CSE 4/562
Database Systems

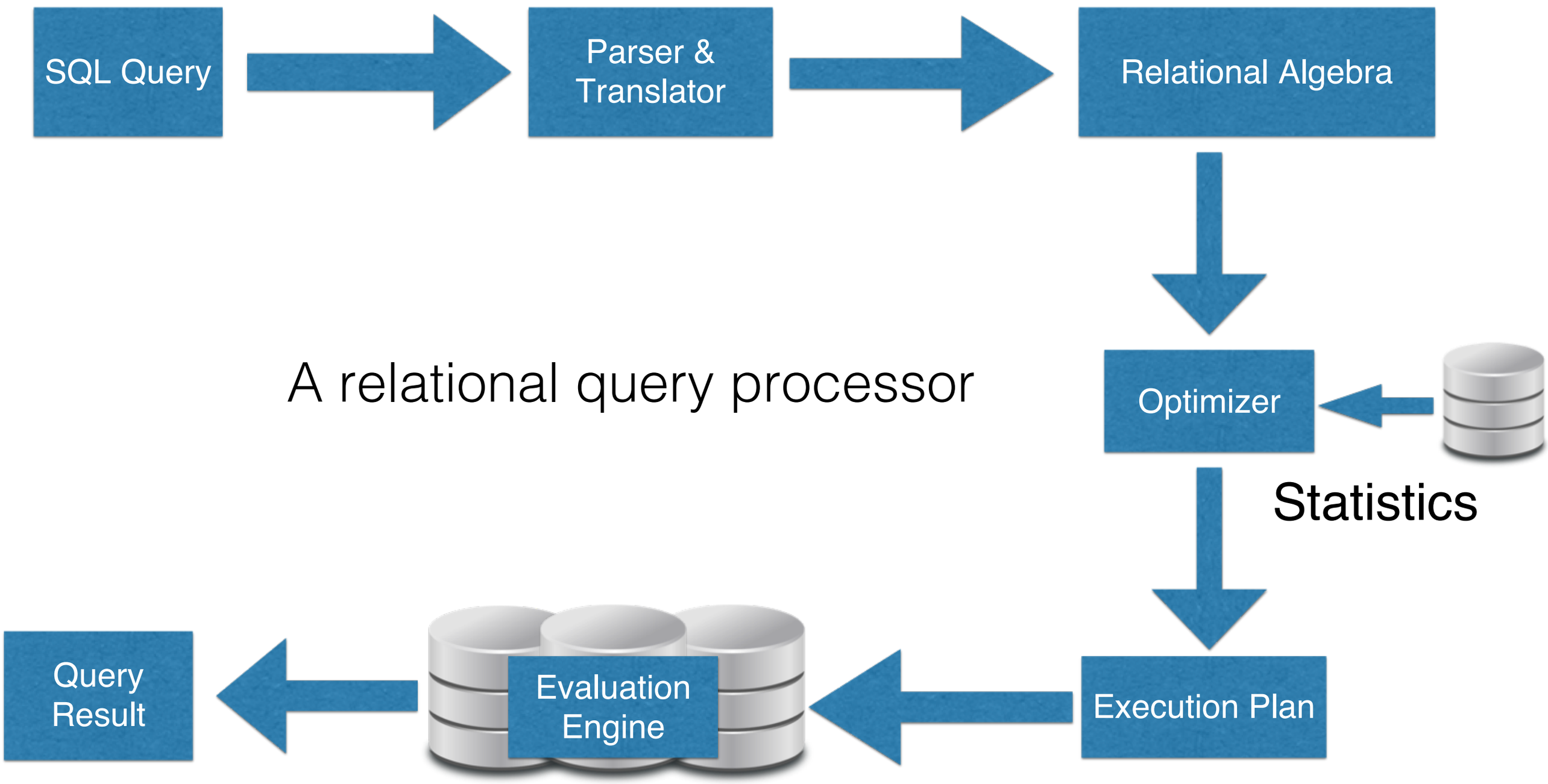
Practicum Component

02/09/2018

Administrivia

- Checkpoint 0 deadline is today!
- Checkpoint 1 queries and data will be released on Monday, 02/12/2018
- Checkpoint 1 grader is going to start taking submissions on Friday, 02/16/2018
- Checkpoint 1 deadline is on Friday, 02/23/2018

