ΚΑΤΑΝΕΜΗΜΕΝΕΣ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

Π. ΒΑΣΙΛΕΙΑΔΗΣ

2.1 ΓΕΝΙΚΑ

Μια *κατανεμημένη βάση δεδομένων (distributed database)* μπορεί να οριστεί σαν μια ομάδα από λογικά συνδεόμενες βάσεις δεδομένων που είναι διεσπαρμένες σε ένα δίκτυο υπολογιστών. Ένα *κατανεμημένο σύστημα διαχείρισης βάσεων δεδομένων (distributed DBMS)* μπορεί να οριστεί σαν ένα πρόγραμμα που επιτρέπει τη διαχείριση μιας κατανεμημένης βάσης δεδομένων με τρόπο που να κάνει την κατανομή της διαφανή στους χρήστες [OV91].

Οι δύο αυτοί ορισμοί δίνουν μια πρώτη εικόνα για το τι μπορεί να είναι το αντικείμενο αυτού του κεφαλαίου. Ο προσεκτικός αναγνώστης θα διαπιστώσει ότι η έμφαση κρύβεται στις εκφράσεις "λογικά συνδεόμενες βάσεις δεδομένων" και "διαφανής κατανομή". Μια κατανεμημένη βάση δεδομένων δεν είναι ένα σύνολο τοπικών βάσεων που βρίσκονται τυχαία διεσπαρμένες σε ένα δίκτυο και επικοινωνούν μεταξύ τους. Υπάρχει αφενός μια λογική συνοχή των τοπικών βάσεων και αφετέρου ένας τρόπος επερώτησής τους χωρίς ο χρήστης/προγραμματιστής να πρέπει να γνωρίζει την εσωτερική δομή της κατανομής.

Η ανάγκη του τεμαχισμού μιας βάσης δεδομένων και της κατανομής της γεωγραφικά στο δίκτυο προέκυψαν ιστορικά από την ανάγκη των μεγάλων οργανισμών να εκμεταλλεύονται το σύνολο των δεδομένων τους που βρίσκεται αποθηκευμένα σε διάφορα τοπικά μηχανογραφικά συστήματα.

Συνοπτικά, οι λόγοι που καθιέρωσαν τα κατανεμημένα συστήματα σαν μια αποδεκτή και χρησιμοποιούμενη στην πράξη, τεχνολογία, μπορούν να περιγραφούν αδρά ως εξής:

- *οργανωτικοί και οικονομικοί λόγοι*, καθώς αποδείχθηκε, πολλές φορές στην πράξη ότι η συντήρηση κατανεμημένων συστημάτων μπορεί να είναι πιο αποδοτική από τη συντήρηση ενός κεντρικού συστήματος,

- *λόγοι αξιοποίησης της πληροφορίας*, καθώς οι κατανεμημένες βάσεις δεδομένων προκύπτουν σαν μια φυσική λύση σε γεωγραφικά κατανεμημένα δεδομένα που προσπελαύνονται από κεντρικές εφαρμογές,
- *λόγοι σταδιακής αύξησης του οργανισμού*, η οποία μπορεί να γίνει πιο εύκολα σε ένα κατανεμημένο, παρά σε ένα κεντρικό περιβάλλον,
- *λόγοι μείωσης του τηλεπικοινωνιακού φορτίου,*
- *λόγοι απόδοσης στην αποτίμηση ερωτήσεων,*
- *λόγοι αξιοπιστίας και διαθεσιμότητας της πληροφορίας*, μέσω πολλαπλών κατανεμημένων αντιγράφων της.

Βέβαια, από την άλλη πλευρά, το τίμημα που πρέπει να πληρώσει κανείς είναι η ανάγκη διαχείρισης και συντήρησης μιας κατανεμημένης και πολύπλοκης εφαρμογής καθώς και το κόστος του επιπλέον λογισμικού και υλικού υποστήριξης.

Τα συστήματα κατανεμημένων βάσεων δεδομένων κλήθηκαν να καλύψουν τις απαιτήσεις των διαφόρων οργανισμών με την ίδια αξιοπιστία και ταχύτητα που τα κλασικά συστήματα βάσεων δεδομένων ανταποκρίνονταν στις αντίστοιχες απαιτήσεις. Βασικά χαρακτηριστικά ενός κατανεμημένου συστήματος βάσεων δεδομένων είναι τα εξής:

- *Έλεγχος του συστήματος*. Ο έλεγχος του συστήματος επιτελείται και τοπικά από τοπικούς διαχειριστές (administrators) και καθολικά, από κάποιον επιβλέποντα διαχειριστή.
- *Ανεξαρτησία από τα δεδομένα*. Βασικό χαρακτηριστικό των κατανεμημένων βάσεων δεδομένων είναι ο βαθμός στον οποίο η φυσική οργάνωση των δεδομένων είναι διαφανής στον προγραμματιστή.
- *Ύπαρξη αντιγράφων*. Σε ένα κατανεμημένο σύστημα, η μείωση του πλεονασμού των δεδομένων (που ήταν από τα βασικά ζητούμενα στις μη-κατανεμημένες βάσεις δεδομένων) δεν είναι πλέον τόσο σημαντικό -αντιθέτως, η ύπαρξη τοπικών αντιγράφων της πληροφορίας είναι ως ένα βαθμό και επιδιωκώμενη, για την αύξηση της απόδοσης του συστήματος.
- Η *βελτιστοποίηση των ερωτήσεων*, σε περιβάλλοντα κατανεμημένων βάσεων δεδομένων δεν γίνεται μόνο τοπικά, αλλά προηγείται και ένα στάδιο καθολικής βελτιστοποίησης.
- Η *αξιοπιστία*, ο *έλεγχος συντονισμού*, η *ανάνηψη* των συστημάτων, καθώς και η *δυνατότητα ορισμού δικαιωμάτων* στος χρήστες, είναι από τα βασικά χαρακτηριστικά και των κατανεμημένων συστημάτων βάσεων δεδομένων.

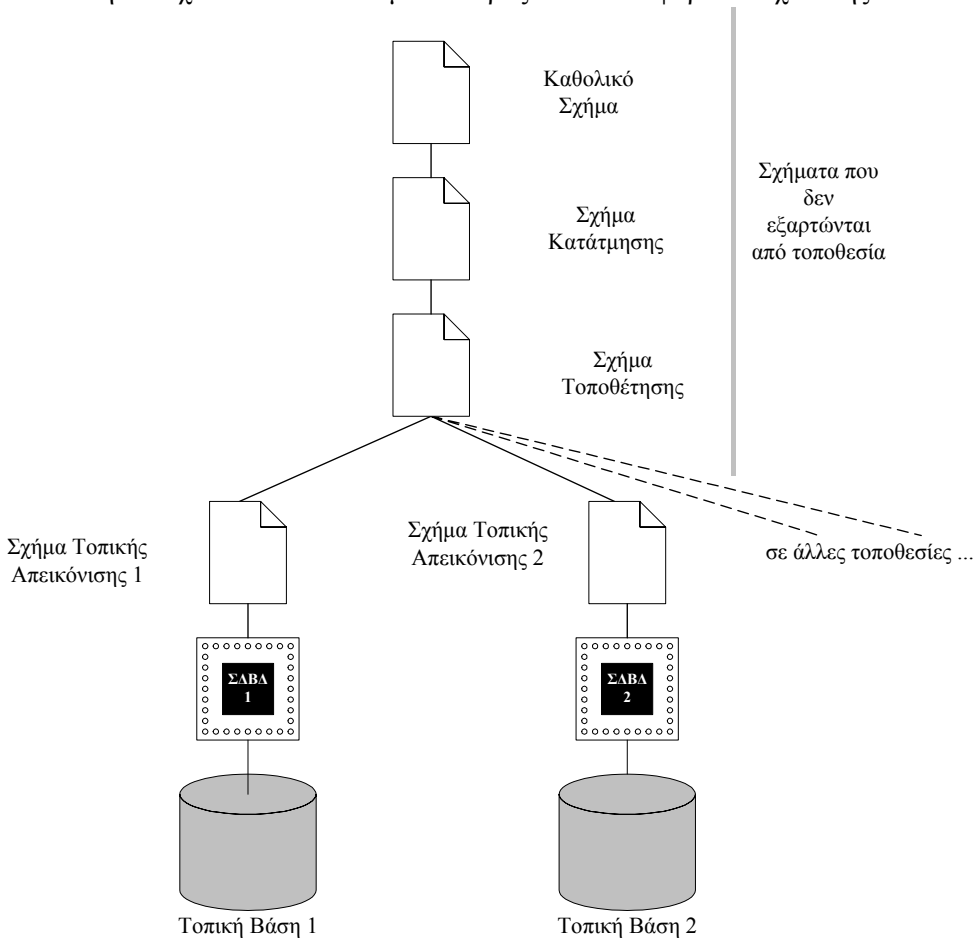
Με βάση τα παραπάνω, τα πιο βασικά προβλήματα που προέκυψαν (και εξακολουθούν -περισσότερο ή λιγότερο- να απασχολούν τον επιστημονικό κόσμο ακόμα) ήταν το πρόβλημα της σχεδίασης, της βελτιστοποίησης ερωτήσεων, του ελέγχου συντονισμού και της ανάνηψης σε περιβάλλοντα κατανεμημένων βάσεων δεδομένων. Αυτά είναι και τα προβλήματα που θα προσπαθήσουμε να καλύψουμε στο κεφάλαιο αυτό.

2.2 ΔΙΑΦΑΝΕΙΑ ΣΤΗΝ ΚΑΤΑΝΟΜΗ

Στις εφαρμογές βάσεων δεδομένων, ένας γενικός στόχος είναι η ανεξαρτησία των εφαρμογών από την τοποθεσία των δεδομένων. Στην περίπτωση των κατανεμημένων βάσεων δεδομένων, εκτός από την κλασική προσέγγιση στο πρόβλημα αυτό, υπάρχει και ένας επιπλέον παράγων δυσκολίας που πρέπει να ληφθεί υπόψη: η κατανομή των δεδομένων σε περισσότερες της μιας τοποθεσίες. Στη συνέχεια της ενότητας αυτής θα παρουσιάσουμε μια αφαιρετική αρχιτεκτονική για ένα σύστημα κατανεμημένων βάσεων δεδομένων, καθώς και διαφορετικούς τρόπους κατάτμησης. Για την παρουσίαση αυτή, θα στηριχτούμε κυρίως στο [CP84].

2.2.1 Αφαιρετική κατανομή των κατανεμημένων βάσεων δεδομένων

Στο Σχήμα 2.1 φαίνεται μια αφαιρετική αρχιτεκτονική μιας κατανεμημένης βάσης δεδομένων. Στη συνέχεια θα αναλύσουμε ένα προς ένα τα διάφορα στοιχεία της.



Σχήμα 2.1 Μια αφαιρετική αρχιτεκτονική κατανεμημένης βάσης δεδομένων

Στο υψηλότερο επίπεδο βρίσκεται το *καθολικό σχήμα (global schema)*. Το καθολικό σχήμα ορίζει όλα τα δεδομένα που βρίσκονται στην κατανεμημένη βάση δεδομένων, σε μια μορφή όπου η κατανομή είναι μη ορατή. Έτσι, το σχήμα είναι το ίδιο με την περίπτωση όπου η βάση είναι μη κατανεμημένη. Το καθολικό σχήμα (στην περίπτωση του σχεσιακού μοντέλου) αποτελείται από *καθολικές σχέσεις (global relations)*.

Κάθε καθολική σχέση μπορεί να διαχωριστεί σε διάφορα, μη επικαλυπτόμενα κομμάτια, τα οποία ονομάζονται *τμήματα (fragments)*. Η αντιστοίχιση των διαφόρων τμημάτων με τις καθολικές σχέσεις, ονομάζεται *σχήμα κατάτμησης (fragmentation schema)*. Στη συνέχεια, θα ακολουθήσουμε το συμβολισμό του [CP84] και θα συμβολίζουμε τα διάφορα τμήματα με το όνομα της καθολικής σχέσης στην οποία ανήκουν, έχοντας σαν δείκτη το όνομά τους. Έτσι, για παράδειγμα, με R_i ονομάζουμε το τμήμα i της καθολικής σχέσης R.

Τα τμήματα είναι οντότητες που ανήκουν στο λογικό μοντέλο της βάσης. Κάθε ένα από αυτά, κατά συνέπεια, έχει ένα ή περισσότερα αντίστοιχα τμήματα στο φυσικό μοντέλο. Έτσι, τα τμήματα μπορούν να είναι τοποθετημένα σε μία ή περισσότερες τοποθεσίες του δικτύου. Η αντιστοίχιση αυτή καθορίζεται από το *σχήμα τοποθέτησης (allocation schema)*. Είναι σαφές ότι

στην περίπτωση που ένα τμήμα απεικονίζεται σε περισσότερες της μίας τοποθεσίες, έχουμε πλεονάζουσα πληροφορία -πράγμα που δεν είναι πάντα ανεπιθύμητο. Όλα τα τμήματα που αντιστοιχούν στην ίδια καθολική σχέση *R* και βρίσκονται στην ίδια τοποθεσία *j* αποτελούν τη *φυσική εικόνα (physical image)* της σχέσης *R* στην τοποθεσία *j*, την οποία και συμβολίζουμε με R^j. Το *αντίγραφο ενός τμήματος (copy of a fragment)* R_i στην τοποθεσία j συμβολίζεται με R^i_j.

Τέλος, η απεικόνιση των φυσικών εικόνων στις διάφορες τοποθεσίες στα συγκεκριμένα ΣΔΒΔ που εξυπηρετούν τις τοποθεσίες αυτές, γίνεται μέσα από ένα *τοπικό σχήμα απεικόνισης (local mapping schema)* που είναι διαφορετικό για κάθε ΣΔΒΔ και εξαρτάται από αυτό.

Η γενική αυτή αρχιτεκτονική προέκυψε σαν μια αφαίρεση από τις αρχιτεκτονικές των υπαρχόντων συστημάτων. Τα συστήματα κατανεμημένων βάσεων δεδομένων μέσα στα πλαίσια αυτής της αρχιτεκτονικής προσπαθούν να επιτύχουν τους εξής σκοπούς:

- *Διαχώριση της διαδικασίας κατάτμησης με τη διαδικασία τοποθέτησης των δεδομένων*. Υπάρχουν δύο επίπεδα *διαφάνειας* στην κατανομή: η *διαφάνεια στην κατάτμηση (fragmetnation transparency)* και η *διαφάνεια στην τοποθεσία (location transparency)*. Η διαφάνεια στην κατάτμηση προσφέρει τη δυνατότητα στις εφαρμογές να μπορούν να γραφτούν γνωρίζοντας μόνο τις καθολικές σχέσεις, ενώ η διαφάνεια στην τοποθεσία επιτρέπει στις εφαρμογές να μπορούν να γραφτούν με γνώση μόνο των διαφορετικών τμημάτων. Στην ενότητα "Σχεδίαση Κατανεμημένων Βάσεων Δεδομένων" οι έννοιες αυτές θα αναδειχτούν περαιτέρω.
- *Ανεξαρτησία από το χρησιμοποιούμενο ΣΔΒΔ*. Το συγκεκριμένο ΣΔΒΔ που χρησιμοποιείται σε κάθε τοποθεσία, λαμβάνεται υπόψη μόνο στο επίπεδο του τοπικού σχήματος απεικόνισης. Η ιδιότητα αυτή ονομάζεται *διαφάνεια τοπικής απεικόνισης (local mapping transparency)* και ουσιαστικά διευκολύνει σημαντικά τη διαδικασία σχεδίασης της κατανεμημένης βάσης δεδομένων.
- *Διαχείριση της πλεονάζουσας πληροφορίας*. Η πλεονάζουσα πληροφορία είναι ιδιαίτερα χρήσιμη στις εφαρμογές των κατανεμημένων βάσεων δεδομένων, κυρίως για λόγους απόδοσης. Ο σαφής ορισμός του πλεονασμού της πληροφορίας βοηθά στη σωστή σχεδίαση, αλλά και συντήρηση ενός τέτοιου συστήματος. Στην περίπτωση δε, που ο προγραμματιστής μπορεί να κατασκευάσει εφαρμογές χωρίς να χρειάζεται να λαμβάνει υπόψη του την τοποθεσία των αντιγράφων, τότε έχουμε την περίπτωση της *διαφάνειας αντιγραφής (replication transparency)*.

Στη συνέχεια θα παρουσιάσουμε τους διαφορετικούς τρόπους με τους οποίους μπορεί να κατατμηθεί μια κατανεμημένη βάση δεδομένων.

2.2.2 Είδη κατάτμησης μιας κατανεμημένης βάσης δεδομένων

Υπάρχουν δύο βασικοί τρόποι για να διαμερίσουμε μια σχέση: η *οριζόντια κατάτμηση (horizontal fragmentation)* και η *κάθετη κατάτμηση (vertical fragmentation)*. Μπορούμε, βέβαια, να χρησιμοποιούμε και το συνδυασμό τους, όταν αυτό είναι απαραίτητο. Προτού παρουσιάσουμε κάθε έναν από αυτούς τους τρόπους, θα παρουσιάσουμε τρεις βασικές προϋποθέσεις που πρέπει να πληρεί ένα σχήμα κατάτμησης:

1. *Συνθήκη πληρότητας (completeness)*. Όλα τα δεδομένα μιας καθολικής σχέσης πρέπει να απεικονιστούν σε τμήματα (με άλλα λόγια δεν πρέπει να υπάρχουν δεδομένα που να ανήκουν στην καθολική σχέση αλλά όχι σε κάποιο τμήμα).
2. *Συνθήκη ανασύνθεσης (reconstruction)*. Πρέπει να είναι πάντα δυνατόν να ανασυνθέσουμε μια καθολική σχέση από τα διάφορα τμήματά της. Η συνθήκη αυτή είναι απαραίτητη καθώς η καθολική σχέση είναι, εν γένει, ιδεατή και πρέπει να μπορούμε να

την ανασυνθέσουμε από τα διάφορα τμήματά της που είναι οι οντότητες που αποθηκεύουμε στην πράξη.

3. *Συνθήκη διαχωρισιμότητας (disjointness)*. Έχει αποδειχτεί στην πράξη, ότι είναι χρήσιμο τα τμήματα να είναι διαφορετικά μεταξύ τους (να μην έχουν, δηλαδή, επικαλύψεις). Η διαχείριση των αντιγράφων είναι βολικό να γίνεται στο επίπεδο του σχήματος τοποθέτησης.

2.2.3 Οριζόντια κατάτμηση (horizontal fragmentation)

Η οριζόντια κατάτμηση συνίσταται στο διαχωρισμό των πλειάδων (tuples) μιας καθολικής σχέσης σε υποσύνολα (για παράδειγμα, σε μια γεωγραφικά κατανεμημένη βάση, να κρατάμε σε κάθε τοποθεσία μόνο τα τοπικά δεδομένα). Αυτό μπορεί να επιτευχθεί με μια συνθήκη επιλογής (selection) στην καθολική σχέση. Έστω για παράδειγμα η καθολική σχέση

PROJECT (PNUM, NAME, BUDGET, CITY)

και μια πιθανή κατάτμηση

$PROJECT_1 = \sigma_{CITY='ATH'}\ PROJECT$

$PROJECT_2 = \sigma_{CITY='SAL'}\ PROJECT$

Η παραπάνω κατάτμηση πληρεί τις συνθήκες που έχουμε ήδη προαναφέρει, καθώς:

- η συνθήκη πληρότητας πληρείται αν υποθέσουμε ότι δεν υπάρχουν άλλες τιμές στο πεδίο CITY στην καθολική σχέση PROJECT,
- η συνθήκη ανασύνθεσης πληρείται γιατί μπορούμε πάντα να εφαρμόσουμε την πράξη,

$$PROJECT = PROJECT_1 \cup PROJECT_2$$

- η συνθήκη διαχωρισιμότητας πληρείται εμφανώς

Αν γενικεύσουμε, μπορούμε να πούμε ότι η συνθήκη πληρότητας ικανοποιείται αν όλες οι πιθανές τιμές καλύπτονται από τις συνθήκες των διαφόρων τμημάτων. Η συνθήκη ανασύνθεσης ικανοποιείται πάντα από την πράξη ένωσης, ενώ η συνθήκη διαχωρισιμότητας απαιτεί τον αμοιβαίο αποκλεισμό (XOR) των συνθηκών.

