Introduction to Data Management
CSE 344

Unit 4: RDBMS Internals
Logic and Physical Plans
Query Execution
Query Optimization

(4 lectures)
Introduction to Data Management
CSE 344

Lecture 15: Introduction to Query Evaluation
Class Overview

• Unit 1: Intro
• Unit 2: Relational Data Models and Query Languages
• Unit 3: Non-relational data
  • Unit 4: RDMBS internals and query optimization
• Unit 5: Parallel query processing
• Unit 6: DBMS usability, conceptual design
• Unit 7: Transactions
• Unit 8: Advanced topics (time permitting)
From Logical RA Plans to Physical Plans
Query Evaluation Steps Review

SQL query

Parse & Rewrite Query

Select Logical Plan

Select Physical Plan

Query Execution

Disk

Logical plan (RA)

Physical plan

Query optimization
Relational Algebra Operators

- Union \( \cup \), intersection \( \cap \), difference \( - \)
- Selection \( \sigma \)
- Projection \( \pi \)
- Cartesian product \( \times \), join \( \bowtie \)
- (Rename \( \rho \))
- Duplicate elimination \( \delta \)
- Grouping and aggregation \( \gamma \)
- Sorting \( \tau \)
Physical Operators

• For each operators above, several possible algorithms
• Main memory or external memory algorithms
• Examples:
  – Main memory hash join
  – External memory merge join
  – External memory partitioned hash join
  – Sort-based group by
  – Etc, etc
Main Memory Algorithms

Logical operator:

\[
\text{Supplier} \bowtie_{\text{sid} = \text{sid}} \text{Supply}
\]

Three algorithms:
1. Nested Loops
2. Hash-join
3. Merge-join
1. Nested Loop Join

Logical operator:
\[
\text{Supplier} \bowtie_{\text{sid}=\text{sid}} \text{Supply}
\]

```plaintext
for x in Supplier do
  for y in Supply do
    if x.sid = y.sid
      then output(x,y)
```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

1. Nested Loop Join

Logical operator:
Supplier \bowtie_{sid=\text{sid}} \text{Supply}

for x in Supplier do
    for y in Supply do
        if x.sid = y.sid
            then output(x,y)

If $|R|=|S|=n$, what is the runtime?
1. Nested Loop Join

Logical operator:

\[ \text{Supplier} \Join_{\text{sid} = \text{sid}} \text{Supply} \]

for \( x \) in \( \text{Supplier} \) do
for \( y \) in \( \text{Supply} \) do
  if \( x.\text{sid} = y.\text{sid} \) then output\((x,y)\)

If \( |R| = |S| = n \), what is the runtime?

\( O(n^2) \)
BRIEF Review of Hash Tables

Separate chaining:

A (naïve) hash function:

\[ h(x) = x \mod 10 \]

Operations:

- find(103) = ??
- insert(488) = ??

Duplicates OK

WHY ??
BRIEF Review of Hash Tables

- insert(k, v) = inserts a key k with value v

- Many values for one key
  - Hence, duplicate k’s are OK

- find(k) = returns the list of all values v associated to the key k
2. Hash Join

Logical operator:
$$\text{Supplier} \bowtie_{\text{sid} = \text{sid}} \text{Supply}$$

for x in Supplier do
    insert(x.sid, x)

for y in Supply do
    x = find(y.sid);
    output(x, y);
2. Hash Join

Logical operator:

\[
\text{Supplier} \bowtie_{\text{sid} = \text{sid}} \text{Supply}
\]

```cpp
for x in Supplier do
    insert(x.sid, x)
for y in Supply do
    x = find(y.sid);
    output(x, y);
```

If \(|R|=|S|=n\), what is the runtime?
2. Hash Join

Logical operator:
Supplier \( \bowtie_{\text{sid} = \text{sid}} \) Supply

```plaintext
for x in Supplier do
  insert(x.sid, x)
for y in Supply do
  x = find(y.sid);
  output(x, y);
```

If \(|R| = |S| = n\), what is the runtime?

\( O(n) \)
2. Hash Join

Logical operator:
\[ Supplier \bowtie_{\text{sid} = \text{sid}} Supply \]

for y in Supply do
insert(y.sid, y)

for x in Supplier do
???

Change join order
2. Hash Join

Logical operator:
\[ \text{Supplier} \Join_{\text{sid} = \text{sid}} \text{Supply} \]

for y in Supply do
    insert(y.sid, y)

for x in Supplier do
    for y in find(x.sid) do
        output(x, y);

Change join order
2. Hash Join

Logical operator:
Supplier $\bowtie_{\text{sid}=\text{sid}}$ Supply

for y in Supply do
    insert(y.sid, y)

for x in Supplier do
    for y in find(x.sid) do
        output(x, y);

If $|R|=|S|=n$, what is the runtime?

Change join order
2. Hash Join

Logical operator:
Supplier \( \bowtie_{\text{sid} = \text{sid}} \) Supply

for y in Supply do
    insert(y.sid, y)

for x in Supplier do
    for y in find(x.sid) do
        output(x, y);

If \(|R| = |S| = n\), what is the runtime?

O(n)

But can be O(n^2) why?

Change join order
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

2. Hash Join

Logical operator:
Supplier \text{\texttt{\textbullet}}_{\texttt{sid}=\texttt{sid}} \text{Supply}

for y in Supply do
  insert(y.sid, y)

for x in Supplier do
  for y in find(x.sid) do
    output(x, y);

If |R|=|S|=n, what is the runtime?
O(n)
But can be O(n^2) why?

Change join order

Why would we change the order?
2. Hash Join

Logical operator:
\[
\text{Supplier} \bowtie_{\text{sid} = \text{sid}} \text{Supply}
\]

for y in Supply do
    insert(y.sid, y)

for x in Supplier do
    for y in find(x.sid) do
        output(x, y);

If |R| = |S| = n,
what is the runtime?

O(n)

But can be O(n^2) why?

Why would we change the order?

When |
Supply| << |
Supplier|
Supplier($sid, sname, scity, sstate$)
Supply($sid, pno, quantity$)

3. Merge Join

Logical operator:

$$\text{Supplier} \bowtie_{sid=sid} \text{Supply}$$

Sort(Supplier); Sort(Supply);

x = Supplier.first();
y = Supply.first();
3. Merge Join

**Logical operator:**

Supplier ⨝ \( sid = sid \) Supply

Sort(Supplier); Sort(Supply);

\[ x = \text{Supplier.first}(); \]
\[ y = \text{Supply.first}(); \]

while \( y \neq \text{NULL} \) do

\[ \text{case:} \]
\[ x.sid < y.sid: ??? \]
\[ x.sid = y.sid: ??? \]
\[ x.sid > y.sid: ??? \]
3. Merge Join

Logical operator:

\[ \text{Supplier} \bowtie_{\text{sid}=\text{sid}} \text{Supply} \]

Sort(Supplier); Sort(Supply);

x = Supplier.first();
y = Supply.first();

while y != NULL do

  case:
    x.sid < y.sid: x = x.next()
    x.sid = y.sid: ???
    x.sid > y.sid: ???
3. Merge Join

Logical operator:

\[ \text{Supplier} \Join_{\text{sid} = \text{sid}} \text{Supply} \]

Sort(Supplier); Sort(Supply);

