CSE544
Data Management
Lectures 4-6
Query Execution
Announcements

• HW1 due on Friday

• No lecture on Monday

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

• Project groups by next Friday (email to me)
Outline for the Next 3 Lectures

- Architecture of a DBMS
- Steps involved in processing a query
- Main Memory Operators
- Storage
- External Memory Operators
Architecture of DBMS
DBMS are monoliths: components cannot be isolated

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

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

- Local Client Protocols
- Remote Client Protocols
- Client Communications Manager
- Query Parsing and Authorization
- Query Rewrite
- Query Optimizer
- Plan Executor
- Relational Query Processor (Section 4)
- Access Methods
- Buffer Manager
- Lock Manager
- Log Manager
- Transactional Storage Manager (Sections 5 & 6)
- 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

• Handles multiple requests concurrently; multiple processes
Process Models

• Process per DBMS worker
• Thread per DBMS worker
• Process pool

Next week’s review:
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

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

SQL query

Logical plan
Physical plan

Query optimization
Example Database Schema

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 sno, sname FROM NearbySupp
WHERE sno IN ( SELECT sno
                FROM Supply
                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:

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

View rewriting
= view inlining
= view expansion
Flattening
= unnesting

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

NearbySupp(sno, sname)
View Rewriting, Flattening

Original query:

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

Rewritten query:

```
SELECT S.sno, S.sname
FROM Supplier S, Supply U
WHERE S.scity='Seattle' AND S.sstate='WA'
AND S.sno = U.sno
AND U.pno = 2;
```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

Decorrelation

Find all suppliers in ‘WA’ that supply *only* parts under $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)

Find all suppliers in ‘WA’ that supply only parts under $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

De-Correlation
(SELECT Q.sno FROM Supplier Q WHERE Q.sstate = 'WA')
EXCEPT
(SELECT P.sno FROM Supply P WHERE P.price > 100)
EXCEPT = set difference

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)

Un-nesting
(SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA')
EXCEPT
(SELECT P.sno
FROM Supply P
WHERE P.price > 100)

Decorrelation
Lifecycle of a Query (2)

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

- **A query plan is**
  - **Logical query plan**: a relational algebra tree
  - **Physical query plan**: add specific algorithms
Five Basic Relational Operators

- **Selection**: $\sigma_{\text{condition}}(S)$
- **Projection**: $\pi_{\text{list-of-attributes}}(S)$
- **Union**: $(\cup)$
- **Set difference**: $(\neg)$,
- **Cross-product/cartesian product**: $(\times)$,
  Join: $R \bowtie_\theta S = \sigma_\theta(R \times S)$

Other operators: semi-join, anti-semijoin
Extended Operators of Relational Algebra

• **Duplicate elimination (δ)**
  – Convert a bag to a set
  – Can be expressed as a group-by γ

• **Group-by/aggregate (γ)**
  – Example: γpcolor, max(psize)→m, avg(psize)→s(Part)
  – Min, max, sum, average, count
  – Partitions tuples of a relation into “groups”
  – Aggregates can then be applied to groups

• **Sort operator (τ)**
Logical Query Plan

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

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

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

Supplier

Suply

CSE 544 - Winter 2021
Logical Query Plan

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

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

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

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

Algorithm

Supplier (File scan)

Suply (Index lookup)

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

Physical Query Plan

\( \text{Supplier(sno,sname,scity,sstate)} \)
\( \text{Part(pno,pname,psize,pcolor)} \)
\( \text{Supply(sno,pno,price)} \)
Final Step in Query Processing

• **Step 4: Query execution**
  – Choice of algorithm
  – How to pass data between operators, e.g. materialized, or pipelined
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
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 $\bowtie_{\text{sid} = \text{sid}}$ 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:

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

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

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

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

Separate chaining:

A (naïve) hash function:

$$h(x) = x \text{ mod } 10$$

Duplicates OK

WHY??
BRIEF Review of Hash Tables

Separate chaining:

A (naïve) hash function:
\[ h(x) = x \mod 10 \]

