Announcements

• HW2 has been posted on sakai
  – Due on Oct 18, 11 pm
  – Start early!
  – Your AWS account should have $100 credit
  – ALWAYS Remember to stop instance!

• Midterm next week
  – Wednesday, Oct 11, in class
  – Closed book, closed electronic devices
  – Everything until and including Lecture 12
Reading Material

• [RG]
  – Query optimization: Chapter 15 (overview only)

• [GUW]
  – Chapter 16.2-16.7

• Original paper by Selinger et al.:
  – P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. *Access Path Selection in a Relational Database Management System*
    Proceedings of ACM SIGMOD, 1979. Pages 22-34
  – No need to understand the whole paper, but take a look at the example (link on the course webpage)

Acknowledgement:
• The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.
• Some of the following slides have been created by adapting slides by Profs. Shivnath Babu and Magda Balazinska
Query Evaluation and Operator Algorithm

Continued from Lecture 10
Hash-Join

- Partition both relations using hash function $h$
- $R$ tuples in partition $i$ will only match $S$ tuples in partition $i$

- Read in a partition of $R$, hash it using $h_2$ ($\neq h$).
- Scan matching partition of $S$, search for matches.
Cost of Hash-Join

• In partitioning phase
  – read+write both relns; \(2(M+N)\)
  – In matching phase, read both relns; \(M+N\) I/Os
  – remember – we are not counting final write

• In our running example, this is a total of 4500 I/Os
  – 3 * (1000 + 500)
  – Compare with the previous joins

\[
\begin{align*}
N &= 500 \text{ pages in } S \\
p_s &= 80 \text{ tuples per page} \\
M &= 1000 \text{ pages in } R \\
p_R &= 100 \text{ tuples per page}
\end{align*}
\]
Sort-Merge Join vs. Hash Join

• Both can have a cost of $3(M+N)$ I/Os
  – if sort-merge gets enough buffer (see 14.4.2)
• Hash join holds smaller relation in buffer—better if limited buffer
• Hash Join shown to be highly parallelizable
• Sort-Merge less sensitive to data skew
  – also result is sorted
General Join Conditions

• Equalities over several attributes
  – e.g., \( \text{R.sid} = \text{S.sid} \text{ AND R.rname} = \text{S.sname} \)
  – For Index Nested Loop, build index on \(<\text{sid, sname}>\) (if S is inner); or use existing indexes on \text{sid} or \text{sname}
  – For Sort-Merge and Hash Join, sort/partition on combination of the two join columns.

• Inequality conditions
  – e.g., \( \text{R.rname} < \text{S.sname} \)
  – For Index NL, need (clustered) B+ tree index.
  – Hash Join, Sort Merge Join not applicable
Review: Join Algorithms

- Nested loop join:
  - for all tuples in R.. for all tuples in S....
  - variations: block-nested, index-nested
- Sort-merge join
  - like external merge sort
- Hash join

- Make sure you understand how the I/O varies
- No one join algorithm is uniformly superior to others
  - depends on relation size, buffer pool size, access methods, skew
Algorithms for Selection

- No index, unsorted data
  - Scan entire relation
  - May be expensive if not many `Joe's`
- No index, sorted data (on `rname`)
  - locate the first tuple, scan all matching tuples
  - first binary search, then scan depends on matches
- B+-tree index, Hash index
  - Discussed earlier
  - Cost of accessing data entries + matching data records
  - Depends on clustered/unclustered
- More complex condition like \( \text{day<8/9/94 AND bid}=5 \text{ AND sid}=3 \)
  - Either use one index, then filter
  - Or use two indexes, then take intersection, then apply third condition
  - etc.

```
SELECT *
FROM Reserves R
WHERE R.rname = 'Joe'
```
Algorithms for Projection

- Two parts
  - Remove fields: easy
  - Remove duplicates (if distinct is specified): expensive
- Sorting-based
  - Sort, then scan adjacent tuples to remove duplicates
  - Can eliminate unwanted attributes in the first pass of merge sort
- Hash-based
  - Exactly like hash join
  - Partition only one relation in the first pass
  - Remove duplicates in the second pass
- Sort vs Hash
  - Sorting handles skew better, returns results sorted
  - Hash table may not fit in memory – sorting is more standard
- Index-only scan may work too
  - If all required attributes are part of index
Algorithms for Set Operations

