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

Final Project Documentation in Database Management System 1 w/Lab

Final Project

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:

Bumatay, Jay-Roween B.
Galpo, Ayessa Apple A.
Lacasa, Trishia May S.
Quizon, Kate Ashley G.
Yaranon, Rica Mae A.

2.4 BSIT
National Teachers College

Instructor:
Ms. Justin Louise R. Neypes

Date:
December 2025

Table of Contents

I. Introduction

1.1. Project Overview & UN SDG Target

The Philippines is one of the countries in the world that is most prone to natural disasters, such as typhoons, earthquakes, and floods. Communities are usually left helpless, with delayed assistance from either local government or non-governmental organizations. Even with donations and pledges for relief operations, many people still are not getting the urgent attention and assistance they need. This usually creates a gap between disaster response and actual recovery. Therefore, it is essential to establish a reliable, data-driven system to ensure that aid can reach those who need it most. This project is aligned with the **United Nations Sustainable Development Goal (SDG) 11: Sustainable Cities and Communities**, specifically **Target 11.5**, reducing the number of people affected by disasters and ensuring prompt assistance during emergencies.

This project aims to build a data-driven disaster response and monitoring system that will enable local governments to determine which areas need immediate support. Through its organization and analysis of disaster-related data, the system advances accountability and transparency in the distribution of aid, enhances the operations of local governments and non-governmental organizations, and empowers communities by providing clear guidance on how to minimize harm during emergencies and by identifying high-risk areas that need stronger protection.

The ultimate result is that this project helps in building safer, more durable communities. When reliable and comprehensive data are available, protection can be afforded to the people, governments can respond more effectively, and dignified recovery can be supported. By aligning with SDG 11.5, the project addresses immediate disaster response needs but at the same time fosters long-term resilience and preparedness for future challenges. Reliable and comprehensive data becomes a solution to help people be safe and support recovery with dignity.

1.2. Problem Statement

The disaster response and monitoring system is designed to address several real-world disaster issues that affect disaster response, community safety, and government planning. In line with SDG 11.5, which focuses on reducing the loss of lives, people affected, and economic damage caused by disasters, the system strengthens how information is collected, organized, and accessed. By improving how information is collected, organized, and accessed, the system provides a clearer foundation for decision-making and public service. It aims to solve the following problems:

- Incompleteness and unreliable record of families affected by disasters, making it hard for the government and donors to accurately determine the extent of losses and the specific needs of each household.
- Lack of transparency in monitoring disaster-related projects, leading to uncertainty about whether planned government projects are established or still under construction, or delivered in substandard form.
- Insufficient information on vulnerable groups, such as low-income families and communities living in high-risk areas, which results in insufficient protection and support in times of disasters.
- Limited access to accurate and updated data in disasters, which is causing delays in early preparation, planning, and immediate emergency response, leading to increased deaths, injuries, and the amount of people affected.
- Inability to determine high-risk areas due to poor infrastructure-related data, resulting in inadequate preventive measures and greater damage to important facilities.
- Local authorities and NGOs lack reliable, organized data on which areas and households were impacted during the specific disaster, making it harder for them to make informed decisions during disaster preparedness, response, and recovery efforts.

II. Requirements & Analysis

2.1. Functional Requirements and Non-Functional Requirements

The following tables present the functional and non-functional requirements of the system. All requirements have been fully implemented and complied with in the final system, ensuring both technical accuracy and overall reliability.

Table 1. Functional Requirements (FR)

ID	Requirement	Alignment	Compliance
FR1	System must successfully load and store input data into the fully designed relational schema.	DDL/DML	Data is inserted correctly into the normalized tables with proper relationships.
FR2	System must implement the **three minimum mandatory DBMS concepts** correctly.	Core Concepts	Tables, constraints, and VIEWs are applied correctly.
FR3	System must execute a **transactional operation** (e.g., recording a major event) using a Stored Procedure that explicitly enforces **ACID properties**.	Finals Concepts	The stored procedures use BEGIN, COMMIT, ROLLBACK, that triggers ensure valid dates.
FR4	System must generate the required SDG reports via VIEWS or DQL and display the results clearly in the chosen interface.	DQL/Reporting	Reports show the affected families, damages, and the vulnerable areas clearly.

Table 2. Non-Functional Requirements (NFR)

ID	Requirement	Metric	Compliance
NFR1	**Robustness:** Program/Scripts must handle invalid input/operations gracefully without violating database integrity.	Error messages should be informative.	CHECKs and triggers block invalid input with clear messages.
NFR2	**Maintainability:** Schema and stored logic code must be modular, well-commented, and adhere to SQL best practices.	Use of comments and logical VIEW definitions.	The scripts and VIEWs are commented and easy to maintain.
NFR3	**Performance:** The three required reports (FR4) must execute efficiently.	Reports should return results in under 1 second on standard hardware with test data.	Queries runs quickly on sample data.

2.2. Data Requirements

This system uses an exact and complete dataset, showcasing the entities and their attributes. The major tables consist of 50 records generated by ChatGPT, specifically the Disaster_event, Population_segment, Human_impact, Asset, and Impacted_asset. Supporting tables such as Hazard_type (has 6 records), Region (has 17 records), Municipality (has 50 records), and Barangay (has 50 records) are also filled to create realistic relationships between locations and disaster events. This system is developed to handle bigger datasets for real-time usage that supports more that 50 entries indicated in major tables.

The Region table has only 17 records because the Philippines is composed of 17 administrative regions, and it would be unrealistic to create 50 regions since the country does not have that many. Similarly, the Hazard_type table has only 6 records because there are only a few major types of hazards that frequently occur in the Philippines and commonly recognized in disaster management, such as typhoons, earthquakes, floods, landslides, storm surges, and fires. These hazard types do not reach anywhere near 50 in real-world classifications. Keeping the number of hazard types accurate ensures that the data reflects actual disaster categories and remains meaningful for analysis. By using realistic counts for regions and hazard types, the system maintains accurate relationships and avoids creating artificial data that does not exist in real-world disaster management frameworks.

To keep the consistency and validity of the data, several rules are applied. Numeric values must be only between the given ranges. For poverty_rate, it must be between 0-100 only (following the PSA's 2023 Statistics), hazard severity between 1 and 10, and vulnerability scores between 1 and 5. Also, date attributes (Start_Date and End_Date) are validated to ensure that every end_date is not less than the start_date and vice versa. Other specific attributes rely on fixed and allowable values such as the Damage_Level (Minor, Moderate, Severe, Destroyed) and Evacuation_Status (Safe, At Risk, or Evacuated) to reduce errors in entering data. Overall, these rules that every data entered in the system are clean, consistent, and reliable.

Here are the tables that define the database structure, each representing a key entity in the system. For every table, the attribute name, data type, and description are clearly specified to ensure consistency and understanding of this project.

Table 3. Region

Attribute Name	Data type	Description
Region_ID (PK)	Integer	Unique identifier for the region.
Region_Name	String	Name of the region.
Poverty_Rate	Float	Poverty rate in the region.

Table 3.1 Percentage Scale of Poverty_Rate

Poverty Rate	Level	Description
0–5%	Minimal / Low	Very few people are below poverty, the region is generally well-off.
6–15%	Moderate	Some households are vulnerable; moderate need for social services.
16–25%	High	A large part of the population is poor; priority region for aid and disaster support.

26–100%	Very High	Extreme poverty; urgent attention required, especially during disasters.

Table 4. Municipality

Attribute Name	Data Type	Description
Municipality_ID (PK)	Integer	Unique identifier for the municipality.
Region_ID (FK)	Integer	Foreign key linking to the region.
Municipality_Name	String	Name of the municipality.

