**CSCE 315 Assignment 2 Design Document**

**Team Information:**

**Team Leader:** Michael Stephens

**Team Members:** Anish Ali, Clifton Rapier, Michael Stephens, Nathan Kocmoud

**Lab Number:** 501

**Section 1 - Purpose**

Design a Relational Database Management System (RDBMS), a parser to use the designed system, and an application to utilize both of these to fulfill some real-world need. For our specific real-world need, we plan to create a database for a car dealership used to track sales and the relations involved with them including customers, cars, and car salesmen.

**Section 2 - Design of high level entities**

*PHASE I:*

**Entity Name:** Database

**Purpose:** To store data as a collection of relations

**Contains:** Relations

**Entity Name:** Relations

**Purpose:** To create entities and relationships between those entities

**Contains:** Tuples

**Entity Name:** Attribute

**Purpose:** To provide information about a row (one or more attribute will be used as a key in each relation)

**Entity Name:** Tuple

**Purpose:** To group related data in the form of cells into a row

**Contains:** Cells

**Entity Name:** Cell

**Purpose:** A container for the data in the tuple

**Entity Name:** Condition

**Purpose:** Created by the parser which passes them to the database in order to evaluate things

**Entity Name:** Runtime Exception

**Purpose:** To give universal error handling to the database

**Entity Name:** Parser

**Purpose:** To take in various strings and tokenize them in order to interpret commands from them and go on to communicate with the engine.

*PHASE II:*

**Entity Name:**  Dealership

**Purpose:** To connect to the parser and to add information (people, cars, salesmen, and the relationships between them) to the database / To get information involved with evaluating the performance of employees or the interest of the customers

**Section 3 - Design of lower levels of each entity**

*PHASE I:*

**Model:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parser**  ***Returns Void:***  **parser\_commandline()** - gets the line from the command window, uses parse\_command to parse it  **parse\_command(string)** - takes the command line and calls the tokenizer to get a list of tokens to use in the parse command  **parse(vector<string>)** - checks and sees if the line is a command or query and calls the various parse command associated with what is being requested  *Other functions:* **tokenize(string, vector<string>), parse\_create(), parse\_insert(), parse\_show(), parse\_delete(), parse\_update(), parse\_query(), parse\_open(), parse\_close(), parse\_write()**  ***Returns Bool:***  **is\_delimiter(char), is\_token(char), is\_first\_double\_token(char), is\_double\_token(char)**  ***Returns Relation:***  **evaluate\_atomic\_expression(vector string)** - this functions evaluates atomic expressions tokens by finding exactly what is being requested  ***Returns Misc:***  **create\_condition(vector<string>),create\_conjunction(vector<string>), create\_comparison(vector<string>)**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Database**  ***Returns Void:***  **add\_relation(Relation)** - adds an existing relation to the database  **create\_relation(string, vector<string>, vector<int>, vector<int>, vector<string>** - creates a new relation and adds it to the database  **drop\_relation(string)** - removes a relation from the database  **insert\_into(string, vector<string>)** - insert tuples into a relation in the database  **insert\_into(string, Relation)** - insert all the tuples from a relation into the database  **update(string, vector<string>, vector<string>, Condition)** - updates specific attributes of all tuples in a relation within the database  **delete\_from(string, Condition)** - deletes tuples of a certain condition from the database  **write\_relation(string)** - writes a relation to a file  ***Returns Int:***  **get\_num\_relations()**  ***Returns Relation:***  **select(Condition, Relation)** - checks if the tuples are being selected based on the conditions and returns a relation of those tuples  **project(vector<string>, Relation)** - returns a subset of the relation based on a condition  **rename(vector<string>, Relation)** - renames the relation by renaming all the attributes  **set\_union(Relation, Relation)** - returns a relation consisting of all the similar elements of two relations  **set\_difference(Relation, Relation)** - returns a relation consisting of the differing elements of two relations  **product(Relation, Relation)** - performs a cross product between two relations  **join(Relation, Relation)** - performs a natural join of two relations  *Other Functions:* **get\_relation(string)**   |  |  |  |  | | --- | --- | --- | --- | | **Relations**  ***Returns Void:***  **insert\_tuple(vector<string>)** - inserts tuples into a relation  **insert\_from\_relation(Relation)** - insert tuples from a relation to another relation  **delete\_from(Condition)** - deletes tuples within a certain condition  **update(vector<string>, vector<string>, Condition)** - updates attributes within a certain condition  **select(Condition, Relation)** - creates a relation using only the tuples associated with a certain condition  **project(vector<string, Relation)** - takes a relation and becomes a subset of it based on the attributes in the vector of strings  **rename(vector<string>, Relation)** - becomes the relation of a different name and inputs a vector to rename all the attributes  **natural\_join(Relation, Relation)** - relation becomes a natural join of the relation called on and the relation given  **rename\_relation(string)** - renames the relation  ***Returns Int:***  **find\_attribute\_column(string), get\_num\_tuples(), get\_num\_attributes()**  ***Returns Bool:***  **has\_attribute(string)**  ***Returns String:***  **get\_relation\_name(), get\_attributes(), get\_key\_list**  ***Returns Attribute:***  **get\_attributes\_list()**  ***Returns Tuple:***  **get\_tuple(), get\_tuple\_list()**  ***Returns Relation:***  **operator+(Relation, Relation)** - relation becomes the union of itself and given relation  **operator-(Relation, Relation)** - relation becomes the difference of itself and given relation  **operator\*(Relation, Relation)** - relation becomes the cross product of itself and given relation  ***Returns OStream:***  **operator<<(Relation)** - prints out the relation   |  |  |  | | --- | --- | --- | | **Attribute**  ***Returns Void:***  **update\_name(string)**  ***Returns Int:***  **get\_attribute\_type(), get\_attribute\_length()**  ***Returns Bool:***  **operator==(Attribute, Attribute)** - returns true if Attributes are equal  ***Returns String:***  **get\_attribute\_name()** | **Tuple**  ***Returns Void:***  **update\_cell(string, int)**  ***Returns Int:***  **get\_num\_cells()**  ***Returns Bool:***  **operator==(Tuple, Tuple)** - returns true if Tuples are equal  ***Returns String:***  **get\_values(), get\_cell\_data**  ***Returns Cell:***  **get\_cells()**  ***Returns OStream:***  **operator<<(Tuple)** - prints out the Tuple   |  | | --- | | **Cell:**  ***Returns Void:***  **update\_data(string)**  ***Returns Int:***  **get\_type()**  ***Returns Bool:***  **operator==(Tuple, Tuple)** - returns true if the cells are equal  ***Returns String:***  **get\_value** | | | | |