Operations:
\[ \text{find}(103) = ?? \]

Duplicates OK
WHY ??
BRIEF Review of Hash Tables

Separate chaining:

A (naïve) hash function:

$$h(x) = x \mod 10$$

Operations:

find(103) = ??
BRIEF Review of Hash Tables

Separate chaining:

A (naïve) hash function:

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

Operations:

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

Duplicates OK WHY ??
BRIEF Review of Hash Tables

Separate chaining:

A (naïve) hash function:

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

Operations:

\[ \text{find}(103) = ?? \]
\[ \text{insert}(488) = ?? \]
2. Hash Join

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

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

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

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

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?
2. Hash Join

Logical operator:

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

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

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

$O(n)$
2. Hash Join

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

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

for x in Supplier do
  ????
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);

Change join order
2. Hash Join

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

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

If \(|R| = |S| = n\), what is the runtime?
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
    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:
\[ \text{Supplier} \bowtie_{\text{sid} = \text{sid}} \text{Supply} \]

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?

\[ O(n) \]

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

Logical operator:

Supplier $\Join_{\text{sid} = \text{sid}}$ 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);
\( x = \text{Supplier}.\text{first}(); \)
\( y = \text{Supply}.\text{first}(); \)
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.sid = y.sid: ???
    x.sid > y.sid: ???
3. Merge Join

Logical operator:

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

Sort(Supplier); Sort(Supply);
\( x = \) Supplier.first();
\( y = \) Supply.first();
while \( y \neq \) 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:

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

Sort(Supplier); Sort(Supply);

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

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

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

\hspace{1em} case:

\hspace{2em} \( x.\text{sid} < y.\text{sid} : x = x.\text{next}(); \)

\hspace{2em} \( x.\text{sid} = y.\text{sid} : \text{output}(x,y); y = y.\text{next}(); \)

\hspace{2em} \( x.\text{sid} > y.\text{sid} : y = y.\text{next}(); \)
3. Merge Join

Logical operator:

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

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

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

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

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

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

• Project teams due by Friday (email to me)

• HW2 posted, we use Snowflake
  – Consider using Snowflake in your project!

• Architecture paper was due today
Discuss Architecture Paper

[Diagram showing different components of architecture including 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), Query Parsing and Authorization, Query Rewrite, Query Optimizer, Plan Executor, Relational Query Processor (Section 4), Access Methods, Buffer Manager, Lock Manager, Log Manager, Transactional Storage Manager (Sections 5 & 6), DDL and Utility Processing]
Recap: 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

```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

```java
interface Operator {

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

}
```
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 ();

    close ();
}

class Select implements 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 ();
}
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

definition

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

definition
### 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 () {
        // ... (implement selection logic)
    }
}
```
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

class Select implements Operator {
    void open (Predicate p, Operator c) {
        this.p = p; this.c = c; c.open();
    }
    Tuple next () {
        boolean found = false;
        Tuple r = null;
        while (!found) {
            r = c.next();
            if (r == null) break;
            found = p(r);
        }
    }
}

interface Operator {

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

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

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

}
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

class Select implements Operator {
    void open (Predicate p, Operator c) {
        this.p = p; this.c = c; c.open();
    }
    Tuple next () {
        boolean found = false;
        Tuple r = null;
        while (!found) {
            r = c.next();
            if (r == null) break;
            found = p(r);
        }
        return r;
    }
}

