ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Κατασκευή ενός Πολυμορφικού Συστήματος Διαχείρισης Βάσεων Δεδομένων: Σχεδιασμός και Υλοποίηση Υποσυστημάτων Αποθήκευσης και Εκτέλεσης Τελεστών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΓΕΡΑΓΓΕΛΟΣ Σ. ΣΤΕΦΑΝΟΣ

Επιβλέπων : Τίμος Σελλής
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2009

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Κατασκευή ενός Πολυμορφικού Συστήματος Διαχείρισης Βάσεων Δεδομένων: Σχεδιασμός και Υλοποίηση Υποσυστημάτων Αποθήκευσης και Εκτέλεσης Τελεστών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΓΕΡΑΓΓΕΛΟΥ ΣΤΕΦΑΝΟΥ

Επιβλέπων : Τίμος Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 26η Οκτωβρίου 2009.

.....	.....	.....
Τίμος Σελλής Καθηγητής Ε.Μ.Π.	Ιωάννης Βασιλείου Καθηγητής Ε.Μ.Π.	Νεκτάριος Κοζύρης Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2009

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Συστημάτων Βάσεων και Γνώσεων Δεδομένων (ΕΒΓΔ) του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Υπεύθυνος καθηγητής είναι ο κ. Τίμος Σελλής, τον οποίο αισθάνομαι την ανάγκη να ευχαριστήσω για την εμπιστοσύνη και το ενδιαφέρον που μου έδειξε καθ' όλη τη διάρκεια της εκπόνησης της παρούσας εργασίας. Θα ήθελα, επίσης, να ευχαριστήσω τον συν-επιβλέποντα της διπλωματικής, υποψήφιο διδάκτορα του ΕΒΓΔ Γιάννη Ρούσσο, καθώς με βοήθησε σημαντικά και αφιέρωσε αρκετό χρόνο και γνώση στην προσπάθειά μου αυτή. Τέλος, ευχαριστώ εκ βάθους καρδίας τους γονείς μου, Σωτήρη και Αλεξάνδρα, και την αδελφή μου, Αθηνά, για την ουσιαστική συμπαράστασή τους στη έως τώρα πορεία μου, αλλά και γενικότερα στη ζωή μου.

Οκτώβριος 2009

Γεράγγελος Σ. Στέφανος

Περίληψη

Ο στόχος της παρούσας διπλωματικής είναι ο σχεδιασμός του πρώτου ερευνητικού σχεσιακού Συστήματος Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ), το οποίο να ενσωματώνει το context στη διαχείριση της πληροφορίας. Συγκεκριμένα, η εν λόγω διπλωματική περιλαμβάνει το σχεδιασμό και την υλοποίηση των Υποσυστημάτων "**Εκτέλεσης Πλάνων**", "**Εκτέλεσης Τελεστών**" και "**Αποθήκευσης context-aware Δεδομένων**". Το σύστημα που υλοποιήθηκε προσφέρει τη δυνατότητα δημιουργίας context-aware βάσεων δεδομένων και εκτέλεσης ερωτημάτων.

Το συγκεκριμένο σύστημα αποτελείται από τις ακόλουθες δομικές μονάδες: το "Υποσύστημα **Αποθήκευσης και Διεπαφής με το Σχεσιακό ΣΔΒΔ** (DataBase Layer)", που είναι υπεύθυνο για την επικοινωνία του συστήματος με ένα απλό σχεσιακό ΣΔΒΔ, το "Υποσύστημα του **Καταλόγου** (Catalogue)", ο οποίος διαχειρίζεται τα μετα-δεδομένα που αφορούν στο context και στα context-aware δεδομένα, το "Υποσύστημα **Διαχείρισης Δεδομένων** (Data)", το οποίο αναλαμβάνει τις εισαγωγές/διαγραφές/ενημερώσεις δεδομένων στο σύστημα, το "Υποσύστημα **Εκτέλεσης Τελεστών** (Operators)", που είναι υπεύθυνο για την εκτέλεση όλων των context-aware τελεστών, που έχουν οριστεί για να υποστηρίζεται το μοντέλο, και το "Υποσύστημα **Εκτέλεσης Πλάνων** (Query Executor)", το οποίο δέχεται ένα ερώτημα, με τη μορφή ενός πλάνου, και αναλαμβάνει την εκτέλεσή του.

Σημαντικό κομμάτι της διπλωματικής ήταν ο σχεδιασμός σε επίπεδο συστήματος (system software engineering) και σε επίπεδο διαχείρισης βάσεων δεδομένων (data base management). Τέλος, κατά τη διάρκεια της παρούσας διπλωματικής πραγματοποιήθηκαν δύο διαφορετικές υλοποιήσεις στο σχεσιακό ΣΔΒΔ. Πραγματοποιήθηκαν, επίσης, πειράματα σε μία context-aware βάση δεδομένων, προκειμένου να μελετηθεί η απόδοση του συστήματος και να συγκριθούν τα αποτελέσματα για τις δύο διαφορετικές υλοποιήσεις.

Λέξεις Κλειδιά: context, συμφραζόμενα, ερμηνευτικό περιβάλλον, context-aware συστήματα, context-aware ΣΔΒΔ, πολυμορφικό ΣΔΒΔ, polymorphs, morphs, context-aware τελεστές.

Abstract

The aim of this Diploma Thesis is the design and implementation of the first research relational Data Base Management System (DBMS), which integrates the context in the management of information. Particularly, this diploma thesis includes the design and implementation of the Subsystems of "**Query Execution**", "**Operators Execution**" and "**Storage of context-aware Data**". The implemented system offers the possibility to create context-aware data bases and execute queries.

This system is composed of the following structural modules: a **DataBase Layer**, which is responsible for the communication of the system with the simple relational DBMS, the **Catalogue**, which manages the metadata that refers to the context and to the context-aware data, the module of **Data**, which takes over the inserts/deletes/updates of data in the system, the module of **Operators Execution**, which is responsible for the execution of all context-aware operators, that have defined in order the model to be fully supported, and the module of **Query Execution**, which takes a query, as a plan, and executes it.

An important part of this diploma thesis was the design in the system level (system software engineering) and in the data base management level. Finally, during this diploma thesis two different implementations have been made in the relational DBMS. Also, experiments have been made in a context-aware data base, in order to study the performance of the system and compare the results of the different implementations.

Keywords: context, context-aware systems, context-aware DBMS, polymorphic DBMS, polymorphs, morphs, context-aware operators.

Πίνακας περιεχομένων

1

Περιγραφή θέματος

Στο πρώτο αυτό κεφάλαιο, συγκεκριμένα στην ενότητα 1.1, αναλύουμε το αντικείμενο της παρούσας διπλωματικής εργασίας. Στην ενότητα 1.2 προχωρούμε σε μία επισκόπηση της οργάνωσης του τόμου.

1.1 Αντικείμενο της διπλωματικής

Κατά τη διάρκεια των τελευταίων δεκαετιών η ταχύτατη ανάπτυξη του διαδικτύου και των αντίστοιχων τεχνολογιών έχει προκαλέσει τη δημιουργία μεγάλου πλήθους ετερογενών συλλογών δεδομένων, τα οποία συνεχώς διαφοροποιούνται ανάλογα με τις εξωτερικές παραμέτρους. Οι χρήστες (καταναλωτές της πληροφορίας) θεωρούν πλέον ότι θα τους επιστραφούν εξατομικευμένα και κατάλληλα φιλτραρισμένα αποτελέσματα ανάλογα με τις προτιμήσεις τους, τη συσκευή που χρησιμοποιούν, τη θέση τους ή την ώρα της ημέρας κλπ. Έτσι, ειδικά σε διαδραστικές (interactive) εφαρμογές και συστήματα, όπου οι πληροφορίες που παρέχονται στο χρήστη επηρεάζονται άμεσα από τέτοιες διαφοροποιήσεις, έχει δημιουργηθεί η ανάγκη διαχείρισης αυτών των δεδομένων με έναν αποτελεσματικό και εύχρηστο τρόπο.

Οι ερευνητές της πληροφορικής ονομάζουν όλες αυτές τις εξωτερικές παραμέτρους που επηρεάζουν την πληροφορία, η οποία επιστρέφεται στο χρήστη, χρησιμοποιώντας τον όρο context. Για παράδειγμα, σε μία εφαρμογή που διαχειρίζεται τουριστικές πληροφορίες για

κάποια χώρα, το αποτέλεσμα σε ένα απλό ερώτημα της μορφής "Βρες ένα εστιατόριο" μπορεί να εξαρτηθεί α) από την τοποθεσία στην οποία βρίσκεται ο χρήστης που υποβάλλει το ερώτημα, έτσι ώστε να επιστρέφονται τα εστιατόρια που πχ βρίσκονται κοντά σε αυτόν, β) από το αν έχει στη διάθεσή του κάποιο μεταφορικό μέσο ή αν είναι με τα πόδια, οπότε να επιστραφούν εστιατόρια τα οποία βρίσκονται σε μεγαλύτερη ή μικρότερη απόσταση, γ) από το αν ο χρήστης είναι μόνος ή με παρέα, δ) από την εποχή (χειμώνας ή καλοκαίρι), ε) από τον καιρό (βροχή, λιακάδα, συννεφιά), στ) από τις προσωπικές γαστρονομικές προτιμήσεις του χρήστη κλπ.

Έτσι, έχει προκύψει η ανάγκη δημιουργίας context-aware συστημάτων, δηλαδή συστημάτων των οποίων η απόκριση σε ερωτήσεις (queries) εξαρτάται από τις διαφοροποιήσεις του context. Στο χώρο της διαχείρισης δεδομένων ένα context-aware σύστημα επιτρέπει στο χρήστη την υποβολή context-aware ερωτημάτων. Γενικά, ένα context-aware ερώτημα είναι ένα ερώτημα, το αποτέλεσμα του οποίου εξαρτάται όχι μόνο από την εκάστοτε βάση δεδομένων, αλλά και από το context του χρήστη με βάση το οποίο εκτελείται το ερώτημα. Επομένως, οι παράμετροι από τις οποίες εξαρτάται η επεξεργασία και η εκτέλεση ενός context-aware ερωτήματος είναι α) η βάση δεδομένων και β) το context. Το ίδιο ερώτημα (query), εκτελούμενο από διαφορετικούς χρήστες ή από τον ίδιο χρήστη αλλά κάτω από διαφορετικά contexts παράγει γενικά διαφορετικό αποτέλεσμα. Για παράδειγμα, ένα ερώτημα σε μία σχέση μίας βάσης δεδομένων ονόματι Προορισμός από μία πόλη Α σε μία πόλη Β θα επιστρέψει διαφορετικό αποτέλεσμα αν το ερώτημα πραγματοποιηθεί τη νύχτα, καθώς το ερώτημα θα απορρίψει τη διαδρομή που περνάει από ένα δάσος για λόγους ασφαλείας, ενώ αν το ερώτημα πραγματοποιηθεί κατά τη διάρκεια της ημέρας θα απορρίψει τη διαδρομή που διέρχεται από το κέντρο της πόλης, προκειμένου να αποφευχθεί η κυκλοφοριακή κίνηση. Σε αντιδιαστολή, όταν χρησιμοποιείται από μία εφαρμογή ένα Σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ), η απάντηση σε ένα ερώτημα εξαρτάται μόνο από το περιεχόμενο της βάσης δεδομένων και είναι πάντα η ίδια ανεξάρτητα από το ποιος το εκτελεί ή ποιο είναι το context του.

Αρκετές προσπάθειες έχουν γίνει για τη μοντελοποίηση του context στην πληροφορική, αλλά και πιο συγκεκριμένα στον τομέα της διαχείρισης δεδομένων. Οι περισσότερες από αυτές χαρακτηρίζονται από την προσπάθεια διαχείρισης του context και των context-aware δεδομένων σε επίπεδο εφαρμογής (application layer). Όμως, ακόμα και στα μοντέλα που προτείνεται η ένταξη του context στο επίπεδο των δεδομένων (data layer) δεν υπάρχει κάποια προσπάθεια ανάπτυξης ενός context-aware σχεσιακού μοντέλου.

Έτσι, λοιπόν, ο στόχος της παρούσας διπλωματικής είναι ο σχεδιασμός και η υλοποίηση του πρώτου ερευνητικού σχεσιακού Συστήματος Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ), το οποίο θα ενσωματώνει το context στη διαχείριση της πληροφορίας. Για το σκοπό αυτό είναι

απαραίτητη η εισαγωγή μίας νέας γλώσσας επεξεργασίας ερωτημάτων που θα εκμεταλλεύεται το context. Έτσι, προτείνεται μία επέκταση της κλασικής σχεσιακής άλγεβρας που υποστηρίζει context-aware ερωτήματα μέσω του ορισμού context-aware τελεστών. Αντίστοιχα, για τις ανάγκες του context-aware συστήματος αυτού ορίζεται και η αντίστοιχη επέκταση της SQL, η extended SQL (ESQL), μέσω της οποίας ο χρήστης μπορεί να εισάγει context-aware ερωτήματα στο σύστημα.

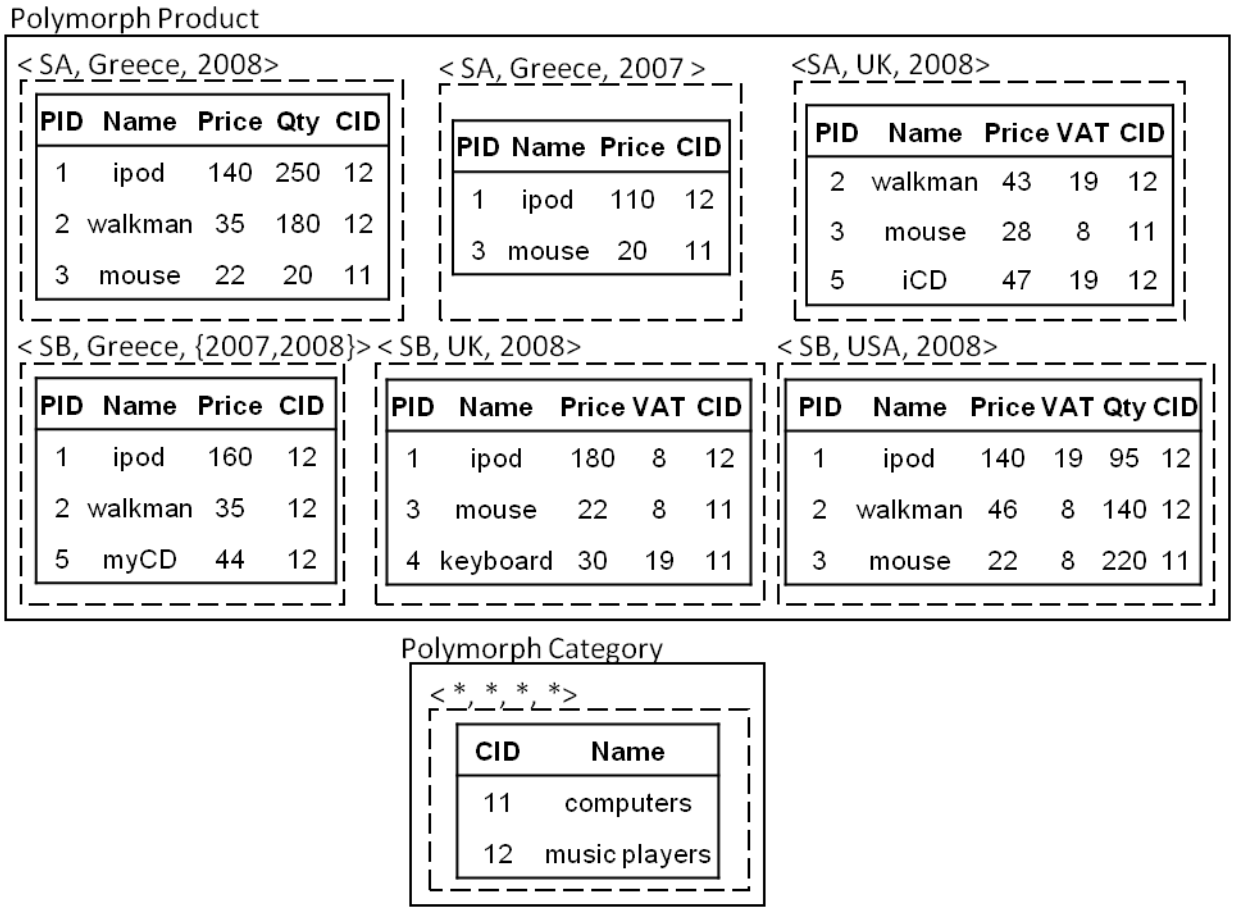
Τα ερωτήματα αυτά είναι τα ίδια για όλους τους χρήστες, αλλά το αποτέλεσμα μεταβάλλεται δυναμικά ανάλογα με το context του κάθε χρήστη, καθώς χρησιμοποιείται αυτόματα κάθε φορά το κατάλληλο σχήμα. Αντίθετα, σε μια υλοποίηση που χρησιμοποιεί το κλασικό σχεσιακό μοντέλο, η επιλογή των σωστών εγγραφών και η παρουσίαση των κατάλληλων πεδίων πρέπει να γίνει προγραμματιστικά σε κάθε εφαρμογή ξεχωριστά.

Σύμφωνα με το [RS09], όπου προτείνεται το μοντέλο στο οποίο βασίστηκε το σύστημα που υλοποιήσαμε, μια context aware relation (την ονομάζουμε polymorph) αποτελείται από ένα σύνολο από κλασικές σχέσεις (τις ονομάζουμε morphs), μια για κάθε context στο οποίο ορίζεται η context aware relation. Επιπλέον, έχοντας στη διάθεσή μας ένα σύνολο από τελεστές για τη διαχείριση σχέσεων με πολλαπλά σχήματα τα οποία εξαρτώνται από το context, μπορούμε γράφοντας απλά ερωτήματα να εκτελέσουμε πολύπλοκες ερωτήσεις ανάλυσης.

Με τη δημιουργία ενός τέτοιου συστήματος θα μπορούν να δημιουργηθούν πολύπλοκες βάσεις δεδομένων, οι οποίες θα αναπαριστούν πειστικά πληροφορίες άμεσα εξαρτώμενες από το context. Για παράδειγμα, ας θεωρήσουμε μια πολυεθνική εταιρεία που πουλάει τα προϊόντα της σε δεκάδες χώρες του κόσμου μέσω του διαδικτύου. Όταν ένας χρήστης εισέρχεται στο ηλεκτρονικό της κατάστημα με στόχο να αγοράσει κάποια προϊόντα, συλλέγονται πληροφορίες όπως η συσκευή που χρησιμοποιεί, το bandwidth της σύνδεσής του και η χώρα προέλευσής του, τα οποία ορίζουν και το context του χρήστη. Χρησιμοποιώντας ένα Context Aware ΣΔΒΔ, μπορούμε να ορίσουμε ένα σύνολο από διαφορετικά σχήματα για τη σχέση Προϊόν, τα οποία, για παράδειγμα, θα περιέχουν περισσότερα και πιο πλούσια πεδία, όπως π.χ. video ή μεγαλύτερου μεγέθους εικόνες, όσο πιο γρήγορη είναι η σύνδεση του χρήστη και όσο πιο ισχυρή είναι η συσκευή που χρησιμοποιεί.

Αντίστοιχα, κάθε προϊόν μπορεί να έχει διαφορετική τιμή ανάλογα με τη χώρα, διαφορετικούς χρόνους παράδοσης και πιθανώς η περιγραφή του να είναι στη γλώσσα κάθε χρήστη. Έτσι, στο προαναφερθέν σύστημα μπορούμε, πλέον, χρησιμοποιώντας τους extended σχεσιακούς αλγεβρικούς τελεστές ή την extended SQL, να εκτελέσουμε ερωτήματα που καλύπτουν ένα συγκεκριμένο ή και ολόκληρο το φάσμα του context, όπως πχ, σύγκριση τιμών σε διαφορετικά contexts, εύρεση των χωρών ή της κατηγορίας των χρηστών για τις οποίες έχουμε τις μεγαλύτερες πωλήσεις, κλπ. Ένα στιγμιότυπο μιας τέτοιας βάσης

δεδομένων παρουσιάζεται στο Σχήμα 1.1, όπου το context ορίζεται ως <Supplier, Location, Year>. Ο ίδιος προμηθευτής προϊόντων (Supplier) πουλάει διαφορετικά προϊόντα και σε διαφορετικές τιμές ανάλογα με τη χώρα και τη χρονολογία. Όπως φαίνεται στο Σχήμα 1.1, για διαφορετικές τιμές του context, διαφοροποιούνται εκτός από τα δεδομένα, επίσης, και το σχήμα της βάσης δεδομένων.



Σχήμα 1.1. Στιγμιότυπο για μία context-aware βάση δεδομένων

Το Context-Aware Σύστημα Διαχείρισης Βάσεων Δεδομένων που υλοποιήσαμε αποτελείται από διάφορα υποσυστήματα. Στην παρούσα διπλωματική πραγματοποιήσαμε το σχεδιασμό και την υλοποίηση του low level μέρους του συγκεκριμένου συστήματος. Συγκεκριμένα, υλοποιήθηκε το "Υποσύστημα **αποθήκευσης του context και των context-aware δεδομένων**" και το "Υποσύστημα **εκτέλεσης των context-aware τελεστών**" καθώς και το "Υποσύστημα **εκτέλεσης συνολικών context-aware ερωτημάτων**". Έτσι, το σύστημα που καλύπτει η συγκεκριμένη διπλωματική δέχεται ως είσοδο ένα context-aware ερώτημα με τη μορφή ενός πλάνου εκτέλεσης, το εκτελεί και αποθηκεύει τα context-aware δεδομένα, ενώ ταυτόχρονα προωθεί το εκάστοτε αποτέλεσμα προς τα "πάνω", έχοντας ως τελικό προορισμό τη διεπαφή (interface) του συστήματος με τα χρήστη. Κάθε τέτοιο υποσύστημα αποτελείται

από επιμέρους δομικές μονάδες (modules), η αρχιτεκτονική των οποίων περιγράφεται αναλυτικά στο κεφάλαιο 3.

1.2 Οργάνωση του τόμου

Ο παρών τόμος αποτελείται από 7 κεφάλαια και αναλύει πλήρως την ανάπτυξη της εργασίας.

Στο ***δεύτερο κεφάλαιο*** γίνεται μια εισαγωγή των εννοιών του context και των context-aware συστημάτων. Επίσης, περιγράφονται σχετικές εργασίες με το θέμα καθώς και η εργασία στην οποία βασίστηκε το σύστημα που υλοποιήσαμε.

Στο ***τρίτο κεφάλαιο*** αναλύεται διεξοδικά η γενική αρχιτεκτονική του συστήματος και των επιμέρους μονάδων που το αποτελούν.

Στο ***τέταρτο κεφάλαιο*** γίνεται η περιγραφή της υλοποίησης του συστήματος. Αρχικά τεκμηριώνεται η επιλογή της πλατφόρμας και των προγραμματιστικών εργαλείων που επιλέξαμε για την ανάπτυξη και στη συνέχεια προχωρούμε στην αναλυτική περιγραφή των κλάσεων του κώδικα που υλοποιήσαμε.

Το ***πέμπτο κεφάλαιο*** πραγματεύεται τον έλεγχο του συστήματος. Παρουσιάζεται αρχικά η μεθοδολογία του ελέγχου και κατόπιν παρατίθενται αναλυτικά τα αποτελέσματα του ελέγχου με βάση μία συγκεκριμένη context-aware βάση δεδομένων και ερωτήματα που εκτελέσθηκαν σε αυτή.

Στο ***έκτο κεφάλαιο***, που αποτελεί τον επίλογο της διπλωματικής, γίνεται επισκόπηση της εργασίας και παρουσιάζονται ορισμένες ιδέες, που αφορούν βελτιώσεις και μελλοντικές επεκτάσεις του συστήματος.

Στο ***έβδομο κεφάλαιο***, τέλος, δίνεται η βιβλιογραφία και γενικότερα οι πηγές από τις οποίες αντλήθηκαν οι απαραίτητες πληροφορίες για τη συγγραφή της διπλωματικής.

2

Περιγραφή θέματος

Στο συγκεκριμένο κεφάλαιο παρουσιάζουμε εκτενέστερα το αντικείμενο της διπλωματικής. Αρχικά, κάνουμε μία εισαγωγή στην έννοια του context και στον τρόπο χρήσης του στο χώρο της επιστήμης των υπολογιστών. Στη συνέχεια, κάνουμε μία σύντομη αναφορά στις σχετικές εργασίες που έχουν γίνει στον ερευνητικό χώρο της διαχείρισης δεδομένων που αφορούν το context. Τέλος, περιγράφουμε το μοντέλο που προτείνεται στο [RS09], στο οποίο βασίστηκε και η παρούσα διπλωματική.

Οι πηγές παρουσίασης των εισαγωγικών εννοιών είναι για την παράγραφο 2.1 τα [DA00] και [FAJ04], για την παράγραφο 2.2 το [SP04], για την παράγραφο 2.3 τα [Sta03], [FAJ04], [SPV05] και [EAM09] και για την παράγραφο 2.4 το [RS09].

2.1 Εισαγωγή στην έννοια του context και των context-aware συστημάτων

Η ανθρώπινη επικοινωνία είναι αρκετά αποτελεσματική στην ανταλλαγή απόψεων, ιδεών, μηνυμάτων κλπ. Αντίθετα, κατά την επικοινωνία μεταξύ ανθρώπου-μηχανής(υπολογιστή), κάτι τέτοιο καθίσταται αρκετά πολύπλοκο και για το λόγο αυτό πρέπει σε κάθε μοντέλο τέτοιας επικοινωνίας να ορίζεται σαφώς η είσοδος από την πλευρά του ανθρώπου,

προκειμένου ο υπολογιστής να είναι σε θέση να επεξεργαστεί συγκεκριμένα και μη διφορούμενα δεδομένα.

Τις πρώτες δεκαετίες ανάπτυξης των υπολογιστών και των υπολογιστικών και πληροφοριακών συστημάτων η γενική τάση των επιστημόνων της πληροφορικής ήταν η σχεδίαση συστημάτων, σύμφωνα με τα οποία το παραγόμενο κάθε φορά αποτέλεσμα εξαρτάτο αποκλειστικά και μόνο από την άμεση είσοδο του χρήστη. Τα συστήματα αυτά, επομένως, δεν μπορούσαν να εκμεταλλευτούν οποιαδήποτε επιπλέον πληροφορία, πέραν αυτής που ρητά εισάγει ο εκάστοτε χρήστης. Με το πέρας των χρόνων και καθώς η ανάγκη για περαιτέρω αξιοποίηση των υπολογιστών και των δυνατοτήτων που αυτοί προσφέρουν γινόταν όλο και πιο έντονη, υπήρξε η προσπάθεια οι διαδραστικές (interactive) αυτές εφαρμογές να κάνουν χρήση όσο το δυνατόν περισσότερης διαθέσιμης πληροφορίας. Αυτή η έμμεση πληροφορία, που είναι γενικά διαφορετική από την άμεση είσοδο του χρήστη, αποτελεί το context. Η ελληνική μετάφραση του όρου context είναι ‘συμφραζόμενα’ ή ‘ερμηνευτικό περιβάλλον’. Από εδώ και στο εξής θα κάνουμε χρήση του όρου context, αποφεύγοντας την αντίστοιχη ελληνική, που προκαλεί αμφισημίες.

Πολλοί ορισμοί έχουν δοθεί για το context, αλλά στο σημείο αυτό αξίζει να παραθέσουμε τον ορισμό που δίνεται στο [DA00], τον οποίο θεωρούμε πιο αντιπροσωπευτικό για την παρούσα διπλωματική αλλά και πιο γενικό και ολοκληρωμένο. Σύμφωνα, λοιπόν, με αυτόν *context είναι οποιαδήποτε πληροφορία μπορεί να χρησιμοποιηθεί για να χαρακτηρίσει την κατάσταση μίας οντότητας. Μία οντότητα είναι ένα άτομο, μία τοποθεσία ή ένα αντικείμενο που θεωρείται σχετικό με την αλληλεπίδραση μεταξύ ενός χρήστη και μίας εφαρμογής, συμπεριλαμβανομένων του χρήστη και των εφαρμογών καθαυτών.*

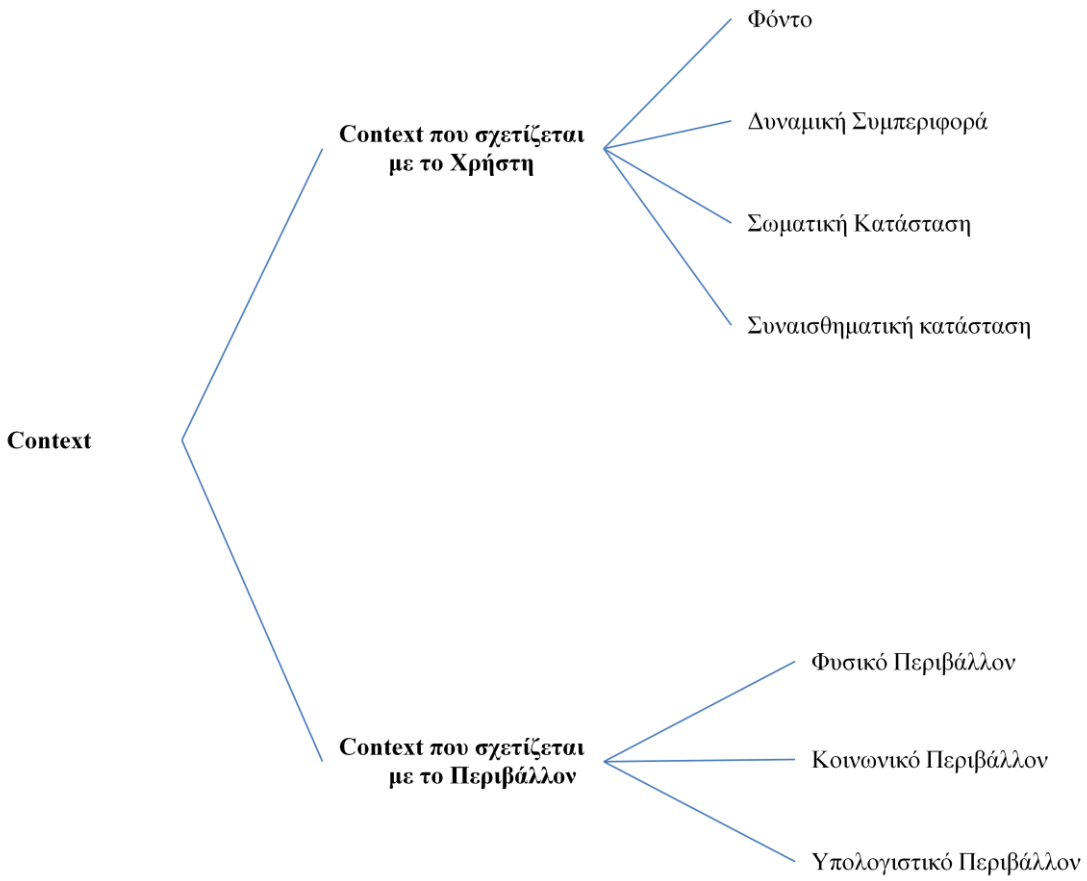
Για να γίνει περισσότερο κατανοητή η παραπάνω ανάλυση κάνουμε χρήση ενός παραδείγματος επικοινωνίας. Έστω το ερώτημα “Τί ώρα είναι;”. Αν αυτό το ερώτημα γίνει από έναν άνθρωπο σε έναν άλλο άνθρωπο, τότε ο ερωτηθείς μπορεί εύκολα να απαντήσει χωρίς να χρειαστεί κάποια άλλη πληροφορία (εφόσον έχει κάποια πρόσβαση σε ένα ρολόι). Αν, όμως, το ερώτημα αυτό δοθεί σε έναν υπολογιστή ή σε ένα πρόγραμμα, χωρίς κανένα άλλο δεδομένο, τότε δεν μπορεί να υπάρξει απάντηση, γιατί σε αυτό το πλαίσιο το ερώτημα δεν είναι σαφές. Δεν έχει, οριστεί για παράδειγμα η τοποθεσία για την οποία ζητείται η ώρα. Διαφορετική είναι η ώρα στη Νέα Υόρκη από την Αθήνα. Επομένως, για το συγκεκριμένο παράδειγμα ως context θα μπορούσε να οριστεί η παράμετρος *τοποθεσία*. Κατά την ανθρώπινη επικοινωνία δε θα υπήρχε κανένα πρόβλημα καθώς αυτή η έμμεση πληροφορία, το context δηλαδή, εννοείται, ο ερωτών ενδιαφέρεται να μάθει την ώρα στο σημείο που βρίσκεται. Αντίθετα, όμως, σε μία επικοινωνία ανθρώπου-μηχανής αυτή η πληροφορία θα πρέπει να είναι με κάποιο τρόπο διαθέσιμη, ακόμα και αν ο χρήστης δεν την εισάγει άμεσα, προκειμένου να απαντηθεί το εν λόγω ερώτημα. Για να προχωρήσουμε λίγο παρακάτω το

παράδειγμα, έστω ότι η απάντηση σε ένα τέτοιο ερώτημα είναι “Η ώρα είναι 11.30”. Τώρα θα μπορούσε να προστεθεί στο context το αν μιλάμε για μέρα ή για νύχτα, δηλαδή αν είναι 11.30 πμ ή 11.30 μμ. Πάλι στην περίπτωση μίας ανθρώπινης επικοινωνίας αυτό το δεδομένο δεν είναι απαραίτητο να δοθεί καθώς ο άνθρωπος που λαμβάνει την απάντηση μπορεί εύκολα να διαπιστώσει αν είναι 11.30 πμ ή 11.30 μμ απλά κοιτώντας τον ουρανό, ενώ στην περίπτωση επικοινωνίας με μηχανή αυτή η πληροφορία είναι απαραίτητη.

Οι πρώτες απόπειρες των ερευνητών για ορισμό του context και ενσωμάτωση του σε κάποια εφαρμογή είχαν ως δεδομένο ότι το context περιλαμβάνει μόνο τις παραμέτρους τοποθεσία ή/και χρόνο. Προχωρώντας αυτή τη σκέψη, μπορούμε να διευρύνουμε την έννοια του context, ώστε να περιλαμβάνει και άλλες παραμέτρους εκτός από τις προαναφερθείσες και γενικά όποιες παραμέτρους απαιτούνται για το εκάστοτε πρόβλημα. Έτσι, λοιπόν, παραθέτουμε κάποιες κατηγοριοποιήσεις για το context που έχουν γίνει στα [DA00] και [FAJ04].

Πιο συγκεκριμένα, στο [DA00] γίνεται ένας διαχωρισμός μεταξύ *βασικών* και *δευτερευόντων* τύπων context. Σύμφωνα με αυτό, οι βασικοί τύποι context είναι η τοποθεσία (location), η ταυτότητα (identity), η δραστηριότητα (activity) και ο χρόνος (time). Αυτές οι κατηγορίες μπορούν να χαρακτηρίσουν την κατάσταση μιας συγκεκριμένης οντότητας και γνωρίζοντας ένα συνδυασμό αυτών μπορούμε να βρούμε και παραμέτρους που αφορούν το δευτερεύον context. Έτσι, στο [DA00] προτείνεται ένα σύστημα δύο επιπέδων. Στο πρώτο επίπεδο ανήκουν οι προαναφερθείσες 4 βασικές κατηγορίες και στο δεύτερο ανήκουν όλες οι υπόλοιπες δευτερεύουσες κατηγορίες.

Στο [FAJ04] γίνεται μία πιο λεπτομερής κατηγοριοποίηση του context. Έτσι, σε πρώτο επίπεδο γίνεται ένας διαχωρισμός μεταξύ του context που αφορά το χρήστη (user-centric context) και αυτού που επηρεάζεται από το περιβάλλον (environmental context). Το user-centric context αναλύεται περαιτέρω στις παραμέτρους που αφορούν το φόντο (background), όπως ενδιαφέροντα, συνήθειες, προτιμήσεις, τομέας εργασίας, κλπ, τη δυναμική συμπεριφορά (dynamic behavior) , όπως ένταση, εργασίες, δραστηριότητα κλπ, τη σωματική κατάσταση (physiological state), όπως θερμοκρασία σώματος, παλμοί καρδιάς κλπ ή τη συναισθηματική κατάσταση (emotional state), όπως ευτυχία, δυστυχία, απέχθεια, φόβος, θυμός, έκπληξη, ηρεμία κλπ. Κατά τον ίδιο τρόπο, γίνεται διαχωρισμός για το context που σχετίζεται με το περιβάλλον σε φυσικό περιβάλλον (physical environment), όπως χρόνος, τοποθεσία, θερμοκρασία, υγρασία, θόρυβος, ένταση φωτός, κραδασμοί κλπ, σε κοινωνικό περιβάλλον (social environment), όπως κυκλοφοριακή κίνηση, πληροφορίες που αφορούν εκπτώσεις, άνθρωποι που συνυπάρχουν τριγύρω κλπ και σε υπολογιστικό περιβάλλον (computational environment), όπως διαθέσιμες συσκευές κλπ. Αυτή η κατηγοριοποίηση παρουσιάζεται σχηματικά στο Σχήμα 2.2.



Σχήμα 2.1. Κατηγοριοποίηση context σύμφωνα με το [FAJ04]

Τελειώνοντας με την προσπάθεια ορισμού και κατηγοριοποίησης του context, πρέπει να προχωρήσουμε και στον ορισμό των context-aware συστημάτων στο χώρο της πληροφορικής. Η ανάγκη πρόσβασης στο context για κάποιες εφαρμογές προκύπτει από το γεγονός ότι αυτό αλλάζει συνεχώς και δραματικά και έτσι απαιτείται διαρκής παρακολούθησή του και προσαρμογή στα νέα context δεδομένα. Επομένως, θα μπορούσαμε να ορίσουμε ως context-aware συστήματα τα συστήματα τα οποία επηρεάζονται από την αλλαγή του context. Ένας πιο αυστηρός ορισμός δίνεται στο [DA00], σύμφωνα με το οποίο, *ένα σύστημα είναι context-aware αν χρησιμοποιεί το context για να παρέχει σχετικές πληροφορίες ή/και υπηρεσίες στο χρήστη, όπου η σχετικότητα αυτή εξαρτάται από την εκάστοτε εργασία του χρήστη*. Επίσης, στο [DA00] δίνεται και μία εξίσου γενική κατηγοριοποίηση των context-aware συστημάτων και έτσι προκύπτουν οι τρεις ακόλουθες κατηγορίες:

- τα συστήματα που κάνουν χρήση του context για να παρουσιάσουν στο χρήστη πληροφορίες και υπηρεσίες (services),
- τα συστήματα στα οποία η αλλαγή του context προκαλεί αυτόματη εκτέλεση μίας υπηρεσίας (service) και

- τα συστήματα που συσχετίζουν (tagging) την πληροφορία με συγκεκριμένο context για μελλοντική προσπέλαση.

Η προσπάθεια αυτή προσέγγισης της έννοιας του context και των context-aware συστημάτων θα βοηθήσει προκειμένου αργότερα να παρουσιάσουμε το σχεσιακό μοντέλο δεδομένων για context που προτείνεται στο [RS09] και στο οποίο βασίστηκε η εν λόγω διπλωματική. Στην επόμενη ενότητα παρουσιάζουμε την κατηγοριοποίηση που έχει γίνει στο [SP04] και αφορά τα μοντέλα για την αναπαράσταση του context.

2.2 Κατηγοριοποίηση μοντέλων για την αναπαράσταση του context

Η προηγούμενη ανάλυση και οι ορισμοί του context πρέπει με κάποιο τρόπο να μοντελοποιηθεί, έτσι ώστε να είναι δυνατή η ανάπτυξη πραγματικών συστημάτων που κάνουν χρήση του context. Στο σημείο αυτό θα παραθέσουμε μία κατηγοριοποίηση που έχει γίνει στο [SP04] και αφορά αυτά τα μοντέλα.

2.2.1 Μοντέλο Κλειδιού-Τιμής (Key-Value)

Το μοντέλο κλειδιού-τιμής είναι η πιο απλή δομή δεδομένων για μοντελοποίηση της πληροφορίας σχετικά με το context. Σύμφωνα με αυτό το μοντέλο, καθίσταται δυνατό για κάποια context-aware εφαρμογή να χρησιμοποιήσει μία καθολική μεταβλητή η οποία παίρνει ως τιμές αυτές του context. Το μοντέλο κλειδιού-τιμής χρησιμοποιείται περισσότερο σε αρχιτεκτονικές κατανεμημένων υπηρεσιών, όπου η υπηρεσία περιγράφεται ως μία λίστα από context ιδιότητες (attributes) και η εν λόγω υπηρεσία λειτουργεί με έναν αλγόριθμο "ακριβούς ταιριάσματος" (exact matching) σε αυτές τις ιδιότητες. Στην πράξη, το μοντέλο κλειδιού-τιμής είναι εύκολο στη διαχείριση, αλλά δεν υπάρχει η δυνατότητα για δημιουργία πολύπλοκης context δομής, καθώς δεν υπάρχουν αποτελεσματικοί αλγόριθμοι ανάκτησης του context.

Ένα παράδειγμα του μοντέλου κλειδιού-τιμής είναι ένα σύστημα τα δεδομένα του οποίου διαφοροποιούνται ανάλογα με την Τοποθεσία και το Έτος στο οποίο αναφερόμαστε. Έτσι, για το συγκεκριμένο σύστημα ως context ορίζεται το σύνολο <Location, Year> και συγκεκριμένες τιμές για αυτό είναι οι ακόλουθες <USA, 2008>, <USA, 2009>, <Greece, 2009>, <UK, 2010> κλπ. Στη συγκεκριμένη κατηγορία ανήκει το μοντέλο που προτείνεται στο [RS09] (βλ. ενότητα 2.4), στο οποίο βασίζεται το σύστημα που υλοποιήσαμε στην παρούσα διπλωματική. Επιλέχθηκε αυτό το μοντέλο, καθώς είναι ένα μοντέλο εύκολο στη

διαχείριση του context και αντιμετωπίζει το context ως μαύρο κουτί. Έτσι, με το μοντέλο που προτείνεται στο [RS09] η διαχείριση του context και των context-aware δεδομένων γίνεται με ένα ενιαίο και απλό τρόπο.

2.2.2 Μοντέλο Περιγραφικής Απεικόνισης (Markup Scheme)

Πρόκειται για μία ιεραρχική δομή δεδομένων αντίστοιχης εκείνης των markup γλωσσών (XML, HTML κλπ) με markup ετικέτες (tags), ιδιότητες (attributes) και περιεχόμενο. Τυπικά αντιπροσωπευτικό δείγμα αυτού του μοντέλου είναι τα λεγόμενα profiles. Υπάρχουν και διάφορες άλλες προσεγγίσεις που ανήκουν σε αυτό το μοντέλο. Αναφέρουμε ενδεικτικά μερικές: Composite Capabilities / Preferences Profile (CC/PP) [W3C07], User Agent Profile (UAProf) [WAP09], Comprehensive Structured Context Profiles (CSCP) [HBS02], Pervasive Profile Description Language (PPDL) [CF03].

2.2.3 Γραφικό (Graphical) Μοντέλο

Ένα ευρέως γνωστό εργαλείο γενικού σκοπού μοντελοποίησης είναι η Unified Modeling Language (UML), η οποία έχει ένα πολύ ισχυρό γραφικό υπόβαθρο, τα διαγράμματα UML. Χάρη στη γενική δομή της, η UML είναι κατάλληλη και για τη μοντελοποίηση του context. Ένα άλλο παράδειγμα που υπάγεται στο γραφικό μοντέλο είναι αυτό που προτείνεται στο [HIR03], το οποίο αποτελεί μία επέκταση του ORM (Objective-Role Modeling). Από το γραφικό μοντέλο για context μπορεί να προκύψει και το επίσης γνωστό E-R μοντέλο (Entity-Relation/Οντότητας-Σχέσης), το οποίο είναι ένα πολύ χρήσιμο εργαλείο για μία σχεσιακή βάση δεδομένων σε ένα context-aware πληροφοριακό σύστημα.

2.2.4 Αντικειμενοστραφές (Object Oriented) Μοντέλο

Το αντικειμενοστραφές μοντέλο για context κάνει χρήση της ενθυλάκωσης (encapsulation), της επαναχρησιμοποίησης (reusability) και της κληρονομικότητας (inheritance). Έτσι, οι λεπτομέρειες του context ενθυλακώνονται σε κάθε επίπεδο και συνεπώς κρύβονται για τα υπόλοιπα στοιχεία, εμφανίζοντάς τους μόνο την απαραίτητη πληροφορία που αυτά χρειάζονται. Οι νέοι τύποι context πληροφορίας, όπως οι κλάσεις ή τα αντικείμενα, χρησιμοποιούνται από ένα τέτοιο σύστημα ακόμα και με έναν κατανεμημένο τρόπο. Γενικά, τα πλεονεκτήματα και μειονεκτήματα του αντικειμενοστραφούς μοντέλου είναι παρόμοια με αυτά των αντικειμενοστραφών γλωσσών προγραμματισμού σε σχέση με τις κλασσικές.

2.2.5 Λογικό (Logic Based) Μοντέλο

Η λογική ορίζει τις συνθήκες στις οποίες μία έκφραση (expression) ή ένα γεγονός (fact) μπορεί να προέλθει από ένα σύνολο άλλων εκφράσεων ή γεγονότων. Για να περιγραφούν αυτές οι συνθήκες πρέπει να χρησιμοποιηθεί ένα τυπικό σύστημα. Έτσι, λοιπόν, σε ένα λογικό μοντέλο για context, το context ορίζεται ως γεγονότα, εκφράσεις και κανόνες. Το κοινό στοιχείο σε όλα τα συστήματα που ανήκουν στο logic-based μοντέλο είναι ότι αυτά υποστηρίζουν ένα μεγάλο βαθμό τυπικότητας (formality). Οφείλουμε να τονίσουμε ότι παρά την τυπικότητα που αυτή η κατηγορία προσφέρει, τα συστήματα που υλοποιούνται με βάση αυτή έχουν την τάση να είναι επιρρεπή σε λάθη.