Table 5. Barangay

Attribute Name	Data Type	Description
Barangay_ID (PK)	Integer	Unique identifier for the barangay.
Municipality_ID (FK)	Integer	Foreign key linking to the municipality.
Barangay_Name	String	Name of the barangay.

Table 6. Population Segment

Attribute Name	Data Type	Description
Segment_ID (PK)	Integer	Unique identifier for the population segment.
Barangay_ID (FK)	Integer	Foreign key linking to the barangay.
Description	String	Description of the segment.
Total_Count	Integer	Total number of people in the segment.
Vulnerability_Index	Float	Vulnerability score.
Evacuation_Status	String	Status of evacuation (e.g., Safe, At Risk, Evacuated).

Table 6.1 Vulnerability Index Levels

Index	Risk Level	Description / Example
1	Minimal	Small households in well-prepared areas have imited exposure to disaster effects.
2	Low	Families in low-lying areas but with access to early warnings.
3	Moderate	Communities without immediate disaster infrastructure, moderate exposure.
4	High	Elderly, Children, and PWD in vulnerable areas are likely to be affected without intervention.
5	Very High	Informal settlers in flood-prone areas or communities with no evacuation plans.

Table 7. Asset

Attribute Name	Data Type	Description
Asset_ID (PK)	Integer	Unique identifier for the asset.
Barangay_ID (FK)	Integer	Foreign key linking to the barangay.
Asset_Type	String	Type of asset (e.g., School, Hospital, Road, Farmland, House).
Pre_Disaster_Value	Float	Value of assets before disaster.
Coordinates	String	Location point.

Table 8. Human Impact

Attribute Name	Data Type	Description
Human_Impact_ID (PK)	Integer	Unique identifier for the human impact record.
Event_ID (FK)	Integer	Foreign key linking to the disaster event.
Segment_ID (FK)	Integer	Foreign key linking to the population segment.

Deaths_Count	Integer	Number of deaths.
Injuries_Count	Integer	Number of injuries.
People_Affected_Count	Integer	Total number of people affected.

➢ **LIMITATION:** In the SDG Goal 11.5, it indicates that missing persons are included. However, there are people who are missing and no longer have any connection with their family, neighbors, or those around them. That is why missing persons were not included since the database will be inconsistent and unreliable.

Table 9. Disaster Event

Attribute Name	Data Type	Description
Event_ID (PK)	Integer	Unique identifier for the disaster event.
Hazard_Type_ID (FK)	Integer	Foreign key linking to the hazard type.
Barangay_ID (FK)	Integer	Location of the event.
Start_Date	Date	When the disaster started.
End_Date	Date	When the disaster ended.
Severity_Scale	Float	Severity rating.
Description	String	Description of the event.

Table 9.1 Severity_Scale

Scale	Intensity	Description / Example
1–3	Low	Minor disaster; minimal damage to buildings or infrastructure; e.g., light typhoon with wind < 60 km/h.
4–6	Moderate	Moderate disaster; some buildings damaged, roads affected; e.g., moderate typhoon or localized earthquake.

7–8	High	Severe disaster; large-scale infrastructure damage, many houses affected; e.g., strong typhoon, magnitude 6 earthquake.
9–10	Very High	Catastrophic disaster; widespread destruction, high fatalities; e.g., super typhoon, major earthquake.

Table 10. Impacted Asset

Attribute Name	Data Type	Description
Impact_Asset_ID (PK)	Integer	Unique identifier for the impact record.
Event_ID (FK)	Integer	Foreign key linking to the disaster event.
Asset_ID (FK)	Integer	Foreign key linking to the asset.
Damage_Level	String	Severity of damage.
Estimated_Loss_Value	Float	Estimated financial loss.

Table 11. Hazard Type

Attribute Name	Data type	Description
Hazard_Type_ID (PK)	Integer	Unique identifier for the hazard type.
Name	String	Name of the hazard (e.g., Flood, Earthquake, Typhoon).
Category	String	Classification of hazard (e.g., Geological, Hydrometeorological).

2.3. Schema Normalization Analysis

The tables inside this database were verified to ensure that it follows the Data Normalization, specifically the 3rd Normal Form of Boyce-Codd Normal Form (BCNF). EVERY table uses a single, unique ID (or Primary Key) that determines all other attributes. No table contains **Partial Dependencies** (where only part of a key determines another attribute) or **Transitive Dependencies** (where a non-key attribute depends on another non-key). Applying normalization in the system ensures that all tables avoid duplication and update anomalies. Since each table's functional dependencies are determined solely by its primary key, all five major tables comply with 3NF/BCNF. Overall, the database is normalized and guarantees clean, reliable, and efficient data.

Table 4. Functional Dependencies and Normalization Levels of Complex Entities/Tables of SDG 11 System

Major Entity/Table Name	Primary Key	Functional Dependency	Normalization Level
1. Disaster_Event	Event_ID	Event_ID ● All non-key attributes depend only on Event_ID (PK). No other attribute determines everything.	BCNF
2. Population_Segment	Segment_ID	Segment_ID ● Segment_ID uniquely determines all details. No extra dependencies.	BCNF
3. Human_Impact	Human_Impact_ID	Human_Impact_ID ● The Primary Key (Human_Impact_ID) determines all fields without redundancy.	3NF/BCNF
4. Asset	Asset_ID	Asset_ID ● Primary Key (Asset_ID) controls all other attributes in this table. No partial and transitive dependency	BCNF
5. Impacted_Asset	Impact_Asset_ID	Impact_Asset_ID ● One unique ID in every damaged asset record. All non-key attributes depend on Impact_Asset_ID	3NF/BCNF

III. Design Specification

3.1. Core DBMS Concepts Used

To ensure the accuracy of the data, efficient reporting, and reliable system operations, the Core Database Management System (DBMS) concepts from Prelim, Midterm, and Finals were applied to the disaster response and monitoring system. First, the Prelim concepts or lessons applied in building the structure of the database are the Data Definition Language (DDL), which is responsible for creating the database and creating the details of each table, and the Data Manipulation Language, which is responsible for the insertion of all the data in the database. These two concepts create normalized tables, define primary and foreign keys, and run initial queries to check if the data and relationships were correct. Moreover, the creation of a complex VIEW was inserted to join multiple related tables. The VIEW makes reporting easier because it allows one to reuse one complex query instead of rewriting it multiple times, improving clarity and maintainability.

For the Midterm, the SQL constraints, primarily the CHECK constraints, were applied to the database, which are used in many tables to enforce valid and consistent input values within specific fields (such as valid damage levels, severity ranges, and category restrictions). With the help of CHECK constraints, it controls what values can be stored in certain fields, such as allowable damage levels, severity ranges, or acceptable categories. This ensures that the basic rules are already correct before any other logic happens.

The 3NF/BCNF normalization was implemented to the whole database to make sure that each attribute depends only on the primary key. This prevents duplication of data, such as repeating location names, by separating Region, Municipality, Barangay, and Disaster_Event into their own tables instead of putting all of that in one table which would cause redundancy and inconsistencies

Furthermore, the TRIGGER function was used in the Disaster_Event table of the database to ensure the End_Date is never earlier than the Start_Date. A trigger was chosen to apply in the Disaster_Event table since CHECK constraints in phpMyAdmin cannot properly compare two different columns, while a trigger can perform this logic and display a clear error message.

Lastly, Stored Procedure was applied along with the Transaction concept to always make sure that connected data, specifically a disaster event and its corresponding human impact record, is inserted together. By applying Database Transaction concept in the Stored Procedure, the system ensures that both inserts are successful as one complete operation, and if an error occurs, everything is rolled back to prevent incomplete or inconsistent data in the database. This keeps the database always accurate, consistent, and reliable when storing disaster-related data.

