Distributed Databases

CS347

Lecture 13

May 23, 2001

Expected Background

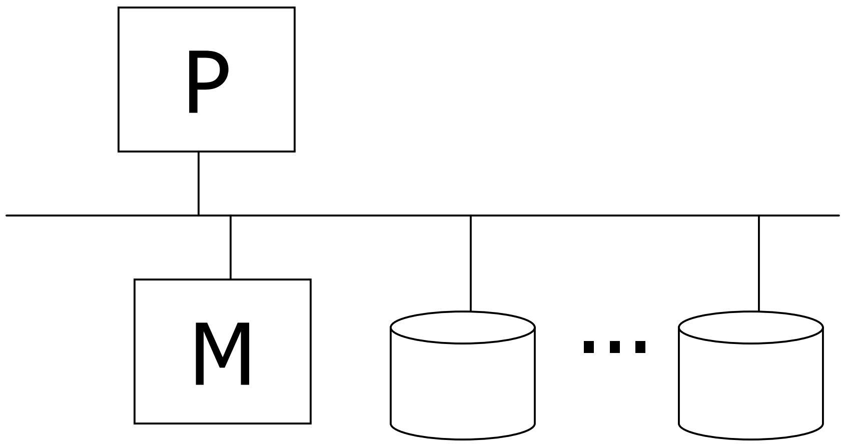
- Basic SQL
- Relational algebra
- Following aspects of centralized DB
  - Query processing: query plans, cost estimation, optimization
  - Concurrency control techniques
  - Recovery methods

Reading Material

- Primarily lecture notes
- No required textbook
- Some lecture material drawn from

  M. Tamer Ozsu and Patrick Valduriez, "Principles of Distributed Database Systems," Second Edition, Prentice Hall 1999.

Centralized DBMS



Software:

Application
SQL Front End
Query Processor
Transaction Proc.
File Access

- Simplifications:
  - single front end
  - one place to keep locks
  - if processor fails, system fails, …..