2.2.6 Οντολογικό (Ontology Based) Μοντέλο

Οι οντολογίες είναι ένα πολλά υποσχόμενο εργαλείο που επιτρέπει τον καθορισμό περιεχομένων και συσχετίσεων. Είναι, επίσης, κατάλληλο για την προβολή και μετατροπή μέρους της πληροφορίας, οι οποίες περιγράφουν και χρησιμοποιούνται στην καθημερινή ζωή (όπως είναι αυτές που υπάγονται στο context), για δομές δεδομένων χρησιμοποιήσιμες από τους υπολογιστές.

Μετά την σύντομη παρουσίαση των παραπάνω μοντέλων για τα context-aware συστήματα, το [SP04] καταλήγει ότι το περισσότερο ελπιδοφόρο και αποτελεσματικό μοντέλο για αναπαράσταση της context πληροφορίας είναι το οντολογικό μοντέλο.

2.3 Σχετικές εργασίες για context-aware συστήματα που διαχειρίζονται δεδομένα

Στην ενότητα αυτή θα περιγράψουμε εν συντομία τα μοντέλα που έχουν προταθεί σε προηγούμενες εργασίες για διαχείριση δεδομένων σε context-aware συστήματα. Συγκεκριμένα, θα αναφερθούμε στα μοντέλα που αναφέρονται στο [Sta03], όπου προτείνεται ένα μοντέλο που βασίζεται σε πολυδιάστατα ημιδομημένα δεδομένα (multidimensional semistructured data), στο [FAJ04], όπου προτείνεται ένα context-aware μοντέλο διαχείρισης δεδομένων για την *Περιβάλλουσα Νοημοσύνη* (Ambient Intelligence), στο [SPV05], όπου προτείνεται ένα context-aware σύστημα Βάσεων Δεδομένων που βασίζεται στις *Προτιμήσεις* (Preferences) και στο [EAM09], όπου προτείνεται ένα νέο μοντέλο που εισάγει το context εντός των Συστημάτων Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ/DBMS) ονόματι Χαμελέων (Chameleon).

2.3.1 Ένα μοντέλο που βασίζεται σε πολυδιάστατα ημιδομημένα δεδομένα (multidimensional semistructured data)

Το συγκεκριμένο μοντέλο, το οποίο εισάγει την έννοια του context στη διαχείριση της πληροφορίας, όχι όμως για σχεσιακά συστήματα αλλά για ημιδομημένα δεδομένα, προτείνεται στο [Sta03]. Περιγράφουμε συνοπτικά το μοντέλο αυτό. Τα πολυδιάστατα δεδομένα είναι ένα μοντέλο θεώρησης δεδομένων, με τη βοήθεια του οποίου είναι δυνατή η οργάνωση και διαχείριση ημιδομημένων δεδομένων με έναν εμπλουτισμένο τρόπο. Τα πολυδιάστατα ημιδομημένα δεδομένα είναι βασικά και αυτά ημιδομημένα δεδομένα, που παρουσιάζουν όμως διαφορετικές όψεις (facets), δηλαδή διαφορετικό περιεχόμενο, κάτω από διαφορετικά περιβάλλοντα που αυτά μπορούν να έχουν υπόσταση.

Τα περιβάλλοντα αυτά ονομάζονται κόσμοι (worlds), και προκειμένου να οριστούν χρησιμοποιούμε ένα σύνολο παραμέτρων, οι οποίες ονομάζονται διαστάσεις (dimensions). Καταυτόν τον τρόπο, ένας κόσμος ορίζεται μέσω της απόδοσης σε κάθε διάσταση μιας τιμής. Το ζεύγος διάσταση-τιμή αποτελεί ουσιαστικά την περιγραφή μιας συνθήκης που ικανοποιείται στα πλαίσια αυτού του κόσμου.

Για παράδειγμα, θα μπορούσαμε να περιγράψουμε έναν κόσμο ορίζοντας την τιμή των διαστάσεων: γλώσσα, λεπτομέρεια, μορφή αρχείου. Έτσι, αν δώσουμε τις αντίστοιχες τιμές ελληνικά, υψηλή, PDF, θα έχουμε μια έκφραση της μορφής:

```
[lang=gr, detail=high, format=pdf]
```

Η έκφραση αυτή ονομάζεται στα πλαίσια των πολυδιάστατων ημιδομημένων δεδομένων “context specifier”. Ένας context specifier μπορεί να ορίζει όχι μόνο μεμονωμένους κόσμους, αλλά και σύνολα αυτών, όπως στο εξής παράδειγμα:

```
[lang=gr, detail=high]
```

όπου αναφερόμαστε στους κόσμους όπου η γλώσσα είναι ελληνικά, η λεπτομέρεια είναι υψηλή και το format μπορεί να είναι οτιδήποτε.

Με αυτό τον τρόπο είναι δυνατό να έχουμε διαφορετικά στιγμιότυπα της ίδιας πληροφορίας, με το καθένα από αυτά να ισχύει κάτω από ένα διαφορετικό σύνολο κόσμων. Μια τέτοια πληροφοριακή οντότητα που περικλείει έναν αριθμό από διαφορετικά στιγμιότυπα (όψεις) λέγεται Πολυδιάστατη Οντότητα (Multidimensional Entity). Εάν οι όψεις e1, e2, .. en μιας πολυδιάστατης οντότητας ισχύουν κάτω από ένα κόσμο w (ή κάτω από κάθε κόσμο που ορίζει ένας context specifier c), τότε λέμε ότι η e αποτιμάται σε e1, e2, .. en κάτω από τον w ή κάτω από τον c).

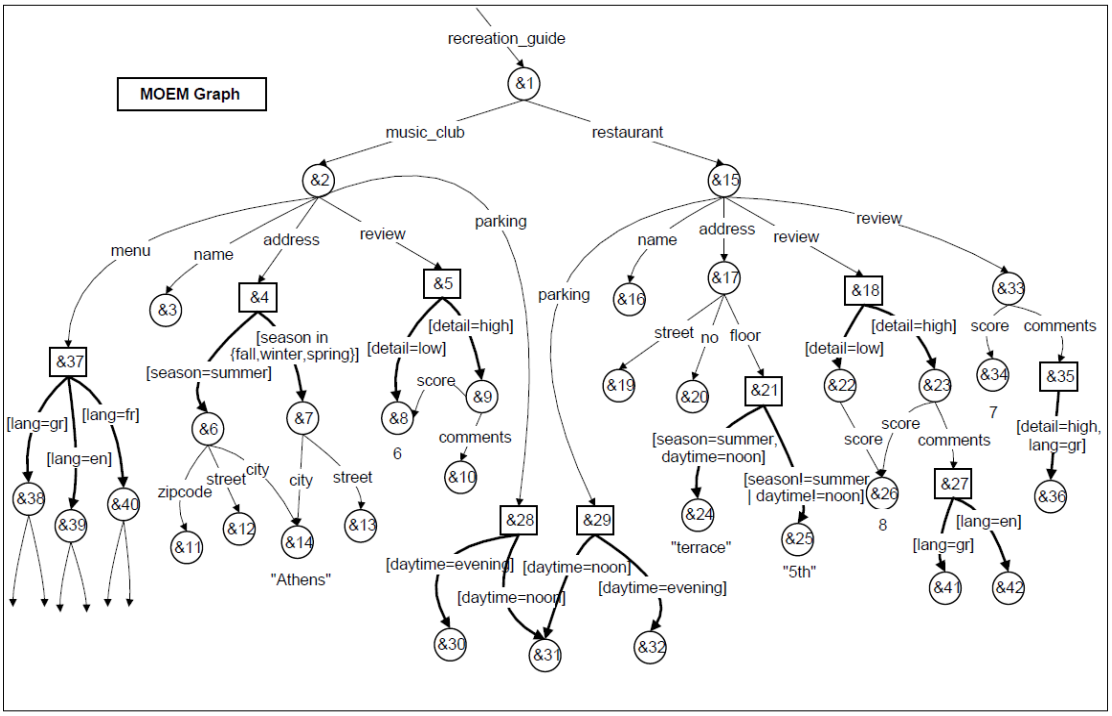
Το μοντέλο MOEM

Το μοντέλο MOEM είναι μια επέκταση του OEM [SG01], η οποία διατηρεί την ευελιξία του OEM, αλλά παράλληλα εισάγει σε αυτό την έννοια των κόσμων κάτω από τους οποίους ισχύουν τα δεδομένα που είναι αποθηκευμένα στο γράφο.

Δύο είναι τα βασικά στοιχεία που υλοποιούν αυτή την επέκταση:

- **Πολυδιάστατοι Κόμβοι**: Ένας πολυδιάστατος κόμβος (multidimensional node) αναπαριστά μία πολυδιάστατη οντότητα και χρησιμοποιείται για την ομαδοποίηση κόμβων που παριστάνουν διαφορετικές όψεις της οντότητας αυτής. Στο συγκεκριμένο μοντέλο δεδομένων που περιγράφουμε, οι πολυδιάστατοι κόμβοι έχουν ορθογώνιο σχήμα για να τους διακρίνουμε από τους συμβατικούς κυκλικούς κόμβους.
- **Context Ακμές**: Οι context ακμές είναι κατευθυνόμενες ακμές που συνδέουν έναν πολυδιάστατο κόμβο με τις εναλλακτικές του μορφές. Το όνομα μιας context ακμής που δείχνει στη όψη r, είναι ένας context specifier που ορίζει το σύνολο κόσμων κάτω από τους οποίους η r έχει υπόσταση. Οι context ακμές σχεδιάζονται σαν παχιές ή διπλές γραμμές, για να διακρίνονται από τις συμβατικές ακμές.

Το νέο μοντέλο που περιγράφουμε ονομάζεται *Multidimensional Object Exchange Model* (*MOEM*). Στο MOEM οι συμβατικοί κυκλικοί κόμβοι ονομάζονται context κόμβοι και αναπαριστούν εναλλακτικές όψεις σχετιζόμενες με κάποιο context. Οι συμβατικές ακμές του OEM (λεπτές γραμμές) ονομάζονται entity ακμές (οντοτήτων).



Σχήμα 2.2. Παράδειγμα ενός οδηγού ψυχαγωγίας σε μορφή MOEM γράφου

Όπως και στο OEM, όλοι οι MOEM κόμβοι θεωρούνται αντικείμενα και χαρακτηρίζονται από ένα μοναδικό αναγνωριστικό (object identifier). Στη συνέχεια, στο πλαίσιο του MOEM, οι όροι κόμβος και αντικείμενο θα χρησιμοποιούνται εναλλακτικά. Όσον αφορά στα context αντικείμενα, ισχύουν όσα έχουμε περιγράψει για τους OEM γράφους.

Στο Σχήμα 2.2 έχουμε ένα παράδειγμα ενός οδηγού ψυχαγωγίας σε μορφή MOEM γράφου. Για λόγους απλότητας ο γράφος δεν έχει αναπτυχθεί πλήρως και μερικοί από τους atomic κόμβους δεν έχουν τιμές.

Για το παράδειγμα του σχήματος 2.2, οι διαστάσεις και τα πεδία τιμών τους είναι τα ακόλουθα:

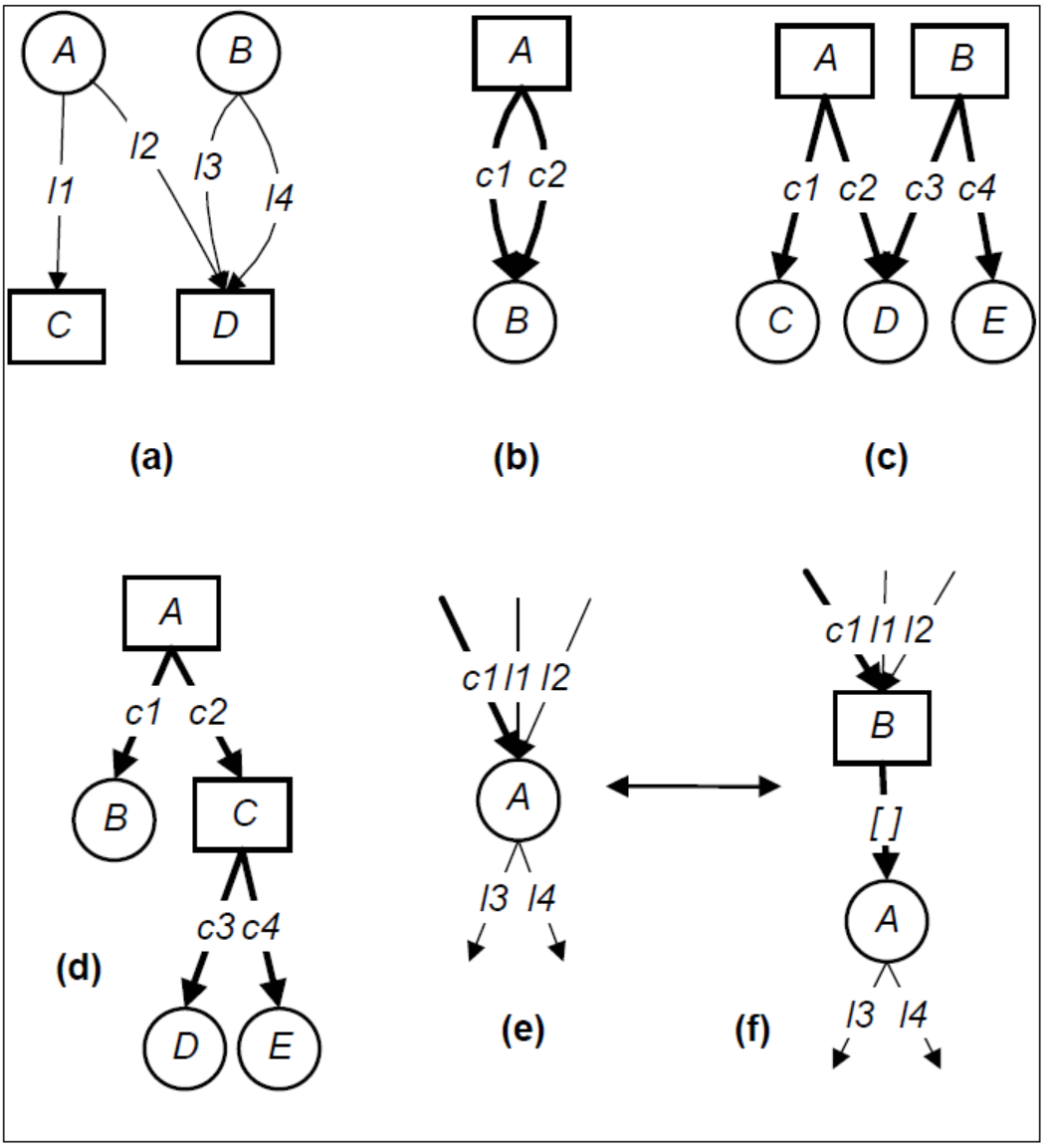
- season με τιμές{summer, fall, winter, spring}
- daytime με τιμές {noon, evening}
- detail με τιμές {high, low} και
- lang με τιμές {en, fr, gr}

Το restaurant με αναγνωριστικό &15 κανονικά βρίσκεται στον πέμπτο όροφο, αλλά τα μεσημέρια του καλοκαιριού βρίσκεται στη βεράντα. Για αυτό το λόγο ο κόμβος με αναγνωριστικό &21 είναι ένα πολυδιάστατο αντικείμενο του οποίου η (atomic) τιμή εξαρτάται από τις διαστάσεις summer και daytime. Εκτός από διαφορετικές τιμές, τα context αντικείμενα μπορούν να έχουν και διαφορετική δομή, όπως στην περίπτωση των κόμβων &6 και &7 που είναι εναλλακτικές μορφές του πολυδιάστατου αντικείμενου address με αναγνωριστικό &4.

Μία context ακμή δεν μπορεί να ξεκινά από έναν context κόμβο και μία απλή ακμή δεν μπορεί να ξεκινά από έναν πολυδιάστατο κόμβο. Αυτοί οι δύο είναι όλοι οι περιορισμοί στη μορφολογία του MOEM γράφου.

Στο σχήμα 2.3 φαίνονται μερικές όχι και τόσο απλές, αλλά νόμιμες MOEM δομές. Στο παράδειγμα (b) περισσότερες από μια context ακμές συνδέουν έναν πολυδιάστατο κόμβο με έναν άλλο context κόμβο. Ο πολυδιάστατος κόμβος A αποτιμάται στον B κάτω από την ένωση των κόσμων που ορίζονται από τους c_1, c_2. Έτσι οι δύο context ακμές μπορούν να αντικατασταθούν με μία με context specifier που εκφράζει την τομή των συνόλων που εκφράζουν οι c_1, c_2. Το παράδειγμα (c) δείχνει έναν context κόμβο D που αποτελεί κομμάτι δύο πολυδιάστατων κόμβων A και B. Αυτή είναι η περίπτωση του κόμβου &31 στο παράδειγμα του σχήματος 2.2. Μια πολυδιάστατη οντότητα μπορεί να είναι μέρος μιας άλλης πολυδιάστατης οντότητας , όπως φαίνεται στο παράδειγμα (d). Η δομή του παραδείγματος (e) είναι ισοδύναμη με αυτή του (f): από την οπτική των l_1, l_2 ο context κόμβος A δε σχετίζεται

μέσω μιας context ακμής, και έτσι θεωρείται η μοναδική όψη της πολυδιάστατης οντότητας που αναπαρίσταται με B και έχει νόημα κάτω από οποιοδήποτε κόσμο.



Σχήμα 2.3. Κάποιες MOEM δομές

Εύκολα καταλαβαίνουμε ότι το OEM είναι μια ειδική περίπτωση πολυδιάστατου γράφου δεδομένων, όπου δεν υπάρχουν πολυδιάστατοι κόμβοι και context ακμές.

Στο [SG01] παραθέτονται τυπικοί ορισμοί για όλες τις προηγούμενες έννοιες. Στην ίδια εργασία ορίζονται και αλγόριθμοι με τη βοήθεια των οποίων μπορούμε να λάβουμε τον OEM γράφο κάθε κόμβος του οποίου αποτελεί όψη ενός MOEM γράφο κάτω από έναν συγκεκριμένο κόσμο (reduction).

Όπως έχει αναπτυχθεί στο [ABS00] ο θεμελιώδης λίθος του συντακτικού για ημιδομημένα δεδομένα είναι η *ssd-expression*. Στο [SG01] έχει οριστεί η *mssd-expression* επεκτείνοντας το

συντακτικό για SSD ώστε να συμπεριλάβει context specifier.

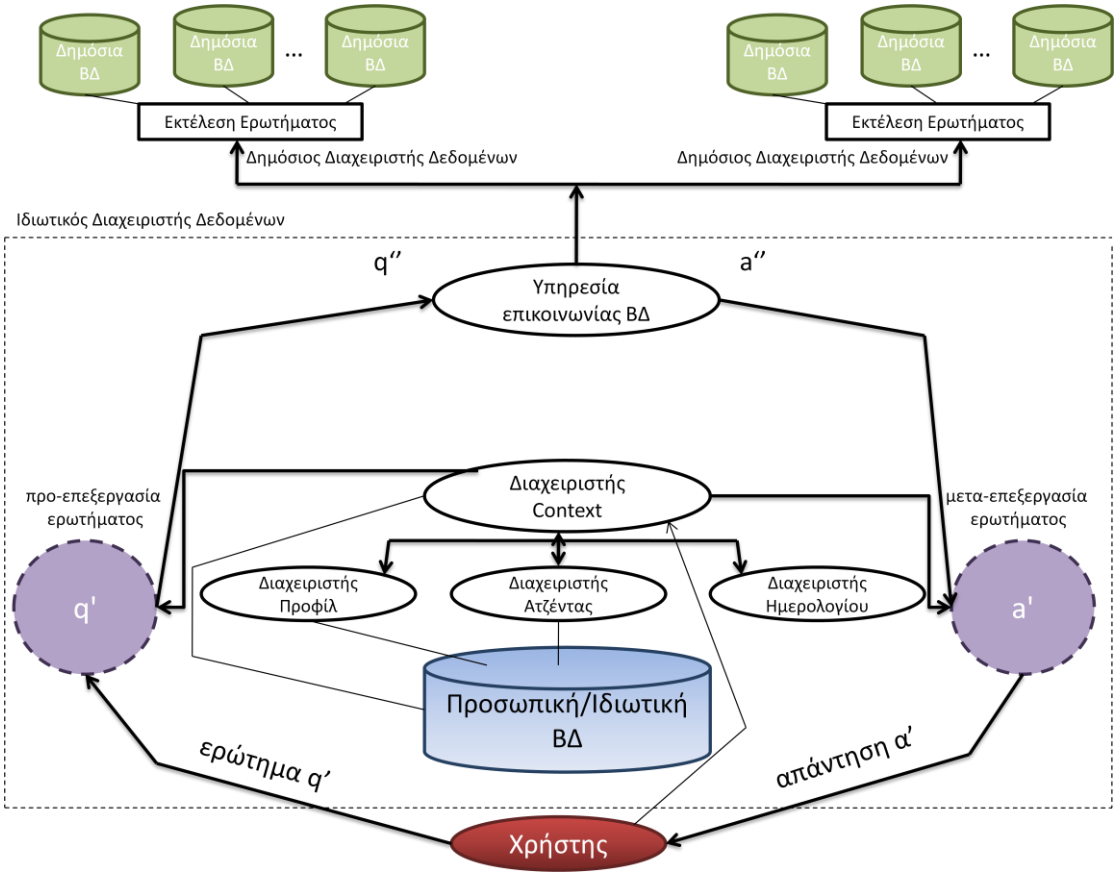
Τέλος, για το συγκεκριμένο μοντέλο ορίζεται και η αντίστοιχη γλώσσα MQL, η οποία περιγράφεται στο [Sta02].

2.3.2 Ένα context-aware μοντέλο διαχείρισης δεδομένων για την Περιβάλλουσα Νοημοσύνη (Ambient Intelligence)

Στο μοντέλο αυτό, το οποίο παρουσιάζεται, στο [FAJ04] προτείνεται ένα σύστημα για context-aware data management και αποτελεί ουσιαστικά ένα middleware, το οποίο δεν ενσωματώνεται σε ΣΔΒΔ. Το σύστημα αυτό περιλαμβάνει ένα Δημόσιο Διαχειριστή Δεδομένων (Public Data Manager) και έναν Ιδιωτικό Διαχειριστή Δεδομένων (Private Data Manager). Τα βασικά συστατικά που αποτελούν τον Ιδιωτικό Διαχειριστή Δεδομένων είναι τα ακόλουθα:

- Ο *διαχειριστής context* (context manager) ορίζει και συντηρεί τον χώρο του context. Διευκολύνει τη χρήση του context με τη δημιουργία context attributes από διάφορες εξωτερικές πηγές.
- Ο *διαχειριστής προφίλ* (profile manager) είναι υπεύθυνος για τη διαχείριση και παροχή πληροφορίας για το προφίλ του χρήστη, συμπεριλαμβανομένων των ενδιαφερόντων του χρήστη, του τομέα εργασίας, των συνηθειών, των προτιμήσεων κλπ.
- Ο *διαχειριστής ατζέντας* (agenda manager) διαχειρίζεται την καθημερινή ατζέντα του χρήστη. Δοθείσης μίας πρόσβασης στο ιστορικό της βάσης δεδομένων, προσπελαύνει προηγούμενες δραστηριότητες και context για τα δεδομένα.
- Ο *διαχειριστής ημερολογίου* (log manager) διαχειρίζεται και αποθηκεύει το ιστορικό των προσπελάσεων στη βάση δεδομένων του χρήστη. Έτσι, ο log manager σε συνεργασία με τον agenda manager ανακαλεί το προηγούμενο context και παρέχει υποστήριξη στα context-aware ερωτήματα.
- Η *υπηρεσία επικοινωνίας βάσης δεδομένων* (database service communicator) λειτουργεί ως μία γέφυρα μεταξύ του ιδιωτικού διαχειριστή δεδομένων και των εξωτερικών δημόσιων διαχειριστών δεδομένων. Οι δημόσιοι διαχειριστές δεδομένων πρέπει να κάνουν register στον communicator ως πάροχοι υπηρεσιών βάσης δεδομένων (database service providers), προκειμένου να χρησιμοποιηθούν από τον ιδιωτικό χρήστη. Ο communicator πρέπει να αντιμετωπίσει την ετερογένεια διαφορετικών δημόσιων διαχειριστών δεδομένων, ενώ ταυτόχρονα παρέχει στον ιδιωτικό χρήστη μία προβολή (view) του μοντέλου δεδομένων και του σχήματος της βάσης δεδομένων που χρησιμοποιείται.

- Η *πολυ-σχηματική διεπαφή* (multi-modal interface) σε εξωτερικές συναρτήσεις/υπηρεσίες προσφέρει είσοδο και έξοδο που προσαρμόζεται στο context. Ένα τέτοιο παράδειγμα είναι η μετάφραση της εξόδου ως κείμενο σε έξοδο ως λόγο, όταν ο χρήστης πχ οδηγεί.
- Η *context-aware μονάδα συντονισμού ερωτημάτων* (context-aware query coordinator) αναλαμβάνει την εκτέλεση των ερωτημάτων με βάση κάποια στρατηγική. Πιο συγκεκριμένα ζητήματα για την επεξεργασία ερωτημάτων αναφέρονται παρακάτω.



Σχήμα 2.4. Η αρχιτεκτονική του συστήματος που προτείνεται στο [FAJ04]

Αυτές οι σχέσεις (relations) αποκαλούνται στο [FAJ04] *σχέσεις βάσης δεδομένων ευαίσθητες στο context* (context-sensitive database relations). Αντίθετα, τα αποτελέσματα των ερωτημάτων στα κλασικά ΣΔΒΔ εξαρτώνται μόνο από τη βάση δεδομένων στην οποία αυτά εκτελούνται και είναι πάντα ίδια για το ίδιο ερώτημα και την ίδια βάση δεδομένων.

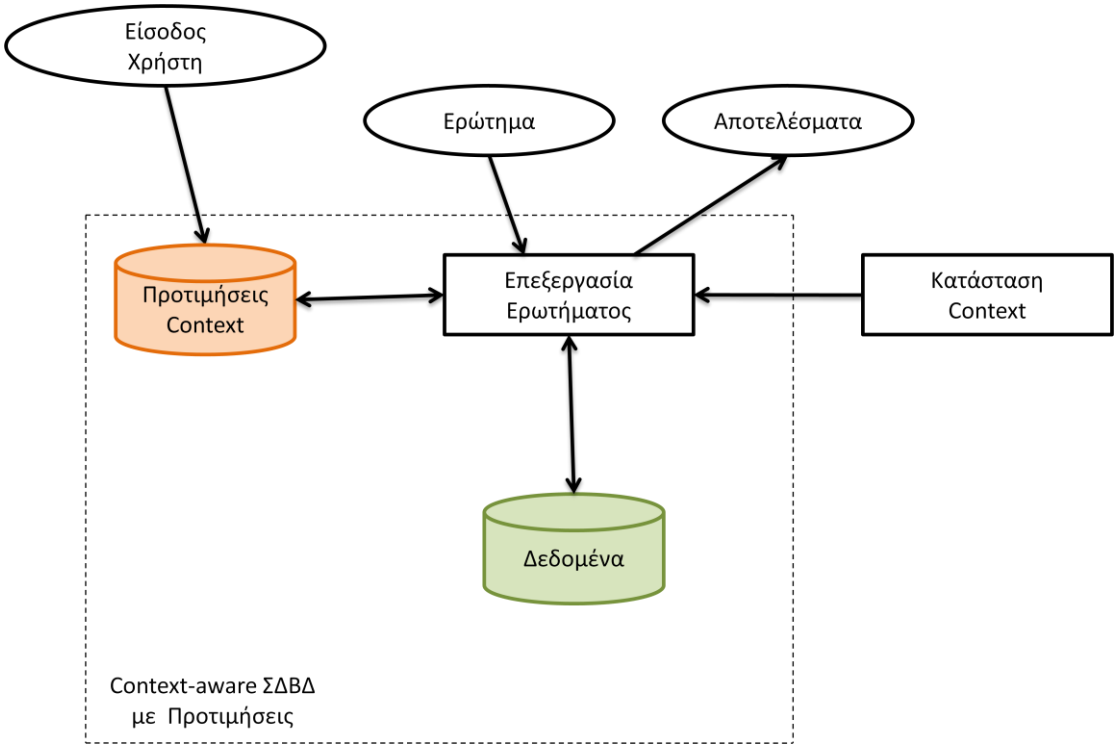
Τέλος, όπως αναφέρεται στο [FAJ04], δεδομένου ενός context-aware ερωτήματος , η επεξεργασία ερωτήματος διαιρείται σε 3 φάσεις, συγκεκριμένα στην *προ-επεξεργασία ερωτήματος* (query pre-processing), στην *εκτέλεση ερωτήματος* (query execution) και στην *μετα-επεξεργασία ερωτήματος* (post-processing query). Οι φάσεις αυτές, όπως και γενικότερα

η αρχιτεκτονική του συστήματος, παρουσιάζονται στο Σχήμα 2.4. Εκτός από την εκτέλεση ερωτημάτων στους δημόσιους διαχειριστές δεδομένων, τα υπόλοιπα ερωτήματα εκτελούνται στον ιδιωτικό διαχειριστή δεδομένων μέσω του context-aware query coordinator. Τέλος, μετά την εκτέλεση του ερωτήματος, η φάση της μετα-επεξεργασίας ερωτήματος αναδιαμορφώνει το αποτέλεσμα του ερωτήματος με βάση το context του χρήστη, μέσω της ταξινόμησης του αποτελέσματος.

2.3.3 Ένα context-aware σύστημα Βάσεων Δεδομένων που βασίζεται στις Προτιμήσεις (Preferences)

Το συγκεκριμένο σύστημα προτείνεται στο [SPV05] και εισάγει τις προτιμήσεις (preferences) του χρήστη δημιουργώντας με αυτόν τον τρόπο ένα context-aware μοντέλο προτιμήσεων για βάσεις δεδομένων. Στην εν λόγω εργασία παρέχονται κάποιοι ορισμοί, τους οποίους είναι χρήσιμο να αναφέρουμε σε αυτό το σημείο:

- *Domains.* Ένα *Domain* είναι ένα πεπερασμένο μετρήσιμο σύνολο τιμών.
- *Ιδιότητες* (Attributes) και *Σχέσεις* (Relations). Έστω ένα μετρήσιμο σύνολο από ονόματα ιδιοτήτων. Κάθε ιδιότητα χαρακτηρίζεται από ένα όνομα και ένα domain. Κατά τα γνωστά, ένα σχεσιακό σχήμα (relation schema) είναι ένα πεπερασμένο σύνολο από attributes και ένα σχεσιακό στιγμιότυπο (relation instance) είναι ένα πεπερασμένο υποσύνολο του καρτεσιανού γινομένου των domains του σχεσιακού σχήματος.
- *Παράμετροι* (Parameters) *Context*. Η μοντελοποίηση του context γίνεται μέσω ενός πεπερασμένου συνόλου attributes που ονομάζονται παράμετροι context. Δοθείσης μίας context-aware εφαρμογής, το περιβάλλον context αυτής της εφαρμογής αποτελείται από ένα σύνολο παραμέτρων context.
- *Κατάσταση* (State) *Context*. Γενικά, μία κατάσταση context είναι μία ανάθεση τιμών στις παραμέτρους context.
- *Ιεραρχία Ιδιοτήτων*. Οι ιδιότητες που αποτελούν το context είναι δυνατό να ανήκουν σε κάποια ιεραρχία, προκειμένου τα context-aware δεδομένα να είναι πιο αντιπροσωπευτικά.
- *Δυναμικές* (Dynamic) και *Στατικές* (Static) *Παράμετροι Context*. Σε αυτό το σημείο γίνεται διάκριση μεταξύ των δυναμικών και των στατικών παραμέτρων. Οι στατικές παράμετροι παίρνουν μία συγκεκριμένη τιμή. Από την άλλη πλευρά, στις δυναμικές παραμέτρους ανατίθενται τιμές από μία συνάρτηση, το αποτέλεσμα της οποίας είναι κάθε φορά το αποτέλεσμα του domain της context παραμέτρου.



Σχήμα 2.5. Η αρχιτεκτονική του μοντέλου που προτείνεται στο [SPV05]

- *Προτιμήσεις Context*. Οι προτιμήσεις context αφορούν το χρήστη και χρησιμοποιούνται στο προτεινόμενο σύστημα έτσι ώστε να αποθηκεύεται ένας βαθμός για κάποια στιγμιότυπα context για κάθε χρήστη. Το παράδειγμα που αναφέρεται στο [SPV05] είναι το εξής: έστω μία βάση δεδομένων που περιλαμβάνει χρήστες και εστιατόρια τα οποία οι χρήστες επισκέπτονται. Έστω, επίσης, ότι το context για κάποια εφαρμογή περιλαμβάνει τα attributes Τοποθεσία και Καιρός. Έστω, τέλος, η χρήστης Μαίρη και το εστιατόριο BeauBrummel, στο οποίο βρίσκεται κοντά στην Ακρόπολη της Αθήνας και σερβίρει γαλλική κουζίνα. Η Μαίρη προτιμά τη γαλλική κουζίνα όταν καιρός είναι συννεφιασμένος. Έτσι, όταν αυτή βρίσκεται κοντά στην Ακρόπολη και ο καιρός είναι συννεφιασμένος ανατίθεται ένας μεγάλος βαθμός για το εστιατόριο BeauBrummel. Έχουμε, δηλαδή

```
preference1 (Acropolis, BeauBrummel, Mary) = 0.8,
preference2 (cloudy, BeauBrummel, Mary) = 0.9
```

Οι παραπάνω *βασικές* προτιμήσεις αντιστοιχούν η κάθε μία σε μία context παράμετρο. Ορίζονται, επίσης, οι συγκεντρωτικές (aggregate) προτιμήσεις, κάθε μία από τις οποίες λαμβάνει υπόψη μία ή περισσότερες παραμέτρους context. Αυτό ονομάζεται στο [SPV05] *interest score*. Δηλαδή για ένα σύνολο από παραμέτρους context c_i και ένα σύνολο από παραμέτρους μη-context A_i έχουμε:

```
Preference (c1, . . . ck,Ak+1, . . . ,An) = interest score
```

Επιπλέον για κάθε εφαρμογή ορίζεται μία τιμή-βάρος (weight) για κάθε προτίμηση ανάλογα με το πόσο σημαντική είναι αυτή η προτίμηση.

Τέλος, στο [SPV05] συζητείται ο τρόπος υλοποίησης του παραπάνω μοντέλου χρησιμοποιώντας ένα σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ). Ουσιαστικά, δηλαδή, περιγράφεται ο τρόπος αποθήκευσης της πληροφορίας (context και μη-context) και ο τρόπος εκτέλεσης ερωτημάτων. Οι προτιμήσεις οργανώνονται σε κύβους δεδομένων (data cubes), ακολουθώντας το μοντέλο OLAP (On-Line Analytical Processing) συστημάτων. Έτσι, δημιουργούνται τόσοι κύβοι όσες και οι παράμετροι context. Όσον αφορά την ιεραρχία context, αυτή αποθηκεύεται με τη βοήθεια ενός επιπλέον attribute, του επιπέδου (Level), οι τιμές του οποίου αντιστοιχούν στο εκάστοτε επίπεδο της ιεραρχίας. Έχοντας επιτύχει την αποθήκευση του μοντέλου που περιγράφουμε σε ένα σχεσιακό ΣΔΒΔ, τα ερωτήματα μπορούν να γίνουν με χρήση κλασικής SQL χωρίς να χρειαστεί να γίνει επέκταση στο συντακτικό αυτής. Συνολικά η αρχιτεκτονική του μοντέλου παρουσιάζεται στο Σχήμα 2.5.

2.3.4 Chameleon: Ένα context-aware μοντέλο για ΣΔΒΔ

Στο [EAM09] προτείνεται ένα context-aware μοντέλο για ΣΔΒΔ, το οποίο ονομάζεται Chameleon. Για να περιγραφεί η λειτουργία και δομή του μοντέλου σε πρώτη φάση γίνεται ένας διαχωρισμός μεταξύ του *context αντικειμένου* (object) και του *context χρήστη* (user). Το context αντικειμένου είναι το context που διαχειρίζεται δεδομένα που μπορούν να συμπεριληφθούν σε ερωτήματα. Το context χρήστη αναφέρεται στο context αυτού που υποβάλει το ερώτημα. Αποτελείται από κάθε πληροφορία σχετική με το χρήστη. Επίσης, εισάγονται τρεις διαστάσεις (dimensions) για το context, η διάσταση 1) *πρόσημο context* (context sign), η διάσταση 2) *σχέση context* (contextual relation) και η διάσταση 3) *λίστα δεδομένων* (listing of data). Το context sign μπορεί να πάρει τις τιμές θετικό (positive) / αρνητικό (negative). Το θετικό context ορίζει τι είναι το context, ενώ το αρνητικό context ορίζει τι δεν είναι το context. Η contextual relation είναι μία relation που αφορά context δεδομένα. Τέλος, το listing of data αναφέρεται στο πως τα δεδομένα πρέπει να τοποθετηθούν σε λίστα.

Στο Chameleon ορίζεται μία παραλλαγή της κλασσικής SQL και εισάγονται νέες DDL εντολές, προκειμένου να υποστηρίζεται η διαχείριση της context πληροφορίας. Αυτές είναι οι ακόλουθες:

- *Δημιουργία Context Αντικειμένου*: Στο Chameleon χρησιμοποιείται το `CREATE OBJECT CONTEXT` για να ορίσει ένα context αντικειμένου όταν αυτό δεν είναι μέρος της σχέσης αντικειμένου. Η εντολή συντάσσεται ως εξής:

```
CREATE OBJECT CONTEXT contextname (
{col_spec | table_constraint} [,...]
, table_binding
);
```

- *Binding Table*: Η εντολή `BINDING KEY` χρησιμοποιείται για να συνδέει το αντικείμενο με το context του. Συντάσσεται ως εξής:

```
BINDING KEY ([col_name [,...]])
REFERENCES ref_table [( ref_col [,...] )] WITH bool_expr
```

Το πρώτο μέρος της εντολής είναι παρόμοιο με `FOREIGN KEY`, με τη διαφορά ότι το `BINDING KEY` δε χρειάζεται να αναφέρεται σε ένα πρωτεύον κλειδί (primary key).

- *Δημιουργία Context Χρήστη*: Για κάθε πίνακα που επηρεάζεται από ένα context χρήστη, χρησιμοποιείται ένα binding key για να φανεί πως το context αντικατοπτρίζεται στον πίνακα. Η σύνταξη που χρησιμοποιείται στο σύστημα Chameleon για να οριστεί ένα context χρήστη είναι η ακόλουθη:

```
CREATE [context_sign] CONTEXT contextname (
{col_spec | table constraint} [,...]
, table binding [,...]
[, substituting_key [,...]]
)
[AS contextual_relation_clause]
[WITH UNLISTED unlisted status];
context_sign: positive | negative
contextual_relation: equivalence | partial order
| total order [USING ordering func]
unlisted_status: excluded | included
```

Με τη δημιουργία ενός context χρήστη δημιουργείται επίσης μία στήλη (column), στην οποία αποθηκεύεται το username για κάθε χρήστη. Συνεπώς, κάθε τιμή context σχετίζεται με ένα συγκεκριμένο χρήστη. Επίσης, αν μία σχέση χρησιμοποιείται για αποθήκευση context, τότε άλλη μία στήλη δημιουργείται, η οποία αποθηκεύει το βαθμό (rank) κάθε context τιμής. Αυτός ο βαθμός μπορεί είτε να εισαχθεί απευθείας από της εφαρμογή ή να υπολογιστεί με τη χρήση μιας συνάρτησης.

- *Καθολική Αλλαγή* (Global Substitution): Η καθολική αλλαγή εφαρμόζεται στην περίπτωση που κάποια attributes πρέπει να τροποποιηθούν για λόγους παρουσίασης. Για παράδειγμα, αν το context ενός χρήστη είναι η περιοχή, και ο χρήστης βρίσκεται στη Γαλλία, τότε μπορεί να θέλουμε να μετατρέψουμε όλες τις τιμές σε όλους τους πίνακες σε Ευρώ. Αυτή η μετατροπή ονομάζεται καθολική αλλαγή, καθώς αυτή συμβαίνει σε όλους τους πίνακες που αφορούν το τρέχον context. Η αντίστοιχη εντολή συντάσσεται ως εξής:

```
SUBSTITUTE table_name(col_name) BY expression;
```

- *Ορίζοντας τα Ενεργά Contexts*: Η εκάστοτε εφαρμογή ενδέχεται να διαχειρίζεται πολλά contexts, τα οποία δε χρειάζονται όλα να είναι ενεργά όλη την ώρα. Για αυτό

το λόγο, στο Chameleon εισάγεται η εντολή `SET ACTIVE CONTEXT` για να οριστούν τα τρέχοντα/ενεργά contexts για τον εν λόγω χρήστη. Η εντολή συντάσσεται ως εξής:

```
SET ACTIVE CONTEXT [FOR USER user_name]
AS context_name [,...];
{ [WITH RANKING ORDER context_name [,...] ]
| [WITH RANKING EXPRESSION expression
| [WITH SKYLINE OF expression {MAX|MIN} [,...] ] };
```

Η εντολή αυτή παρέχει τη δυνατότητα διαχείρισης σύνθετων contexts που προέρχονται από βασικά contexts.

2.4 Ένα μοντέλο για context-aware σχεσιακές βάσεις δεδομένων

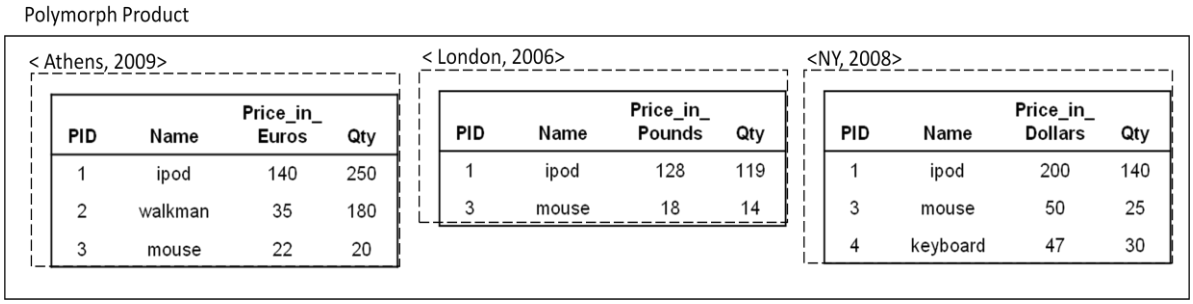
Το μοντέλο που περιγράφεται στην ενότητα 2.3.1 προτείνει μία προσέγγιση του θέματος για ημιδομημένα (XML) δεδομένα. Επίσης, στην ενότητα 2.3.4 το σύστημα που προτείνεται αποτελεί ουσιαστικά ένα middleware χωρίς να ενσωματώνει το context στο επίπεδο των δεδομένων (data layer). Τέλος, στις ενότητες 2.3.2 και 2.3.3 τα προτεινόμενα μοντέλα αφορούν τη μοντελοποίηση του context ως προς τις προτιμήσεις (preferences) του χρήστη. Γίνεται, λοιπόν, σαφές ότι στο χώρο της διαχείρισης δεδομένων δεν έχει προταθεί ένα μοντέλο για σχεσιακά ΣΔΒΔ το οποίο να ενσωματώνει το context στο επίπεδο των δεδομένων.

Έτσι, λοιπόν, στην παρούσα ενότητα προχωρούμε στην παρουσίαση του μοντέλου για context-aware σχεσιακές βάσεις δεδομένων που προτείνεται στο [RS09], στο οποίο βασίστηκε η παρούσα διπλωματική, και αφορά την ενσωμάτωση του context σε ένα context-aware σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων.

Στο [RS09] επεκτείνεται η κατηγοριοποίηση για το context που προτείνεται στο [FAJ04] και που αναφέραμε στην ενότητα 2.1, προσθέτοντας στις κατηγορίες α) του context που σχετίζεται με το χρήστη και β) του context που σχετίζεται με το περιβάλλον, την κατηγορία γ) του σχετικού με το σύστημα context, όπως για παράδειγμα χαρακτηριστικά της εκάστοτε συσκευής ή του εκάστοτε δικτύου. Αναφέρουμε και για αυτό το μοντέλο κάποιους όρους και συμβολισμούς που πρέπει να περιγραφούν:

- Το *context schema* C_s είναι το σύνολο των attributes που το αποτελούν, για παράδειγμα <Location, Date>.
- Το *context instance* C_i αποτελεί μία ανάθεση τιμών για ένα συγκεκριμένο context schema, για παράδειγμα <Athens, 2009>.