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

Example “on the fly” selection operator

```java
interface Operator {

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

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

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

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)

Pipelining

(On the fly)

π\text{\text{\textit{sname}}}

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

(Nested loop)

sid = sid

Supply (File scan)  Supplier (File scan)

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

Pipelining

(On the fly)

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

(On the fly)

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

(Nested loop)

sid = sid

Supply (File scan)

Supplier (File scan)

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

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

**Pipelining**

(On the fly)

(On the fly)

(On the fly)

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

\(\pi_{\text{surname}}\)

(Concerns: open/next/close for nested loop join)

(sid = sid)

Supply (File scan)

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

(On the fly)

(On the fly)

(Discuss: open/next/close for nested loop join)

(Nested loop)
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

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

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

sid = sid

open()

open()

open()

open()

open()

open()

open()

open()

open()
Pipelining

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

(On the fly)

next()

(On the fly)

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

(On the fly)

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

(Nested loop)

sid = sid

Supply (File scan)

Supplier (File scan)

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

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

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

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

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

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

(On the fly)

(On the fly)

(Nested loop)

Discuss: open/next/close for nested loop join

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

Discuss: open/next/close for nested loop join

(On the fly)
next()

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

(Nested loop)
sid = sid
next()

Supply (File scan)
next()

Supply (File scan)
next()

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

(On the fly)

(On the fly)

(On the fly)

(Nested loop)

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

\( \pi \text{sname} \)

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

Supply (File scan)

Supplier (File scan)

Discussion: open/next/close for nested loop join

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

(On the fly)

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

(Nested loop)

Supply (File scan)

Next()

Next()

Next()

Next()

Next()

Next()

Supplier (File scan)

Next()

Next()

Next()

Next()

Discuss: open/next/close for nested loop join

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

Supply($sid$, $pno$, quantity)
Discuss hash-join in class

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

(On the fly)

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

(Hash Join)

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

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

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

\( \text{Supplier} \) (File scan)
Discuss hash-join in class

(On the fly)

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

(Hash Join)

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

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

Supply (File scan)

Supplier (File scan)

Tuples from here are “blocked”
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Pipelining

(On the fly)

(On the fly)

(Hash Join)

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

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

Tuples from here are “blocked”

Tuples from here are pipelined

Supply (File scan)

Supplier (File scan)

Discuss hash-join in class
Blocked Execution

(On the fly)

(On the fly)

(Merge Join)

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

Discuss merge-join in class

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

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

Supply (File scan)

Supplier (File scan)

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

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

**Merge Join**

*On the fly*

\[ \pi_{sname} \]

*On the fly*

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

**Merge Join**

\[ \text{Supply} \]

\[ \text{Supplier} \]

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

*File scan*

*File scan*

*Blocked*
Pipeline v.s. Blocking

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

• Blocking
  – Compute and store on disk entire subplan
  – Advantage: needs less memory
  – Disadvantage: slower
Outline

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

[Diagram showing various processes and components related to database management systems, with a focus on the Buffer Manager.]
