CSCI 3901 - Project

External Documentation

Name: Benny Daniel Tharigopala

Banner ID: B00899629

Overview

This project requires the creation of a system which can accurately store and manage information relevant to People and Media Files. Media Files are not actually stored in the database. Rather the name and attributes of the files are recorded in the database.

The major requirement stated in the problem statement was to build a Family Tree comprising of individuals present in the system and reporting the Biological relation between them. The relation is reported in terms of Degree of Cousinship and Level of Removal.

Subsequently, various functions are required to be implemented to retrieve information pertaining to either people, media files or both.

Disclaimer

Multiple sites, relevant to Canadian Laws & Marriages, were referred, to implement as accurate a solution as possible, for recording marriages and dissolutions in the system. The solution or reasoning does not represent actual laws. Please refer to the Canadian legislature for accurate information on laws.

Data structures

Classes:

1. **PersonIdentity** – Manages data relevant to individuals in the system.

   Data Structures:

   i. Set<PersonIdentity> Children – To store the information of children for each parent.

   ii. Map<Integer, List<String>> Notes – To store notes on an individual

   iii. Map< Integer, List<String>> References – To store references on an individual

   iv. Map<String, String> Attributes – To store attributes of an individual

2. **FileIdentifier**– Manages data relevant to media files in a archive.

   Data Structures:

   i. Map< Integer,List<String>> Tags – To store Tags relevant to a media file.

   ii. Set<PersonIdentity> PeopleInMedia – To store the identifiers for people who appear in a media file.

   iii. Map<String, String> Attributes – To store attributes of a media file.

3. **BiologicalRelation** – Manages Data relevant to relationships between individuals in the system.

   Data Structures:

   i. Set<PersonIdentity> Descendants.
   ii. Set<PersonIdentity> Ancestors.
   iii. Map< List<Integer>, String> Relationships – To store the relationship between any two individuals.

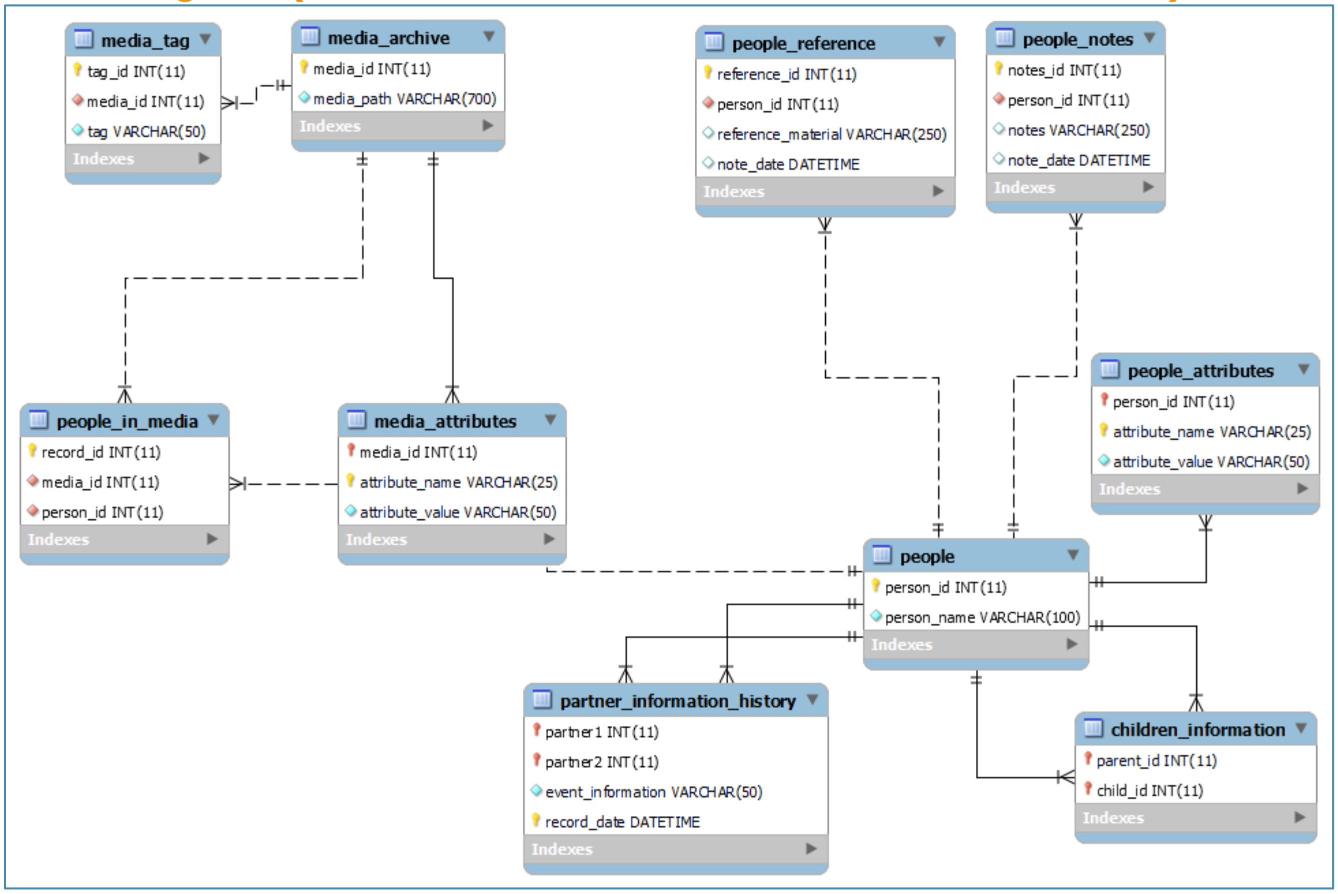
4. **Genealogy** – Parent Class which encompasses all other classes.

