Problem

The problem is based on the genealogy field which is the study to track family relations however this study does not provide much information about the person like gender, occupation, images of the individuals. So the problem will provide the solution to store the biological relationship between people and store the information about it in the database to retrieve whenever it is needed.

Assumption:

The program will only consider the family relation system team has two components, family tree database, and media archive database to store relation and its metadata into the database.

The Family tree has the following tables to provide the relations and the information about that person which are given below:

Database design for the Family Relation system

- **PersonIdentity:** this table is required to identify the individuals from the family tree and it includes personId and personName.
- **BiologicalParentingRelation:** this table is to store the relation between two members of the family. The relation stored in this table will be the ancestor and descendent. The table also stores the level of the generation as generation in the table.
- **BiologicalPartneringRelation:** the table stores the partnering and the dissolution relation between two people.
- **Attributes:** this table stores the key of the attribute such as date of birth, location of the birth, date of death, location of death, occupation and its value like "gender" as key and the "male" as value to that key.
- **SourceReferences:** this table will store the references of the file for a particular person
- **Notes:** the table stores notes for the family member.

Database design for the Media Archive:

- **FileIdentifier:** The table identifies the file uniquely and stores it along with its id
- **MediaAttribute:** media attributes will store the media attribute like the date of the picture taken location of the image, tags, and individuals in the images.
- **Tags:** the table will store the tags for the file
- **Individuals:** the table will store the person's IDs that are in the specific image.

CRC (Class Responsibility Collaborator):

PersonIdentity	
PersonId	Attributes Notes BiologicalParentingRelation BiologicalPartneringRelation Individuals SourceReference
PersonName	

FileIdentifier	
fileId	Individuals Tags mediaAttributes
Filename	

Entity Relation Diagram

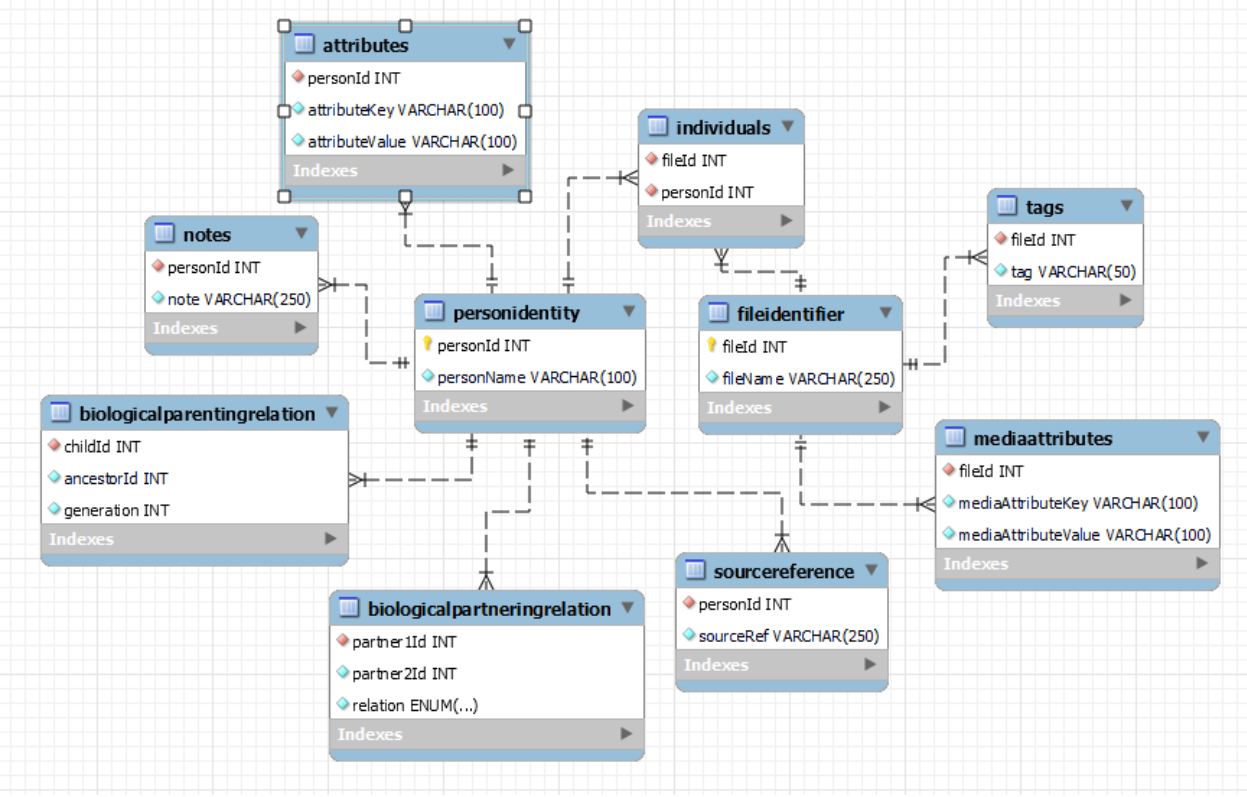

Fig. ERD for genealogy database

Files Listed for the solution :
Persistant: the package for all classes to manage family tree relations and the media archive

Includes files:

- **PersonIdentity:** it identifies the individual from the family relation tree.

