Chapter 19: Distributed Databases

Database System Concepts, 6th Ed.

Chapter 19: Distributed Databases

- Heterogeneous and Homogeneous Databases
- Distributed Data Storage
- Distributed Transactions
- Commit Protocols
- Concurrency Control in Distributed Databases
- Availability
- Distributed Query Processing
- Heterogeneous Distributed Databases
- Directory Systems

Distributed Database System

- A distributed database system consists of loosely coupled sites that share no physical component
- Database systems that run on each site are independent of each other
- Transactions may access data at one or more sites

Homogeneous Distributed Databases

- In a homogeneous distributed database
  - All sites have identical software
  - Are aware of each other and agree to cooperate in processing user requests.
  - Each site surrenders part of its autonomy in terms of right to change schemas or software
  - Appears to user as a single system
- In a heterogeneous distributed database
  - Different sites may use different schemas and software
    - Difference in schema is a major problem for query processing
    - Difference in software is a major problem for transaction processing
  - Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing

Distributed Data Storage

- Assume relational data model
- Replication (Αντιγραφή)
  - System maintains multiple copies of data, stored in different sites, for faster retrieval and fault tolerance.
- Fragmentation (Κατακερματισμός)
  - Relation is partitioned into several fragments stored in distinct sites
- Replication and fragmentation can be combined
  - Relation is partitioned into several fragments (τμήματα): system maintains several identical replicas of each such fragment.

Data Replication

- A relation or fragment of a relation is **replicated** if it is stored redundantly in two or more sites.
- Full replication(πλήρης αντιγραφή) of a relation is the case where the relation is stored at all sites.
- Fully redundant databases are those in which every site contains a copy of the entire database.

Data Replication (Cont.)

- Advantages of Replication
  - **Availability (Διαθεσιμότητα)**: failure of site containing relation *r* does not result in unavailability of *r* if replicas exist.
  - **Parallelism (Παραλληλισμός)**: queries on *r* may be processed by several nodes in parallel.
  - **Reduced data transfer (Μειωμένος χρόνος μεταφοράς)**: relation *r* is available locally at each site containing a replica of *r*.
- Disadvantages of Replication
  - Increased cost of updates: each replica of relation *r* must be updated.
  - Increased **complexity of concurrency control**: concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented.
    - One solution: choose one copy as **primary copy (πρωτεύον αντίγραφο)** and apply concurrency control operations on primary copy

Data Fragmentation (Κατακερματισμός Δεδομένων)

- Division of relation r into fragments r_1, r_2, …, r_n which contain sufficient information to reconstruct relation r.
- **Horizontal fragmentation (Οριζόντιος Κατακερματισμός)**: each tuple of *r* is assigned to one or more fragments
- **Vertical fragmentation (Κατακόρυφος Κατακερματισμός)** : the schema for relation *r* is split into several smaller schemas
  - All schemas must contain a **common candidate key υποψήφιο κλειδί** (or superkey υπερ-κλειδί) to ensure lossless join property.
  - A special attribute, the **tuple-id attribute (κωδικός εγγραφής)** may be added to each schema to serve as a candidate key.

Horizontal Fragmentation of *account* Relation

branch_name	*account_number*	*balance*
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

$$account_1 = \sigma_{branch\_name=\text{“Hillside”}}(account)$$

branch_name	*account_number*	*balance*
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

$$account_2 = \sigma_{branch\_name=\text{“Valleyview”}}(account)$$

Vertical Fragmentation of *employee_info* Relation

branch_name	*customer_name*	*tuple_id*
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

$deposit_1 = \Pi_{branch\_name,\ customer\_name,\ tuple\_id}\ (employee\_info\ )$

account_number	*balance*	*tuple_id*
A-305	500	1
A-226	336	2
A-177	205	3
A-402	10000	4
A-155	62	5
A-408	1123	6
A-639	750	7

$deposit_2 = \Pi_{account\_number,\ balance,\ tuple\_id}\ (employee\_info\ )$

Advantages of Fragmentation

- Horizontal:
  - allows parallel processing on **fragments of a relation**
  - allows a relation to be split so that tuples are located where they are most frequently accessed
- Vertical:
  - allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed
  - tuple-id attribute allows efficient joining of vertical fragments
  - allows parallel processing on a **relation**
- Vertical and horizontal fragmentation can be mixed.
  - Fragments may be successively fragmented to an arbitrary depth.

