DSL to generate SQL SELECT Statements

Problem:

As we can see that the retail stores across use a sophisticated UI to get the details from the current warehouse and all of these from the UI inputs try to run the SQL statements on the database that is present in the local or the cloud storage.

We would want to solve this by having a simpler UI and a language that can be inhibited to the user which is like talking to the warehouse directly and this language on the input from the user can generate the SQL SELECT statements to search to the warehouse.

Need:

No as we see that the SQL statements to be written by a non tech user is never easy and even if we teach the simple statements to them and try to ask them to run it, it will be difficult to give access of the database to the non tech user which will put the security aspect of database in the risk.

Now the sophisticated UI in general to use by the user leaves the user with bunch of options and learning to be given. Now instead if we can provide them a language to communicate with the database as if they are communicating with a person it will be much easier and faster.

Assumptions:

The assumptions for the DSL is that we have been provided with the number of columns in the table that has been stored in the relational database. When we talk about the relational operators that are used in the system they include:

> | < | = | >= | <= | !=

When we talk about the logical operators we are specially interested in the two of the logical operators:

AND | **OR**

Concrete Syntax and Abstract Syntax:

As per our assumptions the user needs to specify the number of columns to be used in the select clause. For the user can enter the column names with a space if there are more than two.

For a simple select clause to fetch the columns from a table, the following will be the concrete syntax.

1. <Expression> (valid if its valid column name)
2. <Expression> <Expression>

The valid abstract syntax for the above two expression are:

1. (Id? Expression) -> #T ===> (id <expression>)
2. (AND ((Id? Expression) -> #T) ((Id? Expression) -> #T)) -> #T ===> (id <expression>) (id <expression>)

```
Welcome to DSL for SQL Statement-Select on the table : STORE

The columns present in the table are : ItemName ItemCost ItemQuantity ItemColor

Press q to quit

Column:> ItemName

Parsed Successfully : ( id ItemName )
Commands :
D: Done

Cond:> D
Ouput:> SELECT ItemName FROM STORE;



Column:> ItemName ItemColor

Parsed Successfully : ( id ItemName )( id ItemColor )
Commands :
D: Done

Cond:> D
Ouput:> SELECT ItemName,ItemColor FROM STORE;
```

The above is the application output. We have shown to generate one and more than one columns.

For the statements where we have a condition to be given the concrete syntax is as follows:

1. (Itemcost > 10)
2. (ItemName = “lays”)

The abstract syntax for the above is:

1. (AND (AND ((Id? <expression>) -> #T (relationalop? ‘>)-> #T) -> #T (number? 10)-->#T))-> #T ===> ((id <expression>) (relationalop>) (num 10))
2. (AND (AND ((Id? <expression>) -> #T (relationalop? ‘>)-> #T) -> #T (string? 10)-->#T))-> #T ===> ((id <expression>) (relationalop>) (string “lays”))

```
Column:> ItemName

Parsed Successfully : ( id ItemName )
Commands :
D: Done

Cond:> ( ItemCost > 10 )

Parsing . . . ( ItemCost > 10 )


Parsing . . . ItemCost

Parsed successfully : ItemCost


Parsing . . . >

Parsed successfully : >


Parsing . . . 10

Parsed successfully : 10

(( id ItemCost )( relationalop > )( num 10 ))

Ouput:> SELECT ItemName FROM STORE WHERE ( ItemCost > 10 );
```

```
Column:> ItemName

Parsed Successfully : ( id ItemName )
Commands :
D: Done

Cond:> ( ItemName = "lays" )

Parsing . . . ( ItemName = "lays" )


Parsing . . . ItemName

Parsed successfully : ItemName


Parsing . . . =

Parsed successfully : =


Parsing . . . "lays"

Parsed successfully : "lays"

(( id ItemName )( relationalop = )( string "lays" ))

Ouput:> SELECT ItemName FROM STORE WHERE ( ItemName = "lays" ) ;
```

The above snippets include the discussed possible relational conditions.