\( x = \text{Supplier} . \text{first}(); \)
\( y = \text{Supply} . \text{first}(); \)

while \( y \neq \text{NULL} \) do

\text{case:}

\( x . \text{sid} < y . \text{sid} : x = x . \text{next}(); \)
\( x . \text{sid} = y . \text{sid} : \text{output}(x,y); y = y . \text{next}(); \)
\( x . \text{sid} > y . \text{sid} : \text{???} \)
3. Merge Join

Logical operator:

\[
\text{Supplier} \Join_{\text{sid} = \text{sid}} \text{Supply}
\]

Sort(Supplier); Sort(Supply);

\[
x = \text{Supplier}.\text{first}();
\]

\[
y = \text{Supply}.\text{first}();
\]

while \(y \neq \text{NULL}\) do

\[
\text{case:}
\]

\[
x.\text{sid} < y.\text{sid}: x = x.\text{next}();
\]

\[
x.\text{sid} = y.\text{sid}: \text{output}(x,y); y = y.\text{next}();
\]

\[
x.\text{sid} > y.\text{sid}: y = y.\text{next}();
\]
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

3. Merge Join

Logical operator:

\[
\text{Supplier} \bowtie_{\text{sid}=\text{sid}} \text{Supply}
\]

Sort(Supplier); Sort(Supply);
\[x = \text{Supplier.first}();\]
\[y = \text{Supply.first}();\]
while \( y \neq \text{NULL} \) do
  case:
    \[x.\text{sid} < y.\text{sid}: x = x.\text{next}();\]
    \[x.\text{sid} = y.\text{sid}: \text{output}(x, y); y = y.\text{next}();\]
    \[x.\text{sid} > y.\text{sid}: y = y.\text{next}();\]

If \(|R| = |S| = n\), what is the runtime?
3. Merge Join

Logical operator:

\[ \text{Supplier} \Join_{\text{sid}=\text{sid}} \text{Supply} \]

Sort(Supplier); Sort(Supply);

\[ x = \text{Supplier}.\text{first}(); \]
\[ y = \text{Supply}.\text{first}(); \]

while \( y \neq \text{NULL} \) do

\[ \text{case:} \]
\[ \quad \text{x.sid < y.sid: } x = x.\text{next}(); \]
\[ \quad \text{x.sid = y.sid: } \text{output}(x,y); y = y.\text{next}(); \]
\[ \quad \text{x.sid > y.sid: } y = y.\text{next}(); \]

If \(|R|=|S|=n\), what is the runtime?

\[ O(n \log(n)) \]
Main Memory Algorithms

• Join ⨝:
  – Nested loop join
  – Hash join
  – Merge join

• Selection σ
  – “on-the-fly”
  – Index-based selection (next lecture)

• Group by γ
  – Hash-based
  – Merge-based
How Do We Combine Them?
How Do We Combine Them?

The *Iterator Interface*

- `open()`
- `next()`
- `close()`
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

```java
interface Operator {

    void open (...);
    // calls next() on its inputs
    // processes an input tuple
    // produces output tuple(s)
    Tuple next ();
    // cleans up (if any)
    void close ();
}
```
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

```java
interface Operator {

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

}
```
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

```java
interface Operator {

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

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

    void close () {
        child.close ();
    }
}
```

Example “on the fly” selection operator

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

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

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

Example “on the fly” selection operator

```java
interface Operator {

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

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

    // cleans up (if any)
    void close ();
}
```
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

```java
interface Operator {

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

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

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

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

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

    void close () {
        child.close ();
    }
}
```

Example “on the fly” selection operator
interface Operator {

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

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

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

class Select implements Operator {
    void open (Predicate p,
                Operator c) {
        this.p = p; this.c = c; c.open();
    }
    Tuple next () {
        // example "on the fly" selection operator
    }
}
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

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

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

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

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

    Tuple next () {
        boolean found = false;
        Tuple r = null;
        while (!found) {
            r = c.next();
            if (r == null) break;
            found = p(r);
        }
    }
}
```
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

```java
interface Operator {

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

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

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

class Select implements Operator {...
    void open (Predicate p,
               Operator c) {
        this.p = p; this.c = c; c.open();
    }
    Tuple next () {
        boolean found = false;
        Tuple r = null;
        while (!found) {
            r = c.next();
            if (r == null) break;
            found = p(r);
        }
        return r;
    }
}
```
Implementing Query Operators with the Iterator Interface

interface Operator {

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

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

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

Example “on the fly” selection operator

class Select implements Operator {

    void open (Predicate p, Operator c) {
        this.p = p; this.c = c; c.open();
    }

    Tuple next () {
        boolean found = false;
        Tuple r = null;
        while (!found) {
            r = c.next();
            if (r == null) break;
            found = p(r);
        }
        return r;
    }

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

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

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

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

/* Query plan execution */
Operator q = parse("SELECT ...");
q = optimize(q);
q.open();
while (true) {
    Tuple t = q.next();
    if (t == null) break;
    else printOnScreen(t);
}
q.close();
```
Supplier\((\text{sid}, \text{sname}, \text{scity}, \text{sstate})\)
Supply\((\text{sid}, \text{pno}, \text{quantity})\)

(On the fly)

\(\pi_{\text{sname}}\)

(On the fly)

\(\sigma_{\text{scity} = 'Seattle' \text{ and } \text{sstate} = 'WA' \text{ and } \text{pno} = 2}\)

(Nested loop)

\(\text{sid} = \text{sid}\)

Supplier (File scan)

Supply (File scan)

Discuss: open/next/close for nested loop join
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Pipelining

(On the fly)

open()

\[ \pi_{sname} \]

(On the fly)

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

(Nested loop)

sid = sid

Supplier (File scan)

Supply (File scan)

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

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

(On the fly)

\[
\text{open()}
\]

(On the fly)

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

(On the fly)

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

(Nested loop)

\[
\text{sid = sid}
\]

\[
\text{Supplier (File scan)}
\]

\[
\text{Supply (File scan)}
\]

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

(On the fly)

(On the fly)

(Nested loop)

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

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

\(\sigma_{\text{scity} = \text{'Seattle'} \text{ and sstate = 'WA' and pno=2}}\)

\(\pi_{\text{sname}}\)

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

(On the fly) 

\( \pi_{\text{name}} \)

(On the fly)

\( \sigma_{\text{city}=\text{Seattle} \text{ and state} = \text{WA} \text{ and } pno=2} \)

(Nested loop)

\( \text{Sid} = \text{Sid} \)

Supplier (File scan)

Supply (File scan)

Discuss: open/next/close for nested loop join
Supplier \((\text{sid, sname, scity, sstate})\)
Supply \((\text{sid, pno, quantity})\)

Pipelining

(On the fly)

(On the fly)

(Nested loop)

\(\pi_{\text{sname}}\)

\(\sigma_{\text{scity} = 'Seattle' \text{ and } \text{sstate} = 'WA' \text{ and } \text{pno}=2}\)

\(\text{sid} = \text{sid}\)

Discuss: open/next/close for nested loop join

Supplier (File scan)

Supply (File scan)
Supplier(\textit{sid, sname, scity, sstate})
Supply(\textit{sid, pno, quantity})

(On the fly)