   Data Structures:

   i. Set<PersonIdentity> People – To add a person to the family tree.
   ii. Set<FileIdentifier>Media – To add Media files to an archive.
   iii. Set<FileIdentifier> MediaByTag – To store Media files for a given tag or location.
   iv. List<FileIdentifier> MediaByPeople – To store Media files for a given set of people or a person's children.

Code Design

1. ER Diagram (Available under the folder "Documentation):



2. Store Information

A major part of code design is to obtain information from the user and store it in the Database.

The following information will be stored and managed in a Database:

1. Name of a person and a unique identifier for the person.
2. Attributes of a person.
3. Notes and source reference materials for a person.
4. Partnering information between 2 people.
5. Records which describe a biological "Parent-Child" relationship between two individuals.
6. Media file paths (absolute paths) a unique identifier for the files.
7. Attributes of a media file.
8. Tags for a media file.
9. Relationship between individuals and media files. That is, records which store the identifiers of individuals who appear in a media file.

3. Report Information

Once the information from users is stored. The next step is to manipulate the data and render details which satisfy requirements.

1. The relationship between two individuals can be determined through Algorithm 1 as shown in the next section.
2. The set of ancestors and descendants within a particular generation gap can be obtained by traversing the Family Tree.
3. Notes & references for individuals can be obtained by a "Select" query in the database.
4. Media files can be filtered and returned on the basis of "Tags", "Location" and Individuals who appear in the media file.

Key Algorithms

One of the key algorithms in the project determines the relationship between two individuals in the system. The relationship is expressed in terms of fields known as "**Cousinship**" & "**Degree of Removal**".

1. The algorithm to determine the relationship between any two individuals is as follows:
2. Obtain the identifiers for the two individuals Person A and Person B.
3. Determine the nearest common ancestor of Person A and Person B. Let this be Person Z.
4. Determine the generation gap between Person A and Person Z. Let this be gAZ.
5. Determine the generation gap between Person B and Person Z. Let this be gBZ.
6. Assign the value [Minimum(GAZ,GBZ)-1] as the Cousinship between the individuals.
7. Assign the value [GAZ-GBZ] as the Degree of Removal between the individuals.
8. Return the Cousinship and Degree of Removal values.

