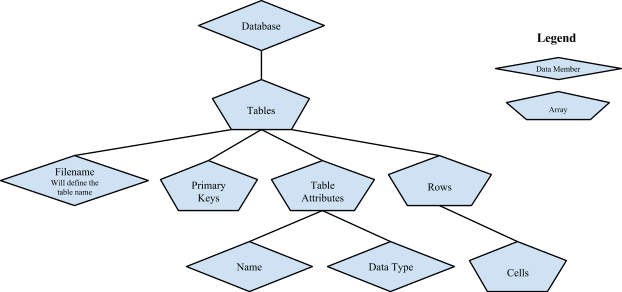
Relational Database Management System (RDBMS)

Design documents for Phases I and II

1. Project Objective

To create a database and an accompanying management system. The database will be then used to implement an application that uses the database system. We will be creating a car catalog application to demonstrate all the features the database has to offer and how it is managed through the use of our data definition language (DDL) and data manipulation language (DML).

In order to run the car catalog, our database must be able to store car data entries and create relation tables based on specific attributes. Specific cars in the database will also be related to user profiles through the use of search filters.The application will allow users to search and compare cars by these data attributes and hopefully find one they like. The database and its management system can later be used to develop many applications that require efficient data storage and manipulation.

2. High-level Design Overview (Phase I)

The entire database will be represented by an array of “table” objects. Each table will consist of a filename, which will define the name of the table, an array of attribute names (e.g., “firstName,” “lastName,” “email”), and an array of attribute data types (e.g., int, char, varchar). The table will also contain an array of rows, and each row will contain an array of cells, which will contain the data we are storing in the database.

3. Low-level Design Overview (Phase I)

**Summary:**

Recall that the database contains an array of all tables. A database has a number of functions that perform operations on tables by calling functions on table objects. The database also allows for direct access of table objects through the findTable function. In general, the functions in Database and Table are used in commands, and the functions used in TableOperations are used in queries.

**Database Functions:**

* void openTable(string filename)
  + Opens a table in local memory
* void closeTable(string tablename)
  + Writes table to file then erases from memory
* void writeTable(string tablename)
  + Writes table to file
* void showTable(string tablename)
  + Prints table to screen
* void createTable(string name, vector<string> attributeNames, vector<string> dataTypeNames, vector<string> primaryKeyNames)
  + Build a new table and adds it to memory
* void updateTable(string relationName, string attributeNameToChange, string valueToChange, string conditionAttributeName, string conditionValue)
  + Updates a given value of a table based on a condition
* void deleteFromTable(string name, string attributeName, string dataName)
  + Deletes a value from the table
* void insertIntoTable(string tablename, vector<string> values)
  + Calls Tables insert
* void insertIntoTable(string tablename, Table relation)
  + Inserts into the first table the value from the second
* bool tableExists(string name)
  + Checks if the table has been written to file
* void addTable(Table table)
  + Adds a table to memory
* Table& findTable(string name)
  + Returns a table from memory

**Table Functions:**

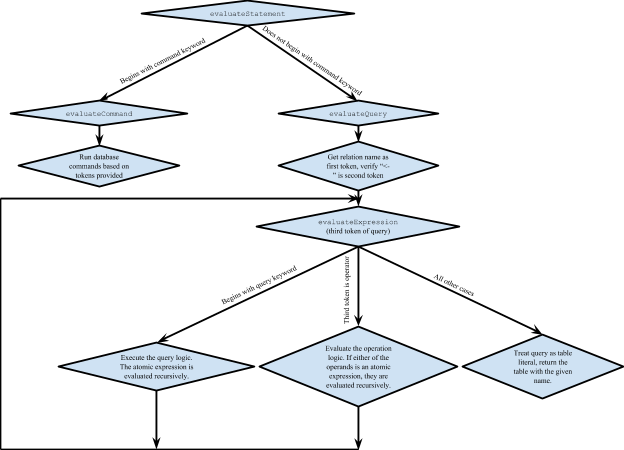
* Table(string name, vector<string> attributeNames, vector<string> dataTypeNames, vector<string> primaryKeyNames)
  + Table Constructor
* Table(string name, vector<TableAttribute> attributes, vector<string> primaryKeyNames)
  + Table Constructor with Attributes
* Table()
  + Default constructor
* Table(string tablename)
  + Default constructor with name
* string getName()
  + Returns a tables name
* void setName(string s)
  + Overwrites a tables name
* vector<TableAttribute>& getAttributes()
  + Returns a vector of the first row of a table
* vector<string> getPrimaryKeys()
  + Returns a vector of the Table’s keys data member
* vector< vector<string> > getTableData()
  + Returns a vector of vector containing all the Tables data
* void appendToRow(int row, vector<string> additions)
  + Adds new values to a row
* void addAttribute(vector<TableAttribute> attributes)
  + Adds a new attribute to the table
* void writeTable()
  + Writes the table to a file prepended with “.db”
* void showTable()
  + Prints the table to Output
* void insert(vector<string> values)
  + Inserts a new row into table
* void deleteFromTable(string attributeName, string dataName)
  + Deletes a value from the table
* int findAttributebyName(string attributeName)
  + Returns the position of the given attribute
* void changeAttributeName(string attributeName, string newAttributeName)
  + Overwrites the name of an attribute
* void update(int& changeAttributePos, int& conditionAttributePos, string& conditionAttributeData, string& changeAttributeData)
  + Updates a value in the table

