CSEP 544: Lecture 04

Query Execution
Announcements

Homework 2: due on Friday

Homework 3:
• We use AWS
• You need to get an access code: https://aws.amazon.com/education/awseducate/members/
Where We Are

• We have seen:
  – Disk organization = set of blocks(pages)
  – The buffer pool
  – How records are organized in pages
  – Indexes, in particular B+ -trees

• Today: query execution, optimization
Steps of the Query Processor

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

SQL query

Query optimization

Logical plan
Physical plan
Disk
Steps in Query Evaluation

- **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, etc.
Steps in Query Evaluation

• **Step 3: Query optimization**
  – Find an efficient query plan for executing the query

• **Step 4: Query execution**
  – Each operator has several implementation algorithms
SQL Query

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
    x.price > 100 and z.city = 'Seattle'
Logical Plan

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Temporary tables T1, T2, ...

Final answer

T4(name,name)

T3(...)

T1(pid,name,price,pid,cid,store)
T2(....)

δ

Π

σ

price>100 and city='Seattle'

cid=cid

Customer

Product

Purchase

x.name, z.name
Logical v.s. Physical Plan

- Physical plan = Logical plan plus annotations
- Access path selection for each relation
  - Use a file scan or use an index
- Implementation choice for each operator
- Scheduling decisions for operators
Logical Query Plan

\[ \Pi_{\text{name, price}} \]

\[ \sigma_{\text{name='Gizmo' \land store='GizmoMart'}} \]

\[ \sigma_{\text{pid = pid}} \]

Product(pid, name, price) 
Purchase(pid, cid, store) 
Customer(cid, name, city)
Physical Query Plan

\( \sigma \text{name='Gizmo' \land \text{store}='GizmoMart'} \)

\( \Pi \text{name, price} \)

Product

Purchase

\( \text{File scan} \)

\( \text{File scan} \)

Customer
Outline of the Lecture

• Physical operators: join, group-by

• Query execution: pipeline, iterator model

• Database statistics
Extended Algebra Operators

- Union $\cup$, difference $-$
- Selection $\sigma$
- Projection $\Pi$
- Join $\Join$ -- also: semi-join, anti-semi-join
- Rename $\rho$
- Duplicate elimination $\delta$
- Grouping and aggregation $\gamma$
- Sorting $\tau$
Sets v.s. Bags

- Sets: \{a,b,c\}, \{a,d,e,f\}, \{\}\, \ldots
- Bags: \{a, a, b, c\}, \{b, b, b, b, b\}, \ldots

Relational Algebra has two semantics:
- Set semantics (paper “Three languages…”)
- Bag semantics
Physical Operators

Each of the logical operators may have one or more implementations = physical operators

Will discuss several basic physical operators, with a focus on join
Main Memory Algorithms

Logical operator:

\[ \text{Product}(\text{pid}, \text{name}, \text{price}) \bowtie_{\text{pid} = \text{pid}} \text{Purchase}(\text{pid}, \text{cid}, \text{store}) \]

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

1.
2.
3.
Main Memory Algorithms

Logical operator:

\[ \text{Product}(\text{pid}, \text{name}, \text{price}) \Join_{\text{pid} = \text{pid}} \text{Purchase}(\text{pid}, \text{cid}, \text{store}) \]

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

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

Logical operator:

\[ \text{Product}(\text{pid}, \text{name}, \text{price}) \bowtie_{\text{pid} = \text{pid}} \text{Purchase}(\text{pid}, \text{cid}, \text{store}) \]

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

1. Nested Loop Join \( O(n^2) \)
2. Merge join \( O(n \log n) \)
3. Hash join \( O(n) \ldots O(n^2) \)
BRIEF Review of Hash Tables

Separate chaining:

A (naïve) hash function:

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

Operations:

\[ \text{find}(103) = ?? \]
\[ \text{insert}(488) = ?? \]
BRIEF Review of Hash Tables

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

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

• \text{find}(k) = \text{return the } \textbf{list} \text{ of all values } v \text{ associated to the key } k
External Memory Algorithms

