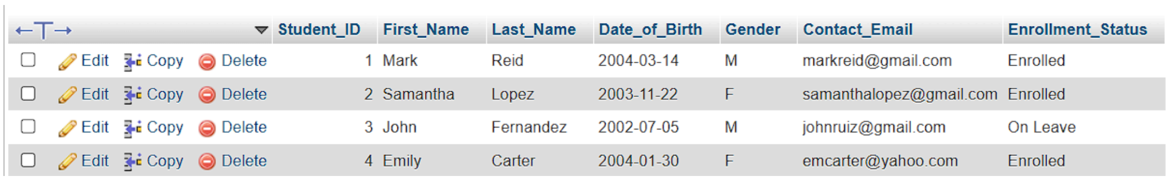
1. **3NF: The separation of entities (Student, Subject, Offering) demonstrates 3NF implementation. For example, student details such as First_Name, Last_Name, and Contact_Email are placed exclusively in the student table, using Student_ID as the Primary Key. The enrollment table only stores the Student_ID Foreign Key, along with attributes dependent solely on the enrollment itself, such as the Enrollment_Date and Final_Grade. This structure prevents the redundant storage of student information for every course they take.**

- **Enrollment table**



Enrollment_ID	Student_ID	Offering_ID	Enrollment_Date	Final_Grade	Enrollment_Status
1	1	1	2025-01-15	4.0	NULL
2	2	2	2025-01-15	3.7	NULL
3	3	3	2025-01-15	3.3	NULL
4	4	4	2025-01-15	3.0	NULL

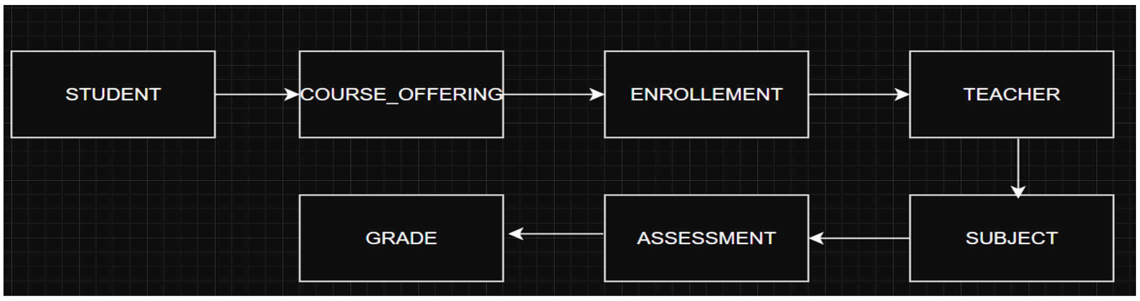
- **Student table**



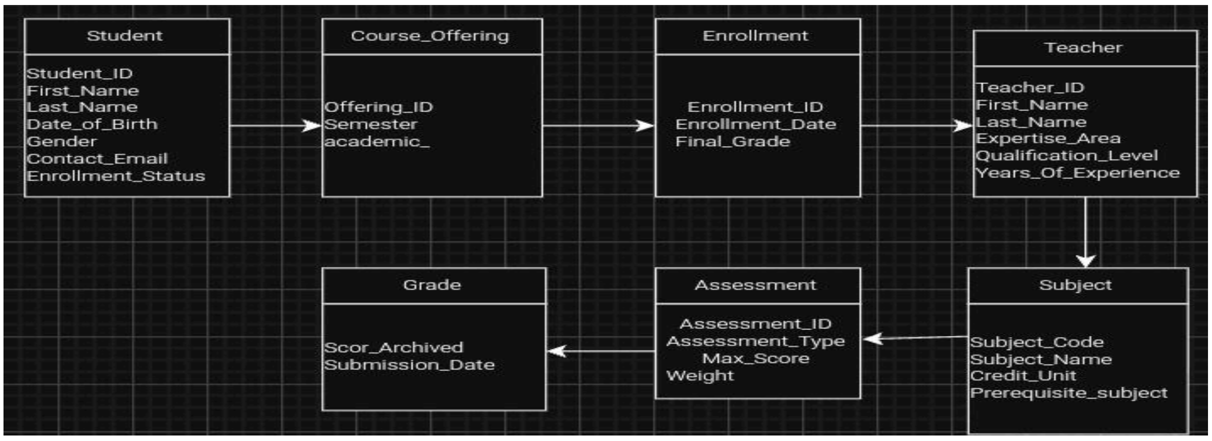
Student_ID	First_Name	Last_Name	Date_of_Birth	Gender	Contact_Email	Enrollment_Status
1	Mark	Reid	2004-03-14	M	markreid@gmail.com	Enrolled
2	Samantha	Lopez	2003-11-22	F	samanthalopez@gmail.com	Enrolled
3	John	Fernandez	2002-07-05	M	johnruiz@gmail.com	On Leave
4	Emily	Carter	2004-01-30	F	emcarter@yahoo.com	Enrolled