For the statements where we have multiple conditions to be given the concrete syntax is as follows:

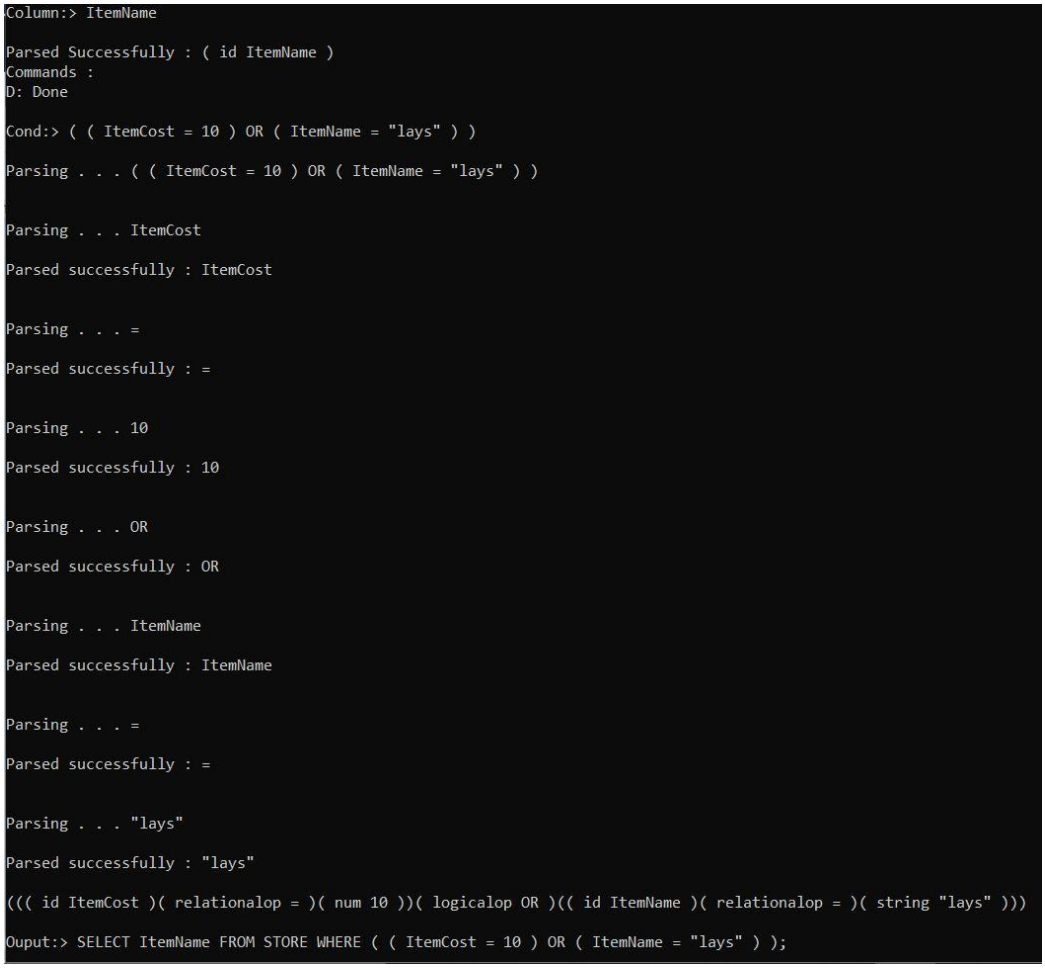
1. ((Itemcost > 10) AND (ItemName = “lays”))
2. ((Itemcost > 10) OR(ItemName = “lays”))

The abstract syntax for the above is:

1. (AND (AND (Condition? <expression>)-> #T (logicalop? ‘AND)->#T)(Condition? <expression>)->#T) -> #T ===> ((id <expression>) (relationalop>) (num 10)) (logicalop AND) ((id <expression>) (relationalop>) (string “lays”)) [Here the condition as in its routing to the step 2 of abstract syntax]
2. (AND (AND (Condition? <expression>)-> #T (logicalop? OR)->#T)(Condition? <expression>)->#T) -> #T ===> ((id <expression>) (relationalop>) (num 10)) (logicalop OR) ((id <expression>) (relationalop>) (string “lays”)) [Here the condition as in its routing to the step 2 of abstract syntax]

```
Column:> ItemName

Parsed Successfully : ( id ItemName )
Commands :
D: Done

Cond:> ( ( ItemCost > 10 ) AND ( ItemName = "lays" ) )

Parsing . . . ( ( ItemCost > 10 ) AND ( ItemName = "lays" ) )


Parsing . . . ItemCost

Parsed successfully : ItemCost


Parsing . . . >

Parsed successfully : >


Parsing . . . 10

Parsed successfully : 10


Parsing . . . AND

Parsed successfully : AND


Parsing . . . ItemName

Parsed successfully : ItemName


Parsing . . . =

Parsed successfully : =


Parsing . . . "lays"

Parsed successfully : "lays"

((( id ItemCost )( relationalop > )( num 10 ))( logicalop AND )(( id ItemName )( relationalop = )( string "lays" )))

Ouput:> SELECT ItemName FROM STORE WHERE ( ( ItemCost > 10 ) AND ( ItemName = "lays" ) );
```