  Methods included:
  - Getter-Setter method for the class
- **Genealogy**: class manages all the operation done on the person on the family relation tree.

  Methods included:

  - addPerson() : will add the person to the database.
  - recordAttributes(): will record the given HashMap of the attributes to the database.
  - recordReference(): will record the reference of the person to the database.
  - recordNote(): will record the notes for the specific person to the database.
  - findPerson(): will find the object of the person class if given the name of the person
  - findName(): will return the name of the person if the PersonIdentity class object is given
  - findRelation(): will return the BiologicalRelation object containing the degree of coushinship and degree of the removal for the relation between two person
  - notesAndReferences(): will return the list of the notes and references for particular person

- **BiologicalRelation:** it creates the relation between two members and stores it into the database.
  Mehods included:

  - recordChild(): will store the relation between parent and child to the database
  - recordPartnering(): will store the partnering relation between two person
  - recordDissolute(): will store the dissolution relation between two person if they are married.
  - descendents(): will return the list of the descendants of the person
  - ancestors(): will return the list of the ancestors of the person

- **FileIdentifier**: it is the class for creating the file identity object to uniquely identify media file.

  Methods included:
  Getter-Setter method for the class

- **MediaArchive:** it saves the metadata of the media file and its metadata.
  Methods included:
  - addMediaFile() : will add the media file to the data base
  - recordMediaAttributes() : will add the attributes of the HashMap to the media archive
  - peopleInMedia(): will record the set of the individual that shown in the media file.
  - tagMedia(): will record the tags into the media archive.
  - findMediaFile(): will find if the media is exist in the media archive

- findMediaByTag(): will find all the media for the specific tag.
- findMediaByLocation(): will find all the media files based on the location.
- findIndividualsMedia(): will return the media files for given sets of the PersonIdentity class object
- findBiologicalFamilyMedia(): will return the files for which includes the immediate child of the person.

Data Structure:

The solution has used the tree data structure to store the family relations, and all the attributes and its values are stored into the HashMap.

Key algorithm with example:

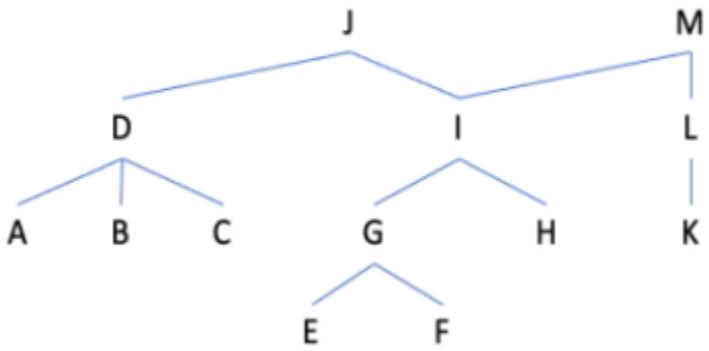

Fig. 2 Family relation diagram

For above family tree here, the data will be store in the database in the following order.
Let's assume in the begging database is empty, now user added person "J" into the database using addPerson method. In this manner user added all the family members into the table but still there is no relation added into the table. All the data for person is stored into the personIdentity table. Now let's say, user added parent child relation between the person "A" and person "D". so now in the row will be added into biologicalRelation table where parentId will be person D's personId and the childId will be A's personId and for relation and level field it will be "parent" and 1 respectively. Now in the next step, system will itetrate throught all the ancestors of parents of person A, and it will add the data for it into the biologicalRelation table. For example, all the ancestors of G will be store in the biological as described below.

Table 1 : example for biologicalParentingRelation

childId	parentId	Level
1 <personId for G>	2 <personId for I>	1
2 <personId for I>	3 <personId for J>	2
3 <personId for I>	4 <personId for M>	2
5 <personId for A>	6 <personId for D>	1
6 <personId for D>	3 <personId for J>	2

So now to find the relation between person G and person A. first extract all the rows of childId of G with parent relation and store it into the map where the key is parentId of the row and the value will be the level of the parent. Now for finding the relation between two members in we have levels of each member so cousinship will be the smaller level minus one and difference between two levels will be the level of separation.

