**University of Niagara Falls Canada**

**Master of Data Analytics**

**SQL Databases (CPSC-500-5)**

**Fall 2024**

**Final Group Project - Database Design and Implementation**

**For**

**NGO Operations**

**Due date: Monday Dec. 9th, 11:59 AM**

**Submitted By**

**Group-9**

**Sridevi Pemmasani (NF1002949)**

**Chandan Rana (NF1004077)**

**Setuben Chaudhari (NF1009018)**

**Professor: Omid Isfahanialamdari**

December 2024

Table of Contents

[1 Introduction 2](#_Toc184628107)

[2 Objectives 3](#_Toc184628108)

[3 Database Design 3](#_Toc184628109)

[3.1 Entity-Relationship Diagram (ERD) 3](#_Toc184628110)

[3.2 Description of key tables and their relationships 4](#_Toc184628111)

[3.2.1 Strong Entities 4](#_Toc184628112)

[3.2.2 Weak Entities 7](#_Toc184628113)

[3.2.3 Strong Relationships 8](#_Toc184628114)

[3.2.4 Weak Relationships 9](#_Toc184628115)

[4 Data Definition Language Statements (DDL) 9](#_Toc184628116)

[5 Data Manipulation Language Statements (DML) 17](#_Toc184628117)

[6 Basic Data Retrieval Statements (DQL) 18](#_Toc184628118)

[7 Advanced SQL Functions - Aggregations using SQL 29](#_Toc184628119)

[8 Advanced SQL Analytics - Windows functions, subqueries 30](#_Toc184628120)

[9 CTEs 49](#_Toc184628121)

[10 Views 55](#_Toc184628122)

[11 Stored Procedures 60](#_Toc184628123)

[12 Technology Stack 68](#_Toc184628124)

[12.1 Database Management System (DBMS) 68](#_Toc184628125)

[12.2 SQL (Structured Query Language) 69](#_Toc184628126)

[12.3 MySQL Server 69](#_Toc184628127)

[12.4 Draw.io 69](#_Toc184628128)

[13 Conclusion 69](#_Toc184628129)

[14 References 70](#_Toc184628130)

# Introduction

A Non-Governmental Organization (NGO) is a nonprofit, voluntary group organized to address social, cultural, environmental, or humanitarian issues. NGOs operate independently of government influence, although they may receive government funding in some cases. Their primary goal is to improve societal conditions by tackling issues such as poverty, education, healthcare, environmental conservation, disaster relief, and human rights.

NGOs serve as a bridge between communities and resources, working to ensure equitable distribution of resources and opportunities. They play a crucial role in society by advocating for the marginalized, fostering social change, and addressing gaps that public and private sectors cannot fully cover.

# Objectives

This project aims to design and implement a comprehensive database system to streamline the operational processes of an NGO. The database will centralize and manage information related to staff, volunteers, donors, beneficiaries, projects, activities, expenses, and feedback. The system ensures data consistency, efficient tracking of resources and efforts, and provides actionable insights for decision-making. It will support the NGO in managing its diverse projects and activities, enhance transparency, and foster better collaboration among stakeholders

# Database Design

## Entity-Relationship Diagram (ERD)

A diagram of a computer

Description automatically generated

Figure 1: ER Diagram for NGO Operations

## Description of key tables and their relationships

### Strong Entities

* + - 1. Staff:

Description: Represents the permanent employees of the NGO who play various roles in managing operations and implementing projects.

Attributes:

StaffID: Unique identifier for each staff member.

FirstName, LastName: Name of the staff member.

RoleID: Specifies the role of the staff (e.g., Program Manager, Finance Manager).

Email, PhoneNumber: Contact details of the staff.

HireDate: The date the staff member joined the NGO.

* + - 1. Staff Role:

Description: Defines various roles that staff members can have within the NGO, such as NGO Head, Fundraising Manager, or Operations Manager.

Attributes:

RoleID: Unique identifier for each role.

RoleName: Name of the role.

Description: Brief description of the responsibilities of the role.

* + - 1. Volunteers:

Description: Represents individuals who contribute their time and skills to the NGO's projects and activities on a voluntary basis.

Attributes:

VolunteerID: Unique identifier for each volunteer.

FirstName, LastName: Name of the volunteer.

Email, PhoneNumber: Contact details.

Availability: Indicates whether the volunteer is currently available (Yes/No).

JoinDate: The date the volunteer started working with the NGO.

* + - 1. Projects:

Description: Represents the various initiatives undertaken by the NGO to achieve its mission.

Attributes:

ProjectID: Unique identifier for each project.

ProjectName: Name of the project (e.g., Community Health Initiative).

ProjectDescription: Detailed description of the project.

StartDate, EndDate: The duration of the project.

Budget: Financial allocation for the project.

Status: Current status of the project (e.g., Active, Completed, On-Hold).

* + - 1. ProjectActivities:

Description: Represents specific activities that fall under a project. Each project can have multiple activities.

Attributes:

ActivityID: Unique identifier for each activity.

ProjectID: References the project to which the activity belongs.

ActivityName: Name of the activity.

ActivityDescription: Description of the activity.

