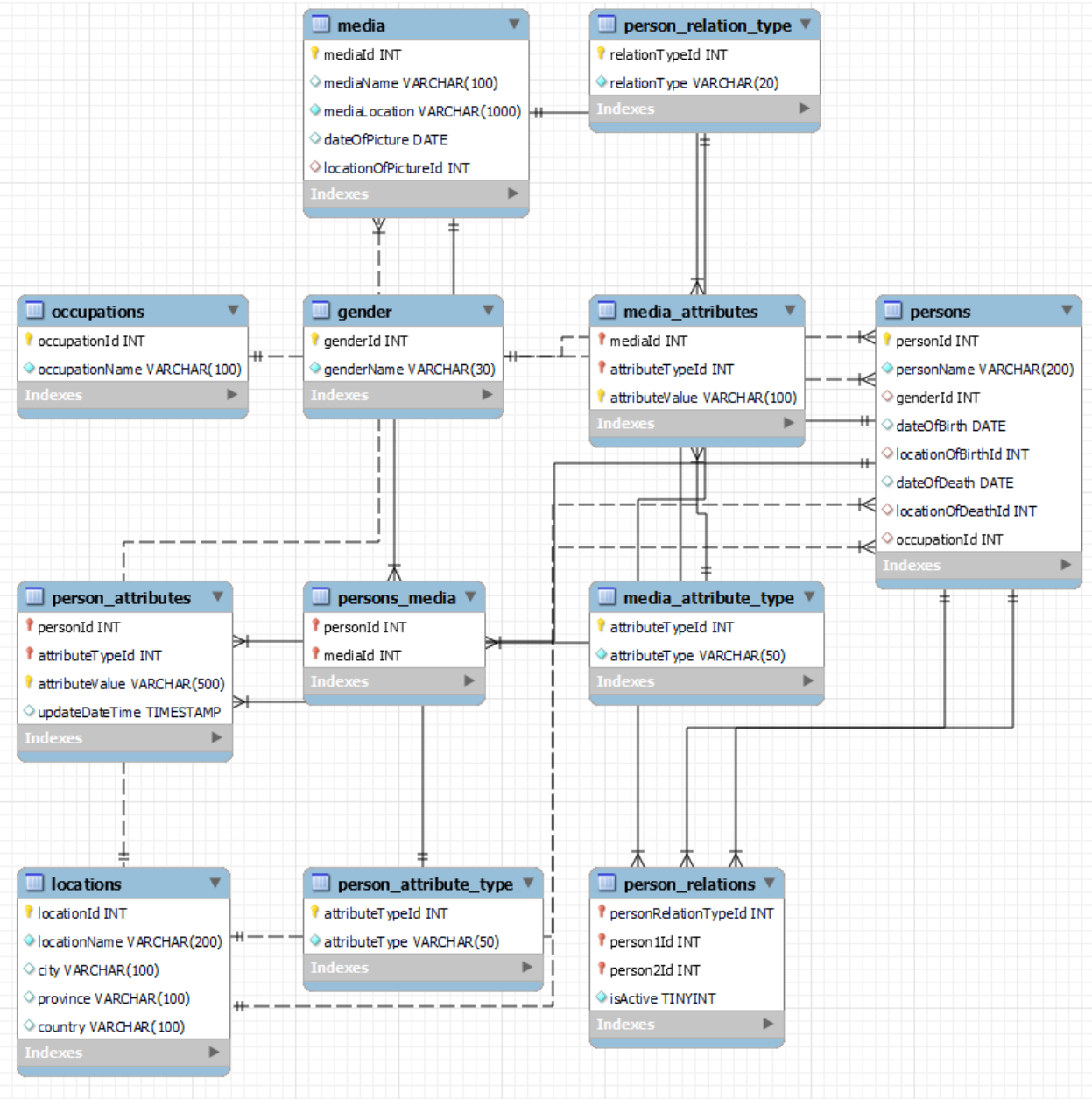
Find your family app

Database design



Tables

1. **locations:** Locations table is an independent table used to store location name, city, province, and country for location of birth, location of death, and location of picture taken. One can reused the already stored location. Referred by persons and media table.

2. **gender:** Gender table is an independent table used to store gender for a person. One can reuse the already stored gender. Referred by person's table.

3. **occupations:** Occupations table is an independent table used to store profession of people. One can reuse the already stored occupation. Referred by person's table.

