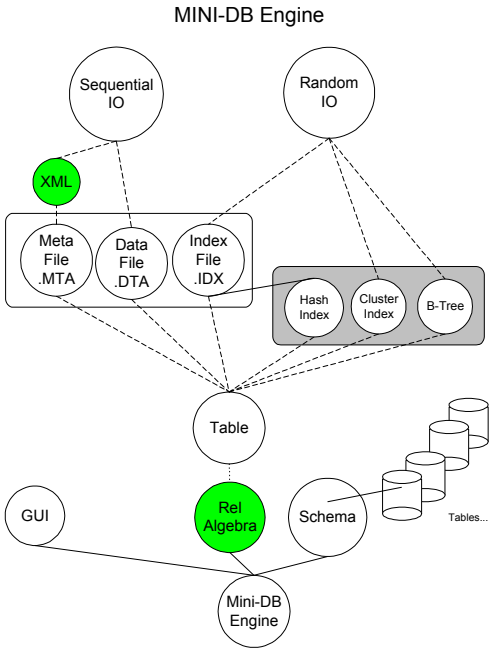
Assignment-3	MINI DB-ENGINE (Phase II) Building the Relational Algebra Class	

Develop a MINI_Relational_Algebra class. This class makes use the classes developed in (Phase I) assignment 2. The relational algebra operators can be separated in to two categories:

1) Update operations:
   (insert, delete, modify)
2) Retrieval operations:
   (select, project, cartesian product)



During Phase II, our goal is to implement the following operations:

```
class Mini_Rel_Algebra {

       bool Create(relation_name, schema);

       bool Insert(relation_name, attribute_list, value_list);
       int  Delete(relation_name, attribute_name, condition, attribute_value);
       bool Modify(relation_name, search_attribute_name, condition, search_attribute_value,
                           modify_attribute_list, modify_value_list);

       result_rel Select(relation_name, attribute_name, condition, attribute value);   // for extra
                                                          credit points, design your method to
                                                          accommodate repeating group of
                                                          attributes, conditions, and values.
       result_rel Project(relation_name, attribute_list);
       result_rel CartesianProduct(relation_name_1, relation_name_2);
};
```

Functional Specification:

- ***bool create(relation_name, schema);***

  Using the "schema" parameter such as:

  "TABLE_NM=^Table_Name~NUM_FILDS=^Number_of_Fields_In_Table~FN=^Field Name~FS=^Field Size~FT=^Field Type~..."

  build a XML based meta file, an Index file (on the key field) and a data file. Returns TRUE if Successful, FALSE otherwise.

  <TABLE_NAME>
  Table Name
  </TABLE_NAME>
  <NUM_FIELDS>
  Number_of_Fields_In_Table
  </NUM_FIELDS>
  <FIELD>
  <FIELD_NAME>
  Field Name
  </FIELD_NAME>
  <FIELD_SIZE>
  Field Size
  </FIELD_SIZE>
  <FIELD_TYPE>
  Field Type // for now don't worry about fields other than character strings.
  </FIELD_TYPE>
  </FIELD>

:
:
<PRIMARY_KEY>
Field Name
</PRIMARY_KEY>
<FOREIGN_KEY>
Field Name
<REFERENCES_FOREIGN_TABLE>
Table Name
</REFERENCES_FOREIGN_TABLE>
<FOREIGN_TABLE_ATTRIBUTE>
Field Name
</FOREIGN_TABLE_ATTRIBUTE>
</FOREIGN_KEY>

- ***bool Insert(relation_name, attribute_list, value_list);***

  This should be similar to insert in assignment 2, however with a twist. Allow multiple fields to be passed to the insert method. Perhaps a string separated by "^" and "~"!! For example:

  insert(employee, "NAME^SSN^PHONE~ ", "John^333227777^5742374554~"); Returns TRUE if Successful, FALSE otherwise.

  (Note that this would be a good time to check your insert information against your meta data about each field! So, if you want to refine your insert statement here, you want to check the fields against your meta-file) (not required at this time!)

- ***int Delete(relation_name, attribute_name, condition, attribute_value);***

  Two types of **delete** must be considered.

  In the first case, the attribute provided can be a key (we would know this if we looked at the meta-file). In this case, we would find the record in the index file and mark it as deleted.

  Hacker's Corner: In the second case, the attribute is a non-key attribute. In this case we would have to sequentially search the data-file, find the record(s) in the relation, with its attribute_value matching the specified value. (Note that this require you to know the schema of the table, so that you can determine which field to match against.) Once the record is found, you need to get its key and mark its corresponding index record as deleted (in the Index-file). Repeat the process for all matching records, and return the number of records deleted.

  For now the **condition** is only restricted to equality.

- *bool Modify(relation_name, search_attribute_name, condition, search_attribute_value, modify_attribute_list, modify_value_list);*

  Two types of **modify** must be considered.

  In the first case, the search-attribute-name provided can be a key (we would know this if we looked at the meta-file). In this case, we would append the new (modified record) at the end of the Data-file, then find the record in the index file and change its pointer to the new location of this record.

  Hacker's Corner: In the second case, the attribute is a non-key attribute. In this case we would have to sequentially search the data-file, find the record(s) in the relation, with its attribute_value matching the specified value. (Note that this require you to know the schema of the table, so that you can determine which field to match

against.) Once the record is found, you need to get its key, write the new record at the end of the data-file, and replace the old file pointer for this record in the index file. Repeat the process for all matching records, and return the number of records modified. (You must know when to stop. Remember not to modify the newly written records at the end of the file!!)

For now the **condition** is only restricted to equality.

Returns TRUE if you were successful to find the record and modify it, FALSE otherwise

- *result_rel Select(relation_name, attribute_name, condition, attribute value);*
  Let the user search for a field value. When a match is found extract the record(s) and place them into a new relation (for debugging purposes, also display the records). Return the resulting relation name.

  Two types of **select** must be considered.

  In the first case, the attribute-name provided can be a key. This of course is a trivial case.

  In the second case, the attribute is a non-key attribute. In this case we would have to sequentially search the data-file, find the record(s) matching the specified value, and insert them into a new relation.

  For now, the **condition** is only restricted to equality.

- *result_rel Project(relation_name, attribute_name1, ..., attribute_name_n.);*
  Project one or more columns from the relation. Returns a relation name which corresponds to the resulting relation. (for debugging also display the records)

- *result_rel* CartesianProduct
  Produce the cartesian product of two relations. Returns a relation name which corresponds to the resulting relation. (for debugging purposes, also display the records)

What to hand in:

- **Cover page** with proper title, your name, course # and name, assignment #, date, etc....
- **Source code** (documented)
- **Sample runs** (annotated if necessary)