Πανεπιστήμιο Κρήτης
Σχολή Θετικών Επιστημών
Τμήμα Επιστήμης Υπολογιστών

TELQUEL: Μια ερωτηματική γλώσσα για το SIS

Ιωάννης Ζηδιανάκης

Μεταπτυχιακή Εργασία

Ηράκλειο, Νοέμβριος 1998

Πανεπιστήμιο Κρήτης
Σχολή Θετικών Επιστημών
Τμήμα Επιστήμης Υπολογιστών

TELQUEL: Μια ερωτηματική γλώσσα για το SIS

Εργασία που υποβλήθηκε από τον
Ιωάννη Ζηδιανάκη για την μερική
εκπλήρωση των απαιτήσεων για την
απόκτηση του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ
ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

_____

Ιωάννης Ζηδιανάκης
Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

_____

Πάνος Κωνσταντόπουλος, Καθηγητής, Επόπτης

_____

Κατερίνα Χούστη, Αναπληρώτρια Καθηγήτρια, Μέλος

_____

Γιώργος Γεωργακόπουλος, Επίκουρος Καθηγητής, Μέλος

_____

Αναστασία Αναλυτή, Εντεταλμένη Ερευνήτρια, Επιβλέπουσα

Δεκτή:

_____

Πάνος Κωνσταντόπουλος, Καθηγητής
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Ηράκλειο, Νοέμβριος 1998

TELQUEL: Μια ερωτηματική γλώσσα για το SIS

Γιάννης Ζηδιανάκης

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

ΠΕΡΙΛΗΨΗ

Το SIS (Semantic Index System) μπορεί να χαρακτηριστεί ως ένα οντοκεντρικό σύστημα διαχείρισης βάσεων δεδομένων που στηρίζεται στη γλώσσα παράστασης γνώσεων TELOS. Η γλώσσα TELOS χρησιμοποιείται για τον ορισμό του σχήματος (DDL) και τη διαχείριση των αντικειμένων (DML) της βάσης και στηρίζεται σε τρεις μηχανισμούς αφαίρεσης: της ταξινόμησης (classification), της απόδοσης γνωρίσματος (attribution) και της γενίκευσης–εξειδίκευσης (specialization).

Αν και στο SIS υπάρχει τρόπος να υποβάλει κάποιος ερωτήσεις μέσω του QI (Query Interface) δεν υπάρχει μια υψηλού επιπέδου ερωτηματική γλώσσα. Το κενό αυτό φιλοδοξεί να αναπληρώσει η TELQUEL. Αν και δεν είναι ιδιαίτερα πλούσια γλώσσα, παραμένει αρκετά εκφραστική και ευέλικτη αφού υποστηρίζει τόσο όλα τα σχεσιακά χαρακτηριστικά όσο και αρκετές λειτουργίες μεταβατικής κλειστότητας.

Η υλοποίηση της TELQUEL βασίζεται στη μετατροπή μιας ερώτησης TELQUEL σε μια ακολουθία κλήσεων QI. Για να είναι εφικτή η υλοποίηση της γλώσσας με αυτό τον τρόπο αναγκαστήκαμε να χωρίσουμε τη διαδικασία της μετάφρασης σε 2 φάσεις.

Στην πρώτη φάση της μετάφρασης μετατρέπεται η ερώτηση σε μια ισοδύναμή της στη λογική γλώσσα Datalog. Στη δεύτερη φάση επιτελείται ο μετασχηματισμός από Datalog σε QI.

Η εργασία επικεντρώνεται περισσότερο:

1. Στη προσπάθεια για την επέκταση του QI ώστε να υποστηρίζει σχέσεις και στις δυσκολίες που υπάρχουν στην υποστήριξη σχέσεων από ένα σύστημα που υποστηρίζει σύνολα και
2. Τη μετατροπή μιας ερώτησης από μια δηλωτική γλώσσα όπως είναι η Datalog, σε μια διαδικαστική όπως είναι το QI, κατά τη δεύτερη φάση.

Παρουσιάζουμε ακόμη τη σύνταξη και τη σημασιολογία της γλώσσας αλλά και τον τρόπο με τον οποίο επιτελείται η πρώτη φάση της μετάφρασης.

Επόπτης: Πάνος Κωνσταντόπουλος, Καθηγητής
Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Επιβλέπουσα: Αναστασία Αναλυτή, Εντεταλμένη Ερευνήτρια
Ινστιτούτο Πληροφορικής,
Ίδρυμα Τεχνολογίας και Έρευνας

TELQUEL: A query language for SIS

John Zidianakis

MASTER OF SCIENCE THESIS

Department of Computer Science
University of Crete

ABSTRACT

SIS (Semantic Index SYSTEM) can be viewed as an Object Oriented Database Management System (OODBMS) that is based on the knowledge representation language TELOS. TELOS is used both for schema declaration (DDL) and data manipulation (DML) and is based on three abstraction mechanisms: classification, attribution and specialization.

Although there is a way to submit queries in SIS through QI (Query Interface), the lack of a high level query language is more than obvious. TELQUEL remedies this situation. TELQUEL is a query language with an SQL–like syntax. Though it is not very rich, it remains expressive and flexible supporting not only relational algebra operations but also transitive closure operations.

The implementation of TELQUEL is based on the transformation of a query in a sequence of QI calls. To achieve this goal we had to break down the compilation into two phases. In the first phase we transform the query to a DATALOG program. On second phase we transform the DATALOG program to a sequence of QI calls.

In our work special emphasis is given to:

1. The extension of QI so that it can support not only sets, but also relations, and
2. the translation of a query expressed in the declarative language DATALOG to a sequence of QI calls.

We also present the syntax and semantics of the language, as well as the way the first phase is performed.

Advisor: Panos Constantopoulos, Professor
Department of Computer Science, University of Crete

Supervisor: Anastasia Analyti, Associate Researcher
Institute of Computer Science,
Foundation for Research and Technology – Hellas

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΕΥΧΑΡΙΣΤΙΕΣ

Ο γράφων θα ήθελε να ευχαριστήσει την ερευνήτρια Αναστασία Αναλυτή για τη καθοδήγηση της και τις συμβουλές της στην ολοκλήρωση της παρούσας εργασίας που χωρίς τη βοήθεια της δεν θα ήταν εφικτή.

Ευχαριστίες ανήκουν δικαιωματικά και στον καθηγητή μου Πάνο Κωσταντόπουλο του οποίου η υπομονή επανειλημμένως δοκιμάστηκε.

Ακόμη ευχαριστιών δικαιούνται ο Βασίλης Χριστοφίδης, η Σταυρούλα Κιζλαρίδου και ο Χρήστος Γεωργής για τις πληροφορίες που πρόσφεραν στην τεχνολογία των ερωτηματικών γλωσσών αλλά και στα ενδότερα του SIS.

Κεφάλαιο 1

ΕΙΣΑΓΩΓΗ

1.Τα οντοκεντρικά μοντέλα

Η ανάγκη χρήσης βάσεων δεδομένων για τη διαχείριση μεγάλων ποσοτήτων πληροφορίας είναι πλέον στις μέρες μας δεδομένη. Ο τελικός, βέβαια, σκοπός αποθήκευσης μιας πληροφορίας είναι η εύρεση και ανάκτηση της όποτε αυτό κριθεί σκόπιμο. Προκειμένου η ανάκτηση ενός επιθυμητού συνόλου πληροφοριών μέσα από ένα πολύ μεγαλύτερο σύνολο να είναι εφικτή θα πρέπει τα δεδομένα να είναι οργανωμένα με τον καλύτερο δυνατό τρόπο. Επίσης θα πρέπει να υπάρχουν εύχρηστα εργαλεία που να επιτρέπουν την ανάκτηση ενός συγκριτικά μικρού τμήματος του συνόλου της αποθηκευμένης πληροφορίας με βάση τα εκάστοτε κριτήρια της επιλογής μας.

1.1 Τα χαρακτηριστικά του οντοκεντρικού μοντέλου

Όσον αφορά το πρώτο ζήτημα της οργάνωσης των δεδομένων έχουν προταθεί διάφοροι τρόποι–μοντέλα για αυτό το σκοπό. Μεταξύ αυτών των 'μοντέλων δεδομένων', όπως ονομάζονται βρίσκονται το ιεραρχικό, το δικτυωτό, το σχεσιακό και το οντοκεντρικό. Πέρα του ότι είναι το πιο σύγχρονο, το οντοκεντρικό μοντέλο θεωρείται και σαν το πιο ισχυρό και ευέλικτο από όσα έχουν παρουσιαστεί μέχρι σήμερα.

Το οντοκεντρικό μοντέλο αντλεί τη δύναμή του από την ικανότητά του να διαχειρίζεται αντικείμενα. Για να εξασφαλίζει την αποδοτική διαχείριση των αντικειμένων, το μοντέλο χρησιμοποιεί ένα σύνολο από μηχανισμούς [Βασ97], [Kim].

- Την απόδοση *ταυτότητας* (OID) σε κάθε αντικείμενο. Συνήθως η ταυτότητα είναι ένας αριθμός ή και αναγνωριστικό που χαρακτηρίζει το αντικείμενο σε σχέση με όλα τα υπόλοιπα που υπάρχουν στην βάση.
- Την οργάνωση των αντικειμένων με σημασιολογική ομοιότητα σε σύνολα που ονομάζονται *κλάσεις* και που στα περισσότερα συστήματα είναι με την σειρά τους και αυτές αντικείμενα. Τα αντικείμενα που απαρτίζουν την κλάση ονομάζονται περιπτώσεις αυτής.
- Την ιεράρχηση των κλάσεων μέσω της σχέσης *κλάση –υποκλάση* που δηλώνει ότι τα αντικείμενα που ανήκουν στην υποκλάση ανήκουν και στην κλάση. Αυτός ο μηχανισμός δίνει το δικαίωμα στις περιπτώσεις της υποκλάσης να αποκτούν αυτόματα τα χαρακτηριστικά που έχουν οι περιπτώσεις της κλάσης Με άλλα λόγια *κληρονομούν* τα χαρακτηριστικά της γενικότερης κλάσης.
- Δυνατότητα ενσωμάτωσης κώδικα και δεδομένων σε ένα αντικείμενο. Οι ρουτίνες που καθορίζουν τη λειτουργία του αντικειμένου ονομάζονται *μέθοδοι* και αποτελούν αναπόσπαστο κομμάτι του μαζί με την πληροφορία για την κατάσταση του αντικειμένου (τα δεδομένα).
- Δημιουργία πολύπλοκων και σύνθετων γενικά αντικειμένων. Το μοντέλο συνήθως παρέχει μερικούς βασικούς τύπους (literals) όπως π.χ. ακεραίους, συμβολοσειρές (strings) κ.λ.π. και μερικούς κατασκευαστές (constructors) όπως οι 'set of ‹type›', 'array of ‹type›' κ.α. που δίνουν τη δυνατότητα από απλούστερα αντικείμενα να κατασκευάζονται συνθετότερα.

Οι παραπάνω μηχανισμοί δίνουν μεγαλύτερη εκφραστικότητα στο οντοκεντρικό μοντέλο σε σχέση με τους ανταγωνιστές του και κυρίως το σχεσιακό μοντέλο που είναι ο βασικός του αντίπαλος.

Το οντοκεντρικό μοντέλο μπορεί να μοντελοποιήσει περισσότερες καταστάσεις του πραγματικού κόσμου και κυρίως με μεγαλύτερη

ευκολία από ότι το σχεσιακό μοντέλο. Επιπλέον το οντοκεντρικό μοντέλο προσφέρει περισσότερες δυνατότητες για επαναχρησιμοποίηση του κώδικα λόγω του μηχανισμού κληρονομικότητας που διαθέτει.

Φυσικά, το οντοκεντρικό μοντέλο δεν διαθέτει μόνο πλεονεκτήματα έναντι του σχεσιακού. Η απουσία κοινά αποδεκτού φορμαλισμού όπως ισχύει για το σχεσιακό μοντέλο [Codd] καθώς και η αυξημένη πολυπλοκότητα του είναι τα βασικά μειονεκτήματα του. Ακόμη, η απουσία φορμαλισμού και άλγεβρας δημιουργεί παραπέρα προβλήματα στην βελτιστοποίηση των ερωτήσεων προς την βάση [ODMG93].

1.2 Οι οντοκεντρικές ερωτηματικές γλώσσες

Όπως προαναφέρθηκε ο λόγος που αποθηκεύουμε την πληροφορία σε βάσεις δεδομένων είναι για να μπορούμε αργότερα με γρήγορο, εύκολο και ασφαλή τρόπο να μπορούμε να ανακτήσουμε (εν γένει να διαχειριστούμε) τμήμα αυτής, με βάση, κριτήρια της επιλογής μας. Ο ρόλος των ερωτηματικών γλωσσών (query languages) στην ανάκτηση πληροφορίας είναι κυρίαρχος. Αν και έχουν παρουσιαστεί αρκετές προτάσεις η σύνταξη που χρησιμοποιήθηκε από την ερωτηματική γλώσσα SQL [SQL] για σχεσιακά συστήματα είναι αυτή που είναι ευρύτερα γνωστή και παρόμοια της υιοθετήθηκε και σε πληθώρα οντοκεντρικών συστημάτων όπως στην O2 [Βασ97], την CQL++ [Dar] και το Gemstone [Βασ97]. Επιπλέον παρόμοια σύνταξη ακολουθεί και η ερωτηματική γλώσσα OQL [ODMG93] η οποία αποτελεί το standard για οντοκεντρικά συστήματα διαχείρισης βάσεων δεδομένων που προτείνεται από το ODMG. Θεωρείται λοιπόν λογική αν όχι επιβεβλημένη η χρήση μια σύνταξης παρόμοιας της SQL στην ερωτηματική γλώσσα που περιγράφουμε σ' αυτή την εργασία.

Πέρα όμως από τη σύνταξη οι οντοκεντρικές ερωτηματικές γλώσσες οφείλουν να εκμεταλλεύονται και τα ιδιαίτερα χαρακτηριστικά των οντοκεντρικών συστημάτων μέσα στα οποία υφίστανται. Μια λοιπόν

βασική ικανότητα ενός οντοκεντρικού συστήματος είναι να μπορεί να διασχίζει σύνθετες δομές [Βασ97] άρα, θα πρέπει η γλώσσα να επιτρέπει κάτι τέτοιο. Προς τούτο η γλώσσα θα πρέπει να υποστηρίζει τον τελεστή '.' (π.χ. john.address.street). Οι εκφράσεις που σχηματίζονται με αυτό τον τρόπο λέγονται *εκφράσεις μονοπατιού* (path expressions). Λογικό είναι τέτοιες εκφράσεις να μπορούν να εμφανιστούν οπουδήποτε εμφανίζονται αντικείμενα.

Ένα άλλο χαρακτηριστικό που πρέπει να εκμεταλλεύεται η γλώσσα είναι η ιεραρχία των κλάσεων. Εάν μια ερώτηση εφαρμόζεται πάνω σε μια κλάση θα πρέπει αυτόματα να εφαρμόζεται και στις υποκλάσεις της. Η δυνατότητα όμως να περιοριστούμε στη δοθείσα κλάση δεν θα πρέπει να αποκλείεται. Επιπρόσθετα μια οντοκεντρική ερωτηματική γλώσσα θα πρέπει να υποστηρίζει [Βασ97]:

- Συνδέσεις (joins). Οι συνδέσεις είναι ευρύτερα γνωστές λόγω της SQL. Επειδή υπάρχουν ερωτήσεις στις οποίες είναι απαραίτητες αποτελεί πλεονέκτημα για την γλώσσα να τις υποστηρίζει.

- Αναδρομή. Τα οντοκεντρικά συστήματα έχουν περισσότερο ανάγκη από τα σχεσιακά από την υποστήριξη αναδρομικών ερωτήσεων. Δεδομένου ότι η SQL δεν υποστηρίζει αναδρομή η επέκταση της γλώσσας είναι αναγκαία.

Τέλος να σημειωθεί ότι μερικές ερωτηματικές γλώσσες, όπως η SQL, επιτρέπουν την ενημέρωση της βάσης (updates) ενώ άλλες όχι [Βασ97].

1.3 Το SIS ως οντοκεντρικό μοντέλο

Το SIS (Semantic Index System) [Anal], [Const94], [Const95] αποτελεί ένα ολοκληρωμένο σύστημα διαχείρισης οντοκεντρικών βάσεων δεδομένων. Χαρακτηρίζεται σαν "σύστημα αποθήκευσης πληροφορίας υψηλών επιδόσεων" και διαθέτει προσαρμοζόμενη επαφή χρήσης (user interface). Προορίζεται για την καταγραφή

δεδομένων που έχουν μεγάλη αλληλοσχέτιση μεταξύ τους, εξελίσσονται στο χρόνο και παρουσιάζουν μεγάλη ποικιλομορφία. Το SIS περιλαμβάνει σημαντικό μέρος της γλώσσας παράστασης γνώσεων Telos [Myl90], αποκλείοντας τις δηλώσεις λογικών κανόνων [SISTELOS]. Η SIS–TELOS αναλαμβάνει το ρόλο της DDL όσο και της DML για το σύστημα. Στα αντικείμενα της δεν υποστηρίζει μεθόδους. Υπάρχει όμως διαρκής (persistent) αποθήκευση των αντικειμένων και ομαδοποίηση αυτών (clustering) κατά την αποθήκευσή τους. Υπάρχει ακόμη πρωτόκολλο για τη διεξαγωγή δοσοληψιών (transactions) και μηχανισμοί caching. Τέλος υπάρχει μια εφαρμογή και ένα API (QI–PQI) που επιτρέπει την επερώτηση της βάσης με χρήση ερωτήσεων σχετικά χαμηλού επιπέδου. Θα περιγράψουμε εν συντομία την Telos για να μπορέσουμε στην συνέχεια να κατανοήσουμε και την ερωτηματική γλώσσα που θα παρουσιάσουμε. Πλήρης περιγραφή της μπορεί να βρεθεί στο [Myl90].

Στην Telos τα δεδομένα οργανώνονται μέσω τριών μηχανισμών: του μηχανισμού ταξινόμησης (IN), απόδοσης γνωρίσματος (attribute) και γενίκευσης–εξειδίκευσης (ISA). Περιγράφουμε τον καθένα ξεχωριστά:

1.3.1 Μηχανισμός ταξινόμησης

Τα αντικείμενα στην Telos, όπως και σε κάθε οντοκεντρικό σύστημα ανήκουν σε κλάσεις. Επειδή στην TELOS και οι κλάσεις είναι αντικείμενα, ανήκουν και αυτές με την σειρά τους σε *μετακλάσεις* κ.ο.κ, η παραπάνω κατάσταση όπου κάθε αντικείμενο αποτελεί περίπτωση μιας κλάσης αποτελεί απαίτηση της γλώσσας. Η ιεραρχία των κλάσεων που δημιουργείται με αυτό τον τρόπο ξεκινάει από τα Tokens (αντικείμενα που δεν είναι κλάσεις, δεν έχουν περιπτώσεις) συνεχίζει με την S_Class, ακολουθεί η M1_Class και για την παρούσα υλοποίηση φτάνουμε μέχρι την M4_Class. Κάθε κλάση από τις παραπάνω χαρακτηρίζεται από ένα ακέραιο αριθμό που ονομάζεται στάθμη και αυξάνει κατά ένα κάθε φορά που ανεβαίνουμε στην ιεραρχία. Πέρα από αυτές τις κλάσεις υπάρχουν και οι κλάσεις συστήματος Object, Attribute και Individual που ονομάζονται *ω*–

κλάσεις γιατί έχουν περιπτώσεις τους αντικείμενα με διαφορετικές στάθμες. Στην κλάση Object βρίσκονται όλα τα αντικείμενα της βάσης. Στην κλάση Attribute βρίσκονται τα αντικείμενα που είναι γνωρίσματα, δηλώνουν δηλαδή μια σχέση μεταξύ 2 άλλων αντικειμένων. Στην κλάση Individual βρίσκονται όσα αντικείμενα δεν είναι περιπτώσεις της κλάσης Attribute.

Είναι δυνατόν ένα αντικείμενο να δηλωθεί ως περίπτωση σε παραπάνω από μια κλάσεις αρκεί να έχουν όλες την ίδια στάθμη.

1.3.2 Απόδοση γνωρίσματος

Τα γνωρίσματα χρησιμοποιούνται για να δηλώσουν μια σχέση μεταξύ 2 αντικείμενων. Από τα δύο αντικείμενα εκείνο που έχει το γνώρισμα ονομάζεται *αφετηρία* του γνωρίσματος (from–value) ενώ το αντικείμενο που προσδιορίζεται από το γνώρισμα ονομάζεται τιμή (to value). Περιορισμός που υπάρχει είναι το γνώρισμα ως αντικείμενο να μην έχει στάθμη μικρότερη της αφετηρίας και της τιμής του. Φυσικά αφού και το ίδιο το γνώρισμα είναι αντικείμενο επιτρέπεται να έχει με τη σειρά του άλλα γνωρίσματα.

1.3.3 Γενίκευση – εξειδίκευση (ISA)

Ο μηχανισμός αυτός υλοποιεί την έννοια της κλάσης–υποκλάσης όπως περιγράφηκε παραπάνω. Χαρακτηριστικά υλοποιείται η κληρονομικότητα των γνωρισμάτων ενώ επιτρέπεται και η πολλαπλή κληρονομικότητα [Myl90].

1.4 Η TELQUEL

Με δεδομένη την ύπαρξη του QI για την υποβολή ερωτήσεων σε βάσεις SIS άλλα και τη δυσχρηστία του (έχει 'νοοτροπία' παρόμοια της γλώσσας μηχανής) είναι απαραίτητη η υλοποίηση μιας ερωτηματικής γλώσσας υψηλού επιπέδου για το SIS. Η TELQUEL έρχεται να συμπληρώσει αυτό το κενό. Με σύνταξη που μοιάζει στην SQL και δηλωτικό χαρακτήρα, αν και δεν έχει την εκφραστική ισχύ που έχει το QI, θα προτιμηθεί από τον απλό χρήστη.

Η ιδιομορφία που παρουσιάζεται στην υλοποίηση της γλώσσας προκύπτει από την επιθυμία μας να εκμεταλλευτούμε το ήδη υπάρχον

QI. Εάν δε προκύψουν σημεία όπου το QI αδυνατεί να σχηματίσει μια ισοδύναμη ερώτηση με κάποια που υποβάλλεται στην TELQUEL επιλέγουμε να επεκτείνουμε το QI κατάλληλα έτσι ώστε κάθε ερώτηση σε TELQUEL να έχει μια ισοδύναμη σε QI. Επομένως ο σκοπός της παρούσας εργασίας είναι διττός: πρώτον να επεκτείνουμε κατάλληλα το QI και δεύτερον να υλοποιήσουμε έναν parser που να μετατρέπει τις ερωτήσεις της TELQUEL σε ισοδύναμες ερωτήσεις σε QI.

Με βάση αυτό τον προσανατολισμό καταλαβαίνουμε ότι η TELQUEL είναι μάλλον ένας μεταφραστής (compiler), παρά ένας διερμηνευτής (interpreter). Η 'γλώσσα μηχανής' (QI) στην οποία θα μεταφράζονται οι ερωτήσεις είναι διαδικαστική (procedural) σε αντίθεση με την TELQUEL που είναι δηλωτική γλώσσα (declarative). Η διαφορά αυτή στην φιλοσοφία των 'γλωσσών' δημιουργεί δυσκολίες στην μετάφραση και μας ανάγκασε να την χωρίσουμε σε δύο φάσεις. Επιπλέον οι περιορισμένοι τύποι δεδομένων που υποστηρίζει το QI αποτελούν μια παραπάνω δυσκολία. Απ' την άλλη μεριά η ιδιαίτερη μορφή του SIS και ειδικά της TELOS όπου σχεδόν οτιδήποτε υπάρχει στη βάση χαρακτηρίζεται ως αντικείμενο επιτρέπει ως ένα βαθμό την ομοιόμορφη αντιμετώπιση των περιεχομένων της βάσης. Έτσι η έλλειψη τύπων χαρακτηρίζεται μόνο δυσκολία της μετάφρασης και δεν αποτελεί (παρότι υφίσταται) έλλειψη και στη γλώσσα.

