Announcements

- WQ5 (datalog) due tomorrow
- HW4 (datalog) due tomorrow
- Midterm review session this evening
  - 5:30pm, CSE 2nd Floor Breakout

Class Overview

- Unit 1: Intro
- Unit 2: Relational Data Models and Query Languages
- Unit 3: Non-relational data
  - Unit 4: RDBMS Internals and Query Optimization
- Unit 5: Parallel Query Processing
- Unit 6: DBMS Usability, Conceptual Design
- Unit 7: Transactions
- Unit 8: Advanced topics (time permitting)

Query Evaluation Steps Review

From Logical RA Plans to Physical Plans
Logical vs Physical Plans

• Logical plans:
  – Created by the parser from the input SQL text
  – Expressed as a relational algebra tree
  – Each SQL query has many possible logical plans

• Physical plans:
  – Goal is to choose an efficient implementation for each operator in the RA tree
  – Each logical plan has many possible physical plans

Physical Plan v.s. Physical Plan

• Logical Plan = a Relational Algebra tree
• Physical Plan = a Logical Plan plus annotation of each operator with an algorithm

Query Optimization and Execution

• Query optimizer:
  – Choose a good logical plan
  – Refine it to a good physical plan
  – Sometimes these steps are intertwined
• Query execution
  – Execute the physical plan

Physical Operators

Relational algebra operators:
• Selection, projection, join, union, difference
• Group-by, distinct, sort

Physical operators:
• For each operators above, several possible algorithms
• Main memory algorithms, or disk-based algorithms
Main Memory Algorithms

Logical operator:
\text{Supplier} \bowtie \text{Supply}

Propose three physical operators for the join, assuming the tables are in main memory:
1. Nested Loop Join \quad O(n^2)
2. Merge join \quad O(n \log n)
3. Hash join \quad O(n)

CSE 414 - Autumn 2018
13

Main Memory Algorithms

Logical operator:
\text{Supplier} \bowtie \text{Supply}

Propose three physical operators for the join, assuming the tables are in main memory:
1. Nested Loop Join \quad O(??)
2. Merge join \quad O(??)
3. Hash join \quad O(??)

CSE 414 - Autumn 2018
14

BRIEF Review of Hash Tables

A (naïve) hash function:
\[ h(x) = x \mod 10 \]

Operations:
\begin{align*}
\text{find}(103) &= 103 \\
\text{insert}(488) &= 488 \\
\end{align*}

CSE 414 - Autumn 2018
15

Query Execution

- Join \text{R} \bowtie \text{S}, e.g. using hash-join:
  - Nested-loop: forall \( x \) in \text{R} forall \( y \) in \text{S} do ...
  - Hash-join: build a hash table on \text{S}, probe \text{R}
- Selection: \( \sigma(\text{R}) \), e.g. "on-the-fly"
- But what about a larger plan?
  - Each operator implements the iterator interface

CSE 414 - Autumn 2018
17
Implementing Query Operators with the Iterator Interface

Each operator implements three methods:

• open()
• next()
• close()

interface Operator {
    // initializes operator state
    // and sets parameters
    void open (...);

    // calls next() on its inputs
    // processes an input tuple
    // produces output tuple(s)
    // returns null when done
    Tuple next ();

    // cleans up (if any)
    void close ();
}

class Select implements Operator {
    void open (Predicate p, Operator child) {
        this.p = p; this.child = child;
    }

    Tuple next () {
        boolean found = false;
        while (!found) {
            Tuple in = child.next ();
            if (in == EOF) return EOF;
            found = p(in);
        }
        return in;
    }

    void close () {
        child.close ();
    }
}
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

```java
interface Operator {
   // initializes operator state
   // and sets parameters
   void open (...);
   // calls next() on its inputs
   // processes an input tuple
   // produces output tuple(s)
   // returns null when done
   Tuple next ();
   // cleans up (if any)
   void close ();
}

class Select implements Operator {
   void open (Predicate p, Operator child) {
      this.p = p; this.child = child;
   }
   Tuple next () {
      boolean found = false;
      Tuple r = null;
      while (!found) {
         r = child.next ();
         if (r == null) break;
         found = p(in);
      }
      return r;
   }
   void close () {
      child.close ();
   }
}
```

