**Quin Reidy**

**3/12/2019**

**CSC 352**

**HW6**

**CODE FOR THE DATABASE:**

**package** hw6;

**import** java.io.File;

**import** java.io.IOException;

**import** java.sql.Connection;

**import** java.sql.Date;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** java.util.Scanner;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** HW6Final {

**static** Connection *conn* = **null**;

**static** Statement *stmt* = **null**;

**public** **static** **void** main(String [] args){

*connectToDatabase*();

*createAgeTable*();

*createJobTable*();

*createMovieDetails*();

*createMoviecategory*();

*createUserDetails*();

*createRatings*();

//Populates the two tables based on the values in the readme attached to the dataset

*populateAgeTable*();

*populateJobTable*();

//Reads the file, inserts into movie details

*populateMovieDetails*();

//Reads the file, inserts into user details

*populateUserDetails*();

//reads the file, inserts into movie categories

*populateMovieCategorys*();

//Reads the file, inserts into ratings

*populateRatings*();

//Close the connection

**try**{

System.***out***.println("Closing the database connection.");

*stmt*.close();

*conn*.close();

}

**catch**(SQLException se){

System.***out***.println("Something went horribly wrong: " + se);

System.*exit*(1);

}

}

//helper method to avoid bloating the main

//attempts to load the jdbc driver, then connect to the depaul database

//don't enjoy that the username and password has to be hardcoded, but oh well

**public** **static** **void** connectToDatabase(){

//Load the JDBC driver

System.***out***.println ("Attemping to load JDBC driver...");

**try** {

Class.*forName*("oracle.jdbc.OracleDriver");

System.***out***.println("Successfully loaded the JDBC driver.");

}

**catch** (ClassNotFoundException e){

System.***out***.println("Something went horribly wrong: " + e);

System.*exit*(1);

}

//Attempt to connect to the database

**try**{

System.***out***.println("Attempting to connect to the DePaul database...");

*conn* = DriverManager.*getConnection*("jdbc:oracle:thin:@acadoradbprd01.dpu.depaul.edu:1521:ACADPRD0",

"qreidy", "cdm1355894");

System.***out***.println("Successfully connected to the DePaul database.");

*stmt*=*conn*.createStatement();

}

**catch** (SQLException se){

System.***out***.println("Something went horribly wrong: " + se);

System.*exit*(1);

}

}

//creates the age\_key table schema

**public** **static** **void** createAgeTable() {

//attempts to drop the table, and does nothing if the table doesn't exist

**try**{

System.***out***.println("Dropping the age\_key table...");

String dropString = "DROP TABLE age\_key CASCADE CONSTRAINTS";

*stmt*.executeUpdate(dropString);

System.***out***.println("Successfully dropped the age\_key table.");

}

**catch**(SQLException se) {/\*Do nothing\*/};

//creates the age\_key table

**try**{

//create the table

System.***out***.println("Creating the age\_key table...");

String createString = "CREATE TABLE age\_key \r\n" +

" (age\_code NUMBER (2), \r\n" +

" age\_translate VARCHAR (10), \r\n" +

" PRIMARY KEY (age\_code, age\_translate)\r\n" +

" ) ";

*stmt*.executeUpdate(createString);

System.***out***.println ("Successfully created the age\_key table.");

}

**catch** (SQLException se) {

System.***out***.println("Something went horribly wrong: " + se);

System.*exit*(1);

}

}

//creates the job\_key table schema

**public** **static** **void** createJobTable() {

//attempts to drop the table, and does nothing if the table doesn't exist

**try** {

System.***out***.println("Dropping the job\_key table...");

String dropString = "DROP TABLE job\_key CASCADE CONSTRAINTS";

*stmt*.executeUpdate(dropString);

System.***out***.println("Successfully dropped the job\_key table.");

}

**catch** (SQLException se) {/\*Do nothing\*/};

//creates the job\_key table

**try** {

System.***out***.println ("Creating the job\_key table...");

String createString = "CREATE TABLE job\_key (\r\n" +

" job\_code NUMBER (2),\r\n" +

" job\_translate VARCHAR (30),\r\n" +

" \r\n" +

" PRIMARY KEY (job\_code))";

*stmt*.executeUpdate(createString);

System.***out***.println("Successfully created the job\_key table.");

}

**catch** (SQLException se) {

System.***out***.println ("Something went horribly wrong: " + se);