Data Transparency (Διαφάνεια Δεδομένων)

- **Data transparency**: Degree to which system user may remain unaware of the details of how and where the data items are stored in a distributed system
- Consider transparency issues in relation to:
  - Fragmentation transparency (Διαφάνεια κατακερματισμού)
  - Replication transparency (Διαφάνεια αντιγραφής)
  - Location transparency (Διαφάνεια θέσης)

Naming of Data Items – Criteria
Κριτήρια Ονομασίας Στοιχείων Δεδομένων

1. Every **data item** must have a **system-wide unique name**.
2. It should be possible to find the location of data items efficiently.
3. It should be possible to change the location of data items transparently.
4. Each site should be able to create new data items autonomously.

Centralized Scheme - Name Server
Κεντρικός Διακομιστής Ονομάτων

- Structure:
  - name server assigns all names
  - each site maintains a record of local data items
  - sites ask name server to locate non-local data items
- Advantages:
  - satisfies naming criteria 1-3
- Disadvantages:
  - does not satisfy naming criterion 4
  - name server is a potential performance bottleneck
  - name server is a single point of failure

Use of Aliases
Χρήση Ψευδονύμων

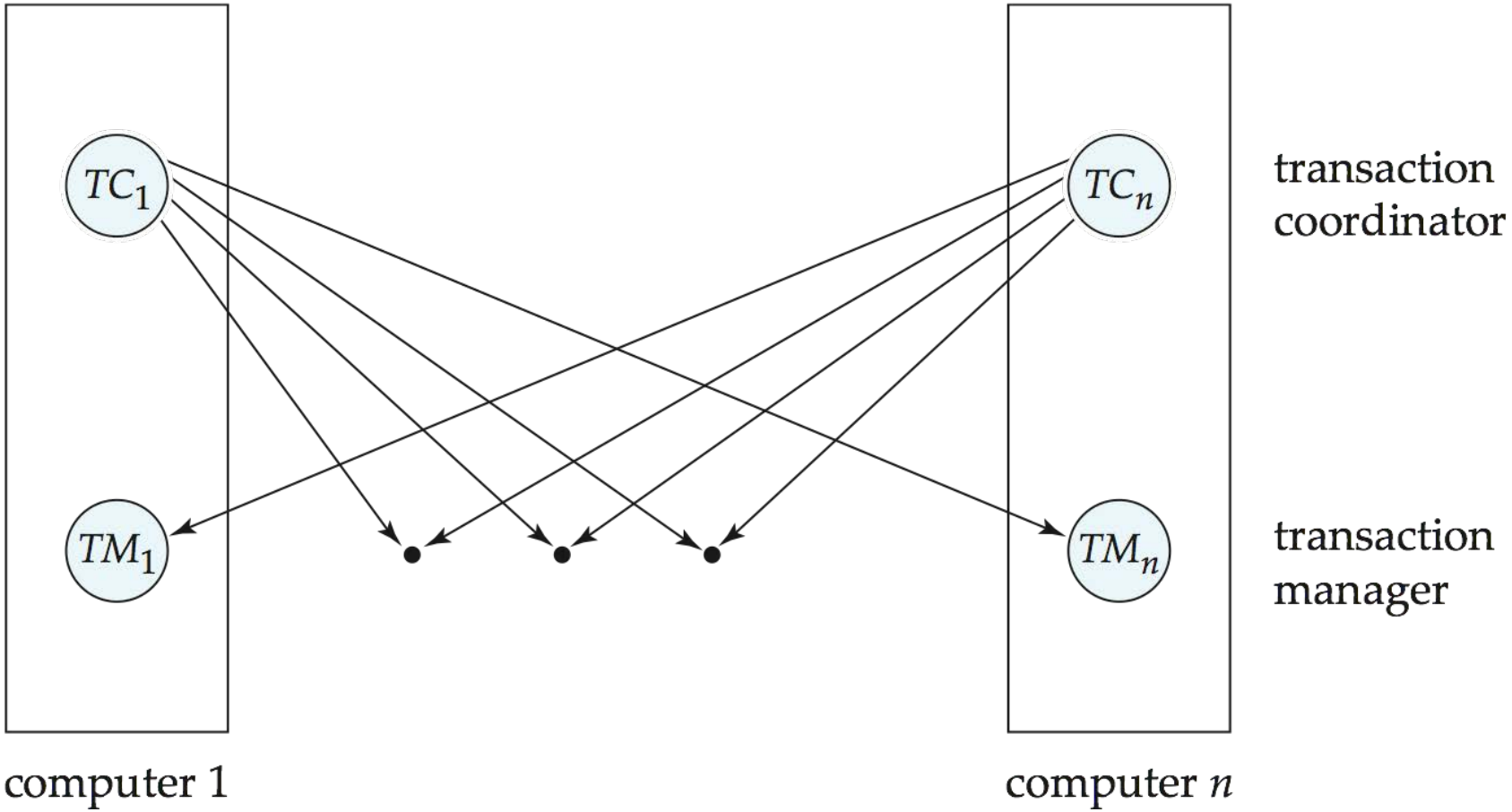
- Alternative to centralized scheme: each site prefixes its own site identifier to any name that it generates i.e., *site* 17.a*ccount.*
  - Fulfills having a unique identifier, and avoids problems associated with central control.
  - However, fails to achieve network transparency.
- Solution: Create a set of **aliases** for data items; Store the mapping of aliases to the real names at each site (θέση).
- The user can be unaware of the physical location of a data item, and is unaffected if the data item is moved from one site to another.