* + - 1. Donors:

Description: Represents individuals or organizations providing financial contributions to the NGO.

Attributes:

DonorID: Unique identifier for each donor.

FirstName, LastName: Name of the donor.

Email, PhoneNumber: Contact details.

Address: Address of the donor.

* + - 1. Beneficiaries:

Description: Represents the individuals or groups who benefit from the NGO’s projects.

Attributes:

BeneficiaryID: Unique identifier for each beneficiary.

FirstName, LastName: Name of the beneficiary.

Email: Contact details.

DOB: Date of birth.

Gender: Gender of the beneficiary.

IncomeLevel: Socio-economic classification.

FamilySize: Number of family members.

Address: Address of the beneficiary.

* + - 1. Expenses:

Description: Tracks the financial expenditure associated with each project.

Attributes:

ExpenseID: Unique identifier for each expense.

ProjectID: References the project for which the expense was incurred.

ExpenseDescription: Description of the expense (e.g., supplies, transportation).

Amount: Cost of the expense.

ExpenseDate: Date when the expense was made.

### Weak Entities

* + - 1. VolunteerHours:

Description: Tracks the hours contributed by volunteers to specific project activities.

Attributes:

VolunteerID: References the volunteer.

ActivityID: References the specific project activity.

WorkDate: Date when the volunteer contributed their time.

HoursWorked: Total hours worked by the volunteer on that date.

* + - 1. Beneficiaries\_Feedback:

Description: Captures feedback provided by beneficiaries on the projects they were part of.

Attributes:

BeneficiaryID: References the beneficiary providing the feedback.

ProjectID: References the project for which feedback is given.

FeedbackDate: Date the feedback was submitted.

Feedback: The content of the feedback.

* + - 1. Donations:

Description: Tracks financial contributions made by donors to specific projects.

Attributes:

DonationID: Unique identifier for each donation.

DonorID: References the donor who made the donation.

ProjectID: References the project the donation is intended for.

DonationAmount: Amount donated.

DonationDate: Date of the donation.

### Strong Relationships

* + - 1. Staff manages Projects (Staff to Projects, 1:N)
      2. Projects include ProjectActivities (Projects to ProjectActivities, 1:N)
      3. Donors contribute to Projects through Donations (Donors to Donations, N:M via Projects)
      4. Beneficiaries provide Feedback on Projects (Beneficiaries to Feedback, N:M via Projects)
      5. Volunteers contribute hours to ProjectActivities (Volunteers to VolunteerHours, N:M via ProjectActivities)

### Weak Relationships

* + - 1. VolunteerHours is associated with Volunteer and Activities (N:M:M relationship with Volunteers and Activities)
      2. Donations depend on Donors and Projects
      3. Beneficiaries\_Feedback depends on Beneficiaries and Projects

# Data Definition Language Statements (DDL)

**/\*\*Create Database and tables for the NGO database\*\*/**

CREATE DATABASE ngooperations;

USE ngooperations;

**-- Creating StaffRole table**

CREATE TABLE StaffRole (

RoleID INT AUTO\_INCREMENT PRIMARY KEY,

RoleName VARCHAR(50) UNIQUE NOT NULL,

Description VARCHAR(255)

);

**-- Creating Staff table**

CREATE TABLE Staff (

StaffID INT PRIMARY KEY AUTO\_INCREMENT,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

RoleID INT,

Email VARCHAR(100) UNIQUE NOT NULL,

PhoneNumber VARCHAR(15),

HireDate DATE,

FOREIGN KEY (RoleID) REFERENCES StaffRole(RoleID) ON DELETE SET NULL

);

**-- Creating Volunteers table**

CREATE TABLE Volunteers (

VolunteerID INT AUTO\_INCREMENT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL,

PhoneNumber VARCHAR(15),

Availability ENUM('Yes', 'No') NOT NULL DEFAULT 'No',

JoinDate DATE

);

**-- Creating Donors table**

CREATE TABLE Donors (

DonorID INT PRIMARY KEY AUTO\_INCREMENT,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL,

PhoneNumber VARCHAR(15),

Address TEXT

);

**-- Creating Beneficiaries table**

CREATE TABLE Beneficiaries (

BeneficiaryID INT PRIMARY KEY AUTO\_INCREMENT,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL,

DOB DATE,

Gender ENUM('Male', 'Female','Other') NOT NULL,

IncomeLevel ENUM('Low', 'Medium', 'High'),

FamilySize INT CHECK (FamilySize >= 0),

Address TEXT

);

**-- Creating Projects table**

CREATE TABLE Projects (

ProjectID INT PRIMARY KEY AUTO\_INCREMENT,

ProjectName VARCHAR(100) NOT NULL,

ProjectDescription VARCHAR(255),

StartDate DATE,

EndDate DATE,

Budget DECIMAL(15, 2) CHECK (Budget >= 0),

Status ENUM('Planned', 'On-Going', 'Completed','Cancelled','Paused') NOT NULL,

CHECK (StartDate IS NULL OR EndDate IS NULL OR StartDate <= EndDate)

);