System.*exit*(1);

}

}

//creates the movie\_details table schema

//AND THAT IS IS- values will be read in later

**public** **static** **void** createMovieDetails() {

//attempts to drop the table, and does nothing if the table doesn't exist

**try**{

System.***out***.println("Dropping the movie\_details table...");

String dropString = "DROP TABLE movie\_details CASCADE CONSTRAINTS";

*stmt*.executeUpdate(dropString);

System.***out***.println("Successfully dropped the movie\_details table.");

}

**catch**(SQLException se) {/\*Do nothing\*/};

//creates the movie\_details table

**try**{

System.***out***.println("Creating the movie\_details table...");

String createString = "CREATE TABLE movie\_details (\r\n" +

" movie\_id NUMBER (20),\r\n" +

" movie\_title VARCHAR(100),\r\n" +

" movie\_year VARCHAR (4),\r\n" +

" \r\n" +

" PRIMARY KEY (movie\_id)\r\n" +

" )";

*stmt*.executeUpdate(createString);

System.***out***.println("Successfully created the movie\_details table.");

}

**catch**(SQLException se){

System.***out***.println("Something went horribly wrong: " + se);

System.*exit*(1);

}

}

//creates the movie\_category table schema

//AND THAT IS IS- values will be read in later

**public** **static** **void** createMoviecategory (){

//attempts to drop the table, and does nothing if the table doesn't exist

**try**{

System.***out***.println("Dropping the movie\_category table...");

String dropString = "DROP TABLE movie\_category CASCADE CONSTRAINTS";

*stmt*.executeUpdate(dropString);

System.***out***.println("Successfully dropped the movie\_category table.");

}

**catch**(SQLException se) {/\*do nothing \*/};

//creates the movie\_category table

**try**{

System.***out***.println ("Creating the movie\_category table...");

String createString = "CREATE TABLE movie\_category (\r\n" +

" movie\_id NUMBER (20),\r\n" +

" movie\_category VARCHAR (30),\r\n" +

" \r\n" +

" PRIMARY KEY (movie\_id, movie\_category),\r\n" +

" FOREIGN KEY (movie\_id) REFERENCES movie\_details(movie\_id)\r\n" +

" )";

*stmt*.executeUpdate(createString);

System.***out***.println("Successfully created the movie\_category table.");

}

**catch** (SQLException se){

System.***out***.println("Something went horribly wrong: " + se);

System.*exit*(1);

}

}

//Creates the user details table schema

//AND THAT IS IT- values will be read in later

**public** **static** **void** createUserDetails(){

//attempts to drop the table, and does nothing if the table doesn't exist

**try**{

System.***out***.println("Dropping the user\_details table...");

String dropString = "DROP TABLE user\_details CASCADE CONSTRAINTS";

*stmt*.executeUpdate(dropString);

System.***out***.println("Successfully dropped the user\_details table.");

}

**catch**(SQLException se) {/\*do nothing \*/};

//creates the user\_details table

**try**{

//creates the user\_details table

System.***out***.println ("Creating the user\_details table...");

String createString = "CREATE TABLE user\_details (\r\n" +

" user\_id NUMBER (20),\r\n" +

" gender VARCHAR (1),\r\n" +

" age NUMBER (2),\r\n" +

" occupation NUMBER (2),\r\n" +

" zip VARCHAR (20),\r\n" +

" \r\n" +

" PRIMARY KEY (user\_id),\r\n" +

" CONSTRAINT check\_occupation\_code CHECK (occupation BETWEEN 0 AND 20),\r\n" +

" CONSTRAINT check\_age\_code CHECK (age BETWEEN 1 AND 56)\r\n" +

" )";

*stmt*.executeUpdate(createString);

System.***out***.println("Successfully created the user\_details table.");

}

**catch** (SQLException se){

System.***out***.println("Something went horribly wrong: " + se);

System.*exit*(1);

}

}

//creates the ratings table schema

//AND THATS IT- values read in later

**public** **static** **void** createRatings(){

//attempts to drop the table, and does nothing if the table doesn't exist

**try**{

System.***out***.println("Dropping the ratings table...");

String dropString = "DROP TABLE ratings CASCADE CONSTRAINTS";

*stmt*.executeUpdate(dropString);

System.***out***.println("Successfully dropped the ratings table.");