(On the fly) \quad \pi_{\text{sname}}

(On the fly) \quad \sigma_{\text{scity}=\text{Seattle} \text{ and } \text{sstate}=\text{WA} \text{ and } \text{pno}=2}

(Nested loop)

sid = sid

Supplier (File scan)

Supply (File scan)

Discuss: open/next/close for nested loop join
Supplier\((sid, sname, scity, sstate)\)
Supply\((sid, pno, quantity)\)

Pipelining

(On the fly)

(On the fly)

(Nested loop)

\(\pi_{sname}\)

\(\sigma_{scity=\text{Seattle} \text{ and } sstate=\text{WA} \text{ and } pno=2}\)

\(\text{sid } = \text{ sid}\)

Supplier
(File scan)

Supply
(File scan)

Discuss: open/next/close for nested loop join
Nested loop (On the fly)

On the fly (On the fly)

Nested loop (Nested loop)

Supplier (File scan)

Supply (File scan)

Discuss: open/next/close for nested loop join

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

(On the fly)

\(\pi_{\text{name}}\)

(On the fly)
\(\sigma_{\text{scity} = 'Seattle' \text{ and } sstate = 'WA' \text{ and } pno = 2}\)

(Nested loop)

\(\text{sid} = \text{sid}\)

Suppliers (File scan)

Supply (File scan)

Discuss: open/next/close for nested loop join
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Pipelining

(On the fly)

(On the fly)

(On the fly)

(Nested loop)

Discuss: open/next/close for nested loop join

\(\pi_{sname}\)

\(\sigma_{\text{scity}= 'Seattle' \quad \text{and} \quad \text{sstate}= 'WA' \quad \text{and} \quad \text{pno}=2}\)

sid = sid

Suppliers (File scan)

Supply (File scan)

CSE 414 - 2019sp 53
Nested loop

On the fly

On the fly

On the fly

Next()

Next()

Next()

Next()

Next()

Next()

Next()

Next()

Next()

Next()

Next()

Next()

Next()

Discuss: open/next/close for nested loop join

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Pipelining

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

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

sid = sid

Supplier

(File scan)

Supply

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

Pipelining

(On the fly)

(On the fly)

(Hash Join)

Discuss hash-join in class

σ_{scity='Seattle' and sstate='WA' and pno=2}

π_{sname}

sid = sid

Supplier

(On the fly)

Supplier (File scan)

Supply

(On the fly)

Supply (File scan)
Discuss hash-join in class

(On the fly) 

(On the fly) 

(Hash Join) 

\[
\pi_{\text{fname}} \left\{ \sigma_{\text{scity} = \text{Seattle} \ \text{and} \ sstate = \text{WA} \ \text{and} \ pno = 2} \right\}
\]

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

Supplier (File scan) 

Supply (File scan) 

Tuples from here are “blocked”
Suppliers($sid$, $sname$, $scity$, $sstate$)
Supplies($sid$, $pno$, quantity)

Discuss hash-join in class

(On the fly) $\pi_{sname}$

(On the fly) $\sigma_{scity = 'Seattle' \land sstate = 'WA' \land pno = 2}$

(Hash Join) $sid = sid$

Tuples from here are pipelined

Tuples from here are "blocked"
Blocked Execution

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

(On the fly)

(On the fly)

\(\sigma_{\text{scity}='Seattle' \text{ and } \text{sstate}='WA' \text{ and } \text{pno}=2}\)

(Merge Join)

\(\pi_{\text{sname}}\)

Discuss merge-join in class

\(\text{sid} = \text{sid}\)

Supplier (File scan)
Supply (File scan)
Blocked Execution

(On the fly)

(On the fly) $\sigma_{\text{scity} = \text{Seattle} \text{ and } \text{sstate} = \text{WA} \text{ and } \text{pno} = 2}$

(Merge Join)

Discuss merge-join in class

Supplier (sid, sname, scity, sstate)
Supply (sid, pno, quantity)
Pipeline v.s. Blocking

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

• Blocking
  – The entire result of the subplan is computed (and stored to disk) before the first tuple is sent up the plan
  – Advantage: saves memory
  – Disadvantage: slower
Introduction to Database Systems
CSE 344

Lecture 16:
Basics of Data Storage and Indexes
Query Performance

To understand query performance and query optimization we need to understand:

• How is data organized on disk
• How to estimate query costs

We focus on disk-based DBMSs
Hard Disk

- Disks are mechanical devices
- A block = unit of read/write
- Once in main memory we call it a page
- Read only at the rotation speed
- Consequence: sequential scan faster than random
  - Fast: read blocks 1,2,3,4,5,…
  - Slow: read blocks 2342, 11, 321,9, …

- Rule of thumb:
  - Random read 1-2% of file ≈ sequential scan entire file;
  - 1-2% decreases over time, because of increased density
Data Storage

- DBMSs store data in **files**
- Most common organization is row-wise storage
- On disk, a file is split into blocks
- Each block contains a set of tuples

<table>
<thead>
<tr>
<th>ID</th>
<th>fName</th>
<th>lName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Tom</td>
<td>Hanks</td>
</tr>
<tr>
<td>20</td>
<td>Amy</td>
<td>Hanks</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>Tom</td>
<td>Hanks</td>
</tr>
<tr>
<td>20</td>
<td>Amy</td>
<td>Hanks</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>220</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>240</td>
<td></td>
<td></td>
</tr>
<tr>
<td>420</td>
<td></td>
<td></td>
</tr>
<tr>
<td>800</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the example, we have 4 blocks with 2 tuples each.
Data File Types

The data file can be one of:

- **Heap file**
  - Unsorted

- **Sequential file**
  - Sorted according to some attribute(s) called *key*

<table>
<thead>
<tr>
<th>ID</th>
<th>fName</th>
<th>lName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Tom</td>
<td>Hanks</td>
</tr>
<tr>
<td>20</td>
<td>Amy</td>
<td>Hanks</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Index

• An **additional** file, that allows fast access to records in the data file given a search key
Index

- An **additional** file, that allows fast access to records in the data file given a search key

- The index contains (key, value) pairs:
  - Key = an attribute value (e.g., student ID or name)
  - Value = a pointer to the record OR the record itself
Index

• An **additional** file, that allows fast access to records in the data file given a search key

• The index contains (key, value) pairs:
  – Key = an attribute value (e.g., student ID or name)
  – Value = a pointer to the record OR the record itself

• Could have many indexes for one table

Key = means here search key
This Is Not A Key

Different keys:

• **Primary key** – uniquely identifies a tuple
• **Key of the sequential file** – how the data file is sorted, if at all
• **Index key** – how the index is organized
Example 1: Index on ID

Index **Student_ID** on **Student.ID**

Data File **Student**

<table>
<thead>
<tr>
<th>ID</th>
<th>fName</th>
<th>lName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Tom</td>
<td>Hanks</td>
</tr>
<tr>
<td>20</td>
<td>Amy</td>
<td>Hanks</td>
</tr>
</tbody>
</table>

Index can be:

- **Dense** = one entry per record
- **Sparse** = one entry per block
Example 2: 
Index on fName

Index `Student_fName` on `Student.fName`

Data File `Student`

<table>
<thead>
<tr>
<th>ID</th>
<th>fName</th>
<th>lName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Tom</td>
<td>Hanks</td>
</tr>
<tr>
<td>20</td>
<td>Amy</td>
<td>Hanks</td>
</tr>
</tbody>
</table>

Index can be:
- Dense only
Index Organization

• Hash table

• B+ trees – most common
  – They are search trees, but they are not binary instead have higher fan-out
  – Will discuss them briefly next

• Specialized indexes: bit maps, R-trees, inverted index; won’t discuss
B+ Tree Index by Example

\(d = 2\)

Find the key 40

\[
\begin{array}{c}
10 & 15 & 18 \\
20 & 30 & 40 & 50 \\
60 & 65 \\
80 & 85 & 90 \\
10 & 15 & 18 \\
20 & 30 & 40 & 50 \\
60 & 65 \\
80 & 85 & 90
\end{array}
\]
Every table can have **only one** clustered and **many** unclustered indexes.