Justification and Implementation Details

Presented below are the justification and implementation details of the system's database features. These highlight how triggers, views, and stored procedures were applied to strengthen data validation, reporting, and transactional reliability

1. **TRIGGER**

   In the system, the Disaster_Event table uses a TRIGGER instead of CHECK constraint to validate the Start_Date and End_Date of a specific disaster. By using a TRIGGER, the system reliably checks that the Start_Date should not be later than the End_Date every time a record is updated or inserted in the database. Invalid inputs will display a message "Start_Date cannot be later than End_Date". TRIGGER ensures that all disaster event data is accurate, prevents incorrect date entries, and displays a clear error message when the rule is broken. Overall, the trigger makes the validation stronger and more consistent for the system.

SQL CODE

```
DELIMITER $$

CREATE TRIGGER trg_validate_disaster_dates
BEFORE INSERT ON disaster_event
FOR EACH ROW
BEGIN
```

```
    IF NEW.Start_Date > NEW.End_Date THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Start_Date cannot be later than End_Date.';
    END IF;
END$$

DELIMITER ;
```

2. VIEW

The system used VIEWS to simplify complex queries and make reporting much easier. Instead of rewriting long JOINs or recalculating totals many time. This helps generate summaries and important disaster information instantly, which is very useful for analysis, decision-making, and reporting of the system. In short, VIEWS help keep the queries cleaner, faster, and more organized by automatically preparing the data needed.s which is time-consuming, VIEWS can save these operations as reusable virtual tables

VIEW #1: Disaster Summary

SQL CODE

```
CREATE VIEW view_disaster_summary AS
SELECT
    de.Event_ID,
    ht.Name AS Hazard_Type,
    b.Barangay_Name,
    de.Start_Date,
    de.End_Date,
    de.Severity_Scale,
    de.Description
FROM disaster_event de
JOIN hazard_type ht ON de.Hazard_Type_ID = ht.Hazard_Type_ID
JOIN barangay b ON de.Barangay_ID = b.Barangay_ID;
```

➢ First, the VIEW was created for the Disaster Summary to make the data easier to access and understand. Since disaster information comes from many related tables, the SQL queries needed to combine them are long and repetitive. By creating a VIEW, there is no longer need to rerun these JOIN queries every time. Just simply select from the view as if it were a table in the database. This is greatly helpful when querying the same details since immediate summaries of disaster events, affected areas, and impacts are often needed. Overall, VIEW was used to simplify repeated queries, reduce errors, and present disaster data in a clear and organized way.

VIEW #2: Total Losses in Each Event

SQL CODE

```
CREATE VIEW view_total_losses_per_event AS
SELECT
   de.Event_ID,
   ht.Name AS Hazard_Type,
   COALESCE(SUM(ia.Estimated_Loss_Value), 0) AS Total_Loss
FROM disaster_event de
LEFT JOIN impacted_asset ia ON de.Event_ID = ia.Event_ID
JOIN hazard_type ht ON de.Hazard_Type_ID = ht.Hazard_Type_ID
GROUP BY de.Event_ID, ht.Name;
```

➢ Second, the VIEW was created for computing the total losses per disaster event to make economic damage reporting faster and more efficient. Instead of manually writing a SUM() function every time the total estimated losses from impacted assets is needed, the VIEW automatically computes it for every disaster event. This saves time and makes the database consistent, especially when executing reports for LGU budgeting of the impacted assets. By storing this calculation inside a created VIEW, it can access updated totals instantly without repeating complex queries or risking errors.

3. STORED PROCEDURE

In the system, the Stored Procedure with an ACID transaction was used to ensure that disaster event data is always complete and reliable. It handles the adding of the Disaster_Event and its corresponding Human_Impact record, then the transaction ensures that these operations completely follow the **ACID Properties**. Through **Atomicity**, both inserts happen together and if one fails, nothing will proceed. This prevents situations where a disaster event exists without its human impact data. **Consistency** is still kept since the database will never store incomplete or missing data. With the help of Rollback, it safely ensures that the system automatically cancels all changes if an error (such as foreign key mismatch or invalid input) occurs and the transaction is rolled back to its prior valid state.

Also, **Isolation** automatically happens when the stored procedure starts the transaction. Starting from the *START TRANSACTION* part to *COMMIT* part, nobody else can see the ongoing transaction and affect the partial changes. Lastly, **Durability** makes sure that once the data is committed, both records are permanently and securely stored. Applying a transactional stored procedure helps keep the disaster information consistent and accurate. Invalid input will display an error message "Transaction was failed and rolled back".

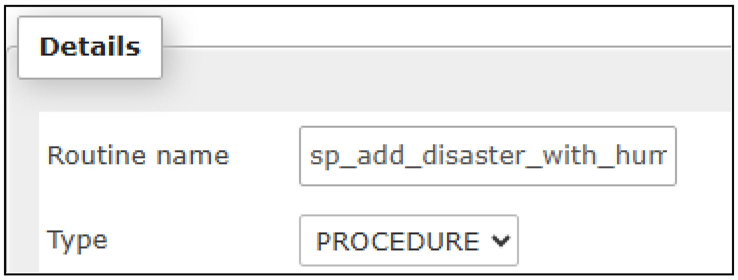
DIRECT SQL CODE

```
DELIMITER $$


CREATE PROCEDURE sp_add_disaster_with_human_impact(
   IN p_Event_ID INT,
   IN p_Hazard_Type_ID INT,
   IN p_Barangay_ID INT,
   IN p_Start DATE,
   IN p_End DATE,
   IN p_Severity INT,
   IN p_Description VARCHAR(255),
    IN p_Human_Impact_ID INT,
    IN p_Segment_ID INT,
    IN p_Deaths INT,
    IN p_Injuries INT,
    IN p_Affected INT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Transaction failed and
was rolled back.';
    END;

    START TRANSACTION;

    INSERT INTO disaster_event
     (Event_ID, Hazard_Type_ID, Barangay_ID, Start_Date, End_Date, Severity_Scale,
Description)
    VALUES
        (p_Event_ID, p_Hazard_Type_ID, p_Barangay_ID, p_Start_Date, p_End_Date,
p_Severity_Scale, p_Description);

    INSERT INTO human_impact
          (Human_Impact_ID, Event_ID, Segment_ID, Deaths_Count, Injuries_Count,
People_Affected_Count)
    VALUES
                (p_Human_Impact_ID, p_Event_ID, p_Segment_ID, p_Deaths_Count,
p_Injuries_Count, p_People_Affected_Count);
```

```
    COMMIT;
END$$

DELIMITER ;
```

