Επεξεργασία Ερωτήσεων σε Ad
Δίκτυα Ομότιμων Πρακτόρω

Νικόλαος Φωλίνας

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕ

— ♦ —

Ιωάννινα, Μάρτιος 2006

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF IOANNINA

ΕΠΕΞΕΡΓΑΣΙΑ ΕΡΩΤΗΣΕΩΝ ΣΕ AD-HOC ΔΙΚΤΥΑ ΟΜΟΤΙΜΩΝ ΠΡΑΚΤΟΡΩΝ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από τον

Νικόλαο Φωλίνα

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΑ ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜΑΤΑ

Μάρτιος 2006

ΑΦΙΕΡΩΣΗ

Στη **Φλω** μου που...!

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω θερμά τον κύριο Παναγιώτη Βασιλειάδη για την τεράστια βοήθεια που μου έχει προσδώσει και τον απεριόριστο χρόνο που μου έχει αφιερώσει όλο αυτό το διάστημα. Θα ήθελα να τον ευχαριστήσω ιδιαίτερα και για το αίσθημα ασφάλειας που μου ενέπνεε απέναντι σε οποιαδήποτε δυσκολία συναντούσαμε.

Επίσης θα ήθελα να ευχαριστήσω την κυρία Ευαγγελία Πιτουρά, τον κύριο Απόστολο Ζάρρα και τον κύριο Ευάγγελο Παπαπέτρου για τις πολύτιμες συμβουλές τους σε κρίσιμα σημεία της εργασίας αυτής.

Ακόμη θα ήθελα να ευχαριστήσω και όλους όσους με βοήθησαν όλα αυτά τα χρόνια, ο καθένας με τον τρόπο του. Αυτοί ξέρουν! Όσο για μένα, χωρίς να εξαιρώ τους παραπάνω, προτιμώ αντί να τους γράψω σε μία κόλλα χαρτί να τους έχω βαθιά χαραγμένους στο μυαλό μου και την καρδιά μου.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΠΕΡΙΛΗΨΗ

Νικόλαος Φωλίνας του Χρήστου και της Παρασκευής. MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Μάρτιος, 2006. Επεξεργασία Ερωτήσεων σε Ad-Hoc Δίκτυο Ομότιμων Πρακτόρων. Επιβλέποντας: Παναγιώτης Βασιλειάδης.

Το πρόβλημα με το οποίο ασχολούμαστε στην εργασία αυτή είναι η επεξεργασία ερωτήσεων σε ad-hoc δίκτυα ομότιμων πρακτόρων. Σε ένα τέτοιο περιβάλλον κάθε ομότιμος πράκτορας διαθέτει μία βάση δεδομένων πάνω στην οποία επιθυμούμε να εκτελούμε ερωτήσεις. Η βάση δεδομένων αυτή, αποτελείται από πίνακες που είναι τοπικά αποθηκευμένοι και από πίνακες που είναι νοητοί ή υβριδικοί, δηλαδή πίνακες που περιέχουν δεδομένα που προέρχονται από τους υπόλοιπους ομότιμους πράκτορες που βρίσκονται στο δίκτυο εκείνη τη στιγμή. Λόγω αυτού του χαρακτηριστικού, η επεξεργασία ερωτήσεων δε μπορεί να πραγματοποιηθεί με τον παραδοσιακό τρόπο. Έτσι, αρχικά ορίζουμε αυστηρά το μοντέλο του συστήματος και αναζητούμε τις απαιτήσεις της εφαρμογής μας. Αναδεικνύοντας τις διαφορές σε σχέση με την παραδοσιακή SQL, ορίζουμε την SQL^P, μία γλώσσα που επεκτείνει την κλασσική SQL ώστε να υποστηρίζει τις απαιτήσεις του υπό εξέταση περιβάλλοντος. Επιπλέον ορίζεται αυστηρά η σημασιολογία της γλώσσας SQL^P, καθώς και το ακριβές συντακτικό της. Στη συνέχεια, παρουσιάζεται ένας αρχικός αλγόριθμος εκτέλεσης ερωτήσεων και δίνεται ο τυπικός ορισμός όλων των τελεστών που μπορεί να λάβουν μέρος σε κάποιο πλάνο εκτέλεσης ερώτησης. Για κάθε έναν από τους τελεστές αυτούς ορίζεται μία φόρμουλα κόστους, κατασκευάζοντας ένα ολοκληρωμένο μοντέλο κόστους για τα πλάνα ερώτησης. Τέλος, στα πλαίσια της εργασίας αυτής υλοποιήθηκε εφαρμογή σε Java που δίνει την δυνατότητα σε κάποιον χρήστη να θέτει μία ερώτηση σε γλώσσα SQL^P πάνω σε κάποιον ομότιμο πράκτορα του συστήματος και να παίρνει ως αποτέλεσμα ένα πλάνο εκτέλεσης της ερώτησης αυτής, το οποίο παρουσιάζεται ως γράφημα με τη βοήθεια του εργαλείου οπτικοποίησης γραφημάτων Yed.

EXTENDED ABSTRACT IN ENGLISH

Folinas, Nikolaos, C. P. MSc, Computer Science Department, University of Ioannina, Greece. March, 2006. Query Processing in Ad-Hoc Peer-to-Peer Networks. Thesis Supervisor: Panos Vassiliadis.

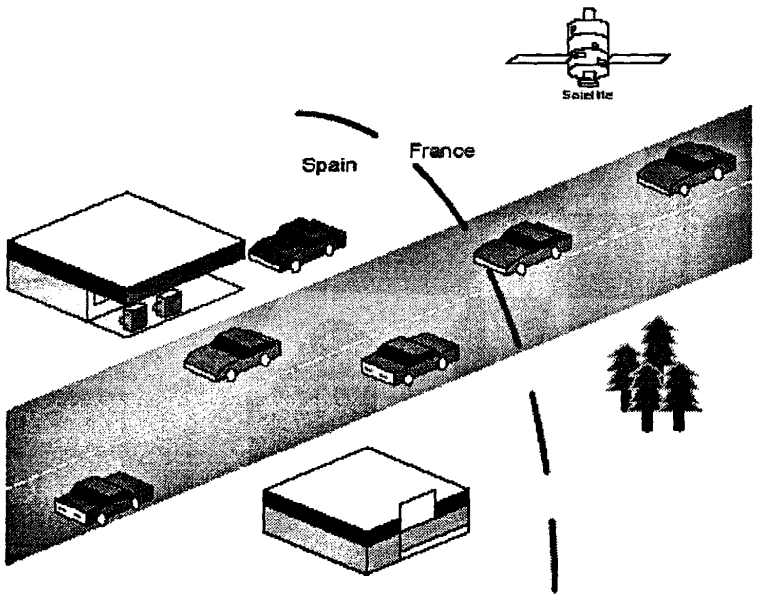
The topic of this thesis involves query processing in ad-hoc peer-to-peer networks. Each peer in such an environment has a database over which users want to execute queries. This database involves (a) relations which are locally stored and (b) relations which are virtual or hybrid. In the case of virtual relations, all the tuples of the relation are collected from peers that are present in the network at the time when the query is posed. Hybrid relations involve both locally stored tuples and tuples collected from the network. Due to this transitive nature of the extent of these relations, we cannot perform query processing in the traditional way. To deal with the aforementioned problem we provide the following contributions. Firstly, we formally define the system model. Next, we define SQL^P, an extension of traditional SQL on the basis of environment requirements that concern the termination of queries, the failure of individual peers and the semantic characteristics of the peers of the network. In addition, we precisely define the semantics of the language SQL^P, and its syntax in BNF, too. Moreover, we present an initial query execution algorithm as well as the typical definition of all the operators which can take place in a query execution plan. We define a cost formula for each of these operators, constructing a full cost model for query plans. Finally, we have implemented an application in Java, which (a) allows the user to write a query in SQL^P over the database of a peer of the network and (b) constructs the execution plan of this query. This execution plan is presented as a graph, with the help of Yed, a graph visualization tool.

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

Στην εποχή μας φαίνεται ότι το μέλλον των κατανεμημένων συστημάτων βρίσκεται στη γενική ιδέα του παγκόσμιου υπολογισμού (pervasive computing) [ZaVP05]. Το γεγονός αυτό οδηγεί στην βαθμιαία απομάκρυνση από τους σταθερούς σταθμούς εργασίας και την κατανομή της πληροφορίας και της επεξεργαστικής ισχύς στο περιβάλλον μέσα στο οποίο οι χρήστες τέτοιων συστημάτων μένουν και εργάζονται.

Θεωρείστε, για παράδειγμα, ότι οχήματα κινούνται σε ένα μεγάλο εθνικό δρόμο. Μπορεί να γίνει διαχωρισμός μεταξύ των διάφορων οχημάτων σε πολιτικά αυτοκίνητα, περιπολικά, φορτηγά και ασθενοφόρα. Κάθε όχημα μπορεί να παρέχει στους γύρω του δυναμικά μεταβαλλόμενες πληροφορίες οι οποίες αναφέρουν την τοποθεσία του οχήματος, την ταχύτητά του και τα αποθηκευμένα καύσιμά του. Επίσης κάθε όχημα μπορεί να έχει και στατικές πληροφορίες όπως ο τύπος του και διάφορα τεχνικά χαρακτηριστικά του. Πάνω στο δρόμο υπάρχουν έξοδοι σε περιοχές στάθμευσης στις οποίες μπορούμε να βρούμε βενζινάδικα, εστιατόρια, κέντρα πρώτων βοηθειών και εμπορικά καταστήματα. Κάθε ένα από αυτά μπορεί να παρέχει πληροφορίες οι οποίες ποικίλουν από πολύ απλές όπως αναφορά ύπαρξης κάποιων υπηρεσιών, ως περισσότερο πολύπλοκες οι οποίες παρέχουν πληροφορίες σχετικά με την τιμή κάποιων αγαθών, τη διαθεσιμότητα κάποιων αγαθών, το πλήθος των ασθενών που αναμένουν για ιατρικά περίθαλψη κ.τ.λ. Οι οδηγοί των οχημάτων έχουν τη δυνατότητα να χρησιμοποιήσουν-καλέσουν αυτές τις παρεχόμενες υπηρεσίες και να συλλέξουν την πληροφορία που τους ενδιαφέρει. Στη γενική περίπτωση, μπορεί

κανείς να φανταστεί ότι οι οδηγοί που βρίσκονται στα οχήματα, ή στα παρακείμενα σημεία στάθμευσης, θέτουν ερωτήσεις που αφορούν την κατάσταση του περιβάλλοντος γύρω τους. Οι ερωτήσεις αυτές για να απαντηθούν, χρειάζεται να συλλεχθούν πληροφορίες από όλους τους παρακείμενους παρόχους πληροφοριών, και να συνδυαστούν κατάλληλα.



Σχήμα 1.1: Παράδειγμα Αναφοράς

1.1 Αντικείμενο της Εργασίας

Το πρόβλημα το οποίο πραγματεύεται αυτή η εργασία είναι η επεξεργασία ερωτήσεων σε ad-hoc δίκτυα ομότιμων πρακτόρων. Υπάρχει ένα σύνολο από ομότιμους πράκτορες οι οποίοι επικοινωνούν μεταξύ τους σχηματίζοντας ένα ad-hoc δίκτυο. Η επικοινωνία, για λόγους διαλειτουργικότητας επιτυγχάνεται μέσω υπηρεσιών διαδικτύου (web services). Ο κάθε ομότιμος πράκτορας έχει μία τοπική βάση δεδομένων η οποία αποτελείται από δεδομένα είναι τοπικά αποθηκευμένα σε αυτήν, αλλά και από δεδομένα τα αφορούν τους άλλους ομότιμους πράκτορες του δικτύου. Τα δεδομένα που είναι αποθηκευμένα σε διαφορετικούς ομότιμους πράκτορες δεν είναι ομοιογενή μεταξύ τους. Επίσης ο κάθε ομότιμος πράκτορας του δικτύου συνοδεύεται από κάποια χαρακτηριστικά τα οποία δημοσιοποιούνται προς τους άλλους ομότιμους πράκτορες κι έτσι γίνονται γνωστά σε όλους τους ομότιμους

πράκτορες του συστήματος. Τα χαρακτηριστικά αυτά είναι η διαθεσιμότητά του, ο χρόνος απόκρισής του και η κλάση των ομότιμων πρακτόρων στην οποία ανήκει. Στα πλαίσια αυτού του περιβάλλοντος ζητούμενο είναι να έχει τη δυνατότητα κάθε ομότιμος πράκτορας να πραγματοποιήσει κάποια ερώτηση στην βάση δεδομένων του και να δέχεται ορθό αποτέλεσμα.

Η παραδοσιακή επεξεργασία ερωτήσεων σε βάσεις δεδομένων θεωρεί ότι όλα τα δεδομένα είναι τοπικώς αποθηκευμένα και διαθέσιμα κατά την επεξεργασία μιας ερώτησης. Ακόμα και η επεξεργασία ερωτήσεων σε κατανεμημένες βάσεις δεδομένων στηρίζεται στην ιδέα ότι υπάρχει ένας μικρός αριθμός τοποθεσιών, στις οποίες η πληροφορία είναι τοπικά αποθηκευμένη. Μέχρι στιγμής, δεν υπάρχει σχετική δουλειά στο αντικείμενο της επεξεργασίας παραδοσιακών ερωτήσεων σε βάσεις δεδομένων των οποίων η πληροφορία δεν είναι στατικά αποθηκευμένη σε κάποια τοποθεσία, αλλά μεταβάλλεται ως αποτέλεσμα της αλλαγής ενός υποκείμενου δικτύου ομότιμων πρακτόρων. Σε ότι αφορά το πρόβλημα της ετερογένειας των τοποθεσιών, στο παρελθόν έχει πραγματοποιηθεί σχετική εργασία έτσι ώστε να δίνεται η δυνατότητα σε κάποιον ομότιμο πράκτορα να πραγματοποιεί ερωτήσεις πάνω σε ετερογενείς βάσεις δεδομένων. Η αρχιτεκτονική που χρησιμοποιήθηκε στο παρελθόν είναι αυτή των mediator-wrappers. Βάσει αυτής της αρχιτεκτονικής, όταν υποβάλλεται μια ερώτηση, ο mediator είναι ο συντονιστής της διαδικασίας, ενώ οι wrappers αποτελούν το συστατικό μέσω του οποίου επικοινωνεί ο mediator με τις διάφορες ετερογενείς βάσεις δεδομένων. Εργασία του wrapper επίσης είναι η κατάλληλη μορφοποίηση της πληροφορίας των βάσεων δεδομένων, ώστε η πληροφορία που φτάνει στον mediator να είναι πάντα ομοιόμορφη. Πάνω σε αυτήν την αρχιτεκτονική στο [HKWY97] παρουσιάζεται ένας τρόπος παραγωγής πλάνων εκτέλεσης ερώτησης σε ένα τέτοιο περιβάλλον. Παρ' όλα αυτά, οι διαφορές της υπάρχουσας σχετικής εργασίας με το υπό εξέταση περιβάλλον, είναι σημαντικές και συγκεκριμένα, οι ακόλουθες.

- Στο υπό εξέταση περιβάλλον, δεν υπάρχει ένα καθολικό σχήμα μίας βάσης δεδομένων, ούτε κάποιος κεντρικός mediator, αλλά κάθε ομότιμος πράκτορας περιλαμβάνει την δική του τοπική βάση. Έτσι, κάθε φορά που ένας ομότιμος πράκτορας συλλέγει πληροφορίες από τους γύρω του, η εισερχόμενη

πληροφορία πρέπει να μετασχηματισθεί με βάση το τοπικό σχήμα της βάσης δεδομένων του συγκεκριμένου ομότιμου πράκτορα.

- Είναι πολύ βασικό να επισημανθεί ότι η συλλογή της πληροφορίας προκύπτει πάνω σε ένα ad hoc δίκτυο ομότιμων το οποίο μεταβάλλεται συνεχώς με το χρόνο. Η πληροφορία, λοιπόν, εξαρτάται από την κατάσταση του υποκείμενου δικτύου τη δεδομένη χρονική στιγμή που υποβάλλεται μια ερώτηση.
- Εκτός από τη μεταβλητότητα του υποκείμενου δικτύου ομότιμων, ο αριθμός των συμμετεχόντων παρόχων πληροφορίας σε μια ερώτηση μπορεί να είναι πολύ μεγάλος. Έτσι, πρέπει να οριστεί μία νέα γλώσσα ερωτήσεων που να λαμβάνει υπόψη της και την πιθανότητα κάποιος ομότιμος πράκτορας να μην παράσχει τη ζητηθείσα πληροφορία, αλλά και την ανάγκη η ερώτηση να περιοριστεί σε ένα πεπερασμένο πλήθος ομότιμων πρακτόρων, ενδεχομένως με ειδικά χαρακτηριστικά.
- Η γλώσσα αυτή ερωτήσεων πρέπει να περιλαμβάνει τουλάχιστον την παραδοσιακή SQL (ώστε οι τοπικές ερωτήσεις να εκτελούνται διαφανώς) -- σε αντιδιαστολή με το state-of-the-art σύστημα Clio [HKWY97] το οποίο περιορίζεται στην κατασκευή πλάνων για ερωτήσεις SPJ, δηλαδή ερωτήσεις του τύπου Select-Project-Join.

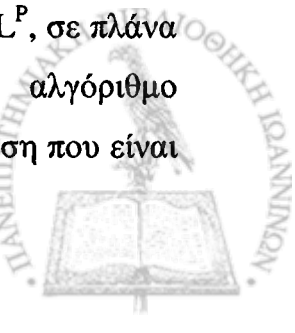
Στη συνέχεια ακολουθεί μία περιγραφή των κεντρικών σημείων της εργασίας μας.

Στην εργασία αυτή, ξεκινάμε με την θεωρητική θεμελίωση του προβλήματος. Προσεγγίζουμε το πρόβλημα γραφοθεωρητικά και κατασκευάζουμε ένα γράφο ομότιμων πρακτόρων, κάθε κόμβος του οποίου μοντελοποιεί έναν ομότιμο πράκτορα και κάθε ακμή μοντελοποιεί τη δυνατότητα επικοινωνίας μεταξύ των δύο ομότιμων πρακτόρων. Ο γράφος μεταβάλλεται σε σχέση με το χρόνο, καθώς ομότιμοι πράκτορες εισέρχονται και εξέρχονται από το σύστημα. Κάθε χρονική στιγμή, ένας ομότιμος πράκτορας γνωρίζει ένα υποσύνολο του γράφου, το οποίο ονομάζουμε *σύνολο κόμβων αναφοράς* του εν λόγω ομότιμου πράκτορα. Επιπλέον, οι ομότιμοι πράκτορες οργανώνονται σε *κοινότητες* (που αποτελούν συνεκτικούς υπογράφους κόμβων με σημασιολογική ομοιότητα) και *κλάσεις* (που αποτελούν διακριτά σύνολα κόμβων, κάθε ένα από τα οποία έχει παρόμοια δομή).

Ο κάθε ομότιμος πράκτορας του συστήματος λειτουργεί με δύο ρόλους, αυτόν του πελάτη (client) κι αυτόν του εξυπηρετητή (server). Ως πελάτης, ο ομότιμος πράκτορας έχει μία βάση δεδομένων η οποία αποτελείται από τριών κατηγοριών πίνακες, τους τοπικά αποθηκευμένους, τους νοητούς και τους υβριδικούς. Οι τοπικά αποθηκευμένοι πίνακες περιέχουν πληροφορία η οποία δεν μεταβάλλεται με το χρόνο και επιπλέον δεν αφορά τους υπόλοιπους ομότιμους πράκτορες του δικτύου. Οι νοητοί, είναι πίνακες που περιέχουν δεδομένα που προέρχονται από τους υπόλοιπους ομότιμους πράκτορες που ανήκουν στο σύστημα της στιγμή της υποβολής μίας ερώτησης. Οι υβριδικοί πίνακες συνδυάζουν τις δύο παραπάνω κατηγορίες πινάκων, δηλαδή έχουν εγγραφές που είναι τοπικά αποθηκευμένες, ενώ οι υπόλοιπες συλλέγονται δυναμικά από το δίκτυο, όπως και στους νοητούς πίνακες. Αν η ερώτηση αφορά νοητούς ή υβριδικούς πίνακες, ο ομότιμος πράκτορας λειτουργεί σαν πελάτης που πρέπει να συλλέξει δεδομένα από τους άλλους ομότιμους πράκτορες. Ως εξυπηρετητής από την άλλη, κάθε ομότιμος πράκτορας δημοσιοποιεί ένα σύνολο από web services, τις οποίες κάποιος άλλος ομότιμος πράκτορας του συστήματος μπορεί να καλέσει.

Λόγω της φύσης των πινάκων της βάσης δεδομένων ενός ομότιμου πράκτορα, η έννοια της ερώτησης στο περιβάλλον του συστήματός μας δεν είναι ίδια με αυτή στις παραδοσιακές βάσεις δεδομένων. Επειδή οι ομότιμοι πράκτορες όπως προαναφέραμε έχουν κάποια χαρακτηριστικά, πρέπει να μπορούμε να επιλέξουμε αυτούς που μας ενδιαφέρουν με βάση τις τιμές που έχουν στα χαρακτηριστικά αυτά. Επίσης θα πρέπει να αντιμετωπίζουμε το πρόβλημα κάποιος ομότιμος πράκτορας να μην απαντήσει, όπως επίσης και το ζήτημα η ερώτηση να τερματίζεται πάντα. Έτσι κατασκευάζουμε μία γλώσσα, λοιπόν ως επέκταση της παραδοσιακής SQL, η οποία να καλύπτει τις προαναφερθείσες απαιτήσεις για ερωτήσεις προς τη βάση δεδομένων ενός ομότιμου πράκτορα του συστήματος. Τέλος, πρέπει να είναι διαφανές στο χρήστη αν οι πίνακες τους οποίους χρησιμοποιεί σε μία ερώτηση είναι τοπικά αποθηκευμένοι ή όχι.

Το επόμενο βήμα είναι η αντιστοίχηση της γλώσσας που εισάγουμε, SQL^P, σε πλάνα μίας επεκτεταμένης σχεσιακής άλγεβρας. Κατασκευάζουμε έναν αλγόριθμο παραγωγής πλάνων εκτέλεσης ερωτήσεων, έχοντας ως είσοδο μία ερώτηση που είναι



γραμμένη στην γλώσσα SQL^P. Ο αλγόριθμος αυτός γεννά την ανάγκη ορισμού ενός συνόλου επιπρόσθετων τελεστών οι οποίοι υποστηρίζουν τις ενέργειες που αναφέρονται κατά τα βήματα του αλγορίθμου κατασκευής πλάνου εκτέλεσης ερώτησης. Με βάση τον αλγόριθμο αυτό κατασκευάζουμε πλάνα ερώτησης σύμφωνα με τον παραδοσιακό τρόπο, με τη διαφορά ότι κάθε νοητός ή υβριδικός πίνακας αποτελεί τη ρίζα ενός υπόδεντρου, το οποίο αντιστοιχεί στη διαδικασία που ακολουθείται για την τροφοδότηση του πίνακα αυτού με πλειάδες από τους ομότιμους πράκτορες που ανακαλύπτουμε ότι ανήκουν στο σύνολο κόμβων ενδιαφέροντος για τη δεδομένη ερώτηση. Η διαδικασία αυτή αποτελείται από την κλήση κατάλληλων web services των ομότιμων πρακτόρων αυτών, τα αποτελέσματα των οποίων μετασχηματίζονται με κατάλληλο τρόπο έτσι ώστε να εισαχθούν με τη μορφή πλειάδων στον αντίστοιχο νοητό ή υβριδικό πίνακα. Επίσης παρουσιάζουμε έναν εναλλακτικό τρόπο εκτέλεσης μίας ερώτησης, δίνοντας τη δυνατότητα σε κάποιον ερωτώντα ομότιμο πράκτορα του συστήματος να ζητήσει τον κατάλογο κάποιου άλλου ομότιμου πράκτορα ή να ζητήσει από κάποιον άλλο ομότιμο πράκτορα να αποτελέσει αντιπρόσωπό του, δηλαδή ο δεύτερος ομότιμος πράκτορας να εκτελέσει την ερώτηση του πρώτου για λογαριασμό του.

Ακόμη ορίζουμε ένα μοντέλο κόστους που παρουσιάζει το κόστος που αντιστοιχεί κατά την εφαρμογή καθενός από τους τελεστές που μπορεί να λάβουν μέρος σε μία ερώτηση. Με βάση το μοντέλο κόστους αυτό, μας δίνεται η δυνατότητα να υπολογίζουμε το συνολικό εκτιμούμενο κόστος ενός πλάνου αθροίζοντας τα επιμέρους κόστη των τελεστών που το αποτελούν.

Στα πλαίσια της παρούσας εργασίας, έχουμε υλοποιήσει και μια εφαρμογή που δίνει τη δυνατότητα σε κάποιον χρήστη να θέτει κάποια ερώτηση σε SQL^P και να έχει ως αποτέλεσμα ένα πλάνο εκτέλεσης της ερώτησης αυτής. Η υλοποίηση έχει πραγματοποιηθεί σε γλώσσα Java κι έχει ως αποτέλεσμα-έξοδο ένα αρχείο σε γλώσσα GML η οποία είναι μία εξειδικευμένη γλώσσα περιγραφής γραφημάτων. Το αρχείο αυτό χρησιμοποιείται ως είσοδος σε ένα εργαλείο, το Yed, το οποίο με είσοδο ένα αρχείο σε τέτοια γλώσσα πραγματοποιεί την αντίστοιχη οπτικοποίησή του. Έτσι συνολικά, θέτοντας κάποιος χρήστης μία ερώτηση στην SQL^P, παίρνει ως αποτέλεσμα το δέντρο εκτέλεσης της ερώτησης αυτής οπτικά-ως γράφημα. Επίσης

έχει κατασκευαστεί και μία διεπαφή της εφαρμογής αυτής έτσι ώστε να διευκολύνεται η διαχείρισή της.

Πιο συνοπτικά τα κεντρικά σημεία της εργασίας είναι τα εξής:

- Ορισμός του μοντέλου του συστήματος και των απαιτήσεων του περιβάλλοντος.
- Επέκταση της παραδοσιακής SQL και ορισμός της γλώσσας SQL^P, έτσι ώστε να υποστηρίζει τις απαιτήσεις του περιβάλλοντος.
- Ορισμός αλγορίθμου εκτέλεσης ερώτησης σε γλώσσα SQL^P και ορισμός τελεστών που τον υποστηρίζουν.
- Ορισμός μοντέλου κόστους των τελεστών που λαμβάνουν μέρος στο πλάνο εκτέλεσης μίας ερώτησης.
- Υλοποίηση εφαρμογής σε Java και οπτικοποίηση του αποτελέσματος της ερώτησης-πλάνου εκτέλεσης της ερώτησης με τη βοήθεια του εργαλείου Yed.

1.2 Δομή της Διατριβής

Στο κεφάλαιο 2 παρουσιάζεται η σχετική εργασία με ανάλυση του κάθε άρθρου. Τα άρθρα τα οποία αναλύονται ομαδοποιούνται σε κάποιες κατηγορίες ανάλογα με το περιεχόμενό τους. Οι κατηγορίες των άρθρων του κεφαλαίου 2 είναι οι εξής: context, context-web services, web services, ετερογενή συστήματα βάσεων δεδομένων, επεξεργασία/βελτιστοποίηση ερωτήσεων, benchmarks και ταίριασμα σχημάτων.

Στο κεφάλαιο 3 παρατίθεται η επέκταση της παραδοσιακής SQL έτσι ώστε να υποστηρίζει τις απαιτήσεις της εφαρμογής μας. Στην ενότητα 3.1 πραγματοποιείται μία τυπική περιγραφή του συστήματος. Στην ενότητα 3.2 αναλύονται οι διαφορές με την παραδοσιακή SQL, ενώ στην 3.3 παρουσιάζονται οι απαιτήσεις για την επέκταση της SQL. Στην ενότητα 3.4 δίνεται το γενικό μοντέλο ερώτησης στην γλώσσα που εισάγουμε και στην ενότητα 3.5 δίνονται οι επεκτάσεις των εντολών CREATE TABLE, INSERT, DELETE, UPDATE.

Στο κεφάλαιο 4 γίνεται η αντιστοίχηση της επεκτεταμένης SQL που εισάγουμε σε πλάνα σχεσιακής άλγεβρας. Αρχικά στην ενότητα 4.1 δίνεται η αρχιτεκτονική του

συστήματός μας. Στην ενότητα 4.2 αναλύεται ο τρόπος εκτέλεσης ερωτήσεων σε συστήματα ετερογενών βάσεων δεδομένων. Ο τυπικός ορισμός των τελεστών που λαμβάνουν μέρος στα πλάνα εκτέλεσης της ερώτησης πραγματοποιείται στην ενότητα 4.3. Στην ενότητα 4.4 αναλύεται ο τρόπος εκτέλεσης μία ερώτησης σε γλώσσα SQL^P και στην ενότητα 4.5 παρουσιάζεται ένας εναλλακτικός τρόπος εκτέλεσης μίας ερώτησης.

Στη συνέχεια, στο κεφάλαιο 5, αναλύεται το μοντέλο κόστους για τους διάφορους τελεστές που λαμβάνουν μέρος στα πλάνα εκτέλεσης ερώτησης. Στην ενότητα 5.1, παρουσιάζεται το κόστος των τελεστών που αντιστοιχούν σε πράξεις της παραδοσιακής σχεσιακής άλγεβρας. Στην ενότητα 5.2, παρουσιάζεται το κόστος των τελεστών που εφαρμόζονται στον κατάλογο του ερωτών peer, ενώ στην ενότητα 5.3 παρουσιάζεται το κόστος των τελεστών που εισάγουμε και λαμβάνουν μέρος στα πλάνα εκτέλεσης ερωτήσεων.

Στο κεφάλαιο 6 παρουσιάζεται η υλοποίηση της εφαρμογής μας. Στην ενότητα 6.1 αναλύονται τα βήματα που ακολουθήσαμε κατά την υλοποίησή της, ενώ στην ενότητα 6.2 παρουσιάζεται ο τρόπος διαχείρισης της διεπαφής της εφαρμογής.

Τέλος στο κεφάλαιο 7 παρουσιάζονται τα συμπεράσματα της εργασίας και η μελλοντική εργασία.

ΚΕΦΑΛΑΙΟ 2. ΣΧΕΤΙΚΗ ΕΡΓΑΣΙΑ

Τα άρθρα που θα αναλυθούν παρακάτω, ομαδοποιούνται σε κάποιες κατηγορίες. Οι κατηγορίες αυτές είναι οι εξής: άρθρα σχετικά με context, άρθρα σχετικά με context και με web services, άρθρα σχετικά μόνο με web services, άρθρα σχετικά με ετερογενή συστήματα βάσεων δεδομένων (αρχιτεκτονική mediator-wrappers), άρθρα σχετικά με επεξεργασία/ βελτιστοποίηση ερωτήσεων, άρθρα σχετικά με benchmarks και άρθρα σχετικά με ταίριασμα σχημάτων.

2.1 Context

Στο [Dey01] παρουσιάζεται ένας ορισμός του context, ένας ορισμός του context-aware υπολογισμού, τα χαρακτηριστικά που πρέπει να υποστηρίζει μία context-aware εφαρμογή και τα χαρακτηριστικά που περιέχονται στην Context Toolkit η οποία είναι μία αρχιτεκτονική που έχει ως σκοπό την υποστήριξη context-aware εφαρμογών. Ως context ορίζεται κάθε πληροφορία η οποία μπορεί να χρησιμοποιηθεί ώστε να χαρακτηρίσει μία οντότητα. Μία οντότητα είναι ένα πρόσωπο, ένα μέρος, ή αντικείμενο που είναι σχετικό με την αλληλεπίδραση μεταξύ χρήστη-εφαρμογής συμπεριλαμβανομένου του χρήστη και της εφαρμογής. Ένα σύστημα ορίζεται ως context-aware αν αυτό χρησιμοποιεί context ώστε να παρέχει σχετική πληροφορία

και/ή υπηρεσίες στον χρήστη, όπου η σχετικότητα έχει να κάνει με την εργασία του χρήστη. Μία context-aware εφαρμογή θα πρέπει να υποστηρίζει τρεις κατηγορίες από χαρακτηριστικά: παρουσίαση πληροφορίας και υπηρεσιών στον χρήστη, αυτόματη εκτέλεση μίας υπηρεσίας για έναν χρήστη και tagging του context σε πληροφορία για μελλοντική ανάκτηση. Το σύστημα Context Toolkit υποστηρίζει τα συνήθη χαρακτηριστικά που απαιτούνται από context-aware εφαρμογές: συλλογή και προσπέλαση του context, αποθήκευση, κατανομή και ανεξάρτητη εκτέλεση από τις εφαρμογές. Η αρχιτεκτονική αυτή παρέχει τρεις abstractions: τα widgets, τους μεταγλωττιστές και τους aggregators. Επίσης εισάγεται η situation abstraction ένα επίπεδο πάνω από τις τρεις προαναφερθείσες.

Στο [DeAb99] παρουσιάζονται διάφοροι ορισμοί που έχουν κατά καιρούς δοθεί για το context και την context-awareness. Προτάσσονται ορισμοί για τα θέματα αυτά και πραγματοποιείται επίσης μία προσπάθεια για κατηγοριοποίηση του context και των χαρακτηριστικών για context-aware εφαρμογές. Η κατηγοριοποίηση αυτή έχει ως σκοπό να γίνει μία μοντελοποίηση του context και των χαρακτηριστικών που πρέπει να υποστηρίζουν οι context-aware εφαρμογές ώστε να διευκολύνονται σε μεγάλο βαθμό ζητήματα που έχουν να κάνουν με την εύρεση του context, αναγνώριση αν μία εφαρμογή είναι context-aware, σχεδιασμό και ανάπτυξη μίας context-aware εφαρμογής.

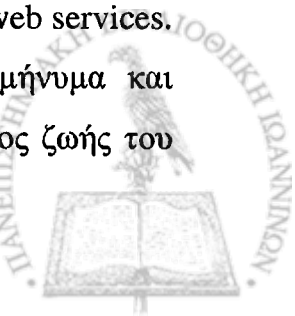
Στο [ChKo00] παρουσιάζονται επίσης διάφοροι ορισμοί του context που έχουν δοθεί κατά καιρούς. Προτάσσεται ένας ορισμός για το context, σύμφωνα με τον οποίο context είναι το σύνολο των καταστάσεων και των ρυθμίσεων του περιβάλλοντος που είτε καθορίζουν τη συμπεριφορά μίας εφαρμογής είτε μέσα στο οποίο ένα γεγονός συμβαίνει και ενδιαφέρει τον χρήστη. Το ίδιο ακριβώς ακολουθείται και για το θέμα του context-aware υπολογισμού και προτείνονται δύο ορισμοί σύμφωνα με τους οποίους ενεργητική context-awareness είναι όταν μία εφαρμογή αυτόματα προσαρμόζεται για την εύρεση context αλλάζοντας την συμπεριφορά της εφαρμογής, ενώ παθητική context-awareness είναι όταν μία εφαρμογή παρουσιάζει το νέο ή τροποποιημένο context σε έναν ενδιαφερόμενο χρήστη ή αποθηκεύει context για τον χρήστη για μελλοντική ανάκτηση. Στη συνέχεια παρουσιάζεται ένα σύνολο από γνωστές context-aware εφαρμογές. Ακόμη αναλύονται θέματα σχετικά με την

ανάκτηση του context και τυχουσών αλλαγών σε αυτό, όπως επίσης και κάποιοι τρόποι για μοντελοποίησή του. Τέλος δίνονται δύο αρχιτεκτονικές μία κεντρικοποιημένη και μία κατανεμημένη για ένα middleware επίπεδο μεταξύ της επεξεργασίας χαμηλού επιπέδου δεδομένων που γίνονται sensing και υψηλού επιπέδου εφαρμογές.

Στο [GrSa01] ως context ορίζεται τυπικά να είναι η τοποθεσία, ταυτότητα, κατάσταση ανθρώπων, ομάδων και υπολογιστικών και φυσικών αντικειμένων. Ως αισθητό (sensed) context ορίζεται το context που έχει να κάνει με το φυσικό περιβάλλον ή εναλλακτικά είναι οι ιδιότητες των φαινομένων. Ως φαινόμενο ορίζεται με τη σειρά του ένα συμβάν, ένα περιστατικό το οποίο είναι αισθητό από τις αισθήσεις. Στη συνέχεια πραγματοποιείται μοντελοποίηση του sensed context. Το sensed context εμφανίζεται στη φύση με τη μορφή «Το φαινόμενο Α έχει την ιδιότητα α». Τις sensed context πληροφορίες μπορούμε να τις μορφοποιήσουμε και να οδηγηθούμε σε πολύπλοκες εκφράσεις sensed context συνδέοντάς τες και με μετα-προτασιακές ιδιότητες. Στο μοντέλο sensed context αξιοποιούνται χαρακτηριστικά όπως το περιεχόμενο πληροφορίας και οι μετα-πληροφορίες. Το περιεχόμενο πληροφορίας σχετίζεται με τις sensed ιδιότητες των φαινομένων και με θέματα του sensing, ενώ οι μετα-πληροφορίες σχετίζονται με πληροφορίες ποιοτικών γνωρισμάτων και πληροφορίες σχετικά με τον πόρο του περιεχομένου. Πραγματοποιείται αξιοποίηση της context πληροφορίας και της μετα-πληροφορίας για το σχεδιασμό και ανάπτυξη εφαρμογών. Αυτό γίνεται πάνω σε μία κλασσική context-aware εφαρμογή, έναν οδηγό η ξενάγησης σε μουσείο. Τέλος παρουσιάζεται μία αρχιτεκτονική η οποία βασίζεται στη χρήση ενός συνόλου από συναρτησιακά χαρακτηριστικά, τα οποία οργανώνονται σε δύο επίπεδα: επίπεδο ανάκτησης context, επίπεδο μετατροπής context. Η αρχιτεκτονική αυτή χρησιμοποιείται στο παράδειγμα της εφαρμογής ξενάγησης σε μουσείο.

2.2 Context & Web Services

Στο [KeKe04] παρουσιάζεται ένας τρόπος να εντάξουμε το context στις web services. Το context μεταφέρεται σε μία web service μέσα σε ένα SOAP μήνυμα και συγκεκριμένα στο header block αυτού. Στη συνέχεια αναλύεται ο κύκλος ζωής του



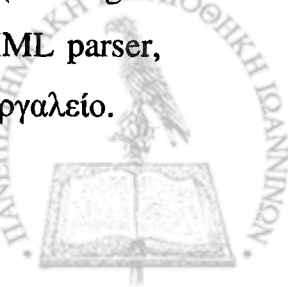
context, το οποίο ξεκινώντας από έναν client και αφού αυτός κάνει μία αίτηση σε μία web service και πάρει την απαιτούμενη απάντηση, ξαναγυρίζει πάλι σε αυτόν με κάποιες πιθανές τροποποιήσεις. Παρουσιάζονται επίσης δύο τρόποι επεξεργασίας του context, η απόλυτη (implicit) και η αυτόματη. Κατά την αυτόματη πραγματοποιείται επεξεργασία του context σε τέσσερις πιθανές περιπτώσεις και υπάρχουν αντίστοιχες διαδικασίες που υλοποιούν την κάθε μία ξεχωριστά, οι οποίες ονομάζονται λειτουργίες context. Για την αυτόματη επεξεργασία του context χρησιμοποιούνται συστατικά, τα οποία είναι τα context plugins και οι context υπηρεσίες. Ακόμη αναλύεται η χρήση των οδηγιών της επεξεργασίας context που έχουν σκοπό να θέσουν κάποιους κανόνες διαδικαστικούς σχετικά με την επεξεργασία του context ώστε να αποφευχθούν δυσάρεστα φαινόμενα. Τέλος εισάγονται και οι κατευθυντήριες γραμμές επεξεργασίες οι οποίες καθορίζουν τον τύπο του συστατικού που χρησιμοποιείται κάθε φορά για επεξεργασία ενός συγκεκριμένου context block, όπως επίσης και οι host όπου θα πραγματοποιηθεί αυτή η επεξεργασία.
Στο [NoPa03] αρχικά αναλύεται το ζήτημα της διανομής πληροφορίας που εξαρτάται από τον context. Δίνεται επίσης ένα μοντέλο context, έτσι ώστε η πληροφορία context που συλλέγεται να μπορεί να μοντελοποιηθεί σε μία μορφή συμβολική, ευέλικτη, εύκολα κατανοήσιμη και εκμεταλλεύσιμη. Επίσης παρουσιάζεται μία πλατφόρμα για web publishing η οποία ονομάζεται OMSwe η οποία είναι μία γενική πλατφόρμα για web publishing και βασίζεται στο OMS σύστημα διαχείρισης αντικειμένων δεδομένων.

Στο [ZaVP05] πραγματοποιείται αρχικά μία περιγραφή της εφαρμογής μας. Βασικά χαρακτηριστικά είναι η συμβατότητα και η προσαρμοστικότητα των υπηρεσιών οι οποίες παρέχονται και χρησιμοποιούνται από τους διάφορους peers. Δίνεται ένας ορισμός του context ως οτιδήποτε το οποίο μπορεί να επηρεάσει την κατάσταση μίας οντότητας η οποία παίζει κάποιο ρόλο στο σύστημα. Μία οντότητα είναι ένας άνθρωπος, μία τοποθεσία, ένα στοιχείο του συστήματος κ.τ.λ. Το σύστημα αποτελείται από δύο υποσυστήματα τον επεξεργαστή ερωτήσεων και τον διαχειριστή του context. Στη συνέχεια παρουσιάζεται το motivating example που δίνει ένα παράδειγμα πρακτικής χρήσης του συστήματος. Επίσης αναλύεται η αρχιτεκτονική του συστήματος με μεγάλη λεπτομέρεια όπου δίνεται η όλη διαδικασία η οποία πραγματοποιείται από τη στιγμή που κάποιος χρήστης κάνει κάποια ερώτηση μέχρι

τελικά να δοθεί το αποτέλεσμά της ακολουθώντας όλα τα ενδιάμεσα στάδια που απαιτούνται για να γίνει αυτό. Αναλύεται ο επεξεργαστής ερωτήσεων δίνοντας και έναν αλγόριθμο κατασκευής του δέντρου εκτέλεσης για δεδομένη ερώτηση. Ακόμη παρουσιάζεται και ένα παράδειγμα απάντησης σε μία ερώτηση εφαρμόζοντας διαφορετικά workflows ανά διαφορετικού τύπου peer. Τέλος ακολουθεί ανάλυση του διαχειριστή context ο οποίος ως μέλημά του έχει να επιλύσει κάποια ζητήματα συμπεριλαμβανομένου του καθορισμού των peers που θα ερωτηθούν, τον χρονισμό των ερωτήσεων, του καθορισμού των workflows για κάθε peer και την εκτέλεση των workflows.

2.3 Web Services

Στο [Iss+05] δίνεται ένας ορισμός των ambient intelligence συστημάτων τα οποία είναι συστήματα που παρέχουν στους καταναλωτές οικουμενική και άμεση προσπέλαση στο διαθέσιμο περιεχόμενο και στις υπηρεσίες μαζί με τρόπους για την αποτελεσματική εκμετάλλευσή τους. Πραγματοποιείται προσπάθεια για σχεδιασμό και προτυποποίηση μίας εφαρμογής ενός γενικού πλαισίου που να δίνει τη δυνατότητα για ambient intelligence εφαρμογές. Χρησιμοποιείται ένα σενάριο που αναπτύχθηκε στο INRIA πάνω από το WSAMI middleware. Το σενάριο αυτό δίνει ένα συγκεκριμένο παράδειγμα μίας τέτοιας εφαρμογής. Στη συνέχεια παρουσιάζεται η WSAMI η οποία είναι μία γλώσσα που χρησιμοποιείται για τέτοιες εφαρμογές, η οποία κάνει δυνατό τον καθορισμό των Web Services έτσι ώστε να μπορούν να συντεθούν δυναμικά σύμφωνα με το περιβάλλον στο οποίο οι υπηρεσίες απαιτούνται, ενώ επιβάλλουν την ποιότητα της υπηρεσίας. Χαρακτηριστικό αυτής της γλώσσας είναι ότι καθορίζονται μη-συναρτησιακές ή ποιοτικές ιδιότητες που σχετίζονται με τις υπηρεσίες, όπως ασφάλεια και απόδοση. Ακόμη παρουσιάζεται και το αντίστοιχο με την προαναφερθείσα γλώσσα, WSAMI middleware. Τέλος παρουσιάζεται μία πρωτότυπη εφαρμογή. Αναπτύχθηκε ένα πρότυπο βασισμένο σε Java του WSAMI core middleware, χρησιμοποιήθηκε το IEEE 802.11b και όλες οι υπηρεσίες εφαρμόζονται ως web services πάνω από το WSAMI. Το WSAMI core middleware πρότυπο διαχωρίζεται: στον WSAMI core broker και στην ND υπηρεσία (Naming & Discovery Service). Ενώ ο WSAMI core broker αποτελείται από: τον XML parser, τον SOAP container, το WSDL2WSAMI εργαλείο και το InstallWSAMI εργαλείο.



Στο [PiTs01] παρουσιάζεται αρχικά η αρχιτεκτονική των e-services η οποία περιέχει τρεις ρόλους, service provider, service requestor, service broker και υποστηρίζει τις βασικές λειτουργίες describe, publish, unpublish, update, discover, invoke. Στη συνέχεια παρουσιάζονται τα πλεονεκτήματα των e-services και ακολουθεί ανάλυση των προτύπων που υποστηρίζουν τις βασικές λειτουργίες των e-services, τα οποία είναι το WSDL, το SOAP και το UDDI. Επίσης πραγματοποιείται κατηγοριοποίηση των τεχνικών προκλήσεων των e-services ως προς διαφορετικές κατευθύνσεις. Η κατηγοριοποίηση αυτή γίνεται ως προς την πολυπλοκότητα των e-services, την αναγκαιότητα των λειτουργιών και το επίπεδο της αφαίρεσης. Ως προς την πολυπλοκότητά τους οι e-services διακρίνονται σε απλές και σύνθετες. Οι λειτουργίες από την πλευρά τους διαχωρίζονται σε βασικές και value-added, ενώ το επίπεδο αφαίρεσης διακρίνεται σε υψηλότερο και χαμηλότερο. Έτσι παίρνοντας όλους τους δυνατούς συνδυασμούς των πιθανών ενδεχομένων των τριών κατευθύνσεων, προκύπτουν οκτώ διαφορετικές κατηγορίες τεχνικών προκλήσεων των e-services.

Στο [POSV04] παρουσιάζεται το MWSAF (METEOR-S Web Service Annotation Framework), ένα πλαίσιο για ημι-αυτόματο ταίριασμα περιγραφών web services με οντολογίες. Έχουν αναπτυχθεί αλγόριθμοι ταιριάσματος WSDL αρχείων με σχετιζόμενες οντολογίες. Πραγματοποιείται χρήση πεδίων οντολογιών έτσι ώστε να κατηγοριοποιηθούν οι web services σε πεδία. Επίσης παρουσιάζεται η αρχιτεκτονική του συστήματος η οποία αποτελείται από τρία κύρια συστατικά: την αποθήκη οντολογιών, τη βιβλιοθήκη του μεταφραστή και τη βιβλιοθήκη του ταιριαστή (matcher). Τέλος δίνονται αποτελέσματα από πειράματα του συστήματος.

Στο [Car+04] παρουσιάζονται ζητήματα σχετικά με ποιότητα υπηρεσιών για workflows και web service διεργασίες. Συστήματα διαχείρισης workflows έχουν χρησιμοποιηθεί για να υποστηρίξουν διαφόρων τύπων business διεργασίες. Στα workflows για εφαρμογές ηλεκτρονικού εμπορίου και εφαρμογές web service, ορίζεται ένα συμβόλαιο μεταξύ του παραγωγού και του καταναλωτή που καθορίζει κάποιες στοιχεία ποιότητας υπηρεσιών. Τέτοια στοιχεία αποτελούν τα προϊόντα ή οι υπηρεσίες που πρέπει να παραδοθούν, η ημερομηνία τελικής παράδοσης, η ποιότητα των προϊόντων και το κόστος των υπηρεσιών. Η διαχείριση των μετρικών ποιότητας

υπηρεσιών είναι άμεσα σχετιζόμενες με την επιτυχία οργανισμών που ασχολούνται με το ηλεκτρονικό εμπόριο. Έτσι όταν προϊόντα ή υπηρεσίες δημιουργούνται ή διαχειρίζονται χρησιμοποιώντας workflows, η workflow μηχανή θα πρέπει να είναι σε θέση να δεχτεί τα τεχνικά χαρακτηριστικά και να μπορεί να εκτιμήσει, να παρακολουθεί και να ελέγξει την ποιότητα υπηρεσιών που αποδίδονται στους πελάτες. Στο άρθρο αυτό, παρουσιάζεται ένα προγνωστικό μοντέλο ποιότητας υπηρεσιών που καθιστά ικανή την διαδικασία υπολογισμού της ποιότητας υπηρεσιών για workflows αυτόματα, με βάση γνωρίσματα ποιότητας υπηρεσίας ατομικών εργασιών. Ακόμη παρουσιάζεται η εφαρμογή αυτού του μοντέλου για το METEOR workflow σύστημα. Περιγράφονται τα συστατικά που έχουν τροποποιηθεί ή προστεθεί και αναλύεται ο τρόπος με τον οποίο επιδρούν ώστε να γίνεται δυνατή η διαχείριση της ποιότητας των υπηρεσιών.

2.4 Ετερογενή Συστήματα Βάσεων Δεδομένων (Mediator-Wrappers)

Στο [RoSc97] παρουσιάζεται το Garlic το οποίο είναι ένα middleware σύστημα που παρέχει μία ενοποιημένη όψη μίας ποικιλίας από πηγές δεδομένων χωρίς να αλλάζει πώς ή πού αποθηκεύονται τα δεδομένα. Αναλύεται η αρχιτεκτονική του Garlic και δίνονται οι βασικοί στόχοι αυτής οι οποίοι είναι οι εξής: το αρχικό κόστος για να γραφτεί ένας wrapper πρέπει να είναι μικρό, οι wrappers θα πρέπει να είναι ικανοί να αναπτύσσονται και να εμπλουτίζονται όποτε αυτό κρίνεται αναγκαίο, η αρχιτεκτονική θα πρέπει να είναι ευέλικτη και να επιτρέπει πολύ μεγάλη ανάπτυξη έτσι ώστε εύκολα να είναι δυνατή η προσθήκη ή η προσαρμογή wrappers για νέες πηγές δεδομένων και τέλος η αρχιτεκτονική πρέπει να είναι άμεσα προσαρμόσιμη από μόνη της για βελτιστοποίηση ερωτήσεων. Στη συνέχεια αναλύεται η διαδικασία κατασκευής ενός Garlic wrapper. Οι υπηρεσίες που παρέχει ο wrapper είναι η μοντελοποίηση των δεδομένων ως αντικείμενα, η επίκληση μεθόδων, ο σχεδιασμός ερωτήσεων και η εκτέλεση ερωτήσεων. Η πρώτη υπηρεσία που παρέχει ένας wrapper είναι να μετατρέψει τα δεδομένα, που περιέχονται στις πηγές δεδομένων που βρίσκονται κάτω από την διαχείρισή του, σε αντικείμενα προσπελάσιμα από το Garlic. Κάθε Garlic αντικείμενο έχει ένα interface και μία εφαρμογή (implementation). Η δεύτερη παρεχόμενη υπηρεσία από τους wrappers είναι η κλήση μεθόδων πάνω σε αντικείμενα τα οποία βρίσκονται σε πηγές δεδομένων οι οποίες

είναι κάτω από τη διαχείρισή τους. Η κλήση μεθόδων μπορεί να παραχθεί από την μηχανή επεξεργασίας ερωτήσεων του Garlic ή από μία εφαρμογή του Garlic η οποία έχει κάποια αναφορά σε ένα αντικείμενο. Η τρίτη υπηρεσία η οποία παρέχεται από έναν wrapper είναι να λαμβάνει μέρος κατά τον σχεδιασμό μίας ερώτησης. Ο στόχος του σχεδιασμού μίας ερώτησης είναι να παραχθούν όλα τα εναλλακτικά πλάνα που απαντούν στην ερώτηση και στη συνέχεια να επιλεχθεί το πιο αποτελεσματικό από όλα αυτά. Αυτό πραγματοποιείται με τη βοήθεια του Garlic βελτιστοποιητή ο οποίος είναι ένας βελτιστοποιητής που επιτελεί αυτήν την εργασία με βάση ένα μοντέλο κόστους. Η συμμετοχή ενός wrapper στον σχεδιασμό ερωτήσεων ελέγχεται από ένα σύνολο από μεθόδους τις οποίες ο βελτιστοποιητής μπορεί να καλέσει κατά την αποτίμηση πλάνων. Η plan_access() μέθοδος χρησιμοποιείται για να παράγουμε πλάνα προσπέλασης συλλογών δεδομένων. Η plan_join() μέθοδος χρησιμοποιείται για να ώστε να παράγουμε πλάνα με πολλαπλά joins. Τέλος η plan_bind() μέθοδος χρησιμοποιείται ώστε να παράγουμε ένα ειδικό είδος πλάνων τα οποία μπορούν να συμπεριφερθούν σαν το εσωτερικό stream σε ένα bind join. Το bind join είναι ένα ειδικό join κατά το οποίο οι τιμές που παράγονται από τον εξωτερικό κόμβο του join περνάνε στον εσωτερικό κόμβο και αυτός τις χρησιμοποιεί έτσι ώστε να αποτιμήσει ένα υποσύνολο από τις συνθήκες που εφαρμόζονται στο join. Η τελευταία υπηρεσία ενός wrapper είναι η συμμετοχή του κατά την μετάφραση ενός πλάνου σε μορφή που είναι κατάλληλη για την εκτέλεσή του και στη συνέχεια εφόσον πραγματοποιηθεί αυτή η διαδικασία η συμμετοχή του κατά την εκτέλεση αυτού του πλάνου. Οι παραπάνω οι υπηρεσίες που παρέχονται από τους wrappers, σαν τελευταία εργασία που πρέπει να επιτελέσει ο συγγραφέας ενός wrapper, πρέπει να συνενωθούν όλες μαζί έτσι ώστε να αποτελούν έναν ενιαίο και πλήρη wrapper. Αυτή η εργασία ονομάζεται πακετάρισμα ενός wrapper.