}

**catch**(SQLException se) {/\*do nothing \*/};

//creates the user\_details table

**try**{

//creates the user\_details table

System.***out***.println ("Creating the ratings table...");

String createString = "CREATE TABLE ratings(\r\n" +

" user\_id NUMBER,\r\n" +

" movie\_id NUMBER,\r\n" +

" stars NUMBER,\r\n" +

" date\_submitted DATE,\r\n" +

" \r\n" +

" PRIMARY KEY (user\_id, movie\_id),\r\n" +

" FOREIGN KEY (user\_id) REFERENCES user\_details(user\_id),\r\n" +

" FOREIGN KEY (movie\_id) REFERENCES movie\_details(movie\_id)\r\n" +

" )";

*stmt*.executeUpdate(createString);

System.***out***.println("Successfully created the ratings table.");

}

**catch** (SQLException se){

System.***out***.println("Something went horribly wrong: " + se);

System.*exit*(1);

}

}

//Populates the age key table

**public** **static** **void** populateAgeTable() {

**int** count = 0;

//keys and their corresponding values

//values are dictated by the readme attached to the dataset

String[] ageKeys = **new** String []{"1", "18", "25", "35", "45", "50", "56"};

String[] ageTrans = **new** String []{"Under 18", "18-24", "25-34", "35-44", "45-49", "50-55", "56+"};

**try** {

//populate the table

System.***out***.println("Beginning to populate age\_key table...");

PreparedStatement insertAges =

*conn*.prepareStatement("INSERT INTO age\_key VALUES (?, ?)");

//replace the values in the prepared statement

**for** (**int** i = 0; i < ageKeys.length; i++){

insertAges.setString(1, ageKeys[i]);

insertAges.setString(2, ageTrans[i]);

insertAges.executeUpdate();

count++;

}

System.***out***.println("Successfully inserted " + count + " rows into age\_key table.");

System.***out***.println("Done inserting values into age\_key");

}

**catch** (SQLException se) {

System.***out***.println("Something went horribly wrong: " + se);

System.*exit*(1);

}

}

//populates the job\_key table

**public** **static** **void** populateJobTable() {

**int** count = 0;

//job titles listed in order based on job key described

//job key will be inserted based on the loop variable

String[] jobTitles = **new** String[] {"Other or not specified",

"academic/educator", "artist", "cleritcal/admin",

"college/gradstudent", "customer service", "doctor/health care",

"executive/managerial", "farmer", "homemaker", "K-12 student",

"lawyer", "programmer", "retired", "sales/marketing",

"scientist", "self-employed", "technician/engineer",

"tradesman/craftsman", "unemployed", "writer"};

**try** {

System.***out***.println("Beginning to populate the job\_key table...");

PreparedStatement insertJobs =

*conn*.prepareStatement("INSERT INTO job\_key VALUES (?, ?)");

//Replace the values in the prepared statement

**for** (**int** i = 0; i < jobTitles.length; i++){

insertJobs.setInt(1, i);

insertJobs.setString(2, jobTitles[i]);

insertJobs.executeUpdate();

count++;

}

System.***out***.println("Successfully inserted " + count + " rows into the job\_key table.");

System.***out***.println("Done inserting into job\_key table.");

}

**catch** (SQLException se) {

System.***out***.println("Something with horribly wrong: " + se);

System.*exit*(1);

}

}

//Method reads in the movies.dat file that is found in the source folder

//Movies.dat

**public** **static** **void** populateMovieDetails(){

System.***out***.println ("Beginning to poulate the movie\_details table...");

File mov = **new** File (System.*getProperty*("user.dir") + "\\movies.dat");

**int** count = 0;

String [] insertArray;

**try**{

Scanner movScan = **new** Scanner (mov);

//make sure the file isn't empty

**if**(!movScan.hasNext())

System.*exit*(1);

//gets the first row

String currentRow = movScan.nextLine();

count++;

//initializes the size of the insert array based on the number of collumns

insertArray = **new** String[*getNumberOfEntries*(currentRow)];

//takes the first row and insert array

insertArray = *returnEachString*(currentRow);

//Creates the prepare statement for details

//Calls the fixMovieCatagories method to prepare/insert in catagories

//This eliminates the need for temporary tables

PreparedStatement insertDetails =

*conn*.prepareStatement("INSERT INTO movie\_details VALUES (?, ?, ?)");

