Lectures 21-23: Query Optimization

Introduction to Database Systems
CSE 444, Winter 2011

http://www.cs.washington.edu/education/courses/cse444/11wi/
### Where we are / and where we go

| Date   | Transactions: Concurrency Control lecture 14-15 Midterm review on the board | Midterm | Data Storage and Indexing lecture 16
|--------|-----------------------------------------------|---------|-----------------------------------|
| Feb 7  | **Database Tuning lecture 17**               | **Relational Algebra lecture 18** | **Query Processing Overview lecture 19**
| Feb 14 | **No class (Presidents Day)**                | **Operator Algorithms**           | **Project 3 due**
| Feb 21 | **Query Optimization**                       | **Query Optimization**            | **Homework 3 due**
| Feb 28 | **Query Optimization**                       | **Query Optimization**            | **Parallel and Distributed DBMSs**
| Mar 7  | Pig Latin                                     | TBA                               | **Wrap-up Project 4 due**
| Mar 14 | Final Exam                                    | **Thursday, March 17, 8:30am-10:20am, in class** |
Review Relational Algebra

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

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

Give a relational algebra expression for this query:

\[ \pi \text{sname}\left(\sigma \text{scity='Seattle' \land sstate='WA' \land pno=2}\left(Supplier \bowtie_{\text{sid = sid}}\text{Supply}\right)\right) \]
Key Idea: Algebraic Optimization

\[ N = \frac{((z*2) + ((z*3) + y))}{x} \]

Given \( x = 1 \), \( y = 0 \), and \( z = 4 \), solve for \( N \).

In what order did you perform the operations?

And how many operations?
Key Idea: Algebraic Optimization

\[ N = \frac{((z*2) + ((z*3) + 0))}{1} \]

Given \( x = 1, y = 0, \) and \( z = 4, \) solve for \( N \) again, but now assume:
* costs 10 units
+ costs 2 units
/ costs 50 units

Which execution plan offers the lowest cost?
Key Idea: Algebraic Optimization

\[ N = \frac{(z \times 2) + ((z \times 3) + 0)}{1} \]

Algebraic Laws:

1. (+) identity: \( x + 0 = x \)
2. (/) identity: \( x / 1 = x \)
3. (*) distributes: \( n \times x + n \times y = n \times (x + y) \)
4. (*) commutes: \( x \times y = y \times x \)

Apply rules 1, 3, 4, 2:
\[ N = (2 + 3) \times z \]

two operations instead of five, no division operator
Optimization with Relational Algebra

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

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

Here is a different relational algebra expression for this query:

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

Query Optimization Goal:
For a query, there may exist many logical and physical query plans. Query Optimizer needs to pick a "good" one.
Hands-on Example

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

Some statistics

- $T(\text{Supplier}) = 1000$ records
- $T(\text{Supply}) = 10,000$ records
- $B(\text{Supplier}) = 100$ pages
- $B(\text{Supply}) = 100$ pages
- $V(\text{Supplier,scity}) = 20$
- $V(\text{Supplier,state}) = 10$
- $V(\text{Supply,pno}) = 2,500$
- Both relations are clustered
- $M = 10$
Physical Query Plan 1

\[
\begin{align*}
T(\text{Supplier}) &= 1,000 \\
T(\text{Supply}) &= 10,000 \\
B(\text{Supplier}) &= 100 \\
B(\text{Supply}) &= 100 \\
V(\text{Supplier, scity}) &= 20 \\
V(\text{Supplier, state}) &= 10 \\
V(\text{Supply, pno}) &= 2,500 \\
M &= 10
\end{align*}
\]

1. (Block-nested loop)
   \[
   \text{Supplier}(\text{sid}, \text{sname}, \text{scity}, \text{sstate})
   \]
   \[
   \text{Supply}(\text{sid}, \text{pno}, \text{quantity})
   \]
   \[
   \pi_{\text{sname}}\ \sigma_{\text{scity}=\text{Seattle} \land \text{sstate}=\text{WA} \land \text{pno}=2}
   \]
   \[
   \frac{B(\text{Supplier}) + B(\text{Supplier}) \cdot B(\text{Supply})}{M} = 100 + 100 \cdot 100/10 = 1,100 \text{ I/Os}
   \]

2. (On the fly)
   \[
   \pi_{\text{sname}}
   \]

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

Selection and project on-the-fly
\[\rightarrow\text{No additional cost.}\]

Cost = 1,100 I/Os
Physical Query Plan 2

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

\[ T(\text{Supplier}) = 1,000 \quad B(\text{Supplier}) = 100 \quad V(\text{Supplier, scity}) = 20 \quad M = 10 \]
\[ T(\text{Supply}) = 10,000 \quad B(\text{Supply}) = 100 \quad V(\text{Supplier, state}) = 10 \]
\[ V(\text{Supply, pno}) = 2,500 \]

\[ \begin{align*}
\text{Cost} & \approx 204 \text{ I/Os} \\
\pi_{\text{sname}} & \text{(On the fly)} \\
\sigma_{\text{scity}=\text{Seattle} \land \text{sstate}=\text{WA}} & \text{(Scan write to T1)} \\
\sigma_{\text{pno}=2} & \text{(Scan write to T2)} \\
\end{align*} \]
Physical Query Plan 3

**Supplier**
- **sid**
- **sname**
- **scity**
- **sstate**

**Supply**
- **sid**
- **pno**
- **quantity**

T(Supplier) = 1,000
B(Supplier) = 100
V(Supplier, scity) = 20
M = 10
V(Supplier, state) = 10
V(Supplier, pno) = 2,500

T(Supply) = 10,000
B(Supply) = 100

(On the fly) \[\pi_{sname}\]

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

(Use index) \[\sigma_{pno=2}\]

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

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

(Index nested loop) \[\text{sid } = \text{sid}\]

Cost \approx 5 \text{ I/Os}
Simplifications

- In the previous examples, we assumed that all index pages were in memory.