Distributed Transactions and 2 Phase Commit (Κατανεμημένες Συναλλαγές & Πρωτόκολλο Ολοκλήρωσης 2 Φάσεων)

Distributed Transactions (Κατανεμημένες Συναλλαγές)

- Transaction may access data at several sites (Θέσεις).
- Each site has a local **transaction manager (διαχειριστής συναλλαγών)** responsible for:
  - Maintaining a log for recovery purposes
  - Participating in coordinating the concurrent execution of the transactions **executing at that site**.
- Each site has a **transaction coordinator (συντονιστής συναλλαγών)**, which is responsible for:
  - Starting the execution of transactions that **originate at the site**.
  - Distributing **subtransactions at appropriate sites for execution**.
  - Coordinating the termination of each transaction that originates at the site, which may result in the transaction being committed at all sites or aborted at all sites.

Transaction System Architecture



System Failure Modes (Καταστάσεις Αποτυχίας του Συστήματος)

- Failures unique to distributed systems:
  - Failure of a site.
  - Loss of messages
    - Handled by network transmission control protocols such as TCP-IP
  - Failure of a communication link
    - Handled by network protocols, by routing messages via alternative links
  - **Network partition (τμηματοποίηση δικτύου)**
    - A network is said to be **partitioned** when it has been split into two or more subsystems that lack any connection between them
      - Note: a subsystem may consist of a single node
- Network partitioning and site failures are generally indistinguishable.