The cost of an operation = total number of I/Os

Cost parameters (used both in the book and by Shapiro):

- \( B(R) \) = number of blocks for relation R (Shapiro: |R|)
- \( T(R) \) = number of tuples in relation R
- \( V(R, a) \) = number of distinct values of attribute a
- \( M \) = size of main memory buffer pool, in blocks

Facts: (1) \( B(R) << T(R) \):
(2) When a is a key, \( V(R, a) = T(R) \)
    When a is not a key, \( V(R, a) << T(R) \)
Cost of an Operator

**Assumption**: runtime dominated by # of disk I/O’s; will ignore the main memory part of the runtime

- If R (and S) fit in main memory, then we use a main-memory algorithm
- If R (or S) does not fit in main memory, then we use an external memory algorithm
Ad-hoc Convention

• The operator *reads* the data from disk
  – Note: different from Shapiro
• The operator *does not write* the data back to disk (e.g.: pipelining)
• Thus:

Any main memory join algorithms for \( R \bowtie S \): Cost = \( B(R) + B(S) \)

Any main memory grouping \( \gamma(R) \): Cost = \( B(R) \)
Nested Loop Joins

- Tuple-based nested loop $R \bowtie S$

```plaintext
for each tuple $r$ in $R$ do
  for each tuple $s$ in $S$ do
    if $r$ and $s$ join then output $(r,s)$
```

- Cost: $T(R) B(S)$
Examples

M = 4

• Example 1:
  – B(R) = 1000, T(R) = 10000
  – B(S) = 2, T(S) = 20
  – Cost = ?

• Example 2:
  – B(R) = 1000, T(R) = 10000
  – B(S) = 4, T(S) = 40
  – Cost = ?

Can you do better with nested loops?
for each (M-2) blocks $bs$ of $S$ do
  for each block $br$ of $R$ do
    for each tuple $s$ in $bs$
      for each tuple $r$ in $br$ do
        if “$r$ and $s$ join” then output($r,s$)
Block-Based Nested-loop Join

for each (M-2) blocks $bs$ of $S$ do
  for each block $br$ of $R$ do
    for each tuple $s$ in $bs$
      for each tuple $r$ in $br$ do
        if “$r$ and $s$ join” then output($r,s$)

Terminology alert: sometimes $S$ is called the *inner* relation
Block-Based Nested-loop Join

for each (M-2) blocks $bs$ of $S$ do
  for each block $br$ of $R$ do
    for each tuple $s$ in $bs$
      for each tuple $r$ in $br$ do
        if “$r$ and $s$ join” then output($r,s$)

Terminology alert: sometimes $S$ is called $S$ the inner relation
Block Nested-loop Join

- **R & S**
- **Hash table for block of S (M-2 pages)**
- **Input buffer for R**
- **Output buffer**
- **Join Result**
Examples

M = 4

• Example 1:
  – B(R) = 1000, T(R) = 10000
  – B(S) = 2, T(S) = 20
  – Cost = B(S) + B(R) = 1002

• Example 2:
  – B(R) = 1000, T(R) = 10000
  – B(S) = 4, T(S) = 40
  – Cost = B(S) + 2B(R) = 2004

Note: T(R) and T(S) are irrelevant here.
Cost of Block Nested-loop Join

• Read S once: cost $B(S)$
• Outer loop runs $\frac{B(S)}{M-2}$ times, and each time need to read R: costs $\frac{B(S)B(R)}{M-2}$

Cost = $B(S) + \frac{B(S)B(R)}{M-2}$
Index Based Selection

Recall IMDB; assume indexes on Movie.id, Movie.year

\[
\text{SELECT } * \text{ FROM Movie WHERE id = '12345'}
\]

B(Movie) = 10k
T(Movie) = 1M

What is your estimate of the I/O cost?

\[
\text{SELECT } * \text{ FROM Movie WHERE year = '1995'}
\]
Index Based Selection

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

• **Clustered** index on a: cost ?

