Advanced Topics in Data Management

Lecture 2
Announcements

• Review 1 was due today
• Review 2 due next Thursday

• Today: continue query processing
• Next Thursday:
  – Rebecca Taft, Cockroach Labs optimizer
  – Max Willsey, EGG equality saturation
Outline for Today

• Recap
• Dynamic Programming
• Subqueries
• Operator Interface
• Cardinality Estimation
Recap

• Relational data model:

• Logical Data Model = relations
• Physical Data Model = NONE!

• Physical data independence
Recap

• SELECT-FROM-WHERE
  – Nested loop semantics

• Complications:
  – NULLs
  – Group-by / Aggregates
  – Outer Joins
  – Subqueries
Recap: Relational Algebra

- Selection: $\sigma$
- Projection: $\Pi$
- Join: $\bowtie$
- Union: $\cup$
- Group-by, aggregate: $\gamma$
- Outer-join: $\ltimes$, $\ltimes$, $\lhd$
- Semi-join: $\ltimes$, Anti-semi-join: $\lhd$
Recap: Physical Plans

• Joins:
  – Nested loop join
  – Hash-join
  – Merge-join
  – Extensions: disk-based, distributed

• Group-by:
  – Hash, merge

• Selection: on-the-fly
Dynamic Programming
Dynamic Programming

• Let n = number of relations to join

• For s = 1, n do:
  
  – For each subset S of of size s do:
    
    • Split S into [relation R] + [set of s-1 relations S’]
    
    • Lookup Cost(S’)
    
    • Cost(S) := \( \min_{\text{splits}} (\text{Cost}(S’) + \text{cost-of}(R \bowtie S’)) \)
    
    • Memorize (S, Cost(S))

• Return Cost(All-relations)
Paper Discussion

• Analysis of Two Existing and One New Dynamic Programming Algorithm, VLDB’2006

• Assumes only plans without cartesian products (why?)
Cartesian Products

\[ R(A, B) \bowtie_{R.B=S.B} S(B, C) \bowtie_{S.C=T.C} T(C, D) \]
Cartesian Products

\[ R(A,B) \bowtie_{R.B=S.B} S(B,C) \bowtie_{S.C=T.C} T(C,D) \]
Cartesian Products

\[ R(A, B) \bowtie_{R.B=S.B} S(B, C) \bowtie_{S.C=T.C} T(C, D) \]
Cartesian Products

\[ R(A,B) \bowtie_{R.B=S.B} S(B,C) \bowtie_{S.C=T.C} T(C,D) \]

Without cartesian product
Cartesian Products

$R(A, B) \bowtie_{R.B = S.B} S(B, C) \bowtie_{S.C = T.C} T(C, D)$

Without cartesian product

With cartesian product
**Cartesian Products**

\[ R(A, B) \bowtie_{R.B=S.B} S(B, C) \bowtie_{S.C=T.C} T(C, D) \]

**Without cartesian product**

\[ R(A, B) \bowtie_{R.B=S.B} \bowtie_{S.C=T.C} T(C, D) \]

**With cartesian product**

\[ R(A, B) \bowtie_{R.B=S.B} \bowtie_{S.C=T.C} T(C, D) \]

When could this plan be better?
Left-Deep v.s. Bush Trees
Some Popular Query Shapes

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph:

```
R_1 ---- R_2 ---- R_3 ---- \ldots ---- R_n
```
Some Popular Query Shapes

Chain query: \[ R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n) \]

Query graph: 
\[ R_1 \quad \cdots \quad R_n \]

Star query: \[ S(X_1, X_2, \ldots, X_n) \bowtie R_1(X_1) \bowtie R_2(X_2) \bowtie \cdots \bowtie R_n(X_n) \]

Query graph: 
\[ \text{Star graph} \]
Some Popular Query Shapes

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph

Star query: $S(X_1, X_2, \ldots, X_n) \bowtie R_1(X_1) \bowtie R_2(X_2) \bowtie \cdots \bowtie R_n(X_n)$

Query graph

Clique query: very rare in practice
Number of Left-Deep Plans

Chain query: \( R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n) \)

Query graph: \( R_1 \longrightarrow R_2 \longrightarrow R_3 \longrightarrow \cdots \longrightarrow R_n \)
Number of Left-Deep Plans

Chain query: \[ R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n) \]

Query graph

W/ cartesian product:
Number of Left-Deep Plans

Chain query: \[ R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n) \]

Query graph

With cartesian product: Can place relations in any order
Number of Left-Deep Plans

Chain query: \[ R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n) \]

Query graph: \[ R_1 \quad R_2 \quad R_3 \quad \ldots \quad R_n \]

With cartesian product:

Can place relations in any order

n!
Number of Left-Deep Plans

Chain query: \[ R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n) \]

Query graph

W/ cartesian product:

W/o cartesian product:
Number of Left-Deep Plans

Chain query: \[ R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n) \]

Query graph

W/ cartesian product:

W/o cartesian product:

May start anywhere

n!
Number of Left-Deep Plans

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph:

W/ cartesian product:

W/o cartesian product:

May start anywhere

Add either left or right
Number of Left-Deep Plans

Chain query:
\[ R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \ldots \bowtie R_n(X_{n-1}, X_n) \]

Query graph
\[ R_1 \quad R_2 \quad R_3 \quad \ldots \quad R_n \]

W/ cartesian product:
\[ R_1 \bowtie R_2 \bowtie R_9 \bowtie R_{15} \bowtie \ldots \bowtie R_n \]

W/o cartesian product:
\[ R_1 \bowtie R_2 \bowtie R_7 \bowtie \ldots \bowtie R_n \]

May start anywhere
Add either left or right
Number of Left-Deep Plans

Chain query:
\[ R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n) \]

Query graph

W/ cartesian product:

W/o cartesian product:

May start anywhere

Add either left or right
Number of Left-Deep Plans

Chain query:

\[ R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n) \]

Query graph:

\[ R_1 \quad R_2 \quad R_3 \quad \ldots \quad R_n \]

W/ cartesian product:

\[ R_7 \quad R_9 \quad R_{15} \quad \ldots \]

W/o cartesian product:

\[ R_7 \quad R_8 \quad R_9 \quad 2^{n-1} \]
Discussion

- n! can be much worse than $2^{n-1}$.

- Other query shapes (star, clique, various) have similar behavior

- Hence: avoid cartesian products
Left-Deep, No C.P.

