Chapter 2 Relational Algebra Interpreter

A relational algebra interpreter is presented in this chapter. The interpreter is implemented in Java using the parser generator tools JCup and JFlex. The system is simple to use and comes with a database engine that implements a set of basic relational algebraic operators. The relational algebra interpreter reads as input a query in relational algebra and performs the following three steps:

(1) Syntax Checking: The syntax of the query is verified in this step.
(2) Semantics Checking: Type checking and verification of valid column references are performed in this step.
(3) Query Evaluation: The query is evaluated using the database engine and the results are displayed.

Relational Algebra Syntax

A subset of Relational Algebra that includes the union, minus, intersect, Cartesian product, natural join, select, project, and rename operators is implemented in the interpreter. The context-free grammar for this subset is shown below:

```
<Query> ::= <Expr> SEMI;
<Expr>  ::= <ProjExpr>  | <RenameExpr>    | <UnionExpr> |
            <MinusExpr> | <IntersectExpr> | <JoinExpr> |
            <TimesExpr> | <SelectExpr>    | RELATION
<ProjExpr>   ::= PROJECT [<AttrList>] (<Expr>)
<RenameExpr> ::= RENAME [<AttrList>] (<Expr>)
<AttrList>   ::= ATTRIBUTE | <AttrList> , ATTRIBUTE
<UnionExpr>  ::= (<Expr> UNION <Expr>)
<MinusExpr>  ::= (<Expr> MINUS <Expr>)
<IntersectExpr>  ::= (<Expr> INTERSECT <Expr>)
<JoinExpr>   ::= (<Expr> JOIN <Expr>)
<TimesExpr>  ::= (<Expr> TIMES <Expr>)
<SelectExpr> ::= SELECT [<Condition>](<Expr>)
<Condition>  ::= <SimpleCondition> |
                 <SimpleCondition> AND <Condition>
<SimpleCondition> ::= <Operand> <comparison> <Operand>
<Operand>    ::= ATTRIBUTE | STRING-CONST | NUMBER-CONST
<Comparison> ::= < | <= | = | <> | > | >=
```

All relational algebra queries must be terminated with a semi-colon. A relational algebra query in the simplest form is a “relation name”. For example the following terminal session with the interpreter illustrates the execution of this simple query form:

```
[~/elmasri-navathe/ra][12:05pm] java RA company
RA> departments;
SEMANTIC ERROR in RA Query: Relation DEPARTMENTS does not exist
```

```
RA> department;
DEPARTMENT(DNAME:VARCHAR,DNUMBER:INTEGER,MGRSSN:VARCHAR,MGRSTARTDA
TE:VARCHAR)

Number of tuples = 6
Research:5:333445555:22-MAY-1978:
Administration:4:987654321:01-JAN-1985:
Headquarters:1:888665555:19-JUN-1971:
Software:6:111111100:15-MAY-1999:
Hardware:7:444444400:15-MAY-1998:
Sales:8:555555500:01-JAN-1997:

RA> exit;
[~/elmasri-navathe/ra][12:06pm]
```

More complicated relational algebra queries involve one or more applications of one or more of the several operators such as select, project, times, join, union, etc. For example, consider the query "*Retrieve the names of all employees working for Dept. No. 5*". This would be expressed by the query execution in the following RA session:

```
RA> project[fname,lname](select[dno=5](employee));
temp1(FNAME:VARCHAR,LNAME:VARCHAR)

Number of tuples = 4
Franklin:Wong:
John:Smith:
Ramesh:Narayan:
Joyce:English:

RA>
```

The RA interpreter assigns temporary relation names such as `temp0, temp1`, etc. to each intermediate relation encountered in the execution of the entire query. The RA interpreter also employs the following rules as far as naming of attributes/columns of intermediate relations:

1. Union, Minus, and Intersect: The attribute/column names from the left operand are used.
2. Times (Cartesian Product): Attribute/Column names from both operands are used. Attribute/Column names that are common to both operands are prefixed by relation name (`tempN`).
3. Select: Same attribute/column names as that of the operand.
4. Project, Rename: Attribute/Column names present in the attribute list parameter of the operator. Duplicate attribute/column names are not allowed in the attribute list.
5. Join (Natural Join): Attribute/Column names from both operands are used. Common attribute/column names appear only once.

As another example, consider the query "*Retrieve the social security numbers of employees who either work in department 5 or directly supervise an employee who works in department 5*". The query is illustrated in the following RA session:

```
RA> (project[ssn](select[dno=5](employee))
RA> union project[superssn](select[dno=5](employee)));
temp4(SSN:VARCHAR)

Number of tuples = 5
333445555:
123456789:
666884444:
453453453:
888665555:

RA>
```

The queries from Section 6.5 of the Elmasri-Navathe text modified to work with RA are shown below:

Query 1: Retrieve the name and address of employees who work for the "Research" department.

```
project[fname,lname,address](
  (rename[dname,dno,mgrssn,mgrstartdate](
      select[dname='Research'](department))
   join
   employee
  )
);
```

Query 2: For every project located in "Stafford", list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
project[pnumber,dnum,lname,address,bdate](
  (
   (select[plocation='Stafford'](projects)
    join
    rename[dname,dnum,ssn,mgrstartdate](department)
   )
   join employee
  )
);
```

Query 3: Find the names of employees who work on all the projects controlled by department number 5.

```
project[lname,fname](
 (employee
  join
  (project[ssn](employee)
```

```
   minus
   project[ssn](
    (
      (project[ssn](employee)
       times
       project[pnumber](select[dnum=5](projects))
      )
      minus
      rename[ssn,pnumber](project[essn,pno](works_on))
    )
   )
  )
 )
);
```

Query 4: Make a list of project numbers for projects that involve an employee whose last name is "Smith", either as a worker or as a manager of the department that controls the project.

```
( project[pno](
   (rename[essn](project[ssn](select[lname='Smith'](employee)))
    join
    works_on
   )
  )
 union
  project[pnumber](
   ( rename[dnum](project[dnumber](select[lname='Smith'](
       (employee
        join
        rename[dname,dnumber,ssn,mgrstartdate](department)
       )
       )
       )
     )
     join
     projects
    )
  )
);
```

Query 5: List the names of all employees with two or more dependents.

```
project[lname,fname](
(rename[ssn](
 project[essn1](
  select[essn1=essn2 and dname1<>dname2](
   (rename[essn1,dname1](project[essn,dependent_name](dependent))
```

```
    times
    rename[essn2,dname2](project[essn,dependent_name](dependent)))
   )
  )
 )
join
employee)
);
```

Query 6: Retrieve the names of employees who have no dependents.

```
project[lname,fname](
 ( ( project[ssn](employee)
     minus project[essn](dependent)
   )
   join
   employee
 )
);
```

Query 7: List the names of managers who have at least one dependent.

```
project[lname,fname](
  ((rename[ssn](project[mgrssn](department))
    join
    rename[ssn](project[essn](dependent))
   )
  join
  employee
  )
);
```