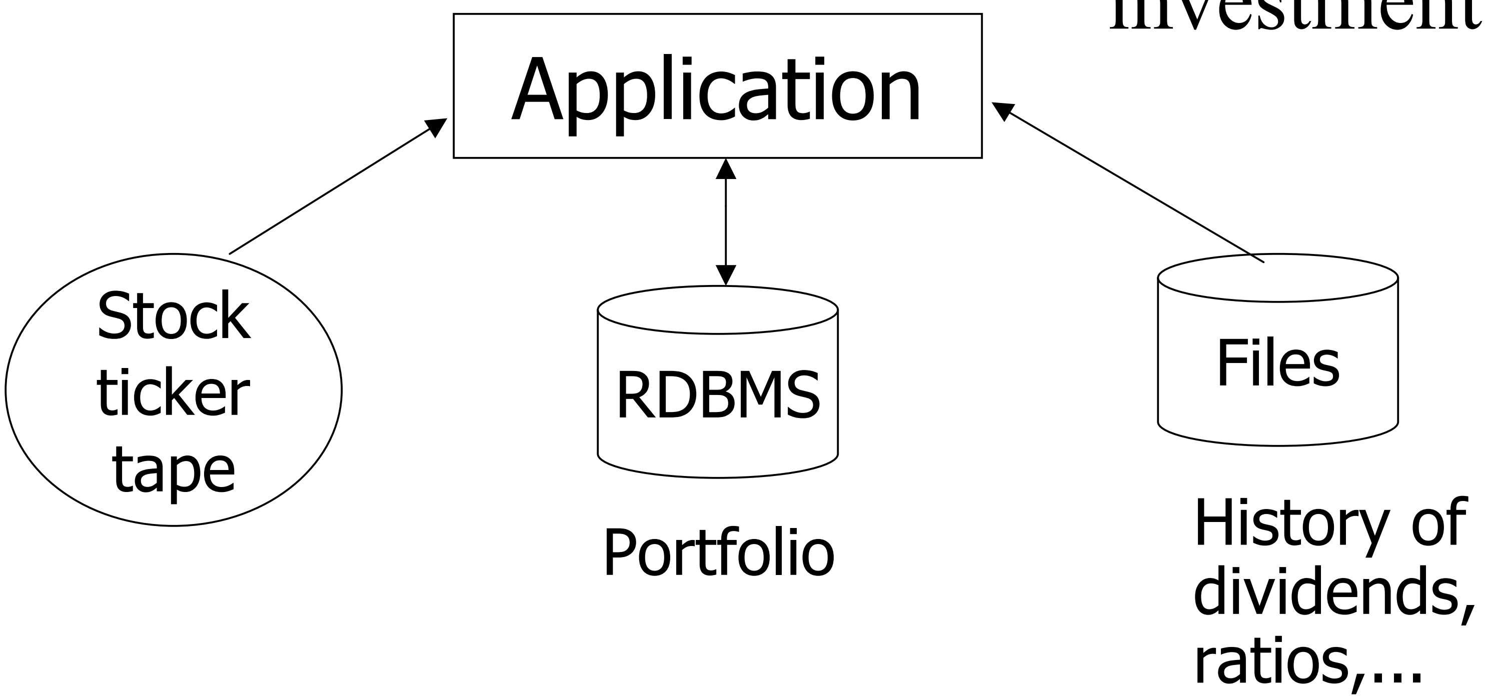
Distributed DB

- Multiple processors, memories, and disks
  - Opportunity for parallelism (+)
  - Opportunity for enhanced reliability (+)
  - Synchronization issues (-)

- Heterogeneity and autonomy of "components"
  - Autonomy example: may not get statistics for query optimization from a site

Heterogeneity

Select new investments



Big Picture

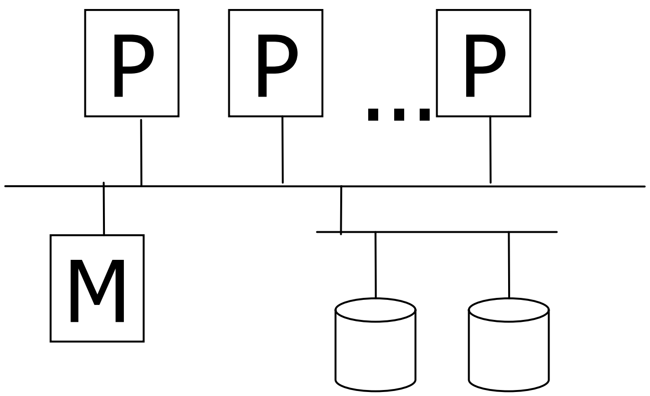
Data management with multiple processors and possible autonomy, heterogeneity.
Impacts:

- Data organization
- Query processing
- Access structures
- Concurrency control
- Recovery

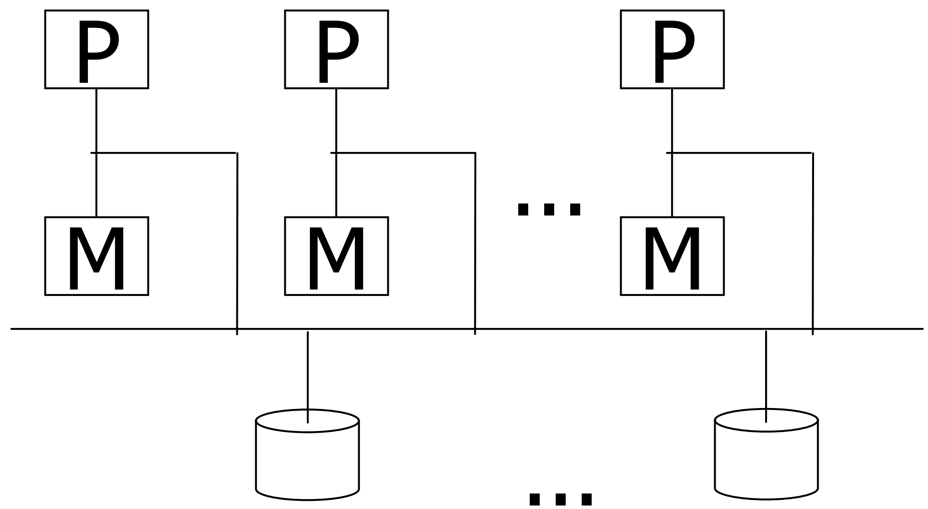
Today's topics

- Introductory topics
  - Database architectures
  - Distributed versus Parallel DB systems
- Distributed database design
  - Fragmentation
  - Allocation

