National Teachers College
629 J Nepomuceno, Quiapo, Manila, 1001 Metro Manila
Bachelor of Science in Information Technology

Employment Monitoring and Job Tracking System for SDG 8.5

A Final Project
Presented to the Faculty of the
College of Information Technology

In partial fulfillment

of the Course Requirements for the degree

of Bachelor of Science in Information Technology

Submitted by:

Jezren R. Atienza
Rimuel D. Loking
Jamel P. Macadaub
Patrick Jay M. Canlas
John Rafael D. Sullera
Gregg Martin D. Cometa

BSIT 2.4
National Teachers College

Professor: *Justin Louise R. Neypes*

December 15, 2025

TABLE OF CONTENTS

I. INTRODUCTION

1.1 Project Overview & UN SDG Target

This project presents the design and development of a relational SQL database system that supports the monitoring and achievement of United Nations Sustainable Development Goal 8, specifically Target 8.5, which aims to ensure “full and productive employment and decent work for all women and men, including young people and persons with disabilities”.

Our system will provide a centralized platform for capturing, validating, and analyzing employment-related data which includes Peoples profile, Training programs and enrollment, Job postings, Job applications, and Training feedback. The database enables organizations and agencies to track employment status and participation in training and monitor unemployed youth and other vulnerable groups. It also matches qualified individuals to job opportunities, analyzes outcomes of skills development programs, and produces data-driven insights for labor policies

Using SQL queries, constraints, stored procedures, and an ACID-compliant transaction workflow, the system ensures data integrity and improves evidence-based reporting for SDG 8.5.

1.2 Problem Statement

Labor agencies often struggle with incomplete, outdated, or scattered employment data. This leads to difficulties such as Inaccurate monitoring of unemployment trends, limited evaluation of training effectiveness, lack of visibility of skill gaps, challenges supporting unemployed youth and vulnerable groups, Inefficiencies in job matching, and limited insights for policymaking

To address this issue, the project proposes a centralized SQL-based database system that integrates information on individuals, training programs, and job applications. By enforcing data integrity rules and secure transaction workflows, the system improves the accuracy of employment monitoring and supports skills-based job matching to contribute to the achievement of SDG 8.5.

II. REQUIREMENTS & ANALYSIS

2.1 Functional Requirements and Non-Functional Requirements

Functional Requirements (FR)

ID	Title	Description
FR1	**User Registration and Profiling**	The system must store individual records in the People table, ensuring that each person is uniquely identified with valid demographic and skills information.
FR2	**Training Enrollment Management**	The system must allow individuals to enroll in training programs and track enrollment status as Enrolled, Completed, or Dropped.
FR3	**Job Application Processing**	The system must allow individuals to apply for job postings and maintain accurate records by linking applicants to specific jobs.
FR4	**Training Feedback Submission**	Individuals who complete training programs must be able to submit feedback. This rule is enforced through application logic and validated against completed training records.

Non-Functional Requirements (NFR)

ID	Title	Description
NFR1	**Data Integrity**	The system must enforce data validity using PRIMARY KEY, FOREIGN KEY, UNIQUE, and CHECK constraints.
NFR2	**ACID Compliance**	The system must support atomic, consistent, isolated, and durable transactions, particularly for job application processing.
NFR3	**Performance**	The database must efficiently handle datasets of at least 50 records each table for queries and reports through the use of indexed primary keys, indexed foreign keys, and optimized join operations.
NFR4	**Maintainability**	The database schema must be normalized, clear, and easy to maintain.

2.2 Data Requirements

Input Data
The system supports initial data loading of at least 50 records across all tables for demonstration and testing purposes. Sample data includes individuals, training programs, job postings, enrollments, job applications, and feedback.

Data Model
The database consists of six main tables with defined structures and constraints:

1. **People**
   PersonID: INT, PRIMARY KEY, auto-increment
   FullName: VARCHAR(150), NOT NULL
   Age: INT, NOT NULL, CHECK (AGE BETWEEN 18 AND 60)
   Status: VARCHAR(50), NOT NULL, CHECK (Status IN ('Unemployed', 'Employed', 'In Training', 'Trained'))
   Skills: VARCHAR(255), optional
   RegistrationDate: DATE, NOT NULL

2. **Jobs**
   JobID: INT, PRIMARY KEY, auto-increment
   JobTitle: VARCHAR(150), NOT NULL
   CompanyName: VARCHAR(150), NOT NULL
   Location: VARCHAR(150), optional
   Salary: DECIMAL(10,2), optional
   JobType: VARCHAR(100), optional

3. **TrainingPrograms**
   ProgramID: INT, PRIMARY KEY, auto-increment
   ProgramName: VARCHAR(150), NOT NULL
   SkillTaught: VARCHAR(100), NOT NULL
   DurationWeeks: INT, NOT NULL, CHECK (DurationWeeks>0)
   AvailableTo: VARCHAR(100), NOT NULL
   JobID: INT, optional, FOREIGN KEY references Jobs(JobID)

