PURDUE UNIVERSITY

CS54200: Distributed Database Systems

Final Review

1 May 2009

Prof. Chris Clifton

Indiana Center for Database Systems

2 Phase Locking

1. To grant a lock, the scheduler *checks if a conflicting lock* has already been assigned, if so, *delay*, otherwise *set lock and grant* it.
2. A lock cannot be released at least until the DM acknowledges that the operation has been performed.
3. *Once the scheduler releases a lock* for a txn, it may *not subsequently acquire any more locks* (on any item) for that txn.

2

Distributed 2PL

- 2PL easily extends to the distributed case.
- Each scheduler follows the same rules as before – if a lock can be acquired, process the operation.
- *No communication* needed – good.
- *Tricky issue: releasing locks!*
- In general would require communication.
- However, if STRICT 2PL is followed everywhere, then no communication is needed.
- Distributed, Strict 2PL is correct (*assuming that abort and commit operations are carried out atomically* – important issue that we will address later).

14

Distributed Deadlocks

- As with centralized 2PL, *distributed 2PL suffers from deadlocks*. Moreover, these can be distributed deadlocks! E.g. if *x* and *y* are at different sites.
- Solutions:
  - Timeouts
  - Deadlock Detection
  - Deadlock Prevention
- Timeouts are easy – local decision, but may be overreacting.

15

Timestamp Ordering

- The TM assigns each txn, T_i, a unique timestamp, *ts(Ti).*
- No two txns share a timestamp.
- A TO scheduler enforces:
- TO Rule: if $p_i[x]$ and $q_j[x]$ are conflicting operations, then the DM processes $p_i[x]$ before $q_j[x]$ iff $ts(T_i) < ts(T_j)$.

16

Distributed Timestamp Ordering

- Distributed TO: How can TO be modified for distributed sites?
- Simple – nothing special needed as long as ….
- *Timestamps are unique across sites!*
- Easy to enforce this.
- Much better than distributed 2PL – no need for inter-site communication, unlike 2PL which requires communication for deadlocks.

18

Recovery

- We will focus on system failures.
- Following the failure, the DBMS is restarted.
- At the start of recovery, the contents of *volatile storage are discarded.*
- The stable storage is potentially inconsistent
- A CONSISTENT database state corresponding to exactly the set of txns that had committed (as far as the DM is concerned) must be reconstructed, i.e. C(*H*).
- This reconstruction uses only data in stable storage – Stable DB and the LOG.

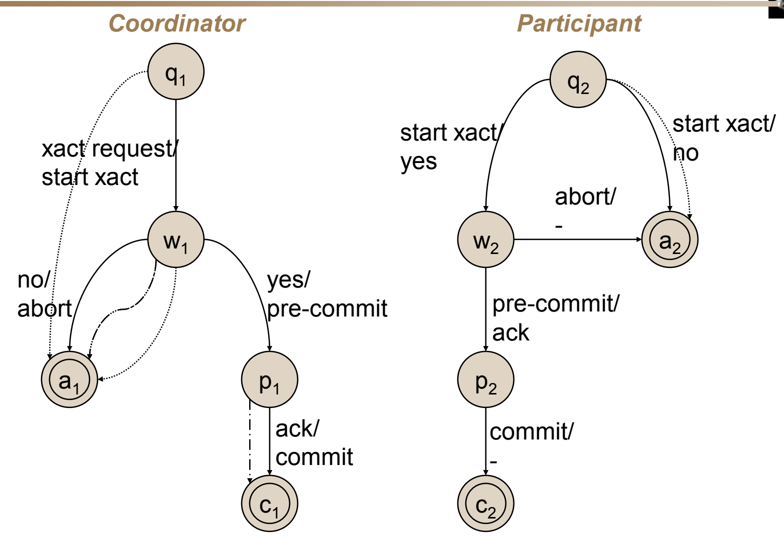
19

Atomic Commit Protocol: Requirements

- AC1: All processes that reach a decision reach the same one.
- AC2: A process cannot reverse its decision after it has reached one.
- AC3: The *Commit* decision can only be reached if *all* processes voted *Yes*.
- AC4: If there are no failures and all processes voted *yes*, then the decision will be to commit.
- AC5: Consider any execution containing only failures that the ACP is designed to tolerate. At any point in this execution, if all existing failures are repaired and no new failures occur for sufficiently long, then all processes will eventually reach a decision.