Σε σύγκριση με το πρότυπο ODMG (OQL), η TELQUEL δεν είναι πλήρης. Π.χ. της λείπουν δηλώσεις όπως 'group by' και 'having', αθροιστικές συναρτήσεις (aggregate functions) κ.α. Πάρα ταύτα παραμένει ισχυρή γλώσσα με οντοκεντρικά χαρακτηριστικά όπως οι εκφράσεις μονοπατιών τα οποία παρουσιάζει με 'σχεσιακό στυλ' αφού τα αποτελέσματα μιας ερώτησης είναι πάντα σχέσεις. Επιπρόσθετα αν και σε σχέση με το QI οι αναδρομικές ικανότητες είναι περιορισμένες εξακολουθούν να επιτρέπουν μεγάλη γκάμα αναδρομικών ερωτήσεων που θα καλύψει αρκετές ανάγκες.

Επιπλέον οι δυνατότητες για συνδυασμό ερωτήσεων (ένωση, τομή, διαφορά) αλλά και φωλιασμένων υποερωτήσεων προσθέτει στην ευελιξία της και την φέρνει συντακτικά ακόμη πιο κοντά στην SQL.

Στα μειονεκτήματα της γλώσσας βρίσκεται η αδυναμία να κατασκευάσει καινούργιους τύπους από ήδη υπάρχοντες και πολύ περισσότερο να ενημερώσει την βάση (updates). Στις αδυναμίες της εντάσσεται και έλλειψη υποστήριξης όψεων (views) αλλά λόγω της παρουσίας του 'GAIN' [GAIN] η αδυναμία αυτή δεν αναμένεται να γίνει ιδιαίτερα αισθητή.

Στα επόμενα κεφάλαια θα περιγράψουμε τόσο την ίδια την γλώσσα όσο και τον τρόπο που υλοποιήθηκε. Θα ξεκινήσουμε από την περιγραφή της γλώσσας, θα συνεχίσουμε με την υλοποίηση του parser και στη συνέχεια θα αναφέρουμε τις επεκτάσεις που έγιναν στο Ql προκειμένου να είναι πλήρες και να μπορούν να μεταφραστούν σ' αυτό όλες οι δηλώσεις της γλώσσας. Ακολούθως θα περιγράψουμε την καθ' αυτό μετάφραση δηλαδή το back-end του parser και θα ολοκληρώσουμε με συμπεράσματα και μελλοντικές επεκτάσεις.

Κεφάλαιο 2

ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΓΛΩΣΣΑΣ

2. Εισαγωγικά

Οι ερωτήσεις (queries) στην TELQUEL σχηματίζονται εφαρμόζοντας τις βασικές συνολοθεωρητικές πράξεις (ένωση, τομή, διαφορά) στην 'βασική μορφή' ερώτησης. Η βασική μορφή ερώτησης μοιάζει στην σύνταξη με την τυπική μορφή μιας SQL ερώτησης, έχει δηλαδή την μορφή 'SELECT...FROM...WHERE...'. Οι ομοιότητες όμως με την SQL δεν σταματούν εδώ. Η TELQUEL παρέχει μια "σχεσιακή" παρουσίαση ενός οντοκεντρικού μοντέλου δεδομένων όπως είναι το SIS. Τα αποτελέσματα των ερωτήσεων είναι σχέσεις (relations) και υποστηρίζονται οι βασικοί τελεστές της σχεσιακής άλγεβρας: επιλογή (select), προβολή (project) και καρτεσιανό γινόμενο (product).

Πέρα όμως από τις γνωστές δυνατότητες του σχεσιακού μοντέλου η TELQUEL εκμεταλλεύεται και τα οντοκεντρικά χαρακτηριστικά του SIS. Έτσι οι συνθήκες που 'φιλτράρουν' τα δεδομένα μπορούν να περιέχουν *path expressions,* να επιλέγουν δεδομένα με βάση τις υποκλάσεις τους (subclasses) ή τις υπερκλάσεις (superclasses) τους ή ακόμη τις μετακλάσεις τους και τις περιπτώσεις τους. Τα *path expressions* δίνουν την δυνατότητα να γίνει επιλογή των δεδομένων με γνώμονα το ποια γνωρίσματα έχουν (τα δεδομένα) ή/και ποιες είναι οι τιμές αυτών των γνωρισμάτων. Επιπλέον δίνουν την δυνατότητα για μεταβατικές (transitive) ερωτήσεις αφού είναι εφικτό, για παράδειγμα, να ζητήσει κανείς τα δεδομένα που κάτω από μια συγκεκριμένη κατηγορία (δηλ. κλάση γνωρισμάτων) συνδέονται σε ένα ή περισσότερα βήματα με ένα άλλο δεδομένο.

Στην συνέχεια θα περιγράψουμε με μεγαλύτερη λεπτομέρεια την TELQUEL και τη σημασιολογία της. Η περιγραφή μας βασίζεται στην σχεδίαση της γλώσσας όπως δίδεται από το [TEL]. Εν πρώτοις θα εξετάσουμε την 'βασική μορφή' ερώτησης SELECT....FROM....WHERE.... . Θα αναλύσουμε το τι ρόλο επιτελεί κάθε τμήμα μιας τέτοιας δήλωσης δίνοντας περισσότερο βάρος στο WHERE τμήμα. Ακολούθως θα αναφέρουμε τους τελεστές με τους οποίους μπορούμε να συνδυάσουμε τις βασικές ερωτήσεις και τέλος θα παραθέσουμε τη γραμματική της γλώσσας.

2.1 Η Βασική SFW ερώτηση

2.1.1 Συνολική δομή του SFW

Η δήλωση SELECT... FROM... WHERE... αποτελείται φυσικά από τρία τμήματα. Στο πρώτο τμήμα, το SELECT, γίνεται η δήλωση των μεταβλητών της συγκεκριμένης ερώτησης. Με κάθε καινούργια *SFW* δήλωση δημιουργείται και ένα καινούργιο scope (= περιοχή προγράμματος–ερώτησης στην οποία επιτρέπεται να αναφέρεται μια μεταβλητή). Στο SELECT τμήμα της SFW αναφέρονται οι μεταβλητές που ανήκουν σε αυτό το scope. Σε αντιστοιχία με την σημασιολογία της SQL οι μεταβλητές αυτές είναι εκείνες πάνω στις οποίες θα επιστραφεί το αποτέλεσμα της ερώτησης.

Στο δεύτερο τμήμα της δήλωσης ορίζουμε τα πεδία τιμών (range) κάθε μεταβλητής. Αυτό μπορεί να γίνει είτε άμεσα, δηλώνοντας την κλάση της οποίας το σύνολο των περιπτώσεων θα αποτελέσει το πεδίο τιμών της μεταβλητής, είτε έμμεσα, ορίζοντας το πεδίο τιμών μέσω ενός φωλιασμένου (nested) *SFW*.

Τέλος, στο WHERE τμήμα της δήλωσης αναφέρουμε τις συνθήκες και περιορισμούς που πρέπει να πληρούν οι τιμές των μεταβλητών προκειμένου να συμπεριληφθούν στο αποτέλεσμα της ερώτησης. Όπως και στην SQL οι επιμέρους συνθήκες συνδυάζονται μέσω κατάλληλων τελεστών, για να σχηματίσουν τη συνθήκη που προσδιορίζει το αποτέλεσμα της ερώτησης.

2.1.2 Σύνταξη του SELECT

Όπως προαναφέρθηκε στο τμήμα αυτό της δήλωσης αναφέρονται οι μεταβλητές πάνω στις οποίες θα επιστραφεί το αποτέλεσμα της ερώτησης. Ο τρόπος που γίνεται αυτή η αναφορά έχει την παρακάτω σύνταξη:

SELECT ‹ΟΝΟΜΑ ΜΕΤ/ΤΗΣ› , ‹ΟΝΟΜΑ ΜΕΤ/ΤΗΣ› , ...

Ως όνομα μεταβλητής μπορεί να είναι οποιοδήποτε συμβολοσειρά η οποία ξεκινάει με γράμμα και συνεχίζει με μια ακολουθία γραμμάτων και αριθμών. Η ίδια μεταβλητή είναι δυνατόν να εμφανίζεται περισσότερο από μια φορά. Αυτό σημαίνει ότι στην σχέση–αποτέλεσμα οι τιμές που θα πάρει θα εμφανιστούν στις αντίστοιχες στήλες στις οποίες δηλώθηκε. Για παράδειγμα αν γράψουμε:

SELECT X,Y,X,Z,X

Η πρώτη, η τρίτη και η πέμπτη στήλη της σχέσης–αποτέλεσμα θα είναι ίδιες.

Παρατήρηση: η χρήση κεφαλαίων ή πεζών τόσο στις μεταβλητές όσο και στις λέξεις κλειδιά (keywords) έχει σημασία. Έτσι η μεταβλητή Χ είναι διαφορετική από την x ενώ οι λέξεις κλειδιά θα πρέπει να πληκτρολογούνται με έναν από τους εξής τρεις τρόπους:

- Αποκλειστικά με κεφαλαίους χαρακτήρες.
- Αποκλειστικά με πεζούς χαρακτήρες.
- Τον πρώτο χαρακτήρα κεφαλαίο και τους υπόλοιπους πεζούς.

2.1.3 Σύνταξη του FROM

Με το FROM περιορίζουμε το σύνολο των δυνατών τιμών που μπορεί να πάρει μια μεταβλητή σε ένα αρχικό σύνολο. Τελικά από αυτές τις τιμές πολύ λιγότερες μπορεί να συμπεριληφθούν στο αποτέλεσμα της ερώτησης λόγω της επιπλέον συνθήκης που δηλώνεται (αν δηλώνεται) στο WHERE. Χρειαζόμαστε όμως για λόγους ασφαλείας (safety) να περιορίσουμε κάθε μεταβλητή σε ένα αρχικό σύνολο.

Ο τρόπος που συντάσσεται το FROM έχει ως ακολούθως:

FROM μετ/τη$_1$ **IN** σύνολο$_1$, μετ/τη$_2$ **IN** σύνολο$_2$, ...

Η κάθε μεταβλητή που αναφέρεται στο FROM μπορεί να είναι κάποια από αυτές που αναφέρθηκαν στο SELECT, ή μια καινούργια, βοηθητική μεταβλητή. Στην δεύτερη περίπτωση γίνεται ταυτόχρονα και δήλωση της βοηθητικής μεταβλητής.

Οι τρόποι που επιτρέπει η γλώσσα για την περιγραφή συνόλων είναι τρεις:

1. Μέσω μιας φωλιασμένης ερώτησης. Στην περίπτωση που η φωλιασμένη ερώτηση επιστρέφει μια σχέση με περισσότερες από μια στήλες, η μεταβλητή θα πάρει τιμές από την πρώτη στήλη. Με αυτό τον τρόπο επιτυγχάνεται η ατομικότητα (atomicity) στις τιμές των μεταβλητών.

2. Μέσω μιας σταθεράς. Η σταθερά αυτή μπορεί να είναι είτε η ταυτότητα κάποιου αντικειμένου (το λογικό του όνομα για την ακρίβεια) είτε κάποια πρωταρχική (primitive) σταθερά τύπου ακεραίου ή πραγματικού αριθμού ή ακόμα συμβολοσειράς (string). Φυσικά, νόημα έχει μόνο η πρώτη περίπτωση.

3. Μέσω μιας μεταβλητής. Π.χ *FROM Z IN S_Class, X IN Z*. Στην περίπτωση αυτή η μεταβλητή θα πρέπει να έχει πιο μπροστά δηλωθεί. Αυτό σημαίνει ό,τι είτε θα πρέπει να την έχουμε συμπεριλάβει στη λίστα μεταβλητών που αναφέρονται στο SELECT, είτε – όπως στο παράδειγμα – να έχει προηγηθεί δήλωση που να ορίζει από πιο σύνολο θα πάρει τιμές. Στο παράδειγμά μας η μεταβλητή X θα έχει ως πεδίο τιμών την ένωση όλων των περιπτώσεων των κλάσεων που προκύπτουν από κάθε επιτρεπτή τιμή της Z.

Τέλος, αναφέρουμε ότι είναι δυνατόν να περιορίσουμε μια μεταβλητή σε δύο ή και περισσότερα σύνολα. Να γράψουμε π.χ.:

FROM x IN File, x IN Database.

Σε αυτή την περίπτωση η μεταβλητή θα πάρει τιμές από την τομή των συνόλων.

2.1.4 Σύνταξη του WHERE

Η χρήση του WHERE έγκειται στο να ορίσει μια συνθήκη–περιορισμό που πρέπει να ικανοποιούν οι εκάστοτε τιμές των μεταβλητών προκειμένου να συμπεριληφθούν στο αποτέλεσμα. Αν και η λειτουργία είναι αντίστοιχη με αυτή της SQL η διατύπωση των συνθηκών έχει περισσότερες διαφορές από ό,τι κοινά σημεία. Εξετάζουμε λοιπόν τη σύνταξη των συνθηκών αυτών ξεκινώντας από μια συνολική περιγραφή τους.

2.1.4.1 Συνολική περιγραφή των συνθηκών

Η γενική συνθήκη που ακολουθεί τη λέξη κλειδί WHERE σχηματίζεται από άλλες απλούστερες οι οποίες συνδέονται μεταξύ τους με τους γνωστούς τελεστές σύζευξης και διάζευξης (and, or). Είναι επιπλέον δυνατό να 'αντιστραφεί' η λειτουργία μιας συνθήκης χρησιμοποιώντας τον τελεστή άρνησης (not).

Οι απλούστερες συνθήκες μπορεί με την σειρά τους να αναλύονται παραπέρα χρησιμοποιώντας πάλι τους ίδιους τελεστές, είτε να ανήκουν στην κατηγορία των 'βασικών συνθηκών'. Υπάρχουν 5 είδη βασικών συνθηκών:

- οι εκφράσεις μονοπατιών (path expressions).
- οι συγκρίσεις ρ_1 op ρ_2 όπου op ένα από τα <,<=,=,>=,>
- οι δηλώσεις υποκλάσης (αντικείμενο$_1$ ISA αντικείμενο$_2$).
- οι δηλώσεις περίπτωσης (αντικείμενο$_1$ IN σύνολο).
- οι υπαρξιακές δηλώσεις (EXISTS (σύνολο)).

Το πρώτο είδος εισάγει την απαίτηση ορισμένα αντικείμενα (σταθερές ή μεταβλητές) να έχουν συγκεκριμένα γνωρίσματα και ενδεχομένως οι τιμές (το values) αυτών των γνωρισμάτων να είναι επίσης συγκεκριμένες. Η δεύτερη κατηγορία αξιώνει να ισχύει η συγκεκριμένη σχέση (ισότητα, ανισότητα, ομοιότητα) μεταξύ των ορισμάτων της. Οι δηλώσεις υποκλάσης και περίπτωσης επαληθεύονται όταν το πρώτο αντικείμενο είναι υποκλάση ή περίπτωση του δευτέρου, αντίστοιχα. Εάν κάποιο/α από τα αντικείμενα είναι μεταβλητή τότε εξετάζονται μία προς μία οι δυνατές τιμές του και απορρίπτονται εκείνες που δεν επαληθεύουν τη συνθήκη. Οι υπαρξιακές δηλώσεις επαληθεύονται όταν το 'σύνολο' δεν είναι κενό.

Έπειτα από αυτή τη συνοπτική περιγραφή παρουσιάζουμε με περισσότερη λεπτομέρεια τόσο τους τελεστές όσο και τις βασικές συνθήκες, ξεκινώντας από τους πρώτους.

2.1.4.2 Τελεστές: and, or, not

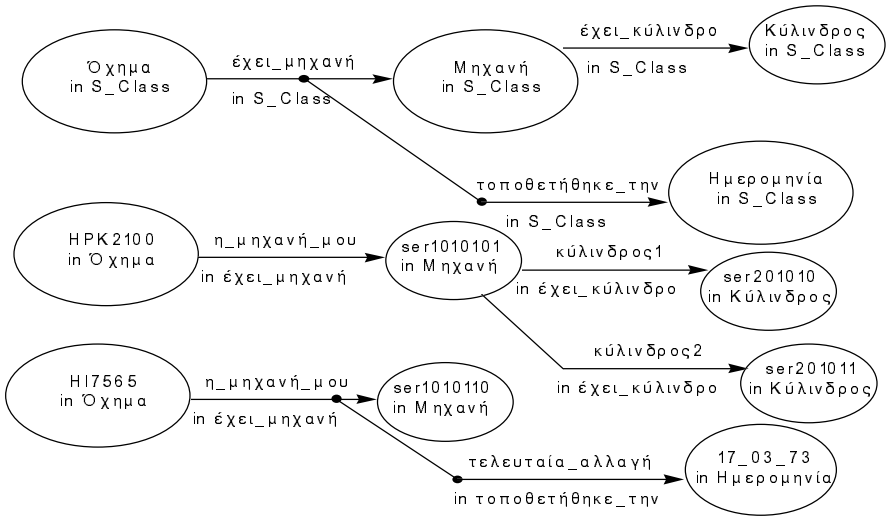
Οι τελεστές είναι ευρύτατα γνωστοί από τις γλώσσες προγραμματισμού. Η λειτουργία τους στην TELQUEL είναι αντίστοιχη με αυτή που έχουν σε άλλες ερωτηματικές γλώσσες. Η σύνταξη τους επίσης είναι αντίστοιχη. Οι **and** και **or** χρησιμοποιούν ενθεματικό (infix) συμβολισμό ενώ ο **not** σαν εναδικός (unary) χρησιμοποιεί προθεματικό (prefix). Το **not** έχει την υψηλότερη προτεραιότητα και το **or** την χαμηλότερη. Όλοι προσεταιρίζουν αριστερά.

2.1.4.3 Εκφράσεις μονοπατιών (path expressions)

Η απλούστερη μορφή που μπορεί να έχει μια έκφραση μονοπατιού είναι η ακόλουθη:

αντικείμενο.κατηγορία$_1$.κατηγορία$_2$....κατηγορία$_v$

Η δήλωση επαληθεύεται εφόσον το αντικείμενο έχει τουλάχιστον ένα γνώρισμα που ανήκει (είναι περίπτωση) στην κατηγορία$_1$ και η τιμή αυτού του γνωρίσματος (το-value$_1$) έχει γνώρισμα που ανήκει στην κατηγορία$_2$ του οποίου η τιμή πρέπει με την σειρά της να έχει ένα γνώρισμα που να ανήκει στην κατηγορία$_3$ κ.ο.κ. Για παράδειγμα θεωρείστε το παρακάτω μοντέλο:



Σχήμα 1

Αν στο παραπάνω μοντέλο εφαρμοστεί η ερώτηση:

SELECT X

FROM X IN Όχημα

WHERE X.{Όχημα έχει__μηχανή}.{Μηχανή έχει__κύλινδρο}

Θα επιστρέψει μόνο το HPK2100 διότι από τις δυνατές τιμές του X (HPK2100 και HI7565 αφού X IN Όχημα) μόνο το HPK2100 έχει γνώρισμα που είναι περίπτωση του 'έχει_μηχανή' – το γνώρισμα 'η_μηχανή_μου' – και η τιμή αυτού του γνωρίσματος –ser1O1O1O1– έχει με την σειρά της γνώρισμα που είναι περίπτωση του 'έχει_κύλινδρο' (για την ακρίβεια έχει 2 τέτοια γνωρίσματα: το 'κύλινδρος1' και το 'κύλινδρος2')

Όπως γίνεται αντιληπτό από το παραπάνω παράδειγμα ένας τρόπος για να προσδιορίσει κανείς μια κατηγορία είναι να δώσει το λογικό όνομα του 'from value' της κατηγορίας αυτής μαζί με το λογικό όνομα της ίδιας της κατηγορίας. Κάτι τέτοιο θεωρείται απαραίτητο για να μπορεί να διακρίνει κανείς το γνώρισμα 'η_μηχανή_μου' του αντικειμένου 'HPK2100' από το ομώνυμο γνώρισμα του αντικειμένου 'HI7565'. Ένας δεύτερος τρόπος για τον προσδιορισμό κατηγορίας είναι η χρήση μεταβλητών. Συμμετρικά η αφετηρία μιας έκφρασης μονοπατιού μπορεί να είναι σταθερά (κατ' ουσία λογικό όνομα αντικειμένου). Για παράδειγμα η ερώτηση:

SELECT X

FROM X IN S__Class

WHERE HPK2100.{Όχημα έχει__μηχανή}.X

Θα επιστρέψει ως αποτέλεσμα όλα τις περιπτώσεις της S_Class. Αυτό το μάλλον απρόσμενο αποτέλεσμα λαμβάνεται διότι όταν χρησιμοποιείται μια μεταβλητή ως κατηγορία σε έκφραση μονοπατιού η έκφραση επαληθεύεται πάντα όταν έστω και μια από τις επιτρεπτές τιμές της μεταβλητής την επαληθεύει. Δηλαδή η παραπάνω έκφραση είναι ισοδύναμη με την:

HPK2100.{Όχημα έχει__μηχανή}.v_1 OR...OR HPK2100.{Όχημα έχει__μηχανή}.v_n

Όπου $v_1, v_2, .., v_n$ είναι οι επιτρεπτές τιμές του X, δηλαδή τις περιπτώσεις της S_Class.

Γνωρίσματα με αφετηρία γνώρισμα

Μέχρι στιγμής διαχωρίζουμε τις διαδοχικές κατηγορίες στις εκφράσεις μονοπατιών με μία τελεία. Προκειμένου να αναφερθούμε σε γνωρίσματα που έχουν αφετηρία (from value) άλλα γνωρίσματα η γλώσσα δίνει την δυνατότητα να διαχωρίσουμε 2 διαδοχικές κατηγορίες με τον χαρακτήρα '^'. Όταν γίνεται τέτοιος διαχωρισμός μεταξύ της κατηγορίας_i και της κατηγορίας_{i+1} (γράψουμε δηλαδή: $\text{κατηγορία}_i\text{^κατηγορία}_{i+1}$) η έκφραση επαληθεύεται για τις περιπτώσεις της κατηγορίας_{i+1} που έχουν ως αφετηρία τις περιπτώσεις της κατηγορίας_i και όχι τις τιμές αυτών. Λόγου χάριν η ερώτηση:

SELECT X
FROM X IN Όχημα
WHERE X.{Όχημα έχει_μηχανή}^{έχει_μηχανή τοποθετήθηκε_την}

Θα επιστρέψει μόνο το Η7565 γιατί από όλα τα οχήματα (σύνολο 2) είναι το μόνο που έχει γνώρισμα που είναι περίπτωση του 'έχει_μηχανή' (η_μηχανή_μου) το οποίο με την σειρά του έχει γνώρισμα που είναι περίπτωση του 'τοποθετήθηκε_την' (τελευταία_αλλαγή).

Μετάβαση μέσω συγκεκριμένου γνωρίσματος

Στην περίπτωση που σε κάποιο βήμα του μονοπατιού δεν θέλουμε να προχωρήσουμε γενικά μέσω οποιασδήποτε περίπτωσης μιας κατηγορίας αλλά μίας συγκεκριμένης η TELQUEL το επιτρέπει. Εκείνο που χρειάζεται είναι πριν από την κατηγορία να γράψουμε το συγκεκριμένο γνώρισμα που επιθυμούμε διαχωρίζοντας το από την κατηγορία με το χαρακτήρα ':' . Π.χ. προκειμένου να αναφέρουμε στο γνώρισμα 'τελευταία_αλλαγή' θα γράφαμε:

SELECT X
FROM X IN Όχημα
WHERE X.{Όχημα έχει_μηχανή}^
{η_μηχανή_μου τελευταία αλλαγή}:{έχει_μηχανή τοποθετήθηκε_την}

Το αποτέλεσμα στην παραπάνω ερώτηση είναι πάλι 'Η7565' και αυτό γιατί επιλέξαμε το γνώρισμα από το οποίο θα 'περάσει' το μονοπάτι να είναι το ίδιο με αυτό της προηγούμενης ερώτησης.