**-- Creating ProjectActivities table**

CREATE TABLE ProjectActivities (

ActivityID INT PRIMARY KEY AUTO\_INCREMENT,

ProjectID INT,

ActivityName VARCHAR(100) NOT NULL,

ActivityDescription VARCHAR(255),

FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID) ON DELETE SET NULL,

INDEX projectactivities\_project\_idx (ProjectID)

);

**-- Creating StaffProjects table**

CREATE TABLE StaffProjects (

StaffID INT,

ProjectID INT,

FromDate DATE NOT NULL,

ToDate DATE,

PRIMARY KEY (StaffID,ProjectID,FromDate),

FOREIGN KEY (StaffID) REFERENCES Staff(StaffID) ON DELETE CASCADE,

FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID) ON DELETE CASCADE,

CHECK (ToDate IS NULL OR FromDate <= ToDate)

);

**-- Creating Expenses table**

CREATE TABLE Expenses (

ExpenseID INT PRIMARY KEY AUTO\_INCREMENT,

ExpenseDescription VARCHAR(255) NOT NULL,

Amount DECIMAL(10, 2) CHECK (Amount >= 0),

ExpenseDate DATE,

ProjectID INT NOT NULL,

FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID) ON DELETE CASCADE,

INDEX expenses\_project\_idx (ProjectID)

);

**-- Creating VolunteerHours table**

CREATE TABLE VolunteerHours (

VolunteerID INT,

ActivityID INT,

WorkDate DATE NOT NULL,

HoursWorked INT CHECK (HoursWorked BETWEEN 0 AND 8),

PRIMARY KEY (VolunteerID,ActivityID,WorkDate),

FOREIGN KEY (VolunteerID) REFERENCES Volunteers(VolunteerID) ON DELETE CASCADE,

FOREIGN KEY (ActivityID) REFERENCES ProjectActivities(ActivityID) ON DELETE CASCADE,

INDEX volunteerhours\_volunteer\_idx (VolunteerID),

INDEX volunteerhours\_projectactivities\_idx (ActivityID),

INDEX volunteerhours\_volunteer\_projectactivities\_idx (VolunteerID, ActivityID)

);

**-- Creating Donations table**

CREATE TABLE Donations (

DonationID INT PRIMARY KEY AUTO\_INCREMENT,

DonorID INT,

DonationAmount DECIMAL(10, 2) CHECK (DonationAmount >= 0),

DonationDate DATE NOT NULL,

ProjectID INT,

FOREIGN KEY (DonorID) REFERENCES Donors(DonorID) ON DELETE CASCADE,

FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID) ON DELETE SET NULL,

INDEX donations\_donor\_idx (DonorID),

INDEX donations\_project\_idx (ProjectID),

INDEX Donations\_donor\_project\_idx (DonorID, ProjectID)

);

**-- Creating Beneficiaries\_Feedback table**

CREATE TABLE Beneficiaries\_Feedback (

BeneficiaryID INT,

ProjectID INT,

FeedbackDate DATE NOT NULL,

Feedback VARCHAR(255),

PRIMARY KEY (BeneficiaryID,ProjectID,FeedbackDate),

FOREIGN KEY (BeneficiaryID) REFERENCES Beneficiaries(BeneficiaryID) ON DELETE CASCADE,

FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID) ON DELETE CASCADE,

INDEX feedback\_beneficiary\_idx (BeneficiaryID),

INDEX feedback\_project\_idx (ProjectID),

INDEX beneficiaries\_feedback\_beneficiary\_project\_idx (BeneficiaryID,ProjectID)

);

**/\* Additional DDL Statements \*/**

-- Q26: Add a new column Status to the projects table, verify and remove the new column after testing

ALTER TABLE projects ADD Status1 VARCHAR(50);

SELECT \* from projects LIMIT 5;

ALTER TABLE projects DROP COLUMN Status1;

SELECT \* from projects LIMIT 5;

-- Q27: Create a new table based on other and Drop after testing.

CREATE TABLE staffCopy LIKE staff;

SELECT \* FROM staffCopy;

DROP TABLE staffCopy;

# Data Manipulation Language Statements (DML)

**/\* DML Statements \*/**

-- Q28:Insert a new staff record.

INSERT INTO staff (FirstName, LastName, Email, PhoneNumber, HireDate, RoleID)

VALUES ('Sridevi', 'Pemmasani', 'sridevi.pemmasani@examplengo.com', '1234567890', '2024-01-01', 1);

-- Q29: Update the Budget of a project with ProjectID = 103.

SELECT Budget from projects where ProjectID = 1;

UPDATE projects SET Budget = 200000 WHERE ProjectID = 1;

-- Q30: Delete a beneficiary with BeneficiaryID = 15.

DELETE FROM beneficiaries WHERE BeneficiaryID = 15;

# Basic Data Retrieval Statements (DQL)

**/\* Basic Data Retrieval Statements (DQL) \*/**

-- Q1: Retrieve all data from Staff

SELECT \* FROM staff;

-- Q2: Retrieve all projects with their ProjectName and Budget.

SELECT

ProjectName,

Budget

FROM

projects;

-- Q3: Fetch the FirstName, LastName, and Email of volunteers.