Common DB architectures



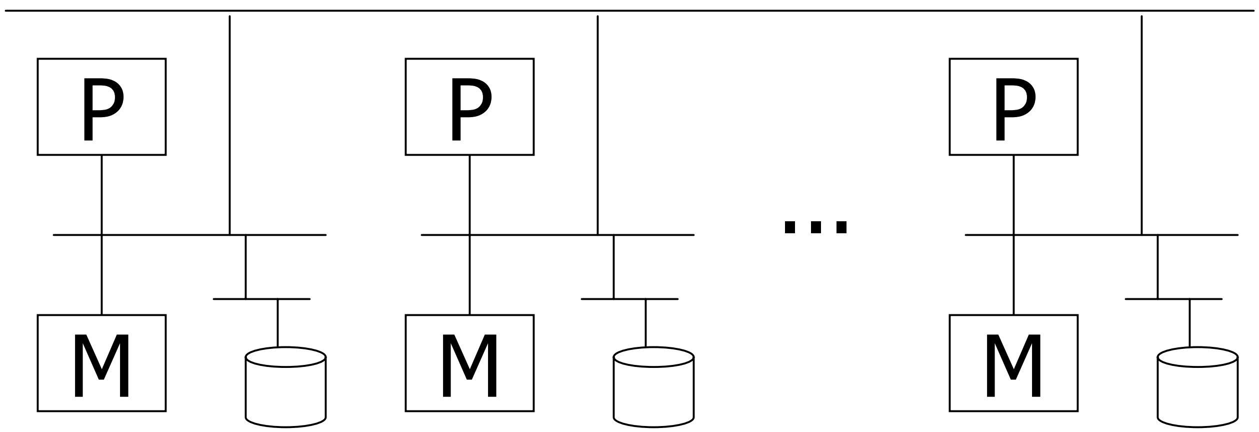
Shared memory



Shared disk

Common DB architectures

Shared nothing



Number of other "hybrid" architectures are possible.

Selecting the “right” architecture

- Reliability
- Scalability
- Geographic distribution of data
- Performance
- Cost

Parallel vs. Distributed DB system

- Typically, parallel DBs:
  - Fast interconnect
  - Homogeneous software
  - Goals: High performance and Transparency

- Typically, distributed DBs:
  - Geographically distributed
  - Disconnected operation possible
  - Goal: Data sharing (heterogeneity, autonomy)

Typical query processing scenarios

- Parallel DB:
  - Distribute/partition/sort…. data to make certain DB operations (e.g., Join) fast
- Distributed DB:
  - Given data distribution, find query processing strategy to minimize cost (e.g. communication cost)

Distributed DB Design

Top-down approach:

- have a database
- how to split and allocate to individual sites

Multi-databases (or bottom-up):

- combine existing databases
- how to deal with heterogeneity & autonomy

Two issues in top-down design

- Fragmentation
- Allocation

Note: issues not independent, but studied separately for simplicity.

Example

Employee relation E (#,name,loc,sal,…)

40% of queries:

Qa: select *
from E
where loc=Sa
and…

40% of queries:

Qb: select *
from E
where loc=Sb
and ...

Motivation: Two sites: Sa, Sb

Qa → Sa ← Sb

E

#	Name	Loc	Sal
5	Joe	Sa	10
7	Sally	Sb	25
8	Tom	Sa	15
:			:

$\mathbf{F} = \{F_1, F_2\}$

At Sa

5	Joe	Sa	10
8	Tom	Sa	15
:			

At Sb

7	Sally	Sb	25
:			

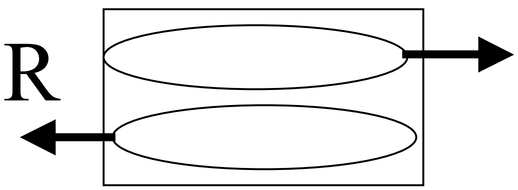
$F_1 = \sigma_{loc=Sa}(E)$ $F_2 = \sigma_{loc=Sb}(E)$

$\Rightarrow$ <u>primary horizontal</u> fragmentation

Fragmentation

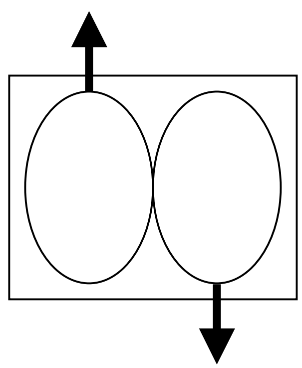
- Horizontal → Primary
  - depends on local attributes
- Derived
  - depends on foreign relation