**Entity Name:** Database

**Usage:** The object will be used to create a database in memory. The database will be of the general form of Relations (Tables) with columns (Attributes) and rows (Tuples). The database will provide methods for creating and manipulating relations.

**Configuration:** A vector of Relations.

**Interaction:** The database will provide methods for creating new Relations as well as performing the following operations on relations (selection, projection, renaming, union, difference, cross product, and natural join).

**Entity Name:** Relations

**Usage:** This object will be used to define the different objects in the database as lower-level entities within the database by giving them specific data grouping. These entities can then be related to each other using whatever it was they were grouped by.

**Configuration:** A vector of Attributes and a map of Tuples

**Interaction:** A Relation will provide a way to add new Attributes and Tuples to the Relation as well as return or modify any other data the database will need.

**Entity Name:** Attribute

**Usage:** The attributes will be used to define the data that will be stored in the database and used to create the separate entities within the database and therefore also the relations between those entities.

**Configuration:** A string for name and a Domain.

**Interaction:** An Attribute will provide access to check its name and valid data for it’s Domain.

**Entity Name:** Tuple

**Usage:** This object will be used for storing data and each row in the database will consist of a tuple.

**Configuration:** A vector of Cells

**Interaction:** A tuple will enable you to retrieve data from it and save data to the appropriate fields in the tuple.

**Entity Name:** Cell

**Usage:** This object will be used for storing the data that each tuple contains.

**Configuration:** A struct that efficiently stores information for a data type.

**Interaction:** A cell will be the method in which data is stored as Tuples.

**Entity Name:** Database Parser

**Usage:** This object will be used to properly parse input from the user and correctly act on those commands by calling the correct functions on the Database. When the parser closes any executed commands for a particular relation will be saved to a file, while queries that generate views will see the views discarded.