- Ο *context specifier* C_p, στο οποίο μπορεί να αποθηκευτεί ένα context instance ή ένα σύνολο από context instances, για παράδειγμα {<Athens, 2009>, <London, 2006>, <NY, 2008>}.
- Μία *context relation* R_c αποτελείται από α) ένα context schema και β) ένα σύνολο από ζεύγη μιας κλασικής σχέσης (relation) και ενός context instance. Κάθε τέτοια (απλή) σχέση, που ορίζεται βάσει ενός context instance, ονομάζεται *morph*. Σε αυτήν την περίπτωση, η context relation μπορεί εναλλακτικά να ονομαστεί και *polymorph*. Έτσι, συνοψίζοντας, ένα polymorph ορίζεται με βάση ένα context schema και αποτελείται από ένα ή περισσότερα morphs, κάθε ένα από τα οποία ορίζεται με βάση ένα διαφορετικό context instance του προαναφερθέντος context schema. Αντίστοιχα με το context schema ορίζεται και το polymorph schema. Για παράδειγμα, έστω το polymorph Product, που ορίζεται με βάση το context schema <Location, Date>. Κάθε morph που κρατάει πληροφορίες για διάφορα προϊόντα περιέχει γενικά διαφορετικά δεδομένα και σχήμα για διαφορετικές περιοχές και χρονολογίες. Έτσι, για τη NY, όπου ορίζεται ο φόρος, το αντίστοιχο attribute (VAT) θα μπορούσε να ορισθεί, ενώ πχ για την Αθήνα όχι. Το polymorph Product, λοιπόν, θα μπορούσε να αποτελείται από τα ακόλουθα 3 morphs: α) για το context instance <Athens, 2009>, το morph με σχήμα (Product_id, Name, Price_in_Euros, Quantity), β) για το context instance <London, 2006>, το morph με σχήμα (Product_id, Name, Price_in_Pounds, Quantity) και γ) για το context instance <NY, 2008> το morph με σχήμα (Product_id, Name, Price_in_Dollars, Quantity, VAT). Φυσικά, για το κάθε ένα από τα 3 αυτά morphs εκτός από το σχήμα διαφοροποιούνται και τα δεδομένα. Ένα τέτοιο παράδειγμα παρουσιάζεται στο Σχήμα 2.6.

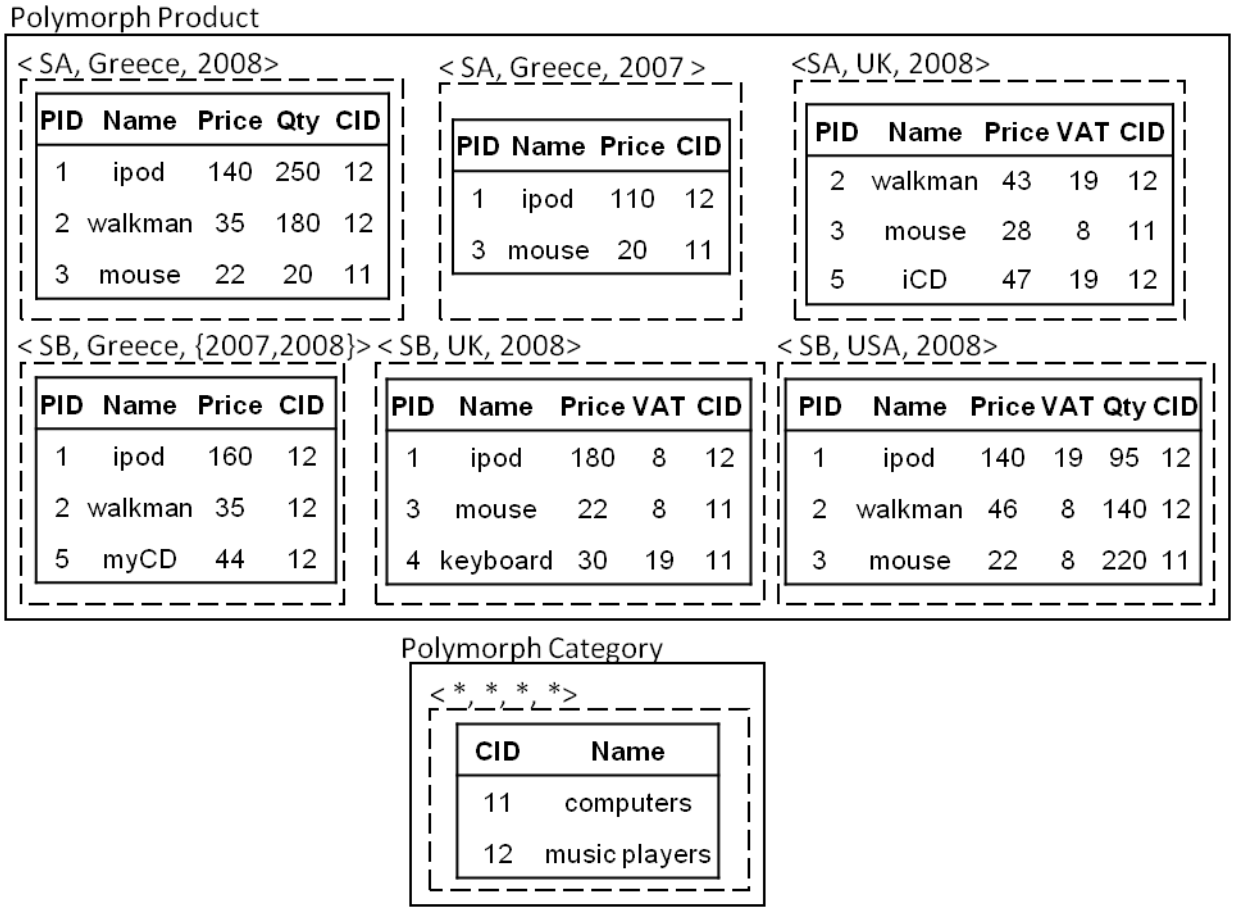


Σχήμα 2.6. Παράδειγμα του polymorph Product, το οποίο έχει οριστεί με βάση το context <Location, Date>

Όπως, ίσως, έγινε φανερό από τα παραπάνω, σε ότι αφορά το context χρησιμοποιείται ο συμβολισμός < >. Από εδώ και στο εξής, θα κάνουμε χρήση αυτού του συμβολισμού, όπως επίσης και των όρων που αναφέρθηκαν προηγουμένως.

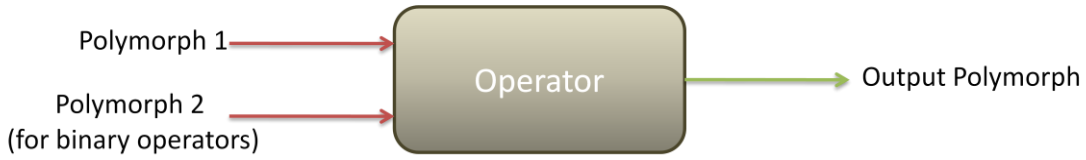
Για να μπορέσει το μοντέλο που περιγράφουμε να είναι ολοκληρωμένο και να υλοποιηθεί ως ένα context-aware Σύστημα Διαχείρισης Βάσεων Δεδομένων, στο [RS09] γίνεται μία επέκταση της κλασικής σχεσιακής άλγεβρας. Επίσης, επεκτείνονται οι αντίστοιχοι σχεσιακοί τελεστές και ορίζονται κάποιοι νέοι, προκειμένου να είναι δυνατή η προσπέλαση των νέων context-aware δεδομένων και να μπορεί ο χρήστης να κάνει ερωτήματα πάνω σε αυτό το context-aware σύστημα.

Στο σημείο αυτό παραθέτουμε ένα πιο ολοκληρωμένο παράδειγμα (Σχήμα 2.7), προκειμένου να δείξουμε τα αποτελέσματα της εφαρμογής συγκεκριμένων τελεστών σε αυτό το παράδειγμα. Σύμφωνα με αυτό, ορίζονται δύο polymorphs, το Product και το Category, τα οποία μοντελοποιούν μία αγορά ηλεκτρονικών προϊόντων. Αυτά τα polymorphs ορίζονται με βάση το context schema <Supplier, Location, Date>. Το καινούριο context attribute Supplier αναφέρεται σε διαφορετικούς πωλητές για τα συγκεκριμένα προϊόντα. Στο παράδειγμά μας ορίζονται δύο διαφορετικοί πωλητές, ο SA και ο SB.



Σχήμα 2.7. Παράδειγμα context-aware βάσης δεδομένων

Ένας ***τελεστής*** (operator) μπορεί να είναι είτε unary είτε binary. Ένας unary τελεστής παίρνει ως είσοδο ένα polymorph και παράγει ως έξοδο ένα polymorph. Από την άλλη πλευρά, ένας binary τελεστής παίρνει ως είσοδο δύο polymorphs και παράγει ως έξοδο ένα polymorph. Σχηματικά, αυτό φαίνεται στο Σχήμα 2.8.



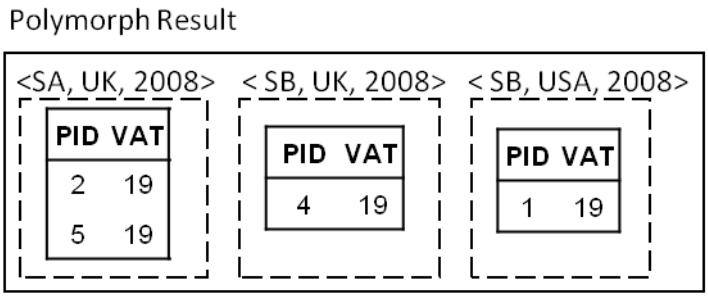
Σχήμα 2.8. Γενικό διάγραμμα τελεστή

Έτσι, αναφέρουμε ακολούθως τις επεκτάσεις των τελεστών για το context-aware μοντέλο που προτείνεται.

Ο ***τελεστής Project*** (προβολή) συμβολίζεται $\pi_{\{A1,A2,\ldots,An\}}R_c$.

Ο project πραγματοποιεί ένα κλασικό project σε κάθε morph του polymorph R_c στο οποίο ορίζονται όλα τα attributes A_1, A_2,…, A_n. Αν σε ένα morph δεν ορίζεται έστω και ένα από τα προαναφερθέντα attributes, τότε αυτό το morph αποκλείεται από το τελικό polymorph που παράγεται ως αποτέλεσμα. Για παράδειγμα, στη βάση δεδομένων του Σχήματος 2.7 η εκτέλεση του τελεστή $\pi_{\{VAT,\ Qty\}}$Product θα επιστρέψει ένα polymorph με ένα μοναδικό morph με βάση το context <SB, USA, 2008>.

Ο ***τελεστής Select*** (επιλογή) συμβολίζεται $\sigma_{\{CONDITIONS\}}R_c$.



Σχήμα 2.9. Αποτέλεσμα του select τελεστή για τη βάση δεδομένων του Σχήματος 2.7

Ο select πραγματοποιεί ένα κλασικό select σε κάθε morph του polymorph R_c στο οποίο ορίζονται όλα τα attributes που εμφανίζονται στις CONDITIONS. Ομοίως, με τον project, έτσι και στον select, αν σε ένα morph δεν ορίζεται έστω και ένα από τα προαναφερθέντα attributes, τότε αυτό το morph αποκλείεται από το τελικό polymorph που παράγεται ως

αποτέλεσμα. Επιστρέφοντας στο παράδειγμα του Σχήματος 2.7, έστω ένα ερώτημα για τα προϊόντα που έχουν υψηλό φόρο (VAT). Το αποτέλεσμα αυτού του ερωτήματος παρουσιάζεται στο Σχήμα 2.9.

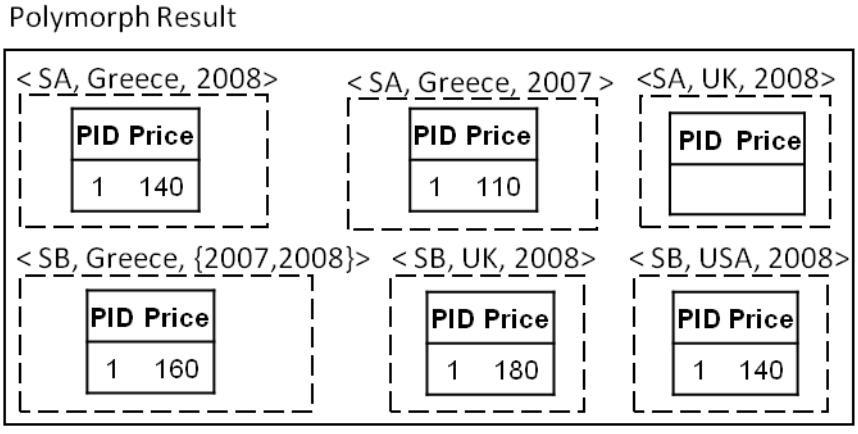
Για τους binary τελεστές που ακολουθούν, πρέπει να αναφέρουμε ότι για να είναι συμβατή μία πράξη με binary τελεστή, πρέπει τα δύο input polymorphs να έχουν οριστεί με βάση το ίδιο context schema. Σε αντίθετη περίπτωση, η πράξη δε θεωρείται έγκυρη και συνεπώς δεν μπορεί να εκτελεστεί ο αντίστοιχος τελεστής.

Ο ***τελεστής Cartesian Product*** (καρτεσιανό γινόμενο) συμβολίζεται $R_c \times S_c$.

Όπως αναφέραμε προηγουμένως, για να είναι έγκυρο το καρτεσιανό γινόμενο, πρέπει τα polymorphs R_c, S_c να έχουν οριστεί με βάση το ίδιο context schema. Κατά τη διάρκεια εκτέλεσης ενός καρτεσιανού γινομένου πραγματοποιείται ένα καρτεσιανό γινόμενο για κάθε ζευγάρι morphs των polymorphs R_c, S_c που έχουν οριστεί με βάση το ίδιο context instance. Αν υπάρχει ένα morph σε ένα από τα δύο polymorphs (έστω στο R_c), ορισμένο για το context instance C_i και δεν υπάρχει για το C_i κάποιο morph ορισμένο στο άλλο polymorph (S_c), τότε το morph του R_c δε συμπεριλαμβάνεται στο αποτέλεσμα. Μπορούμε να χρησιμοποιήσουμε τον τελεστή του καρτεσιανού γινομένου για να ζητήσουμε το id και την τιμή των προϊόντων στην κατηγορία "music players" που κοστίζουν περισσότερο από 50, δηλαδή το ερώτημα:

Result= $\pi_{\{Pid,\ Price\}}(\sigma_{(Price\ >\ 50)\ AND\ (Category.Name\ =\ "music\ players")\ AND\ (Product.CID\ =\ Category.CID)}$ Product × Category))

Το αποτέλεσμα του ανωτέρω ερωτήματος παρουσιάζεται στο Σχήμα 2.10. Αξίζει να σημειωθεί ότι καθώς το μοναδικό morph του polymorph Category έχει οριστεί με βάση το context <*, *, *>, ταιριάζει με οποιοδήποτε morph του polymorph Product κατά το καρτεσιανό γινόμενο.



Σχήμα 2.10. Αποτέλεσμα του τελεστή καρτεσιανό γινόμενο για τη βάση δεδομένων του Σχήματος 2.7

Οι ***τελεστές συνόλων Union*** (ένωση), ***Intersection*** (τομή), ***Difference*** (διαφορά) συμβολίζονται $R_c \cup S_c$, $R_c \cap S_c$, $R_c - S_c$ αντίστοιχα.

Κατά τα γνωστά, για να είναι έγκυρη κάθε πράξη συνόλων, πρέπει τα polymorphs R_c, S_c να έχουν οριστεί με βάση το ίδιο context schema. Όσον αφόρα τα morphs, η πράξη $R_c \cup S_c$ πραγματοποιεί μία ένωση στα morphs ανά δύο (για τα polymorphs R_c, S_c) που έχουν οριστεί με βάση το ίδιο context-instance και επίσης είναι union-compatible (το union-compatibility ορίζεται όπως και στο απλό σχεσιακό μοντέλο). Επίσης, η ένωση συμπεριλαμβάνει και τα morphs εκείνα, που υπάρχουν για κάποιο context instance μόνο στο ένα από τα δύο polymorphs, δηλαδή μία ένωση του morph με το κενό σύνολο (ουσιαστικά καθεαυτό το morph). Στον τελεστή της τομής συμπεριλαμβάνονται μόνο τα morphs που έχουν οριστεί εκατέρωθεν με βάση το ίδιο context instance. Τέλος, στον τελεστή της διαφοράς συμπεριλαμβάνονται τα morphs εκείνα που έχουν οριστεί εκατέρωθεν με βάση το ίδιο context instance καθώς και εκείνα τα morphs, τα context instances των οποίων έχουν οριστεί μόνο στο αριστερό από τα δύο polymorphs, (στην πράξη $R_c - S_c$ αριστερό polymorph εννοούμε το polymorph R_c).

Ο ***τελεστής select schema*** συμβολίζεται $\sigma^c_{\{\text{CONDITIONS}\}} R_c$.

Ο select schema πραγματοποιεί ένα κλασικό select στο context. Το τελικό αποτέλεσμα περιλαμβάνει εκείνα τα morphs τα οποία έχουν οριστεί με βάση τα context instances που ικανοποιούν το προαναφερθέν select.

Ο ***τελεστής map*** συμβολίζεται Map(R_c, c_attr, ctxt_expression).

Κατά τον τελεστή map σε πρώτη φάση εντοπίζεται το context schema με βάση το οποίο έχει οριστεί το polymorph R_c και στη συνέχεια το context attribute c_attr. Αυτό το context attribute παίρνει ως τιμή το αποτέλεσμα της έκφρασης ctxt_expression. Αν μετά την εφαρμογή του τελεστή map σε ένα polymorph προκύψουν δύο ή περισσότερα context instances με τις ίδιες τιμές, τότε αυτά συγχωνεύονται σε ένα context instance και επίσης συγχωνεύονται τα αντίστοιχα morphs (με την ένωση αυτών, εφόσον είναι union-compatible) σε ένα morph ορισμένο με βάση το συγχωνευθέν context instance.

Ο ***τελεστής Add Context Attribute*** συμβολίζεται Add_Ctxt_attr(R_c, c_attr=c_val).

Κατά τον τελεστή add context attribute σε πρώτη φάση εντοπίζεται το context schema με βάση το οποίο έχει οριστεί το polymorph R_c και στη συνέχεια προστίθεται σε αυτό το context schema το context attribute c_attr με τιμή τη c_val για όλα τα ήδη ορισμένα morphs.

Ο ***τελεστής Delete Context Attribute*** συμβολίζεται Del_Ctxt_attr(R_c, c_attr).

Κατά τον τελεστή delete context attribute σε πρώτη φάση εντοπίζεται το context schema με βάση το οποίο έχει οριστεί το polymorph R_c και στη συνέχεια διαγράφεται από αυτό το context schema το context attribute c_attr. Αν μετά την εφαρμογή του τελεστή delete context

attribute σε ένα polymorph προκύψουν δύο ή περισσότερα morphs που ορίζονται με βάση τα ίδια context instances με τις ίδιες τιμές, τότε αυτά συγχωνεύονται σε ένα context instance και επίσης συγχωνεύονται τα αντίστοιχα morphs (με την ένωση αυτών, εφόσον είναι union-compatible) σε ένα morph ορισμένο με βάση το συγχωνευθέν context instance.

3

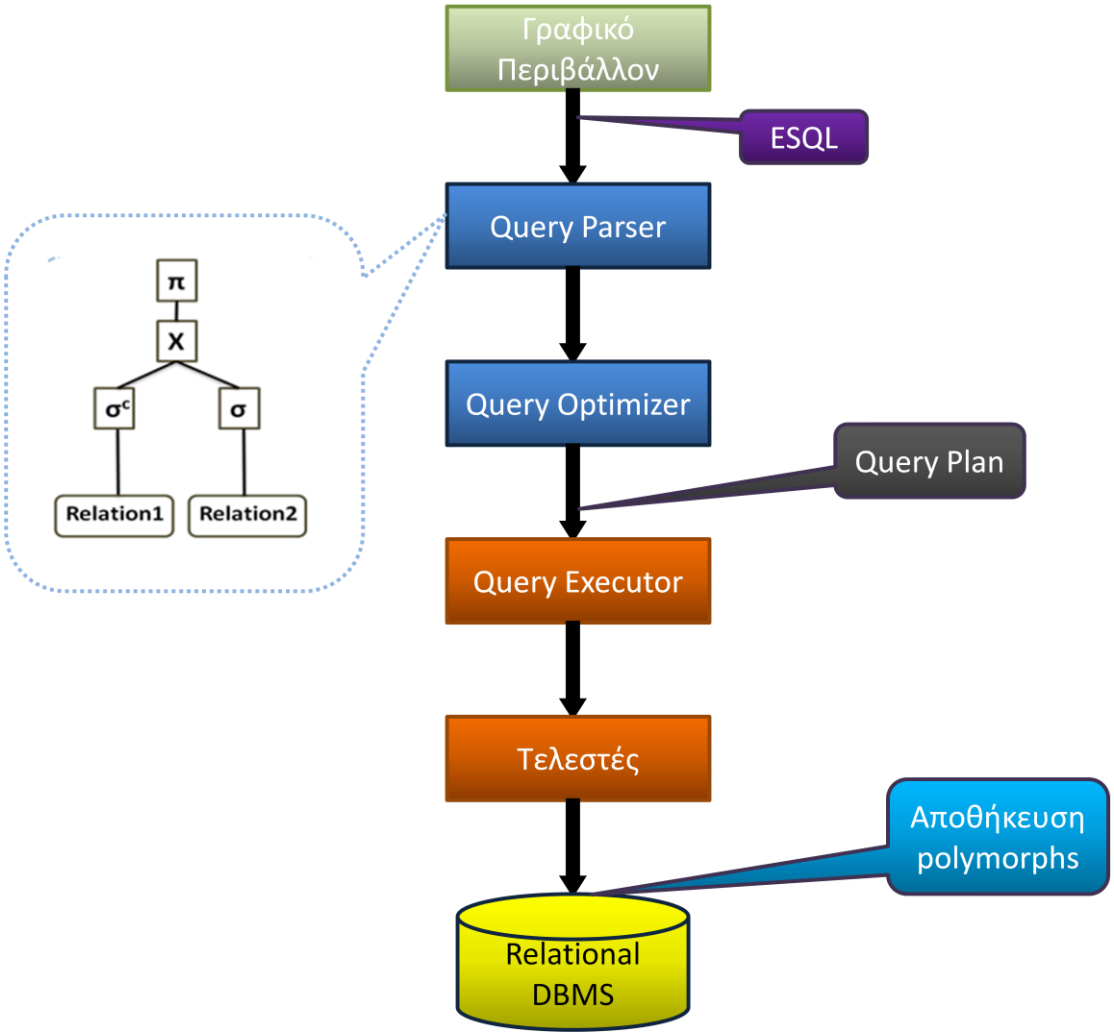
Ανάλυση και Σχεδίαση

Στο συγκεκριμένο κεφάλαιο περιγράφουμε την ανάλυση και τη σχεδίαση του συνολικού συστήματος, δηλαδή του context-aware Συστήματος Διαχείρισης Βάσεων Δεδομένων με βάση το μοντέλο που περιγράφηκε στο 2.4. Στην ενότητα 3.1 γίνεται περιγραφή της γενικής αρχιτεκτονικής του συστήματος, ενώ στην ενότητα 3.2 προχωρούμε σε μία πιο λεπτομερή ανάλυση της δομής και των λειτουργιών του υποσυστημάτων αποθήκευσης των context-aware relations και εκτέλεσης των τελεστών, τα οποία υλοποιούνται στην παρούσα διπλωματική. Τέλος, στην ενότητα 3.3 περιγράφονται οι διαφορετικές υλοποιήσεις που πραγματοποιήθηκαν, όσον αφορά στην αποθήκευση στο σχεσιακό ΣΔΒΔ.

3.1 Περιγραφή γενικής αρχιτεκτονικής συστήματος

Η γενική αρχιτεκτονική του context-aware Συστήματος Διαχείρισης Βάσεων Δεδομένων παρουσιάζεται σχηματικά στο Σχήμα 3.1. Ξεκινώντας από το κοινό σημείο διεπαφής του χρήστη με το σύστημα, υπάρχει αρχικά ένα γραφικό περιβάλλον. Το GUI (Graphical User Interface) αυτό δίνει τη δυνατότητα στο χρήστη να εισάγει ένα context-aware ερώτημα μέσω κατάλληλων κουμπιών (buttons), boxes κλπ. Σε αυτό το σημείο πρέπει να αναφέρουμε ότι η ροή του συστήματος είναι εκτός από «πάνω προς τα κάτω» όταν ο χρήστης εισάγει ένα ερώτημα (input), αλλά και από «κάτω προς τα πάνω» όταν το output του συστήματος προωθείται προς τα πάνω με τελικό προορισμό το σημείο διεπαφής του συστήματος με το

χρήστη. Επομένως, εκτός από την εισαγωγή του ερωτήματος από το χρήστη, το υποσύστημα που περιγράφουμε χρησιμεύει ακόμα στην προβολή των αποτελεσμάτων της εκτέλεσης του δοθέντος context-aware ερωτήματος με έναν εύχρηστο και γραφικό τρόπο για το χρήστη. Εξίσου σημαντικός, συνεπώς, είναι και ο τρόπος αυτός παρουσίασης των αποτελεσμάτων στο χρήστη, όπως πχ παρουσίαση των αποτελεσμάτων σε κύβο, δεντρική μορφή, ή την παρουσίαση συγκεκριμένων εκφάνσεων της πληροφορίας.



Σχήμα 3.1. Γενική αρχιτεκτονική context-aware συστήματος

Προχωρώντας προς τα κάτω, ο χρήστης εισάγει το context-aware ερώτημα μέσω της extended SQL (ESQL). Στα πλαίσια αυτού του υποσυστήματος, προβλέπεται η υλοποίηση ενός πλήρως λειτουργικού Query Parser. Έτσι, το ερώτημα, που έχει συνταχθεί σε ESQL, περνάει από τη διαδικασία του parsing και δημιουργούνται διάφορα πλάνα εκτέλεσης για αυτό το ερώτημα. Τα πλάνα εκτέλεσης είναι προφανώς δομημένα μέσω των νέων ορισμένων context-aware σχεσιακών τελεστών, οι οποίοι έχουν περιγραφεί συνοπτικά στην ενότητα 2.4

και παρουσιάζονται λεπτομερώς στην ενότητα 3.2.4. Έχοντας φτάσει σε αυτό το σημείο, πρέπει να επιλεχθεί το βέλτιστο (ή αν μη τι άλλο ένα αρκετά αποδοτικό) πλάνο εκτέλεσης, Έτσι, υλοποιούνται κάποιοι αλγόριθμοι βελτιστοποίησης και μέσω αυτών επιλέγεται το καταλληλότερο πλάνο εκτέλεσης όσον αφορά στην αποδοτικότητά του. Οι αλγόριθμοι αυτοί λαμβάνουν υπόψη το κόστος σε χώρο αλλά κυρίως σε χρόνο εκτέλεσης του ερωτήματος στο context-aware Σύστημα Διαχείρισης Βάσεων Δεδομένων. Τέλος, κατά την αντίστροφη ροή του συστήματος το τελικό αποτέλεσμα της εκτέλεσης του context-aware ερωτήματος περνάει και από αυτό το στάδιο στην πορεία του προς το interface το οποίο "βλέπει" ο χρήστης.

Τα επόμενα στάδια είναι το ***Υποσύστημα "εκτέλεσης πλάνου"*** και το ***Υποσύστημα "εκτέλεσης τελεστών".*** Σε αυτό το σημείο ως input θεωρείται το (βέλτιστο) πλάνο εκτέλεσης το οποίο, όπως αναφέραμε νωρίτερα, αποτελείται από ένα σύνολο από context-aware τελεστές. Έτσι, το εν λόγω υποσύστημα αναλαμβάνει την εκτέλεση του πλάνου εκτέλεσης μέσω της εκτέλεσης κάθε τελεστή του πλάνου από κάτω προς τα πάνω.

Το ***Υποσύστημα "αποθήκευσης των polymorphs"*** είναι υπεύθυνο για την αποθήκευση των δεδομένων που αναφέραμε σε ένα ήδη υπάρχον σχεσιακό ΣΔΒΔ. Πραγματοποιήθηκαν δύο υλοποιήσεις που αφορούν το μοντέλο της αποθήκευσης αυτής, οι οποίες περιγράφονται εκτενέστερα στην επόμενη ενότητα. Μία τρίτη υλοποίηση αποθήκευσης καθώς, επίσης, και μία native υλοποίηση του συστήματος απευθείας στο δίσκο θα μπορούσαν να αποτελούν μελλοντικές επεκτάσεις του παρόντος συστήματος. Τα θέματα αυτά μελετώνται εκτενέστερα στο Κεφάλαιο 6. Τέλος, για αυτά τα υποσυστήματα κατά την αντίστροφη ροή των δεδομένων προς το χρήστη, τα αποτελέσματα προωθούνται προς τα πάνω.

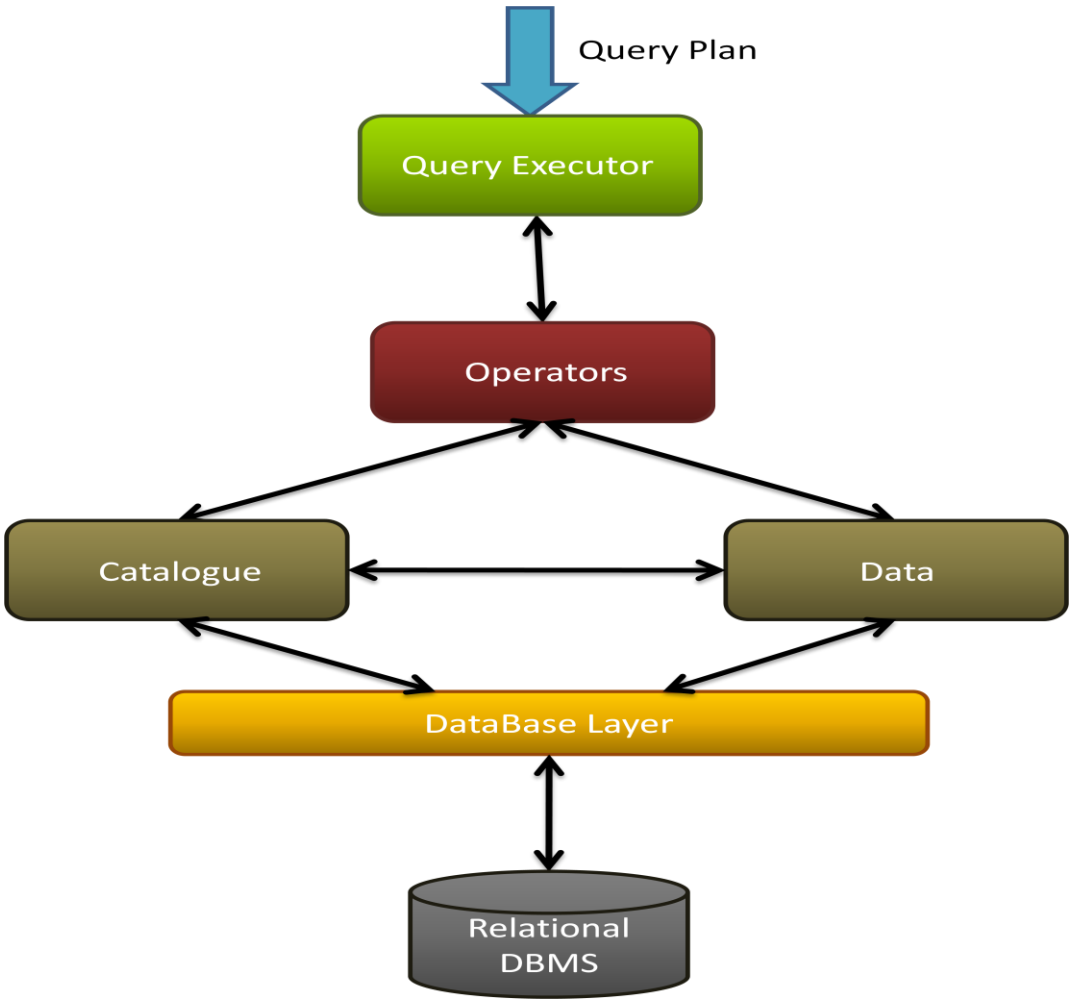
Κλείνοντας τη γενικότερη αναφορά μας στο συνολικό Context-Aware ΣΔΒΔ, όπως τονίσαμε και νωρίτερα, αναφέρουμε ότι τα τελευταία υποσυστήματα της εκτέλεσης του πλάνου, της εκτέλεσης των τελεστών και της αποθήκευσης των polymorphs είναι αυτά που υλοποιήθηκαν κατά τη διάρκεια της διπλωματικής. Στην επόμενη ενότητα, λοιπόν, αναλύουμε λεπτομερέστερα τη δομή και λειτουργία αυτών των υποσυστημάτων και σχολιάζουμε τα προβλήματα που προέκυψαν κατά τη σχεδίαση αυτού καθώς και τις αποφάσεις που πήραμε προκειμένου να τα υπερκεράσουμε.

3.2 Αρχιτεκτονική επιμέρους Υποσυστημάτων

Όπως αναφέραμε παραπάνω, το σύστημα που υλοποιήσαμε στην παρούσα διπλωματική αποτελείται από τα ***Υποσυστήματα "εκτέλεσης πλάνου", "εκτέλεσης τελεστών" και "αποθήκευσης τελεστών"***. Αυτά με τη σειρά τους αποτελούνται από επιμέρους μονάδες

(modules). Τα κύρια modules, λοιπόν, που συνιστούν το σύστημα φαίνονται στο Σχήμα 3.2 και είναι τα ακόλουθα:

- το "Υποσύστημα **Αποθήκευσης και Διεπαφής με το Σχεσιακό ΣΔΒΔ** (DataBase Layer)", δηλαδή ένα επίπεδο επικοινωνίας του κυρίως συστήματος με το σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων,
- το "Υποσύστημα του **Καταλόγου** (Catalogue)", το οποίο χειρίζεται τα μετα-δεδομένα του context-aware συστήματος,



Σχήμα 3.2. Αρχιτεκτονική context-aware υποσυστημάτων Εκτέλεσης και Αποθήκευσης Τελεστών

- το "Υποσύστημα **Διαχείρισης Δεδομένων** (Data)", το οποίο χειρίζεται τις εισαγωγές/ενημερώσεις/διαγραφές δεδομένων στο σύστημα και είναι επίσης υπεύθυνο για όλες τις λειτουργίες αποθήκευσης και ανάκτησης της context aware πληροφορίας προς και από το Σχεσιακό ΣΔΒΔ που χρησιμοποιούμε ως physical layer,

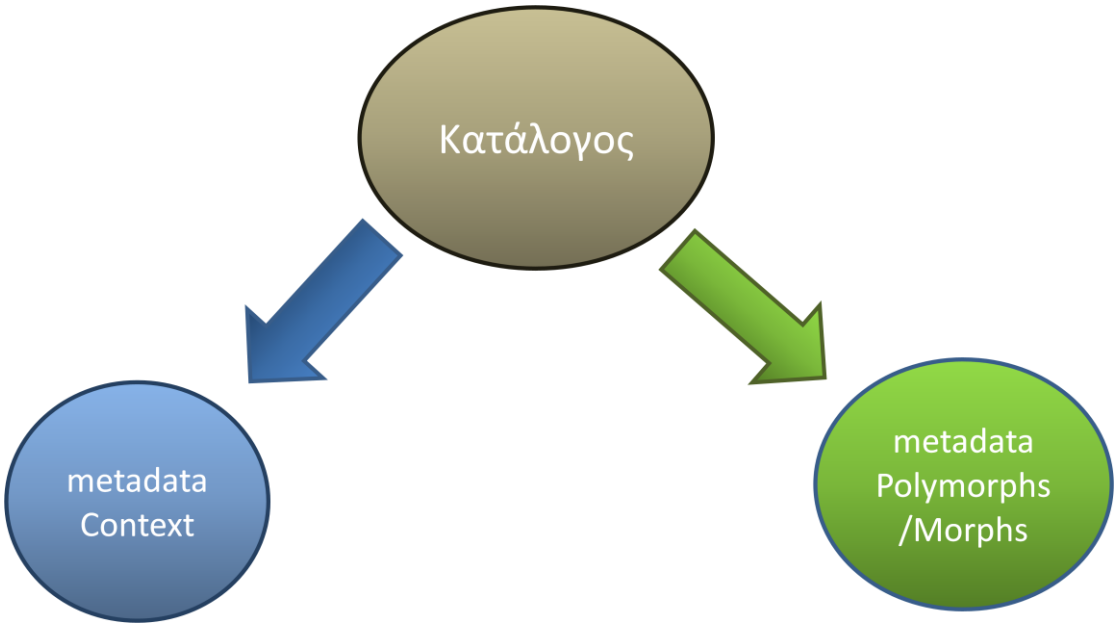
- το "Υποσύστημα **Εκτέλεσης Τελεστών** (Operators)", το οποίο περιλαμβάνει την υλοποίηση και εκτέλεση όλων των context-aware τελεστών και τέλος
- το "Υποσύστημα **Εκτέλεσης Πλάνων** (Query Executor)", η οποία αναλαμβάνει να εκτελέσει το εισαχθέν πλάνο εκτέλεσης, αφού ελέγξει πρώτα την εγκυρότητα του.

3.2.1 Υποσύστημα Αποθήκευσης και Διεπαφής με το Σχεσιακό ΣΔΒΔ (DataBase Layer)

Το module του DataBase Layer αποτελεί ένα "στρώμα" επικοινωνίας ολόκληρου του συστήματος με το σχεσιακό ΣΔΒΔ που τρέχει στο κατώτερο επίπεδο. Η επιλογή δημιουργίας μιας τέτοιας μονάδας έγινε, έτσι ώστε να παρέχεται ένα πιο high-level interface στα υψηλότερα modules όταν αυτά χρειάζεται να επικοινωνήσουν με τη βάση για να εκτελέσουν ερωτήματα ή DDL (Data Definition Language)/DML (Data Manipulation Language) εντολές.

3.2.2 Υποσύστημα του Καταλόγου (Catalogue)

Τα περισσότερα Συστήματα Διαχείρισης Βάσεων Δεδομένων διαθέτουν κάποιο υποσύστημα διαχείρισης των metadata της κάθε βάσης δεδομένων που δημιουργείται, το οποίο συνήθως ονομάζεται Κατάλογος. Αντίστοιχα, στο context-aware σύστημα που υλοποιήσαμε χρειάστηκε να δημιουργήσουμε μία μονάδα καταλόγου για αυτό τον σκοπό. Σε αυτή τη λογική αρχικά πραγματοποιήσαμε μία διάκριση μεταξύ των metadata που αφορούν το context και των metadata που αφορούν τα polymorphs.



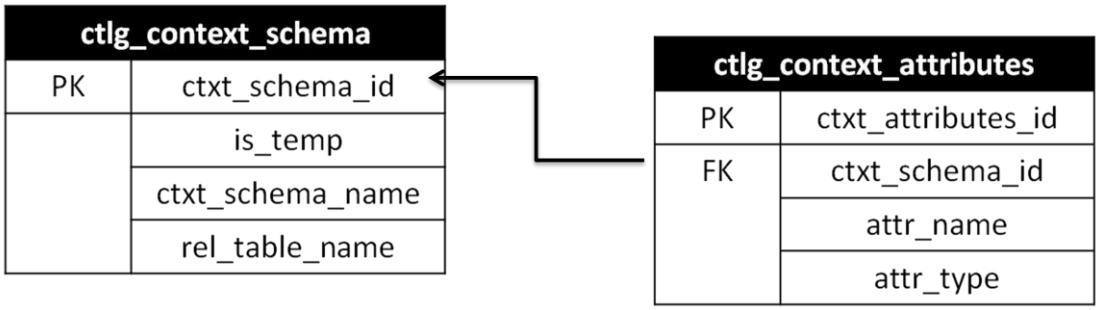
Σχήμα 3.3. Γενική λειτουργία του Καταλόγου

Πρέπει να τονίσουμε ότι για τις διαφορετικές υλοποιήσεις ως προς την αποθήκευση των polymorphs που πραγματοποιήθηκαν, το τμήμα του καταλόγου που διαχειρίζεται το context παραμένει αμετάβλητο. Από την άλλη πλευρά, επειδή η παράμετρος που διαφοροποιείται μεταξύ των υλοποιήσεων αυτών είναι η αποθήκευση των polymorphs, το τμήμα του καταλόγου που αφορά την διαχείριση των μετα-δεδομένων τους επίσης διαφοροποιείται για την κάθε υλοποίηση, προκειμένου να υποστηρίζει τη διαχείριση των metadata των polymorphs αποτελεσματικά ανάλογα με την υλοποίηση. Η γενική λειτουργία του Καταλόγου παρουσιάζεται στο Σχήμα 3.3.

Context

Το τμήμα διαχείρισης των metadata που αφορά το context αποτελείται από δύο πίνακες, τους ctlg_context_schema και ctlg_context_attributes. Όπως φαίνεται και από τις ονομασίες των πινάκων, στον πρώτο αποθηκεύονται πληροφορίες για το context schema (όνομα context, όνομα πίνακα που αποθηκεύει το context, προσωρινό (temporary) ή όχι context), ενώ στο δεύτερο πληροφορίες που αφορούν κάθε context attribute ξεχωριστά (όνομα context attribute, τύπος context attribute). Έτσι, για όλες τις υλοποιήσεις το σχήμα του πίνακα ctlg_context_schema είναι το ακόλουθο

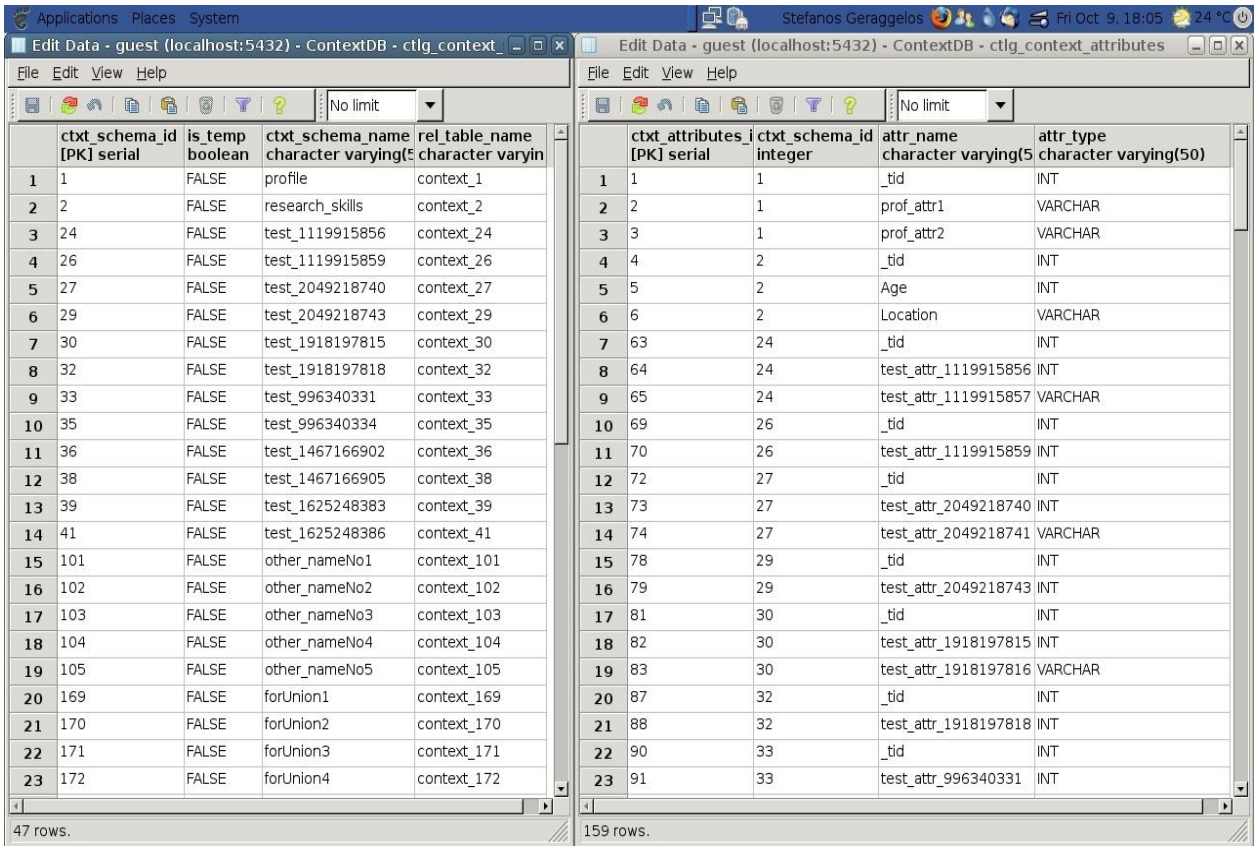
```
(ctxt_schema_id [primary key], is_temp [boolean], ctxt_schema_name [varchar(50)], rel_table_name [varchar(50)])
```



Σχήμα 3.4. Πίνακες καταλόγου που αφορούν το context

Το attribute ctxt_schema_id είναι το πρωτεύον κλειδί (primary key) για τον πίνακα ctlg_context_schema, προκειμένου το σύστημα να μπορεί να αναφερθεί σε κάθε context schema. Το is_temp είναι ένα boolean attribute που δείχνει αν το εν λόγω context schema είναι προσωρινό ή όχι. Η έννοια των προσωρινών contexts είναι αντίστοιχη των προσωρινών polymorphs για τα polymorphs. Μερικοί τελεστές παράγουν, όπως φαίνεται παρακάτω, εκτός από temporary polymorphs και temporary contexts. Έτσι, προκύπτει η ανάγκη δημιουργίας

κάποιων προσωρινών contexts ως αποτέλεσμα εκτέλεσης κάποιων τελεστών, προκειμένου να υπάρχει η δυνατότητα στο σύστημα να διαγράφει όλους τους προσωρινούς πίνακες απευθείας. Πρέπει να τονίσουμε, επίσης, ότι ένα προσωρινό polymorph μπορεί να οριστεί με βάση είτε ένα προσωρινό είτε ένα μη προσωρινό context, ενώ αντίθετα ένα μη προσωρινό polymorph μπορεί να οριστεί μόνο με βάση ένα μη προσωρινό context. Το attribute ctxt_schema_name περιέχει το όνομα του context schema και το attribute rel_table_name περιέχει το όνομα του table που έχει αποθηκευτεί στη βάση και περιέχει τα δεδομένα για το εκάστοτε context schema. Τα δύο αυτά τελευταία attributes έχουν οριστεί ως UNIQUE (το κάθε ένα από αυτά), καθώς δεν έχει νόημα να υπάρχουν δύο contexts με το ίδιο όνομα και επίσης ο database server εμποδίζει την ύπαρξη δύο tables με το ίδιο όνομα. Οι τιμές για το attribute context_schema αν πρόκειται για μη προσωρινό context, ορίζονται τελικά από το χρήστη, ενώ αν πρόκειται για προσωρινό context, είναι της μορφής *TMP*_<ctxt_schema_id>. Τέλος, έχουμε ορίσει τις τιμές για το attribute rel_table_name να διαμορφώνονται ως εξής: για τα μη προσωρινά contexts είναι της μορφής *context*_<ctxt_schema_id> και για τα προσωρινά contexts είναι της μορφής *tmp_ctxt*_<ctxt_schema_id>.