- When this is not the case, we need to add the cost of fetching index pages from disk.
Query Optimization Goal / Algorithm

- **Query Optimization Goal**
  - For a query, there exist many logical and physical plans. Query optimizer needs to pick a good one. How?

- **Query Optimization Algorithm**
  - Enumerate alternative plans
  - Compute estimated cost of each plan
    - Compute both number of I/Os, and CPU cost
  - Choose plan with lowest cost
    - This is called cost-based optimization
Lessons

- Need to consider several physical plan
  - even for one, simple logical plan
- No magic “best” plan: depends on the data
- In order to make the right choice
  - need to have statistics over the data
  - the B’s, the T’s, the V’s
Outline

- **Search space**
- Algorithm for enumerating query plans
- Estimating the cost of a query plan
Relational Algebra Equivalences

- **Selections**
  - Commutative: $\sigma_{c_1}(\sigma_{c_2}(R))$ same as $\sigma_{c_2}(\sigma_{c_1}(R))$
  - Cascading: $\sigma_{c_1 \land c_2}(R)$ same as $\sigma_{c_2}(\sigma_{c_1}(R))$

- **Projections**
  - Projections can be added as long as all attributes are kept that are used in later operators or the results

- **Joins**
  - Commutative: $R \bowtie S$ same as $S \bowtie R$
  - Associative: $R \bowtie (S \bowtie T)$ same as $(R \bowtie S) \bowtie T$
Left-Deep Plans and Bushy Plans

4 relations:
• # different tree shapes = 5
• # different orders = 4! = 24
• # different join trees = 5 * 24 = 120
Commutativity, Associativity, Distributivity

\[
R \cup S = S \cup R, \quad R \cup (S \cup T) = (R \cup S) \cup T
\]

\[
R \triangleleft S = S \triangleleft R, \quad R \triangleleft (S \triangleleft T) = (R \triangleleft S) \triangleleft T
\]

\[
R \triangleleft (S \cup T) = (R \triangleleft S) \cup (R \triangleleft T)
\]
Example

Which plan is more efficient: 
R ⨝ (S ⨝ T) or (R ⨝ S) ⨝ T?

- Assumptions:
  - Every *join selectivity* is 10%
    - That is: $T(R \bowtie S) = 0.1 \times T(R) \times T(S)$ etc.
  - $B(R)=100$, $B(S) = 50$, $B(T)=500$
  - All joins are main memory joins
  - All intermediate results are materialized

Note: sometimes defined differently!
Laws involving selection:

\[ \sigma_{C \text{ AND } C'}(R) = \sigma_C(\sigma_{C'}(R)) = \sigma_C(R) \cap \sigma_{C'}(R) \]

\[ \sigma_{C \text{ OR } C'}(R) = \sigma_C(R) \cup \sigma_{C'}(R) \]

\[ \sigma_C(R - S) = \sigma_C(R) - S \]

\[ \sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S) \]

\[ \sigma_C(R \Join S) = \sigma_C(R) \Join S \]

When \( C \) involves only attributes of \( R \)
Example: Simple Algebraic Laws

Example: R(A, B, C, D), S(E, F, G)

$$\sigma_{F=3} (R \bowtie_{D=E} S) = ?$$

$$= R \bowtie_{D=E} (\sigma_{F=3} S)$$

$$\sigma_{A=5 \text{ AND } G=9} (R \bowtie_{D=E} S) = ?$$

$$= \sigma_{A=5} (\sigma_{G=9} (R \bowtie_{D=E} S))$$

$$= (\sigma_{A=5} R) \bowtie_{D=E} (\sigma_{G=9} S))$$
Laws Involving Projections

\[ \Pi_M(R \Join S) = \Pi_M(\Pi_P(R) \Join \Pi_Q(S)) \]

\[ \Pi_M(\Pi_N(R)) = \Pi_M(R) \]

/* note that \( M \subseteq N */

- Example \( R(A,B,C,D), S(E, F, G) \)
  \[ \Pi_{A,B,G}(R \Join_{D=E} S) = \Pi_? (\Pi_?(R) \Join_{D=E} \Pi_?(S)) \]
Laws involving grouping and aggregation

\[
\delta(\gamma_A, \text{agg}(B)(R)) = \gamma_A, \text{agg}(B)(R)
\]

\[
\gamma_A, \text{agg}(B)(\delta(R)) = \gamma_A, \text{agg}(B)(R)
\]

if agg is “duplicate insensitive”

Which of the following are “duplicate insensitive”? sum, count, avg, min, max
Laws Involving Constraints

Product(pid, pname, price, cid)
Company(cid, cname, city, state)

\[ \Pi_{\text{pid, price}}(\text{Product} \bowtie_{\text{cid} = \text{cid}} \text{Company}) = \Pi_{\text{pid, price}}(\text{Product}) \]

Need a second constraint for this law to hold. Which?
Example

Product(pid, pname, price, cid)
Company(cid, cname, city, state)

CREATE VIEW CheapProductCompany
  SELECT *
  FROM Product x, Company y
  WHERE x.cid = y.cid and x.price < 100

SELECTpname, price
FROM CheapProductCompany

SELECTpname, price
FROM Product
WHERE price < 100
Laws with Semijoins

Recall the definition of a semijoin:

- \( R \bowtie S = \Pi_{A_1,\ldots,A_n} (R \bowtie S) \)

- Where the schemas are:
  - Input: \( R(A_1,\ldots,A_n), S(B_1,\ldots,B_m) \)
  - Output: \( T(A_1,\ldots,A_n) \)

Remember from lecture 18:

\[ R \bowtie S = (R \bowtie S) \bowtie S \]

Observe the "dangling" triangle, doesn't "join" with any content, poor lonely triangle 😞
Laws with Semijoins

Example:

\[ Q = R(A,B) \Join S(B,C) \]

A reducer is:

\[ R_1(A,B) = R(A,B) \Join S(B,C) \]

The rewritten query is:

\[ Q = R_1(A,B) \Join S(B,C) \]

\[ R \Join S = (R \Join S) \Join S \]

Why else would we do this?
Why Would We Do This?

