Introduction to Data Management
CSE 344

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

(3 lectures)
Introduction to Data Management
CSE 344

Lecture 15: Introduction to Query Evaluation
Announcement

• WQ5 (datalog) due tomorrow

• HW4 (datalog) due tomorrow

• Midterm: Wednesday, 1:30 in class
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

- Parse & Rewrite Query
- Select Logical Plan
- Select Physical Plan
- Query Execution

Query optimization

SQL query

Logical plan (RA)

Physical plan

Disk
Query Execution
Physical Operators

Relational algebra operators:
• Selection, projection, join, union, difference
• Group-by, distinct, sort

Physical operators:
• For each operators above, several possible algorithms
• Main memory algorithms, or disk-based algorithms
Main Memory Algorithms

Logical operator:

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

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

1.
2.
3.
Main Memory Algorithms

Logical operator:

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

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

1. Nested Loop Join \( O(??) \)
2. Merge join \( O(??) \)
3. Hash join \( O(??) \)
Logical operator:

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

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

1. Nested Loop Join \( O(n^2) \)
2. Merge join \( O(n \log n) \)
3. Hash join \( O(n) \ldots O(n^2) \)
BRIEF Review of Hash Tables

Separate chaining:

A (naïve) hash function:

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

Operations:

- \( \text{find}(103) = ?? \)
- \( \text{insert}(488) = ?? \)

Duplicates OK

Why??
BRIEF Review of Hash Tables

• \(\text{insert}(k, v) = \text{inserts a key } k \text{ with value } v\)

• Many values for one key
  – Hence, duplicate \(k\)’s are OK

• \(\text{find}(k) = \text{return the list of all values } v \text{ associated to the key } k\)
Query Execution

- Join $R \bowtie S$:
  - Nested-loop: forall $x$ in $R$ forall $y$ in $S$ do …
  - Hash-join: build a hash table on $S$, probe $R$
- Selection: $\sigma(R)$: e.g. “on-the-fly”
- But what about a larger plan?
  - Each operator implements the Iterator Interface
Implementing Query Operators with the Iterator Interface

Each operator implements three methods:

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

Example “on the fly” selection operator

interface Operator {

}
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 ();
}
```
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 ();
}
```

Example "on the fly" selection operator
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

```java
interface Operator {
    void open (...);

    // 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 () ;
    }
}
```
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 () {

    }
}
```
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(in);
        }
    }
}
```
Implementing Query Operators with the Iterator Interface

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(in);
        }
        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 ();
}

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(in);
        }
        return r;
    }

close () { c.close(); }
}
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 ();
}

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

Pipelining

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

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

(Nested loop)  \[ \text{sno} = \text{sno} \]

Suppliers (File scan)

Supplies (File scan)

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

Suppliers

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

Supplies

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

(On the fly)

(On the fly)

(On the fly)

(Nested loop)

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

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

\[ \text{sno} = \text{sno} \]

Suppliers

(File scan)

Supplies

(File scan)

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

\textbf{Pipelining}

(On the fly)

(On the fly)

(Nested loop)

\(\Pi_{\text{sname}}\)

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

\(\text{sno} = \text{sno}\)

Suppliers (File scan)

Supplies (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)

Suppliers (File scan)

Supplies (File scan)

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

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

Supplies

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

Discuss: open/next/close for nested loop join

(On the fly)

(On the fly)

(Nested loop)
Suppliers

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

Supplies

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

Discuss: open/next/close for nested loop join

\( \text{Suppliers} \) (File scan)

\( \text{Supplies} \) (File scan)
Pipelining

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

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

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

(Nested loop)
\[ \text{sno} = \text{sno} \]

Suppliers (File scan)

Supplies (File scan)

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

\(sno = sno\)

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

\(\Pi_{\text{sname}}\)

\(\text{next()}\)

\(\text{next()}\)

\(\text{next()}\)

\(\text{next()}\)

\(\text{next()}\)

\(\text{next()}\)

\(\text{next()}\)

Suppliers

(File scan)

Supplies

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

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

(Nested loop) \( \text{sno} = \text{sno} \)