Project Outline

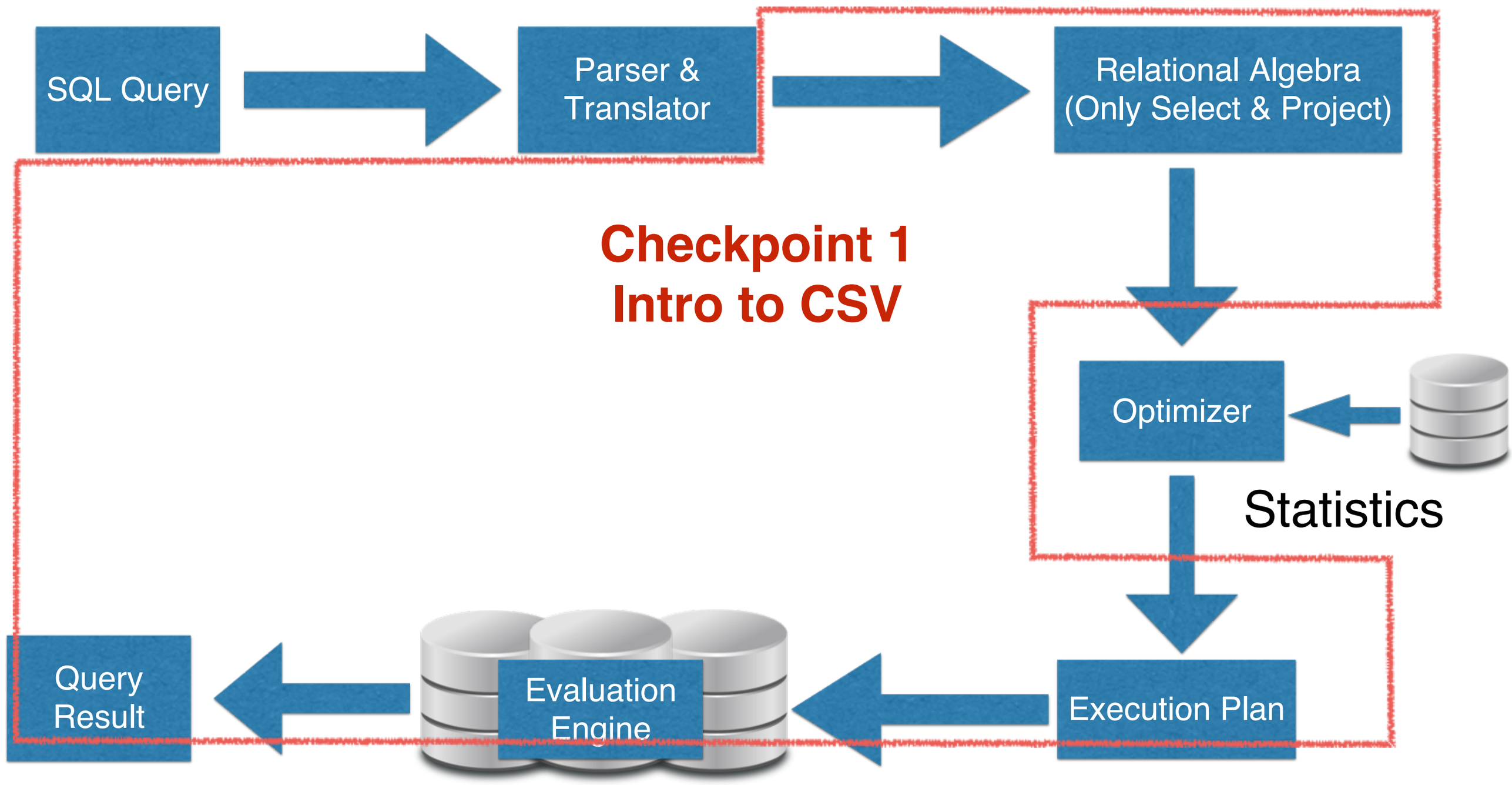


A relational query processor

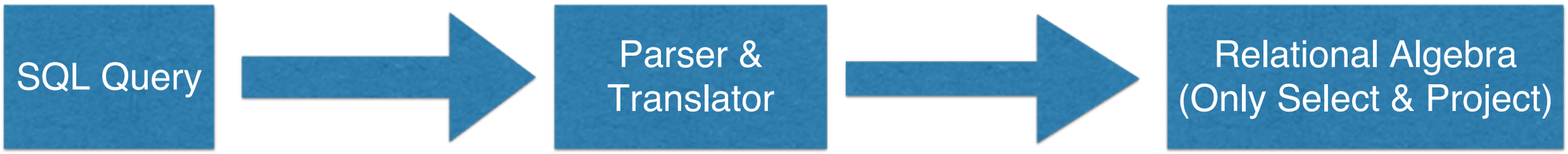
Libraries

- **Download the libraries from the course website!**
- **JSqlParser (Forked version)**
  - Text to SQL Parse Tree
- **EvalLib**
  - Arithmetic Expression Evaluator
- **Apache Commons CSV**
  - CSV Format Support / Parsing CSV

