CSE 544
Principles of Database Management Systems

Lectures 7 and 8
DBMS Architecture and Query Execution
Announcements

• Project proposals: please sign up for a 15’ meeting on Friday
  – You will present your proposal (5’)
  – We discuss it (5’)
  – Additional questions/comments (5’)

• Homework 2 is due on Friday

• Homework 3 is posted
Outline

• Architecture of a DBMS

• Steps involved in processing a query

• Operator implementations
Architecture of DBMS

• Reading:
  Architecture of a DBMS, chap. 1 and 2
Architecture of DBMS

[Diagram showing the architecture of a DBMS with various components such as Local Client Protocols, Remote Client Protocols, 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

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

Discuss pro/cons for each model

• Process per DBMS worker

• Thread per DBMS worker

• Process pool
Admission Control

What is it?
Outline

• Architecture of a DBMS

• Steps involved in processing a query

• Operator implementations
Query Optimization
Lifecycle of a Query

1. SQL query
2. Parse & Rewrite Query
3. Select Logical Plan
4. Select Physical Plan
5. Query Execution

Query optimization

Logical plan

Physical plan

Disk
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

```sql
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 )

Rewritten query:
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)

Correlation!
Decorrelation

\[
\text{SELECT Q.sno} \\
\text{FROM Supplier Q} \\
\text{WHERE Q.sstate = 'WA'} \\
\text{and not exists} \\
\text{(SELECT *} \\
\text{FROM Supply P} \\
\text{WHERE P.sno = Q.sno} \\
\text{and P.price > 100)} \\
\]

\[
\text{SELECT Q.sno} \\
\text{FROM Supplier Q} \\
\text{WHERE Q.sstate = 'WA'} \\
\text{and Q.sno not in} \\
\text{(SELECT P.sno} \\
\text{FROM Supply P} \\
\text{WHERE P.price > 100)} \\
\]
Decorrelation

(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)

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

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

Finally…

\[\pi_{\text{sno}} \sigma_{\text{sstate}='WA'} \sigma_{\text{Price} > 100}\]

\[\pi_{\text{sno}} \]

\[\text{Supplier} \]

\[\text{Supply} \]

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
Lifecycle of a Query (2)

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

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

- Union $\cup$, intersection $\cap$, difference $-$
- Selection $\sigma$
- Projection $\pi$
- Join $\bowtie$
- Duplicate elimination $\delta$
- Grouping and aggregation $\gamma$
- Sorting $\tau$
- Rename $\rho$

Bag semantics!
Logical Query Plan

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

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

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

Suppliers

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
Typical Plan For Block

\[ \pi \text{ fields} \]

\[ \gamma \text{ fields, sum/count/min/max(fields)} \]

\[ \sigma_{\text{having-condition}} \]

\[ \sigma_{\text{where-condition}} \]

\[ \text{join condition} \]

\[ \ldots \ldots \ldots \]

SELECT-PROJECT-JOIN Query
Physical Query Plan

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

$\pi_{\text{sname}}$

$\text{(On the fly)}$

$\text{(Nested loop)}$

Suppliers (File scan)

Supplies (Index lookup)

Physical plan = Logical plan + choice of algorithms + choice of 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
Implementing Query Operators with the Iterator Interface

Each operator implements three methods:

- open()
- next()
- close()
interface Operator {

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 (...);

}
```
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

```java
interface Operator {

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

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

    void close ();

}
```
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

```java
public interface Operator {

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

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

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

}
```
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

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

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

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

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

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

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

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

Example “on the fly” selection operator

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

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

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

class Select implements Operator {
    void open (Predicate p, Operator child) {
        this.p = p;
        this.child = child;  // Assuming class Operator has a child attribute
        child.open();
    }

    Tuple next () {
        // Implementation
    }
}
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