SELECT

FirstName,

LastName,

Email

FROM

volunteers;

/\* **WHERE QUERIES** \*/

SELECT \*

FROM

staff

WHERE

HireDate > '2023-01-01';

-- Q5: Retrieve projects where the budget exceeds $100,000.

SELECT \*

FROM

projects

WHERE

Budget > 100000;

-- Q6: Find all donors who donated more than $1,000.

SELECT \*

FROM

donations

WHERE

DonationAmount > 4000;

/\* **LIMIT QUERIES** \*/

SELECT \*

FROM

projects

ORDER BY StartDate ASC

LIMIT 5;

-- Q8: Retrieve the top 10 donors by DonationAmount.

SELECT \*

FROM

donations

ORDER BY DonationAmount DESC

LIMIT 10;

/\* **Arithmetic Operations** \*/

SELECT

SUM(Budget) AS TotalBudget

FROM

projects;

-- Q10: Find the average donation per donor.

SELECT

AVG(DonationAmount) AS AvgDonation

FROM

donations;

-- Q11: Increase all DonationAmount by 10% and display the new values.

SELECT

DonationID,

DonationAmount AS existingDonation,

DonationAmount \* 1.10 AS UpdatedDonation

FROM

donations;

/\* **Logical Operations** \*/

SELECT \*

FROM

staff

WHERE

HireDate > '2019-01-01' AND RoleID = 2;

-- Q13: List volunteers who are available and joined before 2022.

SELECT \*

FROM

volunteers

WHERE

Availability = 'Yes'

AND JoinDate < '2022-01-01';

/\* **Inline Functions** \*/

SELECT

UPPER(ProjectName) AS UpperProjectName

FROM

projects;

-- Q15: Concatenate the FirstName and LastName of beneficiaries into a full name.

SELECT

CONCAT(FirstName, ' ', LastName) AS FullName

FROM

beneficiaries;

/\* **IN Clause** \*/

SELECT \*

FROM

projects

WHERE

ProjectID IN (1 , 3, 5);

-- Q17: List staff who belong to RoleID 1 or 2.

SELECT \*

FROM

staff

WHERE

RoleID IN (1 , 2);

/\* LIKE Clause \*/

SELECT \*

FROM

beneficiaries

WHERE

FirstName LIKE 'A%';

-- Q19: Find projects with ProjectName containing the word "Health".

SELECT \*

FROM

projects

WHERE

ProjectName LIKE '%Health%';

/\* **CASE Statement** \*/

SELECT

DonorID,

DonationAmount,

CASE

WHEN DonationAmount > 500 THEN 'More than $1000'

ELSE 'Less than $500'

END AS DonationCategory

FROM

donations;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Categorising Donations by Amount

-- Q21: Categorize projects as 'Active' or 'Completed' based on their EndDate.

SELECT

ProjectID,

ProjectName,

CASE

WHEN

EndDate IS NULL

OR EndDate > CURRENT\_DATE

THEN

'Active'

ELSE 'Completed'

END AS Status

FROM

projects;

/\* **JOINS** \*/

SELECT

S.StaffID,

S.FirstName,

R.RoleName

FROM

staff S

INNER JOIN

staffrole R ON S.RoleID = R.RoleID

ORDER BY S.StaffID;

-- Q23: List all volunteers and their total hours worked using a LEFT JOIN.

SELECT

V.VolunteerID,

V.FirstName,

SUM(VH.HoursWorked) AS TotalHours

FROM

volunteers V

LEFT JOIN

volunteerhours VH ON V.VolunteerID = VH.VolunteerID

GROUP BY V.VolunteerID , V.FirstName;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Volunteers Total Hours Worked

-- Q24: Fetch all projects and their corresponding expenses using a RIGHT JOIN.

SELECT

P.ProjectID,

P.ProjectName,

E.Amount as ExpenseAmount

FROM

projects P

RIGHT JOIN

expenses E ON P.ProjectID = E.ProjectID;

-- Q25: Display all project activities along with their volunteers

SELECT

PA.ActivityID,

PA.ActivityName,

CONCAT(V.FirstName, ' ', V.LastName) AS VolunteerName,

VH.WorkDate

FROM

projectactivities PA

JOIN

volunteerhours VH ON PA.ActivityID = VH.ActivityID

JOIN

volunteers V ON VH.VolunteerID = V.VolunteerID;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Volunteers working on Project Activities

# Advanced SQL Functions - Aggregations using SQL

**/\* Aggregations \*/**

-- Q33: Count the total number of projects.

SELECT COUNT(\*) AS TotalProjects FROM projects;

-- Q34: Find the maximum donation amount.

SELECT MAX(DonationAmount) AS MaxDonation FROM donations;

-- Q35: Calculate the average Budget across all projects.

SELECT AVG(Budget) AS AvgBudget FROM projects;

-- Q36: Group projects by Status and count each group.

SELECT Status, COUNT(\*) AS TotalProjects

FROM projects

GROUP BY Status;

-- Q37: Order beneficiaries by their IncomeLevel.

