CSE544: Principles of Database Systems

Query Execution
Announcements

• Homework 2 is posted, due May 6
  – SimpleDB
  – Understand existing code PLUS write more code
  – Start early!!

• Review 3 (Selectivity estimation): due April 24

• Project meetings: tomorrow, April 24

• Project M2 (Proposal) due April 26
  – Please try to choose your project by Wednesday
  – Proposal: define clear, limited goals! Don’t try too much
Outline

• Relational Algebra: Ch. 4.2

• Query Evaluation: Ch. 12-14
Steps of the Query Processor

1. Parse & Rewrite Query
2. Select Logical Plan
3. Select Physical Plan
4. Query Execution
5. Disk

Query optimization

Logical plan

Physical plan
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and 
    x.price > 100 and z.city = ‘Seattle’
Relational Algebra = HOW

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Temporary tables T1, T2, . . .

\[ T1(\text{pid}, \text{name}, \text{price}, \text{pid}, \text{cid}, \text{store}) \]

\[ T2(\ldots) \]

\[ \sigma_{\text{price}>100 \text{ and city}='Seattle'}(\ldots) \]

\[ \Pi_{\text{x.name}, \text{z.name}}(\ldots) \]

\[ \delta_{\text{cid}='cid'}(\ldots) \]

\[ \Pi_{\text{x.name}, \text{z.name}}(\ldots) \]

\[ T4(\text{name}, \text{name}) \]

\[ T3(\ldots) \]

Final answer
Extended Algebra Operators

- Union $\cup$
- Difference $-$
- Selection $\sigma$
- Projection $\Pi$
- Join $\bowtie$
- Rename $\rho$
- Duplicate elimination $\delta$
- Grouping and aggregation $\gamma$
- Sorting $\tau$

Relational Algebra

Extended Relational Algebra
Relational Algebra: Sets v.s. Bags Semantics

- Sets: \{a,b,c\}, \{a,d,e,f\}, \{\}\ldots
- Bags: \{a, a, b, c\}, \{b, b, b, b, b\}, \ldots

Relational Algebra has two semantics:
- Set semantics
- Bag semantics
Union and Difference

R1 \cup R2
R1 \setminus R2

What do they mean over bags?
What about Intersection?

- Derived operator using minus
  \[ R_1 \cap R_2 = R_1 - (R_1 - R_2) \]

- Derived using join (will explain later)
  \[ R_1 \cap R_2 = R_1 \Join R_2 \]

What is the meaning of \( \cap \) under bag semantics?
Projection

•Eliminates columns

\[ \Pi_{A_1,\ldots,A_n}(R) \]

•Example:
  - \[ \Pi_{SSN,Name}(\text{Employee}) \]
  - Answer(SSN, Name)

Semantics differs over set or over bags
Employee

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>5423341</td>
<td>John</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>John</td>
<td>20000</td>
</tr>
</tbody>
</table>

\[ \Pi_{\text{Name,Salary}}(\text{Employee}) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>John</td>
<td>60000</td>
</tr>
<tr>
<td>John</td>
<td>20000</td>
</tr>
</tbody>
</table>

Bag semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>20000</td>
</tr>
</tbody>
</table>

Set semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>John</td>
<td>60000</td>
</tr>
</tbody>
</table>

Which is more efficient?
Natural Join

\[ R1 \Join R2 = \Pi_A(\sigma(R1 \times R2)) \]

- **Meaning:** 
  \[ R1 \Join R2 = \Pi_A(\sigma(R1 \times R2)) \]

- **Where:**
  - \( \sigma \) checks equality of all common attributes
  - \( \Pi_A \) eliminates the duplicate attributes
Natural Join

\[
R \Join S = \Pi_{ABC} (\sigma_{R.B=S.B} (R \times S))
\]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>U</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
<td>W</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>U</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
</tr>
</tbody>
</table>
Natural Join

• Given schemas $R(A, B, C, D)$, $S(A, C, E)$, what is the schema of $R \bowtie S$ ?

• Given $R(A, B, C)$, $S(D, E)$, what is $R \bowtie S$ ?