4. **TrainingEnrollment**
   EnrollmentID: INT, PRIMARY KEY, auto-increment
   PersonID: INT, NOT NULL, FOREIGN KEY references People(PersonID)
   ProgramID: INT, NOT NULL, FOREIGN KEY references TrainingPrograms(ProgramID)

EnrollmentDate: DATE, NOT NULL
Status: VARCHAR(50) NOT NULL CHECK (Status IN ('Enrolled', 'Completed', 'Dropped'))
UNIQUE (PersonID, ProgramID)

5. **JobApplications**
   ApplicationID: INT, PRIMARY KEY, auto-increment
   PersonID: INT, NOT NULL, FOREIGN KEY references People(PersonID)
   JobID: INT, NOT NULL, FOREIGN KEY references Jobs(JobID)
   ApplicationDate: DATE, NOT NULL
   Status: VARCHAR(50), DEFAULT 'Pending'
   UNIQUE(PersonID, JobID) to prevent duplicate applications

6. **TrainingFeedback**
   FeedbackID: INT, PRIMARY KEY, auto-increment
   PersonID: INT, NOT NULL, FOREIGN KEY references People(PersonID)
   ProgramID: INT, NOT NULL, FOREIGN KEY references TrainingPrograms(ProgramID)
   Rating: INT, NOT NULL, CHECK (RATING BETWEEN 1 AND 5)
   Comments: VARCHAR(500), optional
   FeedbackDate: DATE, NOT NULL

Data Integrity
All input data is validated at the database level using PRIMARY KEY, FOREIGN KEY, UNIQUE, and CHECK constraints to ensure referential integrity, prevent duplicate or invalid entries, and enforce valid attribute values such as age ranges, status fields, and ratings.

2.3 Schema Normalization Analysis

Example Table: TrainingPrograms
Functional Dependencies:

- ProgramID → ProgramName, SkillTaught, DurationWeeks, AvailableTo, JobID
- JobID → JobTitle, CompanyName

3NF Justification
The TrainingPrograms table satisfies Third Normal Form (3NF) because:

- It has a single primary key (ProgramID), eliminating partial dependencies.
- Job-related attributes are stored in a separate Jobs table, preventing transitive dependencies.
- All non-key attributes depend entirely on ProgramID.

III. DESIGN SPECIFICATION

3.1 Core DBMS Concepts Used

Our design specification defines how the database maintains data integrity, manages relationships, and processes operations. It includes constraints, relationships, views, triggers, and stored procedures.

A. Constraints

Rules that control what data can be stored in a table to ensure accuracy, consistency, and validity.

System example:

```
Age INT NOT NULL CHECK (Age BETWEEN 18 AND 60),
Rating INT NOT NULL CHECK (Rating BETWEEN 1 AND 5),
Status VARCHAR(50) NOT NULL CHECK (Status IN ('Unemployed', 'Employed', 'In Training', 'Trained'))
```

Other Types of Constraints Implemented:

- **PRIMARY KEY**: Ensures each record is uniquely identifiable (e.g., PersonID in People).
- **FOREIGN KEY**: Maintains valid relationships between tables (e.g., ProgramID in TrainingEnrollment references TrainingPrograms).
- **NOT NULL**: Prevents mandatory fields from being left empty (e.g., FullName in People).
- **CHECK**: Restricts allowable values and ranges (e.g., Age between 18–60, Status must be 'Enrolled', 'Completed', or 'Dropped').

Sample Implementation:

```
INSERT INTO People (FullName, Age, Status, Skills, RegistrationDate)
VALUES ('Test User', 16, 'Unemployed', 'None', CURDATE());
```

Result:

```
mysql> INSERT INTO People (FullName, Age, Status, Skills, RegistrationDate)
    -> VALUES ('Test User', 16, 'Unemployed', 'None', CURDATE());
ERROR 3819 (HY000): Check constraint 'people_chk_1' is violated.
```

The image returns an error as CHECK constraint is violated.

Purpose: To ensure accurate, consistent, and valid data across the database. It maintains relationships, prevents invalid entries, and enforces important data standards, such as age limits and valid status values.

B. Relationships

Rules that define how tables are connected, e.g., one-to-one, one-to-many, many-to-many.

System example:

- People → TrainingEnrollment (1-to-Many): A person can enroll in multiple programs, but only once per program. Enforced using a UNIQUE(PersonID, ProgramID) constraint.
- TrainingPrograms → TrainingEnrollment (1-to-Many): A program can have many participants.
- Jobs → JobApplications (1-to-Many): A job can receive multiple applications.
- People → JobApplications (1-to-Many): A person can apply to multiple jobs.

Sample Implementation:

```
INSERT INTO TrainingEnrollment (PersonID, ProgramID, EnrollmentDate, Status)
VALUES (1, 9999, CURDATE(), 'Enrolled');
```

Result:

```
mysql> INSERT INTO TrainingEnrollment (PersonID, ProgramID, EnrollmentDate, Status)
    -> VALUES (1, 9999, CURDATE(), 'Enrolled');
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`test_final_sdg_8_5_db`.`trainingenrollment`,
 CONSTRAINT `trainingenrollment_ibfk_2` FOREIGN KEY (`ProgramID`) REFERENCES `trainingprograms` (`ProgramID`) ON DELETE CASCADE)
```

The error occurs because the ProgramID (9999) does not exist in the TrainingPrograms table. Since ProgramID in TrainingEnrollment is a foreign key referencing TrainingPrograms, the database cannot insert the record, as it violates the foreign key constraint that ensures data integrity between the two tables.

