NGO Management System

A PROJECT REPORT

Submitted by

Ashish Anil Singh (RA2211033010155)

Seshadri Patra (RA2211033010182)

Under the Guidance of

Dr. R. Beaulah Jeyavathana

Assistant Professor, Department of Computational Intelligence

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

RA2211033010155 and RA2211033010182 Certified to be the bonafide work done by **Ashish Anil Singh and Seshadri Patra** of II year/IV sem B.Tech Degree Course in the Project Course – **21CSC205P Database Management Systems** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**, Kattankulathur for the academic year 2023-2024.

Date: 3Rd MAY 2024

Faculty in Charge
Dr. R. Beaulah Jeyavathana
Assistant Professor
Department of Computational Intelligence
SRMSIT -KTR

HEAD OF THE DEPARTMENT
Dr. R. Annie Uthra
Professor and Head
Department of Computational Intelligence
SRMIST - KTR

ABSTRACT

The NGO Management System project is a comprehensive initiative aimed at developing a robust database management solution tailored specifically to the intricate operational needs of non-governmental organizations (NGOs). At its core, the project seeks to harness the principles of Database Management Systems (DBMS) to streamline and optimize various administrative tasks essential for the effective functioning of NGOs. With a focus on enhancing efficiency, transparency, and accountability, the system will address key areas such as donor management, volunteer coordination, project tracking, financial management, and reporting.

Central to the NGO Management System is its capability to effectively manage donor relations. By employing a relational database model, the system will enable NGOs to meticulously record and track donor information, including donation history, preferences, and communication interactions. This functionality will empower NGOs to cultivate stronger relationships with their donors, tailor communication strategies to individual preferences, and ultimately optimize fundraising efforts. Moreover, the system will facilitate targeted outreach campaigns, analyze donation patterns, and generate comprehensive reports to inform strategic decision-making.

Volunteer coordination is another critical aspect of NGO operations that the system aims to streamline. Through a centralized database of volunteer profiles, including skills, availability, and contributions, NGOs will be able to efficiently manage their volunteer base. The system will automate volunteer assignment and scheduling processes, matching volunteers with tasks that align with their skills and availability. This automation will not only save time and resources but also ensure that volunteer efforts are maximized and effectively utilized to support the organization's mission and projects.

