Project

Name: Hardee Rameshchandra Garala **Std Id:** B00869195

Faculty: Prof. Mike McAllister **Date:** 14th Dec 2021

Overview:

The purpose of the project is to record the data related to families and the files or media they are a part of and report the required details using genealogy.

File and external data:

Below files are used to implement the following problem.

1. BiologicalRelation.java: This class declares degree of cousinship and removal. It also contains getter and setter method to access these variables.
2. FileIdentifier.java: This class contains private variable for id of a file, file location and file name. Also, it contains getter and setter methods to access all these private variables.
3. Genealogy.java: This file contains all the methods to add data of a person or a file and the methods to retrieve the required information
4. PersonIdentity.java: This file contains private variables for id and name of a person. It also contains count to define at what level it is with respect to other person while finding relation.
5. Main.java: it contains switch cases to access different methods of the program for personal testing and accessing the functionality.
6. Project.sql: This file contains the queries to create the database.

Assumptions:

1. The object passed in parameters is same as the one returned while adding a person or media file to database through addMediaFile() and addPerson() method, meaning that all the object passed have the data present in the database.
2. The main.java is not used for testing all the methods of family tree management, media archive management and reporting methods.

Data Structures, design and algorithms:

I have used hashset, arraylist, graph (using adjacency list) and arrays in this project. I have implemented below methods for the project:

PersonIdentity addPerson(String name): The name of the person is passed as a parameter and that person is stored in the table “person”. The id for this table is set to auto increment, so whenever a name is added a unique id is created which is +1 to the last created id. These id and name are fetched to create an object of type PersonIdentity, while loop is used to get the id in order to get the last added person’s id if two or more same names exist in the table. This new created object is returned.

Boolean recordAttributes(PersonIdentity person, Map<String, String> attributes): This method is used to store attributes of a person in the database. For now, I have added pre-defined set of attributes and accept keys like that only. The list of these keys is:

- birthdate: the date is accepted in yyyy-mm-dd format
- birthlocation
- deathdate: the date is accepted in yyyy-mm-dd format
- deathlocation
- gender: gender is accepted either as 'M' or 'F'
- occupation

Any key value other than this will not be stored, and a message will be returned to the user mentioning which key was not accepted. All the other attributes which have the listed attribute will be stored. Even if one attribute is not recorded the method returns false as we can say that all attributes are not recorded. If all the attributes are recorded true is returned.
We can accommodate other attributes by giving an option to add other attribute in drop down menu, where user can give an attribute, they want to add and is not present in the list, which can be verified and a new column can be added for that in person table and an option can be provided to add that attribute later.

Boolean recordReference(PersonIdentity person, String reference): This method stores the reference into the table reference_detail along with the id of the person the reference is related to and the time and date when the data is stored. I have created different table or recording references as one person can have 0 or more references so I have connected person table to reference_detail table using foreign_key. If the data is stored it returns true otherwise throws an exception.

Boolean recordNote(PersonIdentity person, String note): This method stores the id of the person and the note associated with the person. Similar to reference I have created a different table for recording notes as there can be 0 or more notes associated with a person. When the data related to notes is stored in the table, I also store the time and date when the data is entered in the table. The method returns true if the note is recorded otherwise throws an exception.

Boolean recordChild(PersonIdentity parent, PersonIdentity child): In this method I add the id of parent and child to parentchild table where both column are foreign key to id of person table. If the relation already exist in the table the method returns true as the relation is already there in the table, otherwise relation does not exist then the ids are added to the table and method returns true.

Boolean recordPartnering(PersonIdentity partner1, PersonIdentity partner2) and **Boolean recordDissolution(PersonIdentity partner1, PersonIdentity partner2):**

This method checks whether same relation exist between same 2 people, if the relation exist then relationExist variable value is changed to 1. If the value is 0 then only the id's of both person are added to the table. For both condition whether the relationship already exist in table or a new row added to the table method returns true as we can say that relation is established. The id's are stored in partner and dissolution table respectively for partnering relation and dissolution relation.