Example “on the fly” selection operator

```java
interface Operator {
   // initializes operator state
   // and sets parameters
   void open (...);
   // calls next() on its inputs
   // processes an input tuple
   // produces output tuple(s)
   // returns null when done
   Tuple next ();
   // cleans up (if any)
   void close ();
}

class Select implements Operator {
   void open (Predicate p, Operator child) {
      this.p = p; this.child = child;
   }
   Tuple next () {
      boolean found = false;
      Tuple r = null;
      while (!found) {
         r = child.next ();
         if (r == null) break;
         found = p(in);
      }
      return r;
   }
   void close () {
      child.close ();
   }
}
```

Example “on the fly” selection operator

```java
interface Operator {
   // initializes operator state
   // and sets parameters
   void open (...);
   // calls next() on its inputs
   // processes an input tuple
   // produces output tuple(s)
   // returns null when done
   Tuple next ();
   // cleans up (if any)
   void close ();
}

class Select implements Operator {
   void open (Predicate p, Operator child) {
      this.p = p; this.child = child;
   }
   Tuple next () {
      boolean found = false;
      Tuple r = null;
      while (!found) {
         r = child.next ();
         if (r == null) break;
         found = p(in);
      }
      return r;
   }
   void close () {
      child.close ();
   }
}
```

Query plan execution

```
Operator q = parse("SELECT ...");
q = optimize(q);
q.open();
while (true) {
   Tuple t = q.next();
   if (t == null) break;
   else printOnScreen(t);
}
q.close();
```

Pipelining

```
Supplier(sid, sname, scity, sstate)
```

Discuss: open/next/close for nested loop join
Suppliers
Supplies

\( s_{no} = s_{no} \)
\( \sigma_{scity = ‘Seattle’ \text{ and } sstate = ‘WA’ \text{ and } pno = 2} \)
\( \pi_{sname} \)

(On the fly)
File scan

(On the fly)

(On the fly)

(On the fly)

(Nested loop)

(On the fly)

(On the fly)

(On the fly)

(On the fly)

Suppliers (File scan)

Supplies (File scan)

Suppliers (File scan)

Supplies (File scan)

Discuss: open/next/close for nested loop join

open()

open()

open()

open()

Supplier(\( sid, sname, scity, sstate \))

Supply(\( sid, pno, quantity \))

Discuss: open/next/close for nested loop join
Suppliers

\[ \text{Supplier}(sid, sname, scity, sstate) \]

Supplies

\[ \text{Supply}(sid, pno, quantity) \]

\[ \pi_{sname} (\text{File scan}) \]

\[ \sigma_{\text{scity} = \text{Seattle} \land \text{sstate} = \text{WA} \land pno = 2} (\text{On the fly}) \]

\[ \text{(Nested loop)} \]

Discuss: open/next/close

for nested loop join

Supplier(
\[ sid \],
\[ sname \],
\[ scity \],
\[ sstate \])

Supply(
\[ sid \],
\[ pno \],
\[ quantity \])

Discuss hash-join
in class
Pipelining

Suppliers

Supplies

Pipeline v.s. Blocking

- Pipeline
  - A tuple moves all the way through up the query plan
  - Advantages: speed
  - Disadvantage: need all hash at the same time in memory

- Blocking
  - The entire result of the subplan is computed (and stored to disk) before the first tuple is sent up the plan
  - Advantage: saves memory
  - Disadvantage: slower

Discussion on Physical Plan

More components of a physical plan:

- **Access path selection** for each relation
  - Scan the relation or use an index (next lecture)

- **Implementation choice** for each operator
  - Nested loop join, hash join, etc.

- **Scheduling decisions** for operators
  - Pipelined execution or intermediate materialization