Dare to be first.

UNIVERSITY OF DELAWARE

Creating an SQL Parser for Information Analytics

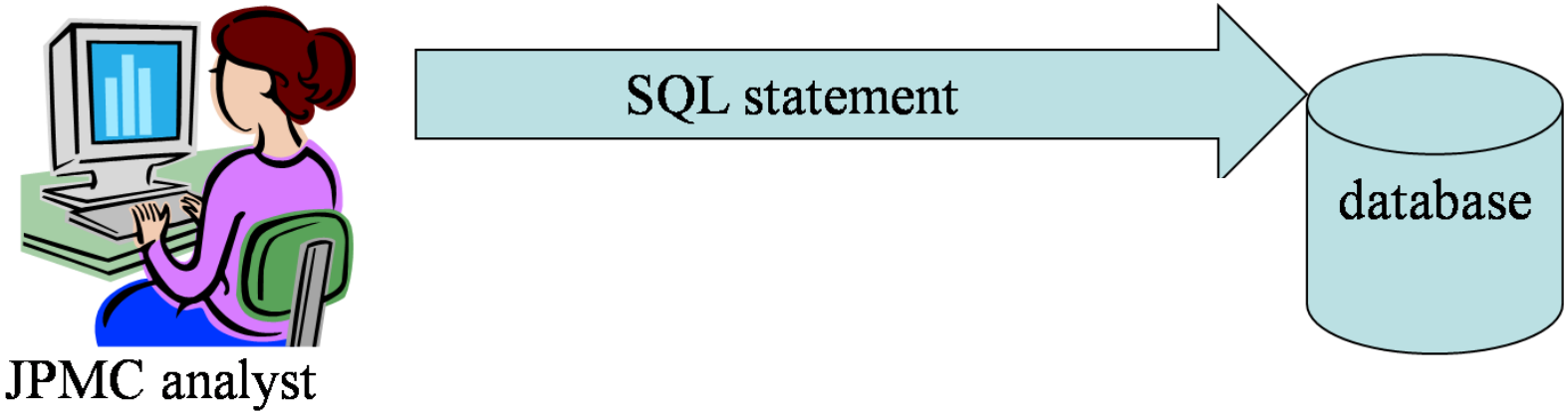
Wei Wang

Oct 7, 2011

Ongoing work with ECE and JPMC

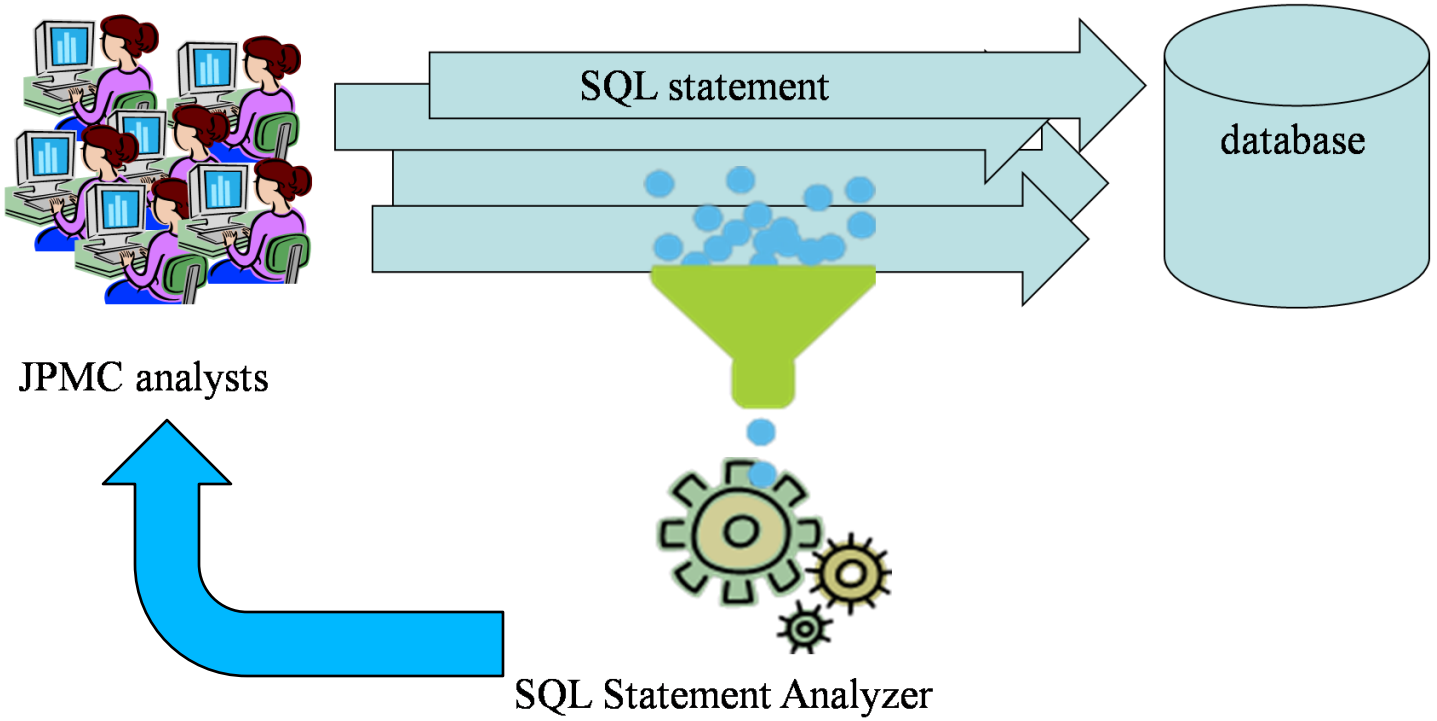
Motivation

- Understand how JPMC analysts use information
  - Will improve business processes
- Previous attempts were unsatisfactory
- Analytics derived from analyzing SQL statements written/executed by JPMC analysts



SQL Analyzer

- Build effective parser that recognizes SQL
- Build suite of information analytic engines



Parser and Analytic Engines

Mechanisms

- Develop an SQL grammar
- Parse queries into parse trees for analysis
- Analytic Engines process parse trees
  - Tree/Graphical analysis
  - Social network analysis

ANTLR Overview

- Another Tool for Language Recognition
- ANTLR takes as input a grammar that specifies a language
- Generates as output source code for a recognizer for that language.
- Grammar for SQL Statements

Example: SQL select

```
SELECT  ACCT_REF_NB, DCLSE_GRP_ID FROM ACCT_CURR;
```

Example: SQL select grammar

A Reasonable Grammar:

```
SELECT  ACCT_REF_NB, DCLSE_GRP_ID FROM ACCT_CURR;
```

```
SELECT_STATEMENT
 : k_select SELECT_LIST k_from TABLE_LIST;
SELECT_LIST: COLUMN (COMMA COLUMN)*;
COLUMN: 'A' .. 'Z' ( 'A' .. 'Z' | '0' .. '9' )*;
TABLE_LIST: 'A' .. 'Z' ( 'A' .. 'Z' )*;
COMMA: ,;
```

```
CLASS
 ::= CLASS TYPEID:n LBRACE feature_list:f RBRACE SEMI
 | …
 ;
```

Overall SQL Select Grammar Syntax

```
select_statement
        :