Purpose: to ensure that related data is properly linked through foreign key constraints. They prevent orphan records and ensure that some data connection, such as a person being able to enroll only once per training program, are maintained.

C. Views

Virtual tables created from one or more tables using queries. They provide a simplified interface for frequent reports.

- **Distinct Report 1:**

Sample Implementation:

```
CREATE VIEW CompletedTrainingReport AS
SELECT
    P.PersonID,
    P.FullName,
    TP.ProgramName,
    TE.EnrollmentDate,
    TE.Status
FROM TrainingEnrollment TE
JOIN People P ON TE.PersonID = P.PersonID
JOIN TrainingPrograms TP ON TE.ProgramID = TP.ProgramID
WHERE TE.Status = 'Completed';

-----------------------------------------------------------------

SELECT * FROM CompletedTrainingReport;
```

Result:

```
mysql> SELECT * FROM CompletedTrainingReport;
+----------+-----------------+---------------------------------+----------------+-----------+
| PersonID | FullName        | ProgramName                     | EnrollmentDate | Status    |
+----------+-----------------+---------------------------------+----------------+-----------+
|        2 | Maria Santos    | Advanced Excel Training         | 2025-02-16     | Completed |
|        5 | Mark Reyes      | Graphic Design Workshop         | 2025-05-12     | Completed |
|        7 | Luis Garcia     | Social Media Strategies         | 2025-07-12     | Completed |
|        9 | Ramon Mendoza   | Software Testing Fundamentals   | 2025-09-06     | Completed |
|       12 | Grace Lim       | Graphic Illustration Bootcamp   | 2025-02-22     | Completed |
|       15 | Carlos Ramos    | SEO and Content Marketing       | 2025-05-15     | Completed |
|       17 | Antonio Bautista| Finance Officer Training        | 2025-07-20     | Completed |
|       20 | Jasmine Ortiz   | Call Center Leadership          | 2025-10-28     | Completed |
|       22 | Samantha De Leon| UX/UI Design Training           | 2025-02-12     | Completed |
|       25 | Daniel Morales  | Logistics Coordination Workshop | 2025-05-20     | Completed |
|       27 | Miguel Alvarado | Business Analysis Training      | 2025-07-25     | Completed |
|       30 | Lorena Cruz     | Networking Essentials           | 2025-10-30     | Completed |
|       32 | Carla Reyes     | SEO Specialist Bootcamp         | 2025-02-14     | Completed |
|       35 | Jorge Ramirez   | Finance Analyst Program         | 2025-05-24     | Completed |
|       37 | Alvin Mendoza   | Operations Supervisor Program   | 2025-07-28     | Completed |
|       40 | Melanie Cruz    | IT Technician Training          | 2025-10-05     | Completed |
|       42 | Clara Reyes     | Graphic Illustration Advanced   | 2025-02-09     | Completed |
|       45 | Emilio Bautista | Content Strategist Workshop     | 2025-05-16     | Completed |
|       47 | Roberto Morales | Logistics Manager Program       | 2025-07-20     | Completed |
|       50 | Angela Fernandez| IT Consultant Program           | 2025-10-26     | Completed |
+----------+-----------------+---------------------------------+----------------+-----------+
20 rows in set (0.00 sec)
```

This view quickly returns all participants who have completed the training by consolidating relevant data from People, TrainingEnrollment, and TrainingPrograms.

- **Distinct Report 2:**

Sample Implementation:

```
CREATE VIEW EmploymentStatusReport AS
SELECT
   Status,
   COUNT(*) AS TotalPeople
FROM People
GROUP BY Status;

---------------------------------------------------------------------

SELECT * FROM EmploymentStatusReport;
```

Result:

```
mysql> SELECT * FROM EmploymentStatusReport;
+-------------+-------------+
| Status      | TotalPeople |
+-------------+-------------+
| Trained     |          13 |
| Employed    |          16 |
| In Training |          13 |
| Unemployed  |          12 |
+-------------+-------------+
4 rows in set (0.01 sec)
```

The EmploymentStatusReport view summarizes the current employment distribution of individuals in the system by grouping them according to their status (Unemployed, Employed, In Training, Trained).

Purpose: To create virtual tables that simplify querying complex data and make frequently run reports more efficient.

- The CompletedTrainingReport consolidates data from multiple tables to quickly retrieve participants who have completed their training.
- The EmploymentStatusReport groups individuals by their employment status, offering an efficient way to monitor and report the progress of SDG 8.5.

By utilizing views, we reduce the complexity of the underlying queries and ensure that frequently accessed data can be retrieved more quickly and easily.

D. Triggers

Automated actions that execute when certain events occur in the database.

