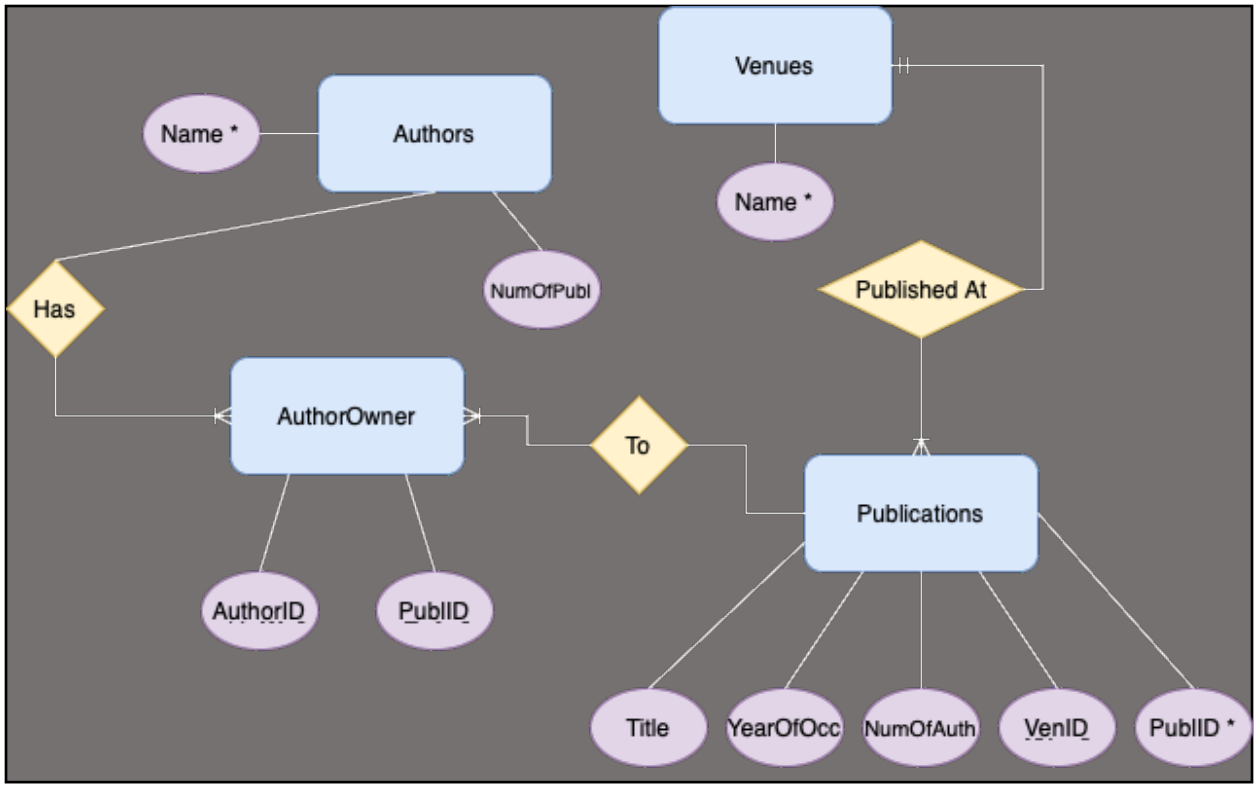
Overview:

This practical asked us to develop and implement a java program which uses JDBC to create and alter data from a connection to a database. Using methods developed in the previous practical, CS1003 P2, data can be pulled from a website API or cache directory of xml files. The data is then accessed via queries which use JDBC to call SQL retrieval statements. The implementation of this program required the use of several different methods, conditional statements, exceptions, xml document viewing/writing, and JDBC connections to SQL databases.

My solution achieved all the required aspects of the specification, and it passed all the rigorous individual tests that I created. My program contains four classes which all work together to create a SQL database, populate it with information pertaining to authors, publications and venues. Finally, the program can query that database to retrieve hard coded specific search results. At the very end there is a Test class which exists purely to test the functionality of the various methods. Using code from a previous practical submission, the PopulateDB class can pull information from the API or from a cache directory.

