CS-A1155 DATABASE FOR DATA SCIENCE

PROJECT PART 1

Group 10

Tran Dang Manh Hoang

La Thuy Linh

Ta Hau Dang Tinh

To Huy

TABLE OF CONTENT

1. Introductions:

1.1. Project topic:

This report regarding the construction of a Volunteer Matching System (VMS) for the Finnish Red Cross (FRC). It aims to develop a database and define its usage to support the matching of Red Cross Volunteer Capacity (supply) with "Local Multidimensional Vulnerabilities and Crises" (demand).

1.2. Tools:

In this project, we use these definitions and tools:

Unified Modeling Language (UML) is a standardized visual modeling language used in software engineering to design, specify, visualize, and document software systems. UML provides a set of graphical notations that allow software developers to represent various aspects of a system, including its structure, behavior, and interactions.

Relational database is a type of database management system (DBMS) that organizes and stores data in a tabular form, consisting of rows and columns. In a relational database, data is organized into tables, where each table represents an entity or a relationship between entities. Each row in a table represents a specific record or instance of the entity, and each column represents a specific attribute or property of the entity. The intersection of a row and a column is called a cell, which holds the actual data value.

1.3. Purpose:

In this project, an UML diagram is created to visualize the data organized for the VMS and converted into relational database schemas. Furthermore, the functional dependencies of these relations are established, scrutinized, and modified to fit Boyce–Codd Normal Form (BCNF).

2. Assumptions

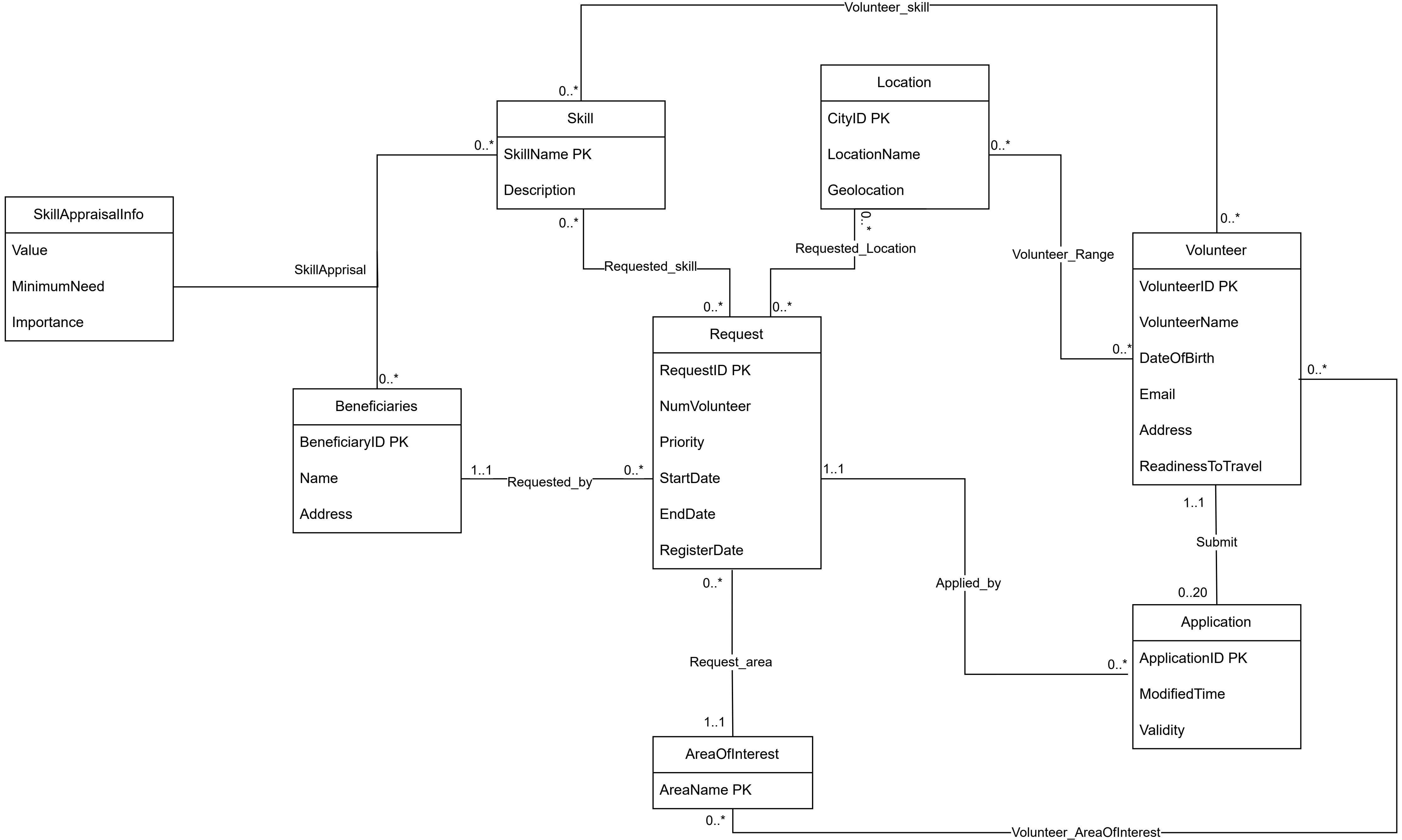
The UML is created based on these sets of assumptions:

– Skills are universal and the Skill relation can be used to describe both the Volunteer possessed skills and Beneficiaries required skills.

- Location is universal, so Location relation can be used to describe the Volunteer operation ranges and the Requested location of need.
- AreaOfInterst is universal so the relation can be used to describe both the Requested Area of needs and Volunteer areas of interest.
- One RequestID only concerns one AreaOfInterest.
- In this project, the term **location** and **city** are considered similar.
- In this project, a city's **geolocation** is considered just a text. In real life, it can be the postal code, longitude, and latitude of a location.
- Each Request can contain multiple Skills.

3. Designing Unified Modeling Language for the project

The image below illustrates the UML we build for this project



Volunteer_skill
0..*
0..*
Skill
SkillName PK
Description
Location
CityID PK
LocationName
Geolocation
0..*
0..*
SkillAppraisalInfo
Value
MinimumNeed
Importance
SkillApprisal
0..*
Requested_skill
0..*
Requested_Location
0..*
0..*
Volunteer_Range
0..*
Volunteer
VolunteerID PK
VolunteerName
DateOfBirth
Email
Address
ReadinessToTravel
0..*
0..*
Request
RequestID PK
NumVolunteer
Priority
StartDate
EndDate
RegisterDate
Beneficiaries
BeneficiaryID PK
Name
Address
1..1
Requested_by
0..*
1..1
Applied_by
1..1
Submit
0..20
0..*
Request_area
1..1
0..*
Application
ApplicationID PK
ModifiedTime
Validity
AreaOfInterest
AreaName PK
0..*
Volunteer_AreaOfInterest

4. **How does this UML fulfill the requirements**:

In this section, we will present how each class and association fulfill the requirements of the project.

