CSE544
Data Management

Lectures 6-8
Query Execution
Announcements

• Review 2 due on Wednesday (Ch. 1&2 only)

• Friday: both HW2 \textit{and} project proposals due

• Next Friday: will meet with each team to discuss the project proposals
Outline

• Architecture of a DBMS

• Steps involved in processing a query

• Main Memory Operators

• Storage

• External Memory Operators
Architecture of DBMS

- Admission Control
- Dispatch and Scheduling
- Process Manager (Section 2)

- Client Communications Manager
  - Local Client Protocols
  - Remote Client Protocols

- Query Parsing and Authorization
- Query Rewrite
- Query Optimizer
- Plan Executor
- Relational Query Processor (Section 4)

- Transactional Storage Manager (Sections 5 & 6)
  - Access Methods
  - Lock Manager

- DDL and Utility Processing
- Buffer Manager
- Log Manager

- Catalog Manager
- Memory Manager
- Administration, Monitoring & Utilities
- Replication and Loading Services
- Batch Utilities
- Shared Components and Utilities (Section 7)
Warning: it will be confusing…

DBMS are monoliths: all components must work together and cannot be isolated

• Good news:
  – Hole system has rich functionality and is efficient

• Bad news:
  – Hard to discuss components in isolation
  – Impossible to use components in isolation
Multiple Processes

Admission Control

Dispatch and Scheduling

Process Manager (Section 2)

Client Communications Manager

Local Client Protocols

Remote Client Protocols

Query Parsing and Authorization

Query Rewrite

Query Optimizer

Plan Executor

Relational Query Processor (Section 4)

Transaction Storage Manager (Sections 5 & 6)

Access Methods

Buffer Manager

Lock Manager

Log Manager

Catalog Manager

Memory Manager

Administration, Monitoring & Utilities

Replication and Loading Services

Batch Utilities

Shared Components and Utilities (Section 7)
Why Multiple Processes

• DBMS listens to requests from clients

• Each request = one SQL command

• Need to handle multiple requests concurrently, hence, multiple processes
Process Models

• Process per DBMS worker

• Thread per DBMS worker

• Process pool

Discuss pro/cons for each model
Outline

• Architecture of a DBMS
• Steps involved in processing a query
• Main Memory Operators
• Storage
• External Memory Operators
Query Optimization
Lifecycle of a Query

SQL query

Parse & Rewrite Query

Select Logical Plan

Select Physical Plan

Query Execution

Disk

Logical plan

Physical plan

Query optimization
Example Database Schema

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

View: Suppliers in Seattle

CREATE VIEW NearbySupp AS
SELECT sno, sname
FROM Supplier
WHERE scity='Seattle' AND sstate='WA'
Example Query

• Find the names of all suppliers in Seattle who supply part number 2

```
SELECT sname FROM NearbySupp
WHERE sno IN ( SELECT sno
FROM Supplies
WHERE pno = 2 )
```
Lifecycle of a Query (1)

• **Step 0: admission control**
  – User connects to the db with username, password
  – User sends query in text format

• **Step 1: Query parsing**
  – Parses query into an internal format
  – Performs various checks using catalog:
    Correctness, authorization, integrity constraints

• **Step 2: Query rewrite**
  – View rewriting, flattening, decorrelation, etc.
View Rewriting, Flattening

Original query:

```
SELECT sname
FROM NearbySupp
WHERE sno IN (SELECT sno
              FROM Supplies
              WHERE pno = 2 )
```

View rewriting
= view inlining
= view expansion
Flattening
= unnesting
View Rewriting, Flattening

Original query:

```sql
SELECT sname
FROM NearbySupp
WHERE sno IN (SELECT sno
               FROM Supplies
               WHERE pno = 2 )
```

Rewritten query:

```sql
SELECT S.sname
FROM Supplier S, Supplies U
WHERE S.scity='Seattle' AND S.sstate='WA'
    AND S.sno = U.sno
    AND U.pno = 2;
```

View rewriting = view inlining
= view expansion
Flattening = unnesting
Decorrelation

```
SELECT  Q.sno
FROM    Supplier Q
WHERE   Q.sstate = 'WA'
        and not exists
            (SELECT *
             FROM Supply P
             WHERE P.sno = Q.sno
             and P.price > 100)
```
SELECT  Q.sno
FROM    Supplier Q
WHERE   Q.sstate = 'WA'
        and not exists
            (SELECT *
             FROM Supply P
             WHERE P.sno = Q.sno
                and P.price > 100)
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and not exists
    (SELECT *
     FROM Supply P
     WHERE P.sno = Q.sno
     and P.price > 100)

SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and Q.sno not in
    (SELECT P.sno
     FROM Supply P
     WHERE P.price > 100)
Decorrelation

Un-nesting

\[
\begin{align*}
\text{(SELECT } & Q.sno \\
\text{FROM } & \text{Supplier Q} \\
\text{WHERE } & \text{Q.sstate = 'WA')} \\
\text{EXCEPT} & \\
\text{(SELECT } & P.sno \\
\text{FROM } & \text{Supply P} \\
\text{WHERE } & \text{P.price > 100)})
\end{align*}
\]

