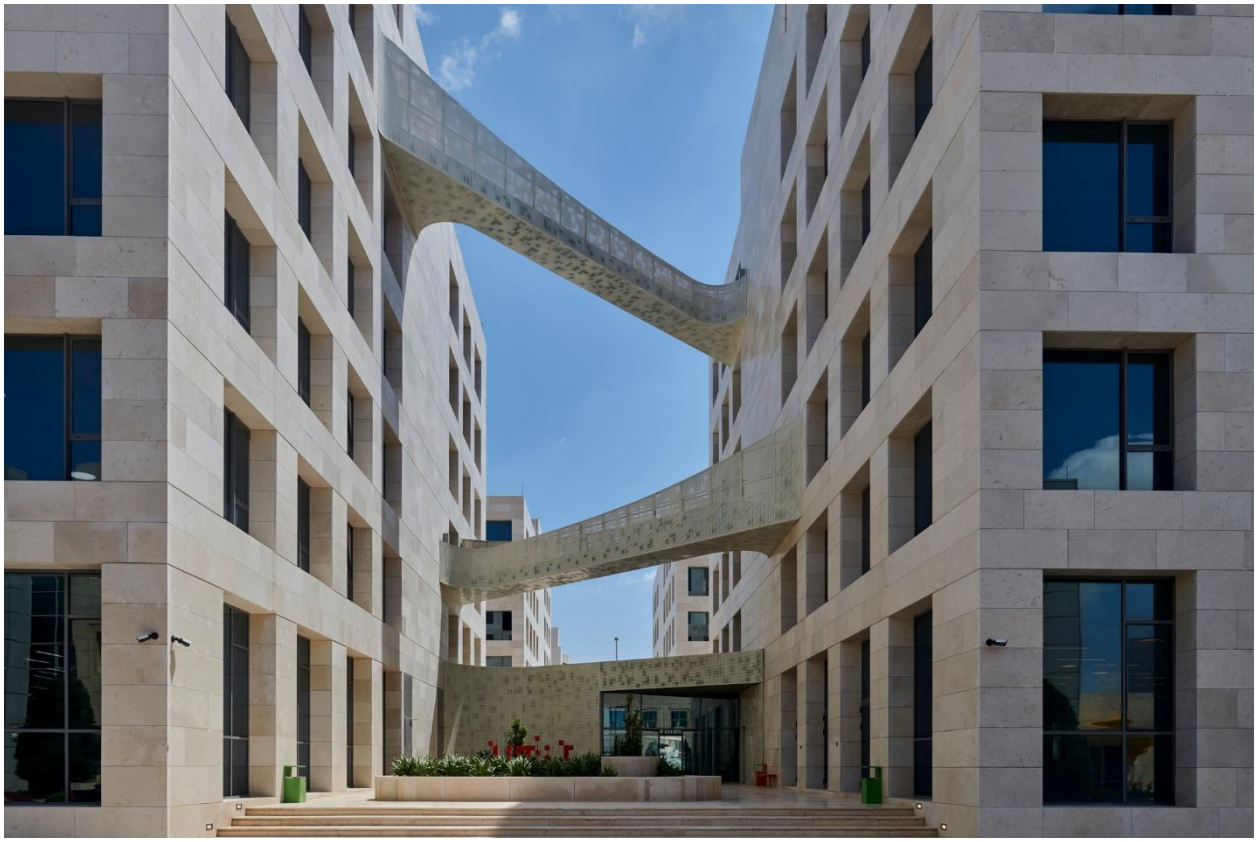
جامعة الحسين التقنية
Al Hussein Technical University



Student Name	Osama Mohammed Ziad Hasan
Student ID	23110709
HTU Course Number and Title	10204282 Database Design & Development
Academic Year	2024-2025 Fall
Assignment Author	Salem Alemaishat
Course Tutor	Ashraf Al-Samadi
Assignment Title	Donor Tracking System Database Design and Development
Submission Date	29/1/2025

Table of Contents

Technical Documentation

1 Introduction

Building a comprehensive database is not an easy task, a small wrong step can lead to a completely wrong and different result than what is required. I am aiming to build a database to donate for the EDU Youth Foundation through non-profit charitable events organized by the Foundation, through which donations are made by donors, to reach the final goal, of developing education for every child and youth financially at the local and regional levels.

To reach the database that I aim, there are clear steps that must be followed one by one, it is not easy but by setting a clear approach, then a clear goal and mastering the work and dealing with it we can reach that.

2 Database Requirement

2.1 User and System Requirement

• First, we must define the basic user requirement for our database in detail to avoid any complications or forgetting any of them when starting.

User Requirement:

- Event organizer: He is the boss, who sets the rules and foundations for the event he is appointed to, and determines the material and financial requirements for the event. He also chooses from the volunteers who apply for acceptance, and determines the total expected amount for donations.
- Donor: Which is the main link and the reason for this event. He donates the amount he wants to help the organization.
- Volunteer: They are who volunteered to manage this event. The organizer divide them and each of them takes his role during the event.

• After defining and enumerating the user requirements, it is time for the system requirements, which are the systems we need to lay the foundation stone for this project.

System Requirement:

- Donations Managing and Tracking: Donors are allowed to participate in any much charity events as they want and track each donation.

- Events management: Scheduling events, making plans, determining the goals for each event and estimated number of volunteers and donors desired. I also need to create statistics to see last events and evaluate every event success.
- Handling Donor and Volunteer Information: Donors and volunteers details must be stored, to ensure that multiple donors and volunteers can participate in different events.

2.2 Data Requirements

• Data Requirements are the data that is desired for the database.

- The organization holds **events**. Each event has an id, name, date, number of donations at it, location, organizer.
- **Volunteers** are assigned to help organize the event while it is happening, which can volunteer at several events, each volunteer has an id, name, age, contact information, the event he is participating in, his role at the event.
- **Donors** comes to the event to donate, they also can donate at more than one event organized by the organization, Each donor has an id, name, age, date of birth, contact information, career details.
- Donors make **donations**, which occur during the event. Each donation has an id, amount, payment method, in which event, from who

3 Database Design

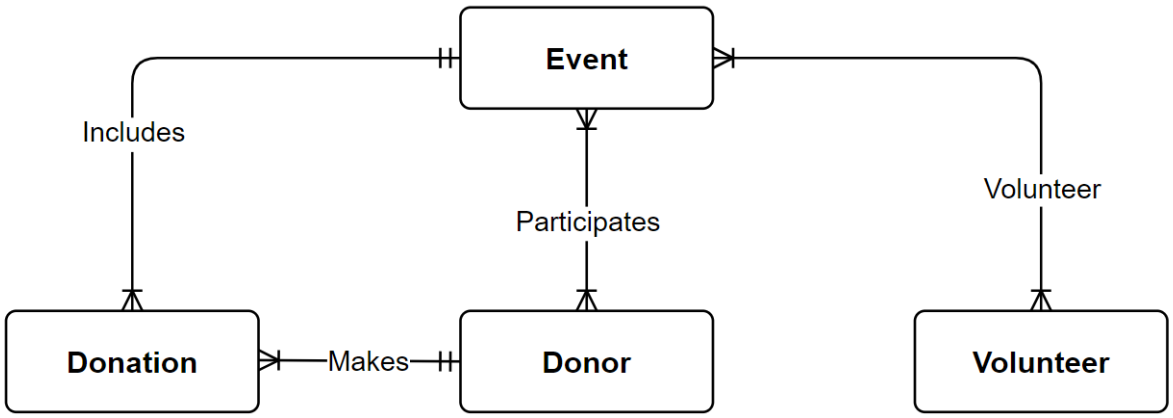
3.1 Conceptual Design

The conceptual design is a description of the basic entities and the relationships that connect them to each other before starting any Advanced stage. Therefore, it forms the cornerstone of the entire database.

So, we conclude that components of logical design are:

• Entities, Entity types
• Relationships
• Constraints

My Conceptual Design:



My entities are (Event, Donor Donation Volunteer), here are some details of these entities:

Event: The place and time through which the donation is made.
Donor: The person who goes to the event for a financial purpose.
Donation: The donation in which the donor makes (The donor can make more than one donation).
Volunteer: The volunteer in the charitable event, he does things like guiding the donors during the event, or other different roles.

3.2 Schema and Mapping

Basic step, schema:

Event (Event_ID, Event_Name, Date, Location, Number_of_Donations, Organizer_Name)
Donor (Donor_ID, Event_ID, Name, Phone_Number, Email, Address, Date_Of_birth, Age, Occupation)
Donation (Donation_ID, Donor_ID, Event_ID, Donation_Amount, Payment_Method)
Volunteer (Volunteer_ID, Event_ID, Name, Phone_Number, Email, Address, Date_Of_birth, Age, Role)

Strong and Weak Entities:

All entities we have are strong at this point

Derived attributes:

Donor: Age should not be mapped because we can calculate it from Date_Of_birth attribute.
Volunteer: Age should not be mapped because we can calculate it from Date_Of_birth attribute.

Donor and volunteer entities before defining derived attributes:

Donor (Donor_ID, Event_ID, Name, Phone_Number, Email, Address, Date_Of_birth, Age, Occupation)
Volunteer (Volunteer_ID, Event_ID, Name, Phone_Number, Email, Address, Date_Of_birth, Age, Role)

Donor and volunteer entities after defining derived attributes:

Donor (Donor_ID, Event_ID, Name, Phone_Number, Email, Address, Date_Of_birth, Occupation)
Volunteer (Volunteer_ID, Event_ID, Name, Phone_Number, Email, Address, Date_Of_birth, Role)

Multivalve attributes:

Event: There are no multivalve attributes here
Donor: Phone_Number
Donation: There is no multivalve attributes here
Volunteer: Phone_Number

Before defining multivalue attributes:

Donor (Donor_ID, Event_ID, Name, Phone_Number, Email, Address, Date_Of_birth, Occupation)
Volunteer (Volunteer_ID, Event_ID, Name, Phone_Number, Email, Address, Date_Of_birth, Role)

After defining multivalue attributes:

Donor (Donor_ID, Name, Email, Address, Date_Of_birth, Occupation)
Volunteer (Volunteer_ID, Event_ID, Name, Email, Address, Date_Of_birth, Role)
Donor_Phone_Numbers (Donor_ID, Phone_Number)
Volunteer_Phone_Numbers (Volunteer_ID, Phone_Number)

Composite attributes:

Event: Location (Country, City, Street_name, building number), Organizer_Name (Organizer_First_Name, Organizer_Last_Name)
Donor: Name (First_name, Last_name), Address (Country, City, Street_name, building_number)
Donation: There is no composite attributes here
Volunteer: Name (First, Last), Address (Country, City, Street_name, building number)

Event, Donor and Volunteer entities before defining composite attributes:

Event (Event_ID, Event_Name, Date, Location, Number_of_Donations, Organizer_Name)
Donor (Donor_ID, Event_ID, Name, Email, Address, Date_Of_birth, Occupation)
Volunteer (Volunteer_ID, Event_ID, Name, Email, Address, Date_Of_birth, Role)

Event, Donor and Volunteer entities after defining composite attributes:

Event (Event_ID, Event_Name, Date, Country, City, Street_name, Building_Number, Number_of_Donations, Organizer_First_Name, Organizer_Last_Name)
Donor (Donor_ID, Event_ID, First_Name, Last_Name, Email, Country, City, Street_name, Building_Number, Date_Of_birth, Occupation)
Volunteer (Volunteer_ID, Event_ID, First_Name, Last_Name, Email, Country, City, Street_name, Building_Number, Date_Of_birth, Role)

Complex attributes:

Event: Location, Organizer_Name
Donor: Name, Address, Phone_Number
Donation: There are no complex attributes here
Volunteer: Name, Address, Phone_Number

1:1 relationship:

There are no 1:1 relationships

1:M relationship:

Event : Donation → 1:M

Donor : Donation → 1:M

N:M relationship:

Event : Donor → N:M
Event : Volunteer → N:M

Before defining N:M relationships:

Event (Event_ID, Event_Name, Date, Country, City, Street_name, Building_Number, Number_of_Donations, Organizer_First_Name, Organizer_Last_Name)
Donor (Donor_ID, Event_ID, First_Name, Last_Name, Email, Country, City, Street_name, Building_Number, Date_Of_birth, Occupation)
Volunteer (Volunteer_ID, Event_ID, First_Name, Last_Name, Email, Country, City, Street_name, Building_Number, Date_Of_birth, Role)

After defining N:M relationships:

Event (Event_ID, Event_Name, Date, Country, City, Street_name, Building_Number, Number_of_Donations, Organizer_First_Name, Organizer_Last_Name)
Donor (Donor_ID, First_Name, Last_Name, Email, Country, City, Street_name, Building_Number, Date_Of_birth, Occupation)
Volunteer (Volunteer_ID, First_Name, Last_Name, Email, Country, City, Street_name, Building_Number, Date_Of_birth)
Event_Donor (Event_ID, Donor_ID)
Event_Volunteer (Event_ID, Volunteer_ID, Role)

Final Schema (With detecting strong and weak entities):

Event (Event_ID, Event_Name, Date, Country, City, Street_name, Building_Number, Number_of_Donation, Organizer_First_Name, Organizer_Last_Name)
Donor (Donor_ID, First_Name, Last_Name, Email, Country, City, Street_name, Building_Number, Date_Of_birth, Occupation)
Volunteer (Volunteer_ID, First_Name, Last_Name, Email, Country, City, Street_name, Building_Number, Date_Of_birth)
Donation (Donation_ID, Donor_ID, Event_ID, Donation_Amount, Payment_Method)
Donor_Phone_Numbers (Donor_ID, Phone_Number)
Volunteer_Phone_Numbers (Volunteer_ID, Phone_Number)
Event_Donor (Event_ID, Donor_ID)
Event_Volunteer (Event_ID, Volunteer_ID, Role)

Note: Weak entities are colored red.

3.3 Logical Design

We can consider logical design as advanced conceptual design. This stage comes after mapping, where the final schema is ready and complete. The attributes are placed in each

table under the name of the entities, and the primary and foreign keys are represented in the diagram. Its goal is to describe the data in as much detail as possible, without regard to how they will be physically implemented in the database, to have a normalized business rules.

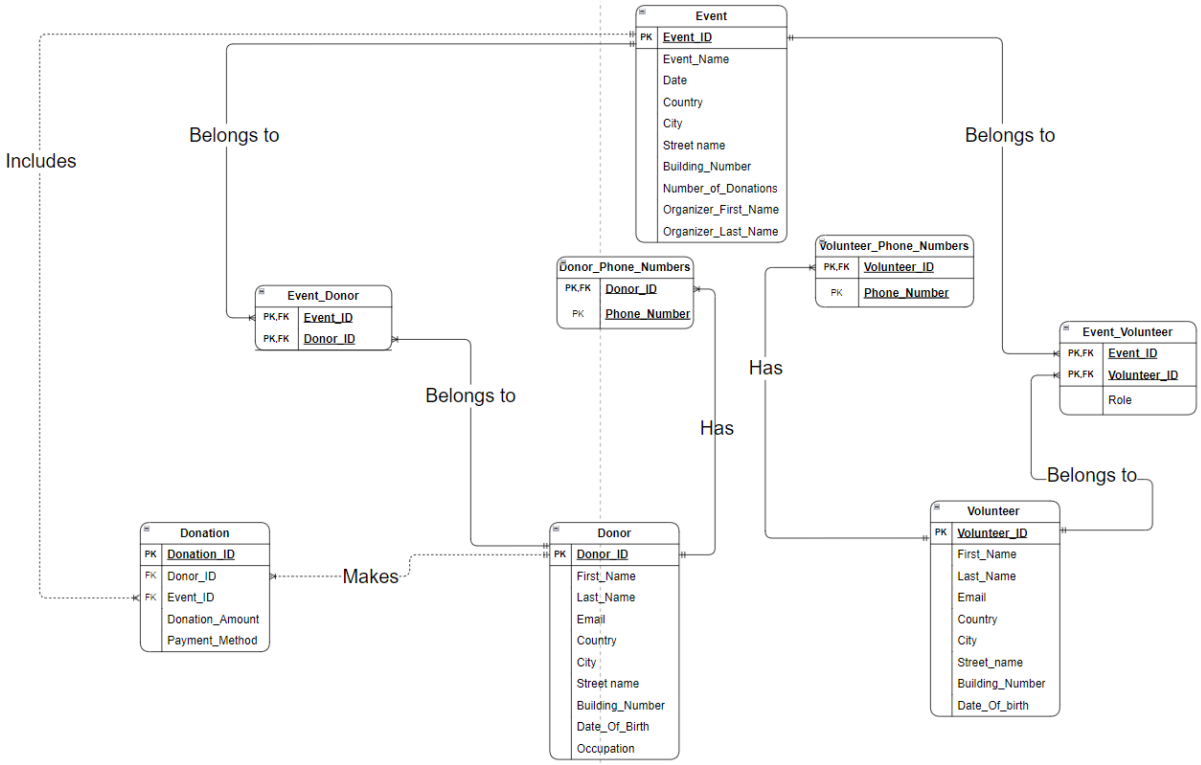
So, we conclude that components of logical design are:

• All entities.