Why?
Index Classification

• **Clustered/unclustered**
  – Clustered = records close in index are close in data
    • Option 1: Data inside data file is sorted on disk
    • Option 2: Store data directly inside the index (no separate files)
  – Unclustered = records close in index may be far in data
Index Classification

• **Clustered/unclustered**
  – Clustered = records close in index are close in data
    • Option 1: Data inside data file is sorted on disk
    • Option 2: Store data directly inside the index (no separate files)
  – Unclustered = records close in index may be far in data

• **Primary/secondary**
  – Meaning 1:
    • Primary = is over attributes that include the primary key
    • Secondary = otherwise
  – Meaning 2: means the same as clustered/unclustered
Index Classification

• **Clustered/unclustered**
  – Clustered = records close in index are close in data
    • Option 1: Data inside data file is sorted on disk
    • Option 2: Store data directly inside the index (no separate files)
  – Unclustered = records close in index may be far in data

• **Primary/secondary**
  – Meaning 1:
    • Primary = is over attributes that include the primary key
    • Secondary = otherwise
  – Meaning 2: means the same as clustered/unclustered

• **Organization** B+ tree or Hash table
Summary So Far

- Index = a file that enables direct access to records in another data file
  - B+ tree / Hash table
  - Clustered/unclustered

- Data resides on disk
  - Organized in blocks
  - Sequential reads are efficient
  - Random access less efficient
  - Random read 1-2% of data worse than sequential
Main Memory Algorithms

• Selection $\sigma$
  – “on-the-fly”
  – Index-based selection

• Join $\Join$
  – Nested loop join
  – Hash join
  – Merge join
  – Index join
Student(ID, fname, lname)
Takes(studentID, courseID)

**Selection**

```
SELECT *
FROM Takes y
WHERE y.courseID = 300
```

Logical plan:

```
σ^{300}
Takes
```
Student(ID, fname, lname)
Takes(studentID, courseID)

```
SELECT *
FROM Takes y
WHERE y.courseID = 300
```

On-the-fly selection

Selection

Logical plan

On-the-fly \( \sigma_{300} \)

\( \sigma_{300} \)

Physical plan

\( \sigma_{300} \)
Student(ID, fname, lname)
Takes(studentID, courseID)

**Selection**

```sql
SELECT *
FROM Takes y
WHERE y.courseID = 300
```

On-the-fly selection

Logical plan

```
σ_{300} \\
Takes
```

Physical plan

```
for y in Takes
  if courseID = 300 then
    output y
```
Selection

On-the-fly selection

Index-based selection:

\[ \text{Takes\_courseID} = \text{index on Takes.courseID} \]

Logical plan:

\[ \sigma_{\text{courseID} = 300} \]

Physical plan:

\[ \text{for } y \text{ in Takes} \]
\[ \text{if courseID = 300 then output } y \]
Student(ID, fname, lname)
Takes(studentID, courseID)

SELECT *
FROM Takes y
WHERE y.courseID = 300

Selection

Logical plan
\[ \sigma_{300} \]
\[ \text{Takes} \]

Physical plan
\[ \text{for } y \text{ in Takes} \]
\[ \text{if } \text{courseID} = 300 \text{ then} \]
\[ \text{output } y \]

On-the-fly selection

Index-based selection:
\[ \text{Takes.courseID} = \]
index on Takes.courseID

Index-based
\[ \sigma_{300} \]
\[ \text{Takes} \]

Index-based
rid_list = Takes_courseID.get(300)
for r in rid_list
y = Student.getRecord(rid)
output y
Student(ID, fname, lname)
Takes(studentID, courseID)

**Selection**

On-the-fly selection

Index-based selection:
\[ \text{Takes\_courseID} = \text{index on Takes.courseID} \]

**Logical plan**

\[ \sigma_{300} \]

\[ \text{Takes} \]

**Physical plan**

\[ \text{for } y \text{ in Takes} \]
\[ \text{if courseID} = 300 \text{ then} \]
\[ \text{output } y \]

Index-based selection:
\[ \text{Takes\_courseID} = \text{index on Takes.courseID} \]

\[ \sigma_{300} \]

\[ \text{Takes} \]

\[ \Rightarrow \]

\[ \sigma_{300} \]

\[ \text{Takes\_courseID} \]

\[ \Rightarrow \]

\[ \sigma_{300} \]

\[ \text{Takes} \]

\[ \Rightarrow \]

\[ \sigma_{300} \]

\[ \text{Takes\_courseID} \]

SQL Server represents index-based selection as a join

\[ \text{rid\_list} = \text{Takes\_courseID.get(300)} \]
\[ \text{for } r \text{ in rid\_list} \]
\[ y = \text{Student.getRecord(rid)} \]
\[ \text{output } y \]
Discussion

Can the optimizer use the index Takes_courseID to answer these queries?

\[
\begin{align*}
\text{SELECT} & \ *
\text{FROM} & \ Takes\ y
\text{WHERE} & \ y.\text{courseID} = 300 \text{ or } y.\text{courseID} = 444
\end{align*}
\]

\[
\begin{align*}
(\text{SELECT} & \ *
\text{FROM} & \ Takes\ y
\text{WHERE} & \ y.\text{courseID} = 300)
\text{UNION ALL}
(\text{SELECT} & \ *
\text{FROM} & \ Takes\ y
\text{WHERE} & \ y.\text{courseID} = 444)
\end{align*}
\]
Can the optimizer use the index Takes_courseID to answer these queries?

SELECT *
FROM Takes y
WHERE y.courseID = 300 or y.courseID = 444

(SELECT *
FROM Takes y
WHERE y.courseID = 300)
UNION ALL
(SELECT *
FROM Takes y
WHERE y.courseID = 444)

Probably not

Recall HW3!!