SELECT \* FROM beneficiaries ORDER BY IncomeLevel ASC;

/\* Filtering with HAVING \*/

-- Q38: Find projects with an average activity budget greater than $5,000.

SELECT

ProjectID,

ROUND(AVG(Budget),2) AS AvgActivityBudget

FROM projects

GROUP BY ProjectID

HAVING AVG(Budget) > 5000;

# Advanced SQL Analytics - Windows functions, subqueries

/\* Window Functions and Subqueries \*/

-- Q39: Assign a row number to each project based on StartDate.

SELECT

ProjectID,

ProjectName,

ROW\_NUMBER() OVER (ORDER BY StartDate) AS RowNum

FROM projects;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Projects Ranking based on Start Date

-- Q40: Rank beneficiaries by their DOB.

SELECT

BeneficiaryID,

CONCAT(FirstName," ", LastName) as Name,

DOB,

RANK() OVER (ORDER BY DOB ASC) AS BeneficiaryRank

FROM beneficiaries;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Ranking Beneficiaries by DOB

-- Q41: Use a subquery to find the project with the highest budget

SELECT \*

FROM

projects

WHERE

Budget =

(SELECT

MAX(Budget)

FROM

projects);

-- Q42: Partition expenses by ProjectID and calculate their sum

SELECT

ProjectID,

ExpenseID,

SUM(Amount) OVER (PARTITION BY ProjectID) AS TotalExpense

FROM expenses;

**Result:**

A screenshot of a data

Description automatically generated

Figure : Total Expenses by Project

/\* DESCRIBE \*/

-- Q43: Use DESCRIBE to show the structure of the beneficiaries table.

DESCRIBE beneficiaries;

/\* Summary Statistics \*/

SELECT

MAX(DonationAmount) - MIN(DonationAmount) AS DonationRange

FROM

donations;

-- Q45: Find the standard deviation of project budgets.

SELECT

STDDEV(Budget) AS StdDevBudget

FROM

projects;

-- Q46: Calculate the variance of hours worked by volunteers.

SELECT

VARIANCE(HoursWorked) AS VarianceHours

FROM

volunteerhours;

/\* Date Functions \*/

SELECT

VolunteerID,

EXTRACT(MONTH FROM JoinDate) AS JoinMonth,

EXTRACT(YEAR FROM JoinDate) AS JoinYear

FROM

volunteers;

SELECT

VolunteerID,

MONTHNAME(JoinDate) AS JoinInMonth,

DAYNAME(JoinDate) AS JoinedON

FROM

volunteers;

-- Q48: Find all projects that started in the year 2022.

SELECT \*

FROM

projects

WHERE

YEAR(StartDate) = 2022;

/\* Multiple Joins \*/

SELECT

S.StaffID,

CONCAT(S.FirstName, ' ', S.LastName) AS Name,

P.ProjectName,

PA.ActivityName

FROM

staff S

JOIN

staffprojects SP ON S.StaffID = SP.StaffID

JOIN

projects P ON SP.ProjectID = P.ProjectID

JOIN

projectactivities PA ON P.ProjectID = PA.ProjectID

ORDER BY S.StaffID;

/\* CTE \*/

-- Q50: Create a CTE to calculate the total budget for each project and fetch projects where the total exceeds $100,000.

WITH TotalBudget AS (

SELECT

ProjectID,

SUM(Budget) AS TotalBudget

FROM projects

GROUP BY ProjectID

)

SELECT \* FROM TotalBudget WHERE TotalBudget > 100000;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Project having more budget than 100000

-- Q51: Find the top 3 donors for each project by donation amount

SELECT \*

FROM

(

SELECT

P.ProjectID,

P.ProjectName,

D.DonorID,

CONCAT(D.FirstName, " ", D.LastName) as DonarName,

DN.DonationAmount,

RANK() OVER (PARTITION BY P.ProjectID ORDER BY DN.DonationAmount DESC) AS DonarRank

FROM

projects P

JOIN

donations DN ON P.ProjectID = DN.ProjectID

JOIN

donors D ON DN.DonorID = D.DonorID

GROUP BY P.ProjectID,P.ProjectName, D.DonorID, DonarName,DN.DonationAmount

) As DonarRankData

WHERE DonarRankData.DonarRank IN (1,2,3);

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Top 3 Donors for each project

-- Q52: Calculate the cumulative total of donations received by each project

SELECT

P.ProjectID,

P.ProjectName,

D.DonationAmount,

SUM(D.DonationAmount) OVER (PARTITION BY P.ProjectID ORDER BY D.DonationDate) AS CumulativeDonations

FROM

projects P

JOIN

donations D ON P.ProjectID = D.ProjectID;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Cumulative Donations for projects

-- Q53: Identify the most active volunteers based on total hours worked

SELECT

V.VolunteerID,

CONCAT(V.FirstName, " ", V.LastName) as VolunteerName,

SUM(VH.HoursWorked) AS TotalHours,

RANK() OVER (ORDER BY SUM(VH.HoursWorked) DESC) AS VolunteerRank

FROM

volunteers V

JOIN

volunteerhours VH ON V.VolunteerID = VH.VolunteerID

GROUP BY

V.VolunteerID, VolunteerName;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Ranking Volunteers based on Hours worked