Αντί να περιορίσουμε το μονοπάτι μέσω ενός σταθερού γνωρίσματος μπορούμε να το περιορίσουμε με την χρήση μεταβλητής όπως παρακάτω:

```
SELECT X
FROM X IN Όχημα, Y IN SELECT A
                      FROM A IN {Μηχανή έχει__κύλινδρο}
                      WHERE  A = {ser1010101 κύλινδρος1}
WHERE X.{Όχημα έχει__μηχανή}^Y:{έχει__μηχανή έχει__κύλινδρο}
```

Με την παραπάνω ερώτηση θα βρίσκαμε το όχημα που η μηχανή του έχει τον 'κύλινδροι'.

Περιορισμός της τιμής του γνωρίσματος

Όπως χρησιμοποιήσαμε σταθερές ή μεταβλητές για να περιορίσουμε τα γνωρίσματα από τα οποία θα περάσει ένα μονοπάτι μπορούμε να περιορίσουμε το μονοπάτι έτσι ώστε να περάσει από γνωρίσματα που *έχουν συγκεκριμένες τιμές*. Αν για παράδειγμα θέλαμε να βρούμε το όχημα πού έχει τη μηχανή ser1010101 θα γράφαμε:

```
SELECT X
FROM X IN Όχημα
WHERE X.{Όχημα έχει__μηχανή}[ser1010101]
```

Φυσικά μπορούμε να περιορίσουμε ταυτόχρονα τόσο τις τιμές όσο και τα ίδια τα γνωρίσματα από τα οποία θα περάσει το μονοπάτι. Π.χ.

```
SELECT X
FROM X IN Όχημα
WHERE X.{HPK2100 η__μηχανή__μου}:{Όχημα έχει__μηχανή}[ser1010101]
```

Διάσχιση με περισσότερες από μία κατηγόριες

Μέχρι τώρα σε κάθε 'βήμα' της έκφρασης μονοπατιού αναφέραμε και μία κατηγορία της οποίας περιπτώσεις θα έπρεπε να είναι όλα τα γνωρίσματα που περνούσαν από το συγκεκριμένο σημείο του μονοπατιού. Η TELQUEL επιτρέπει να αναφέρουμε σε κάθε βήμα του μονοπατιού περισσότερες από μία κατηγορίες. Έτσι αν θέλαμε να βρούμε τα άτομα που είτε εργάζονται είτε διαμένουν στην οδό Δαιδάλου θα γράφαμε:

```
SELECT X
FROM X IN Άτομο
WHERE X.({Άτομο οδός__κατοικίας}|{Άτομο οδός__εργασίας})[Δαιδάλου]
```

Αντίθετα αν θέλαμε τα άτομα που και εργάζονται και διαμένουν στην οδό Δαιδάλου θα γράφαμε:

```
SELECT X
FROM X IN Άτομο
WHERE X.({Άτομο οδός__κατοικίας}&{Άτομο οδός__εργασίας})[Δαιδάλου]
```

Όπως είναι εμφανές από τα προηγούμενα δύο παραδείγματα τις κατηγορίες των οποίων οι περιπτώσεις θα χρησιμοποιηθούν σε κάθε βήμα του μονοπατιού μπορούμε να τις διαχωρίσουμε είτε με το '|' είτε με το '&'. Στην πρώτη περίπτωση το μονοπάτι θα 'περάσει' από την ένωση των περιπτώσεων των κατηγοριών ενώ στην δεύτερη από την τομή. Μια τρίτη δυνατότητα που δίνεται είναι να αποκλείσουμε τις περιπτώσεις μιας κατηγορίας. Αν θέλουμε τα άτομα που κατοικούν στη Δαιδάλου αλλά δεν εργάζονται εκεί θα σχηματίζαμε την ερώτηση:

```
SELECT X
FROM X IN Άτομο
WHERE X.({Άτομο οδός__κατοικίας}&-{Άτομο οδός__εργασίας})[Δαιδάλου]
```

Αναζήτηση με βάση Μετακατηγορίες

Στην περίπτωση που μας ενδιαφέρουν οι περιπτώσεις των κατηγοριών που είναι κ στάθμες (αντί για μία) πιο 'κάτω' από μια κατηγορία μπορούμε να το προσδιορίσουμε αυτό στην γλώσσα. Το μόνο που χρειάζεται να γίνει είναι να εισάγουμε κ-1 χαρακτήρες ':' πριν το συγκεκριμένο βήμα του μονοπατιού. Υποθέστε φέρ' ειπείν ότι όλα τα γνωρίσματα (S_Classes) του μοντέλου που παρουσιάσαμε είναι περιπτώσεις της μετακατηγορίας 'αποτελείται_από' η οποία είναι γνώρισμα του 'αντικείμενο' με τιμή επίσης 'αντικείμενο'. Επομένως αν θέλαμε να βρούμε όλα τα αντικείμενα (σε επίπεδο token) που αποτελούνται από απλούστερα μέρη θα σχηματίζαμε την ερώτηση:

SELECT X
FROM X IN S__Class
WHERE X.:{αντικείμενο αποτελείται__από}

Να διευκρινίσουμε ότι η χρήση του ':' είναι *καθολική* για κάθε βήμα του μονοπατιού. Αυτό σημαίνει ότι δεν επιτρέπεται να έχουμε εκφράσεις του τύπου: **X.(:::κατηγορία$_1$ | κατηγορία$_2$ | :κατηγορία$_3$)** αλλά μόνο σαν **X.::(κατηγορία$_1$ | κατηγορία$_2$ | κατηγορία$_3$).**

Μεταβατικές (transitive) ή αναδρομικές ερωτήσεις

Υπάρχουν περιπτώσεις που θέλουμε σε ένα συγκεκριμένο βήμα του μονοπατιού να μην 'διασχίσουμε' μόνο ένα γνώρισμα αλλά μια ακολουθία γνωρισμάτων όπου η τιμή του ενός είναι η αφετηρία (from value) του επόμενου. Με τα μέχρι τώρα γνωστά χαρακτηριστικά της γλώσσας αν θέλαμε να βρούμε τα αφεντικά του 'Γιάννη' θα διατυπώναμε την εξής ερώτηση:

SELECT X
FROM X IN Άτομο
WHERE Γιάννης.{Άτομο έχει__αφεντικό}[X]

Τι γίνεται όμως αν θέλουμε να βρούμε όλους τους προϊσταμένους του 'Γιάννη' "όσο ψηλά –στην ιεραρχία– και αν βρίσκονται"; Η TELQUEL

επιτρέπει με μια μικρή παραλλαγή της προηγούμενης ερώτησης να απαντήσουμε και σε αυτό το ερώτημα:

```
SELECT X
FROM X IN Άτομο
WHERE Γιάννης.{Άτομο έχει_αφεντικό}+[X]
```

Η χρήση του '+' επιτρέπει στην έκφραση μονοπατιού να επαληθευθεί όχι μόνο για τις περιπτώσεις του 'έχει_αφεντικό' που έχουν ως αφετηρία (from_value) τον 'Γιάννη' αλλά και εκείνες τις περιπτώσεις που η αφετηρία τους είναι η τιμή μιας περίπτωσης που ήδη έχουμε διαπιστώσει ότι επαληθεύει την έκφραση.

Πέρα από το χαρακτήρα '+' η TELQUEL υποστηρίζει και τους '*' και '?' για μεταβατικές-αναδρομικές ερωτήσεις. Ο '*' λειτουργεί όπως ο '+' με την διαφορά ότι η έκφραση επαληθεύεται ακόμη και χωρίς να διασχιστεί περίπτωση της κατηγορίας. Π.χ. στην ερώτηση;

```
SELECT X
FROM X IN Άτομο
WHERE Γιάννης.{Άτομο έχει_αφεντικό}*[X]
```

Το αποτέλεσμα θα είναι το ίδιο με προηγουμένως προσαυξημένο με το αντικείμενο 'Γιάννης'.

Ακόμη η χρήση του '?' μοιάζει με την περίπτωση που δεν έχουμε αναδρομική ερώτηση (*ακριβώς μία* περίπτωση διασχίζεται) με την διαφορά ότι το 'ακριβώς ένα' γίνεται 'το πολύ ένα'.

Τέλος σημειώνουμε ότι η χρήση των '+', '*' και '?' είναι όπως η χρήση του συμβόλου μετακατηγορίας ':' *καθολική*. Εφαρμόζεται δηλαδή ταυτόχρονα σε όλες τις κατηγορίες ενός βήματος.

2.1.4.4 Δηλώσεις υποκλάσης και περίπτωσης

Περιγράψαμε ήδη τη λειτουργία τους, τώρα θα πούμε πως μπορούμε να περιγράψουμε τα αντικείμενα που είναι ορίσματα τους. Πέρα από μεταβλητές τα αντικείμενα μπορεί να είναι και σταθερές. ερωτήσεις

στις οποίες είναι απαραίτητες. Επίσης επιτρέπεται και τα δύο ορίσματα να είναι σταθερές. Δηλαδή στην περίπτωση της δήλωσης περίπτωσης είναι δυνατόν το 'σύνολο' να περιγραφεί από το λογικό όνομα μιας κλάσης. Πέρα από λογικά ονόματα αντικειμένων μπορούμε και εδώ να χρησιμοποιήσουμε ακεραίους, πραγματικούς και συμβολοσειρές. Όπως εύκολα παρατηρεί κανείς η χρήση τέτοιων σταθερών αρκετές φορές δεν έχει νόημα αφού ποτέ δεν επαληθεύει την δήλωση. Π.χ η

123.4 IN M1_Class είναι πάντα ψευδής.

2.1.4.5 Συγκρίσεις (‹,‹=,=,›=,›)

Η σύνταξη των δηλώσεων σύγκρισης έχει την μορφή:

μεταβλητή/σταθερά τελ. σύγκρισης μεταβλητή/σταθερά

Οι τελεστές που υποστηρίζονται είναι οι: ‹,‹=,=,›= και ›. Προκειμένου ένα ζεύγος τιμών–τελεστέων να καταστήσει αληθή μια σύγκριση θα πρέπει οι τελεστέοι να είναι του ιδίου τύπου και να ισχύει μεταξύ τους η σχέση διάταξης που ορίζει η σύγκριση. Π.χ. για δυο ακέραιες τιμές και την σύγκριση '‹' θα πρέπει φυσικά η πρώτη τιμή να είναι μικρότερη της δεύτερης. Δεδομένου ότι δεν υπάρχει διάταξη μεταξύ αντικειμένων ο μόνος τελεστής που ενδέχεται σε σύγκριση 2 αντικειμένων να επαληθευτεί είναι ο =. Σε όλες τις άλλες περιπτώσεις η σύγκριση δεν επαληθεύεται. Για παράδειγμα η ερώτηση:

SELECT X
FROM X IN Άτομο, Y IN Άτομο
WHERE X.{Άτομο έχει__αφεντικό}*[Y] and X ‹ Y

Θα επιστρέφει πάντα το κενό σύνολο διότι οι 2 τελεστέοι (μεταβλητές X και Y) της σύγκρισης παίρνουν τιμές αντικείμενα τα οποία δεν είναι μεταξύ τους διατεταγμένα.

2.2 Τελεστές μεταξύ SFW ερωτήσεων

Οι τελεστές που υποστηρίζονται είναι 3: η ένωση, η τομή και η διαφορά. Τους παρουσιάζουμε εν συντομία.

2.2.1 Ένωση

Προκειμένου να λάβουμε την ένωση των αποτελεσμάτων δύο SFW ερωτήσεων θα πρέπει να το δηλώσουμε με τον τελεστή 'union' χρησιμοποιώντας ενθεματικό (infix) συμβολισμό ανάμεσα στις δύο ερωτήσεις. Επιπλέον απαίτηση είναι οι δύο SFW ερωτήσεις να έχουν στο αποτέλεσμά τους τον ίδιο αριθμό στηλών (arity). Ο τελεστής union προσεταιρίζει αριστερά. Π.χ.:

SELECT X	SELECT X
FROM X IN Άτομο	FROM X IN ¨Άτομο
union	union
SELECT X	SELECT X,Y
FROM X IN Ζώο	FROM X IN Ζώο,Y IN Ψάρι
Σωστό	Λάθος

2.2.2 Τομή

Όσα ισχύουν για την ένωση ισχύουν και για τη τομή. Η μόνη διαφορά είναι το όνομα του τελεστή (intersect) και το αποτέλεσμα που παράγει.

2.2.3 Διαφορά

Όσα ισχύουν για την ένωση ισχύουν και για τη διαφορά. Διαφέρει μόνο το όνομα (minus) και η λειτουργία του τελεστή.

Αξίζει να επισημάνουμε ότι αντίθετα με τους τελεστές and, or και not της συνθήκης του where, οι union, intersect και minus έχουν την ίδια προτεραιότητα.

2.3 Η γραμματική της TELQUEL

Παραθέτουμε τη γραμματική της TELQUEL χρησιμοποιώντας τη γλώσσα περιγραφής γραμματικών του yacc. Τα τερματικά σύμβολα

αναγράφονται με κεφαλαία και το αρχικό σύμβολο της γραμματικής είναι η κεφαλή (head) του πρώτου κανόνα.

```
query:        set_expr

set_expr:      set_expr UNION set_expr
             | set_expr INTERSECT set_expr
             | set_expr MINUS set_expr
             | '(' set_expr ')'
             | sfw_expr
             | atom

atom:         ID
             | '{' ID ID '}'
             | INTEGER
             | REAL
             | STRING

sfw_expr:      SELECT var_list FROM range_list WHERE bool
             | SELECT var_list FROM range_list

var_list:     ID
             | var_list ',' ID

range_list:    range_spec
             | range_list ',' range_spec

range_spec:    ID IN set_expr

bool:         bool AND bool
             | bool OR bool
             | NOT bool
             | '(' bool ')'
             | EXISTS '(' set_expr ')'
             | atom IN set_expr
             | atom ISA atom
             | atom COMPARISON atom
             | path_expr

path_expr:     atom "." path_elt
             | path_expr "." path_elt
             | path_expr "^" path_elt

path_elt:      attr_spec
             | atom ':' attr_spec
             | attr_spec '[' atom ']'
             | atom ':' attr_spec '[' atom ']'
```

```
attr_spec:      nr_attr_spec
                | nr_attr_spec '?'
                | nr_attr_spec '+'
                | nr_attr_spec '*'

nr_attr_spec:   '(' attr_conj ')'
                | '(' attr_disj ')'
                | '(' nr_attr_spec ')'
                | ':' nr_attr_spec
                | attr_literal

attr_conj:      attr_literal '&' attr_literal
                | attr_conj '&' attr_literal

attr_disj:      attr_literal '|' attr_literal
                | attr_disj '|' attr_literal

attr_literal:   ID
                | '{' ID ID '}'


                | '-' attr_literal
```

Κεφαλαίο 3

ΥΛΟΠΟΙΗΣΗ: FRONT END

3. Συνολική περιγραφή

Η μετάφραση μιας ερώτησης TELQUEL σε ένα πρόγραμμα QI περνάει από δύο στάδια. Στο πρώτο στάδιο γίνεται ένας αρχικός μετασχηματισμός της ερώτησης σε μια ενδιάμεση γλώσσα που είναι μια παραλλαγή της γλώσσας DATALOG. Ο μετασχηματισμός αυτός δεν είναι πραγματικός με την έννοια ότι δεν παράγεται κάποιο αρχείο κειμένου (εκτός αν ζητηθεί ρητά). Αυτό που γίνεται πραγματικά είναι η δημιουργία μιας εσωτερικής δενδρικής δομής. Οι κόμβοι του δένδρου αυτού έχουν 1–1 αντιστοιχία με μια δήλωση στο ισοδύναμο πρόγραμμα DATALOG. Αυτό έχει ως αποτέλεσμα ο μετασχηματισμός της εσωτερικής δομής σε πρόγραμμα DATALOG να είναι απλός και σχεδόν άμεσος. Η κυριότερη συνεισφορά όμως αυτής της δομής είναι το γεγονός ότι καθιστά εφικτό το μετασχηματισμό της σε ένα ισοδύναμο πρόγραμμα QI. Αυτός ο μετασχηματισμός αποτελεί και το δεύτερο στάδιο της μετάφρασης.

Στο παρόν κεφάλαιο θα ασχοληθούμε με το πρώτο στάδιο της μετάφρασης. Αφού παρουσιάσουμε τις απαραίτητες γνώσεις για την κατανόηση των προγραμμάτων DATALOG θα περιγράψουμε το μηχανισμό για την παραγωγή τέτοιων προγραμμάτων από ερωτήσεις TELQUEL, η επινόηση του μηχανισμού αυτού οφείλεται στους Gerd Hillebrand και Πολύβιο Κλιμαθιανάκη ενώ η περιγραφή μας στηρίζεται στο [TEL] . Θα ολοκληρώσουμε την παρούσα ενότητα περιγράφοντας την εσωτερική δομή που χρησιμοποιεί ο parser για να αναπαραστήσει το πρόγραμμα DATALOG. Κατ' ουσία θα περιγράψουμε μόνο ένα κόμβο της δενδρικής δομής μια και αυτό είναι αρκετό.

3.1 DATALOG: Βασικές γνώσεις

Η DATALOG είναι μια ερωτηματική γλώσσα κατάλληλη για το σχεσιακό μοντέλο που μοιάζει στη λογική γλώσσα prolog και στην γενική της μορφή επιτρέπει αναδρομικές ερωτήσεις. Η γλώσσα έχει ως βασικές ιδιότητες τον μη διαδικαστικό χαρακτήρα και το ότι είναι βασισμένη στην πρωτοβάθμια λογική (first order logic). Η παραλλαγή της DATALOG που θα περιγράψουμε εδώ διαφέρει από τη κλασσική γλώσσα στο ότι έχουν αφαιρεθεί τα αναδρομικά χαρακτηριστικά (παρά ταύτα είναι δυνατό να απαντηθούν μερικές αναδρομικές ερωτήσεις) και στο πλήθος και είδος των βασικών σχέσεων (EDBs = extensional databases) που υποστηρίζει.

3.1.1 Γενική σύνταξη

Σαν ερωτηματική γλώσσα για το σχεσιακό μοντέλο η DATALOG διαχειρίζεται σχέσεις. Χρησιμοποιεί απλούστερες σχέσεις για να κατασκευάσει συνθετότερες μέσω ερωτήσεων που η ορολογία της γλώσσας αποκαλεί 'DATALOG προγράμματα'. Με βάση αυτή την επισήμανση οι σχέσεις στην DATALOG διακρίνονται σε 2 κατηγορίες:

- Στις βασικές σχέσεις (EDBs = extensional databases). Στο σχεσιακό μοντέλο αυτές είναι οι σχέσεις που ορίζει το σχήμα τις βάσης. Στο SIS τα πράγματα είναι διαφορετικά. Πρώτη διαφορά είναι ότι το πλήθος των βασικών σχέσεων δεν είναι πεπερασμένο. Αντ' αυτού υπάρχει πεπερασμένο πλήθος *κατηγοριών* βασικών σχέσεων. Όλες οι βασικές σχέσεις έχουν έναν, λίγο ως πολύ, ορατό τρόπο να προκύψουν από την SIS βάση μέσω QI (υπο)ερωτήσεων. Ανάλογα με την κατηγορία στην οποία ανήκει μια βασική σχέση επιλέγεται και ο κατάλληλος αλγόριθμος που θα παράγει τον αντίστοιχο QI κώδικα στο δεύτερο στάδιο της μετάφρασης.

- Στις παράγωγες σχέσεις (IDBs = intentional databases). Πρόκειται για ενδιάμεσες κυρίως σχέσεις που προκύπτουν από τον υπολογισμό υποερωτήσεων και βοηθούν στο σχηματισμό του τελικού αποτελέσματος.

Ένα πρόγραμμα DATALOG αποτελείται από μια διατεταγμένη ακολουθία δηλώσεων $k_1,k_2,...,k_v$ που ονομάζονται *κανόνες*. Κάθε κανόνας αποτελείται από 2 τμήματα: την *κεφαλή* (head) και το *σώμα* (body). Κάθε κανόνας ορίζει μια καινούργια IDB σχέση. Το όνομα της σχέσης μαζί το ποιες και πόσες στήλες έχει αυτή η σχέση αναφέρεται στην κεφαλή του κανόνα. Στο σώμα του κανόνα αναγράφονται οι σχέσεις (IDBs και EDBs) από τις οποίες προκύπτει η νέα σχέση. Δεδομένου ότι δεν μας ενδιαφέρουν τα αναδρομικά χαρακτηριστικά της γλώσσας στο σώμα του κανόνα k_i ($1 <= i <= n$) επιτρέπεται να αναφέρονται μόνο EDBs και IDBs που ορίστηκαν από τους προηγούμενους κανόνες $k_1,k_2,...k_{i-1}$. Ακόμη, σε καμία περίπτωση δεν επιτρέπεται η σχέση που αναφέρεται στην κεφαλή του κανόνα να αναφέρεται και στο σώμα του ίδιου κανόνα.

Υπάρχει η δυνατότητα μία IDB σχέση να ορίζεται με τη βοήθεια περισσοτέρων του ενός κανόνες. Αν συμβαίνει κάτι τέτοιο οι κανόνες που ορίζουν την ίδια σχέση θα πρέπει να είναι διαδοχικοί στο πρόγραμμα DATALOG, η σχέση να έχει τον ίδιο αριθμό στηλών σε όλους του κανόνες, ενώ το αποτέλεσμα της προκύπτει από την ένωση των αποτελεσμάτων του κάθε κανόνα.

3.1.2 Δηλώσεις EDB

Υπάρχουν 4 κατηγορίες EDB σχέσεων στην DATALOG για το SIS:

- Οι σχέσεις IN. Αυτές έχουν την μορφή IN(X,Y) όπου X και Y είναι μεταβλητές ή σταθερές. Αντιπροσωπεύουν μια σχέση που έχει τόσες στήλες όσες είναι και οι μεταβλητές που περιέχονται στα ορίσματα της (2, 1 ή 0). Θεωρώντας ότι σε κάθε όρισμα αντιστοιχεί μια στήλη, κάθε πλειάδα θα σχηματιζόταν από ζεύγη τιμών όπου το πρώτο μέλος του ζεύγους θα έπαιρνε τιμές από το πεδίο τιμών του πρώτου ορίσματος (στην περίπτωση που το όρισμα είναι σταθερά η τιμή είναι μία) ενώ το δεύτερο μέλος από το πεδίο τιμών του δεύτερου ορίσματος. Επιπλέον, επιτρέπονται μόνο

πλειάδες που το πρώτο μέλος είναι περίπτωση του δευτέρου. Επειδή όμως δεν αντιστοιχεί μια στήλη για κάθε όρισμα αλλά μόνο για τα ορίσματα που είναι μεταβλητές, οι στήλες που αντιστοιχούν σε ορίσματα που είναι σταθερές δεν προβάλλονται στο τελικό αποτέλεσμα.

- Οι σχέσεις ISA. Αυτές έχουν την μορφή ISA(X,Y) όπου X και Y είναι επίσης μεταβλητές ή σταθερές. Δημιουργούν σχέση με τρόπο αντίστοιχο με αυτό που σχηματίζονται οι IN σχέσεις. Η διαφορά τους έγκειται στο ότι εδώ το πρώτο μέλος μιας πλειάδας της 'γενικής σχέσης', της σχέσης δηλαδή που υπάρχει πριν αφαιρέσουμε τις στήλες που αντιστοιχούν σε ορίσματα σταθερές, είναι υποκλάση και όχι περίπτωση του δευτέρου.