Checkpoint 1



SQL to RA


SQL Query
Parser & Translator
Relational Algebra (Only Select & Project)

Using CCJSqlParser

```
// StringReaders create a reader from a string
Reader input = new StringReader("SELECT * FROM R")

// CCJSqlParser takes a Reader or InputStream
CCJSqlParser parser = new CCJSqlParser(input)
```

```
// CCJSqlParser.Statement() returns the next
// complete Statement object from the reader or
// input stream (or null if the stream is empty).
Statement statement = parser.Statement()
```

Using CCJSqlParser

```
// System.in is an InputStream
CCJSqlParser parser = new CCJSqlParser(System.in)

Statement statement = parser.Statement();
// loop until you hit the last statement
while(statement != null){

  //
  // Do something with statement
  //

  // ... then read the next statement
  statement = parser.Statement();
}
```

Abstract Syntax Trees

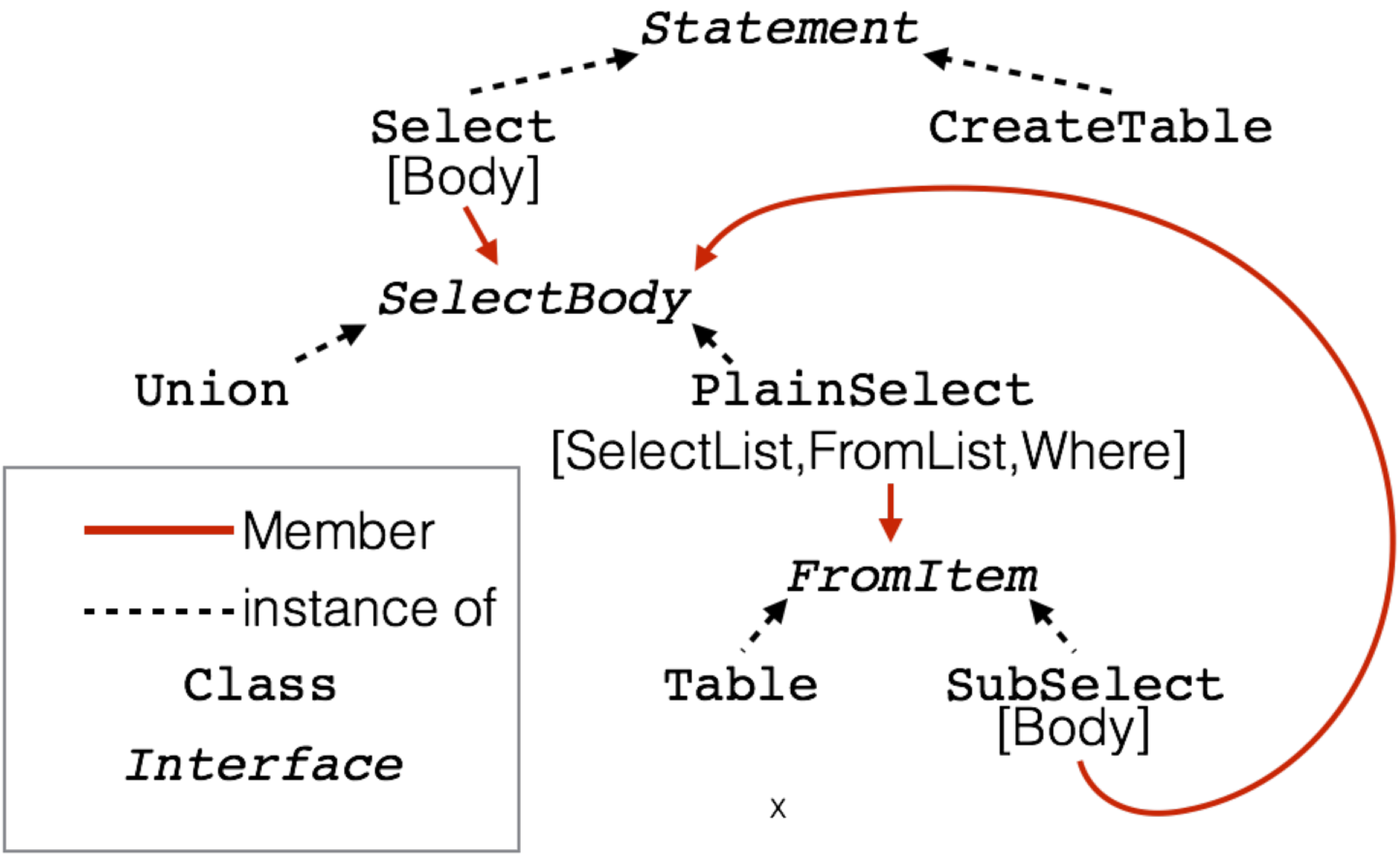
- **Statement:** Select, CreateTable

Using CCJSqlParser

```
while(statement != null){
  if(statement instanceof Select){
    Select select = (Select)statement;
    // Do something with `select`
  } else if(statement instanceof CreateTable){
    CreateTable create = (CreateTable)statement;
    // Do something with `create`

  } else {
    throw new SqlException("Can't handle: "+statement);
  }
  statement = parser.Statement()
}
```

