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
Announcements

Makeup lecture tomorrow, 4:30pm, BAG 131

HW6: we will use AWS. Do the setup early:
• If no account yet, sign up aws.amazon.com
• Request credits aws.amazon.com/awscredits
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
- Disk

SQL query

Query optimization

Logical plan (RA)

Physical plan
Relational Algebra Operators

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

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

Logical operator:

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

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

1. 
2. 
3.
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. Nested Loop Join \(O(??)\)
2. Merge join \(O(??)\)
3. Hash join \(O(??)\)
Main Memory Algorithms

Logical operator:

Supplier \bowtie_{\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 \(O(n) \ldots O(n^2)\)

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

• 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
Recap of Main Memory Algorithms

• Join $\bowtie$
  – Nested loop join
  – Hash join
  – Merge join

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

• Group by $\gamma$
  – Hash–based
  – Merge-based
How Do We Combine Them?
How Do We Combine Them?

The *Iterator Interface*

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

Example “on the fly” selection operator

```java
interface Operator {

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

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

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

}
```
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

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

    Tuple next ();

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

}

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

Example “on the fly” selection operator

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

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

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

class Select implements Operator {
    void open (Predicate p,
               Operator c) {
        this.p = p; this.c = c; c.open();
    }
    Tuple next () {
        boolean found = false;
        while (!found) {
            Tuple in = child.next ();
            if (in == EOF) return EOF;
            found = p(in);
        }
        return in;
    }
    void close () {
        child.close();
    }
}
```

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

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

  // initializes operator state
  // and sets parameters
  Tuple next ();

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

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

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

  Tuple next () {
    // example code
  }

  // example code
}
```