Sample Implementation:

```
DELIMITER $$

CREATE TRIGGER trg_UpdatePersonStatus
AFTER UPDATE ON TrainingEnrollment
FOR EACH ROW
BEGIN
```

```
    IF NEW.Status = 'Completed' THEN
        UPDATE People
        SET Status = 'Trained'
        WHERE PersonID = NEW.PersonID;
    END IF;
END$$

DELIMITER ;

-----------------------------------------------------------------

UPDATE TrainingEnrollment
SET Status = 'Completed'
WHERE EnrollmentID = 1;

-----------------------------------------------------------------

SELECT PersonID, Status FROM People WHERE PersonID = 1;
```

Result:

```
mysql> SELECT PersonID, Status FROM People WHERE PersonID = 1;
+----------+------------+
| PersonID | Status     |
+----------+------------+
|        1 | Unemployed |
+----------+------------+
1 row in set (0.00 sec)

mysql> UPDATE TrainingEnrollment
    -> SET Status = 'Completed'
    -> WHERE EnrollmentID = 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT PersonID, Status FROM People WHERE PersonID = 1;
+----------+---------+
| PersonID | Status  |
+----------+---------+
|        1 | Trained |
+----------+---------+
1 row in set (0.00 sec)
```

First, we returned the table for PersonID 1 to check the status.
Second, we update the status to ‘Completed’.
Third, upon returning the table once again, the Status is automatically marked as ‘Trained’.

Purpose: Automatically updates a person’s overall status to 'Trained' when their training enrollment is marked or updated to 'Completed'.

E. Stored Procedures / ACID Transactions

Predefined SQL routines that execute a series of operations as a single transaction. Ensures Atomicity, Consistency, Isolation, Durability (ACID).

Sample Implementation:

```
DELIMITER $$

CREATE PROCEDURE ApplyForJob (
    IN p_PersonID INT,
    IN p_JobID INT
)
BEGIN
    DECLARE completedTraining INT;
    DECLARE duplicateApp INT;

    START TRANSACTION;

    -- Step 1: Check if training is completed
    SELECT COUNT(*) INTO completedTraining
    FROM TrainingEnrollment
    WHERE PersonID = p_PersonID
      AND Status = 'Completed';

    IF completedTraining = 0 THEN
        ROLLBACK;
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Application failed: Training not completed';
    ELSE
        -- Step 2: Check duplicate application ONLY if training passed
        SELECT COUNT(*) INTO duplicateApp
        FROM JobApplications
```

```
        WHERE PersonID = p_PersonID
          AND JobID = p_JobID;

        IF duplicateApp > 0 THEN
            ROLLBACK;
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Application failed: Duplicate application';
        ELSE
            -- Step 3: Insert application
            INSERT INTO JobApplications (PersonID, JobID, ApplicationDate)
            VALUES (p_PersonID, p_JobID, CURDATE());

            COMMIT;
        END IF;
    END IF;
END$$

DELIMITER ;

-----------------------------------------------------------------

CALL ApplyForJob(2, 1);

-----------------------------------------------------------------

SELECT *
FROM JobApplications
WHERE PersonID = 2 AND JobID = 1;
```

Results:

- **Success Case:**

```
mysql> CALL ApplyForJob(2, 1);
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT *
    -> FROM JobApplications
    -> WHERE PersonID = 2 AND JobID = 1;
+---------------+----------+-------+-----------------+---------+
| ApplicationID | PersonID | JobID | ApplicationDate | Status  |
+---------------+----------+-------+-----------------+---------+
|            51 |        2 |     1 | 2025-12-15      | Pending |
+---------------+----------+-------+-----------------+---------+
1 row in set (0.00 sec)
```

Upon executing the stored procedure, PersonID 2 is now added in the JobApplications as PersonID 2 is eligible to apply for a job by having the status 'Completed'

- **Fail Case:**
    - **Duplicate applications**

Executing CALL ApplyForJob (1, 1); returns an error as PersonID 1 already has an application

```
mysql> CALL ApplyForJob(1, 1);
ERROR 1644 (45000): Application failed: Duplicate application
```

Other Sample:
CAll ApplyForJob (2, 1);

```
mysql> CALL ApplyForJob(2, 1);
ERROR 1644 (45000): Application failed: Duplicate application
```

- Training not complete

Executing CALL ApplyForJob (4, 3) returns an error as PersonID 4 hasn't completed training yet

```
mysql> CALL ApplyForJob(4, 3);
ERROR 1644 (45000): Application failed: Training not completed
```

If we try to return PersonID 4's data,

```
mysql> -- Check training status for PersonID = 4
mysql> SELECT
    ->     TE.EnrollmentID,
    ->     TE.PersonID,
    ->     P.FullName,
    ->     TE.ProgramID,
    ->     TP.ProgramName,
    ->     TE.Status AS EnrollmentStatus
    -> FROM TrainingEnrollment TE
    -> JOIN People P ON TE.PersonID = P.PersonID
    -> JOIN TrainingPrograms TP ON TE.ProgramID = TP.ProgramID
    -> WHERE TE.PersonID = 4;
+--------------+----------+-----------+-----------+--------------------------+------------------+
| EnrollmentID | PersonID | FullName  | ProgramID | ProgramName              | EnrollmentStatus |
+--------------+----------+-----------+-----------+--------------------------+------------------+
|            4 |        4 | Ana Lopez |         4 | Customer Service Training | Enrolled         |
+--------------+----------+-----------+-----------+--------------------------+------------------+
1 row in set (0.00 sec)
```