        subquery_factoring_clause?
        sel=k_select  ( k_distinct | k_union | k_all )? select_list
        k_from
        table_reference_list
        ( where_clause )?
        ( hierarchical_query_clause )?
        ( group_by_clause )?
        ( k_having sql_condition )?
        ( model_clause )?
        ( union_clause )?
        ( order_by_clause )?
        ( for_update_clause )?
```

```
where_clause
        :       k_where sql_condition
```

```
sql_condition
        :       ...
```

Example: SQL insert

```
insert into user_cpu_time (spid,cum_time)
values (233672,270 );
```

Example: SQL insert grammar

Grammar:

```
insert into user_cpu_time (spid,cum_time)
Values (233672,270 );
```

```
INSERT_STATEMENT
 : k_insert k_into T_NAME
      L_PAREN COLUMN (COMMA COLUMN)* RPAREN
      k_values LPAREN EXPR (COMMA EXPR)* RPAREN
COLUMN: 'a' .. 'z' ( 'a' .. 'z' | '0' .. '9' )*;
T_NAME: 'a' .. 'z' ( 'a' .. 'z' | '_' )*;
EXPR
: NUMBER;
NUMBER: '0' .. '9' ('0' .. '9')*;
LPAREN: (;
RPAEEN: );
```

Overall SQL insert Grammar Syntax

insert_statement
:ins=k_insert (single_table_insert | multi_table_insert);
single_table_insert
: insert_into_clause (values_clause returning_clause? | subquery |select_statement) error_logging_clause?
insert_into_clause
: p_into=k_into dml_table_expression_clause t_alias? (LPAREN column_name (COMMA column_name)* RPAREN)?
values_clause
: p_values=k_values LPAREN default_sql_expr (COMMA default_sql_expr)* RPAREN
default_sql_expr
:(p_default = k_default | sql_expression)
...