Figure 1 illustrates the concept of Cousinship and Degree of Removal in a typical Family Tree.

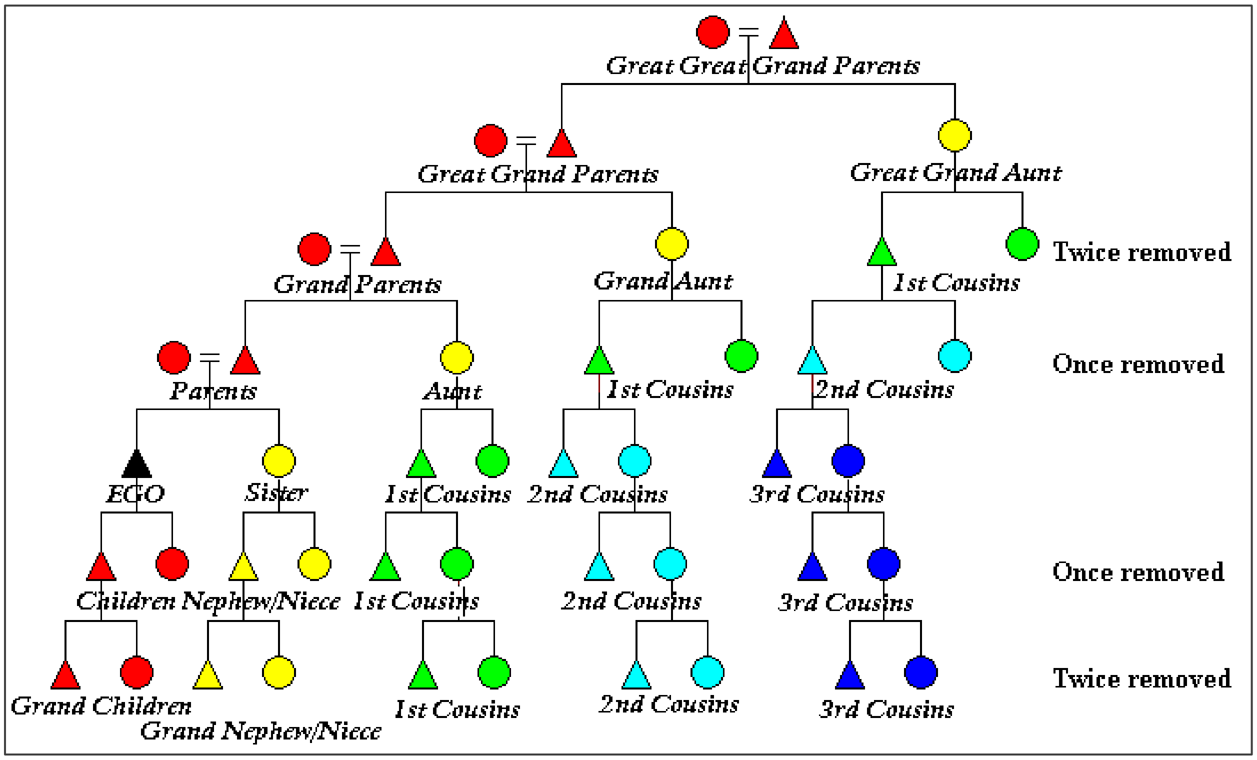


Figure 1: Degree of Cousinship and Removal [1]

Another key algorithm is one, which **retrieves a set of media files** relevant to a set of people or tags or location.

The algorithm is as follows:

1. Obtain the information required to filter the files in the Media Archive. This might be a set of identifiers for people or media files, or, one or more tags, or a location.
2. Retrieve the identifiers of files from the database by filtering files that are relevant to the objects obtained in Step 1. This can be achieved by passing the values in the 'Where" clause of a "Select" statement.
3. Return the identifiers.