• Let $n =$ number of relations to join
• For $s = 1, n$ do:
  – For each connected $S$ of size $s$ do:
    • $S = \{R\} \cup S'$, where $S'$ is connected
    • Lookup $\text{Cost}(S')$
    • $\text{Cost}(S) := \min_{\text{splits}} (\text{Cost}(S') + \text{cost-of}(R \bowtie S'))$
    • Memorize $(S, \text{Cost}(S))$
• Return $\text{Cost(All-relations)}$
Analysis for **Chain Query**

- Let $n =$ number of relations to join
- For $s = 1, n$ do:
  - For each **connected** $S$ of of size $s$ do:
    - $S = \{R\} \cup S'$, where $S'$ is connected
    - Lookup Cost($S'$)
    - Cost($S$) := $\min_{\text{split}} (\text{Cost}(S') + \text{cost-of}(R \bowtie S'))$
    - Memorize ($S$, Cost($S$))
- Return Cost(All-relations)
Analysis for Chain Query

• Let \( n \) = number of relations to join

• For \( s = 1, n \) do:
  – For each connected \( S \) of size \( s \) do:
    • \( S = \{R\} \cup S' \), where \( S' \) is connected
    - Lookup Cost(\( S' \))
    - \( \text{Cost}(S) := \min_{\text{splits}} (\text{Cost}(S') + \text{cost-of}(R \bowtie S')) \)
    • Memorize (\( S, \text{Cost}(S) \))

• Return Cost(All-relations)
Analysis for **Chain Query**

- Let \( n \) = number of relations to join
- For \( s = 1, n \) do:
  - For each **connected** \( S \) of size \( s \) do:
    - \( S = \{R\} \cup S' \), where \( S' \) is **connected**
    - Lookup Cost(\( S' \))
    - Cost(\( S \)) := \( \min_{\text{splits}} (\text{Cost}(S') + \text{cost-of}(R \bowtie S')) \)
    - Memorize (\( S, \text{Cost}(S) \))

**Runtime:**
\[
\sum_{s=1}^{n} (n - s + 1) \cdot 2 = n(n + 1) = O(n^2)
\]
Paper: “Analysis of Two…”

- Bushy plans, no cartesian product
- Challenge: how do we enumerate efficiently the pairs $S_1, S_2$ where:
  - $S_1, S_2$ are disjoint
  - Each of $S_1, S_2$ is connected
  - $S_1$ and $S_2$ are connected to each other
- Solution:
  - EnumerateCSG and EnumerateCMP
  - EnumerateCMP had a bug, reported in an Errata
Discussion

• Dynamic programming can be faster than exploring all possible plans
• Still, it runs in exponential time
• Database systems have a limit (configurable) of how many tables they optimize using dynamic programming; beyond that, they use some heuristics
• Outer-, anti- joins add extra complexity
Subqueries in SQL
Subqueries

• Subquery in SELECT:
  – Must return single value

• Subquery in FROM
  – Like a temporary relation
  – Alternative: use the WITH clause

• Subquery in WHERE or in HAVING
  – Can express sophisticated queries
Subquery in SELECT

Compute the number of products sold by each supplier

```
SELECT x.sno, x.sname, 
    (SELECT count(*)
     FROM Supply y
     WHERE x.sno = y.sno)
FROM Supplier x
```
Subquery in FROM

Better: use the WITH statement!
Subquery in FROM

Better: use the WITH statement!

Find the supplier who supplies the maximum number of parts
Subquery in FROM

Better: use the WITH statement!

Find the supplier who supplies the maximum number of parts

WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
  FROM Supplier x, Supply y
  WHERE x.sno = y.sno
  GROUP BY x.sno, x.sname)

For each supplier, compute how many parts they supply
Subquery in FROM

Better: use the WITH statement!

Find the supplier who supplies the maximum number of parts

```sql
WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
    FROM Supplier x, Supply y
    WHERE x.sno = y.sno
    GROUP BY x.sno, x.sname),
Mx AS (SELECT max(c) as m
    FROM Cnt)
SELECT z.sno, z.sname, m.m
    FROM Cnt z, Mx m
    WHERE z.c = m.m;
```

Find the maximum
Supplier(sno, sname, scity, sstate)
Supply(sno, pno, qty, price)
Part(pno, pname, psize, pcolor)

Subquery in FROM

Better: use the WITH statement!

Find the supplier who supplies the maximum number of parts

WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
FROM Supplier x, Supply y
WHERE x.sno = y.sno
GROUP BY x.sno, x.sname),
Mx AS (SELECT max(c) as m
FROM Cnt)
SELECT z.sno, z.sname, m.m
FROM Cnt z, Mx m
WHERE z.c = m.m;

Find the “witness”, i.e. the supplier that supplies the maximum number of parts; argmax
Subquery in FROM

WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
  FROM Supplier x, Supply y
  WHERE x.sno = y.sno
  GROUP BY x.sno, x.sname),
  Mx AS (SELECT max(c) as m
  FROM Cnt)
SELECT z.sno, z.sname, m.m
FROM Cnt z, Mx m
WHERE z.c = m.m;

Query Plan:
Supplier(sno, sname, scity, sstate)
Supply(sno, pno, qty, price)
Part(pno, pname, psize, pcolor)

Subquery in FROM

WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
FROM Supplier x, Supply y
WHERE x.sno = y.sno
GROUP BY x.sno, x.sname),
Mx AS (SELECT max(c) as m
FROM Cnt)
SELECT z.sno, z.sname, m.m
FROM Cnt z, Mx m
WHERE z.c = m.m;

Query Plan:
Subquery in FROM

WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
  FROM Supplier x, Supply y
  WHERE x.sno = y.sno
  GROUP BY x.sno, x.sname),
  Mx AS (SELECT max(c) as m
  FROM Cnt)

SELECT z.sno, z.sname, m.m
FROM Cnt z, Mx m
WHERE z.c = m.m;

Query Plan:

Needs to be a tree!
Subquery in FROM

```
WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
FROM Supplier x, Supply y
WHERE x.sno = y.sno
GROUP BY x.sno, x.sname),
Mx AS (SELECT max(c) as m
FROM Cnt)
SELECT z.sno, z.sname, m.m
FROM Cnt z, Mx m
WHERE z.c = m.m;
```
Subquery in FROM

```sql
WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
  FROM Supplier x, Supply y
  WHERE x.sno = y.sno
  GROUP BY x.sno, x.sname),
  Mx AS (SELECT max(c) as m
    FROM Cnt)
SELECT z.sno, z.sname, m.m
FROM Cnt z, Mx m
WHERE z.c = m.m;
```

In Relational Algebra, it is easy to compose operations.
Subquery in WHERE

Find suppliers that supply some ‘blue’ parts
Subquery in WHERE

Find suppliers that supply some ‘blue’ parts

```
SELECT x.sno
FROM Supplier x
WHERE exists (SELECT * FROM Supply y, Part z
  WHERE x.sno=y.sno
  and y.pno=z.pno
  and z.pcolor = 'blue');
```
Subquery in WHERE

Find suppliers that supply some ‘blue’ parts

```
SELECT x.sno
FROM Supplier x
WHERE exists (SELECT * FROM Supply y, Part z
WHERE x.sno=y.sno
and y.pno=z.pno
and z.pcolor = 'blue');
```

A correlated subquery: meaning that it depends on the variable x defined by the outer query.
Subquery in WHERE

Find suppliers that supply *only* ‘red’ parts
Subquery in WHERE

Find suppliers that supply only ‘red’ parts

Find the other suppliers
Subquery in WHERE

Find suppliers that supply *only* ‘red’ parts

```sql
SELECT x.sno
FROM Supplier x
WHERE exists (SELECT * FROM Supply y, Part z
    WHERE x.sno=y.sno
    and y.pno=z.pno
    and z.pcolor != 'red');
```