Steps to generate ANTLR Parser

1. Write SQL grammar for ANTLR (oracle.g)
2. Run ANTLR to convert .g to parser
   - Generate an executable (the parser)

Similar?

1. Write cool grammar for JavaCup (cool.cup)
2. Call JavaCup to convert .cup to parser
   - java parser.JavaCup parser/cool.cup
   - Generate CoolParser.java and Sym.java
3. Generate java class files

Generating Parse Tree

Run parser on file and dump output

```
>> ./a.out sql.txt > sql-parser.out
```

Run parser on cool file

```
>>  java parser.Parser hello_world.cl
```

sql-parser.out

- Output of ANTLR Parser (text represents tree)

```
Nodes: (T_SELECT SELECT (T_COLUMN_LIST
(T_SELECT_COLUMN ACCT_REF_NB) , (T_SELECT_COLUMN
DCLSE_GRP_ID)) (T_FROM FROM ACCT_CURR)) ;

// nil-node
//      T_SELECT (8, T_SELECT) [1,0] (3)
//        SELECT (4, T_RESERVED) [1,0] (0)
//        T_COLUMN_LIST (9, T_COLUMN_LIST) [1,8] (3)
//          T_SELECT_COLUMN (10, T_SELECT_COLUMN) [1,8] (1)
//            ACCT_REF_NB (71, ID) [1,8] (0)
//          , (54, COMMA) [1,19] (0)
//          T_SELECT_COLUMN (10, T_SELECT_COLUMN) [1,21] (1)
//            DCLSE_GRP_ID (71, ID) [1,21] (0)
//        T_FROM (11, T_FROM) [1,34] (2)
//          FROM (4, T_RESERVED) [1,34] (0)
//          ACCT_CURR (71, ID) [1,39] (0)
//      ; (49, SEMI) [1,48] (0)


SELECT  ACCT_REF_NB, DCLSE_GRP_ID FROM ACCT_CURR;
```

```
#5
_program
 #5
 _class
  Main
  IO
  "hello_world.cl"
  (
  #4
  _method
   main
   SELF_TYPE
   #4
   _dispatch
    #4
    _object
     self
    : _no_type
    out_string
    (
    #3
    _string
     "Hello, World.\n"
    : _no_type
    )
   : _no_type
  )
```

Steps to Visualize the Tree

- Processing output of ANTLR grammar (text represented tree)
  - Nodes and Edges of the tree more organized

```
1 T_SELECT T_SELECT 0
2 SELECT T_RESERVED 1
3 T_COLUMN_LIST T_COLUMN_LIST 1
4 T_SELECT_COLUMN T_SELECT_COLUMN 3
5 ACCT_REF_NB ID 4
6 , COMMA 3
7 T_SELECT_COLUMN T_SELECT_COLUMN 3
8 DCLSE_GRP_ID ID 7
9 T_FROM T_FROM 1
10 FROM T_RESERVED 9
11 ACCT_CURR ID 9
12 ; SEMI 0
```

Steps to Visualize the Tree (cont'd)