Στο [HKWY97] παρουσιάζεται το Garlic το οποίο είναι ένα middleware σύστημα που σχεδιάστηκε έτσι ώστε να ενοποιεί δεδομένα, που ανήκουν σε μεγάλου εύρους ως προς την ποικιλομορφία τους πηγές δεδομένων, τα οποία χαρακτηρίζονται από πολύ διαφορετικές ιδιότητες και δυνατότητες. Αρχικά αναλύεται η αρχιτεκτονική του Garlic η οποία είναι παρόμοια με αυτήν των τυπικών συστημάτων ετερογενών βάσεων δεδομένων. Στο χαμηλότερο επίπεδο βρίσκονται οι πηγές δεδομένων οι οποίες αποθηκεύουν τα δεδομένα. Κάθε πηγή δεδομένων συνδέεται οπωσδήποτε με

έναν wrapper ο οποίος κρύβει την ετερογένεια αυτών των πηγών από το κύριο κομμάτι της αρχιτεκτονικής που είναι οι Garlic υπηρεσίες ερωτήσεων. Αυτό το τμήμα της αρχιτεκτονικής παίζει αντίστοιχο ρόλο με τον mediator στα συστήματα ετερογενών βάσεων δεδομένων. Διακρίνεται σε δύο συστατικά, τον επεξεργαστή γλώσσας ερωτήσεων και τη μηχανή εκτέλεσης ερωτήσεων. Το πρώτο δέχεται ως είσοδο μία ερώτηση και κατασκευάζει ένα πλάνο εκτέλεσής της. Αφού ο βελτιστοποιητής επιλέξει ένα βέλτιστο πλάνο εκτέλεσης της ερώτησης ακολουθεί η μηχανή εκτέλεσης ερωτήσεων η οποία εκτελεί την ερώτηση με βάση το πλάνο που επέλεξε ο βελτιστοποιητής. Αυτή η διαδικασία πραγματοποιείται περνώντας υποερωτήσεις στους wrappers και στη συνέχεια συνθέτοντας τις αντίστοιχες απαντήσεις από τους wrappers ώστε να προκύψει το τελικό αποτέλεσμα. Στη συνέχεια αναλύεται η βελτιστοποίηση ερωτήσεων στο Garlic. Για να πραγματοποιηθεί αυτό χρησιμοποιούνται τα STARs (Strategy Alternative Rules) τα οποία κατασκευάζουν πλάνα που μπορεί να τα χειριστεί η μηχανή εκτέλεσης ερωτήσεων του Garlic. Τα STARs είναι στρατηγικοί εναλλακτικοί κανόνες και στο Garlic τα STARs είναι γενικά (generic). Τα γενικά STARs τίθενται σε εφαρμογή όταν αποφασίζεται ότι ένα κομμάτι εργασίας πρέπει να πραγματοποιηθεί από κάποιον wrapper. Τα STARs θέτουν τον αντίστοιχο wrapper να κατασκευάσει το δικό τους κομμάτι του πλάνου. Από το προκύπτον σύνολο των ολοκληρωμένων πλάνων ο βελτιστοποιητής επιλέγει αυτό με το μικρότερο κόστος βασισμένος σε ένα μοντέλο κόστος και στη συνέχεια το πλάνο αυτό μεταφράζεται σε εκτελέσιμο. Τα πλάνα στο Garlic είναι δέντρα από τελεστές ή αλλιώς POPs (Plan Operators) και κατασκευάζονται από κάτω προς τα πάνω. Κάθε POP χαρακτηρίζεται από 8 ιδιότητες και αυτές είναι μία συνάρτηση των ιδιοτήτων των POPs που είναι είσοδοι σε αυτόν εάν υπάρχουν. Οι ιδιότητες είναι οι εξής: η ιδιότητα *Tables* στην οποία καθορίζεται το σύνολο των πινάκων που προσπελάζονται και κάνουν join, η ιδιότητα *Columns* όπου παρατίθεται το σύνολο των στηλών που βγαίνει ως έξοδος, η ιδιότητα *Preds* όπου δίνονται οι συνθήκες που εφαρμόζονται, η ιδιότητα *Source* που καθορίζει το πού παράγεται η έξοδος, η ιδιότητα *Mat* παίρνοντας τιμές TRUE ή FALSE αν η έξοδος αποθηκεύεται ή όχι αντίστοιχα, η ιδιότητα *Order* που περιέχει μία έκφραση ταξινόμησης αν οι πλειάδες του αποτελέσματος είναι διατεταγμένες αλλιώς NIL, η ιδιότητα *Cost* δίνοντας το εκτιμούμενο κόστος εφαρμογής του τελεστή και η ιδιότητα *Card* που παρέχει τον εκτιμούμενο αριθμό πλειάδων της εξόδου. Τα STARs είναι οι

κανόνες παραγωγής μίας γραμματικής η οποία παράγει πλάνα. Ένα STAR καθορίζει πώς τα POPs μπορούν να συνδυαστούν σε ένα πλάνο. Ο Garlic βελτιστοποιητής χρησιμοποιεί δυναμικό προγραμματισμό έτσι ώστε να βρει το καλύτερο πλάνο και με λογικό φόρτο. Σε κάθε βήμα της αποτίμησης ενός πλάνου εφαρμόζεται η τεχνική του κλαδέματος. Κατά την τεχνική αυτή κλαδεύονται πλάνα τα οποία έχουν μεγαλύτερο κόστος από κάποιο άλλο και επιπλέον οι ιδιότητές τους είναι υποσύνολο των ιδιοτήτων του άλλου πλάνου. Το κόστος ενός πλάνου είναι το άθροισμα από κόστη για τοπική επεξεργασία, κόστη για επικοινωνία και κόστη για αρχικοποίηση υποερωτήσεων και μεθόδων. Τα δύο τελευταία κόστη αποτιμούνται από συναρτήσεις του Garlic χρησιμοποιώντας σταθερές που αποθηκεύονται στον κατάλογο του Garlic, ενώ τα κόστη της πρώτης κατηγορίας αποτιμούνται από ένα μοντέλο κόστους που παρέχει το Garlic. Στη συνέχεια παρατίθενται πιο πολύπλοκα STARs που δείχνουν τέσσερις τρόπους για την πραγματοποίηση του join. Επίσης με τη βοήθεια ενός παραδείγματος δείχνεται ότι με χρήση STARs είναι εύκολη η μοντελοποίηση wrappers και των πηγών δεδομένων που διαχειρίζονται αυτοί. Το γεγονός ότι οι wrapper STARs είναι τόσο απλοί συμβαίνει για δύο λόγους. Πρώτον το Garlic παρέχει μία πανίσχυρη μηχανή ερωτήσεων η οποία μπορεί να πραγματοποιήσει οποιοδήποτε τμήμα της ερώτησης αδυνατεί να πραγματοποιήσει ο wrapper και δεύτερον τα wrapper STARs μοντελοποιούν το «τι» μπορεί να εκτελέσει ο wrapper κι όχι το «πώς». Με βάση το ίδιο παράδειγμα προκύπτουν κάποια πλεονεκτήματα του συστήματος που αναλύεται στο έγγραφο αυτό. Το πρώτο είναι ότι μπορούμε να ορίσουμε αρχικά ένα απλό ελάχιστο STAR για έναν wrapper κι έτσι εύκολα και γρήγορα να θέσουμε έναν wrapper σε εφαρμογή. Δεύτερον, οι συγγραφείς των wrappers μπορούν να προσθέσουν STARs ή εναλλακτικές για ένα ήδη υπάρχον STAR οποιαδήποτε χρονική στιγμή. Αυτό καθιστά εύκολη την τροποποίηση και την ανάπτυξη των wrappers. Τρίτον, τα STARs κάθε wrapper ορίζονται ανεξάρτητα από αυτά των άλλων wrappers και χωρίς να επηρεάζουν καθόλου ούτε τα Garlic STARs ούτε τις Garlic υπηρεσίες ερωτήσεων. Το γεγονός αυτό καθιστά εύκολη την προσθήκη νέων wrappers στο σύστημα, δηλαδή εύκολη την επεκτασιμότητα του συστήματος. Τέλος δίνεται ένα παράδειγμα βελτιστοποίησης μίας ερώτησης εφαρμόζοντας όλα τα παραπάνω που αναφέραμε με σκοπό να γίνει από τον αναγνώστη πιο κατανοητή η διαδικασία βελτιστοποίησης μίας ερώτησης στην ολότητά της.

2.5 Επεξεργασία/Βελτιστοποίηση Ερωτήσεων

Στο [Koss00] παρουσιάζεται μία γενική περιγραφή σχετικά με την κατανεμημένη επεξεργασία ερωτήσεων. Αρχικά παρουσιάζεται η αρχιτεκτονική του επεξεργαστή ερωτήσεων ο οποίος αποτελείται από τον parser, τον επανεγγραφέα ερωτήσεων (query rewriter), τον βελτιστοποιητή ερωτήσεων (query optimizer), γεννήτορα κώδικα (code generator) και την μηχανή εκτέλεσης ερωτήσεων (query execution engine). Στη συνέχεια αναλύονται διάφορα ζητήματα σχετικά με τη βελτιστοποίηση ερωτήσεων και διάφορα μοντέλα κόστους για να βοηθήσουν σε αυτό. Επίσης αναλύονται τεχνικές για την εκτέλεση ερωτήσεων όπως είναι το row blocking, η βελτιστοποίηση των multicasts, η πολυνηματική εκτέλεση ερωτήσεων, τα joins με οριζόντιο διαχωρισμό δεδομένων, τα semijoins, τα double-pipelined hash joins, τα βασισμένα στους δείκτες joins και η κατανεμημένη assembly αντικειμένων, οι top N και bottom N ερωτήσεις. Μία άλλη ενότητα είναι αυτή η οποία διαπραγματεύεται τα client-server συστήματα βάσεων δεδομένων. Στην ενότητα αυτή παρουσιάζεται ο τρόπος με τον οποίο γίνονται εκμεταλλεύσιμοι οι πόροι του client. Αυτό πραγματοποιείται με τις τεχνικές του query shipping, του data shipping, του υβριδικού shipping κι άλλες εκδοχές του υβριδικού shipping. Ακολουθεί η βελτιστοποίηση ερωτήσεων η οποία διακρίνεται σε κάποια ζητήματα. Αυτά είναι η επιλογή του site, το πού και πότε να γίνει η βελτιστοποίηση και η βελτιστοποίηση δύο βημάτων. Στη συνέχεια κατά τον ίδιο τρόπο παρατίθενται τεχνικές για εκτέλεση ερωτήσεων. Μία άλλη σημαντική ενότητα που παρατίθεται είναι αυτή των συστημάτων ετερογενών βάσεων δεδομένων. Παρουσιάζεται η wrapper αρχιτεκτονική για τέτοια συστήματα όπου αναλύεται ο ρόλος του mediator και του wrapper. Ακολουθείται η ίδια τακτική με πριν οπότε αναλύονται θέματα βελτιστοποίησης ερωτήσεων για τέτοια συστήματα, δίνονται διάφορα μοντέλα κόστους για αυτό και διάφορες τεχνικές για εκτέλεση ερωτήσεων σε τέτοια συστήματα. Ακόμη δίνεται η δυναμική τοποθέτηση δεδομένων όπου γίνεται διαχωρισμός του replication με το caching και μετά αναλύονται αλγόριθμοι δυναμικού replication και εγκατάστασης cache. Τέλος παρουσιάζονται νέες αρχιτεκτονικές για κατανεμημένη επεξεργασία ερωτήσεων, όπου αναλύονται οικονομικά μοντέλα για κατανεμημένη επεξεργασία ερωτήσεων και βασισμένα στη διασπορά πληροφοριακά συστήματα.

Στο [Sel+79] αρχικά παρουσιάζονται οι τέσσερις φάσεις επεξεργασίας μίας δήλωσης στην SQL, που είναι το parsing, η βελτιστοποίηση, η παραγωγή κώδικα και η εκτέλεση. Στη συνέχεια παρουσιάζεται ένας αλγόριθμος που χρησιμοποιείται για τη βελτιστοποίηση ερωτήσεων. Πρώτα ενδιαφερόμαστε για μονοπάτια προσπέλασης των σχέσεων που λαμβάνουν μέρος στην ερώτηση. Απαριθμούμε όλους τους δυνατούς τρόπους προσπέλασης αυτών των σχέσεων και με βάση ένα μοντέλο κόστους αντιστοιχούμε σε κάθε έναν από τους τρόπους αυτούς το κόστος τους. Σε δεύτερη φάση πραγματοποιείται εύρεση όλων των μονοπατιών προσπέλασης για join δύο πινάκων με όλους τους τρόπους με τους οποίους μπορούν να γίνουν τα join αυτά. Με τον ίδιο τρόπο συνεχίζουμε για join τριών πινάκων κ.τ.λ., μέχρις ότου φτάσουμε σε join n πινάκων, δηλαδή όλων των πινάκων που κάνουν join στην ερώτηση. Η όλη αυτή διαδικασία πραγματοποιείται χρησιμοποιώντας δυναμικό προγραμματισμό, έτσι ώστε να κάνουμε κλάδεμα πλάνων τα οποία με βάση το μοντέλο κόστους σίγουρα δεν είναι το βέλτιστο που αναζητούμε. Στην τελική φάση, από όλα τα πλάνα που επιβίωσαν από το κλάδεμα επιλέγουμε αυτό με το μικρότερο κόστος που είναι και το βέλτιστο. Επίσης παρουσιάζεται κι ένα παράδειγμα της παραπάνω διαδικασίας με τη βοήθεια του οποίου γίνεται πιο κατανοητή η φάση της βελτιστοποίησης. Τέλος γίνεται αναφορά στις εμφωλευμένες ερωτήσεις κάνοντας διάκριση μεταξύ αυτών που αναφέρονται σε στήλες πινάκων που βρίσκονται σε blocks υψηλότερου επιπέδου και σε αυτές που δεν αναφέρονται. Στην πρώτη περίπτωση αυτές οι εμφωλευμένες ερωτήσεις αποτιμούνται μία φορά για κάθε μία τιμή της στήλης του πίνακα του υψηλότερου block, ενώ στη δεύτερη αποτιμούνται εξολοκλήρου πριν αποτιμηθεί η ερώτηση του υψηλότερου επιπέδου.

Στο [Ioan96] το βασικό ζήτημα που αναλύεται είναι η βελτιστοποίηση ερωτήσεων. Παρουσιάζεται η αρχιτεκτονική του βελτιστοποιητή ερωτήσεων η οποία αποτελείται από δύο κύρια επίπεδα, το επίπεδο επανεγγραφής και το επίπεδο κατασκευής πλάνων. Στο πρώτο πραγματοποιείται επανεγγραφή της ερώτησης σε μορφή κατάλληλη για την περαιτέρω επεξεργασία της από τον βελτιστοποιητή. Στο δεύτερο επίπεδο πραγματοποιείται η παραγωγή εναλλακτικών πλάνων της ερώτησης. Στο επίπεδο αυτό διακρίνονται κάποια συστατικά που βοηθούν τη διαδικασία παραγωγής πλάνων. Αυτά είναι ο αλγεβρικός χώρος, ο χώρος μεθόδου-δομής, το μοντέλο κόστους και ο εκτιμητής μεγέθους. Με βάση τον αλγόριθμο βελτιστοποίησης που αναπτύσσεται στο

άρθρο [Koss00], πραγματοποιείται προσπάθεια εκτέλεσης του αλγορίθμου αυτού με χρήση ενός παραδείγματος. Στη συνέχεια με αφορμή το συστατικό του εκτιμητή μεγέθους γίνεται μία εισαγωγή στα ιστογράμματα που παίζουν σημαντικό ρόλο στη λειτουργία του. Ακόμη γίνεται αναφορά σε μη-κεντρικοποιημένα περιβάλλοντα, όπως τις παράλληλες βάσεις δεδομένων και τις κατανεμημένες βάσεις δεδομένων και τέλος σε προχωρημένους τύπους βελτιστοποίησης, όπως η σημασιολογική, η ολική και η παραμετρική-δυναμική βελτιστοποίηση ερωτήσεων.

Στο [UrFr99] παρουσιάζεται έναν non-blocking τελεστής join ο οποίος ονομάζεται XJoin. Ο XJoin επεκτείνει το συμμετρικό join κατακερματισμού (symmetric hash join) να χρησιμοποιεί δευτερεύουσα αποθήκευση, πράγμα το οποίο επιτρέπει στο XJoin να χρησιμοποιείται για μεγάλες εισόδους και να εκτελείται ταυτόχρονα με άλλους τελεστές ερώτησης σε ένα θαμνώδες (bushy) πλάνο ερώτησης. Ένα κύριο συστατικό του XJoin είναι μία διεργασία η οποία είναι χρονοπρογραμματισμένη να λειτουργεί αντενεργητικά στο υπόβαθρο (reactively scheduled background process). Το συστατικό αυτό έχει ως αποτέλεσμα να μειώνει τις καθυστερήσεις και να παράγει περισσότερες πλειάδες συντομότερα. Το XJoin έχει δύο βασικές αρχές: βελτιστοποιείται για να παράγει αποτελέσματα αμέσως μόλις αυτά γίνονται διαθέσιμα και επιτρέπει να γίνεται πρόοδος ακόμη κι όταν μία ή παραπάνω πηγές καθυστερούν. Οι κύριες προκλήσεις του τελεστή αυτού είναι να διαχειρίζεται τη ροή πλειάδων μεταξύ μνήμης και δευτερεύουσας αποθήκευσης, να ελέγχει την background διεργασία η οποία αρχικοποιείται όταν οι είσοδοι καθυστερούν, να εξασφαλίζει ότι παράγεται όλη η απάντηση και να εξασφαλίζει ότι δεν παράγονται διπλότιμες πλειάδες. Στη συνέχεια αναλύεται λεπτομερειακά ο τρόπος με τον οποίο λειτουργεί αυτός ο τελεστής. Κεντρική ιδέα αποτελεί ο τεμαχισμός (partitioning) όπου το XJoin χωρίζει και τις δύο εισόδους σε τμήματα με βάση κάποια συνάρτηση κατακερματισμού. Κάθε τμήμα συνθέτεται από ένα memory-resident portion και από ένα disk-resident portion. Στο πρώτο περιέχονται οι πλειάδες που έφτασαν τελευταία ενώ στο δεύτερο όλες οι άλλες. Αφού γίνει ο τεμαχισμός ακολουθούν τρεις φάσεις στις οποίες και πραγματοποιείται το XJoin. Στην πρώτη γίνονται ταιριάσματα πλειάδων οι οποίες και οι δύο είναι στα αντίστοιχα τμήματα στη μνήμη. Όσα ταιριάσματα πραγματοποιηθούν προωθούνται ως αποτελέσματα του τελεστή. Η δεύτερη φάση λειτουργεί όταν μπλοκάρει η πρώτη. Τότε γίνεται επιλογή κάποιου

τμήματος στο δίσκο και ελέγχεται για ταίριασμα με το αντίστοιχο τμήμα στη μνήμη. Εάν βρεθεί κάποιο προωθείται ως αποτέλεσμα του τελεστή αφού πρώτα όμως ελεγχθεί για αποφυγή διπλότιμων. Η τρίτη φάση ξεκινάει αφού όλες οι πλειάδες των δύο εισόδων έχουν παρθεί. Κατά τη φάση αυτή πραγματοποιείται join όλων των τμημάτων και των δύο πηγών είτε είναι στη μνήμη είτε στο δίσκο. Όπως και στη δεύτερη φάση είναι απαραίτητος έλεγχος για την αποφυγή των διπλότιμων. Έτσι παρουσιάζεται επίσης και η όλη διαδικασία που ακολουθείται για να αποφευχθούν τα διπλότιμα. Διπλότιμες πλειάδες μπορούν να παραχθούν μόνο κατά τη δεύτερη και τρίτη φάση οπότε σε αυτές τις φάσεις χρησιμοποιούμε κάποια χρονόσημα με τη βοήθεια των οποίων μπορούμε να αναγνωρίσουμε τα διπλότιμα και να τα αποφύγουμε. Για περισσότερες βελτιστοποιήσεις μπορούμε να προσθέσουμε μία cache όπως επίσης μπορούμε να ελέγχουμε τη δεύτερη φάση. Μπορούμε καθορίζοντας την τιμή ενός ορίου ενεργοποίησης (activation threshold) να κάνουμε τη δεύτερη φάση «επιθετική» ή όχι. Το όριο αυτό περιορίζει τη δεύτερη φάση στο να πραγματοποιείται μόνο όταν αναμένεται η εφαρμογή της να παράγει ποσοστό πλειάδων ως προς τις συνολικές πλειάδες του αποτελέσματος πάνω από αυτό το όριο. Έτσι μπορούμε να ελέγξουμε την εφαρμογή της αργής φάσης δύο μόνο όταν είναι αποδοτική. Τέλος πραγματοποιήθηκαν διάφορα πειράματα για σύγκριση του XJoin με άλλα joins και με άλλες βελτιστοποιημένες εκδοχές του, όπως με χρήση cache και με έλεγχο της δεύτερης φάσης.

2.6 Benchmarks

Στο [HaST05] εισάγεται ένα νέο και διαθέσιμο δημόσια test bed και benchmark που ονομάζεται THALIA (Test Harness for the Assessment of Legacy information Integration Approaches) με σκοπό να απλοποιήσει την αποτίμηση των ήδη υπαρχόντων τεχνολογιών που είναι σχετικές με ενοποίηση ετερογενών δεδομένων. Το THALIA παρέχει μία συλλογή από πηγές δεδομένων που αναπαριστούν καταλόγους μαθημάτων πανεπιστημίων, ένα σύνολο από 12 benchmark ερωτήσεις, όπως επίσης και μία συνάρτηση έτσι ώστε να μετράμε την απόδοση ενός συστήματος ενοποίησης δεδομένων. Το benchmark αυτό επικεντρώνεται κυρίως σε συντακτικές και σημασιολογικές ετερογένειες. Επίσης παρουσιάζεται μία συστηματική κατάταξη των διαφορετικών τύπων συντακτικών και σημασιολογικών ετερογενειών οι οποίες

οδηγούν στις 12 ερωτήσεις του benchmark. Στο τέλος δίνεται και μία δειγματοληπτική αποτίμηση δύο συστημάτων ενοποίησης δεδομένων, έτσι ώστε να φανεί η χρησιμότητα του THALIA κατά την αναγνώριση και τον καθορισμό των προβλημάτων που εμφανίζονται σε αυτά.

Στο [Theo03] παρουσιάζονται οι βασισμένες στην τοποθεσία υπηρεσίες οι οποίες αποτελούν εφαρμογές που περιέχουν spatiotemporal βάσεις δεδομένων. Μελετούνται αυτού του είδους οι εφαρμογές με όρους απαιτήσεων βάσεων δεδομένων και παρέχονται 10 benchmark ερωτήσεις βάσεων δεδομένων (συν άλλες δύο λειτουργίες για φόρτωμα και τροποποίηση δεδομένων). Η λίστα των ερωτήσεων περιέχει ερωτήσεις επιλογής σε στατικά και κινούμενα αντικείμενα αναφοράς, ερωτήσεις με join και εναδικές λειτουργίες πάνω σε τροχιές κινούμενων αντικειμένων. Επίσης παρουσιάζεται σχετική εργασία που έχει πραγματοποιηθεί σχετικά με επεξεργασία ερωτήσεων για τέτοιου είδους-τύπου ερωτήσεις. Ιδιαίτερη έμφαση δίνεται στην ευρετηριοποίηση των κινούμενων αντικειμένων και στην πρόταση υποψηφίων για αποτελεσματική υποστήριξη βάσεων δεδομένων για υπηρεσίες βασισμένες στην τοποθεσία.

2.7 Ταίριασμα Σχημάτων

Στο [RaBe01] παρουσιάζεται μία κατάταξη που καλύπτει πολλές από τις ήδη υπάρχουσες εκδοχές που είναι σχετικές με το ζήτημα του ταιριάσματος του σχήματος. Το ζήτημα αυτό είναι ένα βασικό πρόβλημα το οποίο συναντάται σε πολλά πεδία των εφαρμογών βάσεων δεδομένων, όπως είναι η ενοποίηση δεδομένων, το e-business, οι αποθήκες δεδομένων και η σημασιολογική επεξεργασία ερωτήσεων. Από τη μία στις εφαρμογές που υπάρχουν σήμερα σχετικά με αυτό το ζήτημα, το ταίριασμα σχήματος πραγματοποιείται χειροκίνητα (manually), πράμα το οποίο έχει συγκεκριμένους περιορισμούς. Από την άλλη ερευνητικά άρθρα που έχουν δημοσιευτεί προτείνουν διάφορες τεχνικές με σκοπό την επίτευξη μερικής αυτοματοποίησης της λειτουργίας του ταιριάσματος για συγκεκριμένα πεδία εφαρμογής. Όπως προαναφέραμε στο άρθρο αυτό πραγματοποιείται μία κατάταξη πολλών από τις υπάρχουσες εφαρμογές. Ειδικότερα, γίνεται διαχωρισμός μεταξύ ταιριαστών (matchers) που λειτουργούν σε επίπεδο σχήματος, σε επίπεδο στιγμιότυπου, σε επίπεδο στοιχείου και σε επίπεδο

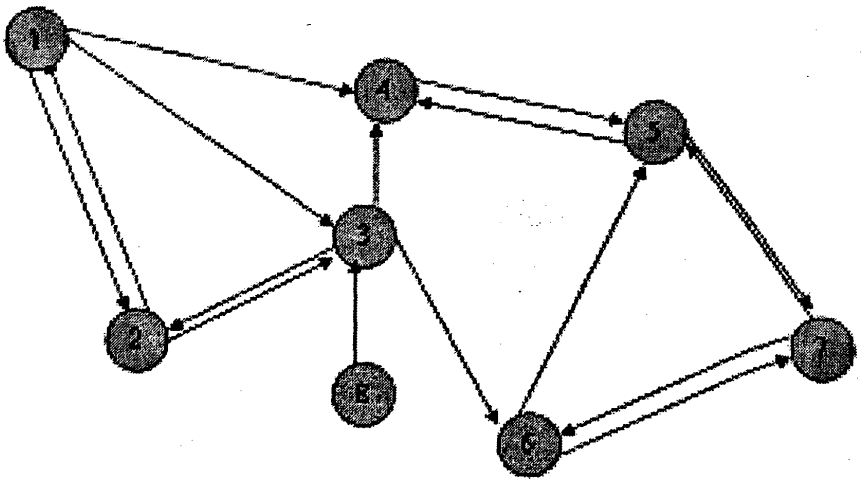
δομής. Επίσης διαχωρισμός γίνεται μεταξύ ταιριαστών που βασίζονται στη γλώσσα και σε αυτούς που βασίζονται σε περιορισμούς. Η κατάταξη αυτή που παρουσιάζεται έχει ως σκοπό να χρησιμεύσει όταν συγκρίνουμε διαφορετικές εκδοχές του ταιριάσματος σχήματος, όταν αναπτύσσουμε έναν νέο αλγόριθμο ταιριάσματος και όταν εφαρμόζουμε ένα συστατικό ταιριάσματος σχήματος.

Το [MaHa03] ασχολείται με το πρόβλημα της σύνθεσης mappings μεταξύ πηγών δεδομένων για mappings τα οποία συγκεκριμενοποιούνται χρησιμοποιώντας τον GLAV φορμαλισμό (γενικεύοντας τους LAV, GAV). Ειδικότερα, δοθέντων σημασιολογικών mappings μεταξύ των πηγών δεδομένων A, B και μεταξύ των B, C ο στόχος είναι η εύρεση ενός άμεσου mapping μεταξύ A, C που είναι ισοδύναμο με τα αρχικά mappings. Ο σκοπός της σύνθεσης mappings, όπως και της σύνθεσης ερωτήσεων, είναι η επιτάχυνση της διαδικασίας της βελτιστοποίησης ερωτήσεων και η παραγωγή καλύτερων πλάνων προς εκτέλεση. Επίσης η σύνθεση mappings παίζει σημαντικό ρόλο κατά τη διατήρηση σημασιολογικών σχέσεων όταν κάποιες πηγές δεδομένων αποκόπτονται από κάποιο κατανεμημένο σύστημα. Αρχικά δείχνεται ότι ακόμη και για απλά mappings, το mapping που αποτελεί προϊόν της σύνθεσης μπορεί να είναι ένα μη-πεπερασμένο σύνολο από GLAV φόρμουλες. Δεύτερον, περιγράφεται ένας αλγόριθμος που κρυπτογραφεί ένα μη-πεπερασμένο σύνολο από GLAV φόρμουλες στην σύνθεση χρησιμοποιώντας μία πεπερασμένη δομή. Ο αλγόριθμος αυτός όταν τερματίσει εγγυάται να παράγει ολόκληρη την σύνθεση. Τρίτον, δείχνεται ότι για την κλάση των CQ_k ερωτήσεων ο αλγόριθμος εγγυάται να τερματίσει και είναι δυνατή η εύρεση κάτω και πάνω ορίου της πολυπλοκότητας του προβλήματος της σύνθεσης. Τέλος μελετάται το ζήτημα της απάντησης σε μία ερώτηση σε μία πηγή δεδομένων όταν η σύνθεση που προκύπτει ως αποτέλεσμα είναι ένα μη-πεπερασμένο σύνολο από GLAV φόρμουλες. Περιγράφεται ένας αλγόριθμος ο οποίος εγγυάται να παράγει όλες τις απαντήσεις σε μία ερώτηση. Αυτός ο αλγόριθμος γενικεύει έναν προηγούμενο αλγόριθμο ο οποίος λαμβάνει υπόψη μόνο LAV mappings και μόνο για περιπτώσεις που μία ισοδύναμη επανεγγραφή της ερώτησης μπορεί να βρεθεί.

ΚΕΦΑΛΑΙΟ 3. ΕΠΕΚΤΑΣΗ ΤΗΣ SQL

3.1 Περιγραφή του Συστήματος

Η υποδομή του συστήματος μπορεί να μοντελοποιηθεί από ένα γράφημα *G* το οποίο αποτελείται από ένα σύνολο από κόμβους *V*, οι οποίοι συνδέονται μεταξύ τους με ένα σύνολο από ακμές *E*. Το γράφημα αυτό συμβολίζεται ως *G(V, E)* και είναι ένα γράφημα όπως αυτό που φαίνεται στο σχήμα 3.1.



Σχήμα 3.1: Γράφημα G

Ένας **γράφος** *G* είναι μία διάταξη από κόμβους *V*, που κάθε **κόμβος** αντιστοιχεί σε κάποιον peer του συστήματος και κάθε **ακμή** $e=<u,v>$ που ανήκει στο *E*, δηλώνει το γεγονός ότι ο κόμβος *u* μπορεί να επικοινωνήσει με τον κόμβο *v*. Η έννοια *μπορεί να επικοινωνήσει* σημαίνει ότι μπορεί να στείλει δεδομένα ή μπορεί να κάνει αίτηση για δεδομένα. Οι ακμές είναι **κατευθυνόμενες** κι έχουν την έννοια ότι ο κόμβος από τον οποίο ξεκινάνε έχει τη δυνατότητα να επικοινωνήσει με τον κόμβο στον οποίο καταλήγουν. Πρέπει να τονίσουμε ότι το αντίστροφο δεν ισχύει, δηλαδή η ακμή $e=<u,v>$ δεν δηλώνει ότι ο κόμβος *v* μπορεί να επικοινωνήσει με τον κόμβο *u*. Κάτι τέτοιο το δηλώνει, αν υπάρχει, μία ακμή $e'=<v,u>$. Οι κόμβοι του γραφήματος *G*, για να διακρίνονται μεταξύ τους, χαρακτηρίζονται με έναν μοναδικό ακέραιο αριθμό, ο οποίος αντιστοιχεί στον peer τον οποίο αναπαριστούν.

Δύο κόμβοι θεωρούμε ότι συνδέονται άμεσα, όταν υπάρχει κάποια ακμή *e*, που ανήκει στο *E*, η οποία να τους ενώνει. Επιπρόσθετα θεωρούμε ότι οι δύο αυτοί κόμβοι απέχουν απόσταση 1 βήματος ή 1 hop.

Διαδρομή μεταξύ δύο κόμβων, έστω u_1 και u_2, αποτελεί ένα οποιοδήποτε σύνολο από ακμές που ανήκουν στο *E*, το οποίο ενώνει τους δύο αυτούς κόμβους. Δηλαδή ένα σύνολο από ακμές $\{(u_1,v_1), (v_1,v_2), ..., (v_n,u_2)\}$.

Απόσταση μεταξύ δύο κόμβων, έστω u_1 και u_2, αποτελεί η ελάχιστη διαδρομή μεταξύ των δύο αυτών κόμβων και είναι ίση με τόσα βήματα όσα το πλήθος των στοιχείων του συνόλου που αποτελεί την ελάχιστη διαδρομή μεταξύ των δύο αυτών κόμβων. Συμβολίζεται ως distance(u_1,u_2) και υπολογίζεται σε βήματα όπως εξηγήσαμε παραπάνω.

Το γράφημα *G* **δεν είναι πάντα συνεκτικό**, δηλαδή δεν έχουμε τη δυνατότητα πάντα από οποιονδήποτε κόμβο του *G*, ακολουθώντας κάποια διαδρομή από ακμές, να φτάσουμε σε οποιονδήποτε κόμβο επιθυμούμε.

Σύνολο κόμβων αναφοράς. Κάθε χρονική στιγμή *T*, ένας κόμβος u γνωρίζει ένα υποσύνολο του γράφου, όπως αυτός ήταν διαμορφωμένος μία προηγούμενη χρονική στιγμή $T'<=T$. Το υποσύνολο του γράφου αυτό, το ονομάζουμε *σύνολο κόμβων*

αναφοράς του δεδομένου κόμβου u ή viewpoint(*u,T*). Το σύνολο viewpoint(*u,T*) αποτελείται από ένα συνδεδεμένο υποσύνολο του *V* το οποίο περιλαμβάνει τον κόμβο *u*.

Κοινότητα. Μία *κοινότητα* από κόμβους είναι ένα υποσύνολο του *V*, για τους οποίους ισχύει ότι έχουν κάποια κοινά σημασιολογικά χαρακτηριστικά και λόγω αυτών διακρίνονται από τους άλλους κόμβους. Ένας κόμβος μπορεί να ανήκει σε πολλές κοινότητες. Επίσης η έννοια της κοινότητας ορίζεται ως η κοινότητα ενός κόμβου με κάποιο όνομα. Έτσι για έναν κόμβο *u* η κοινότητά του με όνομα *c_name*, συμβολίζεται ως community(*c_name,u*) και ορίζεται ως εξής:
community(*c_name,u*)={ *v* | *v* ανήκει στο viewpoint(*u,T*) και ισχύει $\varphi_{c\_name}(v)$=true}, με την φ_{c_name} μία συνάρτηση που επιστρέφει λογικές τιμές true ή false και καθορίζει αν ο κόμβος που δέχεται ως παράμετρος ανήκει στην κοινότητα *c_name* ή όχι.

Έστω $Z_1, Z_2, \ldots, Z_d$ όλες οι κοινότητες των κόμβων που ανήκουν στο σύνολο κόμβων αναφοράς ενός κόμβου u (viewpoint(u,T). Κάθε μία κοινότητα Z_i, με $1<=i<=d$, είναι ένα σύνολο από κόμβους του viewpoint(u,T). Ισχύει ότι η ένωση όλων των κοινοτήτων μας δίνει το σύνολο viewpoint(u,T), δηλαδή $Z_1 \cup Z_2 \cup \ldots \cup Z_d$ = viewpoint(u,T). Για να το εξασφαλίσουμε αυτό εισάγουμε και την κοινότητα *unclassified*, στην οποία περιέχονται όλοι οι κόμβοι που δεν περιέχονται σε καμία άλλη κοινότητα. Έτσι όλοι οι κόμβοι που ανήκουν στο viewpoint(u,T), ανήκουν επίσης και σε κάποια κοινότητα από κόμβους του κόμβου u. Η τομή δύο κοινοτήτων, όπως προαναφέραμε δεν είναι απαραίτητα το κενό σύνολο.

Web Services. Κάθε κόμβος έχει ένα σύνολο από web services τις οποίες δημοσιοποιεί και δίνει την δυνατότητα κλήσης τους από άλλους κόμβους. Δηλαδή κάθε κόμβος *u* που ανήκει στο *V* διαθέτει ένα σύνολο $WS^u=\{ws^u_1, ws^u_2, \ldots, ws^u_m\}$ που είναι το σύνολο των web services που δημοσιοποιεί ο δεδομένος κόμβος. Με ws^u_i, όπου *i* ισχύει ότι $1<=i<=m$, συμβολίζεται μία web service του κόμβου *u*.

Ρολόι κόμβου. Κάθε κόμβος έχει ένα δικό του ρολόι. Τα ρολόγια των κόμβων δεν είναι απαραίτητα συγχρονισμένα.

Κλάση κόμβου. Για έναν γράφο *G*, οι κόμβοι του χωρίζονται σε ξένα σύνολα τα οποία ονομάζονται κλάσεις. Σε κάθε κλάση ομαδοποιούμε όλους τους κόμβους οι οποίοι προσφέρουν τις ίδιες web services. Κάθε κόμβος ανήκει σε μία μόνο κλάση. Έστω $Class_1$, $Class_2$, …, $Class_t$, οι κλάσεις των κόμβων του *G*, τότε ισχύουν τα εξής:

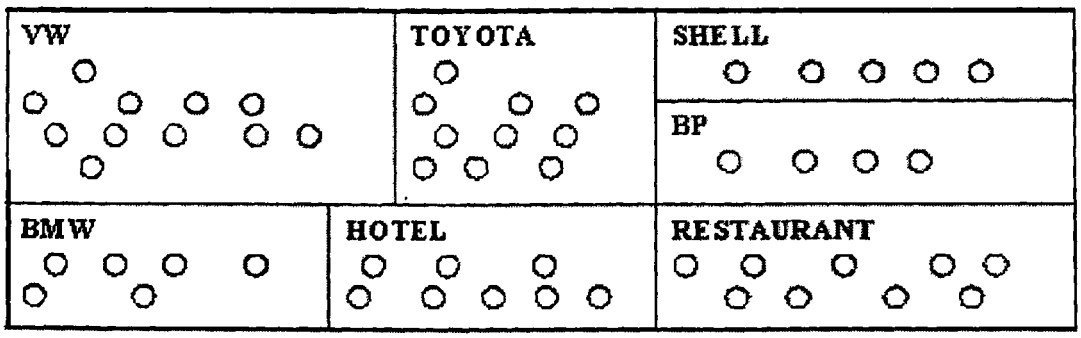
$Class_1 \cup Class_2 \cup \ldots \cup Class_t = V$

$Class_i \cap Class_j = \{\ \}$, για $1<=i<=t$, $1<=j<=t$ και $i \neq j$.

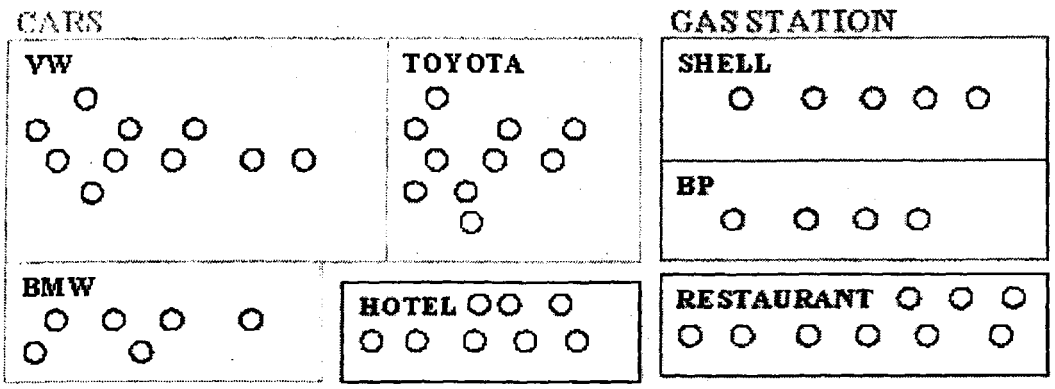
Στο σχήμα 3.2 που ακολουθεί φαίνονται οι κλάσεις VW, BMW, TOYOTA, SHELL, BP, HOTEL, RESTAURANT με τους αντίστοιχους κόμβους. Παρατηρούμε ότι ισχύουν οι δύο παραπάνω ιδιότητες, δηλαδή όλοι οι κόμβοι ανήκουν σε μία κλάση και κανείς κόμβος δεν ανήκει σε πάνω από μία κλάσεις.

Αν χρησιμοποιήσουμε ιεραρχία κλάσεων τότε δεν ισχύει η δεύτερη πρόταση. Δηλαδή υπάρχουν κόμβοι οι οποίοι ανήκουν στην κλάση τους αλλά επίσης ανήκουν και στην κλάση που βρίσκεται από πάνω αυτής, αν υπάρχει. Στο παράδειγμά μας έχουμε μία κλάση CARS πάνω από τις VW, BMW και TOYOTA και μία κλάση GAS STATION πάνω από τις SHELL και BP. Στο σχήμα 3.3 φαίνεται το γεγονός αυτό.

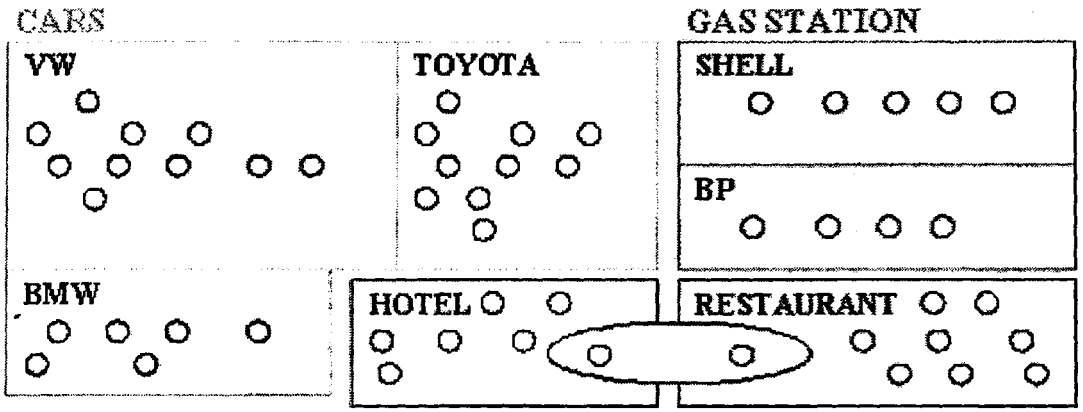
Επειδή ένας κόμβος ανήκει ακριβώς σε μία κλάση, όποιες οντότητες του πραγματικού κόσμου ανήκουν ενδεχομένως σε παραπάνω από μία κλάσεις, αντιμετωπίζονται ως διακριτά αντικείμενα. Για παράδειγμα αν κάποιο ξενοδοχείο διαθέτει δικό του εστιατόριο. Στην περίπτωση αυτή θεωρούμε ότι υλοποιεί τα interface δύο κλάσεων και αντιμετωπίζεται ως δύο κόμβοι. Στο σχήμα 3.4 φαίνεται η περίπτωση αυτή.



Σχήμα 3.2: Οι Κλάσεις του Συστήματος με τους Κόμβους τους



Σχήμα 3.3: Οι Κλάσεις του Συστήματος με τους Κόμβους τους δείχνοντας την Ιεραρχία των Κλάσεων (σε αντιδιαστολή με το σχήμα 3.2)



Σχήμα 3.4: Ένας Κόμβος που Ανήκει και στις δύο Κλάσεις

Διαφορά μεταξύ κοινότητας-κλάσης κόμβου. Στο σημείο αυτό αξίζει να αναφέρουμε τη διαφορά μεταξύ κλάσης και κοινότητας, πράγμα που μπορεί να γίνει περισσότερο κατανοητό με τη βοήθεια ενός παραδείγματος. Ας θεωρήσουμε έναν κόμβο *u* που είναι ένα αυτοκίνητο μάρκας BMW. Μία κοινότητα αυτού του κόμβου μπορεί να είναι όλοι οι κόμβοι που βρίσκονται σε απόσταση μικρότερη από 5 χιλιόμετρα από αυτόν. Η κλάση του όμως, δεν καθορίζεται από τον ίδιο, αλλά είναι ιδιότητα του γράφου που περιλαμβάνει όλους τους κόμβους που δημοσιοποιούν τις ίδιες web services με αυτόν (π.χ., όλα τα αυτοκίνητα μάρκας BMW).

Βάση δεδομένων κόμβου. Κάθε κόμβος έχει μία βάση δεδομένων, η οποία αποτελείται από ένα σύνολο από πίνακες. Οι πίνακες μπορεί να ανήκουν σε μία από τις παρακάτω κατηγορίες:

Τοπικά αποθηκευμένοι πίνακες: πίνακες οι οποίοι είναι αποθηκευμένοι με τα περιεχόμενά τους τοπικά στον δεδομένο κόμβο και είναι σε θέση κάθε στιγμή να ερωτηθούν.

Νοητοί πίνακες: πίνακες με δεδομένο σχήμα των οποίων το περιεχόμενο δεν είναι τοπικά αποθηκευμένο στη βάση δεδομένων του κόμβου, αλλά συλλέγεται ως πλειάδες οι οποίες ζητούνται και έρχονται από άλλους κόμβους. Οι πίνακες αυτοί συλλέγουν τις κατάλληλες εγγραφές από τον γράφο των κόμβων, τις αποθηκεύουν τοπικά και πλέον είναι έτοιμοι να συμμετάσχουν στην επεξεργασία της αρχικής ερώτησης με τον παραδοσιακό τρόπο.

Υβριδικοί πίνακες: πίνακες οι οποίοι είναι συνδυασμός των δύο παραπάνω κατηγοριών. Δηλαδή αποτελούνται εν μέρει από εγγραφές που είναι τοπικά αποθηκευμένες στη βάση δεδομένων του κόμβου, αλλά επίσης και από εγγραφές οι οποίες έρχονται μετά από ανάλογη αίτηση από άλλους κόμβους. Ισχύει ότι και για τους νοητούς πίνακες, δηλαδή συλλέγουν τις κατάλληλες εγγραφές από τον γράφο των κόμβων, τις αποθηκεύουν τοπικά και πλέον είναι έτοιμοι να συμμετάσχουν στην επεξεργασία της αρχικής ερώτησης με τον παραδοσιακό τρόπο.

Για τις δύο τελευταίες κατηγορίες πινάκων η κάθε πλειάδα τους συνοδεύεται από ένα χρονόσημο, με βάση το ρολόι του κόμβου που υποδέχεται και αποθηκεύει την πλειάδα, το οποίο χαρακτηρίζει τη χρονική στιγμή που η συγκεκριμένη πλειάδα εισάχθηκε στον δεδομένο πίνακα.

Χαρακτηριστικά κόμβου. Κάθε κόμβος u έχει κάποια χαρακτηριστικά:

- *Διαθεσιμότητα*, η οποία εκφράζει την πιθανότητα καλώντας κάποια web service αυτού, να πραγματοποιήσει ο δεδομένος κόμβος την εκτέλεσή της. Φυσικά αφού εκφράζει πιθανότητα, είναι μία ποσότητα μεταξύ του 0 και του 1, δηλαδή $0<=availability(u)<=1$.
- *Χρόνος απόκρισης*, ο οποίος εκφράζει το μέσο χρόνο που απαιτείται για την εκτέλεση μίας operation του κόμβου αυτού και την αποστολή των αποτελεσμάτων που προκύπτουν, $response\_time(u) \varepsilon R$ (σύνολο πραγματικών αριθμών).
- Κλάση στην οποία ανήκει, η οποία ορίζεται όπως αναλύσαμε παραπάνω.

Κατάλογος κόμβου. Κάθε κόμβος *u* που ανήκει στο *V*, διαθέτει έναν κατάλογο. Στον κατάλογο αυτόν αποθηκεύονται κάποιες χρήσιμες πληροφορίες, όπως ισχύει και στις παραδοσιακές βάσεις δεδομένων. Επιπλέον αποθηκεύεται η πληροφορία σχετικά με τον τύπο (τοπικά αποθηκευμένος, νοητός, υβριδικός) κάθε πίνακα της βάσης δεδομένων, όπως επίσης αποθηκεύονται πληροφορίες οι οποίες αναφέρονται σε όλους τους γνωστούς σε αυτόν κόμβους και είναι οι εξής:

- σε ποιες κοινότητες του u ανήκουν,
- η απόστασή τους από τον u,
- τα χαρακτηριστικά αυτών των κόμβων, δηλαδή
    - διαθεσιμότητα
    - χρόνος απόκρισης και
    - κλάση στην οποία ανήκουν.

Στη συνέχεια θα θεωρήσουμε τον κατάλογο του συστήματος σαν έναν σχεσιακό πίνακα με σχήμα (Peer ID, Κοινότητα στην οποία ανήκει, Απόσταση σε βήματα, Διαθεσιμότητα, Χρόνος απόκρισης, Κλάση στην οποία ανήκει), όπως φαίνεται στα παραδείγματα παρακάτω.

3.2 Διαφορές σε Σχέση με την Παραδοσιακή SQL

Το σύστημά μας έχει κάποια ιδιαίτερα χαρακτηριστικά τα οποία οδηγούν στην ανάγκη να επεκταθεί η παραδοσιακή SQL, έτσι ώστε να έχουμε τη δυνατότητα να πραγματοποιούμε ερωτήσεις. Στη συνέχεια παρατίθενται αυτές οι διαφορές.

Σε αντίθεση με τις παραδοσιακές βάσεις δεδομένων, στο σύστημά μας οι πίνακες δεν είναι τοπικά αποθηκευμένοι σε μία βάση δεδομένων πάντα. Υπάρχουν τρία είδη πινάκων

- οι τοπικά αποθηκευμένοι στη βάση δεδομένων
- οι νοητοί οι οποίοι αποτελούνται από πλειάδες που είναι διασπαρμένες σε ένα σύνολο από peers και γεμίζουν συλλέγοντας τις πλειάδες αυτές και
- οι υβριδικοί οι οποίοι είναι συνδυασμός των δύο παραπάνω κατηγοριών.

Αποτέλεσμα αυτού του γεγονότος είναι ότι όταν πραγματοποιείται μία ερώτηση που περιέχει κάποιον πίνακα των δύο τελευταίων κατηγοριών θα πρέπει πρώτα να γεμίσει

αυτός με τις απαραίτητες πλειάδες και στη συνέχεια να πραγματοποιηθεί η ερώτηση πάνω του.

Στο σύστημά μας υπάρχει ένα σύνολο από peers, οι οποίοι είναι οργανωμένοι σε ένα γράφο. Με βάση αυτόν τον γράφο και κάποια άλλα κριτήρια όπως διαθεσιμότητα, ταχύτητα απόκρισης και κλάση των παρεχόμενων web services που χαρακτηρίζουν τον κάθε peer του γράφου, καθορίζουμε το υποσύνολο των peers που μας ενδιαφέρει να ρωτήσουμε θέτοντας μία ερώτηση.

Ο γράφος των peers μεταβάλλεται δυναμικά, οπότε ενδέχεται κάποιοι από τους peers που ρωτήθηκαν να μην δώσουν απάντηση. Το γεγονός αυτό μας οδηγεί στην αλλαγή της έννοιας του τερματισμού μίας ερώτησης. Έτσι θέτουμε κάποια κριτήρια επιτυχούς τερματισμού μία ερώτησης, όπως να πάρουμε απάντηση από πλήθος peers που ξεπερνούν ένα ποσοστό από τους συνολικούς που ρωτήθηκαν, να συλλέξουμε πλήθος πλειάδων πάνω από ένα όριο που θέτουμε ή να ξεπεραστεί ένα χρονικό όριο που τίθεται ως μέγιστο χρονικό όριο αποδεκτής απάντησης.

Τέλος, διαφορά αποτελεί και το γεγονός ότι υπάρχει η δυνατότητα να πραγματοποιούμε μία ερώτηση συνεχώς ενημερώνοντας την αντίστοιχη απάντηση ανά κάποια περίοδο.

3.3 Απαιτήσεις για Επέκταση της SQL

α. Ορισμός μεγίστου χρόνου αξιοπιστίας της απάντησης της τοπικής βάσης του peer που πραγματοποιεί την ερώτηση.

Μπορούμε, δηλαδή να θέτουμε ένα χρονικό όριο, το οποίο να καθορίζει ότι αν η ερώτηση αναφέρεται σε σχέσεις που ενημερώθηκαν κάποια χρονική στιγμή που βρίσκεται μέσα στο όριο αυτό τότε να θεωρούμε τα περιεχόμενα των σχέσεων πρόσφατα, ενώ στην αντίθετη περίπτωση να τα θεωρούμε παλιά. Στην περίπτωση των πρόσφατων περιεχομένων, την ερώτηση μπορούμε να την εκτελέσουμε αμέσως πάνω στις τοπικές σχέσεις του peer και να πάρουμε τα αντίστοιχα αποτελέσματα. Στην αντίθετη περίπτωση, η ερώτηση για να εκτελεστεί πρέπει πρώτα να ξεκινήσει η όλη διαδικασία συλλογής πλειάδων από τους peers που μας ενδιαφέρουν και να

τροφοδοτηθούν οι σχέσεις με τα νέα περιεχόμενα. Αφού πραγματοποιηθεί αυτή η διαδικασία, η ερώτηση είναι έτοιμη προς εκτέλεση πάνω στις σχέσεις που μόλις γέμισαν με τα απαραίτητα δεδομένα.

β. Καθορισμός του χρονισμού της ερώτησης.
Μία ερώτηση μπορεί να γίνεται:

- είτε ad-hoc (μια φορά)
- είτε συνεχώς.

Στην πρώτη περίπτωση επίσης πρέπει να καθορίσουμε συνθήκες τερματισμού της φάσης συλλογής δεδομένων και έναρξη της φάσης εκτέλεσης της ερώτησης. Η πιο απλή περίπτωση είναι να αναμένουμε απάντηση από όλους και μόνο όταν ολοκληρωθεί ο κύκλος όλων των απαντήσεων και μόνον τότε να τερματίζεται η πρώτη φάση και να αρχίζει η εκτέλεση της ερώτησης. Μία πιο ρεαλιστική προσέγγιση είναι να θέτουμε κάποια κριτήρια πιο χαλαρά για την ολοκλήρωση της φάσης της συλλογής δεδομένων. Πιθανά τέτοια κριτήρια είναι (α) να λάβουμε απαντήσεις από πλήθος peers που ξεπερνούν ένα ποσοστό σε σχέση με όλους τους peers που μας ενδιαφέρουν, (β) να συλλέγουμε πλειάδες μέχρι να ξεπεράσουν κάποιο όριο που έχει τεθεί και (γ) να τερματίζεται η φάση συλλογής δεδομένων όταν κάποιο χρονικό όριο ξεπεραστεί.

Στη δεύτερη περίπτωση κατά την οποία θέτουμε μία ερώτηση να πραγματοποιείται συνεχώς θα πρέπει να καθορίσουμε τον τρόπο εκτέλεσης αυτής της συνεχούς ερώτησης. Διακρίνουμε δύο περιπτώσεις:

- περιοδικά, ανά κάποια περίοδο που καθορίζουμε, ο peer που κάνει την ερώτηση ζητάει από τους peers που τον ενδιαφέρουν να τον τροφοδοτήσουν με πλειάδες (pull-based),
- κάθε peer από αυτούς που λαμβάνουν μέρος στην ερώτηση ενημερώνει τον peer που κάνει την ερώτηση για οποιεσδήποτε αλλαγές συμβούν στις πλειάδες με τις οποίες τον έχει τροφοδοτήσει (push-based).