- Οι σχέσεις Attribute. Περιγράφονται με την σύνταξη: **A.[meta](&|')([-]category$_1$,...,[-]category$_n$)[+|*|?](X,L,Y)**. Επιστρέφουν μια σχέση κατ' αρχήν πάνω στο σχήμα ‹X,L,Y›. Σε τελικό στάδιο κρατούνται μόνο οι στήλες που αντιστοιχούν σε μεταβλητές. Π.χ. αν το όρισμα X είναι σταθερά η στήλη του δεν θα συμπεριληφθεί στο τελικό αποτέλεσμα. Ο τρόπος ορισμού της σχέσης έχει ως ακολούθως:

1. Εν πρώτοις στη στήλη που αντιστοιχεί σε κάθε όρισμα επιτρέπονται μόνο τιμές από το πεδίο τιμών (range) αυτού του ορίσματος.
2. Η στήλη L περιέχει γνωρίσματα (links). Σε κάθε πλειάδα της σχέσης στη στήλη L βρίσκεται ένα γνώρισμα, στην στήλη X η αφετηρία (from value) του γνωρίσματος και στην στήλη Y η τιμή (to value) του γνωρίσματος.
3. Στην περίπτωση που έχουμε σύζευξη (προσδιορίζεται από το &) θα επιλεγούν από την στήλη L μόνο εκείνα τα γνωρίσματα που είναι περιπτώσεις <u>όλων</u> των 'καταφατικών' (χωρίς πρόθεμα '-') κατηγοριών (categories) αλλά όχι αυτών με άρνηση. Εφόσον κάποια κατηγορία ορίζεται μέσω μεταβλητής θεωρούμε ότι το γνώρισμα είναι περίπτωση αυτής της κατηγορίας αν είναι περίπτωση μίας από τις επιτρεπτές τιμές της.

4. Στην περίπτωση που έχουμε διάζευξη (προσδιορίζεται από το 'I') θα επιλεγούν από τη στήλη L μόνο εκείνα τα γνωρίσματα που είναι περιπτώσεις τουλάχιστον μίας των 'καταφατικών' (χωρίς πρόθεμα '–') κατηγοριών (categories) ή δεν είναι περιπτώσεις κάποιας από τις κατηγορίες με άρνηση.

5. Αναφέραμε προηγουμένως ότι τα γνωρίσματα της στήλης L θα πρέπει να είναι περιπτώσεις των κατηγοριών που αναφέρονται στην Attribute σχέση. Εφόσον το meta, το οποίο είναι ακέραιος αριθμός, είναι θετικό λαμβάνουμε τις περιπτώσεις οι οποίες βρίσκονται meta+1 στάθμες πιο 'κάτω' από την στάθμη των κατηγοριών. Π.χ. αν το meta ισούται με 1 και έχουμε διάζευξη θα πάρουμε τις περιπτώσεις όλων των κατηγοριών και από κάθε μία από αυτές θα πάρουμε πάλι τις περιπτώσεις τους και αυτές οι τελευταίες θα είναι που θα τοποθετηθούν στη στήλη L.

6. Στην περίπτωση που προσδιορίζεται κάποιος από τους τελεστές μεταβατικότητας (*, ?, +) προστίθενται επιπλέον γνωρίσματα στην στήλη L. Φυσικά οι κατάλληλες τιμές συμπληρώνονται στις στήλες X και Y για να σχηματιστεί η εκάστοτε πλειάδα. Αν ο τελεστής είναι ο '+' τότε για κάθε γνώρισμα με τιμή v που υπάρχει στη στήλη L προστίθενται όλα τα γνωρίσματα που έχουν αφετηρία (from value) το v. Θεωρείται αυτονόητο ότι το νέο αυτό γνώρισμα θα πρέπει επιπλέον να πληρεί όλους τους προηγούμενους περιορισμούς. Αυτό συνεπάγεται ότι θα πρέπει να είναι περίπτωση των κατηγοριών και η τιμή του να βρίσκεται μέσα στο πεδίο τιμών του ορίσματος Y. Ο τελεστής '*' παράγει την ίδια σχέση με τον '+' επαυξημένη με τις πλειάδες της μορφής: ‹a,NULL,a› όπου το a ανήκει στην τομή των πεδίων τιμών των ορισμάτων X και Y. Με τις ίδιες πλειάδες προσαυξάνει και ο τελεστής '?' την σχέση που θα προέκυπτε αν δεν υπήρχε καθόλου τελεστής μεταβατικότητας.

- Οι σχέσεις σύγκρισης. Περιγράφονται ως COMP_OP(X,Y) όπου COMP_OP ένα από τα ‹,‹=,=,›=,›. Επιστρέφουν μια σχέση κατ' αρχήν πάνω στο σχήμα ‹X,Y› από το οποίο τελικά κρατούνται μόνο οι στήλες που αντιστοιχούν σε μεταβλητές. Όπως και στις προηγούμενες κατηγορίες EDB σχέσεων οι στήλες παίρνουν τιμές από τα πεδία τιμών των αντίστοιχων ορισμάτων. Το επιπλέον 'φιλτράρισμα' που γίνεται είναι διαγραφή των πλειάδων για τις οποίες η εκάστοτε σύγκριση που ορίζεται από το COMP_OP μεταξύ της πρώτης και της δεύτερης συντεταγμένης της πλειάδας είναι ψευδής. Π.χ. η ‹=(4,Y) θα επιστρέψει μια σχέση με μια στήλη μόνο που θα περιέχει όσους αριθμούς από το πεδίο τιμών το ορίσματος Y είναι μεγαλύτεροι του 4.

Σημείωση: Μέχρι τώρα αναφέραμε ότι οι στήλες που αντιστοιχούν σε ένα όρισμα παίρνουν τιμές από το πεδίο τιμών αυτού του ορίσματος. Στην περίπτωση που το πεδίο τιμών ενός ορίσματος δεν γίνεται γνωστό στην EDB τότε αυτόματα θεωρείται πριν τον υπολογισμό της EDB ότι το πεδίο τιμών του γνωρίσματος είναι όλη η βάση. Μετά τον υπολογισμό της EDB το πεδίο τιμών περιορίζεται στις τιμές που υπάρχουν στη στήλη που αντιστοιχεί στο γνώρισμα. Εξαίρεση αποτελούν οι κατηγορίες στις EDB γνωρισμάτων και τα ορίσματα των EDB σύγκρισης όπου πάντα θα πρέπει το πεδίο τιμών να προσδιορίζεται..

3.1.3 Δηλώσεις IDB

Όπως ήδη έχουμε επισημάνει κάθε IDB ορίζεται μέσω ενός η περισσοτέρων διαδοχικών κανόνων σε ένα πρόγραμμα DATALOG. Στην κεφαλή κάθε κανόνα αναφέρουμε το όνομα της IDB μαζί με το σχήμα της. Το σχήμα της IDB προσδιορίζεται με μια ακολουθία μεταβλητών και σταθερών που χωρίζεται σε δυο τμήματα: τις δεσμευμένες (bound) και τις ελεύθερες (free) μεταβλητές. Αν και η χρήση του όρου 'μεταβλητές' είναι καταχρηστική, την υιοθετούμε γιατί συνήθως αυτή χρησιμοποιείται.

Έτσι στον κανόνα:

R(X,Y; Z) :- IN(X,Z), IN(Z,M2__Class), ISA(X,Y)

R είναι το όνομα της νέας IDB, X και Y είναι οι δεσμευμένες μεταβλητές της και Z είναι η μόνη ελεύθερη μεταβλητή της. Ο παραπάνω κανόνας δημιουργεί μια σχέση με 3 στήλες. Στην πρώτη στήλη κάθε πλειάδας υπάρχει μια 'απλή' κλάση (S_Class), στην δεύτερη υπάρχει η υπερκλάση της και στην τρίτη η μετακλάση της οποίας είναι περίπτωση.

Ο ρόλος των δεσμευμένων μεταβλητών είναι να ορίσει το 'πραγματικό σχήμα' του κανόνα. Αν δηλαδή ο παραπάνω κανόνας ήταν ο τελευταίος σε ένα πρόγραμμα DATALOG και άρα αυτός που θα όριζε τη σχέση–αποτέλεσμα η σχέση αυτή θα αποτελείτο από 2 στήλες. Αν ο κανόνας βρίσκεται στην αρχή ή τη μέση του προγράμματος οι κανόνες που έπονται αυτού θα πρέπει να αναφέρονται σε αυτόν προσδιορίζοντας και τις δεσμευμένες και τις ελεύθερες μεταβλητές. Θα μπορούσαμε λόγου χάριν να ορίσουμε την IDB S συναρτήσει της R ως εξής:

S(X;):- R(X,Y;object)

Με αυτό τον τρόπο θα πάρουμε όλες τις περιπτώσεις της μετακλάσης object.

Ένας επιπλέον περιορισμός που υπάρχει είναι οι μεταβλητές που εμφανίζονται στην κεφαλή του κανόνα να εμφανίζονται και στο σώμα του. Ακόμη, όταν μια IDB χρησιμοποιείται στο σώμα του κανόνα θα πρέπει να συνοδεύεται από το ίδιο πλήθος δεσμευμένων και ελεύθερων μεταβλητών με αυτό που χρησιμοποιήθηκε στον κανόνα(ες) ορισμού της.

Μια επιπλέον σύμβαση που κάνουμε για τα IDBs που ορίζονται μέσω πολλών κανόνων είναι πως αντί να γράφουμε π.χ.:

R(X;) :- S(X;Y), SS(Y;)
R(X;) :- T(X;D), TT(D;S)

γράφουμε:

RS(X;) :- S(X,Y;), SS(Y;)
RT(X;) :- T(X;D), TT(D;S)
R(X;) :- RS(X;) | RT(X;)

Η σχέση που αντιστοιχεί στην κεφαλή ενός κανόνα περιέχει τις πλειάδες εκείνες που αν η τιμή κάθε συντεταγμένης τους δοθεί στην μεταβλητή της αντίστοιχης στήλης να μπορούν να βρεθούν τιμές για τις υπόλοιπες μεταβλητές (που αναφέρονται στο σώμα αλλά όχι στην κεφαλή του κανόνα) έτσι ώστε κάθε σχέση στο σώμα του κανόνα να περιέχει την πλειάδα που σχηματίζεται με τις μεταβλητές που χρησιμοποιεί. Για παράδειγμα στον κανόνα:

R(X;) :- T(X;D), TT(D;S)

Η IDB R θα περιέχει εκείνες τις πλειάδες ‹a› έτσι ώστε αν X=a να μπορούν να βρεθούν τιμές d και s για τις μεταβλητές D και S έτσι ώστε οι πλειάδες ‹X,D›=‹a,d› και ‹D,S›=‹d,s› να ανήκουν στις σχέσεις T και TT αντίστοιχα.

Είδαμε ότι προκειμένου να σχηματιστεί ένας κανόνας αναφέρουμε στο σώμα του μια ακολουθία από EDBs (με την σύνταξη που παρουσιάσαμε παραπάνω) και IDBs χρησιμοποιώντας το σωστό αριθμό ελεύθερων και δεσμευμένων μεταβλητών. Μια επιπλέον δυνατότητα που δίνει η DATALOG είναι αντί να χρησιμοποιηθούν EDBs και IDBs αυτούσιες να αναφέρονται οι αρνήσεις τους. Π.χ αν θέλαμε να βρούμε τα οχήματα που δεν είναι φορτηγά θα γράφαμε:

R(X;) :- IN(X,Όχημα), -IN(X,Φορτηγό)

Σ' αυτή την περίπτωση θα πρέπει οι πλειάδες που σχηματίζονται με την απόδοση τιμών στις μεταβλητές να μην ανήκουν στην σχέση που συμμετέχει με την άρνησή της στον κανόνα.

3.2 Κανόνες μετασχηματισμού

Έχοντας εισάγει τις βασικές γνώσεις για τα DATALOG προγράμματα παρουσιάζουμε τον τρόπο που μια ερώτηση σε γλώσσα TELQUEL μετατρέπεται σε τέτοιο πρόγραμμα.

3.2.1 Γενική φιλοσοφία

Ο τρόπος με τον όποιο σχηματίζεται το πρόγραμμα DATALOG που αντιστοιχεί στην ερώτηση TELQUEL είναι αναδρομικός και 'συμβαδίζει' με τη γραμματική της γλώσσας. Αυτό σημαίνει ότι καθώς κάθε ερώτηση διασπάται σε απλούστερες δηλώσεις, σύμφωνα με την γραμματική της γλώσσας, ο αλγόριθμός μετασχηματισμού μεταφράζει πρώτα τις απλούστερες δηλώσεις σε DATALOG και έπειτα χρησιμοποιεί τις IDBs που προκύπτουν από αυτές για να σχηματίσει τον ή τους νέους κανόνες που θα αντιστοιχούν στην σύνθετη δήλωση. Σύμφωνα με το ανωτέρω σκεπτικό θα είναι και ο τρόπος που θα περιγράψουμε τη μετάφραση. Θα εξηγήσουμε δηλαδή πως σχηματίζεται ο κανόνας για μια σύνθετη δήλωση θεωρώντας ότι οι κανόνες για τις απλούστερες δηλώσεις που τον απαρτίζουν έχουν ήδη δημιουργηθεί και τα αντίστοιχα IDBs είναι διαθέσιμα.

3.2.2 Μετατροπή ενώσεων SFWs

Σύμφωνα με τα όσα αναφέραμε στην περιγραφή της TELQUEL είναι δυνατό να πάρουμε την ένωση δύο SFW για να σχηματίσουμε μια νέα σχέση δηλώνοντας κάτι σαν SFW_1 *union* SFW_2. Εάν η σχέση που αντιστοιχεί στην SFW_1 υπολογίζεται στην IDB $R_1(X;Y)$ και η σχέση που αντιστοιχεί στην SFW_2 στην IDB $R_2(X;Z)$ όπου X είναι το σύνολο των δεσμευμένων μεταβλητών και στους δύο κανόνες, ενώ Y και Z τα σύνολα των ελεύθερων μεταβλητών τότε η σχέση που αντιστοιχεί στην ένωση βρίσκεται στην IDB:

$R(X;Y \cup Z) :- R_1(X;Y) \mid R_2(X;Z)$

Όπου η ερμηνεία της χρήσης του 'ǀ' δόθηκε παραπάνω.

3.2.3 Μετατροπή τομών SFWs

Η μετατροπή μιας δήλωσης τομής δύο SFWs από TELQUEL κώδικα σε κανόνα DATALOG είναι παρόμοια με την μετατροπή της ένωσης με μία μόνο παραλλαγή. Αν σχηματίσουμε την τομή δύο SFWs δηλώνοντας SFW_1 intersect SFW_2 και οι IDBs που αντιστοιχούν στις δύο SFWs είναι όπως προηγουμένως οι $R_1(X;Y)$ και $R_2(X;Z)$ αντίστοιχα τότε η IDB που αντιστοιχεί στην τομή τους ορίζεται από τον κανόνα:

$R(X;Y \cup Z) :- R_1(X;Y), R_2(X;Z)$

Όπου $Y \cup Z$ συμβολίζει την ένωση των συνόλων Y και Z.

3.2.4 Μετατροπή διαφορών SFWs

Στο ίδιο πνεύμα με την ένωση και την τομή βρίσκεται και η διαφορά δύο SFWs. Έτσι ο κανόνας που ορίζει την IDB που αντιστοιχεί στην διαφορά 2 SFWs όπως δηλώνεται μέσω της *SFW_1 minus SFW_2* είναι:

$R(X;Y \cup Z) :- R_1(X;Y), -R_2(X;Z)$

Όπου $R_1(X;Y)$ είναι η IDB που αναδρομικά έχει υπολογιστεί για την SFW_1 και $R_2(X;Z)$ η αντίστοιχη IDB για την SFW_2.

3.2.5 Μετατροπή του FROM τμήματος του SFW.

Υπενθυμίζουμε ότι το From τμήμα μιας SFW έχει την μορφή:

From $μετ_1$ IN $υποερώτηση_1$, $μετ_2$ IN $υποερώτηση_2$,....,$μετ_v$ IN $υποερώτηση_v$

Για να σχηματίσουμε την IDB που αντιστοιχεί σ' αυτή τη δήλωση υποθέτουμε ότι έχουμε διαθέσιμες από αναδρομικό υπολογισμό τις IDBs για τις υποερωτήσεις #1 έως #v έστω: $R_1(X_1;Y_1), R_2(X_2;Y_2),..., R_v(X_v;Y_v)$

Εάν $X_i = \{X_{i1}, X_{i2},...,X_{iv}\}$ και $Y_i = \{Y_{i1}, Y_{i2},...,Y_{iv}\}$ για i = 1,2,...,v τότε σχηματίζουμε τους κανόνες $T_1, T_2,..., T_v$ ως εξής:

$T_i(; \{μετ_i\} \cup Y_i) :- R_i(\{μετ_i\} \cup \{X_{i2}, X_{i3},...,X_{iv}\}; Y_i)$ i=1,2,...,v

Ένας τέτοιος κανόνας ουσιαστικά μεταφράζει την $μετ_i$ IN $υποερώτηση_i$ μετονομάζοντας την μεταβλητή X_{i1} του κανόνα R_i σε $μετ_i$.

Έχοντας τώρα σχηματίσει τις T_i IDBs ορίζουμε αναδρομικά ένα νέο σύνολο κανόνων με την βοήθεια τους ως ακολούθως:

$S_1(;\mathbf{Z}_1)$:- $T_1(;\mathbf{Z}_1)$

$S_i(;\mathbf{Z}_i \text{ U } \text{Freevarsof}(S_{i-1}))$:- $T_i(;\mathbf{Z}_i)$, $S_{i-1}(;\text{Freevarsof}(S_{i-1}))$ i=2,3,...v

Όπου $\mathbf{Z}_i = \{\mu\varepsilon\tau_i\} \text{ U } \mathbf{Y}_i$ i=1,2...,v

Το IDB S_v αποτελεί τη 'μετάφραση' όλης της from δήλωσης. Πρόκειται ουσιαστικά για τη σύζευξη όλων των T_i μόνο που γίνεται εισάγοντας ένα επιπλέον IDB κάθε φορά. Το 'βάθος' στη δενδρική δομή που δημιουργείται συμπτύσσεται στο τέλος από την φάση της απλοποίησης (simplification).

3.2.6 Μετατροπή της WHERE δήλωσης

Το WHERE αποτελεί το τρίτο και τελευταίο κομμάτι μιας SFW. Μέσω αυτού διατυπώνεται η συνθήκη που πρέπει να πληρούν οι μεταβλητές της SFW μεταξύ τους. Τη μετάφραση αυτής της συνθήκης θα περιγράψουμε παρακάτω. Εκείνο που προκαταρτικά πρέπει να διευκρινίσουμε είναι ότι οι IDBs που αντιστοιχούν σε συνθήκες έχουν μόνο ελεύθερες και όχι δεσμευμένες μεταβλητές.

3.2.6.1 Μετατροπή των συνθηκών με τελεστές and, or και not

Η συνθήκη που διατυπώνεται στο WHERE μπορεί να αποτελείται από μια βασική συνθήκη όπως οι εκφράσεις μονοπατιών και οι συγκρίσεις είτε να σχηματίζεται από άλλες απλούστερες μέσω των τελεστών and, or και not. Σ' αυτή την τελευταία περίπτωση η μετάφραση είναι εντελώς ανάλογη με τη μετάφραση των συνολοθεωρητικών πράξεων μεταξύ SFWs. Αν λοιπόν η συνθήκη σχηματίζεται με τους παρακάτω τρόπους:

συνθήκη_1 and συνθήκη_2

συνθήκη_1 or συνθήκη_2

not συνθήκη_1

και οι IDBs που αντιστοιχούν στη συνθήκη$_1$ και συνθήκη$_2$ είναι αντίστοιχα οι $R_1(;Y_1)$ και $R_2(;Y_2)$ τότε η μετάφραση των παραπάνω συνθέτων συνθηκών είναι:

$S_1(;\mathbf{Y_1}\ U\ \mathbf{Y_2})$:- $R_1(;\mathbf{Y_1})$, $R_2(;\mathbf{Y_2})$ για την συνθήκη που χρησιμοποιεί and

$S_1(;\mathbf{Y_1}\ U\ \mathbf{Y_2})$:- $R_1(;\mathbf{Y_1})$ | $R_2(;\mathbf{Y_2})$ για την συνθήκη που χρησιμοποιεί or

$S_1(;\mathbf{Y_1})$:- --$R_1(;\mathbf{Y_1})$ για την συνθήκη που χρησιμοποιεί not

3.2.6.2 Μετατροπή των εκφράσεων μονοπατιών (path expressions)

Σύμφωνα με την γραμματική της γλώσσας κάθε έκφραση μονοπατιού έχει τη μορφή:

{μετ/τη ή σταθερά}.βήμα$_1${. ή |}βήμα$_2${. ή |}.....{. ή |}βήμα$_v$

Η μετάφραση μιας τέτοιας έκφρασης γίνεται ακολουθώντας τα βήματα. Ξεκινάμε παράγοντας DATALOG κώδικα για το πρώτο βήμα και ακολούθως για όλα τα υπόλοιπα. Όπως εξηγήσαμε και όταν παρουσιάσαμε την TELQUEL η σύνταξη του κάθε βήματος έχει μια από τις παρακάτω δύο μορφές:

{link:}::.....:({-}category$_1$&...&{-}category$_n$){*ή+ή?}{[tovalue]}

{link:}::.....:({-}category$_1$|...|{-}category$_n$){*ή+ή?}{[tovalue]}

Όπου ότι περικλείεται μεταξύ { και } είναι προαιρετικό.

Τώρα με βάση την παραπάνω σύνταξη ο μετασχηματισμός του κάθε βήματος είναι φανερός. Αν υποθέσουμε ότι X είναι η μεταβλητή ή η σταθερά με την οποία ξεκινάει η έκφραση μονοπατιού, τότε η μετάφραση του πρώτου βήματος ανάλογα με το ποια σύνταξη χρησιμοποιείται:

A$_1$.[meta](&)([-]category$_1$,...,[-]category$_n$)[+ή*ή?](X, link, tovalue)

A$_1$.[meta](|)([-]category$_1$,...,[-]category$_n$)[+ή*ή?](X, link, tovalue).

Όπου meta ο αριθμός των ':' πριν την '('.

Στην περίπτωση που τα link και tovalue δεν προσδιορίζονται, χρησιμοποιούνται στη θέση τους βοηθητικές μεταβλητές. Επίσης οι τελεστές μεταβατικότητας λαμβάνουν μέρος εφόσον αναφέρονται.

Για το βήμα υπ' αριθμόν i (i>1) η μετάφραση του εξαρτάται τόσο από τον τελεστή που το συνδέει με το προηγούμενο βήμα (^ ή .) όσο και από τα 2 τελευταία ορίσματα του προηγούμενου βήματος (έστω $link_{i-1}$ και $tovalue_{i-1}$). Αν ο τελεστής είναι ο ^ τότε η μετάφραση του βήματος για κάθε μια από τις δύο δυνατές συντάξεις είναι:

$\mathbf{A_i.[meta_i](\&)([-]category_1,...,[-]category_n)[+|*|?]}(link_{i-1},link_i,tovalue_i)$

$\mathbf{A_i.[meta_i](|)([-]category_1,...,[-]category_n)[+|*|?]}(link_{i-1},link_i,tovalue_i)$

Αν αντίθετα ο τελετής είναι ο '.' τότε η μετάφραση είναι:

$\mathbf{A_i.[meta_i](\&)([-]category_1,...,[-]category_n)[+|*|?]}(tovalue_{i-1},link_i,tovalue_i)$

$\mathbf{A_i.[meta_i](|)([-]category_1,...,[-]category_n)[+|*|?]}(tovalue_{i-1},link_i,tovalue_i)$

Όπως στην περίπτωση της μετάφρασης του πρώτου βήματος έτσι και τώρα:

- Το meta είναι ο αριθμός των : πριν την '('.
- Στην περίπτωση που τα $link_i$ και $tovalue_i$ δεν προσδιορίζονται, χρησιμοποιούνται στη θέση τους βοηθητικές μεταβλητές.
- Οι τελεστές μεταβατικότητας λαμβάνουν μέρος εφόσον αναφέρονται.

Η συνολική μετάφραση του μονοπατιού σχηματίζεται από τον κανόνα:

$R(;X,link_1,link_2,...,link_v,tovalue_1,tovalue_2,...,tovalue_v)$**:-**

$A_1[meta_1]....(X,link_1,tovalue_1),.....,A_v[meta_v]...(link_{v-1}/tovalue_{v-1},link_v,tovalue_v)$

Αν κάποιο από τα link ή τα tovalue ή το ίδιο το X είναι σταθερά και όχι μεταβλητή, τότε αυτό δεν συμπεριλαμβάνεται στο σύνολο των ελεύθερων μεταβλητών του κανόνα R.