Find the *other* suppliers
Find suppliers that supply only ‘red’ parts

```
SELECT x.sno
FROM Supplier x
WHERE exists (SELECT * FROM Supply y, Part z
  WHERE x.sno=y.sno
  and y.pno=z.pno
  and z.pcolor != 'red');
```

Find the other suppliers

```
SELECT x.sno
FROM Supplier x
WHERE not exists (SELECT * FROM Supply y, Part z
  WHERE x.sno=y.sno
  and y.pno=z.pno
  and z.pcolor != 'red');
```

Negate to get the right ones
Relational Algebra

• Semijoin: \( R \bowtie S \)
  – Subset of R that joins with S
  – \( R \bowtie S = \Pi_{\text{Attrs}(R)}(R \bowtie S) \)

• Anti-semijoin: \( R \triangleright S \)
  – Subset of R that does not join with S
  – \( R \triangleright S = R - (R \bowtie S) \)
Semi-Join

Find suppliers that supply some ‘blue’ parts

```
SELECT x.sno
FROM Supplier x
WHERE exists (SELECT * FROM Supply y, Part z
               WHERE x.sno=y.sno
               and y.pno=z.pno
               and z.pcolor = 'blue');
```
Semi-Join

Find suppliers that supply some ‘blue’ parts

```
SELECT x.sno
FROM Supplier x
WHERE exists (SELECT * FROM Supply y, Part z
WHERE x.sno=y.sno
    and y.pno=z.pno
    and z.pcolor = 'blue');
```

Semi-join does not introduce duplicates
Anti-semi-Join

Find suppliers that supply \textit{only} ‘red’ parts

\begin{verbatim}
SELECT x.sno
FROM Supplier x
WHERE not exists (SELECT * FROM Supply y, Part z
    WHERE x.sno=y.sno
    and y.pno=z.pno
    and z.pcolor != 'red');
\end{verbatim}
Anti-semi-Join

Find suppliers that supply only ‘red’ parts

```
SELECT x.sno
FROM Supplier x
WHERE not exists (SELECT *
FROM Supply y, Part z
WHERE x.sno=y.sno
    and y.pno=z.pno
    and z.pcolor != 'red');
```
Discussion

• RA does not have variables, no nested expressions
  – Correlated subqueries need to be decorrelated first
  – Nested subqueries then need to be unnested

• Some systems fail to unnest complicated queries: nested loop join
Operator Interface
How Do We Combine Them?
How Do We Combine Them?

Option 1:
materialize intermediate results

Option 2:
Pipeline tuples btw. ops
How Do We Combine Them?

Option 1:
materialize intermediate results

Option 2:
Pipeline tuples btw. ops

Implementation:
Iterator Interface
Operator Interface

Volcano model:

- **open(), next(), close()**
- Pull model
- Volcano optimizer: G. Graefe’s (Wisconsin) → SQL Server
- Supported by most DBMS today
- Will discuss next
Operator Interface

Volcano model:
• `open()`, `next()`, `close()`
• Pull model
• Volcano optimizer: G. Graefe’s (Wisconsin) \(\rightarrow\) SQL Server
• Supported by most DBMS today

Data-driven model:
• `open()`, `produce()`, `consume()`, `close()`
• Push model
• Introduced by Thomas Neumann in Hyper (at TU Munich), later acquired by Tableau
Both joins are hash-Joins

\[
R(A, B, C) \bowtie_{R.A = S.A} S(A, \ldots) \bowtie_{R.B = T.B} T(B, \ldots)
\]
Both joins are hash-Joins

\[ R(A, B, C) \quad \Join_{R.A=S.A} \quad S(A, ...) \quad \Join_{R.B=T.B} \quad T(B, ...) \]

open( )
Operator Interface

Both joins are hash-Joins

R(A, B, C) \bowtie_{R.A=S.A} S(A, ...) \bowtie_{R.B=T.B} T(B, ...)

open( )

Hash-table
Both joins are hash-Joins
Both joins are hash-Joins

\begin{align*}
R(A, B, C) & \bowtie_{R.A=S.A} S(A, ...) \\
& \bowtie_{R.B=T.B} T(B, ...) \\
\end{align*}

open( )

next( )

Hash-table
Both joins are hash-Joins

- \( R(A, B, C) \) \( \bowtie \) \( R.A = S.A \) \( \bowtie \) \( R.B = T.B \)
- \( S(A, ...) \)
- \( T(B, ...) \)

Operator Interface

open( )

next( )

Hash-table

| B   | ...
|-----|--
| b1  |
Both joins are hash-Joins

- \( R.A = S.A \)
- \( R.B = T.B \)

\[ R(A, B, C) \]

\[ S(A, \ldots) \]

\[ T(B, \ldots) \]

Hash-table

- B
- b1

open()

next()
Both joins are hash-Joins

\[ R(A, B, C) \]

\[ S(A, ...) \]

\[ T(B, ...) \]

\[ \bowtie_{R.A = S.A} \]

\[ \bowtie_{R.B = T.B} \]

open( )

Hash-table

<table>
<thead>
<tr>
<th>B</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td></td>
</tr>
</tbody>
</table>

next( )

next( )

next( )
Both joins are hash-Joins

\[ R(A, B, C) \text{ } \bowtie \text{ } R.A = S.A \]

\[ S(A, \ldots) \text{ } \bowtie \text{ } R.B = T.B \]

\[ T(B, \ldots) \]

Hash-table

\begin{array}{c|c}
B & \ldots \\
\hline
b1 & \\
\hline
b2 & \\
\hline
\ldots & \ldots
\end{array}

open( )

next( )

next( )

next( )

…”
Both joins are hash-Joins

$R(A, B, C) \bowtie_{R.A=S.A} S(A, \ldots)$

$S(A, \ldots) \bowtie_{R.B=T.B} T(B, \ldots)$

Hash-table

<table>
<thead>
<tr>
<th>B</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

open( )
close( )
Both joins are hash-Joins

\[ R(A, B, C) \bowtie_{R.A = S.A} S(A, \ldots) \bowtie_{R.B = T.B} T(B, \ldots) \]
Both joins are hash-Joins

- $R(A, B, C)$
- $S(A, ...)$
- $T(B, ...)$

Hash-table

<table>
<thead>
<tr>
<th>$B$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Both joins are hash-Joins

\[ \text{Operator Interface} \]

\[ R(A, B, C) \]

\[ S(A, \ldots ) \ldots \]

\[ T(B, \ldots ) \]

\[ \bowtie_{R.A = S.A} \]

\[ \bowtie_{R.B = T.B} \]

\[ \text{open( )} \]

\[ \text{next( )} \]

\[ \text{open( )} \]

\[ \text{Hash-table} \]

<table>
<thead>
<tr>
<th>B</th>
<th>\ldots</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
</tr>
</tbody>
</table>
Operator Interface

Both joins are hash-Joins

\[ R(A, B, C) \]

\[ S(A, ...) \]

\[ T(B, ...) \]

\[ \bowtie_{R.A=S.A} \]

\[ \bowtie_{R.B=T.B} \]

open( )

close( )