R



- Vertical

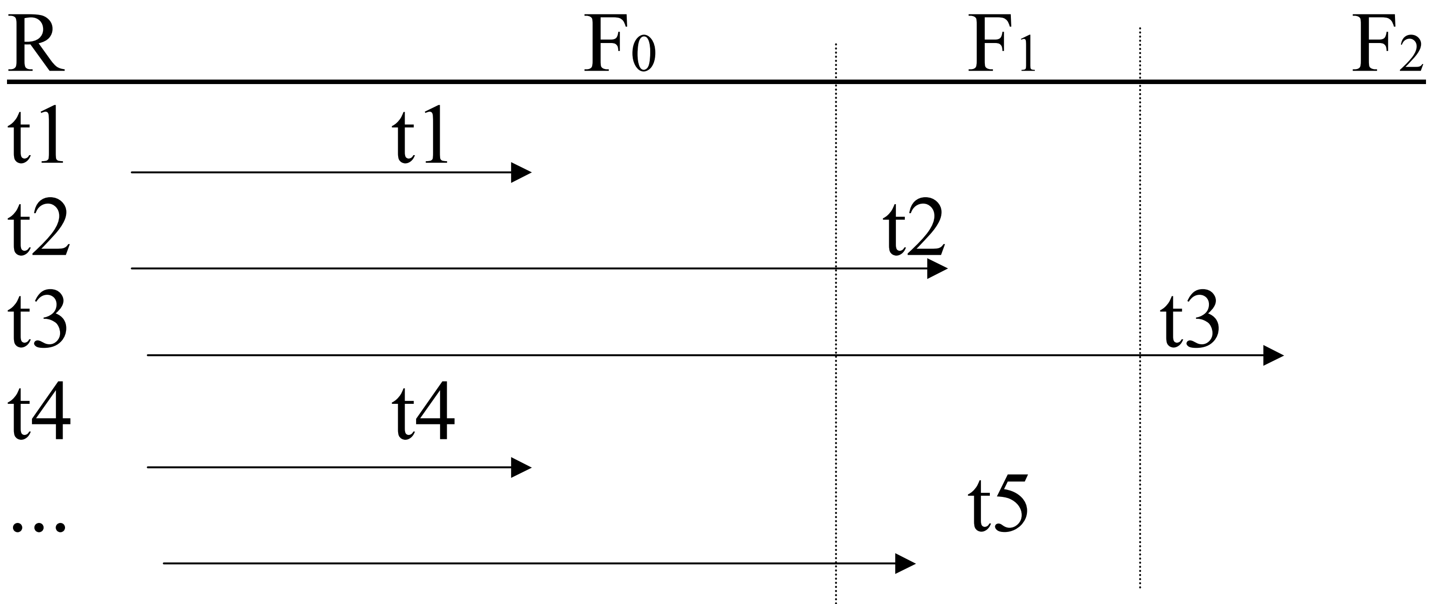
R



Horizontal partitioning techniques

- Round robin
- Hash partitioning
- Range partitioning

Round robin



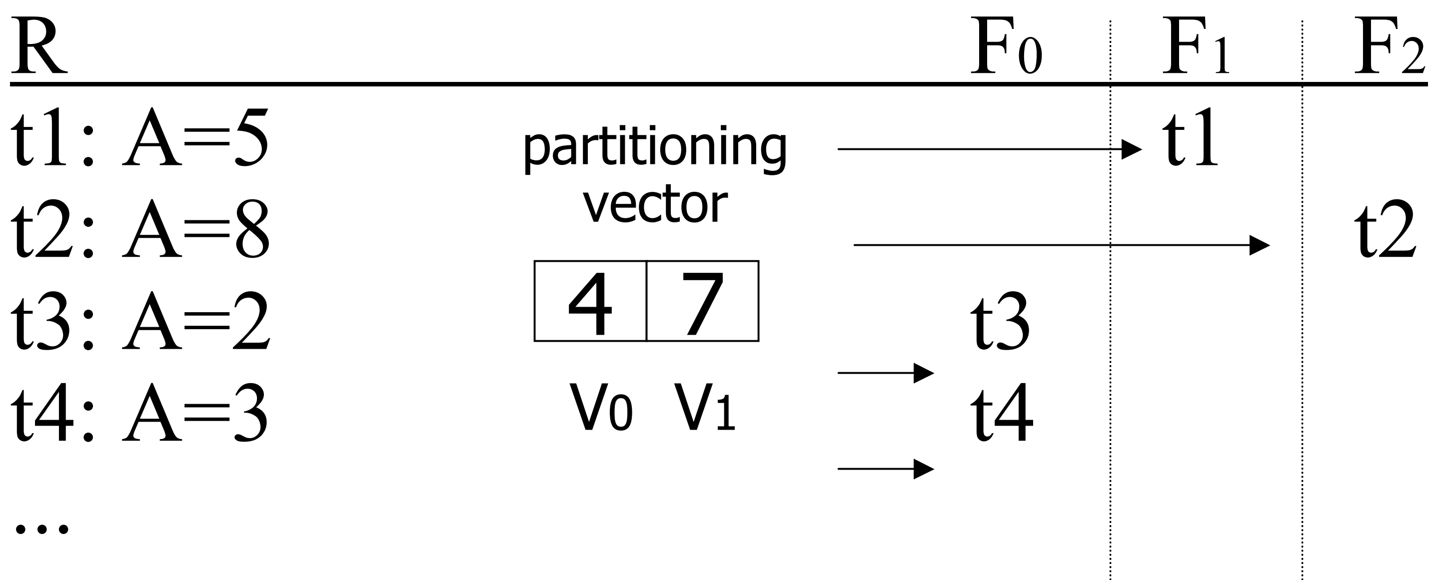
- Evenly distributes data
- Good for scanning full relation
- Not good for point or range queries

Hash partitioning

R	F_0	F_1	F_2
$t1 \rightarrow h(k_1)=2$			$\rightarrow$ t1
$t2 \rightarrow h(k_2)=0$	$\rightarrow$ t2		
$t3 \rightarrow h(k_3)=0$	$\rightarrow$ t3		
$t4 \rightarrow h(k_4)=1$		$\rightarrow$ t4	
...			

- Good for point queries on key; also for joins
- Not good for range queries; point queries not on key
- Good hash function $\Rightarrow$ even distribution

Range partitioning



- Good for some range queries on A
- Need to select good vector: else create imbalance

  → data skew

  → execution skew

Which are good fragmentations?

Example 1: $\mathbf{F} = \{F_1, F_2\}$

$$F_1 = \sigma_{sal<10}(E) \qquad F_2 = \sigma_{sal>20}(E)$$

Problem: Some tuples lost!

Example 2: $\mathbf{F} = \{F_3, F_4\}$