3.2.6.3 Μετατροπή των δηλώσεων exists

Μια από τις βασικές συνθήκες που είναι δυνατόν να υπάρχουν στο WHERE τμήμα μιας SFW είναι η 'exists (υποερώτηση)'. Αν η μετάφραση της υποερώτησης οδηγεί στην IDB R(X;Y) τότε η μετάφραση της συνθήκης exists είναι απλώς η:

E(;Y):- R(X;Y)

3.2.6.4 Μετατροπή της μεταβλητή/σταθερά IN υποερώτηση

Μια άλλη κατηγορία βασικών συνθηκών είναι η 'μεταβλητή/σταθερά' IN υποερώτησή. Αν όπως προηγουμένως $R(X_1,X_2,...,X_m; Y)$ είναι η IDB που αντιστοιχεί στην υποερώτηση και X η σταθερά/μεταβλητή της συνθήκης τότε η μετάφραση της είναι η:

$I(;\{X\}UY) :- R(X,X_2,...,X_m;Y)$

Φυσικά όπως και στις εκφράσεις μονοπατιών αν η X είναι σταθερά, τότε δεν συμμετέχει στο σύνολο των ελεύθερων μεταβλητών της I.

3.2.6.5 Μετατροπή της μεταβλητή/σταθερά ISA μεταβλητή/σταθερά

Η τρίτη κατηγορία συνθηκών δηλώνει σχέσεις ISA μεταξύ μεταβλητών και σταθερών. Αν X είναι η πρώτη μεταβλητή–σταθερά (πριν το ISA) και Y η δεύτερη η μετάφραση αυτής της συνθήκης γίνεται με τον κανόνα:

$R(;X,Y) :- ISA(X,Y)$

Όπως και προηγουμένως η X και η Y περιλαμβάνονται στο σύνολο των ελεύθερων μεταβλητών της R μόνο αν είναι μεταβλητές.

3.2.6.6 Μετατροπή της μεταβλητή/σταθερά op μεταβλητή/σταθερά

Η τελευταία κατηγορία βασικών συνθηκών είναι αυτές που συγκρίνουν με έναν από τους κατάλληλους τελεστές (<,<=,=,>=,>) δυο μεταβλητές ή σταθερές. Εδώ θα εξηγήσουμε πως γίνεται η μετάφραση για τον τελεστή ισότητας. Η μετάφραση για τους άλλους τελεστές είναι πλήρως ανάλογη. Όπως και στην μετάφραση των ISA συνθηκών X είναι η μεταβλητή–σταθερά πριν τον τελεστή και Y αυτή μετά τον τελεστή. Η μετάφραση είναι:

$R(;X,Y) :- =(X,Y)$

Για τις ελεύθερες μεταβλητές του κανόνα ισχύει ότι και στην περίπτωση της ISA.

3.2.7 Συνολική μετατροπή της SFW

Είδαμε πως μεταφράζονται το FROM και το WHERE τμήμα μιας SFW. Τώρα θα εξετάσουμε πως, με δεδομένες τις μεταφράσεις αυτών των τμημάτων μπορούμε να μεταφράσουμε ολόκληρη την SFW. Αν λοιπόν F(;Y) είναι η IDB που προκύπτει από τη μετάφραση του FROM τμήματος και W(;Z) είναι η αντίστοιχη IDB για το WHERE, η IDB για ολόκληρη την SFW προκύπτει από τον κανόνα:

SFW(X;Y U Z):- F(;Y),W(;Z)

Όπου **X** είναι το σύνολο των μεταβλητών που αναφέρονται στο Select τμήμα της SFW.

Υπάρχει μια 'εκφυλισμένη' περίπτωση όπου μια σταθερά (ή στην περίπτωση φωλιασμένης SFW και μεταβλητή) έστω C που θεωρείται ότι είναι κλάση να υποκαθιστά μια ολόκληρη SFW. Τότε η IDB για την SFW ορίζεται από τον κανόνα:

SFW(X;C):– IN(X,C)

Το C συμπεριλαμβάνεται στις ελεύθερες μεταβλητές της SFW μόνο εφόσον είναι μεταβλητή.

3.3 Δομή αναπαράστασης

Όπως εξηγήσαμε στην αρχή του κεφαλαίου η πρώτη φάση της μετάφρασης μιας TELQUEL ερώτησης δεν οδηγεί σε κάποιο πρόγραμμα DATALOG με τη μορφή κειμένου. Δημιουργείται όμως μια δενδρική δομή που (σχεδόν) κάθε κόμβος της είναι ισοδύναμος με τον ορισμό μιας IDB. Τον κόμβο αυτής της δομής, όπως έχει οριστεί στην γλώσσα C περιγράφουμε σ' αυτή την ενότητα. Η δομή αυτή χρησιμοποιείται και για άλλους ρόλους όπως αναπαράσταση μεταβλητών, EDBs και σταθερών. Ο ορισμός της είναι ο ακόλουθος:

```
typedef struct Node {
enum Node_Type {IDB, Comparison, In, Isa, Attribute, Variable,
Object, Integer, Real, String} type;
int num_slots;
```

```
char *name;
int sign;
union Detail {
struct {enum Bool_Op {Conjunction, Disjunction} op;
        int num_bound_vars, num_free_vars, num_children;}
idb;
struct {enum Comp_Op {LT, LE, EQ, NE, GE, GT, SM, NS} op;}
comparison;
struct {enum Bool_Op op; int num_categories, meta;
        enum Transitivity_Flag {EQ1, LE1, GE0, GE1} tc;} attribute;
struct {int scope; struct Node *value, *next;} variable;
struct {SYSID value;} object;
struct {int value;} integer;
struct {float value;} real;
} detail;
struct Node **slots;
} NODE;
```

Το πεδίο type δηλώνει το ρόλο του κόμβου και ανάλογα με αυτόν χρησιμοποιείται και το ομώνυμο μέλος της union detail για να αποθηκεύσει ειδικές πληροφορίες που χρειάζεται αυτός ο κόμβος. Εξαίρεση αποτελούν οι σταθερές τύπου συμβολοσειράς (strings) όπου το μόνο που χρειάζεται είναι η τιμή τους που αποθηκεύεται στο πεδίο name. Ο πίνακας slots αποθηκεύει διαφορετικές πληροφορίες κατά περίπτωση:

- Στην περίπτωση των IN, ISA και COMPARISON EDBs δείχνει στα ορίσματα τους (X και Y).
- Στην περίπτωση της EDB Attribute δείχνει τόσο στα ορίσματα (X, L και Y) όσο και στις κατηγορίες της EDB.

- Στην περίπτωση των IDBs αποθηκεύονται οι ελεύθερες και οι δεσμευμένες μεταβλητές καθώς και οι θυγατρικοί κόμβοι του, οι IDBs δηλαδή που βρίσκονται στο σώμα του κανόνα.

Ακολούθως επεξηγούμε το ρόλο μερικών πεδίων που η λειτουργία τους ενδεχομένως να μην είναι προφανής:

1. Το πεδίο op στη struct idb διακρίνει αν έχουμε 'φυσιολογικό' κανόνα (τιμή conjunction) ή 'ψευδοκανόνα' (τιμή disjunction) δηλ. κανόνα της μορφής: $R(...):- R_1(..) \mid R_2(...) \mid ... \mid R_n(...)$.
2. Το πεδίο op στη struct comparison προσδιορίζει τον τελεστή της σύγκρισης. Η αντιστοιχία μεταξύ τιμών και τελεστών είναι: LT–›'‹', LE–›'‹=', EQ–›'=', NE–›'!=', GE–›'›=', GT–›'›', SM–›'=~', NS–›'!~'
3. Τέλος το πεδίο tc στη struct attribute προσδιορίζει τον τελεστή μεταβατικότητας της εκάστοτε *A.(...)* EDB. Η αντιστοιχία μεταξύ τιμών και τελεστών είναι: **EQ1-›'κανένας τελεστής', LE1-›'τελεστής ?', GE0-›'τελεστής *', GE1-›'τελεστής +'**.

Κεφάλαιο 4

ΥΛΟΠΟΙΗΣΗ: ΕΠΕΚΤΑΣΕΙΣ QL ΚΑΙ PQL

4. Περί QL και PQL

Το SIS πριν την TELQUEL εξυπηρετούσε τις ανάγκες για επερώτηση της βάσης μέσω του QL για περιστασιακές (ad hoc) ερωτήσεις. Το PQL παρέχει τις ίδιες δυνατότητες που παρέχει και το QL αλλά σε προγραμματιστικό περιβάλλον (C/C++). Θα περιγράψουμε πολύ συνοπτικά την λειτουργία του QL. Για μια πλήρη περιγραφή τόσο του QL όσο και το PQL δείτε αντίστοιχα τα [PQL98] και [QL98]. Οι περισσότερες ερωτήσεις του QL εφαρμόζονται σε ένα 'τρέχον αντικείμενο' της βάσης και τοποθετούν τα αποτελέσματα σε μια κατασκευασμένη εκ των προτέρων δομή συνόλου στην οποία τοποθετούν άλλα αντικείμενα. Τα περιεχόμενα του συνόλου–απάντηση εξαρτώνται από το ποια είναι η ερώτηση. Π.χ. αν κατασκευάσουμε ένα καινούργιο κενό σύνολο με την εντολή **sgn** (= set get new) και έπειτα θέσουμε ως τρέχον αντικείμενο την κλάση S_Class (εισάγοντας την εντολή '**scn S_Class**', scn = set current node) μπορούμε στο σύνολο απάντηση να πάρουμε τις περιπτώσεις της με την εντολή **gai**. Τα αποτελέσματα μιας ερώτησης αποθηκεύονται πάντα στο 'τρέχον σύνολο'. Η **sgn** ουσιαστικά κατασκευάζει ένα κενό σύνολο και το θέτει ως τρέχον.

Το QL δίνει την δυνατότητα διαχείρισης συνόλων. Με την εντολή **stor ‹όνομα›** αποθηκεύεται στη μνήμη το τρέχον σύνολο με την ονομασία 'όνομα'. Η αποθήκευση του τρέχοντος συνόλου είναι απαραίτητη μια και με την επόμενη ερώτηση στην βάση του SIS τα περιεχόμενα του θα αλλάξουν.

Επιπλέον δίνεται η δυνατότητα σχηματισμού ένωσης, τομής και διαφοράς μεταξύ του τρέχοντος συνόλου και ενός συνόλου που αποθηκεύτηκε στη μνήμη πιο πριν. (Εντολές **su, si, sd ‹όνομα›**). Ακόμη είναι εφικτό οι ερωτήσεις να μην απευθύνονται στη βάση αλλά σε ένα σύνολο. Σ' αυτή την περίπτωση το αποτέλεσμα προκύπτει εφαρμόζοντας την ερώτηση σε κάθε στοιχείο του συνόλου και σχηματίζοντας την ένωση των επιμέρους αποτελεσμάτων. Ο προσδιορισμός της δομής στη οποία θα εφαρμοστεί η επόμενη ερώτηση (τρέχον αντικείμενο ή ένα σύνολο) επιτελείται μέσω της εντολής **apon ‹όνομα›** (apon = apply on) όπου το όνομα χαρακτηρίζει το σύνολο στο οποίο θα εφαρμοστεί η ερώτηση ή έχει την τιμή **'cn'** για να δείξει στο τρέχον αντικείμενο.

Θα ολοκληρώσουμε την σύντομη εισαγωγή μας στο PQL (που σε καμιά περίπτωση δεν μπορεί να χαρακτηριστεί πλήρης) με την περιγραφή μιας ομάδας εντολών που θα χρησιμοποιήσουμε για να υλοποιήσουμε τα αναδρομικά χαρακτηριστικά της TELQUEL. Η ομάδα αυτή έχει ως βασικό της μέλος την εντολή tal (=Traverse All Links). Η λειτουργία αυτής της εντολής εξαρτάται σε μεγάλο βαθμό από 5 άλλες εντολές που θα περιγράψουμε παρακάτω αλλά σε γενικές γραμμές είναι η εξής: ξεκινώντας από το τρέχον αντικείμενο (ή ένα αντικείμενο από το τρέχον σύνολο εφόσον η ερώτηση εφαρμόζεται σε σύνολο) βρίσκουμε όλα τα γνωρίσματα του. Από αυτά τα γνωρίσματα παρακρατούμε μόνο εκείνα που ικανοποιούν τις συνθήκες που ορίζουμε με την βοήθεια των 5 εντολών για τις οποίες κάναμε νύξη παραπάνω. Για τα γνωρίσματα που παρακρατούμε βρίσκουμε το άλλο άκρο τους (υπενθυμίζουμε ότι ένα γνώρισμα ορίζει μια σχέση μεταξύ 2 αντικειμένων). Για κάθε ένα από τα άκρα που ευρέθησαν επαναλαμβάνουμε την ίδια διαδικασία αναδρομικά: βρίσκουμε τα γνωρίσματα τους κ.τ.λ. Σε κάθε στάδιο της αναδρομής επαυξάνουμε το σύνολο-απάντηση με τα 'παρακρατηθέντα' γνωρίσματα. Η διαδικασία σταματάει όταν το σύνολο-απάντηση δεν 'εμπλουτίζεται' πλέον με νέα γνωρίσματα.

Επισημάναμε ότι στη διαμόρφωση του αποτελέσματος της εντολής *tal* συνεισφέρουν 5 ακόμα εντολές, τις παρουσιάζουμε αφού δώσουμε κάποιες διευκρινήσεις. Στην περιγραφή της λειτουργίας της tal διευκρινίσαμε ότι από κάθε αντικείμενο δεν λαμβάνουμε υπ' όψιν όλα τα γνωρίσματα του αδιακρίτως αλλά μόνο εκείνα που ικανοποιούν κάποια συνθήκη. Μπορούμε να ορίσουμε τη συνθήκη αυτή με βάση τόσο το ίδιο το γνώρισμα και τις ιδιότητες του (σε ποια κλάση ανήκει κλπ.) όσο και τις ιδιότητες του δεύτερου αντικειμένου που συμμετέχει στο γνώρισμα. Ανάλογα με το αν το δεύτερο αντικείμενο είναι η αφετηρία (from value) ή η τιμή του γνωρίσματος διακρίνεται η διάσχιση του γνωρίσματος (η συμπερίληψή του δηλαδή στο σύνολο-απάντηση) σε αντίστροφη (backward) η ευθεία (forward) κατεύθυνση. Τα παραπάνω δικαιολογούν τις 4 από τις 5 εντολές που διαμορφώνουν το αποτέλεσμα της **tal**:

- Η εντολή **sflc** (=Set From-Link Condition) ακολουθείται από μια συνθήκη που πρέπει να ικανοποιείται από τα γνωρίσματα που διασχίζονται ευθέως.
- Η εντολή **stvc** (=Set To-Value Condition) ακολουθείται από μια συνθήκη που πρέπει να ικανοποιείται από τις τιμές (to-values) των γνωρισμάτων που διασχίζονται ευθέως
- Η εντολή **stlc** (=Set To-Link Condition) ακολουθείται από μια συνθήκη που πρέπει να ικανοποιείται από τα γνωρίσματα που διασχίζονται αντιστρόφως.
- Η εντολή **sfvc** (=Set From-Value Condition) ακολουθείται από μια συνθήκη που πρέπει να ικανοποιείται από τις αφετηρίες (from-values) των γνωρισμάτων που διασχίζονται αντιστρόφως.

Πέρα από αυτές τις εντολές υπάρχει και η **sdep** (=Set Depth) που περιορίζει την απόσταση από το αρχικό αντικείμενο από το οποίο ξεκίνησε η ερώτηση. Εκείνο που γίνεται δηλαδή είναι η αναδρομή να σταματάει μετά από τον αριθμό των βημάτων που υπαγορεύει η εντολή.

Χρειάζεται να επεξηγήσουμε λίγο παραπάνω το πως σχηματίζονται οι λογικές συνθήκες που ακολουθούν τις εντολές **sflc, stvc, stlc** και *sfvc*. Δεν θα περιγράψουμε τις λογικές συνθήκες στην πλήρη έκταση τους, θα αναφέρουμε μόνο επιλεκτικά τα βασικότερα χαρακτηριστικά τους. Οι απλούστερες λογικές εκφράσεις είναι η **succ** και η **fail**. Η πρώτη είναι πάντα αληθής και η δεύτερη πάντα ψευδής. Επίσης υπάρχουν οι βασικοί τελεστές για λογικές εκφράσεις **and, or** και **not**. Όλοι τους χρησιμοποιούν προθεματικό (prefix) συμβολισμό χωρίς παρενθέσεις και ασφαλώς χρησιμοποιούνται για να κατασκευάσουν μια συνθετότερη λογική συνθήκη από απλούστερες. Η πρώτη μη τετριμμένη λογική συνθήκη είναι η:

blng ‹αντικείμενο› ‹σύνολο›

η οποία είναι αληθής αν το 'αντικείμενο' είναι στοιχείο του συνόλου. Ο βασικός τρόπος για να περιγράψει κανείς το αντικείμενο είναι μέσω της έκφρασης:

sys ‹system_id›

Όπου 'system_id' είναι το αναγνωριστικό που έχει προσδώσει το σύστημα στο αντικείμενο.

Αντίστοιχα ο βασικός τρόπος για να περιγράψει κανείς ένα σύνολο είναι μέσω της έκφρασης:

set ‹set_id›

Όπου set_id είναι το όνομα που χρησιμοποιήθηκε για να αποθηκευτεί προηγουμένως ένα σύνολο με την εντολή **stor**. Στην ειδική περίπτωση που **set_id=0** νοείται το γνώρισμα ή το άκρο αυτού το οποίο εξετάζεται για να συμπεριληφθεί στο σύνολο-απάντηση.

Πέρα όμως από ήδη υπάρχοντα σύνολα μπορούμε να περιγράψουμε και συνδυασμούς αυτών με το τρέχον σύνολο που προκύπτουν από την ένωση, τομή και διαφορά χρησιμοποιώντας εκφράσεις της μορφής:

su ‹σύνολο›
si ‹σύνολο›
sd ‹σύνολο›

Μπορούμε ακόμη να σχηματίσουμε σύνολα που προκύπτουν από την εφαρμογή μιας βασικής ερώτησης της βάσης (π.χ. **gai = get all instances**) σε ένα σύνολο. Σε μια τέτοια περίπτωση η σύνταξη που ακολουθούμε είναι: **‹όνομα ερώτησης› ‹σύνολο›**.

4.1 Η ανάγκη για επιπλέον QI primitives

Όπως έχουμε ήδη παρουσιάσει η TELQUEL επιστρέφει σαν αποτέλεσμα των ερωτήσεων της σχέσεις (relations). Επιπλέον χρησιμοποιεί σχέσεις και στα ενδιάμεσα στάδια παραγωγής του τελικού αποτελέσματος όπως προκύπτει από την περιγραφή που κάναμε για το front-end της γλώσσας στο προηγούμενο κεφάλαιο. Εφόσον λοιπόν θέλουμε σε τελικό στάδιο οι TELQUEL ερωτήσεις να μεταφράζονται σε ισοδύναμες τους στο QI θα πρέπει το QI (και συνεπώς και το PQI) να υποστηρίζει πέρα από σύνολα και σχέσεις ως δομές, στις οποίες επιστρέφονται αποτελέσματα. Απλώς βέβαια, η προσθήκη μιας τέτοιας δομής στο QI δεν είναι αρκετή. Θα πρέπει να υποστηρίζονται λειτουργίες πάνω σ' αυτή που μέχρι τώρα δεν υπήρχαν γιατί απλούστατα δεν είχε νόημα να υπάρχουν τέτοιες λειτουργίες για σύνολα. Ως ελάχιστο, αλλά και ικανό για τις ανάγκες μας, σύνολο τέτοιων λειτουργιών κρίνεται αυτό της σχεσιακής άλγεβρας (select, project, join). Αυτό το σύνολο λοιπόν, μαζί με την αντίστοιχη δομή επιλέγουμε να υλοποιήσουμε.

Από την άλλη απλά η προσθήκη μιας δομής 'σχέση' στο QI έστω και μαζί με τις προαναφερθείσες λειτουργίες δεν είναι ικανοποιητική αν είναι ξένη ως προς τη δομή σύνολο. Η ερωτήσεις του QI έχουν σχεδιαστεί για να επιστρέφουν αλλά και να εφαρμόζονται πάνω σε σύνολα. Εφόσον λοιπόν δεν επιθυμούμε να ξανά-υλοποιήσουμε τις ερωτήσεις αυτές θα πρέπει η δομή 'σχέση' να συνεργάζεται με την δομή σύνολο. Ο πιο φυσικός τρόπος για να γίνει αυτό είναι να υλοποιήσουμε τη 'σχέση' ως επέκταση της δομής σύνολο. Θεωρούμε λοιπόν ότι ένα σύνολο είναι μια σχέση με μια μόνο στήλη. Επίσης εφόσον δεν αλλάζουμε τις βασικές ερωτήσεις του QI και αυτές εξακολουθούν να επιστρέφουν ως αποτέλεσμα ένα σύνολο

προκειμένου να μπορούν να λειτουργήσουν οι ίδιες ερωτήσεις και με την δομή ‘σχέση’, θεωρούμε ότι αυτή αποτελείται από μια διατεταγμένη ακολουθία στηλών. Κάθε στήλη μπορεί να αντιμετωπιστεί κατά τουλάχιστον τις λειτουργίες που μας ενδιαφέρουν ως σύνολο.

Οι περισσότερες από τις παραπάνω σχεδιαστικές αποφάσεις οφείλονται σε μια πρώτη υλοποίηση της δομής ‘σχέση’ που έγινε από τον Πολύβιο Κλημαθιανάκη. Η (επαν)υλοποίηση που έγινε από τον γράφοντα ήταν:

1. Απαραίτητη, διότι η αρχική υλοποίηση δεν ήταν πλήρης όσον αφορά τις λειτουργίες που υποστήριζε η δομή (δεν είχε υλοποιηθεί καθόλου η λειτουργία select).

2. Βελτιωμένη, μια και εκμεταλλεύτηκε καλύτερα τα οντοκεντρικά χαρακτηριστικά που προσέφερε η γλώσσα υλοποίησης (C++) χρησιμοποιώντας templates για την κατασκευή της δομής ‘στήλη’ κάτι που εκτιμάται ότι θα επιταχύνει τον όλο μηχανισμό.

3. Συμβατή με την προηγούμενη υλοποίηση. Η συμβατότητα είναι πλήρης εξωτερικά όσον αφορά τις λειτουργίες που διατίθενται στο χρήστη, αλλά και εσωτερικά σε μεγάλο βαθμό αφού οι αλγόριθμοι που χρησιμοποιήθηκαν σε πολλά σημεία είναι οι ίδιοι.

4.2 Η δομή column: σύντομη περιγραφή

Το βασικό συστατικό κάθε δομής 'σχέση' κατά την υλοποίηση μας είναι η βοηθητική δομή column που όπως δηλώνει και το όνομα της έχει σκοπό να μοντελοποιήσει μια από τις στήλες της 'σχέσης'. Σκοπός μας είναι η δομή αυτή να είναι όσο το δυνατόν πιο κοντά στη δομή σύνολο έτσι ώστε να μπορεί να χρησιμοποιηθεί τόσο ως αποτέλεσμα όσο και ως είσοδος σε βασικές QI ερωτήσεις. Ταυτόχρονα θα πρέπει να ενσωματώνει τις λειτουργίες που θα την καθιστούν ικανή να λειτουργεί ως τμήμα της ευρύτερης δομής 'σχέση'.