Design & Implementation:

My program is designed with six major key points that all build off the previous, to further expand upon a database. The first step of my project is ER_Diagram.png. This file contains a Chen Diagram of the relationship schema for my database. When I started this project, I realized that it would not be enough to only have three tables. I would need some type of link. Because publications can have multiple authors, I needed to find a way to not have duplicate information in my publications table. To fix this problem I created an AuthorOwner table which contains two foreign keys. One to the author's primary key and one to the publications primary key. This way if multiple authors work together AuthorOwner will add extra entries, but publications still only have one. Below is what my diagram looks like:



The second step of my project is CreateDataBase.txt. This text file contains DDL command prompts to create a specification accurate database in SQLite. I implemented the commands based off my ER diagram. This file has eight commands in it. The first four of these ensure that the database does not contain any tables relating to the ones above. The next four commands create tables with the appropriate format for authors, venues, authorOwner, and publications. Below are contents of this file:

- DROP TABLE IF EXISTS Authors;
- DROP TABLE IF EXISTS Venues;
- DROP TABLE IF EXISTS AuthorOwner;
- DROP TABLE IF EXISTS Publications;
- CREATE TABLE Authors (Name VARCHAR(100) PRIMARY KEY NOT NULL, NumOfPubl INTEGER);
- CREATE TABLE Venues (Name VARCHAR(100) NOT NULL PRIMARY KEY);
- CREATE TABLE AuthorOwner (AuthorID VARCHAR(100) NOT NULL, PublID VARCHAR(100), FOREIGN KEY (AuthorID) REFERENCES Authors (Name), FOREIGN KEY (PublID) REFERENCES Publications (PublID));

- CREATE TABLE Publications (Title VARCHAR(100) NOT NULL, NumOfAuth INTEGER, YearOfOcc VARCHAR(100) NOT NULL, PublID VARCHAR(100) NOT NULL PRIMARY KEY, VenID VARCHAR(100) NOT NULL, FOREIGN KEY (VenID) REFERENCES Venues (Name));

The first class of my source code is InitialiseDB.java. This class contains two methods. The first is the main method which creates a file object linked to the Database and checks to see if that file exists in the directory. If the database does exist, it is deleted. Otherwise, a new file for the database is created. A connection to the newly created file is made. A scanner object reads from CreateDataBase.txt and executes each of the SQL commands in the new file. Afterwards a call to checkInitialized is run to see if the database has been properly formatted. If so, “OK” is printed to the terminal. Check initialized uses an SQL statement which compares the sqlite_master table with those of the individually created ones. Every time a new table is created it is added to the sqlite_master, where these commands can pull to check if the format is correct. Should any of the conditional statement not match a false is returned and the whole class will throw an exception. The second class of my course code is PopulateDB.java. This class uses a signification portion of code from the last practical, CS1003 P2. The main method creates a new PopulateDB object and calls searchAuthor with the three specification specific authors. Search author takes the name and adds it to the end of the private URL attribute. An encoded version of this URL is created to check if the file exists in the cache directory. If the file exists it passes a document the saved file, otherwise it pulls from the API. With the newly created document, callToAuthorAPI is called. This method gets all the nodes from the file with a match to “hit”. Each node grabs the author item, which has author name, and the next URL. These are passed to callToPubl. callToPubl does the same thing as callToAuthor with the exception that it grabs to nodelists, one for inproceedings and one for articles. All of the relevant information is set to an attribute. The last xml viewing method is callToVenue which returns the VenID object but also inserts the name of each venue into the database via insertIntoDBVenue(). This is the same for insertIntoDBOwner(), insertIntoDBPubl() and insertIntoDBAuth(). Those four methods all create a connection to the database and use a hardcoded SQL command to insert the information to the proper table.

The third class in my practical submission is QueryDB.java. This class takes a command line argument (a value from 1 to 5) and runs the appropriate SQL command. The output of these Queries may differ slightly from my collogues and the expected result, however I believe I am justified in my output. For Query 1, I find that the total number of publications from “Ozgur Akgun” or “Ian Gent” or “Alan Dearle” is 250. This result might be slightly different from other answers for two reasons. First, I do not include “Ian Gent” (without a ‘P’) in my query. He only holds one publication and as he is not one of the desired authors, I did not deem it necessary to include him. Second, I only retrieve a count for distinct publications. Because some publications hold the same name i.e., they were revisions republished, I do not add that to the total count. This was my interpretation of the practical specification. The output if I do not include my reasons would be 256. For Query 2, the result is found from a simple SQL command that pulls the NumOfPubl integer from the corresponding author. The result for that is 46. For Query 3, I realized that the values of two tables would need to be joined in order to retrieve all publication title from a specific author. That is done in the SQL command and the result is provided in the testing section. For Query 4, the only addition my program adds is that I create a Year object which grabs the current year. Three is subtracted from this and used in the SQL command to determine the past three years of venues. The last query, Query 5, uses two join statements to retrieve the publication titles and venues for “Ozgur Akgun”. The output for Query 4 and 5 is provided in the testing section.