- Large attributes:
  \[ Q = R(A, B, D, E, F, \ldots) \bowtie S(B, C, M, K, L, \ldots) \]

- Expensive side computations
  \[ Q = (\gamma_{A,B,\text{count}(\ast)}R(A,B,D)) \bowtie (\sigma_{C=\text{value}}S(B,C)) \]

\[ R_1(A,B,D) = R(A,B,D) \times \sigma_{C=\text{value}}(S(B,C)) \]
\[ Q = (\gamma_{A,B,\text{count}(\ast)}R_1(A,B,D)) \bowtie (\sigma_{C=\text{value}}S(B,C)) \]
Laws with Semijoins

Example:

\[ Q = R(A,B) \bowtie S(B,C) \]

A reducer is:

\[ R_1(A,B) = R(A,B) \bowtie S(B,C) \]

The rewritten query is:

\[ Q = R_1(A,B) \bowtie S(B,C) \]

Are there dangling tuples?


Laws with Semijoins

- Example:
  \[ Q = R(A,B) \bowtie S(B,C) \]

- A full reducer is:
  \[ R_1(A,B) = R(A,B) \bowtie S(B,C) \]
  \[ S_1(B,C) = S(B,C) \bowtie R_1(A,B) \]

- The rewritten query is:
  \[ Q = R_1(A,B) \bowtie S_1(B,C) \]

No more dangling tuples
Laws with Semijoins

- More complex example:

\[ Q = R(A,B) \bowtie S(B,C) \bowtie T(C,D,E) \]

- A full reducer is:

\[
\begin{align*}
S'(B,C) &= S(B,C) \bowtie R(A,B) \\
T'(C,D,E) &= T(C,D,E) \bowtie S'(B,C) \\
S''(B,C) &= S'(B,C) \bowtie T'(C,D,E) \\
R'(A,B) &= R(A,B) \bowtie S''(B,C)
\end{align*}
\]

\[ Q = R'(A,B) \bowtie S''(B,C) \bowtie T'(C,D,E) \]
Laws with Semijoins

- **Example:**
  \[
  Q = R(A,B) \bowtie S(B,C) \bowtie T(A,C)
  \]

- Doesn’t have a full reducer (we can reduce forever)

- **Theorem:** A query has a full reducer iff it is **acyclic**
  (see Chapter 20.4)
  - (if interested, you find the proof in the book [1995, Database Theory, by Abiteboul, Hull, Vianu])
Laws with Semijoins

Semijoins

- Given a query: \[ Q = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n \]

- A semijoin reducer for Q is

  \[ R_{i1} = R_{i_1} \bowtie R_{j_1} \]
  \[ R_{i2} = R_{i_2} \bowtie R_{j_2} \]
  \[ \ldots \]
  \[ R_{ip} = R_{i_p} \bowtie R_{j_p} \]

- such that the query is equivalent to:

  \[ Q = R_{k_1} \bowtie R_{k_2} \bowtie \ldots \bowtie R_{kn} \]

- A full reducer is such that no dangling tuples remain
Example with Semijoins

CREATE VIEW DepAvgSal As (  
  SELECT E.did, Avg(E.Sal) AS avgsal  
  FROM Emp E  
  GROUP BY E.did)

SELECT E.eid, E.sal  
FROM Emp E, Dept D, DepAvgSal V  
WHERE E.did = D.did and D.budget > 100k  
  and E.age < 30 and E.did = V.did  
  and E.sal > V.avgsal

Goal: compute only the necessary part of the view
Example with Semijoins

Emp(eid, ename, sal, did)
Dept(did, dname, budget)
DeptAvgSal(did, avgsal) /* view */

New view uses a reducer:

```sql
CREATE VIEW LimitedAvgSal As (
    SELECT E.did, Avg(E.Sal) AS avgsal
    FROM Emp E, Dept D
    WHERE E.did = D.did and D.budget > 100k
    GROUP BY E.did)
```

New Query:

```sql
SELECT E.eid, E.sal
FROM Emp E, Dept D, LimitedAvgSal V
WHERE E.did = D.did and D.budget > 100k
    and E.age < 30 and E.did = V.did
    and E.sal > V.avgsal
```
Example with Semijoins

```
CREATE VIEW PartialResult AS
    (SELECT E.eid, E.sal, E.did
    FROM Emp E, Dept D
    WHERE E.did=D.did and E.age < 30
        and D.budget > 100k)

CREATE VIEW Filter AS
    (SELECT DISTINCT P.did FROM PartialResult P)

CREATE VIEW LimitedDepAvgSal AS
    (SELECT E.did, Avg(E.Sal) AS avgsal
    FROM Emp E, Filter F
    WHERE E.did = F.did
    GROUP BY E.did)
```

PODS'98, by Chaudhuri
Example with Semijoins

Original query:

```
SELECT E.eid, E.sal
FROM Emp E, Dept D, DepAvgSal V
WHERE E.did = D.did and E.did = V.did
  and E.age < 30 and D.budget > 100k
  and E.sal > V.avgsal
```

New query:

```
SELECT P.eid, P.sal
FROM PartialResult P, LimitedDepAvgSal V
WHERE P.did = V.did
  and P.sal > V.avgsal
```
Search Space Challenges