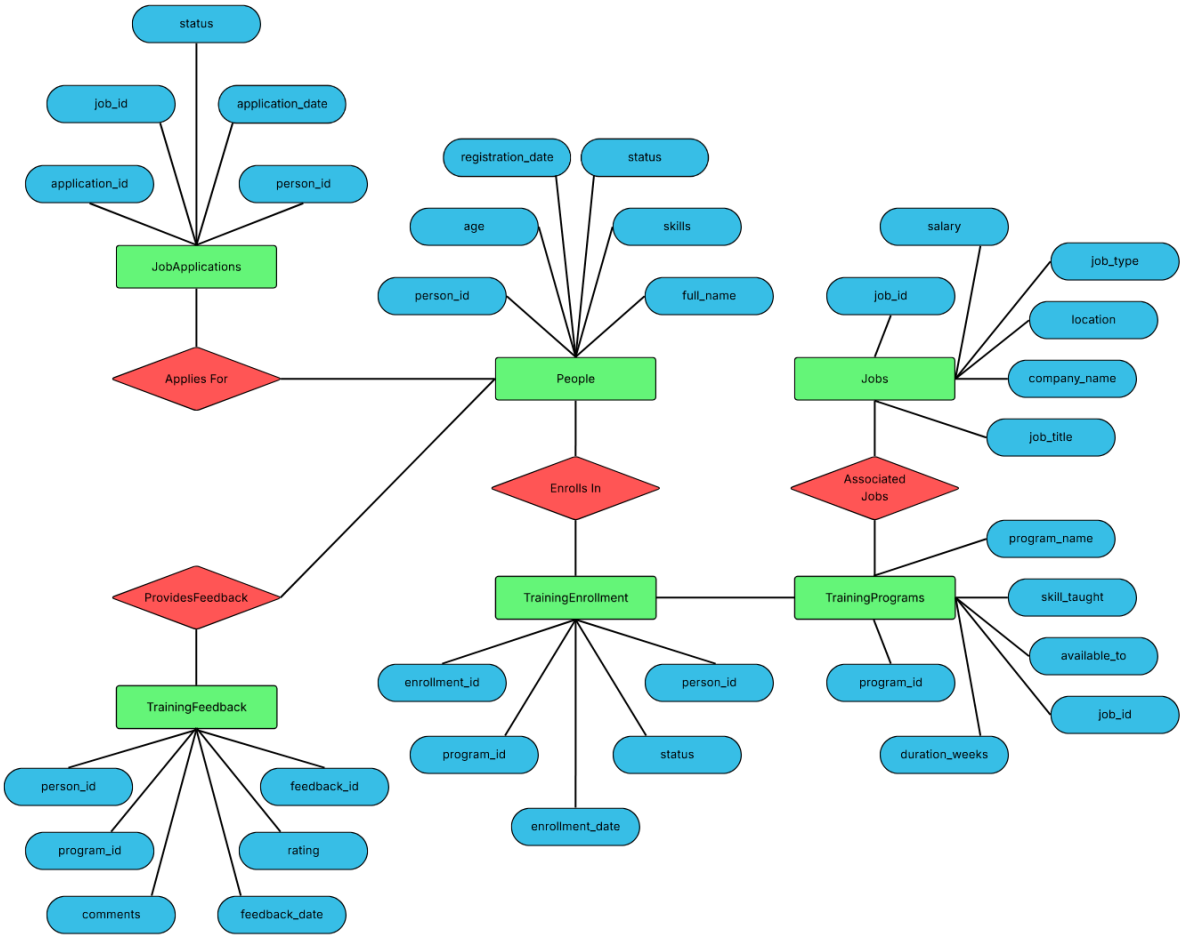
The status is currently 'Enrolled' so PersonID 4 is not eligible to apply for a job yet.

Purpose: To safely process job applications within a single ACID-compliant transaction. It ensures only eligible participants can apply, blocks duplicate applications, and maintains a consistent database state through COMMIT and ROLLBACK operations.

Testing with multiple JobIDs demonstrates the procedure's robust ACID compliance, showing that successful applications are committed while violations (untrained applicant or duplicate application) trigger rollbacks, leaving the database unchanged.