The final part of my practical submission is Test.java. This class contains various testing methods which are all printed in a neat format. The output of this method and further explanation will be provided in the testing section.

Testing:

What is being tested	Name of test method	Pre-conditions	Excepted outcome	Actual outcome	Evidence
Database initialization	testInitialization()	Database does not exist	Database file does not exist!	Database file does not exist!	Code Output 1

Database initialization	testInitialization()	Database is empty	Database file does not exist!	Database file does not exist!	Code Output 2
Database initialization	testInitialization()	Database has entries	Database has been properly initialized: true	Database has been properly initialized: true	Code Output 3 + DBeaver SS 1
All Query check	printAllQuery()	Database has been initialized and populated	Code Output 4	Code Output 4	Code Output 4
Load Author test	loadAuthors()	Database has been initialized and populated	Pulling user from database: Michael John|23	Pulling user from database: Michael John|23	Code Output 3 + DBeaver SS 2

Code Output:

1. Output 1

[redacted] src % javac *.java

[redacted] src % java Test

----------Testing Initialization----------

Checking to see if database has been initialized...

Database file does not exist!

Database has been properly initialized: false

---

2. Output 2

[redacted] src % sqlite3

SQLite version 3.37.0 2021-12-09 01:34:53

Enter ".help" for usage hints.

Connected to a transient in-memory database.

Use ".open FILENAME" to reopen on a persistent database.

sqlite> .open CS1003_P3DataBase

sqlite> ^D

[redacted] src % java Test

----------Testing Initialization----------

Checking to see if database has been initialized...

Database file does not exist!

Database has been properly initialized: false

---

3. Output 3

[redacted] src % javac *.java

[redacted] src % java InitialiseDB

OK

[redacted] src % java PopulateDB

[redacted] src % java Test

----------Testing Initialization----------

Checking to see if database has been initialized...

Database has been properly initialized: true

----------Testing Load Authors----------

Loading the database with a new author...

Creating a new author with name: Michael John, and 23 publications...

Pulling user from database:

Michael John|23

---

4. Output 4

[redacted] src % javac *.java

[redacted] src % java InitialiseDB

OK

[redacted] src % java PopulateDB

[redacted] src % java Test

----------Testing Initialization----------

Checking to see if database has been initialized...

Database has been properly initialized: true

----------Testing Load Authors----------

Loading the database with a new author...

Creating a new author with name: Michael John, and 23 publications...

Pulling user from database:

Michael John|23

----------Testing Print Query----------

Query 1:

Total Number of Publications by “Ozgur Akgun” or “Ian Gent” or “Alan Dearle”: 250

Query 2:

Total Number of Publications by “Özgür Akgün”: 46

Query 3:

A Framework for Generating Informative Benchmark Instances.

Understanding How People Approach Constraint Modelling and Solving.

Finding Subgraphs with Side Constraints.

Effective Encodings of Constraint Programming Models to SMT.

Discriminating Instance Generation from Abstract Specifications: A Case Study with CP and MIP.

Exploiting Incomparability in Solution Dominance: Improving General Purpose Constraint-Based Mining.

Instance Generation via Generator Instances.

Automatic Streamlining for Constrained Optimisation.

Automatic Discovery and Exploitation of Promising Subproblems for Tabulation.