22

3PC assuming timeout on *receipt* of message



Replication Approaches: Consistency

- Write All approach
  - Reads can be satisfied by any copy in the system,
  - Writes must all modify *every* copy of the data item being written.
  - Eliminates the problem of multiple copies, and gives each txn the correct view.
  - It is very poor in terms of performance and progress:
    - Failures have a crippling effect on transactions!
- Write-All-Available
  - Challenge - recovery

30

1 Copy Serializability

- The correctness definition for replicated databases is therefore that it should behave as though all transactions are executed in a ***serial manner on a single copy database***.
- This is the notion of one copy serializability, I.e. 1SR.
- The user must be given a one copy view of the database.
- How is this achieved?
- Read-only is easy. For writes we must manage carefully!

32

Distributed Design Issues

- Why fragment?
- How to fragment?
- How much to fragment?
- How to test correctness?
- How to allocate?
- Information requirements?

34

Correctness of fragmentation

- Completeness
  - Decomposition of Relation R into $R_1, R_2, \ldots R_n$ is complete if and only if each data item in R can also be found in some R_i
- Reconstruction
  - If Relation R is decomposed into $R_1, R_2, \ldots R_n$, then there should exist some operator, that R can be reconstructed from $R_1, \ldots R_n$.
- Disjointness
  - If Relation R is decomposed into $R_1, R_2, \ldots R_n$, and data item d is in R_j, then d should not be in any other fragment R_k, $k <> j$.

35

PHF-Information Requirements

- Application Information
  - Simple predicates: Given $R[A_1, A_2, \ldots, A_n]$, a simple predicate p_j is:
    - P_j: $A_i\ \theta\ Value$
    - where θ is a comparison operator, Value is from the domain of attribute A_i
  - Minterm predicates: Given R and $P_r=\{p_1,p_2, \ldots p_m\}$, define $M=\{m_1, m_2, \ldots, m_z\}$ as

$$M = \{m_i \mid m_i = \wedge_{pj \in \Pr} p_j^*\}, 1 \le i \le z$$

where $p_j^* = p_j$ or $NOT(p_j)$.

36

Primary Horizontal Frag.

- Definition:

$$R_j = \sigma_{Fj}(R), 1 \leq j \leq w$$

  - Where F_j is a selection formula, which is (preferably) a minterm predicate.
- Therefore,
  - A horizontal fragment, R_i of relation R consists of all the tuples of R which satisfy a minterm predicate m_i ➔
  - Given a minterm of predicates M, there are as many horizontal fragments of relation R as there are minterm predicates
  - Set of horizontal fragments also referred to as minterm fragments.

37

PHF - Algorithm

- GIVEN: A relation R, the set of simple predicates P_r
- OUTPUT: The set of fragments of $R = \{R_1, \ldots, R_w\}$ which obey the fragmentation rules.
- Preliminaries:
  - P_r should be complete
  - P_r should be minimal

38

PHF - Example

- Two candidate relations: PAY and PROJ.
- Fragmentation of relation PAY
  - Application: check the salary info and determine raise.
  - Employee records kept at two sites ➔ application run at two sites
  - Simple predicates
    - p_1 : SAL <= 30000
    - p_2 : SAL > 30000
    - $P_r = \{p_1, p_2\}$ which is complete and minimal $P_r' = P_r$
  - Minterm predicates
    - m_1 : *(SAL <= 30000)*
    - m_2 : *NOT(SAL <= 30000) = (SAL>30000)*

39

Fragmentation of PROJ

- Applications:
  - Find the name and budget of projects given their loc.– issued at three sites
  - Access project information according to budget
    - One site accesses <=200000 another accesses > 200000
- Simple Predicates
  - For application 1:
    - p_1 : *LOC = "Montreal"*
    - p_2 : *LOC = "New York"*
    - p_3 : *LOC = "Paris"*
  - For application 2:
    - P_4: *BUDGET <= 200000*
    - P_5: *BUDGET > 200000*
  - $P_r = P_r' = \{p_1, p_2, p_3, p_4, p_5\}$