EXCEPT = set difference
Decorrelation

\[
\begin{align*}
\text{(SELECT Q.sno} & \text{ FROM Supplier Q)} \\
\text{WHERE Q.sstate = 'WA')} & \text{ EXCEPT} \\
\text{(SELECT P.sno} & \text{ FROM Supply P)} \\
\text{WHERE P.price > 100)}
\end{align*}
\]
Lifecycle of a Query (2)

• **Step 3: Query optimization**
  – Find an efficient query plan for executing the query
  – We will spend two lectures on this topic

• **A query plan is**
  – *Logical query plan*: an extended relational algebra tree
  – *Physical query plan*: with additional annotations at each node
Relational Algebra Operators

- Union $\cup$
- Intersection $\cap$
- Difference $-$
- Selection $\sigma$
- Projection $\pi$
- Cartesian product $\times$
- Join $\Join$
- Rename $\rho$
- Duplicate elimination $\delta$
- Grouping and aggregation $\gamma$
- Sorting $\tau$

Extended RA
Logical Query Plan

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

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

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

Suppliers \quad Supplies
Query Block

- Most optimizers operate on individual query blocks

- A query block is an SQL query with no nesting
  - Exactly one
    - SELECT clause
    - FROM clause
  - At most one
    - WHERE clause
    - GROUP BY clause
    - HAVING clause
Query Plan For A Block

$$\sigma_{\text{having-condition}}$$

$$\gamma$$ fields, sum/count/min/max(fields)

$$\pi$$ fields

$$\sigma_{\text{where-condition}}$$

join condition

SELECT-PROJECT-JOIN Query
Physical Query Plan

\[ (\text{On the fly})\]
\[ \pi_{\text{sname}} \]
\[ \sigma_{\text{sscity}='Seattle' \land \text{sstate}='WA' \land \text{pno}=2} \]
\[ (\text{On the fly}) \]
\[ \text{Nested loop} \]

\[ \text{Suppliers (File scan)} \]
\[ \text{Supplies (Index lookup)} \]

Algorithm

Physical plan=
Logical plan
+ choice of algorithms
+ choice of access path

Access path
Final Step in Query Processing

- **Step 4: Query execution**
  - How to synchronize operators
  - How to pass data between operators

- **Standard approach:**
  - Iterator interface and
  - Pipelined execution or
  - Intermediate result materialization
Outline

• Architecture of a DBMS
• Steps involved in processing a query
• Main Memory Operators
• Storage
• External Memory Operators
Multiple Processes
Physical Operators

- For each operator, several algorithms
- Main memory or external memory
- Examples:
  - Main memory hash join
  - External memory merge join
  - External memory partitioned hash join
  - Sort-based group by
  - Hash-based group by
Main Memory Algorithms

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

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

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

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

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

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

If \(|R|=|S|=n\), what is the runtime?
### 1. Nested Loop Join

**Logical operator:**

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

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

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

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

Separate chaining:

A (naïve) hash function:

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

Operations:

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

Duplicates OK WHY ??

0 1 2 3 4 5 6 7 8 9

\[ \begin{array}{c}
0 \rightarrow \\
1 \rightarrow 503 \rightarrow 103 \rightarrow 503 \\
2 \rightarrow \\
3 \rightarrow 76 \rightarrow 666 \\
4 \rightarrow \\
5 \rightarrow \\
6 \rightarrow \\
7 \rightarrow \\
8 \rightarrow 48 \\
9 \rightarrow 
\end{array} \]
BRIEF Review of Hash Tables

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

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

• \text{find}(k) = \text{returns the list of all values } \text{v associated to the key } k
2. Hash Join

Logical operator:

\[ \text{Supplier} \bowtie_{s\text{id}=s\text{id}} \text{Supply} \]

\[
\begin{align*}
\text{for } x \text{ in Supplier} & \text{ do} \\
\text{insert}(x.\text{sid}, x) \\
\text{for } y \text{ in Supply} & \text{ do} \\
x = \text{find}(y.\text{sid}); \\
\text{output}(x, y);
\end{align*}
\]
2. Hash Join

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

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

for $y$ in Supply do
$x = \text{find}(y$.sid$)$;
output($x, y$);

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

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

for \( x \) in Supplier do
  insert(\( x.\text{sid}, x \))

for \( y \) in Supply do
  \( x = \text{find}(y.\text{sid}); \)
  output(\( x, y \));

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

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

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

for x in Supplier do
  ????
```

Change join order
2. Hash Join

Logical operator:

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

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

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

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

2. Hash Join

Logical operator:

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

for \(y\) in Supply do
    insert\((y.\text{sid}, y)\)

for \(x\) in Supplier do
    for \(y\) in find\((x.\text{sid})\) do
        output\((x, y)\);

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

Change join order
2. Hash Join

Logical operator:

```
Supplier \Join_{sid=sid} Supply
```

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

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

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

O(n)

But can be O(n^2) why?
2. Hash Join

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

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

Why would we change the order?

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

\( O(n) \)

But can be \( O(n^2) \) why?
2. Hash Join

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

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

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

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

- \(O(n)\)
- But can be \(O(n^2)\) why?

Why would we change the order?

When \(|\text{Supply}| << |\text{Supplier}|\)

Change join order
3. Merge Join

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

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

Logical operator:

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

\[ \text{Sort(Supplier); Sort(Supply); } \]
\[ x = \text{Supplier.first(); } \]
\[ y = \text{Supply.first(); } \]
\[ \text{while } y != \text{NULL do} \]
\[ \text{case: } \]
\[ \quad x.\text{sid} < y.\text{sid}: ??? \]
\[ \quad x.\text{sid} = y.\text{sid}: ??? \]
\[ \quad x.\text{sid} > y.\text{sid}: ??? \]
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

3. Merge Join

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

Sort(Supplier); Sort(Supply);
\(x = \text{Supplier.first}()\);
\(y = \text{Supply.first}()\);
while \(y \neq \text{NULL}\) do
  case:
    \(x.\text{sid} < y.\text{sid}\): \(x = x.\text{next}()\)
    \(x.\text{sid} = y.\text{sid}\): ???
    \(x.\text{sid} > y.\text{sid}\): ???
3. Merge Join

Logical operator:

\( \text{Supplier} \bowtie_{\text{sid}=\text{sid}} \text{Supply} \)

Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
    case:
        x.sid < y.sid: x = x.next()
        x.sid = y.sid: output(x,y); y = y.next();
        x.sid > y.sid: ???
3. Merge Join

Logical operator:

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

Sort(Supplier); Sort(Supply);
\( x = \text{Supplier}.\text{first}(); \)
\( y = \text{Supply}.\text{first}(); \)
while \( y \neq \text{NULL} \) do
  case:
    \( x.\text{sid} < y.\text{sid} \): \( x = x.\text{next}(); \)
    \( x.\text{sid} = y.\text{sid} \): output\((x,y); \ y = y.\text{next}(); \)
    \( x.\text{sid} > y.\text{sid} \): \( y = y.\text{next}(); \)
Supplier($sid, sname, scity, sstate$)
Supply($sid, pno, quantity$)

3. Merge Join

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

Sort(Supplier); Sort(Supply);
$x = Supplier.first();$
$y = Supply.first();$
while $y ! = NULL$ do
  case:
    $x.sid < y.sid: x = x.next()$
    $x.sid = y.sid: output(x,y); y = y.next()$
    $x.sid > y.sid: y = y.next()$;

If $|R|=|S|=n$, what is the runtime?
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

3. Merge Join

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

Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sid < y.sid: x = x.next()
    x.sid = y.sid: output(x, y); y = y.next();
    x.sid > y.sid: y = y.next();

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

Discuss in class
How Do We Combine Them?
How Do We Combine Them?

The *Iterator Interface*

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

```java
interface Operator {
  void open (...);
  Tuple next ();
  void close ();
}
```
Implementing Query Operators with the Iterator Interface

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

    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 ();

}
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();
    }
}
```

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 () {

    }
}
```
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;
    }
}
```

Example

```java
interface Operator {

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

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

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

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

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

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

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

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;
    }
}

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

interface Operator {

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

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

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

Example “on the fly” selection operator

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

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

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

```
interface Operator {

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

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

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

Query plan execution

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

(On the fly)

π_{sname}

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

(Nested loop)

sid = sid

Supplier (File scan)  Supply (File scan)

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

Pipelining

(On the fly)

(open())

(On the fly)

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

(Nested loop)

(sid = sid)

Supplier
(File scan)

Supply
(File scan)

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

(On the fly)

(On the fly)

(Nested loop)

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

\[
\begin{align*}
\text{π}_{\text{sname}} & \text{} \\
\text{σ}_{\text{scity} = \text{Seattle} \text{ and } \text{sstate} = \text{WA} \text{ and } \text{pno}=2} & \text{} \\
\text{sid} = \text{sid} & \text{}
\end{align*}
\]

\[ \text{Discuss: open/next/close for nested loop join} \]
Nested loop join

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

\textbf{Pipelining}

(On the fly)

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

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

(Nested loop)

\text{sid = sid}

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

(Nested loop)

Discuss: open/next/close for nested loop join

$\pi_{\text{sname}}$

$\text{sid = sid}$

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

Pipelining

(On the fly)

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

(Nested loop)

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

Pipelining

(On the fly)

(On the fly)

(File scan)

(File scan)

(next())

π_{sname}

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

Discuss: open/next/close for nested loop join

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

(On the fly)

(On the fly) σ scity='Seattle' and sstate='WA' and pno=2

(Nested loop)

Discuss: open/next/close for nested loop join

π sname

next()
Pipelining

(On the fly)