//insert for the first row (which set up everything)

insertDetails.setInt (1, Integer.*parseInt*(insertArray[0]));

insertDetails.setString (2, *noYear*(insertArray[1], *onlyYear*(insertArray[1])));

insertDetails.setString (3, *onlyYear*(insertArray[1]));

insertDetails.executeUpdate();

//had to scrap this idea- too many cursors

//calls fixMovieCatagories on this first row

//fixMovieCatagories(insertArray[0], insertArray[2]);

//begins the loop starting on the second row

//had to use the first row to initialize the values

**while** (movScan.hasNextLine()){

count ++;

currentRow = movScan.nextLine();

insertArray = *returnEachString*(currentRow);

insertDetails.setInt (1, Integer.*parseInt*(insertArray[0]));

insertDetails.setString (2, *noYear*(insertArray[1], *onlyYear*(insertArray[1])));

insertDetails.setString (3, *onlyYear*(insertArray[1]));

insertDetails.executeUpdate();

//scraped idea, too many cursors error

//fixMovieCatagories(insertArray[0], insertArray[2]);

}

System.***out***.println ("Successfully inserted " + count + " rows into the movie\_details table.");

System.***out***.println ("Done inserting values into the movie\_details table.");

movScan.close();

}

**catch**(IOException e){

System.***out***.println("Something went horribly wrong with the file: " + e);

System.*exit*(1);

}

**catch**(SQLException se){

System.***out***.println("Something went horribly wrong with the SQL: " + se);

System.*exit*(1);

}

}

//Not super elegant, but opens back up the movies.dat file, and reads the relevant information from that file

//In each row, it will take the movieID as well as the not yet divided category field

//Method will pattern match on | to separate values in category and then insert the values into the movie\_catagories table

**public** **static** **void** populateMovieCategorys(){

//

System.***out***.println ("Beginning to populate movie\_categories table...");

**int** insertID = 0;

String insertCategory = "";

**int** count = 0;

Pattern p = Pattern.*compile*("([^|]+)");

Matcher m;

**try** {

File mov = **new** File (System.*getProperty*("user.dir") + "\\movies.dat");

Scanner mScan = **new** Scanner (mov);

//get the first row to set everything up

String currentRow = mScan.nextLine();

//Initialize array of size equal to the number of entries

String[] valuesArray = **new** String [*getNumberOfEntries*(currentRow)];

//Get the first rows of values into the array

valuesArray = *returnEachString*(currentRow);

//set up the prepared statement

PreparedStatement categoryInsert =

*conn*.prepareStatement ("INSERT INTO movie\_category VALUES (?, ?)");

m=p.matcher(valuesArray[2]);

//insert the first row

**while** (m.find()) {

//covert ID to an INT so that it works with the dbms

insertID = Integer.*parseInt*(valuesArray[0]);

//find each group marked by a |

insertCategory = m.group();

categoryInsert.setInt(1, insertID);

categoryInsert.setString(2, insertCategory);

categoryInsert.executeUpdate();

count ++;

}

//Need the first value (movieID), and the last value (category|category...)

//To do that, we now pass in to the loop the current ID, as well as the current uncompleted category

**while** (mScan.hasNextLine()) {

//get the next row

currentRow = mScan.nextLine();

//get the next set of values

valuesArray = *returnEachString*(currentRow);

//match on the new line

m=p.matcher(valuesArray[2]);

**try** {

//covert the first values (movieID) into an integer

insertID = Integer.*parseInt*(valuesArray[0]);

//run the pattern matcher on the current values array

**while** (m.find()){

insertCategory = m.group();

categoryInsert.setInt(1, insertID);

categoryInsert.setString(2, insertCategory);

categoryInsert.executeUpdate();

count ++;

}

}

**catch** (SQLException se){

System.***out***.println("Something went horribly wrong with the SQL: " + se);

System.***out***.println("Attempt was with: Movie id: " + insertID +" | category parameter: " + insertCategory);

System.*exit*(1);

}

}

System.***out***.println("Successfully inserted " + count + " rows into the movie\_category table.");

mScan.close();

}

**catch** (SQLException se){

System.***out***.println("Something went horribly wrong with the SQL: " + se);

System.***out***.println("Attempt was with: Movie id: " + insertID +" | category parameter: " + insertCategory);

System.*exit*(1);

}

**catch** (IOException io) {

System.***out***.println("Something went horribly wrong with the IO: " + io);

System.*exit*(1);