40

PHF Example

- Fragmentation of PROJ contd:
  - Minterm fragments left after elimination
  - *m_1: (LOC = "Montreal") AND (BUDGET <=200000)*
  - *m_2: (LOC = "Montreal") AND (BUDGET>200000)*
  - *m_3: (LOC = "New York") AND (BUDGET <=200000)*
  - *m_4: (LOC = "New York") AND (BUDGET >200000)*
  - *m_5: (LOC = "Paris") AND (BUDGET <=200000)*
  - *m_6: (LOC = "Paris") AND (BUDGET >200000)*

41

PHF -- Example

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instr.	150000	Montreal

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York

$PROJ_4$

PNO	PNAME	BUDGET	LOC
P3	CAD/CA M	250000	New York

$PROJ_6$

PNO	PNAME	BUDGET	LOC
P4	Maint.	310000	Paris

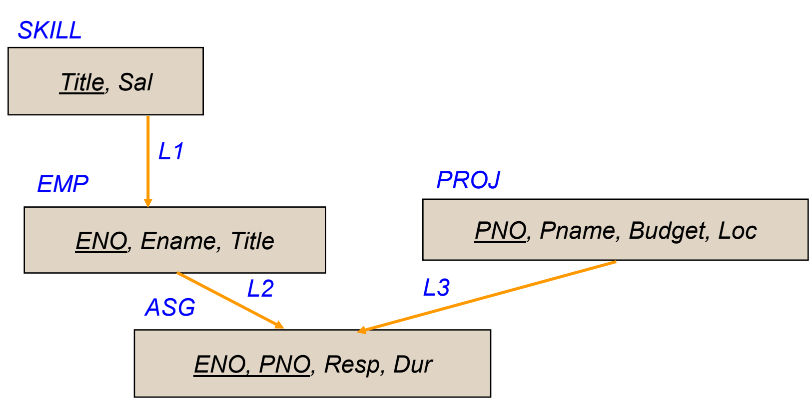
42

Derived Horizontal Fragmentation

- Defined on a member relation of a link according to a selection operation specified on its owner.
  - Each link is an equijoin
  - Equijoin can be implemented by means of semijoins.

43

Derived Horizontal Fragmentation



44

VF – Information Requirements

- Application Information
  - Attribute affinities
    - A measure that indicates how closely related the attributes are
    - This is obtained from more primitive usage data
  - Attribute usage values
    - Given a set of queries $Q=\{q_1, q_2, \ldots, q_k\}$ that will run on the relation $R[A_1, A_2, \ldots, A_n]$,
    - Use(q_i, A_j) = 1 if A_j is referenced by qi, 0 otherwise
    - Use$(qi,.)$ can be defined accordingly

45

VF – Affinity Measure aff(Ai,Aj)

- The attribute affinity measure between two attributes Ai and Aj of a relation R with respect to the set of applications Q={q1, q2, …, qk} is defined as follows:

$$aff(A_i, A_j) = \sum_{\text{all queries that access Ai and Aj}} (\text{query access})$$

$$\text{query access} = \sum_{allsites} \text{access freq of a query} * \frac{\text{access}}{\text{execution}}$$

46

Bond Energy Algorithm

- Input: the AA matrix
- Output: the clustered affinity matrix CA (a perturbation of AA)

1. Initialization: Place and fix one of the columns of AA in CA
2. Iteration: Place the remaining n-I columns in the remaining I+1 positions in the CA matrix. For each column, chose the placement that makes the most contribution to the global affinity measure.
3. Row Order: Order the rows according to the columns.

Selecting Alternatives

SELECT ENAME
FROM EMP, ASG
WHERE EMP.ENO = ASG.ENO
AND DUR > 37.

Strategy 1:

$$\prod_{ENAME}(\sigma_{EMP.ENO=ASG.ENO \wedge DUR>37)}(EMP \times ASG))$$

Strategy 2:

$$\prod_{ENAME}(EMP \rhd\lhd_{ENO} (\sigma_{DUR>37)}(ASG)))$$

Strategy 2, avoids cartesian product.