**Configuration:** Will need the Database to call commands and access data based on what it parses from user input.

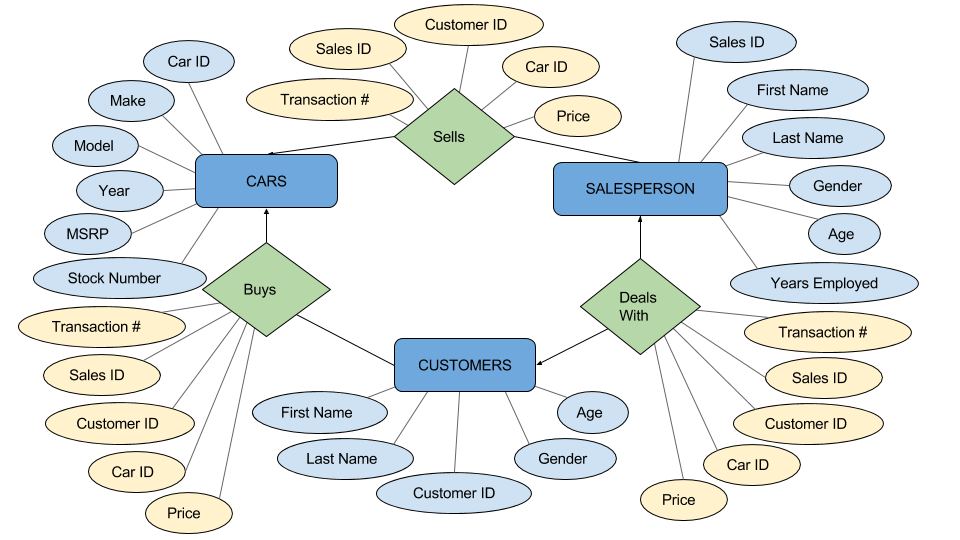
**Interaction:** Will provide an interface for calling commands using the RDBMS language.

*PHASE II:*

|  |
| --- |
| **Parser\***  ***Returns Void:***  **parser\_commandline(),parse\_command(string)**  **parse(vector<string>)**,**tokenize(string, vector<string>), parse\_create(), parse\_insert(), parse\_show(), parse\_delete(), parse\_update(), parse\_query(), parse\_open(), parse\_close(), parse\_write()**  ***Returns Bool:***  **is\_delimiter(char), is\_token(char), is\_first\_double\_token(char), is\_double\_token(char)**  ***Returns Relation:***  **evaluate\_atomic\_expression(vector string)**  ***Returns Misc:***  **create\_condition(vector<string>),create\_conjunction(vector<string>), create\_comparison(vector<string>)**  *\*see database model for more detailed descriptions of Parser methods* |

*↕Communication*

|  |
| --- |
| **Dealership *Returns Void:***  **open\_relation(string)** - opens a file containing commands and builds a database with those commands  **write\_to\_file(string)** - writes the commands generated by the application for creating the database to an output file  **show(string)** - takes a relation name and prints that relation  **add\_customer(), add\_salesperson(), add\_car(), add\_transaction()** - these methods prompt the user for various information involved with what they are trying to add and will create commands for the database to parse  **update\_customer(), update\_salesperson(), update\_car(), update\_transaction()** - these methods prompt the user for various information involved with updating the database and will create commands for the database to parse  **remove\_customer(), remove\_salesperson(), remove\_car(), remove\_transaction()** - these methods prompt the user for information and what they are trying to delete and will create commands for the database to parse  *Other Functions:***main\_menu(), car\_menu(), customer\_menu(), salespeople\_menu(), transaction\_menu(), statistics\_menu(), get\_customer(), get\_salesperson(), get\_car(), get\_transaction()** |



**Entity Name:** Dealership

**Usage:** Provide the command line GUI to edit the database and fetch requested data that will pertain to the interest of a car sale, will be able to add customers, salespeople, cars, and transactions to the database as well as print out various statistics involving those entities in order to evaluate performance of the employees and interests of the customers

**Interaction:** Will make calls through the parser to the database to edit and view the data. In the database it will create 3 relations for customers, salespersons, and cars that represent entities along with the relationships customers buy cars, salesperson sells cars, and customer deals with salesperson.

**Section 4 - Benefits, assumptions, risk/issues**

Benefits:

1. The Database does not need to allocate a large amount of memory initially, but rather it will grow as it needs to
2. Our design can be beneficial in that data can be somewhat protected over the different levels of Entities
3. Could be easier to find specific data by using a key
4. Reduces amount of time spent managing data
5. We can basically infinitely recursive as long as the query is formatted correctly and there’s enough memory

All known current risks:

1. The database could not allocate enough initial data to grow and could be a limited by memory constraints
2. Some possible bad formatting for conditions and atomic expressions which are both recursive which may cause crashes or not evaluate

Resolved Risks:

1. (Data won’t be saved until the close command is called, so potential loss of data could be an issue) - Unless the program crashes this shouldn’t be a problem, the dealership saves when it is destructed
2. (No exception handling or error checking has been discussed) - added a runtime exception class that we use at various times to throw errors and check them

All Assumptions:

1. We currently assume that the person inputting the data is actually competent and currently do have any necessary precautions to avoid human error
2. We assume that our data input will be very small compared to what real-world databases can handle. Thus we do not have any optimization features currently planned
3. We also assume that our program will have enough memory and that the hardware will work correctly without any errors