- Intersection, cross product are special cases of joins
- Union, Except
  - Sort-based
  - Hash-based
  - Very similar to joins and projection
Algorithms for Aggregate Operations

- **SUM, AVG, MIN etc.**
  - again similar to previous approaches

- **Without grouping:**
  - In general, requires scanning the relation.
  - Given index whose search key includes all attributes in the SELECT or WHERE clauses, can do index-only scan

- **With grouping:**
  - Sort on group-by attributes
  - or, hash on group-by attributes
  - can combine sort/hash and aggregate
  - can do index-only scan here as well
Query Optimization
Query Blocks: Units of Optimization

- Query Block
  - No nesting
  - One SELECT, one FROM
  - At most one WHERE, GROUP BY, HAVING

- SQL query
- => parsed into a collection of query blocks
- => the blocks are optimized one block at a time

- Express single-block it as a relational algebra (RA) expression

```
SELECT S.sname
FROM Sailors S
WHERE S.age IN
  (SELECT MAX (S2.age)
   FROM Sailors S2
   GROUP BY S2.rating)
```
Cost Estimation

• For each plan considered, must estimate cost:

  • **Must estimate cost of each operation in plan tree.**
    – Depends on input cardinalities
    – We’ve discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)

  • **Must also estimate size of result for each operation in tree**
    – gives input cardinality of next operators

• **Also consider**
  – whether the output is sorted
  – intermediate results written to disk
Relational Algebra Equivalences

- Allow us to choose different join orders and to `push’ selections and projections ahead of joins.

- **Selections:**
  \[
  \sigma_{c_1 \wedge \ldots \wedge c_n}(R) \equiv \sigma_{c_1} \left( \ldots \sigma_{c_n}(R) \right) \quad \text{(Cascade)}
  \]
  \[
  \sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R)) \quad \text{(Commutative)}
  \]

- **Projections:**
  \[
  \pi_{a_1}(R) \equiv \pi_{a_1} \left( \ldots (\pi_{a_n}(R)) \right) \quad \text{(Cascade)}
  \]

- **Joins:**
  \[
  R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T \quad \text{(Associative)}
  \]
  \[
  (R \bowtie S) \equiv (S \bowtie R) \quad \text{(Commutative)}
  \]

There are many more intuitive equivalences, see 15.3.4 for details.
Notation

- $T(R)$ : Number of tuples in $R$
- $B(R)$ : Number of blocks (pages) in $R$
- $V(R, A)$ : Number of distinct values of attribute $A$ in $R$
Query Optimization Problem

Pick the best plan from the space of physical plans
Cost-based Query Optimization

Pick the plan with least cost

Challenge:

• Do not want to execute more than one plans

• Need to estimate the cost without executing the plan

“heuristic-based” optimizer (e.g. push selections down) have limited power and not used much
Cost-based Query Optimization

Pick the plan with least cost

Tasks:
1. Estimate the cost of individual operators
   - done in Lecture 9-11
2. Estimate the size of output of individual operators
   - today
3. Combine costs of different operators in a plan
   - today
4. Efficiently search the space of plans
   - today
Task 1 and 2
Estimating cost and size of different operators

• Size = #tuples, NOT #pages
• Cost = #page I/O
  • but, need to consider whether the intermediate relation fits in memory, is written back to/read from disk (or on-the-fly goes to the next operator), etc.
Desired Properties of Estimating Sizes of Intermediate Relations

Ideally,

• should give accurate estimates (as much as possible)
• should be easy to compute
• should be logically consistent
  -- size estimate should be independent of how the relation is computed (e.g. which join algorithm/join order is used)

• But, no “universally agreed upon” ways to meet these goals
Cost of Table Scan

Cost: $B(R)$

Size: $T(R)$

$T(R)$: Number of tuples in $R$

$B(R)$: Number of blocks in $R$
Cost of Index Scan

Cost: \( B(R) \) – if clustered
\( T(R) \) – if unclustered

Size: \( T(R) \)