STEP-BY-STEP PROCESS OF EXECUTING STORED PROCEDURE IN ROUTINE TAB

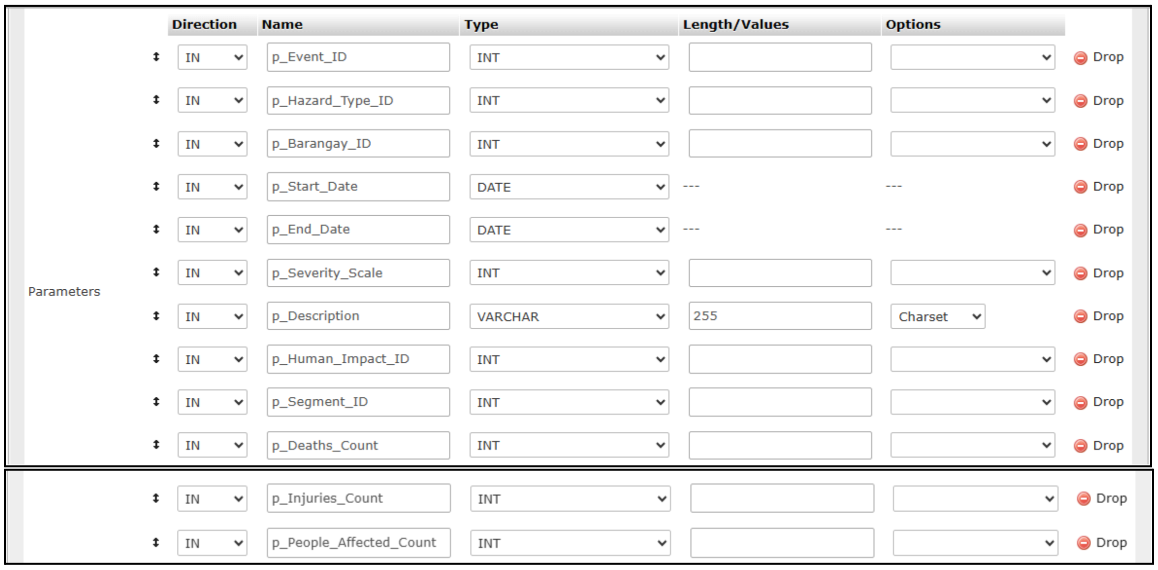
1. Inside phpMyAdmin, select the database
2. Click the **Routines tab → Create New Routine**
3. Set **Routine name** = *sp_add_disaster_with_human_impact* and Choose **PROCEDURE**



Details

Routine name	sp_add_disaster_with_hum
Type	PROCEDURE

4. Add parameters exactly



Parameters

Direction	Name	Type	Length/Values	Options	
IN	p_Event_ID	INT			Drop
IN	p_Hazard_Type_ID	INT			Drop
IN	p_Barangay_ID	INT			Drop
IN	p_Start_Date	DATE	---	---	Drop
IN	p_End_Date	DATE	---	---	Drop
IN	p_Severity_Scale	INT			Drop
IN	p_Description	VARCHAR	255	Charset	Drop
IN	p_Human_Impact_ID	INT			Drop
IN	p_Segment_ID	INT			Drop
IN	p_Deaths_Count	INT			Drop
IN	p_Injuries_Count	INT			Drop
IN	p_People_Affected_Count	INT			Drop

5. In the Description box, insert/paste the stored procedure code between BEGIN and END

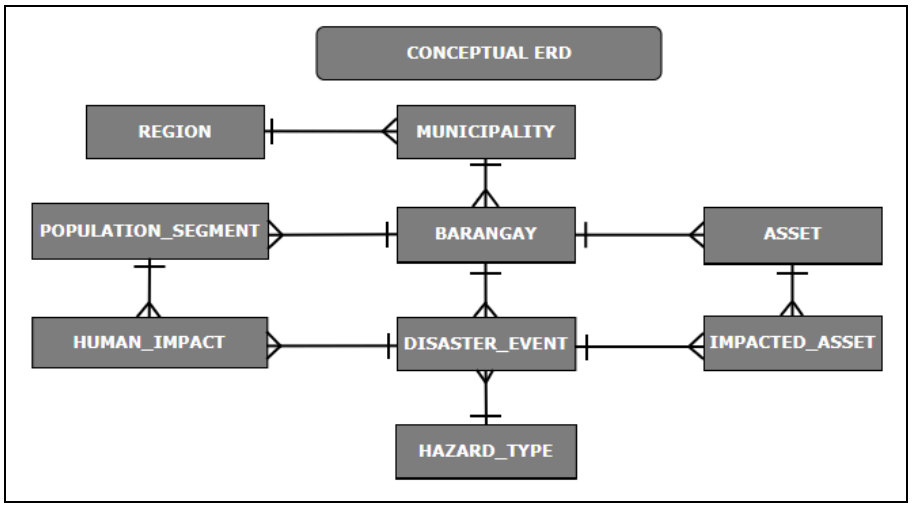
```
BEGIN
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;
         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Transaction failed and was rolled back.';
   END;

   START TRANSACTION;

   INSERT INTO disaster_event
        (Event_ID, Hazard_Type_ID, Barangay_ID, Start_Date, End_Date, Severity_Scale, Description)
   VALUES
          (p_Event_ID, p_Hazard_Type_ID, p_Barangay_ID, p_Start_Date, p_End_Date, p_Severity_Scale, p_Description);

   INSERT INTO human_impact
           (Human_Impact_ID, Event_ID, Segment_ID, Deaths_Count, Injuries_Count, People_Affected_Count)
   VALUES
     (p_Human_Impact_ID, p_Event_ID, p_Segment_ID, p_Deaths_Count, p_Injuries_Count, p_People_Affected_Count);

   COMMIT;
END
```

6. After that, Click **Go**. You should see the routine listed under **Routines** of the **Stored Procedure** of the Database

3.2. ER Diagram

The following presents the complete set of Entity-Relationship Diagrams (ERDs), covering the Conceptual, Logical, and Physical models. All entities, attributes, primary keys, foreign keys, and cardinalities are fully defined, representing the finalized database design."

- **CONCEPTUAL ERD**



View a clearer version at this link: CONCEPTUAL ERD.pdf

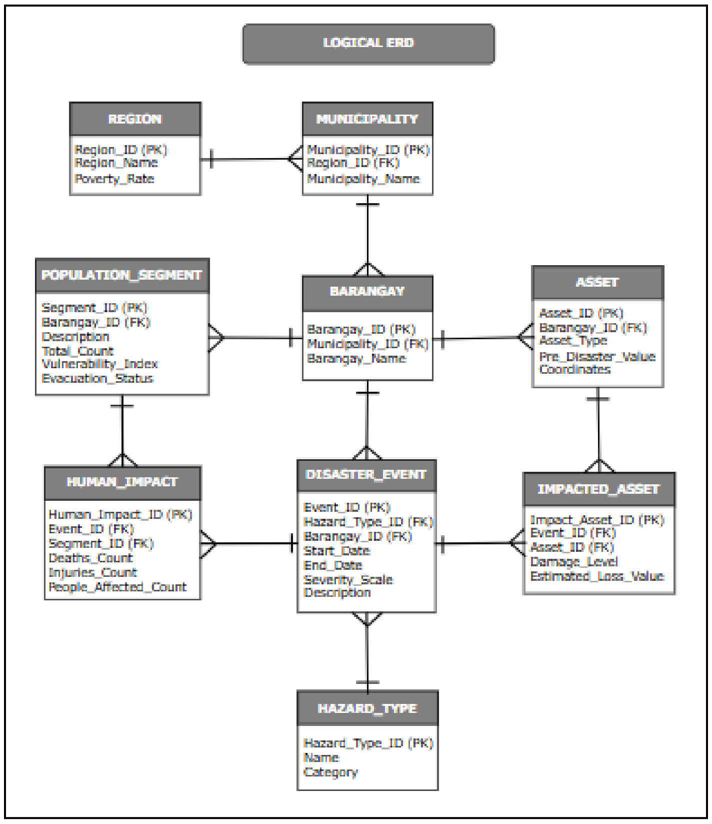
This is the Conceptual ERD that outlines the core entities and their relationships, reflecting both administrative structures and disaster impact analysis.

- **Region** - Represents a broad geographic area that encompasses multiple municipalities.
- **Municipality** - Contains multiple barangay.
- **Barangay** - Each barangay is linked to population segments and assets for detailed vulnerability analysis.
- **Population_Segment** - Used to identify vulnerable populations and tailor disaster response strategies.
- **Asset** - Refers to critical infrastructure, facilities, or resources within a barangay.
- **Human_Impact** - It explains how the disaster affected different population groups and highlights the real human consequences, including lives lost, injuries, and other impacts.

- **Disaster_Event** - A disaster record captures each event in a specific place, enabling tracking of when, where, and what type of disaster occurred.
- **Impacted_Asset** - A list of assets damaged by a disaster and shows the financial losses and helps prioritize repairs.
- **Hazard_Type** - A list of disaster types that commonly occur in the Philippines helps categorize events during data analysis, such as comparing landslides with earthquakes.

