CSCI 3901: Course Project

Milestone 3

Data Structures:

Graph

The system uses Graph data structure for generating and storing the Family Tree Structure. As this data must remain persistent even after program terminates, this structure is defined in the Database Schema itself.

Key Algorithm and Strategy:

The below figure shows the Entity-Relationship diagram for storing the Family Tree information and their corresponding Media files:

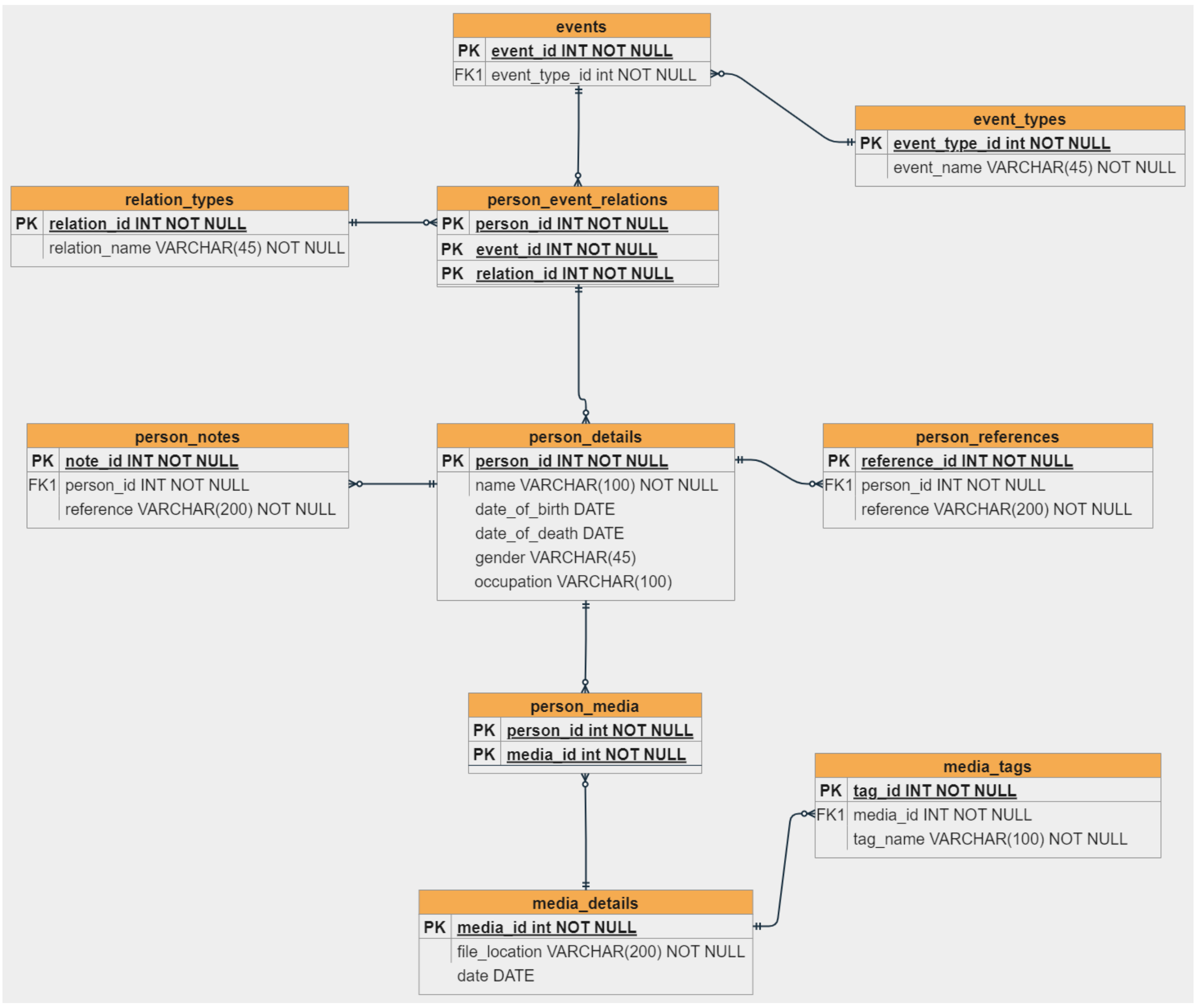

events
PK event_id INT NOT NULL
FK1 event_type_id int NOT NULL
event_types
PK event_type_id int NOT NULL
event_name VARCHAR(45) NOT NULL
relation_types
PK relation_id INT NOT NULL
relation_name VARCHAR(45) NOT NULL
person_event_relations
PK person_id INT NOT NULL
PK event_id INT NOT NULL
PK relation_id INT NOT NULL
person_notes
PK note_id INT NOT NULL
FK1 person_id INT NOT NULL
reference VARCHAR(200) NOT NULL
person_details
PK person_id INT NOT NULL
name VARCHAR(100) NOT NULL
date_of_birth DATE
date_of_death DATE
gender VARCHAR(45)
occupation VARCHAR(100)
person_references
PK reference_id INT NOT NULL
FK1 person_id INT NOT NULL
reference VARCHAR(200) NOT NULL
person_media
PK person_id int NOT NULL
PK media_id int NOT NULL
media_tags
PK tag_id INT NOT NULL
FK1 media_id INT NOT NULL
tag_name VARCHAR(100) NOT NULL
media_details
PK media_id int NOT NULL
file_location VARCHAR(200) NOT NULL
date DATE

Figure 1. Entity-Relationship Diagram

Constructing the Family Tree:

As shown in above Fig. 1, four tables are defined for maintaining the Family Tree information, **events**, **event_types**, **relation_types** and **person_event_relations**, adapted, and modified from [1]. The following points describe the purpose of each table:

- **event_types** table defines the life events that occur for an individual, for example, “marriage”, “divorce” and “birth”, and can be extended in the future to have events such as “death” as well.
- **events** table keeps a record of all the life events that occur for any individual. The type of event is referred from the **event_types** table.
- **relation_types** table defines the role or relation that an individual can have. For example, “mother”, “father”, “child” are the types of relation a person can have.
- **person_event_relations** table combines the information from the above-mentioned tables to define the role of a particular person in an event. The following is the purpose of each column:
    - **person_id** identifies the person involved.
    - **event_id** identifies the event and its type that occurred in the life of that person.
    - **relation_id** identifies the relation/role of the person in that event.

Consider the following example, Person C is the child of Person A and Person B. To store this relationship in the database, we perform the following write operations:

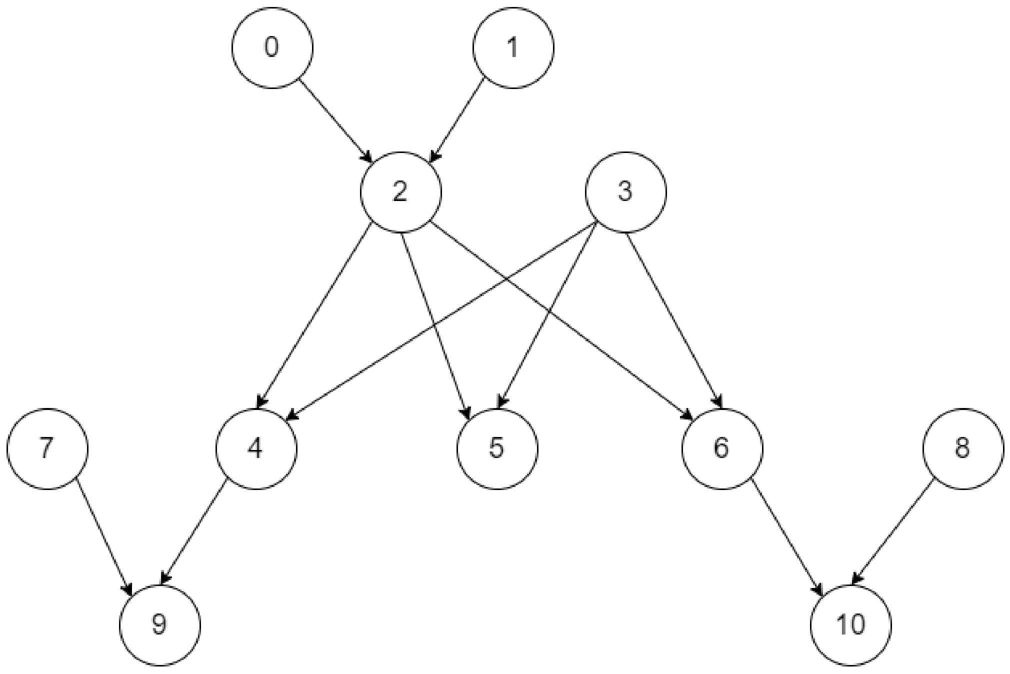
- Record a new event in **events** table, with **event_type_id** of type **“birth”**.
- Record a new entry in the **person_event_relations** table with **person_id** of Person C, **event_id** of above recorded event and **relation_id** of type **“child”**.
- Similarly, for the same **event_id**, we record 2 more entries in **person_event_relations** table with **person_id** of Person A and Person B and **relation_id** of type **“mother”** and **“father”**, respectively.