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

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

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

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

    Tuple next () {
        boolean found = false;
        Tuple r = null;
        while (!found) {
            r = child.next();
            if (r == null) break;
            found = p(r);
        }
    }
}
```
Implementing Query Operators with the Iterator Interface

Example “on the fly” selection operator

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

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

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

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

    Tuple next () {
        boolean found = false;
        Tuple r = null;
        while (!found) {
            r = child.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 ();
}

Example “on the fly” selection operator

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

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

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

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

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

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

Suppliers (File scan)

Supplies (File scan)

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

\begin{itemize}
  \item \textbf{(On the fly)}
  \item \textbf{(On the fly)} \( \sigma_{\textit{scity}=\text{Seattle} \text{ and } \textit{sstate}=\text{WA} \text{ and } \textit{pno}=2} \)
  \item \textbf{(Nested loop)}
\end{itemize}

\begin{itemize}
  \item \textbf{Pipelining}
  \item \textbf{on the fly}
  \item \textbf{open()}
  \item \textbf{\( \pi_{\textit{sname}} \)}
  \item \textbf{\( \sigma_{\textit{scity}=\text{Seattle} \text{ and } \textit{sstate}=\text{WA} \text{ and } \textit{pno}=2} \)}
  \item \textbf{Sno = Sno}
  \item \textbf{Suppliers}
  \item \textbf{Supplies}
  \item \textbf{(File scan)}
\end{itemize}

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

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

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

Supplies

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

\( \text{On the fly} \)

\( \text{On the fly} \)

\( \text{Nested loop} \)

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

Supplies

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

\( \text{Supplies} = \{ \text{sid, pno, quantity} \} \)

Pipelining

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

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

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

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

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

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

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

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

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

Pipelining

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

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

(Nested loop) $sno = sno$

Suppliers (File scan)

Supplies (File scan)

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

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

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

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

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

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

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

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

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

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

Suppliers (File scan)  Supplies (File scan)

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

Suppliers

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

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

Supplies

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

(On the fly)

(On the fly)

(Nested loop)

Suppliers (File scan)

Supplies (File scan)

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

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

### Supplies

\[ \text{Supplies} = \pi_{\text{sno}} \text{next()} \]

#### Pipelining

- (On the fly)
  - \( \pi_{\text{sname}} \text{next()} \)
  - (On the fly)
  - \( \sigma_{\text{scity} = 'Seattle' \text{ and } \text{sstate} = 'WA' \text{ and } \text{pno} = 2} \)
  - (Nested loop)
  - \( \text{sno} = \text{sno} \)
  - Suppliers (File scan)
  - Supplies (File scan)

---

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

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

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

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

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

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

Suppliers (File scan)

Supplies (File scan)

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

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

\( \pi_{sname} \)

Next()

Suppliers

(File scan)

 Suppliers

(File scan)

Next()

Next()

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

Pipelining

\begin{align*}
\text{(On the fly)} & \quad \pi_{\text{sname}} \\
\text{(On the fly)} & \quad \sigma_{\text{scity} = \text{Seattle} \text{ and } \text{sstate} = \text{WA} \text{ and } \text{pno} = 2} \\