These are interconnected by one-to-many relationships that reflect real administrative and disaster structures. A region consists of many municipalities, and each municipality is further subdivided into many barangays, which are the smallest unit of analysis. Each barangay will be associated with many population segments and many assets, representing vulnerable groups and critical infrastructure within the system. Disaster events impact many barangays and are categorized under a single hazard type, while the consequences of these events are represented through human impacts and impacted assets.

- **LOGICAL ERD**



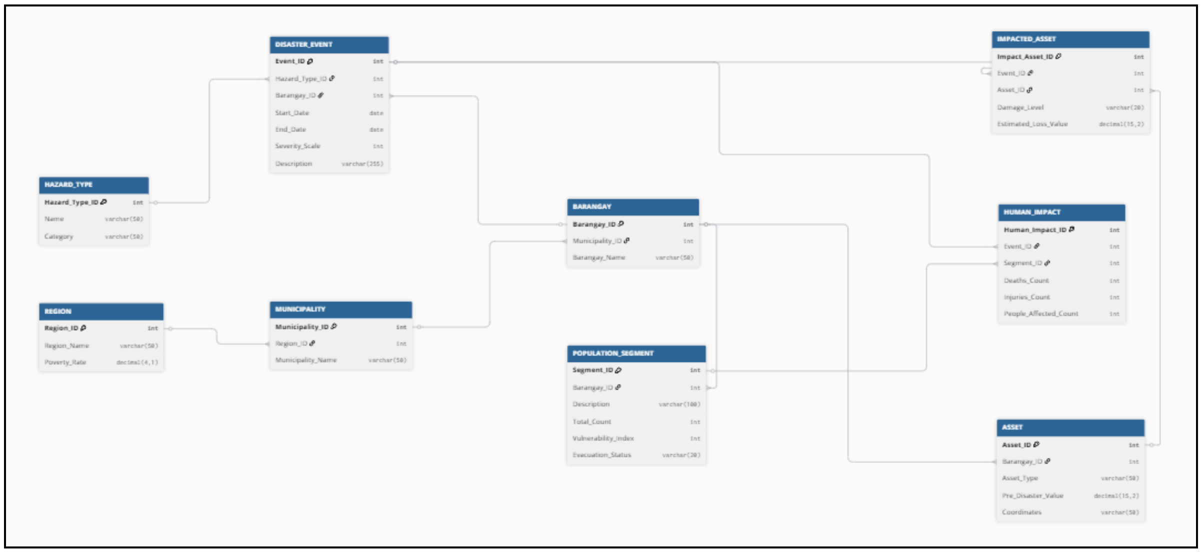
View a clearer version at this link: LOGICAL ERD.pdf

The logical model consists of normalized entities and attributes with relational integrity. Each entity will have a primary key for unique identification and foreign keys to establish one-to-many relationships.

Region (Region_ID PK) is related to Municipality (Region_ID FK), which then relates to Barangay (Municipality_ID FK). Barangay (Barangay_ID PK) is the central entity since it relates both to Asset and Population_Segment through Barangay_ID FKs. Assets (Asset_ID PK) record infrastructure information, while population segments hold demographic data and information on vulnerability.

Disaster_Event (Event_ID PK) is linked to Barangay and Hazard_Type via respective FKs, recording dates, severity, and descriptions. Human_Impact (Human_Impact_ID PK) connects events and population segments, storing information on deaths, injuries, and affected counts. Impacted_Asset (Impact_Asset_ID PK) links events to assets, recording damage and loss values. Finally, Hazard_Type (Hazard_Type_ID PK) classifies disasters by name and category.

- **PHYSICAL ERD**



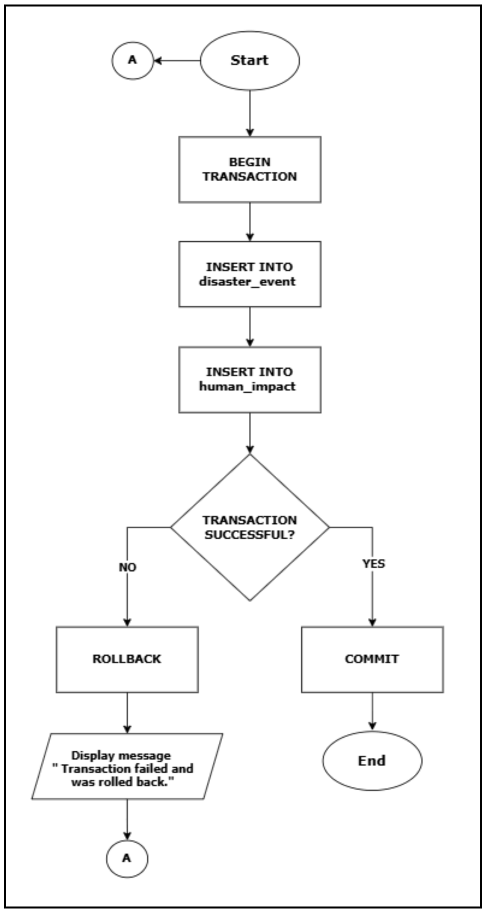
View a clearer version and DBML used at this link: PHYSICAL ERD & DBML

This Physical ERD has entities, attributes, primary keys, foreign keys, and defined data types. The IDs like Region_ID, Municipality_ID, Barangay_ID, Event_ID, and others utilize int for efficient indexing. Names and descriptions are stored as varchar of varying lengths such as

50, 100, 255 for flexible text. Numeric values like poverty rates and financial losses use decimal for precision, while counts of people, injuries, and deaths use int for accuracy. Dates are captured with the date type to track disaster timelines.

3.3. Transaction Flowchart

Below this section is the flowchart that represents the transactional stored procedure for handling disaster event data and its related human impact records. It visually demonstrates how the system ensures data consistency by committing successful operations or rolling back entirely in case of errors.



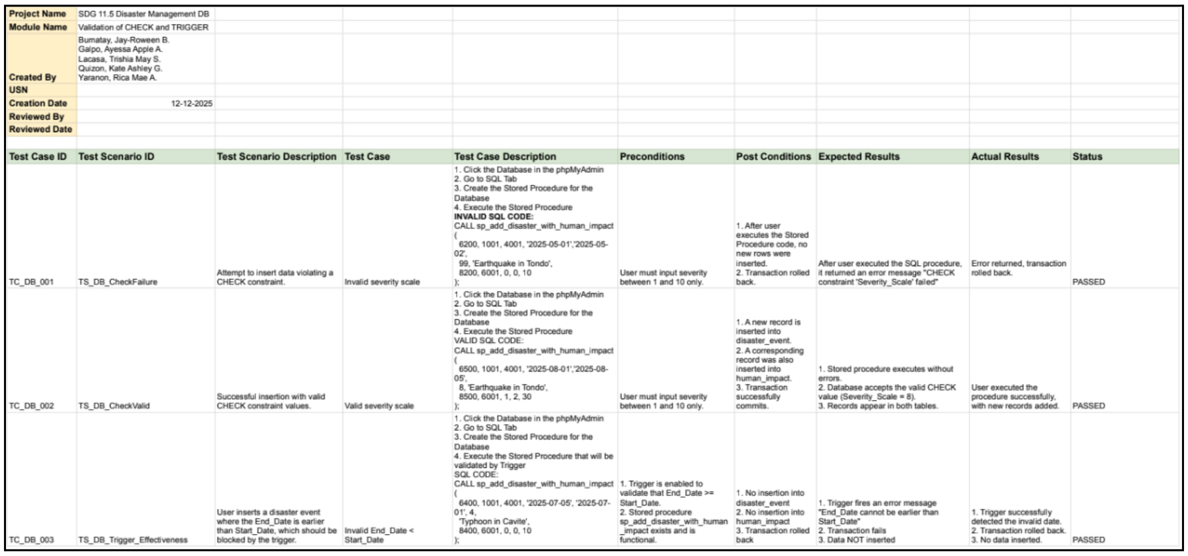
View a clearer version at this link: TRANSACTION FLOWCHART.pdf