Είναι ενδιαφέρον ότι η οριζόντια κατάτμηση μιας σχέσης μπορεί να γίνει, όχι με βάση τα πεδία της ίδιας της σχέσης, αλλά με βάση τις τιμές των πεδίων μιας άλλης σχέσης. Έστω, για παράδειγμα, η σχέση

DEPT_TASK (EMPNUM, PNUM, ROLE)

όπου EMPNUM είναι ο κωδικός κάποιου υπαλλήλου που εργάζεται στο PROJECT με κωδικό PNUM. Μπορεί να έχει νόημα να κατατμήσουμε τη σχέση DEPT_TASK με βάση τις πόλεις στις οποίες διεξάγεται το έργο στο οποίο εργάζεται ο κάθε υπάλληλος. Αυτό μπορεί να γίνει εφικτό, ακόμα και αν το πεδίο CITY δεν ανήκει στη σχέση DEPT_TASK. Ήτοι:

$DEPT\_TASK_1 = DEPT\_TASK \ltimes_{PNUM = PNUM} PROJECT_1$

$DEPT\_TASK_2 = DEPT\_TASK \ltimes_{PNUM = PNUM} PROJECT_2$

Το semi-join[1] εγγυάται ότι αφενός το $DEPT\_TASK_1$ θα έχει το ίδιο σχήμα με το DEPT_TASK αλλά και τις σωστές πλειάδες. Η κατάτμηση αυτού του είδους ονομάζεται *παραγόμενη κατάτμηση (derived fragmentation)*. Στη γενική περίπτωση οι συνθήκες που πρέπει να πληρούνται για μια σωστή παραγόμενη κατάτμηση είναι οι εξής:

---

[1] Ο τελεστής semi-join συμβολίζεται με $\ltimes$ και ορίζεται ως εξής: $R \ltimes_{A = B} S = \pi_{<attributes(R)>}(R \bowtie_{A = B} S)$

- Η πληρότητα απαιτεί να μην υπάρχει έργο στη σχέση DEPT_TASK που δεν βρίσκεται και στη σχέση PROJECT (συνθήκη που μπορεί να υλοποιηθεί άνετα με ένα περιορισμό ξένου κλειδιού).
- Η ανασύνθεση γίνεται πάλι με την πράξη της ένωσης.
- Η διαχωρισιμότητα ικανοποιείται όταν μια πλειάδα της σχέσης DEPT_TASK δεν αντιστοιχεί σε δύο πλειάδες της σχέσης PROJECT που ανήκουν σε διαφορετικά τμήματα. Στην περίπτωση που ο διαχωρισμός γίνεται με βάση ένα πρωτεύον κλειδί, αυτό μπορεί να επιβεβαιωθεί εύκολα -αλλιώτικα είναι δύσκολο να αποδείξουμε τη συνθήκη διαχωρισιμότητας.

2.2.4 Κάθετη κατάτμηση (vertical fragmentation)

Η κάθετη κατάτμηση αποτελεί το διαχωρισμό της καθολικής σχέσης με βάση την προβολή της σχέσης σε ομάδες πεδίων. Η κάθετη κατάτμηση είναι ορθή αν για κάθε πεδίο της καθολικής σχέσης υπάρχει τουλάχιστο ένα πεδίο σε κάποιο τμήμα και η καθολική σχέση μπορεί να ανασυντεθεί συνδέοντας (κάνοντας join) τα διάφορα τμήματα. Για παράδειγμα, έστω η καθολική σχέση:

EMP (EMPNUM, NAME, SALARY, TAX, MGRNUM, DEPTNUM)

Μια κάθετη κατάτμηση αυτής της σχέσης μπορεί να είναι

$EMP_1 = \pi_{EMPNUM,\ NAME,\ MGRNUM,\ DEPTNUM}\ EMP$

$EMP_2 = \pi_{EMPNUM,\ SAL,\ TAX}\ EMP$

Η ανασύνθεση της καθολικής σχέσης μπορεί να γίνει ως εξής

$EMP = EMP_1 \bowtie_{EMPNUM\ =\ EMPNUM} EMP_2$

Η λύση αυτή είναι εφικτή γιατί το EMPNUM είναι κλειδί στο EMP. *Η συμμετοχή του πρωτεύοντος κλειδιού σε κάθε τμήμα είναι ο πιο σίγουρος τρόπος για να εγγυηθεί κανείς ότι η ανασύνθεση είναι εφικτή με μια πράξη σύνδεσης*. Εναλλακτικά, θα μπορούσε κανείς να παράγει τεχνητά κλειδιά στα τμήματα, τα οποία, αφενός δεν μπορεί να είναι πολύ μεγάλα σε μέγεθος και αφετέρου δεν ανανεώνονται από τους χρήστες.

Να σημειώσουμε, επίσης, ότι η παραπάνω φόρμουλα παράγει στη ανασυνθεμένη σχέση δύο πεδία EMPNUM, πράγμα που μπορούμε να εξουδετερώσουμε με μια απλή προβολή.

Τέλος, σε ότι αφορά τη διαχωρισιμότητα των τμημάτων, μπορούμε να πούμε ότι αυτή δεν είναι και τόσο σημαντική όσο στην οριζόντια κατάτμηση. Ήδη, στο παράδειγμα που αναφέραμε, παραβιάσαμε τη διαχωρισιμότητα καθώς το κλειδί κάθε υπαλλήλου επαναλαμβάνεται σε κάθε τμήμα. Η επανάληψη ενός πεδίου (ακόμα και διαφορετικού από το κλειδί) μπορεί -εν γένει- να εξουδετερωθεί κατά την ανασύνθεση της σχέσης, με τις κατάλληλες προβολές.

2.2.5 Μεικτή κατάτμηση (mixed fragmentation)

Τα τμήματα που προκύπτουν από τις προαναφερθείσες κατατμήσεις είναι και αυτά σχέσεις και μπορούν με τη σειρά τους να κατατμηθούν και αυτές. Η ανασύνθεση μπορεί να γίνει εφαρμόζοντας τους κατάλληλους κανόνες με αντίστροφη σειρά από τη σειρά κατάτμησης. Έστω για παράδειγμα ξανά η σχέση

EMP (EMPNUM, NAME, SALARY, TAX, MGRNUM, DEPTNUM)

και η εξής μεικτή κατάτμηση:

$EMP_1 = \sigma_{DEPTNUM\ \le\ 10}\ \pi_{EMPNUM,\ NAME,\ MGRNUM,\ DEPTNUM}\ EMP$

$EMP_2 = \sigma_{10\ <\ DEPTNUM\ \le\ 20}\ \pi_{EMPNUM,\ NAME,\ MGRNUM,\ DEPTNUM}\ EMP$

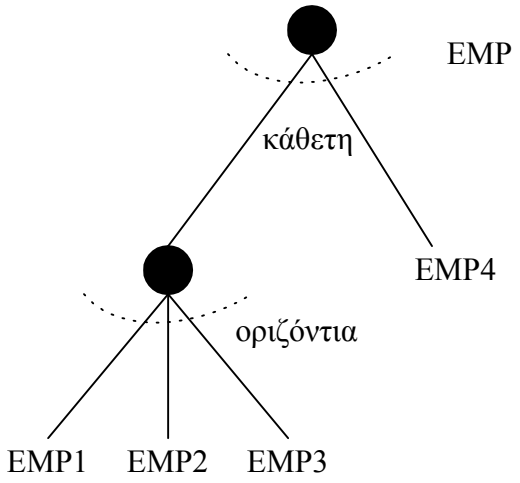
$EMP_3 = \sigma_{DEPTNUM\ >\ 20}\ \pi_{EMPNUM,\ NAME,\ MGRNUM,\ DEPTNUM}\ EMP$

$EMP_4 = \pi_{EMPNUM, NAME, SAL, TAX}\ EMP$

Η ανασύνθεσης της καθολικής σχέσης προκύπτει από την εξής φόρμουλα:

$EMP = \cup(EMP_1, EMP_2, EMP_3) \bowtie_{EMPNUM = EMPNUM} (\pi_{EMPNUM, SAL, TAX}\ EMP_4)$

Η μεικτή αυτή κατάτμηση μπορεί να αναπαρασταθεί σχηματικά όπως φαίνεται στο σχήμα 2.2. Έτσι, βλέπουμε ότι κατασκευάζουμε ένα *δέντρο κατάτμησης (fragmetation tree)*, στο οποίο η ρίζα αντιστοιχεί στην καθολική σχέση, τα φύλλα στα τελικά τμήματα και οι ενδιάμεσοι κόμβοι στα ενδιάμεσα στάδια κατάτμησης.



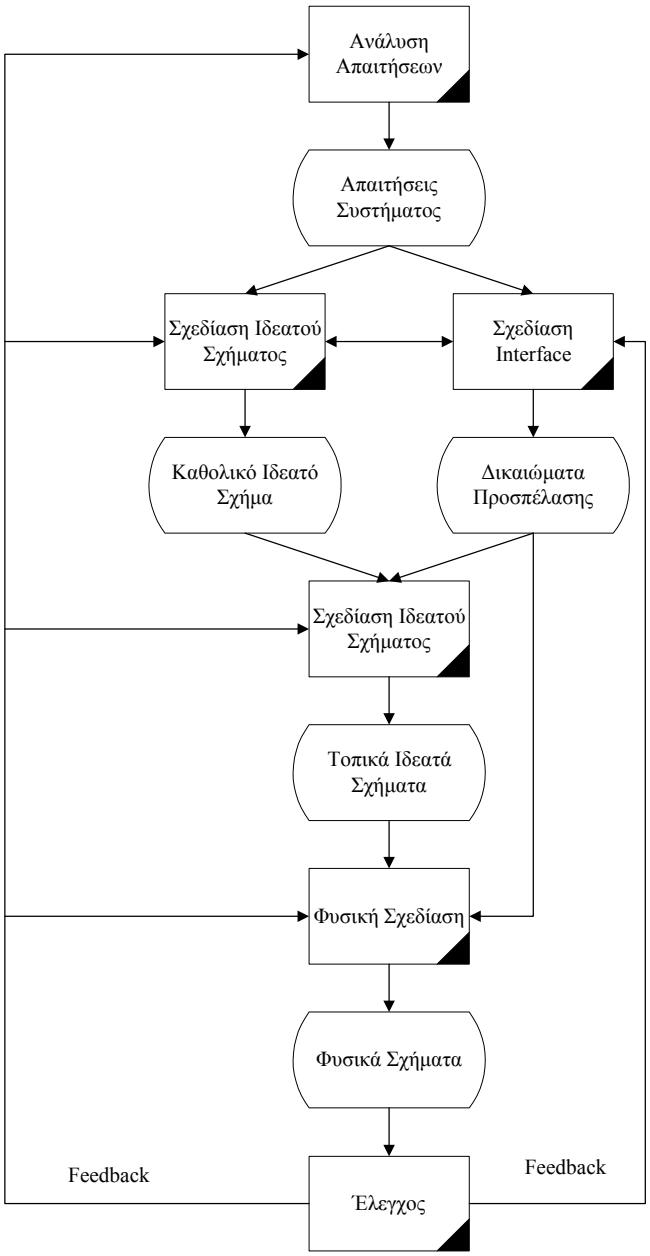
Σχήμα 2.2 Μια μεικτή κατάτμηση

2.3 ΣΧΕΔΙΑΣΗ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Υπάρχουν δύο κύριες στρατηγικές για τη σχεδίαση κατανεμημένων βάσεων δεδομένων: Η top-down και η bottom-up[2]. Όπως φαίνεται από τα ονόματά τους, οι δύο σχεδιάσεις είναι πολύ διαφορετικές μεταξύ τους -πολλές φορές, όμως, μπορούν (ή είναι αναγκαίο) να εφαρμοστούν συμπληρωματικά.

Η *top-down στρατηγική* είναι και η πλέον δημοφιλής, ιδιαίτερα στην περίπτωση που η κατανεμημένη βάση σχεδιάζεται από το μηδέν. Στο σχήμα 2.3 που προέρχεται από το [OV91] παρουσιάζεται σχηματικά η top-down στρατηγική.

[2] Η *bottom-up στρατηγική* ξεφεύγει από το σκοπό των σημειώσεων αυτών και δε θα καλυφθεί εδώ. Ο αναγνώστης παραπέμπεται στα [CP84], [OV91] για περαιτέρω ενημέρωση.



Σχήμα 2.3 Top-down διαδικασία σχεδίασης

Κατ' αρχήν, η διαδικασία ξεκινά με την αποτύπωση των απαιτήσεων του χρήστη που αποτελούν και το στόχο του συστήματος. Το έγγραφο όπου περιγράφονται οι απαιτήσεις των χρηστών αποτελεί είσοδο για δύο παράλληλες διαδικασίες: τη σχεδίαση του interface και τη σχεδίαση του ιδεατού σχήματος (conceptual design). Η διαδικασία αυτή σκοπό έχει την ανακάλυψη οντοτήτων και συσχετίσεων μεταξύ τους, στο περιβάλλον του υπό μοντελοποίηση οργανισμού. Η σχεδίαση του ιδεατού σχήματος, μπορεί εν γένει να σπάσει σε δύο διαδικασίες: στην ανάλυση των οντοτήτων και των συσχετίσεων, αφενός, και στη λειτουργική ανάλυση, αφετέρου, η οποία ασχολείται με τη μοντελοποίηση των βασικών διαδικασιών του υπό

μοντελοποίηση οργανισμού. Οι διαδικασίες σχεδίασης του interface και του ιδεατού σχήματος πρέπει να ελεγχθούν ως προς την συνέπεια μεταξύ τους.

Συνέπεια της διαδικασίας σχεδίασης είναι η παραγωγή του καθολικού ιδεατού σχήματος (global conceptual schema) αφενός, και των δικαιωμάτων πρόσβασης στην πληροφορία αφετέρου. Πρέπει να σημειώσουμε ότι μέχρι στιγμής η διαδικασία δεν έχει ασχοληθεί καθόλου με προβλήματα κατανομής της πληροφορίας και είναι ίδια με τη διαδικασία που ακολουθείται σε μη κατανεμημένα συστήματα.

Το επόμενο στάδιο είναι η διαδικασία κατανομής που σκοπό έχει να παράγει τα τοπικά ιδεατά σχήματα (local conceptual schemata). Από κει και πέρα, επέρχονται οι διαδικασίες κατάτμησης και τοποθέτησης με την κατασκευή των αντίστοιχων σχημάτων.

Στο τελικό στάδιο, έχουμε τη διαδικασία φυσικής σχεδίασης, η οποία λαμβάνει υπόψη της τα φυσικά χαρακτηριστικά των χρησιμοποιούμενων ΣΔΒΔ ανά τοποθεσία και έχει ως αποτέλεσμα την παραγωγή του φυσικού σχήματος. Η διαδικασία ελέγχου και προσαρμογής έρχεται όπως πάντα ως συμπληρωματική της ανάπτυξης ενός ολοκληρωμένου συστήματος.

Στη συνέχεια, θα ασχοληθούμε με το πρόβλημα της σχεδίασης κατανεμημένων βάσεων δεδομένων σε δύο επίπεδα: το επίπεδο της κατάτμησης και το επίπεδο της τοποθέτησης των διαφόρων τμημάτων.

2.3.1 Κατάτμηση

Το πρώτο πρόβλημα που πρέπει να αντιμετωπιστεί στη διαδικασία σχεδίασης είναι η κατάτμηση των καθολικών σχέσεων. Η πρώτη ιδέα είναι να προσανατολιστεί κανείς στον οριζόντιο διαχωρισμό των σχέσεων, στοχεύοντας στη δημιουργία "ομάδων" από πλειάδες. Κάθε τέτοια ομάδα θα πρέπει να παρουσιάζει μια ομοιογένεια που να εξυπηρετεί μια λογική τοποθέτηση στη συνέχεια. Ήτοι, αν δύο πλειάδες πληρούν τις προϋποθέσεις για συμμετοχή σε μία ομάδα (από την πλευρά της τοποθέτησης), τότε η στρατηγική τοποθέτησης θα τις εντάξει στην ίδια τοποθεσία. Ο τελικός στόχος είναι να έχουμε τμήματα που να είναι πολύ κοντά στις φυσικές εικόνες των διαφόρων τοποθεσιών. Περαιτέρω, υπάρχει και η περίπτωση της κάθετης (ή της μεικτής) τοποθέτησης για ειδικότερες περιπτώσεις εφαρμογών. Στη συνέχεια θα μελετήσουμε αναλυτικότερα τους διαφορετικούς τρόπους κατάτμησης.

Οριζόντια κατάτμηση

Η οριζόντια κατάτμηση οφείλει να γίνεται με τέτοιο τρόπο, ώστε να πληρούνται -όσο το δυνατό αυτό είναι εφικτό- οι συνθήκες πληρότητας, διαχωρισιμότητας και αναδόμησης (όπως αυτές ορίστηκαν στην προηγούμενη ενότητα). Υπάρχουν δύο είδη κατάτμησης, η *πρωτεύουσα (primary)*, όπου οι συνθήκες κατάτμησης μιας καθολικής σχέσης αφορούν αποκλειστικά τη σχέση αυτή και η *παραγόμενη (derived)*, όπου οι συνθήκες κατάτμησης μιας καθολικής σχέσης μπορούν να χρησιμοποιούν και άλλες σχέσεις.

Κατ' αρχήν, θα ασχοληθούμε με την πρωτεύουσα κατάτμηση. Προτού παρουσιάσουμε τον αλγόριθμο κατάτμησης, θα δώσουμε μερικούς ορισμούς από το [CP84].

Έστω *R* η καθολική σχέση την οποία θέλουμε να κατατμήσουμε οριζόντια. Ονομάζουμε *απλό κατηγόρημα (simple predicate)* ένα κατηγόρημα της μορφής:

πεδίο = τιμή

Ονομάζουμε *σύνθετο κατηγόρημα (minterm predicate) y* για ένα σύνολο απλών κατηγορημάτων *P*, τη σύζευξη όλων των κατηγορημάτων που εμφανίζονται στο *P* είτε στην κανονική τους μορφή, είτε στην άρνησή τους, αρκεί η έκφραση που παράγεται να μην έχει λογικές αντιφάσεις. Ήτοι:

y = AND (p_i^*), όπου $p_i \in P$, $p_i^* = p_i$ ή $p_i^* =$ NOT p_i, y ≠ false

Ονομάζουμε *τμήμα (fragment)* το σύνολο των πλειάδων που ικανοποιούν ένα σύνθετο κατηγόρημα. Ακόμα, ένα απλό κατηγόρημα p_i είναι *σχετικό* με ένα σύνολο απλών κατηγορημάτων *P* αν υπάρχουν τουλάχιστο δύο σύνθετα κατηγορήματα του *P*, των οποίων η έκφραση διαφέρει μόνο στο p_i και τα οποία χρησιμοποιούνται από τουλάχιστο μία εφαρμογή. Με άλλα λόγια, το p_i εμφανίζεται στο ένα σύνθετο κατηγόρημα ως έχει και στο άλλο εμφανίζεται η άρνησή του -κατά τα λοιπά δε, τα δύο σύνθετα κατηγορήματα είναι ίδια.