Σχήμα 3.5. Screenshot των πινάκων ctlg_context_schema και ctlg_context_attributes που αποτελούν το κομμάτι καταλόγου που αφορά το context

Όσον αφορά τον πίνακα ctlg_context_attributes, το σχήμα του είναι το ακόλουθο `(ctxt_attributes_id [primary key], ctxt_schema_id [integer], attr_name [varchar(50)], attr_type(50)])`

Το attribute ctxt_attributes_id είναι το πρωτεύον κλειδί (primary key) για τον πίνακα ctlg_context_attributes, προκειμένου το σύστημα να μπορεί να αναφερθεί σε κάθε context attribute. Το ctxt_schema_id είναι ένα foreign key στο ctxt_schema_id του πίνακα ctlg_context_schema. Μέσω αυτού του attribute ορίζονται τα attributes που ανήκουν σε κάποιο context schema. Το attribute attr_name περιέχει το όνομα του context attribute και το attribute attr_type περιέχει τον τύπο του context attribute. Οι τύποι για τα context attributes (αλλά και για τα morph attributes, τα οποία περιγράφουμε παρακάτω) που έχουν υλοποιηθεί στο σύστημα είναι integer ή varchar. Επομένως, έχει ορισθεί για το table ctlg_context_attributes constraint, σύμφωνα με το οποίο το attr_type μπορεί να πάρει τιμές INT/VARCHAR. Επίσης, έχει ορισθεί η ένωση των attributes ctxt_schema_id και attr_name, δηλαδή το (ctxt_schema_id, attr_name) ως UNIQUE, καθώς δεν μπορεί ένα context schema να περιέχει δύο attributes με το ίδιο όνομα. Τέλος, αξίζει να αναφέρουμε ότι με τη δημιουργία ενός καινούριου context, δημιουργείται ένας πίνακας με όνομα αυτό του rel_table_name του πίνακα ctlg_context_schema και σε αυτόν τον πίνακα προστίθεται ως primary key το attribute _tid (tuple id), το οποίο χρησιμοποιείται ως αναφορά για κάθε context instance σε αυτό το context. Ένα screenshot των πινάκων ctlg_context_schema και ctlg_context_attributes φαίνεται στο Σχήμα 3.5.

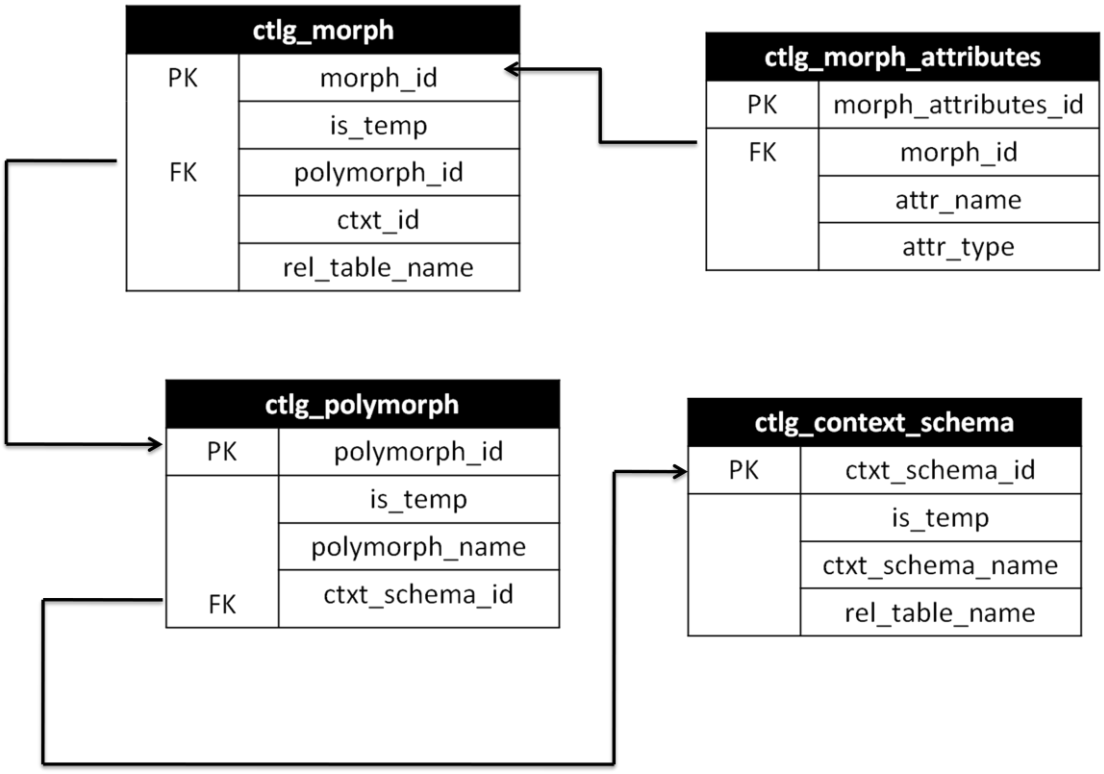
Polymorphs/Morphs

Το τμήμα του καταλόγου που διαχειρίζεται τα metadata που αφορούν τα polymorphs αποτελείται από τους πίνακες ctlg_polymorph, ctlg_morph και ctlg_morph_attributes. Ο πρώτος πίνακας διατηρεί πληροφορίες για τα μετα-δεδομένα του polymorph, ο δεύτερος πληροφορίες για τα μετα-δεδομένα των morphs και ο τρίτος πληροφορίες για τα μετα-δεδομένα των morph attributes. Το σχήμα της σχέσης ctlg_polymorph για την υλοποίηση A (όπου κάθε morph αντιστοιχεί σε έναν πίνακα στη βάση) είναι το ακόλουθο

`(polymorph_id [primary key], is_temp [boolean], polymorph_name [varchar(50)], ctxt_schema_id [integer])`

Το attribute polymorph_id είναι το πρωτεύον κλειδί (primary key) για τον πίνακα ctlg_polymorph, προκειμένου το σύστημα να μπορεί να αναφερθεί σε κάθε polymorph. Το is_temp είναι ένα boolean attribute που δείχνει αν το εν λόγω polymorph είναι προσωρινό ή όχι. Όπως είπαμε νωρίτερα, ένα προσωρινό polymorph μπορεί να οριστεί με βάση είτε ένα προσωρινό είτε ένα μη προσωρινό context, ενώ αντίθετα ένα μη προσωρινό polymorph μπορεί να οριστεί μόνο με βάση ένα μη προσωρινό context. Το attribute polymorph_name

αναφέρεται στο όνομα του εκάστοτε polymorph και το attribute ctxt_schema_id είναι ένα foreign key στο attribute ctxt_schema_id του πίνακα ctlg_context_schema, έτσι ώστε να διακρίνεται με βάση ποιο context schema έχει οριστεί το εκάστοτε polymorph. Οι τιμές για το attribute polymorph_name αν πρόκειται για μη προσωρινό polymorph, ορίζονται τελικά από το χρήστη, ενώ αν πρόκειται για προσωρινό polymorph, είναι της μορφής *TMP*_<ctxt_schema_id>.Το μόνο constraint που ορίζεται για τον πίνακα ctlg_polymorph είναι το uniqueness για το attribute polymorph_name, καθώς προφανώς σε κάθε polymorph πρέπει να αντιστοιχεί ένα μοναδικό όνομα. Ο πίνακας ctlg_morph έχει το ακόλουθο σχήμα `(morph_id [primary key], is_temp [boolean], polymorph_id [integer], ctxt_id [integer], rel_table_name [varchar(50)])`



Σχήμα 3.6. Πίνακες καταλόγου που αφορούν τα polymorphs/morphs

Το attribute morph_id είναι το πρωτεύον κλειδί (primary key) για τον πίνακα ctlg_morph, προκειμένου το σύστημα να μπορεί να αναφερθεί σε κάθε morph. Το is_temp είναι ένα boolean attribute που δείχνει αν το εν λόγω morph είναι προσωρινό ή όχι. Τα temporary morphs μπορούν να ανήκουν μόνο σε temporary polymorphs και αντίστροφα τα temporary polymorphs μπορούν να περιέχουν μόνο temporary morphs. Το ίδιο ισχύει και για τα μη προσωρινά polymorphs και τα μη προσωρινά morphs, δηλαδή τα μη προσωρινά morphs μπορούν να ανήκουν μόνο σε μη προσωρινά polymorphs και αντίστροφα τα μη προσωρινά

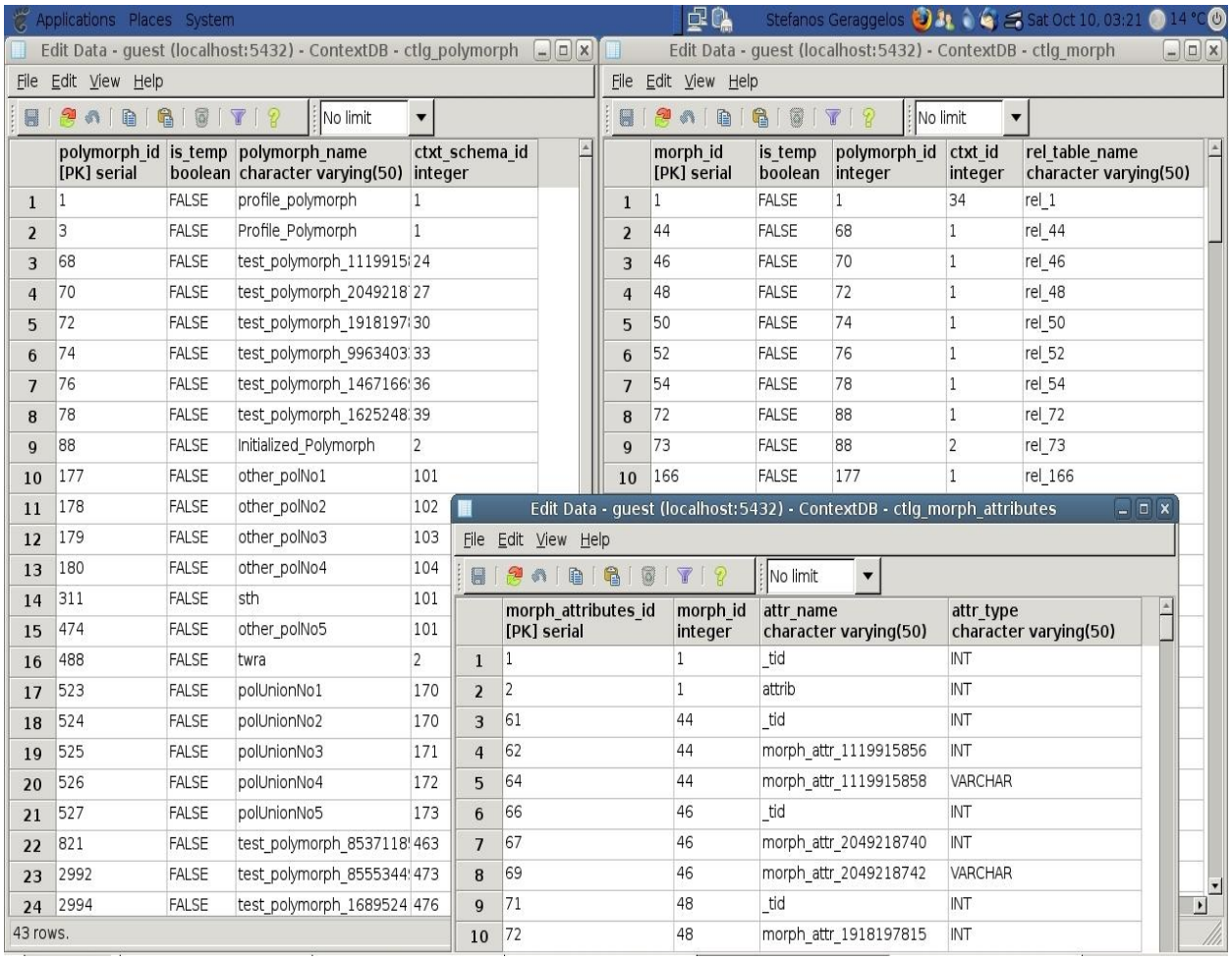
polymorphs μπορούν να περιέχουν μόνο μη προσωρινά morphs. Το polymorph_id είναι ένα foreign key στο attribute polymorph_id του πίνακα ctlg_polymorph, προκειμένου να καθίσταται φανερό ποια morphs ανήκουν σε κάθε polymorph. Το attribute ctxt_id αναφέρεται στο attribute _tid κάθε πίνακα context (δηλαδή των πινάκων *context_*<ctxt_schema_id>-*tmp_ctxt_*<ctxt_schema_id>). Δεν είναι φυσικά δυνατό να ορίσουμε ρητά το ctxt_id ως foreign key σε κάθε _tid attribute κάθε πίνακα που δημιουργείται για το context, γιατί κάτι τέτοιο θα απαιτούσε ένα δυναμικό ορισμό του ctxt_id ως foreign key για όλα τα attributes _tid, ακόμα και σε πίνακες που δεν έχουν ακόμα δημιουργηθεί! Αυτό το πρόβλημα αντιμετωπίστηκε προσομοιώνοντας τη λειτουργία του foreign key για το ctxt_id μέσω κώδικα. Το attribute rel_table_name περιέχει το όνομα του table που έχει αποθηκευτεί στη βάση και περιέχει τα δεδομένα για το εκάστοτε morph. Η ένωση των attributes polymorph_id και ctxt_id έχει οριστεί ως UNIQUE, καθώς σύμφωνα με το μοντέλο μας δεν μπορεί ένα polymorph να περιέχει δύο morphs ορισμένα με βάση το ίδιο context instance. Το attribute rel_table_name έχει, επίσης, οριστεί ως UNIQUE, καθώς ο database server εμποδίζει την ύπαρξη δύο tables με το ίδιο όνομα. Τέλος, έχουμε ορίσει τις τιμές για το attribute rel_table_name να διαμορφώνονται ως εξής: για τα μη προσωρινά morphs *rel_*<morph_id> και για τα προσωρινά contexts *tmp_rel_*<morph_id>.

Όσον αφορά, τέλος, τον πίνακα ctlg_morph_attributes, αυτός διαδραματίζει τον ίδιο ρόλο για τα morphs με αυτόν που παίζει ο ctlg_context_attribute για τα contexts. Συγκεκριμένα, ο ctlg_morph_attributes έχει το ακόλουθο σχήμα

```
(morph_attributes_id [primary key], morph_id [integer], attr_name [varchar(50)], attr_type(50)])
```

Το attribute morph_attributes_id είναι το πρωτεύον κλειδί (primary key) για τον πίνακα ctlg_morph_attributes, προκειμένου το σύστημα να μπορεί να αναφερθεί σε κάθε morph attribute. Το morph_id είναι ένα foreign key στο morph_id του πίνακα ctlg_polymorph. Μέσω αυτού του attribute ορίζονται τα attributes που ανήκουν σε κάποιο morph. Το attribute attr_name περιέχει το όνομα του morph attribute και το attribute attr_type περιέχει τον τύπο του morph attribute. Οι τύποι για τα morph attributes που έχουν υλοποιηθεί στο σύστημα είναι integer ή varchar. Επομένως, έχει ορισθεί για το table ctlg_ morph_attributes constraint, σύμφωνα με το οποίο το attr_type μπορεί να πάρει τιμές INT/VARCHAR. Επίσης, έχει ορισθεί η ένωση των attributes morph_id και attr_name, δηλαδή το (morph_id, attr_name) ως UNIQUE, καθώς δεν μπορεί ένα morph να περιέχει δύο attributes με το ίδιο όνομα. Τέλος, αξίζει να αναφέρουμε ότι με τη δημιουργία ενός καινούριου morph, δημιουργείται ένας πίνακας με όνομα αυτό του rel_table_name του πίνακα ctlg_ morph και σε αυτόν τον πίνακα προστίθεται ως primary key το attribute _tid (tuple id), κατ' αντιστοιχία με το _tid των

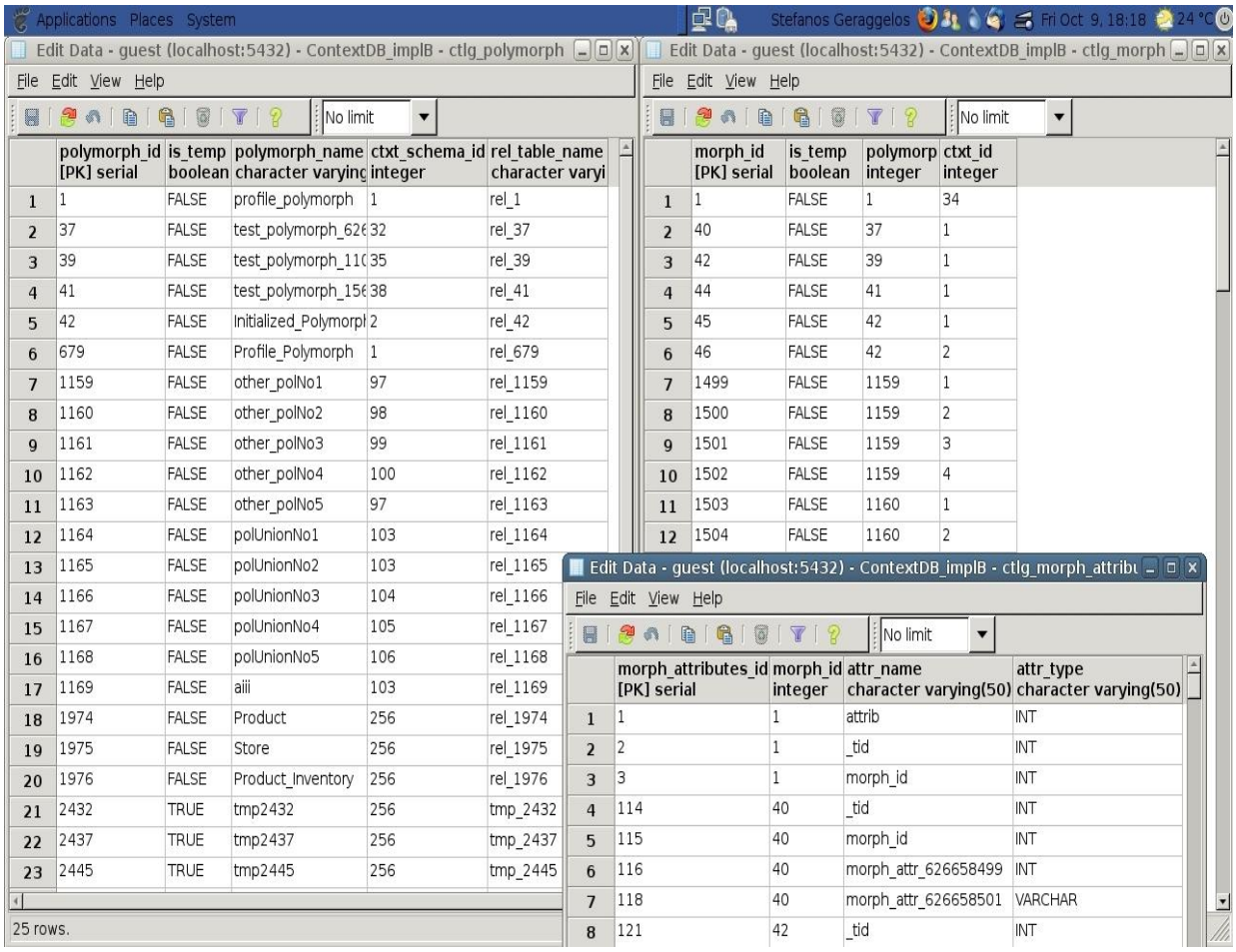
context πινάκων. Ένα screenshot των πινάκων ctlg_polymorph, ctlg_morph και ctlg_ morph_attributes για την υλοποίηση Α φαίνεται στο Σχήμα 3.7.



Σχήμα 3.7. Screenshot των πινάκων ctlg_polymorph, ctlg_morph και ctlg_morph_attributes, που αποτελούν το κομμάτι καταλόγου που αφορά τα polymorphs/morphs, για την υλοποίηση Α

Η παραπάνω ανάλυση για το τμήμα του καταλόγου που χειρίζεται τα metadata των polymorphs/morphs αφορούν στην υλοποίηση Α. Επειδή, όπως, αναφέραμε παραπάνω, σε κάθε υλοποίηση αλλάζει ο τρόπος αποθήκευσης των polymorphs/morphs στη βάση δεδομένων, έπεται ότι διαφοροποιείται και το τμήμα του καταλόγου που χειρίζεται τα metadata αυτών. Οι διαφοροποιήσεις αυτές στον κατάλογο είναι ελάχιστες και η βασική δομή του παραμένει σταθερή. Θα περιγράψουμε στο σημείο αυτό τις αλλαγές αυτές που συμβαίνουν για την υλοποίηση Β στον Κατάλογο. Καθώς στην υλοποίηση Β, κάθε polymorph πλέον αντιστοιχεί σε έναν πίνακα στη βάση δεδομένων, η κύρια διαφοροποίηση για αυτήν την υλοποίηση έγκειται στο γεγονός ότι το attribute rel_table_name, που στην

υλοποίηση Α ανήκει στο σχήμα του πίνακα ctlg_morph, τώρα μεταφέρεται στο σχήμα του πίνακα ctlg_polymorph. Έτσι, πλέον, κατά τη δημιουργία ενός polymorph δημιουργείται, επίσης, αυτόματα ένας πίνακας (με όνομα *rel*_<polymorph_id> ή *tmp_rel*_<polymorph_id>) στη βάση με δύο attributes, τα _tid και morph_id, που είναι (το morph_id) foreign key στο attribute morph_id του πίνακα ctlg_morph (βλ. περιγραφή για την υλοποίηση Β). Αντίστοιχα, το attribute rel_table_name έχει, επίσης, οριστεί ως UNIQUE στον πίνακα ctlg_polymorph, καθώς ο database server εμποδίζει την ύπαρξη δύο tables με το ίδιο όνομα. Ένα screenshot των πινάκων ctlg_polymorph, ctlg_morph και ctlg_ morph_attributes για την υλοποίηση Β φαίνεται στο Σχήμα 3.8.



Σχήμα 3.8. Screenshot των πινάκων ctlg_polymorph, ctlg_morph και ctlg_morph_attributes, που αποτελούν το κομμάτι καταλόγου που αφορά τα polymorphs/morphs, για την υλοποίηση Β

3.2.3 Υποσύστημα Διαχείρισης Δεδομένων (Data)

Η μονάδα διαχείρισης των Δεδομένων (Data) είναι υπεύθυνη για τη διαχείριση των εισαγωγών/διαγραφών/ενημερώσεων (inserts/deletes/updates) των δεδομένων στους πίνακες που έχουν δημιουργηθεί στη βάση δεδομένων. Αυτά τα δεδομένα αναφέρονται τόσο στο καθαρό context όσο και στις relations που εξαρτώνται από αυτό, δηλαδή τις context-aware relations (ή polymorphs). Έτσι γίνεται φανερή μία πρώτη διάκριση μεταξύ του τμήματος της συγκεκριμένης μονάδας που χειρίζεται τα δεδομένα για το context και του τμήματος που χειρίζεται τα δεδομένα για τα polymorphs/morphs.

Κατά τη σχεδίαση της μονάδας των Δεδομένων προέκυψαν κάποια ζητήματα, για τα οποία έπρεπε να πάρουμε αντίστοιχες αποφάσεις. Τα ζητήματα αυτά απορρέουν κυρίως από την αλλαγή δεδομένων για το context. Ένα από αυτά εμφανίζεται στην περίπτωση κατά την οποία επιχειρείται η διαγραφή μίας tuple από έναν context πίνακα, με βάση την οποία (tuple) είχε οριστεί κάποιο morph. Κάτι τέτοιο πρέπει να εμποδίζεται από το σύστημα, γεγονός το οποίο γίνεται εγείροντας την κατάλληλη εξαίρεση (exception). Επίσης, ένα άλλο ζήτημα που απαιτεί ειδική μεταχείριση είναι όταν μετά την ενημέρωση κάποιων context τιμών, προκύψουν δύο ή περισσότερες tuples με τις ίδιες τιμές για όλα τα context attributes. Κάτι τέτοιο, ούτως ή άλλως απαγορεύεται, καθώς με τη δημιουργία κάθε context πίνακα ορίζεται η ένωση όλων των attributes αυτού ως UNIQUE, με αποτέλεσμα να μην είναι η δυνατή η αποθήκευση 2 εγγραφών με τις ίδιες τιμές (διπλοεγγραφές). Όμως, για να μην υπάρξει σε μία τέτοια περίπτωση διακοπή της εκτέλεσης κάποιας διεργασίας, πριν φτάσουμε στην απαγόρευση στο επίπεδο του database server, αντιμετωπίζουμε το πρόβλημα με τη συγχώνευση αυτών των tuples με τις κοινές τιμές (προτού φυσικά δημιουργηθούν) σε μία tuple και το id αυτής ορίζεται ως ctxt_id για όλα εκείνα τα morphs που είχαν οριστεί με βάση κάποιες από τις context tuples που συγχωνεύθηκαν.

Τελικά, τα modules του Καταλόγου και των Δεδομένων συνολικά χρησιμοποιούνται στις περιπτώσεις που ο χρήστης εισάγει, αντί για κάποιο context-aware ερώτημα, κάποια DDL εντολή δημιουργίας ή κάποια DML εντολή εισαγωγής, διαγραφής ή ενημέρωσης. Σε αυτές τις περιπτώσεις δε γίνεται χρήση των Τελεστών, αλλά προσπελαύνονται απευθείας οι μονάδες Καταλόγου και Δεδομένων.

3.2.4 Υποσύστημα Εκτέλεσης Τελεστών (Operators)

Η μονάδα των Τελεστών είναι υπεύθυνη για την εκτέλεση κάθε context-aware τελεστή ξεχωριστά. Σε αυτό το σημείο θα περιγράψουμε κάθε context-aware τελεστή που υλοποιήσαμε, καθώς επίσης και τα προβλήματα που προέκυψαν κατά τη σχεδίαση κάθε τελεστή και τους τρόπους που αυτά αντιμετωπίστηκαν. Όπως αναφέραμε στην ενότητα 2.4,

κάθε context-aware τελεστής μπορεί να είναι είτε unary είτε binary, όπου για τον μεν πρώτο το input είναι ένα polymorph, ενώ για το δεύτερο είναι δύο polymorphs, και το αποτέλεσμα είναι πάντα ένα temporary polymorph. Τα inputs αυτά ισχύουν για όλους τους context-aware τελεστές (ανάλογα με το αν είναι unary ή binary), αλλά εκτός από αυτά υπάρχουν και άλλες είσοδοι ανάλογα με τον τελεστή, οι οποίες περιγράφονται παρακάτω.

Τελεστής Project

Ο τελεστής project είναι ένας unary τελεστής και λαμβάνει ως είσοδο ένα polymorph και μία λίστα από attributes, τα οποία θα προβληθούν. Κατά την εκτέλεσή του πραγματοποιείται ένα project σε εκείνα τα morphs τα οποία περιλαμβάνουν στο σχήμα τους όλα τα attributes που ορίστηκαν στην είσοδο. Αν έστω και ένα από αυτά τα attributes δεν περιλαμβάνεται σε κάποιο από τα morphs του polymorph που ορίστηκε, τότε αυτό το morph δε συμπεριλαμβάνεται στο τελικό παραγόμενο προσωρινό polymorph. Τέλος, στην περίπτωση που το σύνολο των attributes που ορίστηκαν δεν περιλαμβάνεται σε κανένα από τα morphs του δοθέντος polymorph, τότε δημιουργείται απλά ένα temporary polymorph, το οποίο είναι κενό, δηλαδή δεν περιέχει κανένα morph. Θα μπορούσαμε στην περίπτωση αυτή να θεωρούμε το ερώτημα άκυρο και κάποιο error να εγείρεται, όμως προτιμήσαμε την προαναφερθείσα στρατηγική στην προσπάθειά μας να εξαλείψουμε, όπου είναι δυνατό, τα exceptions που διακόπτουν τη ροή μίας διεργασίας. Η απόφασή μας αυτή ακολουθείται στα περισσότερα από τα αντίστοιχα ζητήματα που προκύπτουν για τη σχεδίαση κάθε τελεστή.

Τα παραπάνω αποτελούν τη βασική λειτουργία του τελεστή project. Επιπλέον, έχει υλοποιηθεί και μία επέκταση αυτού του τελεστή, η οποία περιλαμβάνει άλλες τρεις λειτουργίες. Μία από αυτές είναι μαζί με την προβολή κάποιων attributes παράλληλα και η μετονομασία (rename) αυτών για το παραγόμενο προσωρινό polymorph. Έτσι, ορίζεται και άλλη μία λίστα με τα νέα ονόματα των attributes. Ο τελεστής rename έχει υλοποιηθεί και αυτόνομα εκτός του τελεστή project.

Μία άλλη λειτουργία είναι η προβολή κάποιων από τα context attributes (του context schema με βάση το οποίο έχει οριστεί το συγκεκριμένο polymorph) σε κάθε ένα από τα παραγόμενα morphs μαζί με την προβολή των morph attributes, δηλαδή προβάλλονται τα context attributes ως σχήμα του morph. Οι τιμές για αυτά τα context attributes είναι οι τιμές με βάση τις οποίες έχει οριστεί το εκάστοτε polymorph.

Η τελευταία λειτουργία αυτής της extended έκδοσης του project είναι η μετονομασία (rename) αυτών των προβληθέντων context attributes, εφόσον ενεργοποιηθεί η προηγούμενη λειτουργία, αυτή του project κάποιων από τα context attributes.

Τελεστής Select

Ο τελεστής select είναι ένας unary τελεστής, ο οποίος λαμβάνει ως είσοδο ένα polymorph και μία συνθήκη για το WHERE κομμάτι του ερωτήματος καθώς και μία λίστα από τα attributes εκείνα που περιέχονται στη συνθήκη αυτή. Κατά την εκτέλεσή του πραγματοποιείται ένα select σε κάθε ένα από τα morphs τα οποία περιλαμβάνουν στο σχήμα τους όλα τα attributes που ορίστηκαν στην είσοδο. Ομοίως με τον τελεστή project, αν έστω και ένα από αυτά τα attributes δεν περιλαμβάνεται σε κάποιο από τα morphs του polymorph που ορίστηκε, τότε αυτό το morph δε συμπεριλαμβάνεται στο τελικό παραγόμενο προσωρινό polymorph. Έτσι, στην περίπτωση που το σύνολο των attributes που ορίστηκαν δεν περιλαμβάνεται σε κανένα από τα morphs του δοθέντος polymorph, τότε δημιουργείται απλά ένα temporary polymorph, το οποίο είναι κενό, δηλαδή δεν περιέχει κανένα morph.

Τελεστής Cartesian Product

Ο τελεστής cartesian product είναι ένας binary τελεστής και λαμβάνει ως είσοδο αποκλειστικά δύο polymorphs. Τα δύο αυτά polymorphs πρέπει να έχουν οριστεί με βάση το ίδιο context schema, αλλιώς το ερώτημα δεν είναι έγκυρο. Σε αυτό το σημείο πρέπει να αναφέρουμε ότι το context-aware μοντέλο επιτρέπει γενικά την εκτέλεση binary τελεστών για polymorphs ορισμένα με βάση τυπικά διαφορετικά context schemas, αλλά στην ουσία τους ίδια. Αυτό συμβαίνει στις περιπτώσεις που τα ονόματα των context attributes για δύο context schemas είναι διαφορετικά, αλλά το πλήθος τους και οι τύποι τους είναι τα ίδια, οπότε στην ουσία πρόκειται για ισοδύναμα context schemas. Επιλέξαμε να μην επιτρέψουμε κάτι τέτοιο για κανέναν από τους binary τελεστές, καθώς στην περίπτωση αυτή ο έλεγχος των context schemas και των attributes αυτών επιβαρύνει αρκετά την εκτέλεση του binary τελεστή, καθιστώντας το κόστος εκτέλεσης του αρκετά μεγάλο. Για αυτό το πρόβλημα, υπάρχει και μία ενδιάμεση λύση, σύμφωνα με την οποία το overhead για τον παραπάνω έλεγχο μεταφέρεται στη φάση δημιουργίας/διαγραφής ενός context schema, οπότε αν βρεθούν δύο ή περισσότερα context schemas στην ουσία τους ίδια, τότε καθορίζεται με κάποιο τρόπο ότι τα polymorphs που έχουν οριστεί με βάση αυτά είναι συμβατά για εκτέλεση binary τελεστών. Δεν επιλέξαμε αυτήν τη στρατηγική για να μην επιβαρύνουμε την εκτέλεση των DDL/DML εντολών. Στην πράξη δε χάνουμε κάποια λειτουργικότητα από την επιλογή μας αυτή.

Κατά την εκτέλεσή, λοιπόν, του συγκεκριμένου τελεστή πραγματοποιείται ένα καρτεσιανό γινόμενο σε κάθε ζευγάρι από morphs, καθένα από τα οποία ανήκει σε κάθε ένα από τα δύο polymorphs που ορίστηκαν ως είσοδο, με την προϋπόθεση ότι τα δύο αυτά morphs έχουν οριστεί με βάση το ίδιο context instance, δηλαδή έχουν στον κατάλογο την ίδια τιμή για το ctxt_id. Αν υπάρχουν κάποια morphs στο ένα polymorph που δεν έχουν αντίστοιχα morphs στο άλλο polymorph, δηλαδή δεν υπάρχει morph στο άλλο polymorph που να έχει οριστεί με

βάση το ίδιο context instance με το αντίστοιχο morph του πρώτου polymorph, τότε αυτά τα morphs δε συμπεριλαμβάνονται στο τελικό παραγόμενο προσωρινό polymorph.

Ένα πρακτικό ζήτημα που προκύπτει εδώ, είναι για την περίπτωση εκείνη που κάποιο ζευγάρι από morphs έχουν κοινά attributes. Έτσι, στο προσωρινό παραγόμενο morph, που αποτελεί το καρτεσιανό γινόμενο αυτού του ζεύγους, δεν μπορούν να αποθηκευθούν δύο attributes με το ίδιο όνομα. Στην περίπτωση αυτή γίνεται μία μετονομασία των κοινών attributes και αυτά αποθηκεύονται με την εξής μορφή <*polymorph_name*>.<*attribute_name*>. Αν, τέλος, τα δύο polymorphs έχουν το ίδιο όνομα, δηλαδή έχει οριστεί ένα καρτεσιανό γινόμενο ενός polymorph με τον εαυτό του (self-join), τότε τα κοινά attributes (δηλαδή όλα τα attributes) αποθηκεύονται με την εξής μορφή *polymorph1.*<*attribute_name*> για τα attributes του πρώτου polymorph και *polymorph2.*<*attribute_name*> για τα attributes του δεύτερου polymorph.

Τελεστής Union

Ο τελεστής union είναι ένας binary τελεστής και λαμβάνει ως είσοδο αποκλειστικά δύο polymorphs. Τα δύο αυτά polymorphs πρέπει να έχουν οριστεί με βάση το ίδιο context schema, αλλιώς το ερώτημα δεν είναι έγκυρο. Ο union είναι ένας από τους τρεις τελεστές συνόλων (set operators). Κατά την εκτέλεσή του πραγματοποιείται μία ένωση σε κάθε ζευγάρι από morphs, καθένα από τα οποία ανήκει σε κάθε ένα από τα δύο polymorphs που ορίστηκαν ως είσοδο, με την προϋπόθεση ότι τα δύο αυτά morphs έχουν οριστεί με βάση το ίδιο context instance, δηλαδή έχουν στον κατάλογο την ίδια τιμή για το ctxt_id, και επίσης είναι union-compatible. Σε αντίθεση με το καρτεσιανό γινόμενο, στο παραγόμενο προσωρινό polymorph για τον union τελεστή συμπεριλαμβάνεται και τα morphs εκείνα του ενός polymorph που δεν έχουν αντίστοιχα morphs στο άλλο polymorph, δηλαδή δεν υπάρχει morph στο άλλο polymorph που να έχει οριστεί με βάση το ίδιο context instance με το αντίστοιχο morph του πρώτου polymorph. Για αυτά τα morphs γίνεται μία ένωση του κάθε morph με το κενό σύνολο, δηλαδή το αποτέλεσμα είναι το ίδιο το morph.

Τελεστής Intersection

Ο τελεστής intersection είναι ένας binary τελεστής και λαμβάνει ως είσοδο αποκλειστικά δύο polymorphs. Τα δύο αυτά polymorphs πρέπει να έχουν οριστεί με βάση το ίδιο context schema, αλλιώς το ερώτημα δεν είναι έγκυρο. Ο intersection είναι ο δεύτερος από τους τρεις τελεστές συνόλων (set operators). Κατά την εκτέλεσή του πραγματοποιείται μία τομή σε κάθε ζευγάρι από morphs, καθένα από τα οποία ανήκει σε κάθε ένα από τα δύο polymorphs που ορίστηκαν ως είσοδο, με την προϋπόθεση ότι τα δύο αυτά morphs έχουν οριστεί με βάση το ίδιο context instance, δηλαδή έχουν στον κατάλογο την ίδια τιμή για το ctxt_id, και επίσης

είναι union-compatible. Σε αντίθεση με την ένωση, στο παραγόμενο προσωρινό polymorph για τον intersection τελεστή δε συμπεριλαμβάνονται τα morphs κάποιου από τα polymorphs που δεν έχουν αντίστοιχα morphs στο άλλο polymorph, δηλαδή δεν υπάρχει morph στο άλλο polymorph που να έχει οριστεί με βάση το ίδιο context instance με το αντίστοιχο morph του πρώτου polymorph. Αυτό διότι για αυτά τα morphs η τομή με το κενό σύνολο παράγει ως αποτέλεσμα το κενό σύνολο.

Τελεστής Difference

Ο τελεστής difference είναι ένας binary τελεστής και λαμβάνει ως είσοδο αποκλειστικά δύο polymorphs. Τα δύο αυτά polymorphs πρέπει να έχουν οριστεί με βάση το ίδιο context schema, αλλιώς το ερώτημα δεν είναι έγκυρο. Ο difference είναι ο τελευταίος από τους τρεις τελεστές συνόλων (set operators). Κατά την εκτέλεσή του πραγματοποιείται μία διαφορά σε κάθε ζευγάρι από morphs, καθένα από τα οποία ανήκει σε κάθε ένα από τα δύο polymorphs που ορίστηκαν ως είσοδο, με την προϋπόθεση ότι τα δύο αυτά morphs έχουν οριστεί με βάση το ίδιο context instance, δηλαδή έχουν στον κατάλογο την ίδια τιμή για το ctxt_id, και επίσης είναι union-compatible. Η διαφορά πραγματοποιείται έχοντας ως "αριστερό" (στην πράξη της διαφοράς (-)) το πρώτο polymorph και ως "δεξιό" το δεύτερο polymorph. Όσον αφορά στα morphs εκείνα, τα οποία δεν έχουν αντίστοιχα morphs στο άλλο polymorph, λαμβάνονται υπόψη μόνο τα morphs του πρώτου polymorph, ενώ δε συμπεριλαμβάνονται τα αντίστοιχα του δεύτερου polymorph.

Τελεστής Select Schema

Ο τελεστής select schema είναι ένας unary τελεστής και λαμβάνει ως είσοδο ένα polymorph και μία συνθήκη για το WHERE κομμάτι του ερωτήματος που περιέχει context attributes. Το αποτέλεσμα του τελεστή είναι μόνο τα morphs ενός polymorph που ικανοποιούνται από την προηγούμενη συνθήκη. Κατά την εκτέλεση αυτού του τελεστή πραγματοποιείται αρχικά ένα select στον context πίνακα εκείνο, με βάση το οποίο έχει οριστεί το polymorph της εισόδου. Αυτό έχει ως αποτέλεσμα κάποια context instances με τη μορφή μίας λίστας από ctxt_id's (ή από _tid's). Τέλος, τα morphs του polymorph που έχουν οριστεί με βάση αυτά τα ctxt_id's είναι αυτά που συμπεριλαμβάνονται στο τελικό παραγόμενο προσωρινό polymorph.

Τελεστής Rename

Ο τελεστής rename είναι ένας unary τελεστής και λαμβάνει ως είσοδο ένα polymorph και μία λίστα από attributes, τα οποία θα μετονομαστούν, καθώς και μία λίστα από τα νέα ονόματα

για αυτά τα attributes. Το παραγόμενο προσωρινό polymorph περιλαμβάνει το polymorph της εισόδου έχοντας μετονομάσει τα attributes των morphs του με τις νέες ορισθείσες τιμές.

Τελεστής Add Context Attribute

Ο τελεστής add context attribute είναι ένας unary τελεστής και λαμβάνει ως είσοδο ένα polymorph και μία λίστα από τα context attributes, που θα προστεθούν στο context, καθώς και μία λίστα από τις default τιμές για αυτά τα νέα attributes. Έτσι, κατά την εκτέλεση του εν λόγω τελεστή προστίθενται στο context schema του polymorph της εισόδου ,αλλά και στον αντίστοιχο context πίνακα, τα ορισθέντα attributes με τις αντίστοιχες ορισθείσες default τιμές. Εννοείται, βέβαια, ότι τα ονόματα αυτά των attributes δεν πρέπει να χρησιμοποιούνται ήδη στο εν λόγω context schema.

Τελεστής Remove Context Attribute

Ο τελεστής remove context attribute είναι ένας unary τελεστής και λαμβάνει ως είσοδο από ένα polymorph και μία λίστα από τα context attributes, που θα αφαιρεθούν από το context, με βάση το οποίο έχει οριστεί το polymorph της εισόδου. Αν μετά την αφαίρεση αυτών των attributes, προκύπτουν morphs με τα ίδια contexts instances, τότε πραγματοποιείται μία συγχώνευση (merge) αυτών των morphs σε ένα. Αυτό παράγει ένα προσωρινό temporary context schema. Επίσης, για τα morphs που έχουν ορισθεί με βάση αυτά τα merged contexts πραγματοποιείται μία ένωση, εφόσον όλα είναι union-compatible, και το παραχθέν morph ορίζεται στον κατάλογο με βάση το αποτέλεσμα των merged contexts. Αν δεν είναι union-compatible, τότε αυτά δε συμπεριλαμβάνονται στο αποτέλεσμα. Έτσι, το αποτέλεσμα της εκτέλεσης αυτού του τελεστή είναι ένα προσωρινό polymorph και ένα προσωρινό context schema, με βάση το οποίο ορίζεται το προαναφερθέν προσωρινό polymorph.

Τελεστής Map Context

Ο τελεστής map context είναι ένας unary τελεστής και λαμβάνει ως είσοδο εκτός από ένα polymorph, επίσης μία λίστα από τα context attributes, οι τιμές των οποίων θα διαφοροποιηθούν, καθώς και μία λίστα από εκφράσεις (expressions), το αποτέλεσμα κάθε μίας από τις οποίες θα τεθεί ως τιμή για τα αντίστοιχα attributes. Ομοίως με τον τελεστή remove context attribute, αν μετά την τροποποίηση των τιμών για αυτά τα attributes, προκύπτουν morphs με τα ίδια contexts instances, τότε πραγματοποιείται μία συγχώνευση (merge) αυτών των morphs σε ένα. Επίσης, για τα morphs που έχουν ορισθεί με βάση αυτά τα merged contexts πραγματοποιείται μία ένωση, εφόσον όλα είναι union-compatible, και το παραχθέν morph ορίζεται στον κατάλογο με βάση το αποτέλεσμα των merged contexts. Αν

δεν είναι union-compatible, τότε αυτά δε συμπεριλαμβάνονται στο αποτέλεσμα. Έτσι, το αποτέλεσμα της εκτέλεσης και αυτού του τελεστή, όπως και του remove context attribute, είναι ένα προσωρινό polymorph και ένα προσωρινό context schema, με βάση το οποίο ορίζεται το προαναφερθέν προσωρινό polymorph.

