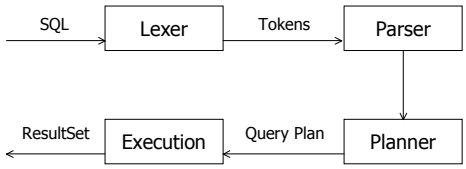
# CS422 Principles of Database Systems

Introduction to Query Processing

Chengyu Sun
California State University, Los Angeles

## Query Processing in SimpleDB




## Schema

- Departments( did, dname )
- Students( sid, sname, dept )

## SQL

```
create table departments (
        did             int;
        dname           varchar(10)
);

select sname, dname
        from students, departments
        where dept = did and sid = 1;
```

## Query Parsing

- Analyze the query string and convert it into some data structure that can be used for query execution
- Syntax
  - A set of rules that describes the strings that could *possibly* be meaningful statements
  - Example: a syntactically wrong statement

```
select from a and b where c - 3;
```

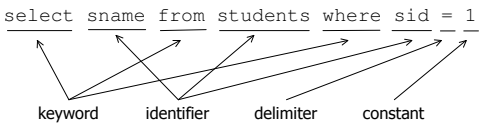
## Semantics

- The *semantics* of a language specify the meaning of a syntactically correct string
- *Is the following statement semantically correct??*

```
select * from a, b where c = 3;
```

## Lexical Analysis

- Split the input string into a series of tokens




## Token

- <type, value>

| Type | Value |
|---|---|
| keyword | select |
| identifier | sname |
| keyword | from |
| identifier | students |
| keyword | where |
| identifier | id |
| delimiter | = |
| intconstant | 1 |

## Lexer API

- Iterate through the tokens
  - Check the current token – "Match"
  - Consume the current token – "Eat"

```
select sname from students where sid = 1
```

current token

```
lexer.matchKeyword("select");
lexer.eatKeyword("select");
```

## SimpleDB Grammar …

```
<Field>      := IdTok
<Constant>   := StrTok | IntTok
<Expression> := <Field> | <Constant>
<Term>       := <Expression> = <Expression>
<Predicate>  := <Term> [ AND <Predicate> ]
```

## … SimpleDB Grammar

```
<Query>     := SELECT <SelectList> FROM <TableList>
               [ WHERE <Predicate> ]
<SelectList> := <Field> [ , <SelectList>]
<TableList>  := IdTok [ , <TableList> ]

<CreateTable> := CREATE TABLE IdTok ( <FieldDefs> )
<FieldDefs>   := <FieldDef> [ , <FieldDefs> ]
<FieldDef>    := IdTok <TypeDef>
<TypeDef>     := INT | VARCHAR ( IntTok )
```

## Using Grammar

- Which of the following are valid SimpleDB SQL statements??

```
create table students (id integer, name varchar(10));
insert into students (1, 'Joe');
select * from students;
```

## From Grammar to Code …

```
public QueryData query()
{
   lex.eatKeyword( "select" );
   Collection<String> fields = selectList();
   lex.eatKeyword( "from" );
   Collection<String> tables = tableList();
   Predicate pred = new Predicate();
   if( lex.matchKeyword("where") )
   {
      lex.eatKeyword("where");
      pred = predicate();
   }
   return new QueryData( fields, tables, pred );
}
```

## … From Grammar to Code

```
public Collection<String> selectList()
{
   Collection<String> L = new ArrayList<String>();
   L.add( field() );
   if( lex.matchDelim(',') )
   {
      lex.eatDelim(',');
      L.addAll( selectList() );
   }
   return L;
}

public String field() { return lex.eatId(); }
```

## Create Table

- Input: table name and Schema
- Create a record file for the table
- Insert the table information into system catalog

## System Catalog

- A.K.A. data catalog, data dictionary
- A set of *tables* containing metadata about the schema elements and data statistics
  - Table, field, view, index information
  - Data statistics
    - E.g. total number of rows in a table and distinct values in a column
    - Used for query optimization

## System Catalog Example

- `tblcat` and `fldcat` in SimpleDB
  - tblcat (TblName, RecLength)
  - fldcat (TblName, FldName, Type, Length, Offset)

## Query Planning

- Break a query into *individual operations*, and organize them into certain order, i.e. a query plan.

## Relational Algebra Operations

- Selection, projection, product
- Join
- Rename
- Set operations: union, intersection, difference
- Extended Relation Algebra operations
  - Duplicate elimination
  - Sorting
  - Extended projection, outer join
  - Aggregation and grouping

## Selection

Input

| sid | sname |
|---|---|
| 1 | Joe |
| 2 | Amy |

sid=1 →

Output

| sid | sname |
|---|---|
| 1 | Joe |

## Projection

Input

| sid | sname |
|---|---|
| 1 | Joe |
| 2 | Amy |

sname →

Output

| sname |
|---|
| Joe |
| Amy |

## Product

Input

| sid | sname | dept |
|---|---|---|
| 1 | Joe | 10 |
| 2 | Amy | 20 |

| did | dname |
|---|---|
| 10 | CS |
| 20 | Math |

⇩

Output

| sid | sname | dept | did | dname |
|---|---|---|---|---|
| 1 | Joe | 10 | 10 | CS |
| 1 | Joe | 10 | 20 | Math |
| 2 | Amy | 20 | 10 | CS |
| 2 | Amy | 20 | 20 | Math |

## Scan

- A scan is an interface to a RA operation implementation

```
public interface Scan {

   public boolean next(); // move to the next result
   public int getInt( String fieldName );
   public String getString( String fieldName );

}
```

## Scan Example: TableScan

```
public TableScan( TableInfo ti, Transaction tx )
{ recordFile = new RecordFile( ti, tx ); }

public boolean next()
{  return recordFile.next(); }

public int getInt( String fieldName )
{  return recordFile.getInt( fieldName ); }

public int getString( String fieldName )
{  return recordFile.getString( fieldName ); }
```

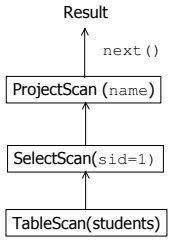
## Scan Example: SelectScan

```
public SelectScan( Scan s, Predicate pred )
{
   this.s = s;
   this.pred = pred;
}

public boolean next()
{
   while( s.next() )
      if( pred.isSatisfied(s) ) return true;
   return false;
}
```

## Query Execution

select name from students where id = 1;

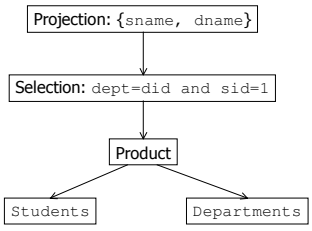



## About Implementations of RA Operations

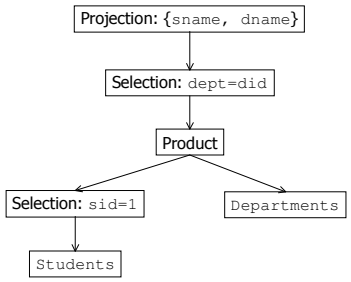
- Each RA operation can be implemented and optimized independently from others
- A RA operation may have multiple implementations
  - E.g. *table scan* vs. *index scan* for selection
- The efficiency of an implementation depends on the characteristics of the data

## A Query Plan

select sname, dname from students, departments
where dept = did and sid = 1;




## A Better Query Plan – Query Optimization




## Readings

- Textbook Chapter 16, 17, 18, 19