• All attributes for each entity.

• specified primary and foreign keys for each entity.

• Resolved relationships among entities.



The steps I took while drawing the logical diagram:

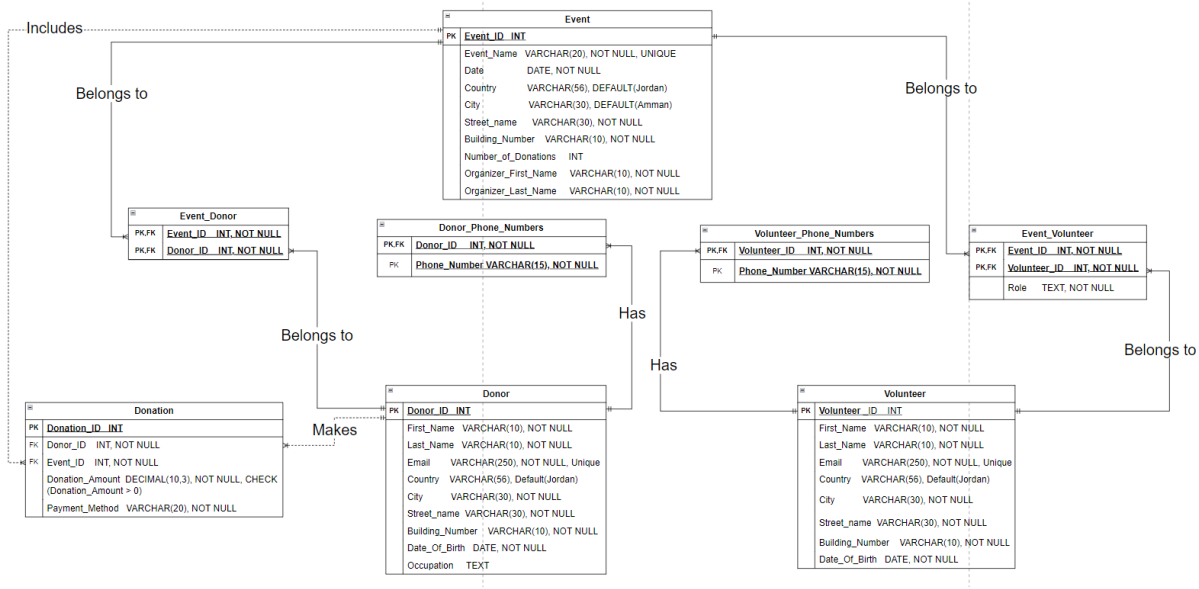
- I made the modifications that were made in the mapping stage to the tables in the conceptual design.
- I defined all the tables, including the new ones that resulted from the multivalues defining and N:M relations defining stages.
- I defined both the primary and foreign keys in each table.
- Resolved relations between entities, including the weak and strong relations.

3.4 Physical Design

Physical Model is a model derived from the logical one, on which a particular relational database management system (RDBMS) is built, in order to determine the datatype for each of the attributes in the schema, with the aim of completing our final form as an integrated model that we can then apply in SQL.

So, we conclude that components of physical design are:

• Defined Data types for fields.
• Defined constraints on these tables such as unique, not null and domain definitions to check if data are out of the range



The steps I took while drawing the Physical diagram:

- Defined all data types for the attributes and applied it to the diagram.
- Defined constraints for the attributes and applied it to the diagram.

3.5 Effectiveness of the design

• My primary entities are Event, Donor, Volunteer, and Donation, they were selected to represent the foundations of the system effectively. Each entity has a clear purpose:

Event: Represents the charitable event organized by the foundation. This is the central entity since donations and volunteers directly participates in the events.

- **Event_ID**: The identifier for each event, This attribute is very important because it indicates and defines the event, and represent its primary key. Its datatype is INT because numeric values are efficient for indexing and searching. And of course, since it is a primary key he must be unique and cannot be null.
- **Event_Name**: The Name of the event. This attribute is necessary to give each event a meaningful name that can identify it from others. Its datatype is VARCHAR(20), which is suitable for naming, and must be unique, meaning that no more than one event with the same name can be held, and it cannot be null.
- **Date**: The date of the event that will be held.This attribute is important to ensure the best scheduling and planning. It hold DATE datatype which is suitable for storing date values, and it cannot be null.
- **Country**: The country where the event is held. This attribute helps further the event location in details with another attributes. Its datatype is VARCHAR(56) which is suitable for naming, and since the largest country's name in the world consists of 56 letters (United Kingdom of Great Britain and Northern Ireland). The default status of the event's country of residence is Jordan, since the company's headquarters is in Jordan and most events held in the past was In Jordan.

- **City**: The city where the event is held. This attribute helps further the event location in details with other attributes as a sub one. It hold VARCHAR(30) datatype, which is suitable for naming. The default status of the event's city of residence is Amman since the company's headquarters is in the capital Amman and most events held was in Amman.
- **Street_name**: The name of the street that the event where the event is held. Same to the City, Country attributes, this attribute helps further the event location in details with other attributes as a sub one. It hold VARCHAR(30) datatype, which is suitable for naming, And it cannot be null.
- **Building_Number**: The number of the building where the event is held. Same as the country, City and Street_name attributes, this attribute helps further the event location in detail with other attributes as a sub one. It hold VARCHAR(10) datatype and it cannot be null.
- **Number_of_Donations**: The total number of donations donors made during the event. Same to the City and Country and Street_name and Building_Number attributes, this attribute helps further the event location in details with other attributes as a sub one. It hold INT because it is a numeric value datatype and it cannot be null.
- **Organizer_First_Name**: The first name of the event organizer. It hold VARCHAR(10) datatype which is suitable for most names, and it cannot be null.
- **Organizer_Last_Name**: The last name of the event organizer last name. This attribute is for ensuring full Organizer's identification. It hold VARCHAR(10) datatype which is suitable for most names, and it cannot be null.

Donor: Represents the individuals that financially donates to the events. This entity is important for tracking contributions and understanding who supports the foundation.

- **Donor_ID**: The identifier for each donor. This attribute is very important because it uniquely identifies each donor in the system and represents the primary key for the Donor table. It hold INT datatype because numeric values are more efficient for indexing and searching that any datatype else. And of course, since it is a primary key he must be unique and cannot be null.
- **First_Name**: The first name of the donor. This attribute helps identify the donor. Its datatype is VARCHAR(10), which is suitable for most names. It cannot be null to ensure his identification.
- **Last_Name**: The last name of the donor. This attribute is for ensuring donor's full identification. Its datatype is VARCHAR(50) which is suitable for most names, and it cannot be null.
- **Email**: The donor's email address. This attribute facilitates communication with the donor and can be used as a secondary identifier. Its datatype is VARCHAR(250), accommodating long email addresses. It must be unique to avoid duplicate records, and it cannot be null.
- **Country**: The country of residence for the donor. This attribute helps further the donor's address in detail with other attributes as a sub one. Its datatype is VARCHAR(56), which is suitable for naming, and since the largest country's name in the world consists of 56 letters (United Kingdom of Great Britain and Northern

Ireland), with a default value of Jordan, reflecting the expected locality of most donors, we can know that by seeing the previous donors records

- **City**: The city of residence for the donor. Same as the country attribute, this attribute helps further the donor's address in detail with other attributes as a sub one. Its datatype is VARCHAR(30) and defaults to Amman, aligning with the foundation's area, since most donors in the previous events are living in Amman.
- **Street_Name**: The street name of the donor's residence. Same as the country, City attributes, this attribute helps further the donor's address in detail with other attributes as a sub one. Its datatype is VARCHAR(50) and cannot be null.
- **Building_Number**: The building number of the donor's residence. Same as the country, City and Street_name attributes, this attribute helps further the donor's address in detail with other attributes as a sub one. Its datatype is VARCHAR(10) and cannot be null.
- **Date_Of_Birth**: The donor's birth date. This attribute is important to verify the donor's age and conduct analysis. Its datatype is DATE and cannot be null.
- **Occupation**: The occupation of the donor. This attribute gives us an additional context about the donor. Its datatype is TEXT and is optional, as not all donors may wish to share this information.

Volunteer: Represents the individuals who assist in organizing events. Volunteers play a crucial role in the event, so we need to record their participations continuously.

- **Volunteer_ID**: The primary key identifier for each volunteer. This attribute uniquely identifies each volunteer in the system since it's the primary key. It holds an INT datatype because numeric values are efficient for indexing and searching. Since it's the primary key, it must be unique and cannot be null.
- **First_Name**: The volunteer's first name. This attribute helps identify the volunteer personally. Its datatype is VARCHAR(10), which is suitable for most names. It cannot be null to ensure his identification.
- **Last_Name**: The last name of the volunteer. This attribute is for ensuring the volunteer's full identification. Its datatype is VARCHAR(10), which is suitable for most names, and it cannot be null.
- **Email**: The volunteer's email address. This attribute facilitates communication with the volunteer and can be used as a secondary identifier. Its datatype is VARCHAR(250), accommodating long email addresses. It must be unique to avoid duplicate entries and cannot be null.
- **Country**: The country of residence for the volunteer. This attribute helps further the volunteer's address in detail with other attributes as a sub one. Its datatype is VARCHAR(56), which is suitable for naming, and since the largest country's name in the world consists of 56 letters (United Kingdom of Great Britain and Northern Ireland), with a default value of Jordan, reflecting the expected locality of most volunteers, we can know that by seeing the previous volunteers records.
- **City**: The city of residence for the volunteer. Same as the country attribute, this attribute helps further the volunteer's address in detail with other attributes as a sub one. Its datatype is VARCHAR(30) and defaults to Amman, aligning with the foundation's area, since most volunteer in the previous events are living in Amman.

- **Street_Name**: The street name of the volunteer's residence. Same as the country, City attributes, this attribute helps further the volunteer's address in detail with other attributes as a sub one. Its datatype is VARCHAR(50) and cannot be null.
- **Building_Number**: The building number of the volunteer's residence. Same as the country, City and Street_name attributes, this attribute helps further the volunteer's address in detail with other attributes as a sub one. Its datatype is VARCHAR(10) and cannot be null.
- **Date_Of_Birth**: The birth date of the volunteer. This attribute is crucial to verify the volunteer's age and gives them a specific roles. It holds DATE datatype and cannot be null.
- **Role**: The role assigned to the volunteer during the event. This attribute identify each donor's responsibilities during the event. It holds VARCHAR(50) datatype, which is suitable for role names, and it cannot be null.

Donation: Represents the financial support made by donors. Each donation must be recorded with its details, such as the amount, payment method used, and the event it is associated with.

- **Donation_ID**: The identifier for each donation. This attribute is very important because it identifies each donation in the system and represents the primary key for the Donation table. It holds INT datatype because numeric values are efficient for indexing and searching. Since it is a primary key, it must be unique and cannot be null.
- **Donor_ID**: The identifier of the donor who made the donation. This attribute establishes the relationship between the donation and the donor. It holds INT datatype and acts as a foreign key referencing the Donor table. It cannot be null, as every donation must be linked to a donor.
- **Event_ID**: The identifier of the event which the donation was made in. This attribute establishes the relationship between the donation and the event. It holds INT datatype and acts as a foreign key referencing the Event table. It cannot be null, as every donation must be associated with an event.
- **Donation_Amount**: The amount of money donated by the donor in 1 time. This attribute is crucial for tracking the financial contributions made during the event. It holds DECIMAL(10,3) datatype to ensure precision, and it cannot be null, and have to be greater than 0.
- **Payment_Method**: The method donor used to make the donation, such as cash or credit card. This attribute provides the feature to track donation movements in detail . It holds VARCHAR(20) datatype, which is suitable for naming, and it cannot be null.

• Relationships:

There is no 1:1 relationship

There is 1:M relationship between Event and Donation since one event can include multiple donations, but one donation belongs to one event.

There is 1:M relationship between Donor and Donation since one donor can make many donations, but each donation is done by one donor.

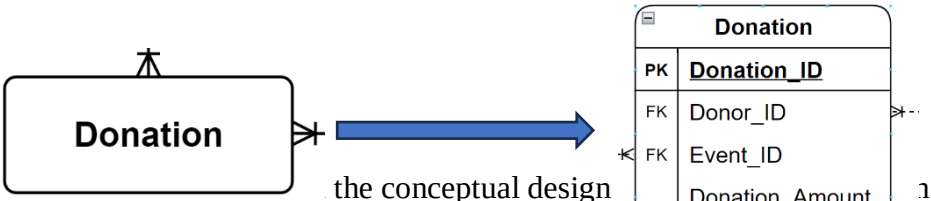
There is an M:N relationship between Event and Donor since a donor can participate in multiple events, and each event can have multiple donors. This was resolved with the Event_Donor table which consists of Event_ID and Donor_ID, both are foreign keys and represent a part of the primary key. I implement that because the logical and physical design can't have multi-value, so I made this table which represents the intermediary between them where it has a 1:M relationship with both of them.

There is an M:N relationship between Event and Volunteer since a Volunteer can Volunteer in multiple events, and each event can have multiple Volunteers. This was resolved with the Event_ Volunteer table which consists of Event_ID and Volunteer_ID, both are foreign keys and represent a part of the primary key. I implement that because the logical and physical design can't have multi-value, so I made this table which represents the intermediary between them where it has a 1:M relationship with both of them.

• I created the conceptual design in order to lay the foundation stone for the database that I will walk on in the mapping steps in order to create the logical and physical design, as it represents the entities and the relationships between them. By using them, I knew the approach that I would follow in order to reach the desired result in the remaining steps. This design is intended for stakeholders, such as the foundation and event organizer, to verify that their requirements are accurately represented in the basic database structure.

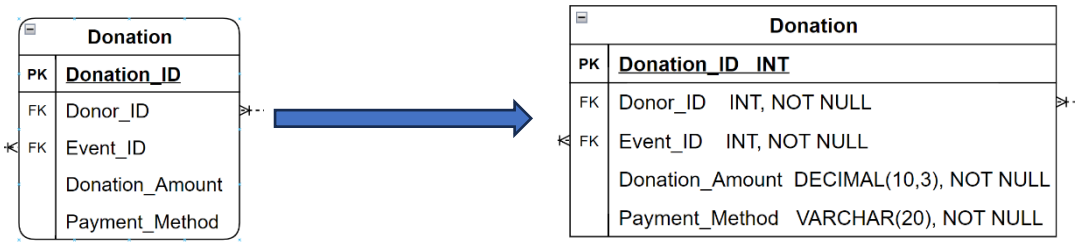
• A schema is a diagram that divides data into tables and defines relationships between them. It differs between its types, but the importance is the same, to convert a design into a format that can be implemented in a database system and benefit from it.

