CS 351 Scott Barlowe

Project 1

LINKED-LIST DATABASE

INTRODUCTION

Databases are useful for storing and organizing large amounts of data. Databases are made of tables containing related information. Tables are made of records (rows) and fields (columns). Figure 1 is a **CollegePersonnel** database with a **Faculty** table and an **Administrator** table. Each table has three records each with a Name, ID, Phone, Division, and Years field.

CollegePersonnel

Faculty Table

Name	ID	Phone	Division	Years
Smith, John	246823	8287228074	1	23
Jones, Bob	548637	8289862451	2	2
Que, Susie	482163	7042683265	6	23

Administrator Table

Name	ID	Phone	Division	Years
Thompson, Cindy	754329	8287528362	3	9
Coolie, Madison	958423	8288242123	2	3
Que, Susie	482163	7042683265	6	23

Figure 1

We can perform operations on one or several tables to extract and then view entries that meet desired criteria. Common operations include SELECT, JOIN, UNION, INTERSECT, and many more. Often, a column assured to have a unique value for each record per table is used to differentiate among records. This is referred to as a ***key***. We will be defining our own operations, but I encourage you to research the formal, accepted definition of these operations on your own.

Your assignment is to implement a small database and several database operations using linked lists. The database consists of two linked lists, each representing a table.

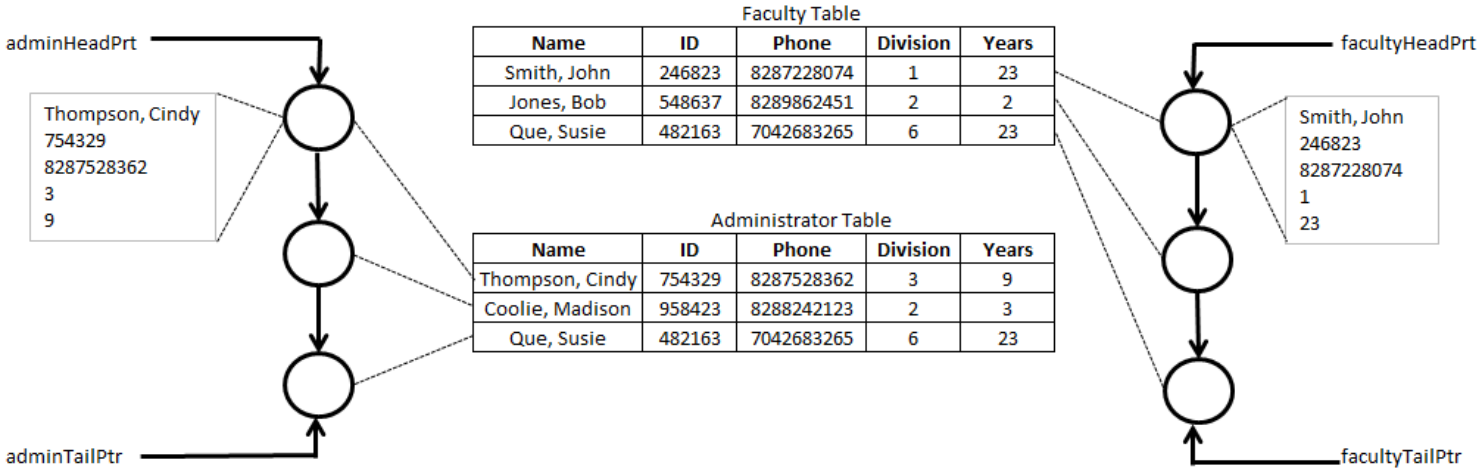

Figure 2

PRO RUCTURE

The program structure is shown in Figure 2. Each list is made of **PersonNode** members representing a row in the table. Each **PersonNode** has fields for **name**, **ID**, **phone**, **division**, and **years** attributes (columns). You should have a **MyDatabase** class that contains 1) the pointers to the two linked lists that hold individual tables and 2) methods for the table operations listed below. **adminHeadPrt** points to the front of a list holding administrators and **adminTailPrt** points to the end of that list. **facultyHeadPrt** points to the front of a list holding faculty and **facultyTailPrt** points to the end of that list. Additionally, you should have methods for creating the list, populating the list, and any other methods that may be suitable. You must use the **printList** method to print all results. Finally, a **DbDriver** class should create a **MyDatabase** object, create the list, populate the list, and repeatedly prompt the user for an operation to execute. YOU MAY NOT USE THE JCF (OR ANY OTHER PREDEFINED LIST STRUCTURES) FOR LIST CREATION OR MANAGEMENT.

PersonNode class

PersonNode next;

String **ID**; // Six digit employee ID number

String **name**; // Employee name having the form "Last name,First name" (no space after the comma)

String **phone**; // Ten digit phone number

String **division**; // Academic or administrative division

String **years**; // Years of service

MyDatabase class

PersonNode **adminHeadPrt**; // Points to front of administrative list

PersonNode **adminTailPtr**; // Points to end of administrative list

PersonNode **facultyHeadPrt**; // Points to front of faculty list

PersonNode **facultyTailPrt**; // Points to end of faculty list

public PersonNode[] insert(int table, String id, String name, String phone, String division, String years)

Return a head and tail pointer to a new list in the specified table where a new node having the above information is added to the end of the list. Used only in building the initial faculty and administrator lists.

public PersonNode [] select(int table, String attribute, String value)

Return a head and tail pointer to a new list comprised of nodes having *value* for a given attribute from a single table (list).

public PersonNode [] intersect(String attribute, String value)

Return a head and tail pointer to a new list comprised of nodes having *value* for a given attribute in both tables (lists). No duplicates are allowed in the result. (Use *ID* as the unique identifier.)