Έστω $P = \{p_1, p_2, ..., p_n\}$ ένα σύνολο από απλά κατηγορήματα. Για να είναι ορθή η κατάτμηση, και το *P* να αποτελεί τη συνθήκη κατάτμησης, πρέπει το *P* να είναι *πλήρες (complete)* και *ελάχιστο (minimal)*. Το *P* είναι πλήρες αν κάθε δύο πλειάδες που ανήκουν στο τμήμα που ορίζει το *P*, χρησιμοποιούνται από τις εφαρμογές με την ίδια πιθανότητα. Το *P* είναι, δε, ελάχιστο, αν όλα τα απλά κατηγορήματά του είναι σχετικά με αυτό.

Αλγόριθμος πρωτεύουσας οριζόντιας κατάτμησης.
Είσοδος: R: σχέση, P: σύνολο απλών κατηγορημάτων
Έξοδος: P': σύνολο απλών κατηγορημάτων
Μεταβλητές: F: σύνολο σύνθετων κατηγορημάτων (f: τμήμα που ορίζεται από ένα σύνθετο κατηγόρημα στο P').

```
begin
    βρες ένα p_i ∈ P, τ.ω. το p_i χωρίζει την R σε δύο τμήματα τα
    οποία προσπελαύνονται με διαφορετικό τρόπο από μία
    τουλάχιστο εφαρμογή;
    P' := {p_i};
    P := P - {p_i};
    F := {f_i};        {το σύνθετο τμήμα που ορίζεται από το p_i}
    do
      begin
          βρες ένα p_j τ.ω. το p_j χωρίζει ένα τμήμα f_j που
          ορίζεται από ένα σύνθετο κατηγόρημα στο P' σε δύο
          τμήματα τα οποία προσπελαύνονται με διαφορετικό τρόπο
          από μία τουλάχιστο εφαρμογή;
          P' := P' ∪ {p_j};
          P := P - {p_j};
          F := F ∪ {f_j};
          if ∃ p_k ∈ P' που είναι μη σχετικό then
          begin
                P' := P' - {p_k};
                F := F - {f_k};
          end-if
    until P' είναι πλήρες
end.
```

Ο αλγόριθμος αυτός μας δίνει ένα πλήρες και ελάχιστο σύνολο από κατηγορήματα *P'*, δεδομένου ενός αρχικού συνόλου *P*. Στην περίπτωση που μια συνολική λύση είναι πολύ δύσκολο να επιτευχθεί από τον αλγόριθμο αυτό, μπορούμε να κάνουμε κάποιους συμβιβασμούς

(π.χ. αναγωγή του προβλήματος σε ένα υποσύνολο κρίσιμων εφαρμογών, μη διάκριση τμημάτων που είναι παρόμοια κλπ.)

Στη συνέχεια, θα δώσουμε ένα συνολικό παράδειγμα για όλα όσα προαναφέρθηκαν, το οποίο βασίζεται κυρίως στο [CP84].

Παράδειγμα.

Έστω η τεχνική εταιρεία ΦΑΤΑΟΥΛΑΣ Α.Ε., η οποία έχει αναλάβει 2 μεγάλα έργα για την Ολυμπιάδα του 4004 που θα γίνει στην επαρχία Ελλάδας των Ενωμένων Ευρωπαϊκών Εμιράτων. Τα έργα γίνονται στην Καστοριά και στη Ρόδο. Τα κεντρικά γραφεία της εταιρείας, όπου γίνεται και η διαχείριση της εταιρείας βρίσκονται -πού αλλού;- στην Αθήνα. Η εταιρεία είναι χωρισμένη σε δύο μεγάλες ομάδες: αυτή που ασχολείται με τα έργα στο Βορρά και αυτή που ασχολείται με τα έργα στο Νότο. Τα έργα που γίνονται στην περιφέρεια της Αθήνας είναι ελάχιστα και είναι κυρίως διαχειριστικής φύσεως, σε σχέση με τα μεγάλα περιφερειακά έργα. Έτσι, εντάσσονται και αυτά στο βόρειο ή νότιο τμήμα, κατά περίπτωση. Κάθε έργο αποτελείται από πολλά υποέργα, τα οποία και αποτελούν τη βασική οντότητα πληροφορίας. Έχουμε τρία πληροφοριακά κέντρα, ένα σε κάθε πόλη, που θέλουμε να μοιράζονται την ίδια κατανεμημένη βάση δεδομένων. Οι καθολικές σχέσεις που έχουμε για την εταιρεία είναι οι εξής:

```
EMP (EMPNUM, NAME, SALARY, TAX, MGRNUM, DEPTNUM)
DEPT (DEPTNUM, NAME, AREA, MGRNUM)
DEPT_TASK(DEPTNUM, TNUM, ROLE, BUDGET)
TASK (TNUM, NAME, PROGRESS, CITY)
```

Κατ' αρχήν, θα ασχοληθούμε με τη σχέση `TASK`. Έχουμε τρεις πόλεις, όπως είπαμε, `'KAST'`, `'RHO'`, `'ATH'` με τις δύο πρώτες να είναι περίπου ίδιες σε συχνότητα στις πλειάδες της καθολικής σχέσης. Ας υποθέσουμε ότι η πιο συχνή εφαρμογή θέλει να βλέπει το όνομα και την πρόοδο κάθε υποέργου. Ήτοι:

```
SELECT NAME, PROGRESS
FROM TASK
WHERE TNUM = $1;
```

Η εν λόγω ερώτηση τίθεται σε οποιαδήποτε πόλη. Στην Καστοριά, η πιθανότητα η ερώτηση να αφορά τα τοπικά έργα είναι 80%. Το αντίστοιχο ισχύει και για τη Ρόδο. Στην Αθήνα η πιθανότητα να ρωτάει κάποιος για έργα στην Καστοριά και στη Ρόδο είναι η ίδια. Μπορούμε ως εκ τούτου να βρούμε τα εξής απλά κατηγορήματα:

```
p1: CITY = 'KAST'
p2: CITY = 'RHO'
```

Μπορεί κανείς να αποδείξει ότι το σύνολο $\{p1, p2\}$ είναι πλήρες και ελάχιστο. Τα σύνθετα κατηγορήματα που μπορούμε να παράγουμε είναι:

```
q1: CITY = 'KAST' AND CITY = 'RHO'
q2: CITY = 'KAST' AND NOT (CITY = 'RHO')
q3: NOT (CITY = 'KAST') AND CITY = 'RHO'
q4: NOT(CITY = 'KAST') AND NOT (CITY = 'RHO')
```

Τα σύνθετα κατηγορήματα *q1* και *q4* εμπεριέχουν αντιφάσεις και ως εκ τούτου δεν λαμβάνονται υπόψη. Επιπλέον, εάν λάβουμε υπόψη και τις λογικές συνεπαγωγές

```
CITY = 'KAST' => NOT(CITY = 'RHO')
CITY = 'RHO' => NOT(CITY = 'CAST')
```

μπορούμε να συνάγουμε ότι τελικά από τα *q2*, *q3* προκύπτουν τα *p1*, *p2* και άρα το σύνολο {*p1, p2*} είναι πλήρες και ελάχιστο.

Το ενδιαφέρον είναι ότι ενώ στο query της εφαρμογής δεν υπάρχει πουθενά το πεδίο CITY, η κατάτμηση της σχέσης γίνεται με βάση αυτό, καθώς συμμετέχει έμμεσα στη σχέση των πεδίων TNUM και CITY.

Η σχέση TASK ως εκ τούτου διαχωρίζεται στα τμήματα:

$TASK_1 = \sigma_{CITY='KAST'}\ TASK$

$TASK_2 = \sigma_{CITY='RHO'}\ TASK$

Η σχέση DEPT χρησιμοποιείται από δύο βασικές εφαρμογές που ψάχνουν για:

- πληροφορίες για τα υποέργα που γίνονται στην Καστοριά και στη Ρόδο. Σε κάθε μία από αυτές τις δύο πόλεις γίνονται και οι ερωτήσεις για τα τμήματα που είναι στο Βορρά (ή στο Νότο αντίστοιχα).
- διαχειριστικές πληροφορίες για τα τμήματα. Για κάθε τμήμα, οι ερωτήσεις γίνονται τοπικά.

Ως εκ τούτου οι ερωτήσεις είναι οι εξής:

```
p1: CITY = 'KAST'
p2: CITY = 'RHO'
p3: CITY = 'ATH'
p4: AREA = 'NORTH'
p5: AREA = 'SOUTH'
```

Έστω ότι έχουμε κάνει την παραδοχή πώς τα τμήματα της Καστοριάς έχουν DEPTNUM από 1 ως 10, της Αθήνας από 11 ως 20 και της Ρόδου από 21 ως 30. Έτσι μπορούμε αν θέλουμε να απλοποιήσουμε τα παραπάνω κατηγορήματα σε:

```
p1: DEPTNUM ≤ 10
p2: 10 < DEPTNUM ≤ 20
p3: DEPTNUM > 20
p4: AREA = 'NORTH'
p5: AREA = 'SOUTH'
```

Τα σύνθετα κατηγορήματα που τελικά παράγουμε (και που το σύνολό τους είναι πλήρες και ελάχιστο) είναι:

```
q1: DEPTNUM ≤ 10
q2: 10 < DEPTNUM ≤ 20 AND AREA = 'NORTH'
q3: 10 < DEPTNUM ≤ 20 AND AREA = 'SOUTH'
q4: DEPTNUM > 20
```

Με βάση τα κατηγορήματα αυτά, η κατάτμηση της σχέσης DEPT γίνεται ως εξής:

$DEPT_1 = \sigma_{DEPTNUM \le 10}\ DEPT$

$DEPT_2 = \sigma_{10 < DEPTNUM \le 20\ AND\ AREA = 'NORTH'}\ DEPT$

$DEPT_3 = \sigma_{10 < DEPTNUM \le 20\ AND\ AREA = 'SOUTH'}\ DEPT$

$DEPT_4 = \sigma_{DEPTNUM > 20}\ DEPT$

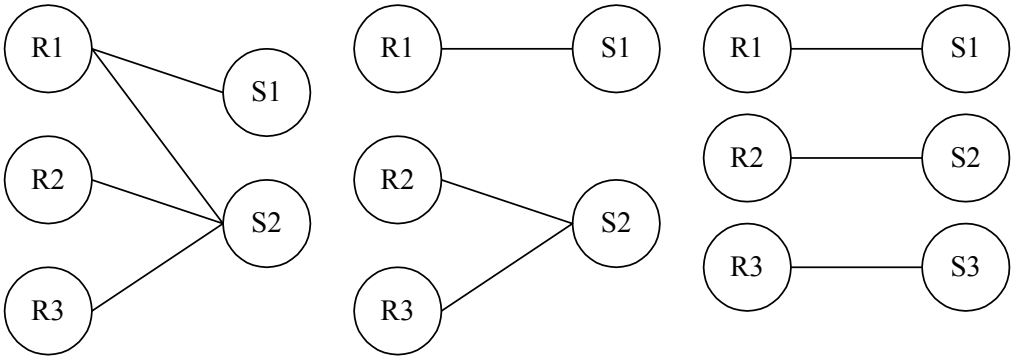
Σε ότι αφορά την *παραγόμενη οριζόντια κατάτμηση*, θα πρέπει αρχικά να εισάγουμε κάποιους όρους. Μια *κατανεμημένη σύνδεση (distributed join)* είναι μια σύνδεση μεταξύ κατανεμημένων σχέσεων. Κανονικά, όταν πρέπει να κάνουμε μια σύνδεση μεταξύ δύο καθολικών σχέσεων *R* και *S*, πρέπει να συνδέσουμε όλα τα τμήματα του *R* με όλα τα τμήματα του *S*. Αυτό μπορεί να αποφευχθεί αν μπορούμε να γνωρίζουμε ότι οι τιμές του πεδίου στο οποίο γίνεται η σύνδεση, είναι διαφορετικές για δύο τμήματα R_i και S_j.

Για να μπορέσουμε να αναπαραστήσουμε μια κατανεμημένη σύνδεση μπορούμε να χρησιμοποιήσουμε ένα *γράφο σύνδεσης (join graph)*. Ένας γράφος σύνδεσης *G (N, E)* μιας κατανεμημένης σύνδεσης μεταξύ δύο σχέσεων *R* και *S* αποτελείται από ένα σύνολο κόμβων *N* που αναπαριστούν τμήματα και ένα σύνολο *E* από μη κατευθυνόμενες ακμές μεταξύ τμημάτων, οι οποίες δεν έχουν ως αποτέλεσμα το κενό σύνολο. Στο σχήμα 2.4α φαίνεται ένας γράφος σύνδεσης.

Ένας γράφος σύνδεσης είναι *καθολικός (total)* αν περιέχει όλες τις πιθανές ακμές και *μειωμένος (reduced)* αν δεν είναι καθολικός. Υπάρχουν δύο είδη μειωμένου γράφου που παρουσιάζουν ιδιαίτερο ενδιαφέρον:

- ο *διαχωρισμένος (partitioned)* γράφος αποτελείται από δύο ή περισσότερους υπογράφους που δεν συνδέονται μεταξύ τους.
- ο *απλός (simple)* γράφος είναι διαχωρισμένος και κάθε υπογράφος αποτελείται από μία μόνο ακμή.

Στο Σχήμα 2.4 φαίνονται παραδείγματα διαχωρισμένου και απλού γράφου.



(α) Γράφος Σύνδεσης

(β) Διαχωρισμένος Γράφος Σύνδεσης

(γ) Απλός Γράφος Σύνδεσης

Σχήμα 2.4 Παραδείγματα γράφων σύνδεσης

Ένας απλός γράφος σημαίνει ότι αν τοποθετήσουμε τα συνδεόμενα τμήματα στην ίδια τοποθεσία, μπορούμε να υπολογίζουμε τα joins τοπικά και να ανασυνθέτουμε το ολικό αποτέλεσμα από τα επί μέρους αποτελέσματα (με κέρδος τη μειωμένη μετακίνηση δεδομένων στο δίκτυο).

Ακόμα, πρέπει να επισημάνουμε ότι *στην περίπτωση που έχουμε semi-join και ισχύουν οι συνθήκες πληρότητας και διαχωρισιμότητας που περιγράφηκαν στην προηγούμενη ενότητα, τότε ο γράφος σύνδεσης είναι απλός*. Στη συνέχεια, θα επεκτείνουμε το παράδειγμα που έχουμε εισάγει ως εξής:

Παράδειγμα.

Ας υποθέσουμε τώρα ότι θέλουμε να κατατμήσουμε την καθολική σχέση DEPT_TASK. Ας υποθέσουμε τις εξής εφαρμογές για τη σχέση DEPT_TASK:

- εφαρμογές για τη διαχείριση των υποέργων (όπου συσχετίζουμε υποέργα με τους εργαζόμενους σε αυτά με μια πράξη DEPT_TASK $\bowtie_{\text{TNUM = TNUM}}$ TASK)

- εφαρμογές για τη διαχείριση των τμημάτων της εταιρείας (όπου συσχετίζουμε τμήματα με τους εργαζόμενους σε αυτά με μια πράξη DEPT_TASK $\bowtie_{\text{DEPTNUM = DEPTNUM}}$ DEPT)

Αν υποθέσουμε ότι οι σχέσεις TASK και DEPT έχουν κατατμηθεί όπως προαναφέρθηκε, τότε έχουμε δύο πιθανές κατατμήσεις

$\text{DEPT\_TASK}_1 = \text{DEPT\_TASK} \ltimes_{\text{TNUM = TNUM}} \text{TASK}_1$

$\text{DEPT\_TASK}_2 = \text{DEPT\_TASK} \ltimes_{\text{TNUM = TNUM}} \text{TASK}_2$

και

$\text{DEPT\_TASK}_1 = \text{DEPT\_TASK} \ltimes_{\text{DEPTNUM = DEPTNUM}} \text{DEPT}_1$

$\text{DEPT\_TASK}_2 = \text{DEPT\_TASK} \ltimes_{\text{DEPTNUM = DEPTNUM}} \text{DEPT}_2$

$\text{DEPT\_TASK}_3 = \text{DEPT\_TASK} \ltimes_{\text{DEPTNUM = DEPTNUM}} \text{DEPT}_3$

$\text{DEPT\_TASK}_4 = \text{DEPT\_TASK} \ltimes_{\text{DEPTNUM = DEPTNUM}} \text{DEPT}_4$

Πρέπει να σημειώσουμε ότι το semi-join είναι επιτρεπτό καθώς οι κατατμήσεις έχουν γίνει με βάση τα κλειδιά των καθολικών σχέσεων. Η επιλογή της τελικής κατάτμησης εξαρτάται από ζητήματα συχνότητας των ερωτήσεων κάθε εφαρμογής. Επίσης, η ανασύνθεση των σχέσεων, σε οποιαδήποτε από τις δύο πιθανές κατατμήσεις, μπορεί να γίνει με τα semi-joins να εκτελούνται τοπικά -ήτοι ο γράφος σύνδεσης είναι απλός.

Κάθετη και μεικτή κατάτμηση

Υπάρχει περίπτωση οι εφαρμογές να επικεντρώνονται στην επερώτηση ομάδων από πεδία. Όπως έχουμε ήδη πει, οι συνθήκες ορθότητας απαιτούν κάθε πεδίο μιας καθολικής σχέσης να ανήκει σε ένα τουλάχιστο τμήμα και κάθε τμήμα να περιέχει είτε το κλειδί της σχέσης, είτε κάποιο τεχνητό κλειδί. Η κάθετη κατάτμηση έχει νόημα όταν οι εφαρμογές σε κάποια τοποθεσία προσπελαύνουν μόνο κάποια από τα πεδία της σχέσης και οι εφαρμογές σε κάποια άλλη τοποθεσία προσπελαύνουν κάποια άλλα. Αξίζει να παρατηρήσουμε εδώ, ότι η σχεδίαση της κάθετης κατάτμησης δεν μπορεί να είναι ανεξάρτητη από τη σχεδίαση της τοποθέτησης.

Αν και δεν υπάρχει κάποιος αλγόριθμος για την κάθετη κατάτμηση, υπάρχουν κάποιες *άπληστες (greedy)* ευρετικές λύσεις. Πιο συγκεκριμένα, στην *προσέγγιση διαχωρισμού (split approach)* οι καθολικές σχέσεις διαχωρίζονται προοδευτικά σε τμήματα, ενώ στην *προσέγγιση ενοποίησης (grouping approach)* πεδία σχέσεων ομαδοποιούνται προοδευτικά, για να κατασκευάσουν τμήματα.

Αν υποθέσουμε ότι τα τμήματα που προκύπτουν από την κάθετη κατάτμηση έχουν επικαλυπτόμενα τμήματα (διαφορετικά από το κλειδί) τότε έχουμε την περίπτωση της *αντιγραφής (replication)*. Η ύπαρξη επικαλύψεων ευνοεί τις διαδικασίες επερώτησης της κατανεμημένης βάσης καθώς η πιθανότητα τοπικής επεξεργασίας είναι μεγαλύτερη. Από την άλλη πλευρά, όμως, οι εφαρμογές ανανέωσης της βάσης έχουν το επιπλέον καθήκον να ενημερώνουν παραπάνω από ένα αντίγραφα κάθε φορά.

Τέλος, στην περίπτωση της μεικτής κατάτμησης, μπορούμε είτε να εφαρμόσουμε κάθετη κατάτμηση σε ήδη υπάρχοντα οριζόντια τμήματα, ή το ανάποδο. Είναι κοινή πρακτική, πάντως, να μην εφαρμόζεται η μεικτή κατάτμηση σε βάθος μεγαλύτερο από δύο.

Παράδειγμα.