• Given $R(A, B)$, $S(A, B)$, what is $R \bowtie S$ ?
Theta Join

• A join that involves a predicate

\[ R1 \bowtie_{\theta} R2 = \sigma_{\theta} (R1 \times R2) \]

• Here \( \theta \) can be any condition
  – Example band join: \( R \bowtie_{R.A-5<S.B \land S.B<R.A+5} S \)
Eq-join

• A theta join where $\theta$ is an equality

$$R1 \Join_{A=B} R2 = \sigma_{A=B} (R1 \times R2)$$

• This is by far the most used variant of join in practice
Semijoin

\[ R \smallfrown_C S = \Pi_{A_1, \ldots, A_n} (R \smallfrown_C S) \]

- Where \( A_1, \ldots, A_n \) are the attributes of \( R \)

\( R \smallfrown_C S \) returns tuples in \( R \) that join with some tuple in \( S \)
- Duplicates in \( R \) are preserved
- Duplicates in \( S \) don’t matter

Semijoin is **important**; we will return to it.
Anti-Semi-Join

- Notation: \( R \triangleright S \)
  - Warning: not a standard notation

- Meaning: all tuples in \( R \) that do NOT have a matching tuple in \( S \)
Set Difference v.s. Anti-semijoin

R(A,B)
S(B)

Plan=

SELECT DISTINCT R.B
FROM R
WHERE not exists (SELECT * FROM S
WHERE R.B=S.B)

SELECT DISTINCT *
FROM R
WHERE not exists (SELECT * FROM S
WHERE R.B=S.B)
Set Difference v.s. Anti-semijoin

R(A,B)
S(B)

Plan=

\[ \Pi_{B} \]

\[ \text{SELECT DISTINCT } R.B \]
\[ \text{FROM } R \]
\[ \text{WHERE not exists } (\text{SELECT } * \]
\[ \text{FROM } S \]
\[ \text{WHERE } R.B=S.B) \]

\[ \text{SELECT DISTINCT } * \]
\[ \text{FROM } R \]
\[ \text{WHERE not exists } (\text{SELECT } * \]
\[ \text{FROM } S \]
\[ \text{WHERE } R.B=S.B) \]
**Set Difference v.s. Anti-semijoin**

**Plan:**

```
SELECT DISTINCT R.B
FROM R
WHERE not exists (SELECT *
    FROM S
    WHERE R.B=S.B)
```

```
SELECT DISTINCT *
FROM R
WHERE not exists (SELECT *
    FROM S
    WHERE R.B=S.B)
```
Set Difference v.s. Anti-semijoin

\[ \text{SELECT DISTINCT } R.B \]
\[ \text{FROM } R \]
\[ \text{WHERE not exists (SELECT * FROM S WHERE R.B=S.B)} \]

Plan:
\[ \Pi_B R(A,B) \]

\[ \text{SELECT DISTINCT } * \]
\[ \text{FROM } R \]
\[ \text{WHERE not exists (SELECT * FROM S WHERE R.B=S.B)} \]

Plan:
\[ \Pi_B R(A,B) \]

Semi-join
Set Difference v.s. Anti-semijoin

Plan = \[ \Pi_B R(A,B) \setminus \text{Anti-semi-join} \]

Plan = \[ \Pi_B R(A,B) \setminus \text{Semi-join} \]

SELECT DISTINCT R.B
FROM R
WHERE not exists (SELECT * FROM S WHERE R.B=S.B)

SELECT DISTINCT *
FROM R
WHERE not exists (SELECT * FROM S WHERE R.B=S.B)
Operators on Bags

• Duplicate elimination $\delta(R) =$

• Grouping $\gamma_{A,sum(B)} (R) =$

• Sorting $\tau_{A,B} (R)$
Complex RA Expressions

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

SELECT u.name, count(*)
FROM Customer x, Purchase y,
    Customer z, Product u
WHERE z.name = 'fred'
    and u.name = 'gizmo'
    and y.cid = z.cid
    and y.pid = u.pid
    and x.cid = z.cid
GROUP BY u.name
Query Evaluation
Physical Operators

Each of the logical operators may have one or more implementations = physical operators

Will discuss several basic physical operators, with a focus on join
Question in Class

Purchase(pid, cid, store) \bowtie_{cid=\text{cid}} \text{Customer}(cid, name, city)

Propose three physical operators for the join, assuming the tables are in main memory:

1.
2.
3.
Question in Class

Purchase(pid, cid, store) ⨝_{cid=cid} Customer(cid, name, city)

Propose three physical operators for the join, assuming the tables are in main memory:

1. Nested Loop Join
2. Merge join
3. Hash join
1. Nested Loop Join

```plaintext
for x in Purchase do {
    for y in Customer do {
        if (x.cid == y.cid) output(x,y);
    }
}
```

Purchase = *outer relation*
Customer = *inner relation*

Note: sometimes terminology is switched

Discuss the possible use of an index Customer(cid)
Hash Tables

Separate chaining:

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

Operations on a hash table:
- find(103) = ??
- insert(488) = ??