Note:
1. size is independent of the implementation of the scan/index
2. Index scan is bad if unclustered
Cost of Index Scan with Selection

\[ X = \sigma_{R.A > 50} R \]

Cost: \( B(R) \times f \) – if clustered
\( T(R) \times f \) – if unclustered

Size: \( T(R) \times f \)

Reduction factor
\[ f = \frac{(\text{Max}(R.A) - 50)}{(\text{Max}(R.A) - \text{Min}(R.A))} \]
assumes uniform distribution

\( T(R) \) : Number of tuples in \( R \)
\( B(R) \) : Number of blocks in \( R \)
Cost of Index Scan with Selection (and multiple conditions)

\[ X = \sigma_{R.A > 50 \land R.B = C} R \]

assume index on \((A, B)\)

What is \(f_1\) if the first condition is \(100 > R.1 > 50\)?

Cost: \(B(R) \times f \) – if clustered

Size: \(T(R) \times f\)

Reduction factors

range selection

\[ f_1 = \frac{\text{Max}(R.A) - 50}{\text{Max}(R.A) - \text{Min}(R.A)} \]

value selection

\[ f_2 = \frac{1}{V(R, B)} \]

\[ f = f_1 \times f_2 \] (assumes independence and uniform distribution)

\(T(R)\) : Number of tuples in \(R\)

\(B(R)\) : Number of blocks in \(R\)

\(V(R, A)\) : Number of distinct values of attribute \(A\) in \(R\)
Cost of Projection

\[ X = \pi_A R \]

Cost: depends on the method of scanning \( R \)
- \( B(R) \) for table scan or clustered index scan

Size: \( T(R) \)
- But tuples are smaller
- If you have more information on the size of the smaller tuples, can estimate \#I/O better
Size of Join

Quite tricky
- If disjoint A and B values
  - then 0
- If A is key of R and B is foreign key of S
  - then T(S)
- If all tuples have the same value of R.A = S.B = x
  - then T(R) * T(S)

\[ R.A = S.B \]

T (R) : Number of tuples in R
B (R) : Number of blocks in R
V(R, A) : Number of distinct values of attribute A in R
Size of Join

Two standard assumptions

1. Containment of value sets:
   - if \( V(R, A) \leq V(S, B) \), then all \( A \)-values of \( R \) are included in \( B \)-values of \( S \)
   - e.g. satisfied when \( A \) is foreign key, \( B \) is key

2. Preservation of value sets:
   - For all “non-joining” attributes, the set of distinct values is preserved in join
   - \( V(R \bowtie S, C) = V(R, C) \), where \( C \neq A \) is an attribute in \( R \)
   - \( V(R \bowtie S, D) = V(S, D) \), where \( D \neq B \) is an attribute in \( S \)
   - Helps estimate distinct set size in \( R \bowtie S \bowtie T \)
Size of Join

Reduction factor
\[ f = \frac{1}{\text{max}(V(R, A), V(S, B))} \]

Size = \( T(R) \times T(S) \times f \)

T (R) : Number of tuples in R
B (R) : Number of blocks in R
V(R, A) : Number of distinct values of attribute A in R
Size of Join

Reduction factor

\[ f = \frac{1}{\max(V(R, A), V(S, B))} \]

Size

\[ \text{Size} = T(R) \times T(S) \times f \]

Why max?

- Suppose \( V(R, A) \leq V(S, B) \)
- The probability of a \( A \)-value joining with a \( B \)-value is \( 1/V(S.B) = \text{reduction factor} \)
- Under the two assumptions stated earlier + uniformity

T (R) : Number of tuples in R
B (R) : Number of blocks in R
V(R, A) : Number of distinct values of attribute A in R

Assumes index on both A and B
if one index: \( 1/V(\ldots, \ldots) \)
if no index: say \( 1/10 \)
Task 3: Combine cost of different operators in a plan

With Examples
“Given” the physical plan

• Size = #tuples, NOT #pages
• Cost = #page I/O
  • but, need to consider whether the intermediate relation fits in memory, is written back to disk (or on-the-fly goes to the next operator) etc.
Example Query

Student (sid, name, age, address)
Book(bid, title, author)
Checkout(sid, bid, date)

