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

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

**Semantics**
- The semantics of a language specify the meaning of a syntactically correct string
- *Is the following statement semantically correct??*
  ```sql
  select * from a, b where c = 3;
  ```
**Lexical Analysis**

» Split the input string into a series of tokens

- select sname from students where sid = 1

**Token**

» <type, value>

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword</td>
<td>select</td>
</tr>
<tr>
<td>identifier</td>
<td>sname</td>
</tr>
<tr>
<td>keyword</td>
<td>from</td>
</tr>
<tr>
<td>identifier</td>
<td>students</td>
</tr>
<tr>
<td>keyword</td>
<td>where</td>
</tr>
<tr>
<td>identifier</td>
<td>id</td>
</tr>
<tr>
<td>delimiter</td>
<td>=</td>
</tr>
<tr>
<td>intconstant</td>
<td>1</td>
</tr>
</tbody>
</table>

**Lexer API**

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

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

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

**SimpleDB Grammar ...**

```java
createQuery := 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 ...

```java
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

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

```java
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

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
</tr>
<tr>
<td>2</td>
<td>Amy</td>
</tr>
</tbody>
</table>

sid=1

Output

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
</tr>
</tbody>
</table>

Projection

Input

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
</tr>
<tr>
<td>2</td>
<td>Amy</td>
</tr>
</tbody>
</table>

Input

Output

<table>
<thead>
<tr>
<th>sname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
</tr>
<tr>
<td>Amy</td>
</tr>
</tbody>
</table>

Product

Input

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Amy</td>
<td>20</td>
</tr>
</tbody>
</table>

Input

Output

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>dept</th>
<th>did</th>
<th>dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>10</td>
<td>10</td>
<td>CS</td>
</tr>
<tr>
<td>2</td>
<td>Amy</td>
<td>20</td>
<td>20</td>
<td>Math</td>
</tr>
</tbody>
</table>

Scan

A scan is an interface to a RA operation implementation

```java
public interface Scan {
    public boolean next(); // move to the next result
    public int getInt( String fieldName );
    public String getString( String fieldName );
}
```

Scan Example: TableScan

```java
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

```java
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

```sql
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 name, dname from students, departments
where dept = did and sid = 1;
```

A Better Query Plan – Query Optimization

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

Readings

- Textbook Chapter 16, 17, 18, 19