Automatic Generation and Selection of Streamlined Constraint Models via Monte Carlo Search on a Model Lattice.
Metamorphic Testing of Constraint Solvers.
Closed Frequent Itemset Mining with Arbitrary Side Constraints.
A Framework for Constraint Based Local Search using Essence.
Using Metric Space Indexing for Complete and Efficient Record Linkage.
Exploiting Short Supports for Improved Encoding of Arbitrary Constraints into SAT.
Automatically Generating Streamlined Constraint Models with Essence and Conjure.
Cloud-based E-Infrastructure for Scheduling Astronomical Observations.
Cloud Benchmarking for Performance.
Optimal Deployment of Geographically Distributed Workflow Engines on the Cloud.
Automatically Improving Constraint Models in Savile Row through Associative-Commutative Common Subexpression Elimination.
Breaking Conditional Symmetry in Automated Constraint Modelling with CONJURE.
Automated Symmetry Breaking and Model Selection in Conjure.
Extensible Automated Constraint Modelling.
Conjure: Automatic Generation of Constraint Models from Problem Specifications.
Enumeration of set-theoretic solutions to the Yang-Baxter equation.
Automatic Tabulation in Constraint Models.
Towards Reformulating Essence Specifications for Robustness.
How People Visually Represent Discrete Constraint Problems.
Towards Portfolios of Streamlined Constraint Models: A Case Study with the Balanced Academic Curriculum Problem.
Exploring Instance Generation for Automated Planning.
Efficient Incremental Modelling and Solving.
Solving Computational Problems in the Theory of Word-Representable Graphs.
Cloud Benchmarking for Maximising Performance of Scientific Applications.
Conjure Documentation, Release 2.3.0.
Towards Improving Solution Dominance with Incomparability Conditions: A case-study using Generator Itemset Mining.
Modelling Langford's Problem: A Viewpoint for Search.

Memory Consistency Models using Constraints.
Automatically improving constraint models in Savile Row.
Extensible automated constraint modelling via refinement of abstract problem specifications.
Declarative Statistics.
Cloud Benchmarking For Maximising Performance of Scientific Applications.
The BIN_COUNTS Constraint: Filtering and Applications.
Conjure Revisited: Towards Automated Constraint Modelling

Query 4:
28th CP 2022: Haifa, Israel
18th CPAIOR 2021: Vienna, Austria
26th CP 2020: Louvain-la-Neuve, Belgium
17th CPAIOR 2020: Vienna, Austria
24th ECAI 2020: Santiago de Compostela, Spain
Artificial Intelligence, Volume 310
Mathematics of Computation, Volume 91
CoRR, February 2022
CoRR, May 2022
CoRR, November 2021
IEEE Transactions on Visualization and Computer Graphics, Volume 26
CoRR, September 2020

Query 5:
A Framework for Generating Informative Benchmark Instances. 28th CP 2022: Haifa, Israel
Understanding How People Approach Constraint Modelling and Solving. 28th CP 2022: Haifa, Israel
Finding Subgraphs with Side Constraints. 18th CPAIOR 2021: Vienna, Austria
Effective Encodings of Constraint Programming Models to SMT. 26th CP 2020: Louvain-la-Neuve, Belgium

Discriminating Instance Generation from Abstract Specifications: A Case Study with CP and MIP. 17th CPAIOR 2020: Vienna, Austria

Exploiting Incomparability in Solution Dominance: Improving General Purpose Constraint-Based Mining. 24th ECAI 2020: Santiago de Compostela, Spain

Instance Generation via Generator Instances. 25th CP 2019: Stamford, CT, USA

Automatic Streamlining for Constrained Optimisation. 25th CP 2019: Stamford, CT, USA

Automatic Discovery and Exploitation of Promising Subproblems for Tabulation. 24th CP 2018: Lille, France

Automatic Generation and Selection of Streamlined Constraint Models via Monte Carlo Search on a Model Lattice. 24th CP 2018: Lille, France

Metamorphic Testing of Constraint Solvers. 24th CP 2018: Lille, France

Closed Frequent Itemset Mining with Arbitrary Side Constraints. 18th ICDM 2018: Singapore - Workshops

A Framework for Constraint Based Local Search using Essence. 27th IJCAI 2018: Stockholm, Sweden

Using Metric Space Indexing for Complete and Efficient Record Linkage. 22nd PAKDD 2018: Melbourne, VIC, Australia

Exploiting Short Supports for Improved Encoding of Arbitrary Constraints into SAT. 22. CP 2016: Toulouse, France

Automatically Generating Streamlined Constraint Models with Essence and Conjure. 21. CP 2015: Cork, Ireland

Cloud-based E-Infrastructure for Scheduling Astronomical Observations. 11th eScience 2015: Munich, Germany

Cloud Benchmarking for Performance. CloudCom 2014: Singapore

Optimal Deployment of Geographically Distributed Workflow Engines on the Cloud. CloudCom 2014: Singapore

Automatically Improving Constraint Models in Savile Row through Associative-Commutative Common Subexpression Elimination. 20. CP 2014: Lyon, France