TABLE OF CONTENTS

Abstract

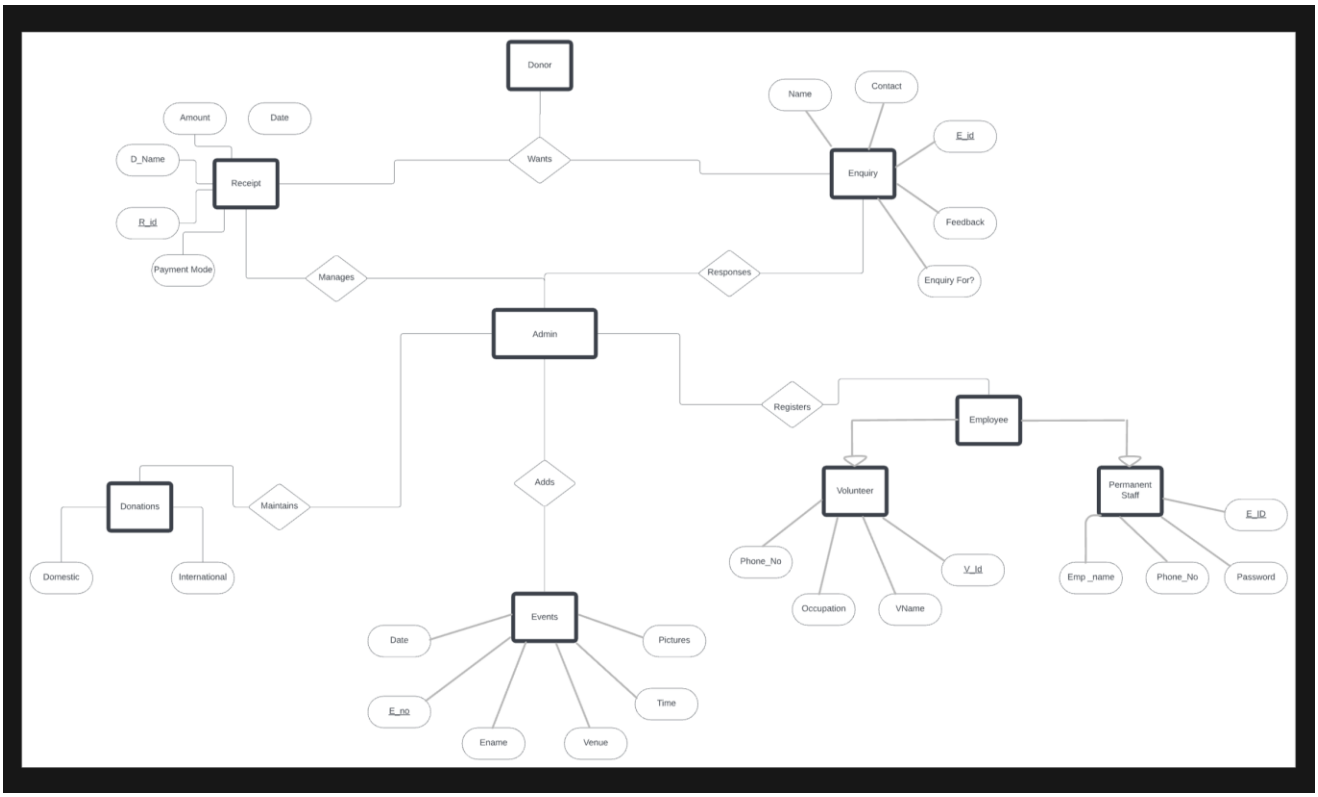
Problem Statement

1.Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the project

PROJECT DESCRIPTION

A Non-Governmental Organization (NGO) management system aims to streamline the operations of an organization dedicated to social welfare, humanitarian aid, or environmental conservation. The system will encompass various functionalities crucial for efficient management, including but not limited to donor management, volunteer tracking, project management, and resource allocation. It should facilitate the organization in maintaining comprehensive records of donors, managing their contributions, and fostering donor relationships. Additionally, the system should enable the efficient management of volunteers, including recruitment, scheduling, and task assignment. Project management features should allow the organization to plan, execute, and monitor projects effectively, tracking progress and allocating resources appropriately. Moreover, the system should support the management of resources such as funds, equipment, and supplies, ensuring their optimal utilization. Overall, the NGO management system aims to enhance organizational efficiency, transparency, and impact in fulfilling its mission of serving communities and addressing social or environmental challenges.

ER Diagram for NGO Management System



The Entity-Relationship (ER) diagram for the NGO management system depicts the relationships between various entities involved in the organization's operations. Entities such as Members, Donors, Volunteers, Tasks, and Transactions are interconnected through relationships like Membership, Donation, Volunteering, Task Assignment, and Financial Transaction. Each entity possesses attributes that describe its properties, while the relationships define how these entities are related to each other.

2. DESIGN OF RELATIONAL SCHEMAS, CREATION OF DATABASE TABLES FOR THE PROJECT.

admin

admin_id	name	email	city_id	phone
1	John Doe	johndoe@example.com	1	1234567890
2	Jane Smith	janesmith@example.com	2	9876543210
3	Alex Lee	alexlee@example.com	3	4567890123

Schema: `admin` (`admin_id`, `name`, `email`, `city_id`, `phone`);

admin_login

login_id	username	password	admin_id
1	admin1	password1	1
2	admin2	password2	2
3	admin3	password3	3

Schema: `admin_login` (`login_id`, `username`, `password`, `admin_id`);

city

city_id	cname
1	New York
2	London
3	Sydney

Schema: `city` (`city_id`, `cname`);

donor

donor_id	email	address	city_id	phone	name
1	donor1@example.com	123 Main St, NY	1	1112223333	Alice
2	donor2@example.com	456 Park Ave, London	2	4445556666	Bob
3	donor3@example.com	789 Beach Rd, Sydney	3	7778889999	Charlie

Schema: `donor` (`donor_id`, `email`, `address`, `city_id`, `phone`, `name`);

donor_login

username	login_id	password	donor_id
donor1	1	pass1	1
donor2	2	pass2	2
donor3	3	pass3	3

Schema: `donor_login` (`username`, `login_id`, `password`, `donor_id`);

items

Item_id	item	donor_id
1	Clothing	1
2	Food	2
3	Books	3

Schema: `items` (`Item_id`, `item`, `donor_id`);

ngo_account

id	bank	ifsc_code	acount	donor_id	city_id	donationS
1	ABC Bank	ABC123	Savings	1	1	1000
2	XYZ Bank	XYZ456	Current	2	2	2500
3	PQR Bank	PQR789	Savings	3	3	500