γ. Καθορισμός σε ποιους peers αναφερόμαστε κάνοντας μία ερώτηση.

Αρχικά μπορούμε να καθορίσουμε γενικά σε ποιους peers τίθεται η ερώτηση. Διακρίνουμε περιπτώσεις ανάλογα με το

(α) αν η ερώτηση αναφέρεται μόνο στον peer που την πραγματοποιεί (τοπικά),

(β) σε κάποια κοινότητα του δεδομένου peer,

(γ) σε όλους τους peers για τους οποίους ισχύει μία συνθήκη που έχει σχέση με το πλήθος των βημάτων που απέχουν από αυτόν,

(δ) σε κάποιους συγκεκριμένους μεμονωμένους peers.

Επίσης μπορούμε να καθορίσουμε τους peers στους οποίους αναφέρεται μία ερώτηση θέτοντας κάποια κριτήρια τα οποία θα πρέπει να ικανοποιούνται από αυτούς. Ένα κριτήριο είναι η διαθεσιμότητα του peer να είναι πάνω από κάποιο όριο, δηλαδή ο peer να είναι ενεργός, ικανός να πραγματοποιήσει τις λειτουργίες του σε ένα ποσοστό χρόνου πάνω από κάποιο όριο. Επίσης μπορούμε να ζητήσουμε ο αναμενόμενος χρόνος απόκρισης σε μία ερώτηση να είναι μικρότερος από κάποιο όριο. Τέλος ένα άλλο κριτήριο αποτελεί η επιλογή των peers οι οποίοι παρέχουν web services που ανήκουν σε μία ή παραπάνω συγκεκριμένη/ες κλάση/εις, ή σε όλες τις κλάσεις.

Η ανάλυση αυτή μπορεί να αναπαρασταθεί σχηματικά με σκοπό την οργάνωση και συμπύκνωση της πληροφορίας που δίνεται παραπάνω, έτσι ώστε να καθίσταται πιο εύκολη η κατανόησή της από τον αναγνώστη (σχήμα 3.5).

--**Ο**ρισμός μεγίστου χρόνου αξιοπιστίας της απάντησης σε μία ερώτηση τοπικά
--**Χ**ρονισμός της ερώτησης
  Συνθήκη τερματισμού μίας Ad-hoc ερώτησης
    - *Π*οσοστό των peers που απάντησαν
    - *Π*λήθος πλειάδων που συλλέχθηκαν
    - *T*imeout που ξεπεράστηκε
  Συνεχείς ερωτήσεις και ανανέωση του αποτελέσματος
    - *P*ull_Based με καθορισμένη περίοδο
    - *P*ush_Based
--**Κ**αθορισμός των peers που συμμετέχουν
  Σε σχέση με τη φυσική τους θέση
    - *Μ*όνο τοπικά
    - *Σ*ε κάποια κοινότητα του peer που πραγματοποιεί την ερώτηση
    - *Σ*ε απόσταση κάποιων βημάτων
    - *Σ*ε συγκεκριμένους peers μεμονωμένα
  Σε σχέση με κάποια κριτήρια
    - *Δ*ιαθεσιμότητα του peer
    - *Τ*αχύτητα απόκρισης του peer
    - *Ε*πιλογή των peers που παρέχουν web services που ανήκουν σε κάποια/ες συγκεκριμένη/ες κλάση/εις, ή σε όλες τις κλάσεις

Σχήμα 3.5: Σύνοψη των Απαιτήσεων για Επέκταση της Παραδοσιακής SQL

3.4 Γενικό Μοντέλο Ερώτησης στην Επεκτεταμένη SQL

Οι απαιτήσεις για επέκταση της SQL που παραθέσαμε παραπάνω, δίνουν τη δυνατότητα σε κάποιον peer να υποβάλλει ερωτήσεις έχοντας παραπάνω δυνατότητες. Με άλλα λόγια ο χρήστης μπορεί να θέτει ερωτήσεις και να μπορεί να καθορίζει ένα σύνολο από επιπρόσθετες παραμέτρους της ερώτησης, οι οποίες δεν αναφέρονται στο σχήμα της βάσης του. Στη ενότητα αυτή δίνεται το γενικό μοντέλο ερώτησης στην επεκτεταμένη SQL που εισάγουμε. Το μοντέλο αυτό φαίνεται στο σχήμα 3.6. Σημειώνουμε ότι μέσα σε [...] υπάρχουν τα προαιρετικά τμήματα αυτού του μοντέλου, τα τμήματα αυτά δηλαδή τα οποία μπορούν να παραληφθούν κατά την εγγραφή κάποιας ερώτησης κι αυτή να παραμένει ορθή. Επίσης η έκφραση *AND ή OR* σημαίνει ότι χρησιμοποιείται το AND ή το OR μόνο στην περίπτωση που απαιτείται, όταν δηλαδή συνδέει δύο συνθήκες μεταξύ τους στο with clause.

SELECT <στήλες προς προβολή>
FROM <πίνακες που συμμετέχουν στην ερώτηση>
[WHERE <συνθήκες που πρέπει να ικανοποιούν οι πλειάδες του αποτελέσματος>]
[GROUP BY <γνωρίσματα ομαδοποίησης>]
[HAVING <συνθήκη ομαδοποίησης>]
[ORDER BY <γνωρίσματα με βάση τα οποία διατάσσονται οι πλειάδες του αποτελέσματος σε φθίνουσα ή αύξουσα διάταξη>]
[WITH
[AGE <μέγιστος χρόνος αξιοπιστίας των περιεχομένων της βάσης δεδομένων τοπικά> ***AND ή OR***]
[**TIMING** <χρονισμός της ερώτησης: είτε *AD-HOC* είτε *CONTINUOUS*
Αν *AD-HOC* καθορίζεται κριτήριο επιτυχούς ερώτησης, είτε ένα *PEERS_PERCENTAGE*, είτε ένα *AMOUNT_TUPLES*, είτε ένα *TIMEOUT*
Αν *CONTINUOUS* τότε επιπλέον καθορίζεται είτε
PULL_BASED WITH PERIOD, είτε *PUSH_BASED*>***AND ή OR****]
[**HORIZON** <καθορισμός των peers που μας ενδιαφέρει να απαντήσουν:
είτε *LOCAL*, είτε *COMMUNITY <όνομα κοινότητας>* είτε *HOPS* < ή > ή = ή <= ή >= *K (*K ακέραιος), είτε PEERS=[$peer_1$,$peer_2$,…, $peer_n$]> ***AND ή OR****]
[**AVAILABILITY** <ελάχιστη διαθεσιμότητα των peers που θα ερωτηθούν> ***AND ή OR****]
[**RESPONSE_TIME** <μέγιστος αποδεκτός χρόνος απάντησης στην ερώτηση> ***AND ή OR****]
[**CLASS** <καθορισμός των peers που μας ενδιαφέρει να απαντήσουν
με βάση την κλάση των web services που παρέχουν> ***AND ή OR****]]

Σχήμα 3.6: Γενικό Μοντέλο Ερώτησης στην Επεκτεταμένη SQL

Στο σημείο αυτό θα αναφέρουμε τις προκαθορισμένες προτάσεις (DEFAULTS) της επεκτεταμένης SQL που εισάγουμε. Δηλαδή όταν κάποια πρόταση του γενικού μοντέλου λείπει από κάποια ερώτηση, ποια έκφραση είναι αυτή που υπονοείται από το σύστημα κατά τη διαδικασία της αποτίμησης της ερώτησης.

(α) Όταν λείπει η πρόταση της ηλικίας των πλειάδων, η προκαθορισμένη έκφραση είναι: AGE <= 0

(β) Όταν λείπει η timing πρόταση, η προκαθορισμένη έκφραση είναι:

TIMING AD-HOC PEERS_PERCENTAGE = 100%

(γ) Όταν λείπει η horizon πρόταση, η προκαθορισμένη έκφραση είναι:

HORIZON HOPS<=0

(δ) Όταν λείπει η availability πρόταση, η προκαθορισμένη έκφραση είναι:

AVAILABILITY >= 0%

(ε) Όταν λείπει η response_time πρόταση, η προκαθορισμένη έκφραση είναι:

RESPONSE_TIME <= ∞

(στ) Όταν λείπει η class πρόταση, η προκαθορισμένη έκφραση είναι:

CLASS = ALL CLASSES

3.4.1 Ερωτήσεις στο Γράφο των Κόμβων

Ερωτήσεις στο γράφο των κόμβων. Έστω ερώτηση Q που υποβάλλεται στον κόμβο u τη χρονική στιγμή T. Έστω $\{R_1, R_2, ..., R_n\}$ οι σχέσεις οι οποίες συμμετέχουν στην ερώτηση Q, δηλαδή που περιέχονται στο FROM clause της ερώτησης. Τότε την ερώτηση μπορούμε να τη γράψουμε στην εξής μορφή: $Q(R_1, R_2, ..., R_n)$. Χωρίς βλάβη της γενικότητας θεωρούμε ότι οι $R_1, ... R_\kappa$, με $k<=n$, είναι νοητές ή υβριδικές σχέσεις. Αυτό που πρέπει να κάνουμε είναι να υλοποιήσουμε (materialize) τις νοητές και υβριδικές σχέσεις και στη συνέχεια να εκτελέσουμε την ερώτηση Q κατά τον παραδοσιακό τρόπο. Για να εξηγήσουμε αυτήν την διαδικασία πρέπει να εισάγουμε πρώτα τις παρακάτω έννοιες.

Σύνολο κόμβων ενδιαφέροντος. Η ερώτηση που μας δίνεται η δυνατότητα να πραγματοποιήσουμε σε κάποιον κόμβο u χωρίζεται σε δύο μέρη. Το πρώτο μέρος αποτελείται από τα clauses της παραδοσιακής SQL, ενώ το δεύτερο τμήμα είναι αυτό που υπάρχει μετά την λέξη κλειδί WITH. Στο δεύτερο τμήμα υπάρχουν συνθήκες που

μπορούμε να εφαρμόσουμε πάνω στην ερώτηση. Αυτές οι συνθήκες συνδέονται με την λέξη AND ή OR κι αν C_1, C_2, …, C_r οι συνθήκες που αναγράφονται στα clauses αυτά, τότε κατά το δεύτερο μέρος της ερώτησης θα πρέπει να αποτιμηθεί μία παράσταση της μορφής: C_1 (AND | OR) C_2 (AND | OR) … (AND | OR) C_r.

Οι συνθήκες που υπάρχουν στο τμήμα της ερώτησης μετά το WITH είναι οι εξής:

- η ηλικία των πλειάδων, που περιέχονται ήδη στους νοητούς ή υβριδικούς πίνακες που συμμετέχουν στην ερώτηση (AGE),
- ο χρονισμός της ερώτησης (TIMING),
- ο ορίζοντας της ερώτησης (HORIZON),
- η διαθεσιμότητα των κόμβων (AVAILABILITY),
- ο χρόνος απόκρισης σε ενδεχόμενη ερώτηση των κόμβων (RESPONSE_TIME) και
- η κλάση στην οποία ανήκουν οι κόμβοι (CLASS).

Από τις παραπάνω συνθήκες, όλες εκτός του χρονισμού (TIMING), καθορίζουν το υποσύνολο των κόμβων που μας ενδιαφέρει να ρωτήσουμε. Δηλαδή υπάρχει μία συνθήκη $C=C_1$ (AND | OR) C_2 (AND | OR) … (AND | OR) C_p, όπου C_i, με $1<=i<=p<=r$, η οποία είναι μία συνθήκη συνδυασμός των clauses AGE, HORIZON, AVAILABILITY, RESPONSE_TIME και CLASS. Επίσης κάθε C_i, με $1<=i<=p<=r$, μπορεί να είναι μία συνθήκη της μορφής: $C_i = a_1$ (AND | OR) a_2 (AND | OR)… (AND | OR) a_q, με $1<=q$. Η εφαρμογή της συνθήκης C, έχει ως αποτέλεσμα ένα σύνολο από κόμβους, οι οποίοι είναι υποσύνολο του συνόλου κόμβων αναφοράς του u (viewpoint(u,T)) και αποτελούν το σύνολο των κόμβων ενδιαφέροντος του κόμβου ο οποίος θέτει την ερώτηση. Ορίζουμε λοιπόν, το σύνολο $V^u(C)$ να είναι το σύνολο των κόμβων που υπάρχουν στον κατάλογο του κόμβου u και για τους οποίους ισχύει η συνθήκη C. Οπότε δημιουργούμε το σύνολο, $V^u(C)$, με $V^u(C)=\{ v \mid v \varepsilon \text{ viewpoint}(u,T): \sigma(v,C)=\text{TRUE}\}$, όπου σ μία συνάρτηση η οποία παίρνει ως ορίσματα έναν κόμβο και μία λογική πρόταση κι επιστρέφει TRUE ή FALSE ανάλογα με το αν η πρόταση αυτή για τον δεδομένο κόμβο, ισχύει ή όχι αντίστοιχα. Παρακάτω θα εξηγηθεί ο τρόπος αποτίμησης της συνθήκης του κάθε clause από αυτά ξεχωριστά.

Συνθήκη χρονισμού μίας ερώτησης. Η συνθήκη αυτή καθορίζει καταρχάς αν η ερώτηση είναι συνεχής ή ad-hoc. Αν είναι συνεχής η ερώτηση καθορίζεται η περίοδος της ερώτησης. Στην περίπτωση αυτή, γεμίζουν με τις απαραίτητες πλειάδες οι νοητοί και υβριδικοί πίνακες πραγματοποιώντας την διαδικασία πλήρης υλοποίησης που θα εξηγήσουμε παρακάτω. Αφού γίνει αυτή η διαδικασία, στη συνέχεια εκτελείται η ερώτηση πάνω σε αυτούς τους πίνακες κατά τα γνωστά. Μετά το πέρας της περιόδου συνεχούς ερώτησης, αρχίζει η διαδικασία εκτέλεσης της ερώτησης από την αρχή. Αν όμως η ερώτηση είναι ad-hoc, καθορίζεται η συνθήκη τερματισμού της ερώτησης. Οπότε οι κόμβοι που έχουν κληθεί μέσω των κατάλληλων web services αυτών, τροφοδοτούν με πλειάδες τους αντίστοιχους νοητούς ή υβριδικούς πίνακες του κόμβου που πραγματοποίησε την κλήση τους, όσο η συνθήκη τερματισμού της ερώτησης αποτιμάται σε false. Μόλις γίνει true, σταματάει η διαδικασία τροφοδοσίας με πλειάδες, οι πίνακες θεωρούνται πλήρεις και εκτελείται κατά τα γνωστά η ερώτηση πάνω στους πίνακες αυτούς. Στην περίπτωση αυτή πραγματοποιείται η μερική υλοποίηση, την οποία θα εξηγήσουμε επίσης παρακάτω. Οι συνθήκη τερματισμού στις ad-hoc ερωτήσεις μπορεί να είναι κάποια από τις εξής:

- να ξεπεραστεί κάποιο χρονικό όριο, το οποίο έχουμε θέσει ως μέγιστο χρονικό όριο συλλογής πλειάδων ή
- να συλλεχθεί πλήθος πλειάδων πάνω από κάποιο όριο, το οποίο είναι ικανοποιητικό έτσι ώστε να θεωρήσουμε τον δεδομένο νοητό ή υβριδικό πίνακα γεμάτο και έτοιμο να λάβει μέρος στην εκτέλεση της ερώτησης στην οποία συμμετέχει ή
- να λάβουμε πλειάδες από ένα σύνολο κόμβων που ξεπερνούν ένα ποσοστό των κόμβων από τους οποίους ζητήθηκε να τροφοδοτήσουν με πλειάδες τον νοητό ή υβριδικό πίνακα.

Έστω c_ad_hoc η συνθήκη που βρίσκεται στο clause του TIMING αν η ερώτηση καθορίζεται ως ad-hoc. Η συνθήκη c_ad_hoc μπορεί να είναι της μορφής:
c_ad_hoc = $c\_ad\_hoc_1$ (AND | OR) ... (AND | OR) $c\_ad\_hoc_s$, με $1<=s<=3$. Η κάθε μία από τις συνθήκες $c\_ad\_hoc_1, \ldots, c\_ad\_hoc_s$, με $1<=s<=3$, μπορεί να είναι μία από τις τρεις προαναφερθείσες κατηγορίες συνθηκών τερματισμού στις ad-hoc ερωτήσεις. Αποτιμούνται και αν η συνολική συνθήκη c_ad_hoc γίνει true, εκείνη τη στιγμή

τερματίζει η διαδικασία τροφοδοσίας με πλειάδες και ξεκινάει η εκτέλεση της ερώτησης κατά τον παραδοσιακό τρόπο. Στη συνέχεια ακολουθεί ανάλυση του τρόπου με τον οποίο αποτιμάται η συνθήκη του δεδομένου clause της ερώτησης.

Mapping των web services των κόμβων στους νοητούς ή υβριδικούς πίνακες. Όταν κάποιος κόμβος *u* πραγματοποιήσει κάποια ερώτηση και ζητήσει από ένα σύνολο κόμβων, έστω u_1, u_2, ..., u_z να του στείλουν τις πλειάδες τους για κάποιον νοητό ή υβριδικό πίνακα, έστω *R*, πραγματοποιείται η εξής διαδικασία. Από τις web services που δημοσιοποιεί ο κάθε κόμβος ότι διαθέτει, καλείται η κατάλληλη κάθε φορά έτσι ώστε να επιστραφεί η πληροφορία που είναι επιθυμητή. Πολλές φορές μία web service δεν είναι αρκετή για αυτήν τη διαδικασία, οπότε απαιτείται συνδυασμός των web services που διαθέτει ένας κόμβος. Οι web services αυτές συνδυάζονται σε ένα workflow το οποίο επιστρέφει το επιθυμητό αποτέλεσμα. Πάντως και στην απλούστερη περίπτωση που απαιτείται μία web service, θεωρούμε ότι είναι ένα workflow από μία web service. Οπότε η ερώτηση ενός κόμβου και η αίτηση για πλειάδες από ένα σύνολο κόμβων έχει ως αποτέλεσμα ένα σύνολο από workflows, ένα για κάθε κόμβο που καλείται. Κάθε ένα από τα workflow αυτά συμβολίζεται ως εξής: $wf^{u.R}(u_i)$, με $1<=i<=z$. Δηλαδή συμβολίζεται ως το workflow του κόμβου u_i που έχει ως σκοπό την τροφοδότηση με πλειάδες της σχέσης *R* του κόμβου *u*. Έστω $R(A_1, A_2, ..., A_v)$ το σχήμα της σχέσης *R*. Η εφαρμογή του workflow $wf^{u.R}(u_i)$ έχει ως αποτέλεσμα ένα σύνολο από εγγραφές με σχήμα $(B_1, B_2, ..., B_\mu)$. Το mapping μεταξύ αυτών των δύο σχημάτων είναι μία συνάρτηση f , με f: $(B_1, B_2, ..., B_\mu) \rightarrow (A_1, A_2, ..., A_v)$. Για κάθε B_j, με $1<=j<=\mu$, υπάρχει 0 ή 1 A_i, με $1<=i<=v$, στο οποίο αντιστοιχίζεται. Η αντιστοίχηση αυτή πραγματοποιείται με τη χρήση μιας βοηθητικής συνάρτησης map, η οποία παίρνει ως ορίσματα ένα A_i, με $1<=i<=v$, κι ένα B_j, με $1<=j<=\mu$ κι επιστρέφει true ή false ανάλογα με το αν τα δύο ορίσματα αναφέρονται στο ίδιο πράγμα ή όχι αντίστοιχα. Εφαρμόζοντας τη συνάρτηση αυτή για όλα τα πιθανά ζεύγη από (A_i, B_j) κρατάμε μόνο αυτά που αποτιμούνται σε true. Στο σχήμα του workflow $wf^{u.R}(u_i)$ αντικαθιστούμε τα B_j με τα αντίστοιχα A_i των ζευγών που αποτιμούνται σε true. Εάν κάποιο B_j δεν έχει κάποιο αντίστοιχο A_i τότε διαγράφεται από το σχήμα του workflow, ενώ όλα τα A_i που δεν έχουν αντίστοιχο B_j, προστίθενται στο σχήμα του workflow με τιμή null. Τέλος πραγματοποιούνται οι κατάλληλες αλλαγές στη σειρά των στοιχείων, αν αυτό απαιτείται, ώστε τα δύο σχήματα να

έρθουν σε πλήρη συμφωνία. Οπότε τελικά πραγματοποιώντας αυτήν την διαδικασία καταφέρνουμε η έξοδος ενός workflow κάποιου κόμβου να παίρνει τη μορφή πλειάδας έτοιμης να εισαχθεί στον πίνακα στόχο. Ακολουθεί παράδειγμα για να γίνει πιο κατανοητή η παραπάνω διαδικασία.

Παράδειγμα: Έστω ο πίνακας R(E_ID,E_SALARY,E_AGE) κι έστω η workflow ενός κόμβου με σχήμα (ID,AGE,NAME). Επιθυμούμε να κάνουμε mapping τα δύο αυτά σχήματα εφαρμόζοντας τη συνάρτηση f, με f από το (ID,AGE,NAME) στο (E_ID,E_SALARY,E_AGE). Εφαρμόζουμε τη συνάρτηση map για όλα τα πιθανά ζεύγη κι έχουμε:

map(E_ID,ID)=true,

map(E_ID,AGE)=false,

map(E_ID,NAME)=false,

map(SALARY,ID)=false,

map(SALARY,AGE)=false,

map(SALARY,NAME)=false,

map(E_AGE,ID)=false,

map(E_AGE,AGE)=true,

map(E_AGE,NAME)=false.

Κρατάμε τα ζεύγη που αποτιμούνται σε true κι αυτά είναι τα εξής:

(E_ID,ID), (E_AGE,AGE).

Στο σχήμα της workflow αντικαθιστούμε όσα στοιχεία περιέχονται στα παραπάνω ζεύγη με τα αντίστοιχα ονόματα του σχήματος της σχέσης *R*. Οπότε το σχήμα της workflow γίνεται: (E_ID,E_AGE,NAME). Στην συνέχεια όλα τα στοιχεία του σχήματος της workflow που δεν υπάρχουν στα ζεύγη που αποτιμούνται σε true. Το στοιχείο NAME δεν υπάρχει σε αυτά, οπότε διαγράφεται και το σχήμα της workflow είναι πια το εξής:

(E_ID,E_AGE). Προσθέτουμε στο σχήμα της workflow τα γνωρίσματα της *R* που δεν υπάρχουν στα ζεύγη των true με τιμές null, επομένως επειδή το γνώρισμα SALARY της *R* δεν υπάρχει σε αυτά το σχήμα της workflow γίνεται:

(E_ID,E_AGE,E_SALARY). Τέλος κάνουμε τις απαραίτητες αλλαγές στη σειρά των στοιχείων του σχήματος της workflow έτσι ώστε να έρθει σε πλήρη συμφωνία με αυτό της *R*. Έτσι αφού γίνει κι αυτό, το τελικό σχήμα της workflow είναι το εξής:

(E_ID,E_SALARY,E_AGE).

Όπως παρατηρούμε μετά το πέρας της διαδικασίας αυτής οι εγγραφές της workflow είναι πια στη μορφή πλειάδων έτοιμων για εισαγωγή στη σχέση-στόχο *R*.

Workflow $wf^{u.R}(u_i)$. Όπως έχει αναφερθεί και παραπάνω το workflow $wf^{u.R}(u_i)$, σημαίνει ένα workflow του κόμβου u_i που τροφοδοτεί πλειάδες τον πίνακα *R* του κόμβου *u*. Ένα workflow του κόμβου u_i είναι η εφαρμογή ενός συνόλου από web services που παρέχει ο κόμβος u_i, οι οποίες σχηματίζουν έναν ακυκλικό κατευθυνόμενο γράφο. Το extension της workflow $wf^{u.R}(u_i)$ είναι το αποτέλεσμα που προκύπτει από την εφαρμογή της δεδομένης workflow, που είναι ένα σύνολο εγγραφών έτοιμων να εισαχθούν στη σχέση *R* του κόμβου *u*.

Πλήρης-Μερική υλοποίηση (materialization). Συμβολίζουμε με $\mu(wf^{u.R}(u_i))$ το γεγονός ότι το workflow $wf^{u.R}(u_i)$ εκτελέστηκε και το extension του workflow αυτού ανήκει πλέον στον πίνακα *R* του κόμβου *u*.

Πλήρης υλοποίηση πραγματοποιείται όταν έχει γίνει επιτυχώς η ενέργεια $\mu(wf^{u.R}(u_i))$ για όλους τους κόμβους u_i, οι οποίοι ανήκουν στο σύνολο κόμβων ενδιαφέροντος του *u* πριν ισχύσει η συνθήκη τερματισμού. Συμβολίζεται με $M(u.R)$ κι έχει την έννοια της πλήρους υλοποίησης της σχέσης *R* του κόμβου *u*.

$M(u.R) = \cup \mu(wf^{u.R}(u_i))$, με $u_i \in V^u(C)$.

Μερική υλοποίηση πραγματοποιείται όταν τη στιγμή που ισχύει η συνθήκη τερματισμού έχει γίνει η ενέργεια $\mu(wf^{u.R}(u_i))$ για κάποιο καθαρό υποσύνολο των κόμβων ενδιαφέροντος του u_i. Μόλις ικανοποιηθεί η συνθήκη τερματισμού της ερώτησης σταματούν όσες ενέργειες $\mu(wf^{u.R}(u_i))$ δεν έχουν προλάβει να ολοκληρωθούν μέχρι τότε. Συμβολίζεται με $M'(u.R)$ κι έχει την έννοια της μερικής υλοποίησης της σχέσης *R* του κόμβου *u*.

$M'(u.R) = \cup \mu(wf^{u.R}(u_i))$, με $u_i \in V'^u(C)$, με $V'^u(C) \subset V^u(C)$ και $V'^u(C)$ είναι το σύνολο των κόμβων του $V^u(C)$ που έχουν προλάβει να στείλουν τα δεδομένα τους πριν ικανοποιηθεί η συνθήκη τερματισμού της ερώτησης.

Εκτέλεση ερώτησης. Στο σημείο αυτό είμαστε σε θέση να δώσουμε τον τρόπο με τον οποίο εκτελείται μία ερώτηση. Όπως αναφέραμε και παραπάνω έστω ότι κάποια χρονική στιγμή *T* τίθεται μία ερώτηση *Q* στον κόμβο *u*. Έστω $\{R_1, R_2, ..., R_n\}$ οι

σχέσεις οι οποίες συμμετέχουν στην ερώτηση Q. Τότε την ερώτηση μπορούμε να τη γράψουμε στην εξής μορφή: $Q(R_1, R_2, ..., R_n)$. Χωρίς βλάβη της γενικότητας θεωρούμε ότι οι $R_1, ..., R_\kappa$, με $k<=n$, είναι νοητές ή υβριδικές σχέσεις. Όλοι οι πίνακες $R_1, ..., R_\kappa$ πρέπει να τροφοδοτηθούν με πλειάδες. Η διαδικασία που ακολουθείται είναι η ίδια για όλους τους πίνακες, οπότε δείχνουμε τη διαδικασία τροφοδοσίας με πλειάδες για τον πίνακα R_1 και το ίδιο ισχύει και για όλους τους υπόλοιπους.

Η πρώτη ενέργεια που πραγματοποιείται είναι η εύρεση του συνόλου κόμβων ενδιαφέροντος του κόμβου u ο οποίος θέτει την ερώτηση ($V^u(C)$), αποτιμώντας τη συνθήκη C η οποία αποτελείται από τις συνθήκες που βρίσκονται στα clauses AGE, HORIZON, AVAILABILITY, RESPONSE_TIME και CLASS πάνω στο σύνολο κόμβων αναφοράς του u (viewpoint(u)).

Έστω ότι $V^u(C) = \{u_1, u_2, ..., u_m\}$. Για κάθε έναν κόμβο από αυτούς πραγματοποιείται κλήση των κατάλληλων web services που διαθέτουν έτσι ώστε να μας δώσουν τις σχετικές εγγραφές κι έστω $wf^{u.R1}(u_1)$, $wf^{u.R1}(u_2)$, ..., $wf^{u.R1}(u_m)$, τα κατάλληλα workflow των κόμβων αυτών.

Το σχήμα κάθε ενός από τα workflow αυτά αντιστοιχίζεται με το σχήμα της σχέσης R_1, η οποία αποτελεί τη σχέση-στόχο. Στη συνέχεια ελέγχεται το clause TIMING, όπου καθορίζεται η ερώτηση σε συνεχής ή ad-hoc και η συνθήκη τερματισμού. Επιχειρείται η υλοποίηση των $wf^{u.R1}(u_i)$ ($\mu(wf^{u.R1}(u_i))$) και πραγματοποιείται πλήρης ή μερική υλοποίηση της R_1 του κόμβου u, ανάλογα με την συνθήκη τερματισμού της ερώτησης, όπως αναλύσαμε παραπάνω.

Πλήρης υλοποίηση: $M(u.R) = \cup \mu(wf^{u.R}(u_i))$, με $u_i \varepsilon V^u(C)$.

Μερική υλοποίηση: $M'(u.R) = \cup \mu(wf^{u.R}(u_i))$, με $u_i \varepsilon V'^u(C)$, με $V'^u(C) \subset V^u(C)$ και $V'^u(C)$ είναι το σύνολο των κόμβων του $V^u(C)$ που έχουν προλάβει να στείλουν τα δεδομένα τους πριν ικανοποιηθεί η συνθήκη τερματισμού της ερώτησης.

Ο πίνακας R_1 έχει τροφοδοτηθεί με τις απαραίτητες πλειάδες κι είναι πια έτοιμος να ερωτηθεί. Η ίδια διαδικασία πραγματοποιείται και για όλους τους υπόλοιπους νοητούς ή υβριδικούς πίνακες, οπότε όλοι οι πίνακες του u είναι έτοιμοι προς

ερώτηση. Στο σημείο αυτό, εκτελείται η ερώτηση που έθεσε ο *u* πάνω στους πίνακες $R_1, R_2, \ldots, R_n$ κατά τον γνωστό και παραδοσιακό τρόπο.

Αποτίμηση της συνθήκης των clauses AGE, HORIZON, AVAILABILITY, RESPONSE_TIME, CLASS.

HORIZON: Η συνθήκη του clause HORIZON καθορίζει τους κόμβους που μας ενδιαφέρει να ερωτηθούν και οι δυνατότητες που υπάρχουν είναι οι εξής:
(α) η ερώτηση να αναφέρεται μόνο στον peer που την πραγματοποιεί (τοπικά),
(β) σε κάποια κοινότητα του δεδομένου peer,
(γ) σε όλους τους peers για τους οποίους ισχύει μία συνθήκη που έχει σχέση με το πλήθος των βημάτων που απέχουν από αυτόν,
(δ) σε κάποιους συγκεκριμένους μεμονωμένους peers.

Έστω C_1 η συνθήκη του δεδομένου clause, η εφαρμογή της δεδομένης συνθήκης γίνεται πάνω στο σύνολο κόμβων αναφοράς του *u* και έχει σαν αποτέλεσμα ένα υποσύνολο αυτού που περιέχει όλους τους κόμβους του viewpoint(u,T) οι οποίοι ικανοποιούν τη συνθήκη C_1. Έστω $VH^u(C_1)$ το σύνολο των κόμβων που προκύπτει, τότε ανάλογα με την περίπτωση από (α) μέχρι (ε) η C_1 και το αντίστοιχο $VH^u(C_1)$ είναι:
α) C_1=LOCAL
$VH^u(C_1)=\{u\}$.
β) C_1=COMMUNITY <C_NAME>
$VH^u(C_1)=\{v \mid v \in \text{viewpoint}(u,T): v \in \text{community}(\text{C\_NAME},u)\}$.
γ) C_1=HOPS φ value, με φ={=,<,<=,>,>=}
$VH^u(C_1)=\{v \mid v \in \text{viewpoint}(u,T): \text{distance}(u,v)\ \delta\ \text{value}\}$, με δ={=,<,<=,>,>=}.
δ) C_1=PEERS={$\text{peer}_1, \text{peer}_2, \ldots, \text{peer}_n$}
$VH^u(C_1)=\{v \mid v \in \text{viewpoint}(u,T): v \in \{ \text{peer}_1, \text{peer}_2, \ldots, \text{peer}_n \}\}$.
Η πληροφορία για την κατασκευή του συνόλου $VH^u(C_1)$ βρίσκεται στον κατάλογο του *u*, στον οποίο περιέχεται η απαραίτητη πληροφορία για όλους τους κόμβους οι οποίοι είναι γνωστοί στον *u*.

AVAILABILITY: Η συνθήκη στο clause AVAILABILITY καθορίζει τους κόμβους που μας ενδιαφέρει να ερωτηθούν σε σχέση με τη διαθεσιμότητα που τους χαρακτηρίζει. Η πληροφορία σχετικά με τη διαθεσιμότητα των κόμβων που είναι γνωστοί στον *u* βρίσκεται στον κατάλογό του. Έστω C_2 η συνθήκη του δεδομένου clause, η οποία είναι της μορφής C_2=AVAILABILITY θ value, με θ={=,>,>=}. Η εφαρμογή της δεδομένης συνθήκης γίνεται πάνω στο σύνολο κόμβων αναφοράς του *u* και έχει σαν αποτέλεσμα ένα υποσύνολο αυτού που περιέχει όλους τους κόμβους του viewpoint(*u,T*) οι οποίοι ικανοποιούν τη συνθήκη C_2. Έστω $VA^u(C_2)$ το σύνολο των κόμβων που προκύπτει, availability(v) η διαθεσιμότητα που χαρακτηρίζει τον δεδομένο κόμβο και value είναι η τιμή της διαθεσιμότητας που δίνεται στην ερώτηση, τότε:

$VA^u(C_2) = \{ v \mid v \varepsilon$ viewpoint(u,T): availability(v) θ value$\}$, με θ={=,>,>=}.

RESPONSE_TIME: Η συνθήκη στο clause RESPONSE_TIME καθορίζει τους κόμβους που μας ενδιαφέρει να ερωτηθούν σε σχέση με το χρόνο απόκρισης που τους χαρακτηρίζει. Η πληροφορία σχετικά με το χρόνο απόκρισης των κόμβων που είναι γνωστοί στον *u* βρίσκεται στον κατάλογό του. Έστω C_3 η συνθήκη του δεδομένου clause, η οποία είναι της μορφής C_3=RESPONSE_TIME θ_1 value, με θ_1={=,<,<=}. Η εφαρμογή της δεδομένης συνθήκης γίνεται πάνω στο σύνολο κόμβων αναφοράς του *u* και έχει σαν αποτέλεσμα ένα υποσύνολο αυτού που περιέχει όλους τους κόμβους του viewpoint(*u,T*) οι οποίοι ικανοποιούν τη συνθήκη C_3. Έστω $VRT^u(C_3)$ το σύνολο των κόμβων που προκύπτει, response_time(v) ο χρόνος απόκρισης που χαρακτηρίζει το δεδομένο κόμβο και value η τιμή του χρόνου απόκρισης που δίνεται στην ερώτηση, τότε:

$VRT^u(C_3) = \{ v \mid v \varepsilon$ viewpoint(u,T): response_time(v) θ_1 value$\}$, με θ_1={=,<,<=}.

CLASS: Η συνθήκη στο clause CLASS καθορίζει τους κόμβους που μας ενδιαφέρει να ερωτηθούν σε σχέση με την κλάση στην οποία ανήκουν. Η πληροφορία σχετικά με την κλάση στην οποία ανήκουν οι κόμβοι που είναι γνωστοί στον *u* βρίσκεται στον κατάλογό του. Έστω C_4 η συνθήκη του δεδομένου clause, η οποία είναι της μορφής C_4=CLASS θ_2 string, με θ_2={=}. Η εφαρμογή της δεδομένης συνθήκης γίνεται πάνω στο σύνολο κόμβων αναφοράς του *u* και έχει σαν αποτέλεσμα ένα υποσύνολο αυτού που περιέχει όλους τους κόμβους του viewpoint(*u,T*) οι οποίοι ικανοποιούν τη

συνθήκη C_4. Έστω $VC^u(C_4)$ το σύνολο των κόμβων που προκύπτει, class(v) η κλάση στην οποία ανήκει ο δεδομένος κόμβος και string το όνομα της κλάσης που δίνεται στην ερώτηση, τότε:

$VC^u(C_4) = \{ v \mid v \varepsilon \text{ viewpoint}(u,T)\text{: class(v)} = \text{string}\}$.

AGE: Ο έλεγχος πραγματοποιείται στις πλειάδες των πινάκων της ερώτησης που είναι νοητοί ή υβριδικοί.

Κατά την πρώτη φάση από τις πλειάδες που περιέχει ο κάθε ένας από αυτούς τους πίνακες, κρατιούνται μόνο αυτές που αναφέρονται σε κόμβους οι οποίοι ανήκουν στο σύνολο κόμβων αναφοράς του u. Για τους νοητούς ή υβριδικούς πίνακες $R_1, \ldots R_\kappa$, με $k<=n$, οι πλειάδες τους έστω ότι είναι οι $[r_{11}, r_{12}, \ldots, r_{1a}]$, $[r_{21}, r_{22}, \ldots, r_{2b}]$, …, $[r_{\kappa 1}, r_{\kappa 2}, \ldots, r_{\kappa c}]$ αντίστοιχα. Πραγματοποιείται προβολή όλων των πλειάδων ως προς το αναγνωριστικό του κόμβου στον οποίο αναφέρεται η πλειάδα και σύγκριση με τους κόμβους του συνόλου viewpoint(u,T). Αν η σύγκριση δεν είναι επιτυχής η δεδομένη πλειάδα διαγράφεται από τον πίνακα που ανήκει, ενώ σε αντίθετη περίπτωση μένει ως έχει.

Σε δεύτερη φάση πραγματοποιείται ένας επιπλέον έλεγχος για όλες τις πλειάδες που έχουν απομείνει στον κάθε νοητό ή υβριδικό πίνακα μετά το πέρας της πρώτης φάσης. Συγκρίνεται η διάρκεια παραμονής της πλειάδας στον πίνακα που ανήκει (T_1), με τη χρονική διάρκεια που αναφέρεται στην ερώτηση στο clause AGE. Η διάρκεια παραμονής μίας πλειάδας σε κάποιον πίνακα ορίζεται ως εξής: αφαιρείται από τη χρονική στιγμή που τέθηκε η ερώτηση (T) το χρονόσημο που συνοδεύει την κάθε πλειάδα που υποδηλώνει πότε η δεδομένη πλειάδα εισάχθηκε στον πίνακα που ανήκει. Αν η σύγκριση αυτή έχει ως αποτέλεσμα η διάρκεια παραμονής της πλειάδας να είναι μικρότερη από τη χρονική διάρκεια που αναφέρεται στο AGE clause της ερώτησης, τότε η πλειάδα αυτή παραμένει ως έχει στον πίνακα που βρίσκεται και δεν ξεκινάει διαδικασία αίτησης προς τον κόμβο που αναφέρεται αυτή η πλειάδα. Στην αντίθετη περίπτωση, η δεδομένη πλειάδα διαγράφεται από τον πίνακα, θεωρώντάς την μη πρόσφατη και ξεκινάει η διαδικασία αίτησης προς τον κόμβο αναφοράς της πλειάδας.

Πίνακας 3.1: Πίνακας Επεξήγησης Συμβόλων της Ενότητας

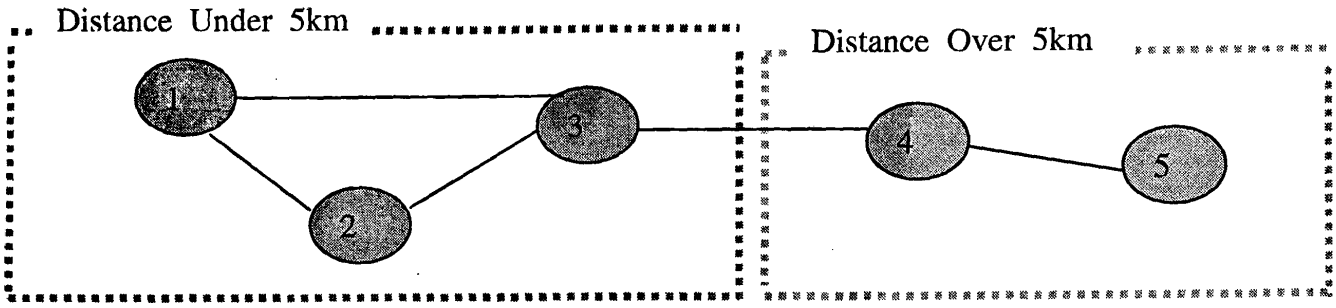
G	Γράφος
V	Σύνολο κόμβων
E	Σύνολο ακμών
u	Κόμβος
T	Χρονική στιγμή
R	Σχέση-Πίνακας της Β.Δ.
Q	Ερώτηση
viewpoint(u,T)	Σύνολο κόμβων αναφοράς του u
community(u)	Κοινότητα του u
WS^u	Σύνολο των web services του u
$Class_i$	Σύνολο κόμβων που ανήκουν στην κλάση i
C_1	Συνθήκη του clause HORIZON πάνω στο σύνολο κόμβων αναφοράς
$VH^u(C_1)$	Υποσύνολο κόμβων αναφοράς του u μετά την εφαρμογή της συνθήκης του clause HORIZON
C_2	Συνθήκη του clause AVAILABILITY πάνω στο σύνολο κόμβων αναφοράς
$VA^u(C_2)$	Υποσύνολο κόμβων αναφοράς του u μετά την εφαρμογή της συνθήκης του clause AVAILABILITY
C_3	Συνθήκη του clause RESPONSE_TIME πάνω στο σύνολο κόμβων αναφοράς
$VRT^u(C_3)$	Υποσύνολο κόμβων αναφοράς του u μετά την εφαρμογή της συνθήκης του clause RESPONSE_TIME
C_4	Συνθήκη του clause CLASS πάνω στο σύνολο κόμβων αναφοράς
$VC^u(C_4)$	Υποσύνολο κόμβων αναφοράς του u μετά την εφαρμογή της συνθήκης του clause CLASS
C_5	Συνθήκη του clause AGE πάνω στο σύνολο κόμβων αναφοράς
$VAGE^u(C_5)$	Υποσύνολο κόμβων αναφοράς του u μετά την εφαρμογή της συνθήκης του clause AGE
C	Συνθήκη των clauses AGE, HORIZON, AVAILABILITY, RESPONSE_TIME, CLASS πάνω στο σύνολο κόμβων αναφοράς
$V^u(C)$	Σύνολο κόμβων ενδιαφέροντος του u
$wf^{u.R}(u_i)$	Workflow του u_i με στόχο την R του u
$\mu(wf^{u.R}(u_i))$	Εκτέλεση της $wf^{u.R}(u_i)$ και εισαγωγή του αποτελέσματος αυτής στη R του u
$M(u.R)$	Πλήρης υλοποίηση της R του u
$M'(u.R)$	Μερική υλοποίηση της R του u

Η συνθήκη αυτή εφαρμόζεται για κάθε νοητό ή υβριδικό πίνακα της ερώτησης κι έχει ως αποτέλεσμα την κατασκευή ενός συνόλου κόμβων, που είναι υποσύνολο του συνόλου κόμβων αναφοράς του u, το οποίο περιέχει τους κόμβους από τους οποίους πρέπει να ζητηθούν πλειάδες. Η συνθήκη, έστω C_5, είναι της μορφής C_5 = AGE θ_3

value, με θ_3={<,<=}. Έστω $VAGE^{u.Ri}(C_5)$ το σύνολο που προκύπτει από την εφαρμογή της C_5, age(v) η ηλικία της πλειάδας στον πίνακα R_i που αναφέρεται στον κόμβο v και value η τιμή της ηλικίας των πλειάδων που δίνεται στην ερώτηση, τότε:
$VAGE^{u.Ri}(C_5)$ ={ v | v ε viewpoint(u,T): age(v) θ_3 value}, με θ_3={<,<=} και age(v)=T-$TS^{u.Ri}$, με T η χρονική στιγμή που τέθηκε η ερώτηση και $TS^{v.Ri}$ το χρονόσημο της πλειάδας του πίνακα R_i που αναφέρεται στον κόμβο v. Κατασκευάζονται m τέτοια σύνολα, ένα για κάθε νοητό ή υβριδικό πίνακα που συμμετέχει στην ερώτηση. Η ένωση όλων αυτών αποτελεί το σύνολο των κόμβων που τελικά απαιτείται να ερωτηθούν λόγω της συνθήκης του clause AGE της ερώτησης.
$VAGE^{u}(C_5)= \cup VAGE^{u.Ri}(C_5)$, με 1<=i<=m.
Οπότε η συνολική συνθήκη C είναι συνδυασμός των συνθηκών C_1, C_2, C_3, C_4,C_5 και το σύνολο κόμβων ενδιαφέροντος του u ($V^u(C)$) προκύπτει με τη βοήθεια των συνόλων $VH^u(C_5)$, $VA^u(C_2)$, $VRT^u(C_3)$, $VC^u(C_4)$ και $VAGE^u(C_5)$.

3.4.2 Παραδείγματα

Παρουσιάζεται στη συνέχεια μία σειρά από παραδείγματα ερωτήσεων στα οποία χρησιμοποιείται η επέκταση της SQL, της οποίας το γενικό μοντέλο ερώτησης αναλύσαμε στην προηγούμενη ενότητα. Η τοπολογία του συστήματος αναφοράς δίνεται στο σχήμα 3.7. Αποτελείται από 5 peers, οι οποίοι αριθμούνται από το ένα μέχρι το πέντε και ερωτήσεις πραγματοποιεί ο peer 1. Οι peers αυτοί οργανώνονται σε δύο κοινότητες του peer 1, στην κοινότητα με τους σκουρόχρωμους peers (Distance_Under_5km) και στην κοινότητα με τους ανοιχτόχρωμους peers (Distance_Over_5km). Επίσης οι δύο κοινότητες διακρίνονται με το αντίστοιχου χρώματος παραλληλόγραμμο σχεδιασμένο με διακεκομμένη γραμμή γύρω τους. Οι peers συνδέονται μέσω ακμών, οι οποίες σημειώνονται με χρώμα μαύρο. Η κάθε ακμή ισοδυναμεί με απόσταση ενός βήματος ανάμεσα στους peers που ενώνει.



Σχήμα 3.7: Τοπολογία του Συστήματος Αναφοράς

Το σχήμα της βάσης του κάθε peer που συμπεριλαμβάνεται στο σύστημα αναφοράς αποτελείται από τους πίνακες που φαίνονται αναλυτικά μαζί με τα γνωρίσματά τους στο σχήμα 3.8. Αξίζει να αναφέρουμε ότι η σχέση BRANDS είναι αποθηκευμένη τοπικά σε κάθε peer ξεχωριστά, σε αντίθεση με τη σχέση CARS η οποία είναι υβριδική σχέση κι αποτελείται από πλειάδες οι οποίες υπάρχουν τοπικά στον peer που εκτελεί την ερώτηση κι από πλειάδες οι οποίες συλλέγονται από όλους τους peers στους οποίους αναφέρεται η ερώτηση. Επίσης το γνώρισμα BRAND του πίνακα CARS αποτελεί ξένο κλειδί με αναφορά στο γνώρισμα BRAND του πίνακα BRANDS. Ακόμη σε κάθε peer υπάρχει ένας κατάλογος στον οποίο αναγράφονται όλοι οι peers που είναι γνωστοί σε αυτόν συμπεριλαμβανομένου και του εαυτού του, συνοδευόμενοι με την κοινότητα στην οποία ανήκουν και το πλήθος των βημάτων που απέχουν από αυτόν. Ακόμη σε κάθε peer, στον κατάλογο υπάρχουν για κάθε γνωστό του peer τρία επιπλέον χαρακτηριστικά: η διαθεσιμότητά του, ο αναμενόμενος χρόνος απόκρισης και η κλάση στην οποία ανήκουν οι web services που παρέχει. Λόγω του γεγονότος ότι ο peer 1 πραγματοποιεί τις ερωτήσεις δίνουμε μόνο τον κατάλογο του peer 1 και αντίστοιχα μπορεί να αντιληφθεί κανείς την μορφή του καταλόγου και στους υπόλοιπους peers. Ο κατάλογος του peer 1 δίνεται στον Πίνακα 3.2.

CARS(ID, PLATE, *BRAND*, VEL)

ID: ένας ακέραιος αριθμός που αποτελεί αναγνωριστικό του κάθε αυτοκινήτου

PLATE: ο αριθμός κυκλοφορίας του αυτοκινήτου

BRAND: η μάρκα του αυτοκινήτου

VEL: η ταχύτητα του αυτοκινήτου σε χιλιόμετρα ανά ώρα

BRANDS(BRAND, COUNTRY, METRICS_SYSTEM)

BRAND: μία μάρκα αυτοκινήτων

COUNTRY: η χώρα κατασκευής της συγκεκριμένης μάρκας αυτοκινήτων

METRICS_SYSTEM: το σύστημα μετρικής που χρησιμοποιείται για να μετρήσει διάφορα χαρακτηριστικά των αυτοκινήτων της δεδομένης μάρκας

Σχήμα 3.8: Το Σχήμα της Βάσης του κάθε Peer του Συστήματος Αναφοράς

Πίνακας 3.2: Τα Περιεχόμενα του Καταλόγου του Peer 1

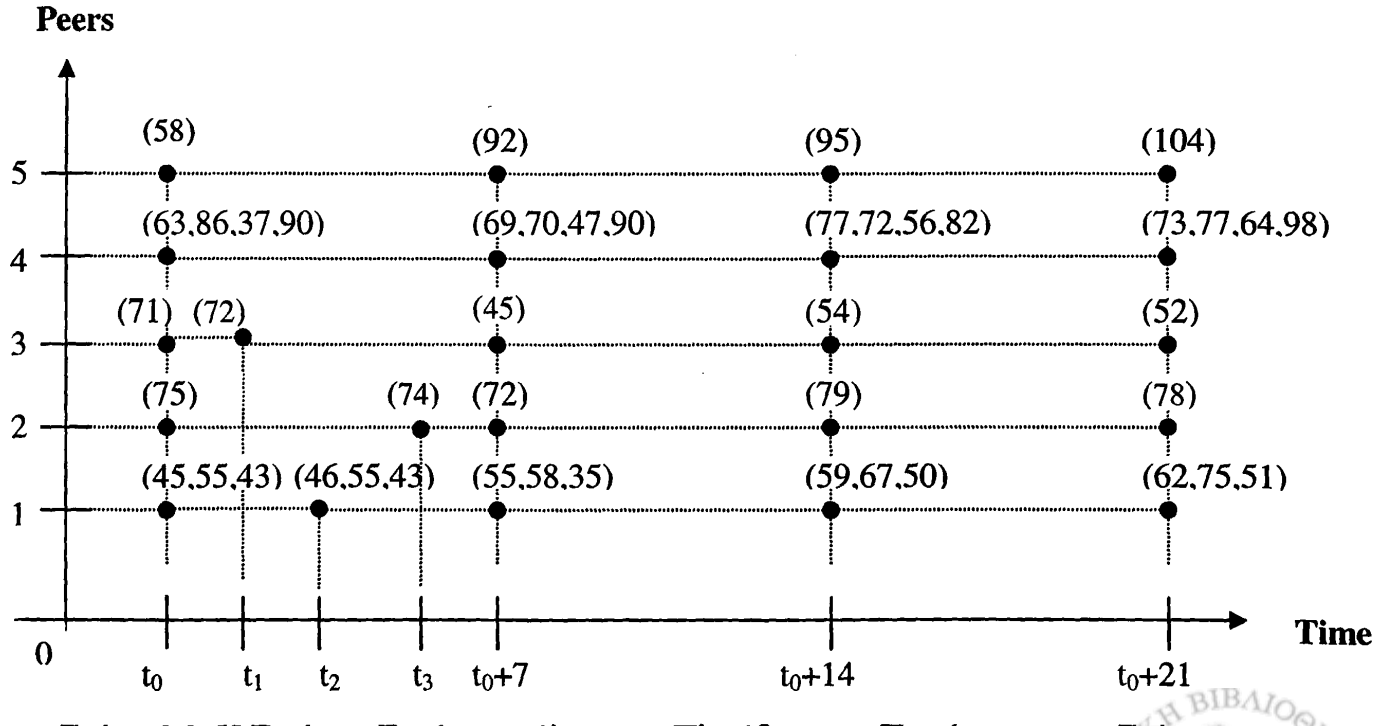
Peer ID	*Κοινότητα που ανήκει*	*Απόσταση σε βήματα*	*Διαθεσιμότητα*	*Χρόνος απόκρισης*	*Κλάση web serv.*
1	Distance_Under_5km	0	80%	0	European
2	Distance_Under_5km	1	75%	2.3	American
3	Distance_Under_5km	1	63%	2.7	European
4	Distance_Over_5km	2	97%	3.9	American
5	Distance_Over_5km	3	91%	4.5	European

Θεωρούμε ότι οι ερωτήσεις που αναφέρονται στα παραδείγματα πραγματοποιούνται τη χρονική στιγμή $t=t_0$. Αυτήν τη χρονική στιγμή δίνεται το στιγμιότυπο των βάσεων δεδομένων των peers του συστήματος. Το στιγμιότυπο της σχέσης BRANDS είναι ίδιο σε όλους τους peers και παραμένει σταθερό σε σχέση με το χρόνο, μιας και είναι σχέση που ανήκει στην κατηγορία των τοπικά αποθηκευμένων στη βάση δεδομένων σχέσεων (Πίνακας 3.3).

Πίνακας 3.3: Στιγμιότυπο της Σχέσης BRANDS (ίδιο για όλους τους Peers)

BRAND	COUNTRY	METRICS_SYSTEM
PEUGEUT	FRANCE	SI
RENAULT	FRANCE	SI
VOLKWAGEN	GERMANY	SI
FORD	GERMANY	SI
OPEL	GERMANY	SI
CHEVROLET	U.S.A.	SI
CRYSLER	U.S.A	SI
NISSAN	JAPAN	SI
TOYOTA	JAPAN	SI

Στο σχήμα 3.9 παρουσιάζεται, με βάση το στιγμιότυπο της σχέσης CARS σε κάθε έναν από τους peers του συστήματος, η τιμή του γνωρίσματος VEL όλων των πλειάδων που περιέχονται σε κάθε μία από τις σχέσεις αυτές. Οι τιμές αυτές δίνονται για τις χρονικές στιγμές $t=t_0$, $t=t_0+7$, $t=t_0+14$, $t=t_0+21$ (συνεχής-περιοδική ερώτηση) όπως επίσης και για τις τρεις πρώτες αλλαγές χρονικά (t_1, t_2, t_3) που συντελούνται στις πλειάδες κάποιου peer που ανήκει στο σύστημά μας.



Σχήμα 3.9: Η Τιμή της Ταχύτητας όλων των Πλειάδων που Περιέχονται στη Σχέση CARS καθενός από τους Peers του Συστήματος για διάφορες Χρονικές Στιγμές.