• Logical design defines entities and their attributes with relationships (of different types), we add information like attributes, resolved relationships between them after mapping, and specified primary and foreign keys for each entity also after mapping are added to the design. It enables us to understand the concept of the database in a simpler and smoother way, as the details in it are much more than the conceptual, and it also leads us to physical design.



the conceptual design … an empty entity connected to other entities, but it became much more m… the logical design with its attributes.

• In the physical design, we add constraints and data types for each attribute, which is the main difference between it and the logical design. It makes it have clear criteria when inserting the data into our database.



For example, here I put the data type and constraints for each attribute in this table, which are rules and conditions we follow when inserting the input.

• Normalization is the process of resetting data to its normal state, in order to improve data integrity and reduce redundancy. There are 3 standard forms in the normalization process, which are:

1st NF: Here, if there is any multi-valued or complex attribute on relation, we should extract them to another relation (new table) using 1:M relationship, and the primary key of the new relation is a combination of the primary key of the original relation plus an attribute from the newly created relation for unique identification. Also, we should remove any composite attribute by taking the simple attributes only. Its purpose is to access a data format with the following characteristics:

- Each cell in it contains single value, no any composite or multi-value attributes
- Each record in it is unique.
- Values stored in a column is of the same domain.
- All the columns in a relation have unique names

For example:

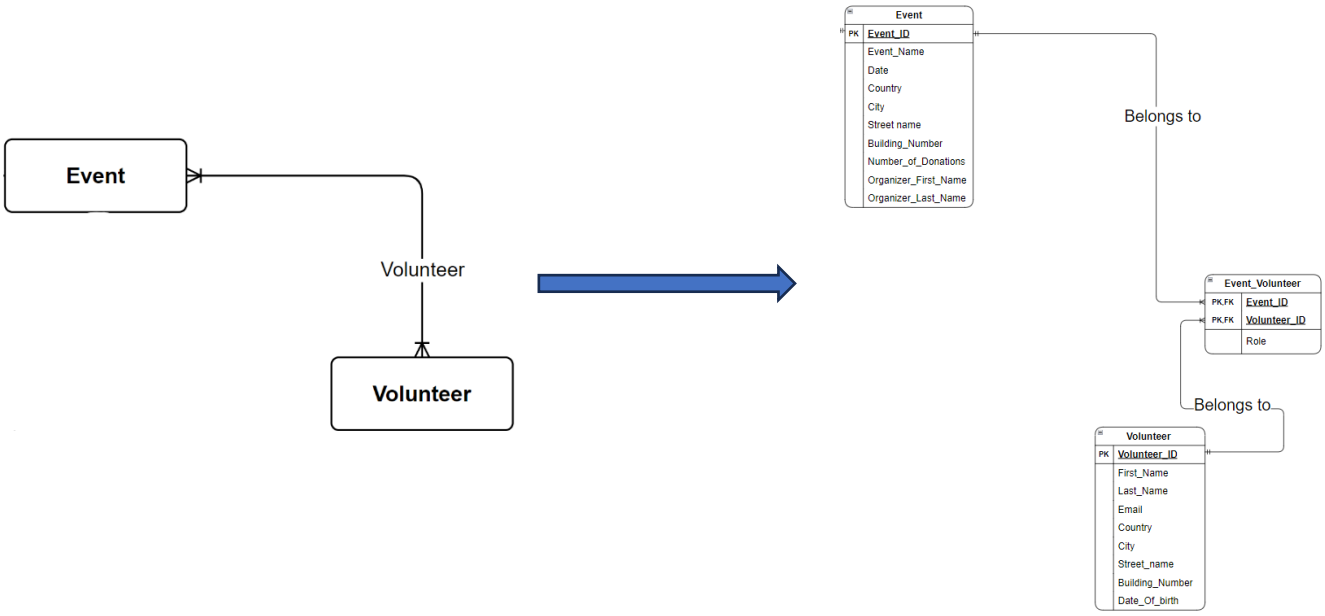
I extracted the multi-value attribute Phone_Number from the Volunteer table into to a new table called (Volunteer_Phone_Numbers) and connected it with the Volunteer table using 1:M relationship. The primary key of the new table is a combination of the Volunteer_ID PK (Which is also an FK from the Volunteer table), and the phone_number attribute.

2nd NF:

Here, we remove all partial dependencies by break the relation to two relations. Remove the attribute which is causing partial dependency and move it to some other relation where it fits in well. Its purpose is to access a data format with the following characteristics:

- All tables are in 1st NF
- All non-key attributes are fully functional dependent on the primary key. (There should be no partial dependency)

For example:



Initially, we had an M:N relationship (Partial Dependency) between the volunteer and the event, I merged them into a single table, and the primary key of the new single table is a composite of the primary keys of the two tables. This will make it much easier for me to deal with the data, without data redundancy in my databases.

3rd NF:

Here, we remove any transitive dependencies; a non-key attribute may not be functionally dependent on another non-key attribute. Its purpose is to access a data format with the following characteristics:

- All tables are in 2nd NF
- The are no transitive functional dependencies. All the attributes in a relation are identified only by the candidate keys of that relation and not by any non-prime attributes

• Dabase design process aims to achieve the best possible result system, if not as desired, at least fulfilling all requirements, including user, system, and data requirements starting from the conceptual design that lays the foundation, to the logical design that contains attributes, keys, and relationships to connect the tables to ensure a better understanding of the database concept, to finally, the physical design implemented constraints and data types to ensure consistency and enforce business rules.

4 Normalization

4.1 1st NF

Relations	Attributes	Violation description	Solution – Relations
The relations schema	The attribute name	Describe why it is not in the 1st NF (the violation)	Show the schema for each affected relation.
Donor (Donor_ID, First_Name, Last_Name, Address, Phone _Number)	Address	Address is a composite attribute	Donor (Donor_ID, First_Name, Last_Name, Country, City, Street_name, building number, Phone _Number)
Donor (Donor_ID, First_Name, Last_Name, Country, City, Street_name, building number, Phone _Number)	Phone _Number	Phone _Number is a multi-value attribute	Donor (Donor_ID, First_Name, Last_Name, Country, City, Street_name, building number) Donor_Phone_Numbers (Donor_ID, Phone_Number)

4.2 2nd NF

Relations	FDs	Violation description	Solution – Relations
The relations schema	Show the functional dependencies causing the violation	Describe why it is not in the 2nd NF (the violation)	Show the schema for each affected relation.
Volunteer (Volunteer_ID, Event_ID, Event_Name, Event_Date, First_Name, Last_Name, Phone _Number, Role)	Volunteer_ID → {Name, phone, Role } Event_ID → {Event_Name, Event_Date}	The non-prime attributes (Name, Phone_Number, and Role) depend on a part of the key Volunteer_ID, and the non-prime attributes (Event_Name and Date) depend on a part of the pk which is Event_ID So we have partial dependencies	**Volunteer** (Volunteer_ID, Event_ID, First_Name, Last_Name, Phone _Number) **Event**(Event_ID, Event_name, Date) **Event_Volunteer** (Event_ID, Volunteer_ID, Role)
Passport (Passport_ID, PassengerID, Passenger_First_Name, Passenger_Last_Name, Passenger_Email,	Passport_ID → { Passport_Number, Issue Issue_Date Date, ExpiryDate}	The non-prime attributes (Passport_Number, Issue_Date and Expiry_Date) depend on a part of the pk	**Passport (**Passport_ID, PassportNumber, Issue Issue_Date Date, ExpiryDate**)**

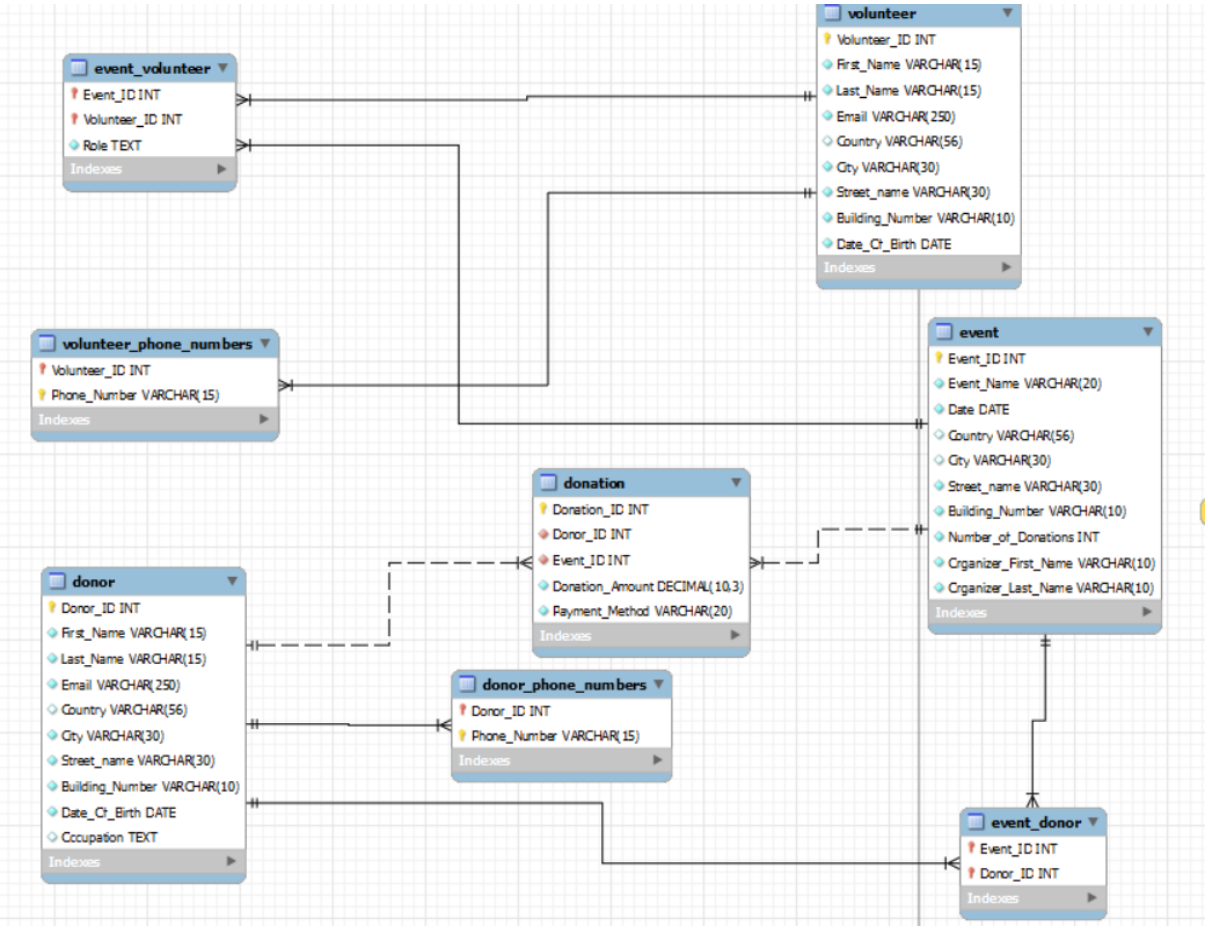
Passport_Number, Issue_Date, Expiry_Date)	Passenger_ID → { Passenger_First_Name, Passenger_Last_Name, Passenger_Email}	which is Passport_ID And the non-prime attributes (Passenger_First_Name, Passenger_Last_Name, Passenger_Email) depend on a part of the key which is Passenger_ID So we have partial dependencies	**Passenger**(Passenger_ID, Passport_ID, Passenger_First_Name, Passenger_Last_Name, Passenger_Email)

4.3 3rd NF

Relations	FDs	Violation description	Solution – Relations
The relations schema	Show the functional dependencies causing the violation	Describe why it is not in the 3rd NF (the violation)	Show the schema for each affected relation.
Volunteer (Volunteer_ID, Event_ID, Event_Name, Event_Date, First_Name, Last_Name, Phone _Number, Role)	Volunteer_ID → Event_ID Event_ID → {Event_Name, Date} Volunteer_ID → {Event_Name, Date}	The PK Volunteer_ID can determine any attribute in the table including Event_ID. At the same time, the Event_ID can determine Event_name, Date. As Event_ID is not a candidate here, we have a transitive dependency.	**Volunteer** (Volunteer_ID, Event_ID, First_Name, Last_Name, Phone _Number) **Event**(Event_ID, Event_name, Date) **Event_Volunteer** (Event_ID, Volunteer_ID, Role)
Passenger(Passenger_ID, Passport_ID, Passenger_First_Name, Passenger_Last_Name, Passenger_Email, Passport_Number, Issue_Date, Expiry_Date)	Passenger_ID → Passport_ID Passport_ID → { Passport_Number, Issue Issue_Date Date, ExpiryDate} Passenger_ID → { Passport_Number, Issue_Date, Expiry_Date}	The PK (ID) can determine any attribute in the table including AuthorID. At the same time, the AuthorID determine fname, lname, and nationality. As AuthorID is not a candidate here,	**Passenger**(Passenger_ID, Passport_ID, Passenger_First_Name, Passenger_Last_Name, Passenger_Email) **Passport (**Passport_ID, PassportNumber, Issue_Date, Issue_Date, ExpiryDate**)**

		we have a transitive dependency.	