4. **persons:** Persons table will store person's unique id, name, date of birth, date of death alongside ids for location of birth, location of death, gender, and occupation. The unique id will be used inside person_attributes to store new attributes of a person and inside persons_media to store media of a person.

5. **person_attribute_type:** As a person is ensured to have only name, and some pre-defined attributes. All the new attributes that will be passed for first time inside a Map for a person, will be added inside this table for current and further use. Attribute type Id tells what's the Id of the attribute which will be used to fill against person's attribute. Increment for the attribute id needs to be done manually, to ensure that if certain rows are deleted and inserted once again, attribute id does not change. By default, it has references and notes stored.

6. **person_attributes:** This table will store attribute values with their corresponding attribute type id. A person can have multiple entries inside the table for references and notes attributes but for other attributes, values will be updated.

7. **media_attribute_type:** As a media is ensured to have only media location, and date of picture. All the new attributes that will be passed for first time inside a Map for a media, will be added inside this table for current and further use. Attribute type Id tells what's the Id of the attribute which will be used to fill against media's attribute.

8. **media:** Media table will store media's unique id, name, location, and date of picture. The unique id will be used inside media_attributes to store attributes of a media and inside persons_media to store media of a person.

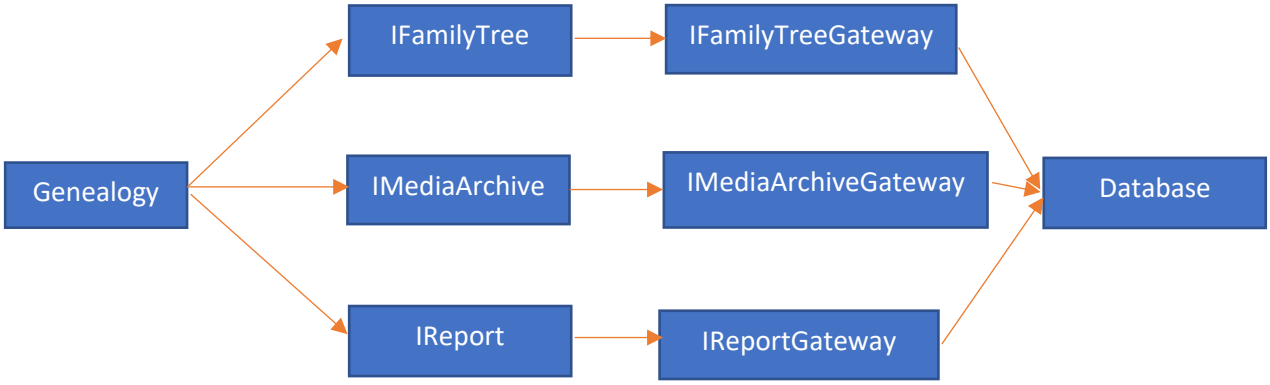
9. **media_attributes:** This table will store attribute values with their corresponding attribute type id. A media can have multiple entries inside the table for different attributes or similar attribute.

10. **persons_media:** This table acts as a storage for many to many relationships between persons and media. A person can have many multiple media, and a media can have multiple persons in them.

11. **person_relation_type:** This table will store relations that can exist between 2 individuals. Currently, this is limited to parent-child and partnership.

12. **person_relations:** This table stores ids of 2 persons, relation between them and if the relation is currently active between them.

Interaction diagram



1. **Wrapper type:** Genealogy type here acts as a wrapper class for providing find your family overall functionality. It combines features of family tree, media archive management and reporting.
   Input validations for the incoming user-defined parameters are done here, to filter out the invalid requests from the user. Once the parameters are validated, filtered requests are converted into internal representation like PersonIdentityInternal and FileIdentifierInternal, to have more control over the information.
   After, each individual operation is done at the business layer, output from the operation is completed to external type. External type ensures 2 things, one is to have representation as required by the user and other is to have only getters, to ensure data is not manipulated and integrity is maintained.
   Every method in Genealogy will throw IOException, as each operation has some work to be done inside database, and if any interaction fails, IOException will let the user know something has failed while saving/retrieving information.

2. **Business layer:** It acts as a second layer of the application where all the manipulations are done on the information coming from the user or going out to the user according to the business rules. e.g., Separating media location into media name and media location to fit database requirements and take out the media with name when asked.

   Business layer is further segregated into family tree, media archive and reporting features.

   1. **Family Tree:**
      IFamilyTree.java: Interface representing a contract for managing family tree. It extends from further 2 interfaces IRecordPersonInfo (add a person, its attributes and record references, notes) and IRecordPersonRelationship (record relationships like parent-child, partnering and dissolution).

