Data Structures

A) PersonIdentity:
The PersonIdentity object stores the following relations:
1. parents: Stores the parent/parents related to the person in a list(List<PersonIdentity>)
2. partner: Stores the partner(PersonIdentity)
3. children: Stores all the children in a list(List<PersonIdentity>)
4. rootAncestors: Stores root ancestors of the person in a set(Set<PersonIdentity>)

B) rootAncestors:
Root ancestors represent the root in the Genealogy tree. There can be multiple root ancestors in the tree. Each individual in the tree stores the corresponding root ancestors.

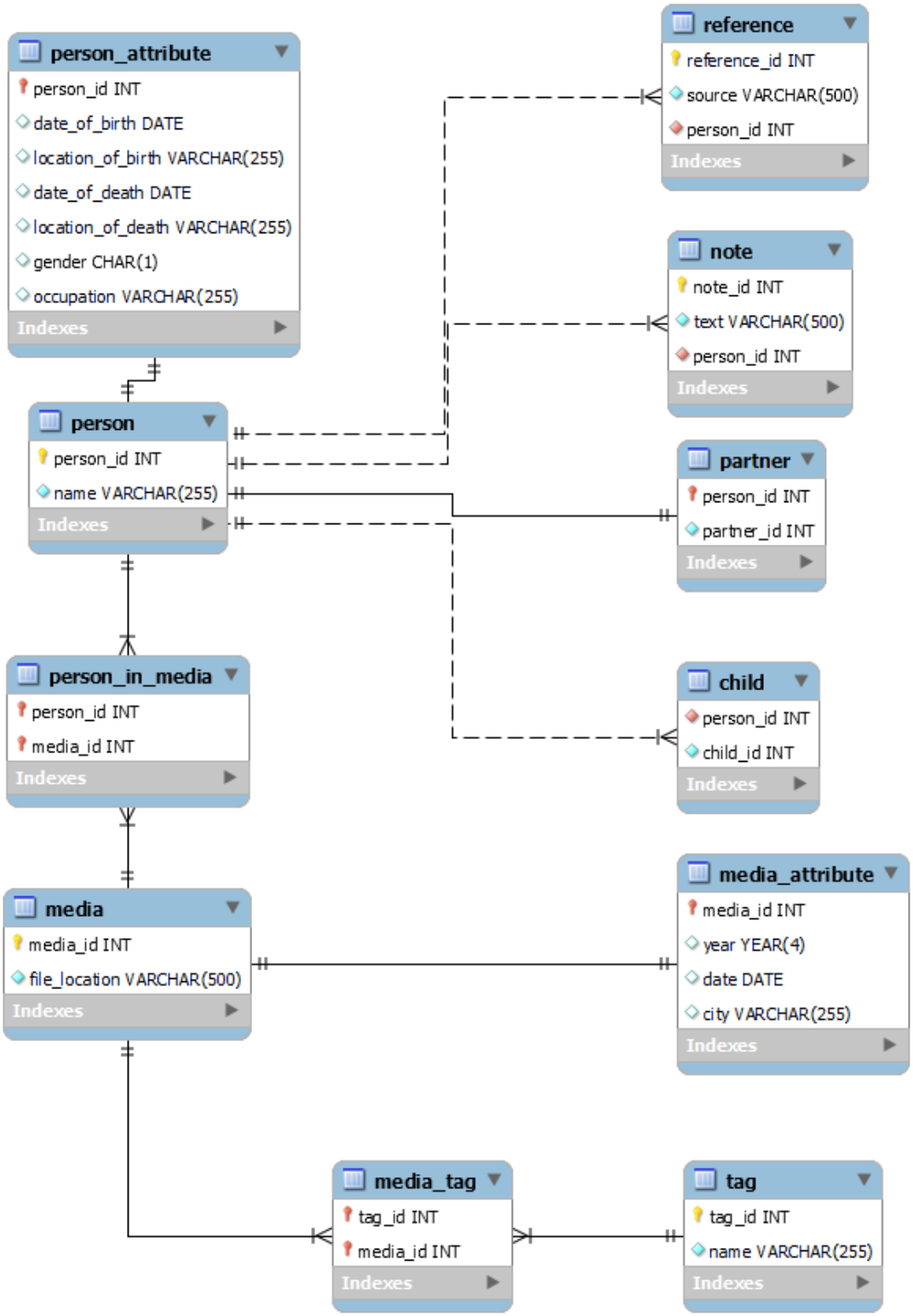
C) FamilyManagement:
The FamilyManagement class stores the following data structures:
1. roots: Represents the root individuals in the genealogy tree(Set<PersonIdentity>)
2. partnered: Represents all the individuals who have a partner(Set< PersonIdentity >)
3. children: Represents all the individuals who are a child of a person(Set<PersonIdentity>)
4. personIds: A map to store the personId and PersonIdentity object(Map<Integer, PersonIdentity>)