Therefore, to find the father of Person C, we get the **event_id** of his birth and for the same **event_id** we find the person with the role of **father**. Similarly, we can iteratively go up in the Family Tree to find the other ancestors as well.

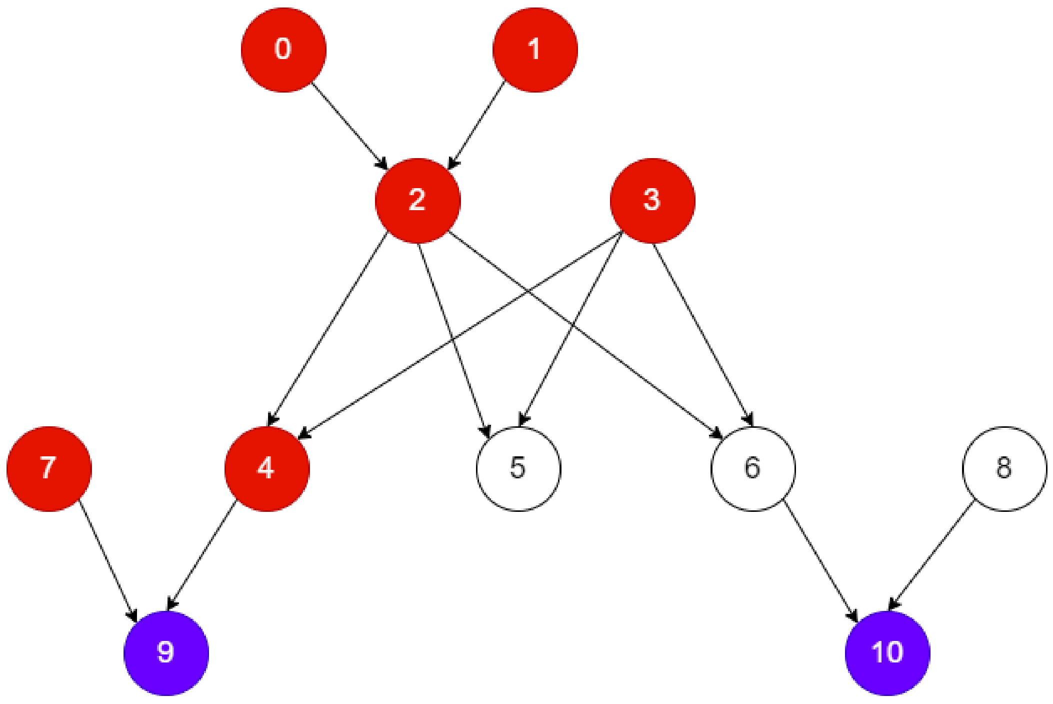
Finding the relation between a pair of individuals:

A Family Tree is usually represented as a **Directed Acyclic Graph (DAG)**. As defined in the assignment text, to find the relationship between a pair of individuals, we must find their **Lowest Common Ancestor** first. To find this, the system will use the following approach:

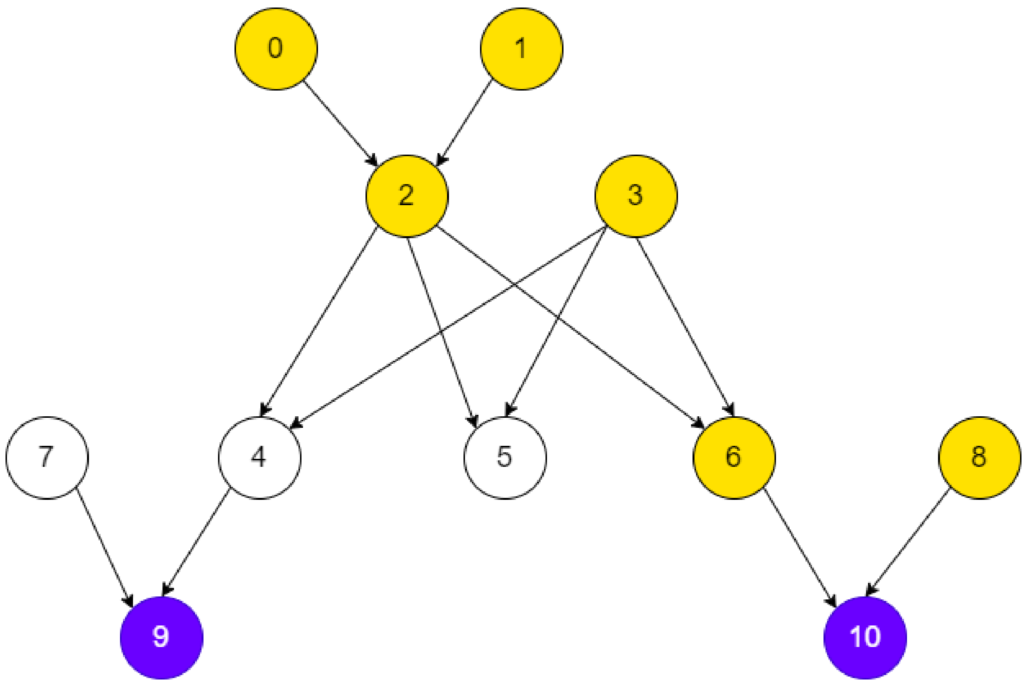
1. Consider the given graph. We want to find the Lowest Common Ancestor of Node 9 and 10.



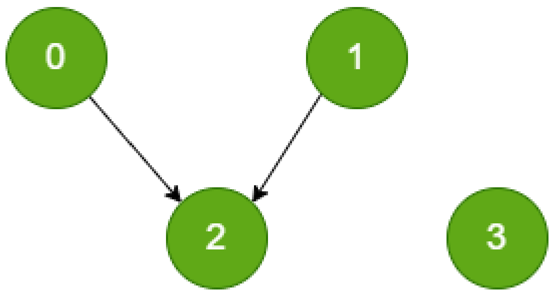
2. We will find all the parents (and their parents) till root for Node 9 using BFS or DFS. These nodes will be marked in Red colour.



3. Now, we will find all the parents of Node 10. They will be marked in Yellow colour.



4. Now, the intersection of Red and Yellow will form a sub-graph. The nodes with **out-degree of 0 (no outgoing edges)** will be the Lowest Common Ancestors. Hence, Node 2 and 3 are the LCAs of Node 9 and 10.



5. Finally, based on the formula provided in assignment text, the smaller of the depths of Node 9 and 10, minus 1 will give the **cousinship**, and the difference of these depth will give the **level of separation**.

Finding the corresponding images of people:

As per the Database Schema mentioned above, there is a **many-to-many relationship** between the people and media files. This relation is shown as a separate table, **person_media** and will store the ID of a person and the ID of their corresponding image. This way the system can fetch the image corresponding to any individual. Also, images can be filtered based on date by performing join operation with the **media_details** table.

Code Design

PersonIdentity
Defines the data for an individual.

FileIdentifier
Defines the data for a media file.

BiologicalRelation
Defines the data and methods for specifying different relationships between a pair of individuals.

Reporting
Defines the data and methods for fetching various information from the database such as searching a person or media file by filters, relation between pair of individuals, and ancestors or descendants for an individual along with their corresponding images.

Genealogy
Defines the data and methods for adding a person or media and recording their attributes in the database. Also contains methods for connecting the people with their corresponding media files.

Additional Tests

- Check whether connection has been established, i.e., Connection object is not null before proceeding with any database operation.
- Check if the Connection is closed at the end of every method that performs a database operation.
- For any method that returns an object, if that object is not found, then it should return null.
- For findPerson() method, even if multiple people with same name exist, then the method should return only one person.

References

[1] E. Pills and D. Blows, “MySQL store relationship (family) Tree,” *Stack Overflow*, 24-Apr-2011. [Online]. Available: https://stackoverflow.com/questions/5773938/mysql-store-relationship-family-tree. [Accessed: 20-Nov-2021].