CSE4/562 Database Systems

Practicum Component

02/16/2018
Recap

How does this work?

Parsed Query

What does this look like?
(last class)

Results

Data

.sql

How does this work?

Employee

Department

2
A Basic SQL Query

(optional) keyword indicating that the answer should **not** contain duplicates

```
SELECT  [DISTINCT] target-list

A list of attributes of relations in relation-list

FROM    relation-list

A list of relation names
(possibly with a range-variable after each name)

WHERE   condition
```

Comparisons (`=`,'<>', '<', '>', '<=', '>=') and other boolean predicates, combined using AND, OR, and NOT (a boolean formula)
SQL

• SQL is a language for querying relations
  • `SELECT` to access (query) data
    • Different features for different access patterns.
  • `INSERT INTO, DELETE FROM` to modify data
  • `CREATE TABLE, DROP TABLE, ALTER TABLE` to modify relations
Relational Algebra Trees

SELECT O.FirstName
FROM Officers O, Ships S
WHERE O.Ship = S.ID
  AND S.Name = 'Enterprise'

πFirstName(Officers ⋈ Ship=ID(σName='Enterprise' Ships))
Relational Algebra Trees

$$\pi_{\text{FirstName}}(\text{Officers} \bowtie_{\text{Ship} = \text{ID}} (\sigma_{\text{Name} = \text{'Enterprise'}} \text{Ships}))$$
Relational Algebra Trees

How many operators do you see here?

\[ \pi_{\text{FirstName}}(\text{Officers} \bowtie \text{Ship=ID}(\sigma_{\text{Name}=\text{'Enterprise'}}\text{Ships})) \]
Statement statement = parser.Statement();

if (statement instanceof Select) {
    Algebra raTree = parseTree((Select)statement);
    evaluate(raTree);
}

else if (statement instanceof CreateTable) {
    loadTableSchema((CreateTable)statement);
}


Recommendation

Start by imagining how you would implement a relational algebra tree - and actually do it!

Determine inputs and outputs for each operator.

Treat table scan as an operator
Syntax Trees in Java

What would a class hierarchy look like for Relational Algebra?
Syntax Trees in Java

Operator

\( \pi \)  \( \sigma \)  \( \sqcup \)  \( \times \)

\([c]\)  \([c]\)  \([l,r]\)  \([l,r]\)
SQL to RA

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

```
U
```

```
lim
```

```
nextselect
```

```
distinct
```

```
order by
```

```
target (π)
```

```
cond2 (σ)
```

```
agg
```

```
cond1 (σ)
```

```
source (X,⋈)
```

13
FROM Clause

FROM $R, S, T, ...$

What happens if I have a FROM-nested query?
FROM Clause

FROM R, (SELECT ...) S, T, ...

Selects are just relations!
FROM Clause

FROM R JOIN S ON cond
FROM Clause

FROM R JOIN S ON cond

\[ \sigma_{\text{cond}} \]

or

\[ R \bowtie S \]
FROM Clause

FROM R NATURAL JOIN S
FROM Clause

FROM R NATURAL JOIN S

\[
\sigma_{\text{cond}} \quad \text{or} \quad R \bowtie S
\]

\[
\text{cond} = \text{schema}(R) \cap \text{schema}(S)
\]

You need to be able to compute the schema of a RA operator
SQL to RA

SELECT [DISTINCT]
    target
FROM source
WHERE cond1
GROUP BY ...
HAVING cond2
ORDER BY order
LIMIT lim
UNION nextselect
SELECT (target) Clause

- \( \pi_{\text{targets}} \)
- \( \text{input} \)

\[
\begin{align*}
\text{SELECT} & \hspace{1em} * \\
\text{no } \pi & \hspace{1em} \text{(or target = schema(input))}
\end{align*}
\]

\[
\begin{align*}
\pi_{\text{targets}} & \hspace{1em} \text{targets} = A, B, \ldots \\
\text{input} & \hspace{1em} \text{SELECT R.*, S.*} \\
& \hspace{1em} \text{targets} = \text{schema(input) from R, S}
\end{align*}
\]

Schemas need both Table Alias & Attribute Name
(see Column class)
SELECT A + B

How do you evaluate A + B

WHERE A + B > 5

How do you evaluate A + B > 5
EvalLib

SELECT A + B

How do you evaluate A + B

WHERE A + B > 5

How do you evaluate A + B > 5

Tip 1: You can evaluate expressions recursively

Tip 2: Use visitor pattern
Evaluating RA
The Evaluation Pipeline

How does this work? (now)
Evaluation Strategies

• **Staged Evaluation**: Start at leaves, Evaluate each operator as one step.

