Lecture 2 Query Execution and Optimization
Announcements

• Paper reviews are due before the beginning of the class

• Project teams due on Friday

• HW1 is due next Monday
Outline for Today

• Discuss *Goes Around* paper

• Discuss query optimization
  – Major paper to read for next time
  – We continue query optimization next time
Discussion of the paper

Data Model

• Enables a user to define the data using high-level constructs without worrying about many low-level details of how data will be stored on disk

• Examples:
  – Relational data model
  – Semistructured data model
  – Graph data model
  – Key-value pairs data model
Paper Discussion

- Early data models: IMS, CODASYL
- Relational data model
- Semistructured data model
Early Proposal 1: IMS*

- What is it?

* IBM Information Management System
Early Proposal 1: IMS*

• What is it?

• Hierarchical data model

• Record
  – Record type, record instance
  – Each instance has a **key**
  – Arranged in a **tree**

* IBM Information Management System
IMS Example

Figure 2 from “What goes around comes around”

What does this mean?
IMS Example

Figure 2 from “What goes around comes around”

File on disk:

| Supp | Part | Part | … | Supp | Part | Part | … | … |

What does this mean?
IMS Example

Figure 2 from “What goes around comes around”

What does this mean?

File on disk:
IMS Limitations
IMS Limitations

- Tree-structured: redundant; existence depends on parent
- Record-at-a-time interface
- Very limited physical independence
- Some logical independence but limited
Data Manipulation Language: DL/1

How does a programmer retrieve data in IMS?
Data Manipulation Language: DL/1

How does a programmer retrieve data in IMS?

- Each record has a hierarchical sequence key (HSK)
- `get_next`; `get_next_within_parent`
- Programmers need to worry about optimization

DL/1 is a record-at-a-time language
Data storage

How is data physically stored in IMS?
Data storage

How is data physically stored in IMS?

• Root records
  – Stored sequentially (sorted on key), or
  – Indexed in a B-tree using the key of the record, or
  – Hashed using the key of the record

• Dependent records: various forms of pointers

• Selected organizations restrict DL/1 commands
Data Independence

What is it?
Data Independence

What is it?

- **Physical data independence**: Applications are insulated from changes in physical storage details

- **Logical data independence**: Applications are insulated from changes to logical structure of the data
Early Proposal 2: CODASYL

What is it?
Early Proposal 2: CODASYL

What is it?
- **Networked data model**
- Organized into **network**
- Multiple parents; arcs = “sets”
- **Record-at-a-time** data manipulation language
CODASYL Example

- Figure 5 from “What goes around comes around”
Paper Discussion

- Early data models: IMS, CODASYL
- Relational data model
- Semistructured data model
Relational Model Overview
Ted Codd 1970

• What was the motivation? What is the model?
Relational Model Overview
Ted Codd 1970

• What was the motivation? What is the model?

• Logical and physical data independence

• Store data in a **simple data structure** (table)
• Access data through **set-at-a-time** language
• No need for physical storage proposal
Great Debate

• Pro relational
  – What were the arguments?

• Against relational
  – What were the arguments?

• How was it settled?
Great Debate

• **Pro relational**
  – CODASYL is too complex
  – No data independence
  – Record-at-a-time hard to optimize
  – Trees/networks not flexible enough

• **Against relational**
  – COBOL programmers cannot understand relational languages
  – Impossible to implement efficiently

• **Ultimately settled by the market place**
Recap

• Physical data independence:
  – SQL: *what* data we want
  – Optimizer: figures out *how* to get it

• Logical data independence
  – Realized in SQL through *views*
Paper Discussion

• Early data models: IMS, CODASYL

• Relational data model

• Semistructured data model
Other Data Models

• Entity-Relationship: 1970’s
  – Successful in logical database design
• Extended Relational: 1980’s
• Semantic: late 1970’s and 1980’s
• Object-oriented: late 1980’s and early 1990’s
  – Address impedance mismatch: relational dbs ↔ OO languages
  – Interesting but ultimately failed (several reasons, see references)
• Object-relational: late 1980’s and early 1990’s
  – User-defined types, ops, functions, and access methods
• Semi-structured: late 1990’s to the present
Semistructured Data Model

