**Syntax**

General Query GuidelinesAny number of spaces or returns can appear between words. All keywords and columns must be separated by at least one space. Commas do not have to be used when querying multiple columns. A semi-colon need not be used to denote the end of a query. Likewise, parenthesis need not be used. Of course if the user desires to include these elements, he/she can feel free to do this, since the application correctly parses these elements out. Any element the user wishes to create such as columns or table names with multiple words in them and strings that contain multiple spaces must be surrounded with brackets [ ]. The entire sum of keywords handled by the program is as follows: create, insert, values, select, from, where, orderby, delete, drop, set, import. These keywords are case-insensitive. Likewise, the entire database is case-insensitive, such that tables and columns and even querying string data present in a table will be selected without regard to exact casing constraints. Columns or tables to be selected can be renamed by appending a colon to the end of the column or table. Types consist but are not limited to short, int, long, float, double, decimal, datetime, timespan, string, char, and bool. Since datetime and timespan values might not be intuitive to users, these types and corresponding values are addressed in the section: Inserting Values. Operators that can be used in the **on** and **where** statements of the application are as follows: <, <=, >, >=, =, !=, and additionally for strings: startswith, endswith, contains, !startswith, !endswith, !contains.

### **Creating a Table**

To create a table type in the **create** keyword and then the table name. Subsequently, type in one or more pairs, where each pair is in the format: ColumnType ColumnName. The list of supported types are listed above.

|  |
| --- |
| ***Create*** *TableName Type1 Column1 Type2 Column2 ….* |

# Inserting Values

To insert values, type in the keyword **insert**, then the table name. To insert values based only on a subset of columns, type in columns that you desire to insert values for. Any other columns will contain the default value associated with them when inserted (empty string for columns of type string, one space for a char, and 0 for ints or floats). To insert values for all columns, you need not specify any column name. Now, type in any number of rows using the **values** keyword. To insert datetime values use the following syntax: mm/dd/yyyy, or optionally, [mm/dd/yyyy hour:minutes:seconds pm]. To insert timestamp values use the following syntax: days.hours:minutes:seconds.milliseconds.

|  |
| --- |
| ***Insert*** *TableName Column1 Column2 ...* ***Values*** *Value1 Value2 …* ***Values*** *Value1 Value2 … ….* |

# Querying Tables

To query tables you will need to make use of the **select**, **from**, **where**, **on**, **orderby** keywords. Many of these keywords are optional, but you must make sure to have at least one select keyword and one from keyword. You can also select \*, where \* selects all possible columns. Select must appear as the first keyword in the query. These function exactly like their SQL counterparts. The minute differences are as follows: To rename columns in the select statement, append the new name to the end of the column with a colon in between. To aggregate columns prepend one of the following to the column name: max!, min!, count!, sum!, avg!. When aggregating (using the previous aggregating functions) it will group rows based on the columns found in the orderby statement. To emulate the count(\*) functionality present in SQL, you can simply not use any orderby statements, and count a specific column: count!columnName. You cannot select columns that are not in the orderby statement without an aggregating function. Furthermore, be sure that if a column is ambiguous or exists in multiple tables that you are selecting, you specify the full column name: TableName.ColumnName to avoid ambiguity. Since the application is very sensitive to errors, an appropriate error message will prompt the user should invalid logical errors reside within a query. Aggregate columns can also be renamed by appending a colon and the alias onto the end of the column name.

|  |
| --- |
| ***Select*** *Table1.Column1 aggregate!Column2:alias ...* ***From*** *Table1 Table2:alias …* ***On*** *Table1.Column1 <= alias.Column2 and …* ***Where*** *Table1.Column1 > alias.Column2* ***OrderBy*** *Col1 Col2 …* ***Set*** *NewTempTable* |

When using the **from** keyword the following should be adhered to. One limitation is that you cannot join tables to themselves, even if you rename the tables to distinct names. To circumvent this issue, one can use the set keyword as extrapolated upon below. If you want to join multiple tables you can, but you must make use of the **on** keyword. Here you can place the column (or fully qualified column name if ambiguities exist) with an operator as specified in the previous sections. For expressions in the on keyword, you can only separate expressions with “and”. “Or’s” are not supported in the onstatement. Thus the join is similar to a theta join where you can join based on more than just equality between columns. You cannot perform cross products between tables and must specify an expression to link/join all tables together that are present in the from keyword.

Optionally, you can make use of the **where** keyword. This is similar to the on keyword except for the fact that you can separate expressions with any number of or’s and and’s. You cannot use parenthesis to force or’s to have a higher precedence than and’s. Thus the only form that is supported is disjunctive normal form. Still, any expression can be reduced to one that is in disjunctive normal form.

Moreover, the **orderby** keyword can optionally be used to simultaneously group and sort data on any number of columns. This will allow you to sort first by one column, then by another, then by another, where each group that is obtained from the previous sort is what is sorted on in the next sort. You can also optionally include the keyword asc, ascending, desc, or descending after each column to sort by a specific ordering. If no directional keyword is placed directly after a column, the default is assumed to be ascending.

Finally, the **set** keyword can optionally be used to store the resulting table (obtained from the query) temporarily in the database in memory until the program is closed and the table is eliminated. This will allow users to explore for more complex querying because subsequent queries can use this temporary table to join on other tables. Users cannot themselves delete, insert, or drop these temporary tables once they have been created. They can only be used in select queries.

### **Deleting Rows or Tables**

To delete rows from a table the keyword **delete** must be used. If no **where** statement is specified immediately after the delete keyword, all rows in the table are deleted, but the table structure is kept intact. If filter criteria are specified, only rows that meet the criteria will be permanently deleted from the table. To drop tables, or permanently remove them from the database, the use should use the **drop** keyword along with exactly one table. Similarly, the **delete** keyword must contain exactly one corresponding table.

|  |
| --- |
| ***Delete*** *Table* ***Where*** *expression(s)* ***Drop*** *Table* |

### **Importing XML Data**

One can specify an xml data source to obtain data. The application will then harvest the data and create a table-like structure along with inserting rows to store the data. This table can then be used to fully query upon relationally. The user should specify the xml data path where the xml file resides, the header tag that will denote a new row along with an optional “:alias” to rename the table in the database to something other than “header”. Furthermore, the user should specify at least one or more descendent tags from which to obtain the data. These will be the columns in the table. These can also be optionally renamed to a more suitable moniker. Since it would be too cumbersome to search every single value in order to determine the best type for each column, and further so that the user need not take on this responsibility themselves, each column is created of type string.

|  |
| --- |
| ***Import*** *FilePath.xml  Header:Table Desc1:Col1 Desc2:Col2 …* |