Hash-table

<table>
<thead>
<tr>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
</tr>
<tr>
<td>b2</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
</tr>
<tr>
<td>a2</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

open( )

open( )
Both joins are hash-Joins

\[ R(A, B, C) \]  \( \bowtie \)  \( R.A = S.A \)  \[ S(A, \ldots) \]  \( \bowtie \)  \( R.B = T.B \)  \[ T(B, \ldots) \]
Operator Interface

Both joins are hash-Joins

$R(A, B, C)$

$S(A, ...)$

$T(B, ...)$

$R.A = S.A$

$R.B = T.B$

Hash-table
Both joins are hash-Joins

\[ R(A, B, C) \]

\[ S(A, ...) \]

\[ T(B, ...) \]

\[ R.B = T.B \]

\[ R.A = S.A \]

Operator Interface
Both joins are hash-Joins
Operator Interface

Both joins are hash-Joins

\[ R(A, B, C) \quad \bowtie_{R.A=S.A} \quad S(A, ...) \quad \bowtie_{R.B=T.B} \quad T(B, ...) \]

next( )

Hash-table

<table>
<thead>
<tr>
<th>B</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

A | ... |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Both joins are hash-Joins

\[ R(A, B, C) \bowtie_{R.A=S.A} S(A, ...) \bowtie_{R.B=T.B} T(B, ...) \]
Both joins are hash-Joins

R(A, B, C) \ join \ S(A, ...) \ join \ T(B, ...)

next( )

Hash-table

\begin{array}{|c|c|}
\hline
B & \ldots \\
\hline
b1 & \ldots \\
\hline
b2 & \ldots \\
\hline
\end{array}

\begin{array}{|c|}
\hline
A \\
\hline
a1 \\
\hline
a2 \\
\hline
\ldots \\
\hline
\end{array}
Both joins are hash-Joins

\begin{align*}
R(A, B, C) & \bowtie_{R.A = S.A} S(A, \ldots) \\
& \bowtie_{R.B = T.B} T(B, \ldots)
\end{align*}
Both joins are hash-Joins

\[ R(A, B, C) \]

\[ S(A, \ldots) \]

\[ T(B, \ldots) \]
Both joins are hash-Joins

\[ R(A, B, C) \quad \bowtie \quad S(A, ...) \quad \bowtie\quad R.B = T.B \]

\[ (a7, b5, c12) \quad \text{next( )} \quad \text{No match} \quad \text{next( )} \]

\[ R \quad A \quad S \quad T \]

\[ a1, a2, ... \]

\[ b1, b2, ... \]

\[ c12 \]

\[ \text{Hash-table} \]
Both joins are hash-Joins

$R(A, B, C)$

$S(A, ...)$

$T(B, ...)$

$\bowtie R.A = S.A$

$\bowtie R.B = T.B$

next( )

next( )

next( )

Hash-table

<table>
<thead>
<tr>
<th>B</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Operator Interface

Both joins are hash-Joins

$R(A, B, C)$
$S(A, ...)$
$T(B, ...)$

$\bowtie_{R.A = S.A}$
$\bowtie_{R.B = T.B}$

next( )

probe

Hash-table

<table>
<thead>
<tr>
<th>B</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Both joins are hash-Joins

3 matches

\[
\begin{array}{|c|c|c|c|}
\hline
a9 & b1 & c7 & \text{val}1 \\
\hline
a9 & b1 & c7 & \text{val}2 \\
\hline
a9 & b1 & c7 & \text{val}3 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
A & \ldots \\
\hline
a1 & \text{} \\
\hline
a2 & \text{} \\
\hline
\ldots & \ldots \\
\hline
\end{array}
\]

\[
\begin{array}{|c|}
\hline
B & \ldots \\
\hline
b1 & \text{} \\
\hline
b2 & \text{} \\
\hline
\ldots & \ldots \\
\hline
\end{array}
\]

Operator Interface
Operator Interface

Both joins are hash-Joins

\[ R(A, B, C) \]

\[ S(A, \ldots) \]

\[ T(B, \ldots) \]

\[ R.A = S.A \]

\[ R.B = T.B \]

next( )

Return 1, save the other 2

\[ R.A = S.A \]

\[ R.B = T.B \]

[Table]

<table>
<thead>
<tr>
<th>a9</th>
<th>b1</th>
<th>c7</th>
<th>val1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a9</td>
<td>b1</td>
<td>c7</td>
<td>val2</td>
</tr>
<tr>
<td>a9</td>
<td>b1</td>
<td>c7</td>
<td>val3</td>
</tr>
</tbody>
</table>

[Table]

<table>
<thead>
<tr>
<th>B</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

[Table]

<table>
<thead>
<tr>
<th>A</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Both joins are hash-Joins

\[
\begin{align*}
R(A, B, C) & \bowtie R.A = S.A \\
S(A, ...) & \\
T(B, ...) & \bowtie R.B = T.B
\end{align*}
\]
Both joins are hash-Joins

R(A, B, C)  \( \bowtie R.A = S.A \)  \( \bowtie R.B = T.B \)

\( S(A, \ldots) \)

\( T(B, \ldots) \)

Hash-table

\begin{array}{|c|}
\hline
B \\
\hline ...
\hline
b1
\hline
b2
\hline ...
\hline
\end{array}

\begin{array}{|c|}
\hline
a9 \\
\hline b1 \\
\hline c7 \\
\hline val1
\hline
\end{array}

\begin{array}{|c|}
\hline
a9 \\
\hline b1 \\
\hline c7 \\
\hline val2
\hline
\end{array}

\begin{array}{|c|}
\hline
a9 \\
\hline b1 \\
\hline c7 \\
\hline val3
\hline
\end{array}

Return: a9 b1 c7 val1 xyz
Both joins are hash-Joins

\[ R(A, B, C) \] \[ S(A, \ldots) \] \[ T(B, \ldots) \]

next( )

\[ R.B = T.B \]

\[ R.A = S.A \]
Operator Interface

Both joins are hash-Joins

\[ R(A, B, C) \]

\[ S(A, \ldots) \]

\[ T(B, \ldots) \]

\[ R.B = T.B \]

\[ R.A = S.A \]
Both joins are hash-Joins

\[
R(A, B, C) \quad \bowtie \quad S(A, ...) \quad \bowtie \quad T(B, ...)\]

\[
\begin{array}{cccc}
   a9 & b1 & c7 & \text{val1} \\
   a9 & b1 & c7 & \text{val2} \\
   a9 & b1 & c7 & \text{val3} \\
\end{array}
\]

\[
\begin{array}{c}
   A \\
   a1 \\
   a2 \\
   ... \\
\end{array}
\]

next( )

next( )

Hash-table

B

b1

b2

...