• Main idea: *schema-last*

• Examples:
  – XML
  – Json
  – Protobuf

• All use a *tree data model*
XML Syntax

<article mdate="2011-01-11" key="journals/acta/GoodmanS83">
  <author>Nathan Goodman</author>
  <author>Oded Shmueli</author>
  <title>NP-complete Problems Simplified on Tree Schemas.</title>
  <pages>171-178</pages>
  <year>1983</year>
  <journal>Acta Inf.</journal>
  <url>db/journals/acta/acta20.html#GoodmanS83</url>
</article>

Semistructured, self-describing schema
JSon

Example from: http://www.jsonexample.com/

myObject = {
    "first": "John",
    "last": "Doe",
    "salary": 70000,
    "registered": true,
    "interests": [ "Reading", "Biking", "Hacking" ]
}

Semistructured, self-describing schema
Discussion of Semistructured

• Stonebraker (circa 1998): niche market

• Today (circa 2020): Json is common

• What changed? Data Science!
Outline for Today

• Discuss *Goes Around* paper

• Discuss query optimization
  – Major paper to read for next time
  – We continue query optimization next time
DBMS Architecture

Process Manager
- Admission Control
- Connection Mgr
- Access Methods
- Lock Manager

Query Processor
- Parser
- Query Rewrite
- Optimizer
- Executor

Storage Manager
- Buffer Manager
- Log Manager

Shared Utilities
- Memory Mgr
- Disk Space Mgr
- Replication Services
- Admin Utilities

Lifecycle of a Query

SQL query

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

Disk

Query optimization

Logical plan

Physical plan
Lifecycle of a Query

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

Logical plan
Physical plan
Query optimization
Relational Algebra

• A set-at-a-time algebra

• Inputs: relations; Output: relation

• E.g. join: $R \bowtie S$

RA details on next slides
Five Basic Relational Operators

- **Selection:** \( \sigma_{\text{condition}}(S) \)
- **Projection:** \( \pi_{\text{list-of-attributes}}(S) \)
- **Union:** \( (U) \)
- **Set difference:** \( (–) \),
- **Cross-product/cartesian product:** \( (\times) \),
  \begin{align*}
  \text{Join: } R \bowtie_{\theta} S &= \sigma_{\theta}(R \times S)
  \end{align*}
Extended Operators of Relational Algebra

- Duplicate elimination ($\delta$)
- Group-by/aggregate ($\gamma$)
- Sort operator ($\tau$)
Logical Query Plan

• Is an expression in RA
• It specifies in which order to execute the operators
Logical Query Plan

- Is an expression in RA
- It specifies in which order to execute the operators

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

- Is an expression in RA
- It specifies in which order to execute the operators

Do this first:

Supplier(x)

\[ x.sno = y.sno \]

Supply(y)

\[ y.pno = z.pno \]

Part(z)

\[ z.psize > 10 \]
Logical Query Plan

- Is an expression in RA
- It specifies in which order to execute the operators

Do this first:

\[
\begin{align*}
\text{Supplier}(x) & \quad \text{Supply}(y) \\
\quad \text{Part}(z)
\end{align*}
\]

\[
\begin{align*}
\text{Join} & \quad x . \text{sno} = y . \text{sno} \\
\text{Join} & \quad y . \text{pno} = z . \text{pno} \\
\text{Select} & \quad \sigma z . \text{psize} > 10
\end{align*}
\]
Logical Query Plan

- Is an expression in RA
- It specifies in which order to execute the operators

Do this first:

\[ \begin{align*}
\text{Supplier} & (\text{sno}, \text{sname}, \text{scity}, \text{sstate}) \\
\text{Supply} & (\text{sno}, \text{pno}, \text{price}) \\
\text{Part} & (\text{pno}, \text{pname}, \text{psize}, \text{pcolor})
\end{align*} \]
Logical Query Plan

- Is an expression in RA
- It specifies in which order to execute the operators

Do this first:

Supplier \( x \)  
Supply \( y \)  
Part \( z \)

\( \Pi_{x.scity} \)

\( \sigma_{z.psize > 10} \)

\( \delta \)

\( y.pno = z.pno \)

\( x.sno = y.sno \)
Converting SQL to RA

1. Convert FROM-WHERE to \( \bowtie \) and \( \sigma \)

2. Convert GROUP-BY to \( \gamma \)

3. Convert HAVING to \( \sigma \), SELECT to \( \Pi \)

4. Decorrelate queries (this is done first)
1. FROM-WHERE to $\bowtie - \sigma$

SELECT ...
FROM R1, R2, ...
WHERE Condition
GROUP BY ...
HAVING ...