• **Pull Model**: Tuple-at-a-time Iterator for each operator (also called *Volcano Operators*) reads from source iterator(s).

• **Push Model**: Thread-per operator reads from input buffer(s) and writes to output buffer.
Evaluation Strategies

- **Pull Model**: Tuple-at-a-time Iterator for each operator (also called *Volcano Operators*) reads from source iterator(s).
Staged Evaluation

\[ \pi_{\text{FirstName}} \Join_{\text{Ship}=\text{ID}} \sigma_{\text{Name}=\text{‘Enterprise’}} \text{Officers} \rightarrow \sigma_{\text{Name}=\text{‘Enterprise’}} \text{Ships} \]

Compute the Output
Staged Evaluation

Compute the Output

**Dependency**: Compute $\pi$

$\pi_{\text{FirstName}}$

$\Join_{\text{Ship}=\text{ID}}$

**Officers** $\sigma_{\text{Name}='\text{Enterprise}'}$

**Ships**
Staged Evaluation

Compute the Output

**Dependency**: Compute $\pi$

**Dependency**: Compute $\bowtie$

$\pi$\text{FirstName}

$\bowtie$\text{Name='Enterprise'}

$\sigma$\text{Name='Enterprise'}

$\pi$\text{FirstName}

$\bowtie$\text{Ship=ID}

$\sigma$\text{Name='Enterprise'}

Ships
Staged Evaluation

Compute the Output

**Dependency**: Compute $\pi$

**Dependency**: Compute $\bowtie$

**Dependency**: Compute $\sigma$

Diagram:

```
  pi FirstName
  \bowtie Ship=ID
  Officers
    \sigma Name='Enterprise'
    Ships
```
Staged Evaluation

Compute the Output

**Dependency**: Compute $\pi$

**Dependency**: Compute $\bowtie$

**Dependency**: Compute $\sigma$

**Dependency**: Load Ships

<table>
<thead>
<tr>
<th>ID, Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1701, Enterprise]</td>
</tr>
<tr>
<td>[DS9, Deep Space 9]</td>
</tr>
<tr>
<td>[74656, Voyager]</td>
</tr>
<tr>
<td>[75633, Defiant]</td>
</tr>
</tbody>
</table>
Staged Evaluation

Compute the Output

**Dependency**: Compute $\pi$

**Dependency**: Compute $\Join$

**Dependency**: Compute $\sigma$

<table>
<thead>
<tr>
<th>ID, Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1701, Enterprise ]</td>
</tr>
<tr>
<td>[DS9, Deep Space 9]</td>
</tr>
<tr>
<td>[74656, Voyager ]</td>
</tr>
<tr>
<td>[75633, Defiant ]</td>
</tr>
</tbody>
</table>
Staged Evaluation

Compute the Output

**Dependency**: Compute $\pi$

**Dependency**: Compute $\bowtie$

$\pi$FirstName

$\bowtie$Ship=ID

Officers $\sigma$Name='Enterprise'