Problem

Site1

$ASG_1 = \sigma_{ENO \leq "E3"}(ASG)$

Site2

$ASG2 = \sigma_{ENO > "E3"}(ASG)$

Site3

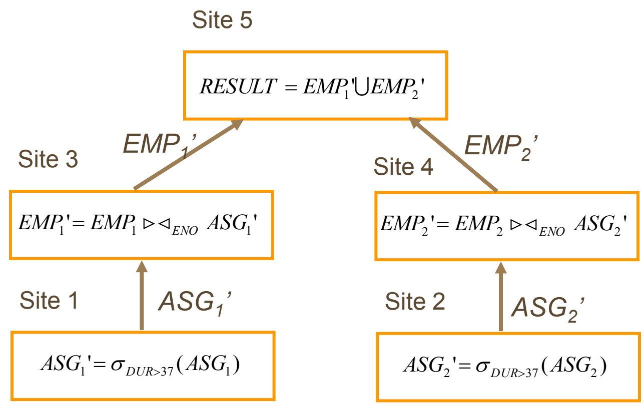
$EMP_1 = \sigma_{ENO \leq "E3"}(EMP)$

Site4

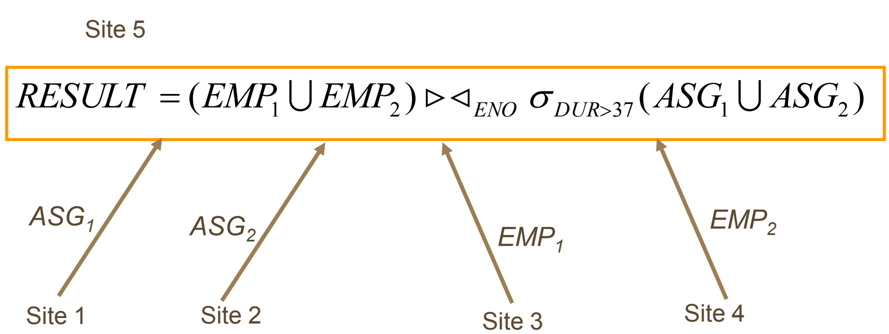
$EMP_2 = \sigma_{ENO > "E3"}(EMP)$

Site5

$RESULT$



Alternative 2



Cost of Alternatives

- Assume
  - Size(EMP) = 400; size(ASG)=1000
  - Tuple access cost (TAC) = 1unit; tuple xfer cost (TXC) =10units
- Strategy 1
  - Produce ASG': (10+10)*TAC = 20
  - Transfer ASG': (10+10)*TXC = 200
  - Produce EMP': (10+10)*TAC*2 = 40
  - Transfer EMP' to result site: (10+10)*TXC = 200
  - Total COST = 460.

Cost of alternatives (cont)

- Strategy 2
  - Transfer EMP to site 5: 400*TXC = 4000
  - Transfer ASG to site 5: 1000*TXC = 10,000
  - Produce ASG': 1000*TAC = 1,000
  - Join EMP and ASG': 400*20*TAC = 8,000

  - TOTAL COST = 23,000!!

Query Optimization Objectives

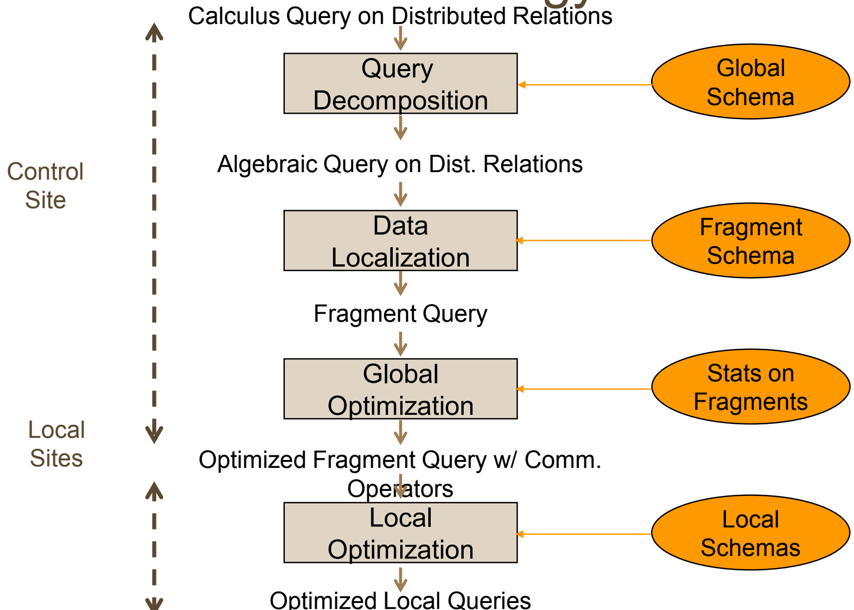
- Minimize a cost function
  - I/O cost + CPU cost + communication cost
- These may have different weights in different distributed environments
- Wide area networks
  - Communication cost will dominate
    - Low bandwidth
    - Low speed
    - High protocol overhead
  - Most algorithms ignore all other cost components