Commit Protocols
Πρωτόκολλα Εκτέλεσης

- Commit protocols are used to ensure atomicity (ατομικότητα) across sites
  - a transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.
  - not acceptable to have a transaction committed at one site and aborted at another
- The ***two-phase commit*** **(2PC) (πρωτόκολλο ολοκλήρωσης 2 φάσεων)** protocol is widely used
- The ***three-phase commit*** **(3PC) (πρωτόκολλο ολοκλήρωσης 3 φάσεων)** protocol is more complicated and more expensive, but avoids some drawbacks of two-phase commit protocol. **This protocol is not used in practice.**

Two Phase Commit Protocol (2PC) (Πρωτόκολλο Ολοκλήρωσης 2 Φάσεων)

The two phase commit protocol is a distributed algorithm which lets all sites in a distributed system agree to commit a transaction. The protocol results in either all nodes committing the transaction or aborting, even in the case of site failures and message losses.

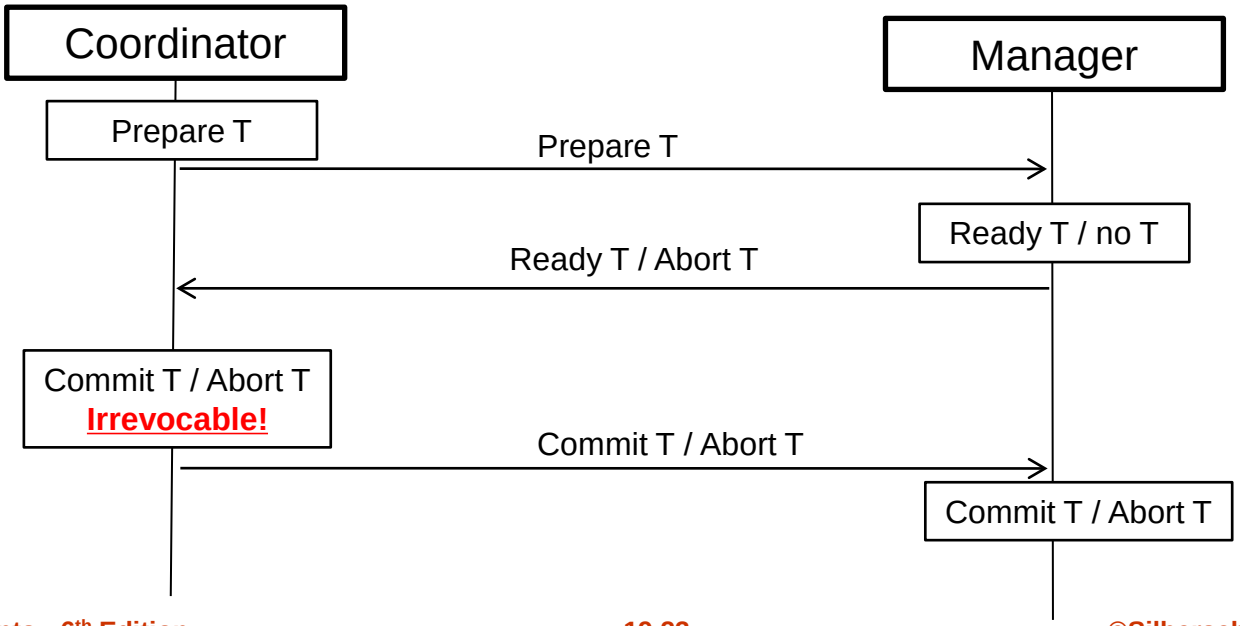
- Assumes **fail-stop** model – failed sites simply stop working, and do not cause any other harm, such as sending incorrect messages to other sites.
- The protocol assumes that there is **stable storage at each node** with a **write-ahead log**, that **no node crashes forever**, that the **data in the write-ahead log is never lost or corrupted in a crash**, and that any two nodes can communicate with each other.
- The protocol involves **all the local sites at which the transaction executed**
- Execution of the protocol is initiated by the **coordinator (συντονιστή) after the last step of the transaction has been reached**.
- The participants then respond with an **agreement message or an abort message** depending on whether the transaction has been processed successfully at the participant.