3.2 ER Diagram

1. **Conceptual ERD**



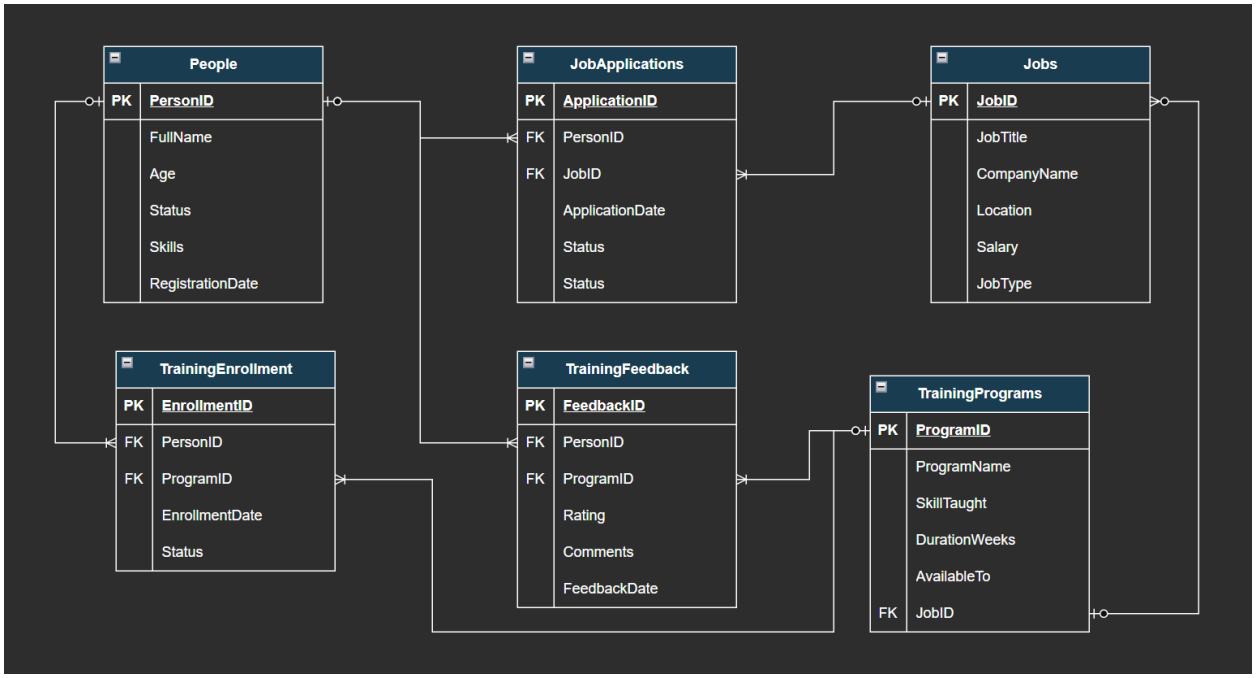
Entities:

- People: Individuals seeking employment or training.
- Jobs: Available employment opportunities.
- TrainingPrograms: Skill development programs.
- TrainingEnrollment: Records of people enrolling in programs
- JobApplications: Records of applications submitted by people for jobs
- TrainingFeedback: Feedback submitted by participants for training programs.

Relationships:

- People can enroll in multiple training programs.
- People can apply to multiple jobs.
- People can provide feedback for multiple training programs.
- Training programs and jobs may be linked to show skill-job alignment.

2. Logical ERD



Entities and Attributes:

- People: PersonID (PK), FullName, Age, Status, Skills, RegistrationDate
- Jobs: JobID (PK), JobTitle, CompanyName, Location, Salary, JobType
- TrainingPrograms: ProgramID (PK), ProgramName, SkillTaught, DurationWeeks, AvailableTo, JobID (FK)
- TrainingEnrollment: EnrollmentID (PK), PersonID (FK), ProgramID (FK), EnrollmentDate, Status
- JobApplications: ApplicationID (PK), PersonID (FK), JobID (FK), ApplicationDate, Status
- TrainingFeedback: FeedbackID (PK), PersonID (FK), ProgramID (FK), Rating, Comments, FeedbackDate