$$F_1 = \sigma_{sal<10}(E) \qquad F_2 = \sigma_{sal>5}(E)$$

Tuples with $5 < sal < 10$ are duplicated

Prefer to deal with replication explicitly

Example: $\mathbf{F} = \{ F_5, F_6, F_7 \}$

$$F_5 = \sigma_{sal<=5}(E)$$
$$F_6 = \sigma_{5 < sal<10}(E)$$
$$F_7 = \sigma_{sal>=10}(E)$$

☞Then replicate F_6 if desired as part of allocation

Horizontal Fragmentation Desiderata

$$R \Rightarrow \mathbf{F} = \{F_1, F_2, \ldots.\}$$

(1) Completeness

$$\forall t \in R, \ \exists F_i \in \mathbf{F} \text{ such that } t \in F_i$$

(2) Disjointness

$$F_i \cap F_j = \emptyset, \ \forall i,j \text{ such that } i \neq j$$

(3) Reconstruction

$$\exists \nabla \text{ such that } R = \overset{i}{\nabla} F_i$$

Generating horizontal fragments

- Given simple predicates $P_r = \{p_1, p_2,.. p_m\}$ and relation R.
- Generate "minterm" predicates

$$M = \{m \mid m = \wedge\, p_k^*,\ \ 1 \le k \le m\},\ \ \text{where}$$

$$p_k^* \text{ is either } p_k \text{ or } \neg\, p_k$$

- Eliminate useless minterms and simplify M to get M'.
- Generate fragments $\sigma_m(R)$ for each $m \in M'$.