3.2.5 Υποσύστημα Εκτέλεσης Πλάνων (Query Executor)

Η ανώτερη μονάδα του συστήματος αναλαμβάνει την εκτέλεση του δοθέντος πλάνου εκτέλεσης, αφού προηγουμένως ελέγξει την εγκυρότητά του. Το πλάνο εκτελείται με την εκτέλεση κάθε τελεστή που το αποτελεί. Για κάθε τελεστή που εκτελείται, παράγεται ένα αποτέλεσμα και αποθηκεύεται ως ένα προσωρινό (temporary) polymorph και η προσωρινή αυτή σχέση προωθείται στον επόμενο τελεστή προς εκτέλεση. Πρέπει να αναφέρουμε ότι η στρατηγική αυτή, δηλαδή η αποθήκευση κάθε αποτελέσματος ως ένα temporary polymorph και επίσης αμέσως μετά την εκτέλεση του πλάνου η διαγραφή όλων των παραχθέντων temporary polymorphs εκτός από την τελευταία (η οποία είναι και το τελικό αποτέλεσμα), δεν είναι η πιο αποδοτική. Πράγματι, η πρόσβαση και η εγγραφή δεδομένων στο δίσκο μετά την εκτέλεση κάθε τελεστή προσδίδει ένα μεγάλο overhead στην εκτέλεση των ερωτημάτων και εν γένει σε ολόκληρη τη λειτουργία του συστήματος. Για να αντιληφθούμε τον όγκο των δεδομένων που αποθηκεύονται, πρέπει να λάβουμε υπόψη ότι το εκάστοτε temporary polymorph, που παράγεται ως αποτέλεσμα, μπορεί γενικά να αποτελείται από αρκετές επιμέρους relations, ακριβώς διότι για κάθε context instance υπάρχει μία relation στο τρέχον polymorph. Θα μπορούσαμε, λοιπόν, να υλοποιήσουμε τους τελεστές χωρίς καμία εγγραφή στο δίσκο, διατηρώντας απλά τα αποτελέσματα στη μνήμη και προωθώντας τα "on the fly" προς τα πάνω. Το βασικό πλεονέκτημα, όμως, αυτής της στρατηγικής, είναι ότι με αυτόν τον τρόπο διευκολύνεται αρκετά η υλοποίηση.

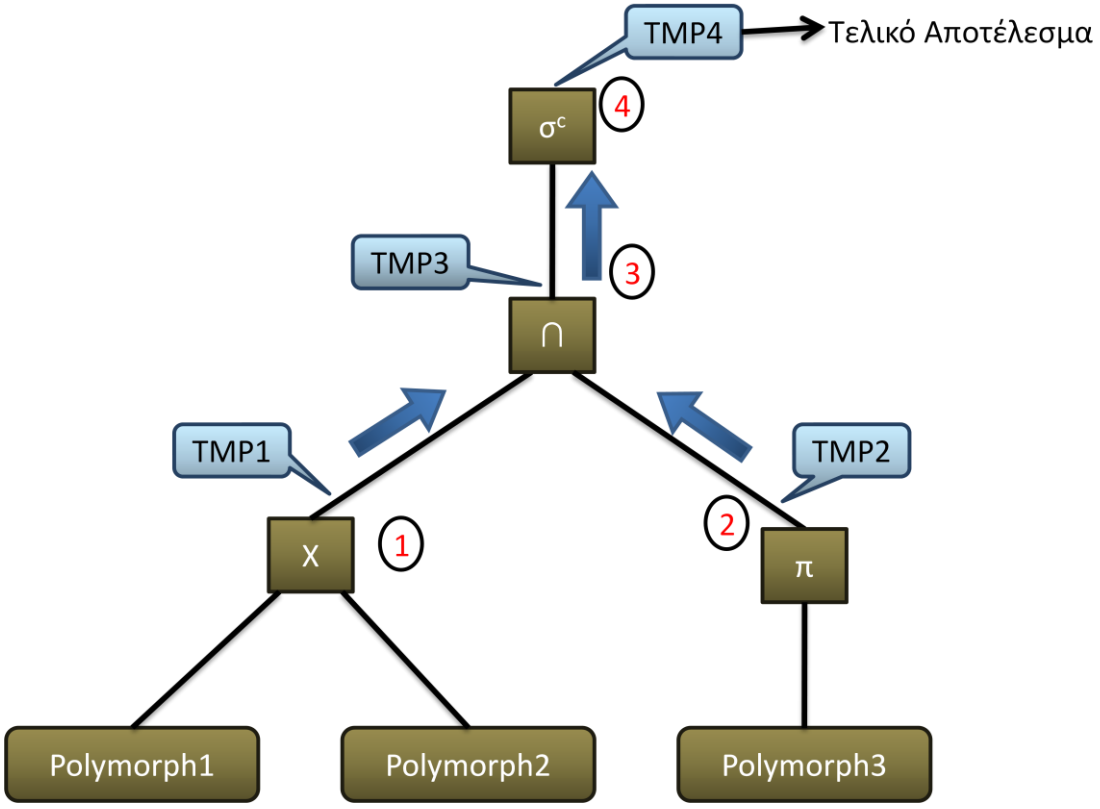
Σε αυτό το σημείο χρειάστηκε να μοντελοποιήσουμε με κάποιο τρόπο το πλάνο εκτέλεσης. Κατασκευάσαμε για το σκοπό αυτό μία δενδρική δομή που αντιστοιχεί στο πλάνο εκτέλεσης που δίνεται. Σε αυτή τη δομή μπορούν να εισαχθούν τριών ειδών κόμβοι, οι unary κόμβοι, οι binary κόμβοι και οι polymorph κόμβοι. Όπως είναι φανερό από την ονομασία τους ένας unary κόμβος αντιστοιχεί σε έναν unary τελεστή, ένας binary κόμβος αντιστοιχεί σε έναν binary τελεστή και ένας polymorph κόμβος αντιστοιχεί σε ένα polymorph. Για να είναι, λοιπόν, έγκυρο ένα τέτοιο πλάνο, πρέπει το αντίστοιχο δένδρο να πληροί ταυτόχρονα όλες τις ακόλουθες προϋποθέσεις:

a. να υπάρχει μία μόνο ρίζα στο δέντρο,
b. κάθε unary κόμβος να έχει ακριβώς ένα "παιδί",
c. κάθε binary κόμβος να έχει ακριβώς δύο "παιδιά"

d. κάθε polymorph κόμβος να μην έχει "παιδιά".
e. κάθε φύλλο του δέντρου να είναι ένας polymorph κόμβος (το οποίο διασφαλίζεται από το συνδυασμό των b,c,d.

Αν δεν ισχύει κάτι από τα παραπάνω για ένα πλάνο εκτέλεσης, τότε αυτό δεν είναι έγκυρο για το σύστημα.

Για να εκτελεσθεί λοιπόν ένα πλάνο εκτέλεσης, εκτελείται αναδρομικά ξεκινώντας από τη ρίζα της δενδρικής δομής. Αυτό έχει ως αποτέλεσμα να διασχίζεται όλο το δέντρο μέχρι τα φύλλα του, δηλαδή τους polymorph κόμβους, και να εκτελείται έτσι από κάτω προς τα πάνω το πλάνο. Όπως έχουμε αναφέρει νωρίτερα, αφού εκτελεστεί ο κατώτερος κόμβος, δημιουργείται ως αποτέλεσμα ένα temporary polymorph, το οποίο προωθείται στον ακριβώς από πάνω κόμβο ως input πλέον για εκτέλεση κοκ, πραγματοποιώντας έτσι ένα pipelining των temporary polymorphs στους τελεστές μέχρι τη ρίζα του δέντρου. Αφού ολοκληρωθεί η εκτέλεση του πλάνου, διαγράφοντα όλα τα ενδιάμεσα προσωρινά polymorphs, εκτός από το τελευταίο, που αποτελεί το τελικό αποτέλεσμα του ερωτήματος.



Σχήμα 3.9. Διαδικασία εκτέλεσης πλάνου

Στο Σχήμα 3.9 παρουσιάζεται σχηματικά η διαδικασία εκτέλεσης ενός πλάνου. Στο συγκεκριμένο σχήμα αρχικά πραγματοποιείται ένα καρτεσιανό γινόμενο μεταξύ των polymorphs Polymorph1 και Polymorph2 και παράγεται ως αποτέλεσμα ένα προσωρινό

polymorph, το TMP1. Στη συνέχεια, πραγματοποιείται ένα project στο polymorph Polymorph3 και παράγεται ως αποτέλεσμα ένα προσωρινό polymorph, το TMP2. Το επόμενο βήμα είναι η τομή μεταξύ των polymorphs TMP1 και TMP2 και η δημιουργία του TMP3 ως αποτέλεσμα. Τέλος, στο polymorph TMP3 εκτελείται ένα select schema και το τελικό αποτέλεσμα είναι το προσωρινό polymorph TMP4.

3.3 Τρόποι αποθήκευσης στο σχεσιακό ΣΔΒΔ

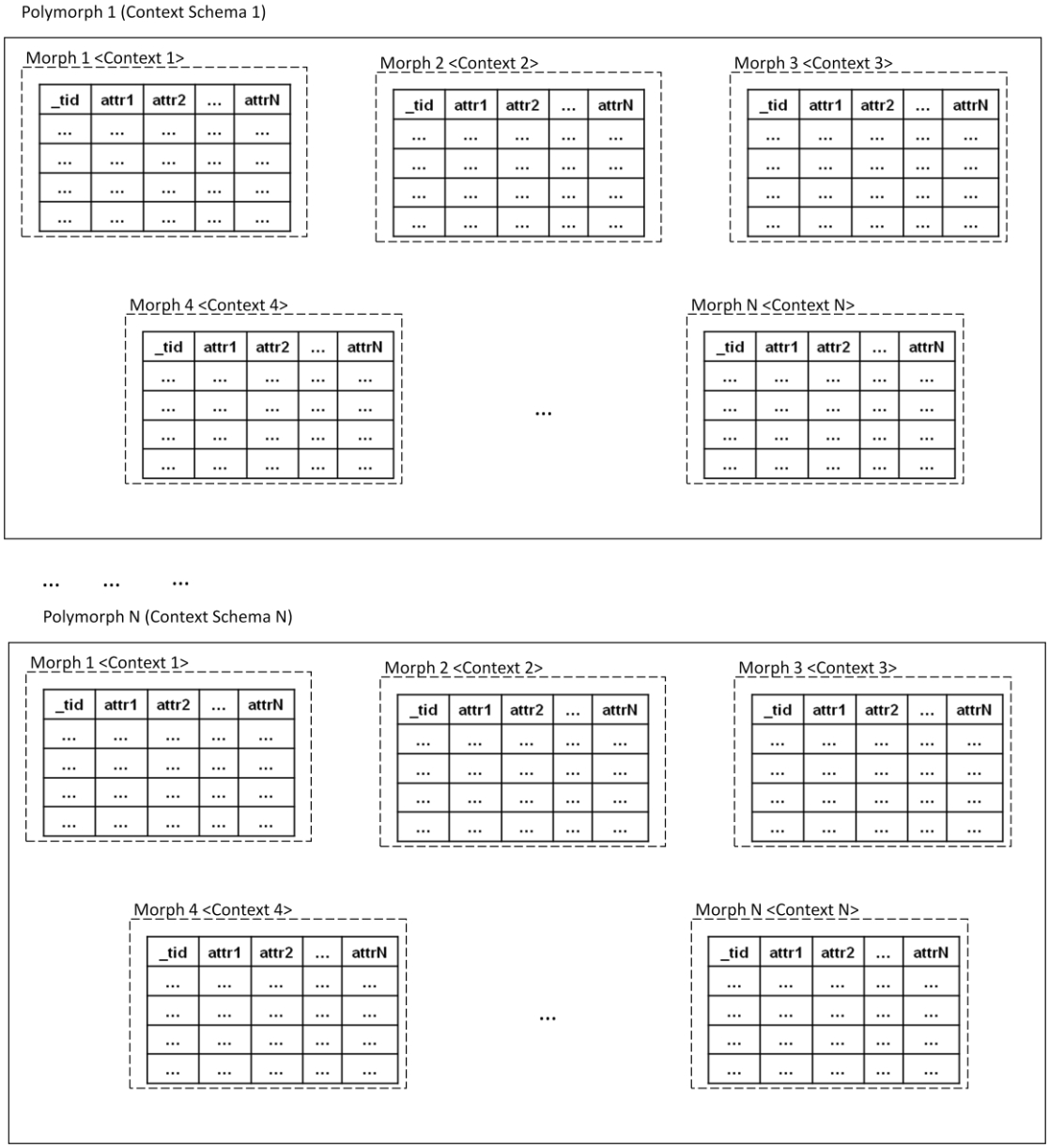
Στο σημείο αυτό θα αναφερθούμε στις δύο υλοποιήσεις που έγιναν και αφορούν την αποθήκευση των polymorphs στο σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων καθώς και στην τρίτη, η οποία ανήκει στις σχεδιαζόμενες μελλοντικές επεκτάσεις του συστήματος, οι οποίες περιγράφονται λεπτομερώς στο Κεφάλαιο 6. Πρέπει σε αυτό το σημείο να γίνει διάκριση μεταξύ της αποθήκευσης του context καθεαυτού και της αποθήκευση των polymorphs (polymorphs). Η αποθήκευση του context παραμένει η ίδια για όλες τις υλοποιήσεις. Αναφέρουμε σε αυτό το σημείο ότι το μοντέλο που χρησιμοποιήσαμε για την αναπαράσταση του context είναι το επίπεδο (flat) μοντέλο κλειδιού-τιμής (key-value), το οποίο περιγράφηκε προηγουμένως στην ενότητα 2.4.

Η διαδικασία της αποθήκευσης συνίσταται ουσιαστικά στην δημιουργία μιας relation στο σχεσιακό ΣΔΒΔ για κάθε context. Κάθε tuple αυτής της relation αποτελεί ένα context instance για το τρέχον context. Αυτό που αποτελεί το κρίσιμο ζήτημα για την ανάγκη δημιουργίας διαφορετικών υλοποιήσεων, προκειμένου να μελετηθεί η αποδοτικότητα καθεμίας για κάθε τύπο ερωτημάτων, είναι η αποθήκευση των polymorphs. Η αποθήκευση των polymorphs είναι η παράμετρος που αλλάζει με την εφαρμογή αυτών των διαφορετικών υλοποιήσεων που αναλύονται παρακάτω. Τα μετα-δεδομένα (metadata) που αφορούν τόσο το context όσο και το polymorphs, επίσης αναλύονται στην περιγραφή του Καταλόγου (catalogue) του συστήματος.

3.3.1 Υλοποίηση Α

Όπως αναφέραμε και στην ενότητα 2.4, όπου αναλύσαμε το μοντέλο πάνω στο οποίο βασίστηκε η διπλωματική αυτή, στο context-aware σύστημα που περιγράφουμε κάθε polymorph αποτελείται από ένα σύνολο από morphs, καθένα από τα οποία ορίζεται με βάση κάποιο context instance. Στην πρώτη αυτή βασική υλοποίηση του συστήματος κάθε morph αποθηκεύεται στο σχεσιακό ΣΔΒΔ ως ένας ξεχωριστός πίνακας. Έτσι, ένα ερώτημα πάνω σε ένα polymorph κάνει προσπέλαση σε τόσους πίνακες, στο επίπεδο του κλασικού σχεσιακού μοντέλου, όσα είναι τα morphs που αποτελούν το συγκεκριμένο polymorph.

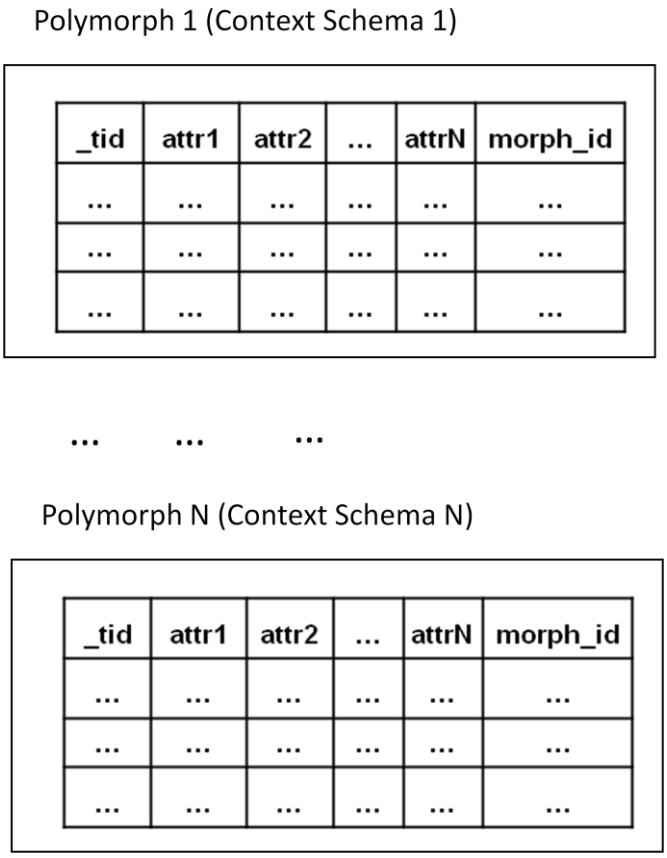
Έτσι, αυτή η υλοποίηση αναμένεται να επιβαρύνει την απόδοση του συστήματος σε περίπτωση που έχουμε polymorphs, τα οποία αποτελούνται από πολλά morphs. Σχηματικά, η υλοποίηση Α για την αποθήκευση των polymorphs στο υπάρχον σχεσιακό ΣΔΒΔ, παρουσιάζεται στο Σχήμα 3.10.



Σχήμα 3.10. Σχηματική αναπαράσταση της αποθήκευσης των polymorphs στο σχεσιακό ΣΔΒΔ για την υλοποίηση Α

3.3.2 Υλοποίηση Β

Σε αυτήν την υλοποίηση ακολουθήσαμε μία εντελώς διαφορετική στρατηγική για την αποθήκευση των polymorphs στο σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων, προκειμένου κατά το testing των υλοποιήσεων να γίνει εμφανές σε ποιες περιπτώσεις ενδείκνυται η κάθε υλοποίηση. Έτσι, στην υλοποίηση Β κάθε polymorph αντιστοιχεί μόνο σε έναν πίνακα του σχεσιακού ΣΔΒΔ. Το σχήμα αυτού του πίνακα περιέχει όλα τα attributes που εμφανίζονται σε όλα τα morphs του polymorph. Εννοείται ότι αν ένα attribute υπάρχει σε περισσότερα του ενός morphs για κάποιο polymorph, τότε αυτό το attribute θα υπάρχει μόνο μία φορά στον πίνακα που δημιουργείται για αυτό το polymorph στο σχεσιακό ΣΔΒΔ.



Σχήμα 3.11. Σχηματική αναπαράσταση της αποθήκευσης των polymorphs στο σχεσιακό ΣΔΒΔ για την υλοποίηση Β

Επίσης, το σχήμα του πίνακα περιλαμβάνει και ένα attribute με το όνομα morph_id, το οποίο είναι foreign key σε πίνακα του καταλόγου (βλ. υποενότητα 3.2.1). Ο πρώτος πίνακας θα περιέχει τιμές για όλα αυτά τα attributes ανάλογα με το morph. Αν ένα morph δεν περιέχει κάποιο από τα attributes που υπάρχουν στον πίνακα του σχεσιακού ΣΔΒΔ, τότε στην αντίστοιχη tuple του πίνακα αυτού (με το αντίστοιχο morph_id) η τιμή για το συγκεκριμένο attribute θα είναι NULL. Έτσι, σύμφωνα με αυτήν την υλοποίηση προκύπτουν πολύ μεγάλες

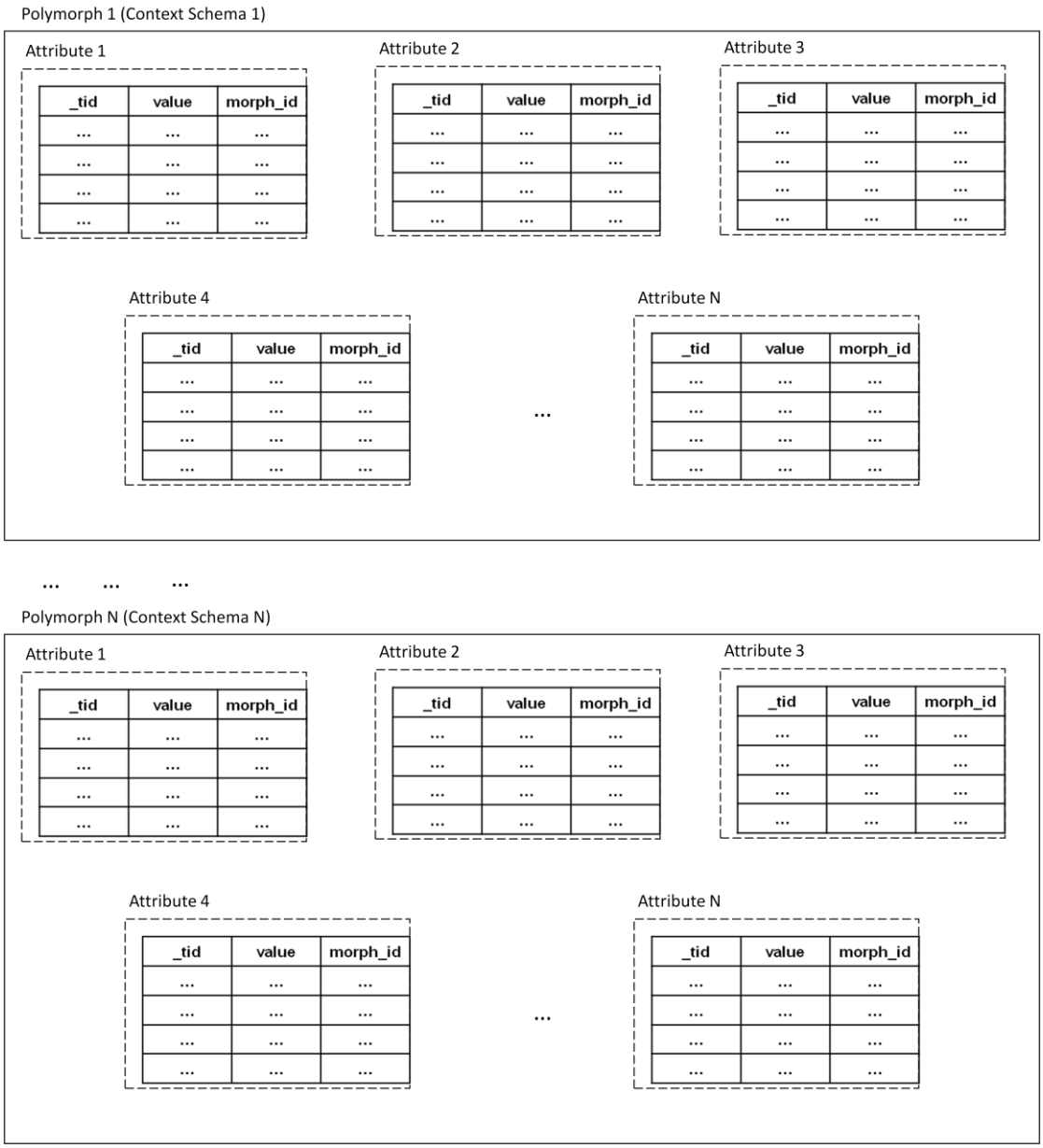
σχέσεις, τόσο σε πλήθος attributes όσο και σε αριθμό από tuples, για κάθε polymorph και πολύ αραιές γενικά. Ένα context-aware ερώτημα, λοιπόν, πάνω σε ένα polymorph κάνει προσπέλαση σε ένα μόνο πίνακα, στο επίπεδο του κλασικού σχεσιακού μοντέλου. Συνεπώς, μερικού τύπου ερωτήματα, όπως πχ project, select, στην υλοποίηση Β αποτελούν ουσιαστικά ένα VIEW πάνω σε ένα (πιθανώς μεγάλο) πίνακα, χωρίς, όμως, να εμπλέκονται άλλες relations, και έτσι αναμένονται αυτά τα ερωτήματα να είναι πιο γρήγορα κατά την εκτέλεσή τους. Σχηματικά, η υλοποίηση Β για την αποθήκευση των polymorphs στο σχεσιακό ΣΔΒΔ παρουσιάζεται στο Σχήμα 3.11.

3.3.3 Υλοποίηση Γ

Η τρίτη αυτή υλοποίηση αποτελεί μία πιο "κάθετη" παραλλαγή της υλοποίησης Α. Συγκεκριμένα, για κάθε polymorph κάθε attribute αποθηκεύεται πάνω στο τρέχον σχεσιακό ΣΔΒΔ ως ένας ξεχωριστός πίνακας. Έτσι, στην υλοποίηση Γ, ένα ερώτημα πάνω σε ένα polymorph κάνει προσπέλαση σε τόσους πίνακες, στο επίπεδο του κλασικού σχεσιακού μοντέλου, όσα είναι τα attributes που υπάρχουν στα morphs του εν λόγω polymorph. Ομοίως με την υλοποίηση Β, αν ένα attribute υπάρχει σε περισσότερα του ενός morphs για κάποιο polymorph, τότε για αυτά τα attributes θα δημιουργηθεί μία relation στο σχεσιακό ΣΔΒΔ. Το σχήμα αυτής της relation, περιλαμβάνει ένα primary key, το _tid (tuple id), ένα attribute με το όνομα value, που περιέχει την τιμή του attribute για το εκάστοτε morph, το οποίο καθορίζεται από ένα τρίτο attribute, το morph_id, το οποίο είναι foreign key σε πίνακα του καταλόγου (βλ. υποενότητα 3.2.1). Σχηματικά, η υλοποίηση Γ για την αποθήκευση των polymorphs στο σχεσιακό ΣΔΒΔ παρουσιάζεται στο Σχήμα 3.12.

Το βασικό κόστος εκτέλεσης ενός τελεστή, και τελικά ενός ερωτήματος, σε οποιοδήποτε σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων συνίσταται στην εγγραφή των δεδομένων στο δίσκο. Ειδικά για το context-aware σύστημα που μελετάμε, για το οποίο, όπως αναφέραμε και πρωτύτερα, τα επιμέρους αποτελέσματα που προκύπτουν από την εκτέλεση του κάθε τελεστή αποθηκεύονται ως προσωρινές σχέσεις στο δίσκο μέσω του υπάρχοντος ΣΔΒΔ, αλλά και λόγω της ενσωμάτωσης του context σε αυτό, η οποία καθιστά τα δεδομένα που αποθηκεύονται στο σύστημα υπερμεγέθη, η αποθήκευση αυτών στο δίσκο καθίσταται ως μία από τις κρισιμότερες παραμέτρους για τη λειτουργία του συστήματος. Οι προαναφερθείσες υλοποιήσεις πραγματοποιήθηκαν προκειμένου να μελετηθεί η απόδοση του συστήματος, αλλά και η απόκρισή του για διάφορους τύπους context-aware ερωτημάτων και σε διάφορες context-aware βάσεις δεδομένων. Κρίσιμα συμπεράσματα προκύπτουν από την εκτέλεση πειραμάτων και σεναρίων ελέγχου (test cases), που πραγματοποιήθηκαν για τις υλοποιήσεις αυτές. Τα ζητήματα αυτά μελετώνται εκτενέστερα στο Κεφάλαιο 5. Η ιδανική υλοποίηση θα ήταν ένας συνδυασμός αυτών ανάλογα με τους τύπους ερωτημάτων που

χρησιμοποιούνται περισσότερο σε κάθε βάση δεδομένων. Κάτι τέτοιο, άλλωστε, χρησιμοποιείται και στα πραγματικά εμπορικά ΣΔΒΔ.



Σχήμα 3.12. Σχηματική αναπαράσταση της αποθήκευσης των polymorphs στο σχεσιακό ΣΔΒΔ για την υλοποίηση Γ

4

Υλοποίηση

Στο κεφάλαιο αυτό περιγράφουμε την υλοποίηση του συνολικού συστήματος, δηλαδή του context-aware Συστήματος Διαχείρισης Βάσεων Δεδομένων. Πιο συγκεκριμένα στην ενότητα 4.1 γίνεται αναφορά στις πλατφόρμες και στα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν, ενώ στην ενότητα 4.2 προχωράμε σε μία πιο εκτενή περιγραφή των λεπτομερειών υλοποίησης.

4.1 Πλατφόρμες και προγραμματιστικά εργαλεία

Το σύστημα έχει αναπτυχθεί με τη γλώσσα προγραμματισμού ANSI C++ [Str97] και τρέχει σε πλατφόρμα Linux (kernel 2.9.11) και συγκεκριμένα σε διανομή Debian 5.0 [DBN09]. Ο database server που τρέχει στο κατώτερο μέρος του συστήματος είναι PostgreSQL 8.3.8 [PSQ]. Η επιλογή τόσο της γλώσσας προγραμματισμού και της γενικότερης πλατφόρμας όσο και του λειτουργικού συστήματος αλλά και του database server έγινε με κριτήριο την όσο το δυνατό απλή ανάπτυξη του συστήματος αλλά και την αποδοτικότερη λειτουργία του.

Το σύστημα, συμπεριλαμβανομένων των κλάσεων ελέγχου και των σχολίων, εκτείνεται σε περισσότερες από 60.000 γραμμές κώδικα για την κάθε υλοποίηση (A και B). Συνολικά, δηλαδή και για τις δύο υλοποιήσεις ο κώδικας ξεπερνάει τις 120.000 γραμμές κώδικα.

Για την ανάπτυξη χρησιμοποιήθηκε ο μεταγλωττιστής g++ (gcc 4.3.2) [GPP] της GNU, το εργαλείο make και το περιβάλλον προγραμματισμού Eclipse 3.4.2 [ECL]. Επίσης, έχει γίνει

χρήση της βιβλιοθήκης libpqxx 3.0.2 [PQX] στη μονάδα DataBase Layer, προκειμένου να καθίσταται δυνατή η επικοινωνία του συστήματος με τον PostgreSQL server. Τέλος, έχει γίνει εκτενής χρήση της standard βιβλιοθήκης προτύπων της C++ (standard template library STL) [STL94] σε όλο τον κώδικα του συστήματος.

4.2 Λεπτομέρειες υλοποίησης

Στη συγκεκριμένη ενότητα παρουσιάζονται οι λεπτομέρειες υλοποίησης των δομικών μονάδων που περιγράφηκαν στην ενότητα 3.2. Έτσι, αναλύεται η υλοποίηση των ακόλουθων μονάδων: DataBase Layer, Κατάλογος (Catalogue), Δεδομένα (Data), Τελεστές (Operators) και Πλάνο Εκτέλεσης (Query Plan). Επίσης, πριν την περιγραφή της υλοποίησης των προηγούμενων μονάδων, παρουσιάζεται η ιεραρχία εξαιρέσεων (exceptions) που υλοποιήθηκε για το σύστημα, προκειμένου να επιτευχθεί ένας αποτελεσματικός χειρισμός των λαθών που προκύπτουν κατά τη λειτουργία του συστήματος.

Μεγάλο μέρος της περιγραφής που ακολουθεί αποτελείται από την παρουσίαση *διαγραμμάτων κλάσεων* (class diagrams). Σε αυτά τα διαγράμματα, μερικές από τις κλάσεις επαναλαμβάνονται όπου είναι αναγκαίο για να δειχθούν σχέσεις ενθυλάκωσης ή κληρονομικότητας με άλλες κλάσεις. Εκτός από τα διαγράμματα, περιγράφονται, επίσης, πιο αναλυτικά τα βασικά attributes, τα οποία αποτελούν τα private members της κάθε κλάσης και οι βασικές μέθοδοι κάθε κλάσης, οι οποίες (εκτός από λίγες εξαιρέσεις) αποτελούν τα public members κάθε κλάσης.

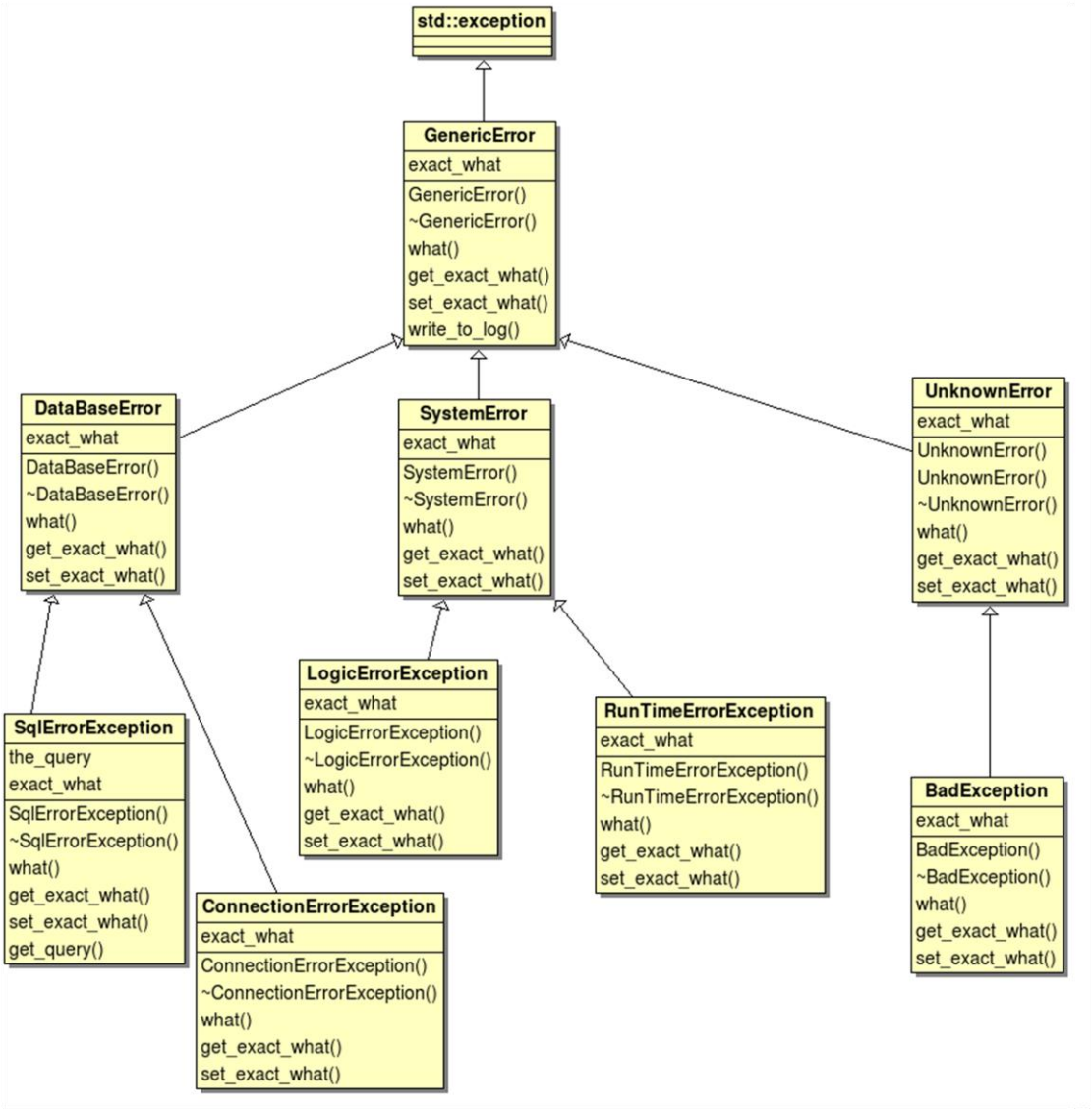
4.2.1 Ιεραρχία Εξαιρέσεων (Exceptions)

Η βασική ιεραρχία των κλάσεων φαίνεται στο Σχήμα 4.1. Οι υπόλοιπες κλάσεις, οι οποίες κληρονομούν από αυτές, παρουσιάζονται παρακάτω. Η βασική κλάση από την οποία κληρονομούν όλες οι υπόλοιπες είναι η `std::exception`, η οποία είναι στις ήδη υπάρχουσες βιβλιοθήκες της C++.

Τα βασικά attributes που χρησιμοποιούνται στις κλάσεις εξαιρέσεων είναι τα ακόλουθα:

- `string exact_what`: Η μεταβλητή exact_what περιέχεται σε όλες τις κλάσεις εξαιρέσεων και περιλαμβάνει το ακριβές μήνυμα του λάθους που ανακύπτει για την κάθε κλάση. Υπάρχει, αντίστοιχα, και το βασικό (πιο γενικό) μήνυμα του λάθους το οποίο επιστρέφεται από τη μέθοδο `what()`, η οποία κληρονομείται από την αντίστοιχη what() της κλάσης std::exception.

- `string the_query`: Η συγκεκριμένη μεταβλητή παρουσιάζεται στην κλάση `SqlErrorException` και σε όλες τις κλάσεις οι όποιες κληρονομούν από αυτήν και περιέχει το ερώτημα (ή DDL/DML εντολή) που προξένησε την εκάστοτε εξαίρεση.



Σχήμα 4.1. Διάγραμμα βασικής ιεραρχίας κλάσεων εξαιρέσεων (exceptions)

Οι βασικές μέθοδοι που υλοποιούνται στις κλάσεις εξαιρέσεων είναι οι ακόλουθες:

- `string what()`: Επιστρέφει το γενικό μήνυμα του λάθους που ανέκυψε και περιέχεται σε όλες τις κλάσεις εξαιρέσεων.

- `string get_exact_what()`: Επιστρέφει το ακριβές μήνυμα του λάθους που ανέκυψε και περιέχεται σε όλες τις κλάσεις εξαιρέσεων. Ουσιαστικά επιστρέφει τη μεταβλητή exact_what.
- `void set_exact_what(string)`: Θέτει το ακριβές μήνυμα λάθους και περιέχεται σε όλες τις κλάσεις εξαιρέσεων. Ουσιαστικά θέτει τιμή για τη μεταβλητή exact_what.
- `string get_query()`: Επιστρέφει το ερώτημα (ή DDL/DML εντολή) που προξένησε την εκάστοτε εξαίρεση και περιέχεται στην κλάση `SqlErrorException` και σε όλες τις κλάσεις οι όποιες κληρονομούν από αυτήν. Ουσιαστικά θέτει τιμή για τη μεταβλητή the_query.

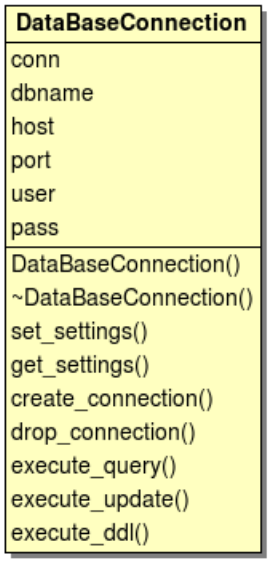
Όλες οι κλάσεις εξαιρέσεων είναι οι ακόλουθες:

```
class GenericError: public exception
class DataBaseError: public GenericError
class SystemError: public GenericError
class UnknownError: public GenericError
class SqlErrorException: public DataBaseError
class ConnectionErrorException: public DataBaseError
class LogicErrorException: public SystemError
class RunTimeErrorException: public SystemError
class BadException: public UnknownError
class TooManyConnectionsException: public ConnectionErrorException
class InsufficientResourcesException: public SqlErrorException
class SyntaxErrorException: public SqlErrorException
class InvalidSqlStatementNameException: public SqlErrorException
class InsufficientPrivilegeException: public SqlErrorException
class InvalidCursorNameException: public SqlErrorException
class InvalidCursorStateException: public SqlErrorException
class IntegrityConstraintViolationException: public SqlErrorException
class DataException: public SqlErrorException
class FeatureNotSupportedException: public SqlErrorException
class DiskFullException: public InsufficientResourcesException
class OutOfMemoryException: public InsufficientResourcesException
class UndefinedTableException: public SyntaxErrorException
class UndefinedFunctionException: public SyntaxErrorException
```

```
class UndefinedColumnException: public SyntaxErrorException
class DomainErrorException: public LogicErrorException
class InvalidArgumentException: public LogicErrorException
class LengthErrorException: public LogicErrorException
class OutOfRangeException: public LogicErrorException
class InternalErrorException: public RunTimeErrorException
class InDoubtErrorException: public RunTimeErrorException
class OverflowErrorException: public RunTimeErrorException
class RangeErrorException: public RunTimeErrorException
class UnderflowErrorException: public RunTimeErrorException
class BadAllocException: public RunTimeErrorException
class BadCastException: public RunTimeErrorException
class BadTypeIdException: public RunTimeErrorException
class IosFailureException: public RunTimeErrorException
class AttributeException: public LogicErrorException
class AttributeNotDefinedException: public AttributeException
class AttributeNotExistsException: public AttributeException
class InvalidPairOfPolymorphException: public LogicErrorException
class DifferentContextSchemaForPairOfPolymorphsException: public
InvalidPairOfPolymorphException
class MorphsAreNotUnionCompatibleException: public
InvalidPairOfPolymorphException
class InvalidQueryPlanException: public LogicErrorException
class InvalidNodeException: public InvalidQueryPlanException
class OperatorNodeException: public InvalidNodeException
class UnaryNodeException: public OperatorNodeException
class BinaryNodeException: public OperatorNodeException
class PolymorphNodeException: public InvalidNodeException
class EmptyQueryPlanException: public LogicErrorException
class InvalidQueryTreeOperationException: public LogicErrorException
class NoPreviousNodeException: public
InvalidQueryTreeOperationException
class NoNextLeftNodeException: public
InvalidQueryTreeOperationException
class NoNextRightNodeException: public
InvalidQueryTreeOperationException
```

4.2.2 Υποσύστημα Αποθήκευσης και Διεπαφής με το Σχεσιακό ΣΔΒΔ (DataBase Layer)

Η κλάση που υλοποιεί τη μονάδα DataBase Layer είναι η DataBaseConnection. Το διάγραμμα κλάσης της DataBaseConnection παρουσιάζεται στο σχήμα 4.2.



Σχήμα 4.2. Διάγραμμα κλάσης DataBaseConnection

Τα attributes της κλάσης DataBaseConnection είναι τα ακόλουθα:

- `connection* conn`: Δείκτης σε ένα αντικείμενο τύπου connection της βιβλιοθήκης libpqxx. Ουσιαστικά μέσω αυτής της μεταβλητής επιτυγχάνεται η επικοινωνία με τη βάση δεδομένων.
- `string dbname`: Περιέχει το όνομα της βάσης δεδομένων στην οποία θα γίνει η σύνδεση.
- `string host`: Περιέχει το όνομα του host του database server.
- `string port`: Περιέχει το port του database server.
- `string user`: Περιέχει το όνομα του χρήστη.
- `string pass`: Περιέχει τον κωδικό του χρήστη.

Οι βασικές μέθοδοι που υλοποιούνται στην κλάση DataBaseConnection είναι οι ακόλουθες:

- `void create_connection()`: Πραγματοποιεί μία σύνδεση σε μία βάση δεδομένων, σύμφωνα με τις ρυθμίσεις που είναι αποθηκευμένες στα προαναφερθέντα attributes.
- `void drop_connection()`: Κλείνει τη σύνδεση με τη βάση δεδομένων.
- `result execute_query(string)`: Εκτελεί το ερώτημα που περιγράφεται στην παράμετρο της μεθόδου και επιστρέφει το αποτέλεσμα.
- `string execute_update(string)`: Εκτελεί την ενημέρωση που περιγράφεται στην παράμετρο της μεθόδου και επιστρέφει τον αριθμό των tuples που διαφοροποιήθηκαν.
- `void execute_ddl(string)`: Εκτελεί το DDL/DML που περιγράφεται στην παράμετρο της μεθόδου.

4.2.3 Υποσύστημα του Καταλόγου (Catalogue)

Οι κλάσεις που υλοποιούν τη μονάδα του καταλόγου είναι η ContextSchema, η οποία διαχειρίζεται τα metadata που αφορούν στο context και η ContextRelationSchema, η οποία διαχειρίζεται τα metadata που αφορούν στα polymorphs. Το διάγραμμα των συγκεκριμένων κλάσεων παρουσιάζεται στο σχήμα 4.3.

Τα attributes της κλάσης ContextSchema είναι τα ακόλουθα:

- `id ctxt_schema_id`: Περιέχει το id του εκάστοτε context schema για τον κατάλογο.
- `string ctxt_schema_name`: Περιέχει το όνομα του context schema.
- `attribute attr_name`: Περιέχει το όνομα του εκάστοτε context attribute.
- `string attr_type`: Περιέχει τον τύπο του context attribute (INT/VARC HAR).
- `list<attribute> names`: Η συγκεκριμένη λίστα περιέχει τα ονόματα των context attributes για τη δημιουργία ενός context schema.
- `list<string> types`: Περιέχει τους τύπους των context attributes για τη δημιουργία ενός context schema.
- `DataBaseConnection DBconn`: Μέσω του συγκεκριμένου αντικειμένου πραγματοποιείται η επικοινωνία με τη βάση.

Οι βασικές μέθοδοι που υλοποιούνται στην κλάση ContextSchema είναι οι ακόλουθες:

- `void create_catalogue_tables()`: Αν δεν υπάρχουν οι πίνακες καταλόγου που αφορούν στο context, η συγκεκριμένη μέθοδος τους δημιουργεί.

Σχήμα 4.3. Διάγραμμα κλάσεων ContextSchema και ContextRelationSchema

- `id create_context_schema()`: Δημιουργεί ένα context schema και επιστρέφει το id του context schema αποθηκεύθηκε στον κατάλογο.