(On the fly)

(Nested loop)

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

(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)

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

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

\( \text{next() } \)

\( \text{next() } \)

\( \text{next() } \)

\( \text{next() } \)

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

Supplier (File scan)

Supply (File scan)
(On the fly)

(On the fly)

(Nested loop)

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

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

(On the fly)

\(\pi_{sname}\)

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

(Nested loop)

\(\text{next}\()\quad \text{next}\()\quad \text{next}\()\quad \text{next}\()

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

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

(On the fly)

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

(On the fly)

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

(Hash Join)

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

Supplier (File scan)

Supply (File scan)

Discuss hash-join in class

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)

(On the fly)

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

(Hash Join)

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

Tuples from here are “blocked”

Discuss hash-join in class

Supplier (File scan)

Supply (File scan)
Discuss hash-join in class

Tuples from here are “blocked”

Tuples from here are pipelined

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

\(\text{On the fly}\)

\(\text{On the fly}\)

\(\text{Hash Join}\)

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

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

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

\(\text{File scan}\)

\(\text{File scan}\)
Blocked Execution

(On the fly)

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

(Merge Join)

$\pi_{\text{sname}}$

Supplier (File scan)

Supply (File scan)

Discuss merge-join in class
Blocked Execution

(On the fly)

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

(On the fly)

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

(Merge Join)

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

Supplier
(File scan)

Supply
(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
Outline

• Architecture of a DBMS
• Steps involved in processing a query
• Main Memory Operators
  • Storage
• External Memory Operators
Multiple Processes

- Admission Control
- Dispatch and Scheduling
- Process Manager (Section 2)
- Local Client Protocols
- Remote Client Protocols
- Catalog Manager
- Memory Manager
- Administration, Monitoring & Utilities
- Replication and Loading Services
- Batch Utilities
- Shared Components and Utilities (Section 7)
- Client Communications Manager
- Query Parsing and Authorization
- Query Rewrite
- Query Optimizer
- Plan Executor
- DDL and Utility Processing
- Relational Query Processor (Section 4)
- Access Methods
- Buffer Manager
- Lock Manager
- Log Manager
- Transactional Storage Manager (Sections 5 & 6)
The Mechanics of Disk

Mechanical characteristics:
- Rotation speed (5400RPM)
- Number of platters (1-30)
- Number of tracks (<=10000)
- Number of bytes/track($10^5$)

Unit of read or write: disk block
Once in memory: page
Typically: 4k or 8k or 16k
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
Disk Access Characteristics

• **Disk latency**
  – Time between when command is issued and when data is in memory
  – Equals = seek time + rotational latency

• **Seek time** = time for the head to reach cylinder
  – 10ms – 40ms

• **Rotational latency** = time for the sector to rotate
  • Rotation time = 10ms
  • Average latency = 10ms/2

• **Transfer time** = typically 40MB/s

**Basic factoid:** disks always read/write an entire block at a time
Buffer Management in a DBMS

Page Requests from Higher Levels

- Data must be in RAM for DBMS to operate on it!
- Table of <frame#, pageid> pairs is maintained

choice of frame dictated by replacement policy
Buffer Manager

Needs to decide on page replacement policy

• LRU
• Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.
Arranging Pages on Disk

A disk is organized into blocks (a.k.a. pages)
• blocks on same track, followed by
• blocks on same cylinder, followed by
• blocks on adjacent cylinder

A file should (ideally) consists of sequential blocks on disk, to minimize seek and rotational delay.

For a sequential scan, pre-fetching several pages at a time is a big win!
Issues

• Managing free blocks

• File Organization

• Represent the records inside a page

• Represent attributes inside the records
Managing Free Blocks

• Linked list of free blocks

• Directory of pages

• Bit map
File Organization

Linked list of pages:

Header page

Data page

Data page

Data page

Full pages

Data page

Data page

Data page

Pages with some free space
Better: directory of pages
File Organization

• Bit map: store compactly the free/full status of each page
Records into a Page

Issues to consider

• 1 page = fixed size (e.g. 8KB)

• Records:
  – Fixed length
  – Variable length

• Record id = RID
  – Typically RID = (PageID, SlotNumber)
Records into a Page

Fixed-length records: packed representation

One page

Rec 1  Rec 2  Rec N

Free space  N

Problems ?
Records into a Page

Variable-length records

Free space

Slot directory
**Record Formats: Fixed Length**

### Product(pid, name, descr, maker)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>Product ID</td>
</tr>
<tr>
<td>name</td>
<td>Product Name</td>
</tr>
<tr>
<td>descr</td>
<td>Description</td>
</tr>
<tr>
<td>maker</td>
<td>Maker</td>
</tr>
</tbody>
</table>

- Information about field types same for all records in a file; stored in *system catalogs*.
- Finding *i*’th field requires scan of record.
- Note the importance of schema information!