}

}

//Method reads in the users.dat file that is found in the source folder

//Method then takes each row and inserts them into the user\_details table

**public** **static** **void** populateUserDetails(){

**int** count = 0;

String[] insertArray;

String currentRow;

System.***out***.println ("Beginning to populate user details table...");

File user = **new** File (System.*getProperty*("user.dir") + "\\users.dat");

**try**{

Scanner userScan = **new** Scanner(user);

//sees if the file is empty

**if** (!userScan.hasNextLine()){

System.***out***.println("Empty or broken file.");

System.*exit*(1);

}

//gets the first line

currentRow = userScan.nextLine();

//initializes the insert array to have length of columns

insertArray = **new** String[*getNumberOfEntries*(currentRow)];

//inserts the first row into insert array broken up

insertArray = *returnEachString*(currentRow);

//According to the README from the data, the rows are as follows

//UserID, Gender, Age, Occupation, Zipcode

PreparedStatement insertDetails = *conn*.prepareStatement("INSERT INTO user\_details VALUES (?, ?, ?, ?, ?)");

//No additional maniuplation needs to be done

insertDetails.setInt (1, Integer.*parseInt*(insertArray[0]));

insertDetails.setString (2, insertArray[1]);

insertDetails.setInt (3, Integer.*parseInt*(insertArray[2]));

insertDetails.setInt (4, Integer.*parseInt*(insertArray[3]));

insertDetails.setString (5, insertArray[4]);

insertDetails.executeUpdate();

**while** (userScan.hasNextLine()){

// Get next line

currentRow = userScan.nextLine();

//Populate insert array

insertArray = *returnEachString*(currentRow);

//Replace values in the prepared statement insert details

insertDetails.setInt (1, Integer.*parseInt*(insertArray[0]));

insertDetails.setString (2, insertArray[1]);

insertDetails.setInt (3, Integer.*parseInt*(insertArray[2]));

insertDetails.setInt (4, Integer.*parseInt*(insertArray[3]));

insertDetails.setString (5, insertArray[4]);

//execute the statement

insertDetails.executeUpdate();

count++;

}

System.***out***.println("Successfully inserted " + count + " rows into user\_details.");

System.***out***.println("Done inserting values into the user\_details table.");

userScan.close();

}

**catch** (IOException io){

System.***out***.println("Something went horribly wrong with the IO: " + io);

System.*exit*(1);

}

**catch** (SQLException se){

System.***out***.println("Something went horribly wrong with the SQL at count: " + count + ". " + se);

System.*exit*(1);

}

}

//Method reads in the ratings.dat THICK file that is found in the source folder

//Method then takes each row and inserts them into the ratings table

//This method must be called AFTER populateMovieDetails AND populateUserDetails to keep referential integrity

//(Uses both userID and movieID as foreign primary keys)

**public** **static** **void** populateRatings(){

**int** count = 0;

String insertRow;

String[] insertValues;

File rat = **new** File (System.*getProperty*("user.dir") + "\\ratings.dat");

System.***out***.println("Beginning to insert into the ratings table...");

**try**{

Scanner ratScan = **new** Scanner(rat);

//check to see if there is a first row

**if** (!ratScan.hasNextLine()){

System.***out***.println("Broken ratings file.");

System.*exit*(1);

}

//get the first row

insertRow = ratScan.nextLine();

//initialize insertValues

insertValues = **new** String[*getNumberOfEntries*(insertRow)];

//break up the first row

insertValues = *returnEachString*(insertRow);

//convert the value (timestamp) into a date object for SQL

//since the value is in seconds since EPOCH, gotta multiply by 1000 since Java Date only works in miliseconds

Date insDate = **new** Date (Integer.*parseInt*(insertValues[3]) \* 1000);

//Insert the values of the first row into the prepared statement

PreparedStatement ratPrep = *conn*.prepareStatement("INSERT INTO ratings VALUES (?, ?, ?, ?)");

//We know from the README that the data is stored with the following organization

//UserID, MovieID, Rating, Timestamp (in seconds since EPOCH)

ratPrep.setInt(1, Integer.*parseInt*(insertValues[0]));

ratPrep.setInt(2, Integer.*parseInt*(insertValues[1]));

ratPrep.setInt(3, Integer.*parseInt*(insertValues[2]));

ratPrep.setDate(4, insDate);

ratPrep.executeUpdate();

count ++;