- `id create_context_schema_with_attrs()`: Δημιουργεί ένα context schema με ονόματα και attributes που έχουν οριστεί στα προαναφερθέντα private members και επιστρέφει το id του context schema που αποθηκεύθηκε στον κατάλογο.
- `int drop_context_schema()`: Διαγράφει το context schema που έχει οριστεί στο ctxt_schema_id ή στο ctxt_schema_name. Η τιμή που επιστρέφει είναι 1, αν πράγματι διεγράφη το εν λόγω context schema ή 0 σε αντίθετη περίπτωση.
- `id add_context_attribute()`: Προσθέτει το context attribute που έχει οριστεί στο context schema που έχει οριστεί. Το id που επιστρέφεται είναι το id του context attribute που προστέθηκε και αποθηκεύθηκε στον κατάλογο.
- `int remove_context_attribute()`: Διαγράφει το context attribute που έχει οριστεί. Η τιμή που επιστρέφει είναι 1, αν πράγματι διεγράφη το εν λόγω attribute ή 0 σε αντίθετη περίπτωση.
- `id create_temp_context_schema()`: Δημιουργεί ένα προσωρινό context schema και επιστρέφει το id του context schema που αποθηκεύθηκε στον κατάλογο.
- `id create_temp_context_schema_with_attrs()`: Δημιουργεί ένα προσωρινό context schema με ονόματα και attributes που έχουν οριστεί στα προαναφερθέντα private members και επιστρέφει το id του context schema αποθηκεύθηκε στον κατάλογο.
- `int drop_all_temp_context_schemas()`: Διαγράφει όλα τα προσωρινά context schemas και επιστρέφει τον αριθμό που διεγράφησαν.
- `id retrieve_context_schema_by_name()`: Επιστρέφει το id ενός context schema, ορίζοντας το όνομά του.
- `string retrieve_context_schema_by_id()`: Επιστρέφει το όνομα ενός context schema, ορίζοντας το id του.
- `list<attribute> retrieve_context_attributes_by_context_schema_name()`: Επιστρέφει μία λίστα με τα ονόματα των attributes ενός context schema, ορίζοντας το όνομά του.
- `list<attribute> retrieve_context_attributes_by_context_schema_id()`: Επιστρέφει μία λίστα με τα ονόματα των attributes ενός context schema, ορίζοντας το id του.
- `string retrieve_context_attribute_type_by_context_attribute_name()`: Επιστρέφει τον τύπο ενός context attribute, ορίζοντας το όνομά του και το context schema στο οποίο ανήκει.

- `string retrieve_context_schema_table_by_context_schema()`: Επιστρέφει το όνομα του table που έχει αποθηκευθεί το context schema που έχει οριστεί.
- `bool is_temp_ctxt_schema()`: Επιστρέφει αν το context schema που έχει οριστεί είναι προσωρινό ή όχι.

Τα attributes της κλάσης ContextRelationSchema είναι τα ακόλουθα:

- `id polymorph_id`: Περιέχει το id του εκάστοτε polymorph για τον κατάλογο.
- `string polymorph_name`: Περιέχει το όνομα του εκάστοτε polymorph για τον κατάλογο.
- `id morph_id`: Περιέχει το id του εκάστοτε morph για τον κατάλογο.
- `attribute morph_attr_name`: Περιέχει το όνομα του εκάστοτε morph attribute.
- `attribute morph_attr_type`: Περιέχει τον τύπο του εκάστοτε morph attribute (INT/VARCHAR).
- `list<attribute> names`: Η συγκεκριμένη λίστα περιέχει τα ονόματα των morph attributes για τη δημιουργία ενός morph.
- `list<string> types`: Περιέχει τους τύπους των morph attributes για τη δημιουργία ενός morph.
- `id ctxt_schema_id`: Περιέχει το id του εκάστοτε context schema για τον κατάλογο.
- `string ctxt_schema_name`: Περιέχει το όνομα του εκάστοτε context schema για τον κατάλογο.
- `id ctxt_id`: Περιέχει το id του εκάστοτε context για το context schema που έχει οριστεί.
- `ContextSchema CtxtSchema`: Μέσω του συγκεκριμένου αντικειμένου αντλούμε πληροφορίες για κάποιο context schema.
- `DataBaseConnection DBconn`: Μέσω του συγκεκριμένου αντικειμένου πραγματοποιείται η επικοινωνία με τη βάση.

Οι βασικές μέθοδοι που υλοποιούνται στην κλάση ContextRelationSchema είναι οι ακόλουθες:

- `void create_catalogue_tables()`: Αν δεν υπάρχουν οι πίνακες καταλόγου που αφορούν στα polymorphs/morphs, η συγκεκριμένη μέθοδος τους δημιουργεί.

- `id create_polymorph()`: Δημιουργεί ένα polymorph με βάση το context schema που έχει οριστεί και επιστρέφει του id του polymorph που αποθηκεύθηκε στον κατάλογο.
- `int drop_polymorph()`: Διαγράφει το polymorph (και τα αντίστοιχα morphs) που έχει οριστεί στο polymorph_id ή στο polymorph_name. Η τιμή που επιστρέφει είναι 1, αν πράγματι διεγράφη το εν λόγω polymorph ή 0 σε αντίθετη περίπτωση.
- `id add_morph()`: Προσθέτει ένα morph με βάση το context που έχει οριστεί στο polymorph που έχει οριστεί. Το id που επιστρέφεται είναι το id του morph που προστέθηκε και αποθηκεύθηκε στον κατάλογο.
- `id add_morph_with_attrs()`: Προσθέτει στο polymorph που έχει οριστεί ένα morph με βάση το context που έχει οριστεί με ονόματα και attributes που έχουν οριστεί στα προαναφερθέντα private members και επιστρέφει το id του morph αποθηκεύθηκε στον κατάλογο.
- `int remove_morph()`: Διαγράφει το morph που έχει οριστεί. Η τιμή που επιστρέφει είναι 1, αν πράγματι διεγράφη το εν λόγω morph ή 0 σε αντίθετη περίπτωση.
- `id add_morph_attribute()`: Προσθέτει το morph attribute που έχει οριστεί στο morph που έχει οριστεί. Το id που επιστρέφεται είναι το id του morph attribute που προστέθηκε και αποθηκεύθηκε στον κατάλογο.
- `int remove_morph_attribute()`: Διαγράφει το morph attribute που έχει οριστεί. Η τιμή που επιστρέφει είναι 1, αν πράγματι διεγράφη το εν λόγω attribute ή 0 σε αντίθετη περίπτωση.
- `id create_temp_polymorph()`: Δημιουργεί ένα προσωρινό polymorph με βάση το context schema που έχει οριστεί και επιστρέφει το id του polymorph που αποθηκεύθηκε στον κατάλογο.
- `int drop_all_temp_polymorphs()`: Διαγράφει όλα τα προσωρινά polymorphs και επιστρέφει τον αριθμό που διεγράφησαν.
- `id add_temp_morph()`: Προσθέτει ένα προσωρινό morph με βάση το context που έχει οριστεί στο polymorph που έχει οριστεί. Το polymorph που έχει οριστεί πρέπει να είναι, επίσης, προσωρινό. Το id που επιστρέφεται είναι το id του morph που προστέθηκε και αποθηκεύθηκε στον κατάλογο.
- `id add_morph_with_attrs()`: Προσθέτει στο polymorph που έχει οριστεί ένα προσωρινό morph με ονόματα και attributes που έχουν οριστεί στα προαναφερθέντα private members και επιστρέφει του id του morph που αποθηκεύθηκε στον κατάλογο. Το polymorph που έχει οριστεί πρέπει να είναι, επίσης, προσωρινό.

- `int remove_all_temp_morphs()`: Διαγράφει όλα τα προσωρινά morphs και επιστρέφει τον αριθμό που διεγράφησαν.
- `id retrieve_polymorph_by_name()`: Επιστρέφει το id ενός polymorph, ορίζοντας το όνομά του.
- `string retrieve_polymorph_by_id()`: Επιστρέφει το όνομα ενός polymorph, ορίζοντας το id του.
- `list<id> retrieve_polymorph_by_context_schema()`: Επιστρέφει τα id's των polymorphs που έχουν ορίζονται με βάση κάποιο context schema.
- `id retrieve_context_schema_by_polymorph()`: Επιστρέφει το id του context schema, με βάση το οποίο έχει οριστεί κάποιο polymorph.
- `list<id> retrieve_morph_by_polymorph()`: Επιστρέφει μία λίστα με τα morphs που ανήκουν στο polymorph που έχει οριστεί.
- `list<attribute> retrieve_morph_attributes_by_morph_id()`: Επιστρέφει μία λίστα με τα ονόματα των attributes ενός morph, ορίζοντας το id του.
- `string retrieve_morph_attribute_type_by_morph_attribute_name()`: Επιστρέφει τον τύπο ενός morph attribute, ορίζοντας το morph στο οποίο ανήκει.
- `string retrieve_morph_table_by_morph_id()`: Επιστρέφει το όνομα του table που έχει αποθηκευθεί το morph που έχει οριστεί. Για την υλοποίηση B, όπου κάθε polymorph αντιστοιχεί σε ένα table, έχει οριστεί αντίστοιχα η μέθοδος `string retrieve_polymorph_table_by_polymorph_id()`.
- `id retrieve_context_id_by_morph_id()`: Επιστρέφει το id του context, με βάση το οποίο έχει οριστεί κάποιο morph.
- `bool is_temp_polymorph()`: Επιστρέφει αν το polymorph που έχει οριστεί είναι προσωρινό ή όχι.

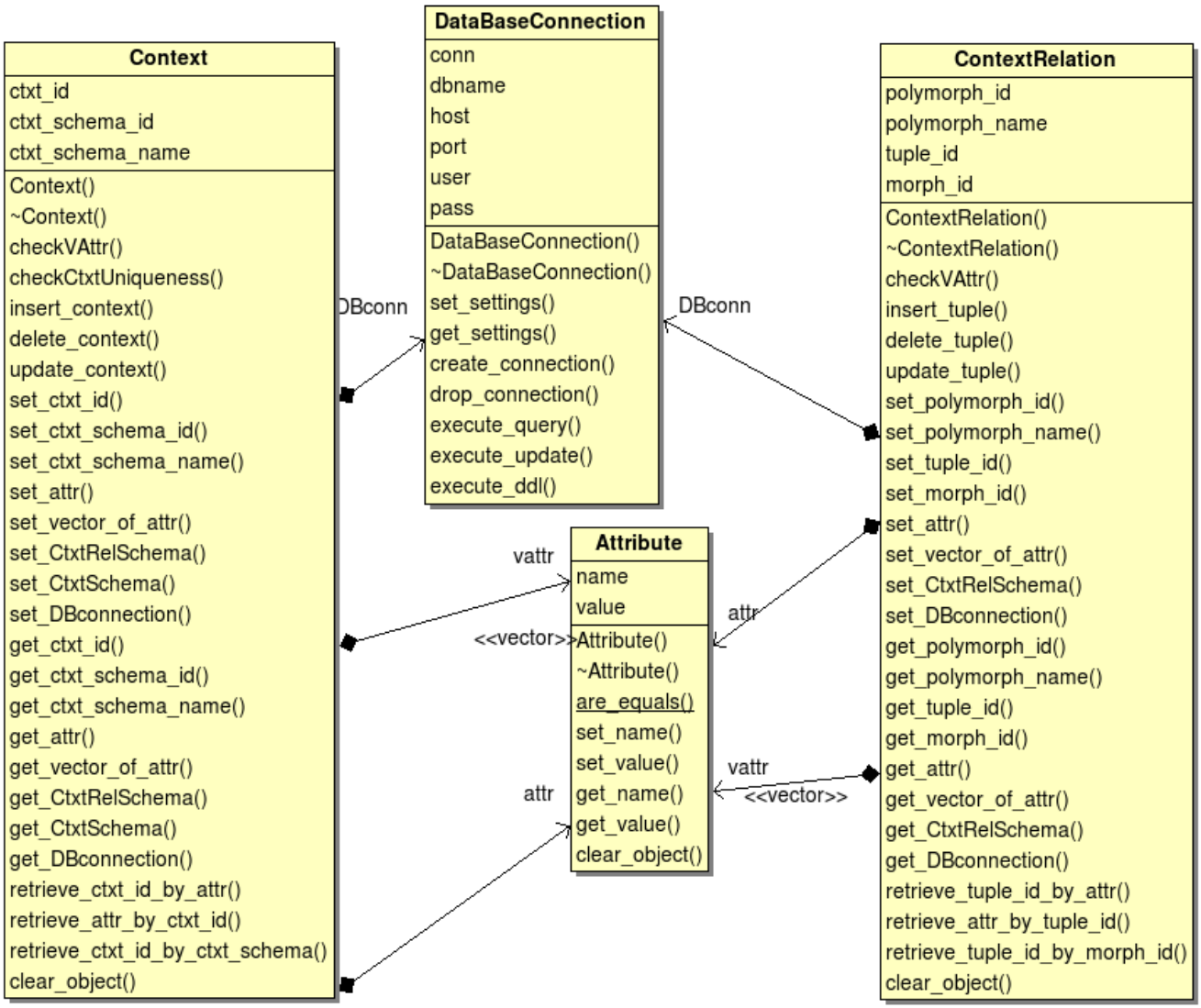
4.2.4 Υποσύστημα Διαχείρισης Δεδομένων (Data)

Οι κλάσεις που υλοποιούν τη μονάδα των δεδομένων είναι η Context, η οποία διαχειρίζεται τα δεδομένα που αφορούν στο context και η ContextRelation, η οποία διαχειρίζεται τα δεδομένα που αφορούν στα polymorphs. Το διάγραμμα των συγκεκριμένων κλάσεων παρουσιάζεται στο σχήμα 4.4.

Τα attributes της κλάσης Context είναι τα ακόλουθα:

- `id ctxt_id`: Περιέχει το id του εκάστοτε context.

- `id ctxt_schema_id`: Περιέχει το id του εκάστοτε context schema για τον κατάλογο.
- `string ctxt_schema_name`: Περιέχει το όνομα του context schema.
- `Attribute attr`: Το αντικείμενο αυτό περιέχει πληροφορίες για κάποιο attribute οι τιμές του οποίου πρόκειται να διαφοροποιηθούν.
- `vector<Attribute> vattr`: Περιέχει πολλά αντικείμενα, για την περίπτωση εκείνη της διαφοροποίησης τιμών για περισσότερα του ενός attribute.
- `DataBaseConnection DBconn`: Μέσω του συγκεκριμένου αντικειμένου πραγματοποιείται η επικοινωνία με τη βάση.



Σχήμα 4.4. Διάγραμμα κλάσεων Context και ContextRelation

Οι βασικές μέθοδοι που υλοποιούνται στην κλάση Context είναι οι ακόλουθες:

- `bool checkVAttr()`: Επιστρέφει true, αν τα attributes που έχουν οριστεί υπάρχουν στον κατάλογο για το συγκεκριμένο context schema και δεν έχει οριστεί κάποιο όνομα attribute περισσότερο από μία φορά.
- `int checkCtxtUniqueness()`: Ελέγχει αν η μετά τις αλλαγές στο context, θα υπάρξουν διπλότυπα και επιστρέφει το id του ήδη υπάρχοντος context. Αν όχι, τότε επιστρέφει μηδέν.
- `id insert_context()`: Εισάγει τιμές για το context και επιστρέφει το id της tuple που αποθηκεύτηκε.
- `int delete_context()`: Διαγράφει κάποιο context, δηλαδή τη συγκεκριμένη tuple που ορίζεται. Αν πράγματι υπήρχε και διεγράφη, επιστρέφει ένα, αλλιώς μηδέν.
- `int update_context()`: Ενημερώνει το context με τις τιμές που ορίστηκαν και επιστρέφει τον αριθμό των tuples που επηρεάστηκαν.
- `list<id> retrieve_ctxt_id_by_attr()`: Επιστρέφει μία λίστα από context id's, τα οποία περιέχουν τις τιμές για τα attributes που έχουν οριστεί.
- `list<Attribute> retrieve_attr_by_ctxt_id()`: Επιστρέφει μία λίστα από αντικείμενα Attribute (ονόματα και τιμές) για το context που ορίστηκε.
- `list<id> retrieve_ctxt_id_by_ctxt_schema()`: Επιστρέφει μία λίστα από context id's, τα οποία ανήκουν στο context schema που έχει οριστεί.

Τα attributes της κλάσης ContextRelation είναι τα ακόλουθα:

- `id polymorph_id`: Περιέχει το id του εκάστοτε `polymorph` για τον κατάλογο.
- `string polymorph _name`: Περιέχει το όνομα του `polymorph`.
- `Attribute attr`: Το αντικείμενο αυτό περιέχει πληροφορίες για κάποιο attribute οι τιμές του οποίου πρόκειται να διαφοροποιηθούν.
- `vector<Attribute> vattr`: Περιέχει πολλά αντικείμενα, για την περίπτωση εκείνη της διαφοροποίησης τιμών για περισσότερα από ένα attribute.
- `id morph_id`: Περιέχει το id του εκάστοτε morph.
- `id tuple_id`: Περιέχει το id της εκάστοτε tuple.
- `DataBaseConnection DBconn`: Μέσω του συγκεκριμένου αντικειμένου πραγματοποιείται η επικοινωνία με τη βάση.

Οι βασικές μέθοδοι που υλοποιούνται στην κλάση ContextRelation είναι οι ακόλουθες:

- `bool checkVAttr()`: Επιστρέφει true, αν τα attributes που έχουν οριστεί υπάρχουν στον κατάλογο για το συγκεκριμένο morph και δεν έχει οριστεί κάποιο όνομα attribute περισσότερο από μία φορά.

- `id insert_tuple()`: Εισάγει τιμές για κάποιο morph και επιστρέφει το id της tuple που αποθηκεύτηκε.
- `int delete_tuple()`: Διαγράφει τη συγκεκριμένη tuple που ορίζεται. Αν πράγματι υπήρχε και διεγράφη, επιστρέφει ένα, αλλιώς μηδέν.
- `int update_tuple()`: Ενημερώνει το morph με τις τιμές που ορίστηκαν και επιστρέφει τον αριθμό των tuples που επηρεάστηκαν.
- `list<id> retrieve_tuple_id_by_attr()`: Επιστρέφει μία λίστα από tuple id's, τα οποία περιέχουν τις τιμές για τα attributes που έχουν οριστεί.
- `list<Attribute> retrieve_attr_by_tuple_id()`: Επιστρέφει μία λίστα από αντικείμενα Attribute (ονόματα και τιμές), για την tuple που ορίστηκε.
- `list<id> retrieve_tuple_id_by_ctxt_schema()`: Επιστρέφει μία λίστα από tuple id's, τα οποία ανήκουν στο morph που έχει οριστεί.

4.2.5 Υποσύστημα Εκτέλεσης Τελεστών (Operators)

Κάθε κλάση που ανήκει σε αυτή τη μονάδα υλοποιεί έναν context-aware τελεστή. Συγκεκριμένα, στο Σχήμα 4.5 παρουσιάζεται το διάγραμμα των κλάσεων ProjectOperator, SelectOperator και SelectSchemaOperator. Στο Σχήμα 4.6 παρουσιάζεται το διάγραμμα των κλάσεων CartesianProductOperator και RenameOperator. Στο Σχήμα 4.7 παρουσιάζεται το διάγραμμα των κλάσεων που υλοποιούν τους τελεστές συνόλων, δηλαδή το διάγραμμα των κλάσεων UnionOperator, IntersectionOperator και DifferenceOperator. Τέλος, στο Σχήμα 4.8 παρουσιάζεται το διάγραμμα των κλάσεων AddContextAttributesOperator, RemoveContextAttributesOperator και MapContextOperator.

Αναφέρουμε σε αυτό το σημείο κάποια κοινά attributes που εμφανίζονται στις κλάσεις που υλοποιούν unary τελεστές:

- `id polymorph_id`: Το id του polymorph που εισάγεται ως input.
- `string polymorph_name`: Το όνομα του polymorph που εισάγεται ως input.
- `id morph_id`: Το id κάποιου morph που ανήκει στο εισαχθέν polymorph.
- `DataBaseConnection DBconn`: Μέσω του συγκεκριμένου αντικειμένου πραγματοποιείται η επικοινωνία με τη βάση.

Αναφέρουμε, ομοίως, κάποια κοινά attributes που εμφανίζονται στις κλάσεις που υλοποιούν binary τελεστές:

- `id first_polymorph_id`: Το id του πρώτου polymorph που εισάγεται ως input.

- `string first_polymorph_name`: Το όνομα του πρώτου polymorph που εισάγεται ως input.
- `id second_polymorph_id`: Το id του δεύτερου polymorph που εισάγεται ως input.
- `string second_polymorph_name`: Το όνομα του δεύτερου polymorph που εισάγεται ως input.
- `id first_morph_id`: Το id κάποιου morph που ανήκει στο εισαχθέν πρώτο polymorph.
- `id second_morph_id`: Το id κάποιου morph που ανήκει στο εισαχθέν δεύτερο polymorph.
- `DataBaseConnection DBconn`: Μέσω του συγκεκριμένου αντικειμένου πραγματοποιείται η επικοινωνία με τη βάση.

Ακολουθούν κάποια από τα attributes που χρησιμοποιούνται σε συγκεκριμένους τελεστές:

- `list<attribute> attributes`: Η συγκεκριμένη μεταβλητή έχει δημιουργηθεί στις κλάσεις ProjectOperator, RenameOperator, AddContextAttributesOperator, RemoveContextAttributesOperator και MapContextOperator και περιέχει εκείνα τα attributes που πρόκειται να προβληθούν, να μετονομαστούν, να προστεθούν στο context schema του εισαχθέντος polymorph, να αφαιρεθούν από το context schema του εισαχθέντος polymorph ή να υποστούν mapping αντίστοιχα.
- `list<attribute> renamed_attributes`: Η συγκεκριμένη μεταβλητή έχει δημιουργηθεί στις κλάσεις ProjectOperator και RenameOperator και περιέχει τα νέα ονόματα των attributes που θα μετονομαστούν.
- `list<attribute> context_attributes`: Η συγκεκριμένη μεταβλητή έχει δημιουργηθεί στην κλάση ProjectOperator και περιέχει τα context attributes που θα προβληθούν.
- `list<attribute> renamed_context_attributes`: Η συγκεκριμένη μεταβλητή έχει δημιουργηθεί στην κλάση ProjectOperator και περιέχει τα νέα ονόματα των context attributes που θα μετονομαστούν.
- `string clause`: Η συγκεκριμένη μεταβλητή έχει δημιουργηθεί στις κλάσεις SelectOperator και SelectSchemaOperator και περιέχει τη συνθήκη που είναι απαραίτητη για την εκτέλεση των τελεστών select και select schema αντίστοιχα.
- `list<string> types`: Η συγκεκριμένη μεταβλητή έχει δημιουργηθεί στην κλάση AddContextAttributesOperator και περιέχει τους τύπους των context attributes που πρόκειται να προστεθούν στο context schema του εισαχθέντος polymorph.

- `list<string> values`: Η συγκεκριμένη μεταβλητή έχει δημιουργηθεί στην κλάση AddContextAttributesOperator και περιέχει τις default τιμές των context attributes που πρόκειται να προστεθούν στο context schema του εισαχθέντος polymorph.
- `list<string> expressions`: Η συγκεκριμένη μεταβλητή έχει δημιουργηθεί στην κλάση MapContextOperator και περιέχει τις εκφράσεις για τα context attributes που είναι απαραίτητες, προκειμένου να εκτελεστεί ο map τελεστής.

Η βασική μέθοδος που υλοποιήθηκε σε όλες τις κλάσεις τελεστών είναι η ακόλουθη:

- `id execute()`: Εκτελεί τον εκάστοτε τελεστή και επιστρέφει το id του προσωρινού polymorph που περιέχει το αποτέλεσμα.

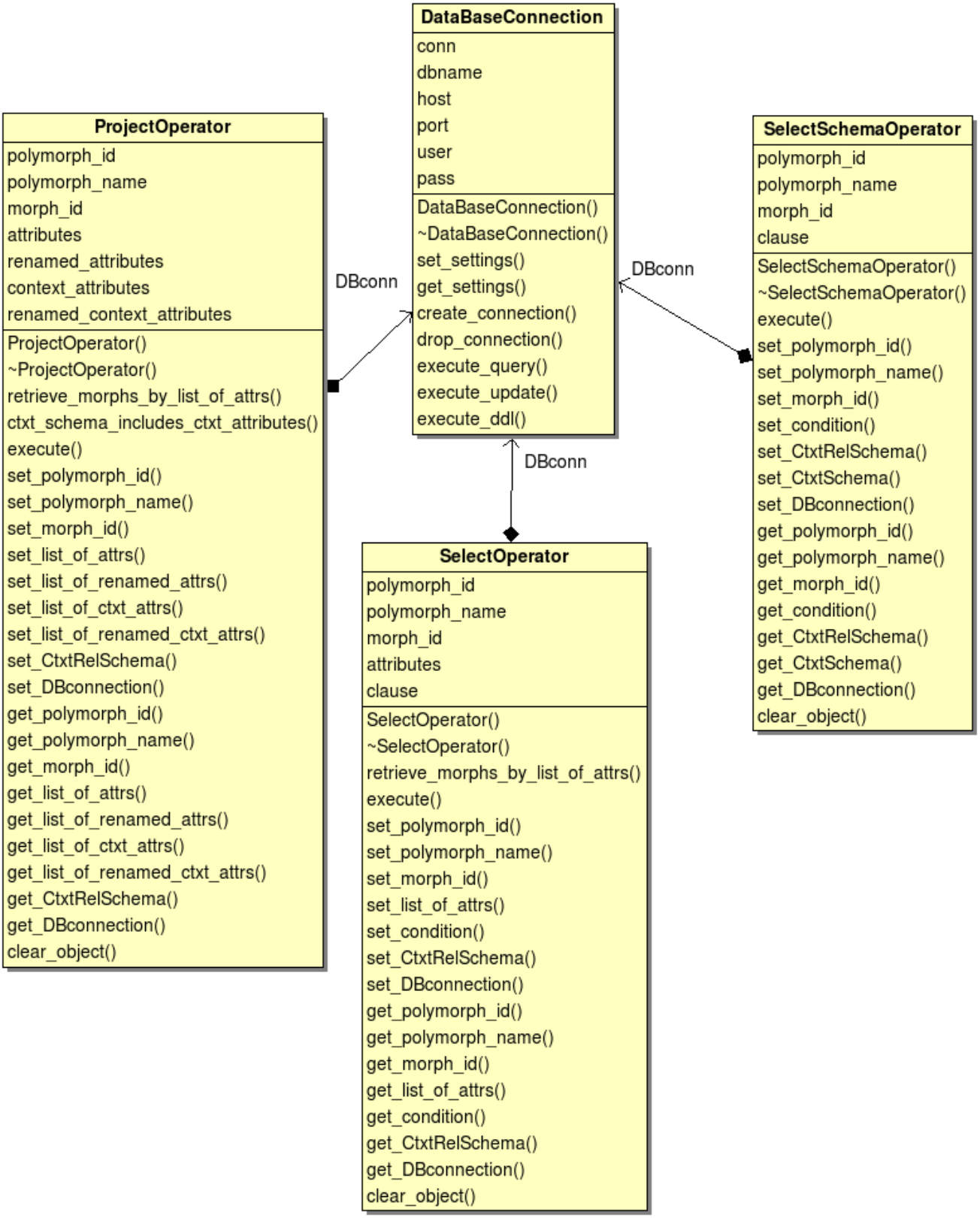
Ακολουθούν κάποιες κοινές μέθοδοι για τις κλάσεις που υλοποιούν τους binary τελεστές:

- `bool polymorphs_have_same_ctxt_schema_by_foreign_key()`: Η συγκεκριμένη μέθοδος ελέγχει αν τα input polymorphs έχουν οριστεί με βάση το ίδιο context schema, ελέγχοντας το αντίστοιχο context schema id στον κατάλογο.
- `bool polymorphs_have_same_ctxt_schema_by_ctxt_attributes()`: Η συγκεκριμένη μέθοδος ελέγχει αν τα input polymorphs έχουν οριστεί με βάση το ίδιο context schema, ελέγχοντας κάθε ένα από τα context attributes. Όπως αναφέραμε σε προηγούμενο κεφάλαιο, η συγκεκριμένη μέθοδος δε χρησιμοποιείται, καθώς για να είναι συμβατή μία πράξη με binary τελεστές, πρέπει αυτοί να έχουν οριστεί με βάση το ίδιο context schema έχοντας το ίδιο context schema id και όχι με διαφορετικά context schemas που έχουν το ίδιο σχήμα, διότι ο έλεγχος αυτός είναι αρκετά κοστοβόρος. Υλοποιήσαμε, λοιπόν, τη συγκεκριμένη μέθοδο για πιθανή μελλοντική χρήση.

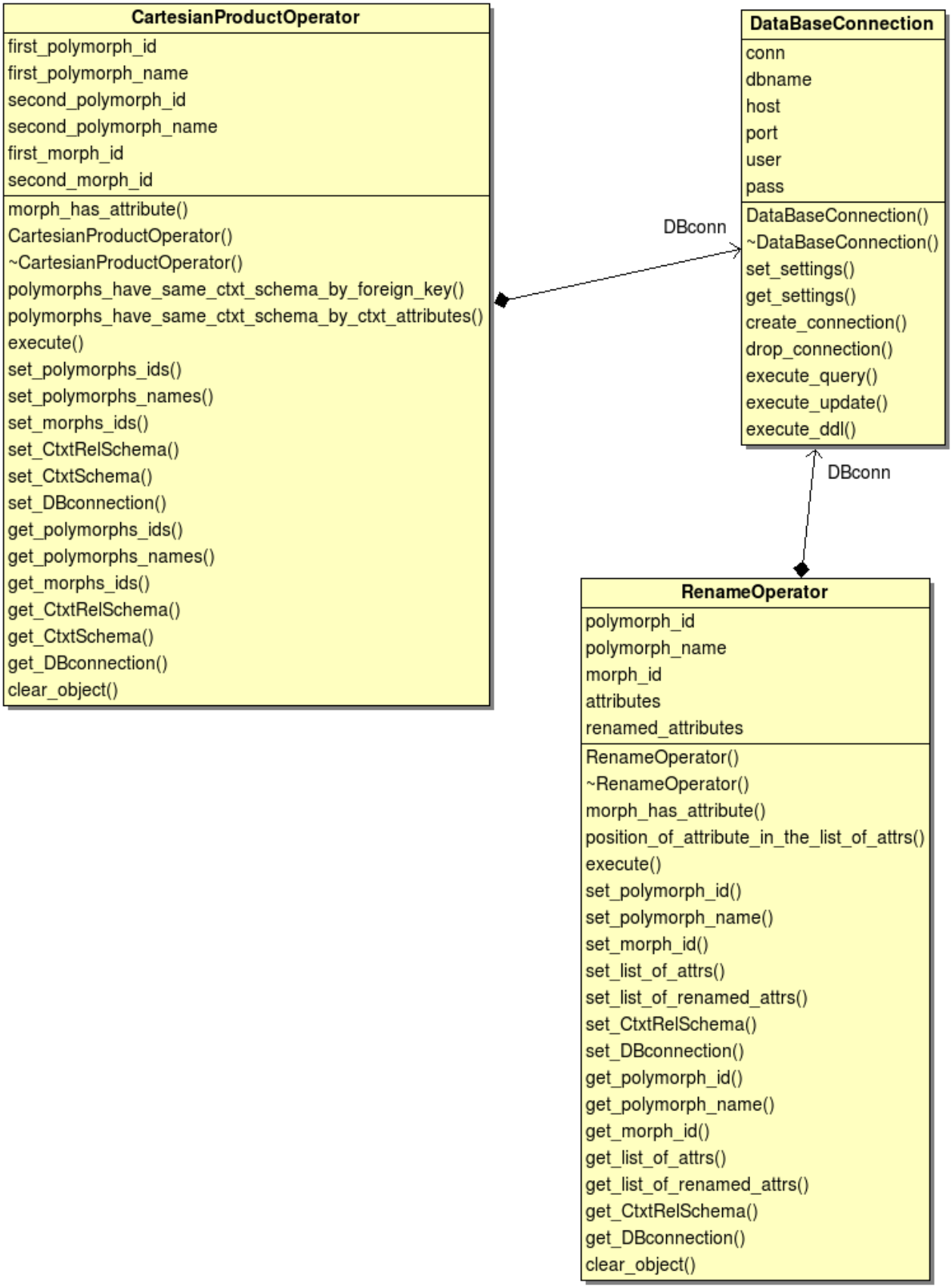
Ακολουθεί κοινή μέθοδος για τις κλάσεις τελεστών συνόλων (set operators), καθώς και για τις κλάσεις RemoveContextAttributesOperator και MapContextOperator, οι οποίες κάνουν merge context και morphs και αυτά τα morphs πρέπει να είναι union-compatibles:

- `bool morphs_are_union_compatible(id,id)`: Η συγκεκριμένη μέθοδος ελέγχει αν τα morphs που δίνονται ως παράμετροι είναι union-compatible μεταξύ τους.

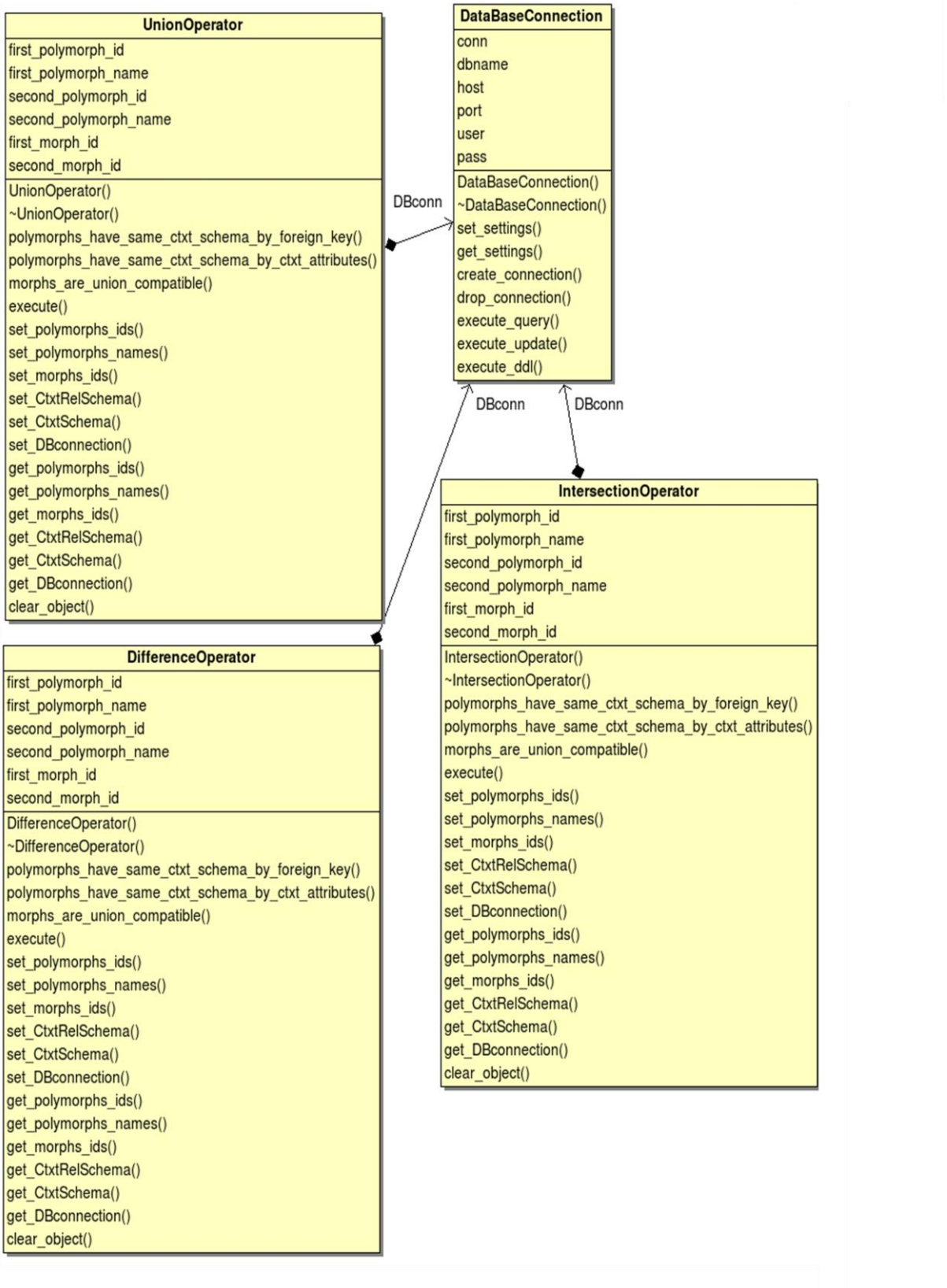
Ακολουθούν τα διαγράμματα κλάσεων που υλοποιούν τους context-aware τελεστές για το σύστημα.



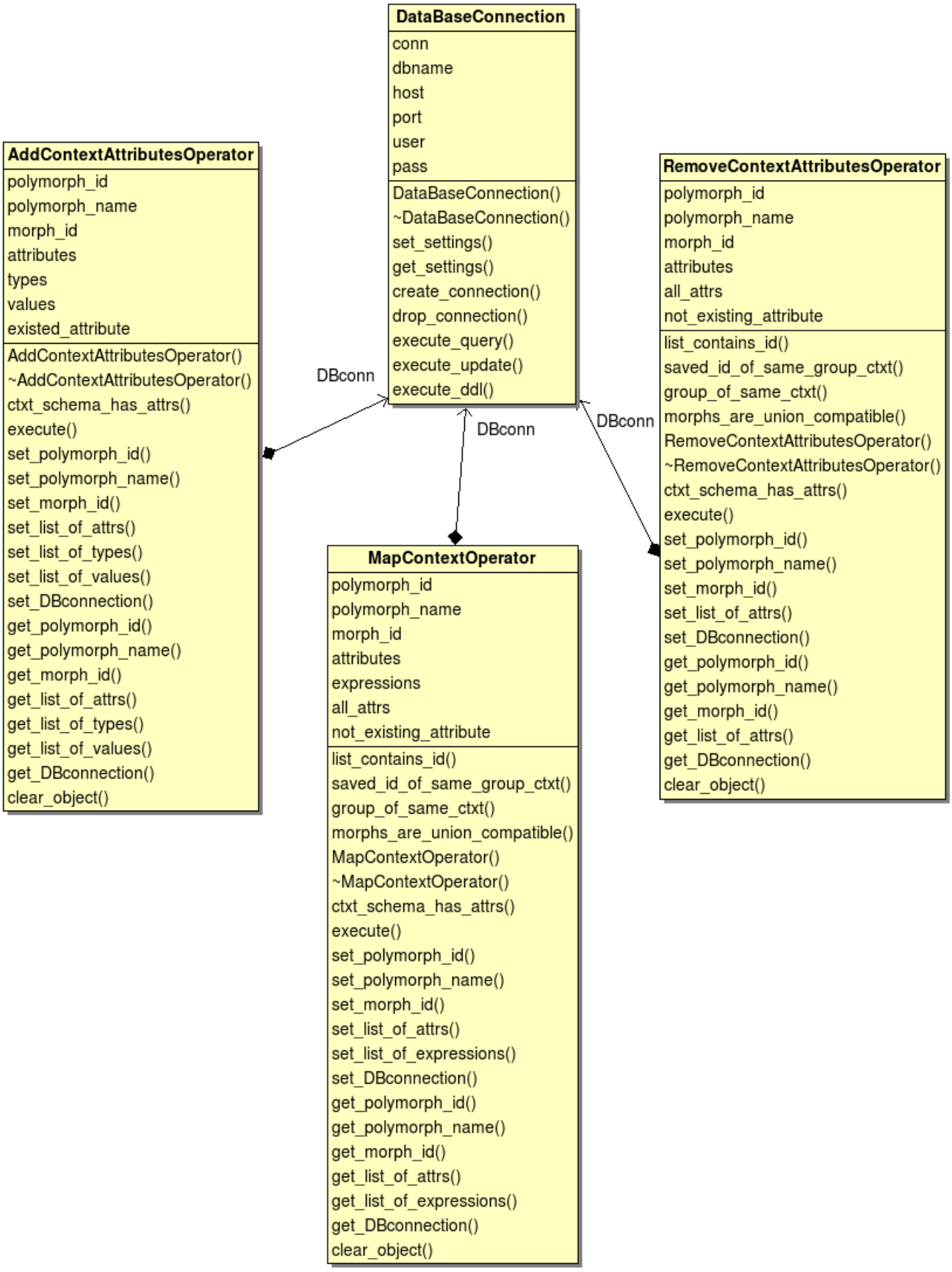
Σχήμα 4.5. Διάγραμμα κλάσεων ProjectOperator, SelectOperator και SelectSchemaOperator



Σχήμα 4.6. Διάγραμμα κλάσεων CartesianProductOperator και RenameOperator



Σχήμα 4.7. Διάγραμμα κλάσεων UnionOperator, IntersectionOperator και DifferenceOperator (τελεστές συνόλων)



Σχήμα 4.8. Διάγραμμα κλάσεων AddContextAttributesOperator, RemoveContextAttributesOperator και MapContextOperator

4.2.6 Υποσύστημα Εκτέλεσης Πλάνων (Query Executor)

Όπως έχουμε αναφέρει σε προηγούμενο κεφάλαιο, έχουμε μοντελοποιήσει ένα πλάνο το οποίο αφού οριστεί, εν συνεχεία εκτελείται. Έχουμε, λοιπόν, δημιουργήσει μία ιεραρχία κλάσεων για το σκοπό αυτό, την οποία περιγράφουμε προτού προχωρήσουμε στην περιγραφή της μονάδας εκτέλεσης.

Η ιεραρχία αυτή αποτελείται στο ανώτερο επίπεδο από μία αφαιρετική κλάση, την κλάση Operator. Η συγκεκριμένη κλάση κληρονομεί στις επίσης αφαιρετικές κλάσεις UnaryOperator και BinaryOperator. Επίσης, έχουν δημιουργηθεί κλάσεις, κάθε μία από τις οποίες αντιστοιχεί σε ένα κόμβο για κάθε τελεστή και έναν κόμβο για τα polymorphs, οι οποίες κληρονομούν από τις κλάσεις UnaryOperator ή BinaryOperator, ανάλογα με το αν ο εκάστοτε τελεστής είναι unary ή binary. Η κλάση-κόμβος που αφορά στα polymorphs κληρονομεί απευθείας από την κλάση Operator. Τα private members αυτών των κλάσεων-κόμβων είναι τα ίδια με τα αντίστοιχα των κλάσεων-τελεστών. Τέλος, έχει δημιουργηθεί η κλάση Node, η οποία αντιπροσωπεύει ένα γενικό κόμβο για το δέντρο-πλάνο. Η συγκεκριμένη κλάση περιέχει αντικείμενα για όλες τις κλάσεις-κόμβους και αρχικοποιεί το αντίστοιχο αντικείμενο για τον εκάστοτε τελεστή.

Τελικά, υλοποιείται η κλάση QueryPlan που μοντελοποιεί το πλάνο εκτέλεσης.

Οι βασικές μέθοδοι της κλάσεις QueryPlan είναι οι ακόλουθες:

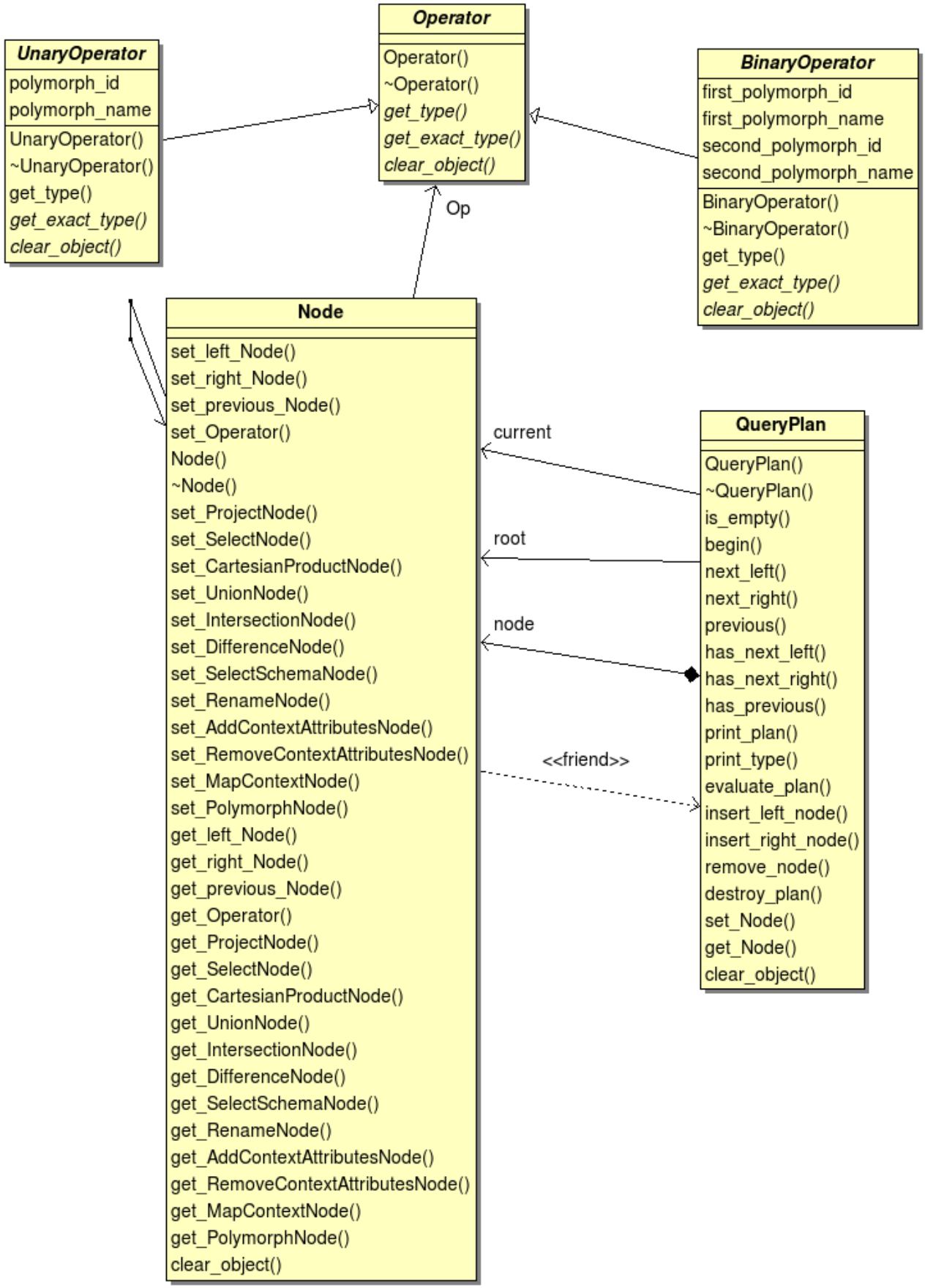
- `Node node`: Η μεταβλητή node περιέχει τον κόμβο που πρόκειται να εισαχθεί/αφαιρεθεί/ελεγχθεί.
- `Node* root`: Δείκτης στον κόμβο-ρίζα του δέντρου.
- `Node* current`: Δείκτης στον τρέχοντα κόμβο του δέντρου.