Ας υποθέσουμε τώρα ότι θέλουμε να κατατμήσουμε την καθολική σχέση EMP. Ας υποθέσουμε τις εξής εφαρμογές για τη σχέση EMP:

- διαχειριστικές εφαρμογές που γίνονται στην Αθήνα και αφορούν τα φορολογικά/μισθολογικά ζητήματα των υπαλλήλων (ήτοι τα πεδία NAME, SAL, TAX)

- εφαρμογές για τη διαχείριση των τμημάτων της εταιρείας (που αφορούν τα πεδία NAME, MGRNUM και DEPTNUM)

Μπορούμε να κατατμήσουμε τη σχέση EMP κάθετα με τον εξής τρόπο:

EMP_1 (EMPNUM, NAME, SALARY, TAX)
EMP_2 (EMPNUM, NAME, MGRNUM, DEPTNUM)

Παρατηρούμε ότι το κλειδί (EMPNUM) βρίσκεται και στα δύο τμήματα. Επίσης, έχουμε επικάλυψη στο πεδίο NAME. Αν το όνομα του υπαλλήλου άλλαζε συχνά, τότε αυτό θα ήταν ένα πιθανό πρόβλημα, καθώς θα έπρεπε να το ανανεώνουμε διπλά. Επειδή, όμως, αυτό δεν ισχύει και επιπλέον, οι αναφορές που παράγουν τα τμήματα μηχανογράφησης αναφέρονται στο όνομα των υπαλλήλων, η αντιγραφή κρίθηκε σκόπιμη.

Αν τώρα υποθέσουμε, ότι οι εφαρμογές για τη διαχείριση των τμημάτων γίνονται από κάθε τμήμα τοπικά με μεγάλη πιθανότητα, τότε μπορούμε να κατατμήσουμε περισσότερο τη σχέση EMP_2 ανάλογα με τα τμήματα της σχέσης DEPT. Έτσι, θα έχουμε την εξής τελική μεικτή κατάτμηση για την EMP:

EMP_1 (EMPNUM, NAME, SALARY, TAX)
EMP_2 (EMPNUM, NAME, MGRNUM, DEPTNUM) –που δε χρησιμοποιείται
$EMP_3 = EMP_2 \ltimes_{DEPTNUM = DEPTNUM} DEPT_1$
$EMP_4 = EMP_2 \ltimes_{DEPTNUM = DEPTNUM} DEPT_2$
$EMP_5 = EMP_2 \ltimes_{DEPTNUM = DEPTNUM} DEPT_3$
$EMP_6 = EMP_2 \ltimes_{DEPTNUM = DEPTNUM} DEPT_4$

Παρατηρούμε ότι ο γράφος σύνδεσης είναι απλός και ότι το semi-join είναι επιτρεπτό αφού γίνεται πάνω στο κλειδί.

2.3.2 Τοποθέτηση τμημάτων

Η *τοποθέτηση των τμημάτων (fragment allocation)* είναι το επόμενο καθήκον του σχεδιαστή μιας κατανεμημένης βάσης δεδομένων, μετά τη σχεδίαση των τμημάτων. Υπάρχει μια βασική σχεδιαστική απόφαση που πρέπει να λάβει κανείς όταν κάνει την τοποθέτηση των τμημάτων: η ύπαρξη, ή όχι, πλεονάζουσας πληροφορίας.

Στην απλή περίπτωση που δεν υπάρχει πλεονάζουσα πληροφορία, ο πιο συνήθης τρόπος σχεδίασης είναι η τακτική του *καλύτερου ταιριάσματος (best fit):* για κάθε δυνατή τοποθέτηση βρίσκουμε ένα μέτρο κόστους και επιλέγουμε την τοποθεσία με το καλύτερο μέτρο. Φυσικά, αυτή η μέθοδος δεν έχει τη δυνατότητα να εκμεταλλευτεί συσχετίσεις μεταξύ τμημάτων.

Εάν, από την άλλη πλευρά, έχουμε την περίπτωση της πλεονάζουσας πληροφορίας, υπάρχουν δύο βασικές μέθοδοι:

- στην τακτική *της εύρεσης των τοποθεσιών που έχουν "κέρδος" (all benefitial sites)* κατασκευάζουμε ένα σύνολο από τοποθεσίες. Βρίσκουμε το κέρδος του να τοποθετήσουμε ένα αντίγραφο ενός τμήματος σε κάθε στοιχείο του συνόλου των τοποθεσιών και επιλέγουμε τις τοποθεσίες με το μεγαλύτερο κέρδος.
- στη μέθοδο της *προσθετικής αντιγραφής (additional replication)* κατασκευάζουμε πρώτα τη λύση του προβλήματος χωρίς αντίγραφα και μετά αρχίζουμε να προσθέτουμε αντίγραφα (με φθίνουσα σειρά κέρδους) μέχρι να βρούμε αρνητικό κέρδος σε κάποια τοποθέτηση.

Στη συνέχεια, θα παρουσιάσουμε πιο αναλυτικά τα μοντέλα κόστους που χρησιμοποιούμε σε κάθε μέθοδο, όπως αυτά παρουσιάζονται στο [CP84]. Θα ξεκινήσουμε με την περιγραφή του συμβολισμού. Συμβολίζουμε:

- με δείκτη *i* τα τμήματα
- με δείκτη *j* τις τοποθεσίες
- με δείκτη *k* τις εφαρμογές
- με f_{kj} τη συχνότητα χρήσης της εφαρμογής *k* στην τοποθεσία *j*
- με r_{ki} τον αριθμό των πράξεων ανάγνωσης που κάνει η εφαρμογή *k* στο τμήμα *i*
- με u_{ki} τον αριθμό των πράξεων ενημέρωσης που κάνει η εφαρμογή *k* στο τμήμα *i*
- με n_{ki} το συνολικό αριθμό των πράξεων που κάνει η εφαρμογή *k* στο τμήμα *i* ($n_{ki} = r_{ki} + u_{ki}$)

Οριζόντια κατάτμηση

Ας εξετάσουμε την *τακτική του καλύτερου ταιριάσματος* (για την περίπτωση που δεν έχουμε αντιγραφή). Έστω ότι θέλουμε να τοποθετήσουμε τη σχέση R_i στην τοποθεσία που ο συνολικός αριθμός πράξεων είναι μέγιστος. Ο αριθμός των τοπικών αναφορών στην τοποθεσία *j* είναι:

$$B_{ij} = \sum_k f_{kj} \; n_{ki}$$

Το τμήμα R_i θα τοποθετηθεί στην τοποθεσία *j* που το B_{ij} είναι μέγιστο.

Στην περίπτωση της τακτικής *της εύρεσης των τοποθεσιών που έχουν "κέρδος"* τοποθετούμε το R_i σε όλες τις τοποθεσίες που το κέρδος από τις αναγνώσεις τοπικά είναι μεγαλύτερο από το κόστος της ανανέωσης του τμήματος από εφαρμογές που τρέχουν σε άλλες τοποθεσίες. Το κέρδος από μια τοποθέτηση του τμήματος *i* στην τοποθεσία *j* είναι:

$$B_{ij} = \sum_k f_{kj} \; r_{ki} - C \times \sum_k \sum_{j' \neq j} f_{kj'} \; u_{ki}$$

Το *C* είναι μια σταθερά που υπολογίζει το λόγο μεταξύ του κόστους ανανέωσης και του κόστους ανάγνωσης (ο οποίος λογικά οφείλει να είναι μεγαλύτερος της μονάδας -η ανάγνωση είναι εν γένει πιο φτηνή από την ανανέωση). Το R_i τοποθετείται σε όλες τις τοποθεσίες που το B_{ij} είναι θετικό. Αν όλα τα B_{ij} είναι αρνητικά, τοποθετείται μοναδικά στην τοποθεσία με τη μέγιστη τιμή.

Στην περίπτωση της μεθόδου της *προσθετικής αντιγραφής*, το κέρδος από ένα παραπάνω αντίγραφο συνίσταται αφενός στην ταχύτητα προσπέλασης της πληροφορίας για κάποιες εφαρμογές και αφετέρου στην αξιοπιστία του συστήματος. Παρ' όλα αυτά, όμως, το κέρδος δεν αυξάνει αναλογικά με τον αριθμό των αντιγράφων, για τον προφανή λόγο ότι τα αντίγραφα πρέπει να κρατιούνται ενήμερα για κάθε αλλαγή που συμβαίνει σε οποιοδήποτε από αυτά. Αν συμβολίσουμε με d_i τον αριθμό των αντιγράφων του τμήματος R_i και με F_i το κέρδος όταν αντιγράψουμε το R_i σε όλες τις τοποθεσίες, μπορούμε να μετρήσουμε το κέρδος β από την αντιγραφή του τμήματος σε όλες τις τοποθεσίες ως εξής:

$$\beta(d_i) = (1 - 2^{1 - d_i}) \; F_i$$

Η εν λόγω συνάρτηση δεν είναι αναλογική, αυξάνει όμως όσο αυξάνει ο αριθμός των τοποθεσιών όπου έχουμε βάλει κάποιο αντίγραφο του R_i. Με βάση τη συνάρτηση αυτή μπορούμε να αποτιμήσουμε το κέρδος από την τοποθέτηση του τμήματος *i* στην τοποθεσία *j*:

$$B_{ij} = \sum_k f_{kj} \; r_{ki} - C \times \sum_k \sum_{j' \neq j} f_{kj'} \; u_{ki} - \beta(d_i)$$

Κάθετη κατάτμηση

Στην περίπτωση της κάθετης κατάτμησης, μπορούμε να έχουμε *επικάλυψη ή όχι*. Στην *περίπτωση που δεν υπάρχει επικάλυψη* αποτιμάται το κέρδος από την κατάτμηση του τμήματος R_i που βρίσκεται στην τοποθεσία *r*, σε δύο τμήματα R_s και R_t που βρίσκονται στις τοποθεσίες *s* και *t* αντίστοιχα. Σαν αποτέλεσμα της κατάτμησης αυτής έχουμε:

- Δύο σύνολα A_s και A_t από εφαρμογές που τρέχουν τοπικά στις τοποθεσίες *s* και *t* αντίστοιχα και χρησιμοποιούν η καθεμία πεδία μόνο της σχέσης R_s και R_t αντίστοιχα. Οι εφαρμογές αυτές επωφελούνται από την κατάτμηση καθώς γλιτώνουν τις αναφορές από μια μη τοπική τοποθεσία.
- Ένα σύνολο εφαρμογών A_1 που προηγουμένως έτρεχε μόνο στο *r* και χρησιμοποιούσε μόνο πεδία του R_s ή του R_t. Οι εφαρμογές αυτές πρέπει τώρα να κάνουν μια παραπάνω, μη τοπική, αναφορά σε μια τοποθεσία διαφορετική από το *r*.
- Ένα σύνολο εφαρμογών A_2 που προηγουμένως έτρεχε τοπικά στο *r* και που χρησιμοποιεί πεδία και του R_s και του R_t. Οι εφαρμογές αυτές πρέπει πλέον να κάνουν δύο επιπλέον προσπελάσεις (στις μη τοπικές τοποθεσίες) *s* και *t*.
- Ένα σύνολο εφαρμογών A_3 που τρέχει σε τοποθεσίες διαφορετικές από τις *r, s, t* και που αναφέρονται σε πεδία και του R_s και του R_t. Οι εφαρμογές αυτές κάνουν μια επιπλέον προσπέλαση σε μη τοπική τοποθεσία (αντί να επικοινωνούν μόνο με το *r*, τώρα πλέον πρέπει να επικοινωνούν και με το *s* και με το *t*).

Το κέρδος από την κάθετη κατάτμηση χωρίς επικάλυψη είναι:

$$B_{ist} = \sum_{k \in A_s} f_{ks}\ n_{ki} + \sum_{k \in A_t} f_{kt}\ n_{ki} - \sum_{k \in A1} f_{kr}\ n_{ki} - \sum_{k \in A2} 2 \times f_{kr}\ n_{ki} - \sum_{k \in A3} \sum_{j \notin r, s, t} f_{kj}\ n_{ki}$$

Στη συνάρτηση αυτή χρησιμοποιούμε προσεγγιστικά το συνολικό αριθμό των προσπελάσεων (n_{ki}). Ο πλήρης υπολογισμός θα απαιτούσε την αντικατάσταση του τελεστή n_{ki} με το άθροισμα $r_{ki} + C \times u_{ki}$.

Στην περίπτωση που έχουμε επικάλυψη, αποτιμάται το κέρδος από την κατάτμηση του τμήματος R_i που βρίσκεται στην τοποθεσία *r*, σε δύο τμήματα R_s και R_t που βρίσκονται στις τοποθεσίες *s* και *t* αντίστοιχα και έχουν ένα σύνολο από επικαλυπτόμενα πεδία *I*. Σε αυτή την περίπτωση, η θεώρηση που έχουμε για τα προαναφερθέντα σύνολα εφαρμογών αλλάζει ως εξής:

- Το σύνολο A_s αποτελείται από εφαρμογές που τρέχουν τοπικά στην τοποθεσία s και χρησιμοποιούν είτε διαβάζουν οποιοδήποτε πεδίο του R_s είτε ανανεώνουν πεδία του R_s που δεν ανήκουν στο *I*. Το ίδιο ισχύει και για το A_t.
- Το σύνολο A_2 περιλαμβάνει εφαρμογές ανανέωσης που προηγουμένως έτρεχαν τοπικά στο r και που χρησιμοποιούν πεδία του *I* (δηλαδή και του R_s και του R_t).
- Το σύνολο A_3 περιλαμβάνει εφαρμογές που τρέχουν σε τοποθεσίες διαφορετικές από τις *r, s, t* και που ανανεώνουν πεδία του *I* και κατά συνέπεια χρησιμοποιούν πεδία και του R_s και του R_t.

Η αποτίμηση του κέρδους γίνεται με χρήση της προαναφερθείσας εξίσωσης.

2.4 ΕΠΕΞΕΡΓΑΣΙΑ ΚΑΙ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΕΡΩΤΗΣΕΩΝ ΣΤΟ ΠΕΡΙΒΑΛΛΟΝ ΤΩΝ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Στην ενότητα αυτή, θα ασχοληθούμε με την επεξεργασία και τη βελτιστοποίηση ερωτήσεων στο περιβάλλον κατανεμημένων βάσεων δεδομένων. Πιο συγκεκριμένα, θα αντιμετωπίσουμε το πρόβλημα της μετατροπής ερωτήσεων που απευθύνονται σε καθολικές σχέσεις, σε ερωτήσεις που απευθύνονται σε συγκεκριμένα τμήματα της κατανεμημένης βάσης. Επιπλέον, θα εξετάσουμε και μεθόδους που βελτιστοποιούν την αποτίμηση μιας ερώτησης. Στη συνέχεια, θα ονομάζουμε *καθολικές ερωτήσεις (global queries)* τις ερωτήσεις που απευθύνονται σε καθολικές σχέσεις και *ερωτήσεις τμημάτων (fragment queries)* τις ερωτήσεις που απευθύνονται

σε τμήματα. Ένας από τους βασικούς στόχους των μετασχηματισμών είναι η πληρότητα και η ορθότητά τους (ήτοι, επιδιώκουμε ο συνδυασμός των ερωτήσεων τμημάτων να επιστρέφει ακριβώς και μόνο το αποτέλεσμα της καθολικής ερώτησης). Στην ενότητα αυτή, θα βασιστούμε κυρίως στο [CP84].

2.4.1 Ισοδυναμίες μετασχηματισμών

Μία σχεσιακή ερώτηση μπορεί να εκφραστεί σαν ο συνδυασμός διάφορων εκφράσεων της σχεσιακής άλγεβρας. Οι σχεσιακές αυτές εκφράσεις παρουσιάζουν αφενός τον τρόπο με τον οποίο θα υπολογιστεί το αποτέλεσμα και αφετέρου τη σειρά με την οποία θα γίνει ο υπολογισμός αυτός. Όμως, είναι γνωστό ότι η σειρά εκτέλεσης των διαφόρων αλγεβρικών πράξεων μπορεί να μην είναι μοναδική. Μπορούμε, δηλαδή, να έχουμε διαφορετικά πλάνα εκτέλεσης για το ίδιο αποτέλεσμα. Για παράδειγμα, οι ακόλουθες πράξεις είναι ισοδύναμες μεταξύ τους:

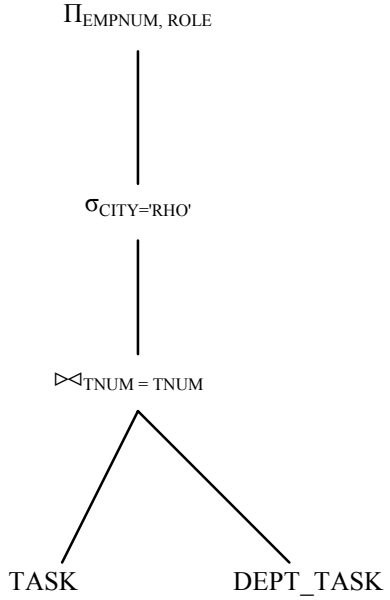
$\pi_{\text{NAME, DEPTNUM}}\ \sigma_{\text{DEPTNUM = 15}}$ EMP

$\sigma_{\text{DEPTNUM = 15}}\ \pi_{\text{NAME, DEPTNUM}}$ EMP

Στην βελτιστοποίηση ερωτήσεων στις βάσεις δεδομένων είναι καθιερωμένη τεχνική να χρησιμοποιούμε ένα *δέντρο τελεστών* (operator tree). Στο επόμενο παράδειγμα θα παρουσιάσουμε τον τρόπο με τον οποίο απεικονίζουμε μια σχεσιακή ερώτηση στο αντίστοιχο δέντρο τελεστών. Έστω η ερώτηση

$$\pi_{\text{TNUM, ROLE}}\ \sigma_{\text{CITY = 'RHO'}}\ (\text{TASK} \bowtie \text{DEPT\_TASK})$$

η οποία μετατρέπεται στο δέντρο τελεστών του σχήματος 2.5:



Σχήμα 2.5. Παράδειγμα δέντρου τελεστών

Υπάρχουν δύο ειδών τελεστές: οι *μοναδιαίοι (unary)* και οι *δυαδικοί (binary)*. Στους μοναδιαίους έχουμε την *επιλογή* και την *προβολή* (select και project), ενώ στους δυαδικούς εντάσσονται *το καρτεσιανό γινόμενο, η ένωση, η διαφορά, η τομή, η σύνδεση, η φυσική σύνδεση και η ημι-σύνδεση* (cartesian product, union, difference, intersection, join, natural join, semi-join).

Στη συνέχεια, θα παρουσιάσουμε κανόνες ισοδυναμιών μεταξύ συνδυασμών τελεστών. Όταν λέμε ότι δύο εκφράσεις είναι ισοδύναμες, εννοούμε ότι αν αντικαταστήσουμε τη μία με την άλλη, το αποτέλεσμα θα παραμείνει το ίδιο, ανεξάρτητα από τα δεδομένα που περιέχει η βάση δεδομένων. Θα χρησιμοποιήσουμε το σύμβολο *U* για τους μοναδιαίους τελεστές και το σύμβολο *B* για τους δυαδικούς, ενώ για τους σχεσιακούς πίνακες θα χρησιμοποιήσουμε το σύμβολο *R*.