Duplicates OK
WHY ??
2. “Classic Hash Join”

for y in Customer do  insert(y);

for x in Purchase do {
    y = find(x.cid);
    if (y != NULL)  { output(x,y); }
}

Purchase = \textit{outer relation}
Customer = \textit{inner relation}

What changes if the join attribute is not a key in the inner relation?
3. Merge Join (main memory)

Purchase1= sort(Purchase, cid);
Customer1=sort(Customer, cid)
x= Purchase1.get_next(); y= Customer1.get_next();

While (x!=NULL and y!=NULL) {
  case:
    x.cid < y.cid:    x = Purchase1.get_next();
    x.cid > y.cid:    y = Customer1.get_next();
    x.cid == y.cid { output(x,y);
                      y = Purchase1.get_next();
    }
}

Why ???

Purchase(pid, cid, store) ×_
Customer(cid, name, city)
The Iterator Model

Each operator implements this interface

• open()
• get_next()
• close()
Main Memory Nested Loop Join

open( ) {
    Purchase.open( );
    Customer.open( );
    x = Purchase.get_next( );
}

get_next( ) {
    repeat {
        y = Customer.get_next( );
        if (y == NULL) {
            Customer.close();
            x = Purchase.get_next( );
            if (x == NULL) return NULL;
            Customer.open( );
            y = Customer.get_next( );
        }
        until (x.cid == y.cid);
        return (x,y)
    }
}

close( ) {
    Purchase.close( );
    Customer.close( );
}

ALL operators need to be implemented this way!
Classic Hash Join

What do these operators do for the classic Hash Join?

- `open()`
- `get_next()`
- `close()`

\[
\text{Purchase}(\text{pid, cid, store}) \bowtie_{\text{cid}=\text{cid}} \text{Customer}(\text{cid, name, city})
\]
Discussion in class

Every operator is a hash-join and implements the iterator model

What happens:
- When we call open() at the top?
- When we call get_next() at the top?
External Memory Algorithms

• Data is too large to fit in main memory

• Issue: disk access is 3-4 orders of magnitude slower than memory access

• Assumption: runtime dominated by # of disk I/O’s; will ignore the main memory part of the runtime
Cost Parameters

The cost of an operation = total number of I/Os

Cost parameters (used both in the book and by Shapiro):

- \( B(R) \) = number of blocks for relation \( R \)
- \( T(R) \) = number of tuples in relation \( R \)
- \( V(R, A) \) = number of distinct values of attribute \( A \)
- \( M \) = size of main memory buffer pool, in blocks

Facts:
1. \( B(R) \ll T(R) \)
2. When \( A \) is a key, \( V(R, A) = T(R) \)
   When \( A \) is not a key, \( V(R, A) \ll T(R) \)
Ad-hoc Convention

• The operator *reads* the data from disk

• The operator *does not write* the data back to disk (e.g.: pipelining)

• Thus:

\[ \text{Any main memory join algorithms for } R \bowtie S: \text{ Cost } = B(R)+B(S) \]
External Memory Join Algorithms

• Nested Loop Joins

• Merge Join

• Hash join: read paper, discuss next week
External Sorting

• Problem: sort a file $R$ of size $B(R)$ with memory $M$

• Concrete:
  – Size of $R$ is $100\text{TB} = 10^{14}$
  – Size of $M$ is $1\text{GB} = 10^9$
  – Page size is $10\text{KB} = 10^4$
External Merge-Sort: Step 1

- Phase one: load M bytes in memory, sort

Can increase to length 2M using “replacement selection” (How?)
External Merge-Sort: Step 2

- Merge $M - 1$ runs into a new run
- Result: runs of length $M (M - 1) \approx M^2$

Assuming $B \leq M^2$, then we are done.

If $B > M^2$, why not merge more than $M$ runs in one step?
Cost of External Merge Sort

• Read+write+read = 3B(R)
  (we don’t count the final write)

• Assumption: B(R) <= M^2
Application: Merge-Join

Join $R \bowtie S$

• Step 1a: initial runs for $R$
• Step 1b: initial runs for $S$
• Step 2: merge and join
Merge-Join

\[ M_1 = \frac{B(R)}{M} \text{ runs for } R \]
\[ M_2 = \frac{B(S)}{M} \text{ runs for } S \]

Merge-join \( M_1 + M_2 \) runs;
need \( M_1 + M_2 \leq M \), or \( B(R) + B(S) \leq M^2 \)