3.2. ER DIAGRAM

- **Conceptual: The conceptual model shows how the Student tracker works without any technical details. It shows only the main parts of the system, like students, teachers, subjects, course offerings, enrollment, and grades. The purpose of this is to clearly show what information does the system need to be stored.**



- **Logical: The logical model shows what data each table should have. The student table stores the information of a student. The teacher table has the teachers information stored. The subject stores the lists of the subjects that are offered by school. This gives us more details about the tables and what are being stored.**



Detailed Description of the Entity Relationship Diagram:

The provided diagram visualizes the database structure, showing the following entities and relationships:

- **Core Entities: STUDENT, TEACHER, and SUBJECT are independent master entities.**
    - **STUDENT attributes include Student_ID (PK) and personal details.**
    - **TEACHER attributes include Teacher_ID (PK) and professional details like Expertise_Area and Years_of_Experience.**
    - **SUBJECT attributes include Subject_Code (PK) and Credit_Units.**
- **One-to-Many (1:M) Relationships:**
    - **TEACHER ←→ COURSE_OFFERING (1:M): A single teacher can teach multiple course offerings. Teacher_ID is an FK in the COURSE_OFFERING table.**
    - **SUBJECT ←→ COURSE_OFFERING (1:M): A single subject can be offered multiple times (different semesters, years, or teachers). Subject_Code is an FK in COURSE_OFFERING.**
    - **COURSE_OFFERING ←→ ASSESSMENT (1:M): Each specific course offering has many assessments (exams, quizzes). Offering_ID is an FK in the ASSESSMENT table.**
    - **Self-Referencing Relationship: The SUBJECT table has a recursive relationship where Prerequisite_Subject (FK) references its own Subject_Code (PK).**

- **Many-to-Many (M:N) Relationships (Resolved by Associative Tables):**
    - **STUDENT ←→ COURSE_OFFERING (M:N): This is resolved by the ENROLLMENT table. The ENROLLMENT table links Student_ID and Offering_ID and stores specific attributes of the enrollment relationship, like Enrollment_Date and Final_Grade.**
    - **STUDENT ←→ ASSESSMENT (M:N): This is resolved by the GRADE table. The GRADE table links Student_ID and Assessment_ID and stores the specific result, Score_Achieved.**

3. Data Integrity Constraints (Uniqueness and Domain)

Constraints are enforced to ensure the data is accurate and valid:

- **Primary Keys (PK): Defined on all entities (e.g., Assessment_ID, Enrollment_ID).**
- **Composite Primary Keys: Used in associative tables to ensure unique combination of values. For example, the PK for the grade table is a combination of (Assessment_ID, Student_ID), meaning a student can only have one score for a given assessment.**
- **Unique Constraints: The system enforces uniqueness on non-key attributes, such as student.Contact_Email.**