Αντιμεταθετικότητα μοναδιαίων τελεστών

$$U_1\ U_2\ R \Leftrightarrow U_2\ U_1\ R$$

Αντιμεταθετικότητα δυαδικών τελεστών

$$R_1\ B\ R_2 \Leftrightarrow R_2\ B\ R_1$$

Προσεταιριστικότητα δυαδικών τελεστών

$$R_1\ B\ (R_2\ B\ R_3) \Leftrightarrow (R_1\ B\ R_2)\ B\ R_3$$

Μοναδική ποσότητα μοναδιαίων τελεστών

$$U\ R \Leftrightarrow U_1\ U_2\ R$$

Επιμεριστικότητα μοναδιαίων τελεστών σε σχέση με δυαδικούς

$$U(R_1)\ B\ U(R_2) \Leftrightarrow U(R_1\ B\ R_2)$$

Οι παραπάνω ιδιότητες δεν ισχύουν πάντα. **Εν γένει, οι ιδιότητες αυτές ισχύουν μόνο όταν κάποιες συνθήκες πληρούνται.** Εμείς, στη συνέχεια, θα παρουσιάσουμε μόνο ένα μικρό υποσύνολο των κανόνων που παρουσιάζουν ενδιαφέρον για το υπό συζήτηση αντικείμενο. Ο ενδιαφερόμενος αναγνώστης παραπέμπεται στα [OV91], [CP84] για μια πλήρη περιγραφή των κανόνων αυτών. Έχουμε λοιπόν, τα εξής χαρακτηριστικά παραδείγματα συνθηκών:

1. *Αντιμεταθετικότητα μεταξύ επιλογής και προβολής.* Η αντιμεταθετικότητα μεταξύ επιλογής και προβολής είναι επιτρεπτή μόνο όταν τα πεδία που λαμβάνουν μέρος στη συνθήκη επιλογής είναι υποσύνολο των πεδίων που αφήνει η προβολή στο τελικό αποτέλεσμα. Ήτοι:
$$\pi_{A1}\ \sigma_{F2}\ R = \sigma_{F2}\ \pi_{A1}\ R, \text{ όταν } Attr(F_2) \subseteq A_1$$
2. *Προσεταιριστικότητα μεταξύ συνδέσεων.* Η προσεταιριστικότητα μεταξύ συνδέσεων είναι επιτρεπτή όταν τα πεδία στα οποία γίνεται η δεύτερη σύνδεση είναι υποσύνολο των πεδίων της δεύτερης και της τρίτης σχέσης.
$$R_1 \bowtie_{F1} (R_2 \bowtie_{F2} R_3) \Leftrightarrow (R_1 \bowtie_{F1} R_2) \bowtie_{F2} R_3, \text{ όταν } Attr(F_2) \subseteq Attr(R_2) \cup Attr(R_3)$$
3. *Μοναδική ποσότητα μεταξύ προβολών.* Η μοναδική ποσότητα μεταξύ προβολών είναι επιτρεπτή όταν τα πεδία της προβολής του αριστερού μέλους της ισότητας είναι τα ίδια με αυτά της πρώτης προβολής του δεξιού σκέλους και υποσύνολο των πεδίων της δεύτερης προβολής.
$$\pi_A\ R = \pi_{A1}\ (\pi_{A2}\ R), \text{ όταν } A = A_1,\ A \subseteq A_2$$
4. *Προσεταιριστικότητα μεταξύ καρτεσιανού γινομένου και προβολής.* Η προσεταιριστικότητα μεταξύ καρτεσιανού γινομένου και προβολής είναι επιτρεπτή όταν ισχύει η εξής συνθήκη:
$$\pi_A\ (R_1 \times R_2) = \pi_{A1}\ (R_1) \times \pi_{A2}\ (R_2), \text{ όταν } A_1 = A - Attr(R_1),\ A_2 = A - Attr(R_2)$$
5. *Προσεταιριστικότητα μεταξύ ένωσης και σύνδεσης.* Η παρακάτω συνθήκη ισχύει πάντα.

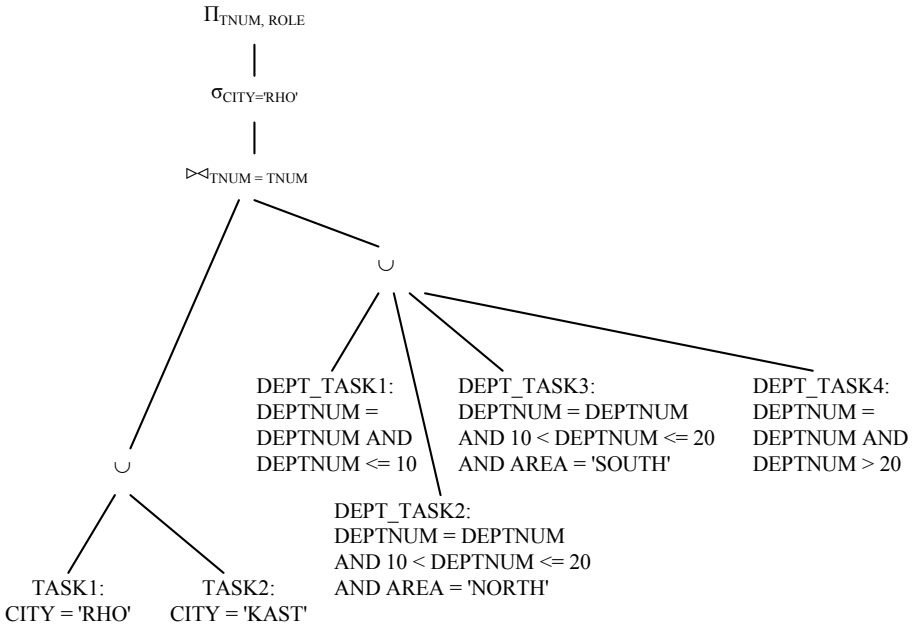
$$(R_1 \cup R_2) \bowtie (S_1 \cup S_2) = (R_1 \bowtie S_1) \cup (R_1 \bowtie S_2) \cup (R_2 \bowtie S_1) \cup (R_2 \bowtie S_2)$$

Όπως μπορεί να παρατηρήσει κανείς, η τελευταία ιδιότητα είναι εξαιρετικά χρήσιμη στον υπολογισμό του αποτελέσματος συνδέσεων κατανεμημένων σχέσεων (αν φανταστούμε ότι τα R_1, R_2, S_1, S_2 είναι τμήματα των καθολικών σχέσεων R, S και εκτελείται η ερώτηση $R \bowtie S$).

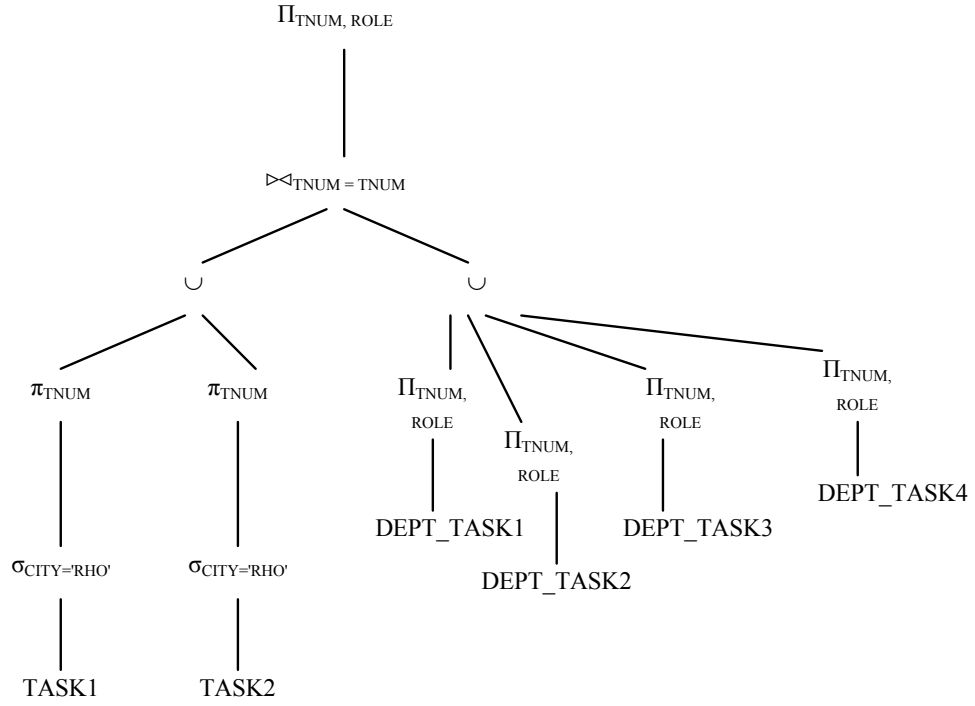
2.4.2 Μετασχηματισμοί καθολικών ερωτήσεων σε ερωτήσεις τμημάτων

Έστω μια αλγεβρική έκφραση μιας καθολικής ερώτησης. Η *κανονική έκφρασή* της *(canonical expression)*, είναι η ισοδύναμη έκφραση, όπου οι καθολικές σχέσεις έχουν αντικατασταθεί από αλγεβρικές εκφράσεις οι οποίες περιγράφουν τον τρόπο με τον οποίο ανασυντίθεται η καθολική σχέση από τα τμήματά της. Ο εν λόγω μετασχηματισμός είναι πολύ βασικός: η κανονική έκφραση περιγράφει τον τρόπο με τον οποίο η ερώτηση θα αποτιμηθεί στην πράξη. Βασική ιδιότητα του εν λόγω μετασχηματισμού είναι η πληρότητα και ορθότητα του υπολογισμού (με την προϋπόθεση, βέβαια, ότι η αναδόμηση της καθολικής σχέσης από τα τμήματά της γίνεται ορθά).

Είναι βασικό να επισημάνουμε ότι η κανονική έκφραση δεν εμπεριέχει κανένα ίχνος βελτιστοποίησης. Εγγυάται μεν την ορθότητα του αποτελέσματος, πλην όμως, στη συνήθη περίπτωση αυτό γίνεται με τον πλέον αργό τρόπο. Όμως, η κανονική έκφραση είναι και αυτή μια αλγεβρική έκφραση και ως εκ τούτου, όλες οι ισοδυναμίες που περιγράφηκαν προηγουμένως μπορούν να χρησιμοποιηθούν. Στη συνέχεια, θα παρουσιάσουμε πιο συστηματικά, κανόνες για τη βελτιστοποίηση ερωτήσεων σε περιβάλλοντα κατανεμημένων βάσεων δεδομένων. Προς το παρόν, στο παράδειγμα των σχημάτων 2.6 και 2.7, δίνουμε μια πρώτη εικόνα για το τι είναι μια κανονική έκφραση και πώς θα μπορούσε πιθανά να βελτιστοποιηθεί. Θα χρησιμοποιήσουμε το παράδειγμα του σχήματος 2.5



Σχήμα 2.6 Κανονική έκφραση μιας αλγεβρικής παράστασης



Σχήμα 2.7 Βελτιστοποίηση μιας κανονικής έκφρασης: οι επιλογές και οι προβολές γίνονται τοπικά

Στη συνέχεια, θα χρησιμοποιήσουμε κάποιες από τις προαναφερθείσες ισοδυναμίες, για να εξάγουμε κανόνες που χρησιμοποιούνται από βελτιστοποιητές ερωτήσεων για την επιτάχυνση της αποτίμησης των ερωτήσεων. Οι κανόνες βασίζονται αφενός σε μοντέλα κόστους και αφετέρου στην πειραματική επαλήθευσή τους. Ο ενδιαφερόμενος αναγνώστης μπορεί να καταφύγει στα [OV91], [CP84] για μια πιο αναλυτική παρουσίαση.

Κανόνας 1. Βασιζόμενοι στην επιμεριστικότητα των μοναδιαίων τελεστών σε σχέση με τους δυαδικούς, σπρώχνουμε επιλογές και προβολές όσο πιο χαμηλά γίνεται στο δέντρο (με αποτέλεσμα να μειώνουμε γρήγορα και τοπικά το μέγεθος της εμπλεκόμενης πληροφορίας). Για παράδειγμα, αντί για να κάνουμε την επιλογή $\sigma_{\texttt{CITY='RHO'}}$ (TASK ⋈ DEPT_TASK), προτιμήθηκε να σπρώξουμε την επιλογή μόνο στα τμήματα $\texttt{TASK}_1$ και $\texttt{TASK}_2$, ώστε να μειώσουμε το κόστος μεταφοράς δεδομένων.

Κανόνας 2. Βασιζόμενοι στην αντιμεταθετικότητα και τη μοναδική ποσότητα των μοναδιαίων τελεστών, παράγουμε συνδυασμούς από επιλογές και προβολές σε κάθε εμπλεκόμενη σχέση. Στο προηγούμενο παράδειγμα, αντί να εμπλέξουμε όλο το $\texttt{TASK}_1$, προτιμήθηκε να χρησιμοποιηθεί η έκφραση $\pi_{\texttt{TNUM}}\ \sigma_{\texttt{CITY='RHO'}}(\texttt{TASK}_1)$, η οποία σαφώς μειώνει το μέγεθος της αποτιμούμενης σχέσης.

Κανόνας 3. Μπορούμε να απαλείφουμε φύλλα του δέντρου (τμήματα δηλαδή των καθολικών σχέσεων), αν ο συνδυασμός των αλγεβρικών εκφράσεων που τα ορίζουν με τις αλγεβρικές εκφράσεις που τους επιβάλλονται έρχεται σε σύγκρουση. Για παράδειγμα, η έκφραση $\sigma_{\texttt{CITY='RHO'}}$ ($\texttt{TASK}_2$) ισοδυναμεί με την έκφραση $\sigma_{\texttt{CITY='RHO'}}\sigma_{\texttt{CITY='KAST'}}$ (TASK), η οποία προφανώς εμπεριέχει αντίφαση και δεν έχει νόημα να αποτιμηθεί. (Αντίστοιχα, σε μια περίπτωση κάθετης κατάτμησης, θα μπορούσε να συμβαίνει το ίδιο με μια προβολή).

Κανόνας 4. Πολλές φορές, εάν ο γράφος σύνδεσης είναι απλός, μπορούμε να εκτελούμε τις συνδέσεις πριν από τις ενώσεις (ήτοι ανεβάζουμε τις ενώσεις όσο πιο ψηλά μπορούμε στο δέντρο). Με τον τρόπο αυτό, εκτελούμε τις συνδέσεις τοπικά, με αποτέλεσμα να μειώνουμε νωρίς την εμπλεκόμενη πληροφορία που διακινούμε στο δίκτυο.

Κανόνας 5. Όπου έχουμε αυξημένη μετα-πληροφορία, μπορούμε να τη χρησιμοποιούμε για να αποφεύγουμε άσκοπες συνδέσεις. Για παράδειγμα, αν γνωρίζουμε ότι DEPTNUM < 10 ⇒ AREA = NORTH, έστω και αν αυτό δε φαίνεται στον ορισμό των τμημάτων, μπορούμε να αποφύγουμε τη σύνδεση ενός τμήματος με AREA = SOUTH με κάποιο τμήμα DEPT_TASK με συνθήκη DEPTNUM < 10.

Κανόνας 6. Χρησιμοποιούμε όσο το δυνατό πιο συχνά semi-joins αντί για joins, με σκοπό να μειώσουμε τη διακινούμενη πληροφορία.

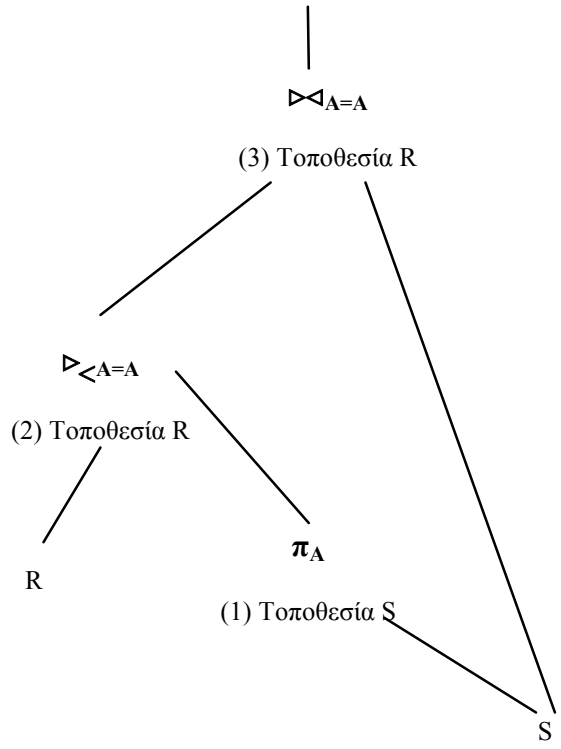
Ο ρόλος των semi-joins στην βελτιστοποίηση ερωτήσεων στις κατανεμημένες βάσεις δεδομένων

Το βασικό πρόβλημα της σύνδεσης (join) είναι ότι για να επιτευχθεί, είναι πιθανά απαραίτητο να μετακινηθούν ολόκληρες σχέσεις στο δίκτυο. Όπως έχουμε ήδη προαναφέρει, σκοπός της βελτιστοποίησης ερωτήσεων είναι να μειώσει το μέγεθος των δεδομένων που διακινούνται στο δίκτυο, καθώς το δίκτυο είναι το πιο ευαίσθητο, αλλά και αργό κομμάτι μιας κατανεμημένης βάσης δεδομένων. Η λύση του semi-join σκοπό έχει να μειώσει λοιπόν, τον όγκο των δεδομένων που μπαίνουν στο δίκτυο.

Αν υποθέσουμε ότι έχουμε μια σύνδεση μεταξύ δύο σχέσεων, τότε έχουμε διάφορους τρόπους να γράψουμε τη σύνδεση αυτή, ισοδύναμα, χρησιμοποιώντας semi-join. Το ποιον από αυτούς θα διαλέξουμε, εξαρτάται από τη φύση της συγκεκριμένης βάσης δεδομένων. Πιο συγκεκριμένα η σύνδεση των σχέσεων *R* και *S* μπορεί να γίνει ως εξής [OV91]:

$$
\begin{aligned}
R \bowtie_{A=A} S \quad &\Leftrightarrow \quad (R \ltimes_{A=A} S) \bowtie_{A=A} S \\
&\Leftrightarrow \quad (R \ltimes_{A=A} \pi_A(S)) \bowtie_{A=A} S \\
&\Leftrightarrow \quad R \bowtie_{A=A} (S \ltimes_{A=A} R) \\
&\Leftrightarrow \quad (R \ltimes_{A=A} S) \bowtie_{A=A} (S \ltimes_{A=A} R)
\end{aligned}
$$

Ας πάρουμε για παράδειγμα τη δεύτερη εναλλακτική λύση. Παρατηρούμε κατ' αρχήν, ότι η σχέση *S* εμφανίζεται δύο φορές στη σχεσιακή έκφραση. Ας υποθέσουμε τώρα ότι τα *R* και *S* βρίσκονται σε δύο διαφορετικές τοποθεσίες. Μπορούμε να κάνουμε γρήγορα μια προβολή στο *S*, να στείλουμε την πληροφορία στο *R* και να εκτελέσουμε το semi-join της πρώτης παρένθεσης τοπικά στην τοποθεσία του *R*. Αν υποθέσουμε ότι το R είναι μια μεγάλη σε όγκο σχέση, το αποτέλεσμα αυτής της πράξης θα είναι μια σαφώς μικρότερη σχέση. Στη συνέχεια, μπορούμε να μεταδώσουμε το αποτέλεσμα αυτό στην τοποθεσία της *S*, όπου και θα γίνει η τελική σύνδεση. Στο σχήμα 2.8, φαίνεται το δέντρο σύνδεσης της εν λόγω πράξης. Σε κάθε ακμή, βάζουμε τη σειρά με την οποία εκτελείται και την τοποθεσία εκτέλεσης. Επιπλέον, έχουμε ενώσει τα δύο φύλλα στα οποία θα εμφανιζόταν κανονικά το *S*, σε ένα, γιατί έτσι αναδεικνύεται καλύτερα το πλεονέκτημα του semi-join.



Σχήμα 2.8. Το πλάνο εκτέλεσης της ερώτησης R ⋉ S, με τη χρήση semi-join

Στο σημείο αυτό, θα πρέπει να σημειώσουμε ότι το semi-join αναδεικνύεται πολύ περισσότερο, σαν η προτιμητέα λύση, στην περίπτωση που εμπλέκονται παραπάνω από μία συνδέσεις στην ερώτηση. Όμως, είναι πιθανό, κάποια από τα ενδιάμεσα αποτελέσματα (ειδικά στην περίπτωση που είναι πολύ μεγάλα) να καθυστερούν την αποτίμηση του αποτελέσματος. Αυτό μπορεί να συμβεί, καθώς οι σχέσεις στις διάφορες τοποθεσίες έχουν συνήθως ευρετήρια (indexes) τα οποία επιταχύνουν τις ερωτήσεις. Αντιθέτως, οι ενδιάμεσες σχέσεις που θα προκύψουν, δεν μπορούν να εκμεταλλευθούν τη δυνατότητα αυτή. Έτσι, αν και η λύση του semi-join είναι γενικά η προτιμητέα, πολλές φορές είναι πιθανό να μην είναι η βέλτιστη.