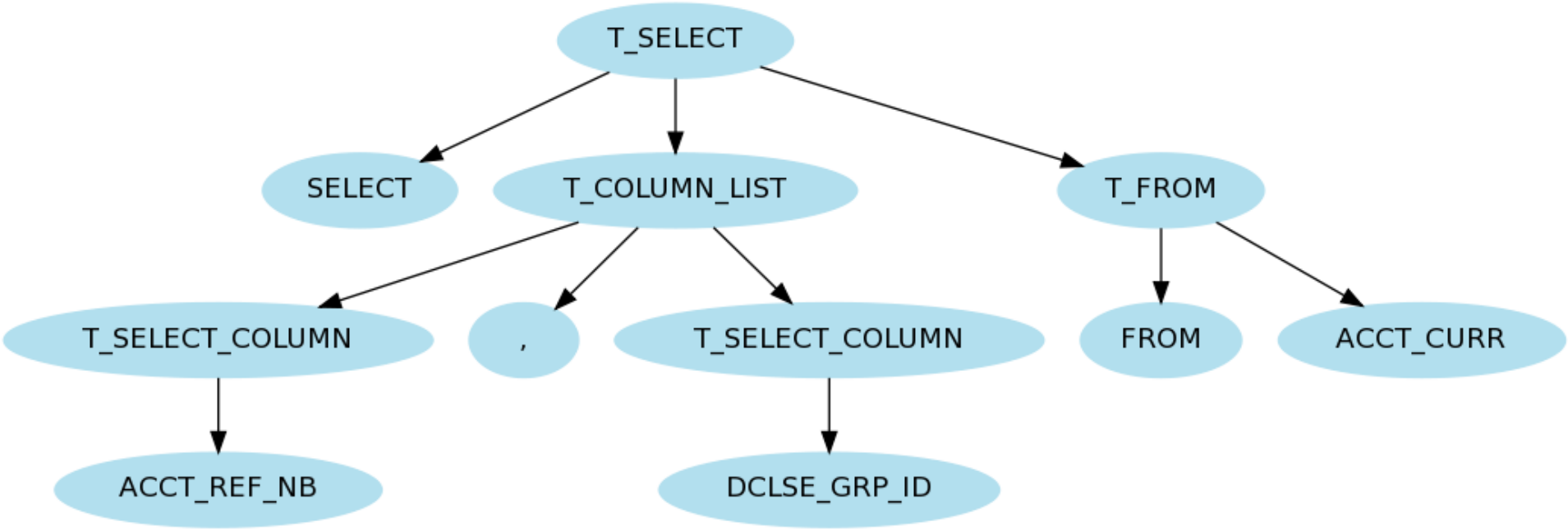
- Get input for .viz file type from the previous output

```
Node1 [label = "T_SELECT" ];
Node2 [label = "SELECT" ];
Node3 [label = "T_COLUMN_LIST" ];
Node4 [label = "T_SELECT_COLUMN" ];
Node5 [label = "ACCT_REF_NB" ];
Node6 [label = "," ];
Node7 [label = "T_SELECT_COLUMN" ];
Node8 [label = "DCLSE_GRP_ID" ];
Node9 [label = "T_FROM" ];
Node10 [label = "FROM" ];
Node11 [label = "ACCT_CURR" ];
```

```
Node1 -> Node2 ;
Node1 -> Node3 ;
Node3 -> Node4 ;
Node4 -> Node5 ;
Node3 -> Node6 ;
Node3 -> Node7 ;
Node7 -> Node8 ;
Node1 -> Node9 ;
Node9 -> Node10
;
Node9 -> Node11 ;
```

