Reed Hampton
Grant Hruzek
Hanzhi Guo

SQL-DBMS Project Design Document

Section 1 - Purpose of the Project

The purpose of this project is to design, from the ground up, a simple, stand-alone, relational database management system. This will be implemented in Java and will utilize a subset of typical algebraic relational operations such as: Selection, Projection, Renaming, Set-union, Set-Difference, and Cross Product in order to execute queries and commands on the database system. The project is set up in a two-part manner as follows:

Database Parser – Reads the users command line input and *tokenizes* it into demarcated commands. The tokens are then *parsed* and passed off to the database engine through means of a function call to be processed.

Database Engine – The DB Engine will mainly consist of functions which process the recently parsed instructions and performs the operation. These operations are either *queries* or *commands*. Queries will involve returning the result of the query specification to the user, whereas Commands will result in the specified table (*.db text file*) of being update with the passed information.

The main goal with this project is to imitate an in-memory SQL-like database with only a subset of operable commands that can execute simple queries/commands on a database. The DB parser will extract the commands/arguments from the user and the DB engine will execute the set of commands on the data specified by the user. The database will also retain system logs to record operations for security/backup purposes. A high-level overlook of the database entities are shown below in the next section.

Section 2 - Definition of high level entities in the design[1]

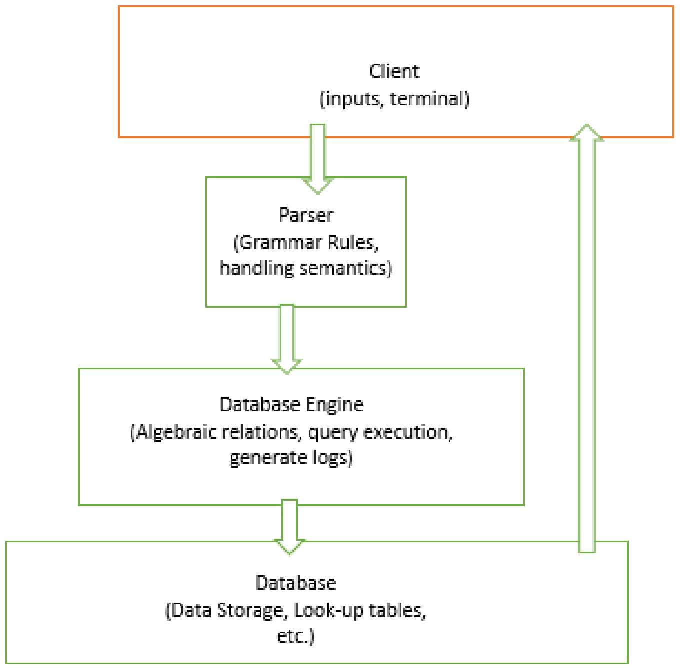
1. **Database Parser –** The parsers job is to read the users input from the command line and perform all preprocessing. This preprocessing includes first tokenizing the input into meaningful tokens. In this projects case that means separating the input into words by using "*whitespace*" as a demarcator. These tokens (words) will then be parsed to inform us of their meaning. The parser will accomplish this by selecting the appropriate function to call based on whichever command/query is passed as the argument.

2. **Database Engine** - The database engine takes in the semantics of the parser and performs common database tasks. These tasks include the basic queries and commands that will be listed below. The DB Engine accomplishes this by accessing (reading/writing) to the designated table given to the program by the user in the argument. These tables are stored as text files.

3. **Database** - The database is where all the entities and relations are stored; it is accessed and modified by the Database Core Functions of the DB Engine. This Database will consist of many text files with a *.db* extension and will be accessed through basic file read/write operations.

[1] See the graphic on the following page for visualization of this process

Reed Hampton
Grant Hruzek
Hanzhi Guo


Client
(inputs, terminal)
Parser
(Grammar Rules, handling semantics)
Database Engine
(Algebraic relations, query execution, generate logs)
Database
(Data Storage, Look-up tables, etc.)

Section 3 - Low level design of each entity

Client Command Line – The command line allows user to input queries or commands, which are then passed through to the parser for lexical analysis, parsing, and processing. This "*Domain Specific Language*" allows for communication with the **DBMS**. Example input commands are as below:

Queries: All queries will be paired with a command which indicates what to process

Relation-Name	Returns a relation name based off specifications
Selection	Selects items from a specified table
Comparison	Used to determine which items are to be queried
Projection	Projects a query onto a specified relation
Attribute-List	Returns the list of attributes of the specified relation
Renaming	Renames a relation
Union	Returns the added items of two sub-queries
Difference	Returns the subtracted items from two sub-queries
Product	Returns the product of items from two sub-queries
Natural-Join	Returns a view of the attributes of two relations

Commands: Allows for input which needs to create, update, store, or delete to/from relations

Open	Opens a database relation
Close	Closes a database relation
Write	Saves any changes to the database relation
Show	Shows the database relation to the user
Create	Creates a new database relation (new .db file)
Update	Updates attributes of existing database relation
Insert	Inserts an item into a database relation
Delete	Deletes an item from a database relation

Reed Hampton
Grant Hruzek
Hanzhi Guo

Database Parser - Receives instructions from the command line as arguments and tokenizes them by using the whitespace as a separator. The new tokens are then parsed based on the preset/provided grammar rules. They will be parsed into the commands and queries viewed above and will call the appropriate functions from the DB Engine for processing. Below are some examples of parsed instructions and how they will complete function calls:

The parser will be autogenerated using a program called ANTLR.