5 Physical Schema



6 Database Development

6.1 Database Overview

Table	Name	Description
1.	Event	It stores all the data about the charity events, which helps us later when counting the events annually such as the total number of donations made in them, the number of volunteers and the number of donors. This data (Attributes) represents:

		Event_ID, the identifier for each event, and the PK for this strong entity, its datatype is INT and carries the constraints NOT NULL and UNIQUE of course because it's the PK **Event_Name**, stores the Name of the event. Its datatype is VARCHAR(20) and carries the constraints (NOT NULL, UNIQUE) **Date**, stores the date of the event that will be held. Its datatype is DATE and carries NOT NULL constraint **Country**, the country where the event is held. Its datatype is VARCHAR(56), carries the DEFAULT(Jordan) constraint of course since the company's headquarters is in Jordan and most events held was In Jordan **City**, city where the event is held. Its datatype is VARCHAR(30), and of course it has DEFAULT(Amman) constraint since the company's headquarters is in the capital Amman and most events held was in Amman **Street_name**, street address of the event location. Its datatype is VARCHAR(30) and carries NOT NULL constraint **Building_Number**, building number where the event will be held. Its datatype is VARCHAR(10) and carries NOT NULL constraint **Number_of_Donations**, total number of donations done from the donors in the event. Its datatype is INT and carries NOT NULL constraint **Organizer_First_Name**, the event organizer first name. Its datatype is VARCHAR(10) and carries NOT NULL constraint **Organizer_Last_Name**, the event organizer last name. Its datatype is VARCHAR(10) and carries NOT NULL constraint There is no FKs here
2.	Donor	Tracking the person who donates money to the charity event and saving his data. This data enables us to make statistics about the average number of consumers in a single event, and invite donors to

		upcoming events, and can also be referred to in the event of any circumstances surrounding a case or felony, in order to ensure that the donor was present at the event as an excuse for absence. Donor's data (Attributes) represents: **Donor_ID**, the identifier for each donor and the PK of this strong entity. Its datatype is INT, and carries NOT NULL and UNIQUE constraints of course since it is the PK **First_Name**, First name of the donor. Its datatype is VARCHAR(10), and carries NOT NULL constraint **Last_Name**, Last name of the donor. Its datatype is VARCHAR(10), and carries NOT NULL constraint **Email**, Email address of the donor. Its datatype is VARCHAR(250), and its constraint is (UNIQUE, NOT NULL) **Country**, the country where the donor lives. Its datatype is VARCHAR(56), and its constraint is DEFAULT(Jordan) since most donors are from Jordan. **City**, the city where the donor lives. Its datatype is VARCHAR(30), and its constraint is NOT NULL **Street_name**, the street name of the donor's house. Its datatype is VARCHAR(30) and carries NOT NULL constraint **Building_Number**, the building number of the donor's house. Its datatype is VARCHAR(10) and carries NOT NULL constraint **Date_Of_Birth**, the donor's date of birth. Its datatype is DATE, and carries NOT NULL constrain **Occupation**, Donor's occupation. Its datatype is TEXT and does not carry constraint. There is no FKs here
3.	Volunteer	Stores information about individuals assisting during events. Volunteers play a crucial role in event management and donor guidance. The benefits of this table are very similar to the Donor's

		table, as it allows us to analyze things like the effectiveness of each volunteer and the number we need for upcoming events. It can also be useful as an excuse for absence in the event in the event that he gets into a problem and is innocent of it. Volunteer's data (Attributes) represents: **Volunteer_ID,** the identifier for each volunteer and the PK of this strong entity. Its datatype is INT, and carries NOT NULL and UNIQUE constraints of course since it is the PK **First_Name**, First name of the volunteer. Its datatype is VARCHAR(10), and carries NOT NULL constraint **Last_Name**, Last name of the volunteer. Its datatype is VARCHAR(10), and carries NOT NULL constraint **Email**, Email address of the volunteer. Its datatype is VARCHAR(250), and carries NOT NULL constraint **Country**, the country where the volunteer lives. Its datatype is VARCHAR(56), and its constraint is DEFAULT(Jordan) since most donors are from Jordan. **City**, the city where the volunteer lives. Its datatype is VARCHAR(30), and carries NOT NULL constraint **Street_name**, the street name of the volunteer's house. Its datatype is VARCHAR(30) and carries NOT NULL constraint **Building_Number**, the building number of the volunteer's house. Its datatype is VARCHAR(10) and carries NOT NULL constraint **Date_Of_Birth**, the volunteer's date of birth. Its datatype is DATE, and carries NOT NULL constrain. There is no FKs here
4.	Donation	The Donation table tracks the financial contributions made by donors during charity events. This table is a critical part of the database as it connects donors to specific events, providing

		valuable insights into donation trends, event success, and individual donor contributions. The table also supports event-based financial reporting and donor recognition. Donation's data (Attributes) represents: **Donation_ID**, the identifier for each donation and the PK of this strong entity. Its datatype is INT, and carries NOT NULL and UNIQUE constraints of course since it is the PK **Donor_ID**, FK that references Donor_ID from Donor tablet to associate each donation with a specific donor. Its datatype is INT, and carries NOT NULL constraints **Event_ID**, FK that references Donor_ID from Donor tablet to link each donation to a specific event. Its datatype is INT, and carries NOT NULL constraints. **Donation_Amount**, The amount of money donated. Its datatype is DECIMAL(10,3), and carries NOT NULL and CHECK (Donation_Amount > 0) constraint. **Payment_Method**, the method used to pay for the donation, like (Credit Card, Cash) for example. Its datatype is VARCHAR(20) and carries NOT NULL constraint
5.	Donor_Phone_Numbers	The Donor_Phone_Numbers table is used to store multiple phone numbers for each donor. This table resolves the multivalued attribute issue for phone numbers in the Donor table, ensuring proper normalization and scalability, and helps us communicate with the donor in the future. Donor_Phone_Numbers' data (Attributes) represents: **Donor_ID**, part of the composite PK that links donors with their phone numbers, and FK that references Donor_ID from Donor table. Its datatype is INT, and carries NOT NULL constraints. **Phone_Number**, part of the composite PK that links donors with their phone numbers. Its datatype

		is VARCHAR(15), and carries NOT NULL constraints.
6.	Volunteer_Phone_Numbers	This table is similar to the Donor_Phone_Numbers table but for volunteers. It stores multiple phone numbers for each volunteer, resolving the multivalued attribute issue in the Volunteer table. Volunteer_Phone_Numbers' data (Attributes) represents: **Volunteer_ID**, part of the composite PK that links Volunteers with their phone numbers, and FK that references Volunteer_ID from Volunteer table. Its datatype is INT, and carries NOT NULL constraints. **Phone_Number**, part of the composite PK that links Volunteers with their phone numbers. Its datatype is VARCHAR(15), and carries NOT NULL constraints.
7.	Event_Donor	The Event_Donor table represents M:N relationship between events and donors. It helps identify which donors participated in which events. Event_Donor's data (Attributes) represents: **Event_ID**, part of the composite PK that links donors with the events, FK that references Event_ID from Event table. Its datatype is INT, and carries NOT NULL constraints. **Donor_ID**, part of the composite PK that links donors with the events, FK that references Donor_ID from Donor table. Its datatype is INT, and carries NOT NULL constraints.
8.	Event_Volunteer	The Event_Volunteer table represents the M:N relationship between events and volunteers. It tracks which volunteers participated in which events. Event_ Volunteer's data (Attributes) represents: **Event_ID**, part of the composite PK that links Volunteers with the events, FK that references Event_ID from Event table. Its datatype is INT, and carries NOT NULL constraints. **Volunteer_ID**, part of the composite PK that links Volunteers with the events, FK that references

		Volunteer_ID from Volunteer table. Its datatype is INT, and carries NOT NULL constraints. **Role**, Which is the volunteer's role in the event. Its datatype is TEXT, and carries NOT NULL constraints.

View	Name	Description
1.	Recent_Events	This view is designed to display the most recent events, sorted by event ID in descending order. It helps event organizers quickly identify newly added or upcoming events based on their unique IDs. For the implementation, I used a SELECT query on the Event table with an ORDER BY clause to sort events in descending order of their IDs.
2.	Top_Donors	This view shows each donor's ID, full name, and their total contributions across all events. It helps event organizers assess donor engagement and identify top contributors for recognition or further campaigns. For the implementation, I used a JOIN between the Donor and Donation tables to calculate the total contributions made by each donor, grouping by donor ID and name.
3.	Volunteers_In_Event	This view lists all volunteers participating in each event. It provides event organizers with a quick overview of who contributed to each event, facilitating better volunteer management and coordination. For the implementation, I combined the Volunteer, Event, and Event_Volunteer tables using JOIN to retrieve volunteer details for each event.
4.	Average_Donation_Amount_Per_Event	This view calculates the average donation amount for each event. It helps event organizers assess donation trends and the financial success of specific events. For the implementation, I used a

		JOIN between the Event and Donation tables and group the results by event ID and name to calculate the average donation amount per event.

Procedure	Name	Description
1.	Add_Event	This process adds a new event after requiring the event name, date, location, and organizer name. It ensures that all event data is captured and stored in the database. For the implementation, the procedure inserts these details into the Event table, ensuring all required fields like the event name, date, and location are recorded accurately.
2.	Events_By_Donor	This procedure shows all the events that a specific donor donated in and the amount he/she donated for each event. This helps event organizers understand how much a donor has contributed and how often he participate in events. For the implementation, I used JOIN between the Event and Donation tables to link the donor's ID to each event they contributed to, and display the total donation amount for each event.
3.	Remove_Volunteer	This procedure removes a volunteer from a specific event. It helps to manage volunteer participants. For the implementation, I used DELETE operation on the Event_Volunteer table where the volunteer's ID and the event ID match, to effectively remove their participation from the event.
4.	Donations_By_Donor	This procedure gives a list of all donations made by a particular donor, including the event name and donation amount. This action helps organizers track how often a donor donates and how much they donate. For the implementation, I used JOIN between the Event and Donation tables to link the donations with the events, to allow the procedure to display the event names alongside the corresponding donation amounts made by the donor.

6.2 Security

Each user who can access the database must have limited access, i.e. to the extent he needs to do his role exactly and no more, to ensure security on the database. Users in my database:

1. Admin, the admin on the whole database.

```
CREATE USER 'Admin' IDENTIFIED BY '%/999';
```

2. Events_User, his role is to follow up on events.

```
CREATE USER 'Events_User' IDENTIFIED BY '&*4532';
```

3. Donations_User, his entire role is based on donations, to ensure that all donations are correct and there are no problems.

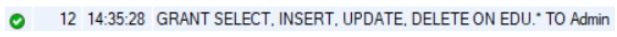
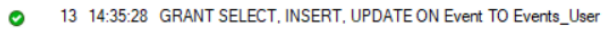
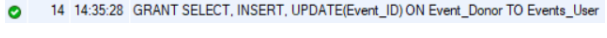
```
CREATE USER 'Donations_User' IDENTIFIED BY '@$2005';
```

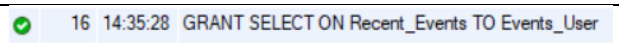
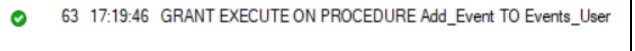
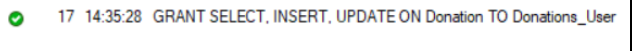
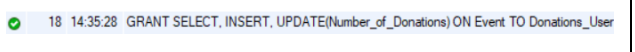
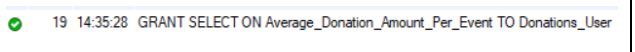
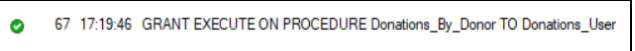
4. Donors_User, his role is to audit everything related to the donors.

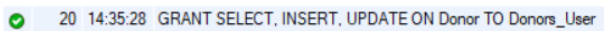
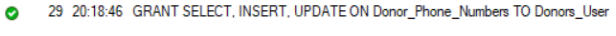
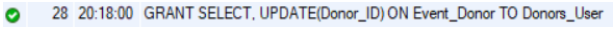
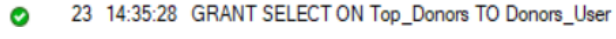
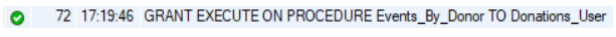
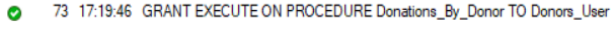
```
CREATE USER 'Donors_User' IDENTIFIED BY '!#9247';
```

5. Ahmed his role is to audit everything related to the volunteers.

```
CREATE USER 'Volunteers_User' IDENTIFIED BY '$%321';
```

User name	Privilege Command	Description	Screenshot
Admin	GRANT SELECT, INSERT, UPDATE, DELETE ON EDU.* TO Admin;	The Admin has full access to all tables and can perform any action seeing, adding, and updating, and deleting on any data within the database.	
Events_User	GRANT SELECT, INSERT, UPDATE ON Event TO Events_User;	The Events_User can view, add, and update Event information like event name, date, and location.	
	GRANT SELECT, INSERT, UPDATE(Event_ID) ON Event_Donor TO Events_User;	The Events_User can manage which donors are linked to each event.	

	GRANT SELECT, INSERT, UPDATE(Event_ID) ON Event_Volunteer TO Events_User;	The Events_User can manage which volunteers are participating in each event.	
	GRANT SELECT ON Recent_Events TO Events_User;	The Events_User can view the most recent events.	
	GRANT EXECUTE ON PROCEDURE Add_Event TO Events_User;	The Events_User can run the Add_Event procedure to create new events.	
Donations_User	GRANT SELECT, INSERT, UPDATE ON Donation TO Donations_User;	He can see every donation made by anyone, and he can update a donation row	
	GRANT SELECT, INSERT, UPDATE(Number_of_Donations) ON Event TO Donations_User;	He can see, insert, and update the number of donations done in an event	
	GRANT SELECT ON Average_Donation_Amount_Per_Event TO Donations_User;	The Donations_User can view the average total donation amount for each event.	
	GRANT EXECUTE ON PROCEDURE	The Events_User can run the	

	Donations_By_Donor TO Donations_User;	Donations_By _Donor procedure to see the donations done by a donor.	
Donors_User	GRANT SELECT, INSERT, UPDATE ON Donor TO Donors_User;	He can see, insert, and update a donor's data	
	GRANT SELECT, INSERT, ON Donor_Phone_Numbers TO Donors_User;	He can see, insert, and update a the Donor_Phone _number table	
	GRANT SELECT, UPDATE(Donor_ID) ON Event_Donor TO Donors_User;	He can see, insert, and update the donor's ID in a event ticket	
	GRANT SELECT ON Top_Donors TO Donors_User;	The Donors_User can view th top donors in all events based on their total donations.	
	GRANT EXECUTE ON PROCEDURE Events_By_Donor TO Donations_User;	The Donors_User can run the Events_By_D onor procedure to see a donor participated in which events.	
	GRANT EXECUTE ON PROCEDURE Donations_By_Donor TO Donors_User;	The Donors_User can run the Donations_By _Donor	

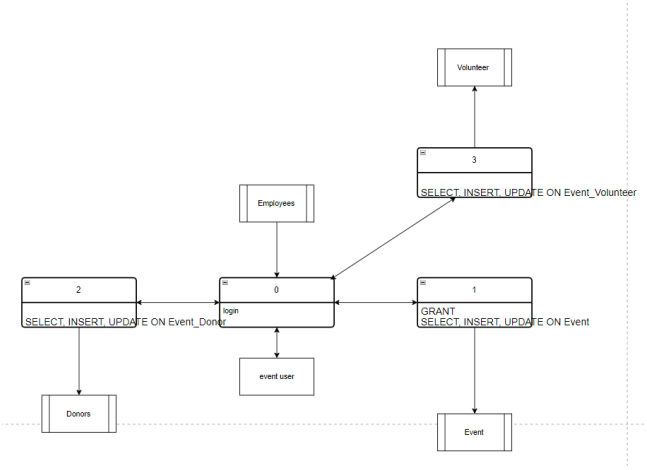
		procedure to see donation details made by a specific donor.	
Volunteers_User	GRANT SELECT, INSERT, UPDATE ON Volunteer TO Volunteers_User;	He can see, insert, and update a volunteer 's data	24 14:35:28 GRANT SELECT, INSERT, UPDATE ON Volunteer TO Volunteers_User
	GRANT SELECT, INSERT, UPDATE ON Volunteer_Phone_Numbers TO Volunteers_User;	He can see, insert, and update a volunteer's phone number	26 20:15:38 GRANT SELECT, INSERT, UPDATE ON Volunteer_Phone_Numbers TO Volunteers_User
	GRANT SELECT, UPDATE(Volunteer_ID) ON Event_Volunteer TO Volunteers_User;	He can see, insert, and update the volunteer 's ID in a event ticket	27 20:16:12 GRANT SELECT, UPDATE(Volunteer_ID) ON Event_Volunteer TO Volunteers_User
	GRANT SELECT ON Volunteers_In_Event TO Volunteers_User;	The Volunteers_User can see the participated volunteers in each event.	27 14:35:28 GRANT SELECT ON Volunteers_In_Event TO Volunteers_User
	GRANT EXECUTE ON PROCEDURE Remove_Volunteer TO Volunteers_User;	The Volunteers_User can run the Remove_Volunteer procedure to remove a volunteer from event.	78 17:19:46 GRANT EXECUTE ON PROCEDURE Remove_Volunteer TO Volunteers_User