Παράδειγμα

Έστω οι σχέσεις `S(S#, SNAME)`, `SP(S#, P#, QTY)` και `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 ⋈ SP ⋈ P

Έστω ότι το κόστος μεταφοράς δεδομένων (το οποίο είναι και το κόστος που θα κρίνει το πλάνο εκτέλεσης) είναι ίσο με το μέγεθος των μεταφερόμενων δεδομένων (M). Θα εξετάσουμε στη συνέχεια τρεις διαφορετικές στρατηγικές αποτίμησης της ερώτησης.

Το κόστος χωρίς λειτουργίες semi-join.

Κατ' αρχήν ας υποθέσουμε ότι δεν χρησιμοποιούνται λειτουργίες semi-join, και θα ψάξουμε να βρούμε το κόστος μεταφοράς των δεδομένων. Από τις πληροφορίες που μας έχουν δοθεί, θα υπολογίσουμε πρώτα το μέγεθος της κάθε σχέσης S, SP και P. Έτσι, έχουμε τα εξής:

- 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)[3]. Άρα το μέγεθος κάθε εγγραφής θα 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*.

Άρα, 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.

Παρατηρούμε, όμως, ότι αν μεταφέρουμε όλες τις σχέσεις S, SP και P στον κόμβο T4 και εκτελέσουμε εκεί την ερώτηση, έχουμε το κέρδος ότι δε θα χρειαστεί να μεταφέρουμε τη λύση.

[3] Το κανονικό σχήμα του αποτελέσματος είναι R(S#, SNAME, P#, PNAME, S#, P#, QTY). Θεωρούμε όμως, ότι τα επαναλαμβανόμενα πεδία δεν ταλαμβάνουμε άμεσα υπόψη.

Το κόστος μεταφοράς των σχέσεων S, SP και P στον κόμβο Τ4, είναι 600+144+1200=*1944 bytes*.

Εύρεση κόστους αν υποθέσουμε ότι κάνουμε S ⋉ SP

Ας υποθέσουμε τώρα ότι αντί για κανονικές συνδέσεις επιλέγουμε να εκτελέσουμε την πράξη S ⋉ SP. Η ερώτηση θα εκτελεστεί ως εξής: κατ' αρχήν θα προβάλλουμε το S# από την SP στον κόμβο Τ2 και θα το στείλουμε στον Τ1. Εκεί θα εκτελέσουμε το semi-join το οποίο θα δημιουργήσει το αποτέλεσμα S'. Έπειτα, θα αποστείλουμε στον κόμβο Τ4, τα S', SP, P και θα εκτελέσουμε την ένωση. Χρήζει ιδιαίτερης προσοχής το γεγονός ότι το SP πρέπει να το στείλουμε πάλι στον Τ4, άσχετα από το αν έχει ήδη συμμετάσχει στο semi-join στον κόμβο Τ1. Αναλυτικά, έχουμε:

Πρώτα θα γίνει στον κόμβο Τ2, 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 Τ1, όπου βρίσκεται η σχέση S. Δηλαδή, μεταφέρονται 3∗4=12 bytes.

Στον κόμβο Τ1, γίνεται το join της σχέσης S και των τιμών S# που ήρθανε από τον κόμβο Τ2, δηλαδή γίνεται η πράξη: $S' = F \bowtie S$. Η σχέση S' έχει μέγεθος 3∗(4+96)=300 bytes.

Στη συνέχεια, μεταφέρεται η σχέση S από τον κόμβο Τ1 στον κόμβο Τ4, δηλαδή μεταφέρονται 300 bytes. Η αξία της κίνησης αυτής είναι ότι με το S ⋉ SP, αντί να μεταφερθεί από τον κόμβο Τ1 στον κόμβο Τ4 ολόκληρη η σχέση S, μεταφέρονται μόνο εκείνες οι εγγραφές που πραγματικά χρειάζονται και θα χρησιμοποιηθούν.

Κατόπιν, όπως προηγουμένως, μεταφέρεται η σχέση SP από τον κόμβο Τ2 στον κόμβο Τ4 (144 bytes) και επίσης μεταφέρεται η σχέση P από τον κόμβο Τ3 στον κόμβο Τ4 (1200 bytes).

Τέλος, στον κόμβο Τ4, πραγματοποιείται η ερώτηση $R = S' \bowtie SP \bowtie P$. Ως εκ τούτου, 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.

Κατά συνέπεια, το καλύτερο πλάνο εκτέλεσης της ερώτησης είναι το εξής:

i) Προβάλλουμε το πεδίο S# της SP στον κόμβο Τ2 και μεταφέρουμε τη σχέση $F=\Pi_{S\#}(SP)$, δηλαδή C1=12 bytes, στον κόμβο Τ1.
ii) Επίσης, προβάλλουμε το πεδίο P# της SP στον κόμβο Τ2 και μεταφέρουμε τη σχέση $G=\Pi_{P\#}(SP)$, δηλαδή C1'=12 bytes, στον κόμβο Τ3.
iii) Κάνουμε τη σύνδεση $F'=S \bowtie_{S\#=S\#} F$ στον κόμβο Τ1 και προκύπτει μια σχέση με μέγεθος : 3∗100=300 bytes.
iv) Επίσης, κάνουμε τη σύνδεση $G'=P \bowtie_{P\#=P\#} G$ στον κόμβο Τ3 και προκύπτει μια σχέση με μέγεθος : 3∗200=600 bytes.
v) Μεταφέρουμε τη σχέση SP από τον κόμβο Τ2 στον κόμβο Τ4, δηλαδή μεταφέρουμε C2=144 bytes.
vi) Μεταφέρουμε την F', δηλαδή C3=300 bytes, από τον κόμβο Τ2 και κάνουμε τη σύνδεση $F"=F' \bowtie_{S\#=S\#} SP$, στον κόμβο Τ4.

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*

2.5 ΔΙΑΧΕΙΡΙΣΗ ΔΟΣΟΛΗΨΙΩΝ

Στην ενότητα αυτή θα ασχοληθούμε με το πρόβλημα της διαχείρισης δοσοληψιών (transactions) στο περιβάλλον μιας κατανεμημένης βάσης δεδομένων. Θα παρουσιάσουμε μια ταξινόμηση των διαφορετικών αλγορίθμων καθώς και τους πιο δημοφιλείς από αυτούς. Πρώτα όμως, θα δώσουμε μερικούς ορισμούς (οι οποίοι δε διαφέρουν ιδιαίτερα από τους αντίστοιχους ορισμούς στις μη κατανεμημένες βάσεις δεδομένων).

Σύμφωνα με το [OV91] μια βάση δεδομένων είναι σε *συνεπή κατάσταση (consistent state)* εάν όλοι οι περιορισμοί ακεραιότητας που έχουν δηλωθεί για αυτήν πληρούνται. Οι αλλαγές στην κατάσταση μιας βάσης δεδομένων προέρχονται από εισαγωγές, διαγραφές και ενημερώσεις των δεδομένων. Όλες αυτές οι αλλαγές επιτελούνται δια μέσου του μηχανισμού των δοσοληψιών.

Σύμφωνα με το [Oz93], μια *δοσοληψία (transaction)* είναι "μια βασική, ατομική μονάδα από συνεπείς και αξιόπιστους υπολογισμούς, αποτελούμενη από μια αλληλουχία από ατομικές εκτελέσεις λειτουργιών". Στο [OV91] η δοσοληψία χαρακτηρίζεται ως "μια αλληλουχία πράξεων ανάγνωσης και εγγραφής στη βάση δεδομένων, μαζί με κάποια υπολογιστικά βήματα".

Κατά τη διάρκεια μιας δοσοληψίας, η βάση μπορεί προσωρινά να βρεθεί σε μη συνεπή κατάσταση. Στο τέλος της δοσοληψίας, όμως, πρέπει οπωσδήποτε να είναι συνεπής. Εν γένει, μια δοσοληψία πρέπει να χαρακτηρίζεται από τέσσερις βασικές ιδιότητες: *αυτονομία, συνέπεια, απομόνωση, διάρκεια*. Το σύνολο αυτών των ιδιοτήτων είναι γνωστό με το ακρωνύμιο *ACID* (Atomicity, Consistency, Isolation, Durability).

Η *αυτονομία (atomicity)* αναφέρεται στο γεγονός ότι μια δοσοληψία είναι μια αδιαίρετη μονάδα λειτουργίας. Είτε θα ολοκληρωθούν όλες οι ενέργειές της, είτε καμία από αυτές δε θα γίνει αποδεκτή.

Η *συνέπεια (consistency)* έχει δύο όψεις: τη *συνέπεια των δοσοληψιών* (transactions consistency) και τη *συνέπεια της παραλληλίας* (concurrency consistency). Συνέπεια των δοσοληψιών σημαίνει ότι μια δοσοληψία παίρνει μια βάση δεδομένων από μια συνεπή μορφή (όπου πληρούνται όλοι οι κανόνες αξιοπιστίας) και την μεταφέρει σε μια νέα μορφή, όπου η βάση εξακολουθεί να είναι συνεπής. Συνέπεια της παραλληλίας σημαίνει ότι υπάρχει η δυνατότητα συντονισμού όλων των δοσοληψιών που διενεργούνται ταυτόχρονα σε μία βάση δεδομένων, έτσι ώστε αυτή να παραμένει συνεπής.

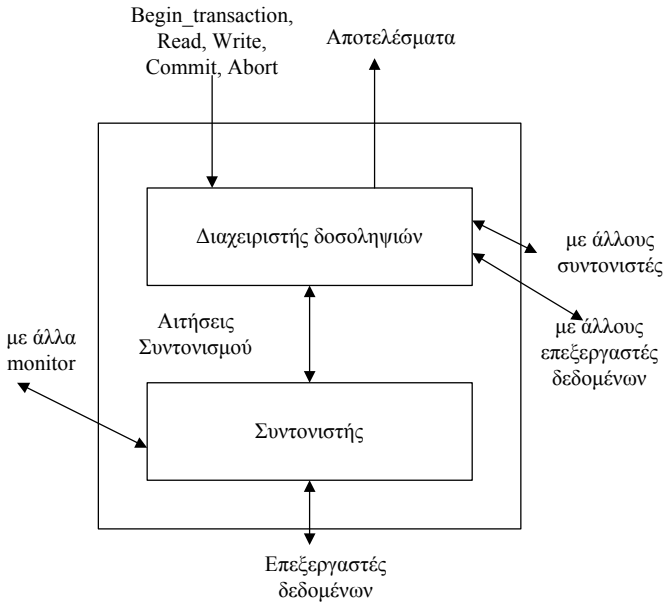
Η *απομόνωση (isolation)* είναι η ιδιότητα εκείνη που εγγυάται ότι μια δοσοληψία θα βλέπει πάντα μια συνεπή βάση δεδομένων. Ακόμα και αν υπάρχουν άλλες δοσοληψίες ταυτόχρονα με αυτή, μια δοσοληψία θα αγνοεί τα αποτελέσματά τους, σα να μην υπάρχουν. Ιδιαίτερο ρόλο παίζει εδώ, η έννοια της *σειριοποιησιμότητας* (serializability), η οποία ελέγχει αν το αποτέλεσμα των ενεργειών κάποιων δοσοληψιών που εκτελούνται συγχρόνως -για λόγους απόδοσης-, είναι το ίδιο με το αποτέλεσμα που θα είχαν, αν εκτελούνταν σειριακά.

Η *διάρκεια (durability)* είναι η ιδιότητα που εγγυάται ότι αν μια δοσοληψία ολοκληρωθεί επιτυχώς (φτάσει σε commit), τα αποτελέσματά της θα είναι διαρκή και δε θα χαθούν από τη βάση, ανεξάρτητα από το αν το σύστημα θα συνεχίσει να λειτουργεί ομαλά, ή θα βρεθεί σε κατάσταση μη ομαλής λειτουργίας.

2.5.1 Αφαιρετική αρχιτεκτονική για την κατανεμημένη εκτέλεση εφαρμογών στο πλαίσιο της διαχείρισης δοσοληψιών.

Στο Σχήμα 2.9 [OV91] φαίνεται μια αφαιρετική αρχιτεκτονική για τον τρόπο που διαχειριζόμαστε την εκτέλεση των εφαρμογών στο περιβάλλον μιας κατανεμημένης βάσης δεδομένων. Το βασικό εργαλείο σε αυτή την αρχιτεκτονική είναι το *monitor κατανεμημένης εκτέλεσης*. Το monitor κατανεμημένης εκτέλεσης αποτελείται από δύο κύρια τμήματα: το *διαχειριστή δοσοληψιών* (transaction manager) και τον *συντονιστή* (scheduler). Ο διαχειριστής δοσοληψιών είναι υπεύθυνος για το συγχρονισμό της εκτέλεσης των διαφόρων λειτουργιών στη βάση δεδομένων εκ μέρους μιας εφαρμογής. Ο συντονιστής είναι υπεύθυνος για την υλοποίηση ενός συγκεκριμένου αλγορίθμου συντονισμού δοσοληψιών για την πρόσβαση στη βάση δεδομένων. Ένα τρίτο κομμάτι -το οποίο δε φαίνεται στο σχήμα- είναι οι τοπικοί *διαχειριστές ανάνηψης* (recovery managers). Οι διαχειριστές ανάνηψης είναι υπεύθυνοι για την επαναφορά της βάσης σε συνεπή κατάσταση, μετά από μια αποτυχία.

Κάθε δοσοληψία ξεκινά από κάποια τοποθεσία. Υπεύθυνο για τη διαχείριση της δοσοληψίας είναι το τοπικό monitor κατανεμημένης εκτέλεσης, το οποίο ως εκ τούτου είναι και υπεύθυνο για την επικοινωνία με άλλα monitor κατανεμημένης εκτέλεσης, σε διαφορετικές τοποθεσίες, έτσι ώστε η κατανεμημένη δοσοληψία να έρθει εις πέρας.



Σχήμα 2.9 Μια αφαιρετική αρχιτεκτονική ενός monitor κατανεμημένης εκτέλεσης

2.5.2 Προβλήματα κατανεμημένων δοσοληψιών

Η *σειριοποιησιμότητα* (serializability), στον έλεγχο συντονισμού είναι η ιδιότητα μιας σειράς πράξεων, η οποία ελέγχει αν το αποτέλεσμα των ενεργειών κάποιων δοσοληψιών που εκτελούνται συγχρόνως -για λόγους απόδοσης-, είναι το ίδιο με το αποτέλεσμα που θα είχαν, αν εκτελούνταν σειριακά. Στην περίπτωση των κατανεμημένων βάσεων δεδομένων, η έννοια της σειριοποιησιμότητας επεκτείνεται σε σχέση με την προσέγγιση που είχαμε στις μη κατανεμημένες βάσεις δεδομένων.

Το *πλάνο εκτέλεσης (schedule)* με την οποία θα εκτελεστούν οι διάφορες πράξεις μπορεί να αντιμετωπιστεί είτε σε σχέση με τη σειρά με την οποία θα γίνει η εκτέλεση σε κάθε τοποθεσία (οπότε αναφερόμαστε στο *τοπικό πλάνο εκτέλεσης -local schedule*) είτε σε σχέση με το *καθολικό πλάνο εκτέλεσης* σε όλες τις τοποθεσίες *(global schedule)*. Στην περίπτωση που δεν υπάρχει αντιγραφή, το καθολικό πλάνο εκτέλεσης είναι το ίδιο με την ένωση όλων των τοπικών πλάνων. Στην περίπτωση που υπάρχουν επικαλύψεις, τα πράγματα γίνονται πιο πολύπλοκα. Είναι πιθανό να έχουμε τοπική σειριοποιησιμότητα χωρίς η καθολική σειριοποιησιμότητα να εξασφαλίζεται. Στο παρακάτω παράδειγμα, από το [OV91], το πρόβλημα αυτό φαίνεται πιο καθαρά

Έστω ότι έχουμε δύο τοποθεσίες, οι οποίες έχουν από ένα αντίγραφο *x* και ξεκινούν οι παρακάτω δοσοληψίες:

```
T1: Read(x)            T2: Read(x)
    x := x + 5             x := 10 * x
    Write(x)               Write(x)
    Commit                 Commit
```

Είναι εμφανές ότι και οι δύο δοσοληψίες πρέπει να εκτελεστούν και στις δύο τοποθεσίες. Έστω ότι έχουμε τα εξής τοπικά πλάνα εκτέλεσης:

S_1 = {$R_1(x)$, $W_1(x)$, C_1, $R_2(x)$, $W_2(x)$, C_2}
S_2 = {$R_2(x)$, $W_2(x)$, C_2, $R_1(x)$, $W_1(x)$, C_1}

Ενώ και τα δύο τοπικά πλάνα εκτέλεσης είναι σειριοποιήσιμα (όντα σειριακά), το καθολικό πλάνο εκτέλεσης είναι μη σειριοποιήσιμο, καθώς οι δύο δοσοληψίες εκτελούνται με την αντίστροφη σειρά σε κάθε τοποθεσία. Ας υποθέσουμε ότι η αρχική τιμή του *x* σε κάθε τοποθεσία είναι 1. Έτσι, ενώ στην πρώτη τοποθεσία τελικά θα έχει τιμή 60, στη δεύτερη θα έχει τιμή 15, με αποτέλεσμα την καταστροφή της συνέπειας της βάσης.

Η αμοιβαία συνέπεια όλων των τοπικών τμημάτων της κατανεμημένης βάσης μπορεί να επιτευχθεί πρακτικά, κρατώντας όλα τα τοπικά πλάνα σειριοποιήσιμα και εκτελώντας όλες τις κατανεμημένες δοσοληψίες με την ίδια σειρά. Τα πλάνα που μπορούν να ικανοποιήσουν αυτές τις ιδιότητες ονομάζονται *σειριοποιήσιμα σε ένα αντίγραφο (one-copy serializable)*.

Έστω ότι έχουμε ένα κομμάτι πληροφορίας *x* και κάποια αντίγραφα x_1, x_2, ..., x_n. Θα αναφερόμαστε στο *x* σαν το *λογικό δεδομένο (logical data)* και στα αντίγραφά του σαν τα *φυσικά δεδομένα (physical data)*. Ο στόχος είναι οι δοσοληψίες των χρηστών να αναφέρονται στα λογικά δεδομένα, ενώ τη διαχείριση των φυσικών δεδομένων να την αναλαμβάνει το πρωτόκολλο διαχείρισης αντιγράφων. Οι πιο συνηθισμένοι αλγόριθμοι, με τους οποίους και θα ασχοληθούμε εμείς, επικεντρώνουν στη δυνατότητα ανάγνωσης από ένα μόνο αντίγραφο και ανανέωσης όλων των αντιγράφων, γιαυτό και το πρωτόκολλο ονομάζεται *read once/write all (ROWA)*.

2.5.3 Ταξινόμηση των αλγορίθμων ελέγχου συντονισμού

Υπάρχουν διάφοροι τρόποι για την ταξινόμηση των αλγορίθμων ελέγχου συντονισμού. Ο πιο βασικός τρόπος είναι αυτός που κατατάσσει τους αλγόριθμους ανάλογα με τα μέσα που χρησιμοποιούν για το συντονισμό. Ένας άλλος -ορθογώνιος σε αυτόν- τρόπος ταξινόμησης, είναι η θεώρηση που κάνει ο αλγόριθμος για το μέλλον μιας δοσοληψίας: οι *αισιόδοξοι*

(optimistic) αλγόριθμοι θεωρούν ότι το πιο πιθανό είναι η δοσοληψία να τερματίσει επιτυχώς, χωρίς συγκρούσεις. Οι *απαισιόδοξοι (pessimistic)* αλγόριθμοι υποθέτουν το αντίθετο. Το αποτέλεσμα είναι ότι οι αισιόδοξοι αλγόριθμοι συγχρονίζουν τις δοσοληψίες στο τέλος τους, ενώ οι απαισιόδοξοι συγχρονίζουν τις δοσοληψίες πολύ νωρίς.