• **Unclustered** index : cost ?
Index Based Selection

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

- **Clustered** index on a: cost \( B(R)/V(R,a) \)

- **Unclustered** index: cost \( T(R)/V(R,a) \)
Index Based Selection

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

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

- **Unclustered** index: cost $T(R)/V(R,a)$

Note: we assume that the cost of reading the index = 0

Why?
Index Based Selection

• Example:
  \[ B(R) = 10k \]
  \[ T(R) = 1M \]
  \[ V(R, a) = 100 \]

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

• Table scan:
  – \[ B(R) = 10k \text{ I/Os} \]

• Index based selection:
  – If index is clustered: \[ B(R)/V(R,a) = 100 \text{ I/Os} \]
  – If index is unclustered: \[ T(R)/V(R,a) = 10000 \text{ I/Os} \]

Rule of thumb:
don’t build unclustered indexes when \( V(R,a) \) is small !
Index Based Join

- $R \Join S$
- Assume $S$ has an index on the join attribute

**for** each tuple $r$ in $R$ **do**

  lookup the tuple(s) $s$ in $S$ using the index

  output $(r,s)$
Index Based Join

Cost:

• If index is clustered:
• If unclustered:
Index Based Join

Cost:

• If index is clustered: \( B(R) + T(R)B(S)/V(S,a) \)
• If unclustered: \( B(R) + T(R)T(S)/V(S,a) \)
Operations on Very Large Tables

• Compute \( R \bowtie S \) when each is larger than main memory

• Two methods:
  – Partitioned hash join (many variants)
  – Merge-join

• Similar for grouping
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, when $B < M^2$
Basic Terminology

• A run in a sequence is an increasing subsequence

• What are the runs?

2, 4, 99, 103, 88, 77, 3, 79, 100, 2, 50
External Merge-Sort: Step 1

• Phase one: load M bytes in memory, sort

Disk → Main memory → Disk

Runs of length M bytes
Basic Terminology

• Merging multiple runs to produce a longer run:
  0, 14, 33, 88, 92, 192, 322
  2, 4, 7, 43, 78, 103, 523
  1, 6, 9, 12, 33, 52, 88, 320

Output:
  0, 1, 2, 4, 6, 7, ?
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
Cost of External Merge Sort

• Read+write+read = 3B(R)

• Assumption: B(R) <= M^2
External Merge-Sort

Can increase to length 2M using “replacement selection”
Group-by

Group-by: $\gamma_a, \text{sum}(b) (R)$

- Idea: do a two step merge sort, but change one of the steps

- Question in class: which step needs to be changed and how?

Cost = $3B(R)$
Assumption: $B(\delta(R)) \leq M^2$
Merge-Join

Join R ⨝ S

• How?....
Merge-Join

Join $R \Join S$

- Step 1a: initial runs for $R$
- Step 1b: initial runs for $S$
- Step 2: merge and join
Merge-Join

\[ M_1 = \frac{B(R)}{M} \text{ runs for } R \]
\[ M_2 = \frac{B(S)}{M} \text{ runs for } S \]
Merge-join \( M_1 + M_2 \) runs;
need \( M_1 + M_2 \leq M \)
Partitioned Hash Algorithms

Idea:

• If \( B(R) > M \), then partition it into smaller files: \( R_1, R_2, R_3, \ldots, R_k \)

• Assuming \( B(R_1) = B(R_2) = \ldots = B(R_k) \), we have
  \[ B(R_i) = \frac{B(R)}{k} \]

• Goal: each \( R_i \) should fit in main memory:
  \[ B(R_i) \leq M \]

How big can \( k \) be?
Partitioned Hash Algorithms

- Idea: partition a relation $R$ into $M-1$ buckets, on disk
- Each bucket has size approx. $B(R)/(M-1) \approx B(R)/M$

Assumption: $B(R)/M \leq M$, i.e. $B(R) \leq M^2$
Grouping

- $\gamma(R)$ = grouping and aggregation
- Step 1. Partition $R$ into buckets
- Step 2. Apply $\gamma$ to each bucket (may read in main memory)