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>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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

**Basic fact:** Disks always read/write an entire block at a time
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 fact**: disks access MUCH slower than main memory
• Data must be in RAM for DBMS to operate on it!
• Table of <frame#, pageid> pairs is maintained
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!
Storing Records On Disk

- Page format: records inside a page
- Record format: attributes inside a record
- File Organization
Page Format

- 1 page = 1 disk block = fixed size (e.g. 8KB)
- Records:
  - Fixed length
  - Variable length
- Record id = RID
  - Typically RID = (PageID, SlotNumber)

Need RID’s for indexes and for transactions
Page Format Approach 1

Fixed-length records: packed representation
Divide page into slots. Each slot can hold one tuple
Record ID (RID) for each tuple is (PageID,SlotNb)

How do we insert a new record?
Page Format Approach 1

Fixed-length records: packed representation
Divide page into **slots**. Each slot can hold one tuple
Record ID (RID) for each tuple is (PageID, SlotNb)

<table>
<thead>
<tr>
<th>Slot₁</th>
<th>Slot₂</th>
<th>Slotₙ</th>
<th>Slotₙ₊₁</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Free Sp.</strong></td>
</tr>
</tbody>
</table>

How do we insert a new record?

Number of records
Page Format Approach 1

Fixed-length records: packed representation
Divide page into slots. Each slot can hold one tuple
Record ID (RID) for each tuple is (PageID, SlotNb)

How do we insert a new record?

How do we delete a record?
Page Format Approach 1

Fixed-length records: packed representation
Divide page into slots. Each slot can hold one tuple
Record ID (RID) for each tuple is (PageID,SlotNb)

How do we insert a new record?

How do we delete a record? Cannot remove record (why?)

How do we handle variable-length records?
Page Format Approach 2

Can handle variable-length records
Can move tuples inside a page without changing RIDs
RID is (PageID, SlotID) combination

Header contains slot directory
+ Need to keep track of nb of slots
+ Also need to keep track of free space (F)
Record Formats

Fixed-length records => Each field has a fixed length (i.e., it has the same length in all the records)

| Field 1 | Field 2 | ... | ... | Field K |

Information about field lengths and types is in the catalog
Record Formats

Variable length records

Field 1  Field 2  ...  ...  Field K

Remark: NULLS require no space at all (why?)
Announcements

• Project teams were due last week

• PAX paper review due on Wednesday

• HW2 Snowflake due on Friday
Quick Review

• What is the unit of access for RAM*?
  What is the unit of access for disk?
  Why the difference?

• What is the Buffer Pool?

• Describe how a table is stored on disk

*RAM = Random Access memory = main memory
Memory hierarchies:

Disk → Main memory → Cache* → CPU

Buffer pool

1 page, e.g. 8KB

1 cache line, e.g. 64 bytes

*aaka CPU cache; several! L3, L2, L1 cache
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`.

- **Index**: 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

• Could have many indexes for one table

*Key = means here search key*
Example 1: Index on ID

Index **Actor_ID** on **Actor.ID**

### Data File Actor

<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>

### Actor

<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>
</tbody>
</table>

...
Example 2: Index on fName

Index **Actor_fName** on **Actor.fName**

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

Data File **Actor**

<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>
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
Discussion on Indices

- What they do: speed up disk access
  What they don’t: speed up RAM algo.

- They benefit only SELECT queries that have some predicate $A = \ldots$ or $A \leq \ldots$

- They hurt all INSERT/UPDATE/DELETE queries (why?)
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 not included
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) =
- V(Supplier, sname) =
- V(Supplier, scity) =
- V(Supplier, sstate) =
- M =
Cost Parameters

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

Block size = 8KB

- \(B(\text{Supplier}) = 1,000,000\) blocks \(= 8\text{GB}\)
- \(T(\text{Supplier}) = 50,000,000\) records \(\sim 50 / \text{block}\)
- \(V(\text{Supplier, sid}) = 50,000,000\) why?
  - \(V(\text{Supplier, sname}) = \)
  - \(V(\text{Supplier, scity}) = \)
  - \(V(\text{Supplier, sstate}) = \)
  - \(M = \)
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 =
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?
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) = 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 $a$:
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** \( a \): \( \text{cost} = B(R) / V(R,a) \)
- **Unclustered index on** \( a \): \( \text{cost} = T(R) / V(R,a) \)
Index Based Selection

- Example:
  - Table scan (assuming R is clustered)
  - B(R) = 2000
  - T(R) = 100,000
  - V(R, a) = 20

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