**Table Operation Functions:**

* static Table select(Table targetTable, string conditionAttribute, string conditionOp, string condition)
  + Performs a selection operation on a table using a given condition
* static Table select(string attributesToInclude, Table targetTable)
  + Unused function
* static bool isNumber(const string s)
  + Determines if a string is numberic
* static bool entriesAreEqual(vector<string> entry1, vector<string> entry2)
  + Determines if two string vectors contain the same data
* static Table setUnion(Table table1, Table table2)
  + Returns set union of two tables
* static Table setDifference(Table table1, Table table2)
  + Returns set difference of two tables
* static Table crossProduct(Table table1, Table table2)
  + Returns cross product of two tables
* static vector<TableAttribute> attributeUnion(Table table1, Table table2)
  + Returns resulting union table attribute values
* static vector<TableAttribute> attributeIntersection(Table table1, Table table2)
  + Returns resulting intersection table attribute values
* static vector<TableAttribute> attributeDifference(Table table1, Table table2)
  + Returns resulting Difference table attribute values
* static Table naturalJoin(Table table1, Table table2)
  + Performs natural join on two tables
* static Table naturalJoin(Table table1, Table table2, string keyAttribute)
  + Unused function
* static Table project(Table table, string name)
  + Performs projection on a table
* static Table combineTables(Table table1, Table table2)
  + Appends the columns of table2 to table1, returns the result
* static Table renamingAttributes(Table table, vector<string> attributeNames)
  + Renames attributes of table to the ones provided in attributeNames

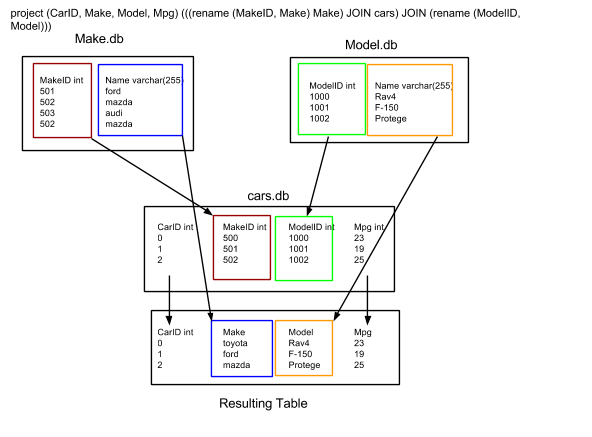
**Parser Functionality:**

The parser uses these functions to manipulate data in the database. The main interface between the application and the database is the evaluateStatement function in the parser. This function takes a statement as a string, written in our DML language, and performs the appropriate action based on it. The process the parser follows can be described by the following diagram:



Command keywords can be defined as one of the following: "OPEN", "CLOSE", "WRITE", "EXIT", "SHOW", "CREATE", "UPDATE", "INSERT", or "DELETE"

Query keywords can be defined as one of the following: "select", "project", or "rename"