Syntax Trees in Java



x

Abstract Syntax Trees

- **Statement:** Select, CreateTable
- **SelectBody:** PlainSelect, Union
- **SelectItem:** AllColumns, AllTableColumns, SelectExpressionItem
- **Expression:** LongValue, AddExpression, GreaterThan, etc…

Using CCJSqlParser

```
Select select = (Select)statement;
SelectBody body = select.getSelectBody();
if(body instanceof /* ... */){
   // ...
}
```

- **PlainSelect:** SELECT * FROM …
- **Union:** (SELECT * FROM …) UNION ALL (SELECT…)

Using CCJSqlParser

```
Select select = (Select)statement;
SelectBody body = select.getSelectBody();
if(body instanceof PlainSelect){
  PlainSelect plain = (PlainSelect)body;
  // Do something with `plain`
}
```

- **PlainSelect:** SELECT * FROM …
- **Union:** (SELECT * FROM …) UNION ALL (SELECT…)

PlainSelect

```
SELECT [distinct] [selectItems]
FROM [fromItem], [joins, ...]
WHERE [where]
GROUP BY [groupByColumnReferences]
HAVING [having]
ORDER BY [orderByElements]
LIMIT [limit]
```

Everything in [brackets] has a method in PlainSelect

SelectItems

AllColumns:
SELECT *

AllTableColumns:
SELECT R.*

SelectExpressionItem:
SELECT R.A or SELECT R.A AS Q

How about FromItem and others?

Explore yourself!

net.sf.jsqlparser.expression.PrimitiveValue

Implementations	Methods
BooleanValue	double toDouble()
LongValue	long toLong()
DoubleValue	string toString()
StringValue	boolean toBool()
DateValue	

EvalLib

net.sf.jsqlparser.eval.Eval

LeafValue eval(Column c)