```sql
SELECT * FROM R WHERE R.a = v
```
Index Based Selection

- Example:
  - Table scan (assuming R is clustered)
    - $B(R) = 2000$ I/Os
  - Index based selection
    - If index is clustered:
    - $T(R) = 100,000$ I/Os
    - If index is unclustered:

\[
\begin{align*}
B(R) &= 2000 \\
T(R) &= 100,000 \\
V(R, a) &= 20 \\
\text{cost of } \sigma_{a=v}(R) &= ?
\end{align*}
\]
Index Based Selection

- Example:
  - Table scan (assuming R is clustered)
    - \( B(R) = 2000 \)
    - \( T(R) = 100,000 \)
    - \( V(R, a) = 20 \)
  - Index based selection
    - If index is clustered: \( B(R)/V(R,a) = 100 \) I/Os
    - If index is unclustered:

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

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

CSE 544 - Winter 2021

SELECT * 
FROM R 
WHERE R.a = v
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

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!

**SELECT * FROM R WHERE R.a = v**

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

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

The 2% rule!
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<?

Cost

Sequential scan

0

Percentage tuples retrieved

0 100

CSE 544 - Winter 2021
SELECT * 
FROM R 
WHERE R.K>? and R.K<?
SELECT * 
FROM R 
WHERE R.K>? and R.K<?
External Memory Joins

- Nested loop join
- Index join, a.k.a. index nested loop join
- Partitioned hash-join, a.k.a. grace join
- Merge-join
Nested Loop Joins

- R ⋈ S

- Naïve nested loop joint: T(R) * B(S) I/Os? WHY? Of course, can switch order: B(R) * T(S)

- We can be much cleverer by using the available main memory: M

- Assume |R| >> |S|. (Outer relation is bigger than inner relation)
Block Nested Loop Join

- Group of \((M-2)\) pages of \(S\) is called a “block”

\[
\text{for each (M-2) pages } ps \text{ of } S \text{ do}
\]
\[
\text{for each page } pr \text{ of } R \text{ do}
\]
\[
\text{for each tuple } s \text{ in } ps
\]
\[
\text{for each tuple } r \text{ in } pr \text{ do}
\]
\[
\text{if } r \text{ and } s \text{ join then output}(r,s)
\]
Block Nested Loop Join

R & S

Hash table for block of S (M-2 pages)

Input buffer for R

Output buffer

Join Result
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: \( B(S) + \frac{B(S)B(R)}{(M-2)} \)
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: \( B(S) + B(S)B(R)/(M-2) \)

Iterate over the smaller relation first!
Index Nested Loop Join

\[ R \bowtie S \]

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

- Cost:
  - If index on S is clustered: \( B(R) + T(R)B(S) / V(S,a) \)
  - If index on S is unclustered: \( B(R) + T(R)T(S) / V(S,a) \)
Two Pass Algorithms

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

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

![Diagram showing two pass algorithm](image)
Two Pass Algorithms

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

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

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

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

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

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

\[ B(R) \]

\[ \text{Disk} \]

\[ \text{Relation } R \]

\[ \text{OUTPUT} \]

\[ \text{INPUT} \]

\[ \text{hash function } h \]

\[ h : M \rightarrow M-1 \]

\[ \text{M main memory buffers} \]

\[ \text{Disk} \]

\[ 1 \]

\[ 2 \]

\[ \ldots \]

\[ \text{Partitions} \]
Two Pass Algorithms

• Idea: partition a relation \( R \) into buckets, on disk
Two Pass Algorithms

• Idea: partition a relation R into buckets, on disk

![Diagram showing the process of partitioning a relation into buckets using hash function h.](image)
Two Pass Algorithms

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

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

When does each bucket fit in memory?
Two Pass Algorithms

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

When $B(R)/M \leq M$, i.e. $B(R) \leq M^2$
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
Partition $R$ using hash fn $h$

$R \bowtie S$

Original Relation  

B main memory buffers  

OUTPUT

INPUT

hash function $h$

Partitions

Disk

Disk
Partition $S$ using hash fn $h$

$$R \bowtie S$$
Partitioned Hash Join

\[ R \bowtie S \]

- Read in partition of S, hash it using \( h_2 \) (\( \neq h \))
- Scan same partition of R, search for matches
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 fin 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: \( \frac{t}{k} \cdot B(S) \leq M \)
• Need room for k-t additional pages: \( k-t \leq M \)
• Thus: \( \frac{t}{k} \cdot B(S) + k-t \leq M \)
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: $(3-2t/k)(B(R) + B(S)) = (3-2M/B(S))(B(R) + B(S))$
ExternalSorting

• 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

- Size $M$ pages

- Runs of length $M$
  
  $\#\text{Runs} = \frac{B(R)}{M}$
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: $B(R) + B(R) \times B(S)/M$

• Index Nested Loop Join: $B(R) + T(R)B(S)/V(S,a)$

• Hash Join: $3(B(R) + B(S))$

  Hybrid Hash Join: $(3-2M/B(S))(B(R) + B(S))$

• Sort-Merge Join: $3B(R)+3B(S)$