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

Example “on the fly” selection operator

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

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

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

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

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

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

Example “on the fly” selection operator

```java
interface Operator {

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

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

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

class Select implements Operator {

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

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

    void 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 ();
}

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

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

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

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();
Pipelining

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

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

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

Supplier (File scan)

Supply (File scan)

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

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

(On the fly) open()

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

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

(Nested loop) sid = sid

Supplier (File scan)

Supply (File scan)

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

Pipelining

(On the fly) open()

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

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

(Nested loop) sid = sid

Supplier (File scan)

Supply (File scan)

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

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

(On the fly)

(On the fly)

(On the fly)

(Nested loop)

Discuss: open/next/close for nested loop join

$\pi_{sname}$

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

sid = sid

Supplier (File scan)

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

(On the fly) ρ_{scity=’Seattle’ and sstate=’WA’ and pno=2}

(On the fly) π_{sname}

(Nested loop) sid = sid

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

**Pipelining**

- **(On the fly)**
  - \(\sigma_{\text{scity}=\text{Seattle} \text{ and } \text{sstate}=\text{WA}} \text{ and } \text{pno}=2\)
  - \(\pi_{\text{sname}}\)
  - \(\text{open}()\)

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

- **(Nested loop)**
  - \(\text{sid} = \text{sid}\)
  - \(\text{open}()\)
  - \(\text{Supplier} \quad \text{(File scan)}\)
  - \(\text{Supply} \quad \text{(File scan)}\)

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)

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

π_{sname}

sid = sid

Discuss: open/next/close for nested loop join

Supplier (File scan)

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

\begin{itemize}
\item \textbf{Pipelining:}
\end{itemize}

\begin{itemize}
\item \textbf{(On the fly)}
\item \textbf{(On the fly)}
\item \textbf{(Nested loop)}
\end{itemize}

\begin{itemize}
\item \textbf{Discussion: open/next/close for nested loop join}
\end{itemize}

\begin{itemize}
\item \textbf{Next():}
\item \textbf{\textit{\pi}_{sname}}
\item \textbf{\textit{\sigma}_{\text{scity}=\text{‘Seattle’} \text{\textbf{\text{\&\text{}}} sstate=\text{‘WA’} \text{\textbf{\text{\&\text{}}} pno=2}}}
\item \textbf{Sid = sid}
\item \textbf{Supplier (File scan)} \quad \textbf{Supply (File scan)}
\end{itemize}
Supplier\((sid, sname, scity, sstate)\)
Supply\((sid, pno, quantity)\)

Pipelining

(On the fly)

\(\pi_{sname}\)

next()

(On the fly)

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

next()

(Nested loop)

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

Supplier
(File scan)

Supply
(File scan)

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

(On the fly)

(On the fly)

(On the fly)

(Nested loop)

\[ \text{π}_{\text{sname}} \]

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

sid = sid

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

next()

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

Pipelining

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

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

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

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

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

sid = sid

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

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

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

(On the fly)
(On the fly)
(Nested loop)

Discuss: open/next/close for nested loop join

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

\( \pi \text{sname} \)
Pipelining

Discuss hash-join in class

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

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

Supplier (File scan)  
Supply (File scan)
Pipelining

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

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

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

Supplier (File scan)  

Supply (File scan)

Discuss hash-join in class

Tuples from here are “blocked”

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

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

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

(Hash Join)  
\[ \text{sid} = \text{sid} \]

Supplier (File scan)  

Supply (File scan)  

Tuples from here are pipelined

Tuples from here are “blocked”
Blocked Execution

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

(On the fly)

(On the fly)

(On the fly)

(Merge Join)

Discuss merge-join in class

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

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

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

Supplier (File scan)

Supply (File scan)

CSE 344 - 2019wi
Blocked Execution

\textbf{(On the fly)}

\textbf{(On the fly)}

\textbf{(Merge Join)}

\textbf{Discuss merge-join in class}

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

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

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

\(\text{sid = sid}\)

\textbf{Blocked}

\textbf{Blocked}

\textbf{File scan}

\textbf{File scan}
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
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
Hard Disk

- Disks are mechanical devices
- A block = unit of read/write
- Once in main memory we call it a page
- Read only at the rotation speed!
- Consequence: sequential scan faster than random
  - 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
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>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>

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

The data file can be one of:

- **Heap file**
  - Unsorted

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

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

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

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

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

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

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

• Could have many indexes for one table

**Key = means here search key**
This Is Not A Key

Different keys:

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

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

Data File **Student**

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

### Index

- Amy
- Ann
- Bob
- Cho
- …
- …
- …
- …
- Tom
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
B+ Tree Index by Example

\[ d = 2 \]

Find the key 40

\[ 40 \leq 80 \]

\[ 20 < 40 \leq 60 \]

\[ 30 < 40 \leq 40 \]
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
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

\[
\text{SELECT *} \quad \\
\text{FROM Student x, Takes y} \quad \\
\text{WHERE x.ID=y.studentID AND y.courseID > 300}
\]
Example

\[
\begin{align*}
\text{Student}(\text{ID}, \text{fname}, \text{Iname}) & \\
\text{Takes}(\text{studentID}, \text{courseID}) & \\
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & * \\
\text{FROM} & \text{Student} \ x, \ \text{Takes} \ y \\
\text{WHERE} & \ x.\text{ID}=y.\text{studentID} \ \text{AND} \ y.\text{courseID} > 300 \\
\end{align*}
\]

\[
\begin{align*}
\text{for } y \ \text{in} \ \text{Takes} & \\
\text{if } \ y.\text{courseID} > 300 \ \text{then} & \\
\text{for } x \ \text{in} \ \text{Student} & \\
\text{if } x.\text{ID}=y.\text{studentID} & \\
\text{output } & *
\end{align*}
\]
Example

Assume the database has indexes on these attributes:

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

Assume the database has indexes on these attributes:

- \textbf{Takes\_courseID} = index on Takes.courseID
- \textbf{Student\_ID} = index on Student.ID

\begin{verbatim}
for y in Takes
    if courseID > 300 then
        for x in Student
            if x.ID=y.studentID
                output *
\end{verbatim}

\begin{verbatim}
for y' in Takes\_courseID where y'.courseID > 300
\end{verbatim}
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
```

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

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

\[
\text{SELECT *}
\text{FROM Student x, Takes y}
\text{WHERE x.ID=y.studentID AND y.courseID > 300}
\]

Assume the database has indexes on these attributes:

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

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

\[
\text{for } y' \text{ in Takes_courseID where } y'.courseID > 300
y = \text{fetch the Takes record pointed to by } y'
\text{for } x' \text{ in Student_ID where } x'.ID = y.studentID
x = \text{fetch the Student record pointed to by } x'
\]
Assume the database has indexes on these attributes:

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

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

Index selection

Index join

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
```

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