$\bowtie$

$\sigma_{\text{Condition}}$

R_n

$\bowtie$

R_4

$\bowtie$

R_3

$\bowtie$

R_2

$\bowtie$

R_1
2. GROUP-BY to $\gamma$

```
SELECT ..., agg_1, agg_2, ...
FROM R1, R2, ...
WHERE Condition
GROUP BY A_1, A_2,
HAVING ... agg'_1, agg'_2, ...
```
3. HAVING to \(\sigma\), SELECT to \(\Pi\)

```
SELECT B1, B2, ..., agg_1, ...
FROM R1, R2, ...
WHERE Condition
GROUP BY A_1, A_2,
HAVING Condition'
```
Example

Find max price of red products for each city that sold > 100 parts
Example

Find max price of red products for each city that sold > 100 parts

```
SELECT x.city, max(y.price)
FROM Supplier x, Supply y, Part z,
WHERE x.sno=y.sno and y.pno=z.pno
    and z.pcolor='red'
GROUP BY x.city
HAVING count(*) > 100
```
Find max price of red products for each city that sold > 100 parts

\[
\text{SELECT } x\text{.city, max(y.price)} \\
\text{FROM Supplier x, Supply y, Part z,} \\
\text{WHERE } x\text{.sno}=y\text{.sno and y.pno}=z\text{.pno} \\
\quad \text{and } z\text{.pcolor}=\text{‘red’} \\
\text{GROUP BY } x\text{.city} \\
\text{HAVING count(*) > 100}
\]
Example

Find max price of red products for each city that sold > 100 parts

```
SELECT x.city, max(y.price)
FROM Supplier x, Supply y, Part z,
WHERE x.sno=y.sno and y.pno=z.pno
    and z.pcolor='red'
GROUP BY x.city
HAVING count(*) > 100
```
Example

Find max price of red products for each city that sold > 100 parts

```
SELECT x.city, max(y.price)
FROM Supplier x, Supply y, Part z,
WHERE x.sno=y.sno and y.pno=z.pno
    and z.pcolor='red'
GROUP BY x.city
HAVING count(*) > 100
```
Find max price of red products for each city that sold > 100 parts

```
SELECT x.city, max(y.price) 
FROM Supplier x, Supply y, Part z,  
WHERE x.sno=y.sno and y.pno=z.pno  
    and z.pcolor='red'    
GROUP BY x.city  
HAVING count(*) > 100
```
4. Decorrelation

- A *correlated SQL subquery* is one that depends on a variable of outer query

- This cannot be converted to RA, because does not have variables

- Solution: decorrelation
4. 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)
4. Decorrelation

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

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

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
(SELECT Q.sno FROM Supplier Q WHERE Q.sstate = 'WA')
EXCEPT
(SELECT P.sno FROM Supply P WHERE P.price > 100)
SQL to RA: Summary

- SQL query to Relational Algebra Plan, which is further optimized

- This is a *logical* plan; specifies the order

- Next: *physical* plan; specifies the implementation
Physical Operators
Lifecycle of a Query

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

SQL query

Query optimization

Logical plan

Physical plan

Disk
Physical Operators

• Each logical operator in RA can be implemented in multiple ways
• An implementation: a physical operator
Physical Operators

• Each logical operator in RA can be implemented in multiple ways
• An implementation: a *physical operator*
  – Physical operators for $\sigma$: [in class]
  – Physical operators for $\cup$: [in class]
Physical Operators

• Each logical operator in RA can be implemented in multiple ways

• An implementation: a physical operator
  – Physical operators for $\sigma$: [in class]
  – Physical operators for $\cup$: [in class]
  – Physical operators for $\bowtie$: next slides
  – Physical operators for $\gamma$, $\delta$: not discussed
  – Physical operators for -: : not discussed
Join

Logical operator:

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

Three algorithms:
1. 
2. 
3. 
Join

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:

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

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

```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?
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)

If \(|R|=|S|=n\), what is the runtime?
O\( (n^2) \)
BRIEF Review of Hash Tables

• Array: map indices to memory locations
  – A[0], A[1], A[2], … sequential in memory

• How to map texts to memory locations?
  – A[“alice”], A[“bob”], A[“carl”]…???

• Hash function: maps strings to indices
Separate chaining:

**BRIEF Review of Hash Tables**

A (naïve) hash function:

\[ h(\text{“abc”}) = (a'b'c') \mod 10 \]
Separate chaining:

BRIEF Review of Hash Tables

A (naïve) hash function:

\[ h(\text{"abc"}) = (\text{a'+'b'+c'}) \mod 10 \]

E.g. \[ h(\text{"ker"}) = (\text{k'+e'+r'}) \mod 10 \]
\[ = (107+101+114) \mod 10 \]
\[ = 2 \]
BRIEF Review of Hash Tables

A (naïve) hash function:

\[ h(\text{"abc"}) = (\text{a}+\text{b}+\text{c}) \mod 10 \]

E.g. \[ h(\text{"ker"}) = (\text{k}+\text{e}+\text{r}) \mod 10 \]
\[ = (107+101+114) \mod 10 \]
\[ = 2 \]
**BRIEF Review of Hash Tables**

A (naïve) hash function:

\[ h("abc") = (a' + b' + c') \mod 10 \]

E.g.

\[ h("ker") = (k' + e' + r') \mod 10 \]
\[ = (107 + 101 + 114) \mod 10 \]
\[ = 2 \]

Separate chaining:

- Insertion: `insert("alice")`
- Find: `find("yu")`

```
0 1 2 3 4 5 6 7 8 9
```

- `ker` at index 2
- `foo` at index 4
- `jon` at index 7
- `xou` at index 8
- `yu` at index 9
- `abc` at the same index as `yu`
Hash Tables: Summary

Key/value pairs; e.g. \((\text{sid}, \text{Supply})\)

- \(\text{insert}(k, v)\)
- \(\text{find}(k) = \text{returns the list of values } v\)
- Time is \(O(1)\), but can become \(O(n)\)
- Collisions!
- Don’t write your own hash function
2. Hash Join

Logical operator:

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

Build phase

for \( x \) in Supplier do

insert(\( x.\text{sid}, x \))
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 = find(y.sid);
  output(x, y);

Build phase
Probe phase
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 = find(y.sid);
    output(x,y);
```

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

Logical operator:

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

$O(n)$
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:

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

```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);
```

Change join order
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

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

Logical operator:

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

For $y$ in Supply do

\text{insert}(y.\text{sid}, y)

For $x$ in Supplier do

For $y$ in find($x.\text{sid}$) do

output($x, y$);

If $|\text{R}| = |\text{S}| = n$, 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.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 \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?

O(n)

But can be O(n^2) why?

Change join order

Why would we change the order?

When |Supply| << |Supplier|

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
3. Merge Join

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

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

Logical operator:

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

Sort(Supplier); Sort(Supply);

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

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

  case:
  \(x.\text{sid} < y.\text{sid}: x = x.\text{next}()\)
  \(x.\text{sid} = y.\text{sid}: ???\)
  \(x.\text{sid} > y.\text{sid}: ???\)
3. Merge Join

Logical operator:

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

Sort(Supplier); Sort(Supply);

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

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

$O(n \log(n))$ (because sorting…)
Summary of Physical Operators

• $\sigma$: on-the-fly

• $\cup$: concatenate, then apply $\delta$

• $\Join$: nested-loop join, hash-join, merge-join

• $\gamma, \delta$: nested-loop, hash-based, sort-based

• $\cdot$: nested-loop, hash-based, sort-based
Query Optimizer
Lifecycle of a Query

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

Query optimization

Disk

Logical plan
Physical plan
Query Optimization

1. Search space

2. Cardinality and cost estimation

3. Plan enumeration algorithms (next time)
Search Space

• Search space = set of rewrite rules that the optimizer implements

• E.g. SQL Server has 400+ rules

We discuss a few basic rewrite rules next
Example Optimization

```
SELECT x.sid, y.pno, y.quantity
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and x.scity = 'Seattle'
```
Example Optimization

```
SELECT x.sid, y.pno, y.quantity
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and x.scity = 'Seattle'
```
Push Selections Down

\[ \prod_{x.\text{sid}, y.\text{pno}, y.\text{quantity}} \]