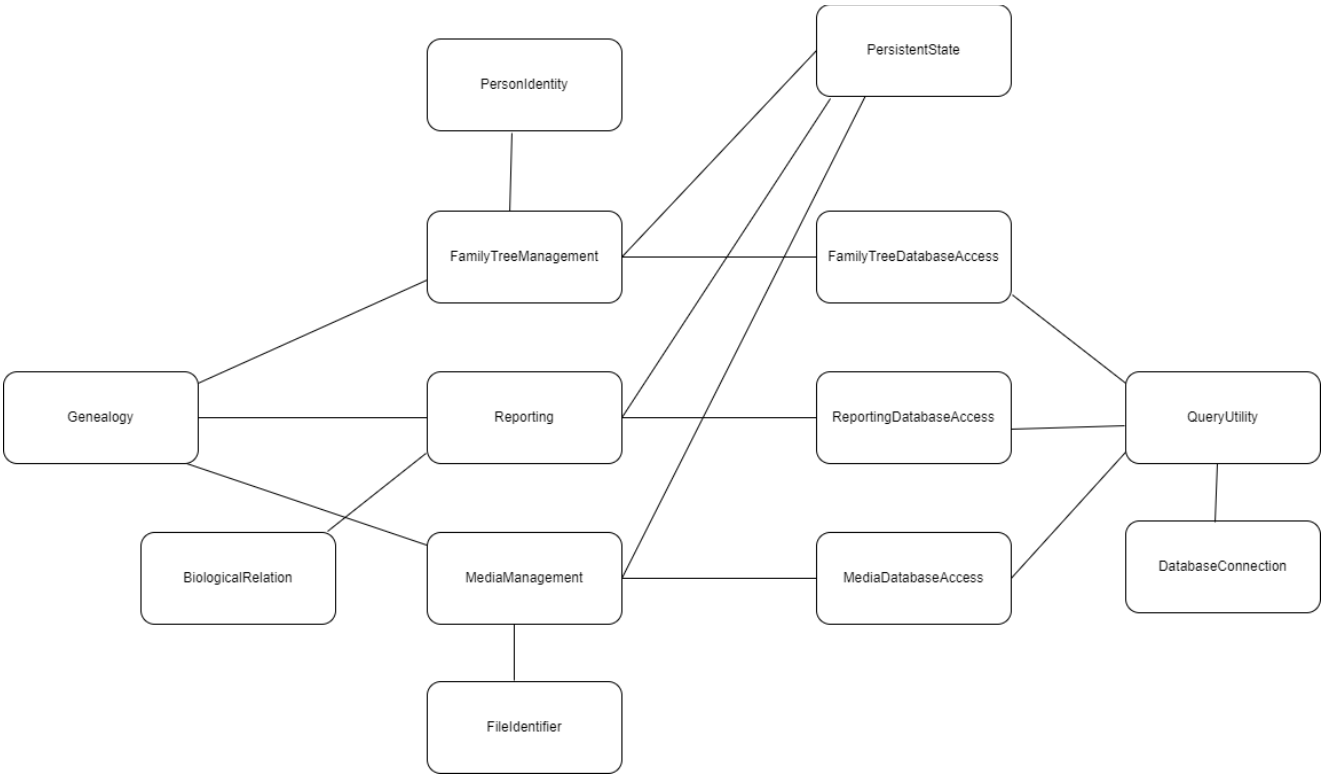
D) MediaManagement:
The media management class stores the following objects:
1. tags: Stores tag name as the key and tagId as the value(Map<String, Integer>)
2. files: Stores fileLocation as the key and FileIdentifier object as the value(Map<String, FileIdentifier>)