6.3 User Interface

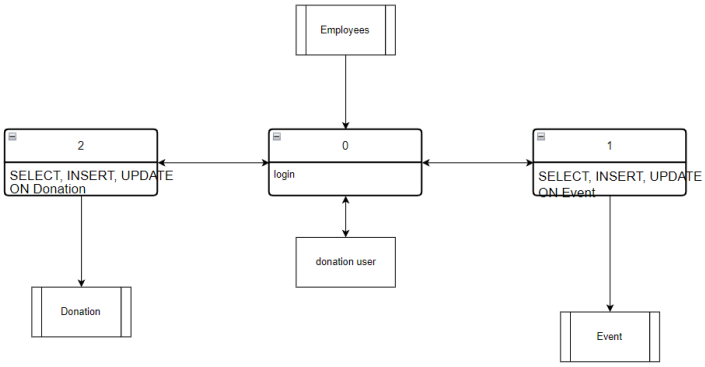
6.3.1 Data Movement Diagrams

DFDs:

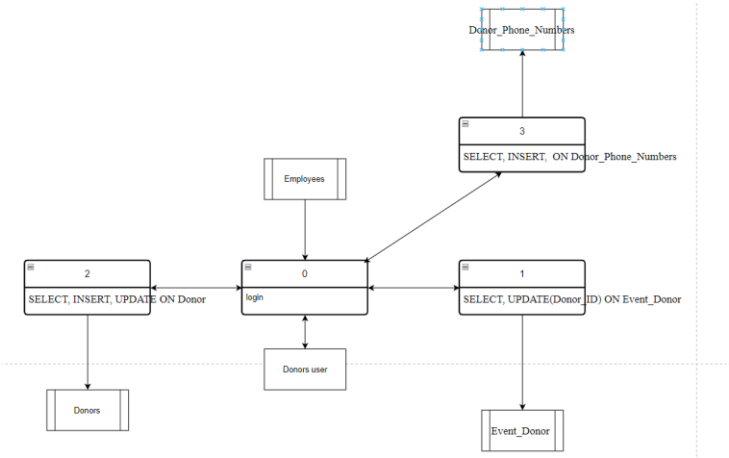
Events_User:



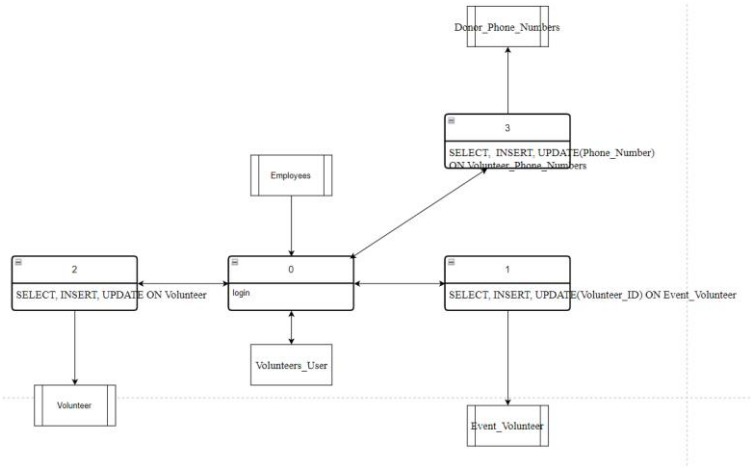
Donations_User:



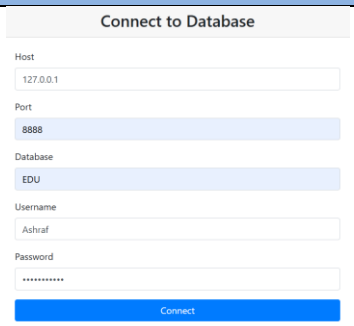
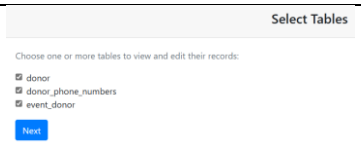
Donors_User:

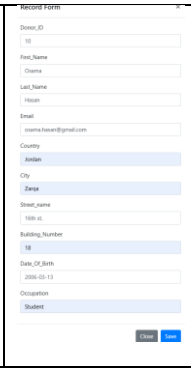


Volunteers_User:



6.3.2 Interfaces Development

Page ID	Title	Description	Screenshot
1.	Connect to database	On this page, the user logs in by entering his username and password. I have logged in using user Ashraf, the donors data administrator.	
2.	Select tables	Here the user chooses what he wants from the tables that he has permissions on. The permissions differ from one table to another according to what I gave him. I picked all the tables he have permissions on.	
3.	Database tables	From here the user can implement his permissions on all the tables he has. I selected the Donor table.	

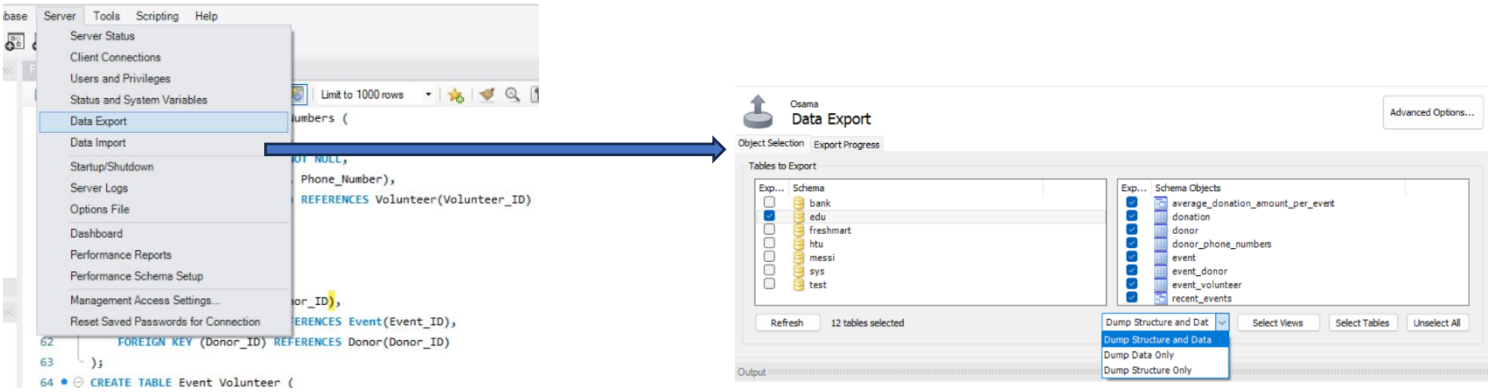
4.	Add, edit record	From here, the user can add or modify a specific record, depending on the permissions he has. User Ashraf has permission to update in the Donor table.	

7 Maintenance

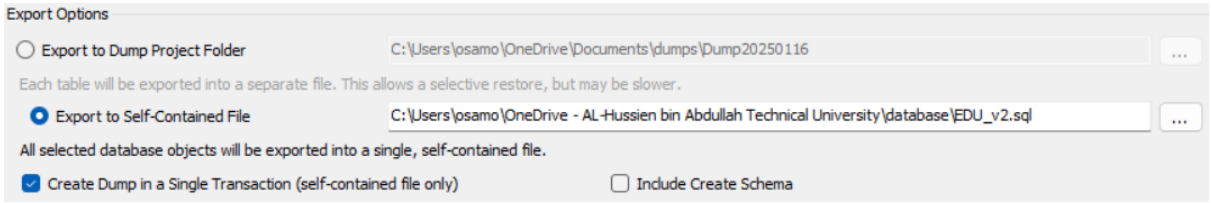
7.1 Database recovery & backups

Any organization must always take precautions to protect data, whether from internal or external breaches. For example, internally, some or all of the data is deleted by an employee, either by mistake or intentionally. For example, externally, availability is compromised and data is deleted from the organization. An important way to take precautions is backups, which means creating spare copies of the database, in order to recover it in case it is hacked.

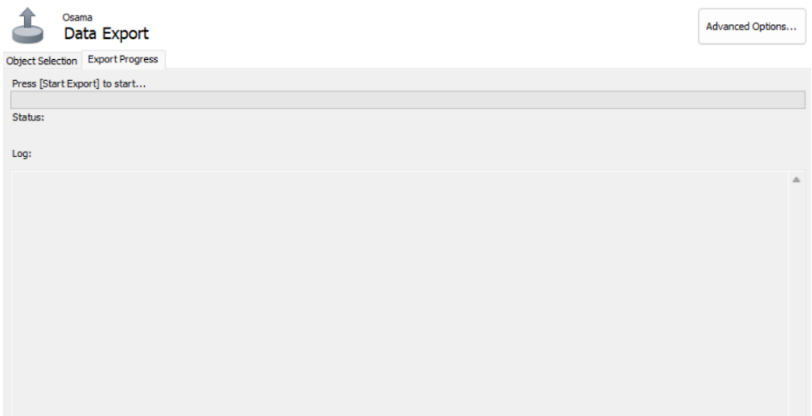
Steps for backups in my database:



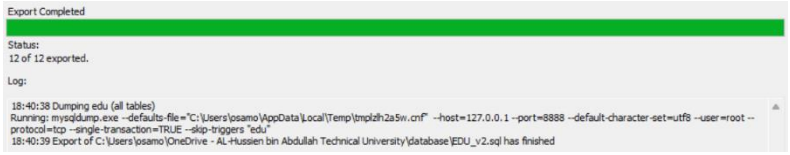
We must select the database that you want to backup, and click on dump structure and data.



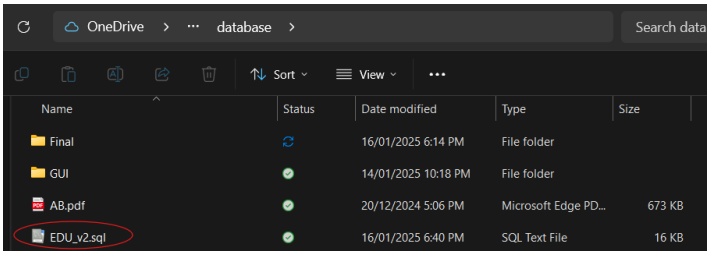
Name the file and choose where to save it



Go to Export Progress and click on Start Export



Now, the backup is completed and saved



7.2 Database maintenance in general

Database Maintenance refers to a series of tasks designed to enhance and optimize your database. These tasks include routines aimed at improving performance, freeing up disk space, detecting data errors, checking for hardware issues, updating internal statistics, and handling other essential functions (but often overlooked).
Workspace has long included a companion program called ‘Database Maintenance’ (previously known as ‘System Utilities’ before 2009), which is installed alongside it. Unfortunately, this program is not well-known. While many people may notice it in the Start Menu, few actually open it, which is a missed opportunity. The ‘Database Maintenance’ program should be run at least once every 14 days to ensure your server remains in optimal ‘health.’

Categories or types of database Maintenance:

1. **Index Defragmentation:**
   SQL databases use indexes to optimize searches by storing sorted copies of data in a fast-access memory. Each table in the Office Tools Professional database has at least one indexed column, often for ID values. Indexes allow the database to quickly locate specific data (e.g., "All time cards for Joe Demo") without scanning every record. A well-maintained index can improve query performance by over 1,000 times.

   Over time, indexes can become fragmented, like a hard drive, with data scattered into multiple parts. This fragmentation slows down the database. To resolve this, the maintenance plan assesses index size, usage, and fragmentation, applying:

- INDEX_REORGANIZE for minor issues.
- INDEX_REBUILD_ONLINE for moderate fragmentation.
- INDEX_REBUILD_OFFLINE for severe cases.

   Regular index maintenance ensures optimal database performance.

2. **Log File Maintenance:**
   SQL databases use log files to track every transaction. These log files are important because they allow us to restore the database to any previous case. If an issue apears, either corruption, accidentally, or maliciously, you can recover your data as it was just before the event. This makes log files an essential part of the system, and require dedicated maintenance.
   Log files grow automatically as more data is added, it creates Virtual Log Files (VLFs) to store operations. These VLFs enable the SQL Server to locate specific operations in the log quickly, avoiding the need to scan the entire file. However, having too many VLFs can destroy the performance by slowing down data insertion into the log file.
   We monitor VLFs to identify performance issues. If necessary, we adjust log file growth settings, such as the size of increments when the file expands. This process compacts the VLFs, restoring performance without losing any log data.

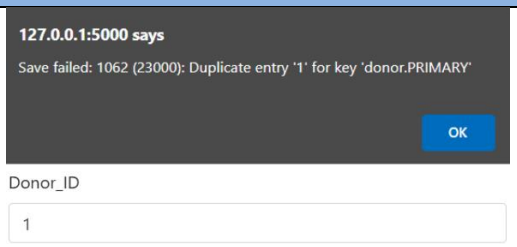
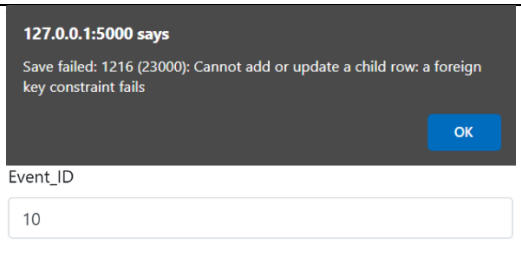
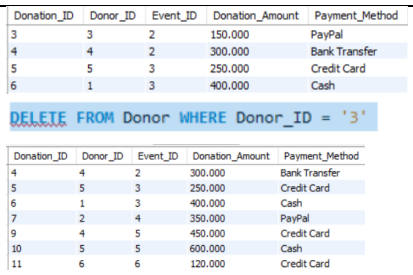
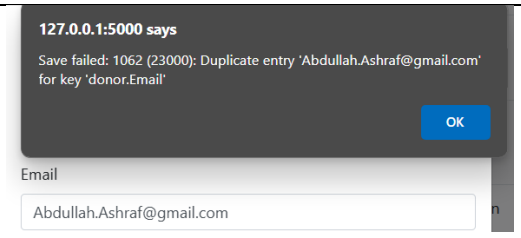
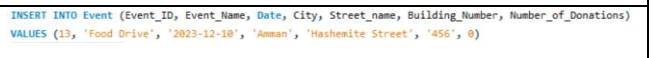
3. **File/Data Compaction:**
   SQL databases grow and shrink by a certain increment as long as we use it, which makes the space required grows by the same certain increment. However, as data is saved, it may become fragmented, with chunks stored in different memory locations. This fragmentation slows down performance, similar to index fragmentation.
   To fix this, we perform 'Data Compaction', which reorganizes the file by grouping related data together. This process improves performance and can free up unused space, which the operating system can reclaim as a free disk space.

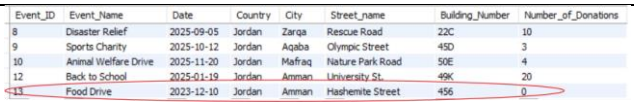
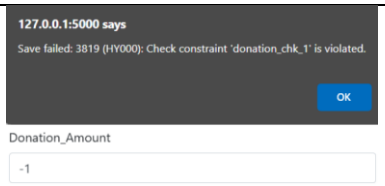
4. **Integrity Check:**
   Over time, databases subject many changes, such as adding/removing data, updating tables and schemas. These changes can start a corruption, leading to performance issues or even catastrophic data loss. To prevent this, a Database Integrity Check should be run weekly. This check analyses the database for corruption, detects issues, and repairs most problems.
   Workspace includes a built-in Integrity Check for admins to run on demand. If corruption is found and resolved, users are informed of the details. If unresolved issues remain, contacting Office Tools Technical Support is recommended for expert assistance. Regular integrity checks are essential for maintaining database health and avoiding potential disruptions.