Base address (B)  Address = B+L1+L2
Need the header because:

- The schema may change for a while new+old may coexist
- Records from different relations may coexist
Variable Length Records

Other header information

header pid name descr maker

length

Place the fixed fields first: F1
Then the variable length fields: F2, F3, F4
Null values take 2 bytes only
Sometimes they take 0 bytes (when at the end)
BLOB

- Binary large objects
- Supported by modern database systems
- E.g. images, sounds, etc.
- Storage: attempt to cluster blocks together

CLOB = character large object
- Supports only restricted operations
File Organizations

• **Heap** (random order) files: Suitable when typical access is a file scan retrieving all records.

• **Sequential file** (sorted): Best if records must be retrieved in some order, or by a `range`.

• **Indexe**: Data structures to organize records via trees or hashing.
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
Different keys:

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

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

Data File **Student**

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

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

Index Student_fName on Student.fName

Data File Student

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

Index can be: Dense only
Index Organization

- Hash table

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

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

d = 2

Find the key 40
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
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N text, P int);
Getting Practical: Creating Indexes in SQL

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

CREATE INDEX V1 ON V(N);

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

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

CREATE INDEX V1 ON V(N)

CREATE INDEX V2 ON V(P, M)

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

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

What does this mean?

```
select * from V where P=55 and M=77
```
Getting Practical: Creating Indexes in SQL

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

CREATE INDEX V1 ON V(N)

CREATE INDEX V2 ON V(P, M)

What does this mean?

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

select * from V
where P=55
Getting Practical: Creating Indexes in SQL

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

What does this mean?

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

select *
from V
where P=55

select *
from V
where M=77
Creating Indexes in SQL

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

```
select * from V where P=55
select * from V where M=77
select * from V where P=55 and M=77
```
Getting Practical: Creating Indexes in SQL

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

CREATE INDEX V1 ON V(N);

CREATE INDEX V2 ON V(P, M);

What does this mean?

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

yes

select * from V where P=55

yes

select * from V where M=77

yes
Getting Practical: Creating Indexes in SQL

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

CREATE INDEX V1 ON V(N);

CREATE INDEX V2 ON V(P, M);

What does this mean?

select *
from V
where P=55

no

select *
from V
where M=77

no

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

yes
Getting Practical: Creating Indexes in SQL

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

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

no

yes

CSE 544 - Winter 2020
Which Indexes?

• How many indexes could we create?

• Which indexes should we create?
Which Indexes?

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

This is called the *Index Selection Problem*

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

V(M, N, P);

Your workload is this
100000 queries:

SELECT * 
FROM V 
WHERE N=?

100 queries:

SELECT * 
FROM V 
WHERE P=?

What indexes?
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)
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?
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)
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 (?, ?, ?)

What indexes?
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)?
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?
Index Selection Problem 4

V(M, N, P);

Your workload is this
1000 queries:

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

100000 queries:

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

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

- Point queries
  - Hash- or B⁺-tree index
- Clustered or not
  - Range queries
  - B⁺-tree index
  - Clustered
To Cluster or Not

Remember:

• Rule of thumb:
  Random reading 1-2% of file ≈ 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<?
Percentage tuples retrieved

<table>
<thead>
<tr>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>

Sequential scan

Clustered index

SELECT *
FROM R
WHERE R.K>? and R.K<?
\texttt{SELECT \ast}
\texttt{FROM R}
\texttt{WHERE R.K>? and R.K<?}
Outline

• Architecture of a DBMS
• Steps involved in processing a query
• Main Memory Operators
• Storage
  • External Memory Operators
Architecture
Cost Parameters

• In database systems the data is on disk
• Parameters:
  – \( B(R) = \# \text{ of blocks (i.e., pages) for relation } R \)
  – \( T(R) = \# \text{ of tuples in relation } R \)
  – \( V(R, a) = \# \text{ of distinct values of attribute } a \)
  – \( M = \# \text{ pages available in main memory} \)
• Cost = total number of I/Os
• Convention: writing the final result to disk is \textit{not included}
Cost Parameters

Supplier\((sid, sname, scity, sstate)\)

Supply\((sid, pno, quantity)\)

Block size = 8KB

- \(B(\text{Supplier}) = 1,000,000\) blocks = 8GB
- \(T(\text{Supplier}) = 50,000,000\) records ~ 50 / block
- \(V(\text{Supplier}, \text{sid}) = \)
- \(V(\text{Supplier}, \text{sname}) = \)
- \(V(\text{Supplier}, \text{scity}) = \)
- \(V(\text{Supplier}, \text{sstate}) = \)
- \(M = \)
Cost Parameters

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

Block size = 8KB

- B(Supplier) = 1,000,000 blocks = 8GB
- T(Supplier) = 50,000,000 records ~ 50 / block
- V(Supplier, sid) = 50,000,000 why?
  - V(Supplier, sname) =
  - V(Supplier, scity) =
  - V(Supplier, sstate) =
- M =
Cost Parameters