To obtain the Media files relevant to the immediate children of a person:

1. Obtain the list of children for the person in context
2. Retrieve the identifiers of files from the database by filtering files that are relevant to the list of identifiers obtained in Step 1. This can be achieved by passing the values in the 'Where" clause of a "Select" statement.
3. Return the identifiers.

Important Operations & their implementation:

Boolean recordChild(PersonIdentity parent, PersonIdentity child)

When an user attempts to record a child for a parent, the program checks if the parent is married. If yes, another record is inserted into the database for the spouse. That is,

Parent1 -> Child

Parent2 -> Child

Boolean recordPartnering(PersonIdentity partner1, PersonIdentity partner2)

A symmetric partnering between two individuals is recorded only when the current status of the two individuals is "Single". This means that both the individuals can be unmarried or divorced. However, the relationship is not recorded when at least any one of the partner is partnered with someone else. In addition, the system considers the fact that a person can be partnered and dissociated multiple times.

Boolean recordDissolution(PersonIdentity partner1, PersonIdentity partner2)

A symmetric dissolution between two individuals is recorded only when the current status of the two individuals is "Married", and the individuals are married to each other.

BiologicalRelation findRelation(PersonIdentity person1, PesonIdentity person2)

In this method, the following steps are performed in order, to retrieve the relation between two people:

1. Find the ancestors for Person 1 and Person 2.
2. Store them ,each, in a Hashmap with the generation level, that is, {Key=Ancestor Id , Value = Level}.
3. Find the common ancestors in the two Hashmaps. Determine the ancestor with the lowest level.
4. Determine the levels of the two persons from this ancestor and substitute them in the formula mentioned above.

Set<PersonIdentity> descendents(PersonIdentity person, Integer generations)

In this method, the following steps are performed in order, to retrieve the ancestors of a Person within certain generations.

1. Retrieve the parent and children information from the database.
2. Build a tree using the information. Each PersonIdentity object has an ArrayList of pointers to store children.
3. Traverse the tree recursively till the desired level/generation is attained.
4. Store all the persons encountered in the tree and return them.

Set<PersonIdentity> ancestores(PersonIdentity person, Integer generations)

In this method, the following steps are performed in order, to retrieve the descendants of a Person within certain generations.

1. Retrieve the parent and children information from the database.
2. Build a tree using the information. Each PersonIdentity object has two pointers Parent1 and Parent to store the parents of a person.
3. Traverse the tree recursively till the desired level/generation is attained.
4. Store all the persons encountered in the tree and return them.

Assumptions

1. When Dates are entered for Media files (as attributes) Users are expected to store the Key Value Pair as "Date: Date_Value". That is, the Key must be named "**Date" (case-sensitive)**.
2. When Locations are entered for Media files (as attributes) Users are expected to store the Key Value Pair as "Location: Location_Value". That is, the Key must be named "**Location" (case-sensitive)**.
3. For a symmetric dissolution to be recorded between two persons, a record of a symmetric partnering relation between them, should exist in the database. **(That is, people should be married before getting divorced)**.
4. If two persons are involved in a symmetric partnering relation and if at least one of the people wants to record a symmetric partnering relation with another person, a symmetric dissolution should exist between the initial couple before recording a new relationship with other people. **(That is, a person should be divorced, before marrying another time).** This is in accordance with a **Canadian law** [6] that states that "**It's against the law to be married to more than 1 person at a time**."
5. Ideally, people do not re-marry their ex-spouse. However, since Canadian law does not explicitly state that you cannot divorce and marry the same person repeatedly, I have assumed that two persons can marry and divorce (each other) multiple times.

   That is, the following scenario is valid:

   A marries B – June 2010

   A divorces B – May 2015

   A marries B – January 2016

   Therefore, the table which records partnering, and dissolutions can have multiple "marriage" / "divorce" records between the **same people**.