(Anon., 2015)

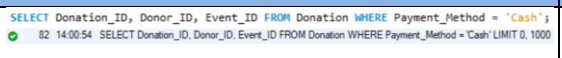
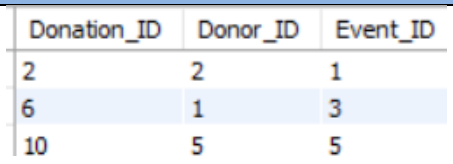
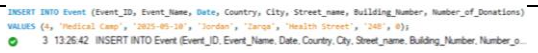
8 Testing

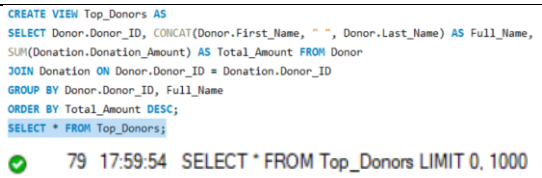
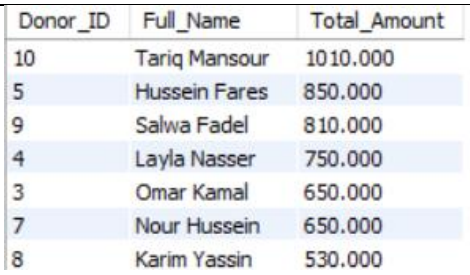
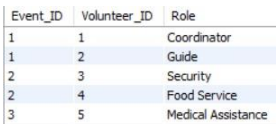
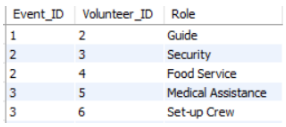
8.1 Data Validation

Number	Type	Description	screenshot
1.	All cases of PK	I tested the uniqueness in the Donor_ID PK by inserting a duplicated value, to make sure that this PK feature is working.	
		I tested the non-null in the Donor_ID PK by inserting a null ID, To make sure that this PK feature is working.	
2.	All cases of FK	I tested the first characteristic of the FK by adding a non-existent value from the PK in the Event table as the Event_ID FK in the Donation table.	
		I deleted the FK (Donor_ID) in the donation table from the original table where it is a PK there, to test the On Delete Cascade	
		I tested the non-null in the Event_ID FK by inserting a null ID, To make sure that this feature is working.	
3.	Unique	I tested the uniqueness in the Email row by inserting a duplicated email, to make sure that this constraint is working.	
4.	Default	I tested the default value (Country) in the Event table by inserting	

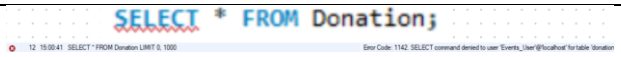
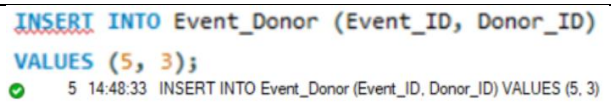
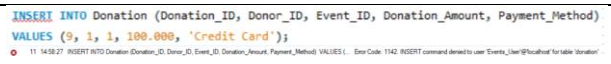
		a null value, to verify that this constraint is working	
5.	Not null	I tested the non-null in the First_Name row by inserting a null value, to make sure that this constraint is working.	
6.	Check	I tested the check constraint in the Donation_Amount by inserting a non-valid amount (based on the check) , to make sure that this constraint is working.	

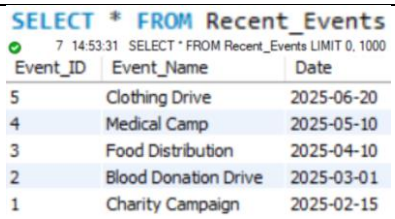
8.2 Output Validation

Number	Query Description	Screenshot (query + result)	Result validation
1.	This query filters donation records and displays only records where payments were made using "cash". This helps analyse how many donations were made in cash and who contributed in this way.		 The result is the same as expected
2.	This query adds a new event to the system. Events are the central activities in this database, and donors contribute funds to them.		 The result is the same as expected

3.	This view calculates the total amount donated by each donor across multiple events. This query helps us analyse and reward the top donors.		 The result is the same as expected
4.	This procedure deletes a volunteer's participation in an event from within the Event_Volunteer table. This query helps us remove a volunteer who has withdrawn or been fired.		Before:  After:  The result is the same as expected

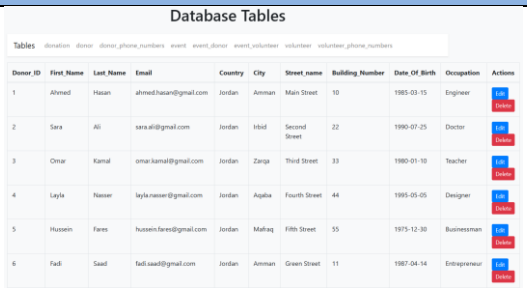
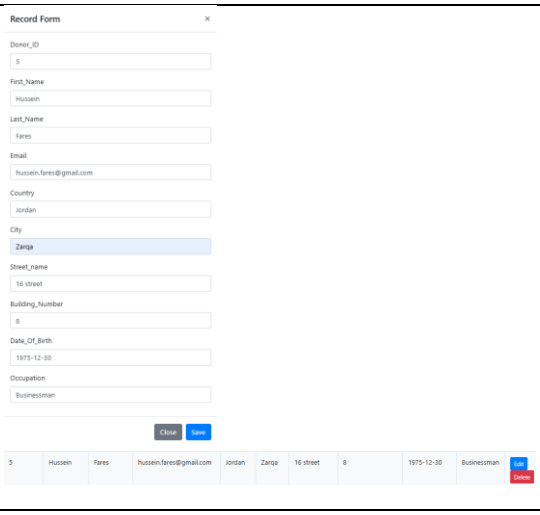
8.3 Security Validation

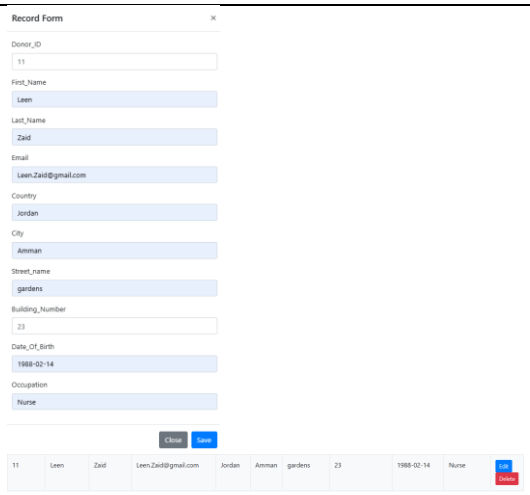
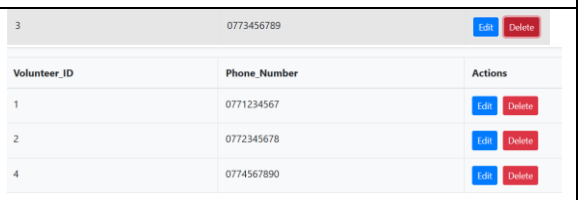
Number	Username	Description of privilege/no privilege	Screenshot (query + result)
1.	Events_User	Retrieve all records from the Event table to view events details/ granted privilege	
2.	Events_User	Retrieve all records from the Donation table to view donations details/ not granted privilege	
3.	Events_User	Assign a donor to an event by inserting a record into Event_Donor/ granted privilege	
4.	Events_User	Insert a new donation record into the Donation table /not granted privilege	

5.	Events_User	Retrieve recent event from the Recent_Events view /granted privilege	
6.	Events_User	Retrieve top donor from the Top_Donors view /not granted privilege	

8.4 GUI Validation

(I will use admin username in all cases)

Number	Description	screenshot
1.	This is the displaying data case, the admin can see all tables including the doner table shown in the screenshot.	
2.	This case is updating data, the admin can edit all tables as per his access. As shown in the screenshot, I have edited the city of Donor with the Donor_ID 5.	

3.	This case is adding a new record, the admin is allowed to add the records he wants, and as shown in the screenshot, I have added a new doner	
4.	This case is a record deletion, it allows the admin to delete any record in the tables, as shown in the screenshot, I deleted the phone number linked to Volunteer_ID 3	

8.5 Assess whether meaningful data has been extracted

This system organizes and extracts useful data about charitable events, donations, and volunteers. It uses tables to store essential information, views to simplify access to data, such as major donors and recent events, and stored procedures to automate tasks. The system efficiently tracks relationships, such as donor participation in multiple events, in order to make it easy to gain valuable insights for decision making. Let's break that step by step:

- Multi-value attributes:
  A multi-value attribute occurs when one entity (like a donor or volunteer) can have multiple values for a specific attribute. These values are stored in separate rows of a table rather than a single column. For example, in my database, a donor can have more than one phone number. Instead of storing multiple phone numbers in one column (which would not be efficient or possible in a database), we create a separate table to store each phone number for a donor, which makes it more meaningful and efficient
- 1:M Relationships:
  A 1:M relationship is when one record in a table is linked to many records from another table. But each record in the second one is linked to only one record in the first table. For example, in my database, a donor can make many donations, but each donation is associated with only one donor. That makes the data meaningful and efficient because we can track how many donations a donor has made and how much they've donated in total, without repeating donor details each time, and by keeping donations in a separate table, we can store donation details like amounts, payment methods, and events, all linked back to the donor.
- M:N relationships
  A M:N relationship is when multiple records in one table can be linked to multiple records from another table. That requires a linking table to establish these

connections. For example, in my database, a donor can donate to many events, and an event can receive donations from many donors. For example, in my database, there was a M:N relationship between Event and Donor since a donor can participate in multiple events, and each event can have multiple donors. This was resolved with the Event_Donor table which consists of Event_ID and Donor_ID, both are foreign keys and represent a part of the primary key. It enables you to track many-to-many relationships, like how each donor participates in multiple events or how each event has multiple donors, which makes the data more meaningful and effective

Views:

Views provide users with a predefined way to query and organize data from multiple tables. They encapsulate complex queries and make them easier to reuse and access. Views are especially useful when you want to grant access to specific parts of a database without exposing all of the underlying data or tables. Examples from my database:

- The Recent_Events view: it's designed to show the most recent events, it simplifies the process of searching for the latest events without having to manually run a query each time. Event managers can benefit from it by quickly see a list of recent events without manually filtering and sorting the data, and also, and anyone needing a quick overview of the latest events, like donors, or guests can use this view.
- The Top_Donors view: It identify and rank donors based on the total amount donated across all events, and by summing the donations for each donor, gives a clear picture of which donors are most generous. By using that view, Fundraising teams can identify top donors, which can help in targeting them for future events or thanking them appropriately.
- The Volunteers_In_Event view: shows the volunteers who are working in each event, to provide information about which volunteers are associated with which events. This view helps in understanding volunteer participation across events, by providing the full names of volunteers and the events they are working in, so they (the volunteers) can use the view to check which events they're assigned to or working in.
- The Average_Donation_Amount_Per_Event view: This view calculates the average donation amount for each event, to help to measure the financial success of each event in terms of donation amounts. This view aggregates the donations for each event and calculates the average donation amount, helping event organizers to see which events were financially most successful. Event organizers and fundraisers can measure the overall financial success of their events and plan future fundraising strategies accordingly.

Procedures:

- The Add_Event procedure: This procedure allows users to add a new event to the Event table by providing necessary detail, such as a event name, date, location, organizer name. Users can easily add events to the database with a single call, making event creation faster and more streamlined. The procedure encapsulates the event entry logic, ensuring data entry consistency. Event managers and organizers can benefit from this by adding new events without directly interacting with the database, reducing human error and speeding up the process.

- The Events_By_Donor procedure: It retrieves all events that a particular donor has contributed to, along with the donation amount and payment method. It provides an easy way to search for all donations made by a particular donor, helping organizations keep track of which events the donor supported and which payment method they used. Donors and support teams can use this procedure to track the contributions of any specific donors and maintain relationships or send targeted communication based on past donations.
- The Remove_Volunteer procedure: Removes a volunteer from an event, effectively cancelling their registration at the event. This allows administrators to remove a volunteer from an event without having to manually delete records from the Event_Volunteer table, That ensures the database consistency and process efficiency. Event coordinators can use this procedure to manage volunteers easily, especially when the adjustments process need to be made to schedules or staffing.
- The Donations_By_Donor procedure: This procedure retrieves us all the donations was made by a specific donor, including the event details and payment method.

In conclusion, I believe I have designed a strong database with views and stored procedures that provides a comprehensive, efficient, and easy-to-use way to manage and interact with data related to events, donors, and volunteers.

8.6 Assess the effectiveness of testing

The testing process is a very important part of developing and maintaining a database system, and it plays a huge role in ensuring the reliability, accuracy, and efficiency of our database. Firstly, it validate the data relationships and integrity constraints, it ensures that referential integrity is preserved, which means that relationships between tables are valid, and foreign keys are not violated. It also optimizes the system's performance identify and optimize any bottleneck, to ensure that the system can handle large datasets effectively.

Steps I followed to test the database system:

- Ensured that the database is user-friendly and intuitive for both technical and non-technical users.
- Confirmed that each part of the system works as intended in isolation, like insuring that all views, and procedures are working as expected
- Confirmed that the database components work together seamlessly, like ensuring that the foreign key relationships between tables work properly
- Confirmed that the system is secure, and user roles are properly enforced, all user roles and permissions are correctly assigned, and no one can get any unauthorized access in the system.

Testing helps to detect and prevent any errors and bugs, especially when we are interacting with large amounts of data or complex operations, in my experience, during the testing process I was able to identify logical errors in stored procedures and views. For example, when I tested the Top_Donors view, I noticed that the total donations were not being correctly calculated for some donors due to a missing group by statement in the SQL query, in that case, testing was the most important step to discover that issue.

9 Evaluation of database solution

9.1 Effectiveness of the database solution based on user and system requirement

The purpose of designing this system was to build a database to donate for the EDU Youth Foundation through non-profit charitable events organized by the Foundation, my final results effectively fulfills its intended purpose of managing, tracking, and organizing donations for the events. In order to understand the effectiveness of the system, let's analyze it in detail:

- The tracking system accurately links donors and volunteers contributions with the events, which makes the system much more flexible. It ensures that every donation made by a donor is associated with the correct event, and every volunteer's role is well cleared, all of that allows the event organizers to track every participation efficiently.
- The system clears the relationship management of donors and volunteers with the events, that was after creating the intermediate tables (Event_Donor, Event_Volunteer)
- The design includes essential user security features, ensuring that only authorized individuals can access or modify certain parts of the data, with clearly defined roles of every user in the system.
- It includes having a backup policy in place to take precautions in case of any natural, internal or external disaster.

A comparison of the initial requirements and what I successfully achieved:

1. Donations Managing and Tracking:
    - Requirement: Donors are allowed to participate in any much charity events as they want and track each donation.
    - Status: Achieved
    - Achievement: We have linked each donation to both donor and event, through the donation table, this ensures us easy tracking of financial records.
2. Events Management:
    - Requirement: Scheduling events, making plans, determining the goals for each event and estimated number of volunteers and donors desired. I also need to create statistics for the event to see last events and evaluate every event success.
    - Status: Achieved
    - Achievement: The Event table stores all event details, including Event name, date, location, and the number of donations during it. Recent_Events, and Volunteers_In_Event views done to see last events and evaluate every event success.
3. Handling Donor and Volunteer Information:
    - Requirement: Donors and volunteers details must be stored, to ensure that multiple donors and volunteers can participate in different events, conducting statistics on donors and volunteers for analysis.
    - Status: Achieved
    - Achievement: Donor and Volunteer tables were created to store their personal details such as names, contact information, and addresses. Top_Donors,

Volunteers_In_Event tables can provide us detailed information about the contributions, helping us with the recognitions and analysis.