- Cost: $3B(R)$
- Assumption: $B(R) \leq M^2$
Note: grace-join is also called partitioned hash-join.
Grace-Join

\[ R \bowtie S \]

- **Step 1:**
  - Hash S into M buckets
  - Send all buckets to disk
- **Step 2**
  - Hash R into M buckets
  - Send all buckets to disk
- **Step 3**
  - Join every pair of buckets

Note: grace-join is also called *partitioned hash-join*
**Grace-Join**

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

![Diagram of Grace-Join process](image)
Grace-Join

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

- Read in a partition of R, hash it using $h2 (<> h!)$. Scan matching partition of S, search for matches.
Grace Join

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

- 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

- Partition $S$ into $k$ buckets
  - $t$ buckets $S_1, ..., S_t$ stay in memory
  - $k-t$ buckets $S_{t+1}, ..., S_k$ 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:
  $(R_{t+1}, S_{t+1}), (R_{t+2}, S_{t+2}), ..., (R_k, S_k)$

Shapiro’s notation:

$1/(B+1) = t/k$ in main memory
$B/(B+1) = (k-t)/k$ go to disk
Hybrid Hash Join Algorithm
Hybrid Join Algorithm

• How to choose $k$ and $t$?
  – Choose $k$ large but s.t. $k \leq M$
  – Choose $t/k$ large but s.t. $t/k \cdot B(S) \leq M$
  – Moreover: $t/k \cdot B(S) + k-t \leq M$

• Assuming $t/k \cdot B(S) \gg k-t$: $t/k = M/B(S)$
Hybrid Join Algorithm

Cost of Hybrid Join:

• **Grace join:** $3B(R) + 3B(S)$
• **Hybrid join:**
  - Saves $2$ I/Os for $t/k$ fraction of buckets
  - Saves $2t/k(B(R) + B(S))$ I/Os
  - Cost:
    $$(3-2t/k)(B(R) + B(S)) = (3-2M/B(S))(B(R) + B(S))$$
Hybrid Join Algorithm

• Question in class: what is the advantage of the hybrid algorithm?
Summary of External Join Algorithms

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

- Index Join: $B(R) + T(R)B(S)/V(S,a)$

- Partitioned Hash: $3B(R)+3B(S)$;
  - $\min(B(R),B(S)) \leq M^2$

- Merge Join: $3B(R)+3B(S)$
  - $B(R)+B(S) \leq M^2$
Outline of the Lecture

• Physical operators: join, group-by

• Query execution: pipeline, iterator model

• Database statistics
Iterator Interface

Each operator implements this interface

• open()
  – Initializes operator state
  – Sets parameters such as selection condition

• get_next()
  – Operator invokes get_next() recursively on its inputs
  – Performs processing and produces an output tuple

• close(): cleans-up state
1. Nested Loop Join

for x in Product do {
    for y in Purchase do {
        if (x.pid == y.pid) output(x,y);
    }
}

Product = *outer relation*
Purchase = *inner relation*

Note: sometimes terminology is switched

When is it more efficient to iterate first over Purchase, then over Product?
It’s more complicated…

- Each **operator implements this interface**
  - `open()`
  - `get_next()`
  - `close()`
Main Memory Nested Loop Join

open ( ) { 
    Product.open( );
    Purchase.open( );
    x = Product.get_next( );
}

close ( ) { 
    Product.close ( );
    Purchase.close ( );
}

get_next( ) { 
    repeat {
        y = Purchase.get_next( );
        if (y == NULL) {
            Purchase.close();
            Purchase.open( );
            x = Product.get_next( );
            if (x== NULL) return NULL;
            y = Purchase.get_next( );
        }
    until (x.pid == y.pid);
    return (x,y)
}

ALL operators need to be implemented this way!
2. Hash Join (main memory)

for x in Product do  insert(x.pid, x);

for y in Purchase do {
    ys = find(y.pid);
    for y in ys do { output(x,y); }
}

