**Western Washington University – CSCI Department**

**CSCI 330 Database Systems**

# SURLY { II } Report

Student Names

| DAVID FUENTES | TAYLOR MORSE | BRIAN STCLAIR |

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

David – Updated Parser, Added Sounds / Ascii Art, Delete Where, Print() Relation

Taylor – Implemented Select Where, Project, Join and Destroy

Brian – Implemented feedback from SURLY I

## What programming language did you select and why?

Java – Due to all of our previous experience with the language it was a unanimous decision.

## List libraries or programming language features you made use of?

* - Java.IO.\* - We used this library in order to implement the Scanner as well as any input required for Surly 2. The main reason that we chose to use the entire library was so that we could future proof the program for any additional features.
* - Java.util.\* - Using the linked list feature of Java allowed us to implement a dynamic array of any time of objects. Rather than using an array, linked lists allowed us to implement the insert function so that the Relation had no limit for the tuples.
* - String.split() – This function allowed us to feed the parser a single line from the input file and turn it into an array of strings that would represent the words themselves. This was a requirement so that we could test each word individually to see which were commands and which were just information/ filler words.
* - Arrays.toString() – We used this for printing the arrays as a debugging tool.
* - JavaX.sound – For audio files
* - Java.util.LinkedList – These were especially helpful for creating dynamic attribute index lists as well as the temporary lists of attributes and tuple for the relations returned from queries. As well as using the iterator method for linked lists.

## 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 | SUCCESSFULLY IMPLEMENTED |
| I | Insert | SUCCESSFULLY IMPLEMENTED |
| I | Print | SUCCESSFULLY IMPLEMENTED |
| I | Heap Storage | SUCCESSFULLY IMPLEMENTED |
| II | Destroy | SUCCESSFULLY IMPLEMENTED |
| II | Delete where … AND/OR | SUCCESSFULLY IMPLEMENTED |
| II | Select where … AND/OR | SUCCESSFULLY IMPLEMENTED |
| optional | Join, Project, Import/Export in XML, CATALOG, GUI, … | SUCCESSFULLY IMPLEMENTED (Join and Project) |

## How did you implement

* **Relations** – This class was used as a container to hold a linked list of Tuples that represent the rows of the table as well a linked list of attributes that were used for white space in printing the table. This class also had to print class itself so that it’d be easier to print the entire data base because you could just print the relation rather than every tuple in every relation.
* **Tuples** – This class held a linked list of strings called “data” to represent the information in the rows of the table. Originally the tuples also had a linked list of attributes that would be used for formatting but we ran into an issue when inserting. The issue was the when inserting a tuple it was too difficult to implement the spacing when constructing a tuple. Because every tuple in a relation has specific formatting rules for the white space, we decided to let the relation itself take care of the formatting. This also allows for the table to be more dynamic; for instance if you wanted change how the table was printed you’d only need to change the relation once and not every tuple in the relation which would save you potentially thousands of operations depending on the size of the relation.
* **Attributes** – This is the most basic class. This class only handles formatting for the relation. This class has 3 strings: name, displayType, and formatSpacing. Name will eventually be used for selecting columns of the tuples. DisplayType would be used for unique formatting based on the type of the objects, as well as logistical computations for ints and things like that.
* **Insert** – This function uses the second word in the list of strings that represent the current line of the input file as the name of the relation that we will be inserting into. The reset of the words get handles in the constructor for tuple to create a new tuple. After the tuple is created we used the linked list function .Insert() to add the tuple into the linked listed of tuples called “Rows” in relation.
* **Destroy** – Created a method in the Database class called remove that takes the relation out of the Linked List of relations in the Database.
* **Delete where** – Used already constructed Select Where method to return a relation that met all of the conditions expected by the input. Using this relation we deleted all tuples that matched the desired Relation to be deleted and the temporary relation constructed from the conditions.
* **Select where** – Used a temporary database to store all the individual results of each where condition. Then it takes all those and combines them with either AND or OR protocol. Deletes the first 2 relation in the database, then adds the new combined relation to the front, to be compared with the next relation of one condition.
* **Join –** Takes in two relations, and creates a new relation with both relations attributes and new tuples of the combined two relations tuples. Gets the join condition indexes and uses those to filter the tuples it’s adding to the new relation.
* **Project –** Creates a new Relationwith dynamically created tuples of the attributes requested. The attribute indexes, will automatically be sorted by the attributes that come first in the relation, first in the projected relation.

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

### Limitations of the current release.

Limited error handling, like if you want multiple join conditions, or if you try to Project the same attribute twice.

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

Sounds, Ascii Art, Help cmd

### Things you are especially proud of

Parser, Ascii Art, Sounds, Dynamically created linked lists of attribute indexes, Temporary Database AND and OR combination of relations.

## Recommendations

### Things you would do differently if starting over now.

Have a design that implements the classes we already wrote, like how we used our select where in the delete where. Reusable code is key to saving time.

### Did SURLY meet your objectives for this course?

Yes it taught us how to create a database and perform queries and other operations.

### Suggestions on how to improve SURLY I/II assignment

Give concrete structural examples about what is a linked list of what.

### Suggestions on how to improve the course?

Present tables in a better format, like increasing the font and using contrasting colors on a projector. Also make sure that the attribute names match up in the example and input file.

### Any other comments?

Keep doing examples, the method of showing what to do then having students work examples are really nice.