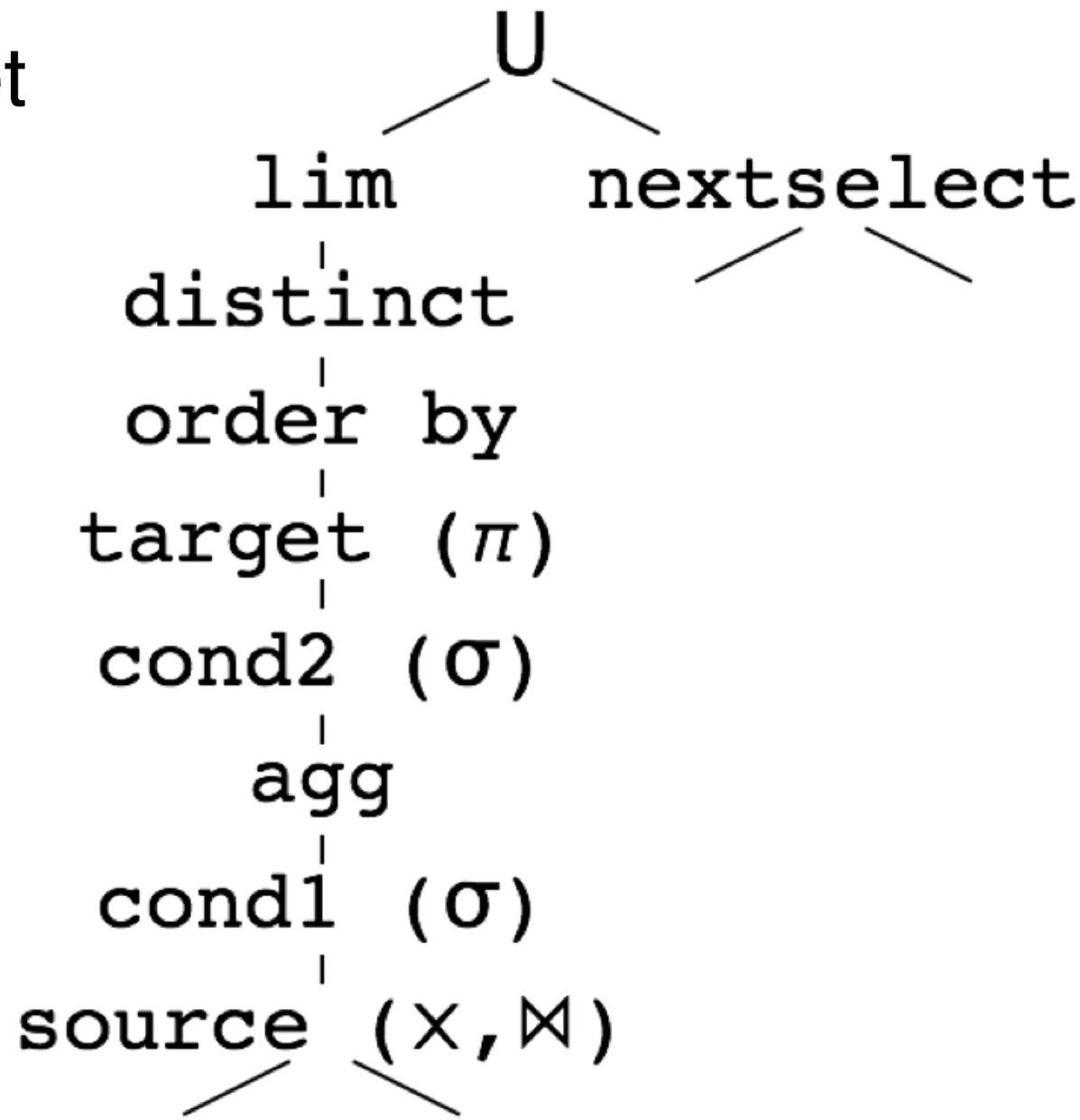
Implement this

LeafValue eval(Expression e)

Then call this from the library

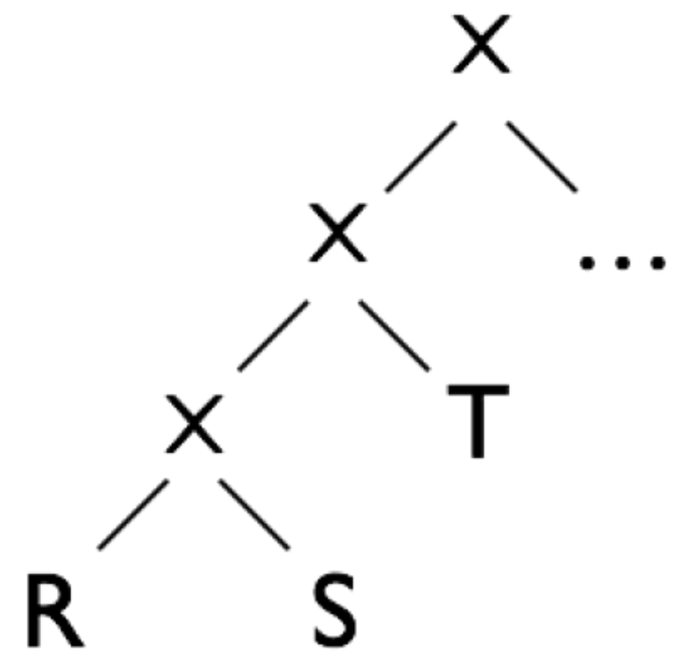
Translating SQL to RA

SELECT [DISTINCT] target
FROM source
WHERE cond1
GROUP BY …
HAVING cond2
ORDER BY order
LIMIT lim
UNION nextselect



FROM Clause

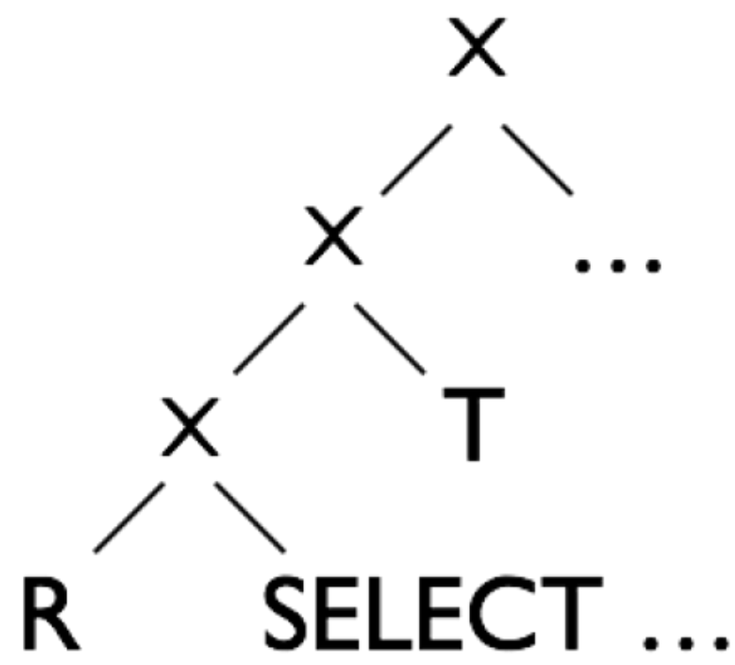
FROM R, S, T, …



What happens when S is another (nested) query?

