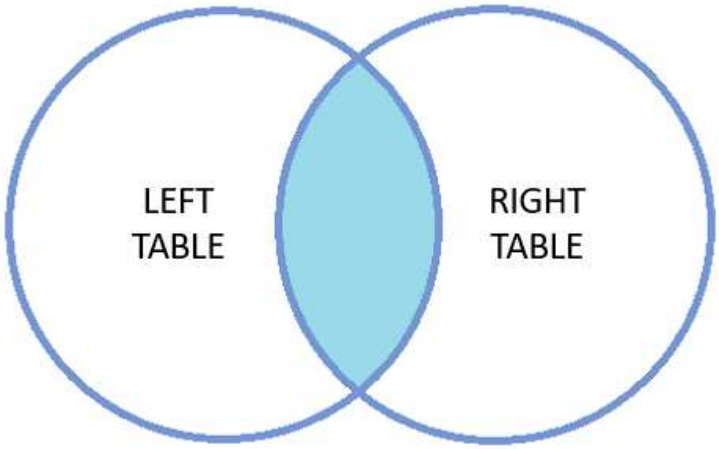
Programming Assignment 3: Table Join Documentation

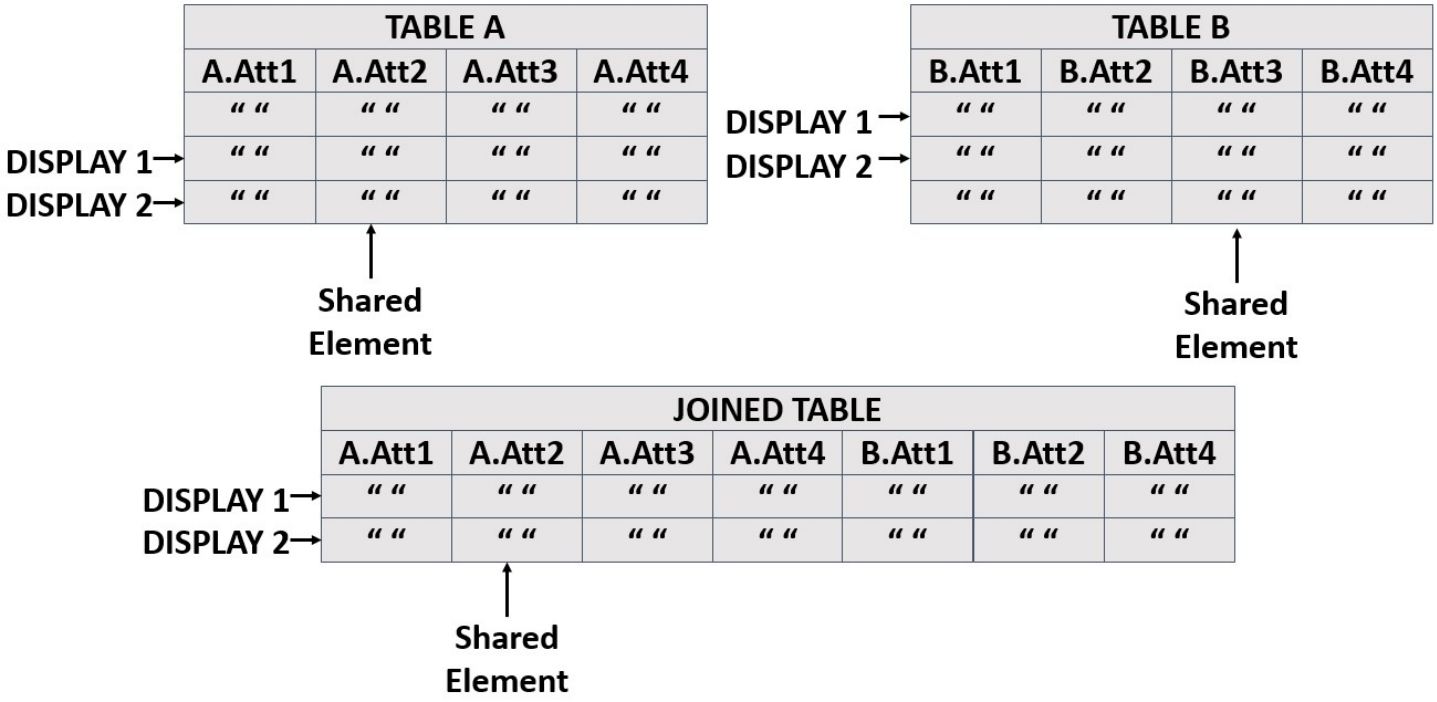
By Mercedes Anderson, Austin Emery, Nickolas Johnson

We have implemented inner join and left outer join into our database management system.. We also fixed a few parsing issues introduced with the latest test script. Setting the Table class as a friend class to the Database Class allowed for easy printing of the attributes and tuples during the joining process.