Complexity of Relational Operators

Assume
Relations of cardinality *n*
Sequential scan

Operation	**Complexity**
Select, Project (without duplicate elimination)	$O(n)$
Project (w/ duplicate elimination) **Group**	$O(n \log n)$
Join **Semijoin** **Division** **Set Operators**	$O(n \log n)$
Cartesian Product	$O(n^2)$

Methodology



(c)Oszu & Valduriez 55

Data Localization

- Assume
  - EMP is fragmented as

$$EMP_1 = \sigma_{ENO \le "E3"}(EMP)$$

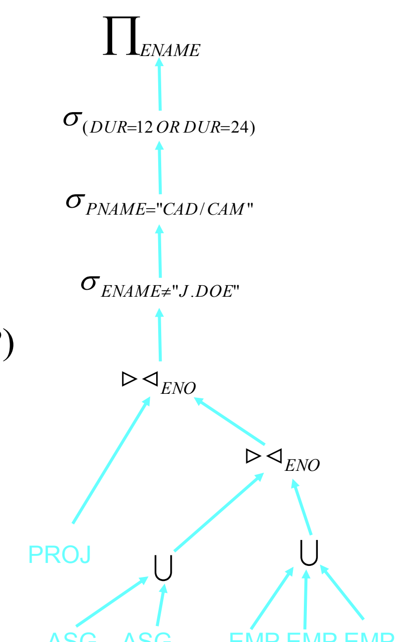
$$EMP_2 = \sigma_{ENO > "E3" \wedge ENO \le "E6"}(EMP)$$

$$EMP_3 = \sigma_{ENO > "E6"}(EMP)$$

  - ASG is fragmented as

$$ASG_1 = \sigma_{ENO \le "E3"}(ASG)$$

$$ASG_2 = \sigma_{ENO > "E3"}(ASG)$$



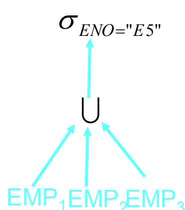
(c)Oszu & Valduriez 56

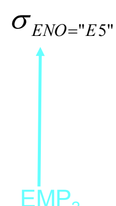
Reduction for PHF

- Reduction with selection
  - Relation R and F_R={R1, …, Rw}, where $R_j = \sigma_{pj}(R)$

$$\sigma_{pj}(R_i) = \phi \quad \text{if } \forall x \text{ in } R: \neg(p_i(x) \wedge p_j(x))$$

  - Example:
    - SELECT * FROM EMP WHERE ENO="E5"





Reduction for PHF

- Reduction with join
  - Possible if fragmentation is done on join attribute
  - Distribute join over unions

$$(R_1 \cup R_2) \bowtie S \Leftrightarrow (R_1 \bowtie S) \cup (R_2 \bowtie S)$$

  - Given $R_i = \sigma_{pi}(R)$ and $R_j = \sigma_{pj}(R)$

$$R_i \bowtie R_j = \phi \quad \text{if } \forall x \text{ in } R_i\ \forall y \text{ in } R_j: \neg(p_i(x) \wedge p_j(y))$$