Discuss: open/next/close for nested loop join

Suppliers
(File scan)

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

\textbf{Pipelining}

(On the fly)

\textbf{Suppliers (File scan)}

\textbf{Supplies (File scan)}

(On the fly)

\[ \sigma_{\text{city} = 'Seattle' \text{ and } \text{state} = 'WA' \text{ and } pno=2} \]

\textbf{Nested loop}

\[ \text{sno} = \text{sno} \]

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

\begin{align*}
\text{Suppliers} & \rightarrow \text{Suppliers (File scan)} \\
\text{Supplies} & \rightarrow \text{Supplies (File scan)} \\
\end{align*}

\begin{align*}
\text{Suppliers} : \sigma_{\text{scity} = \text{'Seattle'} \text{ and sstate} = \text{'WA'} \text{ and pno} = 2} \\
\text{Supplies} : \pi_{\text{sname}}
\end{align*}

Pipelining

(On the fly) 

(On the fly) 

(Nested loop)

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

(Nested loop)

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

\(\text{sno} = \text{sno}\)

Suppliers (File scan)

Supplies (File scan)

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

Supplies

Pipelining

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

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

(Hash Join) \( \bowtie_{\text{sno} = \text{sno}} \)

Suppliers (File scan)

Supplies (File scan)

Discuss hash-join in class
Supplier\((\text{sid, sname, scity, sstate})\)  
Supply\((\text{sid, pno, quantity})\)

Pipelining

\[(\text{On the fly})\quad \pi_{\text{sname}} \quad \sigma_{\text{scity}=\text{Seattle} \text{ and sstate}=\text{WA} \text{ and pno}=2} \quad \text{sno} = \text{sno} \quad \text{Suppliers (File scan)} \quad \text{Supplies (File scan)}\]

Discuss hash-join in class

Tuples from here are pipelined
Suppliers

\[ \text{Suppliers} = \sigma_{\text{scity} = \text{Seattle} \land \text{sstate} = \text{WA} \land \text{pno} = 2} (\text{Supplier}) \]

\[ \text{Supplies} = \pi_{\text{sname}} (\text{Supply}) \]

(Tuples from here are pipelined)

(Tuples from here are "blocked")

(Tuples from here are pipelined)
Blocked Execution

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

(On the fly) Suppliers (File scan) Supplies (File scan)

Discuss merge-join in class
Blocked Execution

\[
\begin{align*}
\text{Suppliers} & \quad \text{Supplies} \\
\text{(File scan)} & \quad \text{(File scan)} \\
\text{Blocked} & \quad \text{Blocked}
\end{align*}
\]

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

\[
\begin{align*}
\Pi_{\text{sname}} \\
\sigma_{\text{scity}=\text{Seattle} \text{ and } \text{sstate}=\text{WA} \text{ and } \text{pno}=2} \\
\text{sno} = \text{sno}
\end{align*}
\]

(On the fly)

Discuss merge-join in class

(Merge Join)
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
Discussion on Physical Plan

More components of a physical plan:

- **Access path selection** for each relation
  - Scan the relation or use an index (next lecture)

- **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 16:
Basics of Data Storage and Indexes
Announcements

- Midterm will be graded next week
- HW5 is posted: NoSQL and SQL++; due Tue
- No WQ until Nov. 30

- In HW6 we will use AWS. Setup early:
  - If no account yet, sign up [aws.amazon.com](http://aws.amazon.com)
  - Request credits [aws.amazon.com/awscreds](http://aws.amazon.com/awscreds)
Query Performance

To understand query performance, we need to understand:

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

- In this course we will focus on disk-based DBMSs
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

In the example, we have 4 blocks with 2 tuples each

<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>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</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>

Student
Data File Types

The data file can be one of:

- **Heap file**
  - Unsorted

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

---

**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>
<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:
  – The key = an attribute value (e.g., student ID or name)
  – The value = a pointer to the record
Index

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

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

• Could have many indexes for one table

**Key =** means here search 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**

---

**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>

---

**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>
<tr>
<td>50</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>200</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>
<tr>
<td>950</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 2: Index on fName