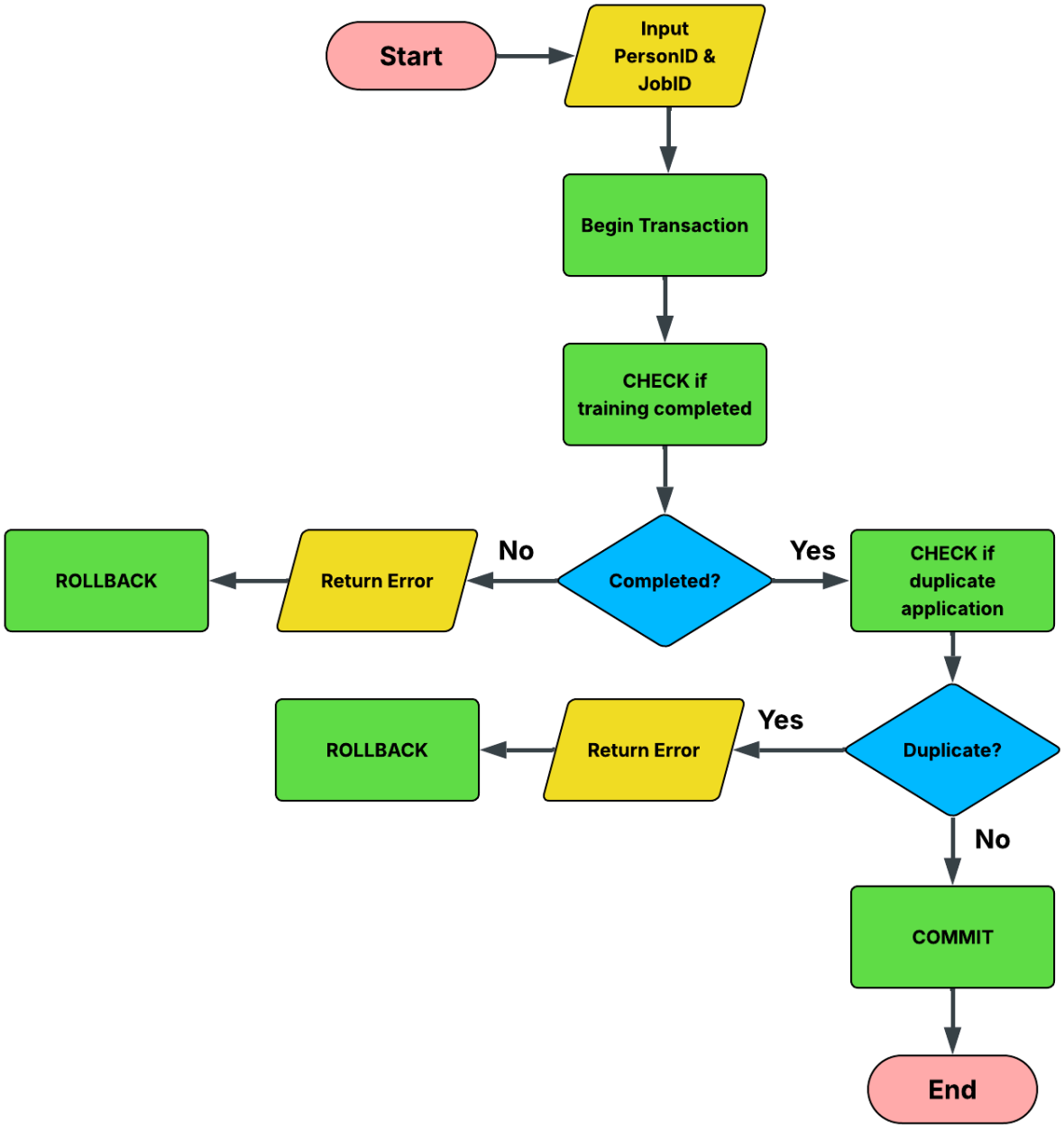
Relationships & Cardinality:

- One-to-many: People → TrainingEnrollment, People → JobApplications, People → TrainingFeedback
- One-to-many: TrainingPrograms → TrainingEnrollment, TrainingPrograms → TrainingFeedback
  One-to-many: Jobs → JobApplications

3. Physical ERD

- Includes actual database implementation with data types, constraints, and keys.
- All tables include appropriate PRIMARY KEY and FOREIGN KEY constraints to enforce referential integrity.
- Constraints such as CHECK, UNIQUE, and NOT NULL maintain data validity and consistency, e.g., age limits, valid status values, and preventing duplicate applications.

3.3 Transaction Flowchart



The transaction flowchart illustrates the ACID-compliant workflow of the ApplyForJob stored procedure, which controls how job applications are processed in the system.

The process begins when a PersonID and JobID are provided as inputs. Once the procedure is called, a database transaction is started to ensure that all succeeding operations are treated as a single atomic unit.

First, the system checks whether the applicant has completed at least one training program. If no completed training record is found, the procedure immediately raises an error and performs a ROLLBACK, to ensure that no partial changes are saved to the database.

If the training requirement is satisfied, the procedure then checks for duplicate job applications by verifying whether the same PersonID and JobID combination already exists in the JobApplications table. If a duplicate is detected, the transaction is again rolled back, and an appropriate error message is returned.

Only when all validation checks pass does the system insert a new job application record into the JobApplications table. The transaction is then committed, permanently saving the changes to the database.

IV. TESTING & RESULTS

4.1 Test Cases

Test Case 1:

Input	Expected Output	Actual Output
PersonID = 5 (completed training), JobID = 10	Row inserted into JobApplications with status = 'Pending'	Row inserted correctly

Implementation:

```
CALL ApplyForJob(5, 10);

SELECT * FROM JobApplications
WHERE PersonID = 1 AND JobID = 3;
```

Result:

```
mysql> CALL ApplyForJob(5, 10);
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql>
mysql>
mysql> SELECT * FROM JobApplications
    -> WHERE PersonID = 5 AND JobID = 10;
+---------------+----------+-------+-----------------+---------+
| ApplicationID | PersonID | JobID | ApplicationDate | Status  |
+---------------+----------+-------+-----------------+---------+
|            54 |        5 |    10 | 2025-12-15      | Pending |
+---------------+----------+-------+-----------------+---------+
1 row in set (0.00 sec)
```

Test Case 2:

Input	Expected Output	Actual Output
PersonID = 18 (not completed training), JobID = 42	Procedure raises error: "Application failed: Training not completed"	Procedure raised error as expected