The above snippet shows the example for the AND condition in the application.



The above example show the snippet of the OR statement.

Error Handling:

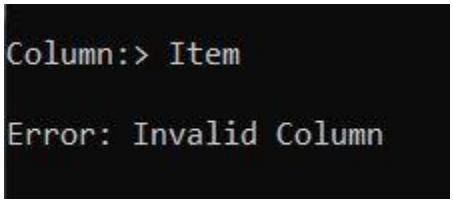
The major error handling we are checking is for the parenthesis and if its a valid abstract syntax or not. For the parenthesis errors, we maintain a stack to check if the valid parenthesis are in place or not.

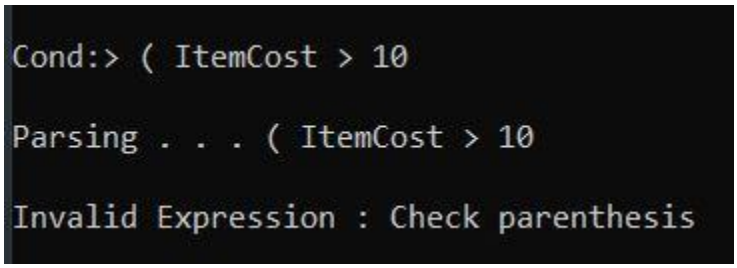
For every valid pair of parenthesis we check if its a valid abstract syntax or not. For each of these errors we raise an error and restart the application.

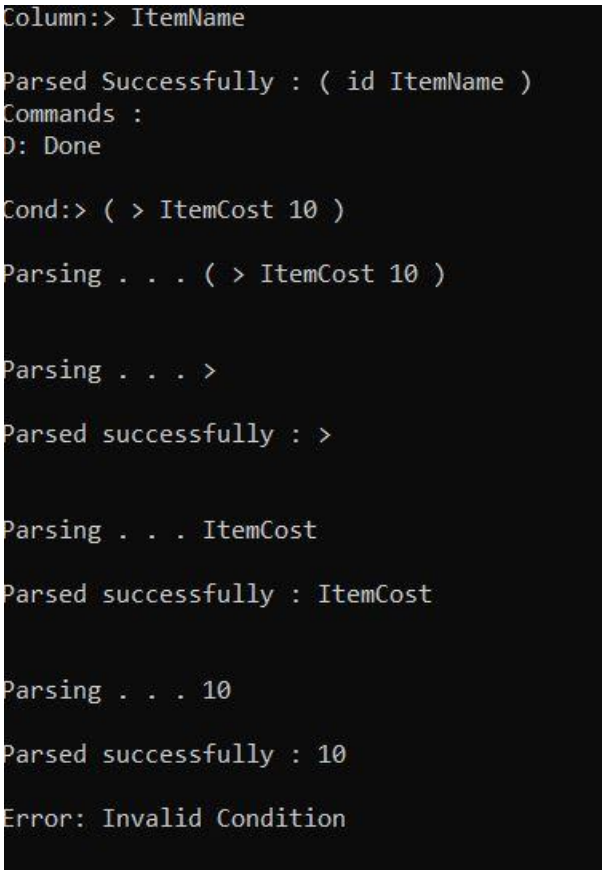
There different functions to check for the parenthesis error, to check the type of the expresson number or string, to check if the given expression is a valid column name or not, to see if the given expression follows the abstract syntax or not.

All of these functions are check during the course of interpreter. It outputs a valid select statement syntax only if the parsed syntax have got not errors.

Screenshots:







There are many other possible cases for error occurrence. Presenting here a few of them to showcase how exactly the error occurs. We have covered three types of errors here and on the full run of the program we can also have able to see all the other errors.