Breaking Conditional Symmetry in Automated Constraint Modelling with CONJURE. 21st ECAI 2014: Prague, Czech Republic

Automated Symmetry Breaking and Model Selection in Conjure. 19. CP 2013: Uppsala, Sweden

Extensible Automated Constraint Modelling. 25th AAAI 2011: San Francisco, California, USA

Conjure: Automatic Generation of Constraint Models from Problem Specifications. Artificial Intelligence, Volume 310

Enumeration of set-theoretic solutions to the Yang-Baxter equation. Mathematics of Computation, Volume 91

Automatic Tabulation in Constraint Models. CoRR, February 2022

A Framework for Generating Informative Benchmark Instances. CoRR, May 2022

Towards Reformulating Essence Specifications for Robustness. CoRR, November 2021

How People Visually Represent Discrete Constraint Problems. IEEE Transactions on Visualization and Computer Graphics, Volume 26

Towards Portfolios of Streamlined Constraint Models: A Case Study with the Balanced Academic Curriculum Problem. CoRR, September 2020

Exploring Instance Generation for Automated Planning. CoRR, September 2020

Efficient Incremental Modelling and Solving. CoRR, September 2020

Solving Computational Problems in the Theory of Word-Representable Graphs. Journal of Integer Sequences, Volume 22

Cloud Benchmarking for Maximising Performance of Scientific Applications. IEEE Transactions on Cloud Computing, Volume 7

Conjure Documentation, Release 2.3.0. CoRR, October 2019

Towards Improving Solution Dominance with Incomparability Conditions: A case-study using Generator Itemset Mining. CoRR, October 2019

Modelling Langford's Problem: A Viewpoint for Search. CoRR, August 2018

Memory Consistency Models using Constraints. CoRR, August 2018

Automatically improving constraint models in Savile Row. Artificial Intelligence, Volume 251

Extensible automated constraint modelling via refinement of abstract problem specifications. Constraints - An International Journal, Volume 22

Declarative Statistics. CoRR, August 2017

Cloud Benchmarking For Maximising Performance of Scientific Applications. CoRR, August 2016

The BIN_COUNTS Constraint: Filtering and Applications. CoRR, November 2016

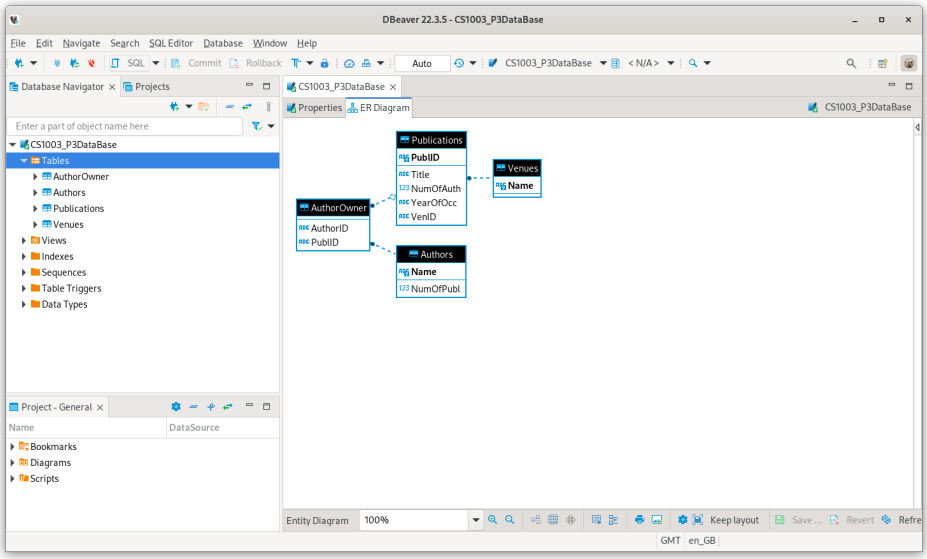
Optimal Deployment of Geographically Distributed Workflow Engines on the Cloud. CoRR, October 2014