-- Q54: Find the projects with the highest number of volunteers

SELECT

P.ProjectID,

P.ProjectName,

COUNT(DISTINCT VH.VolunteerID) AS VolunteerCount

FROM

projects P

JOIN

projectactivities PA ON P.ProjectID = PA.ProjectID

JOIN

volunteerhours VH ON PA.ActivityID = VH.ActivityID

GROUP BY P.ProjectID , P.ProjectName

ORDER BY VolunteerCount DESC;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Project with highest Volunteer Hours count

-- Q55: Calculate the average donation amount for each donor and their rank within all donors

SELECT

D.DonorID,

CONCAT(D.FirstName, " ", D.LastName) AS DonarName,

AVG(DN.DonationAmount) AS AvgDonationAmount,

RANK() OVER (ORDER BY AVG(DN.DonationAmount) DESC) AS DonorRank

FROM

donors D

JOIN

donations DN ON D.DonorID = DN.DonorID

GROUP BY

D.DonorID, DonarName;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Ranking Donors by average donations

-- Q56: Determine the average hours worked per volunteer per project

SELECT

P.ProjectID,

P.ProjectName,

V.VolunteerID,

CONCAT(V.FirstName, ' ', V.LastName) AS VolunteerName,

AVG(VH.HoursWorked) AS AvgHours

FROM

projects P

JOIN

projectactivities PA ON P.ProjectID = PA.ProjectID

JOIN

volunteerhours VH ON PA.ActivityID = VH.ActivityID

JOIN

volunteers V ON VH.VolunteerID = V.VolunteerID

GROUP BY P.ProjectID , P.ProjectName , V.VolunteerID , VolunteerName;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Volunteers average working hours

-- Q57: Identify projects with expenses exceeding 50% of their budget

SELECT

P.ProjectID,

P.ProjectName,

P.Budget,

SUM(E.Amount) AS TotalExpenses,

CASE

WHEN SUM(E.Amount) > (P.Budget \* 0.5) THEN 'Exceeds 50%'

ELSE 'Within Budget'

END AS BudgetStatus

FROM

projects P

JOIN

expenses E ON P.ProjectID = E.ProjectID

GROUP BY P.ProjectID , P.ProjectName , P.Budget;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : projects with expenses exceeding 50% of their budget

-- Q58: Find the percentage of total donations each donor contributed

SELECT

D.DonorID,

D.FirstName,

D.LastName,

SUM(DN.DonationAmount) AS TotalDonations,

ROUND(SUM(DN.DonationAmount) \* 100.0 / (SELECT

SUM(DonationAmount)

FROM

donations),

2) AS PercentageOfTotal

FROM

donors D

JOIN

donations DN ON D.DonorID = DN.DonorID

GROUP BY D.DonorID , D.FirstName , D.LastName;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : percentage of total donations each donor contributed

-- Q59: Find the gap between consecutive donation amounts for each donor

SELECT

D.DonorID,

D.FirstName,

D.LastName,

DN.DonationAmount,

LEAD(DN.DonationAmount) OVER (PARTITION BY D.DonorID ORDER BY DN.DonationDate) - DN.DonationAmount AS DonationGap

FROM

donors D

JOIN

donations DN ON D.DonorID = DN.DonorID;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : gap between consecutive donation amounts for each donor

-- Q60: Calculate the time spent by each volunteer in their longest activity

SELECT

V.VolunteerID,

V.FirstName,

V.LastName,

PA.ActivityName,

MAX(VH.HoursWorked) AS MaxHours

FROM

volunteers V

JOIN

volunteerhours VH ON V.VolunteerID = VH.VolunteerID

JOIN

projectactivities PA ON VH.ActivityID = PA.ActivityID

GROUP BY V.VolunteerID , V.FirstName , V.LastName , PA.ActivityName;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : time spent by each volunteer in their longest activity

# CTEs

**/\* NGO Operations CTEs \*/**

-- CTE1 #1: To get summarized donation details by projects - Which 5 projects have Minimum Average Donation

WITH ProjectDonationsSummary AS

(

SELECT

P.ProjectID,

P.ProjectName,

COUNT(D.DonationID) AS TotalDonations,

SUM(D.DonationAmount) AS TotalDonationAmount,

AVG(D.DonationAmount) AS AverageDonation

FROM

projects P

LEFT JOIN

donations D ON P.ProjectID = D.ProjectID

GROUP BY

P.ProjectID, P.ProjectName

),

MinAverageProject AS

(

SELECT \*

FROM

ProjectDonationsSummary

ORDER BY AverageDonation

LIMIT 5

)

SELECT \* FROM MinAverageProject;

**Result:**

**A screenshot of a computer

Description automatically generated**

Figure : 5 projects have Minimum Average Donation

-- CTE #2: To display volunteers and their total hours worked in Descending order

WITH VolunteerHoursSummary AS

(

SELECT

V.VolunteerID,

CONCAT(V.FirstName, " ", V.LastName) as VolunteerName,

SUM(VH.HoursWorked) AS TotalHoursWorked

FROM

volunteers V

LEFT JOIN

volunteerhours VH ON V.VolunteerID = VH.VolunteerID

GROUP BY

V.VolunteerID, VolunteerName

)