Inner Join:

Inner join is similar to the mathematical term of intersection. It takes a similar element from both tables and displays the tuples from left to right. This may leave out tuples from both tables if they do not share tuples with similar elements. We implemented this using the pseudo code from the assignment. We used a nested loop to compare each element of the first table with each element of the second table. Any intersections yielded in the printing of the tuples to terminal.





Left Outer Join:

Left outer join is an expansion of inner join as it includes all tuples from the left table and prints out blank or null elements if there are no intersections with the right table. We implemented the left outer join in the same way as the inner join function. The slight difference was the addition of any unmatched tuples from the left table still being printed to terminal.

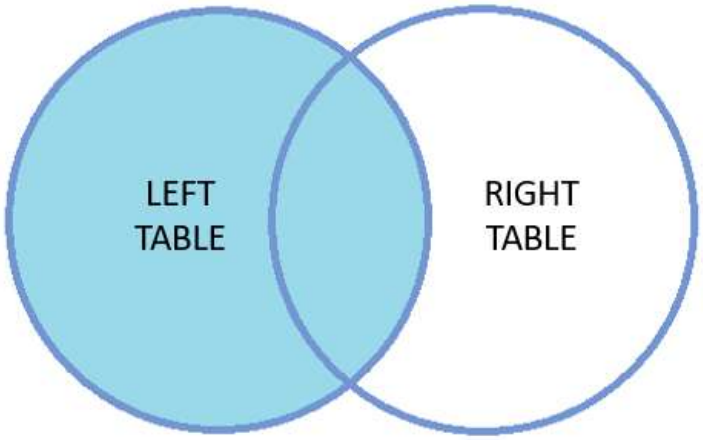


	TABLE A			
	A.Att1	A.Att2	A.Att3	A.Att4
DISPLAY 1→	" "	" "	" "	" "
DISPLAY 2→	" "	" "	" "	" "
DISPLAY 3→	" "	" "	" "	" "

Shared Element

	TABLE B			
	B.Att1	B.Att2	B.Att3	B.Att4
DISPLAY 2→	" "	" "	" "	" "
DISPLAY 3→	" "	" "	" "	" "

Shared Element

	JOINED TABLE						
	A.Att1	A.Att2	A.Att3	A.Att4	B.Att1	B.Att2	B.Att4
DISPLAY 1→	" "	" "	" "	" "	NULL	NULL	NULL
DISPLAY 2→	" "	" "	" "	" "	" "	" "	" "
DISPLAY 3→	" "	" "	" "	" "	" "	" "	" "

Shared Element

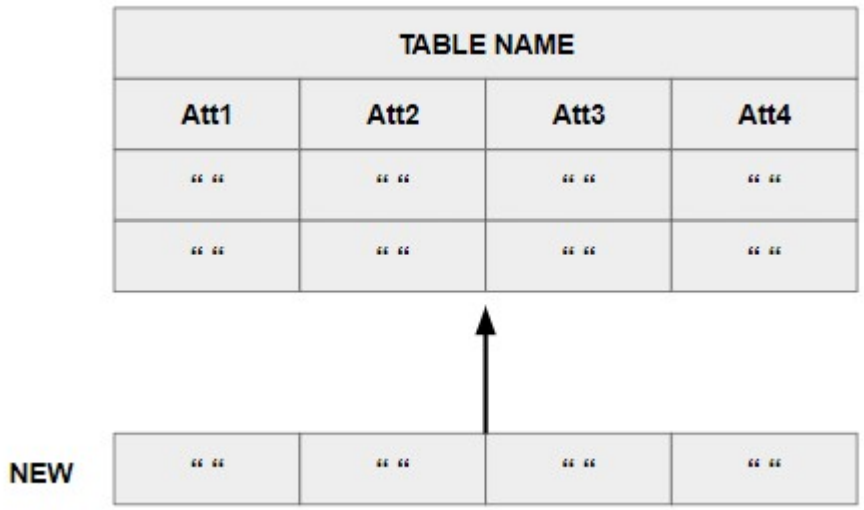
Programming Assignment 2: Basic Data Manipulation Documentation

By Mercedes Anderson, Austin Emery, Nickolas Johnson

We have implemented basic data manipulation into our database management system. Due to using the vector data structure in the design, the table is automatically restructured upon any alteration.