6. Since Canadian law states that another marriage can occur only **31 days** after a divorce [7]. Also, in order to file for divorce in Canada you must first complete a full **one-year separation period** [8].

   Checking for a gap of 31 days or 1 year is beyond the scope of this project. Therefore, I have assumed that [a divorce followed by a second marriage] **(OR)** [a first marriage followed by a divorce] cannot occur on the same day. Therefore, the primary key in the table "**partner_information_history**" is a **composite key** which consists of **(Person 1, Person 2, Date)** where Date represents the date on which an event (Marriage / Divorce) occurred.

Complete Set of Test Cases

1. Add Persons to the Database:

- Add a person to the database when no other individuals exist in the database.
- Add a person to the database when 1 individual exists in the database.
- Add a person to the database when many individuals exist in the database.
- Add a person with an invalid name (Null / Empty String / Special characters).
- Add a person with a name that's identical to an existing name in the database.

2. Record Attributes of Persons:

- Record valid attributes of a person who exists in the database.
- Record valid attributes of a person who doesn't exist in the database.
- Update the attributes of a person who exists in the database.
- Update the attributes of a person who doesn't exist in the database.
- Record the birth date of a person where the date is greater than the current date.
- Record the death date of a person where the date is greater than the current date.
- Record attributes of a person without birth date, that is, all attributes **except** birth date are provided.
- Record attributes of a person without gender.
- Record attributes of a person without occupation.

3. Record a Reference Material for Persons:

- Record a material for a person who exists in the database.
- Record a material for a person who doesn't exist in the database.
- Record multiple materials for a person who exists in the database.

4. Record Notes for Persons:

- Record notes for a person who exists in the database.
- Record notes for a person who doesn't exist in the database.

- Record multiple notes for a person who exists in the database.

5. Record a Child for Persons:

- Record a valid (existent) child for a person who exists in the database.
- Record a child for a person who doesn't exist in the database.
- Record a relationship between the same child and parent multiple times.
- Record multiple children for a person who exists in the database.
- Record a child for a person who exists in the database when the child does not exist.
- Record a relationship between the same person. That is, same person as parent and child.

6. Record a Partnering Relation between Persons:

- Record a valid partnering relation between people who exist in the database.
- Record a partnering relation between people when one person doesn't exist in the database.
- Record a partnering relation between people when both individuals do not exist in the database.
- Record a partnering relation between two people who **already** have a symmetric Partnering relation defined **between them.**
- Record a partnering relation between two people when **both** have a symmetric Partnering relation defined **with other people.** (That is another marriage **without** a divorce).
- Record a partnering relation between two people when **any one** person has a symmetric Partnering relation defined **with other people.** (That is another marriage **without** a divorce).
- Record a partnering relation between two people who have a symmetric dissolution relation defined **with other people.** (That is another marriage **with** a divorce).
- Record a relationship between the same people. That is, same person as Partner 1 and Partner2.
- Record a relationship with different order of person 1 and person 2. (That is, if a marriage between Persons 1 and 2 is recorded, the same should not be entertained for Persons 2 and 1 again).

7. Record a dissolution of a Partnering Relation between Persons:

- Record a dissolution between people who exist in the database and had a partnering relation defined between them.
- Record a dissolution between people who exist in the database but do not have a partnering relation defined between them.
- Record a dissolution between two people who already had a dissolution of relation defined between them.
- Record a dissolution between people when one person doesn't exist in the database.
- Record a dissolution between people when both individuals do not exist in the database.
- Record a dissolution between people who exist in the database and have a partnering relation defined **NOT** between them but with other Persons.

8. Add Media Files to an Archive:

- Add a media file to the archive.
- Add a media file to the archive when the location of the file already exists in the archive.
- Add a media file with an invalid file path.