Phase 1: Obtaining a Decision (Λήψη Απόφασης)

- Let ***T*** be a transaction initiated at site S_i, and let the transaction coordinator at S_i be C_i
- **Coordinator (συντονιστής συναλλαγών)** C_i asks all participants to *prepare* to commit transaction T_i.
  - C_i adds the records (προσθέτει την εγγραφή) <**prepare** *T*> to the **log (αρχείο καταγραφής)** and forces log to **stable storage (σταθερή μνήμη)** .
  - sends **prepare** *T* messages to all sites at which *T* executed
- Upon receiving message, **transaction manager (διαχειριστής συναλλαγών)** at site determines if it can commit the transaction
  - if the transaction **can not** be committed,
    - add a record <**no** *T*> to the log and
    - send **abort** *T* message **to** C_i
  - if the transaction **can** be committed, then:
    - add the record <**ready** *T*> to the log
    - force *all records* for *T* to stable storage
    - send **ready** *T* message **to** C_i

Phase 2: Recording the Decision (Καταγραφή Απόφασης)

- *T* can be committed if C_i received a **ready** *T* message from **all the participating sites**: otherwise *T* must be aborted.
- Coordinator adds a **decision record**, <**commit** *T*> or <a**bort** *T*>, to the log and **forces record onto stable storage**. Once the record stable storage is written **it is irrevocable (even if failures occur)**
- Coordinator **sends a message to each participant** informing it of the decision (**commit** or **abort**)
- Participants take appropriate action locally.



Handling of Failures - Site Failure
Χειρισμός Προβλημάτων – Πρόβλημα σε συμμετέχουσα θέση

When **site S_k** recovers, it examines its log to determine the fate of transactions active at the time of the failure.

- Log contain <**commit** *T*> record: txn had completed, nothing to be done
- Log contains <**abort** *T*> record: txn had not completed, nothing to be done
- Log contains <**ready** *T*> record: site must **consult C_i** to determine the fate of *T*.
  - If *T* committed, **redo** (*T*); write <**commit** *T*> record
  - If *T* aborted, **undo** (*T*)
- The log contains **no control log records *abort, commit, ready* concerning *T* (*eg S_k has received a prepare T)***:
  - Implies that S_k failed before responding to the **prepare** *T* message from C_i
  - since the failure of S_k precludes the sending of a <**ready** *T*> response, coordinator C_i must abort *T*
  - *So* S_k must execute **undo** (*T*)

Handling of Failures- Coordinator Failure
Χειρισμός Προβλημάτων – Πρόβλημα Συντονιστή

- If coordinator fails while the commit protocol for *T* is executing then participating sites must decide on *T*'s fate:
  1. If an active site contains a <**commit** *T*> record in its log, then *T* must be committed.
  2. If an active site contains an <**abort** *T*> record in its log, then *T* must be aborted.
  3. If some active participating site **does not contain a** <**ready** *T*> record in its log (so it contains a <**no** *T*>) , then the failed coordinator C_i cannot have decided to commit *T*.
     - Can therefore **abort** ***T****;* however, such a site must reject any subsequent <**prepare** *T*> message from C_i
  4. If none of the above cases holds, then all active sites must have a <**ready** *T*> record in their logs, but no additional control records (such as <**abort** *T*> of <**commit** *T*>).
     - In this case active sites must wait for C_i to recover, to find decision.
- **Blocking problem (πρόβλημα μπλοκαρίσματος)**: active sites may have to wait for failed coordinator to recover.

