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

(4 lectures)
Introduction to Data Management
CSE 344

Lecture 14: Introduction to Query Evaluation
Announcement

• Midterm review: 5:30pm in Smith 205
• WQ5 (datalog) due tonight
• HW4 (datalog) due on Tuesday
• 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

1. SQL query

2. Parse & Rewrite Query

3. Select Logical Plan

4. Select Physical Plan

5. Query Execution

Query optimization

Logical plan (RA)

Physical plan

Disk
Logical vs Physical Plans

• Logical plans:
  – Created by the parser from the input SQL text
  – Expressed as a relational algebra tree
  – Each SQL query has many possible logical plans

• Physical plans:
  – Goal is to choose an efficient implementation for each operator in the RA tree
  – Each logical plan has many possible physical plans
Relational algebra expression is also called the “logical query plan”
A physical query plan is a logical query plan annotated with physical implementation details.

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

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

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

\[ \pi_{sname} \]

(Hash join)

(On the fly)

Same logical query plan
Different physical plan

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

(File scan)

(File scan)
Physical Query Plan 3

(Scan & write to T1)

(a) $\sigma_{\text{scity} = \text{'Seattle'} \text{ and sstate} = \text{'WA'}}$

(b) $\sigma_{\text{pno} = 2}$ (Scan & write to T2)

(Sort-merge join)

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

(On the fly)

Different but equivalent logical query plan; different physical plan

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

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Query Optimization Problem

• For each SQL query… many logical plans

• For each logical plan… many physical plans

• Next: we will discuss physical operators; how exactly are query executed?
Query Execution
Physical Operators

Each of the logical operators may have one or more implementations = physical operators

Will discuss several basic physical operators, with a focus on join
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.

2.

3.
Main Memory Algorithms

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

1. Nested Loop Join
2. Merge join
3. Hash join
Supplier($sid$, sname, scity, sstate)
Supply($sid$, pno, quantity)

Main Memory Algorithms

Logical operator:

Supplier $\times_{sid=sid}$ 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:

\[ \text{find}(103) = ?? \]
\[ \text{insert}(488) = ?? \]
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`
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 {

}
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection 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 ();
}
```
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 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 ();
}

class Select implements Operator {

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

    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 child) {
        this.p = p; this.child = child;
    }

    Tuple next () {
        boolean found = false;
        Tuple r = null;
        while (!found) {
            r = child.next();
            if (r == null) break;
            found = p(in);
        }
    }
}
```

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 child) {
        this.p = p; this.child = child;
    }

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

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

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

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

Supply

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

**Pipelining**

(On the fly)

(On the fly)

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

(Nested loop)

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

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

Suppliers

(File scan)

Supplies

(File scan)

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

File scan

Nested loop

On the fly

Open/next/close for nested loop join

$$\pi_{sname}$$

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

sno = sno

Suppliers (File scan)

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

Pipelining

(On the fly) π\text{sname}

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

(Nested loop) sno = 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)

(Nested loop)

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

\[ \pi_{sname} \]

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

\[ \text{open()} \]

\[ \text{open()} \]

\[ \text{open()} \]

\[ \text{open()} \]

Suppliers

**(File scan)**

Supplies

**(File scan)**

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

Supplies

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

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

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

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

Suppliers (File scan)

Supplies (File scan)

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

\[ \text{Suppliers} = \pi_{\text{sname}} (\text{Scity} = 'Seattle' \text{ and } \text{sstate} = 'WA' \text{ and } \text{pno} = 2) \]

Supplies

\[ \text{Supplies} = \text{next() for nested loop join} \]

\[ \text{Suppliers} \]

\[ \text{Supplies} \]
Pipelining

Suppliers

\( \pi_{sname} \)

Supplies

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

\( s = s \)

(On the fly)

(On the fly)

(Nested loop)

Discuss: open/next/close for nested loop join

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

\[ \text{Suppliers} \]

\[ \text{Supply} \]

\[ \text{Supplier} \]

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

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

(On the fly)

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

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

\[ \text{next()} \]

(On the fly)

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

\[ \text{next()} \]

(Nested loop)

\[ \text{next()} \]

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

(On the fly)

Suppliers

(On the fly)

Supplies

(Nested loop)

 Suppliers (File scan)

 Supplies (File scan)