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

Block size = 8KB

- B(Supplier) = 1,000,000 blocks = 8GB
- T(Supplier) = 50,000,000 records ~ 50 / block
- V(Supplier, sid) = 50,000,000 why?
- V(Supplier, sname) = 40,000,000 meaning?
- V(Supplier, scity) =
- V(Supplier, sstate) =
- M =
Cost Parameters

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

Block size = 8KB

- B(Supplier) = 1,000,000 blocks = 8GB
- T(Supplier) = 50,000,000 records ~ 50 / block
- V(Supplier, sid) = 50,000,000 why?
- V(Supplier, sname) = 40,000,000 meaning?
- V(Supplier, scity) = 860
- V(Supplier, sstate) =
- M =
Cost Parameters

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

Block size = 8KB

- \( B(\text{Supplier}) = 1,000,000 \) blocks = 8GB
- \( T(\text{Supplier}) = 50,000,000 \) records ~ 50 / block
- \( V(\text{Supplier, sid}) = 50,000,000 \) why?
- \( V(\text{Supplier, sname}) = 40,000,000 \) meaning?
- \( V(\text{Supplier, scity}) = 860 \)
- \( V(\text{Supplier, sstate}) = 50 \) why?
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Cost Parameters

Supplier(sid, sname, scity, sstate)
Block size = 8kB

- B(Supplier) = 1,000,000 blocks = 8GB
- T(Supplier) = 50,000,000 records ~ 50 / block
- V(Supplier, sid) = 50,000,000 why?
- V(Supplier, sname) = 40,000,000 meaning?
- V(Supplier, scity) = 860
- V(Supplier, sstate) = 50 why?
- M = 10,000,000 = 80GB why so little?
Index Based Selection

Selection on equality: $\sigma_{a=v}(R)$
$V(R, a) = \# \text{ of distinct values of attribute } a$

Cost of index-based selection:

- Clustered index on $a$:
- Unclustered index on $a$: 

CSE 544 - Winter 2020
Index Based Selection

Selection on equality: \( \sigma_{a=v}(R) \)
\( V(R, a) = \# \text{ of distinct values of attribute } a \)
Assumptions:
• Values are uniformly distributed
• Ignore the cost of reading the index (why?)

Cost of index-based selection:
• Clustered index on a:
• Unclustered index on a:
Index Based Selection

Selection on equality: $\sigma_{a=v}(R)$
$V(R, a) =$ # of distinct values of attribute $a$

Assumptions:
- Values are uniformly distributed
- Ignore the cost of reading the index (why?)

Cost of index-based selection:
- **Clustered index on $a$:** $\text{cost} = \frac{B(R)}{V(R,a)}$
- **Unclustered index on $a$:** $\text{cost} = \frac{T(R)}{V(R,a)}$
Index Based Selection

**Example:**
- Table scan (assuming \( R \) is clustered)

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

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

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

- Example:
  - Table scan (assuming R is clustered)
    - \( B(R) = 2,000 \) I/Os
  - Index based selection
    - If index is clustered:
    - If index is unclustered:

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

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

- Example:
  - Table scan (assuming R is clustered)
    - $B(R) = 2,000$ I/Os
  - Index based selection
    - If index is clustered: $B(R)/V(R,a) = 100$ I/Os
    - If index is unclustered:

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

Example:

- Table scan (assuming R is clustered)
  - \( B(R) = 2,000 \) I/Os

- Index based selection
  - If index is clustered: \( B(R)/V(R,a) = 100 \) I/Os
  - If index is unclustered: \( T(R)/V(R,a) = 5,000 \) I/Os

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

- Example:
  - Table scan (assuming R is clustered)
    - B(R) = 2,000 I/Os
  - Index based selection
    - If index is clustered: B(R)/V(R,a) = 100 I/Os
    - If index is unclustered: T(R)/V(R,a) = 5,000 I/Os
- Lesson
  - Don’t build unclustered indexes when V(R,a) is small!

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

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

The 2% rule!
External Memory Joins

Recall standard main memory algorithms:

• Hash join
• Nested loop join
• Sort-merge join

Review in class
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:
  - Assuming $R$ is clustered
  - If index on $S$ is clustered: $B(R) + \frac{T(R)B(S)}{V(S,a)}$
  - If index on $S$ is unclustered: $B(R) + \frac{T(R)T(S)}{V(S,a)}$
One Pass Hash Join

Hash join: $R \bowtie S$

- Scan $R$, build buckets in main memory
- Then scan $S$, probe hash table to join

- Cost: $B(R) + B(S)$

- One pass algorithm when $B(R) \leq M$
Nested Loop Joins

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

\[
\begin{align*}
\text{for each tuple } r \text{ in } R & \text{ do} \\
& \text{for each tuple } s \text{ in } S \text{ do} \\
& \quad \text{if } r \text{ and } s \text{ join then output } (r,s)
\end{align*}
\]

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

\[
\text{for each page of tuples } r \text{ in } R \text{ do}
\]
\[
\text{for each page of tuples } s \text{ in } S \text{ do}
\]
\[
\text{for all pairs of tuples}
\]
\[
\text{if } r \text{ and } s \text{ join then output } (r,s)
\]