Με σκοπό την καλύτερη κατανόηση της επεκτεταμένης SQL που εισάγουμε και με βάση το γενικό μοντέλο ερώτησης που δόθηκε, θα παραθέσουμε ένα σύνολο από παραδείγματα ερωτήσεων που πραγματοποιεί ο peer 1 (τη χρονική στιγμή $t=t_0$). Επίσης με βάση τα στιγμιότυπα των σχέσεων των peers του συστήματος μετά την ερώτηση θα ακολουθεί και η αντίστοιχη απάντηση.

Παράδειγμα 0: Δίνεται μία ερώτηση της κλασσικής SQL η οποία θα αποτελέσει βάση για τις επόμενες ερωτήσεις.

Ερώτηση: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής όλων των αυτοκινήτων.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
```

Απάντηση: Από τον κατάλογο του peer 1 όλοι οι peers ικανοποιούν τις συνθήκες της ερώτησης. Με βάση τις πλειάδες των σχέσεων, αφού πρώτα όλοι οι peers τροφοδοτήσουν με τις δικές τους πλειάδες τη σχέση CARS του peer 1, το αποτέλεσμα είναι το εξής:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
GRT8724	75	FRANCE
REB1543	71	GERMANY
BAJ8687	63	JAPAN
GAV2991	58	JAPAN
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.
ASD7295	86	GERMANY
QAF1211	37	U.S.A.
ABC1111	90	FRANCE

Παράδειγμα 1: Με βάση το παράδειγμα 0 θα δώσουμε μία ερώτηση η οποία αποσκοπεί να αναδείξει το γεγονός ότι στην επεκτεταμένη SQL που εισάγουμε έχουμε τη δυνατότητα να καθορίσουμε τους peers που μας ενδιαφέρει να ρωτήσουμε. Ο καθορισμός αυτός μπορεί να γίνει με βάση την θέση τους σε σχέση με τον peer που θέτει την ερώτηση.

Ερώτηση: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής όλων των αυτοκινήτων τα οποία βρίσκονται σε απόσταση 2 και παραπάνω βημάτων και μόνον αυτά.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
HORIZON HOPS>=2
```

Απάντηση: Από τον κατάλογο του peer 1 μόνο αυτοί που είναι σημειωμένοι με μεγάλα και έντονα γράμματα ικανοποιούν τη συνθήκη της πρότασης HORIZON, οπότε μόνο σε αυτούς τίθεται η ερώτηση.

Κατάλογος του peer 1

Peer ID	*Κοινότητα που ανήκει*	*Απόσταση σε βήματα*	*Διαθεσιμότητα*	*Χρόνος απόκρισης*	*Κλάση web serv.*
1	Distance_Under_5km	0	80%	0	European
2	Distance_Under_5km	1	75%	2.3	American
3	Distance_Under_5km	1	63%	2.7	European
4	**Distance_Over_5km**	**2**	**97%**	**3.9**	**American**
5	**Distance_Over_5km**	**3**	**91%**	**4.5**	**European**

Με βάση την παραπάνω επιλογή, οι επιλεχθέντες peers επερωτούνται και απαντούν αποστέλλοντας τις σχετικές πλειάδες. Έτσι το αποτέλεσμα που συλλέγεται είναι ως εξής:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
BAJ8687	63	JAPAN
GAV2991	58	JAPAN
ASD7295	86	GERMANY
QAF1211	37	U.S.A.
ABC1111	90	FRANCE

Παράδειγμα 2: Με βάση το παράδειγμα 0 θα δώσουμε μία ερώτηση η οποία αποσκοπεί να αναδείξει το γεγονός ότι στην επεκτεταμένη SQL που εισάγουμε έχουμε τη δυνατότητα να καθορίσουμε τους peers που μας ενδιαφέρει να ρωτήσουμε. Ο καθορισμός αυτός μπορεί να γίνει με βάση την διαθεσιμότητα που χαρακτηρίζει τον κάθε peer.

Ερώτηση: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής όλων των αυτοκινήτων τα οποία χαρακτηρίζονται από διαθεσιμότητα μεγαλύτερη ίση από 80%.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
AVAILABILITY >= 80 %
```

Απάντηση: Από τον κατάλογο του peer 1 μόνο αυτοί που είναι σημειωμένοι με μεγάλα έντονα γράμματα ικανοποιούν τη συνθήκη της πρότασης AVAILABILITY, οπότε μόνο σε αυτούς τίθεται η ερώτηση.

Κατάλογος του peer 1

Peer ID	*Κοινότητα που ανήκει*	*Απόσταση σε βήματα*	*Διαθεσιμότητα*	*Χρόνος απόκρισης*	*Κλάση web serv.*
1	**Distance_Under_5km**	**0**	**80%**	**0**	**European**
2	Distance_Under_5km	1	75%	2.3	American
3	Distance_Under_5km	1	63%	2.7	European
4	**Distance_Over_5km**	**2**	**97%**	**3.9**	**American**
5	**Distance_Over_5km**	**3**	**91%**	**4.5**	**European**

Με βάση την παραπάνω επιλογή, οι επιλεχθέντες peers επερωτούνται και απαντούν αποστέλλοντας τις σχετικές πλειάδες. Έτσι το αποτέλεσμα που συλλέγεται είναι ως εξής:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
BAJ8687	63	JAPAN
GAV2991	58	JAPAN
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.
ASD7295	86	GERMANY
QAF1211	37	U.S.A.
ABC1111	90	FRANCE

Παράδειγμα 3: Με βάση το παράδειγμα 0 θα δώσουμε μία ερώτηση η οποία αποσκοπεί να αναδείξει το γεγονός ότι στην επεκτεταμένη SQL που εισάγουμε έχουμε τη δυνατότητα να καθορίσουμε τους peers που μας ενδιαφέρει να ρωτήσουμε. Ο καθορισμός αυτός μπορεί να γίνει με βάση τον χρόνο απόκρισης που χαρακτηρίζει τον κάθε peer.

Ερώτηση: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής όλων των αυτοκινήτων τα οποία αναμένεται να απαντήσουν σε χρόνο μικρότερο από 3 δευτερόλεπτα.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
RESPONSE_TIME < 3.0
```

Απάντηση: Από τον κατάλογο του peer 1 μόνο αυτοί που είναι σημειωμένοι με μεγάλα και έντονα γράμματα ικανοποιούν τη συνθήκη της πρότασης RESPONSE_TIME, οπότε μόνο σε αυτούς τίθεται η ερώτηση.

Κατάλογος του peer 1

Peer ID	*Κοινότητα που ανήκει*	*Απόσταση σε βήματα*	*Διαθεσιμότητα*	*Χρόνος απόκρισης*	*Κλάση web serv.*
1	**Distance_Under_5km**	**0**	**80%**	**0**	**European**
2	**Distance_Under_5km**	**1**	**75%**	**2.3**	**American**
3	**Distance_Under_5km**	**1**	**63%**	**2.7**	**European**
4	Distance_Over_5km	2	97%	3.9	American
5	Distance_Over_5km	3	91%	4.5	European

Με βάση την παραπάνω επιλογή, οι επιλεχθέντες peers επερωτούνται και απαντούν αποστέλλοντας τις σχετικές πλειάδες. Έτσι το αποτέλεσμα που συλλέγεται είναι ως εξής:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
GRT8724	75	FRANCE
REB1543	71	GERMANY
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.

Παράδειγμα 4: Με βάση το παράδειγμα 0 θα δώσουμε μία ερώτηση η οποία αποσκοπεί να αναδείξει το γεγονός ότι στην επεκτεταμένη SQL που εισάγουμε έχουμε τη δυνατότητα να καθορίσουμε τους peers που μας ενδιαφέρει να ρωτήσουμε. Ο καθορισμός αυτός μπορεί να γίνει με βάση την κλάση στην οποία ανήκουν οι web services των peers. Έτσι επιλέγουμε αυτούς τους peers οι οποίοι παρέχουν web services που ανήκουν στην ίδια κλάση με αυτές του peer που θέτει την ερώτηση.

Ερώτηση: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής όλων των αυτοκινήτων τα οποία παρέχουν web services της κλάσης European.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
CLASS = "european"
```

Απάντηση: Από τον κατάλογο του peer 1 μόνο αυτοί που είναι σημειωμένοι με μεγάλα έντονα γράμματα ικανοποιούν τη συνθήκη της πρότασης CLASS, οπότε μόνο σε αυτούς τίθεται η ερώτηση.

Κατάλογος του peer 1

Peer ID	*Κοινότητα που ανήκει*	*Απόσταση σε βήματα*	*Διαθεσιμότητα*	*Χρόνος απόκρισης*	*Κλάση web serv.*
1	**Distance_Under_5km**	**0**	**80%**	**0**	**European**
2	Distance_Under_5km	1	75%	2.3	American
3	**Distance_Under_5km**	**1**	**63%**	**2.7**	**European**
4	Distance_Over_5km	2	97%	3.9	American
5	**Distance_Over_5km**	**3**	**91%**	**4.5**	**European**

Με βάση την παραπάνω επιλογή, οι επιλεχθέντες peers επερωτούνται και απαντούν αποστέλλοντας τις σχετικές πλειάδες. Έτσι το αποτέλεσμα που συλλέγεται είναι ως εξής:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
REB1543	71	GERMANY
GAV2991	58	JAPAN
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.

Παράδειγμα 5: Με τα παραπάνω παραδείγματα καλύπτουμε τις διαφορετικές περιπτώσεις κατά τις οποίες μπορούμε να καθορίσουμε τους peers στους οποίους αναφέρεται η ερώτηση. Ο καθορισμός αυτός μπορεί να γίνει με βάση τη θέση των peers σε σχέση με αυτόν που πραγματοποιεί την ερώτηση, τη διαθεσιμότητα, τον χρόνο απόκρισης που χαρακτηρίζει τον κάθε peer και με βάση την κλάση στην οποία ανήκουν οι web services του κάθε peer σε σχέση με αυτές του peer που πραγματοποιεί την ερώτηση.

Ερώτηση: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής όλων των αυτοκινήτων τα οποία ανήκουν στην κοινότητα του peer που θέτει την ερώτηση με όνομα Distance_Under_5km, χαρακτηρίζονται από διαθεσιμότητα μεγαλύτερη από 60%, αναμένεται να απαντήσουν σε χρόνο μικρότερο από 4 δευτερόλεπτα και παρέχουν web services κλάσης european.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
HORIZON COMMUNITY <Distance_Under_5km> AND
AVAILABILITY > 60% AND
RESPONSE_TIME < 4.0 AND
CLASS = "european"
```

Απάντηση: Από τον κατάλογο του peer 1 μόνο αυτοί που είναι σημειωμένοι με μεγάλα έντονα γράμματα ικανοποιούν τις συνθήκες των προτάσεων HORIZON, AVAILABILITY, RESPONSE_TIME και CLASS, οπότε μόνο σε αυτούς τίθεται η ερώτηση.

Κατάλογος του peer 1

Peer ID	*Κοινότητα που ανήκει*	*Απόσταση σε βήματα*	*Διαθεσιμότητα*	*Χρόνος απόκρισης*	*Κλάση web serv.*
1	**Distance_Under_5km**	**0**	**80%**	**0**	**European**
2	Distance_Under_5km	1	75%	2.3	American
3	**Distance_Under_5km**	**1**	**63%**	**2.7**	**European**
4	Distance_Over_5km	2	97%	3.9	American
5	Distance_Over_5km	3	91%	4.5	European

Με βάση την παραπάνω επιλογή, οι επιλεχθέντες peers επερωτούνται και απαντούν αποστέλλοντας τις σχετικές πλειάδες. Έτσι το αποτέλεσμα που συλλέγεται είναι ως εξής:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
REB1543	71	GERMANY
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.

Παράδειγμα 6: Με βάση το παράδειγμα 0 θα δώσουμε μία ερώτηση η οποία αποσκοπεί να αναδείξει το γεγονός ότι στην επεκτεταμένη SQL που εισάγουμε έχουμε τη δυνατότητα να καθορίσουμε τον χρονισμό της ερώτησης. Ο καθορισμός αυτός μπορεί να γίνει διακρίνοντας αν η ερώτηση επιθυμούμε να γίνεται με ad-hoc τρόπο ή συνεχώς. Στην πρώτη περίπτωση πρέπει επίσης να καθορίσουμε την συνθήκη τερματισμού της ερώτησης, ποσοστό επιτυχούς απάντησης από ένα ποσοστό peers, πλήθος πλειάδων που συλλέγονται, λήψη ενός timeout. Στο παράδειγμα αυτό θα ασχοληθούμε με τις ad-hoc ερωτήσεις. Δίνονται τρεις ερωτήσεις για να αναδείξουμε τον τρόπο χρήσης και των τριών συνθηκών τερματισμού της ερώτησης που παρουσιάσαμε παραπάνω.

Ερώτηση α: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής όλων των αυτοκινήτων αυτήν τη στιγμή και επιθυμούμε να τερματιστεί η ερώτηση όταν έχουν απαντήσει πάνω από 70% των peers που ρωτάμε.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
TIMING AD-HOC PEERS_PERCENTAGE > 70 %
```

Απάντηση: Από τον κατάλογο του peer 1 αυτοί που είναι σημειωμένοι με μεγάλα έντονα γράμματα έχουν προλάβει να απαντήσουν (4/5 > 70%). Στο σημείο αυτό η ερώτηση τερματίζεται αφού ικανοποιείται η συνθήκη τερματισμού. Με δεδομένο αυτό μόνο σε αυτούς τους peers τίθεται η ερώτηση.

Κατάλογος του peer 1

Peer ID	*Κοινότητα που ανήκει*	*Απόσταση σε βήματα*	*Διαθεσιμότητα*	*Χρόνος απόκρισης*	*Κλάση web serv.*
1	**Distance_Under_5km**	**0**	**80%**	**0**	**European**
2	**Distance_Under_5km**	**1**	**75%**	**2.3**	**American**
3	**Distance_Under_5km**	**1**	**63%**	**2.7**	**European**
4	Distance_Over_5km	2	97%	3.9	American
5	**Distance_Over_5km**	**3**	**91%**	**4.5**	**European**

Με βάση την παραπάνω επιλογή, οι επιλεχθέντες peers επερωτούνται και απαντούν αποστέλλοντας τις σχετικές πλειάδες. Έτσι το αποτέλεσμα που συλλέγεται είναι ως εξής:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
GRT8724	75	FRANCE
REB1543	71	GERMANY
GAV2991	58	JAPAN
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.

Ερώτηση β: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής όλων των αυτοκινήτων αυτήν τη στιγμή και επιθυμούμε να τερματιστεί η ερώτηση όταν έχουν συλλεχθεί στην σχέση CARS πάνω από 5 πλειάδες.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
TIMING AD-HOC AMOUNT_TUPLES > 5
```

Απάντηση: Από τον κατάλογο του peer 1 αυτοί που είναι σημειωμένοι με μεγάλα έντονα γράμματα έχουν προλάβει να απαντήσουν προσδίδοντας ο καθένας τις δικές του πλειάδες (3+1+4>5). Στο σημείο αυτό η ερώτηση τερματίζεται αφού ικανοποιείται η συνθήκη τερματισμού. Με δεδομένο αυτό μόνο σε αυτούς τους peers τίθεται η ερώτηση.

Κατάλογος του peer 1

Peer ID	*Κοινότητα που ανήκει*	*Απόσταση σε βήματα*	*Διαθεσιμότητα*	*Χρόνος απόκρισης*	*Κλάση web serv.*
1	**Distance_Under_5km**	**0**	**80%**	**0**	**European**
2	**Distance_Under_5km**	**1**	**75%**	**2.3**	**American**
3	Distance_Under_5km	1	63%	2.7	European
4	**Distance_Over_5km**	**2**	**97%**	**3.9**	**American**
5	Distance_Over_5km	3	91%	4.5	European

Με βάση την παραπάνω επιλογή, οι επιλεχθέντες peers επερωτούνται και απαντούν αποστέλλοντας τις σχετικές πλειάδες. Έτσι το αποτέλεσμα που συλλέγεται είναι ως εξής:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
GRT8724	75	FRANCE
GAV2991	58	JAPAN
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.

Ερώτηση γ: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής όλων των αυτοκινήτων αυτήν τη στιγμή και επιθυμούμε να τερματιστεί η ερώτηση όταν έχουν ξεπεραστεί ένα timeout των 7 δευτερολέπτων.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
TIMING AD-HOC TIMEOUT > 7
```

Απάντηση: Από τον κατάλογο του peer 1 αυτοί που είναι σημειωμένοι με μεγάλα έντονα γράμματα έχουν προλάβει να απαντήσουν πριν λήξει το χρονικό όριο. Στο σημείο αυτό η ερώτηση τερματίζεται αφού ικανοποιείται η συνθήκη τερματισμού. Με δεδομένο αυτό μόνο σε αυτούς τους peers τίθεται η ερώτηση.

Κατάλογος του peer 1

Peer ID	*Κοινότητα που ανήκει*	*Απόσταση σε βήματα*	*Διαθεσιμότητα*	*Χρόνος απόκρισης*	*Κλάση web serv.*
1	**Distance_Under_5km**	**0**	**80%**	**0**	**European**
2	**Distance_Under_5km**	**1**	**75%**	**2.3**	**American**
3	**Distance_Under_5km**	**1**	**63%**	**2.7**	**European**
4	Distance_Over_5km	2	97%	3.9	American
5	Distance_Over_5km	3	91%	4.5	European

Με βάση την παραπάνω επιλογή, οι επιλεχθέντες peers επερωτούνται και απαντούν αποστέλλοντας τις σχετικές πλειάδες. Έτσι το αποτέλεσμα που συλλέγεται είναι ως εξής:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
GRT8724	75	FRANCE
REB1543	71	GERMANY
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.

Παράδειγμα 7: Στο παράδειγμα θα ασχοληθούμε με την έννοια του χρονισμού όπως και στο προηγούμενο παράδειγμα. Η διαφορά είναι ότι στο παράδειγμα αυτό θα ασχοληθούμε με τις συνεχείς ερωτήσεις σε αντίθεση με το παράδειγμα 5 το οποίο αναφέρεται σε ad-hoc ερωτήσεις και σε συνθήκες τερματισμού τους. Οι συνεχείς ερωτήσεις διακρίνονται με τη σειρά τους σε pull-based που πραγματοποιούνται ανά κάποια περίοδο που μπορούμε να καθορίσουμε και σε push-based που ο κάθε peer, όποτε χρειάζεται, ενημερώνει για τυχούσες αλλαγές που παρατηρούνται στις πλειάδες που προσδίδει στον peer που πραγματοποιεί την ερώτηση. Δίνονται δύο ερωτήσεις παραδείγματα ερώτησης, ένα για κάθε έναν τρόπο συνεχούς ερώτησης.

Ερώτηση α: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής όλων των αυτοκινήτων συνέχεια με περίοδο ανανέωσης της ερώτησης τα 7 δευτερόλεπτα.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
TIMING CONTINUOUS PULL_BASED_PERIOD = 7
```

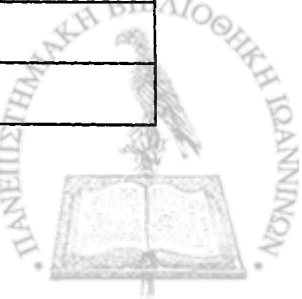
Αυτοκίνητο αναφοράς: το αυτοκίνητο με ID=1 και οι πλειάδες της βάσης

t=0 : CARS(1,TRK4147,NISSAN,70)

Απάντηση: Από τον κατάλογο του peer1 επιλέγονται όλοι οι peers που περιέχονται σε αυτόν. Η ερώτηση πραγματοποιείται με περίοδο 7 δευτερόλεπτα κι έχουμε στιγμιότυπα της βάσης δεδομένων όλων των peers για τις χρονικές στιγμές $t=t_0$, $t=t_0+7$, $t=t_0+14$, $t=t_0+21$. Σαν μία αντιπροσωπευτική απάντηση της συνεχούς ερώτησης θα δώσουμε το αποτέλεσμα της ερώτησης στις χρονικές αυτές στιγμές και αντιλαμβανόμαστε ότι στη συνέχεια η διαδικασία ερώτησης-απάντησης πραγματοποιείται με τον ίδιο τρόπο.

Τη χρονική στιγμή $t=t_0$:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
GRT8724	75	FRANCE
REB1543	71	GERMANY
BAJ8687	63	JAPAN
GAV2991	58	JAPAN
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.
ASD7295	86	GERMANY
QAF1211	37	U.S.A.
ABC1111	90	FRANCE



Τη χρονική στιγμή $t=t_0+7$:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	55	FRANCE
GRT8724	72	FRANCE
REB1543	45	GERMANY
BAJ8687	69	JAPAN
GAV2991	92	JAPAN
TAS4356	58	U.S.A.
YNM3456	35	U.S.A.
ASD7295	70	GERMANY
QAF1211	47	U.S.A.
ABC1111	90	FRANCE

Τη χρονική στιγμή $t=t_0+14$:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	59	FRANCE
GRT8724	79	FRANCE
REB1543	54	GERMANY
BAJ8687	77	JAPAN
GAV2991	95	JAPAN
TAS4356	67	U.S.A.
YNM3456	50	U.S.A.
ASD7295	72	GERMANY
QAF1211	56	U.S.A.
ABC1111	82	FRANCE

Τη χρονική στιγμή $t=t_0+21$:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	62	FRANCE
GRT8724	78	FRANCE
REB1543	52	GERMANY
BAJ8687	73	JAPAN
GAV2991	104	JAPAN
TAS4356	75	U.S.A.
YNM3456	51	U.S.A.
ASD7295	77	GERMANY
QAF1211	64	U.S.A.
ABC1111	98	FRANCE

Στη συνέχεια ακριβώς με τον ίδιο τρόπο ανά 7 δευτερόλεπτα πραγματοποιείται η ερώτηση, γεμίζει με τις νέες πλειάδες η σχέση CARS του peer 1 και εκτελείται η ερώτηση δίνοντας κάθε φορά το αντίστοιχο αποτέλεσμα.

Ερώτηση β: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής όλων των αυτοκινήτων συνέχεια με ενημέρωση για τυχούσες αλλαγές στις πλειάδες από τους peers στους οποίους αναφέρονται.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
TIMING CONTINUOUS PUSH_BASED
```

Απάντηση: Η ερώτηση τίθεται σε όλους τους peers οι οποίοι ανήκουν στον κατάλογο του peer 1. Για οποιαδήποτε αλλαγή που συμβαίνει στην πλειάδα τους ενημερώνουν τον peer 1 και το αποτέλεσμα μεταβάλλεται. Έχουμε τα στιγμιότυπα των τριών σχέσεων CARS των peers στους οποίους συμβαίνουν οι τρεις πρώτες αλλαγές. Θα δείξουμε το αποτέλεσμα της ερώτησης αρχικά ($t=t_0$) και στη συνέχεια κάθε φορά που

πραγματοποιείται μία από αυτές τις αλλαγές. Με μεγάλα έντονα γράμματα σημειώνεται η πλειάδα του αποτελέσματος που έχει μεταβληθεί.

Τη χρονική στιγμή $t=t_0$:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
GRT8724	75	FRANCE
REB1543	71	GERMANY
BAJ8687	63	JAPAN
GAV2991	58	JAPAN
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.
ASD7295	86	GERMANY
QAF1211	37	U.S.A.
ABC1111	90	FRANCE

Μετά την αλλαγή της χρονικής στιγμής $t=t_1$:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
GRT8724	75	FRANCE
REB1543	72	GERMANY
BAJ8687	63	JAPAN
GAV2991	58	JAPAN
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.
ASD7295	86	GERMANY
QAF1211	37	U.S.A.
ABC1111	90	FRANCE

Μετά την αλλαγή της χρονικής στιγμής $t=t_2$:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	46	FRANCE
GRT8724	75	FRANCE
REB1543	72	GERMANY
BAJ8687	63	JAPAN
GAV2991	58	JAPAN
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.
ASD7295	86	GERMANY
QAF1211	37	U.S.A.
ABC1111	90	FRANCE

Μετά την αλλαγή της χρονικής στιγμής $t=t_3$:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
GRT8724	74	FRANCE
REB1543	72	GERMANY
BAJ8687	63	JAPAN
GAV2991	58	JAPAN
TAS4356	56	U.S.A.
YNM3456	43	U.S.A.
ASD7295	86	GERMANY
QAF1211	37	U.S.A.
ABC1111	90	FRANCE

Στη συνέχεια με τον ίδιο τρόπο για κάθε αλλαγή που γίνεται έχουμε και την αντίστοιχη αλλαγή στο αποτέλεσμα της συνεχούς ερώτησης. Στο παράδειγμα αυτό παρουσιάστηκε η διαδικασία που πραγματοποιείται αντιπροσωπευτικά μόνο για τις τρεις πρώτες αλλαγές που συμβαίνουν από τη χρονική στιγμή t_0 που αρχικά τίθεται η ερώτηση.

Παράδειγμα 8: Στο παράδειγμα θα ασχοληθούμε με την έννοια της αξιοπιστίας της απάντησης που μπορούμε να πάρουμε ρωτώντας τοπικά τη βάση μας. Δηλαδή θέτουμε ένα χρονικό όριο μεταξύ της στιγμής που τίθεται η ερώτηση και της στιγμής ολοκλήρωσης της διαδικασίας συλλογής πλειάδων από τη σχέση CARS. Αν αυτό το χρονικό διάστημα ξεπερνά το όριο που θέτουμε, τότε η ερώτηση εκτελείται συλλέγοντας πλειάδες από τους peers, ενώ σε αντίθετη περίπτωση πραγματοποιείται τοπικά θεωρώντας τα δεδομένα που είναι αποθηκευμένα τοπικά στη βάση μας αρκετά πρόσφατα και αξιόπιστα.

Ερώτηση: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής όλων των αυτοκινήτων θέτοντας ως μέγιστο χρόνο αξιοπιστίας των περιεχομένων της τοπικής βάσης δεδομένων ίσο με 5 δευτερόλεπτα.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
AGE < 5
```

Τη χρονική στιγμή που γίνεται η ερώτηση $t=t_0$, ο μετρητής του πίνακα CARS του peer 1 είναι ίσος με 7 το οποίο είναι μεγαλύτερο από το 5 που δίνεται στην πρόταση AGE. Οπότε τα περιεχόμενα του πίνακα εκείνη την στιγμή θεωρούνται μη αξιόπιστα χρονικά και ο peer 1 ζητάει από όλους τους peers που υπάρχουν στον κατάλογό του να του δώσουν τις πλειάδες τους. Αφού συλλέξει τις νέες πλειάδες από όλους τους peers εκτελεί την ερώτηση και δίνει την απάντηση:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
GRT8724	75	FRANCE
REB1543	71	GERMANY
BAJ8687	63	JAPAN

GAV2991	58	JAPAN
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.
ASD7295	86	GERMANY
QAF1211	37	U.S.A.
ABC1111	90	FRANCE

Παράδειγμα 9: Στο παράδειγμα αυτό γίνεται ενοποίηση όλων των παραπάνω παραδειγμάτων σε ένα. Δηλαδή παρουσιάζεται μία ερώτηση η οποία απαιτεί και χρησιμοποιεί όλες τις δυνατότητες της επεκτεταμένης SQL που εισάγουμε. Αυτό πραγματοποιείται έτσι ώστε να δοθεί μία πιο ολοκληρωμένη και συγκεντρωτική εικόνα των ερωτήσεων που μπορούν να τεθούν από κάποιον peer του συστήματος.

Ερώτηση: Επιθυμούμε τον αριθμό κυκλοφορίας, την ταχύτητα και τη χώρα κατασκευής των αυτοκινήτων, τα οποία έχουν διαθεσιμότητα μεγαλύτερη από 60%, χρόνο απόκρισης μικρότερο από 3 δευτερόλεπτα και ανήκουν στην κοινότητα του peer 1 με όνομα Distance_Under_5km. Επίσης ο δεδομένος peer επιθυμεί απάντηση μόνο από αυτοκίνητα των οποίων οι web services ανήκουν στην κλάση European και μάλιστα επιθυμεί η ερώτηση να γίνεται συνεχώς με μία περίοδο 7 δευτερολέπτων. Τέλος ο peer που θέτει την ερώτηση θεωρεί αξιόπιστη απάντηση αυτήν που μπορεί να πάρει τοπικά, αν η σχέση CARS έχει συλλεχθεί το πολύ μέχρι 5 δευτερόλεπτα πριν.