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

\[ \Join_{x.\text{sid} = y.\text{sid}} \]

\[ \text{Supplier x} \]
\[ \text{Supply y} \]
Push Selections Down

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

```
π_{x.sid, y.pno, y.quantity} (π_{x.sid, y.pno, y.quantity} (σ_{x.scity='Seattle'} (Join_{x.sid = y.sid} Supplier x Supply y)))
```

```
π_{x.sid, y.pno, y.quantity} (π_{x.sid, y.pno, y.quantity} (σ_{x.scity='Seattle'} (Join_{x.sid = y.sid} Supplier x Supply y)))
```
Push Selections Down

\[ \sigma_{x.scity='Seattle'}(\Pi_{x.sid,y.pno,y.quantity}(\bowtie_{x.sid=y.sid} \text{Supplier } x \bowtie \text{Supply } y)) \]

\[ \sigma_{x.scity='Seattle'}(\Pi_{x.sid,y.pno,y.quantity}(\bowtie_{x.sid=y.sid} \text{Supplier } x \bowtie \text{Supply } y)) \]

\( \sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S \) when \( C \) refers only to \( R \)
Push Selections Down

\[
\Pi_{x.\text{sid}, y.\text{pno}, y.\text{quantity}}
\]

\[
\sigma_{x.\text{scity}='Seattle'} \text{ and } y.\text{pno}=5
\]

\[
\Join_{x.\text{sid} = y.\text{sid}}
\]

Supplier x

Supply y
Push Selections Down

\[ \Pi_{x \cdot \text{sid}, y \cdot \text{pno}, y \cdot \text{quantity}} \]
\[ \sigma_{x \cdot \text{scity}=\text{\textquoteleft}Seattle\textquoteright} \text{ and } y \cdot \text{pno}=5 \]

\[ \bowtie_{x \cdot \text{sid} = y \cdot \text{sid}} \]
\[ \sigma_{x \cdot \text{scity}=\text{\textquoteleft}Seattle\textquoteright} \]
\[ \Pi_{x \cdot \text{sid}, y \cdot \text{pno}, y \cdot \text{quantity}} \]
\[ \sigma_{y \cdot \text{pno}=5} \]
Push Selections Down

\[ \Pi_{x\.sid, y\.pno, y\.quantity} \]

\[ \sigma_{x\.scity='Seattle'} \text{ and } y\.pno=5 \]

\[ \bowtie \]

\[ \sigma_{x\.scity='Seattle'} \text{ and } y\.pno=5 \]

\[ \Pi_{x\.sid, y\.pno, y\.quantity} \]

\[ \sigma_{y\.pno=5} \]

\[ \sigma_{C1 \text{ and } C2}(R \bowtie S) = \sigma_{C1}(\sigma_{C2}(R \bowtie S)) = \sigma_{C1}(R \bowtie \sigma_{C2}(S)) = \sigma_{C1}(R) \bowtie \sigma_{C2}(S) \]
Join Reorder

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)
Join Reorder

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)
Join Reorder

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

(R \Join S) \Join T = R \Join (S \Join T)
R \Join S = S \Join R
Join Reorder

When is one plan better than the other?

(R \Join S) \Join T = R \Join (S \Join T)

R \Join S = S \Join R
Join Reorder

When is one plan better than the other?

Depends on:

\[ |\text{Supplier} \bowtie \text{Supply}| \leq |\text{Supply} \bowtie \text{Part}| \]

\[ (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T) \]

\[ R \bowtie S = S \bowtie R \]
Join Reorder

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

\[
\begin{align*}
\text{Supplier } x & \bowtie \text{Supply } y \\
& \bowtie \text{Part } z \\
& \sigma_{x.scity = 'Seattle'} \\
& \sigma_{z.pprice > 99}
\end{align*}
\]
Supplier\((\text{sid}, \text{sname}, \text{scity}, \text{sstate})\)
Supply\((\text{sid}, \text{pno}, \text{quantity})\)
Part\((\text{pno}, \text{pname}, \text{pprice})\)
Lesson: need sizes of $\sigma_{x.scity='Seattle'}(Supplier)$, $\sigma_{z.pprice > 99}(Part)$
Search Space: Summary