Requirement	UML class representation
Beneficiaries have unique IDs, names, and addresses	Main class: Beneficiaries Attributes: BenficiaryID (PK), Name, Address
Any Beneficiary can make as many volunteering requests as they need	Association: Requested_by between Beneficiaries and Request Type of association: one-to-many
Requests should include a unique ID, the ID of the beneficiary it was sent by, the number of volunteers needed, a priority value to indicate how urgent the request is, a start date, and end date, and a register by date	Main class: Request Attributes: RequestID (PK), NumVolunteer, Priority, StartDate, EndDate, RegisterDate
Requests also have the area of interest where the request lies	Association: Request_area between Request and AreaOfInterest Type of association: one-to-many
Each of these skills have a unique name and a description	Main class: Skill Attributes: SkillName (PK), Description
Each request skill has a request ID and a skill name	Asosciation: Requested_skill Connect: between Request and Skill Type of association: many-to-many
Each request location has a request ID and a city ID	Association: Request_Location Connect: between Request and Location Type of association: many-to-many
Volunteers have unique ID, name, birthday, email, address, readiness to travel (minutes)	Main class: Volunteer Attribute: VolunteerID (PK), VolunteerName, DateOfBirth, Email, Address, ReadinessToTravel)
A volunteer can choose any combination of areas of interests	Association: Volunteer_AreaOfInterst Connect: between Volunteer and AreaOfInterst Type of association: many-to-many
A volunteer can choose any combination of skills	Association: Volunteer_Skill Connect: between Volunteer and Skill Type of association: many-to-many

Beneficiaries can appraise skills by assigning skills a value, a minimum need, and a value to indicate importance	Association class: SkillAppraisalInfo Attribute: Value, MinimumNeed, Importance Connect between Beneficiaries and Skill.
Applications should include a unique ID, the ID of the request it was made to, the time it was modified, and they should indicate whether they are valid or not	Main class: Application Attribute: ApplicationID, ModifiedTime, Validity
Applications should include the ID of the request it was made to	Association class: Applied_by Connect between Application and Request Type of association: one-to-many
Applications the ID of the request it was made to, the ID of the volunteer it was sent by	Association class: Summit Connect between Application and Volunteer Type of association: one-to-many
Volunteers can sign up to the system, browse through the volunteering requests, and send up to 20 applications where they apply to the requests	Assocation: Submit Connect: between Volunteer and Application Type of association: one-to-many
Each city has an ID, a name, and a geolocation	Main class: Location Attribute: CityID (PK), Name, Geolocation)
Volunteers operate in ranges. Each volunteer range has a volunteer ID and a city ID	Assocation: Volunteer Range Connect: between Volunteer and Location Type of association: many-to-many

Table 1: Explanation how UML fulfill the requirement

5. **Relational model conversion**

5.1. UML to relational model conversion algorithms

Converting UML into Database Schema should be done through the utilizing the following principles:

- Each class gets its own relation, which has the same attributes as the class.

- For each many-many associations, a relation whose attributes are the keys from both the relations the association connects, and the attributes of the association class, if there is one.
- For each many-one association, create a relation in the same manner as with many-many associations or add attributes to the relation that are on the many-side of the association.
- Rename the attributes in the relations, if needed, to make them clear and distinct.
- Consider whether a many-one association should have its own relation, or should the relation be replaced by simply adding the key from the one-side class to the attribute list of the many-side class.
- If an association is a many-many associations, it cannot be combined.

5.2. Relational model conversion

Note: In the relation listed below, the Underscore denotes primary key(s)

Based on the above-mentioned principles, the following main and associations can be listed out as:

- Main relations:

  Request (RequestID, NumVolunteer, Priority, StartDate, EndDate, RegisterDate)

  Location (CityID, LocationName, Geolocation)

  Skill (SkillName, Desciption)

  Beneficiaries (BeneficiaryID, Name, Address)

  Volunteer (VolunteerID, VolunteerName, DateOfBirth, Email, Address, ReadinessToTravel)

  Application (ApplicationID, ModifiedTime, Validity)

  AreaOfInterest (AreaName)

- Association relations:

  SkillApprisal (SkillName, BeneficiaryID, Value, MinimumNeed, Importance)

  Request_by (RequestID, BeneficiaryID)

  Requested_skill (RequestID, SkillName)

  Requested_location (RequestID, CityID)

Requested_area (<u>RequestID</u>, AreaName)

Applied_by (RequestID, <u>ApplicationID</u>)

Submit (<u>ApplicationID</u>, Volunteer)

Volunteer_skill (<u>VolunteerID</u>, <u>SkillName</u>)

Volunteer_Range (<u>VolunteerID</u>, <u>CityID</u>)

Volunteer_AreaOfInterest (<u>VolunteerID</u>, <u>AreaName</u>)

Based on the UML, these many-many relationships are kept as separate relations:

- SkillAppraisal
- Volunteer_skill
- Requested_skill
- Requested_Location
- Volunteer_Range
- Volunteer_AreaOfInterest.