Example

$5 < A < 10$

- Example: say queries use predicates

  $A < 10$, $A > 5$, $Loc = S_A$, $Loc = S_B$

- Eliminate and simplify minterms

  ~~$A < 10 \wedge A > 5 \wedge Loc = S_A \wedge Loc = S_B$~~

  $A < 10 \wedge A > 5 \wedge Loc = S_A \wedge \neg$~~$(Loc = S_B)$~~

- Final set of fragments

  $(5 < A < 10) \wedge (Loc = S_A)$

  $(5 < A < 10) \wedge (Loc = S_B)$

  $(A \leq 5) \wedge (Loc = S_A)$

  $(A \leq 5) \wedge (Loc = S_B)$

  $(A \geq 10) \wedge (Loc = S_A)$

  $(A \geq 10) \wedge (Loc = S_B)$

Work out details for all minterms.

More on Horizontal Fragmentation

- Elimination of useless fragments/predicates depends on application semantics:
  - e.g.: if Loc $\neq S_A$ and $\neq S_B$ is possible, must retain fragments such as $(5 < A < 10) \wedge (Loc \neq S_A) \wedge (Loc \neq S_B)$

- Minterm-based fragmentation generates complete, disjoint, and reconstructible fragments.

Justify this statement.

Choosing simple predicates

- E (#,name,loc,sal,…) with common queries

  Qa: select * from E where loc = S_A and…

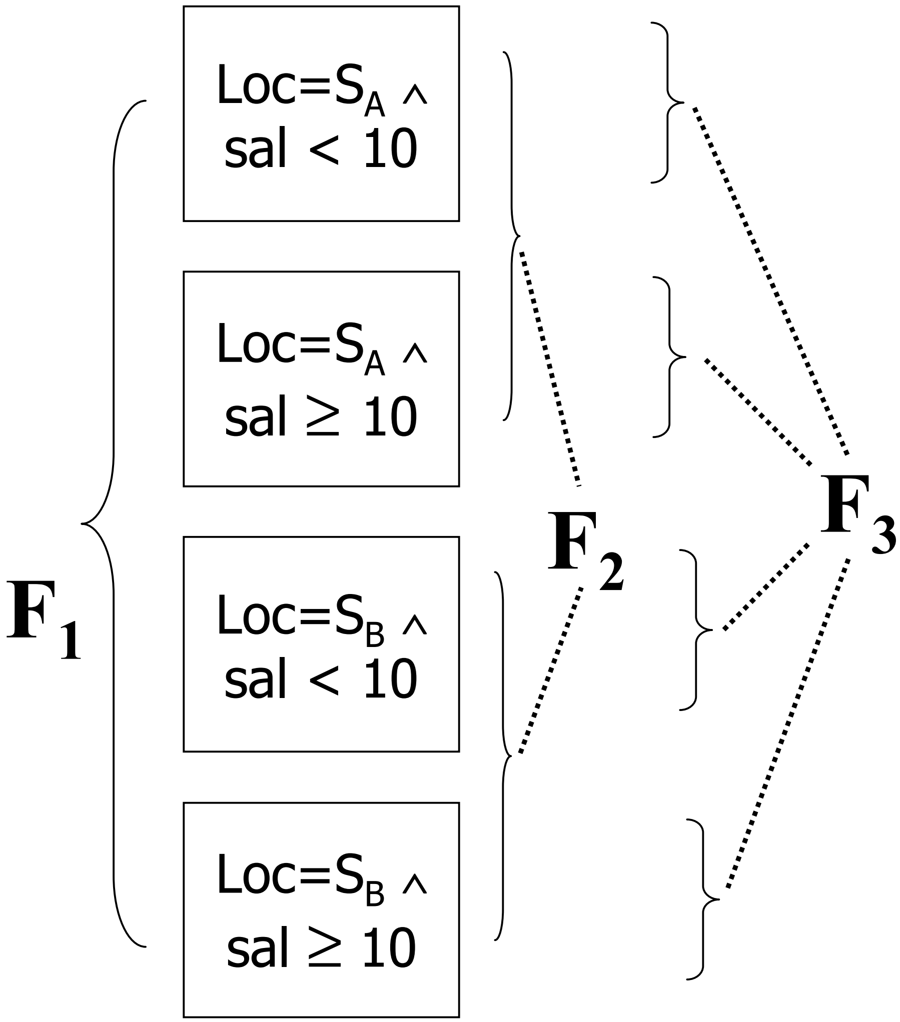
  Qb: select * from E where loc = S_B and…