Index \textit{Student\_fName} on \textit{Student.fName}

Data File \textit{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>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

\begin{itemize}
  \item Amy
  \item Ann
  \item Bob
  \item Cho
  \item ...
\end{itemize}

\begin{itemize}
  \item 10
  \item 20
  \item 50
  \item 200
  \item 220
  \item 240
  \item 420
  \item 800
\end{itemize}
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
Hash table example

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 File (preferably in memory)

Data file (on disk)

CSE 344 - 2018au
B+ Tree Index by Example

d = 2

Find the key 40

10 15 18 20 30 40 50

10 15 18 20 30 40 50 60 65

10 15 18 20 30 40 50 60 65 80 85 90

CSE 344 - 2018au
Clustered vs Unclustered

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
Scanning a Data File

• Disks are mechanical devices!
  – Technology from the 60s;
  – Density increases over time

• Read only at the rotation speed!

• Consequence: sequential scan faster than random
  – **Good**: read blocks 1,2,3,4,5,…
  – **Bad**: 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

• **Solid state (SSD)**: still too expensive today
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
Example

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

Example

for y in Takes
    if courseID > 300 then
        for x in Student
            if x.ID=y.studentID
                output *

\[ \sigma \text{Student} \prec \text{Takes} \]
Example

Assume the database has indexes on these attributes:

- **Takes_courseID** = index on Takes.courseID
- **Student_ID** = index on Student.ID

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

Assume the database has indexes on these attributes:
- **Takes_courseID** = index on Takes.courseID
- **Student_ID** = index on Student.ID

for y’ in Takes_courseID where y’.courseID > 300
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 *

Assume the database has indexes on these attributes:
- Takes_courseID = index on Takes.courseID
- Student_ID = index on Student.ID

for y’ in Takes_courseID where y’.courseID > 300
  y = fetch the Takes record pointed to by y’
Example

Assume the database has indexes on these attributes:

- **Takes_courseID** = index on Takes.courseID
- **Student_ID** = index on Student.ID

```sql
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 *
```

for y' in Takes_courseID where y'.courseID > 300
  y = fetch the Takes record pointed to by y'
  for x' in Student_ID where x'.ID = y.studentID
    x = fetch the Student record pointed to by x'
```
Example

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

Assume the database has indexes on these attributes:

- **Takes_courseID** = index on Takes.courseID
- **Student_ID** = index on Student.ID
SELECT *
FROM Student x, Takes y
WHERE x.ID = y.studentID AND y.courseID > 300

Assume the database has indexes on these attributes:
- **Takes_courseID** = index on Takes.courseID
- **Student_ID** = index on Student.ID

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

```python
for y’ in Takes_courseID where y’.courseID > 300
    y = fetch the Takes record pointed to by y’
    for x’ in Student_ID where x’.ID = y.studentID
        x = fetch the Student record pointed to by x’
        output *
```
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N varchar(20), P int);
CREATE INDEX V1 ON V(N)
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N varchar(20), 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 varchar(20), 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 varchar(20), 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

```sql
CREATE TABLE V(M int, N varchar(20), 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 P=55 and M=77
```

yes
Creating Indexes in SQL

CREATE TABLE V(M int, N varchar(20), P int);

CREATE INDEX V1 ON V(N);

CREATE INDEX V2 ON V(P, M);

What does this mean?

Yes

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 varchar(20), 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

select * from V where P=55

Yes

select * from V where M=77

Yes
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N varchar(20), 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
no
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N varchar(20), 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

Not supported in SQLite

yes

yes

no
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?

In general this is a very hard problem
Which Indexes?

- The *index selection problem*
  - Given a table, and a “workload” (big Java application with lots of SQL queries), decide which indexes to create (and which ones NOT to create!)

- Who does index selection:
  - The database administrator DBA
    - Semi-automatically, using a database administration tool
Which Indexes?

• The *index selection problem*
  – Given a table, and a “workload” (big Java application with lots of SQL queries), decide which indexes to create (and which ones NOT to create!)

• Who does index selection:
  – The database administrator DBA
  – Semi-automatically, using a database administration tool
Index Selection: Which Search Key

• Make some attribute K a search key if the WHERE clause contains:
  – An exact match on K
  – A range predicate on K
  – A join on K
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:

\[
\text{SELECT } * \text{ FROM } V \text{ WHERE } N=\
\]

100 queries:

\[
\text{SELECT } * \text{ FROM } V \text{ WHERE } P=\
\]

What indexes?
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=?

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:

100 queries:

100000 queries:

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

SELECT * 
FROM V 
WHERE P=?

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

Your workload is this

1000 queries:

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

100000 queries:

```
SELECT *
FROM V
WHERE P>? and P<?
```
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
  SELECT * FROM Movie WHERE year = ?

• Range queries
  SELECT * FROM Movie WHERE year >= ? AND year <= ?
Basic Index Selection Guidelines

• Consider queries in workload in order of importance

• Consider relations accessed by query
  – No point indexing other relations

• Look at WHERE clause for possible search key

• Try to choose indexes that speed-up multiple queries
To Cluster or Not

• Range queries benefit mostly from clustering
• An index is called a **covering index** for a query, if the query can be answered directly from the index, without access to the data
  – select count(*) from employee
    where 30<age and age < 35
  – Covering indexes do **not** need to be clustered: they work equally well unclustered
SELECT * 
FROM R 
WHERE R.K>? and R.K<?
SELECT * FROM R WHERE R.K>? and R.K<?
**Percentage tuples retrieved**

**Cost**

- **Sequential scan**
- **Clustered index**

```sql
SELECT * 
FROM R 
WHERE R.K>? and R.K<?
```
```
SELECT * 
FROM R 
WHERE R.K > ? and R.K < ?
```
Introduction to Database Systems
CSE 344

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

• The optimizer considers several plans, estimates their costs, and chooses the cheapest

• This lecture: cost estimation for relational operators

• The cost is always dominated by the cost of reading from, or writing to disk
Cost of Reading
Data From Disk
Cost Parameters

• Cost = I/O + CPU + Network BW
  – We will focus on I/O in this class
• Parameters (a.k.a. statistics):
  – $B(R)$ = # of blocks (i.e., pages) for relation R
  – $T(R)$ = # of tuples in relation R
  – $V(R, a)$ = # of distinct values of attribute a
Cost Parameters

- Cost = I/O + CPU + Network BW
  - We will focus on I/O in this class

- Parameters (a.k.a. statistics):
  - $B(R)$ = # of blocks (i.e., pages) for relation R
  - $T(R)$ = # of tuples in relation R
  - $V(R, a)$ = # of distinct values of attribute a

  When $a$ is a key, $V(R,a) = T(R)$
  When $a$ is not a key, $V(R,a)$ can be anything $\leq T(R)$
Cost Parameters

- Cost = I/O + CPU + Network BW
  - We will focus on I/O in this class

- Parameters (a.k.a. statistics):
  - \( B(R) \) = # of blocks (i.e., pages) for relation R
  - \( T(R) \) = # of tuples in relation R
  - \( V(R, a) \) = # of distinct values of attribute a

- DBMS collects statistics about base tables
  - must infer them for intermediate results

When \( a \) is a key, \( V(R, a) = T(R) \)
When \( a \) is not a key, \( V(R, a) \) can be anything \( \leq T(R) \)
Selectivity Factors for Conditions

• $A = c$  /* $\sigma_{A=c}(R)$ */
  – Selectivity $f = 1/V(R,A)$

• $A < c$  /* $\sigma_{A<c}(R)$ */
  – Selectivity $f = (c - \min(R, A))/(\max(R,A) - \min(R,A))$

• $c_1 < A < c_2$  /* $\sigma_{c_1<A<c_2}(R)$ */
  – Selectivity $f = (c_2 - c_1)/(\max(R,A) - \min(R,A))$

• $\text{Cond1} \land \text{Cond2} \land \text{Cond3} \land \ldots$
  – Selectivity $= f_1 f_2 f_3 \ldots$ (assumes independence)
Cost of Reading Data From Disk

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

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

Note: we ignore I/O cost for index pages
Index Based Selection

- Example:
  - \( B(R) = 2000 \)
  - \( T(R) = 100,000 \)
  - \( V(R, a) = 20 \)

- Table scan:
- Index based selection:

\[
\text{cost of } \sigma_{a=v}(R) = ?
\]
Index Based Selection

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

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

\[
\text{cost of } \sigma_{a=v}(R) = ?
\]
Index Based Selection

- **Example:**
  - Table scan: $B(R) = 2000$ I/Os
  - Index based selection:
    - If index is clustered:
    - If index is unclustered:

<table>
<thead>
<tr>
<th>B(R)</th>
<th>cost of $\sigma_{a=v}(R) =$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td></td>
</tr>
</tbody>
</table>

- $T(R) = 100,000$
- $V(R, a) = 20$
Index Based Selection

- Example:
  \[
  \begin{align*}
  B(R) &= 2000 \\
  T(R) &= 100,000 \\
  V(R, a) &= 20
  \end{align*}
  \]
  Cost of \( \sigma_{a=v}(R) = ? \)

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

- Index based selection:
  - If index is clustered: \( B(R) \times \frac{1}{V(R,a)} = 100 \) I/Os
  - If index is unclustered:
Index Based Selection

Example:

<table>
<thead>
<tr>
<th>B(R)</th>
<th>T(R)</th>