Προκειμένου να ικανοποιηθεί η τελευταία από τις παραπάνω απαιτήσεις η δομή column κρατάει εκτός από τα στοιχεία–τιμές που απαρτίζουν την στήλη επιπλέον πληροφορία. Έτσι για κάθε στοιχείο της στήλης η δομή 'θυμάται' και με ποια στοιχεία της επόμενης στήλης – αν αυτή υπάρχει – 'συνδέεται' αυτό το στοιχείο. Αν για παράδειγμα έχουμε την σχέση:

A1	B1
A1	B2
A2	B1
A2	B2

Πίνακας 1

Τότε η πρώτη στήλη θα είχε τα στοιχεία A1 και A2 και για κάθε ένα από αυτά θα θυμόταν ότι 'συνδέεται' με τα στοιχεία B1 και B2 της δεύτερης στήλης. Αντίστοιχα η δεύτερη στήλη θα περιείχε απλώς τα στοιχεία B1 και B2. Μια άλλη δυνατότητα που υπάρχει είναι να "μαρκάρουμε" ως διαγραφέντα ορισμένα από τα στοιχεία μιας στήλης. Τα στοιχεία δεν διαγράφονται πραγματικά αλλά οι πλειάδες (tuples) που τα περιέχουν δεν συμπεριλαμβάνονται πλέον στην σχέση. Αν λ.χ. 'μαρκάραμε' το A1 στην πρώτη στήλη της προηγούμενης σχέσης θα είχαμε ως αποτέλεσμα την διαγραφή των πλειάδων ‹A1,B1› και ‹A1,B2›. Ο μηχανισμός αυτός θα μας φανεί χρήσιμος όταν θα υλοποιήσουμε τις διάφορες λειτουργίες μεταξύ σχέσεων.

Πέρα από το 'μαρκάρισμα' συγκεκριμένων στοιχείων είναι δυνατόν με την χρήση ενός flag να 'κρύψουμε' μια ολόκληρη στήλη από μια σχέση. Κάτι τέτοιο είναι επιθυμητό στις περιπτώσεις που δεν μας ενδιαφέρουν τα περιεχόμενα της στήλης αλλά η πληροφορία του με ποια στοιχεία της επόμενης στήλης θα πρέπει να συνδεθούμε. Εκτός από την προφανή χρήση για γρήγορες προβολές (projects) οι κρυμμένες στήλες χρησιμοποιούνται στις σχέσεις που προκύπτουν από συνολοθεωρητικές πράξεις (ένωση, τομή, διαφορά).

4.3 Η δομή tuple: αρχές λειτουργίας

Χρησιμοποιώντας έναν πίνακα από 'στήλες', τις δομές δηλαδή που περιγράψαμε προηγουμένως η δομή *tuple* υλοποιεί μια σχέση. Αν και tuple είναι το όνομα που χρησιμοποιείται για να αναφερθούμε σε ένα στοιχείο μιας σχέσης στην υλοποίηση μας χρησιμοποιούμε καταχρηστικά αυτό το όνομα για να αναφερθούμε στην δομή που περιγράφει ολόκληρη την σχέση. Ο λόγος που μας οδήγησε στην χρήση αυτής της (ατυχής) ορολογία ήταν το ότι είχε επιλεγεί από την προηγούμενη υλοποίηση.

Επειδή είναι επιθυμητό και χρήσιμο σε κάθε σχέση να μπορούμε να 'ανακατεύουμε' (αντιμεταθέτουμε) τις στήλες της αλλά κάτι τέτοιο δεν είναι απλό λόγω του ότι κάθε στήλη κρατάει πληροφορία που αφορά και την επόμενή της. Αντί λοιπόν να κάνουμε τροποποιήσεις που μπορεί να επηρεάσουν μεγάλο μέρος της δομής χρησιμοποιούμε σε κάθε σχέση έναν πίνακα *μετάθεσης* που καταγράφει στην #i θέση του σε ποια θέση στον πίνακα των στηλών βρίσκεται η #i στήλη. Αν λόγου χάριν στην θέση Ο του πίνακα αυτού βρίσκεται ο αριθμός 3 αυτό σημαίνει ότι στον πίνακα των στηλών η στήλη #Ο βρίσκεται στην θέση 3 (col[3]). Σημειώνουμε ότι οι τιμές του πίνακα μετάθεσης δεν καλύπτουν όλες τις θέσεις του πίνακα στηλών, αυτό εξηγείται μέσω των 'κρυμμένων' στηλών. Οι στήλες που είναι κρυμμένες δεν εμφανίζονται στο χρήστη και συνεπώς δεν έχουν αριθμό.

Με τη χρήση του πίνακα μετάθεσης η αντιμετάθεση των στηλών #i και #j στη δομή tuple είναι απλή. Το μόνο που χρειάζεται να γίνει είναι η εναλλαγή (swap) τιμών των στοιχείων #i και #j του πίνακα μετάθεσης. Φυσικά οι διάφορες λειτουργίες–μέθοδοι της δομής λαμβάνουν υπ' όψιν τους τον πίνακα μετάθεσης.

Ένας δεύτερος πίνακας που διατηρείται από κάθε δομή tuple είναι ο πίνακας συνδέσεων (join). Ο πίνακας αυτός έχει διαστάσεις Nx2. Στα στοιχεία της μορφής join[i,0] βρίσκονται αριθμοί στηλών της tuple στην οποία ανήκει ο πίνακας. Στα στοιχεία της μορφής join[i,1] βρίσκονται αριθμοί στηλών της δομής tuple με την οποία θα γίνει σύνδεση. Κάθε ζευγάρι τιμών (join[i,0], join[i,1]) δηλώνει ότι σε κάθε πλειάδα της σχέσης που θα προκύψει από τη σύνδεση οι τιμές στην στήλη join[i,0] της πρώτης tuple και στη στήλη join[i,1] της δεύτερης θα πρέπει να έχουν την ίδια τιμή. Εφόσον από αυτή την διαδικασία προκύπτουν στήλες που έχουν τις ίδιες τιμές σε δύο στήλες, η δεύτερη στήλη διαγράφεται (κρύβεται).

Ένα ακόμη στοιχείο που θυμάται η 'tuple' είναι η στήλη που θα αποτελέσει την πραγματική 'είσοδο' (input) για ερώτηση στη βάση στην περίπτωση που όλη η 'tuple' χρησιμοποιηθεί ως είσοδος. Όπως είδαμε οι ερωτήσεις στη βάση υπολογίζουν το αποτέλεσμα τους είτε με βάση ένα (τρέχον) αντικείμενο είτε ένα (τρέχον) σύνολο. Στην δεύτερη περίπτωση είναι δυνατόν τη θέση του συνόλου να καταλάβει μια 'tuple' οπότε για να έχει νόημα η ερώτηση, εφαρμόζεται σε μια στήλη της tuple, η οποία ως δομή είναι ανάλογη του συνόλου. Το αποτέλεσμα μιας τέτοιας ερώτησης αποθηκεύεται σε μια άλλη στήλη της 'tuple'. Το ποια θα είναι αυτή η στήλη εξαρτάται από την κλήση της μεθόδου Tuple::new_column που παρουσιάζουμε παρακάτω.

Η τελευταία πληροφορία που κρατάει η δομή tuple είναι ένας πίνακας με τις θέσεις που βρίσκονται οι κρυμμένες στήλες και αυτό γιατί οι εν λόγω στήλες δεν αναφέρονται στον πίνακα μετάθεσης.

4.4 Οι επιπλέον μέθοδοι/εντολές

Η υλοποίηση της δομής (κλάσης) tuple είχε ως αποτέλεσμα την δημιουργία νέων εντολών στο QI και το PQI που προκύπτουν κυρίως από τις διάφορες μεθόδους που υπάρχουν για την παραπάνω κλάση. Τις μεθόδους αυτές καθώς και τους αλγορίθμους που τις υλοποιούν περιγράφουμε στα επόμενα.

4.4.1 Η Tuple::new_column (επανυλοποίηση)

Η λειτουργία αυτής της μεθόδου συνίσταται στο να δημιουργεί μια καινούργια στήλη στη δομή tuple. Η στήλη αυτή θα 'γεμίσει' με στοιχεία (αντικείμενα) από την εφαρμογή της επόμενης ερώτησης που θα έχει ως είσοδο (input) αυτή την δομή tuple.

4.4.1.1 Ορίσματα–Παράμετροι

Κανένα. Στην περίπτωση του QI η μέθοδος αφορά την τρέχουσα tuple.

4.4.1.2 Αλγόριθμος

Η καινούργια στήλη δεν δημιουργείται την στιγμή που καλείται η μέθοδος. Εκείνο που πραγματικά γίνεται είναι ότι η δομή θυμάται ότι στην επόμενη ερώτηση θα πρέπει να δημιουργήσει μια καινούργια στήλη. Όταν η ερώτηση αυτή εφαρμοστεί πάνω στην δομή 'tuple' εξετάζουμε σε ποια από τις στήλες της θα γίνει πραγματική εφαρμογή της. Εφαρμόζουμε την ερώτηση σε κάθε αντικείμενο αυτής της στήλης. Για κάθε αντικείμενο, από τα έστω n, που έχει η στήλη, η ερώτηση επιστρέφει ένα σύνολο ως απάντηση. Τα στοιχεία της καινούργιας στήλης θα είναι ακριβώς αυτά που ανήκουν στην ένωση των συνόλων. Επιπλέον, προκειμένου να συμπληρωθούν τα tuples με την μία τιμή που τους λείπει, κάθε στοιχείο (αντικείμενο) της στήλης που αποτέλεσε την είσοδο (input) της ερώτησης συνδυάζεται με όλα τα στοιχεία (αντικείμενα) του συνόλου–απάντηση που προέκυψε όταν η ερώτηση εφαρμόστηκε στο συγκεκριμένο στοιχείο.

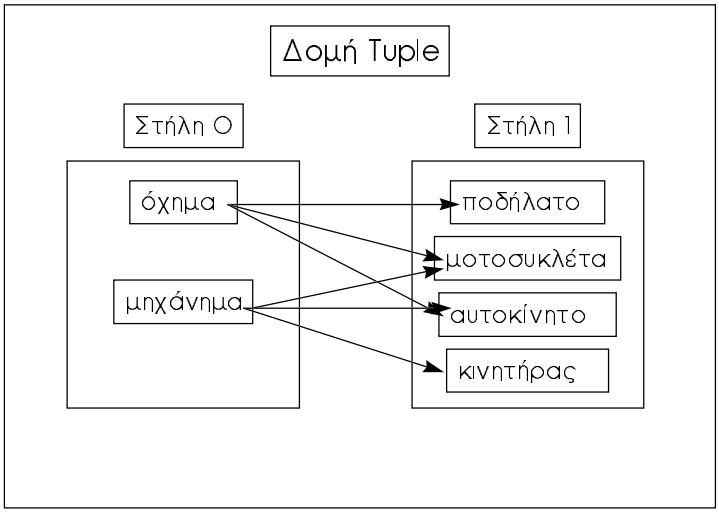
Για παράδειγμα, ας υποθέσουμε ότι αρχικά έχουμε μόνο μια στήλη με στοιχεία τις κλάσεις '*όχημα*' και '*μηχάνημα*' οι οποίες στην βάση μας

έχουν τις υποκλάσεις {*'αυτοκίνητο', 'ποδήλατο', 'μοτοσικλέτα'*} και {*'αυτοκίνητο', 'μοτοσικλέτα', 'κινητήρας'*} αντίστοιχα. Τότε, αν ζητήσουμε μια καινούργια στήλη και στη συνέχεια εφαρμόσουμε την εντολή **gasb** (Get All SuBclasses) με είσοδο την πρώτη στήλη τότε θα προκύψει η σχέση:

όχημα	αυτοκίνητο
όχημα	ποδήλατο
όχημα	μοτοσυκλέτα
μηχάνημα	αυτοκίνητο
μηχάνημα	μοτοσυκλέτα
μηχάνημα	κινητήρας

Πίνακας 2

Η δομή 'tuple' θα αναπαριστά όμως αυτή την σχέση όπως φαίνεται στο σχήμα 2:



Σχήμα 2: Εσωτερική αναπαράσταση της σχέσης που περιγράψαμε

4.4.2 Η Tuple::tuple_no_projection_column

Λειτουργία αυτής της μεθόδου είναι να κρύβει (κατ' ουσία διαγράφει) τη στήλη την οποία προσδιορίζει ο χρήστης.

4.4.2.1 Ορίσματα – Παράμετροι

Πρέπει να προσδιοριστεί ο αριθμός της στήλης την οποία θέλουμε να κρύψουμε.

4.4.2.2 Αλγόριθμος

Θα πρέπει να διευκρινίσουμε ότι πραγματική διαγραφή της στήλης δεν υφίσταται. Απλώς η στήλη που δεν θέλουμε πλέον να φαίνεται 'μαρκάρεται' ανάλογα και το μαρκάρισμα αυτό το λαμβάνουν υπ' όψιν τους οι μέθοδοι που παρουσιάζουν το περιεχόμενο της 'tuple'.

4.4.3 Η Tuple::set_join_pos

Λειτουργία της μεθόδου αυτής είναι προσθέτει άλλο ένα ζευγάρι στηλών στο πίνακα συνδέσεων. Τα ζευγάρια αυτού του πίνακα λαμβάνονται υπ' όψιν την στιγμή που πρόκειται να γίνει σύνδεση με μια δεύτερη 'tuple'.

4.4.3.1 Ορίσματα–Παράμετροι

Οι αριθμοί των 2 στηλών αρκούν.

4.4.3.2 Αλγόριθμος

Τετριμμένος.

4.4.4 Η Tuple::tuple_join

Η tuple_join εκτελεί τη σύνδεση μεταξύ της τρέχουσας 'tuple' και μιας δεύτερης που προσδιορίζεται από το χρήστη λαμβάνοντας υπ' όψιν το πίνακα σύνδεσης της τρέχουσας 'tuple'.

4.4.4.1 Ορίσματα – Παράμετροι

Το λογικό όνομα της δεύτερης 'tuple' πρέπει να δοθεί.

4.4.4.2 Αλγόριθμος

Το πρώτο πράγμα που κάνει η tuple_join είναι ο σχηματισμός του καρτεσιανού γινομένου της τρέχουσας 'tuple' με ένα αντίγραφο της δεύτερης 'tuple'. Το γινόμενο αυτό σχηματίζεται επεκτείνοντας τις στήλες της τρέχουσας 'tuple' με τις στήλες του αντιγράφου. Η επέκταση επιτυγχάνεται δηλώνοντας στον πίνακα στηλών της τρέχουσας 'tuple' τις στήλες του αντιγράφου και ενημερώνοντας κατάλληλα τον πίνακα μετάθεσης και τον πίνακα κρυφών στηλών της τρέχουσας tuple. Ακολούθως η, πριν την σύνδεση, τελευταία στήλη της τρέχουσας 'tuple' ενημερώνεται ότι η επόμενή στήλη της στην 'tuple' είναι η πρώτη στήλη του αντιγράφου (οι στήλες θυμούνται ποια είναι η προηγούμενη και ποια η επόμενη τους μέσα στην 'tuple') και αντίστοιχη ενημέρωση γίνεται για την πρώτη στήλη του αντιγράφου. Ως τελευταίο βήμα για το σχηματισμό του καρτεσιανού γινομένου κάθε στοιχείο της, πριν την σύνδεση, τελευταίας στήλης της τρέχουσας 'tuple' συνδέεται με όλα τα στοιχεία της επόμενης πλέον στήλης της.

Αφού έχει προκύψει το καρτεσιανό γινόμενο σχηματίζεται από αυτό η σύνδεση εξετάζοντας τον πίνακα σύνδεσης της τρέχουσας 'tuple'. Για κάθε ζευγάρι τιμών αυτού του πίνακα ‹join[i,O], join[i,1]› η στήλη που αντιστοιχεί στην τιμή join[i,1] (είναι η στήλη old+join[i,1] όπου old ο αριθμός των στηλών της τρέχουσας 'tuple' πριν την σύνδεση) "κλειδώνεται" στην στήλη join[i,O]. Όταν μια στήλη είναι κλειδωμένη σε κάποια άλλη τότε από όλες τις πλειάδες που υπάρχουν στην 'tuple' θεωρούνται έγκυρες μόνο εκείνες που έχουν τις ίδιες τιμές στις στήλες που είναι 'κλειδωμένες' μεταξύ τους. Τα κλειδώματα των στηλών λαμβάνονται υπ' όψιν από μια ιδιωτική (private) μέθοδο – επαναλήπτη (iterator) όπου η λειτουργία της είναι να επιστρέφει κάθε φορά την επόμενη πλειάδα της Tuple. Έπειτα κρύβονται οι περιττές στήλες.

Παρατηρήστε ότι κάθε στήλη κλειδώνεται (τίθεται ένα flag που δείχνει στον αριθμό της στήλης στην οποία είμαστε "κλειδωμένοι") σε στήλη με αύξοντα αριθμό μικρότερο του δικού της. Αυτή η συνθήκη

εξασφαλίζει ότι δεν πρόκειται να υπάρξουν 'κύκλοι' μεταξύ των κλειδωμένων στηλών και επιπλέον είναι απαραίτητη στον επαναλήπτη διότι προκείμένου αυτός να επιστρέψει μια πλειάδα διασχίζει τις στήλες της 'tuple' από την αρχή προς το τέλος. Αυτή η επιλογή καθιστά ευκολότερο τον έλεγχο για ισότητα τιμών μιας στήλης με στήλες από τις οποίες έχουμε ήδη περάσει – και έχουμε διαλέξει ποιο στοιχείο τους θα μπει στη πλειάδα – παρά με στήλες που θα συναντήσουμε αργότερα.

4.4.5 Η Tuple::tuple_set_input

Ο σκοπός ύπαρξης αυτής της μεθόδου είναι να ορίσει ποια από τις στήλες της τρέχουσας 'tuple' θα αποτελέσει είσοδο (input) για τη επόμενη ερώτηση που θα απευθύνουμε στη βάση. Αν δεν προσδιοριστεί αλλιώς η στήλη αυτή είναι πάντα η τελευταία.

4.4.5.1 Ορίσματα – Παράμετροι

Ο αριθμός της στήλης που θα αποτελέσει είσοδο.

4.4.5.2 Αλγόριθμος

Απλώς τίθεται η κατάλληλη τιμή στη μεταβλητή της tuple που κρατάει πληροφορία για το ποια στήλη αποτελεί είσοδο για την επόμενη ερώτηση.

4.4.6 Η Tuple::return_tuples

Χρησιμοποιούμε αυτή την μέθοδο για να γράψουμε – παρουσιάσουμε τα περιεχόμενα της 'tuple' με την γνώριμη μορφή του πίνακα.

4.4.6.1 Ορίσματα – Παράμετροι

Στο PQl το αρχείο (file descriptor) στο οποίο θα γραφούν τα αποτελέσματα. Στο Ql κανένα.

4.4.6.2 Αλγόριθμος

Παρατηρήστε ότι κάθε 'tuple' στην ουσία είναι ένα σύνολο από δένδρα των οποίων οι ρίζες είναι τα στοιχεία της πρώτης στήλης. Κάθε πλειάδα στη δομή μας αντιπροσωπεύεται από ένα κλάδο αυτού του δένδρου. Συνεπώς αρκεί να διατρέξουμε όλους τους κλάδους, όλων των δένδρων.

Για να το κάνουμε αυτό χρησιμοποιούμε μια ιδιωτική (private) μέθοδο της 'tuple' που μας επιστρέφει με κάθε κλήση της ένα μονοδιάστατο πίνακα με τα στοιχεία που απαρτίζουν τον επόμενο κλάδο. Ουσιαστικά δηλαδή, λαμβάνουμε την επόμενη πλειάδα. Να διευκρινίσουμε ότι η μέθοδος για τη λήψη των 'tuples' δεν τις επιστρέφει σε τελική μορφή. Συγκεκριμένα η μέθοδος δεν λαμβάνει υπ' όψιν τις αντιμεταθέσεις στηλών και επιστρέφει και τις τιμές των κρυφών στηλών. Η μέθοδος από την άλλη μεριά λαμβάνει υπ' όψιν της τυχόν κλειδώματα μεταξύ στηλών καθώς και μαρκαρισμένα στοιχεία των στηλών ως διαγραφέντα. Η τελική πλειάδα κάθε φορά προκύπτει από την εφαρμογή του πίνακα μετάθεσης στην πλειάδα που επέστρεψε η ιδιωτική μέθοδος.

Προκειμένου να θυμάται σε ποια πλειάδα της 'tuple' είχε σταματήσει την προηγούμενη φορά η ιδιωτική μέθοδος–επαναλήπτης χρησιμοποιεί σε κάθε στήλη k_i+1 δείκτες, k_i = αριθμός των στοιχείων στην στήλη i. Ο τελευταίος δείκτης δείχνει σε ποιο στοιχείο της στήλης είχαμε σταματήσει. Οι υπόλοιποι δείκτες δείχνουν για κάθε στοιχείο της στήλης με ποιο στοιχείο της επόμενης πρέπει να προχωρήσουμε. Αλγοριθμικά ο σχηματισμός μιας πλειάδας έχει ως εξής (ξεκινάμε με αρχικοποιημένους τους δείκτες στο πρώτο στοιχείο του εκάστοτε συνόλου):

1. Εξέτασε το τρέχον στοιχείο της στήλης. Αν αυτό είναι 'μαρκαρισμένο' προχώρησε θέσε ως τρέχον το επόμενο για την πρώτη και τελευταία στήλη αλλιώς απάντησε ότι πλειάδα δεν υπάρχει. Ομοίως ενέργησε αν υπάρχει κλείδωμα σε άλλη στήλη και το τρέχον στοιχείο της παρούσας στήλης είναι διαφορετικό από το τρέχον αυτής.
2. Εφόσον μπορούμε να συνεχίσουμε εξέτασε σε ποιο από τα στοιχεία της επόμενης στήλης με τα οποία συνδέεται το τρέχον της παρούσας δείχνει ο αντίστοιχος δείκτης και θέσε αυτό το στοιχείο ως τρέχον στην επόμενη στήλη.

3. Ζήτησε από την επόμενη στήλη να επιστρέψει το υπόλοιπο, δεξί μέρος της πλειάδας. Προσάρτησε σ' αυτή το τρέχον στοιχείο και επέστρεψε το αποτέλεσμα. Αν η επόμενη στήλη απαντήσει ότι δεν διαθέτει τέτοια πλειάδα εξέτασε το επόμενο από τα στοιχεία με τα οποία είναι συνδεδεμένο το τρέχον και αύξησε το σχετικό δείκτη. Ακολούθως επανέλαβε το βήμα 2. Αν έχουν εξαντληθεί όλα τα στοιχεία με τα οποία συνδέεται το τρέχον αρχικοποίησε το σχετικό δείκτη προς αυτά. Αν βρισκόμαστε στην πρώτη στήλη προχώρησε στο επόμενο στοιχείο της στήλης, αν υπάρχει, και επανέλαβε από το βήμα 1. Για τις υπόλοιπες στήλες απάντησε ότι δεν υπάρχει σχετική πλειάδα.

Ο αλγόριθμος τερματίζει όταν διατρέξουμε όλα τα στοιχεία της πρώτης στήλης.

4.4.7 Η Tuple::tuple_union

Ασφαλώς εκείνο που γίνεται είναι το αναμενόμενο. Σχηματίζεται η ένωση μεταξύ της τρέχουσας 'tuple' και της 'tuple' που προσδιορίζει ο χρήστης. Το αποτέλεσμα αποθηκεύεται στην τρέχουσα 'tuple'.

4.4.7.1 Ορίσματα – Παράμετροι

Δίδεται το λογικό όνομα της δεύτερης 'tuple'.

4.4.7.2 Αλγόριθμος

Τα δεδομένα και των 2 'tuples' μεταφέρονται σε μια νέα δομή. Αρχικά δημιουργείται μια ακολουθία από κ 'φανερές' στήλες όπου κ το πλήθος των στηλών των 2 'tuples' που παρουσιάζεται στο χρήστη. Ανάμεσα σε κάθε 2 φανερές στήλες υπάρχει μια κρυφή που τα στοιχεία της είναι αποκλειστικά ακέραιοι διαδοχικοί αριθμοί. Όλες οι κρυφές στήλες είναι κλειδωμένες μεταξύ τους. Σύμφωνα με τα ανωτέρω φτάνουμε σε ένα σύνολο 2κ−1 στηλών. Ακολούθως κατασκευάζονται κατάλληλα ο πίνακας μεταθέσεων και ο πίνακας κρυφών στηλών. Ο πίνακας μεταθέσεων καταγράφει ότι η στήλη i βρίσκεται στη θέση 2*i του πίνακα στηλών.