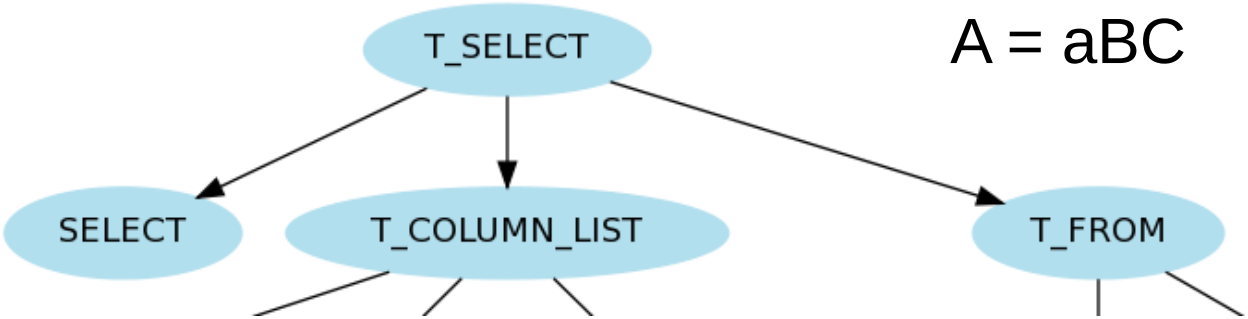
Visualizing the Parse Tree

Run command: dot -Tpng -o filename.png filename.viz

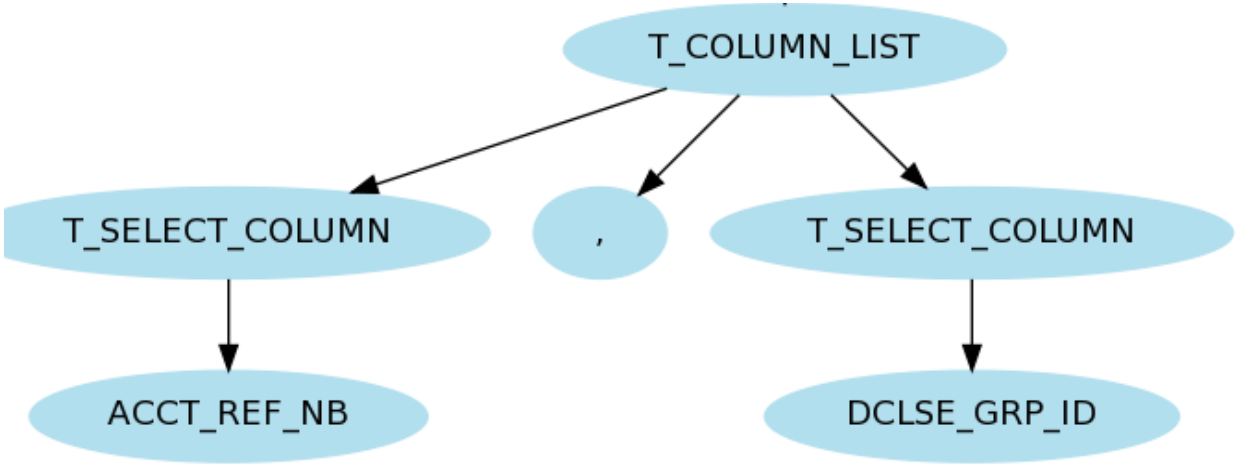


SELECT ACCT_REF_NB, DCLSE_GRP_ID FROM ACCT_CURR;

SQL Parse Tree



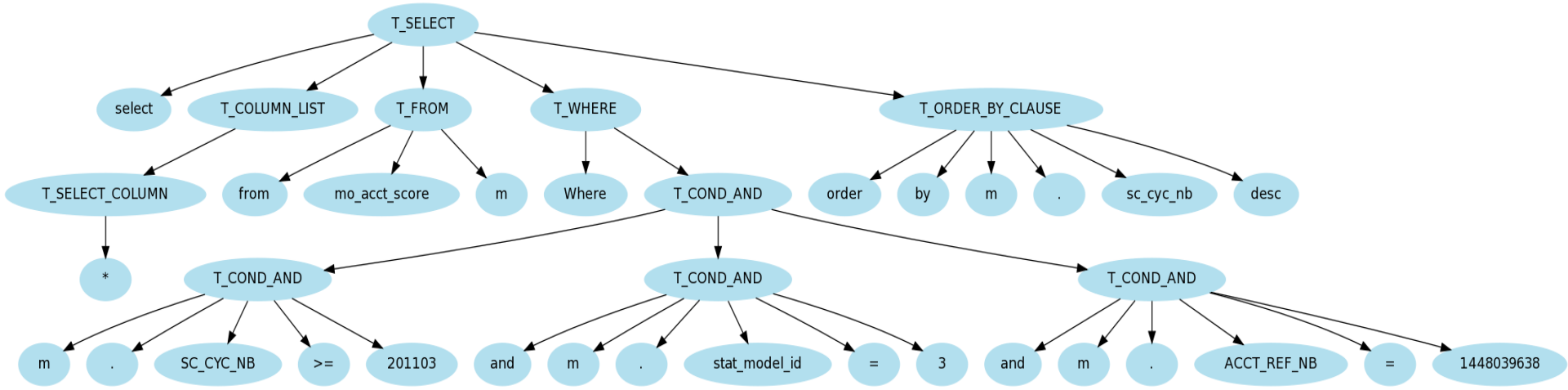
A = aBC



B = D(,D)*

JPMC Parser vs. UD SQL Parser

- The original parser parses 60% of queries: We are able to process about 99% (1948/1974)
- Ours is based on grammar for Oracle's SELECT statement for ANTLR v3



ANTLR Overview