Σε ότι αφορά την ταξινόμηση με βάση τα μέσα με τα οποία επιτελείται ο συντονισμός, υπάρχουν δύο βασικές ομάδες αλγορίθμων: οι αλγόριθμοι που βασίζονται σε *κλειδώματα (lockings)*, και οι αλγόριθμοι που βασίζονται σε *χρονοσήμανση (timestamps)*. Η πράξη έδειξε ότι οι αλγόριθμοι που ανήκουν στην πρώτη ομάδα είναι πιο αποτελεσματικοί, με αποτέλεσμα να είναι και οι πλέον διαδεδομένοι. Έτσι και εμείς στη συνέχεια, θα ασχοληθούμε μόνο με αλγορίθμους κλειδωμάτων.

Στους αλγορίθμους που βασίζονται σε κλειδώματα, λοιπόν, ο συγχρονισμός των δοσοληψιών γίνεται με τη χρήση αμοιβαίου αποκλεισμού των δοσοληψιών, πάνω σε κομμάτια της βάσης δεδομένων. Στους αλγορίθμους αυτούς, ο συντονιστής αναφέρεται και ως *διαχειριστής κλειδωμάτων (lock manager)*. Υπάρχουν τρεις κύριοι αλγόριθμοι για τη διαχείριση των κλειδωμάτων σε μια κατανεμημένη βάση δεδομένων:

- Στην προσέγγιση του *κεντρικού κλειδώματος (centralized locking)*, μια από τις τοποθεσίες ορίζεται σαν η κύρια τοποθεσία όπου βρίσκεται η πληροφορία για τα κλειδώματα και είναι υπεύθυνη για την καθολική διαχείριση των δοσοληψιών.
- Στην προσέγγιση του *κλειδώματος του πρωτεύοντος αντιγράφου (primary copy locking)* ένα από τα αντίγραφα κάθε κλειδώματος, ορίζεται σαν το πρωτεύον αντίγραφο. Το αντίγραφο αυτό κλειδώνεται και όλες οι δοσοληψίες πρέπει να ζητήσουν κλειδί από το συντονιστή της τοποθεσίας του πρωτεύοντος αντιγράφου.
- Στην *αποκεντρωμένη* προσέγγιση *(decentralized locking)*, ο έλεγχος συντονισμού κατανέμεται σε όλους τους συντονιστές της βάσης δεδομένων. Κάθε συντονιστής είναι υπεύθυνος για τα τοπικά του αντίγραφα. Μια δοσοληψία, όμως, είναι υποχρεωμένη να κλειδώσει όλα τα αντίγραφα που υπάρχουν στην κατανεμημένη βάση.

2.5.4 Αλγόριθμοι κλειδωμάτων

Στις μη κατανεμημένες βάσεις δεδομένων ο κλασικός αλγόριθμος που χρησιμοποιείται για τα κλειδώματα είναι ο αλγόριθμος *κλειδώματος σε δύο φάσεις (2 Phase Locking - 2PL)*. Ο αλγόριθμος υποθέτει την ύπαρξη δύο ειδών κλειδώματος: *read-lock (Shared - S)* και *write-lock (eXclusive -X)*. Κάθε φορά που μια δοσοληψία θέλει να διαβάσει ένα αντικείμενο, αποκτά και ένα read-lock για το αντικείμενο αυτό. Αντίστοιχα, κάθε φορά που μία δοσοληψία θέλει να γράψει ένα αντικείμενο, αποκτά και ένα write-lock για το αντικείμενο αυτό. Μία δοσοληψία δεν μπορεί να αφήσει ένα κλείδωμα (lock) προτού είναι σίγουρη ότι δε θα χρειαστεί πλέον νέα κλειδώματα. Ο αλγόριθμος ονομάζεται 2 Phase Locking, γιατί μια δοσοληψία εκτυλίσσεται σε δύο χρονικές φάσεις: στην πρώτη φάση, η δοσοληψία αποκτά κλειδώματα και στη δεύτερη φάση, η δοσοληψία αφήνει τα κλειδώματα αυτά. Στην πράξη η δεύτερη φάση εκφυλίζεται στη στιγμιαία απόδοση όλων των κλειδωμάτων που έχει ανακτήσει η δοσοληψία.

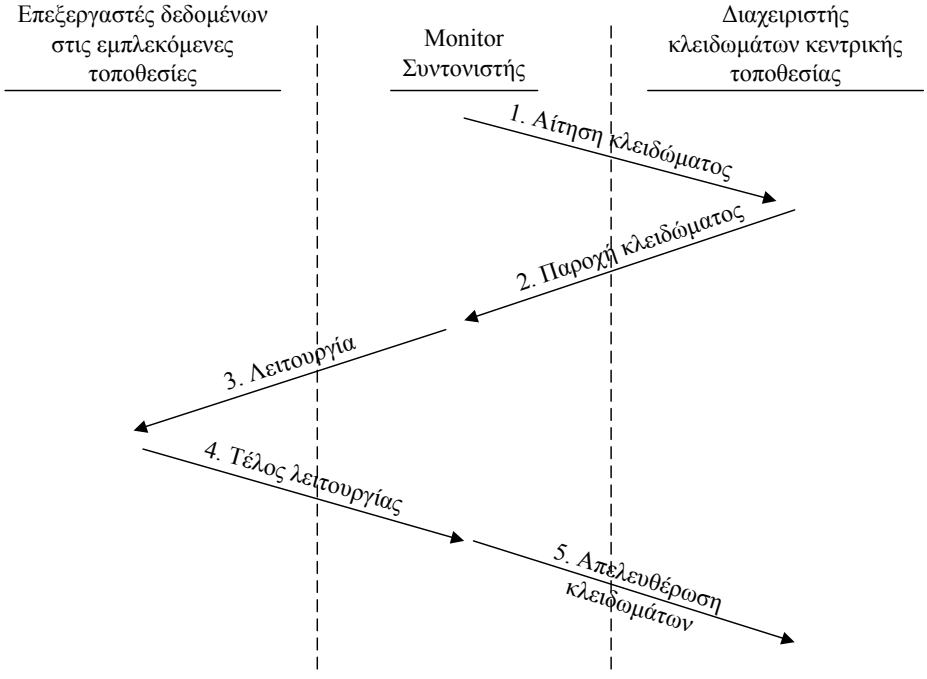
Ας υποθέσουμε ότι μία δοσοληψία T_1 έχει ένα κλείδωμα σε ένα αντικείμενο και μία δοσοληψία T_2 απαιτήσει και αυτή ένα κλείδωμα για το ίδιο αντικείμενο. Το αν θα δοθεί στην T_2 το κλείδωμα που αυτή ζήτησε, καθορίζεται από τον παρακάτω πίνακα (όπου στην οριζόντια στήλη είναι το κλείδωμα που μπορεί να είχε η T_1, στην κάθετη το κλείδωμα που ζήτησε η T_2, και οι τιμές του πίνακα δείχνουν αν επιτρέπεται η T_2 να πάρει τελικά το κλείδωμα που ζήτησε):

	S	**X**
S	Ναι	Όχι
X	Όχι	Όχι

Σε διάφορα εμπορικά συστήματα βάσεων δεδομένων, υπάρχει η έννοια του *φυσικού κλειδώματος* (physical locking) και του *λογικού κλειδώματος* (logical locking). Τα φυσικά κλειδώματα μπαίνουν στις σελίδες όπου περιέχονται τα αντικείμενα, ενώ τα λογικά κλειδώματα κλειδώνουν τα αναγνωριστικά των αντικειμένων. Κατά τη διάρκεια μιας δοσοληψίας, τα λογικά κλειδώματα είναι αυτά που η δοσοληψία κρατά μέχρι το τέλος της. Επειδή, όμως, τα φυσικά κλειδώματα είναι πολύ πιο γρήγορα στη διαχείρισή τους, ο αλγόριθμος που ακολουθούν τα περισσότερα συστήματα, ακολουθεί την εξής την τακτική: Όταν θέλουμε να κλειδώσουμε μια πλειάδα, πρώτα κλειδώνουμε τη σελίδα που το περιέχει με ένα φυσικό κλείδωμα. Μόλις προσπελασθεί η πλειάδα, τίθεται ένα λογικό κλείδωμα σ' αυτό και το φυσικό κλείδωμα ελευθερώνεται. Η δοσοληψία συνεχίζει χρησιμοποιώντας το φυσικό κλείδωμα.

Στις κατανεμημένες βάσεις δεδομένων, όπως ήδη προαναφέραμε, υπάρχουν τρεις βασικοί αλγόριθμοι κλειδώματος. Ο αλγόριθμος *κεντρικού κλειδώματος* επεκτείνει τον κλασικό 2PL αναθέτοντας τη διαχείριση των κατανεμημένων δοσοληψιών σε μία τοποθεσία μόνο. Αυτό σημαίνει ότι μόνο μια τοποθεσία έχει διαχειριστή κλειδωμάτων και οι διαχειριστές δοσοληψιών στις άλλες τοποθεσίες επικοινωνούν με αυτόν.

Στο Σχήμα 2.10 φαίνεται πιο αναλυτικά η διαδικασία εκτέλεσης μιας δοσοληψίας με βάση τον αλγόριθμο κεντρικού κλειδώματος. Ο αλγόριθμος που παρουσιάζεται εδώ αποτελεί μια γενική περιγραφή μιας ευρύτερης ομάδας αλγορίθμων (βλ. και [OV91]).



Σχήμα 2.10 Αλγόριθμος κεντρικού κλειδώματος

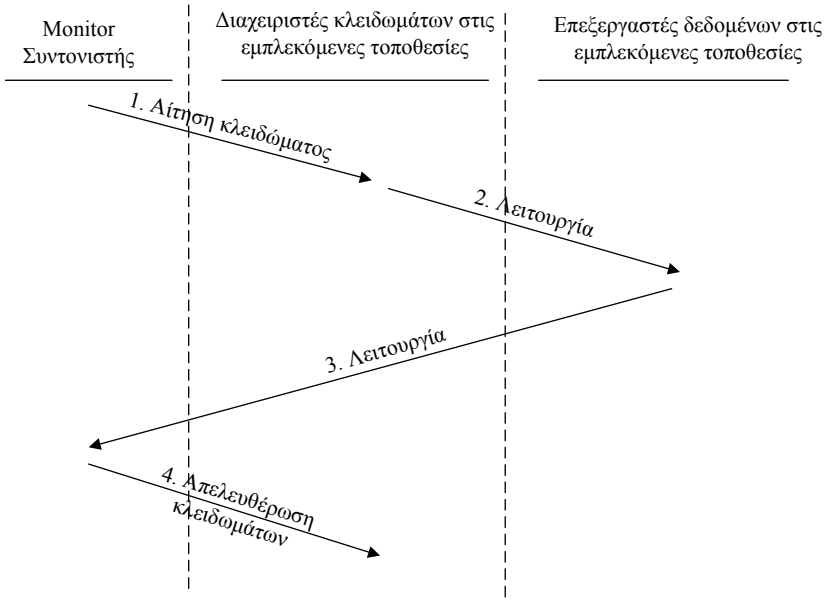
Ένα βασικό μειονέκτημα του αλγορίθμου κεντρικού κλειδώματος είναι ότι η κεντρική τοποθεσία μπορεί να μεταβληθεί στο bottleneck του όλου συστήματος, ιδιαίτερα σε συστήματα

με υψηλό ρυθμό δοσοληψιών. Επίσης, αν για κάποιο λόγο η κεντρική τοποθεσία δεν μπορεί να επικοινωνήσει με τις υπόλοιπες, τότε το σύστημα δεν μπορεί να λειτουργήσει.

Στον αλγόριθμο *κλειδώματος του πρωτεύοντος αντιγράφου*, έχουμε μια επέκταση του αλγορίθμου κεντρικού κλειδώματος. Βασικά επιτρέπουμε σε παραπάνω από μια τοποθεσίες να έχουν διαχειριστή κλειδωμάτων. Κάθε διαχειριστής κλειδωμάτων είναι υπεύθυνος για να διαχειρίζεται κλειδώματα για μια ομάδα από "οντότητες" (πλειάδες, πίνακες) που μπορούν να κλειδωθούν. Οι διαχειριστές δοσοληψιών επικοινωνούν κάθε φορά με τον κατάλληλο διαχειριστή κλειδωμάτων. Έτσι, για κάθε τμήμα πληροφορίας, μπορούμε να πούμε ότι έχουμε ένα πρωτεύον αντίγραφο.

Ο αλγόριθμος κλειδώματος του πρωτεύοντος αντιγράφου έχει το μειονέκτημα ότι απαιτεί επιπλέον μετα-πληροφορία για την περιγραφή της αρχιτεκτονικής κάθε βάσης. Παρ' όλα αυτά, το φορτίο στις διάφορες τοποθεσίες είναι λιγότερο απ' ότι στον αλγόριθμο κεντρικού κλειδώματος.

Τέλος, στον *κατανεμημένο 2PL* αλγόριθμο, θεωρούμε ότι έχουμε διαχειριστές κλειδωμάτων σε κάθε τοποθεσία. Κάθε συντονιστής είναι υπεύθυνος για τα αντίγραφα της πληροφορίας που βρίσκεται τοπικά. Μια δοσοληψία, όμως, είναι υποχρεωμένη να κλειδώσει όλα τα αντίγραφα που υπάρχουν στην κατανεμημένη βάση. Η βασική διαφορά του εν λόγω αλγορίθμου από τον αλγόριθμο κεντρικού κλειδώματος είναι ότι στην εκτέλεση μιας δοσοληψίας εμπλέκονται περισσότεροι του ενός διαχειριστές κλειδωμάτων. Αυτό, ναι μεν επιτρέπει την αποφυγή του bottleneck σε κάποια κεντρική τοποθεσία, αλλά δημιουργεί επιπλέον προβλήματα συγχρονισμού μεταξύ των διαχειριστών κλειδωμάτων, που θα εξετάσουμε στη συνέχεια. Στο Σχήμα 2.11 παρουσιάζεται ο κατανεμημένος 2PL αλγόριθμος.



Σχήμα 2.11 Κατανεμημένος 2PL αλγόριθμος

2.6 ΟΛΟΚΛΗΡΩΣΗ ΚΑΙ ΑΝΑΝΗΨΗ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΔΟΣΟΛΗΨΙΩΝ (COMMIT & RECOVERY OF DISTRIBUTED TRANSACTIONS)

Οι μηχανισμοί ανάνηψης έχουν επινοηθεί με σκοπό να μπορούν να επαναφέρουν τη βάση δεδομένων σε συνεπή κατάσταση μετά από κάποια αποτυχία του συστήματος. Στη συνέχεια

αυτής της παραγράφου θα εξετάσουμε τα διαφορετικά είδη αποτυχιών που υπάρχουν, τους τρόπους με τους οποίους οι αποτυχίες αντιμετωπίζονται στη γενική περίπτωση και τους συγκεκριμένους τρόπους με τους οποίους αντιμετωπίζουμε τις αποτυχίες στην ειδική περίπτωση των κατανεμημένων βάσεων δεδομένων.

2.6.1 Βασικά είδη αποτυχιών μιας βάσης δεδομένων

Οι βασικοί τρόποι με τους οποίους μπορεί να συμβεί μια αποτυχία σε μια βάση δεδομένων είναι οι εξής:

1. *Αποτυχία χωρίς απώλεια πληροφορίας*. Η αποτυχία αυτού του είδους απαντάται μάλλον συχνά και έχει να κάνει, για παράδειγμα, με την περίπτωση που μια δοσοληψία διακόπτεται απότομα (abort) εξ΄ αιτίας ενός απρόσμενου λάθους στα δεδομένα (divide by zero, για παράδειγμα). Στην περίπτωση της αποτυχίας αυτού του είδους, όλα τα δεδομένα για να αναιρέσουμε (rollback) τη δοσοληψία είναι διαθέσιμα στην κεντρική μνήμη.
2. *Αποτυχία με απώλεια κεντρικής μνήμης*. Στο είδος αυτό αποτυχίας, όλα τα δεδομένα της κεντρικής μνήμης χάνονται -όμως οτιδήποτε είναι αποθηκευμένο σε δευτερεύουσα μνήμη (π.χ. σκληρός δίσκος) παραμένει. Κλασικό παράδειγμα τέτοιου είδους αποτυχίας είναι όταν ένα σύστημα πέφτει. Στην περίπτωση αυτή, μπορούμε να αναιρέσουμε τη δοσοληψία με τα δεδομένα που διασώθηκαν στο σκληρό δίσκο.
3. *Αποτυχία με απώλεια δευτερεύουσας μνήμης*. Στην περίπτωση αυτή, χάνονται ακόμα και τα δεδομένα του σκληρού δίσκου. Οι αποτυχίες αυτές ονομάζονται και *αποτυχίες αποθηκευτικών μέσων (media failures)*. Κλασικό παράδειγμα τέτοιων αποτυχιών είναι οι περιπτώσεις που η κεφαλή του δίσκου βρίσκει στο δίσκο. Παρ' όλα αυτά, έχουμε μηχανισμούς για να αντιμετωπίζουμε και αυτού του είδους τις αποτυχίες.

2.6.2 Αντιμετώπιση αποτυχιών - Μηχανισμοί καταγραφής

Ο πιο κλασικός τρόπος που έχουμε για να αντιμετωπίζουμε αποτυχίες είναι η χρήση ενός *μηχανισμού καταγραφής (log)*. Ο μηχανισμός καταγραφής περιέχει την απαραίτητη πληροφορία για να εκτελέσουμε δύο βασικές πράξεις:

- να αναιρέσουμε (γνωστό και ως πράξη *undo*) τις επιπτώσεις όλων των δοσοληψιών που είχαν επηρεάσει τα δεδομένα που βρίσκονται αποθηκευμένα στη βάση δεδομένων, αλλά δεν πρόλαβαν να ολοκληρωθούν επιτυχώς (commit) πριν την αποτυχία, και
- να επαναλάβουμε (γνωστό και ως πράξη *redo*) όλες τις δοσοληψίες που ολοκληρώθηκαν επιτυχώς πριν την αποτυχία, αλλά τα αποτελέσματά τους δεν πρόλαβαν να αποθηκευτούν στη βάση δεδομένων (π.χ. γιατί χάσαμε τους buffers όπου αποθηκεύουμε προσωρινά το αποτέλεσμα μιας πράξης στη μνήμη).

Πρέπει να σημειώσουμε ότι η σειρά με την οποία εκτελούμε τις πράξεις αυτές είναι σημαντική: πρώτα αναιρούμε τις αποτυχημένες δοσοληψίες και μετά επαναλαμβάνουμε τις επιτυχημένες. Για να εκτελέσουμε λοιπόν, τις πράξεις αναίρεσης και επανάληψης πρέπει στο μηχανισμό καταγραφής να έχουμε αποθηκεύσει πληροφορίες για:

- τον κωδικό μιας δοσοληψίας
- τον κωδικό (tupleID) ενός record
- την πράξη η οποία επιτελέστηκε
- την παλιά και τη καινούρια τιμή του record
- τη χρονική στιγμή κατά την οποία επιτελέστηκε η πράξη

Ένα βασικό θέμα είναι ο χρονικός συνδυασμός της καταγραφής της πληροφορίας. Κάθε φορά που μια πράξη συντελείται, πρέπει αφενός να την πραγματοποιούμε και αφετέρου να

ενημερώνουμε το μηχανισμό καταγραφής. Ο συνήθης τρόπος για να το κάνουμε αυτό είναι να *ενημερώνουμε πρώτα το μηχανισμό καταγραφής και μετά να επιτελούμε τις διάφορες λειτουργίες στη βάση δεδομένων*. Ο μηχανισμός αυτός ονομάζεται *write-ahead log*, και σκοπό έχει να εξασφαλίσει την αξιοπιστία της βάσης στην περίπτωση που κάποια αποτυχία συμβεί μεταξύ της εγγραφής στη βάση και της εγγραφής στο μηχανισμό καταγραφής. Ο μηχανισμός του write-ahead log βασίζεται σε δύο βασικούς κανόνες:

- πριν από την ανανέωση κάποιας τιμής, τουλάχιστον το κομμάτι που έχει να κάνει με την πράξη αναίρεσης πρέπει να βρίσκεται αποθηκευμένο στο μηχανισμό καταγραφής.
- πριν από την επιτυχή ολοκλήρωση μιας δοσοληψίας, όλες οι σχετικές με τη δοσοληψία πληροφορίες πρέπει να έχουν καταγραφεί στο μηχανισμό καταγραφής.