Blackbox Testing

PersonIdentity addPerson (String name)

• name as null string or empty string
• name as alphanumeric value
• name as one character string
• name as two or more than two-character string
• Duplicate name
• Call recordChild (), recordPartnering(), recordDissolution() before addperson()

Boolean recordAttributes (PersonIdentity person, Map attributes)

• Attribute name as empty string or null string
• Person as null object
• name as one character string
• name as two or more than two-character string
• Duplicate attribute name
• Call recordAttributes() before add addPerson()

Boolean recordReference(PersonIdentity person, String reference)

• Reference as empty string or null string
• Person as null object
• Duplicate reference
• Call recordReference() before add addPerson()

Boolean recordNote(PersonIdentity person, String note)

• Empty or null string as note
• Person as null object
• Person does not exist

Boolean recordChild (PersonIdentity parent, PersonIdentity child)

• Null object passed as parent and/or child
• Parent already exists
• Child already exits
• Parent has one child exist
• Parent has more than one child exist
• Parent has no child exist
• Child has already one parent
• Child has two parents
• Call record recordChild before addPerson()

Boolean recordPartnering (PersonIdentity partner1, PersonIdentity partner2)

• Null object passed as partner1 and/or partner2
• Partner1 already exist
• Partner2 already exist
• Partner1 is already partner with partner2
• Partner1 has already dissolute with partner2
• Partner1 is ancestors or descendent of partner2

Boolean recordDissolution (PersonIdentity partner1, PersonIdentity partner2)

• Null object passed as partner1 and/or partner2
• Partner1 already exist
• Partner2 already exist
• Partner1 is already partner with partner2
• Partner1 has already dissolute with partner2
• Partner1 is ancestors or descendent of partner2

FileIdentifier addMediaFile (String fileLocation)

• null or empty string passed as fileLocation
• duplicate filename

Boolean recordMediaAttributes(FileIdentifier fileIdentifier, Map attributes)

• Null or empty string passed as fileIdentifier
• fileIdentifier does not exist
• null object passed as attributes
• fileIdentifier already has attributes

Boolean peopleInMedia(FileIdentifier fileIdentifier, List people)

• Null or empty string passed as fileIdentifier
• fileIdentifier does not exist
• null object passed as people
• one or more people element does not exist

Boolean tagMedia(FileIdentifier, fileIdentifier, String tag)

• Null or empty string passed as fileIdentifier
• Null or empty string passed as tag
• Duplicate tag
• fileIdentifier not exist
• call tagMedia() before addMediaFile()

PersonIdentity findPerson(String name)

• Null or empty string passed as name
• Name as one character length name
• Name as two or more-character length name
• Person does not exist in family tree
• Call findPerson() before addPerson()

FileIdentifier findMediaFile (String name)

• Null or empty string passed as name
• Name as one character length name
• Name as two or more-character length name
• Media does not exist in media archive
• Call findMediaFile() before addMediaFile()

String findName(PersonIdentity id)

• Id as null object
• personIdentity does not exist

String findMediaFile (FileIdentifier field)

• FileIdentifier as null object
• Filemedia does not exist

BiologicalRelation findRelation (PersonIdentity person1, PesonIdentity person2)

• Null object passed as person1 and/or person2
• Person1 and/or Person2 does not exist in family tree
• Person1 and Person2 are same object

Set descendents(PersonIdentity person, Integer generations) and
Set ancestores(PersonIdentity person, Integer generations)

- Person as null object
- Negative number as generations
- Zero as generations
- Positive number as generations
- Person does not exist in family tree
- Generations is grater than length of the tree

List notesAndReferences(PersonIdentity person)

- Person as null object
- Person does not exist
- Peron has zero notes and references

Set findMediaByTag(String tag , String startDate, String endDate)

Set findMediaByLocation(String location, String startDate, String endDate)

List findIndividualsMedia(Set people, String startDate, String endDate)

- Empty String or null string passed as tag
- One character string passed as tag
- Two or more than two-character length string passed as tag
- Empty string or null string passed as location
- One character string passed as location
- Two or more than two-character length string passed as location
- Alphanumeric value passed as location
- Null object passed as people
- startDate and endDate as alphanumeric value
- startDate and endDate as numeric value
- startDate and endDate grater than 1 and less than 28,30 or 31
- startDate is less then endDate
- startDate is greater than endDate
- startDate is equals to endDate
- startDate and/or endDate only consist of month and year
- startDate and/or endDate only consist of date
- startDate and/or endDate only consist month or year
- tag does not exist in fileIdentifier
- location does not exist in fileIdentifier
- person does not exist in fileIdentifier

List findBiologicalFamilyMedia(PersonIdentity person)

• Null object as person
• Person does not exist in family tree
• There are no media related to person
• Person has no children
• Person has one immediate child
• Person has two immediate children

Limitations:

- The dates only allowed in YYYY-MM-DD format.
- The family tree only includes the biological relations
- The relations can not be removed after it created into the family tree.
- All the media are stored in terms of URL or references

References

[1] javatpoint. [Online]. Available: https://www.javatpoint.com/example-to-connect-to-the-mysql-database. [Accessed 14 December 2021].

[2] w3schools, "SQL Tutorial," [Online]. Available: https://www.w3schools.com/sql/. [Accessed 14 December 2021].

[3] javacodeexamples, "Java RegEx," [Online]. Available: https://www.javacodeexamples.com/java-regex-validate-dd-mm-yyyy-mm-dd-yyyy-date-format/3457. [Accessed 14 December 2021].

[4] geeksforgeeks. [Online]. Available: https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/. [Accessed 14 December 2021].