• Cost: $B(R) + B(R)B(S)$
Nested Loop Joins

• We can be much more clever

• How would you compute the join in the following cases? What is the cost?
  – B(R) = 1000, B(S) = 2, M = 4
  – B(R) = 1000, B(S) = 3, M = 4
  – B(R) = 1000, B(S) = 6, M = 4
Nested Loop Joins

- Block Nested Loop Join
- Group of (M-2) pages of S is called a “block”

```plaintext
for each (M-2) pages ps of S do
  for each page pr of R do
    for each tuple s in ps
      for each tuple r in pr do
        if r and s join then output(r,s)
```

Main memory hash-join

\[(M-1)ps \bowtie pr\]
Nested Loop Joins

[Diagram showing the process of nested loop joins, including input buffer for R, output buffer, hash table for block of S (M-2 pages), and join result.]
Nested Loop Joins

Cost of block-based nested loop join

• Read S once: \( B(S) \)

• Outer loop runs \( B(S)/(M-2) \) times, each iteration reads the entire R: \( B(S)B(R)/(M-2) \)

• Total cost: \( B(S) + B(S)B(R)/(M-2) \)
Nested Loop Joins

Cost of block-based nested loop join

• Read S once: \( B(S) \)

• Outer loop runs \( B(S)/(M-2) \) times, each iteration reads the entire R: \( B(S)B(R)/(M-2) \)

• Total cost: \( B(S) + B(S)B(R)/(M-2) \)
Nested Loop Joins

Cost of block-based nested loop join

- Read S once: \( B(S) \)
- Outer loop runs \( \frac{B(S)}{(M-2)} \) times, each iteration reads the entire R: \( \frac{B(S)B(R)}{(M-2)} \)
- Total cost: \( B(S) + \frac{B(S)B(R)}{(M-2)} \)

Iterate over the smaller relation first!
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:
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: \( B(R) + B(S) \)
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 \( B(S) + B(R) \leq M \)
Grouping

\[ \gamma_{\text{department}, \sum\text{quantity}} \ (\text{Product}) \]

In class: describe a one-pass algorithms.

Cost=?
Two-Pass Algorithms

• When data is larger than main memory, need two or more passes

• Two key techniques
  – Hashing
  – Sorting
Two Pass Algorithms Based on Hashing

- Idea: partition a relation $R$ into buckets, on disk
Two Pass Algorithms Based on Hashing

- Idea: partition a relation R into buckets, on disk
Two Pass Algorithms Based on Hashing

- Idea: partition a relation R into buckets, on disk
Two Pass Algorithms Based on Hashing

• Idea: partition a relation R into buckets, on disk
Two Pass Algorithms Based on Hashing

- Idea: partition a relation R into buckets, on disk
Two Pass Algorithms Based on Hashing

- Idea: partition a relation $R$ into buckets, on disk

\[ B(R) \rightarrow h \rightarrow \text{hash function} \rightarrow \frac{M-1}{M} \rightarrow \text{OUTPUT} \rightarrow \text{Disk} \]
Two Pass Algorithms Based on Hashing

- Idea: partition a relation $R$ into buckets, on disk

![Diagram](image_url)
Two Pass Algorithms Based on Hashing

- Idea: partition a relation R into buckets, on disk
Two Pass Algorithms Based on Hashing

- Idea: partition a relation $R$ into buckets, on disk
Two Pass Algorithms Based on Hashing

- Idea: partition a relation R into buckets, on disk
Two Pass Algorithms Based on Hashing

- Idea: partition a relation R into buckets, on disk
- Each bucket has size approx. $\frac{B(R)}{M}$
Two Pass Algorithms Based on Hashing

- Idea: partition a relation $R$ into buckets, on disk
- Each bucket has size approx. $B(R)/M$

- Does each bucket fit in main memory?
Two Pass Algorithms Based on Hashing

- Idea: partition a relation $R$ into buckets, on disk
- Each bucket has size approx. $\frac{B(R)}{M}$

Does each bucket fit in main memory?
- Yes when: $\frac{B(R)}{M} \leq M$, i.e. $B(R) \leq M^2$
Hash Based Algorithms for $\gamma$

- Recall: $\gamma(R) =$ grouping and aggregation

- Step 1. Partition $R$ into buckets
- Step 2. Apply $\gamma$ to each bucket

- Cost: $3B(R)$
- Assumption: $B(R) \leq M^2$
Partitioned (Grace) Hash Join

R \bowtie S

• Step 1:
  – Hash S into M-1 buckets
  – Send all buckets to disk

• Step 2
  – Hash R into M-1 buckets
  – Send all buckets to disk

• Step 3
  – Join every pair of buckets
Partitioned Hash Join R

\( R \bowtie S \)

- Partition both relations using hash fn \( h \)
Partitioned Hash Join

\[ R \bowtie S \]