4. Contact Donors and Volunteers:
    - Requirement: Maintain contact information for donors and volunteers to enable the event organizer to communicate To stay in touch with them for event details, any updates.
    - Status: Achieved
    - Achievement: Emails are stored in the Donor and Volunteer tables. For phone numbers, they are stored in the Donor_Phone_Numbers and Volunteer_Phone_Numbers tables, linked to each's donor and volunteer

Unmet requirements:

1. Volunteer Availability Management:
    - Requirement: Track every volunteer's availability for scheduling purposes, ensuring that the event are enough staffed.
    - Status: Not Achieved
    - Explanation: My database currently stores volunteer details but does not capture their availability for even a participation. This would require additional space in the Volunteer table on even a new table.
2. Donation Payment Verification:
    - Requirement: Integrate a feature to verify and confirm donation payments
    - Status: Not Achieved
    - Explanation:

9.2 Suggested improvements

1. **Budgeting and sponsor management:**
   The system doesn't allow event organizers to set or track budgets. There's no way to compare estimated costs with actual expenses and do financial reporting, In addition, The system doesn't track sponsors or their contributions. This makes it difficult to know which sponsors support events and how much they contribute, so a budgeting feature can be added by creating a table to store estimated budgets and actual expenses for each event, including what sponsors pay.

2. **Focus on the donor's social status:**
   In 2018, a group of sociologists conducted an experiment on married couples and found that most couples in the sample made decisions to donate small amounts separately and large amounts together, while this does not mean that married people always donate more, married people are more likely to donate a larger amount than non-married people, especially when they donate together. I lacked focus on these details in designing the database, so I can take the social status of the donor into account to focus more on married donors.
   (Einolf, 2019)

3. **Geographic Donation Analysis:**

My design doesn't have any details about geographic data, this makes it difficult to analyse donation trends by location or identify high-donation areas.. I can include detailed information like postal codes or regions, for donors.

9.3 Evaluation based on improvements needed

Benefit

1. Budgeting and sponsor management:
    - Allows event organizers to track estimated costs, actual expenses, and financial outcomes
    - Enables organizers to assess the financial impact of sponsors and build stronger relationships with them.
    - Increases sponsorship opportunities, it makes it easier to identify and approach to potential sponsors.
2. Focus on the donor's social status:
    - Organizers can design campaigns that encourage joint donations, which are likely to be larger.
    - Helps focus efforts on demographics more likely to donate larger amounts.
    - Builds trust and strengthens relationships.
3. Geographic Donation Analysis:
    - Enables analysis of donation repeating patterns by location, identifying high-donation areas.
    - Campaigns can be adjusted to target those regions more effectively.
    - Allowing organizations to expand into new locations with higher donors potential.

Potential Challenges and Trade-offs:

- Increased development time, designing and implementing new tables and attributes and linking them with events will require additional time for coding, testing, and validation.
- Additional requirements, storing more detailed information increases database size, requiring more requirements, including user, system, and data requirements.
- Data accuracy and maintenance, maintaining geographic data, budgets, and sponsor information requires high data entry and validation accuracy.
- Potential performance issues, Adding tables or complex queries and views may increase query execution time.

10 User Documentation

10.1 System Overview

My database system includes event data management and everything related to it from donors, volunteers and donations. Tables are used to store structured data, such as event details, donor information, and volunteer participation. Views are used to simplify complex

queries, allowing quick access to summary information such as major donors or average donation amounts for each event. Additionally, stored procedures automate repetitive tasks, ensuring data consistency and reducing errors.

Key features and capabilities in the system:

- Donation tracking: The system helps track donations made by donors to different charity events. Donors can see how much they've donated and to which events.
- Event management: The system helps organizers to schedule events, evaluate how successful past events have been by seeing how many volunteers and donors was involved and the total donation amount.
- Users roles and permissions: Each user can access different parts of the system depending on their role, ensuring security and privacy.

Types of data stored and managed in the system:

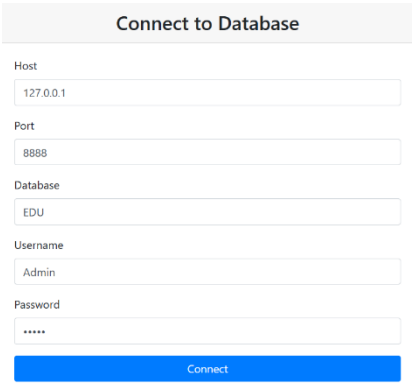
- Event Information: Including event name, date, location, and the organizer name.
- Donor Information: Such as his/her name and contact info, events he donated in (by the Event_Donor table), how much did he/she donate.
- Volunteer Information: Such as his/her name, contact info, event he/she helped with (by the Event_Volunteer table), and his/her role in them.
- Donation Information: Such as the donation amount, payment method

Basic architecture and components of the system:

- Graphical user interface (GUI): The system is easy to use and allows every potential user only to see the data he need to, for doing his work.
- Backend database: All data stored are safely kept and organized.
- Access and security: The system makes sure that no unauthorized access can occur

10.2 Using the system

The login page is unified among all users and appears to them in this format:



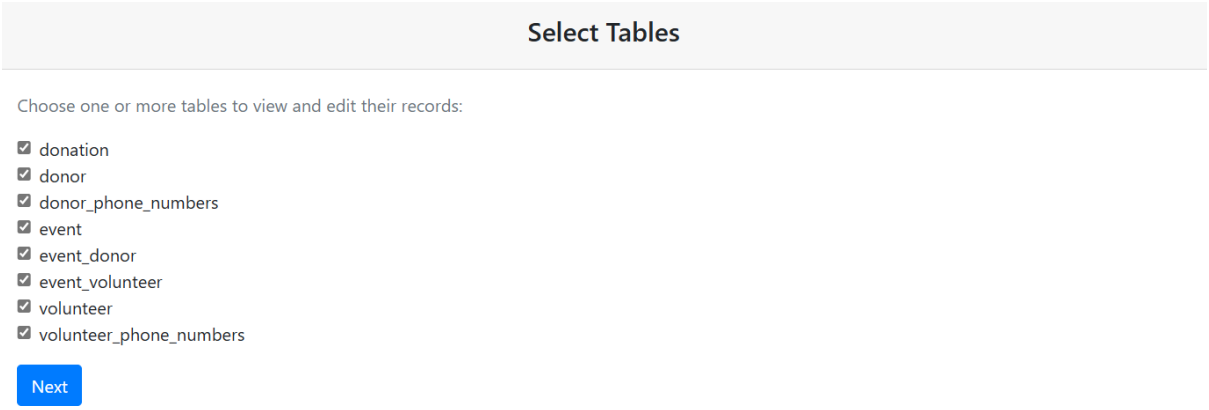
The user should insert the database he need to access, his username, and the password

The GUI for each user:

(Due to GUI tool limitations, I added screenshots for only the Admin user)

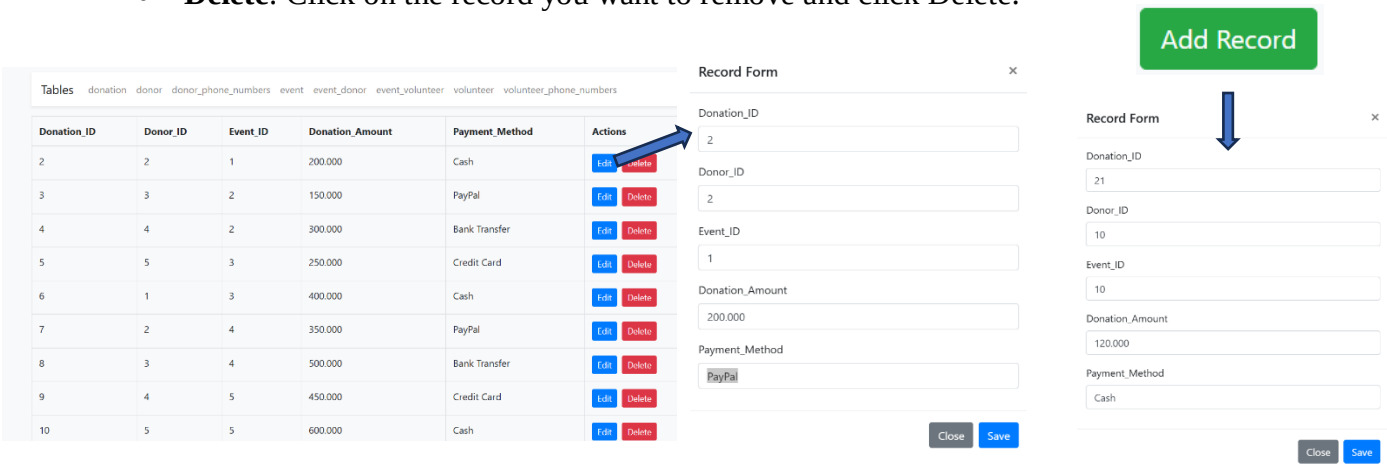
1. **Admin:**

This is the homepage where he has to choose the tables he wants to access, the tables that appear here are the ones that he is at least allowed to select (view), the admin has access to all the tables, so, they all appear on the page.



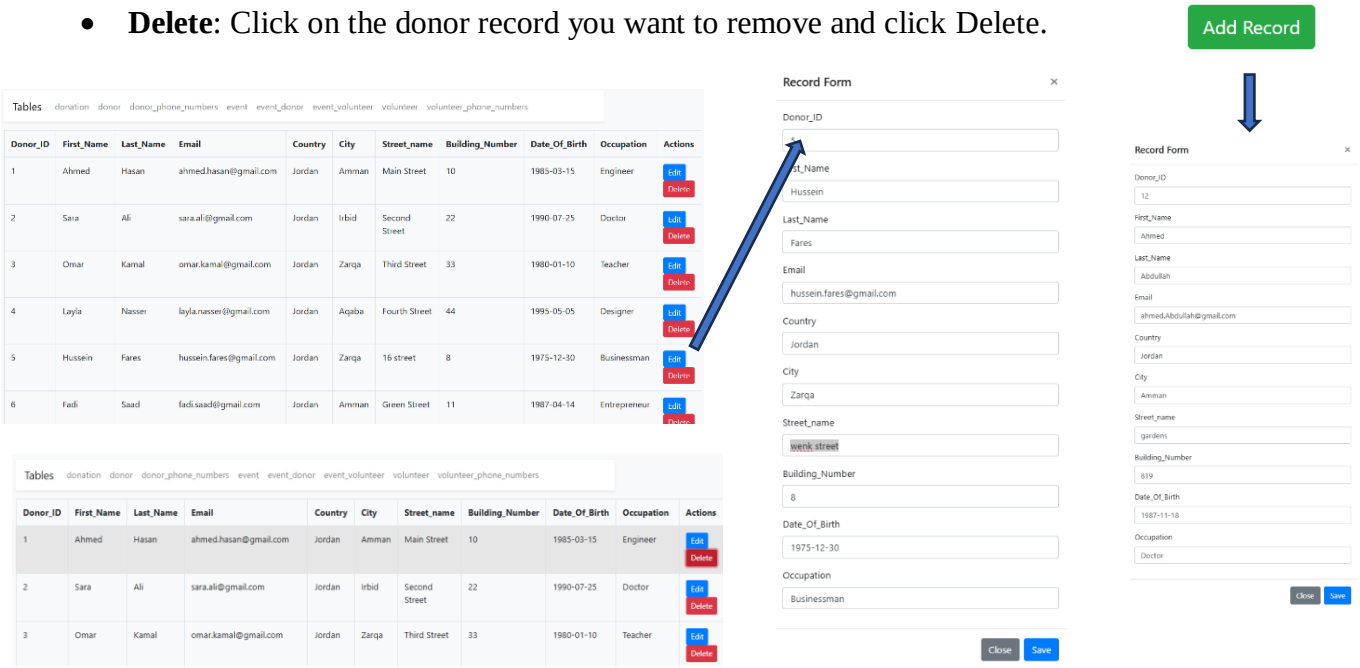
The admin can select, Insert, edit, and delete the record he wants in the donation table:

- **Select**: Click on the Donation table to view donation records.
- **Insert**: Click on Add Record to add a new donation record.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.



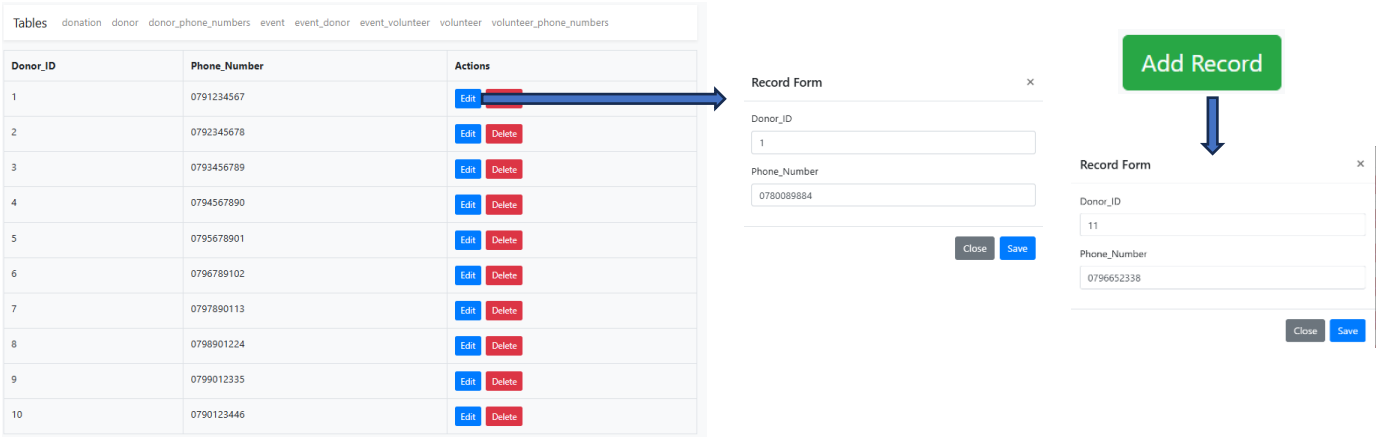
The admin can select, Insert, edit, and delete the record he wants in the donor table:

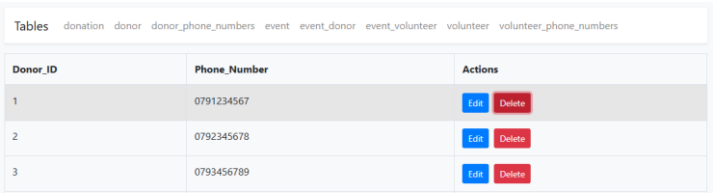
- **Select**: Click on the Donor table to view donor records.
- **Insert**: Click on Add Record to add a new donor.
- **Edit**: Click on Edit in an existing donor record, make changes, and click Save.
- **Delete**: Click on the donor record you want to remove and click Delete.



The admin can select, Insert, edit, and delete the record he wants in the Donor_Phone_Numbers table:

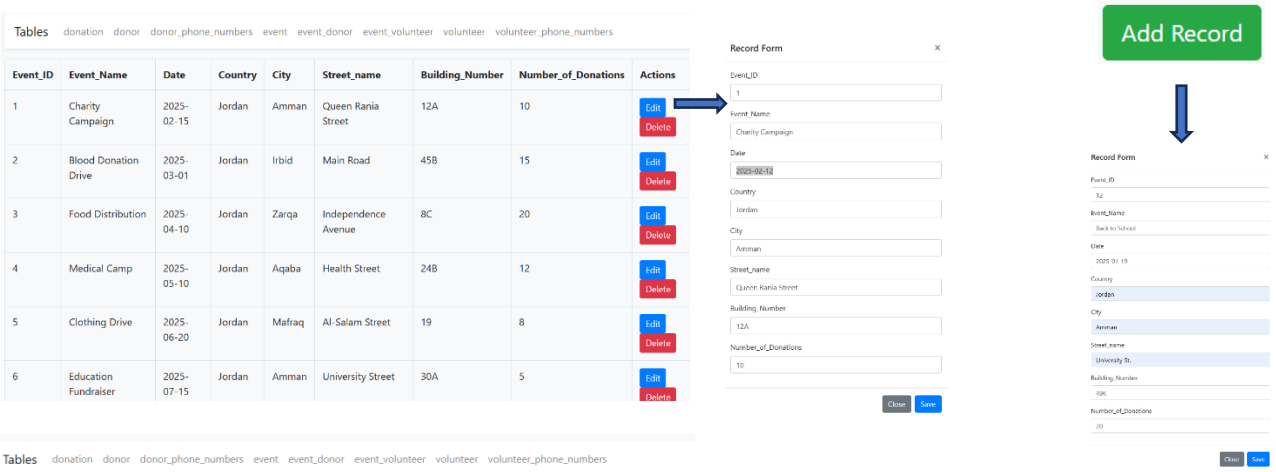
- **Select**: Click on the Donor_Phone_Numbers table to view phone numbers associated with donors.
- **Insert**: Click on Add Record to add a new phone number for a donor.
- **Edit**: Click on "Edit" in an existing phone number record, make changes, and click Save.
- **Delete**: Click on the phone number record you want to remove and click "Delete".