For the one-many relationships we can combine these relations:

- BeneficiaryID from Beneficiaries table can be combined into Request table because one RequestID only corresponds to one BeneficiaryID and remove Requested_by relation.
- RequestID from Request table can be combined into Application table because one ApplicationID only corresponds to one RequestID and remove Applied_by relation.
- VolunteerID from Volunteer can be combined into Application table because on ApplicationID only corresponds to one VolunteerID and remove Submit relation.
- AreaName from AreaOfInterest table can be combined into Request table because one RequestID only corresponds to one AreaName of interest and remove Request_area relation.

Therefore, the final relation model can be defined:

Request (<u>RequestID</u>, BeneficiaryID, NumVolunteer, Priority, AreaName, StartDate, EndDate, RegisterDate)

➔ {RequestID -> BeneficiaryID, NumVolunteer, Priority, AreaName, StartDate, EndDate, RegisterDate}

Requested_location (RequestID, CityID)

➔ {RequestID, CityID -> RequestID, CityID}

Requested_Skill (RequestID, SkillName)

➔ {RequestID, SkillName -> RequestID, SkillName}

Skill (SkillName, Description)

➔ {SkillName -> Description}

Location (CityID, LocationName, Geolocation)

➔ {CityID -> LocationName, Geolocation}

AreaOfInterest (AreaName)

➔ {AreaName -> AreaName}

Beneficiaries (BeneficiaryID, Name, Address)

➔ {BeneficiaryID -> Name, Address}

SkillAppraisal (SkillName, BeneficiaryID, Value, MinimumNeed, Importance)

➔ {SkillName, BeneficiaryID -> Value, MinimumNeed, Importance}

Volunteer (VolunteerID, VolunteerName, DateOfBirth, Email, Address, ReadinessToTravel)

➔ {VolunteerID -> VolunteerName, DateOfBirth, Email, Address, ReadinessToTravel}

Volunteer_Range (VolunteerID, CityID)

➔ {VolunteerID, CityID -> VolunteerID, CityID}

Volunteer_AreaOfInterest (VolunteerID, AreaName)

➔ {VolunteerID, AreaName -> VolunteerID, AreaName}

Volunteer_Skill (VolunteerID, SkillName)

➔ {VolunteerID, SkillName -> VolunteerID, SkillName}

Application (ApplicationID, RequestID, VolunteerID, ModifiedTime, Validity)

➔ {ApplicationID -> ApplicationID, RequestID, VolunteerID, ModifiedTime, Validity}

6. Non-trivial functional dependencies

6.1. Non-trivial functional dependencies definition

If the left-hand side (the determinant set) is a subset or is not contained on among those on the right-hand side, the depdency is (completely) nontrivial

6.2. Retional model non-trivial functional dependencies

Based on the provided definition, the list of non-trivial functional dependencies in this schema are:

Request (<u>RequestID</u>, BeneficiaryID, NumVolunteer, Priority, AreaName, StartDate, EndDate, RegisterDate)

➔ {RequestID -> BeneficiaryID, NumVolunteer, Priority, AreaName, StartDate, EndDate, RegisterDate}

Skill (<u>SkillName</u>, Description)

➔ {SkillName -> Description}

Location (<u>CityID</u>, LocationName, Geolocation)

➔ {CityID -> LocationName, Geolocation}

AreaOfInterest (<u>AreaName</u>)

➔ {AreaName -> AreaName}

Beneficiaries (<u>BeneficiaryID</u>, Name, Address)