\text{(Nested loop)} & \quad \text{sno} = \text{sno}
\end{align*}

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

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

\( \text{Suppliers} = \text{Suppliers} \cap \tau_{\text{scity}=\text{Sealett} \land \text{sstate}=\text{WA} \land pno=2} \\cap \pi_{\text{sname}} \)
Supplier\((sid, sname, scity, sstate)\)
Supply\((sid, pno, quantity)\)

Pipelining

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

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

(Hash Join)  \[sno = sno\]

Suppliers  (File scan)

Supplies  (File scan)

Discuss hash-join in class

Tuples from here are pipelined

CSE 544 - Winter 2018
Suppliers

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

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

Supplies

Tuples from here are pipelined

Tuples from here are “blocked”

Suppliers

(File scan)

Supplies

(File scan)

Discuss hash-join in class

(On the fly)

(On the fly)

(Hash Join)
Blocked Execution

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

(Merge Join)

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

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

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

Discuss merge-join in class

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

\begin{itemize}
\item \textbf{(On the fly)} $\sigma_{\text{scity} = 'Seattle' \text{ and } \text{sstate} = 'WA' \text{ and } \text{pno} = 2}$
\item \textbf{(Merge Join)} $\pi_{\text{sname}}$
\item \textbf{(On the fly)} $\text{sno} = \text{sno}$
\end{itemize}

Suppliers
(File scan)

Supplies
(File scan)

Discuss merge-join in class
Pipelined Execution

• Applies parent operator to tuples directly as they are produced by child operators

• Benefits
  – No operator synchronization issues
  – Saves cost of writing intermediate data to disk
  – Saves cost of reading intermediate data from disk
  – Good resource utilizations on single processor

• This approach is used whenever possible
Pipelined Execution

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

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

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

Suppliers (File scan)

Supplies (Index lookup)
Intermediate Tuple Materialization

• Writes the results of an operator to an intermediate table on disk

• Necessary for some operator implementations
• When operator needs to examine the same tuples multiple times
Intermediate Tuple Materialization

\( \pi \text{ sname} \)

\( \sigma \text{ sscity='Seattle' } \land \text{sstate='WA'} \)

\( \sigma \text{ pno=2} \)

\( \text{Suppliers} \)

\( \text{Supplies} \)

(On the fly)

(Sort-merge join)

(Scan: write to T1)

(Scan: write to T2)

(File scan)
Lifecycle of a Query

1. Parse & Rewrite Query
2. Select Logical Plan
3. Select Physical Plan
4. Query Execution

SQL query -> Parse & Rewrite Query -> Select Logical Plan -> Select Physical Plan -> Query Execution

Query optimization

Disk

Logical plan
Physical plan
Outline

- Architecture of a DBMS
- Steps involved in processing a query
  - Operator implementations
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)

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

DDL and Utility Processing
- Buffer Manager

Catalog Manager
- Memory Manager
- Administration, Monitoring & Utilities
- Replication and Loading Services
- Batch Utilities
- Shared Components and Utilities (Section 7)
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
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
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 the blocks
• Represent attributes inside the records
Managing Free Blocks

- Linked list of free blocks
- Or 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
Page Formats

Issues to consider

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

Why do we need RID’s in a relational DBMS?
Page Formats

Fixed-length records: packed representation

One page

Rec 1  Rec 2  Rec N

Free space

Problems ?
Page Formats

Variable-length records
Record Formats: Fixed Length

Product(pid, name, descr, maker)

\[
\text{Address} = B + L1 + L2
\]

- 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!
Need the header because:

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

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.
- **Sorted Files** Best if records must be retrieved in some order, or only a `range’ of records is needed.
- **Indexes** Data structures to organize records via trees or hashing.
  - Like sorted files, they speed up searches for a subset of records, based on values in certain (“search key”) fields
  - Updates are much faster than in sorted files.
Multiple Processes

- Local Client Protocols
- Remote Client Protocols
- Client Communications Manager
- Query Parsing and Authorization
- Query Rewrite
- Query Optimizer
- DDL and Utility Processing
- 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)

Plan Executor
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
One-pass Algorithms

Selection \( \sigma(R) \), projection \( \Pi(R) \)
- Both are **tuple-at-a-time** algorithms
- Cost: \( B(R) \), the cost of scanning the relation

![Diagram](image-url)
Main Memory Join Algorithms

Three standard main memory algorithms:

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