FamilyTree.java: Implementation for IFamliyTree that acts as a wrapper for managing the family tree. All the methods called inside this class will save person and its attributes, relationship information to the database.

Operations with their features and limitations

PersonIdentity addPerson(String name)

- Multiple persons with the same name can be added. Each person will have its own entry inside the database.
- Name with only numbers, alphanumeric can also be added.

Boolean recordAttributes(PersonIdentity person, Map<String, String> attributes)

- A person can change his/her name.
- Gender, location of birth, location of death and occupation have their own table. Once a value is stored inside in any of the table, that can be reused if same gender, location, or occupation is asked for.
- For locations, if the given location directly matches with the location name column in the locations table, then, only we reuse it otherwise new location gets created.
- It can record new attributes as well. If a new attribute is recorded multiple times. It will update the previous value of the attribute with new value. These new attributes will be returned in map with the name○ additional attributes.
- If notes and references are recorded, they will have multiple entries. Notes and references are returned as individual fields.
- Notes and references can be recorded in record attributes apart from their individual methods.
- If there are 10 attributes, and any one of the attributes is failed. No attribute will be recorded.
- If birth date is greater than date of death, it will not be recorded

- **Limitations:**
  1. Currently if a location comes for an update, and its city, country or province is updated. It will get updated for those persons and media as well. Later, we will cater to this requirement by adding a new location in this case and assign it to current person only.

Boolean recordReference(PersonIdentity person, String reference)

- Multiple references can be added for a person.
- Same reference for a person can’t be added twice.

- **Limitations:**

1. Currently, reference is not verified for its location may be its file path or website lin. Later, it will be updated with validations on reference links as well.

Boolean recordNote(PersonIdentity person, String note)

- Multiple notes can be added for a person.
- Same notes for a person can't be added twice.

Boolean recordChild (PersonIdentity parent, PersonIdentity child)

- When there is already an inverse relation between child and parent. It returns false.
- When a parent-child relation is added, it updates the persons map inside family tree with parent having this child, and child having this parent.
- If the parent has a partner and it does not have current child as added children, update both partner and child with parent-child relationship.
- After all the changes have been reflected inside the database then, only persons map gets updated. No changes are reflected if anu of the change fails inside the database.
- It will return false if there is already a relationship where child is the parent and parent is a child.
- If the parent-child already have a partnership or dissolution relationship, it will not add parent-child relationship and returns false.

Boolean recordPartnering (PersonIdentity partner1, PersonIdentity partner2)

- Records a partnership from partner 1 and partner 2 and vice-versa.
- When there is already a parent-child relationship between partners, it returns false.
- When there is already a dissolution relationship between partners, it updates with partnership.
- If there is already an active partnership with another person, returns false.

Boolean recordDissolution (PersonIdentity partner1, PersonIdentity partner2)

- If there is no partnership between partner 1 and partner 2, we cannot add dissolution first. Partnership should be added first to have a dissolution.
- If there is a partner relationship with any other person. It returns false and do not update dissolution.

2. Media Archive:

IMediaArchive.java: Interface representing a contract for managing media files. It has methods to add media, record their attributes, add people visible inside media, and tagging inside the media.

MediaArchive.java: Implementation for IMediaArchive that acts as a wrapper for managing the media. All the methods called inside this class will save media and related information to the database.

Operations with their features and limitations

FileIdentifier addMediaFile(String fileLocation)

- Adds a media file to the external source with file location segregated into file location and file name with extension.
- If a file with same location is trying to get add, existing file identifier will be returned.
- Files with same name but different location can be added.

Boolean recordMediaAttributes(FileIdentifier fileIdentifier, Map<String, String> attributes)

- A file name and location can be updated.
- For locations, if the given location directly matches with the location name column in the locations table, then, only we reuse it otherwise new location gets created.
- It can record new attributes as well. If a new attribute is recorded multiple times. It will update the previous value of the attribute with new value. These new attributes will be returned in map with the name's additional attributes.
- If tags are recorded, they will have multiple entries.
- Tags can be recorded in record attributes apart from its individual method.
- If there are 10 attributes, and any one of the attributes is failed. No attribute will be recorded.

Boolean peopleInMedia(FileIdentifier fileIdentifier, List<PersonIdentity> people)

- If recording for any of the person in people parameters fails, no media and its person will be recorded to maintain integrity of the system. This ensures user is not under the faulty impression things are updated when they are not.