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
AGE < 5 AND
HORIZON COMMUNITY Distance_Under_5km AND
TIMING CONTINUOUS PULL_BASED_PERIOD = 7 AND
AVAILABILITY > 60% AND
RESPONSE_TIME < 3.0 AND
CLASS = "european"
```

Απάντηση: Τη στιγμή που τίθεται η ερώτηση $t=t_0$ ο μετρητής χρόνου της σχέσης CARS του peer 1 είναι ίσος με 7 > 5, άρα πρέπει να συλλεχθούν νέες πλειάδες από τους peers που μας ενδιαφέρουν. Από τον κατάλογο του peer 1 αυτοί που είναι σημειωμένοι με μεγάλα έντονα γράμματα είναι αυτοί που ικανοποιούν όλες τις συνθήκες της ερώτησης. Με δεδομένο αυτό μόνο σε αυτούς τους peers τίθεται η ερώτηση.

Κατάλογος του peer 1

Peer ID	*Κοινότητα που ανήκει*	*Απόσταση σε βήματα*	*Διαθεσιμότητα*	*Χρόνος απόκρισης*	*Κλάση web serv.*
1	**Distance_Under_5km**	**0**	**80%**	**0**	**European**
2	Distance_Under_5km	1	75%	2.3	American
3	**Distance_Under_5km**	**1**	**63%**	**2.7**	**European**
4	Distance_Over_5km	2	97%	3.9	American
5	Distance_Over_5km	3	91%	4.5	European

Η ερώτηση τίθεται συνεχώς με μία περίοδο 7 δευτερολέπτων. Έχουμε στιγμιότυπα της βάσης δεδομένων όλων των peers για τις χρονικές στιγμές $t=t_0$, $t=t_0+7$, $t=t_0+14$, $t=t_0+21$. Σαν μία αντιπροσωπευτική απάντηση της συνεχούς ερώτησης θα δώσουμε το αποτέλεσμα της ερώτησης στις χρονικές αυτές στιγμές και αντιλαμβανόμαστε ότι στη συνέχεια η διαδικασία ερώτησης-απάντησης πραγματοποιείται με τον ίδιο τρόπο.

Τη χρονική στιγμή $t=t_0$:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	45	FRANCE
REB1543	71	GERMANY
TAS4356	55	U.S.A.
YNM3456	43	U.S.A.

Τη χρονική στιγμή $t=t_0+7$:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	55	FRANCE
REB1543	45	GERMANY
TAS4356	58	U.S.A.
YNM3456	35	U.S.A.

Τη χρονική στιγμή $t=t_0+14$:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	59	FRANCE
REB1543	54	GERMANY
TAS4356	67	U.S.A.
YNM3456	50	U.S.A.

Τη χρονική στιγμή $t=t_0+21$:

CARS.PLATE	*CARS.VEL*	*BRANDS.COUNTRY*
FRA4526	62	FRANCE
REB1543	52	GERMANY
TAS4356	75	U.S.A.
YNM3456	51	U.S.A.

Στη συνέχεια ακριβώς με τον ίδιο τρόπο ανά 7 δευτερόλεπτα πραγματοποιείται η ερώτηση, γεμίζει με τις νέες πλειάδες η σχέση CARS του peer 1 και εκτελείται η ερώτηση δίνοντας κάθε φορά το αντίστοιχο αποτέλεσμα.

3.4.3 Συντακτικό των Ερωτήσεων σε BNF Μορφή

Στην ενότητα αυτή δίνουμε το συντακτικό της επεκτεταμένης SQL που εισάγουμε, σε μορφή BNF. Σημειώνουμε ότι με έντονα γράμματα παριστάνονται οι δεσμευμένες λέξεις της γλώσσας, ενώ με italic οι λέξεις NUMBER και STRING που σημαίνουν ένας αριθμός και μία συμβολοσειρά αντίστοιχα. Επίσης με το ε συμβολίζεται το κενό στην BNF.

<query_statement>::= <select_statement> <from_statement>
<where_statement> <group_by_statement>
<order_by_statement> <with_statement>

<with_statement> ::= {<with> <and_or_with_statement>} | ε

<with> ::= <age_statement> | <timing_statement> |
<horizon_statement> | <availability_statement> |
<response_time_statement> | <class_statement>

<and_or_with_statement> ::= { **AND** | **OR** <with> <and_or_with_statement>} | ε

<select_statement>::= **SELECT** <column> <comma_column>

<comma_column> ::= { ',' <table_name> <dot><column_name><comma_column>}
| ε

<table_name> ::= *STRING*

<dot> ::= '.'

<column_name> ::= *STRING*

<column> ::= <table_name> <dot> <column_name>

<from_statement> ::= **FROM** <table_name> <comma_table_name>

<comma_table_name> ::= {',' <table_name> <comma_table_name>} | ε

<where_statement> ::= { **WHERE** <condition>} | ε

<condition> ::= <term> <operator> <term> <and_or_condition>

<operator> ::= '=' | '<' | '>' | "<=" | ">="

<term> ::= <column> | <value>

<value> ::= NUMBER | STRING

<and_or_condition> ::= { **AND | OR** <condition>} | ' '

<group_by_statement> ::= { **GROUP BY** <column> <comma_column>
<having_statement>} | ε

<having_statement> ::= { **HAVING** <condition>} | ε

<order_by_statement> ::= { **ORDER BY** <column> **DESC | ASC** | ε
<desc_asc_comma_column> } | ε

<desc_asc_comma_column> ::= { ',' <column> **DESC | ASC** | ε
<desc_asc_comma_column> } | ε

<age_statement> ::= **AGE** <less_than> | <less_equal_than> <num_value>

<num_value> ::= *NUMBER*

<timing_statement> ::= { **AD-HOC** <ad-hoc_statement>} |
{ **CONTINUOUS** <continuous_statement>}
<ad-hoc_statement> ::= { **PEERS_PERCENTAGE** <num_value> '%'} |
{ **AMOUNT_TUPLES** <num_value>} |
{ **TIMEOUT** <num_value>}

<continuous_statement> ::= { **PULL_BASED_PERIOD** <equal> <num_value>} |
PUSH_BASED

<equal> ::= '='

<horizon_statement> ::= **HORIZON** <peers_statement>

<peers_statement> ::= LOCAL | COMMUNITY <community_name>
| {**HOPS** <equal> | <less_than> | <greater_than> | <less_equal_than> |
<greater_equal_than> <num_value>} | **PEERS**=[<num_value> <comma_num>]

<community_name> ::= *STRING*

<comma_num> ::= { ',' <num_value> <comma_num>} | ε

<availability_statement> :: **AVAILABILITY** <greater_than> | <greater_equal_than>
<num_value> '%'

<greater_than> ::= '>'

<greater_equal_than> ::= ">="

<response_time_statement> ::= **RESPONSE_TIME** <less_than> |
<less_equal_than> <num_value>

<less_than> ::= '<'

<less_equal_than> ::= "<="

<class_statement> ::= **CLASS** <equal> <string_value>

<string_value> ::= *STRING*

3.5 Επέκταση των CREATE TABLE, INSERT, DELETE, UPDATE

Η εντολή CREATE TABLE παραμένει ως έχει στην standard SQL με τη μόνη διαφορά ότι πριν το όνομα του πίνακα αν αυτός είναι υβριδικός ή νοητός

προσθέτουμε τη λέξη HYBRID ή VIRTUAL αντίστοιχα. Προφανώς στην περίπτωση που ο πίνακας είναι τοπικά αποθηκευμένος δεν προσθέτουμε καμία λέξη κι εννοείται ότι δεν είναι ούτε υβριδικός αλλά ούτε και νοητός.

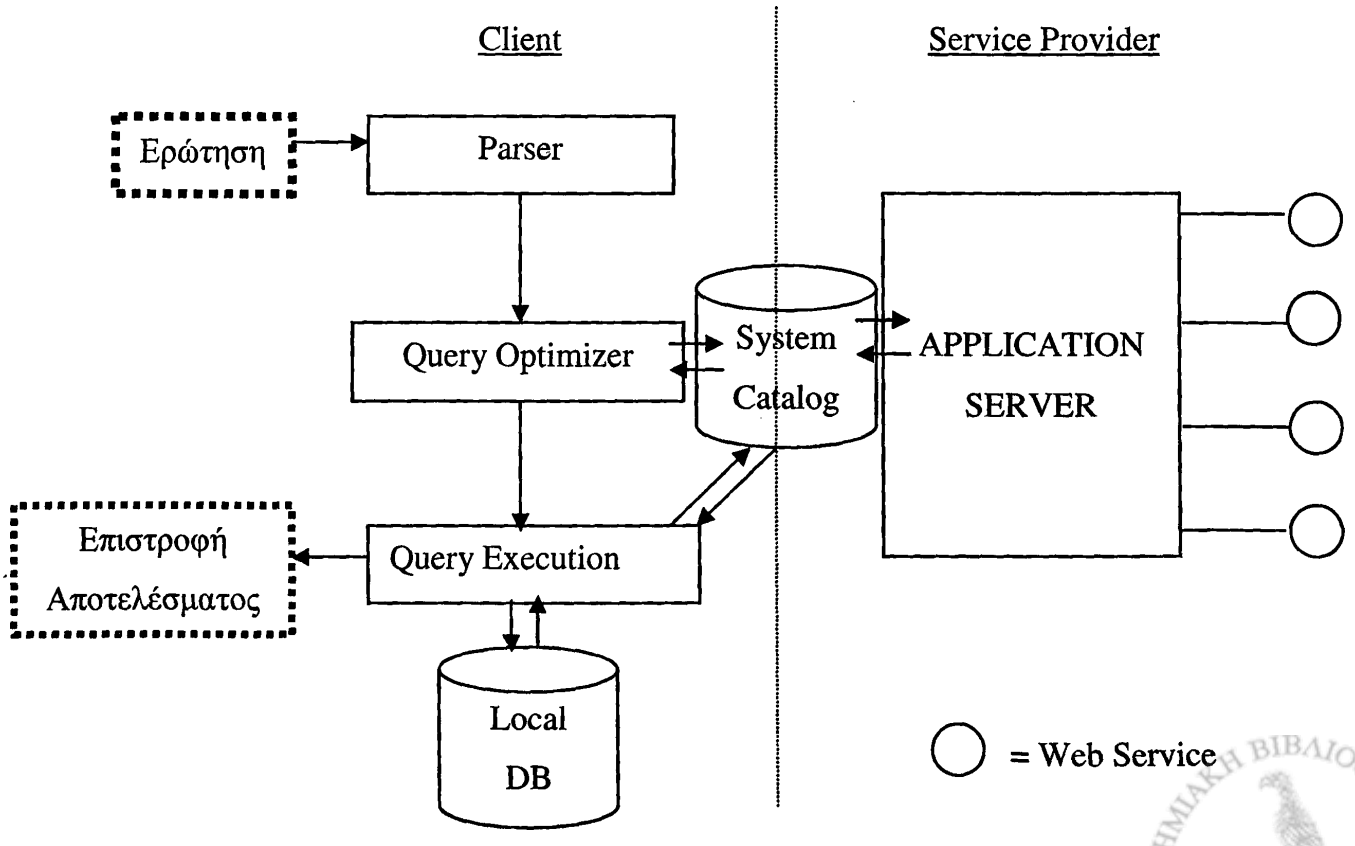
Οι εντολές INSERT, UPDATE, DELETE παραμένουν όπως ακριβώς είναι στην παραδοσιακή SQL χωρίς καμία αλλαγή.

ΚΕΦΑΛΑΙΟ 4. ΑΝΤΙΣΤΟΙΧΙΣΗ ΤΗΣ SQL^P ΣΕ ΠΛΑΝΑ ΣΧΕΣΙΑΚΗΣ ΑΛΓΕΒΡΑΣ

Στο κεφάλαιο αυτό, αρχικά παρουσιάζουμε την διαδικασία της αποτίμησης μίας ερώτησης σε SQL^P. Καταρχήν, στην ενότητα 4.1 παρουσιάζουμε την αρχιτεκτονική του συστήματος. Στην επόμενη ενότητα, την 4.2, αναλύονται τα συστήματα ετερογενών βάσεων δεδομένων. Δίνεται η αρχιτεκτονική (mediator-wrapper) τέτοιων συστημάτων και παρουσιάζεται η διαδικασία παραγωγής πλάνων ερώτησης σε τέτοια συστήματα. Στην ενότητα 4.3, πραγματοποιείται ο τυπικός ορισμός όλων των τελεστών που μπορεί να λάβουν μέρος σε κάποιο πλάνο ερώτησης. Στην ενότητα 4.4 αναλύεται ο τρόπος υλοποίησης των επεκτάσεων που εισάγαμε στην παραδοσιακή SQL. Στην ενότητα 4.5, παρουσιάζεται ο τρόπος κατασκευής πλάνων εκτέλεσης ερώτησης της γλώσσας SQL^P, δίνοντας τον αλγόριθμο εκτέλεσης μίας ερώτησης και ορίζοντας νέους τελεστές που υποστηρίζουν τον αλγόριθμο αυτό. Τέλος παρατίθεται ένας εναλλακτικός τρόπος εκτέλεσης ερώτησης, ο οποίος έχει νόημα όταν κάποιος peer ζητήσει τη συνδρομή κάποιου άλλου peer στη διαδικασία εκτέλεσης μίας ερώτησης.

4.1 Αρχιτεκτονική Συστήματος

Το σύστημα αποτελείται από ένα σύνολο peers οι οποίοι σχηματίζουν ένα γράφο σύμφωνα με όσα περιγράψαμε στο κεφάλαιο 3. Όλοι οι peers του γράφου αυτού υπακούουν σε μία κοινή αρχιτεκτονική. Διακρίνουμε δύο ρόλους σε κάθε peer, τον ρόλο του επερωτώντα (client) και τον ρόλο του παροχέα υπηρεσιών (service provider). Ο κάθε peer συμπεριφέρεται ως client όταν αυτός θέτει κάποια ερώτηση, ενώ συμπεριφέρεται ως παροχέας υπηρεσιών όταν του ζητηθεί να εκτελέσει ένα συγκεκριμένο workflow έτσι ώστε να τροφοδοτήσει με δεδομένα κάποιον άλλο peer. Στο σχήμα 4.1 φαίνεται η αρχιτεκτονική ενός peer του συστήματος, χωρίζοντας με μία κάθετη γραμμή τους δύο ρόλους που μπορεί να παίξει αυτός. Κατά την client εκδοχή, η οποία υπενθυμίζουμε υπάρχει όταν ο δεδομένος peer θέσει κάποια ερώτηση, ο peer αποτελείται από έναν parser, έναν βελτιστοποιητή ερώτησης, μία μηχανή εκτέλεσης ερωτήσεων, μία τοπική βάση δεδομένων και έναν κατάλογο συστήματος. Κατά την εκδοχή του παροχέα υπηρεσιών, η οποία υπενθυμίζουμε ενεργοποιείται όταν καλούνται κάποιες web services ενός peer, υπάρχει ένας application server ο οποίος έχει ένα σύνολο από web service λειτουργίες (operations), τις οποίες δημοσιοποιεί ο δεδομένος peer.



Σχήμα 4.1: Αρχιτεκτονική του κάθε Peer του Συστήματος

Στο σημείο αυτό ορίζουμε την έννοια του πλάνου εκτέλεσης ερώτησης, έννοια την οποία χρησιμοποιούμε συχνά παρακάτω. *Πλάνο εκτέλεσης ερώτησης* είναι μία δενδρική δομή που αντιστοιχεί σε μία έκφραση της σχεσιακής άλγεβρας. Οι σχέσεις εισόδου αναπαριστώνται ως κόμβοι-φύλλα και οι πράξεις της σχεσιακής άλγεβρας ως εσωτερικοί κόμβοι. Εξαίρεση αποτελούν οι σχέσεις που είναι υβριδικές ή νοητές οι οποίες δεν είναι κόμβοι-φύλλα αλλά αποτελούν τη ρίζα ενός υποδέντρου. Το υπόδεντρο αυτό εκφράζει τη διαδικασία τροφοδότησης με πλειάδες της σχέσης που έχει ως ρίζα. Ως φύλλα αυτού του υπόδεντρου, αναπαριστώνται οι peers που ανήκουν στο σύνολο κόμβων ενδιαφέροντος της ερώτησης. Οι εσωτερικοί κόμβοι του υπόδεντρου αντιστοιχούν σε κάποιες πράξεις που πραγματοποιούν τη διαδικασία τροφοδότησης με πλειάδες της σχέσης-ρίζα του υπόδεντρου. Μία εκτέλεση του δέντρου-πλάνου ερώτησης συνίσταται στην εκτέλεση των πράξεων κάθε εσωτερικού κόμβου όταν οι σχέσεις εισόδου του είναι διαθέσιμες και στη συνέχεια αντικατάσταση του εσωτερικού κόμβου με τη σχέση η οποία προκύπτει από την εκτέλεση των πράξεων. Όσες σχέσεις είναι υβριδικές ή νοητές, για να είναι διαθέσιμες πρέπει πρώτα να εκτελεστούν οι πράξεις που αναφέρονται στο υπόδεντρό τους. Η εκτέλεση του πλάνου τερματίζεται όταν εκτελεσθεί ο κόμβος της ρίζας του συνολικού δέντρου-πλάνου, οπότε και παράγεται η σχέση με το τελικό αποτέλεσμα.

Στη συνέχεια, αναλύονται τα διάφορα τμήματα από τα οποία αποτελείται ο peer κατά την client εκδοχή του:

(α) Όταν κάποιος χρήστης υποβάλλει μία ερώτηση, το σύστημα διαχείρισης βάσεων δεδομένων (DBMS) αρχικά ενεργοποιεί τον parser, πραγματοποιεί τη διαδικασία του parsing στην ερώτηση.

(β) Ο βελτιστοποιητής ερωτήσεων, παράγει ένα σύνολο εναλλακτικών πλάνων της ερώτησης και με τη βοήθεια ενός μοντέλου κόστους αποφαίνεται ποιο είναι το πιο φθηνό πλάνο.

(γ) Η μηχανή εκτέλεσης ερωτήσεων, εκτελεί την ερώτηση με βάση το καλύτερο πλάνο που έχει προκύψει από την εφαρμογή της διαδικασίας βελτιστοποίησης της ερώτησης.

(δ) Η τοπική βάση δεδομένων, περιέχει τους πίνακες που είναι τοπικά αποθηκευμένοι στον δεδομένο peer, των οποίων τα δεδομένα δεν επηρεάζονται από τους υπόλοιπους peers του συστήματος και τους νοητούς ή υβριδικούς πίνακες του δεδομένου peer, οι

οποίοι τροφοδοτούνται με δεδομένα από τους άλλους peers του συστήματος, όπως έχει αναλυθεί σε προηγούμενα κεφάλαια.
(ε) Ο κατάλογος συστήματος, περιέχει πληροφορίες οι οποίες είναι απαραίτητες για την εκτέλεση της ερώτησης, όπως τα χαρακτηριστικά των γνωστών peers, την κοινότητα στην οποία ανήκουν και την απόσταση μεταξύ τους σε βήματα ή hops.

Στη συνέχεια πρωτού εισέλθουμε στη διαδικασία εκτέλεσης μίας ερώτησης, θα περιγράψουμε σε ευρύτερη ανάλυση το πώς εκτελούνται οι ερωτήσεις σε συστήματα ετερογενών βάσεων δεδομένων, που αποτελούν την πλησιέστερη τεχνολογία στο πρόβλημά μας. Έπειτα θα συμπληρώσουμε τις τεχνικές αυτές ώστε να καλύπτουν πλήρως τις απαιτήσεις της δικής μας εφαρμογής.

4.2 Συστήματα Ετερογενών Βάσεων Δεδομένων

Τα ετερογενή συστήματα βάσεων δεδομένων [Koss00], είναι συστήματα τα οποία αποτελούνται από πολλές ετερογενείς βάσεις δεδομένων. Δηλαδή εκτός από το πλήθος των βάσεων δεδομένων, τα συστήματα αυτά χαρακτηρίζονται και από την ετερογένεια που υπάρχει μεταξύ των βάσεων δεδομένων αυτών. Λόγω της ετερογένειας αυτής ονομάζονται συστήματα ετερογενών βάσεων δεδομένων (heterogeneous databases systems) και η κάθε μία ξεχωριστά βάση δεδομένων που περιλαμβάνεται σε αυτά, ονομάζεται συστατική βάση δεδομένων (component database).

Με τον όρο ετερογενείς βάσεις δεδομένων, συμπεριλαμβάνουμε δύο έννοιες:
(α) την ετερογένεια σχήματος, κατά την οποία οι πίνακες της βάσεις δεδομένων έχουν διαφορετικά σχήματα, δηλαδή έχουν άλλα πεδία και
(β) τη σημασιολογική ετερογένεια, κατά την οποία είναι διαφορετική η σημασιολογία των πεδίων των πινάκων, δηλαδή διαφορές στην έκφραση της ίδιας πληροφορίας, όπως για παράδειγμα δολλάρια-ευρώ, μίλια-χιλιόμετρα κ.τ.λ.

4.2.1 Εισαγωγή

Η ενότητα αυτή παρουσιάζει πώς μπορεί να πραγματοποιηθεί η επεξεργασία ερωτήσεων σε συστήματα ετερογενών βάσεων δεδομένων. Τέτοια συστήματα έχουν ως σκοπό την ανάπτυξη εφαρμογών οι οποίες απαιτούν την προσπέλαση διαφορετικού είδους βάσεων δεδομένων. Ένα χαρακτηριστικό των συστημάτων αυτών, αποτελεί το γεγονός ότι κάθε μία από τις συστατικές βάσεις δεδομένων του συστήματος μπορεί να έχει διαφορετικές δυνατότητες για την αποθήκευση δεδομένων, για την πραγματοποίηση λειτουργιών βάσεων δεδομένων και για την επικοινωνία με άλλες συστατικές βάσεις δεδομένων του συστήματος. Το συγκεκριμένο χαρακτηριστικό δημιουργεί την εξής πρόκληση: εύρεση πλάνων ερωτήσεων τα οποία εκμεταλλεύονται τις ειδικές δυνατότητες της κάθε μίας συστατικής βάσης δεδομένων του συστήματος με τον καλύτερο δυνατό τρόπο και αποφυγή πλάνων ερωτήσεων τα οποία οδηγούν σε λανθασμένες λειτουργίες πάνω στις συστατικές βάσεις δεδομένων του συστήματος. Μία επιπλέον πρόκληση είναι η ασχολία με την σημασιολογική ετερογένεια.

4.2.2 Αρχιτεκτονική για Συστήματα Ετερογενών Βάσεων Δεδομένων

Η αρχιτεκτονική συστημάτων ετερογενών βάσεων δεδομένων ονομάζεται αρχιτεκτονική mediator-wrappers, παρουσιάζεται στο σχήμα 4.2 και αναλύεται αμέσως παρακάτω.

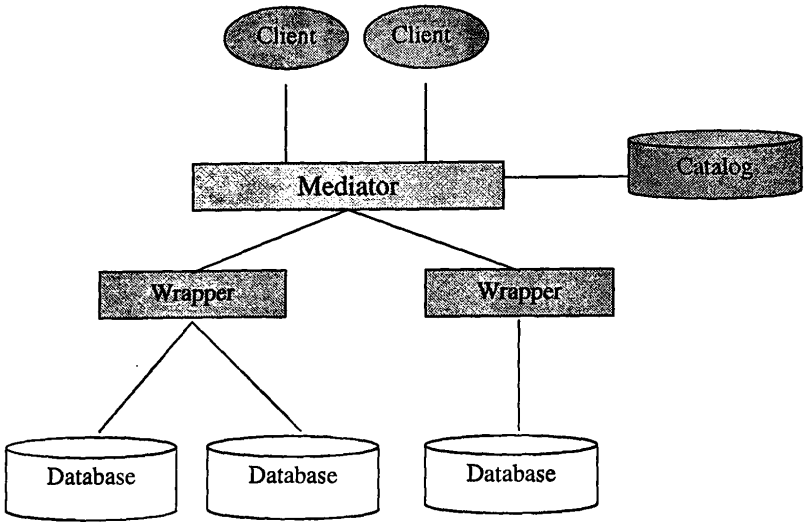
Οι clients συνδέονται με έναν mediator. Ο mediator πραγματοποιεί parsing, επανεγγραφή και βελτιστοποίηση σε μία ερώτηση και εκτελεί κάποιες από τις λειτουργίες (operations) μίας ερώτησης. Επίσης στις αρμοδιότητες του mediator ανήκει η συντήρηση ενός καταλόγου όπου αποθηκεύεται:
(α) το global schema όλου του ετερογενούς συστήματος βάσεων δεδομένων,
(β) το external schema κάθε μίας από τις συστατικές βάσεις του συστήματος και
(γ) στατιστικά για βελτιστοποίηση ερωτήσεων.

Το *global schema* είναι το σχήμα του όλου συστήματος το οποίο χρησιμοποιείται στις ερωτήσεις από προγράμματα της εφαρμογής και από χρήστες. *Το external schema* κάθε μίας συστατικής βάσης δεδομένων του συστήματος είναι το τμήμα του global

schema το οποίο είναι αποθηκευμένο από κάθε μία από τις προαναφερθείσες βάσεις δεδομένων. Ο mediator ακόμη είναι σχεδιασμένος να συνενώνει κάθε είδους συστατική βάση δεδομένων, έτσι ο mediator δεν περιέχει κώδικα που είναι εξειδικευμένος σε κάποια συστατική βάση δεδομένων και ως αποτέλεσμα αυτού ένας mediator δεν μπορεί να αλληλεπιδράσει άμεσα με τις συστατικές βάσεις δεδομένων.

Για να εκμεταλλευτούμε τα δεδομένα των συστατικών βάσεων δεδομένων ένας wrapper συνδέεται με κάθε μία συστατική βάση δεδομένων. Ο wrapper λειτουργεί ανάμεσα από τον mediator και τις συστατικές βάσεις δεδομένων. Η εργασία που επιτελεί ένας wrapper είναι:
(α) από τη μία να μεταφράζει κάθε απαίτηση (request) του mediator έτσι ώστε η απαίτηση αυτή να γίνεται κατανοητή από το API της συστατικής βάσης δεδομένων και
(β) από την άλλη να μεταφράζει τα αποτελέσματα που επιστρέφονται από την συστατική βάση δεδομένων έτσι ώστε αυτά να γίνονται κατανοητά από τον mediator και να είναι συμμορφωμένα με το external schema της συστατικής βάσης δεδομένων και με το global schema του ετερογενούς συστήματος βάσεων δεδομένων.

Οι wrappers συχνά εφαρμόζουν ειδικές τεχνικές όπως row blocking ή caching με σκοπό την βελτίωση της απόδοσης. Επίσης λαμβάνουν μέρος κατά την φάση της βελτιστοποίησης ερωτήσεων. Πάντως είναι γεγονός ότι παρόμοιοι wrappers μπορούν να εργαστούν για πολλές διαφορετικού είδους συστατικές βάσεις δεδομένων, πράγμα που καθιστά εύκολο να προσαρμόσουμε έναν ήδη υπάρχων wrapper με σκοπό να κατασκευάσουμε έναν wrapper για μία νέα συστατική βάση δεδομένων. Όπως φαίνεται και στο σχήμα 4.2 ένας wrapper είναι δυνατό να χειρίζεται πολλές διαφορετικές συστατικές βάσεις δεδομένων. Ένα πολύ σημαντικό χαρακτηριστικό της αρχιτεκτονικής του σχήματος 4.2 αποτελεί το γεγονός ότι η αρχιτεκτονική αυτή είναι επεκτάσιμη. Κάθε στιγμή οι wrappers και οι συστατικές βάσεις δεδομένων μπορούν να αναβαθμιστούν ή νέες συστατικές βάσεις δεδομένων μπορούν να ενοποιηθούν στο σύστημα χωρίς να αλλαχτεί ο mediator ή προσαρμόζοντας ήδη υπάρχοντες wrappers.



Σχήμα 4.2: Mediator-Wrapper Αρχιτεκτονική

Ο wrapper περιγράφει τα δεδομένα τα οποία αποθηκεύονται στις πηγές δεδομένων που έχει υπό τη διαχείρισή του μέσω ενός μοντέλου δεδομένων που παρέχει η εφαρμογή. Τα δεδομένα μέσα στις πηγές φαίνονται με τη μορφή αντικειμένων και το σύστημα μπορεί να αναφερθεί σε αυτά χρησιμοποιώντας ένα αναγνωριστικό αντικειμένου, το OID, το οποίο κατασκευάζεται με βάση την πηγή δεδομένων, τον τύπο του αντικειμένου και ένα μοναδικό κλειδί που καθορίζεται από τον wrapper. Αυτό το OID επιτρέπει στο σύστημα να εφαρμόζει μεθόδους πάνω σε αντικείμενα, επιτρέπει στο σύστημα να καθορίσει τον κατάλληλο wrapper και επιτρέπει στο wrapper να τοποθετήσει τα κατάλληλα δεδομένα και να εφαρμόσει τη μέθοδο. Οι wrappers παρέχουν μεθόδους με τις οποίες μπορούμε να πάρουμε τις τιμές από όλα τα γνωρίσματα ενός αντικειμένου. Επίσης ορίζουν συλλογές αντικειμένων οι οποίες αποτελούν τον στόχο ερωτήσεων.

Ο mediator αποτελείται από δύο βασικά συστατικά: τον βελτιστοποιητή ερωτήσεων και τη μηχανή εκτέλεσης ερωτήσεων. Το πρώτο δέχεται ως είσοδο μία ερώτηση και κατασκευάζει ένα πλάνο εκτέλεσης της ερώτησης αφού πραγματοποιήσει όλες τις φάσεις του parsing, σημασιολογικό έλεγχο, επανεγγραφή ερώτησης και βελτιστοποίηση ερώτησης. Η εργασία του βελτιστοποιητή είναι να κατασκευάσει και να επιλέξει το βέλτιστο πλάνο εκτέλεσης της ερώτησης με βάση κάποιο μοντέλο κόστους που παρέχει το ίδιο το σύστημα. Η κύρια πρόκληση που βρίσκεται στο

σημείο αυτό, είναι η κατασκευή ενός βελτιστοποιητή ερωτήσεων ο οποίος να έχει τη δυνατότητα γρήγορα να βρίσκει καλά πλάνα για την ερώτηση χωρίς να έχει γνώση σχετικά με τις δυνατότητες και ιδιότητες που χαρακτηρίζουν την κάθε πηγή δεδομένων.

Στη φάση κατά την οποία το ζητούμενο πλάνο έχει καθοριστεί από το βελτιστοποιητή, η διαδικασία προχωράει στην εκτέλεση αυτού του πλάνου. Η εκτέλεση του πλάνου συντονίζεται από το δεύτερο συστατικό του mediator που αναφέραμε παραπάνω, την μηχανή εκτέλεσης ερωτήσεων. Το συστατικό αυτό περνάει υποερωτήσεις προς τους wrappers και επίσης συνθέτει το τελικό αποτέλεσμα της ερώτησης συνδυάζοντας τις επιμέρους απαντήσεις που παίρνει από τον κάθε wrapper για την αντίστοιχη υποερώτηση που του ανατέθηκε. Η μηχανή εκτέλεσης ερωτήσεων είναι ένα σύστημα το οποίο έχει τη δυνατότητα να πραγματοποιήσει πολλές λειτουργίες όπως join, εφαρμογή συνθηκών, κλήση μεθόδων, ταξινόμηση, εφαρμογή συναρτήσεων, aggregate και άλλες. Το γεγονός αυτό επιτρέπει στο σύστημα να πραγματοποιεί λειτουργίες οι οποίες δεν μπορούν να πραγματοποιηθούν στις πηγές δεδομένων ή δεν τις παρέχουν οι αντίστοιχοι wrappers, πράγμα το οποίο το καθιστά ιδιαίτερα αποτελεσματικό.

Στο [HKWY97] παρουσιάζεται το Clio, το οποίο είναι ένα σύστημα το οποίο σχεδιάστηκε με σκοπό να ενοποιεί δεδομένα από ένα σύνολο πηγών δεδομένων με διαφορετικές δυνατότητες ερώτησης. Το σύστημα αυτό χρησιμοποιεί την τεχνολογία των wrappers έτσι ώστε να εμφωλεύει τις διαφορετικές δυνατότητες που χαρακτηρίζουν κάθε πηγή δεδομένων. Διαθέτει, έναν βελτιστοποιητή ο οποίος λειτουργεί σε ένα ετερογενές περιβάλλον και με τη βοήθεια κάποιων κανόνων παράγει πλάνα εκτέλεσης ερώτησης πάνω σε ένα τέτοιο περιβάλλον. Στην επόμενη ενότητα αναλύουμε τη διαδικασία παραγωγής πλάνων στο σύστημα αυτό, διότι κάτι αντίστοιχο επιθυμούμε και στη δική μας εφαρμογή.

4.2.3 Διαδικασία Παραγωγής Πλάνων στο Clio

Στην ενότητα αυτή αρχικά πραγματοποιείται μία γενική περιγραφή των συστατικών (STARs, POPs, πλάνα) και της διαδικασίας παραγωγής πλάνων. Στη συνέχεια

ακολουθεί μία επιμέρους ανάλυση για κάθε ένα από αυτά. Έπειτα για SPJ (Select-Project-Join) ερωτήσεις δείχνεται η διαδικασία παραγωγής πλάνων με την εφαρμογή των κατάλληλων STARs και αναλύεται πώς προκύπτει το κόστος ενός τέτοιου πλάνου. Επίσης δίνεται ένα παράδειγμα ερώτησης και η διαδικασία παραγωγής εναλλακτικών πλάνων που ακολουθείται.

Όπως προαναφέραμε ο mediator είναι υπεύθυνος για τη φάση της βελτιστοποίησης και τη φάση της εκτέλεσης μίας ερώτησης. Για να πραγματοποιηθεί η διαδικασία της βελτιστοποίησης μίας ερώτησης ο mediator χρησιμοποιεί ένα σύνολο από εναλλακτικούς κανόνες που ονομάζονται STARs (Strategy Alternative Rules). Τα STARs κατασκευάζουν πλάνα τα οποία μπορεί να χειριστεί η μηχανή εκτέλεσης του συστήματος. Τα πλάνα στο σύστημα αυτό είναι δέντρα από τελεστές ή αλλιώς POPs (Plan Operators). Η διαδικασία παραγωγής πλάνων συνοπτικά έχει ως εξής:
(α) καλούνται κατάλληλα STARs,
(β) χρησιμοποιείται δυναμικός προγραμματισμός και
(γ) τα πλάνα κατασκευάζονται από κάτω προς τα πάνω.

Τα STARs είναι οι κανόνες παραγωγής μίας γραμματικής που παράγει πλάνα. Ένα STAR καθορίζει τον τρόπο με τον οποίο συνδυάζονται οι POPs σε ένα πλάνο. Ένας απλός STAR μπορεί να κατασκευάσει ένα μόνο POP καλώντας τον κατασκευαστή του. Ο κατασκευαστής ενός POP ανακτά χώρο για τον POP, αρχικοποιεί κάποια πεδία και καλεί μία συνάρτηση έτσι ώστε να υπολογιστούν οι ιδιότητες του νέου POP. Πάντως, τα περισσότερα STARs δεν είναι τόσο απλά, αλλά περιέχουν μία συνάρτηση συνθήκης η οποία όταν αποτιμάται σε TRUE το STAR κατασκευάζει το πλάνο του, αλλιώς όχι. Επίσης ένα STAR μπορεί να κατασκευάσει πολλά POPs, πράγμα που συμβαίνει καλώντας τους κατασκευαστές αυτών των POPs ακολουθιακά. Ακόμη, ένα STAR μπορεί να κατασκευάσει πολλά πλάνα, το οποίο γίνεται αρχικοποιώντας το STAR με ένα σύνολο από παραμέτρους, πράγμα που δημιουργεί ένα πλάνο για κάθε ένα στοιχείο του συνόλου αυτού. Στην περίπτωση αυτή η συνάρτηση συνθήκης που αναφέραμε παραπάνω, αποτιμάται για κάθε ένα προκύπτον πλάνο ξεχωριστά. Τέλος, τα STARs μπορούν να καλούν και άλλα STARs.

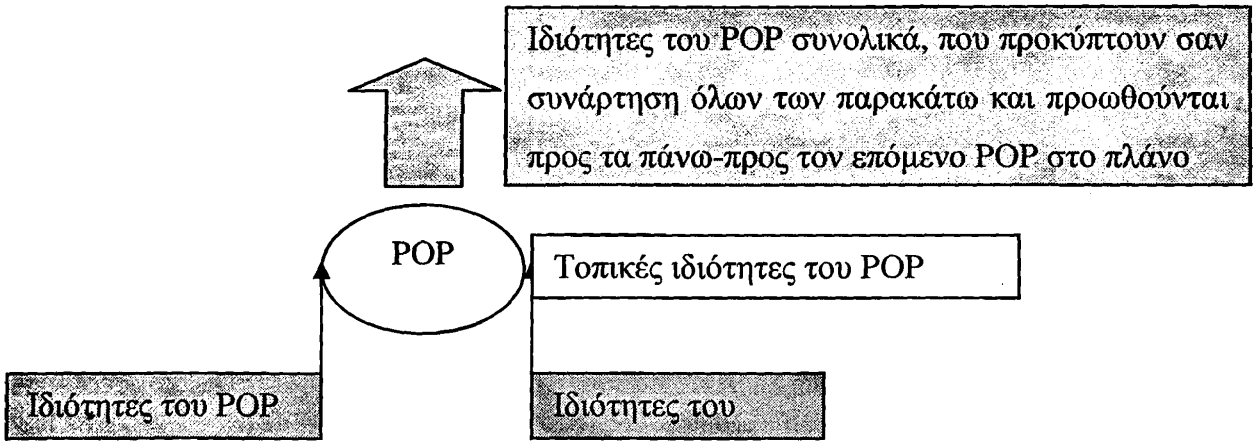
Υπάρχει μία ειδική κατηγορία STARs που ονομάζονται generic STARs τα οποία εφαρμόζονται όταν κάποιο τμήμα εργασίας βρεθεί ότι πρέπει να πραγματοποιηθεί από κάποιον wrapper. Τα generic STARs καλούν τον κατάλληλο wrapper να κατασκευάσει το δικό τους κομμάτι του πλάνου. Από το σύνολο των ολοκληρωμένων πλάνων που απαντούν σε μία ερώτηση ο βελτιστοποιητής επιλέγει το πλάνο που κερδίζει με βάση κάποιο μοντέλο κόστους. Το πλάνο αυτό, στη συνέχεια, μεταφράζεται σε εκτελέσιμη μορφή. Τα πλάνα στο σύστημά μας, όπως αναφέραμε παραπάνω, είναι δέντρα από τελεστές ή POPs. Κάθε POP δέχεται ως είσοδο μία ή πολλές πλειάδες και παράγει κάποια έξοδο. Οι συλλογές πλειάδων παράγονται από άλλα POPs, τα οποία μπορεί να εκτελούν join, ταξινόμηση, εφαρμογή συνθηκών κ.τ.λ. Επίσης το σύστημα παρέχει και ένα generic POP, το PushDown, το οποίο εκφράζει εργασία που πρέπει να πραγματοποιηθεί σε μία πηγή δεδομένων.

Τα πλάνα χαρακτηρίζονται από ένα σύνολο από ιδιότητες. Οι ιδιότητες αυτές είναι ένας εύκολος τρόπος να αναγνωρίζουμε την εργασία που πραγματοποιείται από ένα πλάνο. Είναι ιδιαίτερα σημαντικό να χαρακτηρίζουμε πλάνα με ένα σύνολο από ιδιότητες, διότι τα πλάνα συνθέτονται από generic PushDown POPs. Η εργασία που επιτελούν τα πλάνα αυτά, εξαρτάται από τον wrapper στον οποίο ανήκουν οι πηγές δεδομένων πάνω στις οποίες εφαρμόζονται τα PushDown POPs και δεν είναι γνωστή ούτε από τον mediator ούτε από κάποιον άλλο wrapper του συστήματος. Έτσι οι ιδιότητες των πλάνων παρέχουν στον mediator πληροφορίες ώστε να καταφέρνει να συμπεριλαμβάνει PushDown POPs σε ένα πλάνο. Αυτές οι ιδιότητες λοιπόν που χαρακτηρίζουμε τα πλάνα και την έξοδό τους φαίνονται στον πίνακα 4.1.

Οι ιδιότητες ενός πλάνου προκύπτουν ως εξής. Ένα πλάνο αποτελείται από ένα σύνολο από POPs. Κάθε POP επίσης χαρακτηρίζεται από ένα σύνολο από ιδιότητες. Οι ιδιότητες ενός POP είναι μία συνάρτηση των ιδιοτήτων της/των εισόδου/ων του και υπολογίζονται καθώς κατασκευάζονται τα POPs από STARs (σχήμα 4.3). Το σύνολο των ιδιοτήτων ενός POP και η σημασία της κάθε μίας αναλύονται στον πίνακα 4.2. Τέλος οι ιδιότητες ενός πλάνου είναι οι ιδιότητες του POP που βρίσκεται πιο πάνω από όλα τα άλλα POPs στο δεδομένο πλάνο.

Πίνακας 4.1: Ιδιότητες Πλάνου

Ιδιότητα	Περιγραφή
Tables	το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join στο πλάνο
Columns	το σύνολο των στηλών της εξόδου του πλάνου
Preds	το σύνολο των συνθηκών που έχουν εφαρμοστεί στο πλάνο
Source	εκεί που παράγεται η έξοδος του πλάνου
Mat	TRUE αν η έξοδος γίνεται materialize, αλλιώς FALSE
Order	μία έκφραση ταξινόμησης αν οι πλειάδες της εξόδου είναι διατεταγμένες, αλλιώς NIL
Cost	εκτιμούμενο κόστος του πλάνου
Cardinality	εκτιμούμενο πλήθος πλειάδων της εξόδου του πλάνου



Σχήμα 4.3: Τρόπος που Προκύπτουν οι Ιδιότητες ενός POP

Πίνακας 4.2: Ιδιότητες POP

Ιδιότητα	Περιγραφή
Tables	το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος
Columns	το σύνολο των στηλών της εξόδου του τελεστή
Preds	το σύνολο των συνθηκών που έχουν εφαρμοστεί στον τελεστή ή σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος
Source	εκεί που παράγεται η έξοδος του τελεστή
Mat	TRUE αν η έξοδος του τελεστή γίνεται materialize, αλλιώς FALSE
Order	μία έκφραση ταξινόμησης αν οι πλειάδες της εξόδου του τελεστή είναι διατεταγμένες, αλλιώς NIL
Cost	εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ων που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι
Cardinality	εκτιμούμενο πλήθος πλειάδων της εξόδου του τελεστή

Για ερωτήσεις τύπου *select-project-join* χρησιμοποιούνται τρία STARs: το AccessRoot STAR για προσπέλαση μίας συλλογής δεδομένων το JoinRoot STAR για την εφαρμογή joins και το FinishRoot STAR για την εξασφάλιση ότι το πλάνο είναι

ολοκληρωμένο. Για να επιτραπεί στον βελτιστοποιητή η δυνατότητα να κατασκευάζει πλάνα για ερωτήσεις πάνω σε δεδομένα που βρίσκονται σε ποικιλόμορφες πηγές δεδομένων, ο mediator περιλαμβάνει διάφορα generic STARs. Αυτά τα STARs κατασκευάζουν το generic PushDown POP που αναφέραμε παραπάνω. Για λόγους διάκρισης από τα υπόλοιπα STARs θα χρησιμοποιούμε το πρόθεμα Repo στα ονόματα των generic STARs. Κατά αναλογία με τα STARs που αναφέραμε πριν για ερωτήσεις τύπου *select-project-join* υπάρχουν τα αντίστοιχα generic STARs, RepoAccess STAR και RepoJoin STAR ενώ δεν υπάρχει αντίστοιχο του FinishRoot STAR. Δεν υπάρχει διότι το FinishRoot STAR εφαρμόζεται μόνο στον mediator κι όχι σε κάποια πηγή δεδομένων.

Αξίζει να σημειωθεί ότι και οι wrappers κατασκευάζουν τα πλάνα τους χρησιμοποιώντας STARs. Έτσι όταν θα πρέπει να χαρακτηρίσουμε την εργασία που πραγματοποιήθηκε από κάποιον wrapper θα χρησιμοποιούμε wrapper STARs και wrapper POPs. Στα ονόματα των wrapper STARs θα προσθέτουμε το πρόθεμα plan_ έτσι ώστε να τα ξεχωρίζουμε από τα υπόλοιπα STARs. Τα wrapper STARs έχουν την ιδιότητα ότι είναι απλά, δηλαδή είναι STARs τα οποία δημιουργούν μόνο ένα POP. Δε χρειάζεται ένα wrapper STAR ούτε να κατασκευάζει πολλά POPs ούτε να καλεί άλλα STARs. Το γεγονός αυτό εξηγείται για δύο λόγους. Πρώτον, ο mediator παρέχει μία πανίσχυρη μηχανή εκτέλεσης ερωτήσεων που έχει τη δυνατότητα να πραγματοποιήσει οτιδήποτε παραλείψει ο wrapper. Δεύτερον, οι wrapper STARs μοντελοποιούν το «τι» μπορεί να κάνει ο wrapper κι όχι το «πώς» ακριβώς θα γίνει αυτό.

Ο βελτιστοποιητής του συστήματος αποτιμά πλάνα καλώντας τα κατάλληλα STARs. Τα πλάνα για ερωτήσεις τύπου *select-project-join* αποτιμούνται από κάτω προς τα πάνω σε τρεις φάσεις.

(α) Στην *πρώτη φάση* εφαρμόζεται το AccessRoot STAR σε κάθε σχέση που χρησιμοποιείται στην ερώτηση. Σε αυτή τη χρονική στιγμή ο mediator δεν έχει αποθηκευμένα δεδομένα, οπότε το AccessRoot STAR καλεί το RepoAccess STAR.

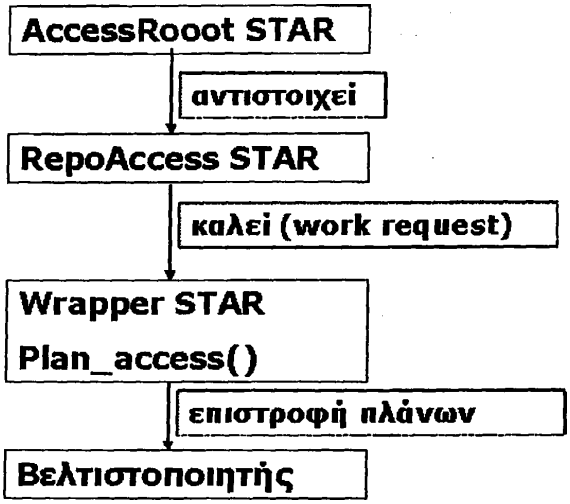
(β) Στη *δεύτερη φάση* εφαρμόζεται το JoinRoot STAR, το οποίο καλεί το RepoJoin STAR όπως επίσης κι άλλα join STARs το καθένα από τα οποία αντιπροσωπεύει μία join μέθοδο του mediator. Το JoinRoot STAR εφαρμόζεται επαναληπτικά περνώντας

σε αυτό κάθε φορά δύο πλάνα και μία συνθήκη join. Αρχικά κάθε πλάνο είναι κάποιο από αυτά που αποτιμήθηκαν στην πρώτη φάση. Όταν όλα τα πιθανά πλάνα για join δύο συλλογών δεδομένων ελεγχθούν, προχωράμε για κατασκευή πλάνων για join τριών συλλογών δεδομένων. Καλείται το JoinRoot STAR να συνδυάσει πλάνα προσπέλασης μίας συλλογής δεδομένων με πλάνα που μόλις πριν κατασκευάστηκαν για join δύο συλλογών δεδομένων. Αυτό έχει ως αποτέλεσμα την κατασκευή πλάνων για join τριών συλλογών δεδομένων και συνεχίζοντας με την ίδια διαδικασία κατασκευάζονται πλάνα για join μεταξύ όλων των συλλογών δεδομένων που παίρνουν μέρος στην ερώτηση. Στο σημείο αυτό πρέπει να σημειωθεί ότι ο βελτιστοποιητής εφαρμόζει δυναμικό προγραμματισμό με σκοπό να βρει το καλύτερο πλάνο και με λογικό κόστος. Σε κάθε βήμα της αποτίμησης, ο βελτιστοποιητής εφαρμόζει την τεχνική του κλαδέματος. Σύμφωνα με αυτήν, κλαδεύεται ένα πλάνο αν έχει μεγαλύτερο κόστος από κάποιο άλλο που κάνει την ίδια εργασία και επιπλέον οι ιδιότητές του είναι υποσύνολο των ιδιοτήτων του άλλου πλάνου.

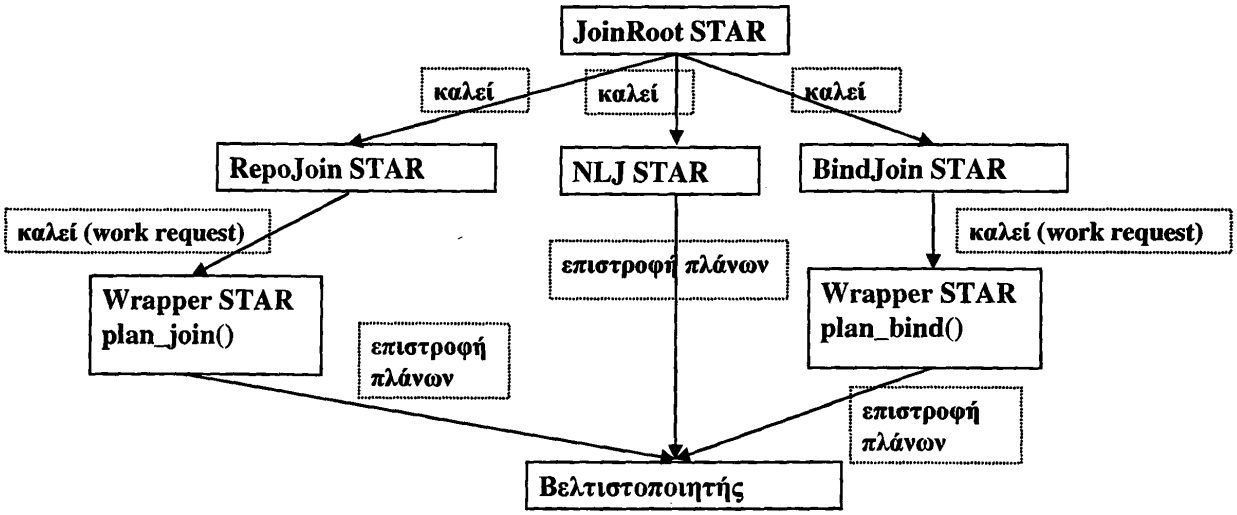
(γ) Κατά την *τρίτη φάση*, εφαρμόζεται το FinishRoot STAR έτσι ώστε όλα τα πλάνα που έχουν επιβιώσει από τη διαδικασία του κλαδέματος στην προηγούμενη φάση να ολοκληρωθούν. Δηλαδή κατά τη φάση αυτή παράγονται τελικά πλάνα της ερώτησης, καθώς εφαρμόζει ο mediator οποιεσδήποτε λειτουργίες έχουν απομείνει για να ολοκληρωθεί η ερώτηση και μέχρι εκείνο το σημείο είχαν παραληφθεί. Από τα τελικά πλάνα που παράγονται, τα οποία όλα χαρακτηρίζονται από τις ίδιες ιδιότητες, επιλέγεται προς εκτέλεση αυτό με το μικρότερο κόστος.

Το κόστος ενός πλάνου είναι το άθροισμα από κόστη για τοπική επεξεργασία, κόστη για επικοινωνία και κόστη για αρχικοποίηση υποερωτήσεων και μεθόδων. Τα δύο τελευταία κόστη αποτιμούνται από συναρτήσεις του mediator χρησιμοποιώντας σταθερές που αποθηκεύονται στον κατάλογό του, ενώ τα κόστη της πρώτης κατηγορίας αποτιμούνται από ένα μοντέλο κόστους. Αυτό το μοντέλο κόστους το παρέχει ο mediator και περιλαμβάνει CPU και I/O κόστη. Τα τοπικά κόστη επεξεργασίας των wrappers και των πηγών δεδομένων τους πρέπει να εκτιμηθούν από μοντέλα κόστους που ορίζονται για κάθε wrapper ξεχωριστά.

Ακολουθεί σχηματική αναπαράσταση των τριών φάσεων αποτίμησης μίας ερώτησης τύπου *select-project-join* από τον βελτιστοποιητή. Παρουσιάζονται οι φάσεις 1, 2 και 3 στα αντίστοιχα σχήματα 4.4, 4.5 και 4.6.



Σχήμα 4.4: Η Φάση 1 της Αποτίμησης Πλάνων



Σχήμα 4.5: Η Φάση 2 της Αποτίμησης Πλάνων

Σχήμα 4.6: Η Φάση 3 της Αποτίμησης Πλάνων

Υπενθυμίζουμε ότι η πρώτη φάση πραγματοποιείται για όλες τις συλλογές δεδομένων που περιλαμβάνονται στην ερώτηση, η δεύτερη φάση πραγματοποιείται επαναληπτικά μέχρι να κάνουν join όλες οι συλλογές δεδομένων που παίρνουν μέρος στην ερώτηση, ενώ η τρίτη φάση πραγματοποιείται για κάθε πλάνο που επιβίωσε από τις δύο πρώτες

Ακολουθεί ένα παράδειγμα ερώτησης έτσι ώστε να γίνει πιο ξεκάθαρη η διαδικασία αποτίμησης μίας ερώτησης και η παραγωγή εναλλακτικών πλάνων εκτέλεσής της. Στο παράδειγμα αυτό υπάρχει ένα σύνολο από αυτοκίνητα τα οποία περιέχουν το καθένα στην τοπική του βάση δύο σχέσεις: την σχέση BRANDS, η οποία είναι αποθηκευμένη εξολοκλήρου στο κάθε αυτοκίνητο και τη σχέση CARS, η οποία αρχικά περιέχει μόνο την πλειάδα που αντιστοιχεί στο αυτοκίνητο στου οποίου τη βάση ανήκει η σχέση. Κατά την απάντηση μίας ερώτησης, η σχέση CARS γεμίζει συλλέγοντας πλειάδες από τα άλλα αυτοκίνητα, κάθε ένα από τα οποία την τροφοδοτεί με την αντίστοιχη πλειάδα του. Το αυτοκίνητο στο οποίο αναφερόμαστε στο παράδειγμά μας είναι το CAR1 του οποίου η βάση περιέχει τις σχέσεις που φαίνονται στο σχήμα 4.7. Σημειώνουμε ότι το γνώρισμα BRAND στη σχέσης CAR1 είναι γραμμένο με italics για να δείξουμε ότι αποτελεί ξένο κλειδί με αναφορά στο γνώρισμα BRAND της σχέσης BRANDS. Στο σχήμα 4.8 δίνεται η ερώτηση που πραγματοποιείται από το αυτοκίνητο CAR1.

CAR1(ID, PLATE, *BRAND*, VEL)
ID: ένας ακέραιος αριθμός που αποτελεί αναγνωριστικό του κάθε αυτοκινήτου
PLATE: ο αριθμός κυκλοφορίας του αυτοκινήτου
BRAND: η μάρκα του αυτοκινήτου
VEL: η ταχύτητα του αυτοκινήτου σε χιλιόμετρα ανά ώρα

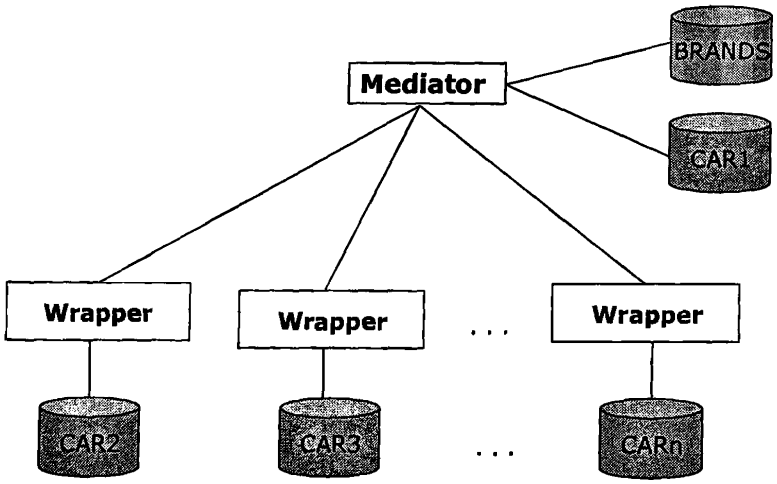
BRANDS(BRAND, COUNTRY, METRICS_SYSTEM)
BRAND: μία μάρκα αυτοκινήτων
COUNTRY: η χώρα κατασκευής της συγκεκριμένης μάρκας αυτοκινήτων
METRICS_SYSTEM: το σύστημα μετρικής που χρησιμοποιείται για να μετρήσει διάφορα χαρακτηριστικά των αυτοκινήτων της δεδομένης μάρκας

Σχήμα 4.7: Το Σχήμα της Βάσης του CAR1