Ships

ID, Name

[1701, Enterprise ]
Staged Evaluation

Compute the Output

**Dependency**: Compute $\pi$

**Dependency**: Compute $\bowtie$

**Dependency**: Load Officers

<table>
<thead>
<tr>
<th>ID,        Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1701, Enterprise ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\text{FirstName}$, $\text{LastName}$, Rank, Ship</th>
</tr>
</thead>
<tbody>
<tr>
<td>[James, Kirk, 4.0, 1701 ]</td>
</tr>
<tr>
<td>[Jean Luc, Picard, 4.0, 1701D]</td>
</tr>
<tr>
<td>[Benjamin, Sisko, 3.0, DS9 ]</td>
</tr>
<tr>
<td>[Kathryn, Janeway, 4.0, 74656]</td>
</tr>
<tr>
<td>[Nerys, Kira, 2.5, 75633]</td>
</tr>
<tr>
<td>[Spock, NULL, 2.5, 1701 ]</td>
</tr>
<tr>
<td>[William, Riker, 2.5, 1701D]</td>
</tr>
<tr>
<td>[Nerys, Kira, 2.5, DS9 ]</td>
</tr>
<tr>
<td>[Chakotay, NULL, 3.0, 74656]</td>
</tr>
</tbody>
</table>
Staged Evaluation

Compute the Output

**Dependency**: Compute $\pi$

**Dependency**: Compute $\bowtie$

<table>
<thead>
<tr>
<th>First Name, Last Name, Rank, Ship, ID, Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[James, Kirk, 4.0, 1701, 1701, Enterprise]</td>
</tr>
<tr>
<td>[Spock, NULL, 2.5, 1701, 1701, Enterprise]</td>
</tr>
</tbody>
</table>
Staged Evaluation

**Dependency**: Compute \( \pi \)

\[
\begin{align*}
\pi & \text{FirstName} \\
\bowtie & \text{Ship=ID} \\
\sigma & \text{Name='Enterprise'} \\
\text{Officers} & \text{Ships}
\end{align*}
\]

- [James, Kirk, 4.0, 1701, 1701, Enterprise]
- [Spock, NULL, 2.5, 1701, 1701, Enterprise]
Staged Evaluation

Compute the Output

\[ \pi \text{FirstName} \\
\left[\begin{array}{c}
\text{FirstName} \\
\text{Spock}
\end{array}\right] \]

Officers \bowtie_{\text{Ship}=\text{ID}} \sigma_{\text{Name}=\text{Enterprise}} \text{Ships}

First Name

[James, Spock]
Staged Evaluation

Can we do better?
Staged Evaluation

- **Expensive**: Lots of Bulk Copies
- **Cache Locality**: Repeated Scans over Full Tables
- **Memory Use**: Working Set is a Full Table (or more)

How do we do better?
The Memory Hierarchy and You

• We want to keep data as close to the CPU as possible
  • Faster memory == Smaller memory

• **Solution 1:** Minimize the *Working Set* Size!
  • (the memory used at any one time)

• **Solution 2:** Aggressively Batch & Reuse Data
Volcano Evaluation

Compute one tuple

\[ \pi \text{FirstName} \]
\[ \bowtie \text{Ship=ID} \]
\[ \text{Officers} \circ \text{Name='Enterprise'} \]
\[ \text{Ships} \]
Volcano Evaluation

Compute one tuple

**Dependency**: Next tuple from $\pi$

$\pi$FirstName

$\bowtie$ Ship=ID

Officers $\sigma$Name=‘Enterprise’

Ships
Volcano Evaluation

Compute one tuple

**Dependency**: Next tuple from $\pi$

**Dependency**: Next tuple from $\bowtie$

$\pi$FirstName

$\bowtie$Ship=ID

Officers $\sigma$Name='Enterprise'

Ships
Volcano Evaluation

Compute one tuple

**Dependency**: Next tuple from \( \pi \)

**Dependency**: Next tuple from \( \bowtie \)

**Dependency**: Next tuple from \( \sigma \)

\[ \pi \text{FirstName} \]

\[ \bowtie \text{Ship=ID} \]

\[ \sigma \text{Name='Enterprise'} \]

Officers

Ships
Volcano Evaluation

Compute one tuple

**Dependency**: Next tuple from \( \pi \)

**Dependency**: Next tuple from \( \bowtie \)

**Dependency**: Next tuple from \( \sigma \)

**Dependency**: Tuple from \( \text{Ships} \)

\[
\begin{align*}
| & ID, & Name \\
| & 1701, & Enterprise \\
\end{align*}
\]
Volcano Evaluation

\[
\begin{align*}
\pi_{\text{FirstName}} \bowtie_{\text{Ship} = \text{ID}} \sigma_{\text{Name} = \text{Enterprise}} \text{Ships}
\end{align*}
\]