Στο επόμενο βήμα εισάγουμε τις πλειάδες που ανήκουν στην ένωση των δυο tuples στην καινούργια δομή αφού τις μετασχηματίσουμε λίγο. Συγκεκριμένα η πλειάδα $\langle a_1,a_2,...,a_n\rangle$ μετασχηματίζεται στην πλειάδα $\langle a_1,i,a_2,i,..,i,a_n\rangle$. Όπου τα i αντιστοιχούν στα περιεχόμενα των κρυφών στηλών, η δε τιμή του i επιλέγεται έτσι ώστε να είναι η αμέσως επόμενη από αυτές που υπάρχουν ήδη στις αριθμητικές στήλες. Έτσι αν στις κρυφές στήλες υπάρχουν οι αριθμοί 1...m κατά την εισαγωγή της παραπάνω πλειάδας το i θα πάρει την τιμή m+1. Υπό αυτή την μορφή εισάγουμε στην νέα δομή χωρίς κανένα έλεγχο όλες τις πλειάδες της πρώτης 'tuple'. Η πλειάδες αυτές παίρνονται με την μορφή που θα ήταν ορατές στο χρήστη και έτσι δεν υπάρχει ανάγκη για την διατήρηση των κλειδωμάτων των παλιών δομών, επιπλέον δεν μεταφέρονται οι κρυφές στήλες. Αφού ολοκληρώσουμε την εισαγωγή των πλειάδων της πρώτης 'tuple' εισάγουμε και τις πλειάδες της δεύτερης 'tuple' που δεν υπάρχουν στην πρώτη. Ο έλεγχος γίνεται με τη χρήση της γνωστής μεθόδου–επαναλήπτη.

Αφού η διαδικασία ολοκληρωθεί καταστρέφεται η παλιά τρέχουσα 'tuple' και την θέση της παίρνει η καινούργια δομή

4.4.8 Η Tuple::tuple_intersection

Όπως είναι λογικό η εντολή–μέθοδος αυτή δημιουργεί την τομή μεταξύ της τρέχουσας 'tuple' και της 'tuple' που προσδιορίζεται από τον χρήστη. Το αποτέλεσμα καταχωρείται στην τρέχουσα 'tuple'.

4.4.8.1 Ορίσματα – παράμετροι

Όπως στην tuple_union.

4.4.8.2 Αλγόριθμος

Δημιουργούμε μια δομή με τον ίδιο τρόπο όπως στην tuple_union (2κ–1 στήλες). Σ' αυτή τη δομή όμως εισάγουμε μόνο τις πλειάδες της τρέχουσας 'tuple' που ανήκουν στην δεύτερη 'tuple'. Ο έλεγχος γίνεται όπως στην περίπτωση της ένωσης.

4.4.9 Η Tuple::tuple_difference

Αντίστοιχα με τα προηγούμενα η εντολή αυτή δημιουργεί την διαφορά μεταξύ της τρέχουσας 'tuple' και μιας δεύτερης 'tuple' που προσδιορίζεται από τον χρήστη. Το αποτέλεσμα καταχωρείται στην τρέχουσα 'tuple'.

4.4.9.1 Ορίσματα – παράμετροι

Όπως στην tuple_union.

4.4.9.2 Αλγόριθμος

Δημιουργούμε μια δομή με τον ίδιο τρόπο όπως στην tuple_union (2κ–1 στήλες). Σ' αυτή τη δομή όμως εισάγουμε μόνο τις πλειάδες της τρέχουσας 'tuple' που δεν ανήκουν στην δεύτερη 'tuple'. Ο έλεγχος γίνεται κατά τα γνωστά.

4.4.10 Η Tuple::tuple_copy

Αντιγράφει την 'tuple' που προσδιορίζεται από το χρήστη στην τρέχουσα 'tuple'. Τα παλιά περιεχόμενα της τρέχουσας 'tuple' χάνονται.

4.4.10.1 Ορίσματα – Παράμετροι

Το λογικό όνομα της 'tuple' από την οποία θα γίνει αντιγραφή είναι φυσικά απαραίτητο.

4.4.10.2 Αλγόριθμος

Πρόκειται για μία προς μια δημιουργία αντιγράφων των δομών της δεύτερης 'tuple' και τοποθέτηση αυτών των αντιγράφων στην τρέχουσα 'tuple'. Το μόνο σημείο που χρειάζεται προσοχή είναι όπου υπάρχουν δείκτες, αυτοί να δείχνουν στα νέα αντίγραφα.

4.4.11 Η Tuple::column_intersect

Η μέθοδος αυτή χρησιμοποιεί τις στήλες που αποτελούν είσοδο για ερωτήσεις στη βάση, θα ονομάζουμε τις στήλες αυτές τρέχουσες για συντομία. Η μέθοδος διαγράφει όλες τις πλειάδες της τρέχουσας 'tuple' πού στην τρέχουσα στήλη έχουν τιμή που δεν ανήκει στην τρέχουσα στήλη μιας δεύτερης 'tuple' που ορίζεται από το χρήστη.

4.4.11.1 Ορίσματα – Παράμετροι

Το λογικό όνομα της δεύτερης 'tuple' είναι απαραίτητο.

4.4.11.2 Αλγόριθμος

Η μέθοδος απλώς 'μαρκάρει' ως διαγραφέντα τα στοιχεία της τρέχουσας στήλης, της τρέχουσας 'tuple' που δεν ανήκουν στην τρέχουσα στήλη της δεύτερης 'tuple'.

4.4.12 Η Tuple::column_copy

Το όνομα της μεθόδου είναι λίγο παραπλανητικό. Αν η τρέχουσα 'tuple' έχει μόνο μια στήλη (είναι δηλαδή σύνολο) αντιγράφει τα περιεχόμενα της τρέχουσας στήλης μιας δεύτερης 'tuple' που ορίζει ο χρήστης στην μοναδική στήλη της τρέχουσας 'tuple'. Αν η τρέχουσα 'tuple' έχει παραπάνω από μια στήλες τότε σχηματίζεται το καρτεσιανό γινόμενο της τρέχουσας 'tuple' με την τρέχουσα στήλη μόνο της δεύτερης tuple.

4.4.12.1 Ορίσματα – Παράμετροι

Χρειάζεται μόνο το λογικό όνομα της δεύτερης 'tuple'.

4.4.12.2 Αλγόριθμος

Απλή, κατά περίπτωση εφαρμογή των αλγορίθμων που περιγράφηκαν παραπάνω.

4.4.13 Η Tuple::swap

Αντιμεταθέτει 2 στήλες που ορίζει ο χρήστης μέσα στην τρέχουσα 'tuple'.

4.4.13.1 Ορίσματα – Παράμετροι

Οι αριθμοί των στηλών που αντιμετατίθενται είναι όλη η πληροφορία που χρειάζεται αυτή η μέθοδος.

4.4.13.2 Αλγόριθμος

Απλώς εναλλάσσονται οι τιμές των στοιχείων i και j του πίνακα μετάθεσης, όπου i και j οι αριθμοί στηλών που παρείχε ο χρήστης.

4.4.14 Η Tuple::clear_null

Λειτουργία της μεθόδου αυτής είναι να διαγράφει από την τρέχουσα 'tuple' τις πλειάδες που στην στήλη που ορίζει ο χρήστης έχουν την τιμή NULL. Η τιμή NULL μπορεί να προκύψει σε μια στήλη κατά την δημιουργία της μέσω μιας ερώτησης που είχε ως είσοδο μια άλλη στήλη. Αν για κάποιο από τα στοιχεία της στήλης-είσοδο η ερώτηση είχε ως απάντηση το κενό σύνολο αυτό παρίσταται με την τιμή NULL. Δεδομένου ότι η απάντηση σε κάθε ερώτηση που έχει ως είσοδο NULL είναι NULL, οι πλειάδες που σε κάποιο στήλη έχουν NULL λογικά θα έχουν και στις επόμενες αυτής, εκτός αν η 'tuple' έχει υποστεί αντιμεταθέσεις στηλών η συνδέσεις. Το NULL σε κάθε στήλη υπάρχει (όταν υπάρχει) ως ξεχωριστό στοιχείο.

4.4.14.1 Ορίσματα –Παράμετροι

Χρειάζεται ο αριθμός της στήλης η οποία θα εξεταστεί για τιμές NULL

4.4.14.2 Αλγόριθμος

Απλώς μαρκάρεται ως διαγραφέν το NULL στοιχείο στη στήλη που όρισε ο χρήστης.

4.4.15 Η Tuple::tuple_less_than (και οι συναφείς)

Η μέθοδος αυτή επιλέγει εκείνες της πλειάδες της τρέχουσας 'tuple' που η τιμή τους στην πρώτη στήλη που όρισε ο χρήστης είναι μικρότερη από την τιμή τους στην δεύτερη στήλη που επίσης όρισε ο χρήστης. Αν οι τιμές δεν είναι συγκρίσιμες, δηλαδή του ιδίου τύπου, η σύγκριση για την εκάστοτε πλειάδα θεωρείται ψευδής. Αντίστοιχη λειτουργία έχουν και οι: tuple_greater_than, tuple_equal, tuple_not_equal, tuple_less_equal, tuple_greater_equal.

4.4.15.1 Ορίσματα – Παράμετροι

Οι αριθμοί των στηλών που θα συγκριθούν δίδονται από το χρήστη.

4.4.15.2 Αλγόριθμος

Λαμβάνουμε μέσω της γνώστης μεθόδου – επαναλήπτη μια – μια της πλειάδες της 'tuple'. Όσες από τις πλειάδες ικανοποιούν τη συνθήκη προστίθενται στην επεκτεταμένη δομή των 2κ–1 στηλών όπως αυτή ορίστηκε στην tuple_union.

4.4.16 Η Tuple::tuple_less_const (και οι συναφείς)

Η μέθοδος αυτή επιλέγει εκείνες της πλειάδες της τρέχουσας 'tuple' που η τιμή τους στη τρέχουσα στήλη είναι μικρότερη από την σταθερά που όρισε ο χρήστης. Αν οι τιμές δεν είναι συγκρίσιμες, δηλαδή του ίδιου τύπου η σύγκριση για την εκάστοτε πλειάδα θεωρείται ψευδής. Ανάλογη λειτουργία έχουν και οι: tuple_greater_than_const, tuple_equal_const, tuple_not_equal_const, tuple_less_equal_const, tuple_greater_equal_const.

4.4.16.1 Ορίσματα – Παράμετροι

Πέραν της σταθεράς δεν απαιτείται καμία άλλη πληροφορία.

4.4.16.2 Αλγόριθμος

Τα στοιχεία της τρέχουσας στήλης εξετάζονται ένα προς ένα, εκείνα που δεν ικανοποιούν την συνθήκη 'μαρκάρονται' ως διαγραφέντα.

Κεφάλαιο 5

ΥΛΟΠΟΙΗΣΗ: BACK END

5. Αρχιτεκτονική του back end

Στο κεφάλαιο που ακολουθεί θα περιγράψουμε πώς από το πρόγραμμα DATALOG που κατασκευάζεται κατά την πρώτη φάση της μετάφρασης μιας ερώτησης TELQUEL προκύπτει το τελικό πρόγραμμα QI. Όπως έχουμε ήδη εξηγήσει η φύση του προβλήματος ευνοεί μια αναδρομική λύση του. Από τη ρίζα του δένδρου που προκύπτει από την πρώτη φάση της μετάφρασης με τον τρόπο που παρουσιάστηκε στο κεφάλαιο 3 προχωρούμε ακολουθώντας τα παρακάτω βήματα–αρχές:

1. Βρισκόμενοι στον κόμβο Χ του δένδρου αν αυτός είναι κόμβος φύλλο κατασκευάζουμε κατευθείαν τον κώδικα που του αντιστοιχεί σύμφωνα με τον τρόπο που περιγράφουμε για κάθε περίπτωση παρακάτω
2. Αν βρισκόμαστε σε ενδιάμεσο κόμβο του δένδρου αποφασίζουμε με έναν αλγόριθμο που επεξηγούμε στα επόμενα με ποια σειρά οι θυγατρικοί του κόμβοι θα υπολογιστούν πρώτα (αναδρομικά). Κάθε κόμβος που υπολογίζεται γράφει τον κώδικά του σε ένα αρχείο και επιστρέφει στον πατρικό του το όνομα της τελικής σχέσης–'tuple' στην οποία αποθήκευσε το αποτέλεσμα. Ο πατρικός κόμβος αφού υπολογιστούν οι θυγατρικοί του χρησιμοποιεί τα αποτελέσματα που παρήγαγαν για να κατασκευάσει το δικό του αποτέλεσμα, του οποίου τον επιπλέον κώδικα κατασκευής γράφει με την σειρά του στο αρχείο.

3. Για κάθε μεταβλητή – όρισμα του εκάστοτε κανόνα στο αποτέλεσμα που κατασκευάζει αντιστοιχεί από μια στήλη ακόμη και για τις ελεύθερες μεταβλητές.

4. Όταν ένας κανόνας υπολογίσει τη σχέση που του αναλογεί καθιστά τις μεταβλητές του περιορισμένες (bound) στο σύνολο των στοιχείων της στήλης που αντιστοιχεί στην καθεμιά. Το γεγονός αυτό το εκμεταλλεύεται ο αλγόριθμος παραγωγής κώδικα κάθε κόμβου. Όταν ο πατρικός κόμβος ζητάει από το θυγατρικό να παράγει τον κώδικα του ταυτόχρονα τον προμηθεύει και με την πληροφορία για το ποιες από τις μεταβλητές του θυγατρικού είναι ήδη δεσμευμένες. Επιπλέον τον πληροφορεί σε ποια στήλη, ποιας 'tuple' βρίσκεται το σύνολο τιμών με το οποίο είναι δεσμευμένη η εκάστοτε μεταβλητή. Ο θυγατρικός κόμβος εκμεταλλεύεται αυτή την πληροφορία για να περιορίσει το εύρος των τιμών στο οποίο θα αναζητήσει το αποτέλεσμα, μάλιστα κατά κανόνα οι κόμβοι (κανόνες DATALOG) χρειάζονται ένα ελάχιστο σύνολο δεσμευμένων μεταβλητών προκειμένου να μπορούν να παράγουν τον κώδικα που τους αντιστοιχεί. Η απαίτηση αυτή των κόμβων αποτελεί και το κριτήριο με το οποίο ο πατρικός προσδιορίζει την σειρά εκτέλεσης των θυγατρικών.

5. Αφότου ο κόμβος υπολογίσει το αποτέλεσμα εξετάζουμε αν είναι ρίζα του δένδρου, αν όντως έτσι είναι κρύβουμε από την σχέση αποτέλεσμα τις στήλες που αντιστοιχούν στις ελεύθερες μεταβλητές (tuple_not_project_column) και επιστρέφουμε το αποτέλεσμα στο χρήστη (return_tuples).

Στα επόμενα εξηγούμε με λεπτομέρεια τα βήματα που αναφέρουμε και το πώς υλοποιούνται.

5.1 Μετάφραση των EDB δηλώσεων

Είδαμε ότι ο κώδικας στα φύλλα του 'DATALOG δένδρου' υπολογίζεται κατευθείαν. Τον κώδικα που παράγεται για αυτά τα φύλλα, δηλ. τις EDB δηλώσεις θα περιγράψουμε παρακάτω.

Κάθε μια κατηγορία από τις EDB δηλώσεις έχει κάποια ορίσματα που μπορεί να είναι είτε μεταβλητές, είτε σταθερές. Εφόσον κάποιο όρισμα είναι σταθερά ο κόμβος πάντοτε το ξέρει καθώς γνωρίζει και ποια είναι η τιμή της σταθεράς. Αν όμως το όρισμα είναι μεταβλητή τότε υπάρχει η περίπτωση, όπως αναφέρθηκε και πριν, για μερικά από αυτά να υπάρχει η απαίτηση να είναι δεσμευμένα σε ένα σύνολο τιμών για να μπορούμε να παράγουμε τον κώδικα που τους αντιστοιχεί. Π.χ. ο κόμβος σύγκρισης <(X,Y) αν και το X και το Y είναι μεταβλητές πρέπει αυτές να είναι δεσμευμένες για να μπορούμε να παράγουμε κώδικα. Δεν είναι δυνατόν να κατασκευάσουμε μια σχέση με όλα τα ζεύγη τιμών που το πρώτο είναι μικρότερο από το δεύτερο. Ο κώδικας που αντιστοιχεί σε κάθε περίπτωση είναι:

5.1.1 Μετάφραση των δηλώσεων IN

Κάθε EDB της μορφής IN(X,Y) Κατασκευάζει μια σχέση με 2 στήλες αν και τα 2 ορίσματα είναι μεταβλητές και με μια στήλη αν μόνο ένα όρισμα είναι μεταβλητή. Ο κόμβος δεν απαιτεί κάποια από τις μεταβλητές να είναι δεσμευμένη αλλά αν είναι το εκμεταλλεύεται είτε αντιγράφοντας την στήλη από την οποία παίρνει τιμές η μεταβλητή, είτε καλώντας την εντολή ci (=column_intersection) για την στήλη που ήδη έχει κατασκευάσει και την στήλη στην οποία είναι ήδη δεσμευμένη η μεταβλητή. Ο κώδικας διακρίνει αρκετές περιπτώσεις ανάλογα με το ποια ορίσματα είναι μεταβλητές και ποια όχι, ποιες μεταβλητές είναι δεσμευμένες και ποιες όχι αλλά σε γενικές γραμμές γίνονται τα εξής:

- Αν η πληροφορία που έχουμε είναι ελάχιστη δηλ 2 μεταβλητές όχι δεσμευμένες κατά αρχήν δημιουργούμε ένα νέο σύνολο όπου τοποθετούμε όλα τις περιπτώσεις, όλων των κλάσεων (από M4_class μέχρι Token) δημιουργούμε έπειτα σε αυτό το σύνολο μια νέα στήλη και με είσοδο την πρώτη στήλη (το αρχικό σύνολο) εφαρμόζουμε την ερώτηση gac (=get all classes). Αποθηκεύουμε το

αποτέλεσμα (stor) Και δηλώνουμε ότι οι μετ/τες X,Y είναι bound στον πατρικό κόμβο.

- Αν ένα από τα ορίσματα είναι σταθερά το θέτουμε ως τρέχον αντικείμενο και ζητάμε είτε όλες του τις περιπτώσεις (εντολή gai) είτε όλες τις κλάσεις του (εντολή gac), ανάλογα. Αν το δεύτερο γνώρισμα είναι δεσμευμένη μεταβλητή δημιουργούμε την τομή της στήλης του με αυτή που μόλις κατασκευάσαμε (εντολή ci=column_intersection).

- Αν έχουμε δυο μεταβλητές εκ των οποίων μια τουλάχιστον είναι δεσμευμένη αντιγράφουμε την στήλη της σε νέο σύνολο–'tuple' (εντολή ccpy = column_copy), κατασκευάζουμε μια νέα στήλη και με είσοδο την πρώτη στήλη εφαρμόζουμε πάλι είτε την εντολή gai είτε την εντολή gac. Αν και η δεύτερη μεταβλητή είναι δεσμευμένη δημιουργούμε την τομή της στήλης που μόλις κατασκευάσαμε με την στήλη στην οποία είναι δεσμευμένη η μεταβλητή. Αν ξεκινήσαμε αν κατασκευάζουμε την 'tuple' από το όρισμα Y αντιμεταθέτουμε τις στήλες.

5.1.2 Μετάφραση των δηλώσεων ISA

Οι δηλώσεις αυτές έχουν την μορφή ISA(X,Y). Υπάρχει 1–1 αντιστοιχία με τις δηλώσεις IN συνεπώς και ο αλγόριθμος παραγωγής κώδικα είναι πλήρως ανάλογος με την διαφορά ότι όπου χρησιμοποιήσαμε την εντολή gac στον κώδικα του κόμβου IN τώρα χρησιμοποιούμε την εντολή gasc (=get all super classes) και όπου την εντολή gai την εντολή gasb (= get all subclasses).

5.1.3 Μετάφραση των δηλώσεων ATTRIBUTE

Ο κώδικας για την παραγωγή των ATTRIBUTE δηλώσεων είναι ο πιο δύσκολα από όλους να παραχθεί. Η διαδικασία διαχωρίζεται σε περιπτώσεις ανάλογα με τον τύπο του κόμβου (σύζευξης ή διάζευξης), τα τυχόν transitivity flags (μόνο το '+' υποστηρίζεται για την ώρα) και το αν και ποια ορίσματα είναι σταθερές η μεταβλητές καθώς και ποιες από τις μεταβλητές είναι δεσμευμένες.

Υπενθυμίζουμε ότι οι κόμβοι ATTRIBUTE έχουν την μορφή:

A.[meta](&I'I')([-]category$_1$,...,[-]category$_n$)[+I*I?](X,L,Y).

Διευκρινίζουμε εδώ ότι οι κόμβοι ATTRIBUTE απαιτούν προκειμένου να υπολογίσουν κώδικα ΟΙ όλες οι μεταβλητές που αναφέρονται στις κατηγορίες τους να είναι δεσμευμένες. Οι ίδιοι οι κόμβοι επιστρέφουν το αποτέλεσμα πάνω στο 'σχήμα' ‹X.L,Y› η για την ακρίβεια στις μεταβλητές αυτού. Ο κώδικας παράγεται με την παρακάτω ακολουθία βημάτων:

- Δημιουργούμε n σύνολα, ένα για κάθε κατηγορία, με τις περιπτώσεις της εκάστοτε κατηγορίας στο καθένα από αυτά στην επιθυμητή στάθμη δηλαδή meta+1 στάθμες πιο κάτω από την στάθμη της κάθε μιας κατηγορίας. Αυτό σημαίνει ότι καλούμε την εντολή gai meta+1 φορές για κάθε κατηγορία, κάθε φορά για εφαρμοστεί στο σύνολο που κατασκεύασε στην προηγούμενη κλήση της. Τα παραπάνω είναι εφικτά διότι όλες οι κατηγορίες είναι είτε σταθερές είτε δεσμευμένες μεταβλητές

- Χωρίζουμε (νοητά) τα σύνολα σε δυο ομάδες: στην πρώτη βρίσκονται αυτά που αντιστοιχούν σε 'καταφατικές' κατηγορίες ενώ στην δεύτερη αυτά που αντιστοιχούν σε κατηγορίες με άρνηση. Αν έχουμε κόμβο σύζευξης σχηματίζουμε την τομή όλων των συνόλων της πρώτης ομάδας (σύνολο Α) και την ένωση όλων των συνόλων της δεύτερης (σύνολο Β), έπειτα σχηματίζουμε την διαφορά των δύο αυτών συνόλων Α−Β. Εφόσον το όρισμα L είναι δεσμευμένο ή σταθερά σχηματίζουμε την τομή του (σύνολο Γ) με το Α−Β αλλιώς Γ=Α−Β. Αν έχουμε κόμβο διάζευξης σχηματίζουμε την ένωση των συνόλων της πρώτης ομάδας (σύνολο Α) και την τομή των συνόλων της δεύτερης (σύνολο Β). Σ' αυτή την περίπτωση καλό είναι και το όρισμα L να είναι δεσμευμένο οπότε το σύνολο Γ προκύπτει από την τομή του Α με το ‹L›−Β αλλιώς κατασκευάζουμε το σύνολο Δ ώστε να περιέχει όλα τα γνωρίσματα της βάσης και τότε Γ = Α U (Δ−Β).

- Η συνέχεια εξαρτάται από το αν υπάρχει ή όχι transitivity flag. Αν δεν υπάρχει τότε στο σύνολο Γ δημιουργούμε μια νέα στήλη (από 'tuple' μιας στήλης αποκτάει και δεύτερη). Με είσοδο την πρώτη στήλη εφαρμόζουμε την ερώτηση gfv (=get from value). Αν το όρισμα X είναι δεσμευμένη μεταβλητή ή σταθερά δημιουργούμε την τομή της στήλης που περιέχει της τιμές του (ci = column_intersect) με την τελευταία στήλη του συνόλου Γ. Αντιμεταθέτουμε τις δύο στήλες του συνόλου–'tuple' Γ. (‹L,X›–› ‹X,L›). Δημιουργούμε και τρίτη στήλη στο Γ. Με είσοδο την δεύτερη στήλη εφαρμόζουμε την ερώτηση gtv (=get to value). Αν το όρισμα Y είναι δεσμευμένη μεταβλητή ή σταθερά δημιουργούμε την τομή της στήλης που περιέχει της τιμές του (ci = column_intersect) με την τελευταία στήλη του συνόλου Γ.