Recall: need to rewrite as open, get_next, close
3. Merge Join (main memory)

Product1 = sort(Product, pid);
Purchase1 = sort(Purchase, pid);

x=Product1.get_next();
y=Purchase1.get_next();

While (x!=NULL and y!=NULL) {
    case:
        x.pid < y.pid:  x = Product1.get_next( );
        x.pid > y.pid:  y = Purchase1.get_next();
        x.pid == y.pid { output(x,y);
            y = Purchase1.get_next();
        }
}

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)
Physical Query Plan

\[ \Pi_{\text{name, price}} \]
\[ \sigma_{\text{name='Gizmo' \land store='GizmoMart'}} \]
\[ \text{pid = pid} \]
\[ \text{Product (On the fly)} \]
\[ \text{Purchase (On the fly)} \]
\[ \text{Customer (Nested loop)} \]

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)
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
Physical Query Plan

\[ \Pi_{\text{name, price}} \]

\( \sigma \text{name='Gizmo' } \land \text{store = 'GizmoMart'} \)

(On the fly)

(Materialize to T1)

(Sort-merge join)

Product(pid, name, price)

Purchase(pid, cid, store)

Customer(cid, name, city)

Product(pid)

Purchase(pid)

Customer(cid)

File scan

File scan

On the fly

Materialize to T1

Sort-merge join
Intermediate Tuple Materialization

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

- No direct benefit but
- Necessary data is larger than main memory
- Necessary when operator needs to examine the same tuples multiple times
Outline of the Lecture

• Physical operators: join, group-by

• Query execution: pipeline, iterator model

• Database statistics
  – Partially based on *Graphical Models* paper
Database Statistics

• Collect statistical summaries of stored data

• Estimate size (=cardinality), bottom-up

• Estimate cost by using the estimated size
Database Statistics

- Number of tuples = cardinality
- Indexes: number of keys in the index
- Number of physical pages, clustering info
- Statistical information on attributes
  - Min value, max value, number distinct values
  - Histograms
- Correlations between columns

Collection approach: periodic, using sampling
Size Estimation Problem

\[ S = \text{SELECT list} \]
\[ \text{FROM} \quad R1, \ldots, \text{Rn} \]
\[ \text{WHERE} \quad \text{cond}_1 \text{AND cond}_2 \text{AND} \ldots \text{AND cond}_k \]

Given \( T(R1), T(R2), \ldots, T(Rn) \)
Estimate \( T(S) \)

How can we do this? Note: doesn’t have to be exact.
Size Estimation Problem

\[
S = \text{SELECT list} \\
\text{FROM } R_1, \ldots, R_n \\
\text{WHERE } \text{cond}_1 \text{ AND } \text{cond}_2 \text{ AND } \ldots \text{ AND } \text{cond}_k
\]

Remark: \( T(S) \leq T(R_1) \times T(R_2) \times \ldots \times T(R_n) \)
Selectivity Factor

• Each condition $cond$ reduces the size by some factor called *selectivity factor*

• Assuming independence, multiply the selectivity factors
Example

R(A,B)  
S(B,C)  
T(C,D)  

SELECT *  
FROM R, S, T  
WHERE R.B=S.B and S.C=T.C and R.A<40

T(R) = 30k, T(S) = 200k, T(T) = 10k

Selectivity of R.B = S.B is 1/3
Selectivity of S.C = T.C is 1/10
Selectivity of R.A < 40 is ½

What is the estimated size of the query output?
Example

\[ \text{SELECT} \ast \]
\[ \text{FROM} \ R, S, T \]
\[ \text{WHERE} \ R.B = S.B \ \text{and} \ S.C = T.C \ \text{and} \ R.A < 40 \]

T(R) = 30k, T(S) = 200k, T(T) = 10k

Selectivity of \( R.B = S.B \) is 1/3
Selectivity of \( S.C = T.C \) is 1/10
Selectivity of \( R.A < 40 \) is \( \frac{1}{2} \)