Yes
Join

Nested Loop Join:

SELECT *
FROM  Student x, Takes y
WHERE x.ID=y.studentID AND y.courseID = 300

for y in Takes
  if courseID = 300 then
    for x in Student
    if x.ID=y.studentID
    output *
Student(ID, fname, lname)  
Takes(studentID, courseID)

SELECT *  
FROM Student x, Takes y  
WHERE x.ID=y.studentID AND y.courseID = 300

Join

Nested Loop Join:

Index Join:

assume the database has these indexes
- Takes_courseID = on Takes.courseID
- Student_ID = on Student.ID
Join

Nested Loop Join:

\[
\text{for } y \text{ in Takes } \\
\quad \text{if } \text{courseID} = 300 \text{ then } \\
\quad \quad \text{for } x \text{ in Student } \\
\quad \quad \quad \text{if } x.\text{ID} = y.\text{studentID} \\
\quad \quad \quad \text{output } * \\
\]

Index Join:

assume the database has these indexes

- Takes\_courseID = on Takes.courseID
- Student\_ID = on Student.ID

\[
\text{for } y' \text{ in Takes\_courseID.get(300) } \\
\quad y = \text{Takes.getRecord}(y') \\
\]
Join

Nested Loop Join:

```
for y in Takes
    if courseId = 300 then
        for x in Student
            if x.ID = y.studentID
                output *
```

Index Join:

```
for y’ in Takes_courseID.get(300)
y = Takes.getRecord(y’)
```

assume the database has these indexes
- **Takes_courseID** = on Takes.courseID
- **Student_ID** = on Student.ID
SELECT *
FROM Student x, Takes y
WHERE x.ID = y.studentID AND y.courseID = 300

Join

Nested Loop Join:

Index Join:

assumes the database has these indexes
- Takes_courseID = on Takes.courseID
- Student_ID = on Student.ID
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N text, P int);

CREATE INDEX V1 ON V(N)
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N text, P int);

CREATE INDEX V1 ON V(N);

CREATE INDEX V2 ON V(P, M);
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N text, P int);

CREATE INDEX V1 ON V(N);

CREATE INDEX V2 ON V(P, M);

What does this mean?
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N text, P int);

CREATE INDEX V1 ON V(N)

CREATE INDEX V2 ON V(P, M)

What does this mean?

select * from V where P=55 and M=77

yes
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N text, P int);
CREATE INDEX V1 ON V(N);
CREATE INDEX V2 ON V(P, M);

What does this mean?

select * from V where P=55
select * from V where P=55 and M=77

yes
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N text, P int);

CREATE INDEX V1 ON V(N);

CREATE INDEX V2 ON V(P, M);

What does this mean?

select *
from V
where P=55 and M=77

select *
from V
where P=55

yes
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N text, P int);

CREATE INDEX V1 ON V(N)

CREATE INDEX V2 ON V(P, M)

What does this mean?

select *
from V
where P=55 and M=77

select *
from V
where P=55

select *
from V
where M=77

yes
Getting Practical: Creating Indexes in SQL

```sql
CREATE TABLE V(M int, N text, P int);

CREATE INDEX V1 ON V(N);
CREATE INDEX V2 ON V(P, M);
```

What does this mean?

```sql
select *
from V
where P=55
```
```
select *
from V
where M=77
```
```
select *
from V
where P=55 and M=77
```

- Yes
- No
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N text, P int);
CREATE INDEX V1 ON V(N);
CREATE INDEX V2 ON V(P, M);
CREATE INDEX V3 ON V(M, N);
CREATE UNIQUE INDEX V4 ON V(N);
CREATE CLUSTERED INDEX V5 ON V(N);

What does this mean?

select * from V where P=55
select * from V where M=77
select * from V where P=55 and M=77

yes
yes
no

Not supported in SQLite
Which Indexes?

• How many indexes **could** we create?

• Which indexes **should** we create?
Which Indexes?

• How many indexes could we create?

• Which indexes should we create?

This is called the **Index Selection Problem**

(not to be confused with the **index selection** operator!)
The Index Selection Problem 1

V(M, N, P);

Your workload is this

100000 queries:

SELECT *
FROM V
WHERE N=?

100 queries:

SELECT *
FROM V
WHERE P=?
The Index Selection Problem 1

V(M, N, P);

Your workload is this

100000 queries:

SELECT * FROM V WHERE N=?

100 queries:

SELECT * FROM V WHERE P=?

What indexes?
The Index Selection Problem 1

V(M, N, P);

Your workload is this

100000 queries:

100 queries:

A: V(N) and V(P) (hash tables or B-trees)
The Index Selection Problem 2

V(M, N, P);

Your workload is this

100000 queries:
SELECT * 
FROM V 
WHERE N>? and N<?

100 queries:
SELECT * 
FROM V 
WHERE P=?

100000 queries:
INSERT INTO V 
VALUES (?, ?, ?)

What indexes?
The Index Selection Problem 2

V(M, N, P);

Your workload is this

100000 queries:

SELECT * FROM V WHERE N>? and N<?

100 queries:

SELECT * FROM V WHERE P=?

100000 queries:

INSERT INTO V VALUES (?, ?, ?)

A: definitely V(N) (must B-tree); unsure about V(P)
The Index Selection Problem 3

V(M, N, P);

Your workload is this

100000 queries: SELECT * FROM V WHERE N=?
1000000 queries: SELECT * FROM V WHERE N=? and P>?
100000 queries: INSERT INTO V VALUES (?, ?, ?)

What indexes?
The Index Selection Problem 3

V(M, N, P);

Your workload is this

100000 queries:

SELECT *
FROM V
WHERE N=?.

1000000 queries:

SELECT *
FROM V
WHERE N=? and P>?.

100000 queries:

INSERT INTO V VALUES (?, ?, ?).

A: V(N, P)

How does this index differ from:

1. Two indexes V(N) and V(P)?
2. An index V(P, N)?
The Index Selection Problem 4

V(M, N, P);

Your workload is this

1000 queries:

SELECT * FROM V WHERE N>? and N<?

100000 queries:

SELECT * FROM V WHERE P>? and P<?

What indexes?
The Index Selection Problem 4

\[ V(M, N, P); \]

Your workload is this

1000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100000 queries:

```
SELECT *
FROM V
WHERE P>? and P<?
```

A: V(N) secondary, V(P) primary index
Two typical kinds of queries

- Point queries
  - Hash- or B\(^+\)-tree index
  - Clustered or not

- Range queries
  - B\(^+\)-tree index
  - Clustered
To Cluster or Not

Remember:

- **Rule of thumb:**
  Random reading 1-2% of file $\approx$ sequential scan entire file;

Range queries benefit mostly from clustering because they may read more than 1-2%
SELECT * 
FROM R 
WHERE R.K>? and R.K<?
SELECT *
FROM R
WHERE R.K>? and R.K<?
SELECT * 
FROM R 
WHERE R.K>? and R.K<?
SELECT * 
FROM R 
WHERE R.K>? and R.K<?
Summary of Physical Plan

More components of a physical plan:

- **Access path selection** for each relation
  - Scan the relation or use an index
- **Implementation choice** for each operator
  - Nested loop join, hash join, etc.
- **Scheduling decisions** for operators
  - Pipelined execution or intermediate materialization
Introduction to Database Systems
CSE 344

Lecture 17:
Basics of Query Optimization and
Query Cost Estimation
Query Optimization

• Main idea: replace a query plan with another one that is equivalent, but cheaper