Schema: `ngo_account` (`id`, `bank`, `ifsc_code`, `acount`, `donor_id`, `city_id`, `donationS`);

task

task_id	task	volunteer_id
1	Event Planning	1
2	Fundraising	2
3	Community Outreach	3

Schema: `task` (`task_id`, `task`, `volunteer_id`);

transaction

id	donor_id	tdate	amount
1	1	2023-05-01 10:00:00	500
2	2	2023-05-02 14:30:00	1000
3	3	2023-05-03 12:15:00	200

Schema: `transaction` (`id`, `donor_id`, `tdate`, `amount`);

volunteer

volunteer_id	name	email	interests	dob	city_id	phone
1	Emily Wong	emilywong@example.com	Environment	1990-03-15	1	1112223333
2	David Kim	davidkim@example.com	Education	1985-07-20	2	4445556666
3	Sarah Lin	sarahlin@example.com	Health	1992-11-05	3	7778889999

CREATE TABLE `volunteer` (`volunteer_id`, `name`, `email`, `intrests`, `dob`, `city_id`, `phone`);

volunteer_login

login_id	username	password	volunteer_id
1	volunteer1	pass1	1
2	volunteer2	pass2	2
3	volunteer3	pass3	3

CREATE TABLE `volunteer_login` (`login_id`, `username`, `password`, `volunteer_id`);

3. COMPLEX QUERIES BASED ON THE CONCEPTS OF CONSTRAINTS, SETS, JOINS, VIEWS, TRIGGERS AND CURSORS

- **CONSTARAINTS**

Primary Key Constraint

```
CREATE TABLE Members (
   member_id INT PRIMARY KEY,
   fname VARCHAR(40) NOT NULL,
   lname VARCHAR(40) NOT NULL,
   email VARCHAR(40) UNIQUE,
   contact VARCHAR(20) NOT NULL
);


-- Foreign Key Constraint
ALTER TABLE doctorapp
ADD CONSTRAINT FK_Trainer
FOREIGN KEY (docapp) REFERENCES Trainer(Trainer_id);


-- Unique Constraint
ALTER TABLE doctorapp
ADD CONSTRAINT UC_Email
UNIQUE (email);
```

-- Not Null Constraint

ALTER TABLE Trainer

MODIFY COLUMN Name VARCHAR(40) NOT NULL;

- **Set Operations**

-- Union

SELECT contact FROM doctorapp

UNION

SELECT customer_id FROM Payment;

-- Intersect

SELECT contact FROM doctorapp

INTERSECT

SELECT customer_id FROM Payment;

-- Except

SELECT contact FROM doctorapp

EXCEPT

SELECT customer_id FROM Payment;

- **Join Queries**

```
SELECT
    da.fname AS Member_First_Name,
    da.lname AS Member_Last_Name,
    pkg.Package_name AS Package_Name,
    pay.Amount AS Payment_Amount,
    pay.payment_type AS Payment_Type
FROM
    doctorapp da
JOIN
    Payment pay
ON
    da.contact = pay.customer_id
JOIN
    Package pkg
ON
    pkg.Package_id = da.docapp
WHERE
    pay.Amount = pkg.Amount;
```

- **View for member Payments**

```
CREATE VIEW MemberPayments AS
SELECT
    da.fname AS Member_First_Name,
    da.lname AS Member_Last_Name,
    pkg.Package_name AS Package_Name,
    pay.Amount AS Payment_Amount,
    pay.payment_type AS Payment_Type
FROM
    doctorapp da
JOIN
    Payment pay
ON
    da.contact = pay.customer_id
JOIN
    Package pkg
ON
    pkg.Package_id = da.docapp;
```

- **Trigger for Automatic Log Update**

```
DELIMITER //
```

```
CREATE TRIGGER UpdateLoginOnNewMember
AFTER INSERT ON doctorapp
FOR EACH ROW
BEGIN
    INSERT INTO logintb (username, password)
    VALUES (NEW.email, 'default_password');
END;
//
DELIMITER ;
```

- **Cursor for Bulk Operations**

```
DELIMITER //
CREATE PROCEDURE UpdateMemberContact(IN trainer_id INT, IN
new_contact VARCHAR(40))
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE contact VARCHAR(40);

    DECLARE member_cursor CURSOR FOR
    SELECT contact FROM doctorapp WHERE docapp = trainer_id;
```

```
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;

    OPEN member_cursor;

    member_loop: LOOP
        FETCH member_cursor INTO contact;

        IF done THEN
            LEAVE member_loop;
        END IF;

        -- Update member's contact information
        UPDATE doctorapp
        SET contact = new_contact
        WHERE contact = contact;
    END LOOP;

    CLOSE member_cursor;
END;
```

//

DELIMITER;

4.ANALYZING THE PITFALLS, IDENTIFYING THE DEPENDENCIES, AND APPLYING NORMALIZATIONS

1. Identify Pitfalls:

- Lack of normalization: The schema appears to have some redundancy, which can lead to data inconsistency and update anomalies.