Handling of Failures - Network Partition
Τμηματοποίηση Δικτύου

- If the coordinator and all its participants remain in one partition, the failure has no effect on the commit protocol.
- If the coordinator and its participants belong to several partitions:
  - Sites that **are not in the partition containing the coordinator** think the coordinator has failed, and execute the protocol to deal with failure of the coordinator.
    - No harm results, but sites may still have to wait for decision from coordinator.
  - The **coordinator and the sites in the same partition as the coordinator** think that the sites in the other partition have failed, and follow the usual commit protocol.
    - Again, no harm results

Recovery and Concurrency Control
Αποκατάσταση & Έλεγχος Συγχρονικότητας

- When site S_k recovers
- **In-doubt transactions (αμφίβολες συναλλαγές)** have a <**ready** *T*>, but neither a <**commit** *T*>, nor an <**abort** *T*> log record.
- The recovering site must **determine the commit-abort status** of such transactions **by contacting other sites**; this can slow and potentially block recovery.
- Recovery algorithms can note lock information in the log.
  - Instead of <**ready** *T*>, write out <**ready** *T*, *L*> *L* = list of locks held by *T* when the log is written (read locks can be omitted).
  - For every in-doubt transaction *T*, all the locks noted in the <**ready** *T*, *L*> log record are reacquired.
- After lock reacquisition, transaction processing can resume; the **commit or rollback** of **in-doubt transactions** is performed concurrently with the execution of new transactions.

Distributed Query Processing Example

Natural join (⋈)

Natural join (⋈) is a binary operator that is written as ($R \bowtie S$) where R and S are relations.[1] The result of the natural join is the set of all combinations of tuples in R and S that are equal on their common attribute names. For an example consider the tables *Employee* and *Dept* and their natural join:

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Employee ⋈ Dept

Name	EmpId	DeptName	Manager
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

Semijoin ($\ltimes$)($\rtimes$)

The semijoin is joining similar to the natural join and written as $R \ltimes S$ where R and S are relations.[2] The result of the semijoin is only the set of all tuples in R for which there is a tuple in S that is equal on their common attribute names. For an example consider the tables *Employee* and *Dept* and their semi join:

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Production

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee $\ltimes$ ***Dept***

Name	EmpId	DeptName
Sally	2241	Sales
Harriet	2202	Production

ΙΣΟΔΥΝΑΜΙΕΣ ΣΕ JOINS:

$$R \triangleright\triangleleft_{A=A} S \Leftrightarrow (R \triangleright<_{A=A} S) \triangleright\triangleleft_{A=A} S$$

$$\Leftrightarrow (R \triangleright<_{A=A} \Pi_A(S)) \triangleright\triangleleft_{A=A} S$$

$$\Leftrightarrow R \triangleright\triangleleft_{A=A} (S \triangleright<_{A=A} R)$$

$$\Leftrightarrow (R \triangleright<_{A=A} S) \triangleright\triangleleft_{A=A} (S \triangleright<_{A=A} R)$$

Παράδειγμα

Έστω οι σχέσεις

```
S (S#, SNAME)
SP (S#, P#, QTY)
P (P#, PNAME)
```

που βρίσκονται αποθηκευμένες στους κόμβους T1, T2 και T3 αντίστοιχα. Ας υποθέσουμε ότι γνωρίζουμε τα παρακάτω στοιχεία για τις σχέσεις αυτές:

S: έχει 6 εγγραφές με κλειδιά s1, s2, s3, s4, s5, s6.
SP: έχει 8 εγγραφές με κλειδιά (s1,p1), (s1,p2), (s1,p3), (s2,p1), (s2,p2), (s2,p3), (s3,p1), (s3,p3).
P: έχει 6 εγγραφές με κλειδιά p1, p2, p3, p4, p5, p6.

Επιπλέον, γνωρίζουμε ότι τα μεγέθη των πεδίων είναι:

Κλειδί	Μέγεθος (bytes)
S#	4
P#	4
QTY	10
SNAME	96
PNAME	196

Έστω, τώρα, ότι μια τοποθεσία T4 ενεργοποιεί την ερώτηση

$$S \bowtie SP \bowtie P$$

Έστω ότι το κόστος μεταφοράς = μέγεθος των μεταφερόμενων δεδομένων (M).

Το κόστος χωρίς λειτουργίες semi-join.