Για να ξεκινήσουμε τη διαδικασία ανάνηψης της βάσης, χρησιμοποιούμε μια τεχνική που στηρίζεται σε *σημεία ελέγχου (checkpoints)* τα οποία συμβαίνουν σε συγκεκριμένες χρονικές στιγμές. Σε συγκεκριμένα (π.χ. περιοδικά) χρονικά διαστήματα, καταγράφουμε την πληροφορία που υπάρχει στη μνήμη, στη βάση δεδομένων και στο σκληρό δίσκο. Για την ακρίβεια:

- καταγράφουμε όλα τα records του μηχανισμού καταγραφής και περνάμε στη βάση όλες τις ανανεώσεις τιμών που βρίσκονται στους buffers
- καταγράφουμε στο μηχανισμό καταγραφής ότι το σημείο ελέγχου συνέβη τη δεδομένη χρονική στιγμή και ποιες δοσοληψίες είναι ενεργές.

Όταν χρειαστεί να γίνει ανάνηψη του συστήματος μετά από μια αποτυχία, οι πράξεις αναίρεσης και επανάληψης, ξεκινούν από το τελευταίο σημείο ελέγχου.

Δεδομένου ότι μια αποτυχία στη δευτερεύουσα μνήμη μας καταστρέφει το σύστημα εντελώς, πολλές φορές παίρνουμε αντίγραφα της βάσης και του log σε άλλα αποθηκευτικά μέσα. **Η διαδικασία αυτή του backup είναι από τις πλέον κρίσιμες για την ομαλή λειτουργία της βάσης δεδομένων**. Είναι επίσης σύνηθες να τοποθετούμε τη βάση δεδομένων σε διαφορετικό δίσκο από το μηχανισμό καταγραφής, ώστε να διασώσουμε τουλάχιστο το ένα από τα δύο σε περίπτωση καταστροφικής αποτυχίας.

2.6.3 Κατανεμημένες δοσοληψίες

Στην περίπτωση των κατανεμημένων δοσοληψιών υπάρχουν επιπλέον προβλήματα που μπορούν να επηρεάσουν την εξέλιξη μιας δοσοληψίας. Στη γενική περίπτωση, το πρωτόκολλο επικοινωνίας μεταξύ των διαφόρων monitors είναι το ίδιο με των μη κατανεμημένων συστημάτων και αποτελείται από το ακόλουθο σύνολο βασικών εντολών: *begin_transaction, read, write, abort, commit, recover*. Οι εντολές *begin_transaction, read* και *write* δεν παρουσιάζουν ιδιαίτερα προβλήματα και εκτελούνται ακολουθώντας τα πρωτόκολλα που περιγράψαμε σε προηγούμενη παράγραφο (π.χ. ROWA). Με τις υπόλοιπες εντολές, το κατανεμημένο περιβάλλον προσθέτει επιπλέον δυσκολίες, με τις οποίες και θα ασχοληθούμε στη συνέχεια. Προς το παρόν, θεωρούμε ότι υπάρχει μια εφαρμογή (κάποιο monitor, δηλαδή) η οποία ξεκινά μια δοσοληψία και είναι ο *συντονιστής (coodinator)* της εν λόγω δοσοληψίας. Ο συντονιστής επικοινωνεί με διάφορα monitors σε άλλες τοποθεσίες, τα οποία βοηθούν στην ολοκλήρωση της εν λόγω δοσοληψίας.

Υπάρχουν δύο βασικά πρωτόκολλα που θα μας απασχολήσουν: τα *πρωτόκολλα τερματισμού (termination)* και τα *πρωτόκολλα ανάνηψης*. Στα πρωτόκολλα τερματισμού, το ζητούμενο είναι να ολοκληρωθεί επιτυχώς, ή να αναιρεθεί μια δοσοληψία, με γενικό στόχο τη συνολική συνέπεια της κατανεμημένης βάσης δεδομένων. Στα πρωτόκολλα ανάνηψης, το ζητούμενο είναι πάλι η συνέπεια της βάσης, μέσα όμως από διαδικασίες επαναλειτουργίας της βάσης, μετά από κάποια αποτυχία.

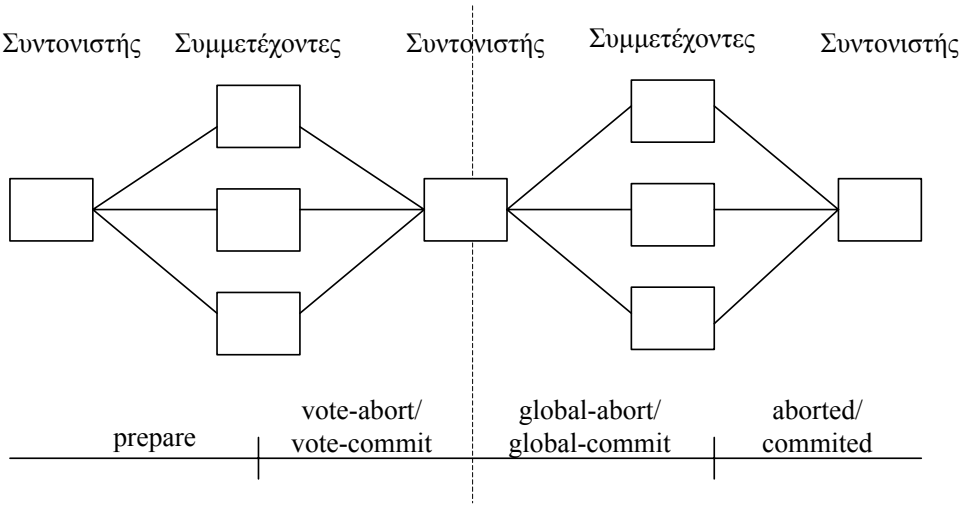
Πρωτόκολλα τερματισμού - Πρωτόκολλο 2 Phase Commit (2PC)

Εν γένει, από ένα πρωτόκολλο τερματισμού έχουμε τις παρακάτω απαιτήσεις:

- Το πρωτόκολλο πρέπει να διατηρεί την ατομικότητα των δοσοληψιών. Ακόμα και αν μια δοσοληψία εμπλέκει παραπάνω από μία τοποθεσίες, πρέπει, είτε να ολοκληρώνεται επιτυχώς συνολικά, είτε να αναιρείται συνολικά.
- Το πρωτόκολλο πρέπει να μπορεί να τερματίζει μια δοσοληψία, χωρίς να χρειάζεται να περιμένει την επάνοδο μιας τοποθεσίας που βρίσκεται εκτός λειτουργίας.

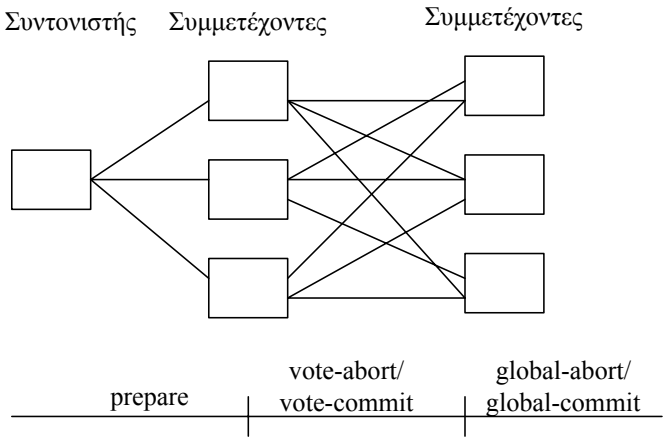
Εμείς στη συνέχεια, θα εξετάσουμε το *πρωτόκολλο ολοκλήρωσης σε δύο φάσης (2 Phase Commit - 2PC)* το οποίο είναι και το πλέον διαδεδομένο. *Το πρωτόκολλο 2PC στηρίζεται στην κεντρική ιδέα ότι όλες οι τοποθεσίες που εμπλέκονται σε μια κατανεμημένη δοσοληψία πρέπει να συμφωνήσουν στην επιτυχή ολοκλήρωσή της (commit) πριν τα αποτελέσματα της δοσοληψίας αποθηκευθούν διαρκώς στη βάση δεδομένων*. (Συνεπαγόμενο αυτής της κεντρικής ιδέας είναι το ότι *αν μία τοποθεσία αρνηθεί την επιτυχή ολοκλήρωση, όλη η δοσοληψία αναιρείται*). Ο αλγόριθμος που χρησιμοποιεί το πρωτόκολλο 2PC στην περίπτωση που δεν λαμβάνονται υπόψη αποτυχίες τοποθεσιών, είναι ο ακόλουθος:

Κατ' αρχήν, ο συντονιστής γράφει στο μηχανισμό καταγραφής ένα record με την ένδειξη "`begin_transaction`" (σηματοδοτώντας έτσι την έναρξη μιας δοσοληψίας). Στη συνέχεια, στέλνει ένα μήνυμα "`prepare`" στις υπόλοιπες τοποθεσίες και μπαίνει σε κατάσταση "`WAIT`". Όταν μια τοποθεσία δέχεται ένα μήνυμα "`prepare`" ελέγχει αν μπορεί να ολοκληρώσει επιτυχώς τη δοσοληψία. Στην περίπτωση που μπορεί, γράφει στο μηχανισμό καταγραφής της ένα record με την ένδειξη "`ready`", στέλνει ένα μήνυμα "`vote-commit`" στο συντονιστή και μπαίνει σε κατάσταση "`READY`". Στην περίπτωση που για οποιοδήποτε λόγο, μια τοποθεσία αδυνατεί να ολοκληρώσει επιτυχώς τη δοσοληψία, γράφει στο μηχανισμό καταγραφής της ένα record με ένδειξη "`ABORT`" και αποστέλλει ένα μήνυμα με ένδειξη "`vote-abort`" στο συντονιστή. Στην περίπτωση αυτή, η τοποθεσία παύει να ασχολείται πλέον με τη δοσοληψία, καθώς είναι σίγουρο ότι η δοσοληψία θα αναιρεθεί συνολικά. Όταν ο συντονιστής έχει λάβει τα μηνύματα από όλες τις εμπλεκόμενες τοποθεσίες αποφασίζει αν θα ολοκληρώσει επιτυχώς τη δοσοληψία. Η απόφαση αυτή είναι θετική, μόνο στην περίπτωση που όλες οι τοποθεσίες έχουν "ψηφίσει" υπέρ της κατάληξης αυτής. Στην περίπτωση έστω και μιας αρνητικής ψήφου, η δοσοληψία αναιρείται. Στην περίπτωση της επιτυχούς ολοκλήρωσης, ο συντονιστής γράφει ένα "`commit`" record στο μηχανισμό καταγραφής του, στέλνει ένα "`global-commit`" record σε όλες τις τοποθεσίες και μπαίνει σε κατάσταση "`COMMIT`". Στην αντίθετη περίπτωση, γράφει ένα "`abort`" record στο μηχανισμό καταγραφής του, αποστέλλει ένα μήνυμα "`global-abort`" στις εμπλεκόμενες τοποθεσίες και μπαίνει σε κατάσταση "`ABORT`". Οι τοποθεσίες είναι υποχρεωμένες να δράσουν ανάλογα με το μήνυμα του συντονιστή και να του επιβεβαιώσουν την πράξη τους. Όταν ο συντονιστής λάβει θετικές επιβεβαιώσεις από όλες τις τοποθεσίες γράφει στο μηχανισμό καταγραφής του ένα "`end-of-transaction`" record και η δοσοληψία τερματίζει. Ο μηχανισμός φαίνεται καλύτερα στο σχήμα 2.12.



Σχήμα 2.12 Πρωτόκολλο 2 Phase Commit - Κεντρικός 2PC

Ο παραπάνω αλγόριθμος ονομάζεται και *κεντρικός 2PC (centralized 2PC)* διότι η επικοινωνία γίνεται πάντα ανάμεσα στον συντονιστή και στις εμπλεκόμενες τοποθεσίες. Υπάρχουν διάφορες παραλλαγές αυτού του αλγορίθμου (βλ. και [OV91]) με πιο βασική αυτή του *κατανεμημένου 2PC (distributed 2PC)*. Στην περίπτωση αυτή, κάθε τοποθεσία, μόλις λάβει το μήνυμα "prepare" ενημερώνει όλες τις εμπλεκόμενες τοποθεσίες για την ψήφο της. Κάθε μία τοποθεσία περιμένει την ψήφο όλων των άλλων εμπλεκομένων τοποθεσιών και αν όλοι έχουν ψηφίσει θετικά, ολοκληρώνει επιτυχώς τη δοσοληψία ανεξάρτητα (ήτοι, χωρίς καμιά περαιτέρω επικοινωνία με τις άλλες τοποθεσίες).



Σχήμα 2.13 Κατανεμημένος 2PC

Ένα άλλο σημείο που πρέπει να τονίσουμε είναι ότι μέχρι στιγμής, υποθέσαμε ότι οι τοποθεσίες μπορούν να επικοινωνούν μεταξύ τους. Δεν ασχοληθήκαμε, δηλαδή, με την περίπτωση που μηνύματα χάνονται στο δίκτυο, τοποθεσίες "πέφτουν" ή η σύνδεση μεταξύ κάποιων τοποθεσιών χάνεται. Είναι απόλυτα σαφές ότι ο 2PC αλγόριθμος πάσχει σε αυτή την περίπτωση, καθώς βασίζεται στην ομοφωνία (ή αντίστοιχα στο veto) των τοποθεσιών για την

ολοκλήρωση μιας δοσοληψίας. Η αντιμετώπιση του εν λόγω προβλήματος μπορεί να γίνει με διάφορους τρόπους (βλ. και [OV91]), ο πιο βασικός εκ των οποίων είναι η χρήση μετρητών χρόνου και timeouts. Για παράδειγμα, μπορούμε να υποθέσουμε τις εξής αποτυχίες:

1. *Κάποιο μήνυμα χάνεται στο δίκτυο.* Στην περίπτωση που το μήνυμα απευθύνεται πρός το συντονιστή, αν λήξει το αντίστοιχο timeout, η δοσοληψία αναιρείται. Αν το μήνυμα απευθύνεται σε κάποιο συμμετέχοντα, ο συμμετέχων, άμα τη λήξη του timeout, μπορεί να ζητήσει επανάληψη της αποστολής του μηνύματος -ενίοτε δε, αυτή είναι και η πολιτική που μπορει να υιοθετήσει κανείς και για τον συντονιστή.
2. *Κάποιος συμμετέχων αποτυγχάνει πριν γράψει το "ready" record στο μηχανισμό καταγραφής του.* Αν το timeout του συντονιστή λήξει, η δοσοληψία αναιρείται. Μόλις ο συμμετέχων ανανήψει, απλά αναιρεί και αυτός τη δοσοληψία.
3. *Κάποιος συμμετέχων αποτυγχάνει αφού γράψει το "ready" record στο μηχανισμό καταγραφής του και αποστείλει το "READY" μήνυμα.* Σε αυτή την περίπτωση, οι υπόλοιποι τερματίζουν τη δοσοληψία χωρίς να τον λαμβάνουν υπόψη τους. Μόλις ο συμμετέχων ανανήψει, ζητά από το συντονιστή να μάθει το αποτέλεσμα της δοσοληψίας και ενεργεί και αυτός κατάλληλα.
4. *Ο συντονιστής αποτυγχάνει αφού έχει γράψει το "prepare" record αλλά πριν γράψει το "global_commit" record στο μηχανισμό καταγραφής του.* Οι συμμετέχοντες περιμένουν τον συντονιστή να ανανήψει. Μόλις ανανήψει, αυτός αποστέλει το "prepare" μήνυμα (πιθανώς ξανά) στις τοποθεσίες και η δοσοληψία συνεχίζει κανονικά. Αν μια τοποθεσία λάβει δύο συνεχόμενα μηνύματα "prepare", πρέπει να είναι σε θέση να καταλάβει τι συμβαίνει και να συνεχίσει κατάλληλα.
5. *Ο συντονιστής αποτυγχάνει αφού έχει γράψει το "global_commit" record στο μηχανισμό καταγραφής του, αλλά πριν γράψει το "end-of-transaction" record.* Το σενάριο και εδώ είναι το ίδιο με την προηγούμενη περίπτωση.
6. Το δίκτυο κόβεται κάπου σε δύο τμήματα. Τότε, σε ότι αφορά το συντονιστή, η κατάσταση είναι ίδια με τις περιπτώσεις 2 και 3, ενώ σε ότι αφορά τους συμμετέχοντες, η κατάσταση είναι ίδια με τις περιπτώσεις 4 και 5.

Πρωτόκολλα ανάνηψης

Εν γένει, από ένα πρωτόκολλο ανάνηψης έχουμε την απαίτηση της *ανεξαρτησίας*, ήτοι την ιδιότητα του τερματισμού μιας ενεργής δοσοληψίας στην περίπτωση μιας αποτυχίας, χωρίς την επικοινωνία με άλλες εμπλεκόμενες τοποθεσίες. Στη συνέχεια, θα ασχοληθούμε με την περίπτωση που το πρωτόκολλο τερματισμού είναι το 2PC. Θα κάνουμε τις εξής βασικές υποθέσεις:

- η πράξη εγγραφής στο μηχανισμό καταγραφής και αποστολής μηνύματος είναι ατομική (δεν υπάρχει περίπτωση, δηλαδή, το σύστημα να πέσει ανάμεσα στις δύο αυτές πράξεις) και
- τα μηνύματα δεν χάνονται ποτέ στο δίκτυο.

Η βασική ιδέα των αλγορίθμων ανάνηψης είναι η εξής: στο 2PC καμιά τοποθεσία δεν έχει δικαίωμα να αλλάξει την ψήφο της. Ως εκ τούτου, αν μια τοποθεσία αποτύχει, όλες οι υπόλοιπες περιμένουν μέχρι η τοποθεσία αυτή να ανανήψει (με το προφανές τίμημα του μπλοκαρίσματος των πόρων και των δεδομένων τους). Στην περίπτωση που η τοποθεσία αυτή ανανήψει (ή παρέλθει ένα κρίσιμο χρονικό διάστημα) ο έλεγχος περνά ξανά στο πρωτόκολλο τερματισμού. Για διάφορες παραλλαγές των αλγορίθμων αυτών μπορεί κανείς να αποταθεί στο [OV91].

ΕΥΧΑΡΙΣΤΙΕΣ

Ο συγγραφέας αισθάνεται την ανάγκη και την υποχρέωση να ευχαριστήσει τη Νάσσια Παπαγιάννη και τον Περικλή Ανδρίτσο, για τη βοήθειά τους στη συγγραφή του κεφαλαίου αυτού.

2.7 ΑΝΑΦΟΡΕΣ

[OV91] M.T. Ozsu, P. Valduriez. Principles of Distributed Database Systems. Prentice-Hall, 1991

[CP84] S. Ceri, G. Pelagatti. Distributed Databases -Principles and Systems. McGraw-Hill, 1984

[Oz93] T. Ozsu, "Transaction models and transaction management in object-oriented database systems", *NATO Advanced Study Institute, Object-Oriented Database System, Kusadasi Turkiye, August 1993*