<th>V(R, a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>100,000</td>
<td>20</td>
</tr>
</tbody>
</table>

cost of $\sigma_{a=v}(R) = ?$

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

Index based selection:

- If index is clustered: $B(R) \times 1/V(R,a) = 100$ I/Os
- If index is unclustered: $T(R) \times 1/V(R,a) = 5,000$ I/Os
Index Based Selection

- Example:

| B(R) = 2000 | T(R) = 100,000 | V(R, a) = 20
| cost of $\sigma_{a=v}(R)$ = ? |

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

Lesson: Don’t build unclustered indexes when $V(R,a)$ is small!
Cost of Executing Operators (Focus on Joins)
Outline

• **Join operator algorithms**
  – One-pass algorithms (Sec. 15.2 and 15.3)
  – Index-based algorithms (Sec 15.6)

• **Note about readings:**
  – In class, we discuss only algorithms for joins
  – Other operators are easier: read the book
Join Algorithms

- Hash join
- Nested loop join
- Sort-merge join
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(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' 'Seattle'</td>
<td>2 'Blue' 123</td>
</tr>
<tr>
<td>2 'Ela' 'Everett'</td>
<td>4 'Prem' 432</td>
</tr>
<tr>
<td>3 'Jill' 'Kent'</td>
<td>4 'Prem' 343</td>
</tr>
<tr>
<td>4 'Joe' 'Seattle'</td>
<td>3 'GrpH' 554</td>
</tr>
</tbody>
</table>

Two tuples per page
Hash Join Example

Patient \Join Insurance

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

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
2 4
Output buffer
2 2

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: pid % 5

Disk
Patient Insurance
1 2
3 4
9 6
8 5

Input buffer
2 4

Output buffer
4 4
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

<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</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Input buffer

| 4 3 |

Output buffer

| 4 4 |

Keep going until read all of Insurance

Cost: B(R) + B(S)
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 t₁ in r, t₂ in s
      if t₁ and t₂ join then output (t₁,t₂)

• Cost: B(R) + B(R)B(S)
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

Input buffer for Patient
1  2
2  4
Input buffer for Insurance
2  2
Output buffer
Page-at-a-time Refinement

Patient Insurance

Disk

Input buffer for Patient

Input buffer for Insurance

Output buffer
Page-at-a-time Refinement

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

for each group of M-1 pages r in R do
  for each page of tuples s in S do
    for all pairs of tuples t₁ in r, t₂ in s
      if t₁ and t₂ join then output (t₁, t₂)

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

Disk

Patient  Insurance
1 2
3 4
9 6
8 5

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

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

Step 2: Scan Insurance and sort in memory

Memory M = 21 pages

Disk

Patient Insurance

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

Step 3: Merge Patient and Insurance

Memory M = 21 pages

Disk

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

Output buffer

1 1
Sort-Merge Join Example

Step 3: **Merge** Patient and Insurance

Memory $M = 21$ pages

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

Output buffer

Keep going until end of first relation
Index Nested Loop Join

\( R \bowtie S \)

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

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

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

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

M=11
Logical Query Plan 1

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

\[ T = 10000 \]

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

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

\[ T < 1 \]

\[ T = 10000 \]

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

\[ \text{Supply} \]

\[ \text{Supplier} \]

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

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

\[ \text{M} = 11 \]
Logical Query Plan 2

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

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

\[ \pi_{sname} \]

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

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

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

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

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?

M = 11
Logical Query Plan 2