• Algebraic identities of the relational algebra

• Will discuss:
  1. Pushing selections down
  2. Join reorder
Push Selections Down

$$\Pi_{x.\text{sname}}$$

$$\sigma_{x.\text{scity}=\text{Seattle}}$$

$$\bowtie_{x.\text{sid} = y.\text{sid}}$$
Push Selections Down

\[
\Pi_{x.\text{sname}} \left( \sigma_{x.\text{scity} = 'Seattle'} \left( \bowtie_{x.\text{sid} = y.\text{sid}} \left( \Pi_{x.\text{sname}} \text{Supplier} \ x \right) \right) \right)
\]
Push Selections Down

\[
\begin{align*}
\Pi_{x.\text{sname}}(x.\text{scity} = \text{'Seattle'}) \\
\bowtie x.\text{sid} = y.\text{sid} \\
\sigma_{x.\text{scity} = \text{'Seattle'}} \end{align*}
\]

\[
\begin{align*}
\Pi_{x.\text{sname}}(x.\text{scity} = \text{'Seattle'}) \\
\bowtie x.\text{sid} = y.\text{sid} \\
\sigma_{x.\text{scity} = \text{'Seattle'}} \end{align*}
\]

\[
\sigma_{C}(R \bowtie S) = \sigma_{C}(R) \bowtie S \quad \text{when } C \text{ refers only to } R
\]
Push Selections Down

\[ \Pi_{x.\text{sname}} \]
\[ \sigma_{x.\text{scity} = 'Seattle'} \text{ and } y.\text{pno} = 5 \]
\[ \bowtie_{x.\text{sid} = y.\text{sid}} \]

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

\[
\left( \prod_{x.sname} \right) \left( \sigma_{x.scity='Seattle'} \land y.pno=5 \right) \left( \times_{x.sid = y.sid} \right) \left( \text{Supplier } x \right) \left( \times_{x.sid = y.sid} \right) \left( \text{Supply } y \right)
\]

\[
\left( \prod_{x.sname} \right) \left( \sigma_{x.scity='Seattle'} \right) \left( \times_{x.sid = y.sid} \right) \left( \text{Supplier } x \right) \left( \times_{x.sid = y.sid} \right) \left( \text{Supply } y \right)
\]

\[
\sigma_{y.pno=5}
\]
Push Selections Down

$\Pi_{x.\text{sname}}$ $\sigma_{x.\text{scity}=\text{‘Seattle’}}$ and $y.\text{pno}=5$

$\bowtie_{x.\text{sid} = y.\text{sid}}$ $\sigma_{x.\text{scity}=\text{‘Seattle’}}$

$\sigma_{y.\text{pno}=5}$

$\Pi_{x.\text{sname}}$

$\sigma_{C_1 \text{ and } C_2}(R \bowtie S) = \sigma_{C_1}(\sigma_{C_2}(R \bowtie S)) = \sigma_{C_1}(R \bowtie \sigma_{C_2}(S)) = \sigma_{C_1}(R) \bowtie \sigma_{C_2}(S)$
Join Reorder

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder

Supplier x ⋈ x.sid = y.sid ⋈ y.pno = z.pno ⋈ Part z

Supplier y ⋈ y.pno = z.pno ⋈ Part z

Supplier x ⋈ x.sid = y.sid ⋈ Supply y

Join Reorder
Join Reorder

Supplier\((sid, sname, scity, sstate)\)
Supply\((sid, pno, quantity)\)
Part\((pno, pname, pprice)\)

\((R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)\)
Join Reorder

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)

Also:
R \bowtie S = S \bowtie R
Join Reorder

When is one plan better than the other?

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

Also:

\[R \bowtie S = S \bowtie R\]
Join Reorder

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

\( \Join_{x \cdot \text{sid} = y \cdot \text{sid}} \)
\( \Join_{y \cdot \text{pno} = z \cdot \text{pno}} \)
\( \sigma_{x \cdot \text{scity}= \text{Seattle}} \)
\( \sigma_{z \cdot \text{pprice} > 99} \)
Join Reorder

When is one plan better than the other?

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

\[
\text{Join } x \bowtie y : y.pno = z.pno \\
\text{Join } x : x.sid = y.sid \\
\text{Join } y : y.pno = z.pno
\]

\[
\text{σ}_{x.scity='Seattle'} \\
\text{σ}_{z.pprice > 99} \\
\text{σ}_{x.scity='Seattle'} \\
\text{σ}_{z.pprice > 99}
\]

\[
\text{Supplier x} \\
\text{Supply y} \\
\text{Part z} \\
\text{Supplier x} \\
\text{Supply y} \\
\text{Part z}
\]
Join Reorder

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

When is one plan better than the other?

Lesson: need sizes of $\sigma_{x.scity='Seattle'}(Supplier)$, $\sigma_{z.pprice > 99}(Part)$
Size and Cost Estimation

Given statistics on the base tables:

- $B(R) = \#$ of blocks (i.e., pages) for relation $R$
- $T(R) = \#$ of tuples in relation $R$
- $V(R, A) = \#$ of distinct values of attribute $A$

Size estimation: estimate the size of a logical subplan
Cost estimation: estimate the cost of a physical subplan
Size Estimation

Problem: estimate the size of a query plan: $|P|$
We consider plans with selections and joins

Worst case sizes:

- Size of a selection: $|\sigma_C(R)| \leq |R|$
- Size of a join: $|R \bowtie S| \leq |R| \times |S|$

Estimate $\approx f \times \text{worst-case}$
where $f$ in $(0,1)$ is called selectivity factor
Estimating Size of a Selection

Assumption 1: uniform distribution of values
- $|\sigma_{A=v}(R)| \approx |T(R)| / V(R,A)$
- Selectivity factor: $f_{A=v} = 1/V(R,A)$

Assumption 2: independence of attributes
- Selectivity factor: $f_{A=v \text{ and } B=w} = f_{A=v} \cdot f_{B=w}$
- $|\sigma_{A=v \text{ and } B=w}(R)| \approx |T(R)| / (V(R,A) \cdot V(R,B))$
Estimating Size of a Join

**Assumption 3:** Inclusion assumption

if $V(R,B) \leq V(S,C)$ then $\Pi_B(R) \subseteq \Pi_C(S)$

- $|R \bowtie S| \approx |R| \times |S| / V(S,C)$

In general:

- $|R \bowtie S| \approx |R| \times |S| / \max(V(R,B),V(S,C))$
Supplier($sid$, $sname$, $scity$, $sstate$)
Supply($sid$, $pno$, quantity)
Part($pno$, $pname$, $pprice$)

### Example

```sql
SELECT *
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(Supplier)$</td>
<td>$100,000$</td>
</tr>
<tr>
<td>$T(Supply)$</td>
<td>$3,000,000$</td>
</tr>
<tr>
<td>$V(Supply,sid)$</td>
<td>$60,000$</td>
</tr>
<tr>
<td>$V(Supply,pno)$</td>
<td>$25,000$</td>
</tr>
<tr>
<td>$T(Part)$</td>
<td>$50,000$</td>
</tr>
</tbody>
</table>
SELECT *
FROM Supplier x, Supply y
WHERE x.sid = y.sid

\[
|Q| = \frac{T(\text{Supplier}) \times T(\text{Supply})}{\max(V(\text{Supplier}, \text{sid}), V(\text{Supply}, \text{sid}))}
\]

T(\text{Supplier}) = 100,000
T(\text{Supply}) = 3,000,000
V(\text{Supply}, \text{sid}) = 60,000
V(\text{Supply}, \text{pno}) = 25,000
T(\text{Part}) = 50,000
Example

SELECT *
FROM Supplier x, Supply y
WHERE x.sid = y.sid

|Q| = T(Supplier) * T(Supply) / max(V(Supplier,sid), V(Supply,sid))
   = 100,000 * 3,000,000 / 100,000

T(Supplier) = 100,000
T(Supply) = 3,000,000
V(Supplier,sid) = 60,000
V(Supply,pno) = 25,000
T(Part) = 50,000
Example