- Search space is huge!
  - Many possible equivalent trees
  - Many implementations for each operator
  - Many access paths for each relation
    - File scan or index + matching selection condition

- Cannot consider ALL plans
  - Heuristics: only partial plans with "low" cost
Outline

- Search space
- Algorithm for enumerating query plans
- Estimating the cost of a query plan
Key Decisions

- **Logical plan**
  - What logical plans do we consider (left-deep, bushy?)
  - *Search Space*
  - Which algebraic laws do we apply, and in which context(s)?
    - *Optimization rules*
  - In what order do we explore the search space?
    - *Optimization algorithm*

- **Physical plan**
  - What physical operators to use?
  - What access paths to use (file scan or index)?
Optimizers

- **Heuristic-based optimizers:**
  - Apply greedily rules that always improve
    - Typically: push selections down
  - Very limited: no longer used today

- **Cost-based optimizers**
  - Use a cost model to estimate the cost of each plan
  - Select the “cheapest” plan
The Search Space

1. Complete plans
2. Bottom-up plans
3. Top-down plans
Seach Space 1: Complete Plans

```
SELECT * FROM R, S, T
WHERE R.B = S.B
    and S.C = T.C
    and R.A < 40

Why is this search space inefficient?
```

```
R(A,B) S(B,C) T(C,D)
```

```
......
```

```
σ_{A<40}

σ_{A<40}

R

S

T
```

```
......
```

```
σ_{A<40}

R

S

T
```

```
......
```

```
σ_{A<40}

R

S

T
```

```
......
```
Seach Space 2: Bottom-up Partial Plans

SELECT * FROM R, S, T WHERE R.B=S.B and S.C=T.C and R.A<40

Why is this better?
Seach Space 3: Top-down Partial Plans

\[
\begin{align*}
\text{SELECT} & \quad * \\
\text{FROM} & \quad R, S, T \\
\text{WHERE} & \quad R.B=S.B \\
& \quad \text{and} \quad S.C=T.C \\
& \quad \text{and} \quad R.A<40
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad * \\
\text{FROM} & \quad R, S \\
\text{WHERE} & \quad R.B=S.B \\
& \quad \text{and} \quad R.A < 40
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad * \\
\text{FROM} & \quad R, S, T \\
\text{WHERE} & \quad R.B=S.B \\
& \quad \text{and} \quad S.C=T.C \\
& \quad \text{and} \quad R.A<40
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad R.A, T.D \\
\text{FROM} & \quad R, S, T \\
\text{WHERE} & \quad R.B=S.B \\
& \quad \text{and} \quad S.C=T.C
\end{align*}
\]

\[
\sigma_{A<40}
\]

......
Plan Enumeration Algorithms

- **Dynamic programming** (in class)
  - Classical algorithm [1979]
  - Limited to joins: *join reordering algorithm*
  - Bottom-up

- **Rule-based algorithm (will not discuss)**
  - Database of rules (=algebraic laws)
  - Usually: dynamic programming
  - Usually: top-down
Dynamic Programming

Originally proposed in System R [1979]

- Only handles single block queries:

```
SELECT list
FROM R1, ..., Rn
WHERE cond_1
    and cond_2
    and ...
    and cond_k
```

- Heuristics: selections down, projections up
Dynamic Programming

- Search space = join trees
- Algebraic laws = commutativity, associativity
- Algorithm = dynamic programming 😊
Join Trees

- $R_1 \Join R_2 \Join \ldots \Join R_n$
- Join tree:

  \[
  \begin{array}{c}
    \Join \\
    R_3 \\
    \Join \\
    R_1 \\
    \Join \\
    R_2 \\
    \Join \\
    R_4
  \end{array}
  \]

- A plan = a join tree
- A partial plan = a subtree of a join tree
## Types of Join Trees

### Left-deep plan

```
  R3
  |   R1
  |   |   R5
  |   |   |   R4
  |   |   |   |   R2
```

### Bushy plan

```
  R3
  |   R2
  |   |   R4
  |   |   |   R1
  |   |   |   |   R5
```

### Right-deep plan

```
  R3
  |   R5
  |   |   R4
  |   |   |   R2
  |   |   |   |   R1
```

<table>
<thead>
<tr>
<th># relations</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td># tree shapes</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>14</td>
<td>42</td>
<td>132</td>
</tr>
<tr>
<td># permutations</td>
<td>2</td>
<td>6</td>
<td>24</td>
<td>120</td>
<td>720</td>
<td>5040</td>
</tr>
<tr>
<td># join trees</td>
<td>2</td>
<td>12</td>
<td>120</td>
<td>1680</td>
<td>&gt;30k</td>
<td>&gt;665k</td>
</tr>
</tbody>
</table>
Join ordering:

- Given: a query $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$
- Find optimal order
- Assume we have a function cost() that gives us the cost of every join tree
Dynamic Programming

- For each subquery $Q \subseteq \{R_1, \ldots, R_n\}$ compute the following:
  - $\text{Size}(Q) = \text{the estimated size of } Q$
  - $\text{Plan}(Q) = \text{a best plan for } Q$
  - $\text{Cost}(Q) = \text{the estimated cost of that plan}$
Dynamic Programming

- **Step 1:** For each \( \{R_i\} \), set:
  - \( \text{Size}(\{R_i\}) = B(R_i) \)
  - \( \text{Plan}(\{R_i\}) = R_i \)
  - \( \text{Cost}(\{R_i\}) = (\text{cost of scanning } R_i) \)

```
SELECT list
FROM R1, ..., Rn
WHERE cond_1 and ... and cond_k
```
Dynamic Programming