SELECT \* FROM VolunteerHoursSummary

ORDER BY TotalHoursWorked DESC;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : volunteers and their total hours worked in Descending order

-- CTE #3: To list beneficiaries and their feedback for projects - Show first 10 based on beneficiaryID

WITH BeneficiaryFeedbackSummary AS

(

SELECT

B.BeneficiaryID,

CONCAT(B.FirstName, " " , B.LastName) as BeneficiaryName,

P.ProjectName,

BF.FeedbackDate,

BF.Feedback

FROM

beneficiaries B

JOIN

beneficiaries\_feedback BF ON B.BeneficiaryID = BF.BeneficiaryID

JOIN

projects P ON BF.ProjectID = P.ProjectID

)

SELECT \* FROM BeneficiaryFeedbackSummary

ORDER BY BeneficiaryID

LIMIT 10;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Top 10 feedback for projects by beneficiary

-- CTE #4: To show staff and their assigned projects which has completed on 2023-06-30

WITH StaffProjectAssignment AS

(

SELECT

S.StaffID,

Concat(S.FirstName, " ", S.LastName) as StaffName,

P.ProjectName,

SP.FromDate,

SP.ToDate

FROM

staff S

JOIN

staffprojects SP ON S.StaffID = SP.StaffID

JOIN

projects P ON SP.ProjectID = P.ProjectID

)

SELECT \* FROM StaffProjectAssignment

WHERE ToDate = '2023-06-30';

**Result:**

A screenshot of a computer

Description automatically generated

Figure : staff and their assigned projects which has completed on 2023-06-30

# Views

**/\* NGO Operations Views \*/**

**-- View #1: To get summarized donation details by projects**

CREATE VIEW ProjectDonationsSummary AS

SELECT

P.ProjectID,

P.ProjectName,

COUNT(D.DonationID) AS TotalDonations,

SUM(D.DonationAmount) AS TotalDonationAmount,

AVG(D.DonationAmount) AS AverageDonation

FROM

projects P

LEFT JOIN

donations D ON P.ProjectID = D.ProjectID

GROUP BY

P.ProjectID, P.ProjectName;

**-- Extract the Project Donations Summary from the View**

SELECT \* FROM ProjectDonationsSummary;

**Result:**

A screenshot of a computer screen

Description automatically generated

Figure : summarized donation details by projects

**-- View #2: To display volunteers and their total hours worked**

CREATE VIEW VolunteerHoursSummary AS

SELECT

V.VolunteerID,

CONCAT(V.FirstName, " ", V.LastName) as VolunteerName,

SUM(VH.HoursWorked) AS TotalHoursWorked

FROM

volunteers V

LEFT JOIN

volunteerhours VH ON V.VolunteerID = VH.VolunteerID

GROUP BY

V.VolunteerID, VolunteerName;

**-- Extract the Volunteer Hours Summary from the View**

SELECT \* FROM VolunteerHoursSummary;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : volunteers and their total hours worked

**-- View #3: To list beneficiaries and their feedback for projects**

CREATE VIEW BeneficiaryFeedbackSummary AS

SELECT

B.BeneficiaryID,

CONCAT(B.FirstName, " " , B.LastName) as BeneficiaryName,

P.ProjectName,

BF.FeedbackDate,

BF.Feedback

FROM

beneficiaries B

JOIN

beneficiaries\_feedback BF ON B.BeneficiaryID = BF.BeneficiaryID

JOIN

projects P ON BF.ProjectID = P.ProjectID;

**-- Extract the beneficiaries and their feedback for projects**

SELECT \* FROM BeneficiaryFeedbackSummary;

**Result:**

A screenshot of a computer

Description automatically generated

Figure : beneficiaries and their feedback for projects

**-- View #4: To show staff and their assigned projects**

CREATE VIEW StaffProjectAssignment AS

SELECT

S.StaffID,

Concat(S.FirstName, " ", S.LastName) as StaffName,

P.ProjectName,

SP.FromDate,

SP.ToDate

FROM

staff S

JOIN

staffprojects SP ON S.StaffID = SP.StaffID

JOIN

projects P ON SP.ProjectID = P.ProjectID;

**-- Extract staff and their assigned projects from View**

SELECT \* FROM StaffProjectAssignment

**Result:**

A screenshot of a computer

Description automatically generated

Figure : staff and their assigned projects

# Stored Procedures

**/\* NGO Operations Stored Procedures \*/**

**-- Stored Procedure #1: To Add a New Donation**

DELIMITER //

CREATE PROCEDURE AddDonation(

IN donorID INT,

IN projectID INT,

IN donationAmount DECIMAL(10,2),

IN donationDate DATE

)

BEGIN

INSERT INTO donations (DonorID, ProjectID, DonationAmount, DonationDate)

VALUES (donorID, projectID, donationAmount, donationDate);

END //

DELIMITER ;

**-- Call Stored Procedure #1 to add a New Donation**

CALL AddDonation(1,2,1500,'2024-12-01')