...
Both joins are hash-Joins

\[ R(A, B, C) \]

\[ S(A, ...) \]

\[ T(B, ...) \]

\[ R.B = T.B \]

\[ R.A = S.A \]

Return 1, save 1

next( )

\[ \text{Hash-table} \]

\begin{array}{|c|c|c|c|}
\hline
a9 & b1 & c7 & val1 \\
\hline
a9 & b1 & c7 & val2 \\
\hline
a9 & b1 & c7 & val3 \\
\hline
\end{array}
Operator Interface

Both joins are hash-Joins

\[ R(A,B,C) \]

\[ S(A,\ldots) \]

\[ T(B,\ldots) \]

\[ \bowtie R.A = S.A \]

\[ \bowtie R.B = T.B \]

next( )

And so on, until all tuples are computed
Discussion

• Most systems adopt the Volcano-model, a.k.a. the iterator interface
• Vectorized processing = iterator interface that processes a block of tuples (vector?) instead of one tuple
• Compiled model = compile to machine code and use the push model
Cardinality Estimation
Cardinality Estimation

**Problem**: given statistics on base tables and a query, estimate size of the answer

Very difficult, because:

- Need to do it very fast
- Need to use very little memory
Statistics on Base Data

- Number of tuples (cardinality) \( T(R) \)
- Number of physical pages \( B(R) \)
- Indexes, number of keys in the index \( V(R,a) \)
- Histogram on single attribute (1d)
- Histogram on two attributes (2d)

Computed periodically, often using sampling
Assumptions

- Uniformity
- Independence
- Containment of values
- Preservation of values
Size Estimation

**Selection**: size decreases by *selectivity factor* $\theta$

$$T(\sigma_{\text{pred}}(R)) = \theta_{\text{pred}} \times T(R)$$
Size Estimation

Selection: size decreases by *selectivity factor* $\theta$

\[
T(\sigma_{\text{pred}}(R)) = \theta_{\text{pred}} \cdot T(R)
\]

\[
T(R \bowtie_{A=B} S) = \theta_{A=B} \cdot T(R) \cdot T(S)
\]
Selectivity Factors

*Uniformity assumption*

Equality:

$$T(\sigma_{\text{pred}}(R)) = \theta_{\text{pred}} * T(R)$$
Selectivity Factors

**Uniformity assumption**

Equality:
- \( \theta_{A=c} = 1/V(R,A) \)

\[ T(\sigma_{\text{pred}}(R)) = \theta_{\text{pred}} \times T(R) \]
Selectivity Factors

**Uniformity assumption**

Equality:

- \( \theta_{A=c} = 1/\sqrt{V(R,A)} \)

Range:

- \( \theta_{c_1<A<c_2} = (c_2 - c_1)/(\max(R,A) - \min(R,A)) \)

\[ T(\sigma_{\text{pred}}(R)) = \theta_{\text{pred}} \times T(R) \]
Selectivity Factors

Uniformity assumption

Equality:

- \( \theta_{A=c} = 1/V(R,A) \)

Range:

- \( \theta_{c1<A<c2} = (c2 - c1)/(\max(R,A) - \min(R,A)) \)

Conjunction

\[ T(\sigma_{\text{pred}}(R)) = \theta_{\text{pred}} \times T(R) \]
Selectivity Factors

**Uniformity assumption**

Equality:

- $\theta_{A=c} = 1/V(R,A)$

Range:

- $\theta_{c1<A<c2} = (c2 - c1)/(\max(R,A) - \min(R,A))$

**Independence assumption**

Conjunction

- $\theta_{\text{pred1 and pred2}} = \theta_{\text{pred1}} \cdot \theta_{\text{pred2}} = 1/V(R,A) \cdot 1/V(R,B)$
Selectivity Factors

Join

\[ T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S) \]
Selectivity Factors

Join
• $\theta_{R.A=S.B} = \frac{1}{\text{MAX}(V(R,A), V(S,B))}$

Why? Will explain next...
Warmup: Key / Foreign-key Join

Supplier(\textit{sno}, \textit{sname}, \textit{scity}, \textit{sstate})
Supply(\textit{sno}, \textit{pno}, \textit{qty}, \textit{price})

We know \( T(\text{Supplier}) \) and \( T(\text{Supply}) \)

Q: How large is Supplier \( \bowtie \) Supply?
Warmup: Key / Foreign-key Join

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)

We know T(Supplier) and T(Supply)

Q: How large is Supplier $\bowtie$ Supply?

A: $T(\text{Supplier} \bowtie \text{Supply}) = T(\text{Supply})$

Make sure you understand why…
Selectivity Factors

**Containment of values:** if \( V(R,A) \leq V(S,B) \), then the set of \( A \) values of \( R \) is included in the set of \( B \) values of \( S \)

- Note: this indeed holds when \( A \) is a foreign key in \( R \), and \( B \) is a key in \( S \)

\[
T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)
\]
Selectivity Factors

Assume $V(R,A) \leq V(S,B)$

- Tuple $t$ in $R$ joins with $T(S)/V(S,B)$ tuples in $S$
Selectivity Factors

Assume $V(R,A) \leq V(S,B)$

- Tuple $t$ in $R$ joins with $T(S)/V(S,B)$ tuples in $S$
- Hence $T(R \bowtie_{A=B} S) = T(R) \cdot T(S) / V(S,B)$
Selectivity Factors

Assume $V(R,A) \leq V(S,B)$

• Tuple $t$ in $R$ joins with $T(S)/V(S,B)$ tuples in $S$
• Hence $T(R \bowtie_{A=B} S) = T(R) \cdot T(S) / V(S,B)$

In general:

• $T(R \bowtie_{A=B} S) = T(R) \cdot T(S) / \max(V(R,A), V(S,B))$
• $\theta_{R.A=S.B} = 1 / (\max(V(R,A), V(S,B)))$

$$T(R \bowtie_{A=B} S) = \theta_{A=B} \cdot T(R) \cdot T(S)$$
Key / Foreign-key Join

Supplier\((sno, sname, scity, sstate)\)
Supply\((sno, pno, qty, price)\)

We know \(T(\text{Supplier})\) and \(T(\text{Supply})\)

Q: How large is \(\text{Supplier} \bowtie \text{Supply}\) ?

A: \(T(\text{Supplier} \bowtie \text{Supply}) = \frac{T(\text{Supplier}) \times T(\text{Supply})}{\max(V(\text{Supplier}, sno), V(\text{Supply}, sno))}\)
Key / Foreign-key Join

Supplier(sno, sname, scity, sstate)
Supply(sno, pno, qty, price)

We know T(Supplier) and T(Supply)

Q: How large is Supplier \( \bowtie \) Supply?

A: \[
T(\text{Supplier} \bowtie \text{Supply}) = \frac{T(\text{Supplier}) \times T(\text{Supply})}{\max(V(\text{Supplier}, \text{sno}), V(\text{Supply}, \text{sno}))}
\]
\[
= \frac{T(\text{Supplier}) \times T(\text{Supply})}{V(\text{Supplier}, \text{sno})}
\]
Key / Foreign-key Join

Supplier(sno, sname, scity, sstate)
Supply(sno, pno, qty, price)

We know $T(\text{Supplier})$ and $T(\text{Supply})$

Q: How large is $\text{Supplier} \bowtie \text{Supply}$?