public PersonNode [] difference(int tableA, int tableB)

Return a head and tail pointer to a new list comprised of nodes found in table (list) A but not table (list) B. (Use *ID* as the unique identifier.)

public PersonNode [] union()

Return a head and tail pointer to a new list comprised of all nodes found in both tables. No duplicates are allowed in the result. (Use *ID* as the unique identifier.)

public PersonNode [] remove(int table, String id)

Return a head and tail pointer to a new list with the node with *id* in the table listed removed. If a node with id is not found, the list should be unchanged.

public PersonNode [] getVeteran(int table)

Return a head and tail pointer to a new list comprised of all the node(s) having the maximum value for the years attribute in the given table.

public PersonNode [] getRookie(int table)

Return a head and tail pointer to a new list comprised of all the node(s) having the minimum value for the years attribute in the given table.

public void printList(int table)

Prints the entire list corresponding to *table* in the form below.

Name	ID	Phone	Division	Years	Classify
Jones, Bob	548637	8289862451	2	2	FAC

INPUT

The data in each file is separated by whitespace and you can assume that there is no missing data. Your program should read in data from **faculty.txt** and **admin.txt** and load the appropriate table. The form of the text files is shown below. An example **faculty.txt** and **admin.txt** is posted in Blackboard.

Name ID Phone Division Years

OUTPUT

Your program should prompt the user to enter an integer corresponding to one of the database operations followed by the necessary arguments. The operation-to-integer mapping is **select → 1**, **intersect → 2**, **difference → 3**, **union → 4**, **remove → 5, getVeteran → 6, getRookie → 7, print → 8**. Entering **0** for the operation should end the program. The table-to-integer mapping is **faculty table → 1** and **administrator table → 2**.

Results from **select**, **intersect**, **difference**, **union, getVeteran, and getRookie** should be printed as part of the operation.

The **remove** operation should print the specified list minus the removed node.

The **print** operation should print out a single table input by the user.

When printing an entry, you should output if the entry can be found in the faculty table, administrator table, or both as shown in the CLASSIFY column below. For example, suppose we have the faculty and administrator tables shown above. A program session would look like:

$ javac *.java
$ java DbDriver
$ Enter operation
$ 1 1 years 23

NAME	ID	PHONE	DIVISION	YEARS	CLASSIFY
Smith,John	246823	8287228074	1	23	FAC
Que,Susie	482163	7042683265	6	23	FAC, ADM

$ Enter operation
$ 2 ID 482163

NAME	ID	PHONE	DIVISION	YEARS	CLASSIFY
Que,Susie	482163	7042683265	6	23	FAC, ADM

$ Enter operation
$ 3 1 2

NAME	ID	PHONE	DIVISION	YEARS	CLASSIFY
Smith,John	246823	8287228074	1	23	FAC
Jones,Bob	548637	8289862451	2	2	FAC

$ Enter operation
$ 3 2 1

NAME	ID	PHONE	DIVISION	YEARS	CLASSIFY
Thompson,Cindy	754329	8287528362	3	9	ADM
Coolie,Madison	958423	8288242123	2	3	ADM

$ Enter operation
$ 4

NAME	ID	PHONE	DIVISION	YEARS	CLASSIFY
Smith,John	246823	8287228074	1	23	FAC
Jones,Bob	548637	8289862451	2	2	FAC
Que,Susie	482163	7042683265	6	23	FAC, ADM
Thompson,Cindy	754329	8287528362	3	9	ADM
Coolie,Madison	958423	8288242123	2	3	ADM

$ Enter operation
$ 6 2

NAME	ID	PHONE	DIVISION	YEARS	CLASSIFY
Que,Susie	482163	7042683265	6	23	FAC, ADM

$ Enter operation
$ 7 1

NAME	ID	PHONE	DIVISION	YEARS	CLASSIFY
Jones,Bob	548637	8289862451	2	2	FAC

$ Enter operation
$ 5 1 482163

NAME	ID	PHONE	DIVISION	YEARS	CLASSIFY
Smith,John	246823	8287228074	1	23	FAC
Jones,Bob	548637	8289862451	2	2	FAC

(Continued on next page)

$Enter operation
$ 8 2

NAME	ID	PHONE	DIVISION	YEARS	CLASSIFY
Thompson,Cindy	754329	8287528362	3	9	ADM
Coolie,Madison	958423	8288242123	2	3	ADM
Que,Susie	482163	7042683265	6	23	ADM

(Note that Susie Que has been removed from the FAC table and now only has the ADM classification.)

$Enter operation
$ 0
$ BYE!

Administrative

1. The project is due at midnight on Monday, the 19th of September.
2. ***The score is 0 if the program does not compile on agora.***
3. The project can be turned in late. Five points will be taken off for every date late (including weekends and holidays) up to a maximum of 25 late points. Programs exceeding 5 days late will not be accepted.
4. Linked list data structures implemented as part of a language or a predefined package are not allowed (e.g. Java Collection Framework). If you have questions regarding what is allowed, please see your instructor.
5. You may complete this assignment in a language other than Java. If you choose to use another language, please see your instructor for submission instructions.
6. You may work in teams of one or two students from the class. You may talk with other students in the class about the concepts involved in doing the project, but anything involving actual code needs to just involve your team. In other words, you cannot show your team's code to students outside of your team and you cannot look at the code of another team.
7. Your solution must follow the course coding standards including

   (a) no lines longer than 80 characters
   (b) no magic numbers; use named constants
   (c) javadoc class headers, fields, constructors, and methods
   (d) use // comments to document code in method bodies.

8. Submit your files as a tar file via **handin** on agora. If your tar file is called project1.tar.gz, then the command will be:

   handin.351.1 1 project1.tar.gz