Query:
SELECT S.name
FROM Student S, Book B, Checkout C
WHERE S.sid = C.sid
AND B.bid = C.bid
AND B.author = 'Olden Fames'
AND S.age > 12
AND S.age < 20
S(sid, name, age, addr)  
B(bid, title, author)  
C(sid, bid, date)  

Assumptions

• Student: S, Book: B, Checkout: C

• Sid, bid foreign key in C referencing S and B resp.
• There are 10,000 Student records stored on 1,000 pages.
• There are 50,000 Book records stored on 5,000 pages.
• There are 300,000 Checkout records stored on 15,000 pages.
• There are 500 different authors.
• Student ages range from 7 to 24.

Warning: a few dense slides next 😊
Physical Query Plan – 1

Q. Compute
1. the cost and cardinality in steps (a) to (d)
2. the total cost

Assumptions (given):
- Data is not sorted on any attributes
- For both in (a) and (b), outer relations fit in memory

(Tuple-based nested loop
B inner)

(On the fly) (d) \( \Pi_{name} \)

(On the fly) (c) \( \sigma_{12<\text{age}<20 \land \text{author} = 'Olden Fames'} \)

(Please provide specific details for the query plan and assumptions based on the image content.)
\[ T(S) = 10,000 \quad B(S) = 1,000 \quad V(B, \text{author}) = 500 \]
\[ 7 \leq \text{age} \leq 24 \]

\[ \text{Cost} = B(S) + B(S) \times B(C) = 1000 + 1000 \times 15000 = 15,001,000 \]

\[ T(C) = 300,000 \]

**Cardinality**
- foreign key join, output pipelined to next join
- Can apply the formula as well

\[ T(S) \times T(C) / \max (V(S, \text{sid}), V(C, \text{sid})) = T(C) \]

since \( V(S, \text{sid}) \geq V(C, \text{sid}) \) and \( T(S) = V(S, \text{sid}) \)
(TUPLE-BASED NESTED LOOP, B INNER)

(ON THE FLY) (d) $\Pi_{\text{name}}$

(ON THE FLY) (c) $\sigma_{12 < \text{age} \leq 20 \land \text{author} = \text{'Olden Fames'}}$

Cost = $T(S \bowtie C) \times B(B) = 300,000 \times 5,000 = 15 \times 10^8$

Cardinality = $T(S \bowtie C) = 300,000$

- foreign key join
- don’t need scanning for outer relation
  - outer relation fits in memory
\( S(\text{sid}, \text{name}, \text{age}, \text{addr}) \quad T(S) = 10,000 \quad B(S) = 1,000 \quad V(B, \text{author}) = 500 \)

\( B(\text{bid}, \text{title}, \text{author}) \quad T(B) = 50,000 \quad B(B) = 5,000 \quad 7 \leq \text{age} \leq 24 \)

\( C(\text{sid}, \text{bid}, \text{date}) \quad T(C) = 300,000 \quad B(C) = 15,000 \)

\[(c, d)\]

\[(\text{On the fly}) \quad (d) \ \Pi_{\text{name}}\]

\[(\text{On the fly}) \quad (c) \ \sigma_{12<\text{age}<20} \ \land \ \text{author} = \text{'Olden Fames'}\]

\( (\text{Tuple-based nested loop}) \quad (\text{B inner}) \)

\( (\text{Page-oriented nested loop,} \quad \text{S outer, C inner}) \)

\( \text{Student S} \quad (\text{File scan}) \)

\( \text{Checkout C} \quad (\text{File scan}) \)

\( \text{Book B} \quad (\text{File scan}) \)

\( \text{Cost = 0 (on the fly)} \)

\( \text{Cardinality = 300,000} \ * \ 1/500 \ * \ 7/18 \)

\( = 234 \ (\text{approx}) \)

(assuming uniformity and independence)
### Query Plan

**Data Sources**
- **S** (Student) with attributes: `sid`, `name`, `age`, `addr`
- **B** (Book) with attributes: `bid`, `title`, `author`
- **C** (Checkout) with attributes: `sid`, `bid`, `date`