The process begins with a "Start" node, followed by a "Begin Transaction" step that initiates the SQL transaction block. Two consecutive process blocks insert data into the disaster_event and human_impact tables, respectively. After these operations, the flow reaches a decision point labeled "Transaction Successful?" which checks whether any SQL exception occurred during the inserts. If the transaction fails, the flow proceeds to a "Rollback" block to undo all changes, followed by a display message stating "Transaction failed and was rolled back." Notably, a connector is used here to redirect the flow back to the "Start" node. This connector simplifies the diagram by avoiding long or overlapping arrows and reinforces the logical loop that allows the system to retry or exit cleanly. If no error occurs, the flow continues to a "Commit" block, finalizing the transaction, and ends with a successful termination. The use of connectors enhances clarity and modularity in the flowchart, especially in handling rollback scenarios.

IV. Testing and Results

4.1. Test Cases

The validation of the SDG 11.5 Disaster Management DB check and trigger mechanisms was carried out through three structured test scenarios. The first scenario assessed the system's ability to enforce constraints by rejecting an invalid severity scale input, which correctly produced an error and rolled back the transaction. The second scenario demonstrated successful insertion when valid severity values were provided, resulting in proper updates to both the disaster and human impact tables. The third scenario confirmed the trigger's reliability by preventing the insertion of records where the end date was earlier than the start date.



Project Name	SDG 11.5 Disaster Management DB
Module Name	Validation of CHECK and TRIGGER
Created By	Bumatay, Jay-Roween B. Galpo, Ayessa Apple A. Lacasa, Trishia May S. Quizon, Kate Ashley G. Yaranon, Rica Mae A.
USN	
Creation Date	12-12-2025
Reviewed By	
Reviewed Date	

Test Case ID	Test Scenario ID	Test Scenario Description	Test Case	Test Case Description	Preconditions	Post Conditions	Expected Results	Actual Results	Status
TC_DB_001	TS_DB_CheckFailure	Attempt to insert data violating a CHECK constraint.	Invalid severity scale	1. Click the Database in the phpMyAdmin 2. Go to SQL Tab 3. Create the Stored Procedure for the Database 4. Execute the Stored Procedure **INVALID SQL CODE:** CALL sp_add_disaster_with_human_impact (6200, 1001, 4001, '2025-05-01','2025-05-02', 99, 'Earthquake in Tondo', 8200, 6001, 0, 0, 10);	User must input severity between 1 and 10 only.	1. After user executes the Stored Procedure code, no new rows were inserted. 2. Transaction rolled back.	After user executed the SQL procedure, it returned an error message "CHECK constraint 'Severity_Scale' failed"	Error returned, transaction rolled back.	PASSED
TC_DB_002	TS_DB_CheckValid	Successful insertion with valid CHECK constraint values.	Valid severity scale	1. Click the Database in the phpMyAdmin 2. Go to SQL Tab 3. Create the Stored Procedure for the Database 4. Execute the Stored Procedure VALID SQL CODE: CALL sp_add_disaster_with_human_impact (6500, 1001, 4001, '2025-08-01','2025-08-05', 8, 'Earthquake in Tondo', 8500, 6001, 1, 2, 30);	User must input severity between 1 and 10 only.	1. A new record is inserted into disaster_event. 2. A corresponding record was also inserted into human_impact. 3. Transaction successfully commits.	1. Stored procedure executes without errors. 2. Database accepts the valid CHECK value (Severity_Scale = 8). 3. Records appear in both tables.	User executed the procedure successfully, with new records added.	PASSED
TC_DB_003	TS_DB_Trigger_Effectiveness	User inserts a disaster event where the End_Date is earlier than Start_Date, which should be blocked by the trigger.	Invalid End_Date < Start_Date	1. Click the Database in the phpMyAdmin 2. Go to SQL Tab 3. Create the Stored Procedure for the Database 4. Execute the Stored Procedure that will be validated by Trigger SQL CODE: CALL sp_add_disaster_with_human_impact (6400, 1001, 4001, '2025-07-05', '2025-07-01', 4, 'Typhoon in Cavite', 8400, 6001, 0, 0, 10);	1. Trigger is enabled to validate that End_Date >= Start_Date. 2. Stored procedure sp_add_disaster_with_human_impact exists and is functional.	1. No insertion into disaster_event 2. No insertion into human_impact 3. Transaction rolled back	1. Trigger fires an error message "End_Date cannot be earlier than Start_Date" 2. Transaction fails 3. Data NOT inserted	1. Trigger successfully detected the invalid date. 2. Transaction rolled back. 3. No data inserted.	PASSED

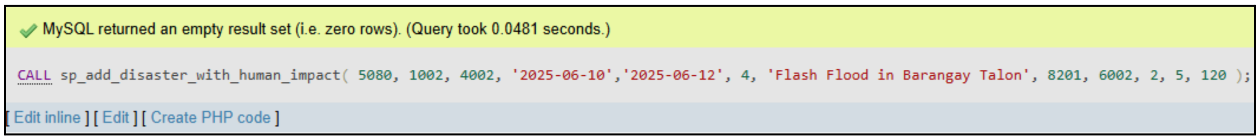
View a clearer version at this link: TEST CASE.pdf

4.2. ACID Compliance Test

The ACID Compliance Test section presents actual screenshots to verify that the FR3 transactional procedure satisfies all ACID properties. In particular, the ROLLBACK demonstration confirms atomicity, while the accompanying outputs provide clear evidence of consistency, isolation, and durability."

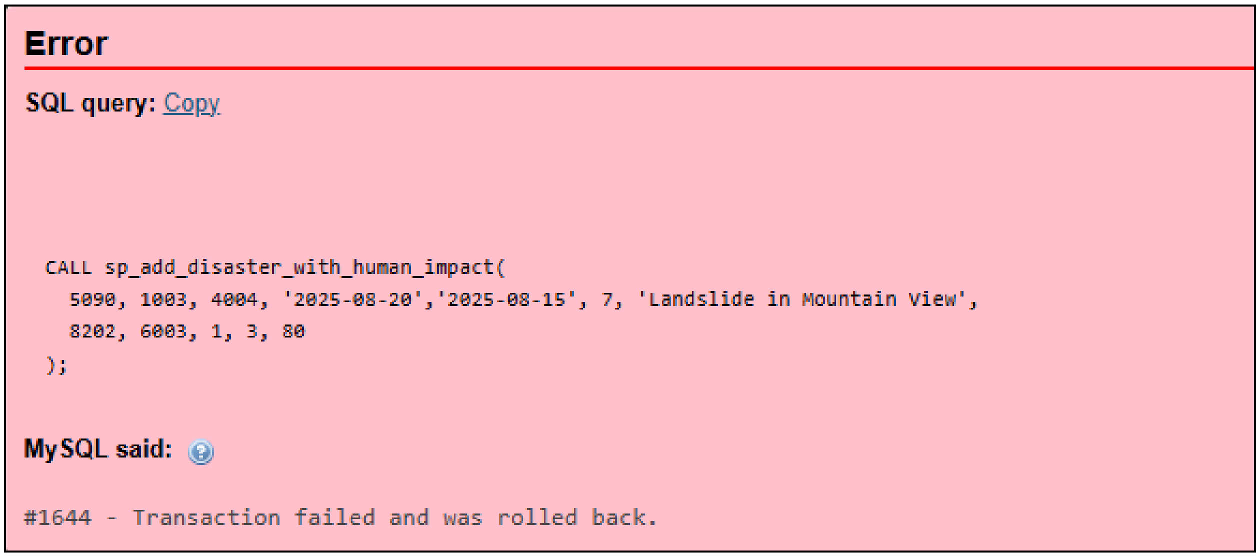
A. ATOMICITY

- Successful COMMIT



```
✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0481 seconds.)

CALL sp_add_disaster_with_human_impact( 5080, 1002, 4002, '2025-06-10','2025-06-12', 4, 'Flash Flood in Barangay Talon', 8201, 6002, 2, 5, 120 );

[ Edit inline ] [ Edit ] [ Create PHP code ]
```

