|  |  |  |
| --- | --- | --- |
| Computer Science 380 | February 14  2013 | |
| Implementation and Experimental Evaluation of Object-Oriented Programs that use the Structured Query Language. | | Braden Licastro |

1. JoSQL offers many features, including but not limited to searching, ordering and grouping java objects. It allows you to apply a structured query language statement to a collection of java objects. Even though JoSQL is not a database tool, it treats java objects in a way that emulates a database. In addition JoSQL supports many SQL statements such as select, insert, delete, and many conditional statements like and, or, <, >, where, and similar.

JoSQL processes queries in a very interesting manner. Each objected is treated as a row from a table. The methods from the objects are treated as columns. When you run a query it can access the entire object tree at one time and perform the requested function.

1. To test a program written in SQL and Java I would take several steps. To begin I would create an outline that would give the results I am looking for. I would then create a randomized dataset with both valid and invalid data and manually add in the results I will be testing for and expect. From here I would run the program and verify the returned data is as expected. If the results were not correct, the code would be corrected and rerun until the values were correct. From here additional tests would be generated and run to verify the query.
2. In order to test the performance of JoSQL I ran several benchmarks consisting in several configurations. Benchmark 1 was designed to test the performance of a basic search query. More specifically a list of names would be provided along with a first name. From here the program would search for and return every full name with a matching first name. For testing purposes two configurations were used. One configuration was a JoSQL query that would perform the search. The other configuration would use a hard coded regular expression to search for the matching names. Regular expressions were chosen because the JoSQL queries use pattern matching to perform the search. For the sake of consistency regular expressions were used to keep the results as accurate as possible. Both of these configurations were implemented using an ArrayList and a LinkedList. The benchmarks were run on datasets of size 500, 5000, 50,000, and 1,000,000 entries.

As an additional note, for the purpose of completing the given tests within the allotted time a restriction was placed on how long each test can take. Given the number of configurations and test cases the time was determined. With three days available after coding for testing and data analysis the restriction was determined by dividing the number of hours in the time period by the number of tests to be performed. This gave the worst case time per test of approximately 2.5 hours, which can be seen in several test cases below.

As seen in the graph below, when using an ArrayList the benchmark runs in a nearly linear time with the procedural approach averaging around the same runtime.

On the contrary, when a LinkedList is used linear time can only be realized by the procedural approach to executing the search as seen in the chart below. When using JoSQL the program runs in exponential time with the 1 million test timing out at 2.5 hours. This is due to the fact that JoSQL has to recall the LinkedList for every next item, and to reach that item it must iterate one at a time through the entire list until the index is reached. This makes a LinkedList a very inefficient data structure to use with this tool. It can also be noted that the ArrayList timed out. Though this result was unexpected it can be explained. Due to the tools need for recalling the data structure for every iteration of the search, there becomes a lag due to garbage collection that cannot be easily seen with smaller sample sizes. Garbage collection in Java is highly inefficient so at larger and larger data sizes this lag becomes relevant to the point where the code fails to complete execution before the time restriction set at 2.5 hours for testing purposes.

This dataset was chosen as it would provide strings to be tested against. The next test used integers in the same setup, with nearly identical results to what were discussed above. These results can be seen below.

In these tests a collection of phone numbers were used and the user would specify an area code to look up. The program would then return a list of the full phone numbers whose area codes matched the one requested.

The second benchmark used the same datasets, and was also implemented using ArrayLists and LinkedLists for the sake of consistency. The only variable changed was the complexity of the search. For names the expression would take a first name, as in the first test, and all matching first names would be returned. In addition to this, similar names would be returned. For example if the user wanted all full names with the first name of Matt the program would return people with the first name of Matt in addition to those whose first name was Matthew. Once again, as seen in the charts below, the results are nearly identical to the previous tests on the strings of names.

As in the first benchmark, the second dataset consisted of integer phone numbers. The results were once again nearly identical to those of the string searches and the previous search on integers. The results can be seen below.

These similarities in runtime can be attributed to the search algorithms used. Both algorithms used pattern matching of one type or another. Even though the query and chosen datasets can alter runtime, keeping these consistent over the duration of both benchmarks gives us a reliable data collection to base results on. By using pattern matching the efficiency benefits of integer comparisons is not relevant as everything is matched by pattern, as is with a string. Due to both datasets being treated as patterns and individual characters, both would experience nearly identical trends.

In conclusion neither procedural nor JoSQL implementations were significantly better or worse than the other. When implemented with a linked data structure, the query became massively inefficient so LinkedLists are strongly discouraged. When tested using an ArrayList the results were so similar they can be considered negligible. The benefit of using the JoSQL implementation will save development time for nearly identical running times. This is because the implementation of a procedural program can be significantly more complex, needing methods of data comparison, handling, and storage, while JoSQL does this natively. For large datasets JoSQL can still be favored for the same reason, but it is necessary to choose a data structure that can first handle the data size efficiently and second be used seamlessly with JoSQL with minimal time and memory overhead.