E) Genealogy Tree:
The fields of the PersonIdentity class discussed above automatically creates a tree like data structure. The tree can be thought of as top to bottom – In this case, the tree will be a multi-node tree. It can also be considered from bottom to top – In this case, the tree will be a binary tree(Since a person can have a maximum of two parents)

Code Design

Entity Relationship Diagram:





I have distributed the classes into three different layers.

A) Input/Output layer:
The user will be able to interact with these classes while running the program.
1. Genealogy: The class which can be used by the user to invoke all the methods
2. BiologicalRelation: The class returned as the output showing relation between two persons
3. PersonIdentifier: An entity that represents a person in the genealogy tree
4. FileIdentifier: An entity for storing media related information.

B) Intermediate/Validation layer:
These classes as an intermediate layer between the input layer and database layer. The input validation is done in this layer.
1. FamilyTreeManagement: Updates/Inserts the family tree related data structures.
2. MediaManagement: Updates/Inserts the media related data structures.
3. Reporting: Retrieval of data from the tree. Traversal of the tree.

C) Database layer:
These classes hold the information related to the database tables.
1. FamilyTreeDatabaseAccess
2. ReportingDatabaseAccess
3. MediaDatabaseAccess

4. PersistentState: This class is used to retrieve and rebuild the data structure of each new session.
5. QueryUtility: Holds commonly used generic query methods.
6. DatabaseConnection: This class is used to establish a common database connection for the entire program.

Key Algorithms

1. Identifying root ancestors:
   It is important to find the root ancestor in the genealogy tree.
   a) Whenever a new person gets added to the tree, the person automatically becomes one of the roots.
   b) When a person is recorded as the child of a parent, the person loses the root status.

2. Updating root ancestors:
   The genealogy tree is a multi-root tree. To find the relationship between two person, it is necessary to narrow down a common root ancestor between two persons. The easiest way to do this is to maintain a rootAncestor set in the PersonIdentity class. The rootAncestors should be updated in two different cases.

   Scenario 1: While loading the data from the database tables
   a) Iterate over all the roots in the tree.
   b) In each iteration, recursively update the root node as the root ancestor for the descendant nodes.

   Scenario 2: While adding a child
   a) Check the root ancestors of the parent nodes.
   b) Add the root ancestors of both the parents as the root ancestors of the child node.
   c) Recursively traverse all the descendants of the child node -> remove the child node from the root ancestors of the descendants -> And add the new root ancestors of the child node to all the descendants.

3. Sequence of loading family tree information from the database:
   a) Get all individuals
   b) Get all roots
   c) Get partnering relationships
   d) Get all children

4. Finding ancestors:
   Recursively traverse all the parents of the current node till the given number of generations to find all the ancestors.

5. Finding descendants:

Recursively traverse all the children of the current node till the given number of generations to find all the descendants.

6. Finding relation:
   a) Before finding relation, check if the two persons have any common root ancestors.
   b) If common root ancestors are not present, then they are not related.
   c) Otherwise, pick any one of the common root ancestors.
   d) Create two double ended queues for each person to find the path from the person node to the picked root ancestor node.
   e) Traverse recursively to add the nodes between the person node and the root node.
   f) After adding nodes to both the queues, remove common nodes between two queues.
   g) The size of the queues represents nX and nY.
   h) Calculate degree of cousinship and level of removal between using nX and nY.