//Complete this for the rest of the file

**while** (ratScan.hasNextLine() && count < 200000){

//get the next row

insertRow = ratScan.nextLine();

//get each value from that row

insertValues = *returnEachString*(insertRow);

//convert the value (time stamp) into a date object

insDate = **new** Date (Integer.*parseInt*(insertValues[3]) \* 1000);

//prepare the prepared statement

ratPrep.setInt(1, Integer.*parseInt*(insertValues[0]));

ratPrep.setInt(2, Integer.*parseInt*(insertValues[1]));

ratPrep.setInt(3, Integer.*parseInt*(insertValues[2]));

ratPrep.setDate(4, insDate);

//execute

ratPrep.executeUpdate();

count++;

}

System.***out***.println("Successfully inserted " + count + " rows into the ratings table.");

System.***out***.println("Done inserting into the ratings table.");

ratScan.close();

}

**catch** (IOException io){

System.***out***.println("Something went horribly wrong with the IO: " + io);

System.*exit*(1);

}

**catch** (SQLException se){

System.***out***.println("Something went horribly wrong with the SQL at count: " + count + ". " + se);

System.*exit*(1);

}

}

//Method that returns ONLY the year from a given string

**public** **static** String onlyYear (String s){

String returnString = s;

Pattern p = Pattern.*compile*("\\((\\d{4})\\)");

Matcher m = p.matcher(returnString);

**if** (m.find())

returnString = m.group();

returnString = returnString.substring(1,5);

**return** returnString;

}

//Method that returns EVERYTHING BUT the year from a given string

//Uses the above onlyYear method to remove that substring from the movie title

**public** **static** String noYear (String s, String r){

String returnString = s;

returnString = returnString.replace("("+r+")", "");

**return** returnString;

}

//Method that returns each match the REGEXP finds as a value in an array

//All entries are treated as Strings and must be cast/converted into their format before inserted

**public** **static** String[] returnEachString(String s){

//getNumberOfEntries scans the first row of the file, does all the pattern matching, and returns the number of times matcher.find() returned true

//This allows us not not have to use dynamic arrays

//Must be re run every time we work with a new table

String[] returnString = **new** String[*getNumberOfEntries*(s)];

String testString = s;

String workingString;

**int** currentIndex = 0;

**int** nextIndex;

//loop variable

**int** tracker = 0;

Pattern p = Pattern.*compile* ("(:{2})");

Matcher m = p.matcher(testString);

**while** (m.find()) {

nextIndex = testString.indexOf(m.group());

workingString = testString.substring(currentIndex, nextIndex);

returnString[tracker] = workingString;

tracker ++;

//trim the string for next

testString = testString.substring(nextIndex+2);

}

returnString[tracker] = testString;

**return** returnString;

}

//Helper method so that we can avoid dynamic arrays

//Takes one row and counts the number of matches the REGEXP finds

**public** **static** **int** getNumberOfEntries (String s){

//variables

**int** count = 0;

String testString = s;

//Pattern matches

Pattern p = Pattern.*compile* ("(:{2})");

Matcher m = p.matcher(testString);

//Loops through and tracks number of times matcher.find is true

**while** (m.find()){

count ++;

}

**return** count+1;

}

**public** **static** ResultSet seeMyTable(String tName){

ResultSet rset = **null**;

**try**{

rset = *stmt*.executeQuery("SELECT \* FROM " + tName);

}

**catch**(SQLException se){

System.***out***.println("Something went horribly wrong with the SQL: " + se);

System.exit(1);

}

**return** rset;

}

}

**CODE FOR THE QUERIES:**

package hw6;

import java.sql.\*;

import java.util.Scanner;

public class RunQueries {

public static Connection conn;

public static Statement stmt;

public static void main(String args[]){

//personal login

setUpDatabase("qreidy", "cdm1355894");

//will accept user input up until newline

Scanner inputScan = new Scanner (System.in);

//May add option to choose multiple options- depends on how long I have leftover

System.out.println("The following query will return the top 10 movies names, "

+ "and the number of times reviewed that they were reviewed for the "

+ "profession you select.");

//Loops so that there can be multiple executions of the query and/or multiple tests

while (true){

System.out.println("Please enter the job name which you would like to search for."