```
SELECT *
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

|Q| = \( T(Supplier) \times T(Supply) / \max(V(Supplier, sid), V(Supply, sid)) \)
  = 100,000 \times 3,000,000 / 100,000
  = 3,000,000

\[
\begin{align*}
T(Supplier) &= 100,000 \\
T(Supply) &= 3,000,000 \\
V(Supply, sid) &= 60,000 \\
V(Supply, pno) &= 25,000 \\
T(Part) &= 50,000
\end{align*}
\]
Example

**SELECT** *
**FROM** Supplier x, Supply y
**WHERE** x.sid = y.sid

\[ |Q| = \frac{T(\text{Supplier}) \times T(\text{Supply})}{\max(V(\text{Supplier}, \text{sid}), V(\text{Supply}, \text{sid}))} \]
\[ = \frac{100,000 \times 3,000,000}{100,000} \]
\[ = 3,000,000 \]

This is obvious!! Why?
Example

```
SELECT *
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
    and z.price = 30
```

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

T(Supplier) = 100,000
V(Supplier,city) = 2000
V(Supplier,state) = 50
T(Supply) = 3,000,000
V(Supply,sid) = 60,000
V(Supply,pno) = 25,000
T(Part) = 50,000
V(Part,price) = 5000
Example

```sql
SELECT *
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
    and z.price = 30
```

Q = \frac{T(Supply)}{V(Supplier,city) \times V(Supplier, state) \times V(Part, price)}

T(Supplier) = 100,000
V(Supplier, city) = 2000
V(Supplier, state) = 50

T(Supply) = 3,000,000
V(Supply, sid) = 60,000
V(Supply, pno) = 25,000

T(Part) = 50,000
V(Part, price) = 5000
Example

```
SELECT *
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
    and z.price = 30
```

\[
Q = \frac{T(\text{Supply})}{V(\text{Supplier,city}) \cdot V(\text{Supplier,state}) \cdot V(\text{Part,price})}
\]

\[
= \frac{3,000,000}{(2000 \cdot 50 \cdot 5000)} < 1
\]
Optimization

- The optimizer considers several plans
- For each plan, it estimates costs
- Then chooses the cheapest plan

Cost estimation: we will consider only the I/O cost.
I/O Cost of Physical Operators
Cost Parameters

Given statistics on the base tables:

- \( B(R) \) = # of blocks (i.e., pages) for relation \( R \)
- \( T(R) \) = # of tuples in relation \( R \)
- \( V(R, A) \) = # of distinct values of attribute \( A \)
I/O Cost of Selection

• Sequential scan for relation R costs $B(R)$

• Index-based selection
  – Estimate selectivity factor $f$
  – Clustered index: $f^*B(R)$
  – Unclustered index $f^*T(R)$

Note: we ignore I/O cost for index pages
Example

- Table scan:
- Index based selection:

\[
\begin{align*}
  B(R) &= 2000 \\
  T(R) &= 100,000 \\
  V(R, A) &= 20
\end{align*}
\]

\[
\text{cost of } \sigma_{A=v}(R) = ?
\]
Example

- Table scan: $B(R) = 2,000$ I/Os
- Index based selection:

\[
\begin{align*}
B(R) &= 2000 \\
T(R) &= 100,000 \\
V(R, A) &= 20
\end{align*}
\]

\[\text{cost of } \sigma_{A=v}(R) = ?\]
Example

- Table scan: $B(R) = 2000$ I/Os
- Index based selection:
  - If index is unclustered: $T(R) * 1/V(R,A) = 5,000$ I/Os

\[
\begin{align*}
B(R) &= 2000 \\
T(R) &= 100,000 \\
V(R, A) &= 20
\end{align*}
\]

Cost of $\sigma_{A=v}(R) =$ ?
Example

- Table scan: \( B(R) = 2,000 \) I/Os
- Index based selection:
  - If index is unclustered: \( T(R) \times 1/V(R,A) = 5,000 \) I/Os
  - If index is clustered: \( B(R) \times 1/V(R,A) = 100 \) I/Os

\[ \text{cost of } \sigma_{A=v}(R) = ? \]
Example

\[
\begin{align*}
B(R) &= 2000 \\
T(R) &= 100,000 \\
V(R, A) &= 20
\end{align*}
\]

- Table scan: \( B(R) = 2,000 \) I/Os

- Index based selection:
  - If index is unclustered: \( T(R) \times \frac{1}{V(R,A)} = 5,000 \) I/Os
  - If index is clustered: \( B(R) \times \frac{1}{V(R,A)} = 100 \) I/Os

Lesson: Don’t build unclustered indexes when \( V(R,A) \) is small!
CSE 414, Spring 2019:

• We will not cover the I/O cost of a join

• Skip slides until “Cost of a query plan”

• Study the size estimate of the logical plan.
I/O Cost of a Join

- Nested loop join
- Hash join
- Sort-merge join
- Index-join

Read: sections 15.2, 15.3, 15.6
Join Example

Patient(pid, name, address)
Insurance(pid, provider, policy_nb)
Patient \( \bowtie \) Insurance

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 'Bob'</td>
<td>2 'Blue'</td>
</tr>
<tr>
<td>2 'Ela'</td>
<td>4 'Prem'</td>
</tr>
<tr>
<td>3 'Jill'</td>
<td>4 'Prem'</td>
</tr>
<tr>
<td>4 'Joe'</td>
<td>3 'GrpH'</td>
</tr>
</tbody>
</table>

Two tuples per page
Nested Loop Joins

• Tuple-based nested loop $R \bowtie S$
• $R$ is the outer relation, $S$ is the inner relation

```
for each tuple $t_1$ in $R$ do
  for each tuple $t_2$ in $S$ do
    if $t_1$ and $t_2$ join then output ($t_1$, $t_2$)
```

What is the Cost?
Nested Loop Joins

• Tuple-based nested loop $R \bowtie S$
• $R$ is the outer relation, $S$ is the inner relation

```
for each tuple $t_1$ in $R$ do
  for each tuple $t_2$ in $S$ do
    if $t_1$ and $t_2$ join then output ($t_1, t_2$)
```

• Cost: $B(R) + T(R) B(S)$
• Multiple-pass since $S$ is read many times
Page-at-a-time Refinement

for each page of tuples \( r \) in \( R \) do
  for each page of tuples \( s \) in \( S \) do
    for all pairs of tuples \( t_1 \) in \( r \), \( t_2 \) in \( s \)
      if \( t_1 \) and \( t_2 \) join then output \((t_1,t_2)\)