**Table Sizes**
- `T(S) = 10,000`
- `T(B) = 50,000`
- `T(C) = 300,000`

**Index Sizes**
- `B(S) = 1,000`
- `B(B) = 5,000`
- `B(C) = 15,000`
- `V(B, author) = 500`

**Conditions**
- `7 <= age <= 24`

**Query**

1. **Tuple-based nested loop**
   - **S** outer, **C** inner
2. **Page-oriented nested loop**
   - **S** outer, **B** inner
   - **On the fly**
     - Join on `sid` (File scan)
     - Join on `bid` (File scan)
     - `σ_{12<age<20 \land \text{author} = 'Olden Fames'}` (On the fly)
     - `Π_{name}` (On the fly)

**Total Cost**

```
Total cost = 1,515,001,000
```

**Final Cardinality**

```
Final cardinality = 234 (approx)
```
Physical Query Plan – 2

Q. Compute
1. the cost and cardinality in steps (a) to (g)
2. the total cost

Assumptions (given):
• Unclustered B+tree index on B.author
• Clustered B+tree index on C.bid
• All index pages are in memory
• Unlimited memory
\begin{itemize}
  \item S(sid, name, age, addr)
  \item B(bid, title, author): Un. B+ on author
  \item C(sid, bid, date): Cl. B+ on bid
\end{itemize}

\begin{align*}
T(S) &= 10,000 & B(S) &= 1,000 & V(B, \text{author}) &= 500 \\
T(B) &= 50,000 & B(B) &= 5,000 & 7 \leq \text{age} \leq 24 \\
T(C) &= 300,000 & B(C) &= 15,000
\end{align*}

\begin{itemize}
  \item \(\sigma_{12<\text{age}<20}\) \(\Pi_{\text{name}}\) \(\Pi_{\text{sid}}\) \(\Pi_{\text{bid}}\)
  \item \(\sigma_{\text{author} = 'Olden Fames'}\) \(\Pi_{\text{bid}}\)
  \item \(\Pi_{\text{sid}}\) \(\Pi_{\text{name}}\)
  \item \(\Pi_{\text{bid}}\)
  \item \(\Pi_{\text{sid}}\)
\end{itemize}

\textbf{Cost =} \\
\(\frac{T(B)}{V(B, \text{author})} = \frac{50,000}{500} = 100\) (unclustered)

\textbf{Cardinality =} \\
100
$S$($sid$, $name$, $age$, $addr$)
$B$($bid$, $title$, $author$): Un. B+ on author
$C$($sid$, $bid$, $date$): Cl. B+ on bid

\begin{align*}
T(S) &= 10,000 & B(S) &= 1,000 & V(B, author) &= 500 \\
T(B) &= 50,000 & B(B) &= 5,000 & 7 \leq age \leq 24 \\
T(C) &= 300,000 & B(C) &= 15,000 & \end{align*}

\begin{align*}
V(B, author) &= 500 \\
7 \leq age \leq 24 & \end{align*}

\begin{align*}
S &\quad \text{(Indexed-nested loop, B outer, C inner)} \\
B &\quad \text{(Index scan)} \\
C &\quad \text{(File scan)} \\
\end{align*}

Cost = 0 (on the fly)
Cardinality = 100
\[ S(\text{sid}, \text{name}, \text{age}, \text{addr}) \]
\[ B(\text{bid}, \text{title}, \text{author}) : \text{Un. B+ on author} \]
\[ C(\text{sid}, \text{bid}, \text{date}) : \text{Cl. B+ on bid} \]

\[
\begin{align*}
T(S) &= 10,000 & B(S) &= 1,000 & V(B, \text{author}) &= 500 \\
T(B) &= 50,000 & B(B) &= 5,000 \\
T(C) &= 300,000 & B(C) &= 15,000
\end{align*}
\]

\[
7 \leq \text{age} \leq 24
\]

- one index lookup per outer B tuple
- 1 book has \( \frac{T(C)}{T(B)} = 6 \) checkouts (uniformity)
- \# C tuples per page = \( \frac{T(C)}{B(C)} = 20 \)
- 6 tuples fit in at most 2 consecutive pages (clustered)
  could assume 1 page as well

Cost \( \leq \)
\[
100 \times 2 = 200
\]