➔ {BeneficiaryID -> Name, Address}

SkillAppraisal (<u>SkillName</u>, <u>BeneficiaryID</u>, Value, MinimumNeed, Importance)

➔ {SkillName, BeneficiaryID -> Value, MinimumNeed, Importance}

Volunteer (<u>VolunteerID</u>, VolunteerName, DateOfBirth, Email, Address, ReadinessToTravel)

➔ {VolunteerID -> VolunteerName, DateOfBirth, Email, Address, ReadinessToTravel}

Application (<u>ApplicationID</u>, RequestID, VolunteerID, ModifiedTime, Validity)

➔ {ApplicationID -> ApplicationID, RequestID, VolunteerID, ModifiedTime, Validity}

7. Boyce–Codd normal form (BCNF)

7.1. BCNF definition

A relation, R is said to be in BCNF if and only if: For any non-trivial when FD A1, A2 , ..., An → B1 , B2 , ...,Bm for R, the attribute set A1, A2, ..., An is a superkey for R. In other words, the closure of the left hand side of any non-trivial FD contains all the attributes. A relation on BCNF does not exhibit these anomalies:

- Redundancy: Repeated information over several tuples in a table (not over several tables). Redundancy in a database my result in anomalies
- Update Anomaly: Sensitivity to mistake in updating repeated information
- Deletion Anomaly: If a tuple needs to be deleted, information might be lost.

7.2. BCNF of the database

Based on the definition of BCNF, the following relations are in BCNF:

- Request (RequestID, BeneficiaryID, NumVolunteer, Priority, AreaName, StartDate, EndDate, RegisterDate). If we know RequestID, we can know all BeneficiaryID, NumVolunteer, Priority, AreaName, StartDate, EndDate, RegisterDate. That means, we have {RequestID}$^+$ = {RequestID, BeneficiaryID, NumVolunteer, Priority, AreaName, StartDate, EndDate, RegisterDate}. So this relation is in BCNF.

- Skill (SkillName, Description). If we know SkillName, we know Description. That means {SkillName}$^+$ = {SkillName, Description}. So this relation is in BCNF.

- Location (CityID, LocationName, Geolocation). If we know CityID, we know all LocationName, Geolocation. That means: {CityID}$^+$ = {CityID,LocationName, Geolocation}. So this relation is in BCNF.

- AreaOfInterest (AreaName). We have {AreaName}$^+$ = {AreaName}. So this relation is in BCNF.

- Beneficiaries (BeneficiaryID, Name, Address). If we know BeneficiaryID, we also know {Name, Address}, which means {BeneficiaryID}$^+$ = {BeneficiaryID, Name, Address}. So this relation is in BCNF.

- SkillAppraisal (SkillName, BeneficiaryID, Value, MinimumNeed, Importance). If we know SkillName and BeneficiaryID, we will also know BeneficiaryID, Value, MinimumNeed, Importance. So, {SkillName, BeneficiaryID}$^+$ = {SkillName, BeneficiaryID, Value, MinimumNeed, Importance}. So this relation is in BCNF.

- Volunteer (VolunteerID, VolunteerName, DateOfBirth, Email, Address, ReadinessToTravel). If we know VolunteerID, we will also know VolunteerName, DateOfBirth, Email, Address, ReadinessToTravel. That means, {VolunteerID}$^+$ = {DateOfBirth, Email, Address, ReadinessToTravel}. So this relation is in BCNF.

- Application (ApplicationID, RequestID, VolunteerID, ModifiedTime, Validity). If we know ApplicationID, we will also know RequestID, VolunteerID, ModifiedTime, Validity. That means, {ApplicationID}$^+$ = {ApplicationID, RequestID, VolunteerID, ModifiedTime, Validity}. So this relation is in BCNF.

Other relations exhibit trivial dependencies, however, since they are in the form of AB -> AB, they cannot be divided further.