- Redundant or unused columns: There might be columns in the tables that are not necessary or could be derived from other columns, leading to unnecessary storage space and complexity.

- Lack of referential integrity: The foreign key constraints might not cover all necessary relationships between tables, leading to orphaned records and data integrity issues.

- Lack of constraints: Some columns might lack constraints such as NOT NULL or UNIQUE, allowing for invalid or duplicate data.

2. Identify Dependencies:

- Identify functional dependencies within each table: Determine which attributes depend on others within the same table.

- Identify relationships between tables: Determine the relationships between tables

using foreign key constraints.

3. Apply Normalizations:

First Normal Form (1NF): Ensure that each table has a primary key, and each column contains atomic values. If there are composite attributes, split them into individual attributes.

Second Normal Form (2NF): Eliminate partial dependencies by moving non-key attributes to separate tables.

Third Normal Form (3NF): Eliminate transitive dependencies by moving attributes that depend on non-key attributes to separate tables.

Here are some potential normalization steps based on the provided schema:

1NF:

- Ensure that each table has a primary key (done).
- Verify that each column contains atomic values (no multi-valued attributes).

2NF:

- For example, consider splitting the `items` table to eliminate partial dependencies. If `item` depends on `donor_id`, move it to a separate table with `donor_id` as the

primary key.

3NF:

- For example, consider splitting the `ngo_account` table to eliminate transitive dependencies. If `bank`, `ifsc_code`, and `account` depend on `id`, move them to a separate table with `id` as the primary key.

5. Implementation of concurrency control and recovery mechanisms

1. Concurrency Control Mechanisms:

- Locking: Implement a locking mechanism to control access to data items. Use locks such as shared locks and exclusive locks to prevent conflicting operations from occurring simultaneously.
- Transaction Isolation Levels: Implement different transaction isolation levels (e.g., Read Uncommitted, Read Committed, Repeatable Read,

Serializable) to control the visibility of data changes made by concurrent transactions.

- Timestamp Ordering: Use timestamp-based concurrency control mechanisms to order transactions based on their timestamps and ensure serializability of transactions.
- Optimistic Concurrency Control: Implement optimistic concurrency control techniques such as validation or versioning to detect conflicts at commit time rather than during transaction execution.
- Two-Phase Locking (2PL): Implement the two-phase locking protocol to ensure serializability by acquiring all locks before performing any modifications and releasing them only after the transaction completes.

2. Recovery Mechanisms:

- Write-Ahead Logging (WAL): Implement WAL protocol to ensure durability by writing transaction updates to the log before modifying the corresponding data in the database. During recovery, redo and undo operations are performed based on the log records to bring the database to a consistent state.
- Checkpointing: Implement periodic checkpointing to reduce recovery time by writing consistent snapshots of the database to stable storage. During

recovery, only the transactions after the last checkpoint need to be replayed from the log.

- Transaction Rollback: Implement transaction rollback mechanisms to undo the effects of incomplete transactions in case of failures. Use the log records to identify and undo the changes made by aborted transactions.
- Recovery Manager: Implement a recovery manager responsible for coordinating recovery operations such as redo, undo, and checkpointing. The recovery manager ensures that the database remains consistent after system failures.

3. Implementation Considerations:

- Concurrency Control Algorithms: Choose appropriate concurrency control algorithms based on the concurrency requirements and workload characteristics of the application.
- Logging and Recovery Strategies: Implement efficient logging and recovery strategies to minimize the overhead of logging and recovery operations.
- Transaction Management: Implement transaction management mechanisms to ensure that transactions are executed atomically, consistently, and durably.
- Testing and Validation: Thoroughly test and validate the concurrency control and recovery mechanisms to ensure their correctness and reliability

under various failure scenarios.