- Three choices for P_r and hence $\mathbf{F}[P_r]$:
  - $P_r = \{\}$ $\mathbf{F_1} = \mathbf{F}[P_r] = \{E\}$
  - $P_r = \{Loc = S_A, Loc = S_B\}$

    $\mathbf{F_2} = \mathbf{F}[P_r] = \{\sigma_{loc=SA}(E), \sigma_{loc=SB}(E)\}$
  - $P_r = \{Loc = S_A, Loc = S_B, Sal < 10\}$

    $\mathbf{F_3} = \mathbf{F}[P_r] = \{\sigma_{loc=SA \wedge sal< 10}(E), \sigma_{loc=SB \wedge sal< 10}(E),$

    $\sigma_{loc=SA \wedge sal \geq 10}(E), \sigma_{loc=SA \wedge sal \geq 10}(E)\}$



Qa: Select … loc = S_A …

Qb: Select … loc = S_B …

Prefer $\mathbf{F_2}$ to $\mathbf{F_1}$ and $\mathbf{F_3}$

Desiderata for simple predicates

- Completeness

Different from completeness of fragmentation

Set of predicates P_r is complete if for every $F_i \in \mathbf{F}[P_r]$, every $t \in F_i$ has equal probability of access by every major application.

- Minimality

Set of predicates P_r is minimal if no $P_r' \subset P_r$ is complete.

To get complete and minimal P_r use predicates that are "relevant" in frequent queries

Derived horizontal fragmentation

- Example: Two relations Employee and Jobs
  - E(#, NAME, SAL, LOC)
  - J(#, DES,…)

- Fragment E into $\{E_1, E_2\}$ by LOC

- Common query:

  "Given employee name, list projects (s)he works in"

E_1

#	NM	Loc	Sal
5	Joe	Sa	10
8	Tom	Sa	15
…			

(at S_a)

E_2

#	NM	Loc	Sal
7	Sally	Sb	25
12	Fred	Sb	15
…			

(at S_b)

J

#	Description
5	work on 347 hw
7	go to moon
5	build table
12	rest
…	

E_1

#	NM	Loc	Sal
5	Joe	Sa	10
8	Tom	Sa	15
…			

(at S_a)

E_2

#	NM	Loc	Sal
7	Sally	Sb	25
12	Fred	Sb	15
…			

(at S_b)

J_1

#	Des
5	work on 347 hw
5	build table
…	

J_2

#	Des
7	go to moon
12	rest
…	

$J_1 = J \ltimes E_1$

$J_2 = J \ltimes E_2$

Derived horizontal fragmentation

R, fragmented as **F** = $\{F_1, F_2, \ldots, F_n\}$

$\Downarrow$

S, derive **D** = $\{D_1, D_2, \ldots, D_n\}$ where $D_i = S \ltimes F_i$

Convention: R is called the owner relation

S is called the member relation

Completeness of derived fragmentation

Example: Say J is

#	Des
…	
33	build chair
…	

- $J_1 \cup J_2 \subset J$ (incomplete fragmentation)
- For completeness, enforce referential integrity constraint

join attribute of member relation

$\Downarrow$

joint attribute of owner relation

E1

#	NM	Loc	Sal
5	Joe	Sa	10
…			

E2

#	NM	Loc	Sal
5	Fred	Sb	20
…			

J

#	Description
5	day off
…	

Fragmentation is not disjoint!

J1

#	Description
5	day off
…	

J2

#	Description
5	day off
…	

Common way to enforce disjointness: make join attribute key of owner relation.