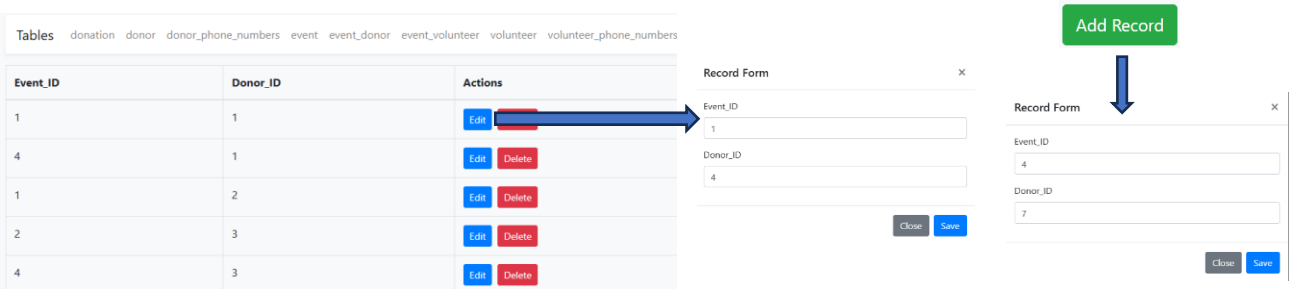
The admin can select, Insert, edit, and delete the record he wants in the Event table:

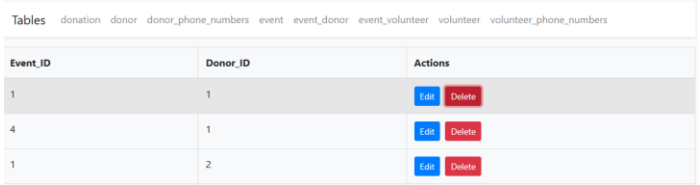
- **Select**: Click on the Event table to view event records.
- **Insert**: Click on Add Record to add a new event.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.



The admin can select, Insert, edit, and delete the record he wants in the Event_Donor table:

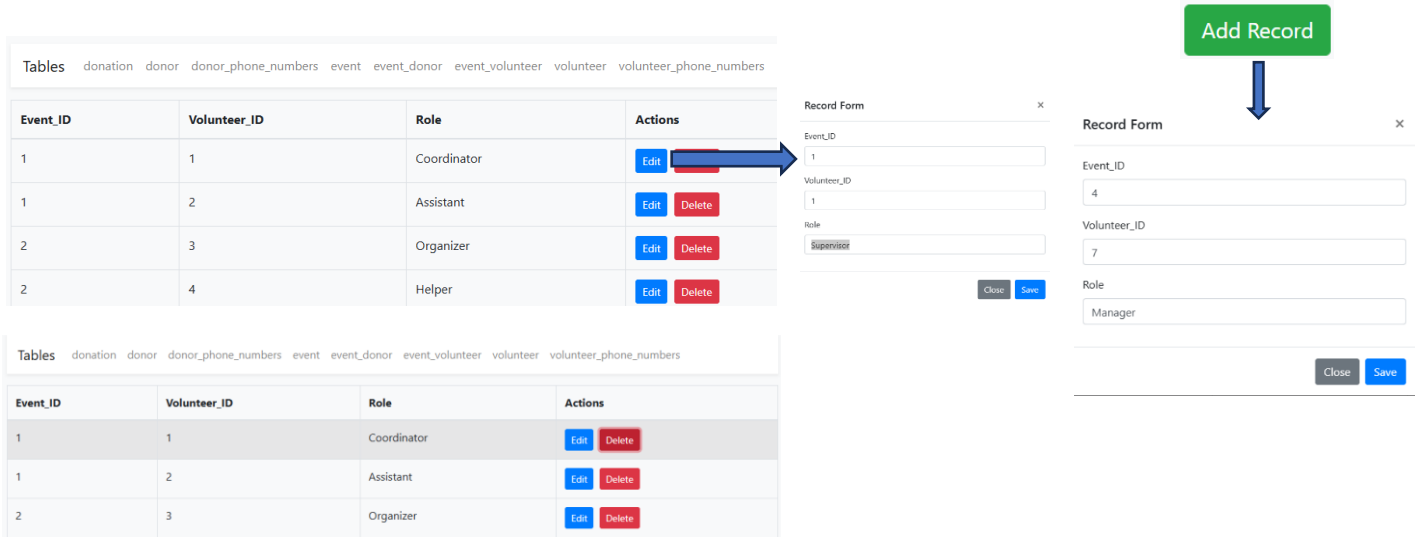
- **Select**: Click on the Event_Donor table to view event-donor records.
- **Insert**: Click on Add Record to add a new event-donor link.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.





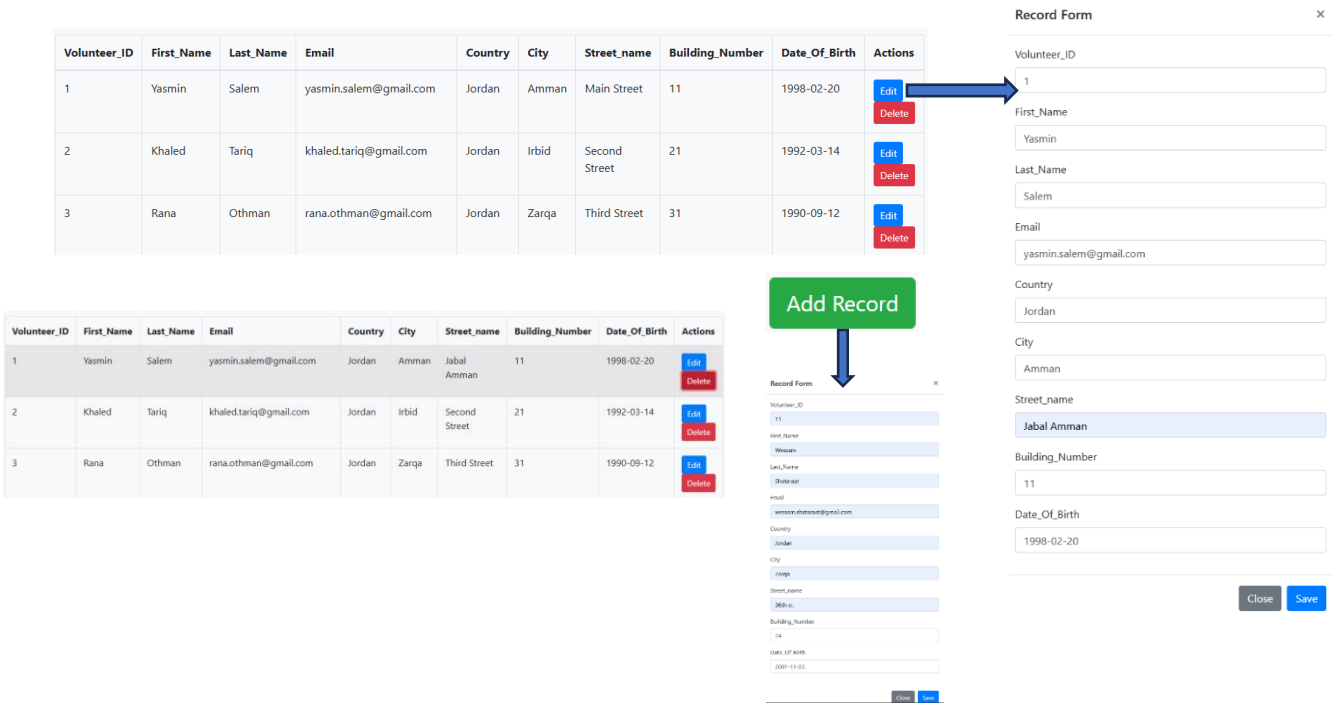
The admin can select, Insert, edit, and delete the record he wants in the Event_Volunteer table:

- **Select**: Click on the Event_Volunteer table to view event-volunteer records.
- **Insert**: Click on Add Record to add a new event-volunteer link.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.



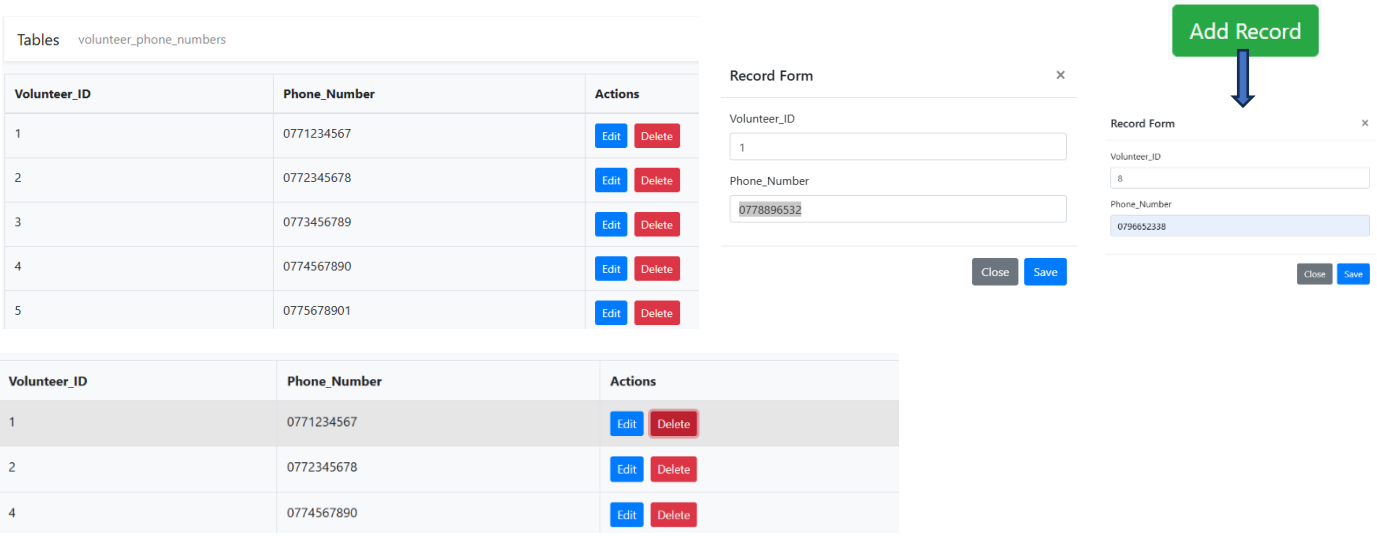
The admin can select, Insert, edit, and delete the record he wants in the Volunteer table:

- **Select**: Click on the Volunteer table to view volunteer records.
- **Insert**: Click on Add Record to add a new volunteer record.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.



The admin can select, Insert, edit, and delete the record he wants in the Volunteer_Phone_Number table:

- **Select**: Click on the Volunteer_Phone_Numbers table to view volunteer phone numbers.
- **Insert**: Click on Add Record to add a new phone number for a volunteer.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.



2. Events_User:

The homepage is where he has to choose the tables he wants to access from the tables that he is at least allowed to select (view).

The events user can select, insert, update, and delete the record he wants in the Event table.

- **Select**: Click on the Event table to view event records.
- **Insert**: Click on Add Record to add a new event.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.

The events user can select, insert, update, and delete the Event_ID attribute in the Event_Donor table.

- **Select**: Click on the Event_Donor table to view event-donor relationships.
- **Insert**: Click on Add Record to link a donor to an event.
- **Edit**: Click on edit in an existing event-donor link, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.

The events user can select, insert, update, and delete the Event_ID attribute in the Event_Volunteer table.

- **Select**: Click on the Event_Volunteer table to view event-volunteer relationships.
- **Insert**: Click on Add Record to link a volunteer to an event.
- **Edit**: Click on edit in an existing event-volunteer link, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.

3. Donations_User:

The donations user can select, insert, update, and delete the record he wants in the Donation table.

- **Select**: Click on the Donation table to view donation records.
- **Insert**: Click on Add Record to add a new donation record.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.

The donations user can select, insert, update, and delete the Number_of_Donations attribute in the Donation table.

- **Select**: View the Number_of_Donations attribute in the Donation table.
- **Insert**: Add or update the Number_of_Donations for an event.
- **Edit**: Modify the Number_of_Donations for an event.
- **Delete**: Remove donation records if necessary.

4. Donors_User:

The donors user can select, insert, update, and delete the record he wants in the Donor table.

- **Select:** Click on the Donor table to view donor records.
- **Insert:** Click on Add Record to add a new donor record.
- **Edit: C**lick on edit in an existing record, make changes, and click on Save.
- **Delete:** Click on the record you want to remove and click Delete.

The donors user can select, insert, update, and delete the record he wants in the Donor_Phone_Numbers table.

- **Select**: Click on the Donor_Phone_Numbers table to view phone numbers associated with donors.
- **Insert**: Click on Add Record to add a new phone number for a donor.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.

The donations user can select, update, and delete the Donor_ID attribute in the Event_Donor table.

- **Insert**: Click on Add Record to add ID.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.

5. Volunteers_User:

The volunteers user can select, insert, update, and delete the record he wants in the Volunteer table.

- **Select**: Click on the Volunteer table to view volunteer records.
- **Insert**: Click on Add Record to add a new volunteer record.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.

The volunteers user can select, insert, update, and delete the record he wants in the Volunteer_Phone_Numbers table.

- **Select**: Click on the Volunteer_Phone_Numbers table to view volunteer phone numbers.
- **Insert**: Click on Add Record to add a new phone number for a volunteer.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.

The volunteers user can select, update, and delete the Volunteer_ID attribute in the Event_Volunteer table.

- **Insert**: Click on Add Record to add ID.
- **Edit**: Click on edit in an existing record, make changes, and click on Save.
- **Delete**: Click on the record you want to remove and click Delete.

10.3 Frequently asked questions

Why can't I access to all tables (as a user)?
Because the database design allows each user to have limited access to perform his job functions only.

Why does my information is not showing up in reports or views?
This issue can happen if you information is not entered correctly or if there are missing connections between different pieces of information.

How can I ensure that there are no duplicate email records?
Each donor and volunteer has a unique email by adding a UNIQUE constraint to the email column in MySQL, so, it's almost impossible to have to records with the same email.

Why is my donation not showing up in the system?
This could be because the donation was not properly recorded or linked to the correct event or donor.

10.4 Contact information

Osama Hasan
+971567551458
osamoh2006@outlook.com

Office Tools, (2025). Database Maintenance Explained. [online] Available at: https://officetools.com/knowledgebase/database-maintenance-explained/

Sage Journals, (2018). Understanding the impact of volunteering and pro-social attitudes. [online] Available at: https://doi.org/10.1177/0899764018757027