```sql
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

```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
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
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

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

yes

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

yes

yes

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

select * from V where P=55

select * from V where M=77

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
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

100,000 queries:

SELECT * FROM V WHERE N=?

100 queries:

SELECT * FROM V WHERE P=?

What indexes?
The Index Selection Problem 1

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

$$V(M, N, P);$$

Your workload is this

100000 queries: 1000000 queries: 100000 queries:

SELECT * FROM V WHERE N=?

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

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:

\[
\begin{align*}
& \text{SELECT } * \\
& \text{FROM } V \\
& \text{WHERE } N>? \text{ and } N<? \\
\end{align*}
\]

100000 queries:

\[
\begin{align*}
& \text{SELECT } * \\
& \text{FROM } V \\
& \text{WHERE } P>? \text{ and } P<? \\
\end{align*}
\]

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
  - What data structure should be used for index?

- Range queries
  - What data structure should be used for index?
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

Remember:

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

Range queries benefit mostly from clustering because they may read more than 1-2%
SELECT * 
FROM R 
WHERE R.K>? and R.K<?
SELECT * 
FROM R
WHERE R.K>? and R.K<?
\begin{verbatim}
SELECT * 
FROM R 
WHERE R.K>? and R.K<?
\end{verbatim}
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

When \( a \) is a key, \( V(R, a) = T(R) \)
When \( a \) is not a key, \( V(R, a) \) can be anything \( <= T(R) \)

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

Main principle:

• Size of the output = some \textit{fraction} of the size of the input

• The \textit{fraction} is called the \textit{selectivity factor}
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)

Will use mostly this…

…and this
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
  - Cost of $\sigma_{a=v}(R) = ?$

- Table scan:
- Index based selection:
Index Based Selection

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

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

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

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

- Index based selection:
  - If index is clustered:
  - If index is unclustered:

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

- 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: \( B(R) = 2,000 \) 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

\[ B(R) = 2000 \]
\[ T(R) = 100,000 \]
\[ V(R, a) = 20 \]

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: \( B(R) \cdot \frac{1}{V(R,a)} = 100 \) I/Os
    - If index is unclustered: \( T(R) \cdot \frac{1}{V(R,a)} = 5,000 \) I/Os

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

• Nested loop join
• Hash join
• Sort-merge join
• Index-join
Join Example

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

\[
\text{Patient} \Join \text{Insurance}
\]

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 'Bob'</td>
<td>'Blue' 123</td>
</tr>
<tr>
<td>2 'Ela'</td>
<td>'Prem' 432</td>
</tr>
<tr>
<td>3 'Jill'</td>
<td>'Prem' 343</td>
</tr>
<tr>
<td>4 'Joe'</td>
<td>'GrpH' 554</td>
</tr>
</tbody>
</table>

Two tuples per page
Nested Loop Joins

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

```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 $(t_1, t_2)$
```

What is the Cost?
Nested Loop Joins

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

\[ \text{Cost: } B(R) + T(R) B(S) \]

- 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

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

Insurance

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

Input buffer for Patient

Input buffer for Insurance

Output buffer

CSE 344 - 2019wi
Page-at-a-time Refinement

<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>

Disk

Input buffer for Patient

Input buffer for Insurance

Output buffer
Page-at-a-time Refinement

Disk

Patient

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

Insurance

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

Input buffer for Patient

Input buffer for Insurance

Keep going until read all of Insurance

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

Output buffer

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) + B(R)B(S)/(M-1)
Hash Join

Hash join: $R \bowtie S$

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

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

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

Patient $\bowtie$ Insurance

Disk

Patient

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

Insurance

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<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>

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()

<table>
<thead>
<tr>
<th>Memory M = 21 pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash h: pid % 5</td>
</tr>
<tr>
<td>5 1 6 2 3 8 4 9</td>
</tr>
</tbody>
</table>

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>
Hash Join Example

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

Memory $M = 21$ pages

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

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

Disk

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

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

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

2 4

4 4
Hash Join Example

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

Memory $M = 21$ pages

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

<table>
<thead>
<tr>
<th>Disk</th>
<th>Input buffer</th>
<th>Output buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient</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</td>
</tr>
<tr>
<td></td>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

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

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

Sort-merge join: \( R \bowtie S \)

- Scan \( R \) and sort in main memory
- Scan \( S \) and sort in main memory
- Merge \( R \) and \( S \)

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

Step 1: Scan Patient and sort in memory

Memory M = 21 pages
Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory

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
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Memory M = 21 pages

Disk

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

Output buffer

1 1
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Memory $M = 21$ pages

Disk

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

Output buffer

Keep going until end of first relation
Index 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:
    \[ B(R) + T(R) \ast (B(S) \ast 1/V(S,a)) \]
  - If index on S is unclustered:
    \[ B(R) + T(R) \ast (T(S) \ast 1/V(S,a)) \]
Cost of Query Plans
Example
Logical Query Plan 1

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

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

**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

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

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

\[ T = 10000 \]

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

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

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

\[ M=11 \]
Logical Query Plan 1

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

\[ T < 1 \]

\[ T = 10000 \]

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

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

\[ M=11 \]

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