Cardinality =
\[
100 \times 6 = 600
\]

\[
= 100 \times \frac{T(C)}{\text{MAX}(100, V(C, bid))}
\]
assuming
\[
V(C, \text{bid}) = V(B, \text{bid}) = T(B) = 50,000
\]
\begin{itemize}
\item \textbf{Student S (sid, name, age, addr)}
\item \textbf{Checkout C (sid, bid, date)}: Cl. B+ on bid
\item \textbf{Book B (bid, title, author)}: Un. B+ on author
\end{itemize}

\begin{align*}
T(S) &= 10,000 & B(S) &= 1,000 & V(B, \text{author}) &= 500 \\
T(B) &= 50,000 & B(B) &= 5,000 & 7 \leq \text{age} \leq 24 \\
T(C) &= 300,000 & B(C) &= 15,000
\end{align*}

\begin{itemize}
\item (On the fly) \textbf{(d) } \Pi_{\text{sid}} \text{ (On the fly)}
\item (Indexed-nested loop, B outer, C inner)
\item (Index scan)
\item (File scan)
\item \textbf{Student S (File scan)}
\item \textbf{Checkout C (Index scan)}
\item \textbf{Book B (Index scan)}
\item \textbf{Cost = 0 (on the fly)}
\item \textbf{Cardinality = 600}
\end{itemize}
\( S(\text{sid}, \text{name}, \text{age}, \text{addr}) \)  
\( T(S) = 10,000 \)

\( B(\text{bid}, \text{title}, \text{author}) : \)  \( \text{Un. B+ on author} \)  
\( T(B) = 50,000 \)

\( C(\text{sid}, \text{bid}, \text{date}) : \)  \( \text{Cl. B+ on bid} \)  
\( T(C) = 300,000 \)

\( B(S) = 1,000 \)

\( B(B) = 5,000 \)

\( B(C) = 15,000 \)

\( V(\text{B, author}) = 500 \)

\( 7 \leq \text{age} \leq 24 \)

---

### (a) \( \sigma \text{author} = \text{‘Olden Fames’} \)

### Book B

### (Index scan)

---

### (b) \( \Pi \text{bid} \)

### Checkout C

### (Index scan)

---

### (c) \( \text{Student S} \)

### (File scan)

---

### (d) \( \Pi \text{sid} \)

### (On the fly)

### (Indexed-nested loop, B outer, C inner)

---

### (e) \( \sigma \text{12<age<20} \)

### (On the fly)

### (Block nested loop, S inner)

---

### (f) \( \Pi \text{name} \)

### (On the fly)

---

### (g) \( \Pi \text{name} \)

### Outer relation is already in (unlimited) memory need to scan S relation

**Cost =**  
\( B(S) = 1000 \)

**Cardinality =**  
600  
(one student per checkout)
\[
\begin{align*}
S(\text{sid, name, age, addr}) & \quad T(S)=10,000 \quad B(S)=1,000 \quad V(\text{B, author}) = 500 \\
B(\text{bid, title, author}): \text{Un. B+ on author} & \quad T(B)=50,000 \quad B(B)=5,000 \\
C(\text{sid, bid, date}): \text{Cl. B+ on bid} & \quad T(C)=300,000 \quad B(C)=15,000 \\
\end{align*}
\]

\[7 \leq \text{age} \leq 24\]
\( S(\text{sid}, \text{name}, \text{age}, \text{addr}) \)
\( B(\text{bid}, \text{title}, \text{author}) \): Un. B+ on author
\( C(\text{sid}, \text{bid}, \text{date}) \): Cl. B+ on bid

\[ T(S) = 10,000 \quad B(S) = 1,000 \quad V(B, \text{author}) = 500 \]
\[ 7 \leq \text{age} \leq 24 \]
\[ T(B) = 50,000 \quad B(B) = 5,000 \]
\[ T(C) = 300,000 \quad B(C) = 15,000 \]

T(S)=10,000 T(B)=50,000 T(C)=300,000
B(S)=1,000 B(B)=5,000 B(C)=15,000

V(B,author) = 500
7 <= age <= 24

(On the fly) \(\Pi_{name}\)
(On the fly) \(\sigma_{12<age<20}\)
(Block nested loop S inner)
(On the fly) \(\Pi_{sid}\)
(Indexed-nested loop, B outer, C inner)
(On the fly) \(\Pi_{bid}\)
(a) \(\sigma_{author = 'Olden Fames'}\)
(b) \(\Pi_{bid}\)
(e)
(c)
Student S
(File scan)

Checkout C
(Index scan)

Book B
(Index scan)

Total cost = 1300
(compare with 1,515,001,000 for plan 1!)