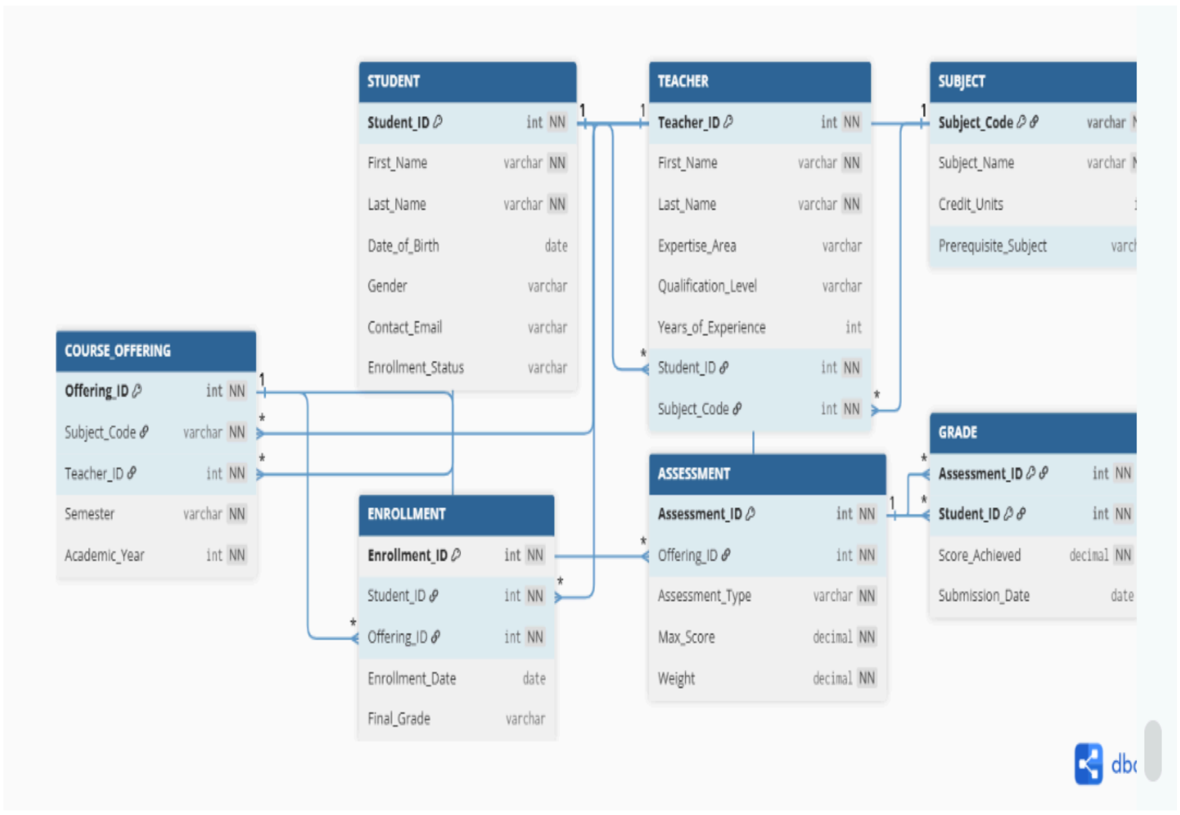
- **Composite Unique Constraints: The enrollment table uses a unique constraint on (Student_ID, Offering_ID), preventing a student from enrolling in the same course offering more than once. The course_offering table uses a unique constraint on (Subject_Code, Teacher_ID, Semester, Academic_Year).**
- **Check Constraints: Enforce domain integrity by restricting the possible values for an attribute:**
  - **assessment: Max_Score > 0 and Weight between 0 and 100.**
  - **grade: Score_Achieved must be >= 0.**