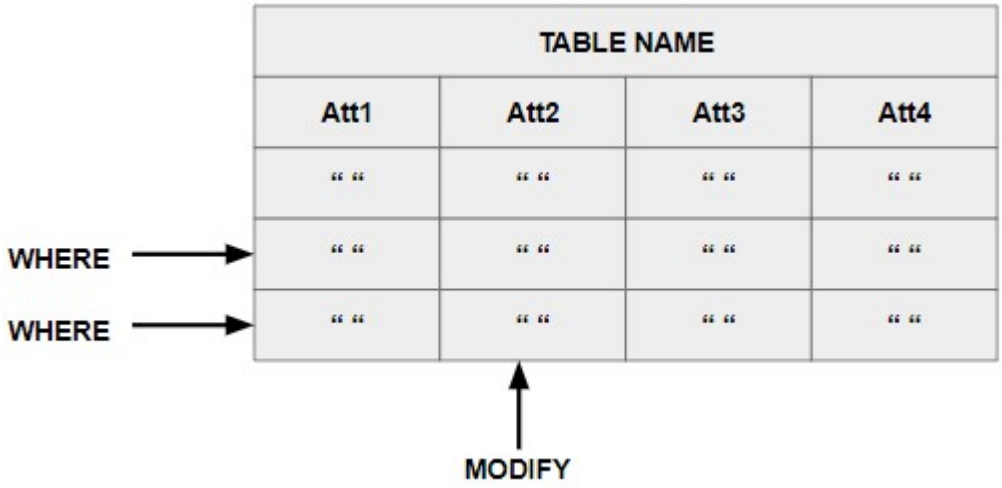
Insert:

Insert is the simplest of basic data manipulation to the existing database management system. The user will insert a tuple following the tables structure. The inserted tuple will be pushed to the back of the vector in a queue-like fashion. Due to the tuple structure being the only restriction, there is no need to compare the data to any other existing tuple and can just be added with no problem.



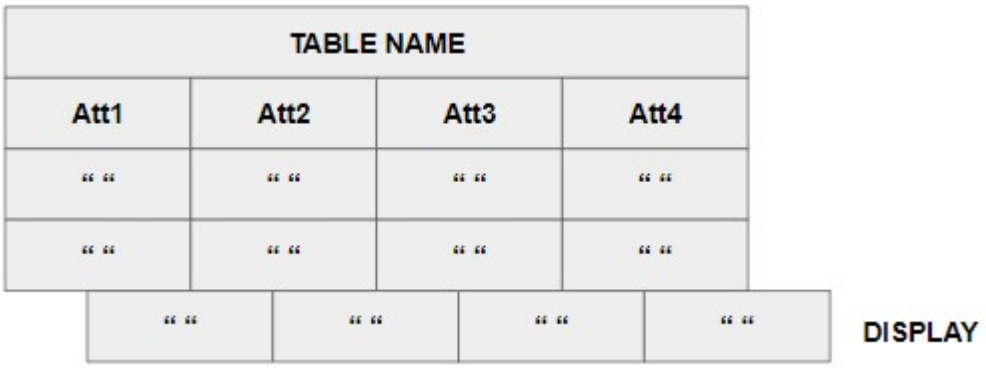
Modify:

Modify creates a parser to determine what the user input is asking. The function then searches the table to find applicable tuples. The tuples are modified in place, so the existing data being modified is overwritten. There is no need for the table to be restructured as no data is moved, only overwritten.



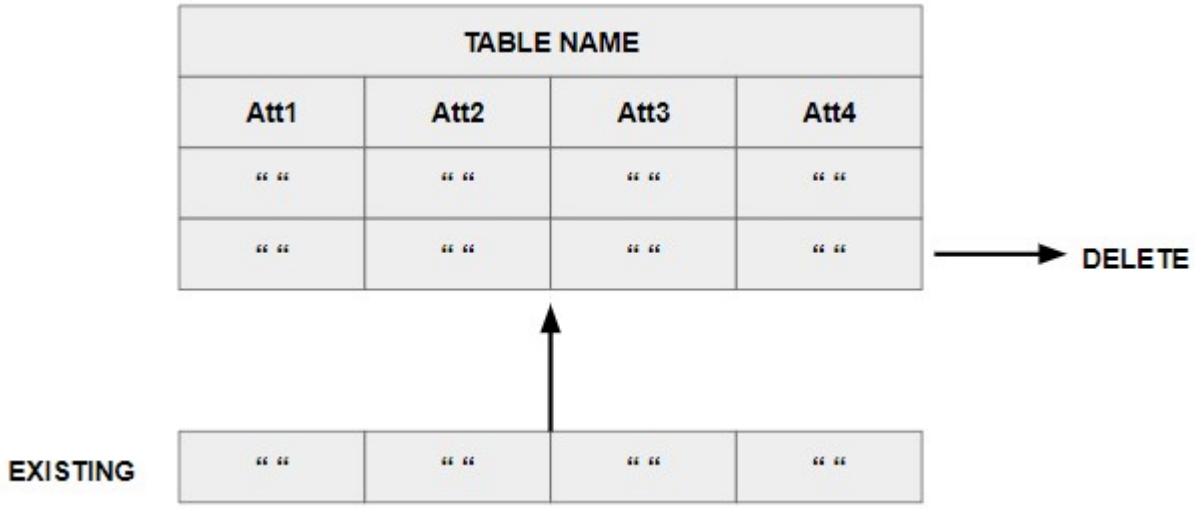
Query:

Query uses a similar parser created in modify to find the tuples applicable to the user input. This data is then printed using a class function from the table class to display the data between pipes. There is no need for the table to be restructured as the data is only being displayed on the console.



Delete:

Delete re-uses a small part of the parser created in modify to compare the requested information to be deleted. It then compares the information in a similar manner as modify to decide which tuples will be deleted on the pass through of the table. Due to the table using the vector class, all that is needed is the existing erase function to remove the desired tuples. The vector class then takes care of restructuring and removing the gap.



Programming Assignment 1: Metadata Management Documentation

By Mercedes Anderson, Austin Emery, Nickolas Johnson

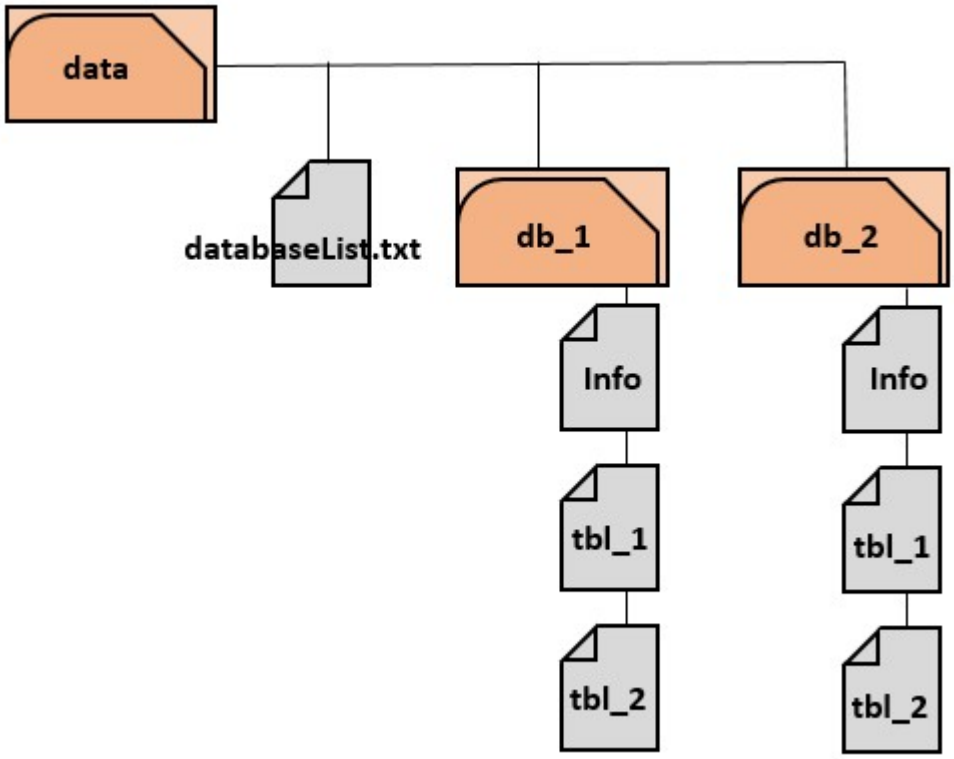
Our database management program is implemented with the programming language C++.

Databases:

Our database management program handles databases using the C++ defined class, vectors. Each database is pushed onto the vector as they are being edited. When the databases are saved, each database is saved as a separate directory included in the program directory. The databases are saved in parallel and may be nested later if needed.

Tables:

Our database management program handles tables using the C++ defined class, vectors. Each table is pushed onto the vector within the database class as they are being edited. When the tables are saved, each table is saved as a separate file included in the database directory. The tables are saved in parallel and may be nested via merging later if needed.



Implementation:

Each function contains a doxygen level comment above the function prototype in the PA1.cpp file. This contains the name of the function, the brief description, the prerequisites, the post effect, and the function type.

Two abstract structures are used to organize our database system, included in classes.h. The abstract structures created are the Table and Database classes. Each class contains a vector for the metadata and a vector for the data, either the tables for the database or the column information for the tables. The vectors expand off of each other creating a 2-D array of data for the tables, depicted below, while the database creates a 1-D array of tables as depicted above.