• Cost: \( B(R) + B(R)B(S) \)
Page-at-a-time Refinement

Disk

Patient | Insurance
---|---
1 2 | 2 4
3 4 | 4 3
9 6 | 2 8
8 5 | 8 9

Input buffer for Patient

Input buffer for Insurance

Output buffer
Page-at-a-time Refinement

Disk

Patient  Insurance

1 2  2 4  6 6
3 4  4 3  1 3
9 6  2 8
8 5  8 9

1 2  Input buffer for Patient
4 3  Input buffer for Insurance

Output buffer
Page-at-a-time Refinement

Disk

Patient

1 2
3 4
9 6
8 5

Insurance

2 4 6 6
4 3 1 3
2 8
8 9

Input buffer for Patient

Input buffer for Insurance

1 2
2 8

Keep going until read all of Insurance

Then repeat for next page of Patient… until end of Patient

Output buffer

2 2

Cost: $B(R) + B(R)B(S)$
Block-Nested-Loop Refinement

\[
\text{for each group of M-1 pages } r \text{ in } R \text{ do}
\]
\[
\text{for each page of tuples } s \text{ in } S \text{ do}
\]
\[
\text{for all pairs of tuples } t_1 \text{ in } r, t_2 \text{ in } s
\]
\[
\text{if } t_1 \text{ and } t_2 \text{ join then output } (t_1, t_2)
\]

- Cost: \( B(R) + \frac{B(R)B(S)}{(M-1)} \)

What is the Cost?
Hash Join

Hash join: \( R \bowtie S \)
- Scan \( R \), build buckets in main memory
- Then scan \( S \) and join
- Cost: \( B(R) + B(S) \)
- Which relation to build the hash table on?
Hash Join

Hash join: \( R \bowtie S \)

- Scan \( R \), build buckets in main memory
- Then scan \( S \) and join
- Cost: \( B(R) + B(S) \)
- Which relation to build the hash table on?

- One-pass algorithm when \( B(R) \leq M \)
  - \( M \) = number of memory pages available
Hash Join Example

Patient $\bowtie$ Insurance

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4 6 6</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3 1 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8 8 9</td>
</tr>
<tr>
<td>8 5</td>
<td></td>
</tr>
</tbody>
</table>

Memory M = 21 pages

Showing pid only

Some large-enough #

This is one page with two tuples
Hash Join Example

Step 1: Scan Patient and build hash table in memory
Can be done in method open()

Memory M = 21 pages
Hash h: pid % 5

Disk

Patient      Insurance
1  2          2  4
3  4          4  3
9  6          2  8
8  5          8  9

Input buffer
Hash Join Example

Step 2: Scan Insurance and probe into hash table
Done during calls to next()

Memory M = 21 pages
Hash h: pid % 5

Disk

Patient | Insurance
---|---
1 2
3 4
9 6
8 5

Input buffer

Output buffer

Write to disk or pass to next operator
Hash Join Example

Step 2: Scan Insurance and probe into hash table

Done during calls to next()

Memory $M = 21$ pages

Hash $h: \text{pid} \mod 5$

<table>
<thead>
<tr>
<th>Disk</th>
<th>Input buffer</th>
<th>Output buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 1 6 2</td>
<td>2 4</td>
<td>4 4</td>
</tr>
</tbody>
</table>
Hash Join Example

Step 2: Scan Insurance and probe into hash table
Done during calls to next()

Memory M = 21 pages
Hash h: pid % 5

Input buffer

Output buffer

Keep going until read all of Insurance

Cost: B(R) + B(S)
Sort-Merge Join

Sort-merge join:  $R \bowtie S$

- Scan $R$ and sort in main memory
- Scan $S$ and sort in main memory
- Merge $R$ and $S$

- Cost: $B(R) + B(S)$
- One pass algorithm when $B(S) + B(R) \leq M$
- Typically, this is NOT a one pass algorithm
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory

Memory M = 21 pages

Patient | Insurance
--- | ---
1 2 | 2 4 6 6
3 4 | 4 3 1 3
9 6 | 2 8
8 5 | 8 9
Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory

Memory M = 21 pages

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4 6 6</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3 1 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8 8 9</td>
</tr>
<tr>
<td>8 5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4</td>
</tr>
<tr>
<td>5 6 8 9</td>
</tr>
<tr>
<td>1 2 2 3</td>
</tr>
<tr>
<td>3 4 4 6</td>
</tr>
<tr>
<td>6 8 8 9</td>
</tr>
</tbody>
</table>
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Disk

Patient

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

Insurance

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Memory M = 21 pages

Output buffer

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Memory $M = 21$ pages

Output buffer

Keep going until end of first relation
Index Join

\[ \text{R } \bowtie \text{ S} \]

- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S

- **Cost:**
  - If index on S is clustered:
    \[ B(R) + T(R) \times (B(S) \times \frac{1}{V(S,a)}) \]
  - If index on S is unclustered:
    \[ B(R) + T(R) \times (T(S) \times \frac{1}{V(S,a)}) \]
Cost of Query Plans
Example
Logical Query Plan 1

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

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

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

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

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

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

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

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

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

M = 11
Logical Query Plan 1

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

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

\[ T < 1 \]

\[ T = 10000 \]

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

\[ \text{SELECT sname} \]

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

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

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

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

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

\[ T(\text{Supply}) = 10000 \]

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

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

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

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

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

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

\[ M=11 \]
Logical Query Plan 2

\[
\begin{align*}
\pi_{\text{sname}} \\
\sigma_{\text{pno}=2} \\
\sigma_{\text{scity}='Seattle' \land \text{sstate}='WA'}
\end{align*}
\]

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

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

\[
\begin{align*}
\text{T(Supply)} & = 10000 \\
\text{B(Supply)} & = 100 \\
\text{V(Supply, pno)} & = 2500
\end{align*}
\]
Logical Query Plan 2

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

\[ \pi_{\text{sname}}(\sigma_{\text{pno}=2} \sigma_{\text{scity}='Seattle'} \land \text{sstate}='WA')(\text{Supplier}) = 1000 \]

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

\[ \text{M=11} \]

\[ \text{Generally wrong! Why?} \]
Logical Query Plan 2

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

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

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

Very wrong! Why?

\(M=11\)
Logical Query Plan 2

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

Very wrong! Why?

Different estimate 😞

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, sstate) = 10
M=11

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
Physical Plan 1

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

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

\[ \text{T} = 10000 \]

Block nested loop join

Scan

\[ \text{Supply} \]

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

Scan

\[ \text{Supplier} \]

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

Total cost:

\[ \frac{100}{10} \times 100 = 1000 \]
Physical Plan 1

\[ \pi_{sname} \]

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

\[ T = 10000 \]

\[ T < 1 \]

Block nested loop join

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

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

Total cost: 100 + 100 * 100 / 10 = 1100
Physical Plan 2

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

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

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

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

Unclustered index lookup Supplier(scity)

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

M=11
Physical Plan 2

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

- \( \text{sid} = \text{sid} \)
- \( \sigma_{\text{sstate}='WA'}(\text{Supplier}) \)
- \( \sigma_{\text{scity}='Seattle'}(\text{Supplier}) \)

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

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

\( M=11 \)
Physical Plan 2

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

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

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

M=11
Physical Plan 3

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

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

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

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

Clustered Index join

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

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

\[ M = 11 \]
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 3

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

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

Clustered Index join

Unclustered index lookup
Supply(pno)

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

M=11
Physical Plan 3

```
T = 4
σ_{scity='Seattle' ∧ sstate='WA'}

T = 4
σ_{pno=2}  
```

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

Unclustered index lookup
Supply(pno)

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

Clustered Index join

Supplier

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

M=11
Query Optimizer Summary

• Input: A logical query plan
• Output: A good physical query plan
• Basic query optimization algorithm
  – Enumerate alternative plans (logical and physical)
  – Compute estimated cost of each plan
  – Choose plan with lowest cost

• This is called cost-based optimization