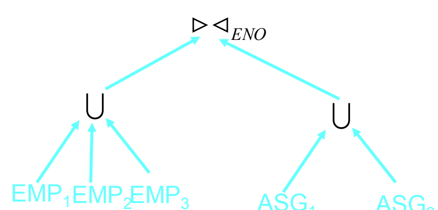
Reduction for PHF

- Reduction with join -- Example
  - Assume EMP fragmented as before, and

$$ASG_1 = \sigma_{ENO \leq "E3"}(ASG)$$

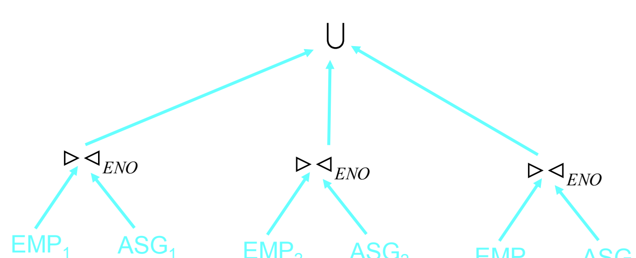
$$ASG_2 = \sigma_{ENO > "E3"}(ASG)$$

  - Example:
    - SELECT * FROM EMP,ASG WHERE EMP.ENO=ASG.ENO



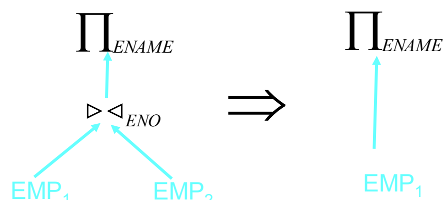
Reduction for PHF

- Reduction with join -- Example
  - Distribute join over unions
  - Apply the reduction rule



Reduction for VF

- Find useless (not empty) intermediate relations
  - Relation R defined over attributes A={A1, …, An} vertically fragmented as $R_i = \Pi_{A'}(R)$ where A' is a subset of A
  - $P_{D,K}(R_i)$ is useless if D is not in A'
  - Example $EMP_1 = \Pi_{ENO,ENAME}(EMP)$, $EMP_2 = \Pi_{ENO,TITLE}(EMP)$
  - SELECT ENAME FROM EMP



Reduction for DHF

- Rule:
  - Distribute join over unions
  - Apply the join reduction for horizontal fragmentation
  - Example

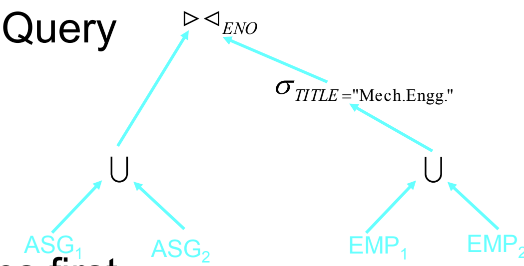
$$ASG_1 = ASG \triangleright\!< _{ENO} (EMP_1)$$
$$ASG_2 = ASG \triangleright\!< _{ENO} (EMP_2)$$
$$EMP_1 = \sigma_{TITLE = \text{"Programmer"}}(EMP)$$
$$EMP_2 = \sigma_{TITLE \neq \text{"Programmer"}}(EMP)$$

  - Query:

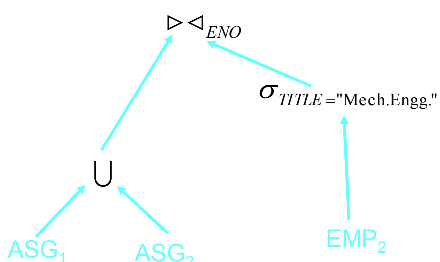
SELECT	*
FROM	EMP, ASG
WHERE	ASG.ENO=EMP.ENO
AND	EMP.TITLE="Mech. Engg"

Reduction for DHF

- Generic Query



- Selections first



Step 3 – Global Optimization

- Input: Fragment query
- Find the best (not necessarily optimal) global schedule
  - Minimize a cost function
  - Distributed join processing
    - Bushy vs. linear trees
    - Which relation to ship where?
    - Ship-whole vs. ship-as-needed
  - Decide on use of semijoins
  - Join methods
    - Nested loop vs. ordered joins (merge join or hash join)

Semijoin Algorithms

- Perform the join
  - Send R to site 2
  - Site 2 computes the join
- Consider semijoin $R \rhd\lhd_A S \Leftrightarrow (R \rhd<_A S) \rhd\lhd_A S$
  - $S' \leftarrow \prod_A(S)$
  - S' → Site 1
  - Site 1 computes $R' = R \rhd<_A S'$
  - R' → Site 2
  - Site 2 computes $R' \rhd\lhd_A S$

Semijoin is better if

$$size(\prod_A(S)) + size(R \rhd<_A S)) < size(R)$$

R* Algorithm

- Performing Joins
- Ship Whole
  - Larger data transfer
  - Smaller number of messages
  - Better if relation are small
- Fetch as needed
  - Number of message – O(card of external relation)
  - Data transfer per message is minimal
  - Better if relations are large and selectivity is good.