4. Joins and Views (Data Retrieval)

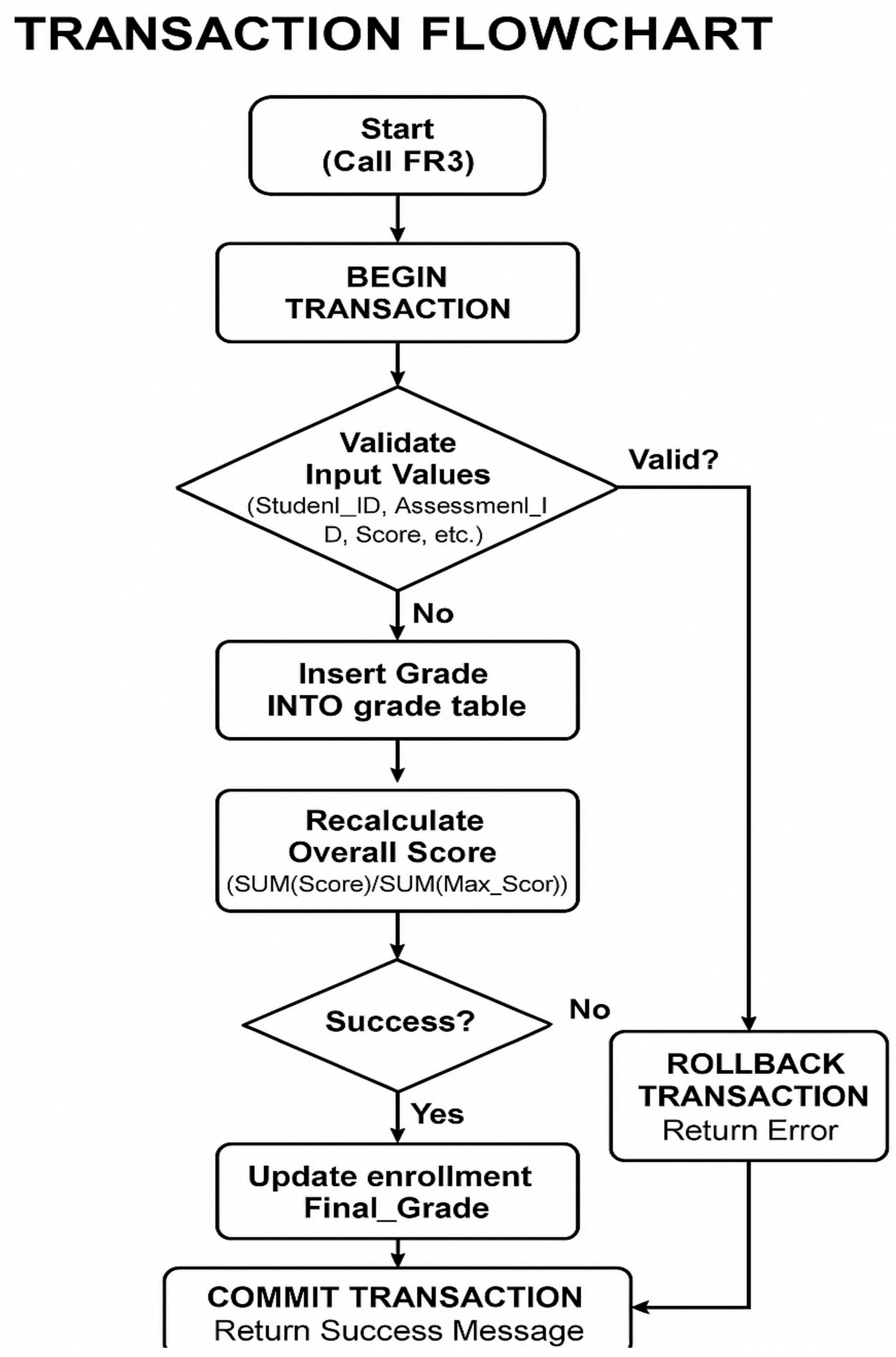
The course_performance_summary is a View that serves as a virtual table, simplifying complex data retrieval.

- **Joins: The View uses multiple JOIN operations (including INNER JOIN in the definition) to combine data from five normalized tables (enrollment, student, course_offering, subject, teacher).**
- **Aggregation and Subqueries: The View's definition includes a Subquery and aggregate functions (SUM) to calculate the Overall_Percentage_Score, demonstrating complex aggregation across the grade and assessment tables.**

ENTITY RELATIONSHIP DIAGRAM:



3.3. Flowchart:



IV. Conclusion and Contributions

- **Conclusion**

The Student Tracker Database project helped us understand how important a well organized and properly designed database system is for a school. Many schools still face problems with messy and outdated record keeping methods, so creating a system that can store student, teacher, subject, enrollment, and grading information in a clear and efficient way can really make a big difference. By building this database, we were able to address issues like slow data retrieval, lost records, and difficulty in monitoring student performance. Throughout the development process, we applied different DBMS concepts such as primary keys, foreign keys, UNIQUE and CHECK constraints, normalization up to 3NF, and the use of joins and views. These helped ensure that the data remains accurate, consistent, and easy to manage. We also created an ER diagram that clearly shows how each entity is connected, helping us better understand the flow of information within the system.

This project not only allowed us to design a functional and organized database but also showed us how data management contributes to better school operations and supports Quality Education (UN SDG 4). As students, this experience helped us appreciate how proper database design improves decision-making, transparency, and student progress tracking. The Student Tracker Database serves as a strong foundation that can be improved further and used in real school settings to make record-keeping faster, easier, and more reliable.

GITHUB:
https://github.com/Kisatsune/Final-Project-Exam-DBMS-for-Sustainable-Development-sdg11-dbms.git

- **Contribution**

PAPER WORKS, FLOWCHART, SQL
Hans Kenneth Dela Cruz

PAPER WORKS, FLOWCHART, DIAGRAM, SQL
Christian Camano

PAPER WORKS, SQL
Harvey Begonte

PAPERWORK
Josh Llames

SQL
Jacob Larman