- Αν το transitivity flag είναι το '+' χρησιμοποιούμε την εντολή tal για αναδρομική διάσχιση των γνωρισμάτων. Κατασκευάζουμε ένα νέο σύνολο το F με τον ακόλουθο τρόπο: Αν X σταθερά ή δεσμευμένη σε μια στήλη τότε F=X αλλιώς το Φ προκύπτει ως το σύνολο – απάντηση της ερώτησης gfv πάνω στο σύνολο Γ. Στην συνέχεια δημιουργούμε μια νέα στήλη για το σύνολο F (γίνεται 'tuple') και εφαρμόζουμε την εντολή tal με είσοδο την πρώτη στήλη. Πριν την εφαρμογή της tal θέτουμε κατάλληλα τις συνθήκες της ως εξής: Θέτουμε τις συνθήκες stlc fail και sfvc fail για να απαγορεύσουμε διάσχιση γνωρισμάτων προς τα πίσω. Θέτουμε sflc bing (sys O, set Γ) έτσι ώστε κάθε γνώρισμα που διασχίζουμε να ανήκει στο σύνολο Γ. Αν το όρισμα Y είναι σταθερά η δεσμευμένη μεταβλητή κατασκευάζουμε ένα σύνολο με τις τιμές που είναι επιτρεπτές για το Y, έστω το σύνολο N, θέτουμε stvc bing (sys O, set N) έτσι ώστε οι τιμές των γνωρισμάτων που διασχίζουμε να είναι μέσα στο σύνολο N. Αν το όρισμα Y δεν έχει περιοριστεί σε ένα σύνολο τιμών τότε θέτουμε απλώς stvc succ για να μην παίζει ρόλο η τιμή του γνωρίσματος στην διάσχιση του. Δημιουργούμε μια νέα στήλη

για το σύνολο F. Με είσοδο την δεύτερη στήλη εφαρμόζουμε την ερώτηση gtv.

- Κρύβουμε τις στήλες του F (αν tf='+') ή του Γ (αν δεν υπάρχει tf) που αντιστοιχούν σε ορίσματα που είναι σταθερές.
- Ελευθερώνουμε όλα τα προσωρινά σύνολα.
- Ενημερώνουμε τον πατρικό ότι όλα τα ορίσματα – μεταβλητές από το σύνολο {X,L,Y} είναι δεσμευμένα στις αντίστοιχες στήλες του συνόλου F ή Γ.

5.1.4 Μετάφραση των δηλώσεων σύγκρισης

Οι δηλώσεις συγκρίσεις έχουν την μορφή OP(X,Y). Για να μπορέσει να παραχθεί κώδικας θα πρέπει και τα δύο ορίσματα είτε να είναι σταθερές είτε να έχει περιοριστεί το σύνολο τιμών τους σε κάποια στήλη κάποιας 'tuple'. Αν και τα 2 ορίσματα είναι μεταβλητές ενεργούμε ως εξής: σχηματίζουμε με χρήση της ccpy (column_copy) το καρτεσιανό γινόμενο των 2 στηλών που αντιστοιχούν στα ορίσματα, έστω ότι προκύπτει η σχέση R. Καλούμε ανάλογα με τον τελεστή OP πάνω στη σχέση R την κατάλληλη tC‹OP› (tuple column) με ορίσματα τις στήλες 0 και 1.

Αν μόνο ένα όρισμα είναι μεταβλητή αντιγράφουμε μόνο τη στήλη που του αντιστοιχεί στη μεταβλητή σε νέο σύνολο R (ccpy) και σ' αυτή ανάλογα με τον τελεστή εφαρμόζουμε την κατάλληλη tc‹OP› (tuple const) με παράμετρο το δεύτερο όρισμα της δήλωσης που είναι σταθερά.

5.2 Μετάφραση των IDB δηλώσεων

Περιγράφουμε τώρα πως μια IDB δήλωση χρησιμοποιώντας τα αποτελέσματα των θυγατρικών της δηλώσεων κατασκευάζει το δικό της αποτέλεσμα αφού πρώτα αποφασίσει με ποια σειρά θα 'καλέσει' τους θυγατρικούς της κόμβους να υπολογίσουν τα αποτελέσματα τους.

5.2.1 Προσδιορισμός σειράς εκτέλεσης

Έχουμε επισημάνει ότι μερικοί EDB κόμβοι θα πρέπει να έχουν περιορισμένα σε ένα αρχικό σύνολο τιμών τα ορίσματα τους προκειμένου να μπορέσουν να παράγουν το δικό τους κώδικα. Αυτό έχει σαν συνέπεια και οι IDB κόμβοι να έχουν την ίδια απαίτηση. Το πρώτο πράγμα λοιπόν που θα πρέπει να μπορεί να κάνει ένας κόμβος είναι να μπορεί να απαντήσει ποιο υποσύνολο των μεταβλητών του είναι απαραίτητο για να μπορεί να λειτουργήσει καθώς και ποιες θα είναι διαθέσιμες μετά την περάτωση του. Τα σύνολα αυτά τα ονομάζουμε *'σύνολα απαιτούμενων και παρεχόμενων μεταβλητών'* των οποίων το τρόπο υπολογισμού αυτών εξηγούμε στη συνέχεια. Η ιδέες είναι βασισμένες στον ορισμό των συνόλων First και Follow που χρησιμοποιούνται για το LR parsing[Aho86]

5.2.1.1 Παρεχόμενες και απαιτούμενες μεταβλητές

Ήδη έχουμε εξηγήσει πως τα σύνολα αυτά ορίζονται για τους EDB κόμβους, Τώρα θα τα ορίσουμε και για τους IDB κόμβους. Ο ορισμός όπως θα ανέμενε κανείς είναι αναδρομικός. Ανάλογα με το είδος του κόμβου (σύζευξη ή διάζευξη) και μας ενδιαφέρει η άρνηση του κόμβου ή όχι το σύνολο Req των 'απαιτούμενων μεταβλητών' ορίζεται ως:

- Για κόμβους που έχουν άρνηση Req={όλες οι δεσμευμένες μετ/τες του κόμβου} U R'. R' = Req κόμβου χωρίς την άρνηση.
- Για καταφατικούς κόμβους σύζευξης Req = (U Req_i) – (U $Prov_i$) όπου Req_i το σύνολο των 'απαιτούμενων μεταβλητών' του θυγατρικού κόμβου i και το $Prov_i$ αντίστοιχο σύνολο των 'παρεχόμενων μεταβλητών'.
- Για καταφατικούς κόμβους διάζευξης Req = (U Req_i) όπου Req_i όπως προηγουμένως

Το σύνολο των παρεχόμενων μεταβλητών για κάθε κόμβο ορίζεται ως {Μεταβλητές του κόμβου} – Req.

5.2.1.2 Εύρεση διάταξης

Έχοντας προσδιορίσει το σύνολο των παρεχόμενων και των απαιτούμενων μεταβλητών για κάθε κόμβο είμαστε σε θέση να εφαρμόσουμε τον αλγόριθμο για την εύρεση της σειράς εκτέλεσης των θυγατρικών κόμβων ενός κόμβου. Ξεκινώντας πρέπει να αναφέρουμε ότι για να λειτουργήσει ο αλγόριθμος θα πρέπει ο πατρικός κόμβος να έχει στην διάθεση του όλες τις απαιτούμενες μεταβλητές του, δηλαδή τα 'tuples' και τους αριθμούς των στηλών στα οποία βρίσκονται τα σύνολα από τα οποία πρέπει να παίρνουν τιμές. Τότε ο αλγόριθμος που σέβεται το γράφο προτεραιότητας ο οποίος προκύπτει σύμφωνα με το [Korth91] ακολουθεί τα βήματα:

1. Θέσε A = ‹απαιτούμενες μεταβλητές του πατρικού κόμβου›
2. Βρες τον πρώτο θυγατρικό κόμβο i για τον οποίο $Req_i \subseteq A$.
3. Προμήθευσε τον κόμβο i με τις μεταβλητές του συνόλου $A \cap$ {μεταβλητές του κόμβου i} (εννοούμε tuples και αριθμούς στηλών) και εκτέλεσε τον κώδικα του κόμβου.
4. Αν κόμβος σύζευξης, θέσε $A = A \cup Prov_i$ κατέγραψε κάπου την 'tuple' και τις στήλες από τις οποίες 'δεσμεύονται' οι μεταβλητές του $Prov_i$ (για να μπορούμε να τις προμηθεύσουμε σε επόμενους κόμβους)
5. "Διέγραψε" τον κόμβο i από το σύνολο των θυγατρικών κόμβων και αν υπάρχουν ακόμα θυγατρικοί κόμβοι επανέλαβε από το βήμα 2.

Προκειμένου να λειτουργήσει ο παραπάνω αλγόριθμος θα πρέπει να μην υπάρχουν κυκλικές εξαρτήσεις μεταξύ των θυγατρικών κόμβων. Δηλ να μην υπάρχει π.χ. μεταβλητή που να ανήκει στο σύνολο των παρεχόμενων μεταβλητών του κόμβου Α και το σύνολο των απαιτούμενων μεταβλητών του κόμβου Β και ταυτόχρονα να υπάρχει μεταβλητή που να ανήκει και στο σύνολο των παρεχόμενων μεταβλητών του κόμβου Β και στο σύνολο των απαιτούμενων μεταβλητών του κόμβου Α.

5.2.2 Δηλώσεις σύζευξης

Εφόσον έχει κατασκευαστεί ο κώδικας και τα αποτελέσματα των θυγατρικών κόμβων μπορούμε να κατασκευάσουμε τον επιπλέον κώδικα που αντιστοιχεί στον πατρικό κόμβο. Για τους κόμβους σύζευξης ο κώδικας κατασκευάζεται ως εξής:

- Αν ο κόμβος είναι καταφατικός (όχι άρνησης) τότε σχηματίζουμε την σύνδεση (join) όλων των αποτελεσμάτων των θυγατρικών κόμβων. Οι στήλες πάνω στις οποίες θα γίνει η σύνδεση είναι αυτές που αντιστοιχούν στις ίδιες μεταβλητές. Έπειτα κάνουμε προβολή μόνο των στηλών που αντιστοιχούν σε μεταβλητές του πατρικού κανόνα και ελευθερώνουμε τα 'tuples' στα οποία φυλάγονται τα αποτελέσματα των θυγατρικών κανόνων. Τέλος κάνουμε τα απαραίτητα swap για να διατάξουμε τις μεταβλητές της σύνδεσης με την σειρά που υπάρχουν στον πατρικό κανόνα. Για παράδειγμα στον κανόνα:

  R(X,Y):- R1(X,Y;Z), R2(A,B;Z)

  έστω ότι οι πλειάδες t_1 και t_2 φυλάσσουν τα αποτελέσματα των κανόνων R_1 και R_2. Τότε η σύνδεση τους γίνεται με τον κώδικα:

  apon t1 //apply on tuple t1 -τρέχουσα 'tuple' η t1

  trjp //tuple reset join pos -καθάρισε τον πίνακα σύνδεσης της t1

  tsjp 2,2 //tuple set join pos - σύνδεση της στήλης #2 της t1 (τρίτη στήλη) με την στήλη #2 της t2 (και οι δύο αντιστοιχούν στη μεταβλητή Z)

  tj t2 //tuple join - Εκτέλεσε την σύνδεση με την t2, προκύπτον σχήμα: ‹X,Y,Z,A,B›

  fs t2 //free set - ελευθέρωσε την t2

  tnpc 4 //tuple not project column - κρύψε την στήλη B

  tnpc 3 // κρύψε την A

  tnpc 2 // κρύψε την Z

Το αποτέλεσμα βρίσκεται στην t_1, (την 'tuple' του πρώτου παιδιού)

- Αν ο κόμβος είναι άρνησης τότε κατασκευάζουμε το καρτεσιανό γινόμενο όλων των μεταβλητών του κόμβου χρησιμοποιώντας επανειλημμένως την ccpy(column copy) και την πληροφορία για το που, σε ποιες στήλες, βρίσκονται τα 'πεδία τιμών' των μεταβλητών του κόμβου. (Αν για μια μεταβλητή δεν υπάρχει σύνολο τιμών θεωρούμε όλη την βάση) Έστω T η tuple που προκύπτει. Ακολούθως υπολογίζουμε τον κόμβο σαν να ήταν καταφατικός, έστω ότι το αποτέλεσμα τοποθετείται στην 'tuple' N τότε η 'tuple' που αντιστοιχεί στην άρνηση είναι η T–N. Π.χ. αν είχαμε την άρνηση του κανόνα στο προηγούμενο παράδειγμα και γνωρίζαμε ότι η μεταβλητή X παίρνει τιμές από την στήλη #2 της s_1 και η Y από την στήλη #3 της s_2 ο συνολικός κώδικας για την άρνηση θα ήταν:

```
sgn //set get new -νέο σύνολο
stor nt //store -αποθήκευσε
apon s1 // apply on s1
tsi 2 // tuple set input - τρέχουσα στήλη η #2
apon nt
ccpy s1 //column copy
apon s2
tsi 3
apon nt
ccpy s2
‹κώδικας όπως προηγουμένως›
apon nt
sd t1 //set (tuple) difference - διαφορά με την t1
fs t1 //free set
```

Τώρα το αποτέλεσμα βρίσκεται στην 'tuple' nt.

5.2.3 Δηλώσεις διάζευξης

Δεδομένων των αποτελεσμάτων των θυγατρικών κόμβων ο επιπλέον κώδικας στον πατρικό κόμβο για τους κόμβους διάζευξης κατασκευάζεται ως εξής:

- Αν ο κόμβος είναι καταφατικός (όχι άρνησης) τότε σχηματίζουμε την ένωση (union) όλων των αποτελεσμάτων των θυγατρικών κόμβων. Για να γίνει αυτό από κάθε αποτέλεσμα θυγατρικού κόμβου κρύβουμε τις στήλες που αντιστοιχούν σε μεταβλητές που δεν υπάρχουν στον πατρικό κόμβο. Επίσης αν στον πατρικό κόμβο υπάρχουν μεταβλητές που δεν υπάρχουν στο θυγατρικό συμπληρώνουμε στήλες για αυτές με χρήση της ccpy. Οι στήλες αυτές προκύπτουν με 2 τρόπους είτε έχουν δοθεί στον πατρικό κόμβο είτε όχι. Στην δεύτερη περίπτωση αναγκαστικά θα πρέπει να κατασκευαστούν και η μόνη επιλογή που έχουμε είναι να κατασκευάσουμε στήλες με περιεχόμενο ολόκληρη την βάση. Ακολούθως κάνουμε τα απαραίτητα swap ώστε όλες οι 'tuple' των θυγατρικών κόμβων να έχουν το ίδιο σχήμα με τον πατρικό τους. Τέλος εκτελούμε την ένωση όλων των θυγατρικών και ελευθερώνουμε τα 'tuples' στα οποία φυλάγονται τα αποτελέσματα των θυγατρικών κανόνων. Για παράδειγμα στον κανόνα:

  $R(X,Y;A,B):- R_1(X,Y;A) \mid R_2(X,Y;B)$

  έστω ότι οι πλειάδες t_1 και t_2 φυλάσσουν τα αποτελέσματα των κανόνων R_1 και R_2, και πως ο R γνωρίζει ότι η μεταβλητή B βρίσκεται στην στήλη #1 της tuple u. Τότε η ένωση τους γίνεται με τον κώδικα:

  apon u // apply on

  tsi 1 //tuple set input - τρέχουσα στήλη της u η 1

  apon t_1

  ccpy u

  apon t_2

ccpy all // All = σύνολο με όλα τα αντικείμενα της βάσης

swap 2 3 // Αντιμετάθεσε τις στήλες A,B

apon t_1

su t_2 //set (tuple) union

fs t_2

fs all

- Αν ο κόμβος είναι άρνησης τότε ακολουθούμε την ίδια διαδικασία που ακολουθήσαμε και στους κόμβους σύζευξης.

Κεφάλαιο 6

ΕΠΙΛΟΓΟΣ

6. Συμπεράσματα

Παρουσιάσαμε την TELQUEL, μια οντοκεντρική ερωτηματική γλώσσα με σύνταξη παρόμοια της SQL, για το σύστημα SIS. Η γλώσσα υποστηρίζει όλες τις λειτουργίες της σχεσιακής άλγεβρας και αρκετά από τα αναδρομικά χαρακτηριστικά που παρέχει το βασικό σύστημα ερωτήσεων του SIS, το QI. Η γλώσσα παρέχει την δυνατότητα για διατύπωση ερωτήσεων σε υψηλό επίπεδο και δεν αναγκάζει το χρήστη να καταφεύγει σε χρήση μακρόσυρτων ερωτήσεων με μνημονικά ονόματα και στοιχειώδεις λειτουργίες. Αυτό έχει ως αποτέλεσμα τη μείωση της πιθανότητας λάθους και τη διατύπωση πιο κατανοητών και ευανάγνωστων ερωτήσεων.

Δεδομένης της ύπαρξης του QI θέλαμε να χρησιμοποιήσουμε τη λειτουργικότητα που παρέχει. Προς τούτο προβήκαμε σε 2 ενέργειες:

1. Επέκταση του QI ώστε να υποστηρίζει τις σχέσεις και όλους τους τελεστές της σχεσιακής άλγεβρας ως φυσική επέκταση της δομής του συνόλου που ήδη υποστήριζε.
2. Καθορισμό της διαδικασίας της μετάφρασης από ερώτηση TELQUEL σε ερώτηση QI. Η μετάφραση γίνεται σε δύο φάσεις. Στην πρώτη φάση η ερώτηση μετασχηματίζεται σε πρόγραμμα DATALOG και στη δεύτερη σε κώδικα QI.

Περιγράψαμε τόσο τον τρόπο με τον οποίο υλοποιήσαμε τις επεκτάσεις στο QI όσο και τις δύο φάσεις της μετάφρασης.

Η υλοποίηση της γλώσσας στην μέχρι στιγμής μορφή της παρουσιάζει τις εξής αδυναμίες:

1. Δεν έχουν υλοποιηθεί οι τελεστές '*' και '?' όπως προβλέπεται από την σχεδίαση της γλώσσας[TEL].
2. Πρέπει ο αλγόριθμος εύρεσης σειράς εκτέλεσης των θυγατρικών κόμβων (κεφάλαιο 5) να γενικευθεί ώστε να αντιμετωπίζει και περιπτώσεις κυκλικών εξαρτήσεων.

6.1 Βελτιώσεις – επεκτάσεις

Το πρώτο που θα έπρεπε να γίνει είναι η εξάλειψη των αδυναμιών που αναφέρονται στο προηγούμενο υποκεφάλαιο. Επόμενα βήματα που θα μπορούσαν να γίνουν είναι η πλήρης υποστήριξη των χρονικών δυνατοτήτων του QL (τελεστές before, after, overlaps κλπ.) καθώς και βελτίωση των αναδρομικών χαρακτηριστικών, π.χ. αναζήτηση μέχρι συγκεκριμένο βάθος. Απώτερος σκοπός θα μπορούσε να είναι η επέκταση της ώστε να έχει τις εκφραστικές δυνατότητες του QL.

Επίσης θα μπορούσαμε να ζητήσουμε την ενσωμάτωση αθροιστικών συναρτήσεων (aggregate functions) όπως sum, count κ.λ.π. Τέλος θα ήταν δυνατόν να επιδιωχθεί η μέγιστη δυνατή συμβατότητα με την OQL, το πρότυπο που προτείνει το ODMG [ODMG93] μια και πλήρης δεν μπορεί να επιτευχθεί.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[Abit95] S. Abiteboul, R. Hull, V Vianu, Foundations of databases, Addison Wesley, 1995

[Aho86] A.V. Aho, R. Sethi, J.D. Ullman, Compilers: Principles, Techniques and Tools. Addison-Wesley, Reading, Mass, 1986

[Bert93] E. Bertino, L. Martino, Object-Oriented Database Systems - Concepts and Architectures, Addison-Wesley, 1993.

[ODMG93] R. G.G. Cattell (ed) The Object Database Standard : ODMG-93, Release 1.2, Morgan Kaufmann Publishers, Inc., 1995.

[Kim] W. Kim, Introduction to Object Oriented Databases, The MIT Press, 1990

[Korth91] H. F. Korth, A. Silberschatz, Database System Concepts, 2nd Edition, McGraw-Hill, 1991

[SQL] C. J. Date, A Guide to the SQL Standard, 2nd Edition, Addison-Wesley, Reading, MA (1989)

[Strou91] B. Stroustrup, The C++ Programming Language, Addison-Wesley, 1991

[Βασ97] Π. Βασιλειάδης, Τ. Σελλής, "Αντικειμενοστρεφή Συστήματα Διαχείρισης Βάσεων Δεδομένων", Παπασωτηρίου και Εθνικό Μετσόβιο Πολυτεχνείο, 1997

[Anal] A. Analyti, P. Costantopoulos, N. Spyratos, SPECIALIZATION BY RESTRICTION AND SCHEMA DERIVATIONS, *Information Systems* 23(1):1-38, 1998

[Codd] E. F. Codd, A Relational Model for Large Shared Data Banks, *Communications of the ACM*, 13(6):377-387, June 1970

[Const94] P. Constantopoulos, M. Doerr, The Semantic Index System: A brief presentation, *Technical Report Institute of Computer Science Foundation for Research and Technology - Hellas (1994)*

[Const95] P. Constantopoulos, M. Doerr, Component classification in the software information base, In: O. Nierstrasz, D. Tsichritzis (Eds.), *Object-Oriented Software Composition*, Prentice-Hall, 1995

[Dar] S. Dar, H. N. Gehani, V. H. Jagadish. *CQL++: An SQL for a C++ based Object-Oriented DBMS,* Technical Report, AT&T Bell Laboratories, Murray Hill.

[GAIN] *SIS – Graphical Analysis Interface User's Manual*, ver 2.2, Institute of Computer Science, Foundation for Research and Technology – Hellas, 1998

[Lex] M.E. Lesk, M. Schmidt. Lex – A Lexical Analyzer Generator. In *Unix Programmer's Manual, Tenth Edition,* AT&T Bell Laboratories, 1989

[Myl90] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis. Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems*, 8(4):352–362, October 1990

[PQl98] C. Dadouris, M. Doerr, A. Kladogenis, C. Georgis, *SIS – Application Programmatic Interface*, ver 2.1.1, Institute of Computer Science, Foundation for Research and Technology – Hellas, 1998

[Ql98] C. Dadouris, M. Doerr, S. Kizlaridou, N. Prekas, *SIS – Query Interpreter: An Interactive Program to Use PQl functions*, ver 2.0, Institute of Computer Science, Foundation for Research and Technology – Hellas, 1998

[SISTELOS] M. Doerr, P. Klimathianakis, Manos Theodorakis, *SIS – Data Entry Language User's Manual*, ver 2.1, Institute of Computer Science, Foundation for Research and Technology – Hellas, 1995

[TEL] G. Hillebrand, P. Klimathianakis, Quering The Software Information Base, Internal Report, Institute of Computer Science, Foundation for Research and Technology – Hellas, 1995

[Yacc] S.C. Johnson, R. Sethi, Yacc: A Parser Generator. In *Unix Programmer's Manual, Tenth Edition,* AT&T Bell Laboratories, 1989

[Χρι94] Μ. Χριστοφοράκη, *Τεκμηρίωση πολιτιστικών αγαθών με το σύστημα ΚΛΕΙΩ*, Εργασία Μεταπτυχιακού Διπλώματος Ειδίκευσης, Πανεπιστήμιο Κρήτης – Σχολή θετικών Επιστημών – Τμήμα Επιστήμης Υπολογιστών, 1994