What is the estimated size of the query output?

\[ 30k \times 200k \times 10k \times \frac{1}{3} \times \frac{1}{10} \times \frac{1}{2} = 1 \text{TB} \]
What is the probability space?

\[
S = \text{SELECT list} \\
\text{FROM } \ R_1 \text{ as } x_1, \ldots, \ R_k \text{ as } x_k \\
\text{WHERE } \text{Cond} \quad \text{-- a conjunction of predicates}
\]
Discussion: Paper

What is the probability space?

\[ S = \text{SELECT list} \]
\[ \text{FROM } R_1 \text{ as } x_1, \ldots, R_k \text{ as } x_k \]
\[ \text{WHERE } \text{Cond} -- \text{a conjunction of predicates} \]

\((x_1, x_2, \ldots, x_k), \text{drawn randomly, independently from } R_1, \ldots, R_k\)

\(\Pr(R_1.A = 40) = \text{prob. that random tuple in } R_1 \text{ has } A=40\)

\(\Pr(R_1.A = 40 \text{ and } J_{R_1.B = R_2.C} \text{ and } R_2.D = 90) = \text{prob. that } \ldots\)

\[ E[ |\text{SELECT ... WHERE Cond}| ] = \Pr(\text{Cond}) * T(R_1) * T(R_2) * \ldots * T(R_k) \]
Discussion: Paper

What is the probability space?

\[ S = \text{SELECT} \ \text{list} \]
\[ \text{FROM} \ \ R_1 \ \text{as} \ x_1, \ldots, \ R_k \ \text{as} \ x_k \]
\[ \text{WHERE} \ \text{Cond} \ -- \ a \ conjunction \ of \ predicates \]

What are the three simplifying assumptions?
What is the probability space?

\[ S = \text{SELECT list} \]
\[ \text{FROM} \quad R_1 \text{ as } x_1, \ldots, R_k \text{ as } x_k \]
\[ \text{WHERE} \quad \text{Cond} \quad \text{-- a conjunction of predicates} \]

What are the three simplifying assumptions?

**Uniform:** \[ \Pr(R_1.A = 'a') = \frac{1}{V(R_1, A)} \]

**Attribute Indep.:** \[ \Pr(R_1.A = 'a' \text{ and } R_1.B = 'b') = \Pr(R_1.A = 'a') \Pr(R_1.B = 'b') \]

**Join Indep.:** \[ \Pr(R_1.A = 'a' \text{ and } J_{R_1.B = R_2.C}) = \Pr(R_1.A = 'a') \Pr(J_{R_1.B = R_2.C}) \]
Rule of Thumb

• If selectivities are unknown, then:
  selectivity factor = 1/10
  [System R, 1979]
Using Data Statistics

- **Condition is** $A = c$  /* value selection on $R$ */
  - Selectivity $= \frac{1}{V(R, A)}$

- **Condition is** $A < c$  /* range selection on $R$ */
  - Selectivity $= \frac{c - \text{Low}(R, A)}{\text{High}(R, A) - \text{Low}(R, A)}T(R)$

- **Condition is** $A = B$  /* $R \bowtie_{A=B} S$ */
  - Selectivity $= \frac{1}{\max(V(R, A), V(S, A))}$
  - (will explain next)
Selectivity of Join Predicates

Assumptions:

• **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 $S$

• **Preservation of values**: for any other attribute $C$, $V(R \bowtie_{A=B} S, C) = V(R, C)$ (or $V(S, C)$)
Selectivity of Join Predicates

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

- Each tuple $t$ in $R$ joins with $T(S)/V(S,B)$ tuple(s) in $S$

- Hence $T(R \bowtie_{A=B} S) = T(R) \cdot T(S) / V(S,B)$

In general: $T(R \bowtie_{A=B} S) = T(R) \cdot T(S) / \max(V(R,A),V(S,B))$
Selectivity of Join Predicates