π_{sname} (File scan)

σ_{scity='Seattle' and sstate='WA' and pno=2} (File scan)

sno = sno

discuss: open/next/close for nested loop join
Suppliers(sno = sno)

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

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

\( \text{Suppliers} \)

File scan

\( \text{Supplies} \)

File scan

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

Pipelining

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

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

Discuss: open/next/close for nested loop join

(On the fly)

next()

(next())

(next())

(next())

(next())

(next())

next()
Pipelining

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

(On the fly)

(On the fly)

(Hash Join)

(On the fly)

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

\[ \text{Suppliers} \quad \text{Supplies} \]

(File scan) (File scan)

Guard conditions:

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

Discuss hash-join in class
Suppliers

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

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

Pipelining

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

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

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

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

(Hash Join)

Suppliers

(File scan)

Tuples from here are pipelined

Discuss hash-join in class
Suppliers

\( \text{Hash Join} \)

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

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

Suppliers

(File scan)

Supplies

(File scan)

Tuples from here are pipelined

Tuples from here are “blocked”

Discuss hash-join in class
Supplier($sid$, $sname$, $scity$, $sstate$)
Supply($sid$, $pno$, quantity)

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

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

(Merge Join) $sno = sno$

Suppliers (File scan) Supplies (File scan)

Discuss merge-join in class
Blocked Execution

\[ \text{Suppliers} \]
\[ \text{Supplies} \]

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

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

\[ \Pi_{\text{name}} \]

Discuss merge-join in class

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

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

\[ (\text{Merge Join}) \]

Suppliers (File scan)

Supplies (File scan)

Blocked

Blocked
Pipelined Execution

- Tuples generated by an operator are immediately sent to the parent

- Benefits:
  - No operator synchronization issues
  - No need to buffer tuples between operators
  - Saves cost of writing intermediate data to disk
  - Saves cost of reading intermediate data from disk

- This approach is used whenever possible
Query Execution Bottom Line

• SQL query transformed into 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

• Pipelined execution of physical plan
Recall: Physical Data Independence

- Applications are insulated from changes in physical storage details

- SQL and relational algebra facilitate physical data independence
  - Both languages input and output relations
  - Can choose different implementations for operators
Introduction to Database Systems
CSE 344

Lecture 15-16:
Basics of Data Storage and Indexes
Announcements

• HW4 (datalog) due tomorrow (Tuesday)

• Midterm: Wednesday

• No sections on Thursday!

• HW5 (SQL++) due next Tuesday
Query Performance

• My database application is too slow… why?
• One of the queries is very slow… why?

• To understand 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

<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>

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>
Data File Types

The data file can be one of:

• Heap file
  – Unsorted

• Sequential file
  – Sorted according to some attribute(s) called `key`

Note: `key` here means something different from primary key: it just means that we order the file according to that attribute. In our example we ordered by `ID`. Might as well order by `fName`, if that seems a better idea for the applications running on our database.
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
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>
<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 **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 Organization

We need a way to represent indexes after loading into memory so that they can be used.

Several ways to do this:

• Hash table
• B+ trees – most popular
  – They are search trees, but they are not binary instead have higher fanout
  – 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>
<tr>
<td>50</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>220</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>200</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>

Index File (preferably in memory)

Data file (on disk)
B+ Tree Index by Example

d = 2

Find the key 40
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 much higher now
- Read only at the rotation speed!
- Consequence:
  Sequential scan is MUCH FASTER than random reads
  - **Good**: read blocks 1,2,3,4,5,…
  - **Bad**: read blocks 2342, 11, 321,9,…
- **Rule of thumb**:
  - Random reading 1-2% of the file ≈ sequential scanning the entire file; this is decreasing over time (because of increased density of disks)
- Solid state (SSD): $$$ expensive; put indexes, other “hot” data there, still too expensive for everything
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
```

Student (ID, fname, lname)
Takes (studentID, courseID)
Example

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)
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
```
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
```

```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
```
Assume the database has indexes on these attributes:

- **Takes**\_\_courseID = index on Takes.courseID
- **Student**\_ID = index on Student.ID

```plaintext
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’
```
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