- Rollback (Unsuccessful COMMIT)



```
Error

SQL query: Copy

  CALL sp_add_disaster_with_human_impact(
    5090, 1003, 4004, '2025-08-20','2025-08-15', 7, 'Landslide in Mountain View',
    8202, 6003, 1, 3, 80
  );

MySQL said:

#1644 - Transaction failed and was rolled back.
```

B. CONSISTENCY

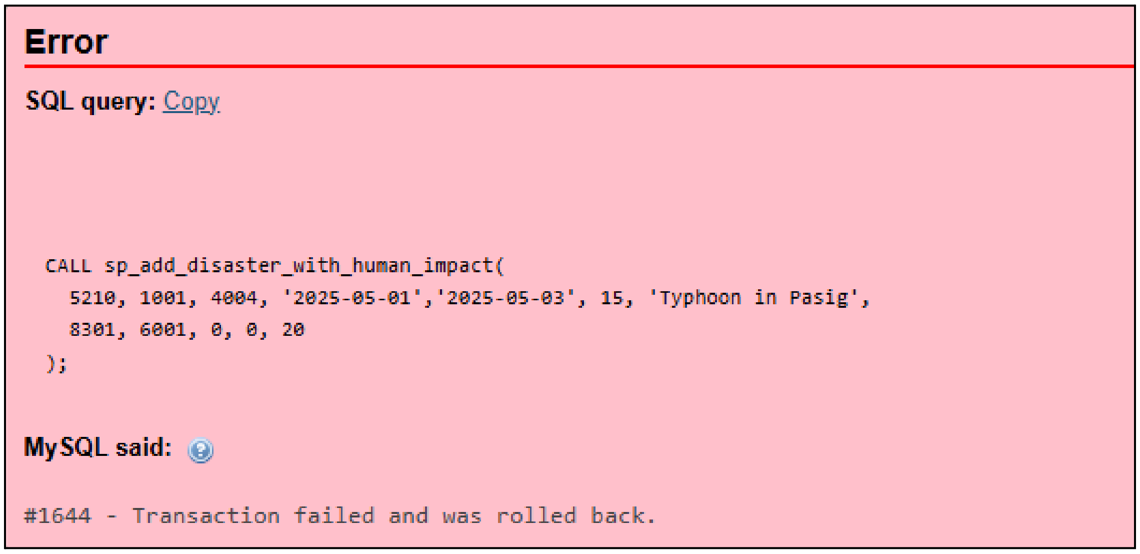
1. CHECK CONSTRAINT

SQL Code applying the CHECK Constraint in a specific table
CREATE TABLE \`disaster_event\` (\`Event_ID\` int(11) NOT NULL, \`Hazard_Type_ID\` int(11) NOT NULL, \`Barangay_ID\` int(11) NOT NULL,

```
`Start_Date` date NOT NULL,
`End_Date` date NOT NULL,
`Severity_Scale` int(11) NOT NULL CHECK (`Severity_Scale` between 1 and 10),
`Description` varchar(255) DEFAULT NULL
);
```

➢ The **CHECK (`Severity_Scale` between 1 and 10)** is declared in the disaster_event table, indicating that the inserted severity scale of every record must only range from 1-10 only. If they inserted a negative value of greater than 10, it will display an error message and rollback.

- **INVALID INPUT OF CHECK CONSTRAINT**



Error

SQL query: Copy

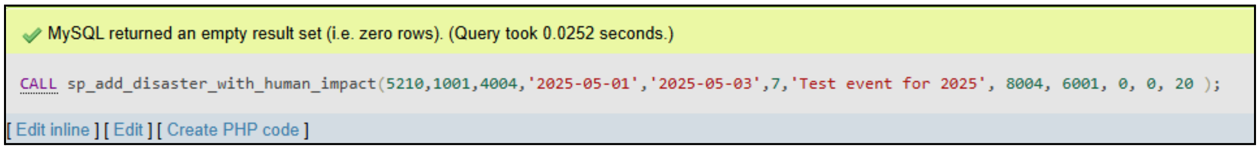
```
CALL sp_add_disaster_with_human_impact(
  5210, 1001, 4004, '2025-05-01','2025-05-03', 15, 'Typhoon in Pasig',
  8301, 6001, 0, 0, 20
);
```

MySQL said:

```
#1644 - Transaction failed and was rolled back.
```

➢ The record attempted to be inserted has an invalid severity scale vale, which is 15 (greater than 10), so the transaction will be rolled back.

- **VALID INPUT OF CHECK CONSTRAINT**



MySQL returned an empty result set (i.e. zero rows). (Query took 0.0252 seconds.)

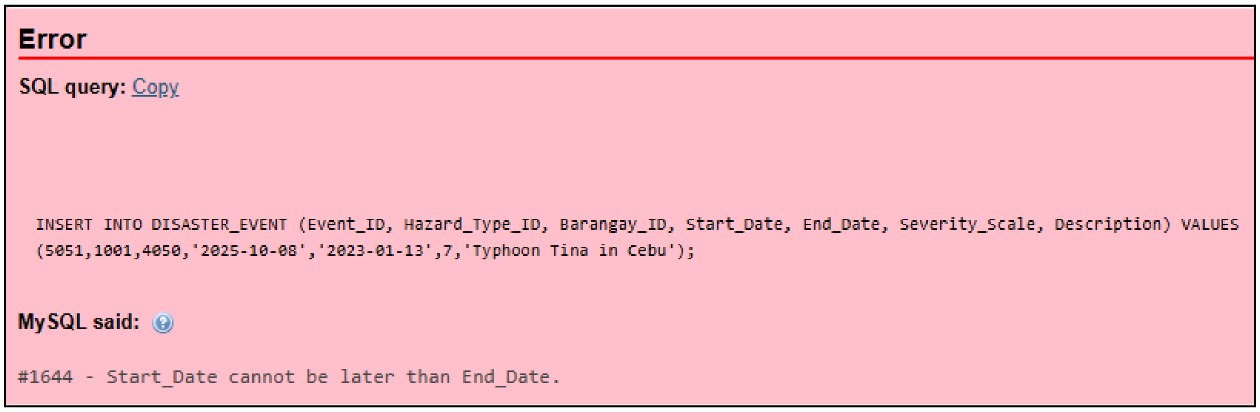
```
CALL sp_add_disaster_with_human_impact(5210,1001,4004,'2025-05-01','2025-05-03',7,'Test event for 2025', 8004, 6001, 0, 0, 20 );
```

[Edit inline] [Edit] [Create PHP code]

➢ The record was successfully inserted, following the rules that severity level must be between 1 to 10 only.