- **Step 2:** For each \( Q \subseteq \{R_1, ..., R_n\} \) involving \( i \) relations:
  - Size\( (Q) \) = estimate it recursively
  - For every pair of subqueries \( Q', Q'' \) s.t. \( Q = Q' \cup Q'' \)
    - compute \( \text{cost(Plan}(Q') \land \text{Plan}(Q'')) \)
    - Cost\( (Q) \) = the smallest such cost
    - Plan\( (Q) \) = the corresponding plan

- **Step 3:** Return \( \text{Plan}(\{R_1, ..., R_n\}) \)

---

```
SELECT list 
FROM R1, ..., Rn 
WHERE cond_1 and ... and cond_k 
```
Example: $R \bowtie S \bowtie T \bowtie U$

To illustrate, ad-hoc cost model (from the book 😊):

- $\text{Cost}(P_1 \bowtie P_2) = \text{Cost}(P_1) + \text{Cost}(P_2) + \text{size(intermediate results for } P_1, P_2)$
- Cost of a scan = 0

Further assumptions:

<table>
<thead>
<tr>
<th>$T(R)$</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(S)$</td>
<td>5000</td>
</tr>
<tr>
<td>$T(T)$</td>
<td>3000</td>
</tr>
<tr>
<td>$T(U)$</td>
<td>1000</td>
</tr>
</tbody>
</table>

All join selectivities = 1%

- $T(R \bowtie S) = 0.01 \times T(R) \times T(S)$
- $T(S \bowtie T) = 0.01 \times T(S) \times T(T)$
- etc.
<table>
<thead>
<tr>
<th>Subquery</th>
<th>Size</th>
<th>Cost</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSTU</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ T(R) = 2000 \]
\[ T(S) = 5000 \]
\[ T(T) = 3000 \]
\[ T(U) = 1000 \]
<table>
<thead>
<tr>
<th>Subquery</th>
<th>Size</th>
<th>Cost</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>100k</td>
<td>0</td>
<td>RS</td>
</tr>
<tr>
<td>RT</td>
<td>60k</td>
<td>0</td>
<td>RT</td>
</tr>
<tr>
<td>RU</td>
<td>20k</td>
<td>0</td>
<td>RU</td>
</tr>
<tr>
<td>ST</td>
<td>150k</td>
<td>0</td>
<td>ST</td>
</tr>
<tr>
<td>SU</td>
<td>50k</td>
<td>0</td>
<td>SU</td>
</tr>
<tr>
<td>TU</td>
<td>30k</td>
<td>0</td>
<td>TU</td>
</tr>
<tr>
<td>RST</td>
<td>3M</td>
<td>60k</td>
<td>(RT)S</td>
</tr>
<tr>
<td>RSU</td>
<td>1M</td>
<td>20k</td>
<td>(RU)S</td>
</tr>
<tr>
<td>RTU</td>
<td>0.6M</td>
<td>20k</td>
<td>(RU)T</td>
</tr>
<tr>
<td>STU</td>
<td>1.5M</td>
<td>30k</td>
<td>(TU)S</td>
</tr>
<tr>
<td>RSTU</td>
<td>30M</td>
<td>60k</td>
<td>(RT)(SU)</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
T(R) &= 2000 \\
T(S) &= 5000 \\
T(T) &= 3000 \\
T(U) &= 1000
\end{align*} \]
Reducing the Search Space

- **Restriction 1: only linear trees (no bushy)**
  Most systems restrict the search space to left-deep plans. Note that for some join algorithms, there exist different conventions about which is the build and which is the probe relation. The convention of our textbook (see example 16.31 p.818) assumes the build relation on the left, and hence calls right-deep plans those with several build relations in main memory. Don't let this detail confuse you. In practice it does not matter, as the optimizer does not actually "draw" these trees. The fact that they are linear is the only thing that matters.

- **Restriction 2: no trees with cartesian product**

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

Plan: \((R(A,B) \Join T(C,D)) \Join S(B,C)\)

has a cartesian product.

Most query optimizers will not consider it.
Dynamic Programming: Summary

- Handles only join queries:
  - Selections are pushed down (i.e. early)
  - Projections are pulled up (i.e. late)

- Takes exponential time in general, BUT:
  - Left linear joins may reduce time
  - Non-cartesian products may reduce time further
Rule-Based Optimizers

- Extensible collection of rules
  Rule = Algebraic law with a direction
- Algorithm for firing these rules
  Generate many alternative plans, in some order
  Prune by cost

- Volcano (later SQL Server)
- Starburst (later DB2)
Completing the Physical Query Plan

- Choose algorithm for each operator
  - How much memory do we have?
  - Are the input operand(s) sorted?

- Access path selection for base tables

- Decide for each intermediate result:
  - To materialize
  - To pipeline
Access Path Selection

- **Access path**: a way to retrieve tuples from a table
  - A file scan
  - An index *plus* a matching selection condition

- **Index matches selection condition** if it can be used to retrieve just tuples that satisfy the condition
  - Example: `Supplier(sid,sname,scity,sstate)`
  - B+-tree index on `(scity,sstate)`
    - matches `scity='Seattle'`
    - does not match `sid=3`, does not match `sstate='WA'`
Access Path Selection

- Relation: Supplier(sid, sname, scity, sstate)
- Selection condition: sid > 300 ∧ scity='Seattle'
- Indexes: B+-tree on sid and B+-tree on scity

- Which access path should we use?
  - We should pick the most selective access path
Access Path Selectivity

Access path selectivity:
- number of pages retrieved if we use this access path
- Most selective retrieves fewest pages

As we saw earlier, for equality predicates:
- Selection on equality: $\sigma_{a=v}(R)$
- $V(R,a) = \# \text{ of distinct values of attribute } a$
- $\frac{1}{V(R,a)}$ is thus the reduction factor
- Clustered index on $a$: cost $\frac{B(R)}{V(R,a)}$
- Unclustered index on $a$: cost $\frac{T(R)}{V(R,a)}$
- (we are ignoring I/O cost of index pages for simplicity)
Materialize Intermediate Results b/w Operators