```
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

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 \(x'\)
  output *
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 *
```
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)
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?
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);

SELECT * FROM V WHERE P=55 AND M=77
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 P=55 and M=77
- select * from V where P=55
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 and M=77

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

select *
from V
where P=55

select *
from V
where M=77

yes

yes

no
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);
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` is supported.
- `select * from V where P=55 and M=77` is supported.
- `select * from V where M=77` is supported.
- `select * from V where M=77` is not supported in SQLite.

Not supported in SQLite.
Which Indexes?

- How many indexes could we create?

- Which indexes should we create?

<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>
Which Indexes?

- How many indexes could we create?
- Which indexes should we create?

In general this is a very hard problem.

<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>
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:

100 queries:

SELECT * 
FROM V 
WHERE N=?

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

V(M, N, P);

Your workload is this
100000 queries:
100 queries:

SELECT * 
FROM V
WHERE N=?

SELECT * 
FROM V
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:
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

Your workload is this:

100,000 queries:  SELECT * FROM V WHERE N=?

100,000 queries:  SELECT * FROM V WHERE N=? and P>?

100,000 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
  
  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
• Covering indexes do not need to be clustered: they work equally well unclustered
SELECT * 
FROM R 
WHERE R.K>? and R.K<?
106 Percentage tuples retrieved

Cost

Sequential scan

SELECT * 
FROM R 
WHERE R.K>? and R.K<?

Cost

0 100

Percentage tuples retrieved

CSE 344 - 2017au
Percentage tuples retrieved

Cost

SELECT *
FROM R
WHERE R.K>?
and R.K<?

Sequential scan

Clustered index

Cost

Percentage tuples retrieved

0

100
Percentage tuples retrieved

Cost

0

100

Sequential scan

Clustered index

Unclustered index

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
Choosing Index is Not Enough

• To estimate the cost of a query plan, we still need to consider other factors:
  – How each operator is implemented
  – The cost of each operator
  – Let’s start with the basics
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 = \( \frac{1}{V(R,A)} \)

- **A < c**  
  
  \[ \sigma_{A<c}(R) \]  
  
  - Selectivity = \( \frac{c - \min(R, A)}{\max(R,A) - \min(R,A)} \)

- **c1 < A < c2**  
  
  \[ \sigma_{c1<A<c2}(R) \]  
  
  - Selectivity = \( \frac{c2 - c1}{\max(R,A) - \min(R,A)} \)
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

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

• Table scan:

• Index based selection:
Index Based Selection

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

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

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

- Index based selection:
Index Based Selection

• Example:

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

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

• Table scan: \( B(R) = 2,000 \) I/Os

• Index based selection:
  – If index is clustered:
  – If index is unclustered:
Index Based Selection

- **Example:**
  - 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: 
      \[
      \text{cost of } \sigma_{a=v}(R) = ?
      \]

- **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 scan: $B(R) = 2000$ I/Os
  - Index based selection:
    - If index is clustered: $B(R) \times \frac{1}{V(R, a)} = 100$ I/Os
    - If index is unclustered: $T(R) \times \frac{1}{V(R, a)} = 5000$ I/Os

$$B(R) = 2000 \quad T(R) = 100,000 \quad V(R, a) = 20$$

Cost of $\sigma_{a=v}(R) = ?$
Index Based Selection

- Example:
  - 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

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

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’</td>
<td>‘Seattle’</td>
</tr>
<tr>
<td>2 ‘Ela’</td>
<td>‘Everett’</td>
</tr>
<tr>
<td>3 ‘Jill’</td>
<td>‘Kent’</td>
</tr>
<tr>
<td>4 ‘Joe’</td>
<td>‘Seattle’</td>
</tr>
</tbody>
</table>

Two tuples per page
Hash Join Example

Patient \( \bowtie \) Insurance

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<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>

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

Disk

Showing pid only

Memory M = 21 pages

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

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: pid % 5

Disk

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

Output buffer

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

| 4 4 |
Hash Join Example

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

Memory $M = 21$ pages
Hash $h: \text{pid} \mod 5$

Input buffer
Output buffer

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

```plaintext
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: $B(R) + T(R) \cdot B(S)$
- 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