Compute one tuple

**Dependency**: Next tuple from \( \pi \)

**Dependency**: Next tuple from \( \bowtie \)

**Dependency**: Next tuple from \( \sigma \)

\[
\text{ID, Name}
\]

\[ [1701, \text{ Enterprise} ] \]
Volcano Evaluation

Compute one tuple

**Dependency**: Next tuple from \( \Pi \)

**Dependency**: Next tuple from \( \Join \)

\( \text{ID, Name} \)

[1701, Enterprise]
Volcano Evaluation

Compute one tuple

**Dependency**: Next tuple from $\pi$

**Dependency**: Next tuple from $\bowtie$

**Dependency**: Tuple from $\sigma$

$\pi_{\text{FirstName}}$

$\bowtie_{\text{Ship}=\text{ID}}$

$\sigma_{\text{Name}='\text{Enterprise}'}$

**Officers**

**Ships**

| ID,     Name          |
|----------|-------------------|
| [1701, Enterprise | ]

<table>
<thead>
<tr>
<th>FirstName, LastName, Rank, Ship</th>
</tr>
</thead>
<tbody>
<tr>
<td>[James,    Kirk,    4.0, 1701 ]</td>
</tr>
</tbody>
</table>
Volcano Evaluation

Compute one tuple

**Dependency**: Next tuple from \( \pi \)

**Dependency**: Next tuple from \( \bowtie \)

\[
\begin{align*}
\text{Officers} & \quad \sigma \text{Name=‘Enterprise’} \\
\text{Ships} & \\
\end{align*}
\]

\[
\begin{align*}
\text{FirstName, LastName, Rank, Ship, ID, Name} \\
\text{[James, Kirk, 4.0, 1701, 1701, Enterprise]}
\end{align*}
\]
Volcano Evaluation

Compute one tuple

**Dependency**: Next tuple from $\pi$

$\pi_{\text{FirstName}}$

$\Join_{\text{Ship}=\text{ID}}$

$\sigma_{\text{Name}='\text{Enterprise}'}$

$\text{Officers} \ O \text{Name}='\text{Enterprise}'$

$\text{Ships}$

$\text{FirstName, LastName, Rank, Ship, ID, Name}$

[James, Kirk, 4.0, 1701, 1701, Enterprise]
Volcano Evaluation

Compute one tuple

\[ \pi_{\text{FirstName}} (\nabla_{\text{Ship}=\text{ID}} (\sigma_{\text{Name}='Enterprise'} (\text{Officers}))) \]

\[ \text{FirstName} = [\text{James}] \]
Iterators

```c
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
}
```
GetNext()

Relation (I called this “scan operator” before)

Read One Line from File

Split Line into Fields

Parse Field Types

Return Tuple

What is the Working Set Size?
GetNext()

**Projection (\(\pi\))**

Read One Tuple

Compute Projected Attributes

**Return Tuple**

What is the Working Set Size?
GetNext()

Selection (σ)

Read One Tuple

Test Condition

Reject Tuple  Return Tuple

What is the Working Set Size?
GetNext()

Union (U)

Read One Tuple from R

R Empty?

Read One Tuple from S ——> Return Tuple

What is the Working Set Size?
GetNext()

**Nested Loop Join/Cross**

- Read (and save) One Tuple from R
- Read One Tuple from S
- S Empty?
  - Reset S (Close(), Open())
  - Construct Joint Tuple: \(< S \circ \cdot < R >\)
  - What is the Working Set Size?
  - Return Tuple but...

Is there a saved tuple? Y N
Implementing: Joins

**Solution 1 (Nested-Loop)**

For Each (a in A) { For Each (b in B) { emit (a, b); }}
Implementing: Joins

**Solution 2** (Block-Nested-Loop)
Implementing: Joins

**Solution 2** (Block-Nested-Loop)

1) Partition into Blocks  
2) NLJ on each pair of blocks