```
SELECT ID,VEL
FROM CARS,BRANDS
WHERE CARS.BRAND=BRANDS.BRAND AND BRANDS.COUNTRY!='GB'
```

Σχήμα 4.8: Η Ερώτηση που Τίθεται στο CAR1

Στο επόμενο σχήμα παρουσιάζουμε την αρχιτεκτονική του συστήματος, που περιγράψαμε παραπάνω (σχήμα 4.9).



Σχήμα 4.9: Αρχιτεκτονική του Συστήματος

Ακολουθεί εφαρμογή των τριών φάσεων αποτίμησης τις ερώτησης.

1. AccessRoot STAR

RepoAccess STAR → PushDown POP για τα CAR2, CAR3, CARn.

AccessRoot STAR → Scan POP για την CAR1 και την BRANDS.

2. JoinRoot STAR

Plan_Join POP

NLJ POP

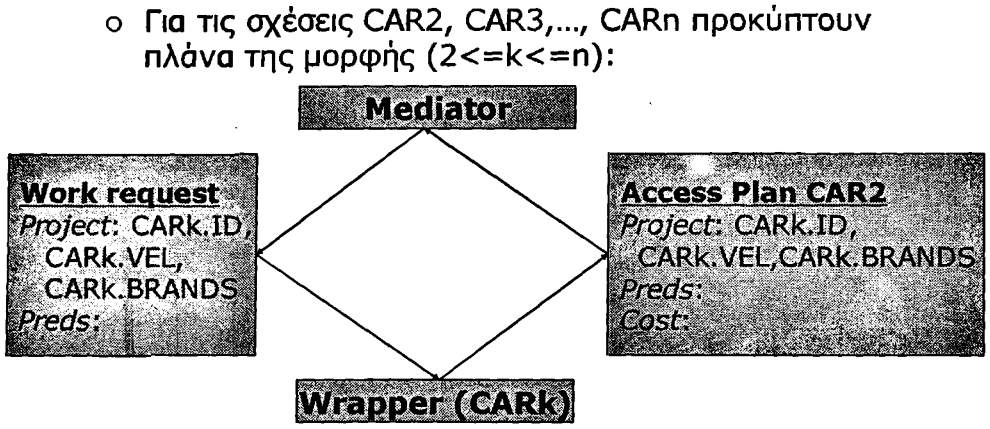
Bind_Join POP

3. FinishRoot STAR

Union

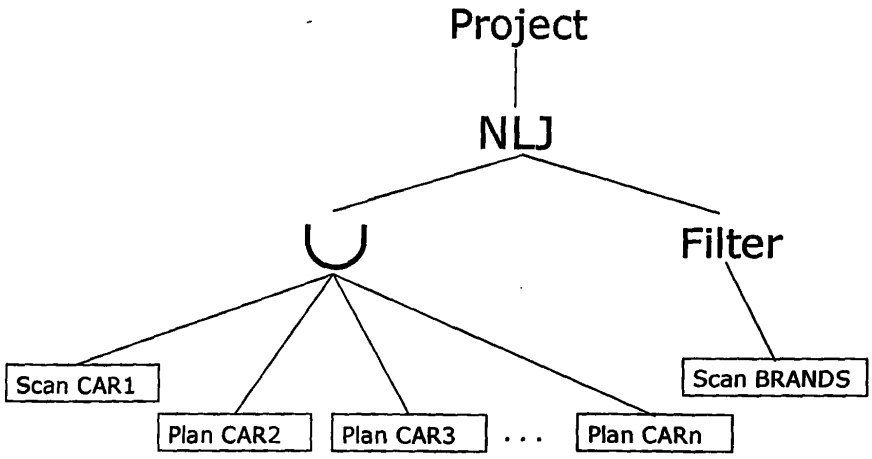
Project

Για τα πλάνα προσπέλασης των σχέσεων που λαμβάνουν μέρος στην ερώτηση ισχύουν τα παρακάτω (σχήμα 4.10):

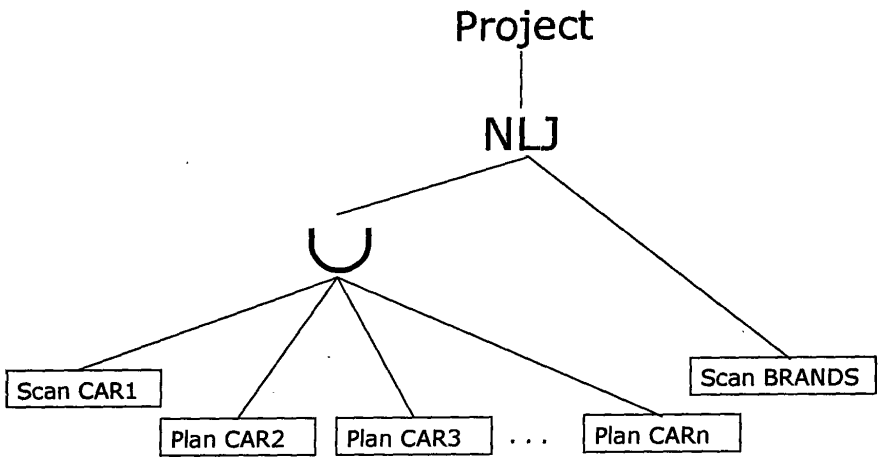


Σχήμα 4.10: Τρόπος που Προκύπτουν τα Πλάνα των Σχέσεων

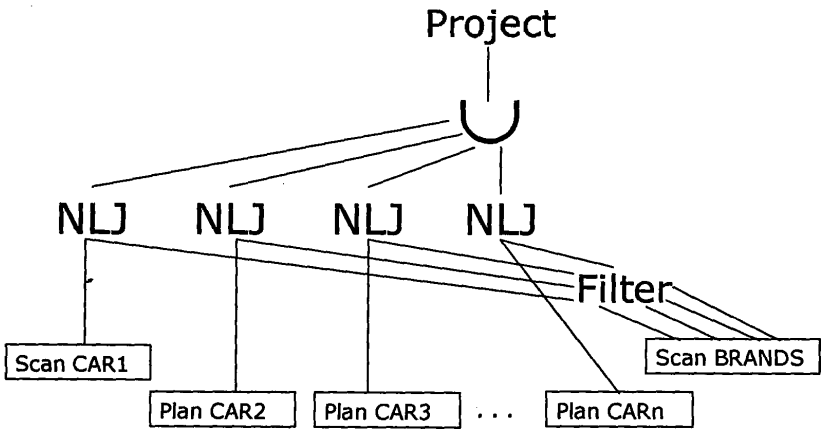
Στα παρακάτω σχήματα αναπαρίστανται όλα τα εναλλακτικά πλάνα εκτέλεσης της ερώτησης του παραδείγματος εφαρμόζοντας τις τρεις φάσεις αποτίμησης της ερώτησης.



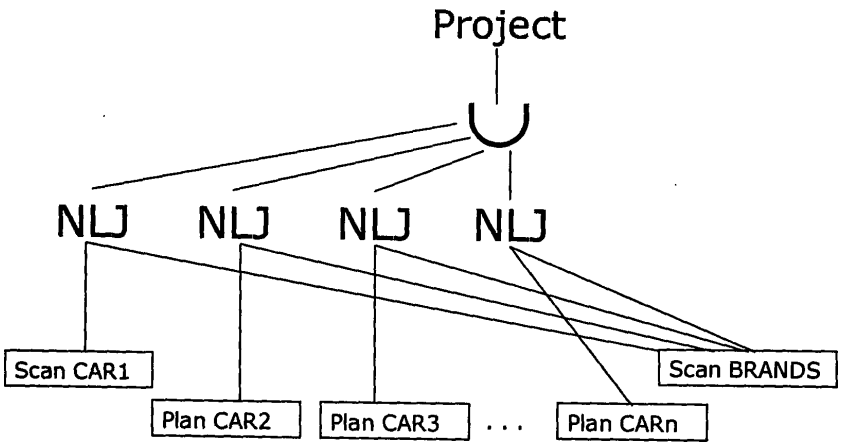
Σχήμα 4.11: Πρώτο Εναλλακτικό Πλάνο



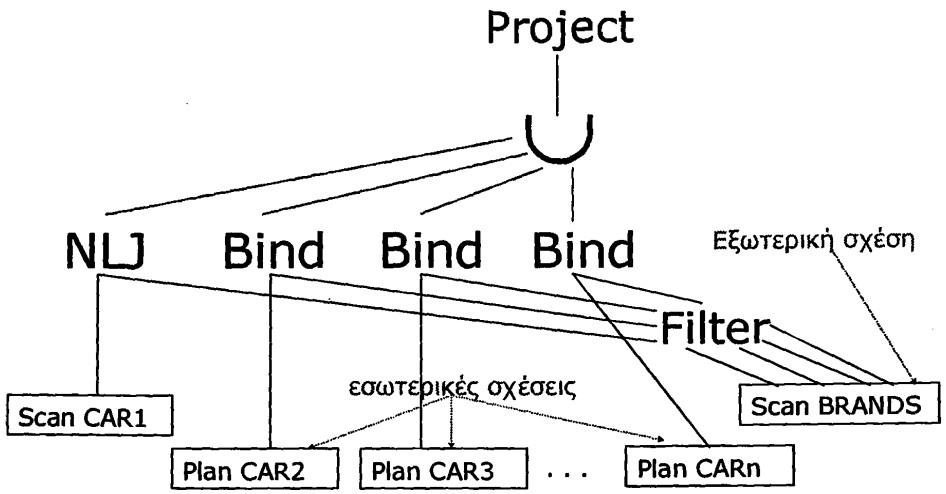
Σχήμα 4.12: Δεύτερο Εναλλακτικό Πλάνο



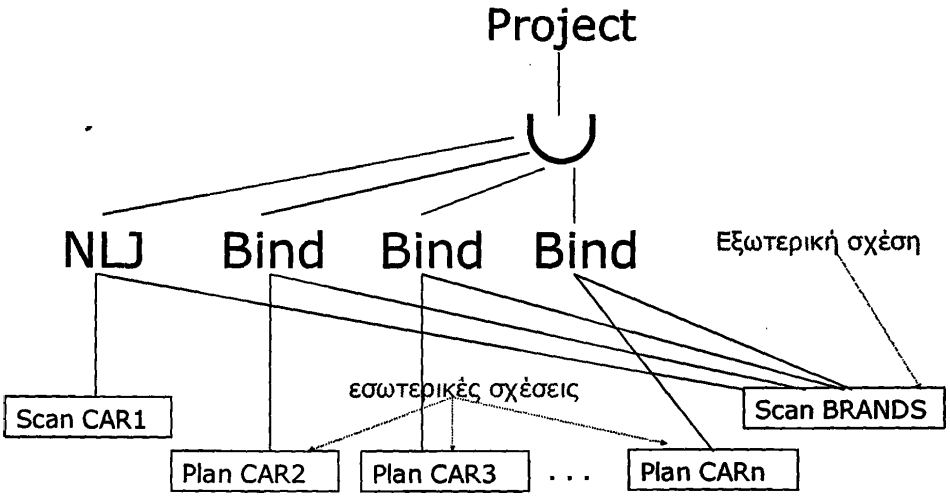
Σχήμα 4.13: Τρίτο Εναλλακτικό Πλάνο



Σχήμα 4.14: Τέταρτο Εναλλακτικό Πλάνο



Σχήμα 4.14: Πέμπτο Εναλλακτικό Πλάνο



Σχήμα 4.15: Έκτο Εναλλακτικό Πλάνο

Το σύστημα Clio μας δίνει ένα υπόβαθρο για επεξεργασία ερωτήσεων σε ένα περιβάλλον ετερογενών βάσεων δεδομένων. Όμως υπάρχουν κάποιες βασικές διαφορές σε σχέση με τη δική μας εφαρμογή οι οποίες επικεντρώνονται σε τρία σημεία:

(α) Στο δικό μας περιβάλλον δεν υπάρχει η έννοια του global σχήματος, αλλά κάθε peer έχει την τοπική του βάση δεδομένων με τους πίνακες που την αποτελούν να έχουν το δικό τους τοπικό σχήμα.

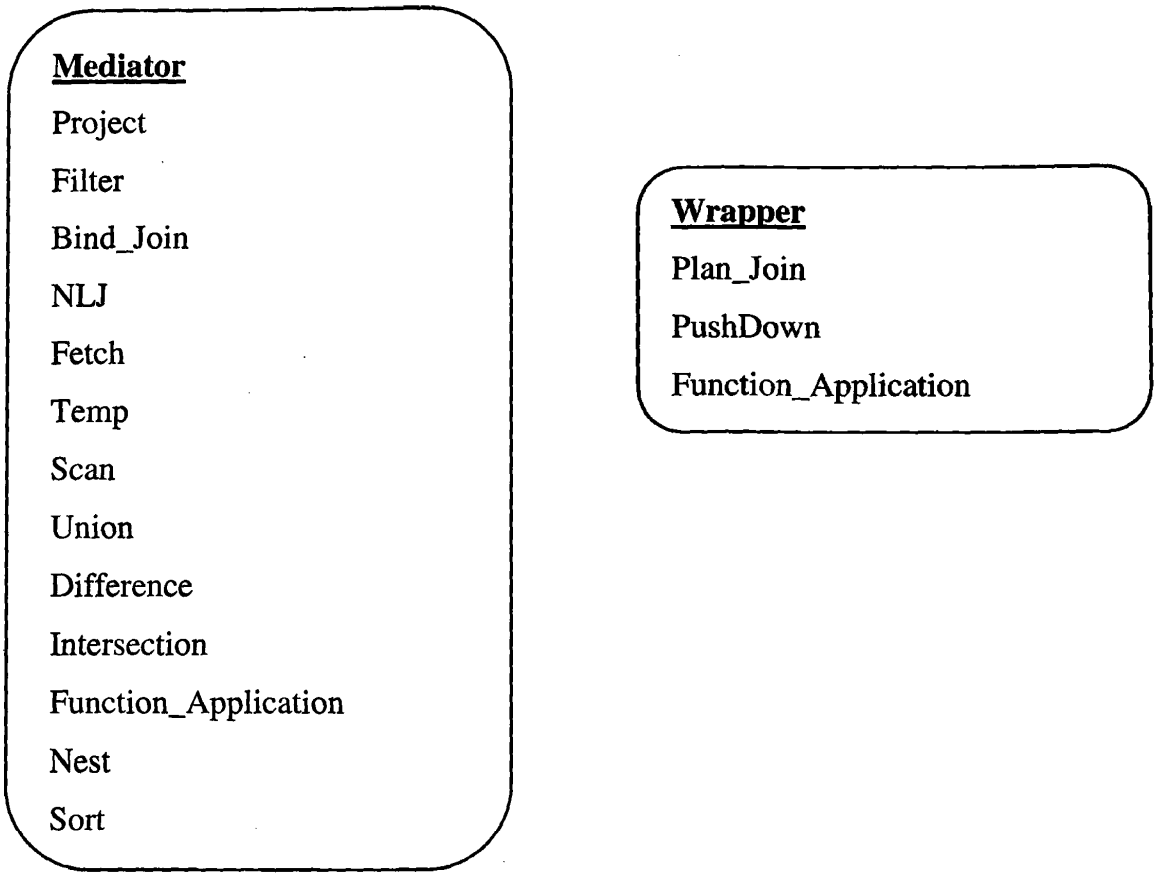
(β) Στο δικό μας περιβάλλον δεν υπάρχει ο κλασσικός wrapper πάνω από κάθε συστατική βάση δεδομένων, όπως στο Clio. Στο Clio, ο mediator δεν επικοινωνούσε ποτέ απευθείας με κάποια συστατική βάση δεδομένων, αλλά πάντα μέσω του ενδιάμεσου wrapper. Στη δική μας εφαρμογή έχουμε απευθείας επικοινωνία του mediator με web services των διάφορων peers.
(γ) Στο Clio αντιμετωπίζονται ερωτήσεις της κατηγορίας των SPJ ερωτήσεων. Αυτές οι ερωτήσεις αποτελούν υποσύνολο των ερωτήσεων που μπορεί να πραγματοποιήσει κάποιος στην SQL. Στη δική μας εφαρμογή, εισάγοντας την γλώσσα SQL^P, ως επέκταση της κλασσικής SQL, μπορούμε να επεξεργαστούμε όλες τις ερωτήσεις που μπορούν να εκφραστούν σε SQL και με κάποιες επιπλέον δυνατότητες που παρέχει η γλώσσα που εισάγουμε.

Αυτές οι βασικές διαφορές μας οδηγούν να χρησιμοποιήσουμε ως υπόβαθρο το Clio, αλλά να αναπτύξουμε κι όλα αυτά τα απαραίτητα στοιχεία έτσι ώστε να ικανοποιήσουμε όλες τις απαιτήσεις της εφαρμογής μας.

Στην επόμενη ενότητα παρατίθεται ο τυπικός ορισμός όλων των τελεστών οι οποίοι λαμβάνουν μέρος σε πλάνα του Clio. Από αυτούς τους τελεστές άλλοι εφαρμόζονται στον mediator, άλλοι στους wrappers κι άλλοι παντού. Από αυτούς τους τελεστές θα αναφέρουμε ποιοι λαμβάνουν μέρος σε πλάνα του δικού μας συστήματος και τους δανειζόμαστε και ποιοι όχι.

4.3 Τυπικός Ορισμός Τελεστών

Στο Clio όταν τίθενται ερωτήσεις, πραγματοποιείται η κατασκευή εναλλακτικών πλάνων εκτέλεσης για κάθε ερώτηση. Τα πλάνα αυτά, όπως έχει ήδη αναφερθεί, είναι δέντρα από τελεστές που εφαρμόζονται από κάτω προς τα πάνω ώστε τελικά να δίνεται ορθή και ολοκληρωμένη απάντηση στην ερώτηση που τίθεται. Οι τελεστές έχουν τη δυνατότητα να εφαρμόζονται είτε στον mediator είτε σε κάποιον wrapper είτε και στα δύο αυτά συστατικά της αρχιτεκτονικής της εφαρμογής μας. Στο σχήμα 4.16 παρουσιάζονται οι τελεστές αυτοί και μάλιστα διαχωρίζοντάς τους ανάλογα με το πού εφαρμόζονται, στον mediator ή σε κάποιον wrapper ή και στα δύο.



Σχήμα 4.16: Παράθεση των Τελεστών

Ακολουθεί για κάθε έναν από τους παραπάνω, ανεξάρτητα σε ποια κατηγορία ανήκουν, ο τυπικός ορισμός τους, ένας ορισμός δηλαδή που περιλαμβάνει την περιγραφή του τελεστή, τον συμβολισμό του, την είσοδό του, το σχήμα εξόδου, το σύνολο πλειάδων εξόδου και τις ιδιότητες του τελεστή.

Project: Ο τελεστής αυτός πραγματοποιεί προβολή κάποιων συγκεκριμένων στηλών ενός πίνακα αγνοώντας τις υπόλοιπες.

Συμβολισμός: *Project(T, L).*

Είσοδος: Ένας πίνακας *T* και μία λίστα *L* από γνωρίσματα του πίνακα *T*. Η λίστα *L* αντιπροσωπεύει τα προβαλλόμενα γνωρίσματα του *T*.

Σχήμα εξόδου: Η έξοδος έχει σχήμα που αποτελείται μόνο από τα γνωρίσματα που περιλαμβάνονται στη λίστα *L*.

Πλειάδες εξόδου: Στην έξοδο περιλαμβάνονται όλες οι πλειάδες του πίνακα *T*, πραγματοποιώντας όμως διαγραφή διπλότυπων που είναι απαραίτητη λόγω της προβολής σε κάποιο υποσύνολο των γνωρισμάτων του *T*.

Ιδιότητες:

Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: το σύνολο των στηλών που ανήκουν στην *L*.

Predicates: το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source: πάντα στον mediator

Materialize: FALSE.

Order: η ίδια με αυτήν του τελεστή που βρίσκεται ακριβώς από κάτω στο δέντρο, αλλιώς NIL.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.

Cardinality: εκτιμούμενο πλήθος πλειάδων της εξόδου του τελεστή.

Filter: Ο τελεστής αυτός πραγματοποιεί επιλογή κάποιων συγκεκριμένων πλειάδων ενός πίνακα σύμφωνα με κάποια/ες συνθήκη/ες επιλογής.

Συμβολισμός: Filter(T, Preds)

Είσοδος: Ένας πίνακας *T* και μία λίστα *Preds* από συνθήκες που εφαρμόζονται πάνω στις πλειάδες του πίνακα *T*.

Σχήμα εξόδου: Ακριβώς το ίδιο με αυτό του πίνακα *T*.

Πλειάδες εξόδου: Οι πλειάδες του *T* οι οποίες ικανοποιούν τις συνθήκες που καθορίζουν τα *Preds*.

Ιδιότητες:

Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: το σύνολο όλων των στηλών του πίνακα *T*.

Predicates: το σύνολο των συνθηκών *Preds* μαζί με το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source: πάντα στον mediator.

Materialize: FALSE.

Order: η ίδια με αυτήν του τελεστή που βρίσκεται ακριβώς από κάτω στο δέντρο, αλλιώς NIL.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.

Cardinality: εκτιμούμενο πλήθος πλειάδων της εξόδου του τελεστή.

Sort: Ο τελεστής αυτός πραγματοποιεί ταξινόμηση των πλειάδων ενός πίνακα ως προς κάποια έκφραση ταξινόμησης που του δίνεται και τελικά διατάσσει τις πλειάδες του πίνακα αυτού με βάση την προαναφερθείσα έκφραση.

Συμβολισμός: Sort(T, Sort_Expression).

Είσοδος: Ένας πίνακας *T* και μία έκφραση ταξινόμησης *Sort_Expression* με βάση την οποία διατάσσονται οι πλειάδες του πίνακα *T*.

Σχήμα εξόδου: Ακριβώς το ίδιο με τον πίνακα *T*.

Πλειάδες εξόδου: Όλες οι πλειάδες του πίνακα *T*.

Ιδιότητες:

Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: το σύνολο των στηλών του πίνακα *T*.

Predicates: το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source: πάντα στο mediator.

Materialize: FALSE.

Order: η έκφραση ταξινόμησης *Sort_Expression* που δίνεται στην είσοδο.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.

Cardinality: ίσο με το πλήθος των πλειάδων του *T*.

Temp: Ο τελεστής αυτός πραγματοποιεί προσωρινή αποθήκευση ενός πίνακα προκειμένου να χρησιμοποιηθεί στη συνέχεια όποτε αυτό κριθεί απαραίτητο από το σύστημα.

Συμβολισμός: *Temp(T)*

Είσοδος: Ένας πίνακας *T*.

Σχήμα εξόδου: Το ίδιο με το σχήμα του *T*.

Πλειάδες εξόδου: Όλες οι πλειάδες του πίνακα *T*.

Ιδιότητες:

Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: όλες οι στήλες του πίνακα *T*.

Predicates: το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source: πάντα ο mediator.

Materialize: TRUE.

Order: η ίδια με αυτήν του τελεστή που βρίσκεται ακριβώς από κάτω στο δέντρο, αλλιώς NIL.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.

Cardinality: ίσο με το πλήθος των πλειάδων του *T*.

Scan: Ο τελεστής αυτός πραγματοποιεί ανάγνωση ενός πίνακα ο οποίος είναι αποθηκευμένος προσωρινά κάπου στον mediator του συστήματος.

Συμβολισμός: *Scan(T)*

Είσοδος: Ο πίνακας *T*.

Σχήμα εξόδου: Ίδιο με αυτό του πίνακα *T*.

Πλειάδες εξόδου: Όλες οι πλειάδες του πίνακα *T*.

Ιδιότητες:

Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: όλες οι στήλες του πίνακα *T*.

Predicates: το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source: πάντα ο mediator.

Materialize: FALSE.

Order: η ίδια με αυτήν του τελεστή που βρίσκεται ακριβώς από κάτω στο δέντρο, αλλιώς NIL.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.

Cardinality: ίσο με το πλήθος των πλειάδων του *T*.

Union: Ο τελεστής αυτός πραγματοποιεί ένωση δύο ή περισσότερων πινάκων (έστω n στο πλήθος) με την έννοια ότι δημιουργεί έναν νέο πίνακα ο οποίος περιέχει όλες τις πλειάδες που περιλαμβάνονται στον καθένας από τους επιμέρους πίνακες ξεχωριστά. Επίσης πραγματοποιείται απαλοιφή διπλότυπων όπου απαιτείται. Σημαντικό στοιχείο που πρέπει να αναφέρουμε αποτελεί το γεγονός ότι οι επιμέρους πίνακες θα πρέπει να έχουν το ίδιο σχήμα έτσι ώστε να ορίζεται η πράξη της ένωσης μεταξύ τους.

Συμβολισμός: Union(T_1, T_2, …, T_n).

Είσοδος: Ένα σύνολο από πίνακες T_1, T_2, …, T_n οι οποίοι όλοι χαρακτηρίζονται από την ιδιότητα ότι έχουν το ίδιο σχήμα.

Σχήμα εξόδου: Ίδιο με το σχήμα των πινάκων T_1, T_2, …, T_n.

Πλειάδες εξόδου: Όλες όσες ανήκουν στους T_1, T_2, …, T_n, λαμβάνοντας υπόψη ότι πραγματοποιείται απαλοιφή διπλότυπων όπου χρειάζεται.

Ιδιότητες:

Tables : το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: οι στήλες ενός οποιουδήποτε από τους πίνακες T_1, T_2, …, T_n.

Predicates: το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source, πάντα στον mediator.

Materialize: FALSE.

Order: NIL.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.

Cardinality: εκτιμούμενο πλήθος πλειάδων της εξόδου του τελεστή, περίπου ίσο με το άθροισμα του πλήθους των πλειάδων των πινάκων T_1, T_2, …, T_n.

Difference: Ο τελεστής αυτός πραγματοποιεί την πράξη της διαφοράς μεταξύ δύο πινάκων. Δηλαδή από τον πρώτο πίνακα αφαιρεί όσες πλειάδες αυτού υπάρχουν επίσης και στον δεύτερο πίνακα. Σε αυτήν την περίπτωση δεν απαιτείται απαλοιφή διπλότυπων, αλλά όπως και στην πράξη της ένωσης υπάρχει η απαίτηση οι εμπλεκόμενοι πίνακες να έχουν οπωσδήποτε το ίδιο σχήμα για να μπορεί να πραγματοποιηθεί η πράξη μεταξύ τους.

Συμβολισμός: *Dif(T, T')*.

Είσοδος: Δύο πίνακες οι *T*, *T'* οι οποίοι έχουν το ίδιο σχήμα.

Σχήμα εξόδου: Το ίδιο με αυτό των *T*, *T'*.

Πλειάδες εξόδου: Όλες οι πλειάδες οι οποίες ανήκουν στον *T* αλλά επίσης δεν ανήκουν στον *T'*.

Ιδιότητες:

Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: οι στήλες ενός οποιουδήποτε από τους πίνακες *T*, *T'*.

Predicates: το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source: εκεί που παράγεται η έξοδος – πάντα στον mediator.

Materialize: FALSE.

Order: Η ίδια με αυτήν του πίνακα *T*, αλλιώς NIL.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.

Cardinality: εκτιμούμενο πλήθος πλειάδων της εξόδου του τελεστή.

Intersection: Ο τελεστής αυτός πραγματοποιεί την πράξη της τομής μεταξύ δύο πινάκων. Με βάση δύο πίνακες δημιουργείται ένας νέος πίνακας ο οποίος περιλαμβάνει όσες πλειάδες ανήκουν τόσο στον ένα όσο και στον άλλο πίνακα. Απαλοιφή διπλότυπων προφανώς δεν χρειάζεται και κατά αναλογία με την ένωση και τη διαφορά οι δύο πίνακες στους οποίους εφαρμόζεται η πράξη πρέπει να έχουν το ίδιο σχήμα.

Συμβολισμός: Intersection(T, T').

Είσοδος: Δύο πίνακες *T, T'* που έχουν το ίδιο σχήμα.

Σχήμα εξόδου: Το ίδιο με αυτό των πινάκων *T, T'*.

Πλειάδες εξόδου: Το σύνολο των πλειάδων οι οποίες ανήκουν στον *T* και ανήκουν επίσης και στον *T'*.

Ιδιότητες:

Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: οι στήλες ενός οποιουδήποτε από τους πίνακες *T, T'*.

Predicates: το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source: εκεί που παράγεται η έξοδος – πάντα στον mediator.

Materialize: FALSE.

Order: η ίδια με αυτήν του πίνακα *T*, αλλιώς NIL.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.

Cardinality: εκτιμούμενο πλήθος πλειάδων της εξόδου του τελεστή.

Fetch: Ο τελεστής αυτός εφαρμόζεται από τον mediator. Με τον τελεστή *Fetch* δίνεται η δυνατότητα στον mediator να ζητήσει και να λάβει κάποιες επιλεγμένες στήλες ενός πίνακα τις οποίες ο wrapper αυτού του πίνακα δεν μπορεί να του παρέχει. Αυτό γίνεται μέσω του κλειδιού του πίνακα το οποίο ο wrapper περνά στον mediator και αυτός με τη σειρά του χρησιμοποιεί για να προσπελάσει τις πλειάδες του πίνακα και να λάβει τις στήλες που επιθυμεί.

Συμβολισμός: *Fetch(T, L).*

Είσοδος: Ένας πίνακας *T* και μία λίστα *L* από γνωρίσματα του *T*.

Σχήμα εξόδου: Αποτελείται από το σύνολο των γνωρισμάτων του πίνακα *T* τα οποία ανήκουν στην λίστα *L*.

Πλειάδες εξόδου: Όλες οι πλειάδες του πίνακα *T*, λαμβάνοντας υπόψη ότι πραγματοποιείται απαλοιφή διπλότυπων όταν απαιτείται.

Ιδιότητες:

Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: το σύνολο των στηλών του *T* που αναφέρονται στη λίστα *L*.

Predicates: το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source: πού παράγεται η έξοδος – πάντα στον mediator.

Materialize: FALSE.

Order: η ίδια με αυτήν του τελεστή που βρίσκεται ακριβώς από κάτω στο δέντρο, αλλιώς NIL.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.

Cardinality: ίσο με το πλήθος των πλειάδων του πίνακα *T*.

Plan_Join: Ο τελεστής αυτός πραγματοποιεί join μεταξύ δύο πινάκων με βάση κάποια συνθήκη για join. Το συγκεκριμένο join έχει εφαρμογή όταν και οι δύο πίνακες είναι πηγές δεδομένων που βρίσκονται κάτω από τον ίδιο wrapper.

Συμβολισμός: Plan_Join(T, T', P).

Είσοδος: Δύο πίνακες *T, T'* που έχουν την ιδιότητα που αναφέραμε παραπάνω.

Σχήμα εξόδου: Αποτελείται από τις στήλες του *T* μαζί με αυτές του *T'*.

Πλειάδες εξόδου: Όσες πλειάδες ανήκουν στο καρτεσιανό γινόμενο των δύο πινάκων *T, T'* και επίσης ικανοποιούν την συνθήκη που δίνεται στο *P*.

Ιδιότητες:

Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: η ένωση των στηλών του *T* και του *T'*.

Predicates: η συνθήκη του join που δίνεται στο *P* μαζί με το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source: εκεί που παράγεται η έξοδος – πάντα στον wrapper.

Materialize: FALSE.

Order: η ίδια με αυτήν του *T*, αλλιώς NIL.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.
Cardinality: εκτιμούμενο πλήθος πλειάδων της εξόδου του τελεστή.

NLJ: Ο τελεστής αυτός πραγματοποιεί join μεταξύ δύο πινάκων οι οποίοι μπορούν να βρίσκονται οπουδήποτε μέσα στο σύστημά μας.
Συμβολισμός: *NLJ(T, T', P).*
Είσοδος: Δύο πίνακες *T, T'*.
Σχήμα εξόδου: Αποτελείται από τις στήλες του *T* μαζί με αυτές του *T'*.
Πλειάδες εξόδου: Όσες πλειάδες ανήκουν στο καρτεσιανό γινόμενο των δύο πινάκων *T, T'* και επίσης ικανοποιούν την συνθήκη που δίνεται στο *P*.
Ιδιότητες:
Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.
Columns: η ένωση των στηλών του *T* και του *T'*.
Predicates: η συνθήκη του join που δίνεται στο *P* μαζί με το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.
Source: εκεί που παράγεται η έξοδος – πάντα στον mediator.
Materialize: FALSE.
Order: η ίδια με αυτήν του *T*, αλλιώς NIL.
Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.
Cardinality: εκτιμούμενο πλήθος πλειάδων της εξόδου του τελεστή.

Bind_Join: Ο τελεστής αυτός πραγματοποιεί join μεταξύ δύο πινάκων. Στην περίπτωση αυτού του τελεστή ακολουθείται μία διαφορετική από τις παραπάνω τεχνικές. Απαιτείται ο εσωτερικός πίνακας να είναι πηγή δεδομένων και να υποστηρίζει τη λειτουργία του bind. Σύμφωνα με αυτήν ο mediator στον οποίο βρίσκεται ο εξωτερικός πίνακας περνάει τιμές για τη συνθήκη join του εξωτερικού πίνακα με μορφή παραμέτρων στον εσωτερικό πίνακα. Ο εσωτερικός πίνακας με τη σειρά του χρησιμοποιεί αυτές τις τιμές ώστε να φιλτράρει τα αποτελέσματα που

επιστρέφει. Η διαδικασία αυτή συνεχίζεται μέχρις ότου περαστούν όλες οι τιμές για τη συνθήκη join από τον εξωτερικό πίνακα στον εσωτερικό οπότε και παράγεται το τελικό αποτέλεσμα της εφαρμογής του τελεστή.

Συμβολισμός: Bind_Join(T, T', P).

Είσοδος: Δύο πίνακες *T, T'* που έχουν την ιδιότητα που αναφέραμε παραπάνω.

Σχήμα εξόδου: Αποτελείται από τις στήλες του *T* μαζί με αυτές του *T'*.

Πλειάδες εξόδου: Όσες πλειάδες προκύπτουν από την επαναληπτική διαδικασία περάσματος με παραμέτρους τιμών για τη συνθήκη του join από τον εξωτερικό πίνακα προς τον εσωτερικό κι αφού ο τελευταίος κάθε φορά πραγματοποιεί το απαραίτητο φιλτράρισμα στα αποτελέσματα που επιστρέφει με βάση αυτές τις τιμές που του περνιούνται. Η επαναληπτική αυτή διαδικασία τερματίζεται όταν περαστούν όλες οι τιμές από τον εξωτερικό κόμβο.

Ιδιότητες:

Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: η ένωση των στηλών του *T* και του *T'*.

Predicates: η συνθήκη του join που δίνεται στο *P* μαζί με το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source: εκεί που παράγεται η έξοδος – πάντα στον mediator.

Materialize: FALSE.

Order: η ίδια με την T, αλλιώς NIL.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.

Cardinality: εκτιμούμενο πλήθος πλειάδων της εξόδου του τελεστή.

Function_Application: Ο τελεστής αυτός πραγματοποιεί μία διαδικασία κατά την οποία με είσοδο έναν πίνακα παράγει ένα επιπλέον από τα ήδη υπάρχοντα γνωρίσματα και το προσθέτει σαν τελευταία στήλη του πίνακα. Το επιπλέον αυτό γνώρισμα προκύπτει από την εφαρμογή μίας συνάρτησης πάνω στα αρχικά γνωρίσματα του πίνακα.

Συμβολισμός: Function_Application(T, F(T)).

Είσοδος: Ένας πίνακας *T* και μία συνάρτηση πάνω στα γνωρίσματα του *T*.

Σχήμα εξόδου: Το σχήμα του *T* ένωση ένα γνώρισμα που αντιπροσωπεύει το αποτέλεσμα της εφαρμογής της συνάρτησης *F(T)*.

Πλειάδες εξόδου: Οι πλειάδες της εξόδου προκύπτουν από τις πλειάδες του *T* με τη διαφορά ότι σε κάθε πλειάδα προστίθεται στο τέλος της μία επιπλέον τιμή που είναι το αποτέλεσμα της εφαρμογής της συνάρτησης *F* για τις τιμές της συγκεκριμένης πλειάδας του *T*.

Ιδιότητες:

Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: οι στήλες του πίνακα *T* μαζί με μία νέα στήλη που εκφράζει το αποτέλεσμα της εφαρμογής της συνάρτησης *F(T)*.

Predicates: το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source: εκεί που παράγεται η έξοδος – είτε στον mediator είτε σε κάποιον wrapper.

Materialize: FALSE.

Order: η ίδια με αυτήν του τελεστή που βρίσκεται ακριβώς από κάτω στο δέντρο, αλλιώς NIL.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.

Cardinality: ίσο με το πλήθος των πλειάδων του πίνακα *T*.

Nest: Ο τελεστής αυτός πραγματοποιεί ουσιαστικά ένα πρώτο βήμα για να κάνουμε aggregation. Δημιουργεί έναν νέο πίνακα τον *T'* στον οποίο ομαδοποιούνται μαζί όλες οι πλειάδες του πίνακα εισόδου *T* που έχουν τις ίδιες τιμές για ένα σύνολο από γνωρίσματα, σε μία νέα πλειάδα του πίνακα T'. Τα υπόλοιπα γνωρίσματα των πλειάδων αυτών ομαδοποιούνται σε bags από τιμές. Έτσι είναι εύκολο στη συνέχεια να πραγματοποιηθεί aggregation πάνω σε αυτά τα bags από τιμές.

Συμβολισμός: *Nest(T, L)*.

Είσοδος: Ένας πίνακας *T* και μία λίστα *L* γνωρισμάτων του πίνακα *T*. Η λίστα *L* είναι οι λίστα των γνωρισμάτων της ομαδοποίησης.

Σχήμα εξόδου: Το σχήμα της εξόδου είναι ίδιο με αυτό του *T* με τη διαφορά ότι τα γνωρίσματα του *T* που δεν ανήκουν στην λίστα *L* από απλά γνωρίσματα μετατρέπονται σε πλειότιμα γνωρίσματα.

Πλειάδες εξόδου: Κάθε πλειάδα του *T* ελέγχει αν υπάρχει στον *T'* πλειάδα με τις ίδιες τιμές στα γνωρίσματα ομαδοποίησης. Αν υπάρχει τότε στον *T'* δεν προστίθεται νέα πλειάδα απλά στην ήδη υπάρχουσα προστίθενται οι τιμές των γνωρισμάτων αυτής που δεν ανήκουν στην *L* στα αντίστοιχα bags του *T'*. Στην αντίθετη περίπτωση προστίθεται στον *T'* ως έχει. Η διαδικασία ολοκληρώνεται όταν πραγματοποιηθεί για όλες τις πλειάδες του *T* οπότε και δημιουργείται ο τελικός πίνακας *T'*.

Ιδιότητες:

Tables: το σύνολο των πινάκων που έχουν προσπελαστεί ή έχουν κάνει join κατά την εφαρμογή του τελεστή ή κάποιου άλλου τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Columns: το σύνολο των στηλών του πίνακα *T*.

Predicates: το σύνολο των συνθηκών που έχουν εφαρμοστεί σε κάποιον άλλον τελεστή που βρίσκεται κάτω από αυτόν στο δέντρο εάν υπάρχει τέτοιος.

Source: εκεί που παράγεται η έξοδος – είτε στον mediator είτε σε οποιονδήποτε wrapper.

Materialize: FALSE.

Order: η ίδια με αυτήν του τελεστή που βρίσκεται ακριβώς από κάτω στο δέντρο, αλλιώς NIL.

Cost: εκτιμούμενο κόστος του τελεστή + εκτιμούμενο κόστος του/των τελεστή/ών που βρίσκονται κάτω από αυτόν στο δέντρο εάν υπάρχουν τέτοιοι.

Cardinality: εκτιμούμενο πλήθος πλειάδων της εξόδου του τελεστή.

4.4 Ορισμός Νέων Τελεστών

Από τους παραπάνω τελεστές επιλέγουμε αυτούς που ταιριάζουν στη δική μας εφαρμογή και συμμετέχουν σε πλάνα εκτέλεσης ερωτήσεων της δικής μας εφαρμογής. Επίσης προκύπτει η ανάγκη ορισμού και εισαγωγής κάποιων νέων τελεστών οι οποίοι θα έχουν τη δυνατότητα να υποστηρίζουν και να υλοποιούν τις επιπλέον απαιτήσεις της εφαρμογής μας.

4.4.1 Επιλογή των Κατάλληλων Τελεστών από το Clio

Με βάση τις απαιτήσεις της δικής μας εφαρμογής, επιλέγουμε τους κατάλληλους τελεστές από αυτούς που λαμβάνουν μέρος σε πλάνα ερωτήσεων του Clio. Τους τελεστές αυτούς τους δανειζόμαστε με σκοπό την εφαρμογή τους και σε πλάνα ερωτήσεων της δικής μας εφαρμογής. Είναι οι εξής:

(α) Project

(β) Filter

(γ) Group

(δ) Order

(ε) Join

4.4.2 Εισαγωγή-Ορισμός Νέων Τελεστών

Όπως προαναφέραμε οι παραπάνω τελεστές δεν αρκούν για την κατασκευή πλάνων εκτέλεσης σε ερωτήσεις της γλώσσας SQL^P. Για αυτόν τον λόγο χρειαζόμαστε και κάποιους επιπλέον τελεστές. Οι νέοι τελεστές διακρίνονται σε δύο κατηγορίες, ανάλογα με το πού εφαρμόζονται. Έτσι έχουμε μία κατηγορία για αυτούς που εφαρμόζονται στον κατάλογο του peer που θέτει την ερώτηση και μία δεύτερη κατηγορία για αυτούς που λαμβάνουν μέρος στα πλάνα των ερωτήσεων. Ο ορισμός αυτών των νέων τελεστών ακολουθεί αμέσως παρακάτω.

Τελεστές που εφαρμόζονται στον κατάλογο ενός peer:

(α) CHECK_TABLES:

Ο τελεστής αυτός ελέγχει τους πίνακες που βρίσκονται στο FROM clause της ερώτησης αν είναι νοητοί, υβριδικοί ή τίποτα από τα δύο.

Είσοδος: Το σύνολο των πινάκων που βρίσκονται στο FROM clause της ερώτησης.

Επεξεργασία: Ελέγχεται ο κάθε πίνακας της εισόδου σε ποια κατηγορία ανήκει, νοητός, υβριδικός ή τίποτα από τα δυο μέσω της αντίστοιχης εντολής του CREATE TABLE.

Έξοδος: Οι πίνακες της εισόδου με μία ένδειξη ο καθένας τους, VIRTUAL, HYBRID, LOCAL αν πρόκειται για νοητό ή υβριδικό ή τοπικά αποθηκευμένο πίνακα αντίστοιχα.

(β) CHECK_PEERS:

Ο τελεστής αυτός αποσκοπεί στον καθορισμό του συνόλου των peers που μας ενδιαφέρει να ρωτήσουμε σε σχέση με όλους τους γνωστούς peers με βάση τα χαρακτηριστικά τους. *Είσοδος*: Ο κατάλογος του peer που θέτει την ερώτηση, ο τύπος του πίνακα στον οποίο αναφερόμαστε και οι προτάσεις HORIZON, AVAILABILITY, RESPONSE_TIME και CLASS που βρίσκονται στο WITH clause της ερώτησης.

Επεξεργασία: Ο τελεστής ελέγχει με βάση τον κατάλογο ποιοι peers από αυτούς που περιέχονται σε αυτόν, ικανοποιούν τις συνθήκες των προτάσεων HORIZON, AVAILABILITY, RESPONSE_TIME και CLASS του WITH clause της ερώτησης. *Έξοδος*: Ένα σύνολο από peers, αυτοί που ικανοποιούν τις συνθήκες που δίνονται. και αποτελούν το σύνολο κόμβων ενδιαφέροντος της ερώτησης.

(γ) CHECK_AGE:

Ο τελεστής αυτός αποσκοπεί στον καθορισμό του συνόλου των peers που μας ενδιαφέρει να ρωτήσουμε σε σχέση με όλους τους γνωστούς peers με βάση την ηλικία των πλειάδων που περιέχονται στους νοητούς ή υβριδικούς πίνακες της ερώτησης. Πραγματοποιεί το υπόλοιπο μέρος του βήματος 2 του δοθέντος αλγορίθμου και σε συνδυασμό με τον τελεστή CHECK_PEERS έχει ως αποτέλεσμα το σύνολο των κόμβων ενδιαφέροντος της ερώτησης.

Είσοδος: Ο κατάλογος του peer που θέτει την ερώτηση, ο τύπος του πίνακα στον οποίο αναφερόμαστε και η πρόταση AGE που βρίσκονται στο WITH clause της ερώτησης.

Επεξεργασία: Ο τελεστής για κάθε νοητό ή υβριδικό πίνακα που συμμετέχει στην ερώτηση με βάση το χρονόσημο που διαθέτει κάθε πλειάδα, ελέγχει αν η ηλικία της κάθε πλειάδας είναι μικρότερη ή μεγαλύτερη από αυτήν που αναφέρεται στο AGE clause της ερώτησης. Στην περίπτωση που ικανοποιείται η συνθήκη του AGE clause, ο peer στον οποίο αναφέρεται η πλειάδα ικανοποιεί τη συνθήκη, ενώ σε αντίθετη περίπτωση όχι.

Έξοδος: Ένα σύνολο από peers, στο οποίο ανήκουν οι peers που δεν ικανοποιούν τη συνθήκη του AGE clause κι έτσι πρέπει να ερωτηθούν ώστε να στείλουν πιο πρόσφατα δεδομένα.

Τελεστές που συμμετέχουν στα πλάνα ερωτήσεων:

(α) CALL_WS:

Ο τελεστής αυτός πραγματοποιεί την διαδικασία κατά την οποία πραγματοποιείται κλήση των κατάλληλων web services κάθε κόμβου που ανήκει στο σύνολο των κόμβων ενδιαφέροντος της ερώτησης. Αντιστοιχεί στο βήμα 3 του αλγορίθμου.

Είσοδος: Ένας κόμβος και μία web service του κόμβου αυτού..

Επεξεργασία: Ο τελεστής πραγματοποιεί κλήση των web services του δεδομένου κόμβου με σκοπό την τροφοδότηση της σχέσης του peer που θέτει την ερώτηση με δεδομένα.

Έξοδος: Τα δεδομένα που επιστρέφονται από την εφαρμογή των κληθεισών web services.

(β) WRAPPER_POP:

Ο τελεστής αυτός χρησιμοποιείται με σκοπό να κατασκευάσει πλειάδες από τα δεδομένα που επιστρέφουν οι διάφορες web services. Αντιστοιχεί στο βήμα 4 του αλγορίθμου που παρουσιάστηκε παραπάνω.

Είσοδος: Τα δεδομένα που επιστρέφονται από την εφαρμογή web services.

Επεξεργασία: Με βάση το σχήμα του πίνακα στόχου κατασκευάζονται πλειάδες έχοντας ως υλικά για την κατασκευή αυτή δεδομένα που επιστρέφονται από την εφαρμογή διάφορων web services. Πραγματοποιείται κατάλληλη επεξεργασία και ενοποίηση των δεδομένων επιστροφής από τις web services με τέτοιον τρόπο ώστε να κατασκευάζονται πλειάδες όπως ακριβώς επιτάσσεται από το σχήμα του πίνακα στόχου.

Έξοδος: Πλειάδες έτοιμες προς εισαγωγή στον πίνακα στόχο.

(γ) FILL:

Ο τελεστής αυτός πραγματοποιεί τη διαδικασία που περιγράφεται στο βήμα 6 του αλγορίθμου και χρησιμοποιείται είτε σε συνεχείς είτε σε ad-hoc ερωτήσεις. Ο τελεστής αυτός γεμίζει τους τοπικούς νοητούς ή υβριδικούς πίνακες με πλειάδες. Επίσης ελέγχει τον χρονισμό της ερώτησης κι ανάλογα με αυτόν σταματάει την διαδικασία τροφοδότησης των πινάκων με πλειάδες όπως επιτάσσεται κάθε φορά.

Είσοδος: Ένα σύνολο από έτοιμες προς εισαγωγή πλειάδες με συγκεκριμένο προορισμό (πίνακα στόχο), η πρόταση TIMING που βρίσκεται στο WITH clause της ερώτησης.

Επεξεργασία: Εισαγωγή των πλειάδων στη σχέση-στόχο με τη σειρά που ετοιμάζονται και έρχονται προς εισαγωγή. Η διαδικασία αυτή σταματάει ανάλογα με τον χρονισμό της ερώτησης. Αν είναι συνεχής η ερώτηση σταματάει να εκτελεί την εργασία του ο τελεστής όταν είτε έχουν απαντήσει όλοι οι peers που ερωτήθηκαν, είτε αν έχει περάσει η περίοδος εκτέλεσης της συνεχούς ερώτησης οπότε τότε θεωρείται ότι πρέπει να σταματήσει να τροφοδοτεί με νέες πλειάδες τους πίνακες στόχους. Στην περίπτωση που η ερώτηση είναι ad-hoc σταματάει να εκτελεί την εργασία του όταν η συνθήκη επιτυχίας της ad-hoc ερώτησης γίνει αληθής. Τότε θεωρείται ότι υπάρχουν αρκετά δεδομένα στους υβριδικούς ή νοητούς πίνακες ώστε να προχωρήσουμε στην εκτέλεση της ερώτησης και να εξάγουμε τα αποτελέσματά της.

Έξοδος: Τους νοητούς ή υβριδικούς πίνακες οι οποίοι είναι γεμάτοι με νέα δεδομένα από τους peers που μας ενδιαφέρουν κι είναι έτοιμοι για περαιτέρω επεξεργασία τοπικά στον peer στου οποίου τη βάση δεδομένων ανήκουν. Επίσης αν ο χρονισμός της ερώτησης είναι συνεχής τότε ως έξοδο έχουμε την ένδειξη YES, ενώ σε αντίθετη περίπτωση το NO.

(δ) ExAg (Execute Again):

Ο τελεστής αυτός είναι χρήσιμος μόνο σε συνεχείς ερωτήσεις και η εργασία που επιτελεί είναι μετά την παραγωγή αποτελέσματος από μία εκτέλεση της συνεχούς ερώτησης να επανεκκινήσει, μετά το πέρας της περιόδου της συνεχούς ερώτησης, την εκτέλεση της ερώτησης από την αρχή.

Είσοδος: Ο χρόνος που πέρασε από τη στιγμή που ξεκίνησε η εκτέλεση της ερώτησης, η περίοδος της συνεχούς ερώτησης και η λέξη YES ή NO που σηματοδοτεί αν πρόκειται για συνεχή ερώτηση ή όχι αντίστοιχα.

Επεξεργασία: Αν η λέξη της εισόδου είναι NO ο τελεστής δεν πραγματοποιεί απολύτως τίποτα. Στην αντίθετη περίπτωση, αν η λέξη της εισόδου είναι YES, η εργασία που επιτελεί ο τελεστής είναι να αναμένει το απαιτούμενο χρονικό διάστημα ώστε να ολοκληρωθεί η περίοδος εκτέλεσης της συνεχούς ερώτησης και μόλις αυτό πραγματοποιηθεί θέτει προς εκτέλεση πάλι την ίδια ερώτηση στον ίδιο peer.

Έξοδος: Η ερώτηση της οποίας η εκτέλεση μόλις ολοκληρώθηκε και επιθυμούμε να εκτελεστεί πάλι από την αρχή.

4.5 Κατασκευή Δέντρου Εκτέλεσης

Στην ενότητα αυτή παρουσιάζεται αναλυτικά τόσο ο αλγόριθμος παραγωγής πλάνου εκτέλεσης ερωτήσεων, όσο κι ο αλγόριθμος εκτέλεσης ερωτήσεων της επεκτεταμένης SQL. Στη συνέχεια πραγματοποιείται περιγραφή των τελεστών που απαιτούνται για την υλοποίηση του αλγορίθμου αυτού, δίνεται ένα παράδειγμα του πλάνου εκτέλεσης μίας ερώτησης και τέλος παρουσιάζεται το μοντέλο κόστους που χρησιμοποιούμε έτσι ώστε να μπορούμε να υπολογίζουμε το εκτιμούμενο κόστος εφαρμογής κάποιου τελεστή ξεχωριστά και ενός πλάνου εκτέλεσης μίας ερώτησης συνολικά.

4.5.1 Αλγόριθμοι Παραγωγής Πλάνου και Εκτέλεσης Ερώτησης

Στον αλγόριθμο που ακολουθεί εξηγούμε τον τρόπο κατασκευής πλάνου εκτέλεσης ερωτήσεων στη γλώσσα SQL^P. Έστω ότι στο σύστημα αναφοράς περιέχονται n peers, οι $peer_1$, $peer_2$, ..., $peer_n$. Υποθέτουμε χωρίς να επηρεάζεται η γενικότητα του αλγορίθμου (προφανώς τα αντίστοιχα ισχύουν αν η ερώτηση γίνεται από οποιονδήποτε άλλο peer) ότι η ερώτηση τίθεται από τον $peer_1$. Επίσης σημειώνουμε ότι όλοι οι peers του συστήματος είναι γνωστοί μεταξύ τους, δηλαδή στον κατάλογο του καθενός από αυτούς περιέχονται τα απαραίτητα στοιχεία για όλους τους peers του συστήματος. Ο αλγόριθμος λοιπόν είναι ο εξής:

Βήμα1: Ανακαλύπτουμε τους νοητούς ή υβριδικούς πίνακες που λαμβάνουν μέρος στην ερώτηση. Για όλους αυτούς θα κατασκευάσουμε ένα υποδέντρο όπως θα εξηγήσουμε στη συνέχεια.

Βήμα 2: Βρίσκουμε το σύνολο κόμβων ενδιαφέροντος. Για κάθε peer που ανήκει στο σύνολο αυτό κατασκευάζουμε έναν κόμβο με το όνομά του. Όλοι οι κόμβοι αυτοί αποτελούν φύλλα στο υποδέντρο που κατασκευάζουμε.

Βήμα 3: Πάνω από κάθε κόμβο που κατασκευάσαμε για κάθε peer που ανήκει στο σύνολο κόμβων ενδιαφέροντος, εισάγουμε έναν κόμβο που αντιστοιχεί στον τελεστή CALL_WS.

Βήμα 4: Πάνω από κάθε κόμβο που αντιστοιχεί στον τελεστή CALL_WS κατασκευάζουμε έναν κόμβο που αντιστοιχεί στον τελεστή WRAPPER_POP.
Βήμα 5: Κατασκευάζουμε έναν κόμβο που αντιστοιχεί στον τελεστή FILL ως ρίζα του υπόδεντρου έχοντας ως κόμβους-παιδιά όλους τους κόμβους που αντιστοιχούν στον WRAPPER_POP τελεστή.
Βήμα 6: Στο σημείο αυτό κατασκευάζουμε το παραδοσιακό αριστεροβαθές πλάνο εκτέλεσης μία ερώτησης στην κλασσική SQL κατά τα γνωστά με μία διαφορά. Σε κάθε κόμβο-πίνακα που έχουμε ανακαλύψει από το βήμα 1 ότι είναι υβριδικός ή νοητός εισάγουμε ως υπόδεντρό του, το υπόδεντρο που έχουμε κατασκευάσει με την εφαρμογή των πρώτων 5 βημάτων του αλγορίθμου.

Στη συνέχεια, με δεδομένο ότι έχουμε κατασκευάσει το πλάνο εκτέλεσης της ερώτησης, παρουσιάζουμε τον αλγόριθμο εκτέλεσης αυτής.

Βήμα 1: Κλήση των κατάλληλων web services των κόμβων ενδιαφέροντος
Έχουμε ήδη ανακαλύψει το σύνολο των κόμβων ενδιαφέροντος της ερώτησης. Σε κάθε έναν από αυτούς τους κόμβους πραγματοποιείται κλήση των κατάλληλων web services που διαθέτουν έτσι ώστε να παραχθούν οι επιθυμητές πλειάδες και να σταλούν στον κόμβο που πραγματοποιεί την κλήση.

Βήμα 2: Επεξεργασία των αποτελεσμάτων των κληθεισών web services
Κατά την εφαρμογή του συγκεκριμένου βήματος παίρνεται η πληροφορία που βγαίνει ως έξοδος από την εφαρμογή των διάφορων web services που καλούνται και υποβάλλεται στη συνέχεια κατάλληλη επεξεργασία ώστε να κατασκευάσουμε πλειάδες οι οποίες είναι έτοιμες προς εισαγωγή στον αντίστοιχο νοητό ή υβριδικό πίνακα. Οι έτοιμες πλειάδες που κατασκευάζονται στο βήμα αυτό αποτελούν την είσοδο για τη διαδικασία που ακολουθεί στο επόμενο βήμα.

Βήμα 3: Γέμισμα των νοητών ή υβριδικών πινάκων
Οι πλειάδες που στο προηγούμενο βήμα ετοιμάζονται για εισαγωγή, στο βήμα αυτό εισάγονται στους πίνακες στόχους. Με αυτόν τον τρόπο τροφοδοτούμε τους πίνακες που ζήτησαν νέες πλειάδες, πράγμα το οποίο γίνεται όσο επιτρέπεται από τη συνθήκη τερματισμού που αναφέρεται στο TIMING clause αν η ερώτηση είναι ad-hoc. Αν η

ερώτηση είναι συνεχής η συνθήκη είναι να μην ξεπεραστεί η χρονική περίοδος της ερώτησης, διότι όταν αυτό γίνει η ερώτηση πρέπει να εκτελεστεί από την αρχή ξανά με τον ίδιο τρόπο. Κατά το βήμα αυτό λοιπόν τροφοδοτούνται με πλειάδες οι νοητοί ή υβριδικοί πίνακες και είναι όλα έτοιμα για το επόμενο που είναι και το τελευταίο βήμα του αλγορίθμου.

Βήμα 4: Εκτέλεση του υπόλοιπου της ερώτησης τοπικά, κατά τα γνωστά
Στο βήμα αυτό όλοι οι πίνακες είναι γεμάτοι με τις πλειάδες που μας ενδιαφέρουν και επιθυμούμε, οπότε πραγματοποιείται η εκτέλεση της ερώτησης στον peer που την έθεσε κατά τα γνωστά και παίρνουμε τα αντίστοιχα αποτελέσματα.

4.5.2 Υλοποίηση των Επεκτάσεων

Στην ενότητα αυτή αναλύουμε τον τρόπο με τον οποίο υλοποιούνται οι επεκτάσεις της SQL που εισάγουμε. Θα γίνει ανάλυση κάθε μίας πρότασης του WITH clause.

AGE: Κάθε πλειάδα μίας νοητής ή υβριδικής σχέσης χαρακτηρίζεται από ένα χρονόσημο. Το χρονόσημο καθορίζει πότε εισήχθη η πλειάδα αυτή στη σχέση. Υπάρχει αντίστοιχο χρονόσημο για όλες τις πλειάδες της σχέσης. Με τη βοήθεια αυτών των χρονόσημων εύκολα όταν πραγματοποιείται μία ερώτηση, η οποία περιέχει κάποια νοητή ή υβριδική σχέση, μπορούμε να υπολογίσουμε για κάθε πλειάδα πόσος χρόνος πέρασε από τη στιγμή που εισάχθηκε η πλειάδα αυτή στη σχέση. Συγκρίνουμε αυτό το χρονικό διάστημα με αυτό που αναγράφεται στην ερώτηση στην πρόταση AGE και πράττουμε αναλόγως με βάση το αν είναι μεγαλύτερο ή μικρότερο. Στην περίπτωση που είναι μεγαλύτερο θεωρούμε την πλειάδα αυτή παλιά, οπότε ζητάμε από τον peer στον οποίο αναφέρεται να τροφοδοτήσει τη σχέση με νέα δεδομένα. Στην αντίθετη περίπτωση θεωρούμε τα περιεχόμενα της δεδομένης πλειάδας πρόσφατα, οπότε δεν χρειάζεται να ζητηθεί από τον peer στον οποίο αναφέρεται η πλειάδα να τροφοδοτήσει τη σχέση με νέα δεδομένα.

TIMING: Ο χρονισμός της ερώτησης διακρίνεται σε AD-HOC και σε συνεχή (CONTINUOUS). Στην περίπτωση της AD-HOC ερώτησης πρέπει να καθοριστεί το

κριτήριο τερματισμού της ερώτησης. Τα πιθανά κριτήρια είναι: να πάρουμε απάντηση από κάποιο ποσοστό των peers που ρωτάμε (PEERS_PERCENTAGE), να συμπληρώσουμε ένα πλήθος από πλειάδες (TUPLES_AMOUNT) ή να λήξει ένα χρονικό όριο που θέτουμε (TIMEOUT). Ακολουθεί ανάλυση για κάθε ένα από αυτά τα κριτήρια τα οποία τίθενται σε AD-HOC ερωτήσεις.

AD-HOC:

PEERS_PERCENTAGE: Ο αλγόριθμος που ακολουθείται στην περίπτωση που έχουμε αυτό το κριτήριο τερματισμού είναι ο εξής:

(1) Με τη βοήθεια του καταλόγου του peer που θέτει την ερώτηση κατασκευάζεται το σύνολο κόμβων ενδιαφέροντος.

(2) Πραγματοποιείται υπολογισμός του ποσοστού των peers που έχουν τροφοδοτήσει με τις πλειάδες τους τη σχέση στόχο σε σχέση με τους peers που ανήκουν στο σύνολο κόμβων ενδιαφέροντος.

(3) Έλεγχος αν το ποσοστό που μόλις υπολογίστηκε είναι μεγαλύτερο από αυτό που αναγράφεται στην ερώτηση μετά την έκφραση PEERS_PERCENTAGE.

(i) Αν ναι, η συνθήκη τερματισμού ισχύει και η ερώτηση εκτελείται ως έχουμε.

(ii) Αν όχι, συνεχίζουμε στο βήμα 4.

(4) Κάποιος peer στέλνει τις πλειάδες του.

(5) Πάμε στο βήμα 2.

TUPLES_AMOUNT: Ο αλγόριθμος που ακολουθείται στην περίπτωση που έχουμε αυτό το κριτήριο τερματισμού είναι ο εξής:

(1) Υπολογισμός του πλήθους των πλειάδων που υπάρχουν στη σχέση.

(2) Έλεγχος αν το πλήθος που μόλις υπολογίστηκε είναι μεγαλύτερο από αυτό που αναγράφεται στην ερώτηση μετά την έκφραση TUPLES_AMOUNT.

(i) Αν ναι, η συνθήκη τερματισμού ισχύει και η ερώτηση εκτελείται ως έχουμε.

(ii) Αν όχι, συνεχίζουμε στο βήμα 3.

(3) Κάποιος peer στέλνει τις πλειάδες του.

(4) Πάμε στο βήμα 1.

TIMEOUT: Ο αλγόριθμος που ακολουθείται στην περίπτωση που έχουμε αυτό το κριτήριο τερματισμού είναι ο εξής:

(1) Όταν πραγματοποιείται η ερώτηση τίθεται ένας timer=0 και στη συνέχεια μετράει το χρόνο που περνάει από αυτήν τη στιγμή.
(2) Έρχονται πλειάδες μέχρι να πάρει ο timer τιμή μεγαλύτερη από αυτήν που αναγράφεται στην ερώτηση μετά την έκφραση TIMEOUT.
(3) Εκτέλεση της ερώτησης ως έχουμε.

CONTINUOUS: Στην περίπτωση των συνεχών ερωτήσεων πρέπει να καθορίσουμε την περίοδο εκτέλεσης αυτών των ερωτήσεων. Ο αλγόριθμος που ακολουθείται είναι ο εξής:
(1) Τίθεται ένας timer=0 και στη συνέχεια μετράει το χρόνο που περνάει από αυτήν τη στιγμή.
(2) Πραγματοποιείται η ερώτηση όπως έχουμε περιγράψει και δίνει τα αντίστοιχα αποτελέσματα.
(3) Μόλις ο timer γίνει ίσως με την περίοδο που θέσαμε στην ερώτηση τότε πάμε στο βήμα 1.

HORIZON: Στην πρόταση αυτή καθορίζουμε ποιοι peers μας ενδιαφέρουν. Οι πιθανές εκφράσεις είναι μόνο τοπικά (LOCAL) ή κάποια κοινότητα του peer που ρωτάει (COMMUNITY <C_NAME>) ή κάποιοι συγκεκριμένοι peers ή peers που απέχουν κάποιο πλήθος βημάτων από τον peers που θέτει την ερώτηση. Ακολουθεί ανάλυση για κάθε μία από τις παραπάνω περιπτώσεις.

LOCAL: Στην περίπτωση αυτή η ερώτηση εκτελείται τοπικά, χωρίς να μας αφορούν οι υπόλοιποι peers που γνωρίζει ο ερωτών peer.

COMMUNITY C_NAME: Στα μηνύματα ύπαρξης που ανταλλάσσονται μεταξύ των peers δίνεται και μία επιπλέον πληροφορία. Ο κάθε peer κάνει γνωστό προς τους υπόλοιπους κάποια σημασιολογικά χαρακτηριστικά του. Με βάση αυτή την πληροφορία ο κάθε peer ταξινομείται σε κάποια κοινότητα ή κάποιες κοινότητες των peers που λαμβάνουν αυτήν την πληροφορία. Οπότε ο peer που θέτει την ερώτηση από τον κατάλογό του είναι σε θέση να γνωρίζει ποιοι peers ανήκουν σε κάποια κοινότητά του και να θέσει την ερώτηση σε αυτούς.

Συγκεκριμένους peers: Όλοι οι peers χαρακτηρίζονται από έναν μοναδικό αριθμό έτσι ώστε να διακρίνονται ο ένας από τον άλλο. Επίσης στα μηνύματα που στέλνουν μεταξύ τους εκτός των άλλων περιέχεται και ο αριθμός που τους χαρακτηρίζει. Έτσι ο peer που θέτει την ερώτηση μπορεί να απευθυνθεί μόνο σε κάποιους συγκεκριμένους peers που αυτός επιθυμεί μέσω του αριθμού που αντιστοιχεί σε αυτούς.

Απόσταση κάποιου πλήθους βημάτων: Ο κάθε peer έχει στον κατάλογό του τις αποστάσεις σε βήματα από κάθε γνωστό του peer. Αυτό το επιτυγχάνει ως εξής:
Η ίδια διαδικασία ακολουθείται για κάθε peer
(1) Πηγαίνω στους άμεσους γείτονές μου (1 βήμα)
(2) Από αυτούς πηγαίνω στους άμεσους γείτονές τους (άλλο ένα βήμα)
Συνεχίζω με τον ίδιο τρόπο μέχρι να εξαντληθούν όλοι οι peers, δηλαδή μέχρι να επισκεφτώ όλους τους γνωστούς peers που υπάρχουν στον κατάλογο.
Αφού τελειώσει η παραπάνω διαδικασία μετράμε τα βήματα που απέχει ο κάθε peer, λαμβάνοντας υπόψη πάντα το ελάχιστο μονοπάτι.

Με βάση αυτήν την πληροφορία που περιέχεται στον κατάλογό του έχει τη δυνατότητα ο peer που θέτει την ερώτηση να απευθυνθεί στους peers οι οποίοι απέχουν από αυτόν κάποιο πλήθος βημάτων.

AVAILABILITY: Ένα άλλο επιπρόσθετο στοιχείο που κάθε peer δημοσιοποιεί προς τους άλλους είναι η διαθεσιμότητά του. Κάθε peer με χρήση ενός timer μετράει πόσο χρόνο είναι ενεργός-διαθέσιμος σε σχέση με το συνολικό χρόνο λειτουργίας του. Η διαθεσιμότητά του ορίζεται σαν το πηλίκο του χρόνου που είναι ενεργός-διαθέσιμος προς τον συνολικό χρόνο λειτουργίας του. Ο κάθε peer ενημερώνει τους άλλους peers για αυτό το χαρακτηριστικό του, οπότε στον κατάλογο του κάθε peer περιέχεται και μία στήλη με τη διαθεσιμότητα όλων των γνωστών σε αυτόν peers. Επίσης, στη διάρκεια του χρόνου αυτό το χαρακτηριστικό μπορεί να μεταβάλλεται, έτσι για μεταβολές μεγαλύτερες από ένα όριο, έστω t_a, ο peer στον οποίο συναντώνται αυτές οι μεταβολές οφείλει να ενημερώσει τους άλλους peers για τη μεταβολή αυτή. Με αυτόν τον τρόπο πάντα η διαθεσιμότητα του κάθε peer ανταποκρίνεται πολύ κοντά στην πραγματικότητα (AVAILABILITY-t_a < πραγματική τιμή < AVAILABILITY+t_a) κι από την άλλη αποφεύγονται τα πολλά μηνύματα μεταξύ των

peers που θα οδηγούσαν σε μεγαλύτερο κόστος επικοινωνίας μεταξύ των peers (δεν αποτελεί ιδιαίτερα σύνηθες το φαινόμενο της μεταβολής της διαθεσιμότητας ενός peer κατά t_a ή περισσότερο).

RESPONSE_TIME: Ένα άλλο χαρακτηριστικό του κάθε peer είναι ο αναμενόμενος χρόνος απόκρισης σε κάποια ερώτηση που θέτει κάποιος peer. Στον κατάλογο του κάθε peer υπάρχει μία στήλη η οποία περιέχει την πληροφορία αυτή, τον αναμενόμενο χρόνο απόκρισης όλων των γνωστών peers. Ο χρόνος αυτός για κάθε peer που περιέχεται στον κατάλογο του peer αναφοράς υπολογίζεται ως το γινόμενο του πλήθους των βημάτων της απόστασής του από τον peer αναφοράς με το βάρος του. Το πλήθος των βημάτων της απόστασης υπάρχει στον κατάλογο του peer και υπολογίζεται όπως αναλύθηκε παραπάνω. Το βάρος του peer είναι μία ποσότητα η οποία χαρακτηρίζει την ποιότητα της επικοινωνίας ενός peer και δημοσιοποιείται στους άλλους peers μέσω των μηνυμάτων ύπαρξης που ανταλλάσσονται μεταξύ τους. Οπότε εύκολα ο κάθε peer έχει τη δυνατότητα, αφού του παρέχονται και οι δύο ποσότητες, να υπολογίσει τον αναμενόμενο χρόνο απόκρισης όλων των γνωστών του peers σε τυχούσα ερώτηση που μπορεί να τεθεί από αυτόν και να τους αφορά.

CLASS: Κάθε peer ανήκει σε μία κλάση από peers με βάση την κλάση στην οποία ανήκουν οι web services που παρέχει. Την πληροφορία αυτή την δημοσιοποιεί κάθε peer στα μηνύματα ύπαρξης που ανταλλάσσει με τους άλλους peers. Η πληροφορία αυτή αποθηκεύεται στον κατάλογο των peers, έτσι ο κάθε peer γνωρίζει την κλάση στην οποία ανήκουν όλοι οι γνωστοί του peers. Με βάση αυτήν την πληροφορία δίνεται η δυνατότητα σε κάποιον peer να θέσει ερώτηση η οποία να αφορά τους peers μίας συγκεκριμένης κλάσης που επιθυμεί αυτός.

4.5.3 Παράδειγμα Πλάνου Εκτέλεσης Ερώτησης

Με σκοπό να δώσουμε ένα παράδειγμα πλάνου εκτέλεσης μίας ερώτησης σύμφωνα με τον αλγόριθμο παραγωγής πλάνου εκτέλεσης ερώτησης της ενότητας 4.5.1 και με τη βοήθεια των τελεστών που εισάγαμε στην ενότητα 4.4.2, θεωρούμε τον γράφο των peers που φαίνεται στο σχήμα 4.17. Η ερώτηση τίθεται στον peer 1 και δίνεται στο σχήμα 4.18, ενώ ο κατάλογός του παρουσιάζεται στον πίνακα 4.3.

Εφαρμογή των βημάτων του αλγορίθμου αναλυτικά:

Βήμα 1: Στην ερώτηση στο FROM clause έχουμε δύο πίνακες, τον CARS και τον BRANDS. Η εφαρμογή του τελεστή CHECK_TABLES πάνω σε αυτούς έχει ως αποτέλεσμα ότι ο πρώτος είναι υβριδικός πίνακας ενώ ο δεύτερος είναι τοπικά αποθηκευμένος.

Βήμα 2: Εφαρμόζεται ο τελεστής CHECK_PEERS ο οποίος έχει ως αποτέλεσμα το σύνολο κόμβων ενδιαφέροντος της ερώτησης. Με τη βοήθεια του κατάλογου που διαθέτει ο peer 1 και λαμβάνοντας υπόψη την ηλικία των πλειάδων που βρίσκονται στον πίνακα CARS, οι κόμβοι που προκύπτει να αποτελούν το σύνολο κόμβων ενδιαφέροντος της δεδομένης ερώτησης είναι οι 1, 2, 3 και 7, οι οποίοι ικανοποιούν τις συνθήκες της ερώτησης και είναι μαρκαρισμένοι με έντονα γράμματα στον κατάλογο του peer 1.

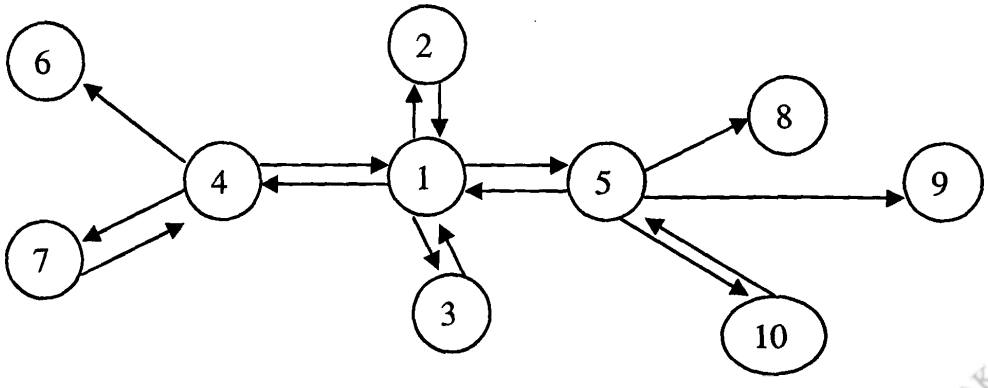
Βήμα 3: Εφαρμογή του τελεστή CALL_WS πάνω σε κάθε peer που ανήκει στο σύνολο κόμβων ενδιαφέροντος.

Βήμα 4: Εφαρμογή του τελεστή WRAPPER_POP πάνω από κάθε τελεστή CALL_WS.

Βήμα 5: Εφαρμογή του τελεστή FILL πάνω από κάθε τελεστή WRAPPER_POP.

Βήμα 6: Κατασκευή του πλάνου της ερώτησης με τον παραδοσιακό τρόπο, με τη διαφορά ότι προσθέτουμε ως υπόδεντρο του πίνακα CARS (υβριδικός) όλο το υπόδεντρο που κατασκευάζεται στα 5 προηγούμενα βήματα.

Ακολουθώντας τα βήματα αυτά, το πλάνο εκτέλεσης της ερώτησης του σχήματος 4.18 που προκύπτει, παρουσιάζεται στο σχήμα 4.19.



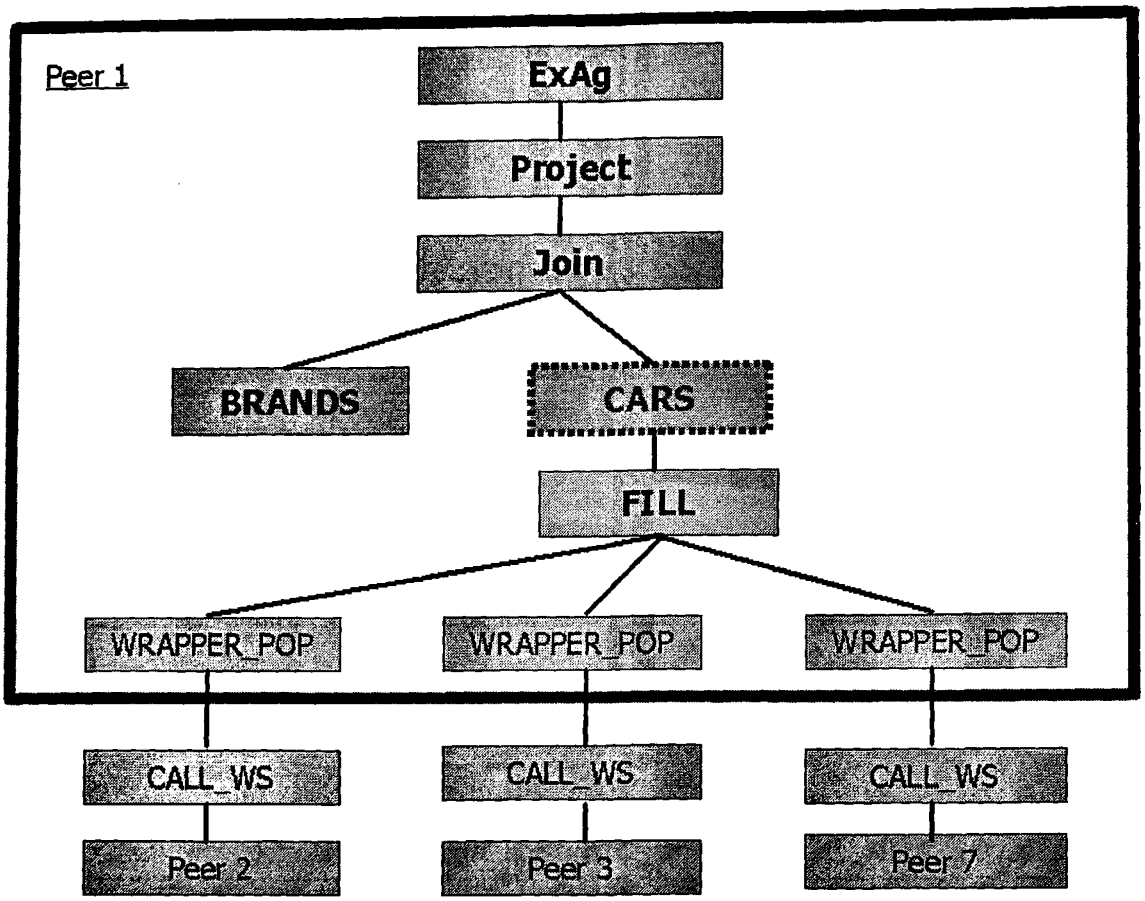
Σχήμα 4.17: Ο Γράφος των Peers του Συστήματος