Cloud Benchmarking for Performance. CoRR, November 2014

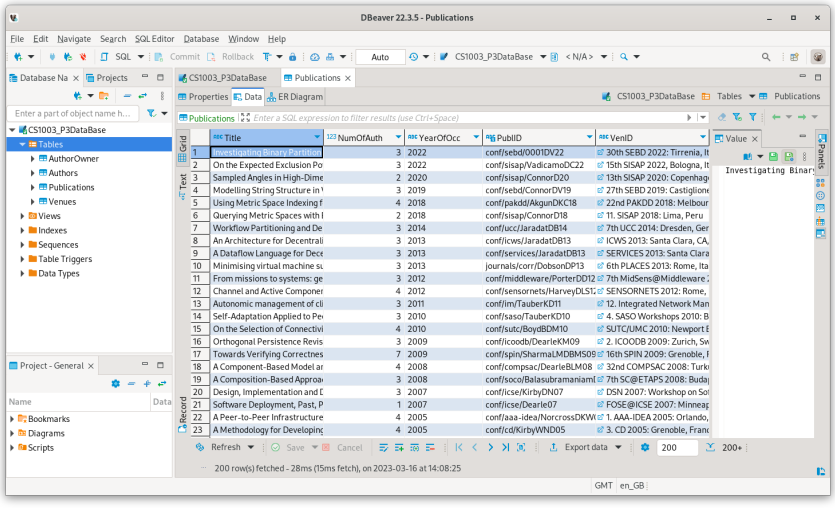
Conjure Revisited: Towards Automated Constraint Modelling CoRR, September 2011

DBeaver SS:

1.



2.



Examples:

Example of running all the code:

[redacted]@pc7-121-l:/cs/home/[redacted]/Documents/CS1003/CS1003_P3/src $ export CLASSPATH=${CLASSPATH}:./sqlite-jdbc-3.40.1.0.jar

[redacted]@pc7-121-l:/cs/home/[redacted]/Documents/CS1003/CS1003_P3/src $ javac *.java

[redacted]@pc7-121-l:/cs/home/[redacted]/Documents/CS1003/CS1003_P3/src $ java InitialiseDB

OK

[redacted]@pc7-121-l:/cs/home/[redacted]/Documents/CS1003/CS1003_P3/src $ java PopulateDB

[redacted]@pc7-121-l:/cs/home/[redacted]/Documents/CS1003/CS1003_P3/src $ java QueryDB 1

Total Number of Publications by “Ozgur Akgun” or “Ian Gent” or “Alan Dearle”: 250

[redacted]@pc7-121-l:/cs/home/[redacted]/Documents/CS1003/CS1003_P3/src $ java QueryDB 4

28th CP 2022: Haifa, Israel

18th CPAIOR 2021: Vienna, Austria

26th CP 2020: Louvain-la-Neuve, Belgium

17th CPAIOR 2020: Vienna, Austria

24th ECAI 2020: Santiago de Compostela, Spain

Artificial Intelligence, Volume 310

Mathematics of Computation, Volume 91

CoRR, February 2022

CoRR, May 2022

CoRR, November 2021

IEEE Transactions on Visualization and Computer Graphics, Volume 26

CoRR, September 2020

[redacted]@pc7-121-l:/cs/home/[redacted]/Documents/CS1003/CS1003_P3/src $

Evaluation:

At the completion of this project, I have found my implementation successful in adhering to the required specifications for the practical. I have tested my project rigorously and found that the outputs provide the desired result. In my testing I have produced a dependable, reliable, and accurate implementation of this practical.

Conclusion:

This practical tested all my knowledge on java, methods, conditional statements, exceptions, xml document viewing/writing and JDBC connections/command execution. The implementation of this practical gave me several goals. However, in the process of trying to achieve these, there were several challenges. Using JDBC to connect to a database proved more complicated than I expected. I would often find SQL errors that gave little to no information regarding what was happening. There were also several errors that arose when trying to retrieve venue information as different venue titles are stored is slightly different nodes. I had to write much of my reused code from the previous practical to obtain a proper populate class. The completion of this practical is a good indication of my current knowledge on java. Had I been given more time to work on this project I would implement several extensions. First, while this code is nearly formatted to populate and query any given author there are a few setbacks. For example, the AuthorID is just the author's name and there may be several authors with the same name. If I were to add to this program, I would produce a unique ID that is given to each author. I would also improve the populate method to accept any authors name via command line arguments. I would also adjust the query method to accept more command line arguments that allow a user to search for data relevant to any specific entry.

Previously Used Code:

Names of all methods taken from previous practical submission:

1. searchAuthor()
2. checkDirectory()
3. checkCache()
4. callToAuthorAPI()
5. callToPubl()
6. callToVenue()
7. writeXMLtoCache()

Every other method and piece of code was written by me and not obtained through any other means.