Vertical fragmentation

Example: E

#	NM	Loc	Sal
5	Joe	Sa	10
7	Sally	Sb	25
8	Fred	Sa	15
…			

E_1

#	NM	Loc
5	Joe	Sa
7	Sally	Sb
8	Fred	Sa
…		

E_2

#	Sal
5	10
7	25
8	15
…	

$$R[T] \Rightarrow R_1[T_1], R_2[T_2], \ldots, R_n[T_n] \qquad T_i \subseteq T$$

➽ Just like normalization of relations

Properties

$$R[T] \Rightarrow R_i[T_i],\ i = 1..\ n$$

- Completeness: $\bigcup T_i = T$
- Reconstruction: $\bowtie R_i = R$ (lossless join)
  - One way to guarantee lossless join: repeat key in each fragment, i.e., key $\subseteq T_i\ \forall\ i$
- Disjointness: $T_i \cap T_j = \{key\}$
  - Check disjointness only on non-key attributes

Grouping Attributes

Example:

E(#,NM,LOC,SAL)

→ { E_1(#,NM,LOC), E_2(#,SAL) }

→ { E_1(#,NM), E_2(#,LOC), E_3(#,SAL) }

→ …..

Which is the right vertical fragmentation?

Attribute affinity matrix

	A_1	A_2	A_3	A_4	A_5
A_1	78	50	45	1	0
A_2	50	25	28	2	0
A_3	45	28	34	0	4
A_4	1	2	0	20	75
A_5	0	0	4	75	40

Cluster attributes based on affinity

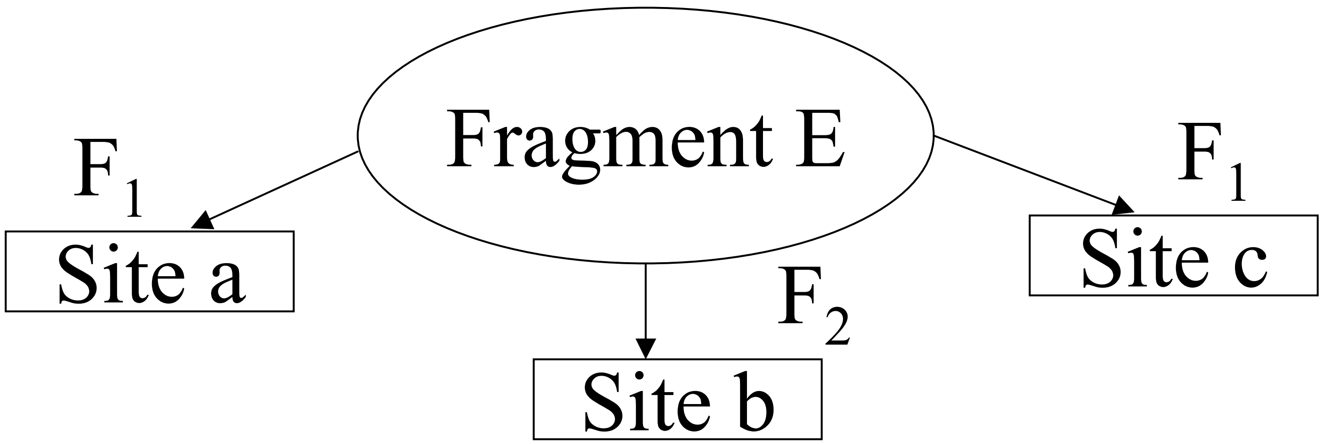
⬇

$R_1[K, A_1, A_2, A_3]$
$R_2[K, A_4, A_5]$

- $A_{i,j} \Rightarrow$ a measure of how "often" A_i and A_j are accessed by the same query
- Hand constructed using knowledge of queries and their frequencies

Allocation

Example: E $\Rightarrow F_1 = \sigma_{loc=Sa}(E)$; $F_2 = \sigma_{loc=Sb}(E)$



- Do we replicate fragments?
- Where do we place each copy of each fragment?

Issues

- Origin of queries
- Communication cost and size of answers, relations, etc.
- Storage capacity, storage cost at sites, and size of fragments
- Processing power at the sites
- Query processing strategy
  - How are joins done? Where are answers collected?
- Fragment replication
  - Update cost, concurrency control overhead

Optimization problem

- What is the best placement of fragments and/or best number of copies to:
  - minimize query response time
  - maximize throughput
  - minimize "some cost"
  - ...
- Subject to constraints
  - Available storage
  - Available bandwidth, processing power,…
  - Keep 90% of response time below X
  - ...

Very hard problem

Looking Ahead

- Query processing
  - Decomposition
  - Localization
  - Distributed query operators
  - Optimization (briefly)

Resources

- Ozsu and Valduriez. "Principles of Distributed Database Systems" – Chapter 5