Example:
• $T(R) = 10000$, $T(S) = 20000$
• $V(R,A) = 100$, $V(S,B) = 200$
• How large is $R \bowtie_{A=B} S$?
Histograms

- Statistics on data maintained by the RDBMS
- Makes size estimation much more accurate (hence, cost estimations are more accurate)
Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Employee, age) = 50
min(age) = 19, max(age) = 68

σ_{age=48}(Employee) = ? \quad σ_{age>28 \text{ and } age<35}(Employee) = ?
Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Employee, age) = 50
min(age) = 19, max(age) = 68

\[ \sigma_{age=48}(Employee) = ? \]
\[ \sigma_{age>28 \text{ and } age<35}(Employee) = ? \]

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
</tbody>
</table>
**Histograms**

**Employee**(ssn, name, age)

T(Employee) = 25000, \( V(Employee, age) = 50 \)
min(age) = 19, max(age) = 68

\[ \sigma_{age=48}(Employee) = ? \quad \sigma_{age>28 \ and \ age<35}(Employee) = ? \]

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
</tbody>
</table>

Estimate = 1200

Estimate = \( 1 \times 80 + 5 \times 500 = 2580 \)
Types of Histograms

• How should we determine the bucket boundaries in a histogram?
Types of Histograms

• How should we determine the bucket boundaries in a histogram?

• Eq-Width
• Eq-Depth
• Compressed
• V-Optimal histograms
**Employee(ssn, name, age)**

### Histograms

**Eq-width:**

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
</tbody>
</table>

**Eq-depth:**

<table>
<thead>
<tr>
<th>Age:</th>
<th>0..20</th>
<th>20..29</th>
<th>30-39</th>
<th>40-49</th>
<th>50-59</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>1800</td>
<td>2000</td>
<td>2100</td>
<td>2200</td>
<td>1900</td>
<td>1800</td>
</tr>
</tbody>
</table>

**Compressed:** store separately highly frequent values: (48,1900)
V-Optimal Histograms

• Defines bucket boundaries in an optimal way, to minimize the error over all point queries
• Computed rather expensively, using dynamic programming
• Modern databases systems use V-optimal histograms or some variations
Difficult Questions on Histograms

• Small number of buckets
  – Hundreds, or thousands, but not more
  – WHY?

• *Not* updated during database update, but recomputed periodically
  – WHY?
Multidimensional Histograms

Classical example:

SQL query:  
```
SELECT ... FROM ...  
WHERE Person.city = 'Seattle' ...
```

User “optimizes” it to:

```
SELECT ... FROM ...  
WHERE Person.city = 'Seattle'  
and Person.state = 'WA'
```

Big problem! (Why?)
Multidimensional Histograms

- Store distributions on two or more attributes
- Curse of dimensionality: space grows exponentially with dimension
- Paper: discusses using only two dimensional histograms
Paper: Bayesian Networks

\[ P_{BN}(A, B, C, D, E) = P(E|D)P(D|B)P(C|A, B) \ P(A)P(B). \]
Paper: Bayesian Networks

\[ P_{BN}(A, B, C, D, E) = P(E|D)P(D|B)P(C|A, B) \ P(A)P(B). \]
$P_{BN}(A, B, C, D, E) = P(E|D)P(D|B)P(C|A, B)P(A)P(B)$. 

Fig. 1 A small graphical model of five binary random variables $A, B, C, D, E$. 

- **a** Bayesian network. 
- **b** Moral graph. 
- **c** Junction tree. 
- **d** Clique potentials

$$P(A, D) = \sum_{B,C} \frac{P(A, B, C)P(B, D)}{P(B)}.$$
Paper Highlights

• Universal table (what is it?)
• Acyclic v.s. Cyclic Schemas
• Within a table: tree-BN only
• Join indicator: two parents only
• Hence: acyclic schema $\rightarrow$ 2D-histograms only in the junction tree
• Simplifies construction, estimation
Next Lecture

Plan:
• Revisit Grace join after you read the paper
• Query optimization
• Latest results in optimal query processing
• Start Parallel DBs