Boolean tagMedia(FileIdentifier fileIdentifier, String tag)

- If same tag is trying to get recorded for a file, it returns true.
- Multiple tags for the same file can be added.
- Same tag can be used for multiple files.

3. Reporting:

PersonIdentity findPerson(String name)

- If it has many persons with the same name, random person will be reported.
- Currently it returns person name, it's pre-defined as well as new attributes, notes, and references, parents, children, siblings and partner, media files.

FileIdentifier findMediaFile(String name)

- If it has many media with the same name, random file will be reported.
- Currently it returns media name, it's pre-defined as well as new attributes, tags and persons that are in this media.

String findName(PersonIdentity id)

- Find the name for a given person id. There can be only 1 unique id per person.

String findMediaFile(FileIdentifier fileId)

- Find the name for a given media id. There can be only 1 unique id per media.

BiologicalRelation findRelation(PersonIdentity person1, PesonIdentity person2)

- Finds the relation between 2 persons by finding the lowest common ancestor to them.
- Biological relation contains counsinship and removal as -1 when no common ancestor is found.
- When siblings are there counsinship and removal is 0.

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

- Gives the descendants of the current person using breadth first search algorithm with a queue.
- Returns all the direct descendants of the current person.

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

- Gives the ascendants of the current person using breadth first search algorithm with a queue.
- Returns all the direct ancestors of the current person.

List<String> notesAndReferences(PersonIdentity person)

- Returns notes and references in the order which they were added for a person.

Set<FileIdentifier> findMediaByTag(String tag , String startDate, String endDate)

- Return distinct files for a given tag within given time frame.

Set<FileIdentifier> findMediaByLocation(String location, String startDate, String endDate)

- Return distinct files for a given location within given time frame.
- If the location with some extra information is passed. No information will be returned.

List<FileIdentifier> findIndividualsMedia(Set<PersonIdentity> people, String startDate, String endDate)

- Returns distinct files for all those persons who exist in the system according to the date it was taken and no date files at the end. If 2 files exist with same date, return it according to the alphabetical order.
- If date range is not passed, returns all the files without any date restriction.
- If any of the person does not exists in the system. Still returns information for other people.

List<FileIdentifier> findBiologicalFamilyMedia(PersonIdentity person)

- Returns distinct files among all the files for immediate children of the person are returned according to the date it was taken and no date files at the end. If 2 files exist with same date, return it according to the alphabetical order.
- If some children do not exist, return files for others.

3. **Database layer:** It acts as a final layer of the application where all the interactions are done with the external source.

   Database layer is further segregated into family tree, media archive and reporting features. All these features are segregated into Interfaces such that if requirements come for new external source, these interfaces can be implemented to interact with data.

4. **Common layer:** It contains features that can be used by any of the layers listed above. It includes enums, file handing and mappers.

   Enums are present for person attributes, media attributes and relationship type. Person and media attributes enums are used to store information for the pre-defined attributes.

Mappers map person identity and file identifier types from internal type to external and vice-versa. Maintaining different types for internal and external representation ensures independent development of the features where external type does not see changes if any of the changes are done to internal types.

5. Models/Data Structures:

Domain models:

Person Identity: External type used to represent person information required at the user end. It consists of information required by the user and does not consist any of the information user should not have control to.

Person Identity Internal: Internal type used to store person information required internally in the application. It has the flexibility of updating the information as required inside the application. All the information is parsed into internal type before it comes in and parsed to external type before it goes out.

Person Identity Lite: Lightweight type for storing person ids. These person ids are used to store partner, parents, and children ids.

PersonRelations.java: Type for storing person relations including partner, parents, and children.

File Identifier: External type used to represent media information required at the user end. It consists of only getters, so that the data cannot get tempered while interaction It consists of information required by the user and does not consist any of the information user should not have control to.

File Identifier Internal: Internal type used to store media information required internally in the application. It has the flexibility of updating the information as required inside the application. All the information is parsed into internal type before it comes in and parsed to external type before it goes out.

Location: Type for storing location related information inside the database. It stores the location information in the form of location name, city, province, and country.

Attribute: Type for storing new attribute type id and the value stored for the person or media inside the database.

DateRecord: Type for storing date related information for the internal use. It helps to store date when either of the month, day or year is missing.

AttributeType: Type for storing new attributes added inside the database apart from pre-defined attributes,

PersonRelationshipLite: Type for storing person ids of 2 people and the type of relationship between them.