• Large set of rewrite rules

• Generates many candidate plans

• Need to estimate their cost, choose best
Query Optimization

1. Search space

2. Cardinality and cost estimation

3. Plan enumeration algorithms (next time)
Cardinality Estimation

**Problem**: given statistics on base tables and a query, estimate size of the answer

Challenging, because:
- Need to do it very fast
- Need to use very little memory
Statistics on Base Data

- Number of tuples (cardinality) \( T(R) \)
- Number of physical pages \( B(R) \)
- Indexes, number of keys in the index \( V(R,a) \)

- Histograms: 1d or 2d (next lecture)

Computed periodically, often using sampling
Assumptions

• Uniformity

• Independence

• Containment of values

• Preservation of values
Size Estimation

Selection: size decreases by *selectivity factor* $\theta$

$$T(\sigma_{\text{pred}}(R)) = \theta_{\text{pred}} \times T(R)$$
Size Estimation

**Selection:** size decreases by *selectivity factor* $\theta$

\[
T(\sigma_{\text{pred}}(R)) = \theta_{\text{pred}} \times T(R)
\]

\[
T(R \bowtie_{A=B} S) = \theta_{A=B} \times T(R) \times T(S)
\]
Selectivity Factors

**Uniformity assumption**

Equality:

\[ T(\sigma_{\text{pred}}(R)) = \theta_{\text{pred}} \times T(R) \]
Selectivity Factors

**Uniformity assumption**

Equality:

- \( \theta_{A=c} = 1/V(R,A) \)

\[ T(\sigma_{\text{pred}}(R)) = \theta_{\text{pred}} \cdot T(R) \]
$T(\sigma_{\text{pred}}(R)) = \theta_{\text{pred}} \times T(R)$

**Selectivity Factors**

**Uniformity assumption**

Equality:
- $\theta_{A=c} = 1/V(R,A)$

Range:
- $\theta_{c_1<A<c_2} = (c_2 - c_1)/(\max(R,A) - \min(R,A))$
Selectivity Factors

Uniformity assumption

Equality:

• \( \theta_{A=c} = 1/V(R,A) \)

Range:

• \( \theta_{c1<A<c2} = (c2 - c1)/(\max(R,A) - \min(R,A)) \)

Conjunction

\[ T(\sigma_{\text{pred}}(R)) = \theta_{\text{pred}} \times T(R) \]
Selectivity Factors

**Uniformity assumption**

Equality:
- \( \theta_{A=c} = 1/V(R,A) \)

Range:
- \( \theta_{c1<A<c2} = (c2 - c1)/(\max(R,A) - \min(R,A)) \)

**Independence assumption**

Conjunction
- \( \theta_{\text{pred1 and pred2}} = \theta_{\text{pred1}} \times \theta_{\text{pred2}} = 1/V(R,A) \times 1/V(R,B) \)

\( T(\sigma_{\text{pred}(R)}) = \theta_{\text{pred}} \times T(R) \)
Selectivity Factors

\[ T(R \bowtie_{A=B} S) = \theta_{A=B} \ast T(R) * T(S) \]
Selectivity Factors

Join

- $\theta_{R.A=S.B} = \frac{1}{\text{MAX}(V(R,A), V(S,B))}$

Why? Will explain next...
Selectivity Factors

**Containment of values**: if $V(R,A) \leq V(S,B)$, then the set of $A$ values of $R$ is included in the set of $B$ values of $S$

- Note: this indeed holds when $A$ is a foreign key in $R$, and $B$ is a key in $S$
Selectivity Factors

Assume $V(R,A) \leq V(S,B)$

• Tuple $t$ in $R$ joins with $T(S)/V(S,B)$ tuples in $S$
Selectivity Factors

Assume \( V(R,A) \leq V(S,B) \)

- Tuple \( t \) in \( R \) joins with \( T(S)/V(S,B) \) tuples in \( S \)
- Hence \( T(R \bowtie_{A=B} S) = T(R) \cdot T(S) / V(S,B) \)
Selectivity Factors