5. CODE FOR THIS PROJECT

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark shadow-lg p-3 mb-5 ">
  <a class="navbar-brand" href="index.php">NGO</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="index.php" aria-
controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNav">
    <ul class="navbar-nav">
      <li class="nav-item active">
        <a class="nav-link" href="#">Admin<span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item ">
```

```
        <a class="nav-link" href="logout.php">
        <?php
          $stmt3 = $pdo->query("SELECT `name` FROM `admin` WHERE `admin_id`
="._SESSION['admin_id']);
          $rows2 = $stmt3->fetchAll(PDO::FETCH_ASSOC);
          echo $rows2[0]['name'];
        ?>
        <span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item ">
        <a class="nav-link" href="update/adminUpdate.php">Edit Profile<span class="sr-
only">(current)</span></a>
      </li>
      <li class="nav-item ">
        <a class="nav-link" href="adminVolunteer.php">Volunteers<span class="sr-
only">(current)</span></a>
      </li>
      <li class="nav-item ">
        <a class="nav-link" href="logout.php">Logout<span class="sr-only">(current)</span></a>
      </li>
    </ul>
  </div>
</nav>

<div class="container  rounded">
  <?php
```

```
    if(isset($_SESSION['success'])){
      echo '<div class="row alert alert-success" role="alert">';
      echo $_SESSION['success'];
      unset ($_SESSION['success']);
      echo '</div>' ;
    }

    if(isset($_SESSION['error'])){
      echo '<div class="row alert alert-danger" role="alert">';
      echo $_SESSION['error'];
      unset($_SESSION['error']);
      echo '</div>';
    }
  ?>
  <div>
    <?php
     $stmt3 = $pdo->query("SELECT SUM(donationS)FROM ngo_account");
     $rows = $stmt3->fetchAll(PDO::FETCH_ASSOC);
     echo "<h2 class='shadow-lg p-3 mb-5 bg-light rounded mx-auto' style='width: 550px;'>The Overall
Donations Are ₹ ".$rows[0]['SUM(donationS)']." </h2>" ;
    ?>
  </div>
  <div class = "row ">
    <div class="col-6 "><h3>Bengaluru Donors</h3></div>
    <div class="col-6"><h3>Hyderabad Donors</h3></div>
  </div>
```

```
  <div class = "row ">
    <div class="col-6">
      <table class="table shadow-lg p-3 mb-5 bg-light rounded ">
        <thead class="thead-dark">
          <tr class="">
            <th class="" scope="col">Sno </th>
            <th class="" scope="col">Donor Name</th>
            <th class="" scope="col">Donation</th>
            <th class="" scope = "col">     </th>
          </tr>
        </thead>
        <tbody class="">
          <?php
            $stmt3 = $pdo->query("SELECT ngo_account.donor_id , donor.name , ngo_account.donationS
FROM donor JOIN ngo_account WHERE donor.donor_id = ngo_account.donor_id AND
ngo_account.city_id = 1");
            $rows = $stmt3->fetchAll(PDO::FETCH_ASSOC);
            $count = 1;
             foreach($rows as $row){
               echo "<tr class=">";
               echo "<th scope='row' class=">".$count."</th>";
               echo "<td class=">".htmlentities($row['name'])."</td>";
               echo "<td class=" >".htmlentities($row['donationS'])." </td>";
               echo("<td class=" > <a class='btn btn-primary btn-sm'
href='admin/delete.php?donor_id=".$row['donor_id']."'>Remove Donor</a></td>");
               echo "</tr>";
```

```
            $count++;
            }
          ?>
        </tbody>
      </table>
    </div>
    <div class="col-6">
      <table class="table shadow-lg p-3 mb-5 bg-light rounded ">
        <thead class="thead-dark">
          <tr class="">
            <th class="" scope="col">Sno </th>
            <th class="" scope="col">Donor Name</th>
            <th class="" scope="col">Donation</th>
            <th class="" scope = "col">     </th>
          </tr>
        </thead>
        <tbody class="">
          <?php
            $stmt3 = $pdo->query("SELECT ngo_account.donor_id , donor.name , ngo_account.donationS
FROM donor JOIN ngo_account WHERE donor.donor_id = ngo_account.donor_id AND
ngo_account.city_id = 2");
            $rows = $stmt3->fetchAll(PDO::FETCH_ASSOC);
            $count = 1;
            foreach($rows as $row){
              echo "<tr class=">";
              echo "<th scope='row' class=">".$count."</th>";
```

