Western Washington University – CSCI Department
CSCI 330 Database Systems

SURLY { I/II } Report

Andrew Nguyen, Jonathan McCamey

Who is on your team and what's the division of labor?

Andrew Nguyen and Jonathan Mccamey

The division of labor for the second portion weighted towards Andrew, but we both contributed.

```
 mccamej <mccamej@wwu.edu>:
  insertions:    612 (18%)
  deletions:     95 (5%)
  files:         35 (19%)
  commits:       35 (55%)
  lines changed: 707
  first commit:  Wed Apr 24 13:01:37 2019 -0700
  last commit:   Sat Jun 1 17:12:36 2019 -0700

 Andrew <osbetel@gmail.com>:
  insertions:    2774 (82%)
  deletions:     1666 (95%)
  files:         145 (81%)
  commits:       29 (45%)
  lines changed: 4440
  first commit:  Wed Apr 24 14:25:55 2019 -0700
  last commit:   Sun Jun 2 23:43:29 2019 -0700

total:
  insertions:    3386 (100%)
  deletions:     1761 (100%)
  files:         180 (100%)
  commits:       64 (100%)
```

What programming language did you select and why?

Java 8 was the programming language for this assignment. Language was assigned.

List libraries or programming language features you made use of?

- This is very useful to transfer into your resume, so provide adequate detail.
- LinkedList
- List
- Collections
- Cloneable (removed later)
- Scanner
- java.io

Deliverables

Checklist of deliverables	
Hardcopy of	I/II/III
This writeup	x
Zip file containing	I/II/III
This writeup	x
Test cases showing input/output	x
Source code	x
README.TXT *	x

Coverage - Did you complete all of SURLY Part I/II - what is missing?

version	Feature	Covered/Comment
I	Relation	
I	Insert	
I	Print	
I	Heap Storage	
I	Catalog	
I	Destroy	
II	Delete where … AND/OR	
II	Select where … AND/OR	
II	Project	
II	Join	This is the only thing that is missing. Partial functionality was implemented (it can join two Relations fine), but the boolean condition (JOIN… ON…) is non-functional. I disabled whatever spaghetti code I had for it so that it wouldn't affect the rest of the program.
other	Import/Export, GUI, …	

How did you implement

- **Relations** – Relations are implemented with two LinkedLists. One for Tuples and one for Attributes.
- **Tuples** – Each Tuple contains a LinkedList of AttributeValues.
- **Attributes** – Attributes and Attribute Values are separate objects, but I later allowed Attribute Values to "know" what their parent type Attribute is. ie: every AttributeValue has a reference to the Attribute it corresponds to.
  The reason for this is that, particularly in Select and Delete parsers, it made the implementation easier if an Attribute Value knew what type it was. Then you could ask for its type and, if it was not a specified Attribute in the boolean condition, get rid of it.
- **Insert** – Insert was completely unchanged from Surly1
- **Catalog** – Unchanged
- **Destroy** – Unchanged

- **Delete where** – Using SelectParser, we can parse the where clause just like in SelectParser, but instead of returning those Tuples in a new Relation, we search for and delete them from the DB.
- **Select where** – Uses BooleanConditionHandler to handle the WHERE clause portion. SelectParser functions very similar to set operations (as Relations/Tables are like sets, so this makes sense) and handles the selection of Tuples that way.
- **...**

Things you did differently (e.g., than the SURLY spec)

Limitations of the current release.

Cannot JOIN on a condition, can only JOIN two Relations outright.

Extra features you added - e.g., going beyond the SURLY I/II spec

–

Things you are especially proud of

The recursive implementation in BooleanConditionHandler's extractTuples for SELECT commands. Additionally, DELETE and SELECT were very similar, so I was able to re-use some of the code. DeleteParser uses SelectParser to function. -Andrew

Recommendations

Things you would do differently if starting over now.

Most of the Select, Delete, Join, parsers function by creating a deep copy of a relation and then removing what isn't necessary under the boolean condition. For small databases this is fine but is computationally inefficient.

Next time I would probably try to pay more attention to computational efficiency. Running a nested O(n^2) for loop doesn't do anything when you have 10 tuples, but I imagine it's super important for large 10k or 100k size databases.

What did you learn about databases from SURLY?

Both computational efficiency and space efficiency matter. In fact they're both very important, so the ability to trade-off space for speed, or speed for space is not really much use here like it is in other types of CS applications.

Any other comments?