FileIdentifier addMediaFile(String fileLocation): I passed the location of file as parameter to this method and as it is mentioned that all pictures in media archive has a filename, I split the location through backslash to get the last part of the location and add the obtained parameter as name of file. This name can be updated if needed through recordMediaAttributes. Once fileLocation is passed, I create a new record in filedetails table where I add file id, file location and file name. I have set the id of file in table to auto increment and hence it is updated on its own when new record is created. In file location four backslash needs to be passes then only one is stored as it takes as escape sequence. For example: when passes “D:\\\\used\\\\charm.jpg” then it stores “D:\used\charm.jpg”. This data added in table is used to create a new object of type FileIdentifies and returned. If the passed file location is null or empty, null is returned.

Boolean recordMediaAttributes(FileIdentifier fileIdentifier, Map<String, String> attributes): This method is implemented similar to recordAttributes for a person. Here, the attributes of a file are recorded in the database. For now, I have added pre-defined set of attributes and accept keys similar to that only. The list of these keys is:

- year
- day
- month
- location
- province
- city
- country
- name
- date

any key value other than this will not be stored and a message will be returned to the user mentioning which key was not accepted. All the other attributes which have the listed attribute will be stored. Even if one attribute is not recorded the method returns false as we can say that all attributes are not recorded. If all the attributes are recorded true is returned. Other new attributes for file can also be handled as mentioned for person.

Boolean peopleInMedia(FileIdentifier fileIdentifier, List<PersonIdentity> people): In this method all the people present in list are checked and added with its respective file id in table peopleinfile as a file can have multiple people associated with it, I have created a different table where person_id is foreign key to id of person table and file_id is foreign key to id of file table.

Boolean tagMedia(FileIdentifier, fileIdentifier, String tag): Similar to peopleInMedia this method stores tag for a media/file. Since a file can have multiple tags associated with it. I have created mediatags table that store id of file and the tag assigned to it.

Reporting Methods:

PersonIdentity findPerson(String name): This method returns the object of type PersonIdentity for the name that is passed as parameter. There are 4 conditions that can occur:

i) The name passed is null or empty. In this case null is returned as no name can be null or empty.

ii) The name passed is present only once in the database. In this case the object of that person is returned.

iii) The name passed is present multiple times in the database. It is logically incorrect to send null as the name is present. We can return a set of type PersonIdentity containing all related Objects, but as the return type here is not set and just PersonIdentity, I am returning the person object that is most recently added or found last in database.

iv) The name passed in parameter is not present in the database. In this case null is returned since name is not present the object does not exist.

FileIdentifier findMediaFile(String name): This method returns the object of type FileIdentifier for the name that is passed as parameter. It is implemented similar to PersonIdentity findPerson(String name).

String findName(PersonIdentity id): The Personidentity object is passed as parameter to this method is checked in the database. If it is present in database, the name of the person is returned otherwise null is returned.

String findMediaFile(FileIdentifier fileId): The object of type FileIdentifier is passed in this method. The id of the passed object is checked in database if it is present the file location is fetched and returned as String, otherwise null is returned.

createFamilyTree(): In this method the graph is created based on parent – child table. As the relation to be found is only based on parent and child, I have used parentchild table. I created the graph using adjacency list, as tree could have only one root node and there can be 2 or more roots for family tree depending on number of parents, I chose to implement through graph. First, I take id of each person from person table and put that id as a key in map and an empty arraylist as value. Once the nodes are created, I check in parentchild table and for each row of table I attach the child id in arraylist that is the value of parent id key. I child already present in the value arraylist, I do not add it. Thus the graph representing relations between people is created.