A: $T(\text{Supplier} \bowtie \text{Supply}) =$

\[
= T(\text{Supplier}) \times T(\text{Supply}) / \max(V(\text{Supplier}, sno), V(\text{Supply}, sno))
\]

\[
= T(\text{Supplier}) \times T(\text{Supply}) / V(\text{Supplier}, sno)
\]

\[
= T(\text{Supply})
\]
Final Assumption

*Preservation of values:* For any other attribute C:

- $V(R \bowtie_{A=B} S, C) = V(R, C)$ or
- $V(R \bowtie_{A=B} S, C) = V(S, C)$

- This is needed higher up in the plan.
Computing the Cost of a Plan

• Estimate cardinalities bottom-up

• Estimate cost by using estimated cardinalities

• Examples next...
Logical Query Plan 1

\[
\pi_{\text{name}} \\
\sigma_{\text{pno}=2 \land \text{scity}='Seattle' \land \text{sstate}='WA'} \\
\text{sid} = \text{sid}
\]

SELECT \text{name} \\
FROM \text{Supplier x, Supply y} \\
WHERE x.\text{sid} = y.\text{sid} \\
and y.\text{pno} = 2 \\
and x.\text{scity} = 'Seattle' \\
and x.\text{sstate} = 'WA'

\text{T(Supply) = 10000} \\
\text{B(Supply) = 100} \\
\text{V(Supply, pno) = 2500} \\
\text{T(Supplier) = 1000} \\
\text{B(Supplier) = 100} \\
\text{V(Supplier, scity) = 20} \\
\text{V(Supplier, state) = 10} \\
\text{M=11}
Logical Query Plan 1

\[ \sigma_{\text{pno}=2 \land \text{scity}='Seattle' \land \text{sstate}='WA'} \]

\[ \pi_{\text{sname}} \]

\[ \text{T} = 10000 \]

Supplier \((\text{sid}, \text{sname}, \text{scity}, \text{sstate})\)
Supply \((\text{sid}, \text{pno}, \text{quantity})\)

\[ \text{T(Supply)} = 10000 \]
\[ \text{B(Supply)} = 100 \]
\[ \text{V(Supply, pno)} = 2500 \]
\[ \text{T(Supplier)} = 1000 \]
\[ \text{B(Supplier)} = 100 \]
\[ \text{V(Supplier, scity)} = 20 \]
\[ \text{V(Supplier, state)} = 10 \]

Estimated (why?)

M=11
Logical Query Plan 1

\[ \sigma_{pno=2 \land scity='Seattle' \land sstate='WA'} \]

\[ \pi_{\text{sname}} \]

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
AND y.pno = 2
AND x.scity = 'Seattle'
AND x.sstate = 'WA'

Because key / foreign-key

Estimated (why?)

T = 10000

M=11

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Logical Query Plan 1

\[
\begin{align*}
\sigma_{pno=2 \land scity='Seattle' \land sstate='WA'} \\
\pi_{sname} \quad \text{T} = 10000
\end{align*}
\]

Because key / foreign-key

\[
\text{sid} = \text{sid}
\]

Also: \( \theta = 1 / \max(V(\text{Supply}, \text{sid}) \cdot V(\text{Supplier}, \text{sid})) = 1 / V(\text{Supplier}, \text{sid}) \)

Supplier

\[
\begin{align*}
T(\text{Supplier}) &= 1000 \\
B(\text{Supplier}) &= 100 \\
V(\text{Supplier}, \text{scity}) &= 20 \\
V(\text{Supplier}, \text{sstate}) &= 10
\end{align*}
\]

\( M = 11 \)

Supply

\[
\begin{align*}
T(\text{Supply}) &= 10000 \\
B(\text{Supply}) &= 100 \\
V(\text{Supply}, \text{pno}) &= 2500
\end{align*}
\]
Logical Query Plan 1

**SELECT** sname
**FROM** Supplier x, Supply y
**WHERE** x.sid = y.sid
and y.pno = 2
and x.scity = ‘Seattle’
and x.sstate = ‘WA’

Estimated (why?)

<table>
<thead>
<tr>
<th>Supplier(sid, sname, scity, sstate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply(sid, pno, quantity)</td>
</tr>
</tbody>
</table>

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

\[ \sigma_{\text{pno}=2 \land \text{scity}='Seattle' \land \text{sstate}='WA'} \]

\[ \pi_{\text{sname}} \]

T < 1

T = 10000

M = 11
Logical Query Plan 2

\[
\text{SELECT } sname \\
\text{FROM Supplier } x, \text{ Supply } y \\
\text{WHERE } x.\text{sid} = y.\text{sid} \\
\text{and } y.\text{pno} = 2 \\
\text{and } x.\text{scity} = \text{`Seattle`} \\
\text{and } x.\text{sstate} = \text{`WA`} \\
\]

\[
\text{T(Supplier)} = 1000 \\
\text{B(Supplier)} = 100 \\
\text{V(Supplier, scity)} = 20 \\
\text{V(Supplier, state)} = 10 \\
\]

\[
\text{T(Supply)} = 10000 \\
\text{B(Supply)} = 100 \\
\text{V(Supply, pno)} = 2500 \\
\]

\[
M=11
\]
Logical Query Plan 2

\[
\begin{align*}
\text{SELECT} & \quad \text{sname} \\
\text{FROM} & \quad \text{Supplier} \ x, \ \text{Supply} \ y \\
\text{WHERE} & \quad x.\text{sid} = y.\text{sid} \\
& \quad \text{and} \ y.\text{pno} = 2 \\
& \quad \text{and} \ x.\text{scity} = \text{‘Seattle’} \\
& \quad \text{and} \ x.\text{sstate} = \text{‘WA’}
\end{align*}
\]

\[
\begin{align*}
\text{M} & = 11 \\
T(\text{Supplier}) & = 1000 \\
B(\text{Supplier}) & = 100 \\
V(\text{Supplier, scity}) & = 20 \\
V(\text{Supplier, sstate}) & = 10
\end{align*}
\]
Logical Query Plan 2

\[
\pi_{\text{sname}}
\]

\[
\sigma_{\text{pno}=2}
\]

\[
\sigma_{\text{scity}=\text{Seattle} \land \text{sstate}=\text{WA}}
\]

\[
\text{SELECT sname}
\]

\[
\text{FROM Supplier x, Supply y}
\]

\[
\text{WHERE x.sid = y.sid}
\]

\[
\text{and y.pno = 2}
\]

\[
\text{and x.scity = 'Seattle'}
\]

\[
\text{and x.sstate = 'WA'}
\]

Very wrong! Why?

\[
\begin{align*}
T(\text{Supply}) &= 10000 \\
B(\text{Supply}) &= 100 \\
V(\text{Supply, pno}) &= 2500 \\
T(\text{Supplier}) &= 1000 \\
B(\text{Supplier}) &= 100 \\
V(\text{Supplier, scity}) &= 20 \\
V(\text{Supplier, state}) &= 10
\end{align*}
\]

\[M=11\]
Logical Query Plan 2

\[
\begin{align*}
\text{SELECT} & \quad \text{sname} \\
\text{FROM} & \quad \text{Supplier} \ x, \ \text{Supply} \ y \\
\text{WHERE} & \quad x.\text{sid} = y.\text{sid} \\
& \quad \text{and} \ y.\text{pno} = 2 \\
& \quad \text{and} \ x.\text{scity} = \text{Seattle} \\
& \quad \text{and} \ x.\text{sstate} = \text{WA}
\end{align*}
\]

Very wrong! Why?