- Η σχέση S (S#, SNAME) έχει 6 εγγραφές και καθεμία από αυτές έχει μέγεθος 4+96=100 bytes. Άρα, το μέγεθος της S είναι 6·100 bytes=600 bytes.
- Η σχέση SP (S#, P#, QTY) έχει 8 εγγραφές και καθεμία από αυτές έχει μέγεθος 4+4+10=18 bytes. Άρα, το μέγεθος της SP είναι 8·18 bytes=144 bytes.
- Η σχέση P (P#, PNAME) έχει 6 εγγραφές και καθεμία από αυτές έχει μέγεθος 4+196=200 bytes. Άρα, το μέγεθος της P είναι 6·200 bytes=1200 bytes.

Το αποτέλεσμα της ερώτησης S JOIN SP JOIN P, είναι μία σχέση R, η οποία αποτελείται από 8 εγγραφές και τα πεδία κάθε εγγραφής είναι τα παρακάτω: R(S#, SNAME, P#, PNAME, QTY). Το μέγεθος κάθε εγγραφής είναι 4+96+4+196+10=310 bytes. Συνεπώς, το μέγεθος της σχέσης R θα είναι 8*310 bytes=2480 bytes.

Αν η ερώτηση S ⋈ SP ⋈ P εκτελεστεί σε οποιονδήποτε άλλο κόμβο εκτός από τον κόμβο T4, αυτό σημαίνει ότι θα πρέπει να μεταφέρουμε το αποτέλεσμα, (το οποίο στη συνέχεια θα ονομάζουμε R) στον κόμβο T4 όπου έχει ενεργοποιηθεί η ερώτηση. Το κόστος μεταφοράς δεδομένων της λύσης R, είναι ίσο με το μέγεθος των μεταφερόμενων δεδομένων, δηλ. είναι C_R= *2480 bytes*.

Αν η ερώτηση εκτελεστεί στον κόμβο T1 ή στον κόμβο T2 ή στον κόμβο T3, το κόστος της ερώτησης θα είναι ίσο με το άθροισμα των διαφόρων μεταφερομένων δεδομένων από τους άλλους δύο κόμβους στον κόμβο που εκτελείται η ερώτηση, προσαυξημένο κατά 2480 bytes, που είναι το κόστος μεταφοράς της λύσης.

Αν μεταφέρουμε όλες τις σχέσεις S, SP και P στον κόμβο T4 και εκτελέσουμε εκεί την ερώτηση, έχουμε κόστος = μεταφορά των σχέσεων S, SP και P στον κόμβο T4, είναι 600+144+1200=*1944 bytes*.

Εύρεση κόστους αν υποθέσουμε ότι κάνουμε S ⋉ SP

Η ερώτηση θα εκτελεστεί ως εξής:

- προβάλλουμε το S# από την SP στον κόμβο T2 και θα το στείλουμε στον T1.
- εκτελουμε το semi-join το οποίο θα δημιουργήσει το αποτέλεσμα S'.
- Στέλνουμε στον κόμβο T4, τα S', SP, P και εκτελουμε την ένωση.

Το SP πρέπει να το στείλουμε **πάλι** στον T4, άσχετα από το αν έχει ήδη συμμετάσχει στο semi-join στον κόμβο T1.

Αναλυτικά, έχουμε:

- Πρώτα θα γίνει στον κόμβο T2, προβολή των τιμών S# της σχέσης SP. Δηλαδή θα γίνει η πράξη: $F = \Pi_{S\#}(SP)$. Η σχέση F έχει 3 εγγραφές: s1, s2, s3. Αυτές οι τιμές μεταφέρονται στον κόμβο T1, όπου βρίσκεται η σχέση S. Δηλαδή, μεταφέρονται 3*4=12 bytes.