Final cardinality = 234 (approx)
(same as plan 1!)

End of Lecture 11
Task 4:
Efficiently searching the plan space

Use dynamic-programming based
Selinger’s algorithm
Heuristics for pruning plan space

• Apply predicates as early as possible
• Avoid plans with cross products
• Only left-deep join trees
Join Trees

Query:  \( R_1 \Join R_2 \Join R_3 \Join R_4 \Join R_5 \)

- Several possible structures of the trees
- Each tree can have \( n! \) permutations of relations on leaves

(physical plan space)
- Different implementation and scanning of intermediate operators for each logical plan
Selinger Algorithm

• Dynamic Programming based

• Dynamic Programming:
  – General algorithmic paradigm
  – Exploits “principle of optimality”
    • Useful reading: Chapter 16, Introduction to Algorithms, Cormen, Leiserson, Rivest

• Considers the search space of left-deep join trees
  – reduces search space (only one structure)
  – but still $n!$ permutations
  – interacts well with join algos (esp. NLJ)
  – e.g. might not need to write tuples to disk if enough memory
Principle of Optimality

Optimal for “whole” made up from optimal for “parts”
Principle of Optimality

Query: \( R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5 \)

Suppose, this is an Optimal Plan for joining \( R1 \ldots R5 \):
Principle of Optimality

Query: \( R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5 \)

Then, what can you say about this sub-plan?

This has to be the optimal plan for joining \( R3, R2, R4, R1 \)

Suppose, this is an Optimal Plan for joining \( R1 \ldots R5 \):
Principle of Optimality

Query:  \( R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5 \)

Then, what can you say about this sub-plan?

We are using the associativity and commutativity of joins:

\[
\begin{align*}
(R \bowtie S) \bowtie T &= R \bowtie (S \bowtie T) \\
R \bowtie S &= S \bowtie R
\end{align*}
\]

Suppose, this is an Optimal Plan for joining R1…R5:

This has to be the optimal plan for joining \( R3, R2, R4 \)
Exploiting Principle of Optimality

Query: \( R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n \)

Both are giving the same result
\( R_2 \bowtie R_3 \bowtie R_1 = R_3 \bowtie R_1 \bowtie R_2 \)

Optimal for joining \( R_1, R_2, R_3 \)

Sub-Optimal for joining \( R_1, R_2, R_3 \)
Exploiting Principle of Optimality

Suppose you chose the sub-optimal one

Leads to sub-Optimal for joining R1, …, Rn

A sub-optimal sub-plan cannot lead to an optimal plan
Notation

\[ \text{OPT} \left( \{ R1, R2, R3 \} \right) : \]

Cost of optimal plan to join \( R1, R2, R3 \)

\[ \text{T} \left( \{ R1, R2, R3 \} \right) : \]

Number of tuples in \( R1 \bowtie R2 \bowtie R3 \)
Simple Cost Model

\[
\text{Cost } (R \bowtie S) = T(R) + T(S)
\]

All other operators have 0 cost

Note: The simple cost model used for illustration only, it is not used in practice
Cost Model Example

Total Cost: $T(R) + T(S) + T(T) + T(X)$
Selinger Algorithm:

\[
\text{OPT ( \{ R1, R2, R3 \} ):}
\]

\[
\begin{align*}
\text{Min} & \quad \text{OPT ( \{ R1, R2 \} )} + T ( \{ R1, R2 \} ) + T (R3) \\
& \quad \text{OPT ( \{ R2, R3 \} )} + T ( \{ R2, R3 \} ) + T (R1) \\
& \quad \text{OPT ( \{ R1, R3 \} )} + T ( \{ R1, R3 \} ) + T (R2)
\end{align*}
\]