9. Record Attributes of a Media file in the Archive:

- Record valid attributes of a media file which exists in the archive.
- Record valid attributes of a media file which doesn't exist in the archive.
- Update the existing attributes of a media file with new ones.
- Record date for a media file when the date is greater than the current date.

10. Record People in Media Files:

- Record People in a file when both the file and people exist.
- Record People in a file when any one of the file and people don't exist.
- Record People in a file when both the file and people don't exist.

- Record redundant information. That is, record that a set of people exist in a media file, multiple times.

11. Record tags for Media Files:

- Record tags for a file which exists in the archive.
- Record tags for a file which doesn't exist in the archive.
- Record empty tags for a file.
- Record multiple tags for a same media file.

12. Report:

- Locate an individual when the individual exists in the database.
- Locate an individual when the individual doesn't exist in the database.
- Location an individual when the individual's name is empty. That is, pass an empty string while attempting to locate a person.
- Report a person's name when the person exists.
- Report a person's name when the person does not exist.
- Report notes and source reference materials for a person who exists in the database.
- Report notes and source reference materials for a person who does not exist in the database.
- Report relations when two people exist.
- Report relations when any one person does not exist.
- Report relations when two people do not exist.
- Report relation between people who have direct ancestor descendant relationship.
- Report relation between people who do not have direct ancestor descendant relationship (Pibling and Nibling included).
- Report ancestors / descendants of a person within a given generation.
- Report the ancestors of a person when the information isn't available in the database.
- Report the descendants of a person when the person doesn't have any descendants.
- Report people who are linked a media file.
- Report people who are linked a media file but the file path doesn't exist in the archive.

- Locate a media file when the file exists in the archive.
- Locate a media file when the file doesn't exist in the archive.
- Location a media file when the file path is empty. That is, pass an empty string while attempting to locate a media file.
- Report files which fall within a given date range.
- Report media files when any one of start date or end date is not passed.
- Report media files when both start date or end date are not passed.
- Report files which are supposedly related to a given set of people who exist in the database.
- Report files which are supposedly related to a given set of people, but the people do not exist.
- Report media files linked to a given tag and timeframe.
- Report media files linked to a given tag and timeframe but the tag doesn't exist.
- Report media files linked to a given location and timeframe.
- Report media files linked to a given location and timeframe but the location doesn't exist.
- Report media files (in chronological order) linked to the immediate children of a person, within a timeframe.
- Report media files (in chronological order) linked to the immediate children of a person, within a timeframe when the person has no immediate children.
- Report media files (in chronological order) linked to the immediate children of a person, within a timeframe when there are no media files.

Exceptions

Custom Exceptions:

1. **PersonNotFoundException –** Thrown when an operation is attempted to be performed on a person who does not exist. This Exception was created to mimic the foreign key constraint in the database.
2. **FileNotFoundException -** Thrown when an operation is attempted to be performed on a media file which does not exist.

3. **DuplicateRecordException –** Thrown when an attempt is made to insert a duplicate record in the database. This Exception was created to mimic the primary key constraint in the database.

Class: Genealogy

1. addperson(String name)
   a. ClassNotFoundException
   b. SQLException
   c. IllegalArgumentException
2. recordReference(PersonIdentity person, String reference)
   a. ClassNotFoundException
   b. SQLException
   c. PersonNotFoundException
3. recordNote(PersonIdentity person, String note)
   a. ClassNotFoundException
   b. SQLException
   c. PersonNotFoundException
4. recordAttribute (PersonIdentity person, Map attributes)
   a. ClassNotFoundException
   b. SQLException
   c. PersonNotFoundException
5. recordChild(PersonIdentity parent, PersonIdentity child)
   a. ClassNotFoundException
   b. SQLException
   c. PersonNotFoundException
   d. DuplicateRecordException
6. recordPartnering(PersonIdentity partner1, PersonIdentity partner2)
   a. ClassNotFoundException
   b. SQLException
   c. PersonNotFoundException