```
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) = 10000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, sstate) = 10
M=11
```

```
T(Supply) = 100000
B(Supply) = 100
V(Supply, pno) = 2500
```

```
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) = 100
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, sstate) = 10
M=11
```
Logical Query Plan 2

\[ \sigma_{\text{scity='Seattle' \land \text{sstate='WA'}}} \] (Supplier, sid = sid) \rightarrow \pi_{\text{sname}}

\[ \sigma_{\text{pno=2}} \] (Supply, sid = sid) \rightarrow T = 4

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

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

\text{M=11}

Very wrong! Why?
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?

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

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

T= 4
T= 5
σ_pno=2
σ_scity='Seattle' ∧ sstate='WA'
π_sname
σ_sid=sid
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'
```

```
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, sstate) = 10
M=11
```
Physical Plan 1

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

\[ \pi_{sname} \]

\[ T < 1 \]

\[ T = 10000 \]

Block nested loop join

Scan

Supply

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

Scan

Supplier

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

Total cost:

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

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

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

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

\[ M = 11 \]

\[ \Phi(pno=2 \land \text{scity}='Seattle' \land \text{sstate}='WA') \]

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

\[ T < 1 \]

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

\[ \text{Total cost: } 100 + 100 \times 100/10 = 1100 \]
Physical Plan 2

\[ \sigma_{\text{sstate} = 'WA'}(\pi_{\text{sname}}(\sigma_{\text{pno} = 2}(\text{Supply}))) \]

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

\[ \sigma_{\text{scity} = 'Seattle'}(\text{Supplier}) \]

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

Unclustered index lookup
Supply(pno)

Unclustered index lookup
Supplier(scity)

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

M = 11
Physical Plan 2

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

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

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

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

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

\[ \sigma_{scity='Seattle'}(\sigma_{sstate='WA'}(\text{Supplier})) \]

\[ \text{Main memory join} \]

\[ T = 4 \]

\[ T = 5 \]

\[ M = 11 \]

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

\[ \text{T(Supplier)} = 1000 \]
\[ \text{B(Supplier)} = 100 \]
\[ \text{V(Supplier, scity)} = 20 \]
\[ \text{V(Supplier, state)} = 10 \]
Physical Plan 2

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

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

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

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

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

Physical Plan 3

\[
\begin{align*}
\pi_{sname} & \quad T = 4 \\
\sigma_{\text{scity}=\text{Seattle} \land \text{sstate}=\text{WA}} & \quad T = 4 \\
\sigma_{\text{pno}=2} & \quad \text{Unclustered index lookup}
\end{align*}
\]

\[
\begin{align*}
\text{Cost of Supply(pno)} &= 4 \\
\text{Cost of Index join} &= 4 \\
\text{Total cost:} &= 8
\end{align*}
\]

\[
\begin{align*}
\text{Unclustered index lookup} & \quad \text{Supply(pno)} \\
\text{Clustered Index join} & \quad \text{Supplier}
\end{align*}
\]

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

M = 11
Physical Plan 3

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

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

\[ M = 11 \]
Physical Plan 3

$\pi_{\text{sname}}$

$\sigma_{\text{scity}=\text{Seattle} \land \text{sstate}=\text{WA}}$

$\text{sid} = \text{sid}$

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

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

Clustered Index join

Supply

$\sigma_{\text{pno}=2}$

Supplier

T(Supplier) = 10000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11
Query Optimizer Summary

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

• This is called cost-based optimization