BiologicalRelation: Type for storing the cousinship and degree of removal between 2 persons.

Key Algorithms

1. **Recording parent-child and caching it:**
    - While recording the parent-child relationship, 2 relationship records are created initially. 1 contains the relationship between the passed parent and child.
    - Then, we check if we have any present partner for the passed parent, if we have 1 more relationship record between this partner and child is added.
    - Once, we update everything to the database, parent, child, and partner are updated with new information inside the persons map that acts as a local cache.

2. **Recording partnership and caching it:**
    - While recoding the partnership, 2 relationships records are created. 1 is between from partner 1 to partner 2 and other is from partner 2 and partner 1, with relation type set to partnership and status set as Active inside the database.
    - Once, the relationship records are updated in the database, partner 1 and partner 2 are updated inside the local cache with the updated information.

3. **Recording dissolution and caching it:**
    - While recording the dissolution, 2 relationship records that were recorded earlier between partner 1 and partner 2 are updated and status is et to not active inside the database.
    - Once, the relationship records are updated in the database, partner 1 and partner 2 are updated inside the local cache with the updated information.

4. **How is persons cache used in getting descendants, ancestors, biological relation, and other information?**
   While recording parent-child, partnership and dissolution, persons map gets updated as soon as relationships are updated inside the database. Now, while getting descendants, ancestor, or anything else if we required any information related to person relations, we check the person's map. If we have its entry inside the persons map, then that is used otherwise we use get_personrelations stored procedure to get all the relations of the person and update it in the cache as well.

5. **Finding biological relation:**

Backtracking for finding the lowest common ancestor is used.

- For finding the biological relation, we first find root of the person 1 and person 2 and it to set of roots.
- While finding the root of first person, we add all the ancestors of that person in the nodes covered set.
- Now while finding the root of person 2, we move upwards, and check if we have found the common ancestor. If we have, this gives us the lowest common ancestor.
- Next, we find the positions of person 1 and person 2 when started from the root.
- Using the positions we find the cousinship by using Min(person 1, person 2 position) – 1 and then, degree of removal by |person 1 – person 2 position|.

6. **Finding ancestors and descendants:**
    - Breadth first search is used with queue for finding descendants and ancestors.
    - We add the given person to the queue, deque it and finds its parents(ancestors)/children(descendants) and enqueue it.
    - We take the first person added to the queue and find its parents(ancestors)/children(descendants) and enqueue them and continue this until we have reached the given number of generations.

7. **Transaction management:**
   Transactions are managed in 2 ways in the application.
   a. While recording information, if any of the attribute, person or media fails to record, nothing gets recorded. This ensures user is not under the impression that information is saved when some information fails to get impression on the database.
   b. While reporting information, if some of the information is not present like person inside findIndividualsMedia method, still media for rest of the persons are reported.

Assumptions

1. Date passed in the methods should be of the format “dd/MM/yyyy”.
2. While saving media, media location is passed. Internally it’s segregated into media name and media location. It is assumed that when media is requested with media name, it includes extension as well. e.g.: TravelAssitant.java.
3. Location information will be entered in the form of location name, city, province, and country. With each of them having their own individual attributes.
4. Date information will be entered in the form of year, month, and day. With each of them having their own individual attributes.
5. Pre-defined attributes for Person Identity to be passed inside map of attributes:
   PERSONNAME,
   GENDER,
   DAYOFBIRTH,
   MONTHOFBIRTH,
   YEAROFBIRTH,
   LOCATIONOFBIRTH,

CITYOFBIRTH,
PROVINCEOFBIRTH,
COUNTRYOFBIRTH,
DAYOFDEATH,
MONTHOFDEATH,
YEAROFDEATH,
LOCATIONOFDEATH,
CITYOFDEATH,
PROVINCEOFDEATH,
COUNTRYOFDEATH,
OCCUPATION,
REFERENCES,
NOTES

6. Pre-defined attributes for File-Identifier to be passed inside map of attributes:

MEDIANAME,
MEDIALOCATION,
DAYOFPICTURE,
MONTHOFPICTURE,
YEAROFPICTURE,
LOCATIONOFPICTURE,
CITYOFPICTURE,
PROVINCEOFPICTURE,
COUNTRYOFPICTURE,
TAG

Prerequisites to run the application

1. Run the database creation script from docs/sqlscripts folder.
2. Update the database name, login id and password inside the config.properties