+ "\n(? for a list of job titles || 0 to exit.)");

String input = inputScan.nextLine();

//testing

input = input.toUpperCase().trim();

//ends the loop

if (input.equals("0"))

break;

//prints the list of options then returns to the start of the loop

else if (input.equals("?")){

listOfOptions();

continue;

}

else

queryOne(input);

}

//shut it all down

inputScan.close();

try{

stmt.close();

conn.close();

}

catch (SQLException se){

System.out.println("Something went horribly wrong while closing the connections: " + se);

System.exit(1);

}

System.out.println("Everything is closed. \nProgram exiting.");

System.exit(0);

}

//Method to establish the database connection without having to bloat the main

//Also helps with practice to isolate the database initiation code

public static void setUpDatabase(String userId, String passWord){

//Attempt to load the JDBC driver

System.out.println ("Attempting to load the JDBC driver...");

try {

Class.forName("oracle.jdbc.OracleDriver");

}

catch(ClassNotFoundException e){

System.out.println ("Something went horribly wrong with the driver: " + e);

System.exit(1);

}

System.out.println ("Successfully loaded the driver.");

//Attempt to connect to the database

System.out.println ("Attemping to connect to the DePaul database...");

try{

conn = DriverManager.getConnection("jdbc:oracle:thin:@acadoradbprd01.dpu.depaul.edu:1521:ACADPRD0",

userId, passWord);

stmt=conn.createStatement();

}

catch (SQLException se){

System.out.println("Something went horribly wrong while connecting to the database: " + se);

System.exit(1);

}

System.out.println ("Successfully connected to the DePaul database.");

}

//Query that searches the database for the top 10 movies reviewed for a specific profession

//Profession is loaded in via the parameter into a prepared statement

public static void queryOne (String jobTitle){

int count = 0;

try{

PreparedStatement prepQuer = conn.prepareStatement(""

+ "SELECT movie\_details.movie\_title, COUNT(ratings.movie\_id)\r\n" +

" FROM movie\_details INNER JOIN (ratings INNER JOIN user\_details \r\n" +

" ON ratings.user\_id = user\_details.user\_id)\r\n" +

" ON movie\_details.movie\_id = ratings.movie\_id\r\n" +

" WHERE occupation = ( \r\n" +

" SELECT job\_code\r\n" +

" FROM job\_key\r\n" +

" WHERE UPPER(job\_translate) LIKE UPPER(?))\r\n" +

" GROUP BY movie\_details.movie\_title\r\n" +

" ORDER BY COUNT(ratings.movie\_id) DESC\r\n");

//Replaces the ? in the nested WHERE statement

prepQuer.setString(1, jobTitle);

//Result set doesn't need to be stored anywhere, so we'll just print it to the screen

ResultSet rset = prepQuer.executeQuery();

System.out.println ("\n\n-------------");

System.out.println ("-------------");

System.out.println ("-------------\n\n");

System.out.println ("Top 10 reviewed movies for " + jobTitle.toLowerCase() + "s.");

while (rset.next() && count < 10){

count ++;

System.out.println ("Movie: " + rset.getString("movie\_title") + "| Times reviewed: " + rset.getString("COUNT(ratings.movie\_id)"));

}

System.out.println ("\n\n-------------");

System.out.println ("-------------");

System.out.println ("-------------\n\n");

rset.close();

}

catch (SQLException se){

System.out.println ("Something went horribly wrong during the final query: " + se);

System.exit(1);

}

}

//Additional option in main

//Simple query to return all the names of the jobs currently stored in the job\_key table

//Feels like a more elegant solution in case job\_key table gets added to in the future

public static void listOfOptions(){

try{

ResultSet rset = stmt.executeQuery("SELECT \* FROM job\_key");

while (rset.next()){

System.out.println (rset.getString ("job\_translate"));

}

}

catch(SQLException se){

System.out.println("Something went horribly wrong with the options query: " + se);

System.exit(1);

}

System.out.println ("\n\n-------------");

System.out.println ("-------------");

System.out.println ("-------------\n\n");

}

public static void showMovieTable() {

int count = 0;

try {

ResultSet rset = stmt.executeQuery("SELECT \* FROM movie\_details");

while (rset.next() && count < 20) {

System.out.println ("Movie name: " + rset.getString("movie\_title") + "| Movie Year: " + rset.getString("movie\_year"));

count ++;

}

}

catch(SQLException se){

System.out.println("Something went horribly wrong with the show query: " + se);

System.exit(1);

}

}

}