Implementation:

CALL ApplyForJob(2, 3);

Result:

```
mysql> CALL ApplyForJob(18, 42);
ERROR 1644 (45000): Application failed: Training not completed
```

Test Case 3:

Input	Expected Output	Actual Output
PersonID = 9, JobID = 12 (already applied)	Procedure raises error: "Application failed: Duplicate application"	Procedure raised error as expected

Implementation:

```
CALL ApplyForJob(9, 12);
```

Result:

```
mysql> CALL ApplyForJob(9, 12);
ERROR 1644 (45000): Application failed: Duplicate application
```

4.2 ACID Compliance Test

Here, we demonstrate atomicity using a rollback scenario:

1. Start a transaction manually that violates business rules.
2. Confirm no partial insert occurs (atomicity).

Step 1 - Start Transaction:

```
START TRANSACTION;
CALL ApplyForJob(23, 32);
```

Result:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL ApplyForJob(23, 32);
ERROR 1644 (45000): Application failed: Training not completed
```

Step 2 – Verify

```
SELECT * FROM JobApplications
WHERE PersonID = 23 AND JobID = 32;
```

Result:

```
mysql> SELECT * FROM JobApplications
    -> WHERE PersonID = 23 AND JobID = 32;
Empty set (0.00 sec)
```

Outcome:

- No new row inserted, shows atomicity of procedure.
- Any error inside the procedure triggers rollback.

V. CONCLUSION AND CONTRIBUTIONS

5.1 Conclusion

The Employment Monitoring and Skills-Based Job Matching System successfully provides a centralized and efficient platform to support SDG 8 Target 8.5. By integrating individual profiles, training programs, and job opportunities into a single relational database, the system enables organizations to monitor employment trends, track skills development, and ensure that job matching is fair and effective. Through automated triggers, constraints, and ACID-compliant transactions, data integrity is maintained, and data consistency is maintained across all operations, reducing errors and improving reliability. The system also allows for actionable insights through reporting views and structured data analysis, supporting evidence-based decision-making and policy development.

The system demonstrates the ability to:

- Track skills and training outcomes to monitor workforce development
- Match qualified individuals to suitable job opportunities
- Support structured monitoring of unemployed youth and vulnerable groups

Overall, this project illustrates how a well-designed database system can streamline employment monitoring, enhance skills-based job matching, and contribute meaningfully to creating productive, inclusive, and decent work opportunities for all.

5.2 Individual Contributions

Name	**Contribution/Distribution**
Jezren R. Atienza (424002335)	Part 3 Documentation, Core Concept, SQL Database Implementation, DQL, DML, Reporting, Revision

Rimuel D. Loking (424003533)	Part 1 & 5, Documentation, Core Concept, SQL Database Implementation, DDL, DML, DQL, TCL, ERD, Flowchart, Reporting, Full Revision, GitHub Repository
Jamel P. Macadaub (424002156)	Part 2, Documentation, Core Concept, SQL Database Implementation, DDL, DML, DQL, TCL, ERD, Reporting, GitHub Repository
Patrick Jay M. Canlas (424002336)	Part 2 Documentation, Core Concept, SQL Database Implementation, Flowchart, DQL, DML, Reporting
John Rafael D. Sullera (424002338)	Part 1 Documentation, Core Concept, SQL Database Implementation, DQL, DML, ERD, Reporting
Gregg Martin D. Cometa (424001164)	Part 3 Documentation, Core Concept, SQL Database Implementation, DQL, DML, Reporting

REFERENCES

Global Goals. (n.d.). *Goal 8: Decent work and economic growth*. https://globalgoals.org/goals/8-decent-work-and-economic-growth/

United Nations. (n.d.). *Goal 8. SDGs Knowledge Platform*. https://sdgs.un.org/goals/goal8

GeeksforGeeks. (2025, October 9). *Introduction to database normalization. GeeksforGeeks*. https://www.geeksforgeeks.org/database-normalization/

W3Schools. (n.d.). *SQL tutorial. W3Schools*. https://www.w3schools.com/sql/

GeeksforGeeks. (n.d.). *ACID properties in DBMS. GeeksforGeeks*. https://www.geeksforgeeks.org/acid-properties-in-dbms/

Lucidchart. (n.d.). *Lucidchart: Diagramming & visual communication*. https://www.lucidchart.com/pages

dbdiagram.io. (n.d.). *dbdiagram: Quick database diagrams*. https://dbdiagram.io/d

diagrams.net. (n.d.). *diagrams.net (formerly Draw.io)*. https://app.diagrams.net/

Visual Paradigm. (n.d.). *Conceptual vs logical vs physical data model*. https://online.visual-paradigm.com/knowledge/visual-modeling/conceptual-vs-logical-vs-physical-data-model

ACKNOWLEDGEMENT

Note: This paper utilized ChatGPT (OpenAI, 2025) as a tool for coding, debugging, data input, fact-checking, and to assist with grammar and sentence structure. All information gathered from this tool was verified using credible sources listed in the References section.