Review in class
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

\[
\text{for each tuple } r \text{ in } R \text{ do for each tuple } s \text{ in } S \text{ do if } r \text{ and } s \text{ join then output } (r,s)
\]

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

\begin{algorithm}
\begin{verbatim}
  for each page of tuples r in R do
    for each page of tuples s in S do
      for all pairs of tuples
        if r and s join then output (r,s)
\end{verbatim}
\end{algorithm}

- 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
\[ ps \bowtie pr \]
Nested Loop Joins

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

Input buffer for R  Output buffer
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: \( B(S) + B(S)B(R)/(M-2) \)

Notice: it is better to 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: $B(R) + B(S)$
- One pass algorithm when $B(S) + B(R) \leq M$
- Typically, this is NOT a one pass algorithm
Example

Grouping:
\[
\text{Product(name, department, quantity)} \quad \gamma_{\text{department, sum(quantity)}} \quad (\text{Product}) \rightarrow \text{Answer(department, sum)}
\]

In class: describe a one-pass algorithms. Cost=?
Outline

• **Steps involved in processing a query**
  – Logical query plan
  – Physical query plan
  – Query execution overview

• **Operator implementations**
  – One pass algorithms
  – Two-pass algorithms
  – Index-based algorithms
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
- Each bucket has size approx. B(R)/M

• Does each bucket fit in main memory?
  – Yes if B(R)/M ≤ M, i.e. B(R) ≤ M^2
Hash Based Algorithms for $\gamma$

- Recall: $\gamma(R) = \text{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

- Partition both relations using hash fn $h$
- R tuples in partition i will only match S tuples in partition i.

Diagram:
- Original Relation
- DISK
- B main memory buffers
- OUTPUT
  - 1
  - 2
  - M-1
- DISK

CSE 544 - Winter 2018
Partitioned Hash Join

- Read in partition of R, hash it using $h2 \neq h$
  - Build phase
- Scan matching partition of S, search for matches
  - Probe phase
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 $S_{t+1}, \ldots, S_k$ go 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: \[ \frac{t}{k} \cdot B(S) \leq M \]
  – Need room for k-t additional pages: \[ k-t \leq M \]
  – Thus: \[ \frac{t}{k} \cdot B(S) + k-t \leq M \]

• Assuming \( \frac{t}{k} \cdot B(S) \gg k-t \): \[ \frac{t}{k} = \frac{M}{B(S)} \]
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 < M^2
External Merge-Sort: Step 1

• Phase one: load $M$ pages in memory, sort
External Merge-Sort: Step 2

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

If $B \leq M^2$ then 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: $B(R) + B(S) \leq M^2$
Index

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

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

• Could have many indexes for one table

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

**Index Student_ID on Student.ID**

### Data File Student

<table>
<thead>
<tr>
<th>ID</th>
<th>fName</th>
<th>lName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Tom</td>
<td>Hanks</td>
</tr>
<tr>
<td>20</td>
<td>Amy</td>
<td>Hanks</td>
</tr>
<tr>
<td>50</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>200</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>220</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>240</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>420</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>800</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

### Index

- 10
- 20
- 50
- 200
- 220
- 240
- 420
- 800
- 950
- ...

CSE 544 - Winter 2018
Example 2: Index on fName

Index `Student_fName` on `Student.fName`
Index Organization

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

Several ways to do this:

• Hash table
• B+ trees – most popular
  – They are search trees, but they are not binary instead have higher fanout
  – Will discuss them briefly next
• Specialized indexes: bit maps, R-trees, inverted index
Review: 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
Clustered vs Unclustered

Every table can have **only one** clustered and **many** unclustered indexes.
Index Based Selection

• Selection on equality: $\sigma_{a=v}(R)$

• $V(R, a) = \#$ of distinct values of attribute $a$

• Clustered index on $a$: cost $B(R)/V(R,a)$

• Unclustered index on $a$: cost $T(R)/V(R,a)$

• Note: we ignored the I/O cost for the index pages (why?)
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: \( \frac{B(R)}{V(R,a)} = 100 \) I/Os
    - If index is unclustered: \( \frac{T(R)}{V(R,a)} = 5,000 \) I/Os

- Lesson
  - Don’t build unclustered indexes when \( V(R,a) \) is small!

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

- Example:
  - B(R) = 2000
  - T(R) = 100,000
  - V(R, a) = 20
  - cost of \( s_{a=v}(R) \) = ?

- 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!

The 2% rule!

Note: the “2” in 2% decreases yearly (why?)
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: $B(R) + T(R)B(S)/V(S,a)$
  - If index on S is unclustered: $B(R) + T(R)T(S)/V(S,a)$
Summary of External Join Algorithms

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

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

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

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