2. TRIGGER CONSTRAINT

○ INVALID INPUT



Error

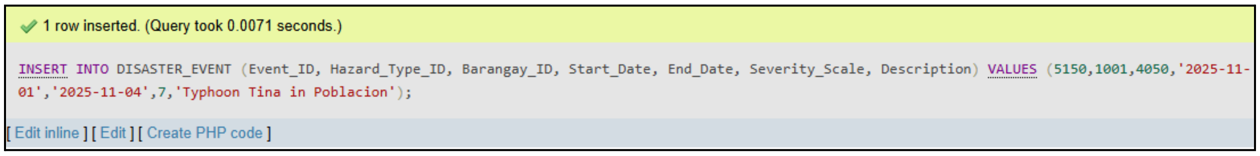
SQL query: Copy

```
INSERT INTO DISASTER_EVENT (Event_ID, Hazard_Type_ID, Barangay_ID, Start_Date, End_Date, Severity_Scale, Description) VALUES
(5051,1001,4050,'2025-10-08','2023-01-13',7,'Typhoon Tina in Cebu');
```

MySQL said:

```
#1644 - Start_Date cannot be later than End_Date.
```

○ VALID INPUT



✓ 1 row inserted. (Query took 0.0071 seconds.)

```
INSERT INTO DISASTER_EVENT (Event_ID, Hazard_Type_ID, Barangay_ID, Start_Date, End_Date, Severity_Scale, Description) VALUES (5150,1001,4050,'2025-11-
01','2025-11-04',7,'Typhoon Tina in Poblacion');
```

[Edit inline] [Edit] [Create PHP code]

C. ISOLATION

- ○ Isolation in ACID Properties happens when an individual ransaction runs as if it is the only one happening, even when other users are also inserting, updating, or deleting data in a database
- ○ There is no interference and no mixing of changes.

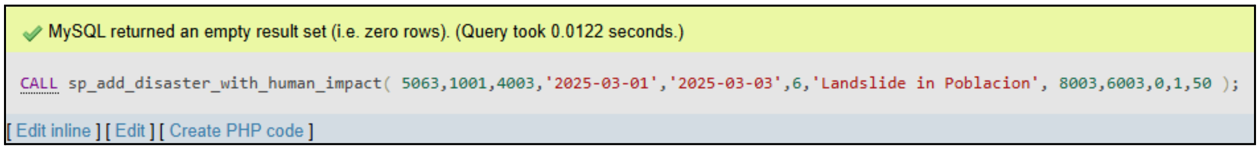
SAMPLE SQL CODE
CALL sp_add_disaster_with_human_impact(5060, 1001, 4003, '2025-03-01','2025-03-03', 6, 'Typhoon Tina in Cebu', 8100, 6001, 0, 1, 50);

➢ This code follows the isolation part of ACID because it runs as a single transaction, only one user is inserting this code and in this way, it prevents other transactions made from other users from interfering.

D. DURABILITY

- Once a transaction is successfully committed, the data will stay saved permanently, even if the database crashes.

STEP 1: SUCCESSFULLY COMMITTED TRANSACTION

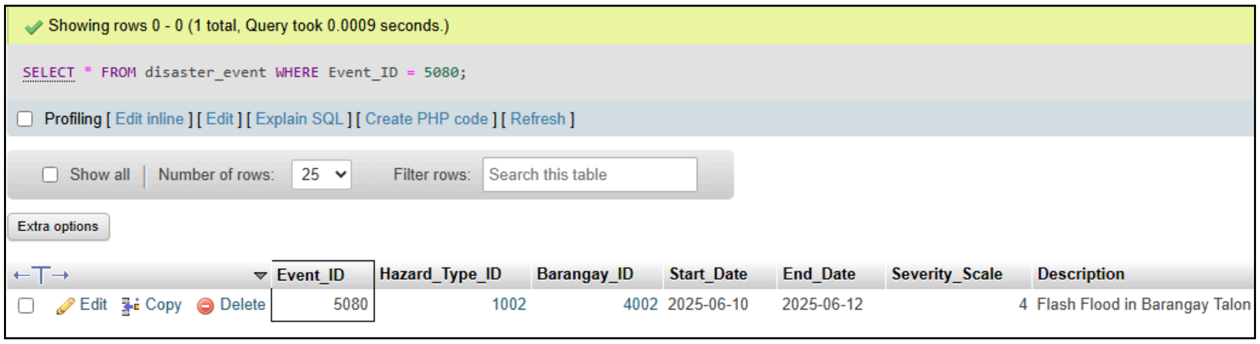


MySQL returned an empty result set (i.e. zero rows). (Query took 0.0122 seconds.)

```
CALL sp_add_disaster_with_human_impact( 5063,1001,4003,'2025-03-01','2025-03-03',6,'Landslide in Poblacion', 8003,6003,0,1,50 );
```

STEP 2: CLOSE THE phpMyAdmin AND TRY TO OPEN IT AGAIN

STEP 3: TRY TO FIND THE SQL CODE INSERTED EARLIER



Showing rows 0 - 0 (1 total, Query took 0.0009 seconds.)

```
SELECT * FROM disaster_event WHERE Event_ID = 5080;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table
Extra options
Edit Copy Delete

Event_ID	Hazard_Type_ID	Barangay_ID	Start_Date	End_Date	Severity_Scale	Description
5080	1002	4002	2025-06-10	2025-06-12	4	Flash Flood in Barangay Talon

➢ The rows are still there, demonstrating the Durability of the Database.

V. Conclusion and Contributions

5.1. Conclusion

The development of the DBMS for Sustainable Development successfully demonstrates how database technology can be applied to address real-world disaster management challenges. By aligning with the United Nations Sustainable Development Goal 11.5, the system provides a structured and reliable way of organizing disaster-related data, ensuring that affected families, vulnerable groups, and damaged assets are properly recorded and monitored. Through normalization, constraints, triggers, and stored procedures, the project guarantees data integrity, transparency, and accountability, which are critical in disaster response and recovery.

Furthermore, the system's design highlights the importance of efficiency and accuracy in reporting. The use of VIEWS simplifies complex queries, enabling faster generation of disaster summaries and economic loss reports, while stored procedures enforce ACID compliance to maintain consistency and reliability in sensitive operations. These features ensure that local governments and NGOs can make informed decisions quickly, reducing delays in aid distribution and improving preparedness for future emergencies. The database structure, with its clear relationships among regions, municipalities, barangays, and population segments, reflects real-world administrative and disaster frameworks, making the system both practical and scalable.

Ultimately, this project contributes to building safer and more resilient communities by bridging the gap between disaster response and recovery. It empowers stakeholders with accurate, timely, and organized information, allowing them to prioritize resources, protect vulnerable populations, and strengthen infrastructure. By combining technical precision with social responsibility, the DBMS not only meets academic requirements but also serves as a meaningful solution that can be expanded for real-world disaster management applications.

5.2. Individual Contributions

Each member actively contributes to the project, ensuring that every section reflects individual input and collaborative effort. The detailed contributions of all members are systematically documented and presented in the table below, highlighting specific responsibilities, and tasks completed.

Name	Contributions
Bumatay, Jay-Roween B.	● 2.1. Functional Requirements and Non-Functional Requirements ● 3.1. Core DBMS Concepts Used ● 3.3. Transaction Flowchart ● GitHub
Galpo, Ayessa Apple A.	● 1.1. Project Overview & UN SDG Target ● 2.2. Data Requirements ● 3.2. ER Diagram ● PPT
Lacasa, Trishia May S.	● 1.1. Project Overview & UN SDG Target ● 3.2. ER Diagram ● 3.3. Transaction Flowchart ● IV. Testing and Results (4.1 & 4.2)
Quizon, Kate Ashley G.	● 1.2. Problem Statement ● 2.3. Schema Normalization Analysis ● 3.1. Core DBMS Concepts Used ● IV. Testing and Results (4.1 & 4.2) ● SQL DATABASE
Yaranon, Rica Mae A.	● 1.2. Problem Statement ● 2.2. Data Requirements ● 5.1. Conclusion ● PPT