Tokenization: Will be of the form of the above queries/commands and will be tokenized as follows.

Parsing: Will parse each token in the command, verify it is a valid and expected token, and call DB Engine functions.

INSERT INTO table_1_name VALUES FROM table_2_name

TOKENIZING PROCESS		**PARSEING PROCESS**
Token 1	“*insert*”	Will call the insert() function and check if next token is “*into*”
Token 2	“*into*”	Verifies we are inserting and will check next token for relation name
Token 3	“*table_1_name*”	If a relation name, open the file and prepare to append to the end
Token 4	“*values*”	Check if token == “*values*” if not it is a bad command
Token 5	“*from*”	Check if token == “*from*” if not it is a bad command
Token 6	“*table_2_name*”	Check if the token is a literal (append literals into opened file) or relation name (open new file for copy)

Since Parser will be auto generated by ANTLR as allowed by Program Specifications function headers have not yet been made and thusly cannot be listed. They will function in accordance to the example above.

Reed Hampton
Grant Hruzek
Hanzhi Guo

Database Engine - Based on the commands/arguments passed by the parser[2] the DB Engine will execute the user-specified commands on the database. The DB Engine will first determine if the instruction is a *Command* or a *Query* and then call the appropriate function. There will exist a function for every command and query variation. The functions will process the arguments based on the associated algebraic relations to the database files. A log will keep a record of all database changes. Below is a list of the subroutine declarations, along with a comment explaining its' function.

```
void processQuery(string[] instructionTokens());  //Will process an expression on the relation (first argument)

    // Will remove the keyword, open relation, check for condition and call an expression if listed
            void select(string relationName, string[] instructionTokens);
    //Will remove the keyword, read any listed attributes, check for any other expressions, and show the view
            void project(string relationName, string[] instructionTokens);
    //Will remove the keyword, read any listed attributes, and call atomicExpression()
            void rename(string relationName, string[] instructionTokens);
                            //----------HELPER FUNCTIONS----------------------//
    //Will remove the two relation names, read from each relation, and select items that fit the condition
            void union(string relationName1, string relationName2, string[] instructionTokens);
    //Will remove the two relation names, read from each relation, and select items that DON'T fit the condition
            void difference(string relationName1, string relationName2, string[] instructionTokens);
    //Will remove the two relation names, read from each relation, and perform BOTH sub-queries
            void product(string relationName1, string[] relationName2);
    //Will remove the two relation names, read from each relation, and combine the distinct attributes into one view
            void naturalJoin(string relationName1, string relationName2, string[] instructionTokens);

void processCommand(string[] instructionTokens());  //Will process a command

    // Will remove the keyword, open the data file
            void open(string relationName);
    //Will remove the keyword, and close WITHOTU saving the data file
            void close();
    //Will remove the keyword, and save and close the opened data file
            void write();
    //Will remove the keyword, and exit the program
            void exit();
    //Will remove the keyword, open the data file, call the appropriate query from above
            void show(string relationName, string query);
    //Will remove the keyword, create a new data file, write the first row as attributes, and select the primary key
            void create(string relationName, string[] attributeList, string primaryKey);
    //Will remove keywords, open the data file, select the attributes that satisfy the condition, and replace with content
            void update(string relationName, string attributes, string condition, string content);
    //Will remove the keyword, open the data file, and append the attribute list or other data file to the end of this file
            void insert(string relationName, string insertionContent);
    //Will remove the keyword, open the data file, and delete each line where the condition is true
            void delete(string relationName, string condition);
```

Database – The database consist of a series of text files with a *.db* extension. Each database file will have a first line containing the entity title, followed by a lost of the attributes.

[2] This is done in part 2 of the project

Reed Hampton
Grant Hruzek
Hanzhi Guo

Section 4 - Benefits, assumptions, risks/issues

Benefits:

1. A standalone client application for the user.
2. A small and light-weight simple database management system.
3. System log-generating functionality for “security”, historical, and backup purposes.
4. Convenient and user-friendly Command-Line Interface.
5. Ability to perform most conventional query operations on a database.

Risks/Assumptions:
There are a few risks associated with security concerns for designing the database system such as information hiding and implementing a secure database that cannot be viewed/manipulated by a user that does not have privileges but it is assumed that this would be out of scope for this project since the database is more of a proof-of-concept working design rather than a fully-functional, real-world, production database system.

Conclusion:
Several design elements will change over the course of development by the team such as the low-level details regarding implementation, but the higher-level design should, for the most part, essentially stay the same as we begin to dive further in the implementation stage of designing the database system.