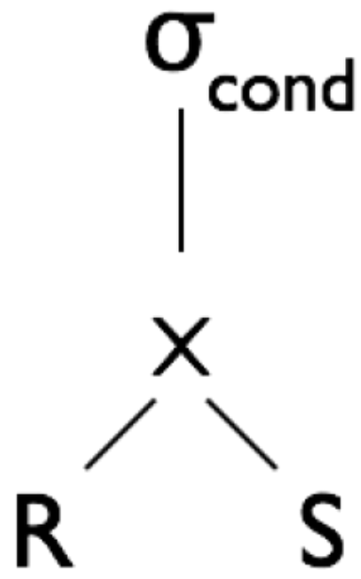
FROM Clause

FROM R, (SELECT …) S, T, …

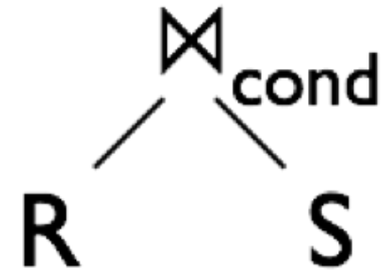


FROM Clause

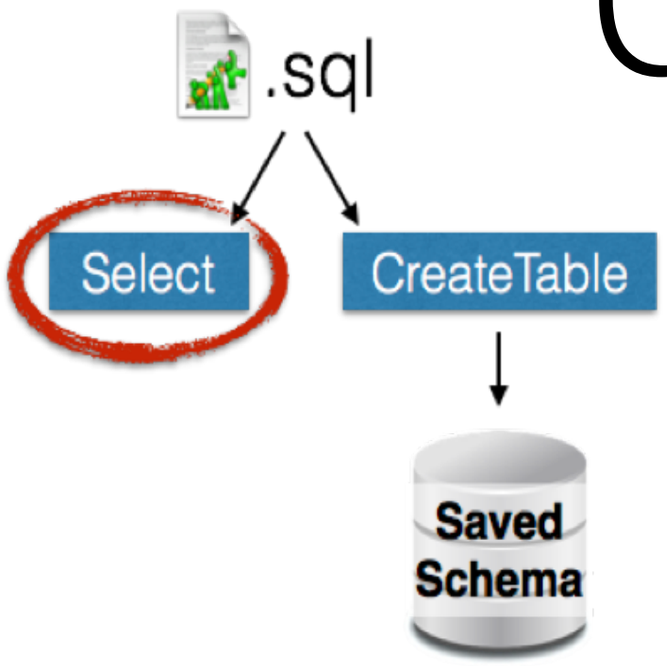
FROM R, JOIN ON cond



OR

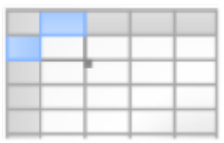


Checkpoint 1




```
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;
```

`PLAYERS.dat`



(Output)

`body.getFromItem()`

↓

query = Players

`body.getWhere()`

↓

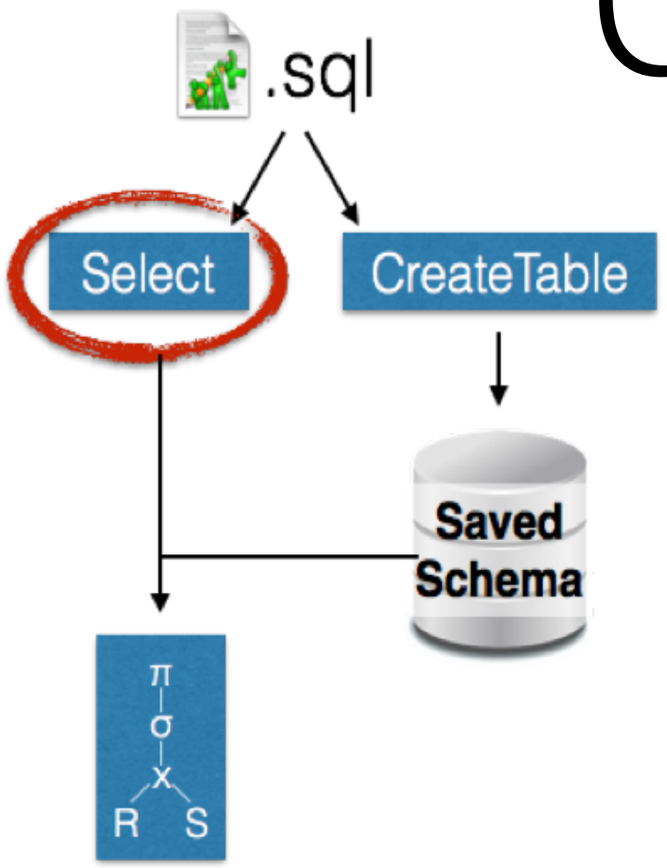
query = σ(where, **query**)