- Read in partition of S, hash it using \( h2 \) (\( \neq h \))
- Scan same partition of R, search for matches

Input buffer for \( R_i \)

B main memory buffers

Join Result

Disk

Hash table for partition \( S_i \) (\(< M-1 \) pages)

\( h2 \)

\( h2 \)

disk fn \( h2 \)

Partitions of R & S
Partitioned Hash Join

- Cost: $3B(R) + 3B(S)$
- Assumption: $\min(B(R), B(S)) \leq M^2$
Hybrid Hash Join Algorithm

• Assume we have **extra memory available**

• Partition S into k buckets
  - t buckets $S_1, \ldots, S_t$ stay in memory
  - k-t buckets $S_{t+1}, \ldots, S_k$ to disk

• Partition R into k buckets
  - First t buckets join immediately with S
  - Rest k-t buckets go to disk

• Finally, join k-t pairs of buckets:
  $(R_{t+1}, S_{t+1}), (R_{t+2}, S_{t+2}), \ldots, (R_k, S_k)$
Hybrid Hash Join Algorithm

How to choose k and t?

• The first t buckets must fit in M:
  \[ \frac{t}{k} \times B(S) \leq M \]
Hybrid Hash Join Algorithm

How to choose k and t?

- The first t buckets must fit in M: \( \frac{t}{k} \times B(S) \leq M \)
- Need room for k-t additional pages: \( k-t \leq M \)
Hybrid Hash Join Algorithm

How to choose k and t?

- The first t buckets must fit in M: \( \frac{t}{k} \times B(S) \leq M \)
- Need room for k-t additional pages: \( k-t \leq M \)
- Thus: \( \frac{t}{k} \times B(S) + k-t \leq M \)
**Hybrid Hash Join Algorithm**

**How to choose k and t?**

- The first $t$ buckets must fit in M: $\frac{t}{k} \times B(S) \leq M$
- Need room for $k-t$ additional pages: $k-t \leq M$
- Thus: $\frac{t}{k} \times B(S) + k-t \leq M$

Assuming $\frac{t}{k} \times B(S) \gg k-t$: $\frac{t}{k} = \frac{M}{B(S)}$
Hybrid Hash Join Algorithm

- How many I/Os?

- Cost of partitioned hash join: $3B(R) + 3B(S)$

- Hybrid join saves 2 I/Os for a $t/k$ fraction of buckets
  - Hybrid join saves $2t/k(B(R) + B(S))$ I/Os

Cost: $(3-2t/k)(B(R) + B(S)) = (3-2M/B(S))(B(R) + B(S))$
External Sorting

• Problem: Sort a file of size $B$ with memory $M$

• Where we need this:
  – ORDER BY in SQL queries
  – Several physical operators
  – Bulk loading of $B+$-tree indexes.

• Will discuss only 2-pass sorting, for when $B \leq M^2$
External Merge-Sort: Step 1

- Phase one: load M pages in memory, sort

- Main memory
  - Size M pages
  - Runs of length M
  - \#Runs = B(R)/M

Disk

Disk
External Merge-Sort: Step 2

- Merge $M - 1$ runs into a new run
- Result: runs of length $M (M - 1) \approx M^2$

Assuming $B \leq M^2$, we are done
External Merge-Sort

• Cost:
  – Read+write+read = 3B(R)
  – Assumption: B(R) <= M^2

• Other considerations
  – In general, a lot of optimizations are possible
Two-Pass Algorithms Based on Sorting

Grouping: $\gamma_a, \text{sum}(b)$ (R)

Sort, then compute the sum(b) for each group of a’s

- Step 1: sort chunks of size M, write
  - cost $2B(R)$
- Step 2: merge M-1 runs, combining groups by addition
  - cost $B(R)$
- Total cost: $3B(R)$, Assumption: $B(R) \leq M^2$
Two-Pass Algorithms Based on Sorting

Join $R \bowtie S$

- Start by creating initial runs of length $M$, for $R$ and $S$:
  - Cost: $2B(R)+2B(S)$
- Merge (and join) $M_1$ runs from $R$, $M_2$ runs from $S$:
  - Cost: $B(R)+B(S)$
- Total cost: $3B(R)+3B(S)$
- Assumption:
  - $R$ has $M_1=B(R)/M$ runs, $S$ has $M_2=B(S)/M$ runs
  - $M_1 + M_2 \leq M$
  - Hence: $B(R)+B(S) \leq M^2$
Summary of External Join Algorithms

- Block Nested Loop Join: $B(R) + B(R)\times B(S)/M$

- Hybrid Hash Join: $(3 - 2M/B(S))(B(R) + B(S))$
  Assuming $t/k \times B(S) >> k-t$

- Sort-Merge Join: $3B(R) + 3B(S)$
  Assuming $B(R) + B(S) \leq M^2$

- Index Nested Loop Join: $B(R) + T(R)B(S)/V(S,a)$
  Assuming $R$ is clustered and $S$ has clustered index on a