Set<PersonIdentity> descendents(PersonIdentity person, Integer generations): In this method I fetch all the PersonIdentity objects that are descendants of given person who are within specified generations. First I create the graph using createFamilyTree() method. I then search the id of the person in key of the map. If found I check the value of that key which gives me all the children of this person. I then call the method "findDescendents(Set<PersonIdentity> personDescendents, ArrayList<Integer> firstDescendents, int count, int generations)" recursively to get the descendants of all people obtained in the list. This loop breaks when count equals to the generations. Initially, the count is taken as 0 and is incremented with each level of the graph. All the descendants obtained are added in personDescendents set of type PersonIdentity. If the value of generations is 0 empty set is returned.

Set<PersonIdentity> ancestores(PersonIdentity person, Integer generations): In this method, unlike descendants method I search for the id of the person in the value of the adjacency list. For an id found in value of a key, we can say that the key is parent of the person. Thus, I recursively call the "findAncestors(Set<PersonIdentity> personAncestors, int child, int count, int generations)" method to get all the parents till given generation. I repeat the process for all the key values till the count is equal to generations we need to gather information for, and return the set containing the ancestors' objects.

BiologicalRelation findRelation(PersonIdentity person1, PesonIdentity person2): This method is used to find relation between two person in terms of degree of cousinship and degree of removal. If both the person are not related to each other then null will be returned. First of all the ancestors of both the person are found. Out of these common ancestors are found. For a person 0 is stored in its object, moving one level to ancestors, all of the ancestor of that person at that level will have 1 stored in count in its object. Thus count is found for all the ancestors of both the person. Then for the same ancestors we check which one is nearer, person1 or person2 by checking minimum count. Whichever is nearer to that ancestor we store in temp and check whether it is smaller as compared to the previous min found if so then that count is considered as min and that node is considered nearest common ancestor. Once the loop is complete we get the nearest common ancestor and its count. To get the desired output we subtract 1 from this. This new value obtained is degree of cousinship.

If there is no degree of cousinship we can say that 2 person are not related and will not have degree of removal, so we return null. But is the cousinship exist we find the max count of an ancestor for person1 and person2, then we find the difference between these to get degree of removal.

List<String> notesAndReferences(PersonIdentity person): This method is used to return list of notes and references associated with a person in order they were added in the table. To get this order while adding data to the table I am also passing current time and date. So that can be used to order the data. Firstly, I take notes of the person from note_detail table and union it with references of the person obtained from reference_detail along with the person_id, time and date. Once this table containing both notes and reference is obtained I order it by entryDate following by entryTime.

Set<FileIdentifier> findMediaByLocation(String location, String startDate, String endDate): This method returns set of FileIdentifier object that are associated with the given location and fall between start and end date. If both date are null, all file object associated with the location are added in the set. If startDate is null then the media whose date is less than the endDate is added to the set. If endDate is null then media whose date is greater than the startDate is added in the set. If both date are present then the file whose date fall between these date are added in set.

Set<FileIdentifier> findMediaByTag(String tag , String startDate, String endDate): This method returns set of FileIdentifier object similar to findMediaByLocation but the only difference is the files are the ones associated with tag and not location.

List<FileIdentifier> findIndividualsMedia(Set<PersonIdentity> people, String startDate, String endDate): This method returns list of FileIdentifier object of files that are associated with a set of people. The date of these files have same condition as findByLocation method. For getting the files associated with all the people, first I am creating a query by appending id of all the people that are given in set. Once all files associated with people are obtained I am grouping it using file_id and then ordering it by mediaDate i.e. the date of the media in descending order. As we need to add the files with null dates at the end I took descending order. When we get the resultset, for null value I store at the end of the arraylist and for the value with date I store at 0th position in arraylist in order to arrange it chronologically.

List<FileIdentifier> findBiologicalFamilyMedia(PersonIdentity person): Since in this method we need to return Individuals media only but only for the immediate children of a person. So I first call the descendants method to get children till 1 generation. This returns a set of people, which I pass as parameter to fileIndividualsMedia to obtain list of FileIdentifier object.