   d. DuplicateRecordException

7. recordDissolution(PersonIdentity partner1, PersonIdentity partner2)
   a. ClassNotFoundException
   b. SQLException
   c. PersonNotFoundException
   d. DuplicateRecordException
8. addMediaFile(String fileLocation)
   a. ClassNotFoundException
   b. SQLException
   c. DuplicateRecordException
   d. IllegalArgumentException
9. recordMediaAttributes(FileIdentifier fileIdentifier, Map<String, String> attributes)
   a. ClassNotFoundException
   b. SQLException
   c. FileNotFoundException
10. Boolean peopleInMedia(FileIdentifier fileIdentifier, List<PersonIdentity> people)
    a. ClassNotFoundException
    b. SQLException
    c. FileNotFoundException

11. Boolean tagMedia(FileIdentifier, fileIdentifier, String tag)
    a. ClassNotFoundException
    b. SQLException
    c. FileNotFoundException

Reporting

12. PersonIdentity findPerson(String name)
    a. SQLException
13. FileIdentifier findMediaFile(String name)

   a. SQLException

14. String findName(PersonIdentity id)
    a. SQLException
15. String findMediaFile(FileIdentifier fileId)
    a. SQLException
16. BiologicalRelation findRelation(PersonIdentity person1, PesonIdentity person2)
    a. SQLException
17. Set<PersonIdentity> descendents(PersonIdentity person, Integer generations)
    a. SQLException
18. Set<PersonIdentity> ancestores(PersonIdentity person, Integer generations)
    a. SQLException
19. List<String> notesAndReferences(PersonIdentity person)
    a. SQLException
20. Set<FileIdentifier> findMediaByTag(String tag , String startDate, String endDate)
    a. SQLException
21. Set<FileIdentifier> findMediaByLocation(String location, String startDate, String endDate)
    a. SQLException
22. List<FileIdentifier> findIndividualsMedia(Set<PersonIdentity> people, String startDate, String endDate)
    a. SQLException
23. List<FileIdentifier> findBiologicalFamilyMedia(PersonIdentity person)
    a. SQLException

What else could have been achieved (or) done better ?

1. The entire set of statements for establishing a connection could have been placed in the constructor of the class Genealogy so that the statements aren't present in each function. This would have made it easier for users to substitute their credentials and desired schema while testing.
2. Parsing dates (Year, Year & Month, Year, Month and Date) into a common format to achieve versatility, wasn't implemented. This would have been achieved after more intensive testing.

3. The solution queries the database each time to retrieve a piece of information. Loading all the data beforehand and storing it in Data Structures would have proven to be more efficient, but at the cost of memory.

References

1. https://www.umanitoba.ca/faculties/arts/anthropology/tutor/descent/cognatic/collateral.html#top
2. https://stackoverflow.com/questions/4085420/how-do-i-read-a-properties-file-and-connect-a-mysql-database
3. https://alvinalexander.com/java/java-mysql-insert-example-preparedstatement/
4. https://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html
5. https://stackoverflow.com/questions/35111475/how-to-pass-arraylist-as-in-clause-in-sql-query-in-mysql
6. https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-usagenotes-last-insert-id.html
7. https://www.javatpoint.com/how-to-compare-two-objects-in-java
8. https://www.baeldung.com/java-new-custom-exception
9. https://www.canada.ca/en/immigration-refugees-citizenship/services/new-immigrants/learn-about-canada/laws/family-law.html#:~:text=Marriage%20and%20divorce,1%20person%20in%20the%20past.&text=You%20can%20only%20remarry%20if,or%20your%20spouse%20has%20died.
10. https://www.torontodivorcelaw.com/long-divorce-canada/#:~:text=to%20be%20resolved.-,How%20long%20do%20you%20have%20to%20wait%20to%20get%20remarried,legally%20separated%20from%20your%20ex.
11. https://divorce-canada.ca/marriage-separation-in-canada