Assume $V(R,A) \leq V(S,B)$
- Tuple $t$ in $R$ joins with $T(S)/V(S,B)$ tuples in $S$
- Hence $T(R \bowtie_{A=B} S) = T(R) \cdot T(S) / V(S,B)$

In general:
- $T(R \bowtie_{A=B} S) = T(R) \cdot T(S) / \max(V(R,A), V(S,B))$
- $\theta_{R.A=S.B} = 1/ (\max(V(R,A), V(S,B)))$
Final Assumption

*Preservation of values*:
For any other attribute C:
- \( V(R \bowtie_{A=B} S, C) = V(R, C) \) or
- \( V(R \bowtie_{A=B} S, C) = V(S, C) \)

• This is needed higher up in the plan
Computing the Cost of a Plan

• Estimate cardinalities bottom-up

• Estimate cost by using estimated cardinalities

• Examples next...
Logical Query Plan 1

\[ \pi_{\text{sname}} \sigma_{pno=2 \land \text{scity}='Seattle' \land sstate='WA'}(x, y) \]

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

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

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

T(Supply) = 10000
V(Supply, pno) = 2500
Logical Query Plan 1

\[
\begin{align*}
\text{Estimated (why?)} \\
\sigma_{\text{pno}=2 \land \text{scity}='Seattle' \land \text{sstate}='WA'} \\
\Pi_{\text{sname}}
\end{align*}
\]

\[T = 10000\]

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

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

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

Supplier(\text{sid, sname, scity, sstate})
Supply(\text{sid, pno, quantity})
Logical Query Plan 1

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

Because key / foreign-key

Estimated (why?)

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

T = 10000

Because key / foreign-key

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

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

Supply

Supplier

**π_{sname}**
Logical Query Plan 1

Estimated (why?)

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

Because key / foreign-key

\[ \text{id} = \text{id} \]

Also: \[ \theta = 1/\max(V(\text{Supply}, \text{id})*V(\text{Supplier}, \text{id}) = 1/V(\text{Supplier}, \text{id}) \]

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

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

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

\text{Supplier(sid, sname, scity, sstate)}
\text{Supply(sid, pno, quantity)}
Logical Query Plan 1

Estimated (why?)

\[ \pi_{\text{snname}} \]

\[ T < 1 \]

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

\[ T = 10000 \]

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

\[ \text{Supply} \]

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

\[ \text{Supplier} \]

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

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

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

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

T(Supply) = 10000
V(Supply, pno) = 2500
Logical Query Plan 2

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

```
π_{sname}(
  σ_{pno=2}(Supply) ∨
  σ_{scity='Seattle'∧sstate='WA'}(Supplier)
)
```

```
T(Supplier) = 1000
V(Supplier, scity) = 20
V(Supplier, state) = 10
T(Supply) = 10000
V(Supply, pno) = 2500

T = 4
T = 5
```

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

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

T(Supply) = 10000
V(Supply, pno) = 2500
Logical Query Plan 2

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

\[
T = 4
\]

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

\[
\text{Supply}
\]

\[
T = 4
\]

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

\[
\text{Supplier}
\]

\[
T = 5
\]

\[
\text{SELECT sname}
\]

\[
\text{FROM Supplier x, Supply y}
\]

\[
\text{WHERE x.sid = y.sid}
\]

\[
\text{and y.pno = 2}
\]

\[
\text{and x.scity = 'Seattle'}
\]

\[
\text{and x.sstate = 'WA'}
\]

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

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

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

Very wrong! Why?
Logical Query Plan 2

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

<table>
<thead>
<tr>
<th>Table</th>
<th>Estimate</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supplier</td>
<td>T(Supplier) = 1000</td>
<td>V(Supplier, scity) = 20</td>
</tr>
<tr>
<td>Supply</td>
<td>T(Supply) = 10000</td>
<td>V(Supply, pno) = 2500</td>
</tr>
</tbody>
</table>

The estimate for the query is different from the actual execution cost.

Very wrong! Why?
Summary

• Optimizer has three components:
  – Search space
  – Cardinality and cost estimation
  – Plan enumeration algorithms (next time)

• Paper *How good are they* does a deep dive into modern optimizers

• Will continue optimizers next week