Additional White Box Tests

1. addPerson():
- When the database connection is not present
- Add two persons using two different Genealogy objects
- Add person when no person is present in the database
- Add person when one person is present in the database
- Add person when many persons are present in the database

2. recordAttributes():
- For a person already present in the database
- For a person added in the same session
- Record attributes in a different Genealogy object
- Record attributes twice for the same person

3. recordReference():
- Same reference for two different individuals
- Add reference for a person added in a different session
- Add reference for a person added in the same session

4. recordNote():
- Same reference for two different individuals
- Add reference for a person added in a different session
- Add reference for a person added in the same session

5. recordChild():

- Record child for partnered individuals
- Record child after dissolution
- Record child twice for the same individual
- Record child twice for two different individuals
- Record child added in the same session as the parent
- Record child for the person added in a different session

6. recordPartnering():
- Record partner for partnered individuals
- Record partner after dissolution
- Record partner twice for the same individual
- Record partner twice for two different individuals
- Record partner added in the same session as the other partner
- Record partner for the person added in a different session

7. recordDissolution():
- Record dissolution for partnered individuals
- Record dissolution after dissolution
- Record dissolution twice for the same individual
- Record dissolution twice for two different individuals
- Record dissolution added in the same session as the other partner
- Record dissolution for the person added in a different session

8. addMediaFile():
- When no media is present in the session
- When no media is present in the database

9. recordMediaAttributes():
- For media already present in the database
- For media added in the current session
- For media added in a different genealogy object

10. peopleInMedia():
- For media already present in the database
- For media added in the current session
- For media added in a different genealogy object
- When the database connection is not present

11. tagMedia():
- For media already present in the database
- For media added in the current session
- For media added in a different genealogy object

12. findPerson():
- Person added in the same session
- Person already present in the database

13. findMediaFile():
- Media added in the same session
- Media already present in the database

14. findRelation():
- For both the persons added in the same session
- For one of the persons added in the same session
- For both the persons present in the database

15. ancestors():
- For the person added in the same session
- For the person added in a different session

16. descendants():
- For the person added in the same session
- For the person added in a different session

17. notesAndReferences():
- For the person added in the same session
- For the person added in a different session

18. findMediaByTag():
- Media added in the same session
- Media already present in the database
- Tag added in the same session
- Tag already present in the database

19. findMediaByLocation():
- Media added in the same session
- Media already present in the database

20. findIndividualsMedia():
- Media added in the same session
- Media already present in the database
- For the person added in the same session
- For the person added in a different session

21. findBiologicalMedia():
- For the person added in the same session
- For the person added in a different session

Progress

1. Create conceptual, logical and physical models for storing the family tree information and media entities.

Tasks	Entities	Tools	Status
Design conceptual/logical model	person, person_attribute, note, reference, partner, parent_child, media, media_attribute, people_media, tag	Draw.io	Completed
Design physical model: Define tables and column constraints		MySQL Workbench	Completed
Design physical model: Develop ERD, Normalize using iterative approach, Forward engineer the model			Completed
Integrate Genealogy database with the java code	NA	Java, Eclipse, MySQL connector	Completed

2. Work on the java classes.

Tasks	Status
1. Create a class called Geneaology implementing above interfaces. 2. Implement FileIdentifier class. 3. Develop SQL queries for adding media, recording media attributes, and tagging media. 4. Implement methods for storing media information in the database: • addMediaFile • recordMediaAttributes • tagMedia	Partially Completed

1. Implement PersonIdentity class. 2. Develop SQL queries for adding person, recording person attribute, recording reference and recording note. 3. Implement methods for storing an individual's information in the database: • addPerson • recordAttributes • recordReference • recordNote 4. Implement the peopleInMedia method.	Partially Completed
1. Implement data structure for storing family tree hierarchy 2. Develop SQL queries for recording child, recording partner, and recording dissolution. 3. Implements methods for defining relationships in the database and in the memory: • recordChild • recordPartnering • recordDissolution 4. Implement methods to retrieve information from the database to build a family tree hierarchy.	Completed
1. Implement methods to find the most recent ancestor in the family tree 2. Develop SQL queries for finding a person, finding a media file, finding name, and finding a media file. 3. Implement reporting functions: • findPerson • findMediaFile • findName • findMediaFile • findMediaByTag • findMediaByLocation • notesAndReferences	Partially Completed
1. Implement reporting functions • findRelation • descendents • ancestors • findIndividualsMedia • findBiologicalFamilyMedia	Partially Completed