`body.getSelectItems()`

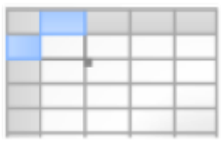
↓

query = π(items, **query**)

Checkpoint 1

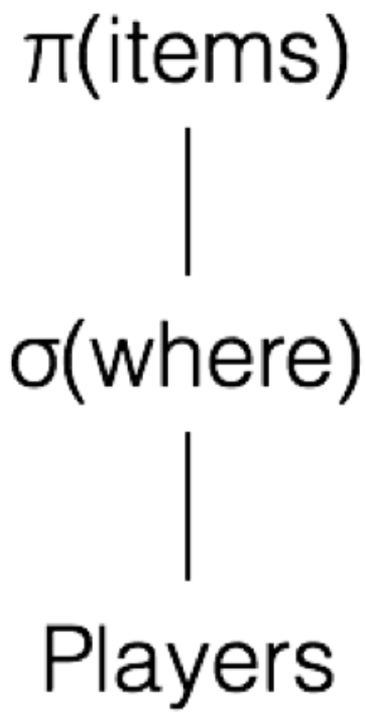


PLAYERS.dat

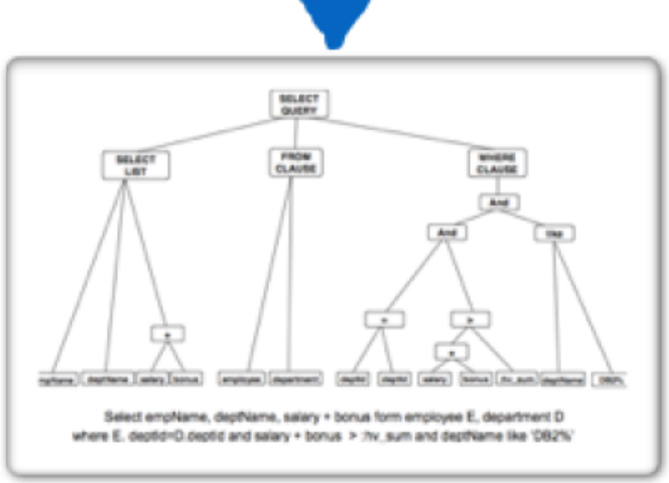


(Output)

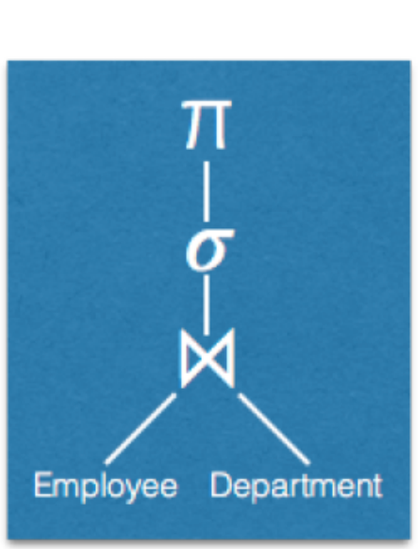
```
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;
```