Convention of the book: build relations on the left.
Materialize Intermediate Results b/w Operators

Question in class

Given B(R), B(S), B(T), B(U)

- What is the total cost of the plan?
  - Cost =

- How much main memory do we need?
  - M =

\[\text{Cost} = \]

\[\text{M} = \]
Pipeline Between Operators

convention of the book: build relations on the left.
Pipeline Between Operators

Question in class

Given $B(R), B(S), B(T), B(U)$

- What is the total cost of the plan?
  - Cost =

- How much main memory do we need?
  - $M =$
Pipeline in Bushy Trees
Example "Star Schema"

**Customers**

<table>
<thead>
<tr>
<th>Cid</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Purchases**

<table>
<thead>
<tr>
<th>Cid</th>
<th>Pid</th>
<th>Store</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>00143</td>
<td>U-district</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Stores**

<table>
<thead>
<tr>
<th>Sname</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>U-district</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Items**

<table>
<thead>
<tr>
<th>Pid</th>
<th>Name</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>00143</td>
<td>Banana</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The diagram illustrates a star schema with tables linked by foreign keys. The **Customers** table has a customer with the name "Alice". The **Purchases** table links customers to purchases made at stores. The **Stores** table categorizes the stores as "small" and "big". The **Items** table lists items purchased, with a specific item "Banana" associated with a purchase."
Possible Naming Confusion

Right-deep plan

Convention in the book and in this class!
See our textbook example 16.31 p.818

Note that you may find the same evaluation strategy (all 3 build relations in main memory) at other places depicted as above. Reason is that build and probe are reversed. Hence they call this a left-deep plan. Don't get confused, just so you know.
Types of Join Trees

- **Left-deep plan**: intermediate results in main memory and thus pipelined
  - E.g. good for "non-expansive" joins
  - Each too big for main memory

- **Bushy plan**: main memory
  - E.g. good for "expansive" joins

- **Right-deep plan**: each too big for main memory
  - E.g. good for "non-expansive" joins
  - Def. "expansive join": $|R \bowtie S| > \max(|A|, |B|)$

Convention in the book and in this class!

Source: [Stocker et al. ICDE 2001]
Example

Logical plan

Naïve evaluation:
- 2 partitioned hash-joins
- Cost: \[ 3B(R) + 3B(S) + n + [3n + 3B(U)] \]
  \[ = 3B(R) + 3B(R) + 3B(U) + 4n \]
  \[ = 75,000 + 4n \]
Smarter:

- Step 1: hash $R$ on $x$ into 100 buckets, each of 50 blocks; to disk
- Step 2: hash $S$ on $x$ into 100 buckets, each of 100 blocks; to disk
- Step 3: read each $R_i$ in memory (50 buffer) join with $S_i$ (1 buffer); hash result on $y$ into 50 buckets (50 buffers) -- here we pipeline
- Cost so far: $3B(R) + 3B(S)$
Example

Continuing:
- How large are the 50 buckets on \( y \)? Answer: \( n/50 \).
- If \( n \leq 50 \) then keep all 50 buckets in Step 3 in memory, then:
  - Step 4: read \( U \) from disk, hash on \( y \) and join with memory
  - Total cost: \( 3B(R) + 3B(S) + B(U) = 55,000 \)
Example

Continuing:

- If $50 < n \leq 5000$ then send the 50 buckets in Step 3 to disk
  - Each bucket has size $n/50 \leq 100$
- Step 4: partition $U$ into 50 buckets
- Step 5: read each partition and join in memory
- Total cost: $3B(R) + 3B(S) + 2n + 3B(U) = 75,000 + 2n$
Example

Continuing:
- If $n > 5000$ then materialize instead of pipeline
- 2 partitioned hash-joins
- Cost $3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4n$
Example in pictures 1

If $n \leq 50$: $3B(R) + 3B(S) + B(U)$

Source: Variation on example 16.36 from book; all cost units are in "blocks" = I/O
If $50 < n \leq 5000$: $3B(R) + 3B(S) + 2n + 3B(U)$

Source: Variation on example 16.36 from book; all cost units are in "blocks" = I/O
If $5000 < n$: $3B(R) + 3B(S) + 4n + 3B(U)$

Source: Variation on example 16.36 from book; all cost units are in "blocks" = I/O
Outline

- Search space
- Algorithm for enumerating query plans
- Estimating the cost of a query plan
Computing the Cost of a Plan

- Collect statistical summaries of stored data
- Estimate size in a bottom-up fashion
- Estimate cost by using the estimated size
Statistics on Base Data

- Collected information for each relation
  - Number of tuples (cardinality)
  - Indexes, number of keys in the index
  - Number of physical pages, clustering info
  - Statistical information on attributes
    - Min value, max value, number distinct values
    - Histograms
  - Correlations between columns (hard)

- Collection approach
  - periodic
  - using sampling
Size Estimation Problem

\[
S = \text{SELECT } \text{list} \\
\text{FROM } \text{R1, ..., Rn} \\
\text{WHERE } \text{cond}_1 \text{ AND } \text{cond}_2 \text{ AND } \ldots \text{ AND } \text{cond}_k
\]

Given T(R1), T(R2), ..., T(Rn)
Estimate T(S)

How can we do this? Note: doesn’t have to be exact.