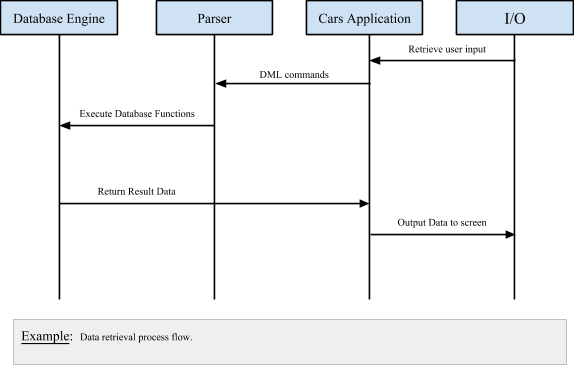
Below is an example of a parser command and a visual representation of its operation. The statement evaluates multiple JOIN functions on tables that have their attributes renamed before performing the joins. After all the tables are formatted and joined a project command is called to hide all the ID related attributes and just show actual car data.

4.Phase II Overview

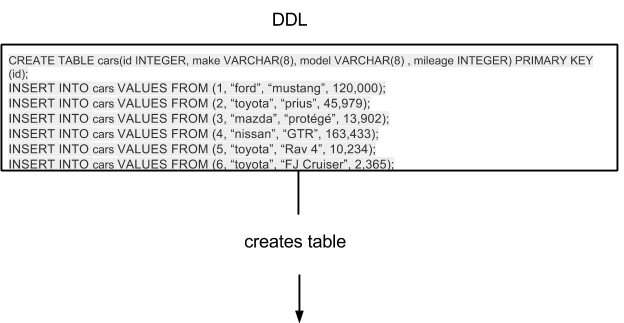
**Summary:**

Our application is a car database, similar to Edmunds.com. The user is presented with a menu with several options to choose from. Based on the option, the application will evaluate several statements against the database using our parser. Statements execute commands such as OPEN and SHOW as well as query operations like JOIN and set difference.

**Interaction:**



**Model:**



|  |  |  |  |
| --- | --- | --- | --- |
| Id | make | model | mileage |
| 1 | ford | mustang | 121,000 |
| 2 | toyota | prius | 45,979 |
| 3 | mazda | protégé | 13,902 |
| 4 | nissan | GTR | 163,433 |
| 5 | toyota | Rav 4 | 10,234 |
| 6 | toyota | FJ Cruiser | 2,365 |

4. Benefits & Issues

**Benefits:**

* A more quick and concise implementation of a database
* Because of our linear storage technique, the database is easier to troubleshoot and maintain
* Predefined attribute enumerations insures data type safety
* Our car application allows flexible entry of user’s car data
* Since all data is output to text files, the information is very portable

**Risks:**

* Performance risks because searching through ASCII text is less efficient than a binary tree.
* No network capabilities
* Memory is not encrypted so it is prone to security risks

**Assumptions:**

* That the user knows the correct data to be entered when queried
* That there will be sufficient memory for all data that needs to be stored

Post Production Notes

We started out having trouble with the design documents because we hadn't written any code yet and had trouble envisioning how our project would turn out. When we got the project running and the code written, it was easy to go back and modify the design documents to match the program we made. Our greatest difficulty came with writing the parser, and having an efficient method of parsing the different commands and queries. We solved it by creating methods that were able to distinguish between command and query input and run the appropriate evaluator. We learned many lessons in the project, including how to deal with two people writing code at the same time and the corresponding merge conflicts that resulted. We also learned about how big of a scale writing a database engine is. Our greatest lesson learned was from working with GitHub, and seeing how useful it is for us all to share our code instantaneously.

Work Load Distribution

* 30% Miguel
  + Wrote multiple relational operations and table manipulating helper functions.
  + Debugged and strengthened overall program robustness.
  + Wrote several functions in Database, Table, and in the Phase II implementation.
* 30% Devin
  + Created majority of program structure
  + Wrote several functions in Database, Table, and in the Phase II implementation
  + Wrote evaluateExpression in the parser, as well as several parser functions
* 25% Corey
  + Wrote several functions in Database, Table, and in the Phase II implementation.
  + Program debugging.
  + Wrote the combineTable function used in some operations
* 15% Sergio
  + Wrote several functions in Database, Table, and in the Phase II implementation