1 2
3 4
9 6
8 5

Insurance

2 4
4 3
1 3
2 8
8 9

Input buffer for Patient

1 2

Input buffer for Insurance

2 4

Output buffer

2 2
Page-at-a-time Refinement

Disk

Patient  Insurance

Patient Buffer: 1 2 3 4 5 6
Insurance Buffer: 2 4 6 6 1 3

Output Buffer: 2 8 8 9

Input buffer for Patient: 1 2
Input buffer for Insurance: 4 3
Page-at-a-time Refinement

Patient
1 2
3 4
9 6
8 5

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

Disk

Input buffer for Patient
1 2

Input buffer for Insurance
2 8

Keep going until read all of Insurance

Output buffer
2 2

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

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 $t_1$ in r, $t_2$ in s
      if $t_1$ and $t_2$ join then output ($t_1$, $t_2$)

• Cost: $B(R) + B(R)B(S)/(M-1)$

What is the Cost?
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
Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory

Memory M = 21 pages

Disk

Patient Insurance

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<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>

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

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

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Memory M = 21 pages

Disk

Patient  Insurance

1 2
3 4
9 6
8 5

Output buffer

[Diagram showing the merge process]

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

Step 3: Merge Patient and Insurance

Memory M = 21 pages

Disk

Patient  Insurance

1 2  
3 4  
9 6  
8 5  

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

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

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

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

\[ \text{M=11} \]
Logical Query Plan 1

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

\[ T = 10000 \]

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

\( M = 11 \)
Logical Query Plan 1

\[ T < 1 \]
\[ \sigma_{pno=2 \land scity='Seattle' \land sstate='WA'} \]
\[ T = 10000 \]
\[ \pi_{sname} \]

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

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

M=11
Logical Query Plan 2

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

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

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

\[ T(\text{Supply}) = 10000 \]
\[ B(\text{Supply}) = 100 \]
\[ V(\text{Supply}, \text{pno}) = 2500 \]
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'
```
Logical Query Plan 2

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

Very wrong! Why?

\[M=11\]

**Logical Query Plan 2**

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

**SUPPLIER(sid, sname, scity, sstate)**

**SUPPLY(sid, pno, quantity)**

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

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

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

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

Very wrong! Why?

M=11
Logical Query Plan 2

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

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

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

M=11

Very wrong! Why?

Different estimate 😞
Physical Plan 1

\[
\text{Total cost: } \frac{T}{10} \times B = 1000
\]
Physical Plan 1

\[
\text{Total cost: } \frac{100}{10} \times 100 = 1000
\]
Physical Plan 2

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

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

\[ \text{Supply} \]

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

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

\[ \text{Supplier} \]

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

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

Cost of Supply(pno) = 4
Cost of Supplier(scity) = 50
Total cost: 54
Physical Plan 2

\[
\text{Physical Plan 2}
\]

\[
\Pi_{\text{sname}}
\]

\[
T = 4
\]

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

\[
\text{Supply}
\]

\[
T = 4
\]

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

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

\[
\tau = 5
\]

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

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

\[
\text{Supplier}
\]

\[
T = 50
\]

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

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

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

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

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

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

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

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

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

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

\[
M = 11
\]

\[
\text{Main memory join}
\]

\[
\text{Unclustered index lookup}
\]

\[
\text{Supply(pno)}
\]

\[
\text{Supplier(scity)}
\]
Physical Plan 2

\[ \Pi_{\text{sname}}(\sigma_{\text{state}} = 'WA')(\sigma_{\text{pno}} = 2)(\text{Supplier}) = 1000 \]
\[ \text{B(}\text{Supplier}) = 100 \]
\[ \text{V(}\text{Supplier, } \text{state}) = 10 \]

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

Main memory join

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

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

\[ \text{T(Supply)} = 10000 \]
\[ \text{B(Supply)} = 100 \]
\[ \text{V(Supply, pno)} = 2500 \]
Physical Plan 3

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

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

Unclustered index lookup
Supply(pno)

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

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

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

M = 11
Physical Plan 3

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

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

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

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

Supply

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

Clustered Index join

\[ T = 4 \]

\[ T = 4 \]

\[ M = 11 \]

Supplier

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

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

Unclustered index lookup

Supply(pno)
Physical Plan 3

```
| π_{sname} |
| σ_{scity='Seattle' \land sstate='WA'} |
| σ_{pno=2} |
```

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

```
M=11
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, sstate) = 10
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
```

Unclustered index lookup

Supply(pno)

Clustered Index join

Supplier

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