Οι βασικές μέθοδοι της κλάσεις QueryPlan είναι οι ακόλουθες:

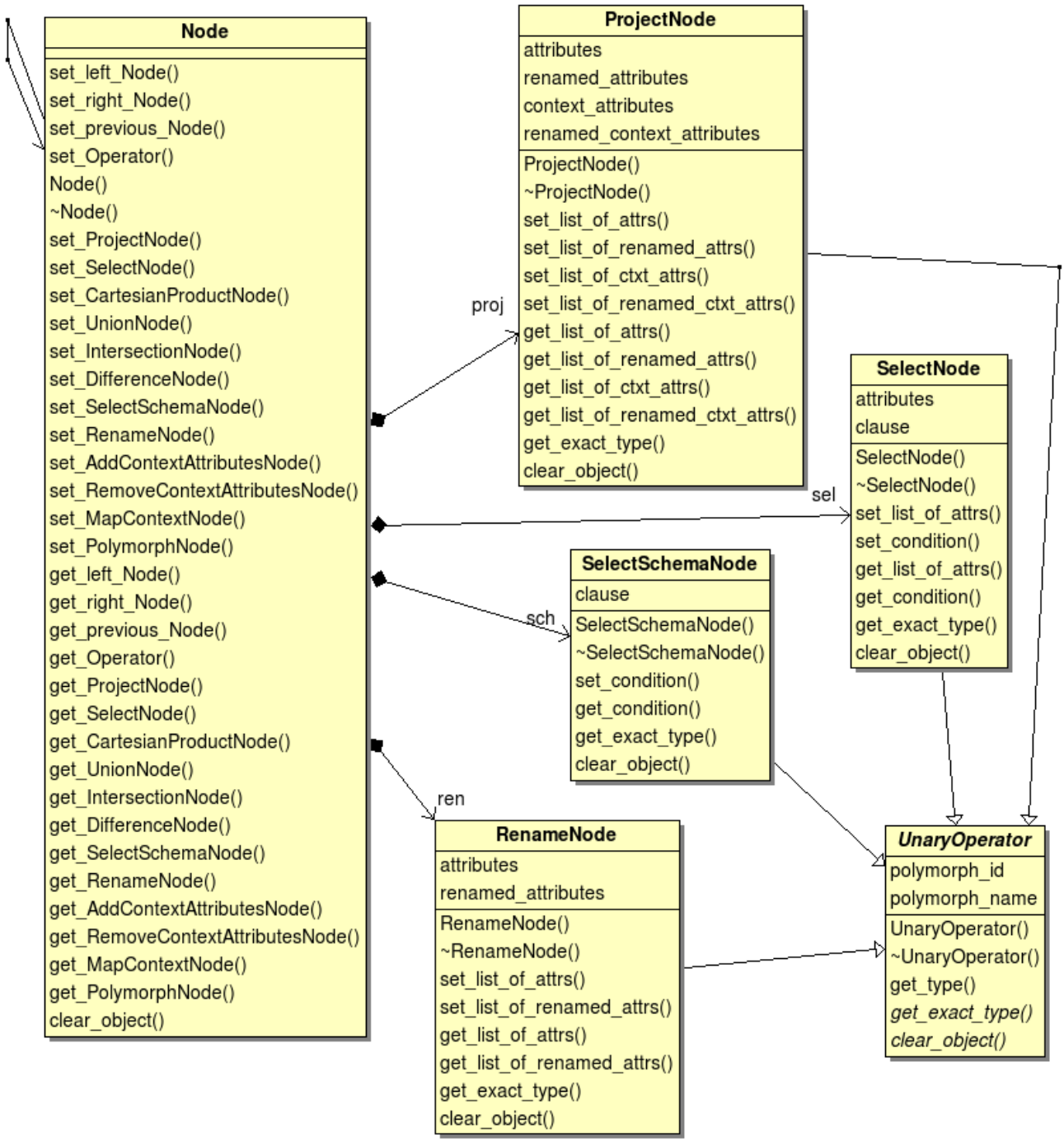
- `bool is_empty()`: Επιστρέφει true, αν το πλάνο εκτέλεσης δεν περιέχει κανένα κόμβο, αλλιώς επιστρέφει false.
- `Node* begin()`: Επιστρέφει ένα δείκτη στον κόμβο-ρίζα του δέντρου, αν υπάρχει, δηλαδή αν το πλάνο δεν είναι άδειο.
- `Node* next_left()`: Επιστρέφει ένα δείκτη στο αριστερό παιδί του τρέχοντος κόμβου, αν υπάρχει.
- `Node* next_right()`: Επιστρέφει ένα δείκτη στο δεξί παιδί του τρέχοντος κόμβου, αν υπάρχει.
- `Node* previous()`: Επιστρέφει ένα δείκτη στον πατέρα του τρέχοντος κόμβου, αν υπάρχει, δηλαδή αν ο τρέχων κόμβος δεν είναι η ρίζα του δέντρου.

- `void print_plan()`: Εκτυπώνει το πλάνο του δέντρου στο terminal.
- `void evaluate_plan()`: Ελέγχει την εγκυρότητα του πλάνου. Κάθε πλάνο για να είναι έγκυρο πρέπει α) να έχει μόνο μία ρίζα, β) κάθε unary κόμβος να έχει ακριβώς ένα παιδί, γ) κάθε binary κόμβος να έχει ακριβώς δύο παιδιά και δ) κάθε polymorph κόμβος να μην έχει κανένα παιδί.
- `void insert_left_node()`: Εισάγει τον κόμβο της μεταβλητής node ως αριστερό παιδί στον τρέχοντα κόμβο.
- `void insert_right_node()`: Εισάγει τον κόμβο της μεταβλητής node ως δεξί παιδί στον τρέχοντα κόμβο.
- `void remove_node()`: Αφαιρεί από το δέντρο τον δοθέντα κόμβο.
- `int destroy_plan()`: Καταστρέφει το πλάνο αφαιρώντας όλους τους κόμβους του δέντρου και επιστρέφει τον αριθμό των κόμβων που αφαιρέθηκαν.

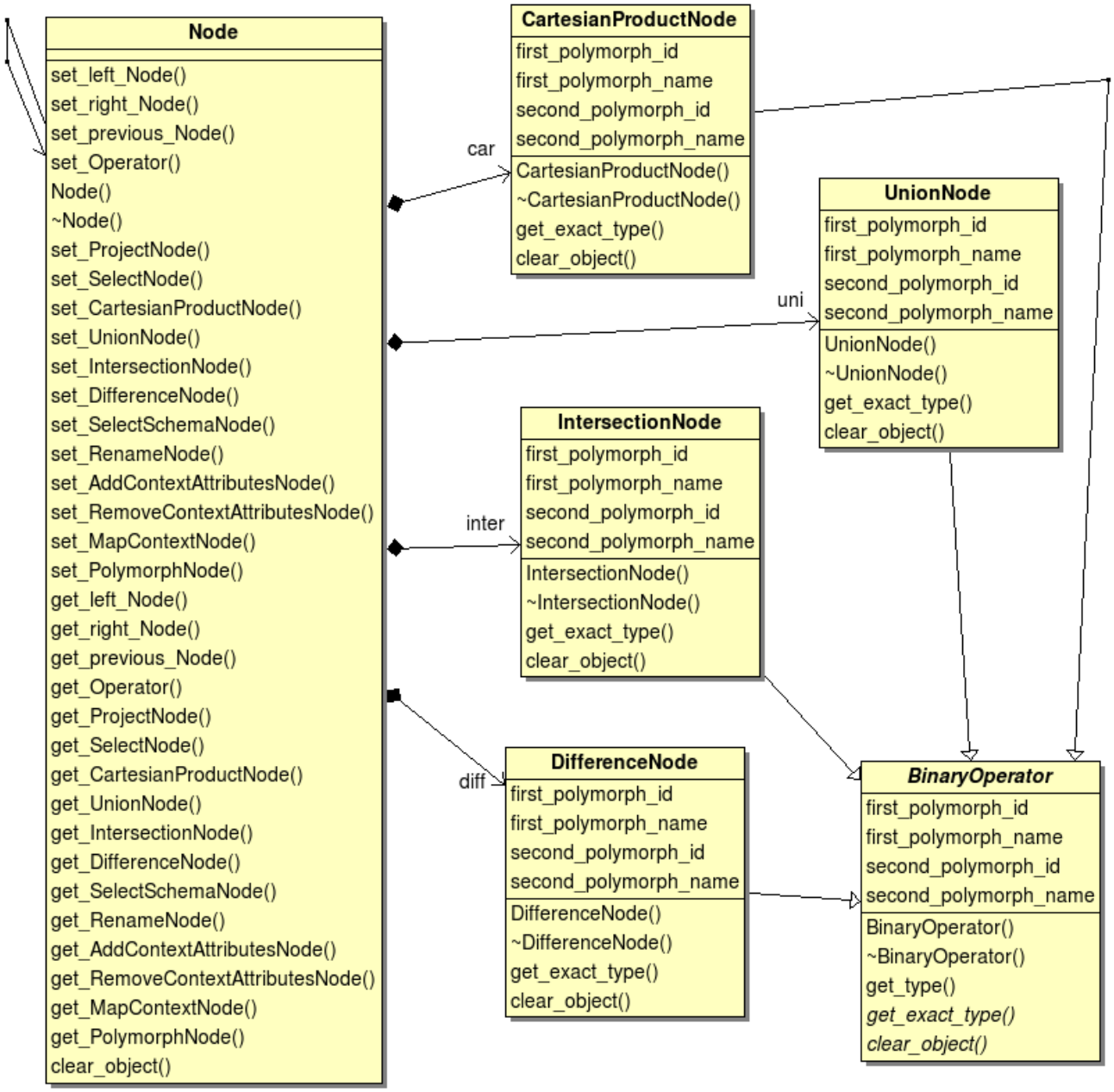
Ακολουθούν τα Σχήματα 4.9-4.12, τα οποία περιέχουν τα διαγράμματα της ιεραρχίας των κλάσεων που περιγράψαμε καθώς και τη κλάσης QueryPlan.



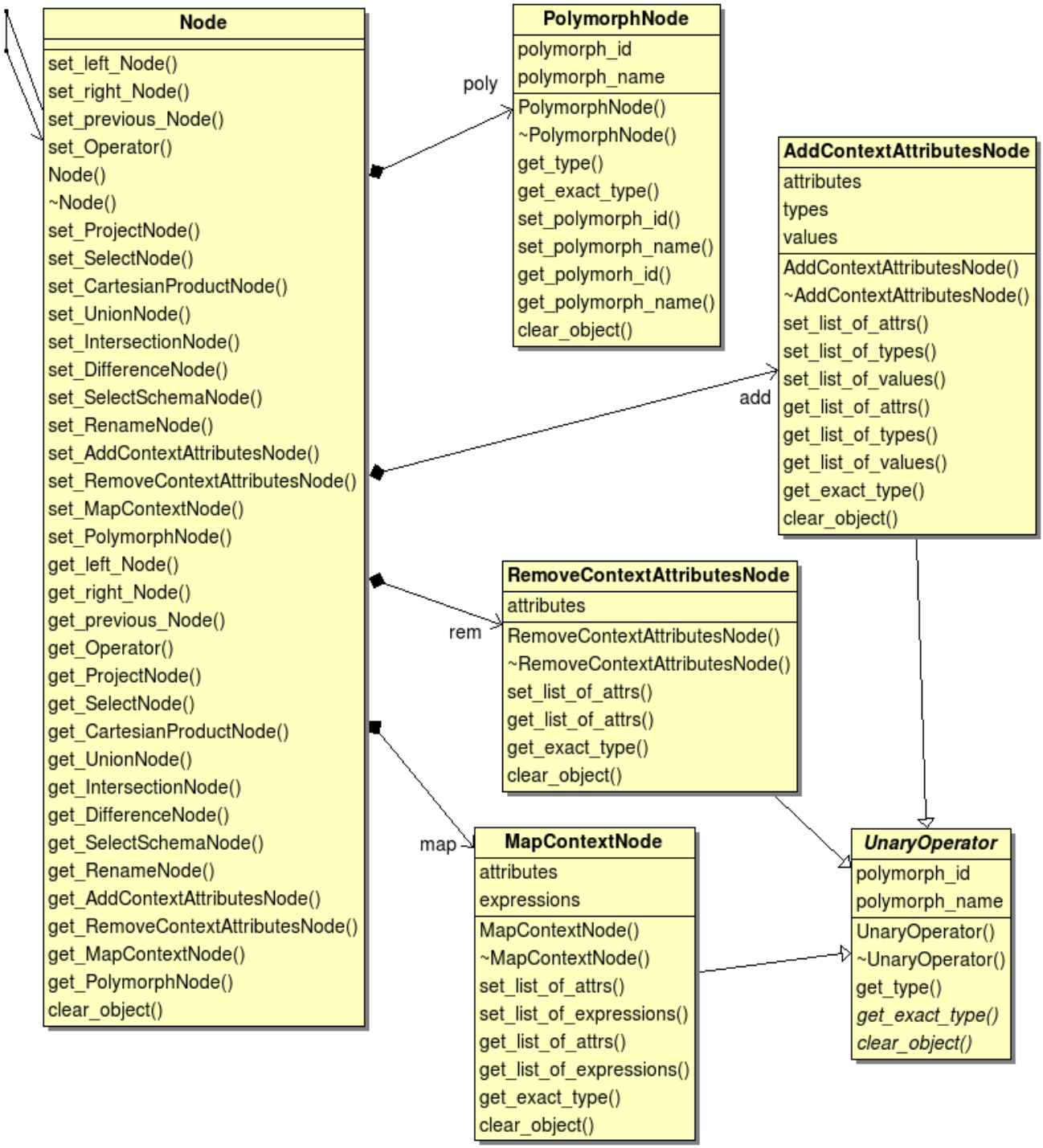
Σχήμα 4.9 Διάγραμμα κλάσεων Operator, UnaryOperator, BinaryOperator, Node και QueryPlan



Σχήμα 4.10 Διάγραμμα κλάσεων ProjectNode, SelectNode, RenameNode και SelectSchemaNode

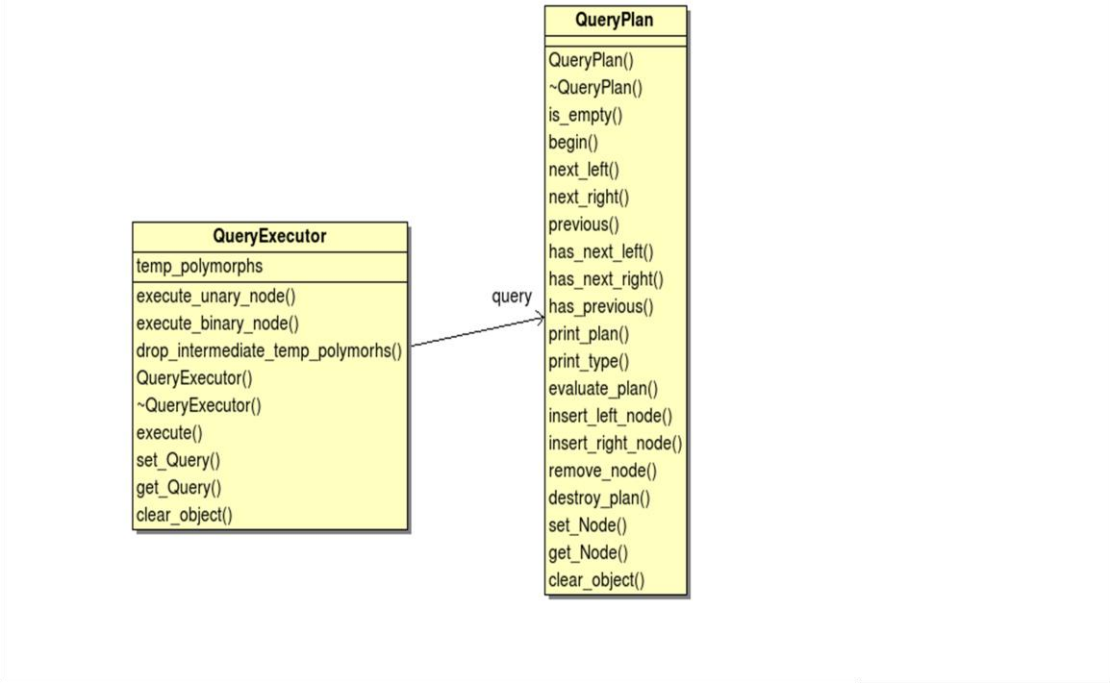


Σχήμα 4.11 Διάγραμμα κλάσεων CartesianProductNode, UnionNode, IntersectionNode και DifferenceNode



Σχήμα 4.12 Διάγραμμα κλάσεων AddContextAttributesNode, RemoveContextAttributesNode, MapContextNode και PolymorphNode

Τέλος, η κλάση που είναι υπεύθυνη για την εκτέλεση ενός πλάνου εκτέλεσης είναι η κλάση QueryExecutor, το διάγραμμα της οποία παρουσιάζεται στο Σχήμα 4.13.



Σχήμα 4.13 Διάγραμμα κλάσης QueryExecutor

Τα βασικά attributes της κλάσης QueryPlan είναι τα εξής:

- `QueryPlan* query`: Δείκτης στο πλάνο που πρόκειται να εκτελεσθεί.
- `ContextRelation* result`: Δείκτης στο προσωρινό polymorph, που αποτελεί το αποτέλεσμα της εκτέλεσης ενός πλάνου.
- `list<id> temp_polymorphs`: Λίστα από τα προσωρινά polymorphs που δημιουργήθηκαν κατά την εκτέλεση του πλάνου.

Οι βασικές μέθοδοι της κλάσης QueryPlan είναι οι ακόλουθες:

- `ContextRelation* execute()`: Εκτελεί το δοθέν πλάνο αναδρομικά και επιστρέφει ένα δείκτη στο τελικό προσωρινό polymorph που αποτελεί το αποτέλεσμα της εν λόγω εκτέλεσης.
- `id execute_unary_node(Node*)`: Εκτελεί τον unary τελεστή που προσδιορίζεται στην παράμετρο και επιστρέφει το id του προσωρινού polymorph που δημιουργήθηκε ως αποτέλεσμα.
- `id execute_binary_node(Node*)`: Εκτελεί το binary τελεστή που προσδιορίζεται στην παράμετρο και επιστρέφει το id του προσωρινού polymorph που δημιουργήθηκε ως αποτέλεσμα.

- `void drop_intermediate_temp_polymorphs(list<id>)`: Διαγράφει τα polymorphs που δίνονται ως παράμετροι. Ουσιαστικά, η μέθοδος χρησιμοποιείται για τη διαγραφή όλων των προσωρινών polymorphs, εκτός από το τελευταίο (που είναι και το τελικό αποτέλεσμα), που δημιουργούνται κατά την εκτέλεση του πλάνου.

5

Έλεγχος

Στο κεφάλαιο αυτό παρουσιάζεται ο έλεγχος του συστήματος. Στην ενότητα 5.1 αναλύεται η γενική μεθοδολογία που χρησιμοποιήσαμε για τον έλεγχο του context-aware ΣΔΒΔ. Στην ενότητα 5.2 περιγράφουμε τη δημιουργία ενός αντιπροσωπευτικού και ρεαλιστικού παραδείγματος με μία context-aware βάση δεδομένων και την εκτέλεση συγκεκριμένων ερωτημάτων στη βάση αυτή για κάθε υλοποίηση (A και B). Επίσης, μετράται ο χρόνος εκτέλεσης του κάθε ερωτήματος και συγκρίνεται η απόδοση μεταξύ των δύο υλοποιήσεων. Τέλος, στην ενότητα 5.3 συνοψίζουμε τα συμπεράσματα που προέκυψαν για την απόδοση των δύο υλοποιήσεων για τα ερωτήματα που εκτελέστηκαν.

5.1 Μεθοδολογία ελέγχου

Κατά τον έλεγχο του συστήματος, μετά την υλοποίηση κάθε κλάσης προχωρούσαμε στην εκτέλεση μίας κλάσης ελέγχου (test class). Αυτή η κλάση ελέγχου περιλαμβάνει την εκτέλεση συγκεκριμένων σεναρίων ανάλογα με τη λειτουργία της εκάστοτε κλάσης. Τα σενάρια αυτά διακρίνονται σε πλάνα που ολοκληρώνονται επιτυχώς και σε παθολογικά πλάνα, προκειμένου να ελέγξουμε τις κλάσεις και τις περιπτώσεις εκείνες που προκύπτει κάποιο λάθος κατά την εκτέλεση. Αυτό μπορεί να είναι κάποια εξαίρεση (exception), κάποιο σφάλμα (bug) στον κώδικα ή κάποιο λογικό σφάλμα.

Σε κάθε περίπτωση πριν την εκτέλεση αυτών των κλάσεων ελέγχου υπήρχε ο προσδιορισμός ενός συγκεκριμένου αποτελέσματος, το οποίο έπρεπε να επαληθευθεί όπως αναμενόταν, ώστε να θεωρηθεί επιτυχής η όλη διαδικασία ελέγχου. Έτσι, για κάθε κλάση, έχει υλοποιηθεί και η αντίστοιχη κλάση ελέγχου. Εκτός από αυτές, έχουν, επίσης, υλοποιηθεί και σενάρια ελέγχου (test cases), μέσω κλάσεων ελέγχου που ελέγχουν ολόκληρο το σύστημα και οι οποίες περιγράφονται στις επόμενες δύο ενότητες. Τόσο οι απλές κλάσεις ελέγχου (test classes), όσο και τα ολοκληρωμένα σενάρια (test cases), ολοκλήρωσαν επιτυχώς την εκτέλεσή τους.

5.2 Αναλυτική παρουσίαση ελέγχου για ένα αντιπροσωπευτικό και ρεαλιστικό παράδειγμα

Στην ενότητα αυτή περιγράφουμε αρχικά, όπως αναφέρθηκε προηγουμένως, τη δημιουργία μίας context-aware βάσης δεδομένων που προσεγγίζει τις ανάγκες αποθήκευσης και ανάκτησης δεδομένων για ένα πραγματικό σύστημα. Με την παρουσίαση του εν λόγω παραδείγματος προσπαθούμε μεταξύ άλλων να αναδείξουμε την ανάγκη και τη χρησιμότητα ύπαρξης ενός context-aware ΣΔΒΔ, το οποίο θα διαχειρίζεται αυτού του είδους τα δεδομένα.

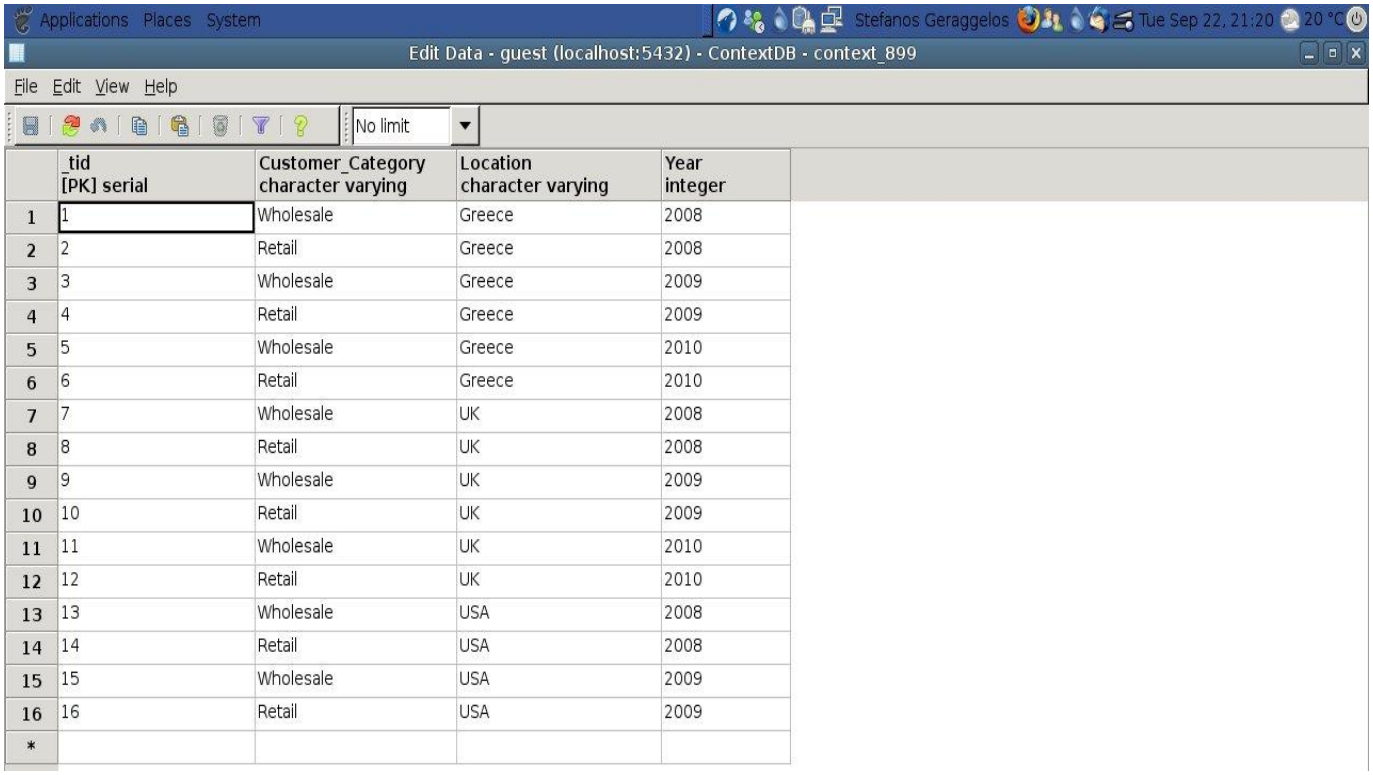
Η βάση δεδομένων που επιλέχθηκε ως γενικό παράδειγμα αφορά μία ηλεκτρονική "αποθήκη", η οποία προμηθεύει με ηλεκτρονικά προϊόντα διάφορα καταστήματα (λιανικό / χονδρικό εμπόριο) σε παγκόσμιο επίπεδο. Η αποθήκη αυτή διατηρεί δεδομένα για τα προϊόντα αυτά, για τα καταστήματα που συνεργάζεται καθώς και για το ποιο/ποια προϊόν/προϊόντα πουλήθηκε/πουλήθηκαν σε ποιο/ποια κατάστημα/καταστήματα και τις λεπτομέρειες αυτών των συναλλαγών.

Η βάση αυτή διαφοροποιείται ως προς τα δεδομένα αλλά και ως προς το σχήμα της ανάλογα με τους διαφορετικούς τύπους πελατών (λιανική / χονδρική), τη διαφορετική τοποθεσία των καταστημάτων και τις διαφορετικές χρονιές κατά τις οποίες έγιναν οι πωλήσεις. Για παράδειγμα, ένα κατάστημα S_A που συνεργάστηκε τη χρονιά 2008 με την ηλεκτρονική αποθήκη μέσω λιανικού εμπορίου και βρίσκεται στην Ελλάδα έχει διαφορετικά δεδομένα και σχήμα για τα προϊόντα από ένα άλλο κατάστημα S_B που συνεργάστηκε τη χρονιά 2009 με την ηλεκτρονική αποθήκη και βρίσκεται στο Ηνωμένο Βασίλειο.

Στο σημείο αυτό, λοιπόν, γίνεται φανερή η ανάγκη ύπαρξης του context και ενός συστήματος που θα διαφοροποιείται σύμφωνα με αυτό. Πιο συγκεκριμένα, το σχήμα του context για το εν λόγω παράδειγμα ορίζεται ως εξής <Customer_Category, Location, Year>. Οι τιμές για κάθε context attribute προέρχονται από τα σύνολα (Wholesale, Retail), (Greece, UK, USA), (2008,

2009, 2010) αντίστοιχα. Συνολικά, έχουμε δημιουργήσει 16 instances για το συγκεκριμένο context, όπως φαίνεται και στο Σχήμα 5.1. Για παράδειγμα, ένα τέτοιο μπορεί να είναι το <Wholesale, USA, 2009>, ενώ ένα άλλο το <Retail, Greece, 2008>.

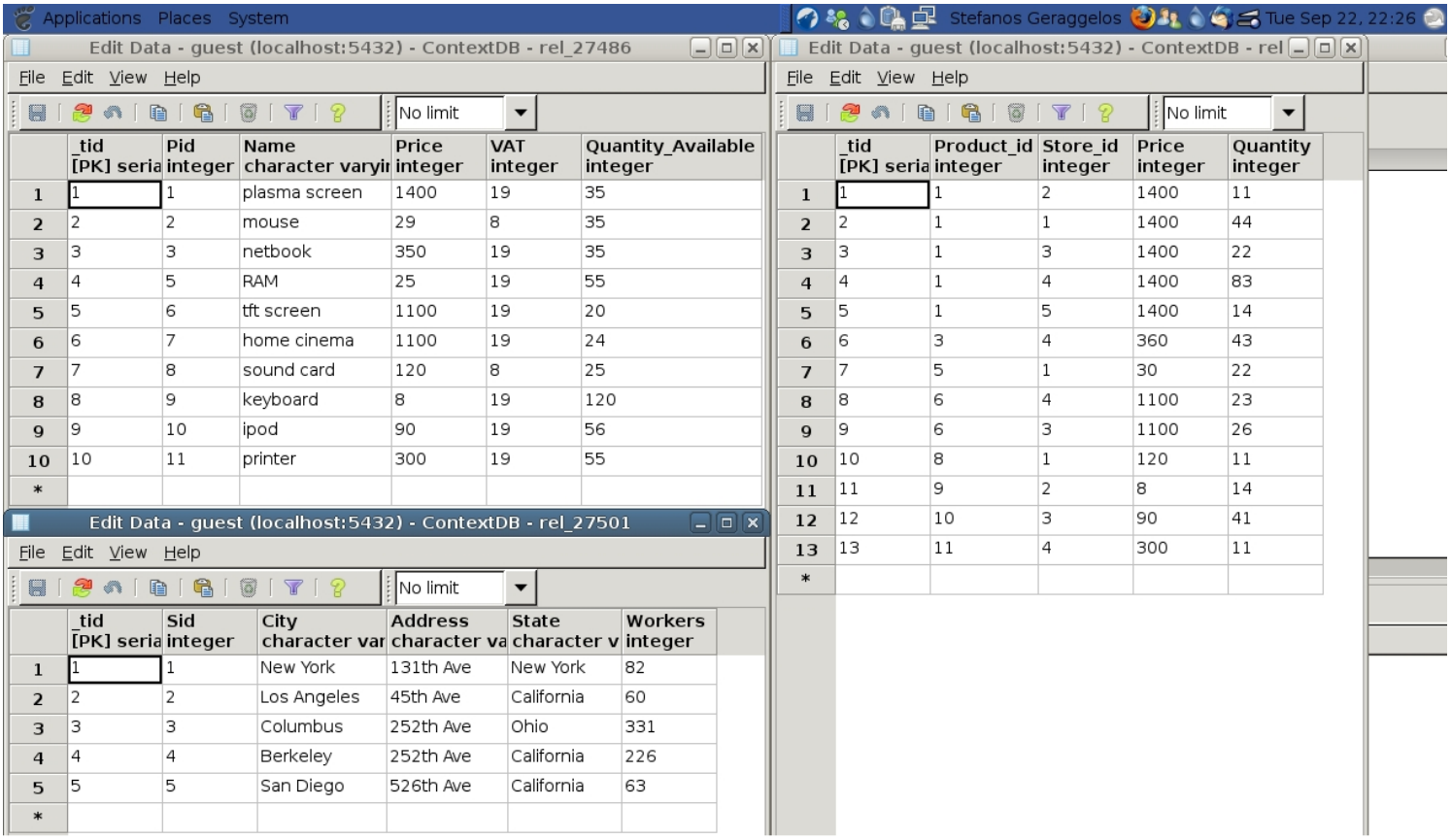
Με βάση αυτό το context ορίζονται τα polymorphs Product, Store, Product_Inventory, τα οποία διαχειρίζονται δεδομένα για τα προϊόντα, τα καταστήματα και τις ακριβείς συναλλαγές που αφορούν το ποια προϊόντα πωλήθηκαν σε ποια καταστήματα αντίστοιχα. Για κάθε ένα από τα παραπάνω polymorphs, έχουν οριστεί 16 morphs (όσες και οι τιμές για το context), κάποια από τα οποία παρουσιάζονται ενδεικτικά στο Σχήμα 5.2 (ένα για κάθε polymorph). Η παρουσίαση αυτή γίνεται για την υλοποίηση A, όπου κάθε morph αντιστοιχεί σε ένα table. Όπως αναφέρθηκε νωρίτερα, για κάθε τέτοιο morph, αλλάζουν όχι μόνο τα δεδομένα, αλλά σε πολλές περιπτώσεις και το σχήμα του. Για παράδειγμα, στα morphs που ανήκουν στο polymorph Product και ορίζονται για το Ηνωμένο Βασίλειο (UK) και τις Ηνωμένες Πολιτείες Αμερικής (USA), στις οποίες χρησιμοποιείται ο φόρος, υπάρχει αυτό το attribute (VAT), ενώ για αυτά που έχουν οριστεί για την Ελλάδα (Greece) όχι. Επίσης, για τα καταστήματα που βρίσκονται στο Ηνωμένο Βασίλειο, αποθηκεύονται πληροφορίες για την περιοχή (region) του καταστήματος, ενώ για αυτά που βρίσκονται στις Ηνωμένες Πολιτείες Αμερικής, αποθηκεύεται η αντίστοιχη πολιτεία (state).



	_tid [PK] serial	Customer_Category character varying	Location character varying	Year integer
1	1	Wholesale	Greece	2008
2	2	Retail	Greece	2008
3	3	Wholesale	Greece	2009
4	4	Retail	Greece	2009
5	5	Wholesale	Greece	2010
6	6	Retail	Greece	2010
7	7	Wholesale	UK	2008
8	8	Retail	UK	2008
9	9	Wholesale	UK	2009
10	10	Retail	UK	2009
11	11	Wholesale	UK	2010
12	12	Retail	UK	2010
13	13	Wholesale	USA	2008
14	14	Retail	USA	2008
15	15	Wholesale	USA	2009
16	16	Retail	USA	2009
*				

Σχήμα 5.1. Το context και οι διαφορετικές τιμές του

Στο σημείο αυτό αναφέρουμε κάποια ποσοτικά δεδομένα για την context-aware βάση δεδομένων που υλοποιήσαμε, για να γίνει κατανοητό το μέγεθος της εν λόγω βάσης. Γενικά, τα morphs που αποτελούν το polymorph Product (προϊόντα) περιέχουν κάποια από τα attributes Pid (Product id), Όνομα (Name), Ενδεικτική τιμή πώλησης του προϊόντος (Price), Φόρος (VAT) και Διαθέσιμη ποσότητα του προϊόντος (Quantity_Available). Τα morphs που αποτελούν το polymorph Store περιέχουν κάποια από τα attributes Sid (Store id), Πόλη (City), Διεύθυνση (Address), Ταχυδρομικός Κώδικας (Postal_code), Περιοχή (Region), Πολιτεία (State) και Αριθμός εργαζομένων (Workers). Τέλος, τα morphs που αποτελούν το polymorph Product_Inventory περιέχουν κάποια από τα attributes Product_id, Store_id, ακριβής Τιμή που πωλήθηκε το συγκεκριμένο προϊόν στο συγκεκριμένο κατάστημα (Price) και ποσότητα του προϊόντος που πωλήθηκε (Quantity).

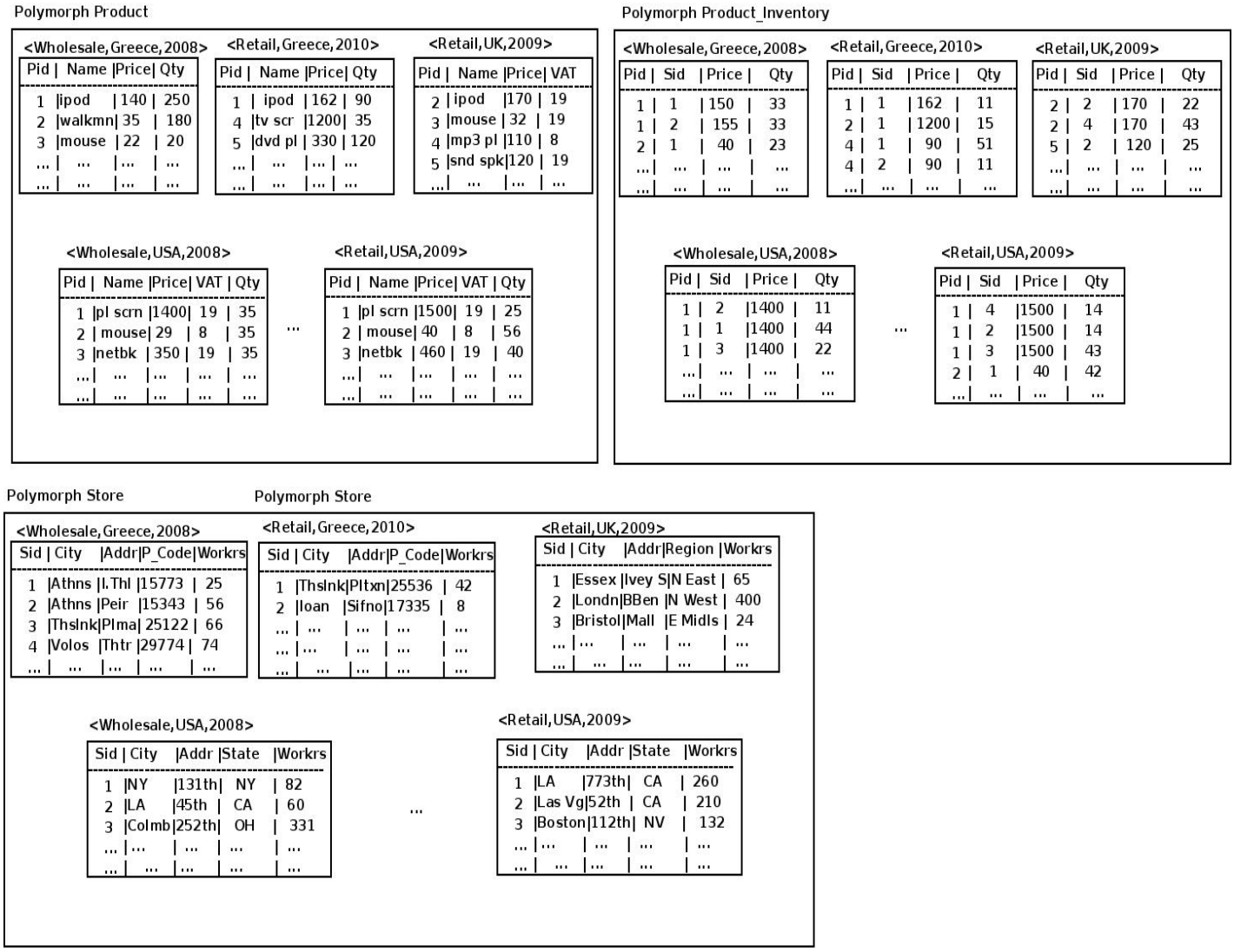


Σχήμα 5.2. Κάποια από τα morphs (1 για κάθε polymorph) και τα δεδομένα τους (συγκεκριμένα τα εν λόγω morphs έχουν οριστεί για το context instance <Wholesale, USA, 2008> και παρουσιάζονται με βάση την υλοποίηση A

Στο σημείο αυτό αξίζει να γίνει μία αναφορά στον όγκο των εγγραφών (tuples) και να δοθεί μία ιδέα για τα δεδομένα των morphs, προκειμένου να γίνει καλύτερη εκτίμηση των χρόνων εκτέλεσης ερωτημάτων, που θα παρουσιαστούν στη συνέχεια. Όσον αφορά, λοιπόν, αυτά τα δεδομένα, υπάρχουν κατά μέσο όρο 10 tuples για κάθε morph του polymorph Product, κατά

μέσο όρο 5 tuples για κάθε morph του polymorph Store και κατά μέσο όρο 15 tuples για κάθε morph του polymorph Product_Inventory.

Παρουσιάζουμε, επίσης, σχηματικά τη δομή της συγκεκριμένης βάσης δεδομένων στην υλοποίηση A. Η δομή αυτή φαίνεται στο Σχήμα 5.3. Τέλος, στην παρούσα context-aware βάση δεδομένων έχουμε εκτελέσει 12 διαφορετικά ερωτήματα. Στις επόμενες υποενότητες περιγράφουμε αναλυτικά αυτά τα ερωτήματα, καθώς και τα αποτελέσματα που παρήγαγε η εκτέλεσή τους, τους χρόνους εκτέλεσης τους για την κάθε υλοποίηση και τους επιμέρους χρόνους κάθε τελεστή κάθε ερωτήματος. Πρέπει να επισημάνουμε σε αυτό το σημείο ότι για να προσδιορίσουμε με μεγαλύτερη ακρίβεια τους χρόνους εκτέλεσης των ερωτημάτων και των επιμέρους τελεστών, εκτελούμε το κάθε ερώτημα 10 φορές και προσδιορίζουμε ένα μέσο όρο για κάθε ερώτημα και για κάθε επιμέρους τελεστή για κάθε μία από τις υλοποίησεις.

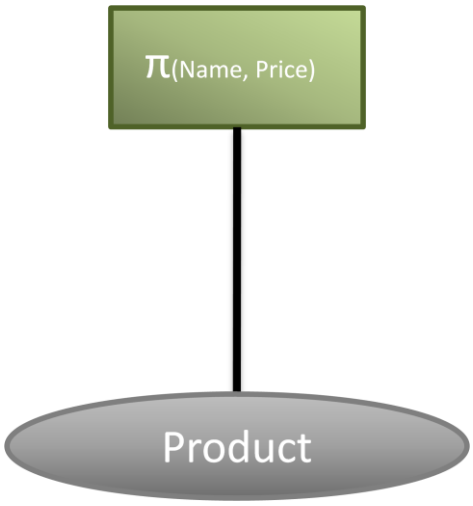


Σχήμα 5.3. Σχηματική αναπαράσταση της context-aware Δεδομένων 'MarketPlace' στην υλοποίηση A

5.2.1 Ερώτημα 1

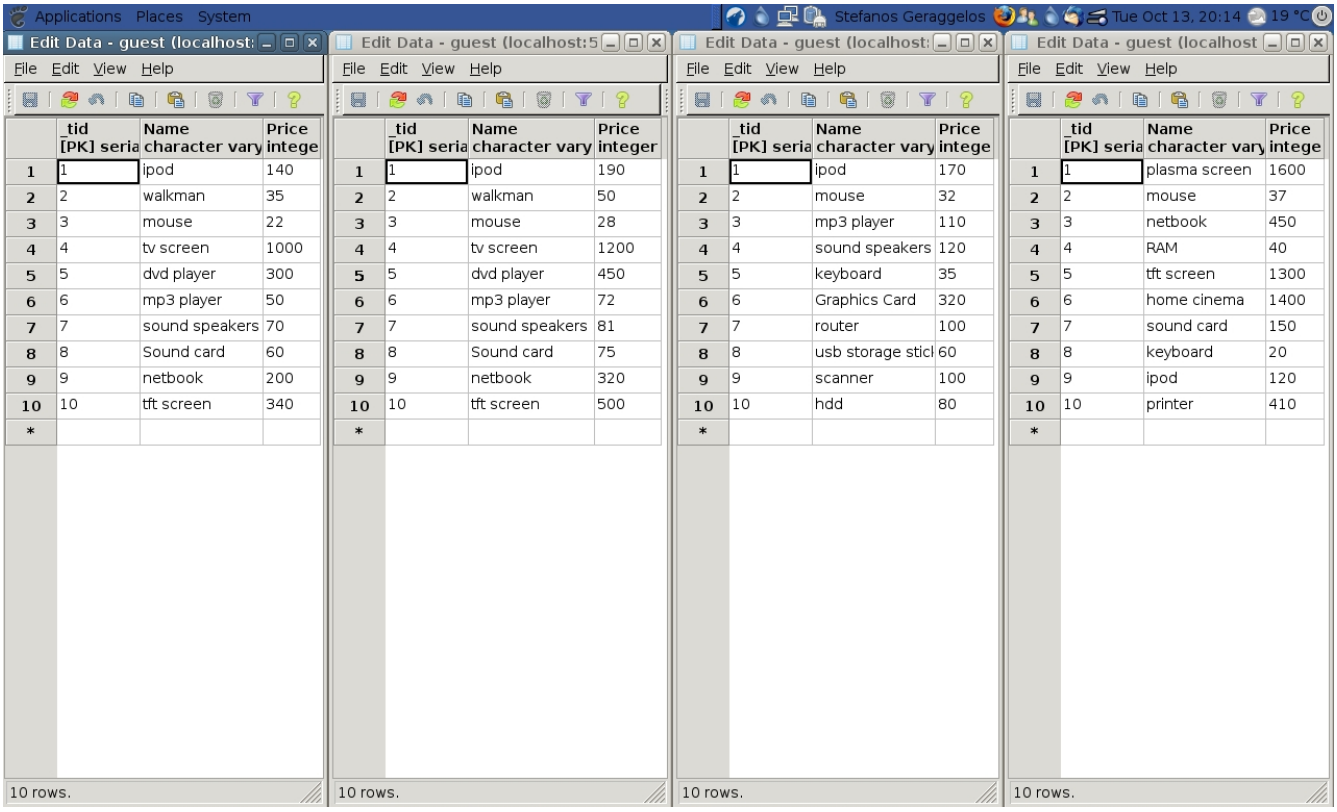
Το πρώτο ερώτημα που εκτελέστηκε στην context-aware βάση δεδομένων που περιγράψαμε στην προηγούμενη ενότητα, αποτελείται από έναν project τελεστή πάνω στο polymorph Product. Η εκτέλεση του ερωτήματος 1 έχει ως αποτέλεσμα την προβολή του ονόματος και της τιμής όλων των προϊόντων. Το πλάνο εκτέλεσης του συγκεκριμένου ερωτήματος παρουσιάζεται στο Σχήμα 5.4.