Remark: \( T(S) \leq T(R1) \times T(R2) \times \ldots \times T(Rn) \)
Selectivity Factor

- Each condition "cond" reduces the size by some factor called **selectivity (factor)**
  - **selection**
    \[
    \text{sel}_\sigma = \frac{|\sigma_C(R)|}{|R|}
    \]
  - **join**
    \[
    \text{sel}_\Join = \frac{|R \Join S|}{|R \times S|}
    \]
- Assuming independence, multiply the selectivity factors
Example

```
SELECT  *                  
FROM    R, S, T           
WHERE   R.B = S.B and S.C = T.C and R.A < 40
```


\[
T(R) = 300, \quad T(S) = 2000, \quad T(T) = 100
\]

Selectivity of R.B = S.B is 1/3
Selectivity of S.C = T.C is 1/10
Selectivity of R.A < 40 is ½

What is the estimated size of the query output?
Rule of Thumb

- If selectivities are unknown, then:
  selectivity factor = 1/10

[System R, 1979]
Selectivities from Statistics

- **Condition is** $A = c$  /* value selection on R */
  - Selectivity = $\frac{1}{V(R, A)}$

- **Condition is** $A < c$  /* range selection on R */
  - Selectivity = $\frac{(c - \text{Low}(R, A))}{(\text{High}(R, A) - \text{Low}(R, A))} \cdot \text{T}(R)$

- **Condition is** $A = B$  /* $R \bowtie_{A=B} S$ */
  - Selectivity = $\frac{1}{\max(V(R, A), V(S, A))}$
  - (will explain next)
Selectivity of $R \bowtie_{A=B} S$

Assumption:
Containment of values:
• if $V(R,A) \leq V(S,B)$, then
  the set of $A$ values of $R$
  is subset of $B$ values of $S$
• Here: $\{1,2,3\} \subseteq \{1,2,3,4\}$

When does this hold for sure?

Conclusion 1:
• A tuple from $R$ joins with expected $\frac{T(S)}{V(S,B)}$ tuples from $S$

Conclusion 2:
• Expected join size is $\frac{T(S) \cdot T(R)}{V(S,B)} = 12.5$

Why different?

| $R \times S|$ = 60
| $R \bowtie_{A=B} S$ | = 12

$T(R) = 10$
$V(R,A) = 3$
$T(S) = 6$
$V(S,B) = 4$

$\text{sel}_{\bowtie} = \frac{12}{60} = \frac{1}{5}$
Selectivity of $R \bowtie_{A=B} S$

**Assumption:**

**Containment of values:**

- if $V(R,A) \leq V(S,B)$, then the set of $A$ values of $R$ is subset of $B$ values of $S$
- Here: $\{1,2,3\} \subseteq \{1,2,3,4\}$
- Holds for sure if $A$ in $R$ is a foreign key on $B$ in $S$ (*not the case here*)

**Conclusion 1:**

- A tuple from $R$ joins with expected $\frac{T(S)}{V(S,B)}$ tuples from $S$

**Conclusion 2:**

- Expected join size is $\frac{T(S) \cdot T(R)}{V(S,B)} = 12.5$
- Expected sel$_{RS} = \frac{1}{V(S,B)} = 0.25$

\[
\begin{align*}
|R \times S| &= 60 \\
|R \bowtie_{A=B} S| &= 26
\end{align*}
\]

\[
\text{sel}_{\bowtie} = \frac{12}{60} = 0.43
\]
Selectivity of $R \bowtie_{A=B} S$

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

$T(R) = 10$
$V(R,A) = 3$
$T(S) = 6$
$V(S,B) = 4$

$|R \times S| = 60$
$|R \bowtie_{A=B} S| = 10$

$\text{sel}_{\bowtie} = 12/60 = 0.17$

Assumption:

Containment of values:
- if $V(R,A) \leq V(S,B)$, then the set of $A$ values of $R$ is subset of $B$ values of $S$
- Here: $\{1,2,3\} \subseteq \{1,2,3,4\}$
- Holds for sure if $A$ in $R$ is a foreign key on $B$ in $S$ (not the case here)

Conclusion 1:
- A tuple from $R$ joins with expected $T(S)/V(S,B)$ tuples from $S$

Conclusion 2:
- Expected join size is
  $T(S) T(R) / V(S,B) = 12.5$
- Expected $\text{sel}_{RS} = 1/V(S,B) = 0.25$
Assumptions

1: 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 $S$

2: Preservation of values: for any other attribute $C$, $V(R \Join_{A=B} S, C) = V(R,C)$ if $C$ is attribute of $R$
Example:
- $T(R) = 10,000$, $T(S) = 20,000$
- $V(R,A) = 100$, $V(S,B) = 200$
- How large is $R \bowtie_{A=B} S$ ?
Histgrams

- Statistics on data maintained by the RDBMS
- Makes size estimation much more accurate
  - hence, cost estimations are more accurate
Histograms

\[
\begin{align*}
\text{Employee}(\text{ssn}, \text{name}, \text{age}) & \quad T(\text{Employee}) = 25,000 \\
& \quad V(\text{Employee}, \text{age}) = 50 \\
& \quad \min(\text{age}) = 18 \\
& \quad \max(\text{age}) = 77 \\
\sigma_{\text{age}=18}(\text{Employee}) = ? & \quad \sigma_{\text{age}>28 \text{ and } \text{age}<35}(\text{Employee}) = ? \\
\end{align*}
\]
Histograms

\[
\begin{align*}
\text{Employee}(\text{ssn, name, age}) & \quad \text{T(Employee) = 25,000} \\
& \quad V(Employee, age) = 50 \\
& \quad \text{min(age) = 18} \\
& \quad \text{max(age) = 77} \\
\sigma_{\text{age}=18}(\text{Employee}) & = ? \\
\sigma_{\text{age}>28 \text{ and age}<35}(\text{Employee}) & = ? \\
\end{align*}
\]