\[
\begin{align*}
\text{T(Supplier)} & = 1000 \\
\text{B(Supplier)} & = 100 \\
\text{V(Supplier, scity)} & = 20 \\
\text{V(Supplier, state)} & = 10 \\
\text{M} & = 11
\end{align*}
\]
Logical Query Plan 2

\[ \pi\text{\textbf{sname}} \]

\[ \sigma\text{\textbf{pno}=2} \]

\[ \text{\textbf{Supply}} \]

\[ \sigma\text{\textbf{scity}='Seattle'} \land \text{\textbf{sstate}='WA'} \]

\[ \text{\textbf{Supplier}} \]

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

SELECT \textbf{sname}

FROM Supplier \textbf{x}, Supply \textbf{y}

WHERE \textbf{x.sid} = \textbf{y.sid}

and \textbf{y.pno} = 2

and \textbf{x.scity} = 'Seattle'

and \textbf{x.sstate} = 'WA'

T = 4

Different estimate 😞

Very wrong! Why?
Recap: External Memory Joins

- Disks are organized into *blocks*
- Typically: 1 block = 4k or 8k or 16k
- Main cost of an external memory algorithm is # blocks read or written
- Notation:
  - $B(R)$ = number of blocks of relation $R$
  - $M$ = number of blocks (=pages) that fit in main memory
Recap: External Memory Joins

\[ R \bowtie S \]

Block nested loop join:
Recap: External Memory Joins

$R \bowtie S$

Block nested loop join:

Repeat:

Read M blocks of R in memory
Scan S (one block at a time):

Join current records in S
with current records in R

Until R is exhausted
Recap: External Memory Joins

\( R \bowtie S \)

Block nested loop join:

Repeat:

Read M blocks of R in memory
Scan S (one block at a time):

Join current records in S
with current records in R

Until R is exhausted

Cost: \( B(R) + B(R)B(S)/M \)
Recap: External Memory Joins

\[ R \Join_{A=B} S \]

Index join: assume an index on S.B
Recap: External Memory Joins

\[ R \bowtie_{A=B} S \]

Index join: assume an index on S.B

Repeat:

Scan R (one block at a time)
for each tuple in current block:
look up R.A using index on S.B

Until R is exhausted
Recap: External Memory Joins

$$R \bowtie_{A=B} S$$

Index join: assume an index on S.B

Repeat:

Scan R (one block at a time) for each tuple in current block:
look up R.A using index on S.B

Until R is exhausted

Cost: \(B(R) + T(R)\times T(S)/V(S,B)\)
Recap: External Memory Joins

\[ R \bowtie_{A=B} S \]

Index join: assume an index on S.B

Repeat:

Scan R (one block at a time)
for each tuple in current block:
look up R.A using index on S.B

Until R is exhausted

Cost: \( B(R) + T(R) \times B(S) / V(S, B) \)

If S.B is a clustered index
Physical Plan 1

\[ \sigma_{pno=2 \land scity='Seattle' \land sstate='WA'} \]

\[ T = 10000 \]

Block nested loop join

\[ \pi_{sname} \]

\[ T < 1 \]

Total cost:

\[ \frac{T(\text{Supply})}{T(\text{Supplier})} \times B(\text{Supplier}) = 100 \]

\[ V(\text{Supplier}, \text{scity}) = 20 \]

\[ V(\text{Supplier}, \text{sstate}) = 10 \]

\[ T(\text{Supplier}) = 1000 \]

\[ B(\text{Supplier}) = 100 \]

\[ V(\text{Supplier}, \text{pno}) = 2500 \]

\[ M = 11 \]
Physical Plan 1

\[ \sigma_{pno=2 \land scity='Seattle' \land sstate='WA'} \]

\[ \pi_{sname} \]

\[ T < 1 \]

\[ T = 10000 \]

Block nested loop join

Supply

\[ T(\text{Supply}) = 10000 \]
\[ B(\text{Supply}) = 100 \]
\[ V(\text{Supply}, \text{pno}) = 2500 \]

Supplier

\[ T(\text{Supplier}) = 1000 \]
\[ B(\text{Supplier}) = 100 \]
\[ V(\text{Supplier}, \text{scity}) = 20 \]
\[ V(\text{Supplier}, \text{sstate}) = 10 \]

Total cost: \( 100 + 100 \times 100/10 = 1100 \)
Physical Plan 2

\[ \pi_{\text{sname}}(\sigma_{\text{sstate} = 'WA'}(\text{Supplier})) \]

\[ \sigma_{\text{pno} = 2}(\text{Supply}) \]

\[ \sigma_{\text{scity} = 'Seattle'}(\sigma_{\text{sstate} = 'WA'}(\text{Supplier})) \]

Cost of \( \text{Supply(pno)} \) = 4

Cost of \( \text{Supplier(scity)} \) = 50

Total cost: 54

\[ \text{T(Supplier)} = 1000 \]
\[ \text{B(Supplier)} = 100 \]
\[ \text{V(Supplier, scity)} = 20 \]
\[ \text{V(Supplier, state)} = 10 \]

\[ \text{M} = 11 \]

\[ \text{T(Supply)} = 10000 \]
\[ \text{B(Supply)} = 100 \]
\[ \text{V(Supply, pno)} = 2500 \]

Main memory join

Unclustered index lookup \( \text{Supplier(scity)} \)

Unclustered index lookup \( \text{Supply(pno)} \)
Physical Plan 2

\[ \sigma \text{pno}=2 \]
\[ \pi_{\text{sname}} \]
\[ \sigma \text{sstate='WA'} \]
\[ \sigma \text{scity='Seattle'} \]

Cost of Supply(pno) = 4
Cost of Supplier(scity) = 50
Total cost: 54

T(\text{Supply}) = 10000
B(\text{Supply}) = 100
V(\text{Supply, pno}) = 2500

T(\text{Supplier}) = 1000
B(\text{Supplier}) = 100
V(\text{Supplier, scity}) = 20
V(\text{Supplier, state}) = 10

\text{M}=11
Physical Plan 2

\[ \sigma_{\text{pno}=2} \]

\[ \text{Supply} \]

\[ \pi_{\text{snname}} \]

\[ \sigma_{\text{sstate}='WA'} \]

\[ \text{Supplier} \]

\[ \sigma_{\text{scity}='Seattle'} \]

Unclustered index lookup \( \text{Supply(pno)} \)

Unclustered index lookup \( \text{Supplier(scity)} \)

\[ \text{T(Supplier)} = 1000 \]
\[ \text{B(Supplier)} = 100 \]
\[ \text{V(Supplier, pno)} = 2500 \]

\[ \text{T(Supply)} = 10000 \]
\[ \text{B(Supply)} = 100 \]
\[ \text{V(Supply, pno)} = 2500 \]

\[ \text{M} = 11 \]

\[ \text{Cost of Supply(pno)} = 4 \]
\[ \text{Cost of Supplier(scity)} = 50 \]
\[ \text{Total cost: 54} \]
Physical Plan 3

\[ \pi_{\text{sname}} \]

\[ \sigma_{\text{scity}='Seattle' \land \text{sstate}='WA'} \]

\[ \text{sid} = \text{sid} \]

\[ \sigma_{\text{pno}=2} \]

Unclustered index lookup
Supply(pno)