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

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

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

\[
\begin{align*}
\text{T} & = 4 \\
\text{M} & = 11
\end{align*}
\]
Logical Query Plan 2

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

Very wrong! Why?

Different estimate 😞

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

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

\[ \text{M=11} \]

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

\[ \text{sid, sname, scity, sstate} \]
\[ \text{sid, pno, quantity} \]
Physical Plan 1

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

Block nested loop join

\[\text{Total cost: } 100/10 \times 100 = 1000\]

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

\[\text{M=11} \]

\[\text{T(Supply)} = 100000 \]
\[\text{B(Supply)} = 100 \]
\[\text{V(Supply, pno)} = 2500 \]
Physical Plan 1

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

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

\[ \text{T} < 1 \]

\[ \text{T} = 10000 \]

\[ \text{Block nested loop join} \]

\[ \text{Total cost: } 100 + 100 \times 100/10 = 1100 \]

\[ \text{Scan} \]

\[ \text{Scan} \]

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

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

\[ \text{M} = 11 \]
Physical Plan 2

\[
\begin{align*}
\text{Cost of Supply(pno) } &= 4 \\
\text{Cost of Supplier(scity) } &= 50 \\
\text{Total cost: } &= 54
\end{align*}
\]

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

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

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

\[
\begin{align*}
\sigma_{pno=2} & \quad \text{Supply} \\
\sigma_{sstate='WA'} \quad T=5 & \quad \text{Unclustered index lookup Supplier(sstate='WA')} \\
\sigma_{scity='Seattle'} \quad T=50 & \quad \text{Unclustered index lookup Supplier(scity='Seattle')} \\
\pi_{sname} \quad T=4 & \quad \text{Unclustered index lookup Supplier(scity='Seattle')} \\
\sigma_{sstart='WA'} \quad T=5 & \quad \text{Main memory join}
\end{align*}
\]
Physical Plan 2

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

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

\[ \text{Supply} \]

\[ \sigma_{\text{sstate} = \text{\textquotedoubleleft}WA\text{\textdoublequoteleft}} \]

\[ \text{Supplier} \]

\[ \sigma_{\text{scity} = \text{\textquotedoubleleft}Seattle\text{\textdoublequoteleft}} \]

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

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

\[ \text{Total cost: 54} \]

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

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

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

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

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

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

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

\[ M = 11 \]
Physical Plan 2

\begin{align*}
\text{Supply}(\text{sid, pno, quantity}) \\
\text{Supplier}(\text{sid, sname, scity, sstate})
\end{align*}

\begin{align*}
\text{Cost of Supply}(\text{pno}) &= 4 \\
\text{Cost of Supplier}(\text{scity}) &= 50 \\
\text{Total cost:} &= 54
\end{align*}
Physical Plan 3

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

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

Unclustered index lookup \text{Supply(pno)}

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

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

\[
M=11
\]
Physical Plan 3

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

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

Unclustered index lookup
Supply(\text{pno})

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

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

Clustered Index join

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

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

M=11
Physical Plan 3

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

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

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

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

\[ \text{Supply} \]

\[ \text{Supplier} \]

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

\[ M=11 \]

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