- Στον κόμβο T1, γίνεται το join της σχέσης S και των τιμών S# που ήρθανε από τον κόμβο T2, δηλαδή γίνεται η πράξη: S' = F ⋈ S. Η σχέση S' έχει μέγεθος 3*(4+96)=300 bytes.
- Στη συνέχεια, μεταφέρεται η σχέση S' από τον κόμβο T1 στον κόμβο T4, δηλαδή μεταφέρονται 300 bytes. Η αξία της κίνησης αυτής είναι ότι με το S ⋉ SP, αντί να μεταφερθεί από τον κόμβο T1 στον κόμβο T4 ολόκληρη η σχέση S, μεταφέρονται **μόνο** εκείνες οι εγγραφές που πραγματικά χρειάζονται και θα χρησιμοποιηθούν.
- Κατόπιν, μεταφέρεται η σχέση SP από τον κόμβο T2 στον κόμβο T4 (144 bytes) και επίσης μεταφέρεται η σχέση P από τον κόμβο T3 στον κόμβο T4 (1200 bytes).
- Τέλος, στον κόμβο T4, πραγματοποιείται η ερώτηση R = S' ⋈ SP ⋈ P. Το κόστος της ερώτησης αν χρησιμοποιηθεί το S ⋉ SP, είναι: 12+300+144+1200=*1656 bytes*.

Εύρεση του βέλτιστου πλάνου, με τη χρήση semi-joins

Παρατηρούμε ότι προβάλλοντας τα πεδία S# και P# της σχέσης SP παίρνουμε 3 τιμές τόσο για το κλειδί S#, όσο και για το P#. Από την άλλη μεριά, προβάλλοντας το πεδίο S# της S και το πεδίο P# της P παίρνουμε 6 τιμές και στις δύο περιπτώσεις. Δεδομένου ότι τα πεδία S# και P# αποτελούν τα πεδία σύνδεσης για τη σύνδεση των σχέσεων S, SP και P, και ότι είναι ίδιου μεγέθους, συμφέρει να προβάλλουμε τα πεδία S# και P#, της σχέσης SP.

Το καλύτερο πλάνο εκτέλεσης της ερώτησης είναι το εξής:

i) Προβάλλουμε το πεδίο S# της SP στον κόμβο T2 και μεταφέρουμε τη σχέση $F=\Pi_{S\#}(SP)$, δηλαδή C1=12 bytes, στον κόμβο T1.

ii) Προβάλλουμε το πεδίο P# της SP στον κόμβο T2 και μεταφέρουμε τη σχέση $G=\Pi_{P\#}(SP)$, δηλαδή C1'=12 bytes, στον κόμβο T3.

iii) Κάνουμε τη σύνδεση $F' = S \ \triangleright\!\triangleleft_{S\#=S\#} \ F$ στον κόμβο T1 και προκύπτει μια σχέση με μέγεθος : 3*100=300 bytes.

iv) Επίσης, κάνουμε τη σύνδεση $G' = P \ \triangleright\!\triangleleft_{P\#=P\#} \ G$ στον κόμβο T3 και προκύπτει μια σχέση με μέγεθος : 3*200=600 bytes.

v) Μεταφέρουμε τη σχέση SP από τον κόμβο T2 στον κόμβο T4, δηλαδή μεταφέρουμε C2=144 bytes.

vi) Μεταφέρουμε την F', δηλαδή C3=300 bytes, από τον κόμβο T2 και κάνουμε τη σύνδεση $F'' = F' \ \triangleright\!\triangleleft_{S\#=S\#} \ SP$, στον κόμβο T4.

vii) Επίσης, μεταφέρουμε την G', δηλαδή C3'=600 bytes, από τον κόμβο T3 και κάνουμε τη σύνδεση $F'' \ \triangleright\!\triangleleft_{P\#=P\#} \ G'$, για να πάρουμε την τελική σχέση R στον κόμβο T4.

Το συνολικό κόστος της ερώτησης είναι:

$C = C1 + C1' + C2 + C3 + C3' = 12 + 12 + 144 + 300 + 600 =$ *1068 bytes*