Cost of Supply(pno) = 4
Cost of Index join = 4
Total cost: 8

Clustered Index join

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11
Physical Plan 3

\[
\pi_{\text{sname}}(\sigma_{\text{scity}=\text{Seattle} \land \text{sstate}=\text{WA}}(\text{Supplier}) \bowtie \text{Supply})
\]

Cost of \text{Supply}(\text{pno}) = 4

Cost of Index join = 4

Total cost: 8

Unclustered index lookup \text{Supply}(\text{pno})

\begin{align*}
T(\text{Supplier}) &= 1000 \\
B(\text{Supplier}) &= 100 \\
V(\text{Supplier, scity}) &= 20 \\
V(\text{Supplier, state}) &= 10
\end{align*}

\(M=11\)
Physical Plan 3

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

\[ \pi_{\text{sname}} \]
\[ \sigma_{\text{scity='Seattle' \land sstate='WA'}} \]
\[ \text{sid} = \text{sid} \]
\[ \sigma_{\text{pno}=2} \]
\[ \text{Supply} \]
\[ \text{Supplier} \]

\[ T(\text{Supply}) = 10000 \]
\[ B(\text{Supply}) = 100 \]
\[ V(\text{Supply, pno}) = 2500 \]
\[ T(\text{Supplier}) = 1000 \]
\[ B(\text{Supplier}) = 100 \]
\[ V(\text{Supplier, scity}) = 20 \]
\[ V(\text{Supplier, state}) = 10 \]

Cost of \text{Supply}(\text{pno}) = 4
Cost of Index join = 4
Total cost: 8
Discussion

• We considered only IO cost; real systems need to consider IO+CPU

• Each system has its own hacks

• We assumed that all index pages were in memory: sometimes we need to add the cost of fetching index pages from disk
Histograms

• T(R), V(R,A) too coarse
• Histogram: separate stats per bucket

• In each bucket store:
  – T(bucket)
  – V(bucket,A) – optional
Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Employee, age) = 50

\( \sigma_{\text{age}=48}(\text{Employee}) = ? \)
**Histognams**

Employee(ssn, name, age)

$T(\text{Employee}) = 25000, \ V(\text{Employee, age}) = 50$

$\sigma_{\text{age}=48}(\text{Employee}) = ?$

Estimate: $T(\text{Employee}) / V(\text{Employee, age}) = 500$
Histograms

Employee(ssn, name, age)

$T(\text{Employee}) = 25000$, $V(\text{Employee, age}) = 50$

$\sigma_{\text{age}=48}(\text{Employee}) = ?$

Estimate: $\frac{T(\text{Employee})}{V(\text{Employee, age})} = 500$

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>T =</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
</tbody>
</table>
Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Employee, age) = 50

$\sigma_{age=48}(Employee) = ?$

Estimate: $T(Employee) / V(Employee, age) = 500$

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
</tbody>
</table>

Assume V = 10
Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Employee, age) = 50

σ_{age=48}(Employee) = ?

Estimate: \( \frac{T(Employee)}{V(Employee, age)} = 500 \)

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>T =</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
</tbody>
</table>

Estimate: \( \frac{12000}{10} = 1200 \)

Assume \( V = 10 \)
Histograms

Employee(ssn, name, age)

$T(\text{Employee}) = 25000$, $V(\text{Employee, age}) = 50$

$\sigma_{\text{age}=48}(\text{Employee}) = ?$

Estimate: $T(\text{Employee}) / V(\text{Employee, age}) = 500$

<table>
<thead>
<tr>
<th>Age</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T =$</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
<tr>
<td>$V =$</td>
<td>3</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Estimate: $12000/10 = 1200$
Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Employee, age) = 50

σ_{age=48}(Employee) = ?

Estimate: T(Employee) / V(Employee, age) = 500

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>T =</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
<tr>
<td>V =</td>
<td>3</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Estimate: 12000/10 = 1200 12000/6 = 2000
Types of Histograms

- Eq-Width
- Eq-Depth
- Compressed: store outliers separately
- V-Optimal histograms
Employee(ssn, name, age)

Histograms

**Eq-width:**

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
</tbody>
</table>

**Eq-depth:**

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..32</th>
<th>33..41</th>
<th>42-46</th>
<th>47-52</th>
<th>53-58</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>1800</td>
<td>2000</td>
<td>2100</td>
<td>2200</td>
<td>1900</td>
<td>1800</td>
</tr>
</tbody>
</table>

**Compressed:** store separately highly frequent values: (48,1900)
V-Optimal Histograms

- Error:
  \[ \sum_{v \in \text{Domain}(A)} \left( |\sigma_{A=v}(R)| - \text{est}_{Hist}(\sigma_{A=v}(R)) \right)^2 \]

- Bucket boundaries = \[\text{argmin}_{Hist}(\text{Error})\]
- Dynamic programming
- Modern databases systems use V-optimal histograms or some variations
Discussion

• Cardinality estimation = still unsolved

• Histograms:
  – Small number of buckets (why?)
  – Updated only periodically (why?)
  – No 2d histograms (except db2) why?

• Samples:
  – Fail for low selectivity estimates, joins

• Cross-join correlation – still unsolved
Yet Another Difficulties

SQL Queries issued from applications:

• Query is optimized once: prepare

• Then, executed repeatedly

Query constants are unknow until execution: optimized plan is suboptimal
select
    o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)
from
    (select YEAR(o_orderdate) as o_year,
        l_extendedprice * (1 - l_discount) as volume,
        n2.n_name as nation
     from part, supplier,
          lineitem, orders,
          customer, nation n1, nation n2, region
     where p_partkey = l_partkey and s_suppkey = l_suppkey
       and l_orderkey = o_orderkey and o_custkey = c_custkey
       and c_nationkey = n1.n_nationkey
       and n1.n_regionkey = r_regionkey
       and r_name = 'AMERICA'
       and s_nationkey = n2.n_nationkey
       and o_orderdate between '1995-01-01'
       and '1996-12-31'
       and p_type = 'ECONOMY ANODIZED STEEL'
       and s_acctbal ≤ C1 and l_extendedprice ≤ C2 ) as all_nations
group by o_year order by o_year
select
    o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)
from
(select YEAR(o_orderdate) as o_year,
    l_extendedprice * (1 - l_discount) as volume,
    n2.n_name as nation
from part, supplier, lineitem, orders,
    customer, nation n1, nation n2, region
where p_partkey = l_partkey and s_suppkey = l_suppkey
    and l_orderkey = o_orderkey and o_custkey = c_custkey
    and c_nationkey = n1.n_nationkey
    and n1.n_regionkey = r_regionkey
    and r_name = 'AMERICA'
    and s_nationkey = n2.n_nationkey
    and o_orderdate between '1995-01-01'
    and '1996-12-31'
    and p_type = 'ECONOMY ANODIZED STEEL'
    and s_acctbal <= C1 and l_extendedprice <= C2 ) as all_nations
group by o_year order by o_year
Different optimal plans for different C1, C2
Discussion

- Cardinality estimation: the weak spot of all query optimizers
- Other approaches:
  - Offline sampling: may lead to 0 estimates
  - Online sampling: sloooooooow
  - ML: large training data, model
  - Theoretical upper bounds: too coarse