```
SELECT CARS.PLATE,CARS.VEL,BRANDS.COUNTRY
FROM   CARS,BRANDS
WHERE  CARS.BRAND=BRANDS.BRAND
WITH
AGE < 5 AND
HORIZON COMMUNITY DIS<5KM AND
TIMING CONTINUOUS PULL_BASED_PERIOD = 7 AND
AVAILABILITY > 60% AND
RESPONSE_TIME < 3.0 AND
CLASS = "european"
```

Σχήμα 4.18: Η ερώτηση που Τίθεται από τον Peer 1 του Συστήματος

Πίνακας 4.3: Ο Κατάλογος του Peer

Peer ID	Κοινότητα που ανήκει	Απόσταση σε βήματα	Διαθεσιμότητα	Χρόνος απόκρισης	Κλάση web services
1	**DIS<5KM**	**0**	**80%**	**0**	**European**
2	**DIS<5KM**	**1**	**75%**	**2.3**	**European**
3	**DIS<5KM**	**1**	**63%**	**2.7**	**European**
4	DIS<5KM	1	97%	3.9	American
5	DIS>5KM	1	82%	4.1	European
6	DIS<5KM	2	77%	2.5	American
7	**DIS<5KM**	**2**	**67%**	**2.9**	**European**
8	DIS>5KM	2	92%	3.3	American
9	DIS>5KM	2	89%	4.3	European
10	DIS>5KM	2	78%	4.6	European



Σχήμα 4.19: Πλάνο Εκτέλεσης της Ερώτησης του Σχήματος 4.18

4.6 Εναλλακτικός Τρόπος Εκτέλεσης Ερώτησης

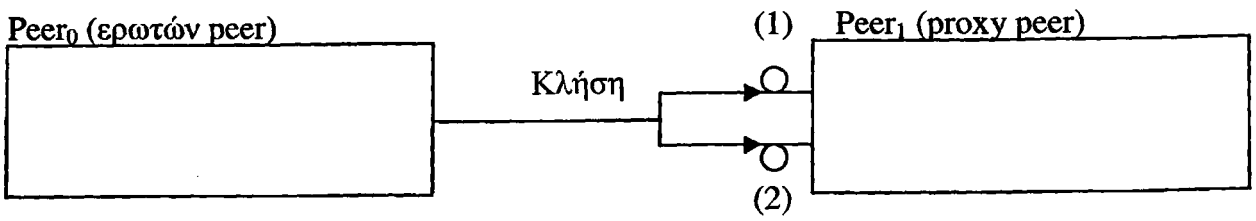
Σύμφωνα με τα μέχρι τώρα λεγόμενα ο peer ο οποίος θέτει την ερώτηση είναι σε θέση ή καλύτερα, υποχρεούται να είναι σε θέση να απαντήσει μόνος του. Με την έννοια είναι σε θέση να απαντήσει μόνος του εννοούμε ότι δεν ζητάει την συνδρομή κανενός άλλου peer στη διαδικασία απάντησης στην ερώτηση. Έτσι είναι πιθανό να γνωρίζει ένα υποσύνολο του συνολικού γράφου των peers και να υποχρεούται να απαντήσει βάσει αυτού. Για να αποφεύγονται περιπτώσεις κατά τις οποίες η απάντηση σε μία ερώτηση είναι ελλιπής κι έτσι μη απόλυτα ορθή, υπάρχει η εναλλακτική να οδηγηθεί η εκτέλεση της ερώτησης μέσω άλλων peers. Αυτό μπορεί να συμβεί όταν ο ερωτών peer έχει περίοδο ανανέωσης του καταλόγου του πιο αργή από τους άλλους peers ή στην ακραία περίπτωση που η περίοδος αυτή είναι άπειρη, οπότε η ανανέωση του καταλόγου του πραγματοποιείται on-demand.

Ένας peer λοιπόν, μπορεί να ζητήσει την συνδρομή κάποιου άλλου peer σε μία ερώτηση,
(α) αν του το ζητήσει ο χρήστης στην ερώτηση,
(β) αν ο ίδιος γνωρίζει πολύ λίγους peer (π.χ αν διαπιστώσει ότι οι κόμβοι που γνωρίζει είναι μόνο ένα βήμα-hop μακριά) και
(γ) για πολύ μακρινούς από αυτόν peers.

Υποθέτουμε ένα μοντέλο συνεργασίας στο οποίο ένας peer έχει τρόπο να ζητήσει από κάποιον άλλον peer να κατασκευάσει πλάνο εκτέλεσης για μία ερώτηση ως αντιπρόσωπός του και να του επιτρέπει στη συνέχεια τη μεταβίβαση αυτού του πλάνου. Επίσης θεωρούμε μία συνάρτηση f η οποία αποφαίνεται αν τμήμα της ερώτησης θα μεταβιβαστεί σε κάποιον άλλο peer και σε ποιον.

Έστω ότι τίθεται μία ερώτηση Q πάνω στον p_0. Όπως έχουμε αναλύσει παραπάνω, ο p_0 εφαρμόζει τους τελεστές CHECK_PEERS και CHECK_AGE έτσι ώστε να ανακαλύψει το σύνολο κόμβων ενδιαφέροντος με βάση τον κατάλογο που είναι αποθηκευμένος σε αυτόν. Σύμφωνα με την τεχνική που εισάγουμε στην ενότητα αυτή, η εύρεση του συνόλου κόμβων ενδιαφέροντος με βάση τον τοπικά αποθηκευμένο κατάλογο του p_0 ενεργοποιεί
(α) είτε ένα τελεστή που ζητάει από τους peers που είναι σε απόσταση ενός βήματος από αυτόν να του δώσουν το δικό τους σύνολο κόμβων ενδιαφέροντος
(β) είτε έναν άλλο τελεστή με τον οποίο ζητείται από κάποιον peer να γίνει ο αντιπρόσωπος του p_0.

Για να πραγματοποιηθεί αυτό, όπως φαίνεται και στο σχήμα 4.20, κάθε peer έχει δύο web services, την proxy_give_peers() και την proxy_execute_query(). Η πρώτη καλείται όταν ζητείται από τον δεδομένο κόμβο να ανακαλύψει και να δώσει το δικό του σύνολο κόμβων ενδιαφέροντος, ενώ η δεύτερη όταν ζητείται να γίνει ο αντιπρόσωπός του για την δεδομένη ερώτηση.



(1) proxy_give_peers web service

(2) proxy_execute_query web service

Σχήμα 4.20: Κλήση της Κατάλληλης Web Service

Για να πραγματοποιηθεί η διαδικασία που αναλύουμε στην ενότητα αυτή είναι απαραίτητοι κάποιοι τελεστές. Αυτοί είναι οι εξής:

(α) Ένας τελεστής που πραγματοποιεί την ενέργεια της αίτησης προς τους γύρω peers να στείλουν το δικό τους σύνολο κόμβων ενδιαφέροντος (Give_Set_of_Interesting_Peers).

(β) Ένας τελεστής που πραγματοποιεί την ενέργεια αίτησης προς έναν κόμβο να εκτελέσει μία ερώτηση, ως αντιπρόσωπος του κόμβου στον οποίο αρχικά είχε τεθεί αυτή (Execute_Query_as_Proxy).

(γ) Ένας τελεστής ο οποίος θα εκτελείται στο μέρος του proxy και θα έχει ως αποστολή να κατασκευάζει πλάνα της ερώτησης αλλά όχι για την τοπική του βάση δεδομένων (Construct_Remote_Plan).

Ακολουθεί ο ορισμός αυτών των τελεστών:

(α) Give_Set_of_Interesting_Peers:

Η εφαρμογή του εν λόγω τελεστή αποσκοπεί να μαζέψει από όλους τους άμεσους γείτονές του όλους τους peers οι οποίοι είναι γνωστοί σε αυτούς.

Είσοδος: Ο τοπικός κατάλογος του peer στον οποίο τίθεται η ερώτηση.

Επεξεργασία: Από τον κατάλογό του εύρεση όλων των γνωστών του κόμβων που απέχουν από αυτόν ένα βήμα (1 hop).

Έξοδος: Ένα μήνυμα για κάθε κόμβο από αυτούς που προκύπτουν από τη φάση της επεξεργασίας, το οποίο θα περιέχει την ερώτηση που τίθεται.

(β) Execute_Query_as_Proxy:

Η εφαρμογή του εν λόγω τελεστή αποσκοπεί στο να κάνει κάποιον peer αντιπρόσωπο αυτού στον οποίο έχει τεθεί η ερώτηση αρχικά.

Είσοδος: Ο τοπικός κατάλογος του peer στον οποίο τίθεται η ερώτηση.

Επεξεργασία: Από τον κατάλογό του εύρεση όλων των γνωστών του κόμβων και επιλογή αυτού που θεωρείται ότι μπορεί να παίξει το ρόλο του αντιπρόσωπού του καλύτερα.

Έξοδος: Ένα μήνυμα προς τον κόμβο που προκρίνεται από τη φάση της επεξεργασίας, το οποίο θα περιέχει την ερώτηση που τίθεται.

(γ) Construct_Remote_Plan:

Η εφαρμογή του εν λόγω τελεστή αποσκοπεί να κατασκευάσει πλάνο εκτέλεσης της ερώτησης για λογαριασμό του peer στον οποίο τέθηκε η ερώτηση αρχικά.

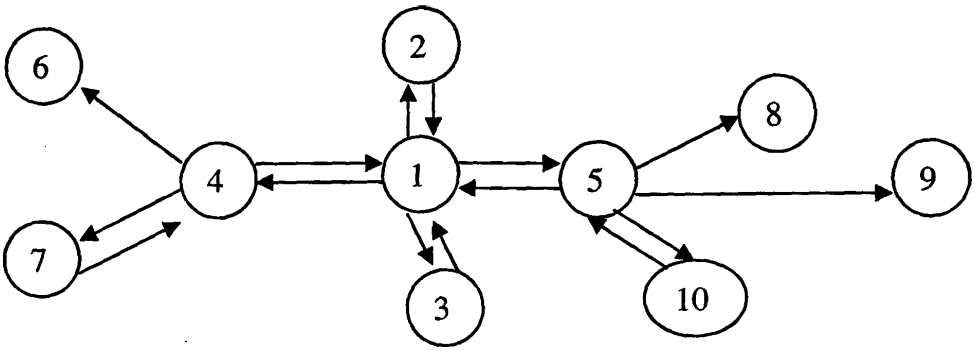
Είσοδος: Η ερώτηση και το σχήμα των υβριδικών ή νοητών πινάκων της βάσης δεδομένων του peer στον οποίο τέθηκε αρχικά η ερώτηση.

Επεξεργασία: Κατασκευή πλάνου εκτέλεσης της ερώτησης αυτής με βάση το σχήμα των πινάκων της εισόδου κι όχι της τοπικής του βάσης δεδομένων.

Έξοδος: Το πλάνο που κατασκευάστηκε στη φάση της επεξεργασίας.

Στο σημείο αυτό αναδύεται ένα θέμα κι είναι αυτό της αντιστοίχησης σχήματος, δηλαδή τι σχήμα θα έχουν οι πλειάδες τις οποία θα στείλει πίσω ο p_1. Λύση στο ζήτημα αυτό μπορεί να δοθεί αν για κάθε πλειάδα του αποτελέσματος επιστρέφεται και η κλάση που ανήκει ο αντίστοιχος peer που την παρήγαγε. Τότε μπορούμε να βρούμε το workflow το οποίο την μετατρέπει σύμφωνα με το σχήμα που είναι επιθυμητό από τον p_0.

Ακολουθεί ένα παράδειγμα για να δούμε πρακτικά τους δύο τρόπους με τους οποίους μπορεί να εκτελεστεί εναλλακτικά μία ερώτηση. Θεωρούμε το γράφο του σχήματος 4.21 και τους καταλόγους των peers που φαίνονται στο σχήμα 4.22. Επίσης θεωρούμε ότι η ερώτηση τίθεται στον peer 1 και στην ερώτηση συμμετέχει ένας υβριδικός πίνακας αυτού ο CARS.



Σχήμα 4.21: Ο Γράφος των Peers του Παραδείγματος Αναφοράς

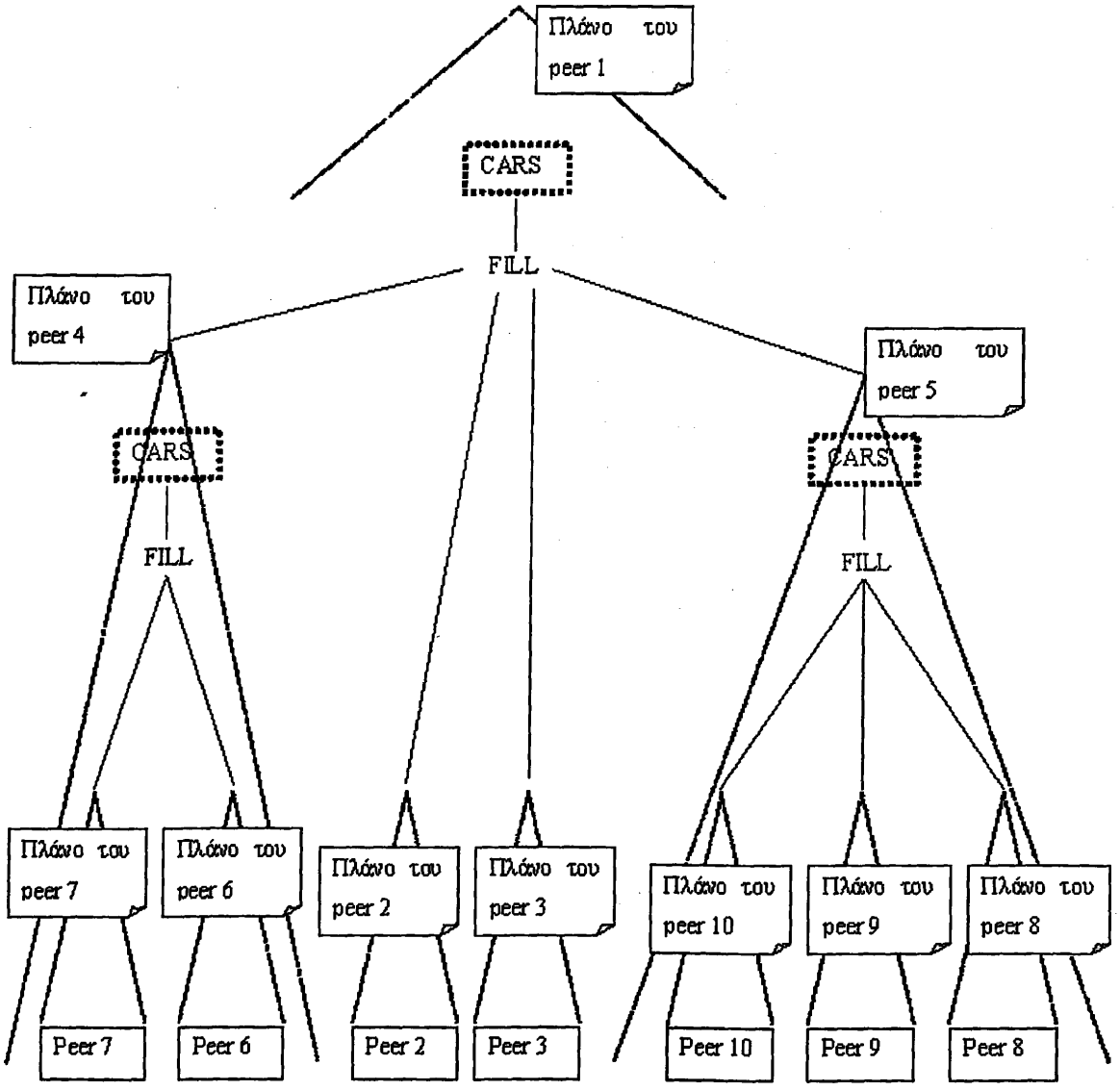
Κατάλογος του peer 1	**Κατάλογος του peer 4**	**Κατάλογος του peer 5**
Γνωστοί κόμβοι:	Γνωστοί κόμβοι:	Γνωστοί κόμβοι:
2	1	1
3	6	8
4	7	9
5		10

Σχήμα 4.22: Οι Κατάλογοι των Peers 1, 4 και 5

Σύμφωνα με τον πρώτο τρόπο εναλλακτικής εκτέλεσης ερώτησης ο peer 1 καλεί την proxy_give_peers() web service όλων των γειτόνων του. Χωρίς να χάνεται η γενικότητα θεωρούμε ότι οι peers 2 και 3 γνωρίζουν μόνο τον peer 1. Η κλήση αυτής της web service των γειτόνων του έχει ως αποτέλεσμα όλοι αυτοί να του στείλουν τον κατάλογό τους. Από τον κατάλογο των peers 2 και 3, σύμφωνα με την παραδοχή που έχουμε κάνει δεν γνωρίζει κάποιον peer άλλον από αυτούς που γνωρίζει ήδη. Όμως από τους peers 4 και 5 σύμφωνα με το σχήμα 4.22 μαθαίνει την ύπαρξη των peers 6, 7 και 8, 9, 10 αντίστοιχα. Προσθέτει αυτούς τους peers στον δικό του κατάλογο και εκτελεί την ερώτηση λαμβάνοντας υπόψη την ύπαρξη όλων αυτών των peers.

Σύμφωνα με τον δεύτερο τρόπο εναλλακτικής εκτέλεσης ερώτησης ο peer 1 καλεί την proxy_execute_query() web service κάποιων peers που επιθυμεί να εκτελέσουν την ερώτησή του σαν αντιπρόσωποί του. Θεωρούμε ότι ο peer 1 επιλέγει τους peers 4 και

5 σαν αντιπροσώπους του. Έτσι αυτοί οφείλουν να κατασκευάσουν πλάνα εκτέλεσης της ερώτησης που έχει τεθεί στον peer 1 σαν να είναι δική τους με τη μόνη διαφορά ότι δεν αναφέρεται στην τοπική τους βάση. Για αυτό το λόγο ο peer 1 τους στέλνει μία λίστα από web services για κάθε κλάση peers, έτσι ώστε να μπορούν οι peers αντιπρόσωποι να εκτελέσουν την εργασία τους σωστά. Οπότε στο γενικό πλάνο της ερώτησης που φαίνεται στο σχήμα 4.23 υπάρχουν 4 υποπλάνα, ένα για κάθε peer που γνωρίζει ο peer 1 (τους peers 2 και 3) και ένα για κάθε αντιπρόσωπο που έχει θέσει (τους peers 4 και 5).



Σχήμα 4.23: Πλάνο Εκτέλεσης της Ερώτησης με τον Εναλλακτικό Τρόπο

ΚΕΦΑΛΑΙΟ 5. ΜΟΝΤΕΛΟ ΚΟΣΤΟΥΣ

Στο κεφάλαιο αυτό θα παρουσιαστεί ένα μοντέλο κόστους για όλους τους τελεστές που μπορεί να εφαρμοστούν κατά την εκτέλεση κάποιας ερώτησης στο σύστημά μας. Οι τελεστές αυτοί χωρίζονται σε τρεις κατηγορίες, έτσι στην ενότητα 5.1 θα δώσουμε το κόστος εφαρμογής των τελεστών της παραδοσιακής άλγεβρας, στην ενότητα 5.2 για αυτούς που εφαρμόζονται στον κατάλογο του ερωτώντα peer και στην ενότητα 5.3 δίνεται το κόστος εφαρμογής των τελεστών που εισάγαμε στο προηγούμενο κεφάλαιο και λαμβάνουν μέρος σε πλάνα εκτέλεσης ερωτήσεων. Με τη βοήθεια αυτού του μοντέλου κόστους έχουμε τη δυνατότητα να συγκρίνουμε τα κόστη από εναλλακτικά πλάνα εκτέλεσης της ίδιας ερώτησης και να επιλέγουμε αυτό που θα έχει το μικρότερο κόστος για την εκτέλεσή της (βελτιστοποίηση ερωτήσεων). Στη συνέχεια παρουσιάζουμε έναν πίνακα με την απαραίτητη επεξήγηση συμβόλων που θα χρησιμοποιήσουμε για να καθορίσουμε τα κόστη των τελεστών αυτών.

5.1 Τελεστές Παραδοσιακής Σχεσιακής Άλγεβρας

Οι τελεστές της παραδοσιακής σχεσιακής άλγεβρας οι οποίοι λαμβάνουν μέρος σε πλάνα ερωτήσεων του συστήματός μας με τα αντίστοιχα κόστη τους παρουσιάζονται αμέσως παρακάτω. Σημειώνουμε ότι τα κόστη εφαρμογής αυτών είναι σε σελίδες μεταφοράς από το δίσκο στη μνήμη.

- Project: Το κόστος εφαρμογής αυτού του τελεστή υπολογίζεται με βάση τα blocks που μεταφέρονται μεταξύ δίσκου και μνήμης. Το κόστος του τελεστή είναι ίσο με το πλήθος των blocks του αρχείου, διότι πρέπει να μεταφέρουμε

όλον τον πίνακα στη μνήμη έτσι ώστε να προβάλλουμε τα κατάλληλα γνωρίσματά του.

Οπότε *Cost(Project)=sizeof(R).*

Πίνακας 5.1: Επεξήγηση Συμβόλων του Μοντέλου Κόστους

Παράμετρος	**Σύμβολο**
Σχέσεις	*R,S*
Μέγεθος πίνακα σε σελίδες	*sizeof)R)*
Μέγεθος πίνακα σε πλειάδες	*tuples(R)*
Πλήθος πλειάδων ανά σελίδα	*tuples_per_page*
Μέγεθος buffers	*B*
Ευρετήριο	*I*
Επιλεκτικότητα (selectivity)	*s*
Πλήθος επιπέδων ευρετηρίου	*H(I)*
Πλήθος blocks του πρώτου επιπέδου	*size(I_top)*
Παράγοντας κλιμάκωσης μεταξύ μνήμης και I/O	*W(m-I/O)*
Web service λειτουργία (operation)	*O*
Μέγεθος αποτελέσματος της O	*tuples(O)*
Κόστος κατασκευής μηνύματος SOAP	*cost_con(S_msg)*
Κόστος αποστολής μηνύματος SOAP	*cost_send(S_msg)*
Αναμονή απάντησης	*time_wai(O)t*
Κόστος για parsing	*cost_parse(S_msg)*
Workflow	*wf*
Αποτέλεσμα workflow σε πλειάδες	*tuples(wf)*
Είσοδος σε workflow	*input_tuples(wf)*
Το κόστος όλου του workflow	*cost(wf)*
Το κόστος του workflow ανά πλειάδα	*cost_per_tuple(wf)*

- Filter: α) Μπορούμε να χρησιμοποιήσουμε τη μέθοδο της σειριακής αναζήτησης, δηλαδή αναζήτηση σε όλα τα block του αρχείου για να ανακτηθούν οι εγγραφές που ικανοποιούν τη συνθήκη επιλογής. Άρα *Cost(Filter)=sizeof(R).*

  Για συνθήκη ισότητας πάνω σε ένα κλειδί, το μισό, κατά μέσον όρο, του αρχείου πρέπει να διαβαστεί μέχρις ότου βρεθεί η εγγραφή, οπότε *Cost(Filter)= sizeof(R)/2.* Αν δε βρεθεί εγγραφή που να ικανοποιεί τη συνθήκη τότε, *Cost(Filter)= sizeof(R).*

  β) Σε περίπτωση που η σχέση είναι αποθηκευμένη σε αρχείο ταξινομημένο με βάση το πεδίο της συνθήκης επιλογής, μπορούμε να χρησιμοποιήσουμε

δυαδική αναζήτηση. Η αναζήτηση αυτή προσπελαύνει προσεγγιστικά $log_2 (sizeof(R))+s/B-1$ blocks αρχείου. Το πλήθος αυτό μειώνεται σε $log_2 sizeof(R)$ αν η συνθήκη ισότητας αναφέρεται σε μοναδικό γνώρισμα, διότι τότε s=1.

γ) Επίσης μπορούμε να χρησιμοποιήσουμε ευρετήριο (B-δέντρο). Σε μία σύγκριση ισότητας, s εγγραφές θα ικανοποιούν τη συνθήκη, όπου s είναι ο πληθάριθμος επιλογής του γνωρίσματος ευρετηριασμού. Κάθε εγγραφή μπορεί να βρίσκεται σε διαφορετικό block, έτσι $Cost(Filter)=H(I)+s$. Αν όμως, το γνώρισμα ευρετηριασμού είναι επίσης και γνώρισμα κλειδί, τότε το εκτιμούμενο κόστος είναι $Cost(Filter)=H(I)+1$.

Αν η συνθήκη σύγκρισης δεν είναι ισότητα, τότε υποθέτοντας ότι οι μισές περίπου εγγραφές ικανοποιούν τη συνθήκη, προσπελαύνονται περίπου τα μισά block πρώτου επιπέδου του ευρετηρίου και οι μισές εγγραφές του αρχείου μέσω του ευρετηρίου. Το κόστος στην περίπτωση αυτή εκτιμάται, ως $Cost(Filter)=H(I)+(size(I\_top)/2)+(tuples(R)/2)$.

- Order: Απαιτείται ταξινόμηση του αρχείου που περιέχει τον πίνακα προς ταξινόμηση ως προς κάποιο γνώρισμα του. Το κόστος αυτής της διαδικασίας είναι το εξής: $Cost(Order)=(2* sizeof(R))+(2* sizeof(R)*log_2 sizeof(R))$.

- Group: Ισχύουν τα ίδια με το Order.
  Συνεπώς $Cost(Group)=(2* sizeof(R))+(2* sizeof(R)*log_2 sizeof(R))$.

- Join: α) Μπορούμε να χρησιμοποιήσουμε τη μέθοδο nested loops για την πραγματοποίηση του join δύο πινάκων. i) Αν ένας από τους δύο πίνακες χωράει στη μνήμη τότε $Cost(Join)=tuples(R)+tuples(S)$. ii) Αν όμως κανείς από τους δύο δε χωράει στη μνήμη τότε $Cost(Join)=[sizeof(R)/B-1]+sizeof(S)$.
  β) Επίσης μπορούμε να χρησιμοποιήσουμε τη μέθοδο merge-join, σύμφωνα με την οποία πρώτα ταξινομούμε τους δύο προς συνένωση πίνακες. Το κόστος για την ταξινόμηση ενός πίνακα, όπως προαναφέραμε, είναι $(2* sizeof(R))+(2* sizeof(R)*log_2 sizeof(R))$. Το κόστος για την συνένωση δύο πινάκων εφόσον είναι ταξινομημένοι είναι $sizeof(R)+ sizeof(S)+(tuples\_per\_page* sizeof(R)* sizeof(S))/B$, οπότε συνολικά έχουμε, $Cost(Join)=(2* sizeof(R)+(2* sizeof(R)*log_2 sizeof(R))+(2* sizeof(S)+$

(2 sizeof(S)*log₂ sizeof(S))+ sizeof(R)+ sizeof(S)+*
(tuples_per_page sizeof(R)* sizeof(S))/B.*

γ) Ακόμη μπορούμε να χρησιμοποιήσουμε τη μέθοδο hash-join. i) Αν ο ένας από τους δύο πίνακες χωράει στη μνήμη τότε το κόστος είναι *Cost(Join)= tuples(R)+tuples(S).* ii) Αν όμως δε χωράει κανείς από τους δύο πίνακες στη μνήμη, τότε ισχύει
Cost(Join)=3(sizeof(R)+ sizeof(S)).*

5.2 Τελεστές που Εφαρμόζονται στον Κατάλογο

Στο κεφάλαιο 4 εισάγαμε κάποιους νέους τελεστές από τους οποίους κάποιοι έχουν τη χρησιμότητα της προεργασίας πριν την κατασκευή του πλάνου εκτέλεσης ερώτησης. Οι τελεστές αυτοί εφαρμόζονται πάνω στον κατάλογο του ερωτών peer και παρουσιάζονται με τα αντίστοιχα κόστη τους εξής αμέσως.

- CHECK_TABLES: Πραγματοποιείται έλεγχος για κάθε πίνακα που υπάρχει στο FROM clause της ερώτησης με τη βοήθεια του καταλόγου, για το αν είναι νοητός ή υβριδικός ή τίποτα από τα δύο. Ο έλεγχος αυτός μπορεί να γίνει σε σταθερό χρόνο μεταφέροντας σε ένα hash-table στην κύρια μνήμη το τμήμα του καταλόγου που περιέχει αυτήν την πληροφορία κατά το ξεκίνημα του συστήματος.
  Άρα *Cost(CHECK_TABLES)=O(1).*

- CHECK_AGE: Για κάθε πίνακα που ανήκει στο FROM clause της ερώτησης εφαρμόζεται η πράξη της επιλογής πάνω στις πλειάδες του, με συνθήκη επιλογής, η ηλικία τους να είναι μικρότερη από αυτήν που αναφέρεται στην ερώτηση στο AGE clause. Οπότε έχουμε:
  $Cost(CHECK\_AGE)=\sum(sizeof(R_i))$, με R_i τον i πίνακα που υπάρχει στο FROM clause της ερώτησης.

- CHECK_PEERS: Στον κατάλογο του ερωτώντα peer υπάρχει ένας πίνακας με τα χαρακτηριστικά του κάθε peer.
  - Αν αυτός ο πίνακας χωράει στην κύρια μνήμη τότε

*Cost(CHECK_PEERS)=sizeof(R)*W(m_I/O)*, διότι παντού μετράμε σε blocks (I/O), ενώ εδώ μετράμε σε μνήμη.

- Αν αυτός ο πίνακας δε χωράει στην κύρια μνήμη τότε *Cost(CHECK_PEERS)=sizeof(R)*.

5.3 Τελεστές που Λαμβάνουν Μέρος στα Πλάνα Εκτέλεσης Ερωτήσεων

Εκτός από τελεστές οι οποίοι εφαρμόζονται πάνω στον κατάλογο του ερωτώντος peer, υπάρχουν και κάποιοι άλλοι οι οποίοι παίρνουν μέρος στην κατασκευή του πλάνου εκτέλεσης ερωτήσεων. Αυτοί οι τελεστές με τα αντίστοιχα κόστη τους παρουσιάζονται στη συνέχεια. Σημειώνουμε ότι κρατάμε στατιστικά για το αναμενόμενο μέγεθος του αποτελέσματος εφαρμογής κάποιας web service όπως και για το αναμενόμενο μέγεθος του αποτελέσματος εφαρμογής κάποιου workflow. Τα στατιστικά αυτά χρησιμεύουν λόγω του ότι αυτές οι ποσότητες αποτελούν είσοδο στον WRAPPER_POP και στον FILL αντίστοιχα και από τη μορφή των πλάνων που κατασκευάζουμε γνωρίζουμε ότι η έξοδος του CALL_WS αποτελεί είσοδο στον WRAPPER_POP, ενώ η έξοδος του WRAPPER_POP αποτελεί είσοδο στον FILL.

- CALL_WS: Το κόστος εφαρμογής αυτού του τελεστή δίνεται από την εξής ποσότητα:
  Cost(CALL_WS)=cost_con(S_msg)+cost_send(S_msg)+time_wait(O)+ cost_parse(S_msg)+ tuples(O).

- WRAPPER_POP: Το κόστος εφαρμογής του εν λόγω τελεστή υπολογίζεται ως το γινόμενο του μεγέθους της εισόδου του με το κόστος εφαρμογής του workflow αυτού προσθέτοντας και το αναμενόμενο μέγεθος του αποτελέσματος της εφαρμογής του. Οπότε:
  *Cost(Wrapper_POP)=input_tuples(wf)*cost(wf)+tuples(wf)* cost_per_tuple(wf).*
  Επισημαίνουμε ότι το πλήθος των εγγραφών που αποτελούν είσοδο του εν λόγω τελεστή είναι το αναμενόμενο μέγεθος του αποτελέσματος της εφαρμογής του αντίστοιχου CALL_WS τελεστή σε ένα πλάνο.

- FILL: Το κόστος εφαρμογής αυτού του τελεστή υπολογίζεται σε σελίδες με τον εξής τρόπο: *Cost(FILL)=∑(tuples(wf)/tuples_per_page)*, με το tuples(wf) να υπολογίζεται σε πλειάδες και να αντιστοιχεί στο αναμενόμενο μέγεθος αποτελέσματος του αντίστοιχου WRAPPER_POP τελεστή.

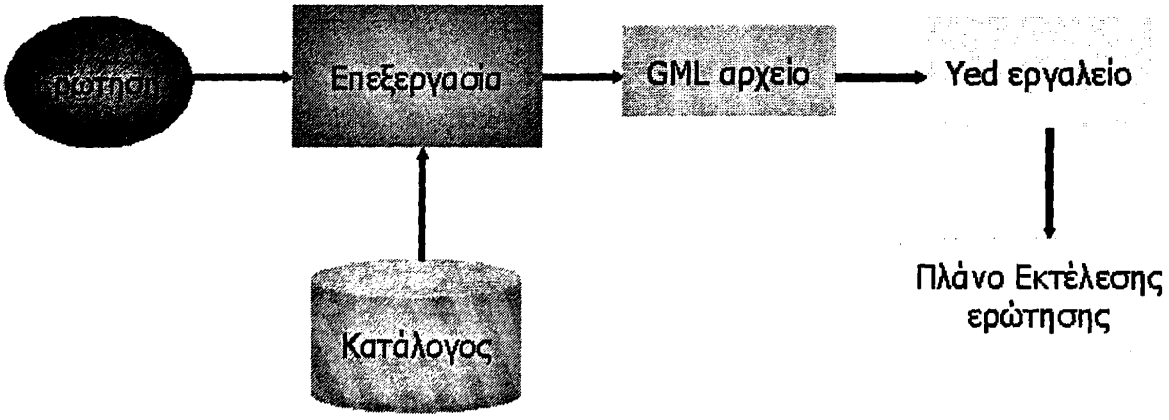
- ExAg: Το κόστος του θεωρείται αμελητέο, διότι ο τελεστής αυτός απλά ενεργοποιεί τη διαδικασία εκτέλεσης της ερώτησης ξανά και ξανά με τη πάροδο συγκεκριμένης χρονικής περιόδου.

ΚΕΦΑΛΑΙΟ 6. ΥΛΟΠΟΙΗΣΗ ΕΦΑΡΜΟΓΗΣ

Στο Κεφάλαιο 6 θα παρουσιάσουμε την υλοποίηση της εφαρμογής μας. Στην πρώτη ενότητα θα δώσουμε μία αναλυτική περιγραφή της εφαρμογής, δίνοντας τον αλγόριθμο που ακολουθήσαμε για την υλοποίησή της. Στη δεύτερη ενότητα παρουσιάζεται η διεπαφή της εφαρμογής, ενώ στην τρίτη ενότητα δίνεται ένα παράδειγμα χρήσης της. Η υλοποίηση της εφαρμογής πραγματοποιήθηκε σε Java με χρήση του εργαλείου NetBeans IDE 4.0.

6.1 Περιγραφή της Εφαρμογής

Αρχικά δίνουμε στο σχήμα 6.1 την αρχιτεκτονική της εφαρμογής που υλοποιήσαμε.



Σχήμα 6.1: Η Αρχιτεκτονική της Εφαρμογής που Υλοποιήσαμε

Ακολουθεί αναλυτική περιγραφή της εφαρμογής αυτής:

α) Διαβάζουμε την ερώτηση που ο χρήστης ζήτησε να εκτελεστεί.

β) Πραγματοποιείται το parsing της ερώτησης, κατασκευάζοντας μία συμβολοσειρά για κάθε clause αυτής.

γ) Διάβασμα του καταλόγου του peer στον οποίο τίθεται η ερώτηση.

δ) Έλεγχος των horizon, availability, response time, class clauses της ερώτησης σε σχέση με τις αντίστοιχες πληροφορίες του καταλόγου του peer που θέτει την ερώτηση.

ε) Με βάση τον παραπάνω έλεγχο, κατασκευή του συνόλου των κόμβων ενδιαφέροντος για τη δεδομένη ερώτηση.

στ) Έλεγχος του timing clause της ερώτησης με σκοπό την εύρεση αν είναι συνεχής ή όχι κι αν όχι ποια είναι η συνθήκη τερματισμού συλλογής πλειάδων.

ζ) Διάβασμα της τοπικής βάσης δεδομένων του peer που θέτει την ερώτηση με σκοπό την εύρεση των νοητών και υβριδικών πινάκων της.

η) Έλεγχος στο from και στο where clause με σκοπό την εύρεση joins μεταξύ των πινάκων του from clause. Οι υπόλοιποι πίνακες στο from clause συνενώνονται με καρτεσιανά γινόμενα.

θ) Δημιουργία δενδρικής δομής στην οποία εισάγονται κόμβοι από κάτω προς τα πάνω και με τέτοιον τρόπο ώστε να οδηγούμαστε σε αριστεροβαθές δέντρο, όπως είναι το επιθυμητό.

ι) Εισαγωγή στην παραπάνω δενδρική δομή από κάτω προς τα πάνω όλων των joins και των καρτεσιανών γινομένων με τους αντίστοιχους πίνακες τους με τη σειρά που τα ανακαλύπτουμε.

κ) Εισαγωγή στη δομή του δέντρου πάντα προς τα πάνω όλων των τελεστών filter. Αυτοί ανακαλύπτονται από το where clause της ερώτησης και κατασκευάζονται ένας για κάθε όρο του where clause που όμως δεν λαμβάνει μέρος σε κάποιο join πινάκων.

λ) Εισαγωγή προς τα πάνω του τελεστή group εάν υπάρχει group by clause στην ερώτηση που μας δίνεται.

μ) Εισαγωγή επίσης προς τα πάνω του τελεστή order εάν υπάρχει order by clause στην ερώτηση.

ν) Εισαγωγή προς τα πάνω του τελεστή project ο οποίος προκύπτει από το select clause της ερώτησης.

ξ) Εάν κατά τον έλεγχο του timing clause της ερώτησης προκύψει ότι είναι συνεχής στην ρίζα του δέντρου εισάγεται ο τελεστής ExAg (Execute Again).

ο) Στη συνέχεια κατασκευάζεται βοηθητικό δέντρο με ρίζα τον τελεστή fill και τόσο παρακλάδια όσοι οι κόμβοι που αποτελούν το σύνολο των κόμβων ενδιαφέροντος για την ερώτηση που μας δόθηκε. Κάθε παρακλάδι έχει από κάτω προς τα πάνω έναν κόμβο για τον peer που αναφερόμαστε, έναν κόμβο για τον τελεστή call_web_services κι έναν κόμβο για τον τελεστή wrapper_pop ο οποίος και ενώνεται με τον τελεστή fill που όπως αναφέραμε είναι η ρίζα αυτού του βοηθητικού δέντρου.

π) Έλεγχος των πινάκων στο from clause της ερώτησης με σκοπό την εύρεση των νοητών ή υβριδικών πινάκων που συμμετέχουν στην ερώτηση.

ρ) Για κάθε έναν από τους νοητούς ή υβριδικούς πίνακες που παίρνουν μέρος στην ερώτηση, εύρεση του αντίστοιχου κόμβου στο αρχικό μας δέντρο και εισαγωγή όλου του βοηθητικού δέντρου που κατασκευάσαμε ως υπόδεντρό τους. Στο βήμα αυτό κατασκευάζεται το συνολικό πλάνο της ερώτησης και είναι αποθηκευμένο στην δενδρική δομή που ορίσαμε.

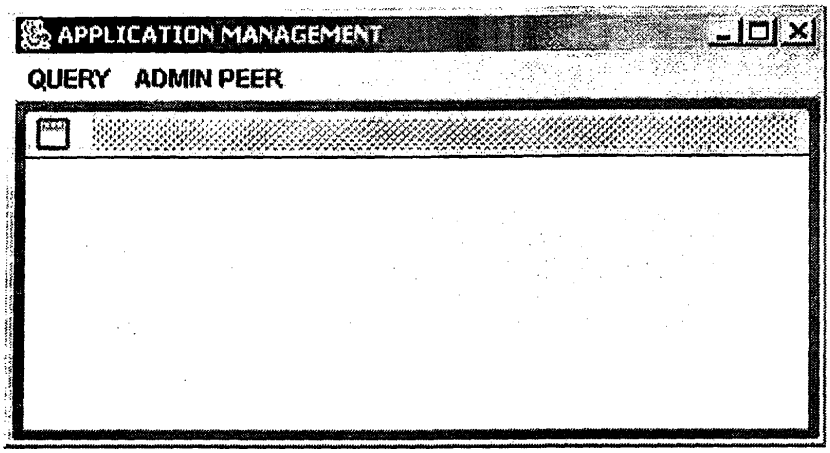
σ) Διάβασμα των κόμβων του δέντρου και κατασκευή του gml αρχείου που αντιστοιχεί στο δέντρο-πλάνο της ερώτησης.

Το gml αρχείο είναι ένα αρχείο, το οποίο περιέχει σε μία γλώσσα περιγραφής γραφημάτων (GML), το γράφημα-δέντρο που προκύπτει ως αποτέλεσμα από την εφαρμογή όλων των παραπάνω βημάτων. Με τη βοήθεια ενός εργαλείου, του Yed, έχουμε τη δυνατότητα να μετατρέψουμε το αρχείο αυτό σε γράφημα, το οποίο προφανώς αντιστοιχίζεται ακριβώς στο γράφημα που περιγράφεται μέσω της γλώσσας GML. Με αυτόν τον τρόπο πετυχαίνουμε να έχουμε ως έξοδο όλης αυτής της διαδικασίας οπτικό αποτέλεσμα, το πλάνο εκτέλεσης της ερώτησης που θέτει κάποιος χρήστης προς έναν peer του συστήματος. Το εργαλείο αυτό βρέθηκε στο διαδίκτυο στη διεύθυνση www.yworks.com/en/products_yed_about.htm.

6.2 Παρουσίαση της Διεπαφής της Εφαρμογής

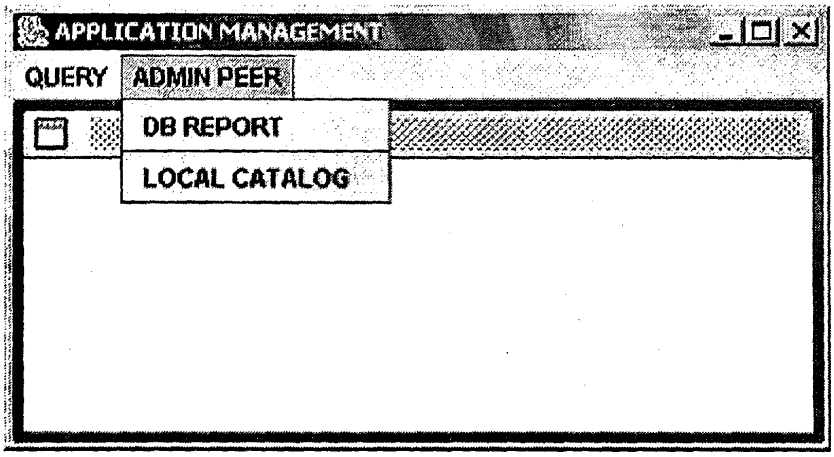
Με σκοπό την πιο εύκολη διαχείριση της εφαρμογής που έχουμε υλοποιήσει, κατασκευάσαμε και μία διεπαφή η οποία μας δίνει διάφορες δυνατότητες. Στην ενότητα αυτή θα παρουσιάσουμε τις δυνατότητες αυτές. Στο σχήμα 6.2 φαίνεται η

αρχική κατάσταση, όπου παρατηρούμε ότι υπάρχουν δύο κουμπιά, ένα για να θέσουμε την ερώτηση που επιθυμούμε κι ένα άλλο για τη διαχείριση του peer στον οποίο θέτουμε την ερώτηση.



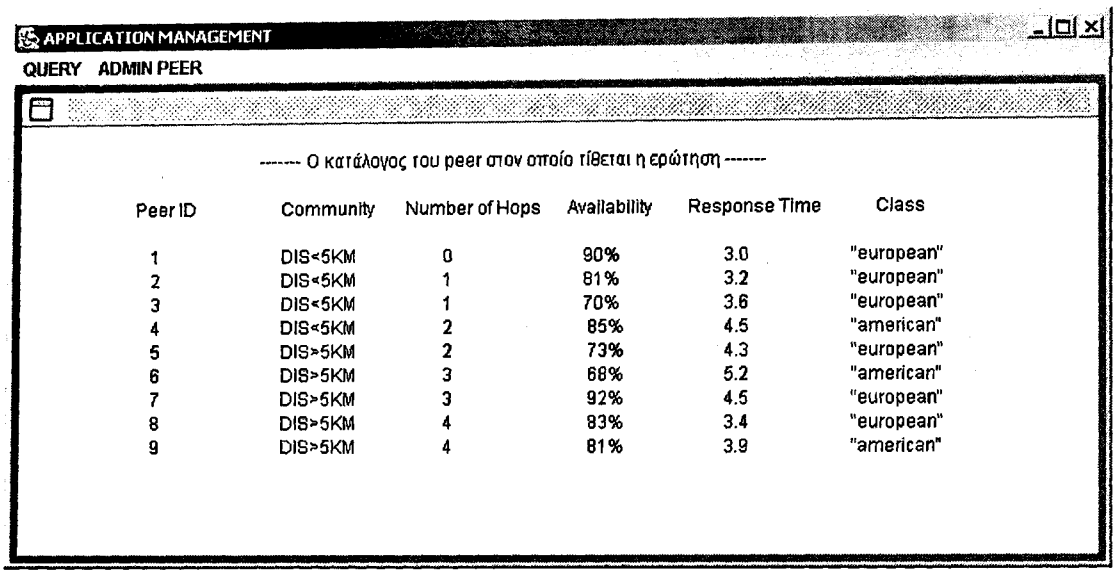
Σχήμα 6.2: Το Αρχικό Παράθυρο της Διεπαφής της Εφαρμογής μας

Εάν επιλέξουμε το κουμπί ADMIN PEER παρουσιάζεται ένα μενού με δύο επιλογές, την DB REPORT και την LOCAL CATALOG. Το μενού με τις δύο αυτές επιλογές φαίνεται στο σχήμα 6.3.

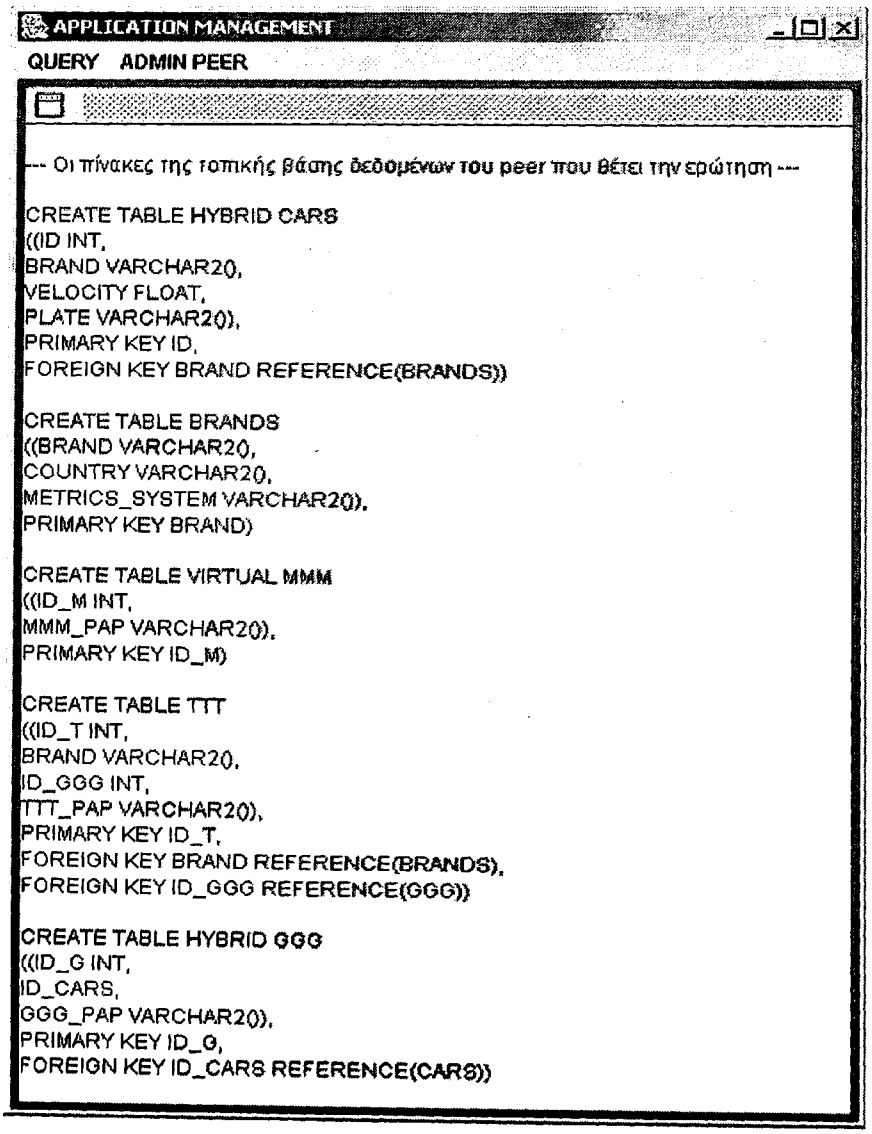


Σχήμα 6.3: Το Μενού Επιλογών αν Διαλέξουμε το Κουμπί ADMIN PEER

Από τις δύο αυτές επιλογές αν διαλέξουμε την LOCAL CATALOG εμφανίζεται ο κατάλογός του (σχήμα 6.4), ενώ αν διαλέξουμε την DB REPORT εμφανίζεται η τοπική βάση δεδομένων του peer στον οποίο θέτουμε την ερώτηση (σχήμα 6.5).

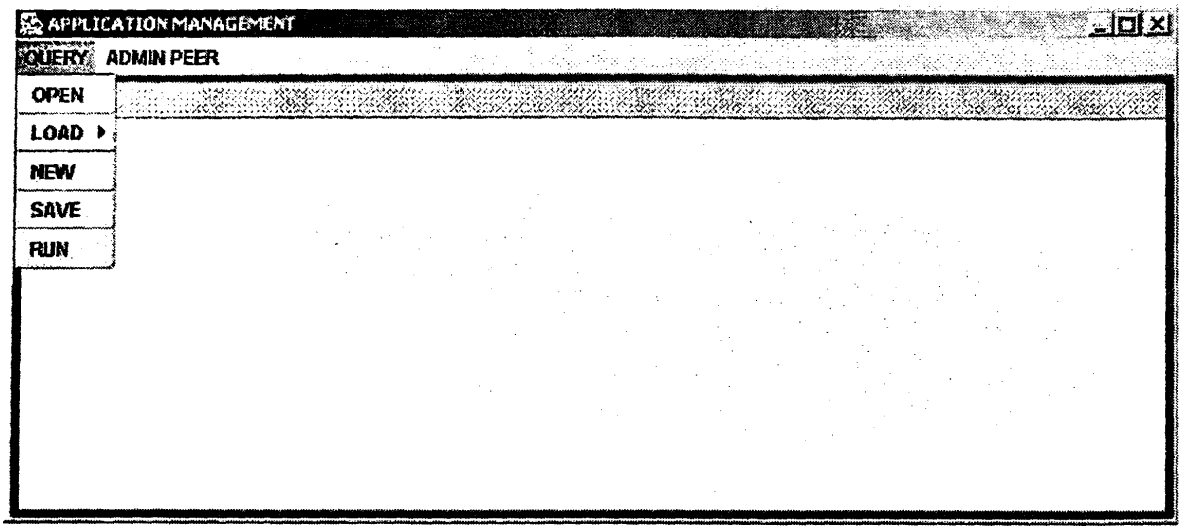


Σχήμα 6.4: Επιλογή του LOCAL CATALOG-Παρουσίαση του τοπικού καταλόγου



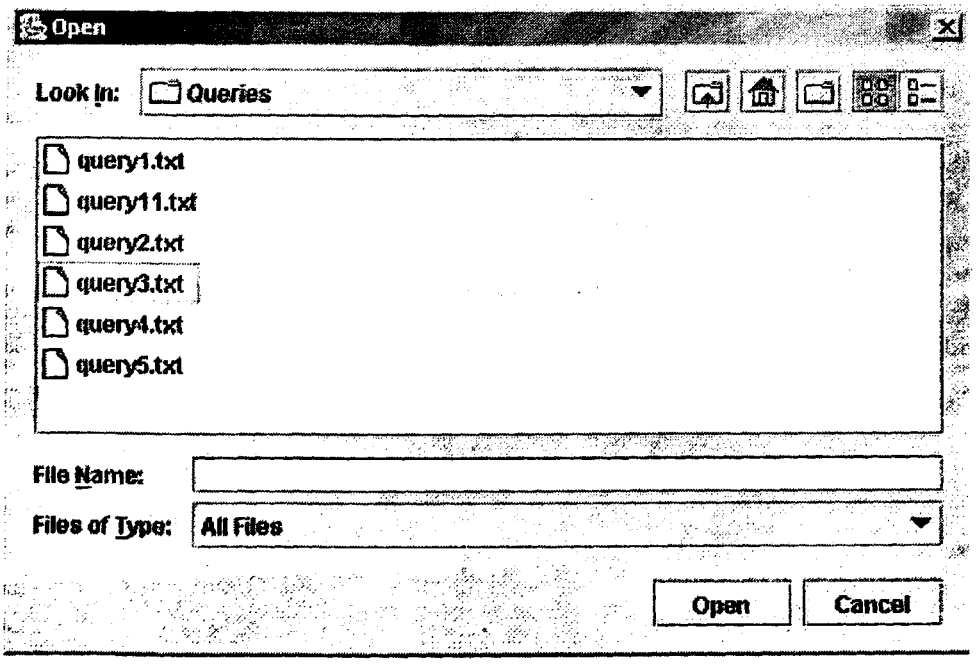
Σχήμα 6.5: Επιλογή DB REPORT-Παρουσίαση της Τοπικής Β.Δ.

Στην περίπτωση που επιλέξουμε το κουμπί QUERY τότε εμφανίζεται ένα μενού επιλογών όπως φαίνεται στο σχήμα 6.6.



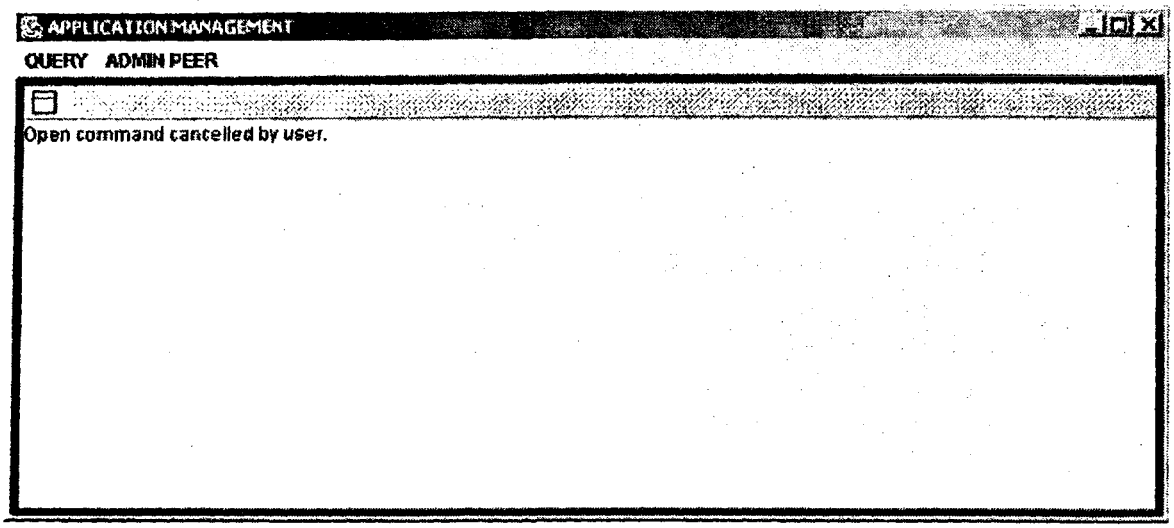
Σχήμα 6.6: Επιλογή του QUERY-Εμφάνιση Μενού

Από το μενού που εμφανίζεται αν επιλέξουμε το OPEN εμφανίζεται ένας επιλογέας αρχείου όπως στα Windows, με τη βοήθεια του οποίου μπορούμε να επιλέξουμε το αρχείο που επιθυμούμε να ανοίξουμε, δηλαδή να επιλέξουμε να ανοίξουμε το αρχείο που περιέχει την ερώτηση που θέλουμε να εκτελέσουμε. Ο επιλογέας φαίνεται στο σχήμα 6.7.



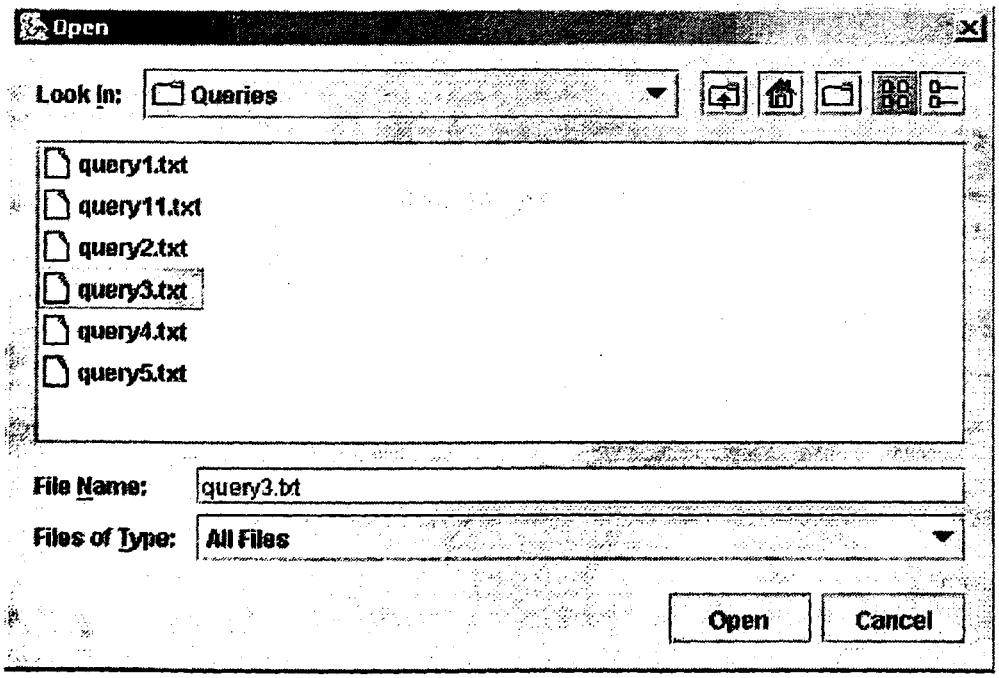
Σχήμα 6.7: Παρουσίαση του Επιλογέα Αρχείου μετά την Επιλογή του OPEN

Επιλέγοντας ένα αρχείο αν πατήσουμε το κουμπί CANCEL ακυρώνεται η διαδικασία ανοίγματος αρχείου και επιστρέφουμε στο αρχικό παράθυρο. Αυτό παρουσιάζεται στο σχήμα 6.8.



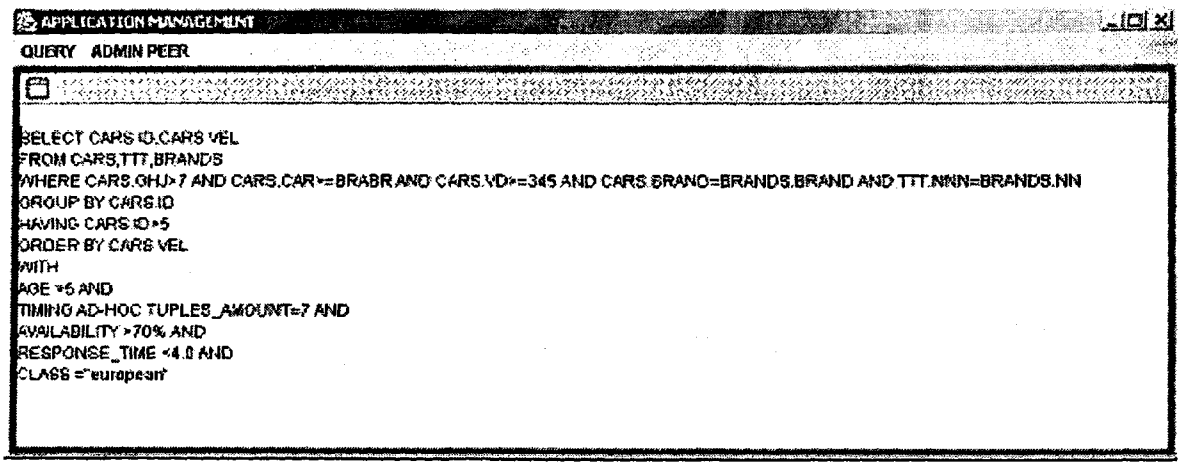
Σχήμα 6.8: Επιστροφή στο Αρχικό Παράθυρο μετα το πάτημα του Κουμπιού CANCEL στον Επιλογέα Αρχείου

Στην περίπτωση που επιλέξουμε κάποιο αρχείο τότε όπως και στα Windows στο πεδίο όνομα αρχείου γράφεται το όνομα του αρχείου που μόλις επιλέξαμε. Η κατάσταση αυτή φαίνεται στο σχήμα 6.9.



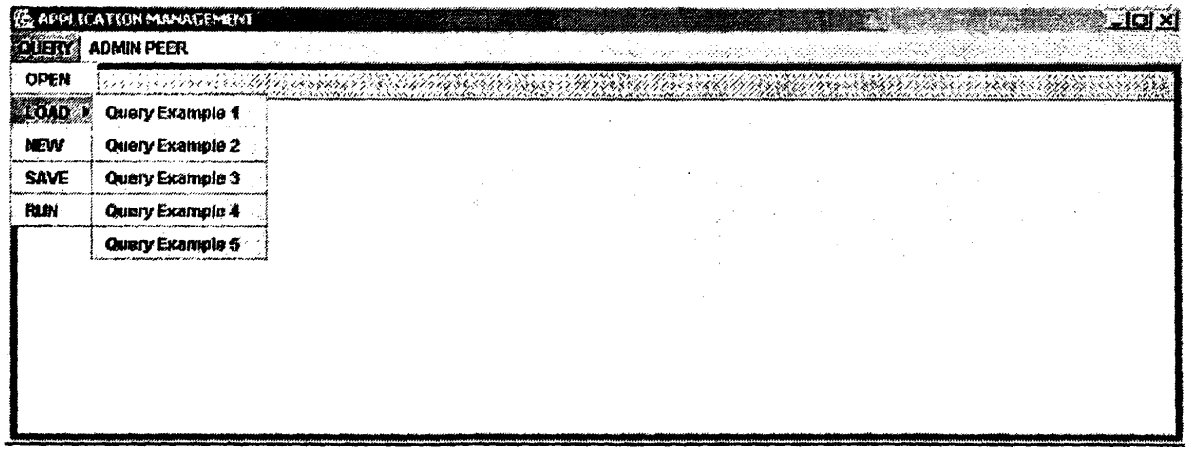
Σχήμα 6.9: Επιλογή του αρχείου query3.txt

Στο σημείο αυτό αν πατήσουμε το κουμπί Cancel τότε ακυρώνουμε την ενέργεια ανοίγματος αρχείου και οδηγούμαστε στην κατάσταση που φαίνεται στο σχήμα 6.8. Αν όμως πατήσουμε το κουμπί Open, τότε ανοίγουμε το αρχείο που έχουμε επιλέξει και στο αρχικό μας παράθυρο παρουσιάζεται το περιεχόμενο του αρχείο αυτού. Το σχήμα 6.10 δείχνει ακριβώς αυτήν την κατάσταση.



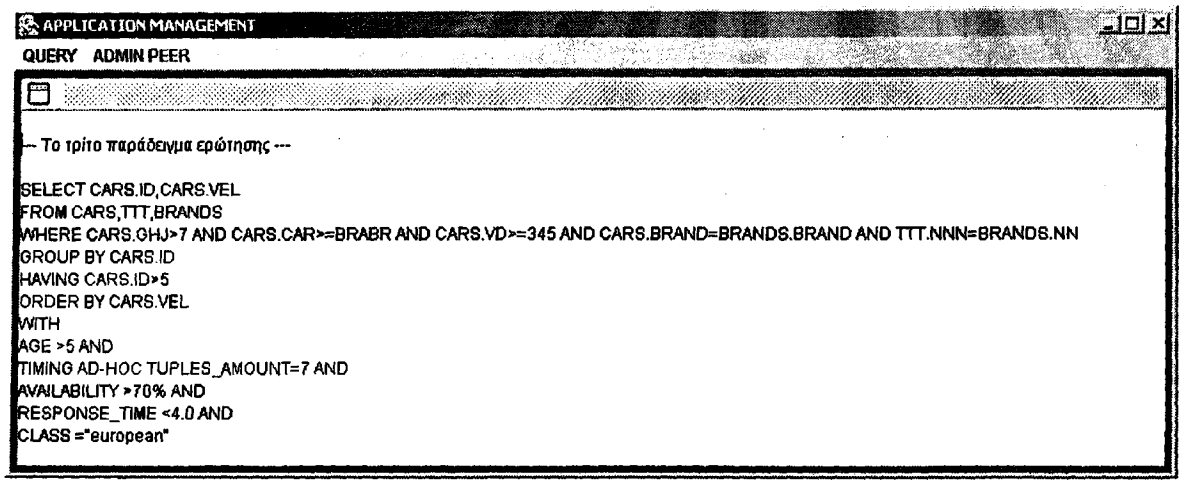
Σχήμα 6.10: Άνοιγμα του Αρχείου query3.txt-Παρουσίαση των περιεχομένων του

Το μενού επιλογών που εμφανίζονται αν επιλέξουμε το κουμπί QUERY, όπως αναφέραμε παραπάνω φαίνεται στο σχήμα 6.6. Αν επιλέξουμε το στοιχείο LOAD εμφανίζεται ένα νέο εσωτερικό μενού με επιλογές 5 αρχεία, δηλαδή 5 προτεινόμενες ερωτήσεις. Το εσωτερικό αυτό μενού φαίνεται στο σχήμα 6.11.



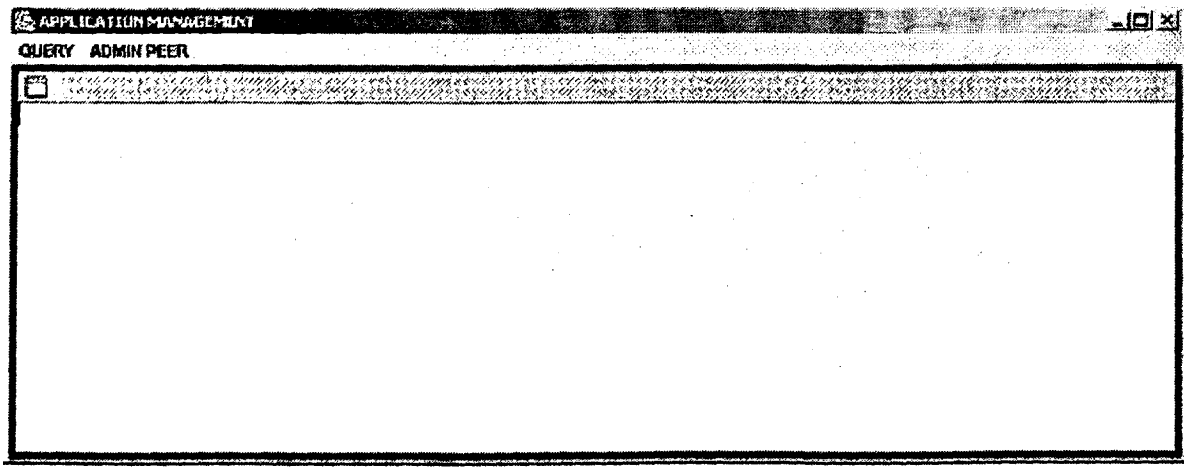
Σχήμα 6.11: Επιλογή LOAD από το Μενού QUERY-Παρουσίαση του Εσωτερικού Μενού με τα 5 Παραδείγματα Ερωτήσεων

Αν επιλέξουμε κάποια από τις 5 προτεινόμενες ερωτήσεις αυτή εμφανίζεται στο αρχικό-κεντρικό παράθυρο της εφαρμογής, όπως δείχνει το σχήμα 6.12 όπου έχουμε διαλέξει την ερώτηση 3.



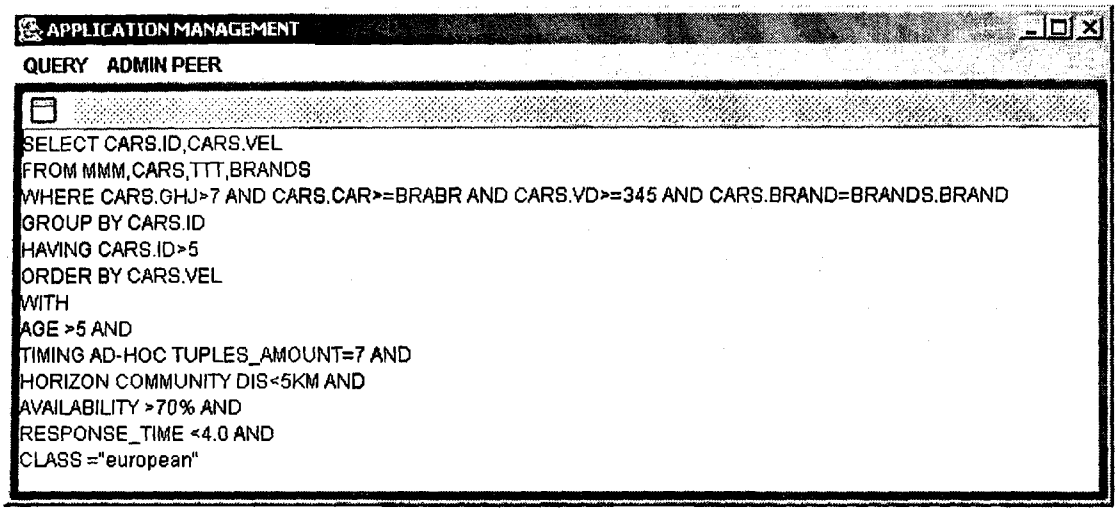
Σχήμα 6.12: Επιλογή-Παρουσίαση της Τρίτης Προτεινόμενης Ερώτησης

Στο σχήμα 6.6 στο οποίο εμφανίζεται το μενού επιλογών αν διαλέξουμε στο αρχικό παράθυρο την επιλογή QUERY, υπάρχει και το στοιχείο του μενού NEW. Αν επιλέξουμε αυτό πρώτον καθαρίζει η περιοχή κειμένου του παραθύρου από οτιδήποτε ήταν γραμμένο πριν και δεύτερον μας δίνεται η δυνατότητα να γράψουμε μία δική μας ερώτηση. Το γεγονός αυτό φαίνεται στο σχήμα 6.13.



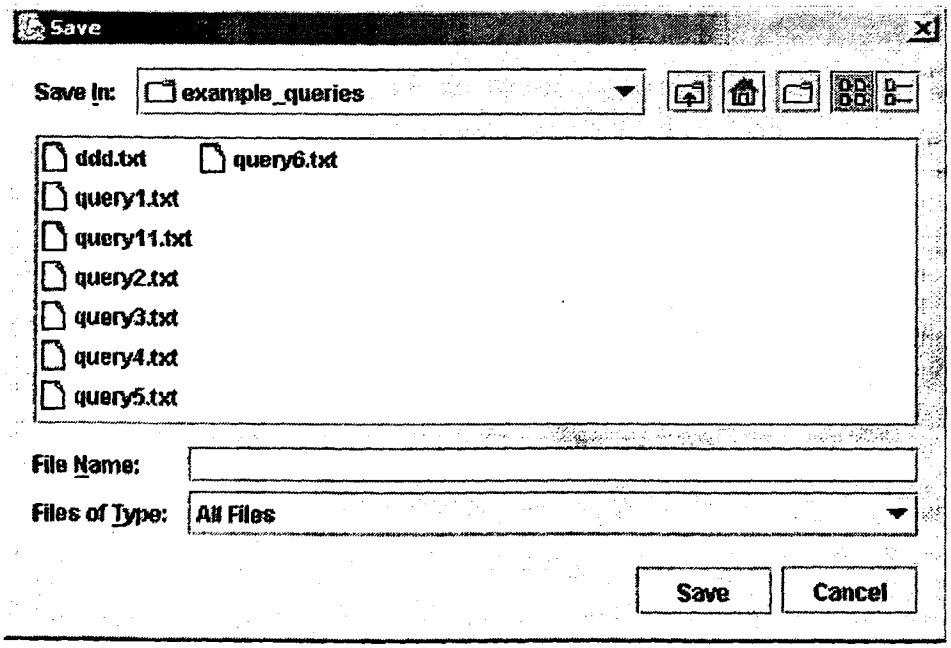
Σχήμα 6.13: Επιλογή του στοιχείου NEW από το Μενού QUERY

Στο σημείο αυτό, όπως προαναφέραμε, έχουμε τη δυνατότητα να γράψουμε οποιαδήποτε ερώτηση επιθυμούμε. Στο σχήμα 6.14 δείχνεται η κατάσταση κατά την οποία έχουμε γράψει μία ερώτηση.



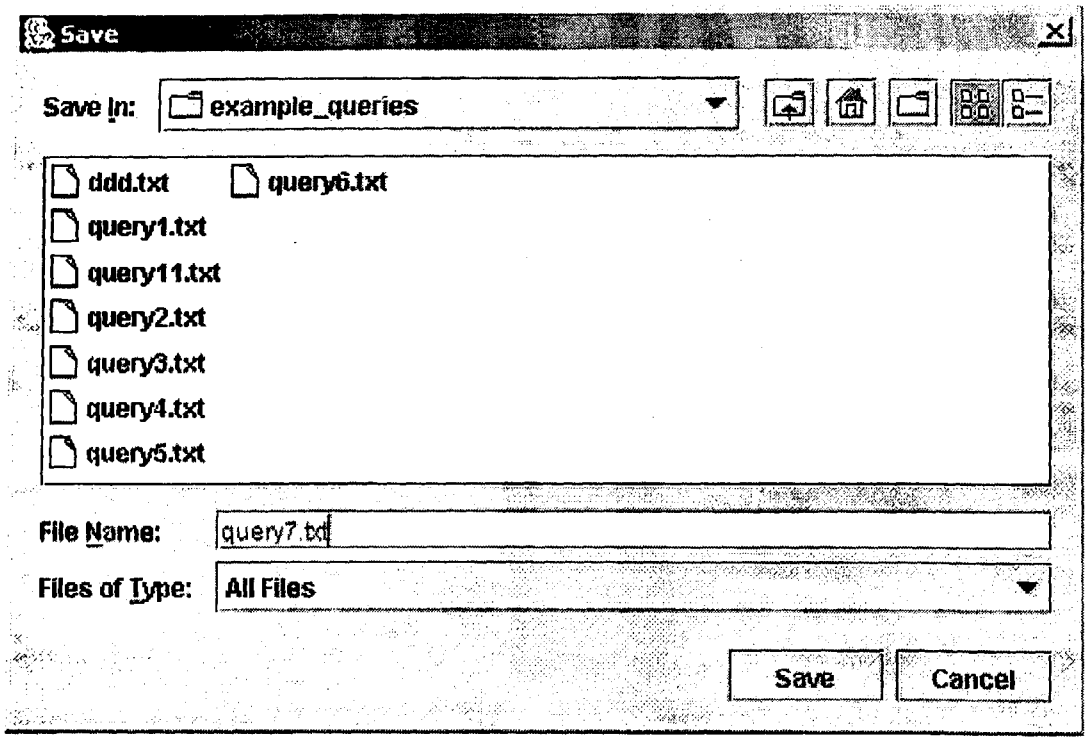
Σχήμα 6.14: Γράψιμο μίας Νέας Ερώτησης

Την δεδομένη ερώτηση που μόλις έχουμε συντάξει, μπορούμε να την αποθηκεύσουμε έτσι ώστε να έχουμε τη δυνατότητα να την ανοίγουμε όποτε επιθυμούμε. Αυτό πραγματοποιείται αν από το μενού QUERY επιλέξουμε το στοιχείο SAVE. Τότε ανοίγει ένα παράθυρο αντίστοιχο με αυτό των Windows για αποθήκευση αρχείου. Το παράθυρο αυτό που ανοίγει φαίνεται στο σχήμα 6.15.



Σχήμα 6.15: Επιλογή του Στοιχείου SAVE από το QUERY Μενού-Παρουσίαση του Παραθύρου για Αποθήκευση Αρχείου

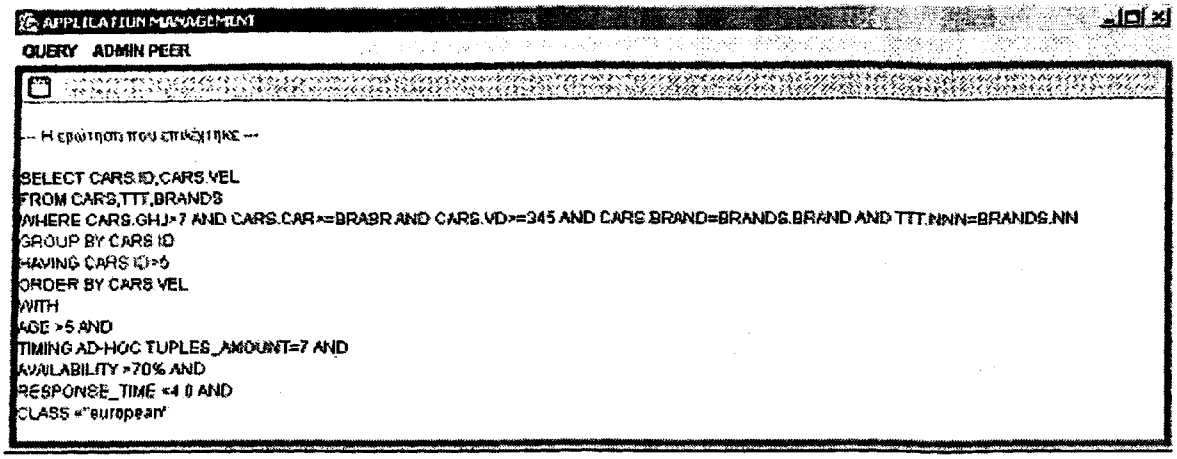
Στο πεδίο File Name μπορούμε να δώσουμε το όνομα που επιθυμούμε για το αρχείο, όπως φαίνεται στο σχήμα 6.16.



Σχήμα 6.16: Ονομάζουμε το προς Αποθήκευση Αρχείο

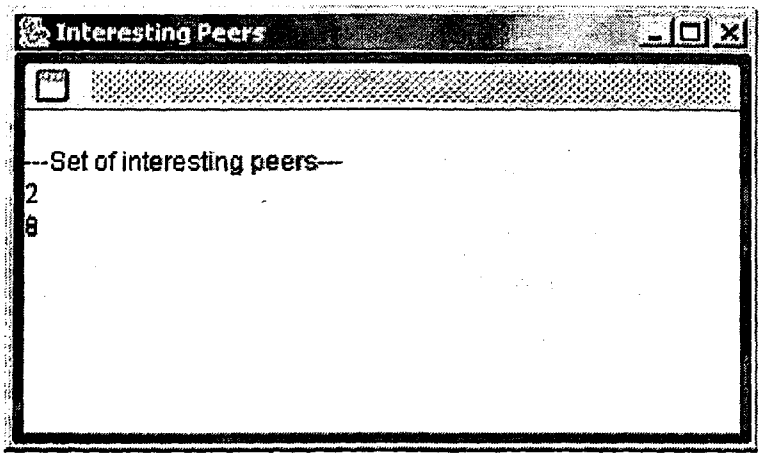
Οποτεδήποτε ενώ το παράθυρο αυτό είναι ανοιχτό, πατήσουμε την επιλογή Cancel, κλείνει το παράθυρο, οδηγούμαστε στο αρχικό-βασικό παράθυρο κι έχουμε ένα μήνυμα ότι ακυρώσαμε την ενέργεια, αντίστοιχα με το σχήμα 6.8. Στην περίπτωση που πατήσουμε το κουμπί Save, τότε η ερώτηση που γράψαμε αποθηκεύεται σε αρχείο με όνομα αυτό που εμείς ορίσαμε, κλείνει αυτό το παράθυρο και οδηγούμαστε στο βασικό παράθυρο.

Το τελευταίο στοιχείο στο μενού QUERY είναι το RUN. Πατώντας το, κατασκευάζεται το πλάνο εκτέλεσης της ερώτησης που έχουμε επιλέξει. Οπότε με βάση και τα παραπάνω, μπορούμε να πατήσουμε στο στοιχείο OPEN του μενού QUERY, μέσω του επιλογέα αρχείων που εμφανίζεται να ανοίξουμε το αρχείο με την ερώτηση που επιθυμούμε να πραγματοποιήσουμε και στη συνέχεια να πατήσουμε το στοιχείο RUN και να κατασκευαστεί το πλάνο εκτέλεσης της ερώτησης που μόλις επιλέξαμε. Έστω ότι με τη βοήθεια του επιλογέα αρχείων, ανοίγουμε-επιλέγουμε την ερώτηση που φαίνεται στο σχήμα 6.17.



Σχήμα 6.17: Η Ερώτηση που Επιλέξαμε

Πατώντας στο RUN που είναι η τελευταία επιλογή στο μενού QUERY, κατασκευάζεται το πλάνο εκτέλεσης της δεδομένης ερώτησης. Επίσης δημιουργούνται άλλα δύο παράθυρα. Στο πρώτο αναγράφονται οι peers που ανήκουν στο σύνολο κόμβων ενδιαφέροντος για τη δεδομένη ερώτηση (σχήμα 6.18) και στο δεύτερο η περιγραφή του δέντρου εκτέλεσης της ερώτησης σε γλώσσα GML.

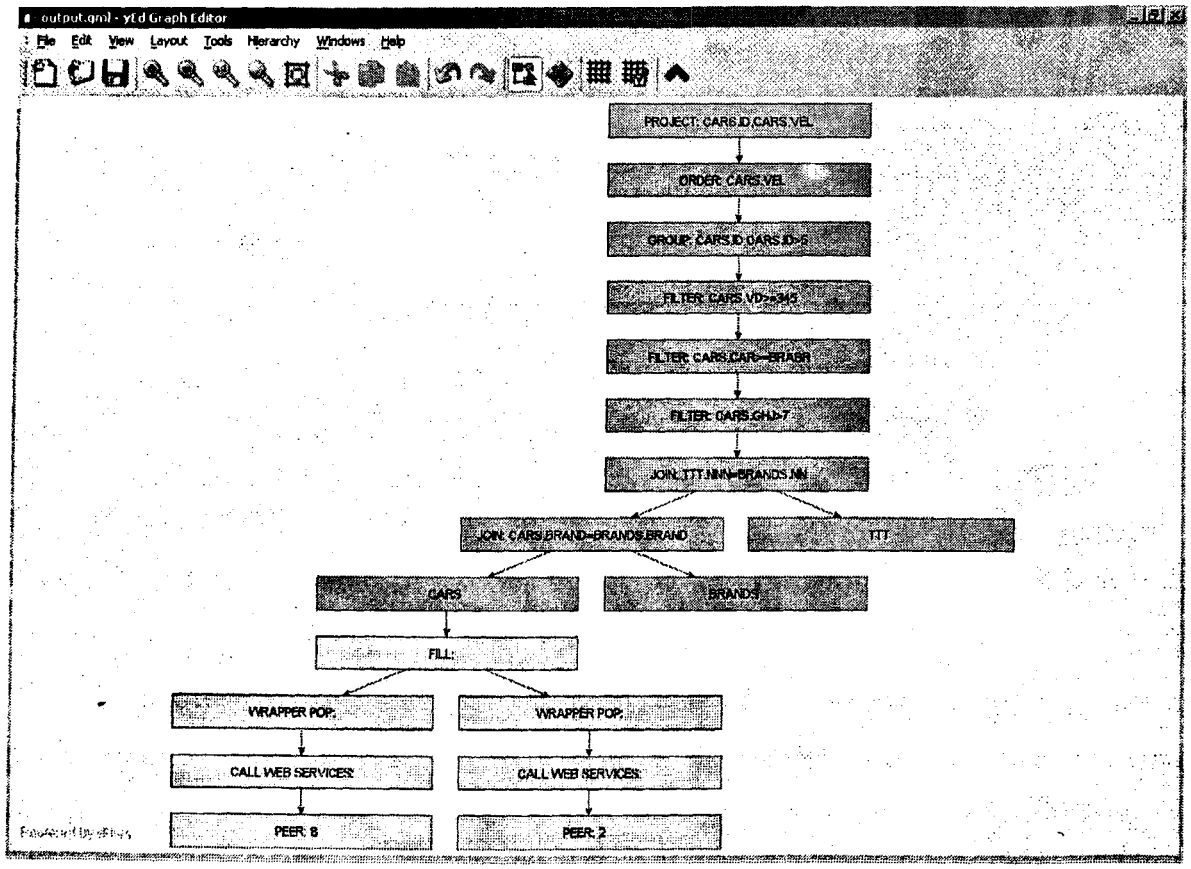


Σχήμα 6.18: Παράθυρο που Αναγράφει το Σύνολο Κόμβων Ενδιαφέροντος για τη Δεδομένη Ερώτησης

Η περιγραφή του πλάνου εκτέλεσης της ερώτησης που εκτελέσαμε, σε γλώσσα περιγραφής γραφημάτων GML, δίνεται στο παράρτημα.

Στο παρακάτω σχήμα, σχήμα 6.19, παρουσιάζεται το πλάνο εκτέλεσης της ερώτησης που επιλέξαμε να εκτελέσουμε, με μορφή γραφήματος, μέσω του εργαλείου Yed. Στο

παρακάτω σχήμα με πιο σκούρο κίτρινο χρώμα συμβολίζονται οι κόμβοι του βασικού δέντρου και με πιο ανοικτό κίτρινο οι κόμβοι του υπόδεντρου που προσθέτουμε κάτω από κάθε νοητό ή υβριδικό πίνακα που συμμετέχει στην ερώτηση.



Σχήμα 6.19: Το Πλάνο Εκτέλεσης της Ερώτησης που Επιλέξαμε

ΚΕΦΑΛΑΙΟ 7. ΣΥΜΠΕΡΑΣΜΑΤΑ

7.1 Επίλογος

Το πρόβλημα με το οποίο ασχοληθήκαμε στην εργασία αυτή είναι η επεξεργασία ερωτήσεων σε ad-hoc δίκτυα ομότιμων πρακτόρων. Σε ένα τέτοιο περιβάλλον θεωρήθηκε ότι κάθε peer διαθέτει μία βάση δεδομένων τοπικά πάνω στην οποία επιθυμούμε να εκτελούμε ερωτήσεις. Η βάση δεδομένων αυτή, αποτελείται από πίνακες που είναι τοπικά αποθηκευμένοι και από πίνακες που είναι νοητοί ή υβριδικοί, δηλαδή πίνακες οι οποίοι αποτελούνται από πλειάδες που αναφέρονται στους άλλους peers του συστήματος. Με βάση τα προαναφερθέντα, η επεξεργασία ερωτήσεων δε θα μπορούσε να πραγματοποιηθεί με τον παραδοσιακό τρόπο, λόγω της ύπαρξης τέτοιων πινάκων στην ερώτηση. Για την επίλυση αυτού του ζητήματος αρχικά ορίστηκε αυστηρά το μοντέλο του συστήματος και αναζητήθηκαν οι απαιτήσεις για μια κατάλληλη γλώσσα επερωτήσεων. Ανακαλύφτηκαν κάποιες διαφορές σε σχέση με την παραδοσιακή SQL κι έτσι οδηγηθήκαμε στην επέκτασή της, ορίζοντας μία νέα γλώσσα, την SQL^P, τέτοια ώστε να επεκτείνει την κλασσική SQL και να υποστηρίζει τις απαιτήσεις της εφαρμογής μας. Ορίστηκε αυστηρά ο τρόπος με τον οποίο πραγματοποιείται η επεξεργασία ερωτήσεων της γλώσσας SQL^P και δόθηκαν αντιπροσωπευτικά παραδείγματα ερωτήσεων για κάθε είδος ερώτησης που μπορούμε να πραγματοποιήσουμε με τις επιπλέον δυνατότητες που μας δίνονται από την γλώσσα SQL^P. Επίσης δόθηκε το ακριβές συντακτικό της γλώσσας που εισάγαμε.

Στη συνέχεια παρουσιάστηκε ένας αρχικός αλγόριθμος εκτέλεσης ερωτήσεων και δόθηκε ο τυπικός ορισμός όλων των τελεστών που μπορεί να λάβουν μέρος σε κάποιο πλάνο εκτέλεσης ερώτησης, συμπεριλαμβάνοντας και κάποιους νέους τελεστές για τους οποίους γεννήθηκε η ανάγκη ορισμού τους για την υλοποίηση των βημάτων του αλγορίθμου. Για κάθε έναν από τους τελεστές αυτούς ορίστηκε μία φόρμουλα κόστους, κατασκευάζοντας ένα ολοκληρωμένο μοντέλο κόστους για τα πλάνα ερώτησης. Τέλος περιγράφηκε η υλοποίηση μίας εφαρμογής υποβολής ερωτήσεων που δίνει την δυνατότητα σε κάποιον χρήστη να θέτει μία ερώτηση σε γλώσσα SQL^P πάνω σε κάποιον peer του συστήματος και να παίρνει ως αποτέλεσμα ένα πλάνο εκτέλεσης της ερώτησης αυτής, το οποίο επιπλέον παρουσιάζεται ως γράφημα με τη βοήθεια ενός εργαλείου οπτικοποίησης γραφημάτων, το Yed.

7.2 Μελλοντική Εργασία

Στην προηγούμενη ενότητα έχουμε αναφέρει επιγραμματικά τα κεντρικά σημεία της εργασίας αυτής. Θεωρούμε ότι αποτελεί ένα ισχυρό υπόβαθρο για περαιτέρω επεκτάσεις στο μέλλον. Δύο σημαντικά σημεία που πιστεύουμε ότι επιδέχονται μελλοντική έρευνα-εργασία είναι η βελτιστοποίηση των ερωτήσεων της γλώσσας που εισάγαμε και ο εναλλακτικός τρόπος εκτέλεσης των ερωτήσεων.

Ο στόχος της δεδομένης εργασίας είναι η επεξεργασία ερωτήσεων σε ένα ad-hoc δίκτυο ομότιμων πρακτόρων. Με τον ορισμό της γλώσσας SQL^P μπορούμε να θέτουμε κάποια ερώτηση πάνω σε έναν peer του συστήματος και να κατασκευάζεται ένα πλάνο εκτέλεσης της ερώτησης αυτής. Επίσης έχουμε ορίσει κι ένα μοντέλο κόστους έτσι ώστε να μπορούμε να εκτιμούμε το κόστος εκτέλεσης ενός πλάνου. Μελλοντικά έχοντας αυτά τα εργαλεία θα ήταν ενδιαφέρον να πραγματοποιηθεί η διαδικασία της βελτιστοποίησης των ερωτήσεων αυτών. Δηλαδή, να παράγουμε όλα τα δυνατά εναλλακτικά πλάνα εκτέλεσης της ερώτησης και με τη βοήθεια του μοντέλου κόστους που έχουμε ορίσει, να επιλέγουμε το φθηνότερο από όλα αυτά προς εκτέλεση.

Επίσης στην τελευταία ενότητα του κεφαλαίου 4 αναφέρουμε έναν εναλλακτικό τρόπο εκτέλεσης μίας ερώτησης. Η ενότητα αυτή αξίζει μελλοντικά περισσότερη έρευνα, διότι θεωρούμε ότι σε κάποιες περιπτώσεις ο εναλλακτικός αυτός τρόπος εκτέλεσης ερώτησης είναι ιδιαίτερα ευεργετικός τόσο ως προς το κόστος εκτέλεσης μίας ερώτησης όσο και ως προς την ποιότητα των δεδομένων που θα παίρνονται ως αποτέλεσμα της εκτέλεσης μίας ερώτησης.

ΑΝΑΦΟΡΕΣ

[Car04] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold and Krys Kochut. Quality of service for workflows and web service processes. In Journal of Web Semantics, pages 281-308, 2004.

[ChKo00] Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.

[DeAb00] Anind K. Dey and Gregory D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In Proceedings of Workshop on The What, Who, Where, When and How of Context-Awareness at the CHI 2000 Conference on Human Factors in Computing Systems (CHI 2000), The Hague, The Netherlands, April 2000.

[Dey01] Anind K. Dey. Understanding and Using Context. Personal and Ubiquitous Computing Journal, 5(1):4-7, 2001.

[GrSa01] Philip Gray and Daniel Salber. Modelling and Using Sensed Context Information in the Design of Interactive Applications. In Proceedings of Eighth IFIP International Conference on Engineering for Human-Computer Interaction (EHCI 2001), pages 317-335, Toronto, Canada, May 2001.

[HaST05] Joachim Hammer, Michael Stonebraker and [illegible]

ΑΝΑΦΟΡΕΣ

[Car+04] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold and Krys Kochut. Quality of service for workflows and web service processes. In *Journal of Web Semantics*, pages 281-308, 2004.

[ChKo00] Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November, 2000.

[DeAb99] Anind K. Dey and Gregory D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of Workshop on The What, Who, Where, When, and How of Context-Awareness* of the *2000 Conference on Human Factors in Computing Systems (CHI 2000)*, The Hague, The Netherlands, April, 2000.

[Dey01] Anind K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing Journal*, 5(1):4-7, 2001.

[GrSa01] Philip D.Gray and Daniel Salber. Modeling and Using Sensed Context Information in the Design of Interactive Applications. In *Proceedings of Eighth IFIP International Conference on Engineering for Human-Computer Interaction (EHCI 2001)*, pages 317-336, Toronto, Canada, May 2001.

[HaST05] Joachim Hammer, Michael Stonebraker and Oguzhan Topsakal.

THALIA: Test Harness for the Assessment of Legacy Information Integration Approaches. In Proceedings of the 21st International Conference on Data Engineering (ICDE), 2005, Tokyo, Japan.

[HKWY97] Laura M. Haas, Donald Kossmann, Edward L. Wimmers and Jun Yang. Optimizing Queries Across Diverse Data Sources. In *Proceedings of 23rd International Conference on Very Large Data Bases, VLDB'97*, pages 276-285, Athens, Greece, August 1997.

[Ioan96] Y. Ioannidis. Query Optimization. ACM Computing Surveys, symposium issue on the 50th Anniversary of ACM, Vol. 28, No. 1, March 1996, pages 121-123.

[Iss+05] Valerie Issarny, Daniele Sacchetti, Ferda Tartanoglu, Francoise Sailhan, Rafik Chibout, Nicole Levy, Angel Talamona. Developing Ambient Intelligence Systems: A Solution based on Web Services. In *Journal of Automated Software Engineering*, 12(1), pp. 101-137, 2005, Kluwer academic publisher.

[KeKe04] Markus Keidl and Alfons Kemper. Towards Context-Aware Adaptable Web Services. In *Proceedings of Ninth International Conference on Extending Database Technology (EDBT 2004)*, pages 826-829, Heraklion, Crete, Greece, March 2004.

[Koss00] Kossmann Donald. The State of the Art in Distributed Query Processing. ACM Computing Surveys 32(4), pages 422–469, 2000.

[MaHa03] J. Madhavan and A. Y. Halevy. Composing Mappings Among Data Sources. In *International Conference on Very Large Data Bases (VLDB)*, pages 572-583, 2003.

[NoPa03] Moira C. Norrie and Alexios Palinginis. Versions for Context

Dependent Information Services. *Springer-Verlag Heidelberg*, pages 503-515, 2003.

[PiTs01] Thomi Pilioura and Aphrodite Tsalgatidou. E-Services: Current Technology and Open Issues. Springer-Verlag Berlin Heidelberg 2001. In *Proceedings of Second International Workshop on Technologies for E-Services (TES 2001)*, pages 1-15, Rome, Italy, September 2001.

[POSV04] Abhijit A. Patil, Swapna A. Oundhakar, Amit P. Sheth, Kunal Verma. METEOR-S Web Service Annotation Framework. In *Proceedings of the International WWW Conference*, New York, USA, 2004.

[RaBe01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *Spriger Verlag, New York*, volume 10, issue 4, pages: 334-350, 2001.

[RoSc97] M. Tork Roth and P. Schwarz. Don't scrap it, wrap it! A wrapper architecture for legacy sources. In *Proceeding of the 23th VLDB Conference,* pages 266-275, Athens, Greece, 1997.

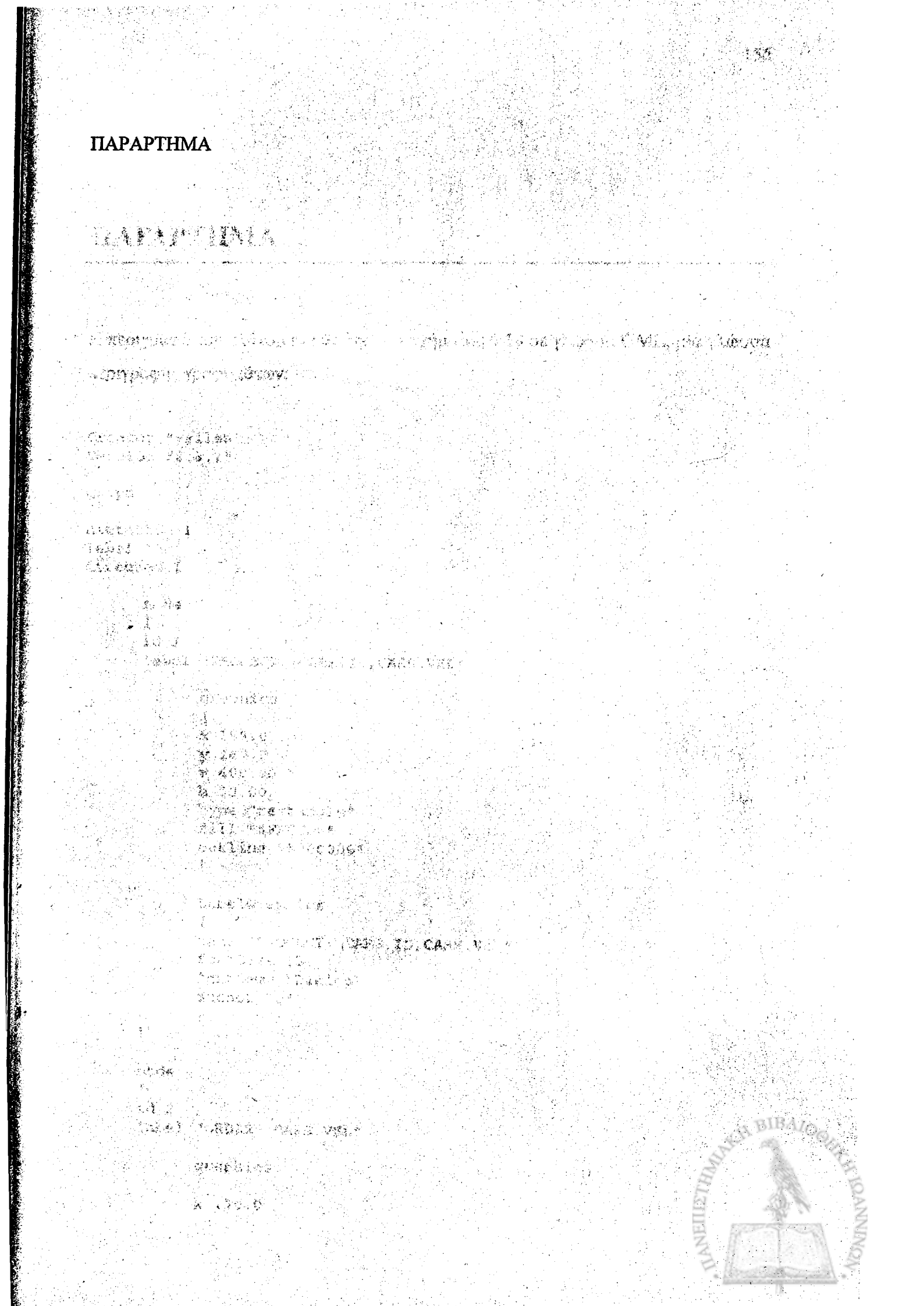
[Sel+79] P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access path selection in a relational database management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 23-34, Boston, MA, USA, 1979.

[Theo03] Yannis Theodoridis. Ten Benchmark Database Queries for Location-based Services. In *Computer Journal* volume 46, issue 6, pages: 713-725, November 2003.

[UrFr99] T. Urhan and M. J. Franklin. XJoin: Getting Fast Answers from Slow and Bursty Networks. *University of Maryland Technical Report, CS-TR-3994.*, February, 1999.

[ZaVP05] A. Zarras, P. Vassiliadis and E. Pitoura. Query Management over Ad-Hoc Communities of Web Services. In *Proceedings of the IEEE International Conference on Pervasive Services 2005 (ICPS'05),* 11-14 July 2005, Santorini, Greece.

ΠΑΡΑΡΤΗΜΑ



ΠΑΡΑΡΤΗΜΑ

Η περιγραφή του πλάνου εκτέλεσης του σχήματος 6.18 σε γλώσσα GML, μία γλώσσα περιγραφής γραφημάτων:

```
Creator "yFiles"
Version "2.3.1"

graph
[
hierachic 1
label ""
directed 1

      node
      [
      id 1
      label "PROJECT: CARS.ID,CARS.VEL"

            graphics
            [
            x 185.0
            y 140.0
            w 400.00
            h 50.00
            type "rectangle"
            fill "#FFCC00"
            outline "#000000"
            ]

            LabelGraphics
            [
            text "PROJECT: CARS.ID,CARS.VEL"
            fontSize 13
            fontName "Dialog"
            anchor "c"
            ]
      ]

      node
      [
      id 2
      label "ORDER: CARS.VEL"

            graphics
            [
            x 170.0
```

```
        y 125.0
        w 400.00
        h 50.00
        type "rectangle"
        fill "#FFCC00"
        outline "#000000"
        ]

        LabelGraphics
        [
        text "ORDER: CARS.VEL"
        fontSize 13
        fontName "Dialog"
        anchor "c"
        ]
]

node
[
id 3
label "GROUP: CARS.ID CARS.ID>5"

        graphics
        [
        x 155.0
        y 110.0
        w 400.00
        h 50.00
        type "rectangle"
        fill "#FFCC00"
        outline "#000000"
        ]

        LabelGraphics
        [
        text "GROUP: CARS.ID CARS.ID>5"
        fontSize 13
        fontName "Dialog"
        anchor "c"
        ]
]

node
[
id 4
label "FILTER: CARS.VD>=345"

        graphics
        [
        x 140.0
        y 95.0
        w 400.00
        h 50.00
        type "rectangle"
        fill "#FFCC00"
        outline "#000000"
        ]

        LabelGraphics
        [
```

```
        text "FILTER: CARS.VD>=345"
        fontSize 13
        fontName "Dialog"
        anchor "c"
        ]
]

node
[
id 5
label "FILTER: CARS.CAR>=BRABR"

        graphics
        [
        x 125.0
        y 80.0
        w 400.00
        h 50.00
        type "rectangle"
        fill "#FFCC00"
        outline "#000000"
        ]

        LabelGraphics
        [
        text "FILTER: CARS.CAR>=BRABR"
        fontSize 13
        fontName "Dialog"
        anchor "c"
        ]
]

node
[
id 6
label "FILTER: CARS.GHJ>7"

        graphics
        [
        x 110.0
        y 65.0
        w 400.00
        h 50.00
        type "rectangle"
        fill "#FFCC00"
        outline "#000000"
        ]

        LabelGraphics
        [
        text "FILTER: CARS.GHJ>7"
        fontSize 13
        fontName "Dialog"
        anchor "c"
        ]
]

node
[
id 7
```

```
label "JOIN: TTT.NNN=BRANDS.NN"

        graphics
        [
        x 95.0
        y 50.0
        w 400.00
        h 50.00
        type "rectangle"
        fill "#FFCC00"
        outline "#000000"
        ]

        LabelGraphics
        [
        text "JOIN: TTT.NNN=BRANDS.NN"
        fontSize 13
        fontName "Dialog"
        anchor "c"
        ]
]

node
[
id 8
label "JOIN: CARS.BRAND=BRANDS.BRAND"

        graphics
        [
        x 80.0
        y 35.0
        w 400.00
        h 50.00
        type "rectangle"
        fill "#FFCC00"
        outline "#000000"
        ]

        LabelGraphics
        [
        text "JOIN: CARS.BRAND=BRANDS.BRAND"
        fontSize 13
        fontName "Dialog"
        anchor "c"
        ]
]

node
[
id 9
label "TTT"

        graphics
        [
        x 100.0
        y 35.0
        w 400.00
        h 50.00
        type "rectangle"
        fill "#FFCC00"
```

```
        outline "#000000"
        ]

        LabelGraphics
        [
        text "TTT"
        fontSize 13
        fontName "Dialog"
        anchor "c"
        ]
]

node
[
id 10
label "CARS"

      graphics
      [
      x 85.0
      y 20.0
      w 400.00
      h 50.00
      type "rectangle"
      fill "#FFCC00"
      outline "#000000"
      ]

      LabelGraphics
      [
      text "CARS"
      fontSize 13
      fontName "Dialog"
      anchor "c"
      ]
]

node
[
id 11
label "BRANDS"

      graphics
      [
      x 105.0
      y 20.0
      w 400.00
      h 50.00
      type "rectangle"
      fill "#FFCC00"
      outline "#000000"
      ]

      LabelGraphics
      [
      text "BRANDS"
      fontSize 13
      fontName "Dialog"
      anchor "c"
      ]
```

```
]

node
[
id 12
label "FILL: "

      graphics
      [
      x 125.0
      y 20.0
      w 400.00
      h 50.00
      type "rectangle"
      fill "#FFFF00"
      outline "#000000"
      ]

      LabelGraphics
      [
      text "FILL: "
      fontSize 13
      fontName "Dialog"
      anchor "c"
      ]
]

node
[
id 13
label "WRAPPER POP: "

      graphics
      [
      x 145.0
      y 20.0
      w 400.00
      h 50.00
      type "rectangle"
      fill "#FFFF00"
      outline "#000000"
      ]

      LabelGraphics
      [
      text "WRAPPER POP: "
      fontSize 13
      fontName "Dialog"
      anchor "c"
      ]
]

node
[
id 14
label "CALL WEB SERVICES: "

      graphics
      [
      x 165.0
```

```
        y 20.0
        w 400.00
        h 50.00
        type "rectangle"
        fill "#FFFF00"
        outline "#000000"
        ]

        LabelGraphics
        [
        text "CALL WEB SERVICES: "
        fontSize 13
        fontName "Dialog"
        anchor "c"
        ]
]
node
[
id 15
label "PEER: 8"

      graphics
      [
      x 185.0
      y 20.0
      w 400.00
      h 50.00
      type "rectangle"
      fill "#FFFF00"
      outline "#000000"
      ]

      LabelGraphics
      [
      text "PEER: 8"
      fontSize 13
      fontName "Dialog"
      anchor "c"
      ]
]

node
[
id 16
label "WRAPPER POP: "

      graphics
      [
      x 205.0
      y 20.0
      w 400.00
      h 50.00
      type "rectangle"
      fill "#FFFF00"
      outline "#000000"
      ]

      LabelGraphics
      [
      text "WRAPPER POP: "
```

```
        fontSize 13
        fontName "Dialog"
        anchor "c"
        ]
 ]

 node
 [
 id 17
 label "CALL WEB SERVICES: "

        graphics
        [
        x 225.0
        y 20.0
        w 400.00
        h 50.00
        type "rectangle"
        fill "#FFFF00"
        outline "#000000"
        ]

        LabelGraphics
        [
        text "CALL WEB SERVICES: "
        fontSize 13
        fontName "Dialog"
        anchor "c"
        ]
 ]
 node
 [
 id 18
 label "PEER: 2"

        graphics
        [
        x 245.0
        y 20.0
        w 400.00
        h 50.00
        type "rectangle"
        fill "#FFFF00"
        outline "#000000"
        ]

        LabelGraphics
        [
        text "PEER: 2"
        fontSize 13
        fontName "Dialog"
        anchor "c"
        ]
]

edge
[
source 1
target 2
```

```
        graphics
        [
        fill "#000000"
        targetArrow "standard"
        ]
]

edge
[
source 2
target 3

        graphics
        [
        fill "#000000"
        targetArrow "standard"
        ]
]

edge
[
source 3
target 4

        graphics
        [
        fill "#000000"
        targetArrow "standard"
        ]
]

edge
[
source 4
target 5

        graphics
        [
        fill "#000000"
        targetArrow "standard"
        ]
]

edge
[
source 5
target 6

        graphics
        [
        fill "#000000"
        targetArrow "standard"
        ]
]

edge
[
source 6
target 7
```

```
        graphics
        [
        fill "#000000"
        targetArrow "standard"
        ]
]

edge
[
source 7
target 8

      graphics
      [
      fill "#000000"
      targetArrow "standard"
      ]
]

edge
[
source 7
target 9

      graphics
      [
      fill "#000000"
      targetArrow "standard"
      ]
]

edge
[
source 8
target 10

      graphics
      [
      fill "#000000"
      targetArrow "standard"
      ]
]

edge
[
source 8
target 11

      graphics
      [
      fill "#000000"
      targetArrow "standard"
      ]
]

edge
[
source 10
target 12
```

```
        graphics
        [
        fill "#000000"
        targetArrow "standard"
        ]
]

edge
[
source 12
target 13

      graphics
      [
      fill "#000000"
      targetArrow "standard"
      ]
]

edge
[
source 13
target 14

      graphics
      [
      fill "#000000"
      targetArrow "standard"
      ]
]

edge
[
source 14
target 15

      graphics
      [
      fill "#000000"
      targetArrow "standard"
      ]
]

edge
[
source 12
target 16

      graphics
      [
      fill "#000000"
      targetArrow "standard"
      ]
]

edge
[
source 16
target 17
```

```
            graphics
            [
            fill "#000000"
            targetArrow "standard"
            ]
        ]

        edge
        [
        source 17
        target 18

            graphics
            [
            fill "#000000"
            targetArrow "standard"
            ]
        ]
]
```

ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ

Ο Φωλίνας Νικόλαος του Χρήστου και της Παρασκευής γεννήθηκε στα Τρίκαλα στις 24 Απριλίου του 1980. Φοίτησε στο 3ο Λύκειο Τρικάλων και το 1998 εισήχθη στο Πανεπιστήμιο Ιωαννίνων στο τμήμα Πληροφορικής. Τον Ιούλιο του 2003 πήρε το πτυχίο της Πληροφορικής με βαθμό 6,71 (λίαν καλώς). Τον Σεπτέμβριο του 2003 εισήχθη στο μεταπτυχιακό πρόγραμμα του τμήματος Πληροφορικής στο Πανεπιστήμιο Ιωαννίνων.