```
            echo "<td class=">".htmlentities($row['name'])."</td>";
            echo "<td class=" >".htmlentities($row['donationS'])." </td>";
            echo("<td class=" > <a class='btn btn-primary btn-sm'
href='admin/delete.php?donor_id=".$row['donor_id']."'>Remove Donor</a></td>");
            echo "</tr>";
           $count++;
           }
          ?>
       </tbody>
     </table>
  </div>

  </div>
  <div class = "row">
   <div class="col-6"><h3>Chennai Donors</h3></div>
   <div class="col-6"><h3>Mumbai Donors</h3></div>

 </div>
  <div class = "row">
   <div class="col-6">
     <table class="table shadow-lg p-3 mb-5 bg-light rounded ">
       <thead class="thead-dark">
         <tr class="">
           <th class="" scope="col">Sno </th>
           <th class="" scope="col">Donor Name</th>
           <th class="" scope="col">Donation</th>
```

```
          <th class="" scope = "col">     </th>
        </tr>
      </thead>
      <tbody class="">
        <?php
          $stmt3 = $pdo->query("SELECT ngo_account.donor_id , donor.name , ngo_account.donationS
FROM donor JOIN ngo_account WHERE donor.donor_id = ngo_account.donor_id AND
ngo_account.city_id = 3");
          $rows = $stmt3->fetchAll(PDO::FETCH_ASSOC);
          $count = 1;
           foreach($rows as $row){
             echo "<tr class=">";
             echo "<th scope='row' class=">".$count."</th>";
             echo "<td class=">".htmlentities($row['name'])."</td>";
             echo "<td class=" >".htmlentities($row['donationS'])." </td>";
             echo("<td class=" > <a class='btn btn-primary btn-sm'
href='admin/delete.php?donor_id=".$row['donor_id']."'>Remove Donor</a></td>");
             echo "</tr>";
            $count++;
            }
          ?>
      </tbody>
    </table>
  </div>
  <div class="col-6">
    <table class="table shadow-lg p-3 mb-5 bg-light rounded ">
```

```
<thead class="thead-dark">
  <tr class="">
    <th class="" scope="col">Sno </th>
    <th class="" scope="col">Donor Name</th>
    <th class="" scope="col">Donation</th>
    <th class="" scope = "col">     </th>
  </tr>
</thead>
<tbody class="">
  <?php
    $stmt3 = $pdo->query("SELECT ngo_account.donor_id , donor.name , ngo_account.donationS
FROM donor JOIN ngo_account WHERE donor.donor_id = ngo_account.donor_id AND
ngo_account.city_id = 4");
    $rows = $stmt3->fetchAll(PDO::FETCH_ASSOC);
    $count = 1;
    foreach($rows as $row){
      echo "<tr class=">";
      echo "<th scope='row' class=">".$count."</th>";
      echo "<td class=">".htmlentities($row['name'])."</td>";
      echo "<td class=" >".htmlentities($row['donationS'])." </td>";
      echo("<td class=" > <a class='btn btn-primary btn-sm'
href='admin/delete.php?donor_id=".$row['donor_id']."'>Remove Donor</a></td>");
      echo "</tr>";
    $count++;
    }
  ?>
```

```
        </tbody>
      </table>
    </div>
```

6. Result and Discussion

7. Conclusion

In conclusion, the project involves the design and implementation of a database schema for managing a non-governmental organization (NGO) and its various operations, including managing members, donors, volunteers, tasks, donations, and financial transactions.

The provided SQL schema outlines the structure of the database, including tables such as `Members`, `donor`, `volunteer`, `task`, `transaction`, and others. Each table contains attributes related to specific entities within the organization, along with constraints, set operations, join queries, views, triggers, and cursors to facilitate data management and retrieval.

However, the schema exhibits certain pitfalls such as redundancy, lack of normalization, and potential data integrity issues. To address these shortcomings, it is essential to identify dependencies, apply normalization techniques, enforce referential integrity constraints, and optimize data integrity constraints. By doing so, the database design can be improved to ensure efficiency, reliability, and maintainability in managing the NGO's operations.

In summary, while the provided schema serves as a foundation for the database, further refinement through normalization, constraint enforcement, and optimization

is necessary to enhance its effectiveness and robustness in supporting the NGO's activities.

8. REFRENCE

Coursera: Database Design and Relational Theory

Udemy: The Complete SQL Bootcamp

TutorialsPoint

W3Schools

GeeksforGeeks

Stack Overflow

Oracle Documentation

SQLCourse

Database Journal