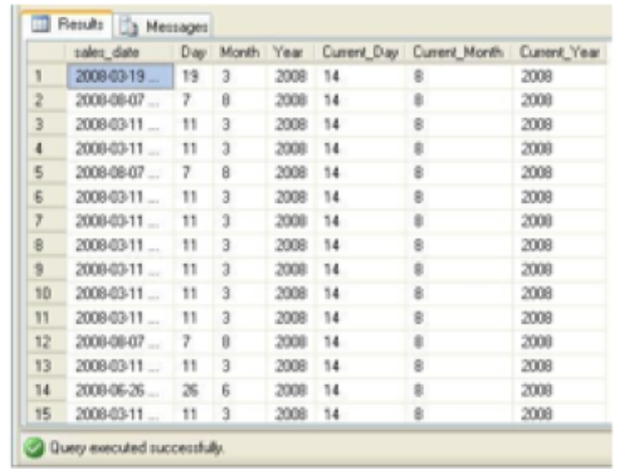


Next Week:
Query Evaluation



Parsed Query



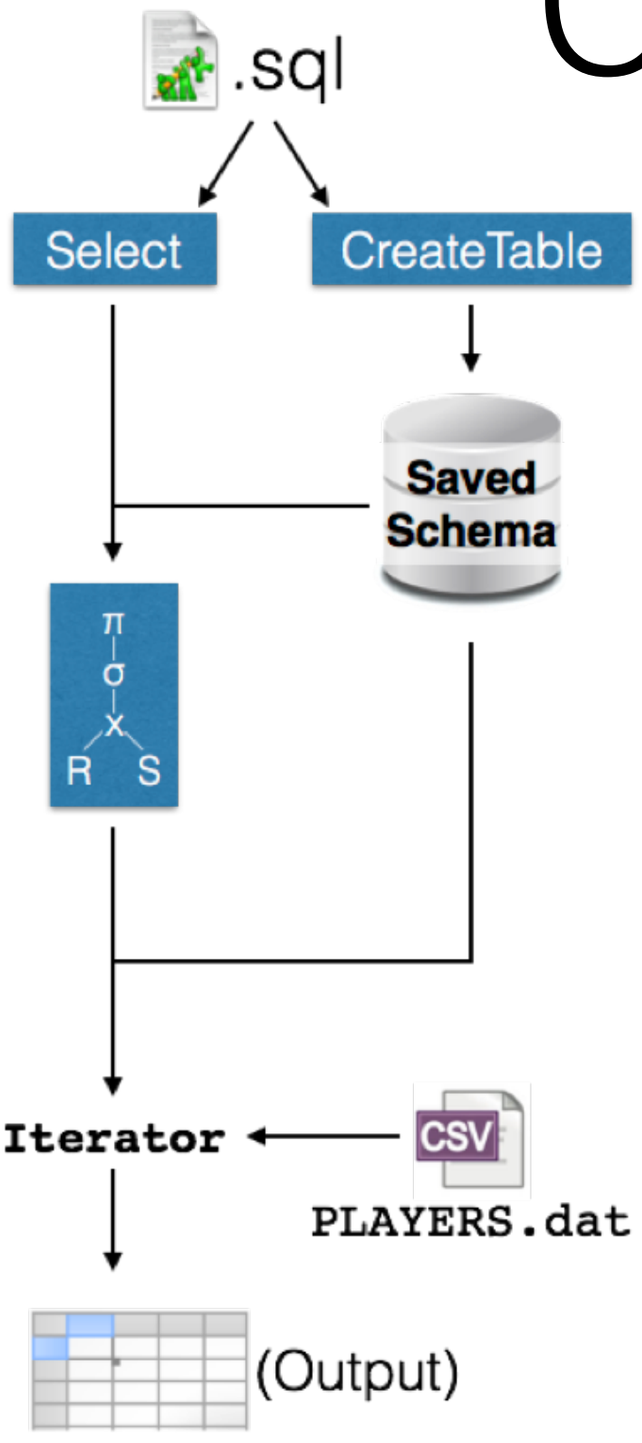


Results

Data

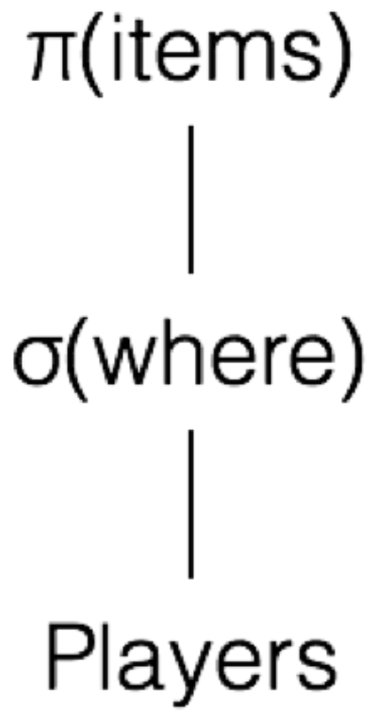
How does this work?

Checkpoint 1



SELECT FIRSTNAME, LASTNAME,
WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;

Iterators:
Read 1 row at a time



Tip for Iterators

```
void open() {
 // call open() on child iterators
 // prepare the iterator
}

Tuple getNext() {
 // read, process, and return a tuple
}

void close() {
 // clean-up the iterator
 // call close() on child iterators
}
```

Questions?