**Result:**

**A screenshot of a data table

Description automatically generated**

Figure : Record added using stored procedure

**-- Stored Procedure #2: To Update a Project Budget**

DELIMITER //

CREATE PROCEDURE UpdateProjectBudget(

IN projectID INT,

IN newBudget DECIMAL(15,2)

)

BEGIN

UPDATE projects P

SET Budget = newBudget

WHERE P.ProjectID = projectID;

END //

DELIMITER ;

DROP PROCEDURE IF EXISTS UpdateProjectBudget;

-- Call SP to update the budget

CALL UpdateProjectBudget(4,200000)

**Result:**

A screenshot of a computer

Description automatically generated

Figure : Record updated with budget by stored procedure

**-- Stored Procedure #3: To Get Project Details with Donor Information**

DELIMITER //

CREATE PROCEDURE GetProjectDonorDetails(IN projectID INT)

BEGIN

SELECT

P.ProjectName,

D.FirstName AS DonorFirstName,

D.LastName AS DonorLastName,

DN.DonationAmount,

DN.DonationDate

FROM

projects P

JOIN

donations DN ON P.ProjectID = DN.ProjectID

JOIN

donors D ON DN.DonorID = D.DonorID

WHERE

P.ProjectID = projectID;

END //

DELIMITER ;

**-- Call SP to retrive project donars for 1**

CALL GetProjectDonorDetails(1);

**Result:**

A screenshot of a computer

Description automatically generated

Figure : List of project donars

**-- Stored Procedure #4: To Delete a Beneficiary Feedback Entry**

DELIMITER //

CREATE PROCEDURE **DeleteBeneficiaryFeedback**(

**IN beneficiaryid INT,**

**IN projectid INT**

)

BEGIN

DELETE FROM beneficiaries\_feedback BFB

WHERE BFB.BeneficiaryID = beneficiaryid AND BFB.ProjectID = projectid;

END //

DELIMITER ;

**-- Call SP to delete a Beneficiary Feedback Entry**

CALL DeleteBeneficiaryFeedback(598,5);

**Results:**

**Before:**

**A screenshot of a computer

Description automatically generated**

Figure : Beneficiary Feedback entries before record deletion

**After:**

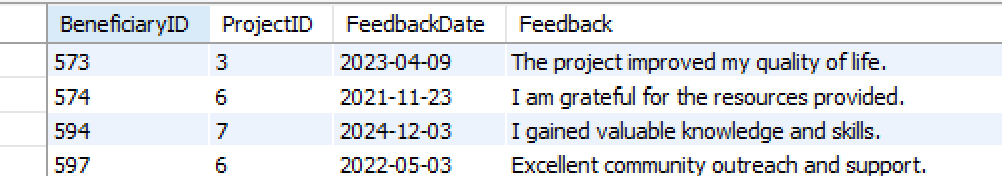
****

Figure : Beneficiary Feedback entries after record deletion

**-- Stored Procedure #5: To Generate Volunteer Summary**

DELIMITER //

CREATE PROCEDURE GetVolunteerSummary()

BEGIN

SELECT

V.VolunteerID,

V.FirstName,

V.LastName,

COUNT(VH.WorkDate) AS TotalDaysWorked,

SUM(VH.HoursWorked) AS TotalHoursWorked

FROM

volunteers V

LEFT JOIN

volunteerhours VH ON V.VolunteerID = VH.VolunteerID

GROUP BY

V.VolunteerID, V.FirstName, V.LastName;

END //

DELIMITER ;

**-- Call SP to retrieve Volunteer Summary**

CALL GetVolunteerSummary;

**Result:**

**A screenshot of a table

Description automatically generated**

Figure : Volunteer Worked Hours Summary

# Technology Stack

## Database Management System (DBMS)

MySQL Workbench - used as the primary database design and management tool. It facilitated the creation of the Entity-Relationship (ER) diagram, schema design, and SQL queries for managing data operations.

## SQL (Structured Query Language)

SQL was used for defining, manipulating, and querying data within the MySQL database. Queries were written for tasks such as creating tables, managing relationships, and retrieving specific information.

## MySQL Server

A MySQL database server was configured to host the database and allow access for authorized users. The server ensures data integrity and provides support for multiple concurrent users.

## Draw.io

This tool was used for ER Modeling.

# Conclusion

The NGO Operations Database project successfully implemented tailored to the unique needs of nonprofit organizations. By centralizing key operational data, the system enhances the efficiency, transparency, and accountability of the NGO's day-to-day activities and long-term projects.

# References

* Dietrich, S. W. (2021). *Understanding databases: Concepts and practice*, first edition. Wiley.
* Foote, K. D. (2023, October 3). *Types of database management systems*. DATAVERSITY. https://www.dataversity.net/types-of-database-management-systems/
* *Microsoft*. Microsoft Support. (n.d.). https://support.microsoft.com/en-us/office/create-a-diagram-with-crow-s-foot-database-notation-1ec22af9-3bd3-4354-b2b5-ed5752af6769
* P., T. R. M. (2021). *SQL for data scientists: A beginner’s guide for building datasets for analysis.* John Wiley & Sons, Incorporated.