Note: Valid only for the simple cost model
Selinger Algorithm:

Query: $R_1 \Join R_2 \Join R_3 \Join R_4$

Progress of algorithm
**Selinger Algorithm:**

Query: \[ R1 \bowtie R2 \bowtie R3 \bowtie R4 \]

Progress of algorithm:

```
{ R1, R2, R3, R4 }
{ R1, R2, R3 }    { R1, R2, R4 }    { R1, R3, R4 }    { R2, R3, R4 }
{ R1, R2 }       { R1, R3 }       { R1, R4 }       { R2, R3 }       { R2, R4 }    { R3, R4 }
{ R1 }           { R2 }           { R3 }           { R4 }           { R2 }       { R3 }       { R4 }
```

Duke CS, Fall 2017
Selinger Algorithm:

Query:  \( R1 \bowtie R2 \bowtie R3 \bowtie R4 \)

Progress of algorithm

\[
\begin{align*}
\{ R1, R2, R3, R4 \} & \\
\{ R1, R2, R3 \} & \rightarrow \{ R1, R2, R4 \} & \rightarrow \{ R1, R3, R4 \} & \rightarrow \{ R2, R3, R4 \} \\
\{ R1, R2 \} & \rightarrow \{ R1, R3 \} & \rightarrow \{ R1, R4 \} & \rightarrow \{ R2, R3 \} & \rightarrow \{ R2, R4 \} & \rightarrow \{ R3, R4 \} \\
\{ R1 \} & \rightarrow \{ R2 \} & \rightarrow \{ R3 \} & \rightarrow \{ R4 \}
\end{align*}
\]

e.g. All possible permutations of R1, R3, R4 have been considered after OPT({\( R1, R3, R4 \)}) has been computed
Query: \( R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \)

Q. How to optimally compute join of \{R1, R2, R3, R4\}?  

Ans: First optimally join \{R1, R3, R4\} then join with R2 as inner.

Selinger Algorithm:
Selinger Algorithm:

Query: \( R1 \bowtie R2 \bowtie R3 \bowtie R4 \)

Q. How to optimally compute join of \{R1, R3, R4\}?  
Ans: First optimally join \{R1, R3\}, then join with R4 as inner.
Selinger Algorithm:

Query: \( R1 \bowtie R2 \bowtie R3 \bowtie R4 \)

Q. How to optimally compute join of \{R1, R3\}?  
Ans: First optimally join \{R3\}, then join with R1 as inner.
Selinger Algorithm:

Query: \( R1 \bowtie R2 \bowtie R3 \bowtie R4 \)

Q. How to optimally compute join of \( \{R3\} \)?

Ans: Single relation – so optimally scan \( R3 \).
Selinger Algorithm:

Query: \( R1 \bowtie R2 \bowtie R3 \bowtie R4 \)

Final optimal plan:

NOTE: There is a one-one correspondence between the permutation (R3, R1, R4, R2) and the above left deep plan
Selinger Algorithm:

Query: \( R1 \bowtie R2 \bowtie R3 \bowtie R4 \)

NOTE: (*VERY IMPORTANT*)
- This is *NOT* done by top-down recursive calls.
- This is done BOTTOM-UP computing the optimal cost of *all* nodes in this lattice only once (dynamic programming).

{ \( R1, R2, R3, R4 \) }

{ \( R1, R2, R3 \) } { \( R1, R2, R4 \) } { \( R1, R3, R4 \) } { \( R2, R3, R4 \) }

{ \( R1, R2 \) } { \( R1, R3 \) } { \( R1, R4 \) } { \( R2, R3 \) } { \( R2, R4 \) } { \( R3, R4 \) }

{ \( R1 \) } { \( R2 \) } { \( R3 \) } { \( R4 \) }

Progress of algorithm
More on Query Optimizations

• See the survey (on course website): “An Overview of Query Optimization in Relational Systems” by Surajit Chaudhuri

• Covers other aspects like
  – Pushing group by before joins
  – Merging views and nested queries
  – “Semi-join”-like techniques for multi-block queries
    • covered later in distributed databases
  – Statistics and optimizations
  – Starbust and Volcano/Cascade architecture, etc