Estimate = 25,000 / 50 = 500

Estimate = 25,000 \times 6 / 60 = 2,500
# Histograms

<table>
<thead>
<tr>
<th>Age:</th>
<th>10..19</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>5,000</td>
<td>12,000</td>
<td>6,500</td>
<td>500</td>
</tr>
</tbody>
</table>

\[ \sigma_{age=18}(\text{Employee}) = ? \] \[ \sigma_{age>28 \text{ and } age<35}(\text{Employee}) = ? \]

T(\text{Employee}) = 25,000
V(\text{Employee, age}) = 50
\text{min(age)} = 18
\text{max(age)} = 77
### Histograms

**Employee(ssn, name, age)**

<table>
<thead>
<tr>
<th>Age:</th>
<th>10..19</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>5,000</td>
<td>12,000</td>
<td>6,500</td>
<td>500</td>
</tr>
</tbody>
</table>

- **T(Employee) = 25,000**
- **V(Employee, age) = 50**
- **min(age) = 18**
- **max(age) = 77**

**σ_{age=18}(Employee) = ?**

**σ_{age>28 and age<35}(Employee) = ?**

**Estimate = 20**

**Estimate = 1*80 + 5*500 = 2580**
Types of Histograms

- How should we determine the bucket boundaries in a histogram?
  - Equi-Width
  - Equi-Depth
    - also called equi-height or equi-sum
  - Compressed
Histograms

1. Equi-Width Histogram

Data values (V = 16)

2. Equi-Depth Histogram (also Equi-Height / Equi-sum)

Note: does not always have to be the same number per bucket, we just try to make it approx. the same

3. Compressed: store separately some highly frequent values: e.g. (10,9)

Source: Numerical values slightly varied from:
# Histograms

Employee(\textit{ssn}, name, age)

## Equi-width

<table>
<thead>
<tr>
<th>Age:</th>
<th>10..19</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>5,000</td>
<td>12,000</td>
<td>6,500</td>
<td>500</td>
</tr>
</tbody>
</table>

## Equi-depth

<table>
<thead>
<tr>
<th>Age:</th>
<th>10..34</th>
<th>35..41</th>
<th>42-45</th>
<th>46-48</th>
<th>49-54</th>
<th>&gt; 55</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>4,200</td>
<td>4,100</td>
<td>4,200</td>
<td>4,300</td>
<td>3,900</td>
<td>4,300</td>
</tr>
</tbody>
</table>
Difficult Questions on Histograms

- Small number of buckets
  - Hundreds, or thousands, but not more
  - WHY?

- *Not* updated during database update, but recomputed periodically
  - WHY?

- Multidimensional histograms rarely used
  - WHY?
Summary of Query Optimization

- Three parts:
  - search space, algorithms, size/cost estimation

- Ideal goal: find optimal plan. But
  - Impossible to estimate accurately
  - Impossible to search the entire space

- Goal of today’s optimizers:
  - Avoid very bad plans
Outline

- Search space
- Algorithm for enumerating query plans
- Estimating the cost of a query plan
- Some extra slides (optional)

The following slides are taken from this German Database textbook. They may provide some alternative intuitions for some of the operators. Topics: Hash Join / value of 2 passes / Merge join / External sort / semi-join / operators.

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung" http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
Partition relations into buckets

\[ R \]

<table>
<thead>
<tr>
<th>...</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 )</td>
<td>5</td>
</tr>
<tr>
<td>( r_2 )</td>
<td>7</td>
</tr>
<tr>
<td>( r_3 )</td>
<td>8</td>
</tr>
<tr>
<td>( r_4 )</td>
<td>5</td>
</tr>
</tbody>
</table>

\[ S \]

<table>
<thead>
<tr>
<th>B</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>( s_1 )</td>
</tr>
<tr>
<td>7</td>
<td>( s_2 )</td>
</tr>
<tr>
<td>10</td>
<td>( s_3 )</td>
</tr>
<tr>
<td>5</td>
<td>( s_4 )</td>
</tr>
</tbody>
</table>

\[ h(A) \]

Bucket 1

| \( r_1 \) | 5 |
| 5 | \( s_1 \) |
| \( r_4 \) | 5 |
| 5 | \( s_4 \) |
| 10 | \( s_3 \) |

\[ h(B) \]

Bucket 2

| \( r_2 \) | 7 |
| 7 | \( s_2 \) |

Bucket 3

| \( r_3 \) | 8 |

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMS_einf/EIS_4_Auflage/index.html (March 2011)
Partitioned Hash Join: **Partitioning**

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung" http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
Partitioned Hash Join: **Build/Probe**

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung" 
http://www3.in.tum.de/research/publications/books/DBMSseinf/EIS_4_Auflage/index.html (March 2011)
Comparing Tuples "on the Diagonal"

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
Comparing Tuples "on the Diagonal"

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
Merge-Join

- Assume R and S are already sorted:
  - Then each relation needs to be read only once

Beispiel:

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung" http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung" http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

merge

3 5 2

3 17 97 5 16 27 2 13 99

run

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung" http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung" http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

merge

run
External Sorting

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung" 
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sorting

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sort w/ Heap-Priority Queue

merge

run

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sort w/ Heap-Priority Queue

merge

run

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung" http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sort w/ Heap-Priority Queue

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sort w/ Heap-Priority Queue

Important: load from green run (i.e. run, from which the object comes)

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung" http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sort w/ Heap-Priority Queue

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sort w/ Heap-Priority Queue

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
External Sort w/ Heap-Priority Queue

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
R \Join S (Natural Join)

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
\[ R \Join S = R \Join (S \Join \Pi_C R) \]

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)
Alternative: $R \Join S = (R \Join S) \Join S$
Logical Algebra → Physical Operators

Notice: in this book, the build relation is on the right (just a convention, not essential)

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)