Επίσης, στο Σχήμα 5.5 παρουσιάζουμε το αποτέλεσμα αυτού του ερωτήματος για την υλοποίηση A, ενδεικτικά για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008>, <Retail, UK, 2009> και <Retail, USA, 2008> με αντίστοιχα context id's 1,2,10 και 14. Ο κάθε ένας από τους 4 πίνακες που παρουσιάζεται αντιστοιχεί σε ένα από τα προαναφερθέντα contexts. Για την υλοποίηση B, παρουσιάζεται ένα screenshot στο Σχήμα 5.6 για το μοναδικό πίνακα που παράγει το ερώτημα 1 και είναι σκιασμένες οι tuples που αναφέρεται στο context <Wholesale, Greece, 2008>.



Σχήμα 5.4. Σχηματική απεικόνιση του πλάνου εκτέλεσης για το ερώτημα 1

Τέλος, στο Σχήμα 5.7 φαίνεται το output στο τερματικό (terminal) που παρήγαγε η εκτέλεση του πειράματος για το ερώτημα 1 για την υλοποίηση A. Στο Σχήμα 5.8 φαίνονται οι αντίστοιχοι μέσοι χρόνοι και για την υλοποίηση B. Θα παρουσιάσουμε αυτά τα outputs ενδεικτικά για κάποια ερωτήματα, ενώ για τα υπόλοιπα θα αναφερθούμε στους μέσους χρόνους ερωτημάτων και επιμέρους τελεστών, χωρίς την παρουσίαση ολόκληρου του output στο terminal.



_tid [PK] serial | Name character varying | Price integer

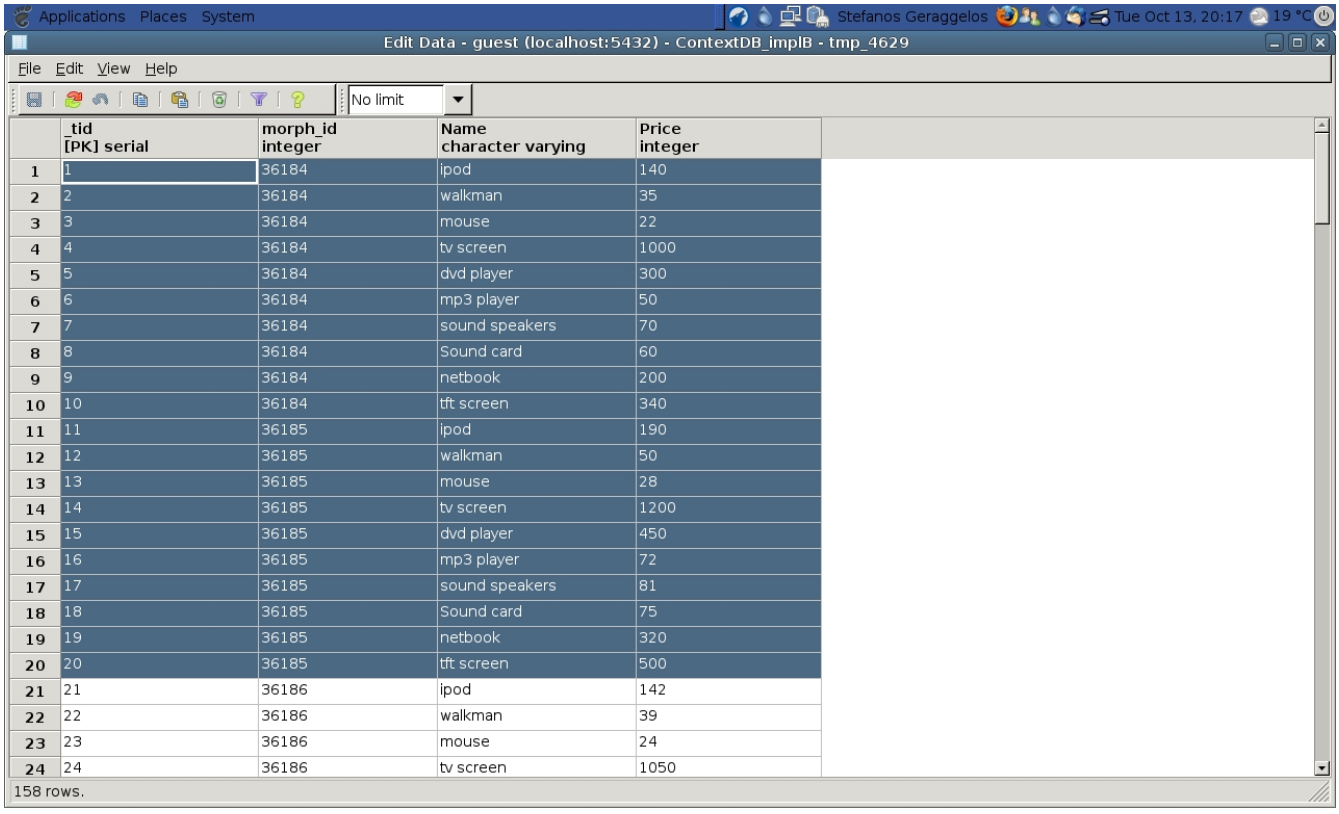
	_tid [PK] serial	Name character varying	Price integer
1	1	ipod	140
2	2	walkman	35
3	3	mouse	22
4	4	tv screen	1000
5	5	dvd player	300
6	6	mp3 player	50
7	7	sound speakers	70
8	8	Sound card	60
9	9	netbook	200
10	10	tft screen	340

	_tid [PK] serial	Name character varying	Price integer
1	1	ipod	190
2	2	walkman	50
3	3	mouse	28
4	4	tv screen	1200
5	5	dvd player	450
6	6	mp3 player	72
7	7	sound speakers	81
8	8	Sound card	75
9	9	netbook	320
10	10	tft screen	500

	_tid [PK] serial	Name character varying	Price integer
1	1	ipod	170
2	2	mouse	32
3	3	mp3 player	110
4	4	sound speakers	120
5	5	keyboard	35
6	6	Graphics Card	320
7	7	router	100
8	8	usb storage stic	60
9	9	scanner	100
10	10	hdd	80

	_tid [PK] serial	Name character varying	Price integer
1	1	plasma screen	1600
2	2	mouse	37
3	3	netbook	450
4	4	RAM	40
5	5	tft screen	1300
6	6	home cinema	1400
7	7	sound card	150
8	8	keyboard	20
9	9	ipod	120
10	10	printer	410

Σχήμα 5.5. Αποτέλεσμα της εκτέλεσης του ερωτήματος 1 για την υλοποίηση A και για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008>, <Retail, UK, 2009> και <Retail, USA, 2008> αντίστοιχα



Edit Data - guest (localhost:5432) - ContextDB_implB - tmp_4629

_tid [PK] serial | morph_id integer | Name character varying | Price integer

	_tid [PK] serial	morph_id integer	Name character varying	Price integer
1	1	36184	ipod	140
2	2	36184	walkman	35
3	3	36184	mouse	22
4	4	36184	tv screen	1000
5	5	36184	dvd player	300
6	6	36184	mp3 player	50
7	7	36184	sound speakers	70
8	8	36184	Sound card	60
9	9	36184	netbook	200
10	10	36184	tft screen	340
11	11	36185	ipod	190
12	12	36185	walkman	50
13	13	36185	mouse	28
14	14	36185	tv screen	1200
15	15	36185	dvd player	450
16	16	36185	mp3 player	72
17	17	36185	sound speakers	81
18	18	36185	Sound card	75
19	19	36185	netbook	320
20	20	36185	tft screen	500
21	21	36186	ipod	142
22	22	36186	walkman	39
23	23	36186	mouse	24
24	24	36186	tv screen	1050

Σχήμα 5.6. Αποτέλεσμα της εκτέλεσης του ερωτήματος 1 για την υλοποίηση B

(σκιασμένες είναι οι tuples που αναφέρονται στα contexts <Wholesale, Greece, 2008> και Retail, Greece, 2008> αντίστοιχα)

```
                              TESTING execute().............

TIME OF PROJECT OPERATOR EXECUTION: 0.640 seconds

                              .............execute()---->OK

TOTAL TIME OF PLAN1 EXECUTION (attempt No.1):   0.950 seconds


                              TESTING execute().............

TIME OF PROJECT OPERATOR EXECUTION: 0.608 seconds

                              .............execute()---->OK

TOTAL TIME OF PLAN1 EXECUTION (attempt No.2):   0.917 seconds


                              TESTING execute().............

TIME OF PROJECT OPERATOR EXECUTION: 0.575 seconds

                              .............execute()---->OK

TOTAL TIME OF PLAN1 EXECUTION (attempt No.3):   0.885 seconds


                              TESTING execute().............

TIME OF PROJECT OPERATOR EXECUTION: 0.589 seconds

                              .............execute()---->OK

TOTAL TIME OF PLAN1 EXECUTION (attempt No.4):   0.902 seconds


                              TESTING execute().............

TIME OF PROJECT OPERATOR EXECUTION: 0.593 seconds

                              .............execute()---->OK
```

```
TOTAL TIME OF PLAN1 EXECUTION (attempt No.5):   0.926 seconds

                              TESTING execute().............

TIME OF PROJECT OPERATOR EXECUTION: 0.654 seconds
                              .............execute()---->OK

TOTAL TIME OF PLAN1 EXECUTION (attempt No.6):   0.981 seconds

                              TESTING execute().............

TIME OF PROJECT OPERATOR EXECUTION: 0.589 seconds
                              .............execute()---->OK

TOTAL TIME OF PLAN1 EXECUTION (attempt No.7):   0.911 seconds

                              TESTING execute().............

TIME OF PROJECT OPERATOR EXECUTION: 0.590 seconds
                              .............execute()---->OK

TOTAL TIME OF PLAN1 EXECUTION (attempt No.8):   0.905 seconds

                              TESTING execute().............

TIME OF PROJECT OPERATOR EXECUTION: 0.630 seconds
                              .............execute()---->OK

TOTAL TIME OF PLAN1 EXECUTION (attempt No.9):   0.977 seconds

                              TESTING execute().............

TIME OF PROJECT OPERATOR EXECUTION: 0.611 seconds
                              .............execute()---->OK

TOTAL TIME OF PLAN1 EXECUTION (attempt No.10):  0.933 seconds
```

```
AVERAGE TOTAL TIME OF PLAN1 EXECUTION:    0.929 seconds


*********************************************
* Plan1 completed as expected!!!             *
*********************************************
```

Σχήμα 5.7.Output εκτέλεσης του ερωτήματος 1 για την υλοποίηση A

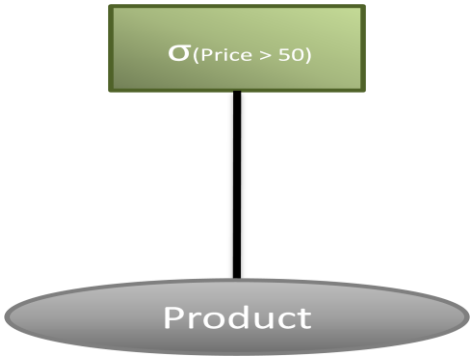
Ερώτημα/Τελεστής	Μέσος χρόνος εκτέλεσης για την υλοποίηση A (seconds)	Μέσος χρόνος εκτέλεσης για την υλοποίηση B (seconds)
Project	0.608	0.304
Ερώτημα 1	0.929	0.625

Σχήμα 5.8. Μέσοι χρόνοι εκτέλεσης για το ερώτημα 1

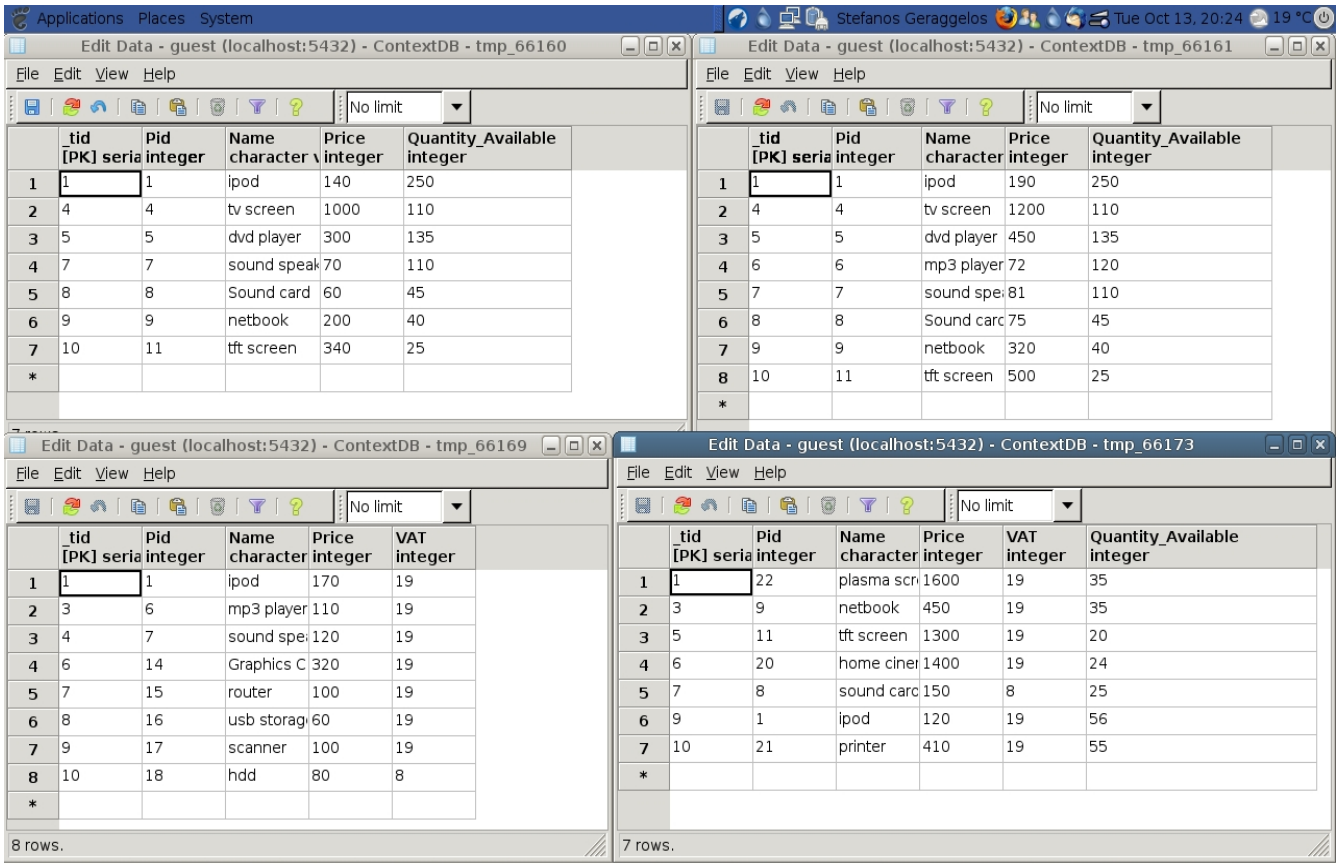
Παρατηρούμε από τον πίνακα του Σχήματος 5.8 ότι για το συγκεκριμένο ερώτημα η υλοποίηση B είναι πιο γρήγορη κατά ~50% από την υλοποίηση A σε ολόκληρο το ερώτημα, ενώ στην εκτέλεση του project είναι πιο γρήγορη κατά 33%. Κάτι τέτοιο αναμενόταν, καθώς στην υλοποίηση B ένα project είναι ουσιαστικά μία προβολή σε ένα μόνο πίνακα, ενώ στην υλοποίηση A απαιτούνται επιμέρους προβολές σε περισσότερους πίνακες.

5.2.2 Ερώτημα 2

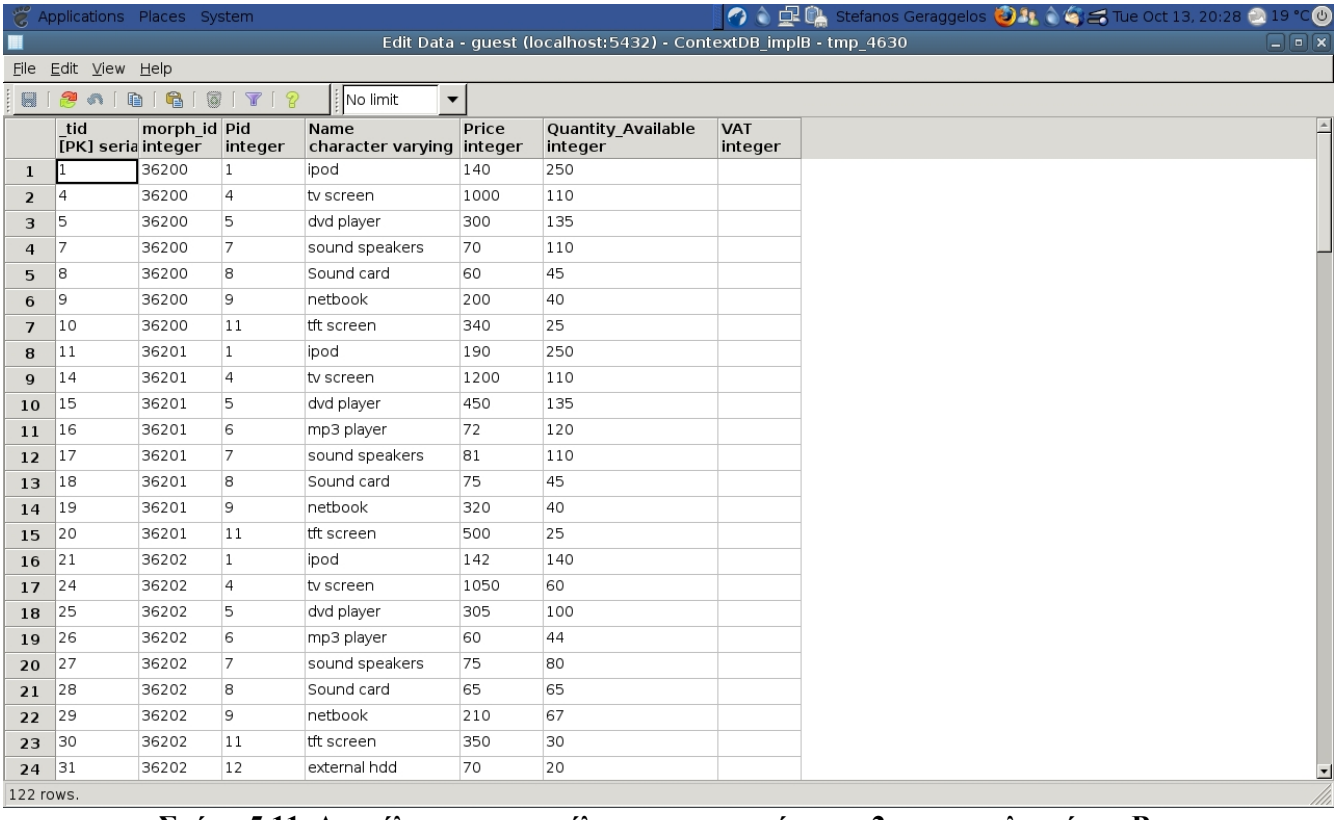
Το ερώτημα 2 αποτελείται, επίσης, από ένα μόνο τελεστή, ένα select στο polymorph Product. Το ερώτημα επιστρέφει εκείνα τα προϊόντα που έχουν τιμή πάνω από 50. Ακολουθούν τα Σχήματα 5.9-5.12, όπου παρουσιάζονται το πλάνο του ερωτήματος, τα αποτελέσματα εκτέλεσης αυτού και οι μέσοι χρόνοι αντίστοιχα.



Σχήμα 5.9. Σχηματική απεικόνιση του πλάνου εκτέλεσης για το ερώτημα 2



Σχήμα 5.10. Αποτέλεσμα της εκτέλεσης του ερωτήματος 2 για την υλοποίηση A και για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008>, <Retail, UK, 2009> και <Retail, USA, 2008> αντίστοιχα



Σχήμα 5.11. Αποτέλεσμα της εκτέλεσης του ερωτήματος 2 για την υλοποίηση B

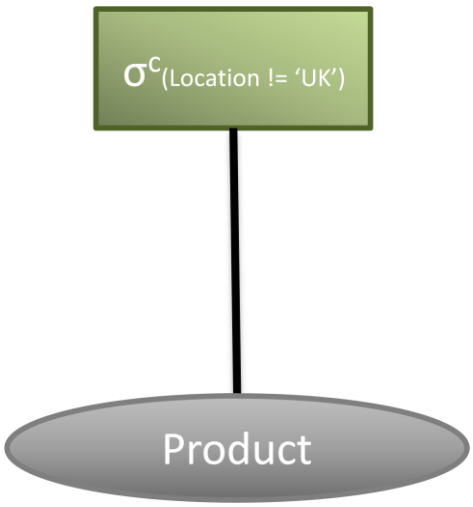
Ερώτημα/Τελεστής	Μέσος χρόνος εκτέλεσης για την υλοποίηση A (seconds)	Μέσος χρόνος εκτέλεσης για την υλοποίηση B (seconds)
Select	0.748	0.444
Ερώτημα 2	1.043	0.731

Σχήμα 5.12. Μέσοι χρόνοι εκτέλεσης για το ερώτημα 2

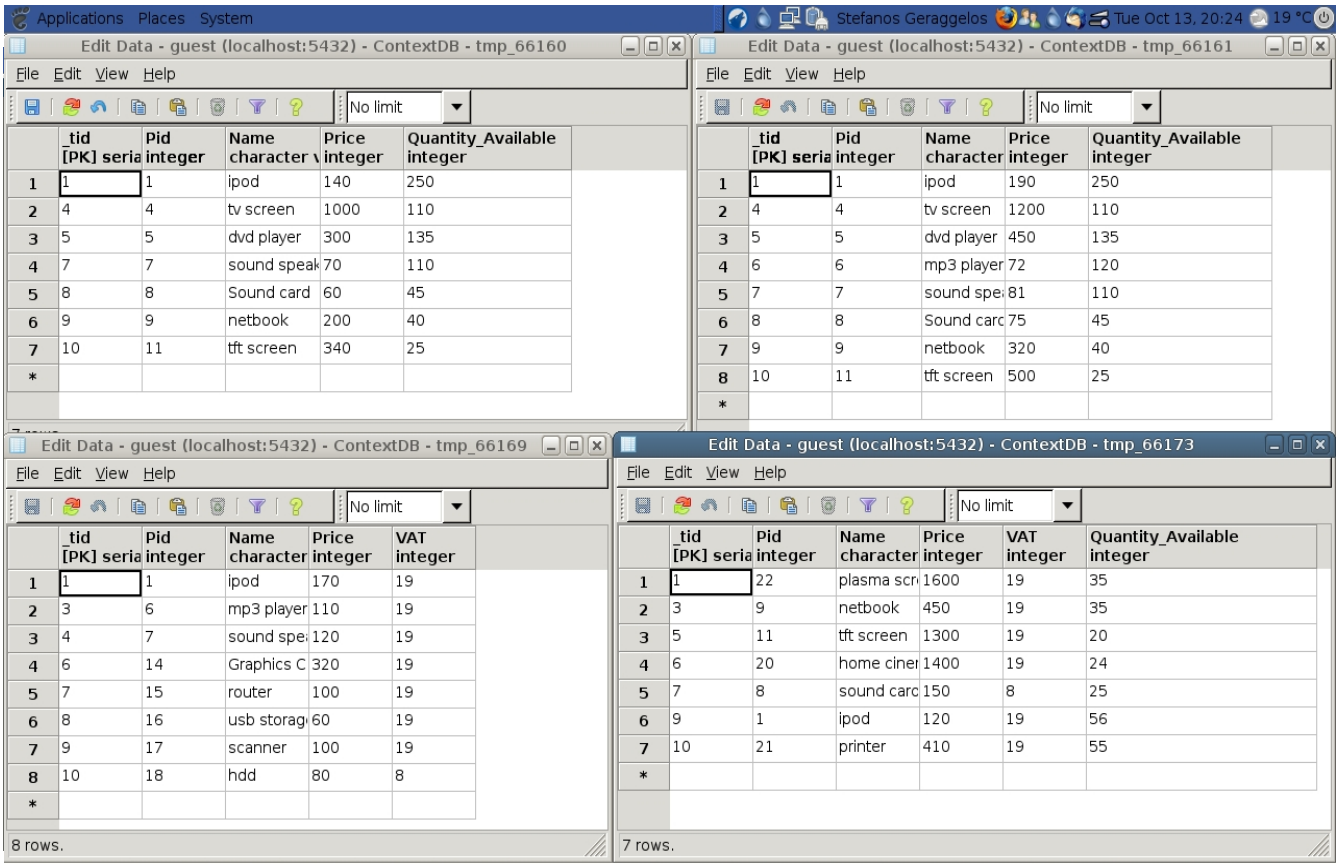
Παρατηρούμε από τον πίνακα του Σχήματος 5.12, όπως και στο ερώτημα 1, ότι για το συγκεκριμένο ερώτημα η υλοποίηση B είναι πιο γρήγορη κατά ~30% από την υλοποίηση A σε ολόκληρο το ερώτημα, ενώ στην εκτέλεση του select είναι πιο γρήγορη κατά ~41%. Ομοίως, κάτι τέτοιο αναμενόταν, καθώς στην υλοποίηση B ένα select είναι ουσιαστικά μία επιλογή σε ένα μόνο πίνακα, ενώ στην υλοποίηση A απαιτούνται επιμέρους επιλογές σε περισσότερους πίνακες.

5.2.3 Ερώτημα 3

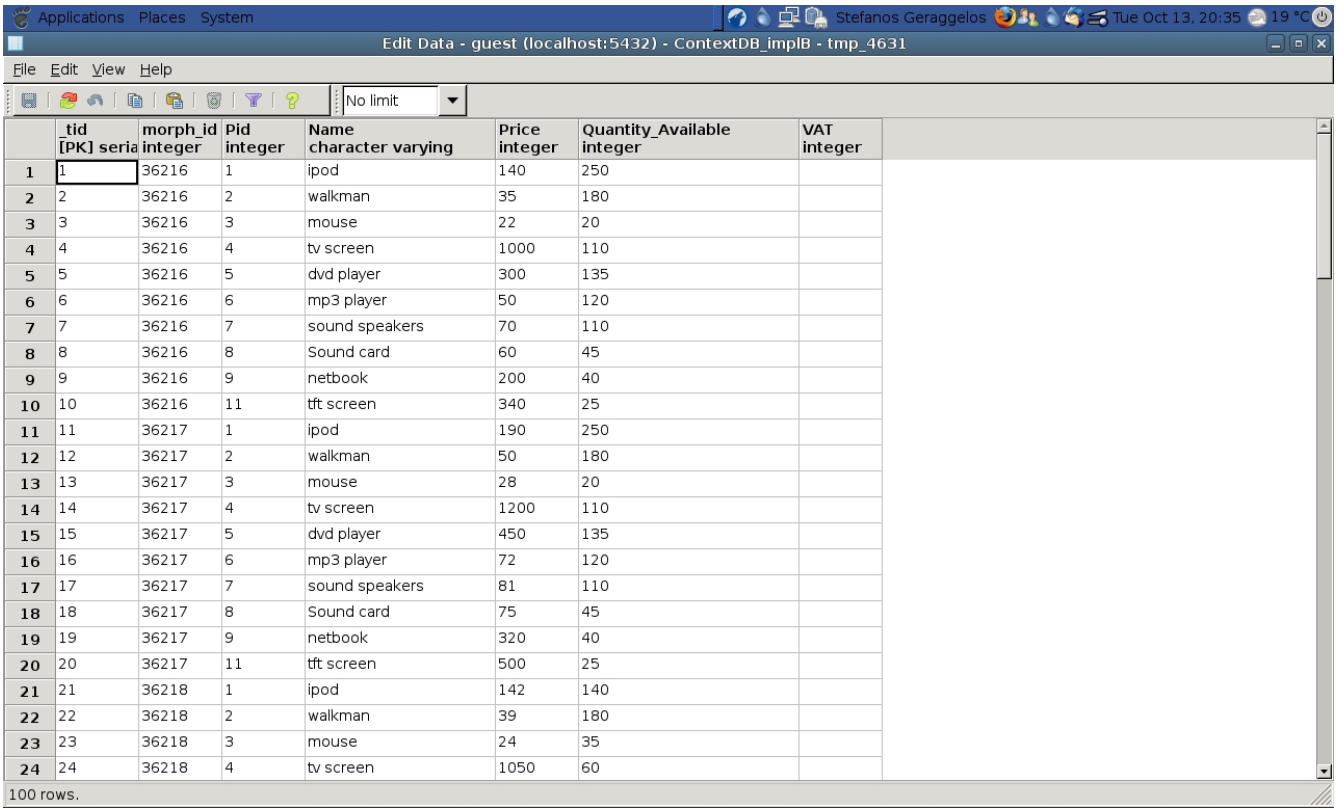
Το ερώτημα 3 αποτελείται, επίσης, από ένα μόνο τελεστή, ένα select schema στο polymorph Product. Συγκεκριμένα, το ερώτημα επιστρέφει εκείνα τα προϊόντα που είναι διαθέσιμα σε όλες τις χώρες εκτός από το Ηνωμένο Βασίλειο. Ακολουθούν τα Σχήματα 5.13-5.16, όπου παρουσιάζονται το πλάνο του ερωτήματος, τα αποτελέσματα εκτέλεσης αυτού και οι μέσοι χρόνοι αντίστοιχα. Στο Σχήμα 5.14 παρουσιάζονται τα αποτελέσματα της υλοποίησης A για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008> και <Retail, USA, 2008>. Το αποτέλεσμα για το context <Retail, UK, 2009> δεν υπάρχει, καθώς το αντίστοιχο morph δε συμπεριλαμβάνεται στο αποτέλεσμα της εκτέλεσης του ερωτήματος 3.



Σχήμα 5.13. Σχηματική απεικόνιση του πλάνου εκτέλεσης για το ερώτημα 3



Σχήμα 5.14. Αποτέλεσμα της εκτέλεσης του ερωτήματος 3 για την υλοποίηση A και για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008> και <Retail, USA, 2008> αντίστοιχα



Σχήμα 5.15. Αποτέλεσμα της εκτέλεσης του ερωτήματος 3 για την υλοποίηση B

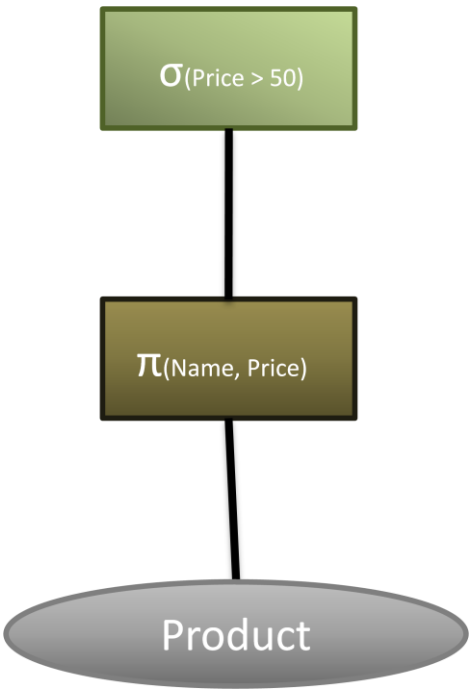
Ερώτημα/Τελεστής	Μέσος χρόνος εκτέλεσης για την υλοποίηση A (seconds)	Μέσος χρόνος εκτέλεσης για την υλοποίηση B (seconds)
Select Schema	0.493	0.303
Ερώτημα 3	0.817	0.626

Σχήμα 5.16. Μέσοι χρόνοι εκτέλεσης για το ερώτημα 3

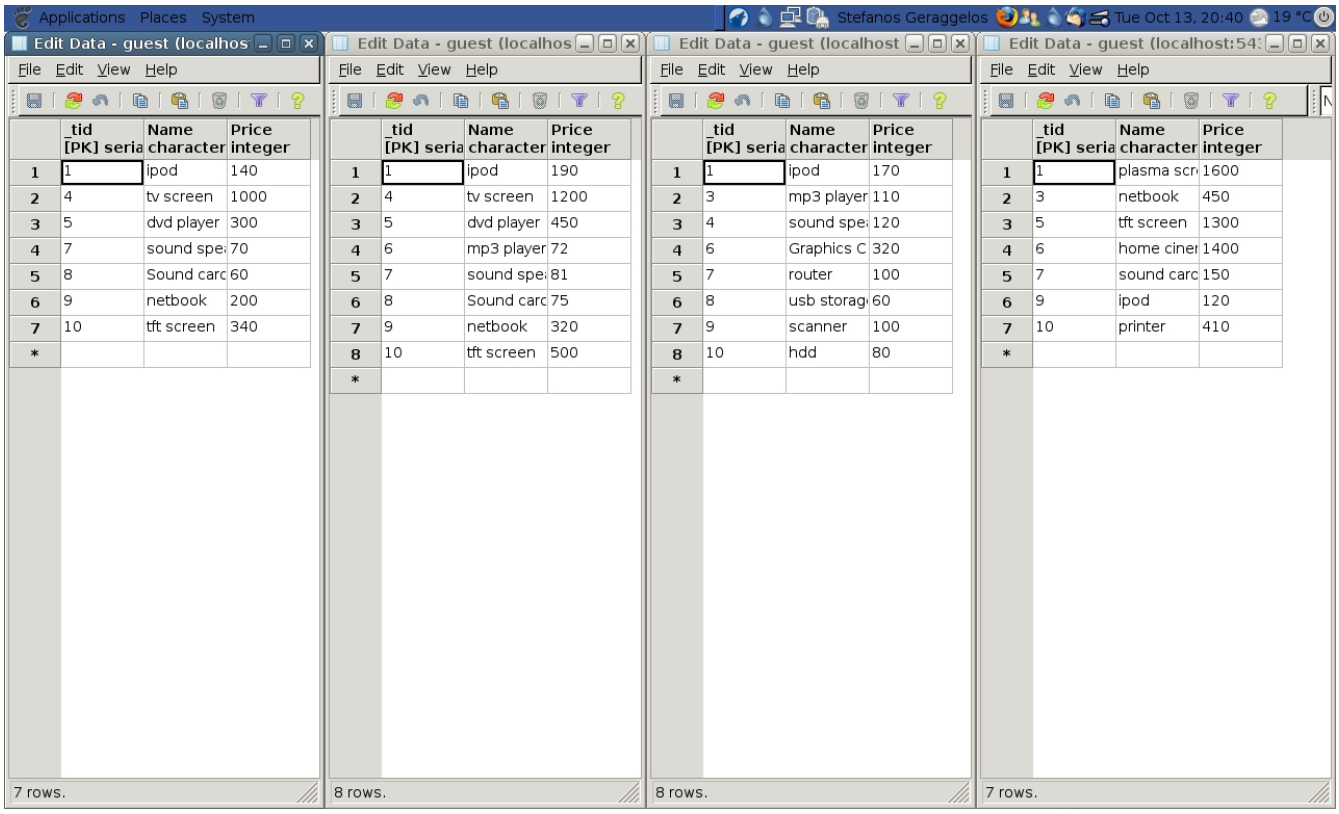
Παρατηρούμε από τον πίνακα του Σχήματος 5.16 ότι για το συγκεκριμένο ερώτημα η υλοποίηση B είναι πιο γρήγορη κατά ~23% από την υλοποίηση A σε ολόκληρο το ερώτημα, ενώ στην εκτέλεση του select είναι πιο γρήγορη κατά ~39%.

5.2.4 Ερώτημα 4

Το ερώτημα 4 αποτελείται από δύο τελεστές, έναν project στο polymorph Product και ένα select στο αποτέλεσμα αυτό. Έτσι, το τελικό αποτέλεσμα παρουσιάζει εκείνα τα ονόματα και τις τιμές εκείνων των προϊόντων, που έχουν τιμή μεγαλύτερη του 50. Ακολουθούν τα Σχήματα 5.17-5.20, όπου παρουσιάζονται το πλάνο του ερωτήματος, τα αποτελέσματα εκτέλεσης αυτού και οι μέσοι χρόνοι αντίστοιχα.



Σχήμα 5.17. Σχηματική απεικόνιση του πλάνου εκτέλεσης για το ερώτημα 4



Edit Data - guest (localhos

File Edit View Help

	_tid [PK] seria	Name character	Price integer
1	1	ipod	140
2	4	tv screen	1000
3	5	dvd player	300
4	7	sound spe	70
5	8	Sound carc	60
6	9	netbook	200
7	10	tft screen	340
*			

7 rows.

Edit Data - guest (localhos

File Edit View Help

	_tid [PK] seria	Name character	Price integer
1	1	ipod	190
2	4	tv screen	1200
3	5	dvd player	450
4	6	mp3 player	72
5	7	sound spe	81
6	8	Sound carc	75
7	9	netbook	320
8	10	tft screen	500
*			

8 rows.

Edit Data - guest (localhost

File Edit View Help

	_tid [PK] seria	Name character	Price integer
1	1	ipod	170
2	3	mp3 player	110
3	4	sound spe	120
4	6	Graphics C	320
5	7	router	100
6	8	usb storag	60
7	9	scanner	100
8	10	hdd	80
*			

8 rows.

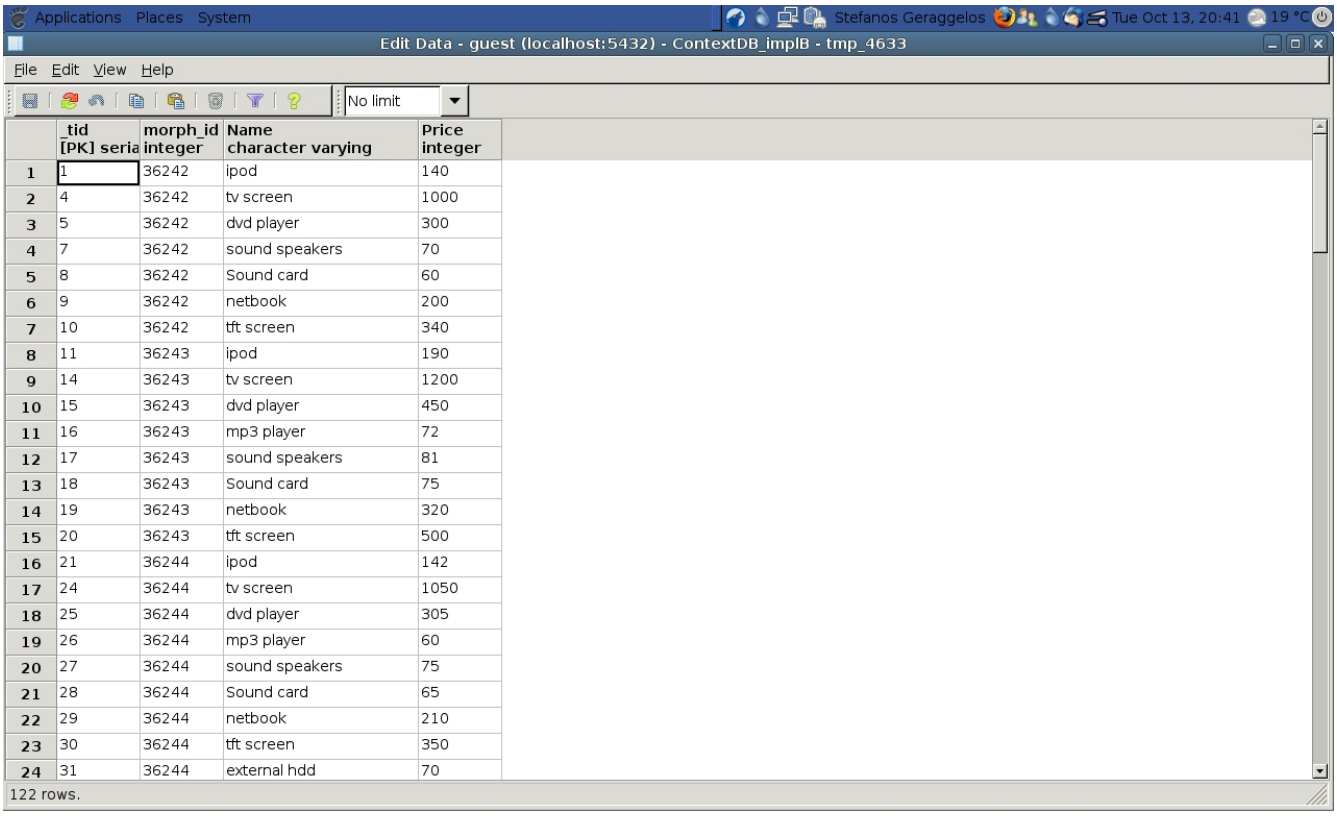
Edit Data - guest (localhost:54

File Edit View Help

	_tid [PK] seria	Name character	Price integer
1	1	plasma scr	1600
2	3	netbook	450
3	5	tft screen	1300
4	6	home ciner	1400
5	7	sound carc	150
6	9	ipod	120
7	10	printer	410
*			

7 rows.

Σχήμα 5.18. Αποτέλεσμα της εκτέλεσης του ερωτήματος 4 για την υλοποίηση A και για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008>, <Retail, UK, 2009> και <Retail, USA, 2008> αντίστοιχα



Edit Data - guest (localhost:5432) - ContextDB_implB - tmp_4633

File Edit View Help

No limit

	_tid [PK] seria	morph_id integer	Name character varying	Price integer
1	1	36242	ipod	140
2	4	36242	tv screen	1000
3	5	36242	dvd player	300
4	7	36242	sound speakers	70
5	8	36242	Sound card	60
6	9	36242	netbook	200
7	10	36242	tft screen	340
8	11	36243	ipod	190
9	14	36243	tv screen	1200
10	15	36243	dvd player	450
11	16	36243	mp3 player	72
12	17	36243	sound speakers	81
13	18	36243	Sound card	75
14	19	36243	netbook	320
15	20	36243	tft screen	500
16	21	36244	ipod	142
17	24	36244	tv screen	1050
18	25	36244	dvd player	305
19	26	36244	mp3 player	60
20	27	36244	sound speakers	75
21	28	36244	Sound card	65
22	29	36244	netbook	210
23	30	36244	tft screen	350
24	31	36244	external hdd	70

122 rows.

Σχήμα 5.19. Αποτέλεσμα της εκτέλεσης του ερωτήματος 4 για την υλοποίηση B

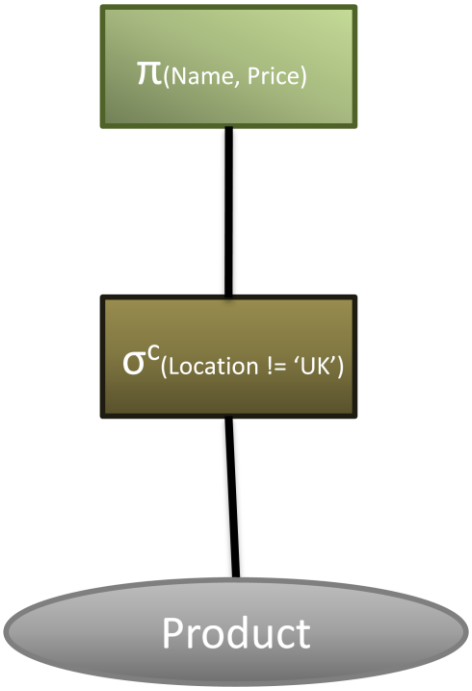
Ερώτημα/Τελεστής	Μέσος χρόνος εκτέλεσης για την υλοποίηση A (seconds)	Μέσος χρόνος εκτέλεσης για την υλοποίηση B (seconds)
Project	0.600	0.313
Select	0.664	0.322
Ερώτημα 4	1.691	1.061

Σχήμα 5.20. Μέσοι χρόνοι εκτέλεσης για το ερώτημα 4

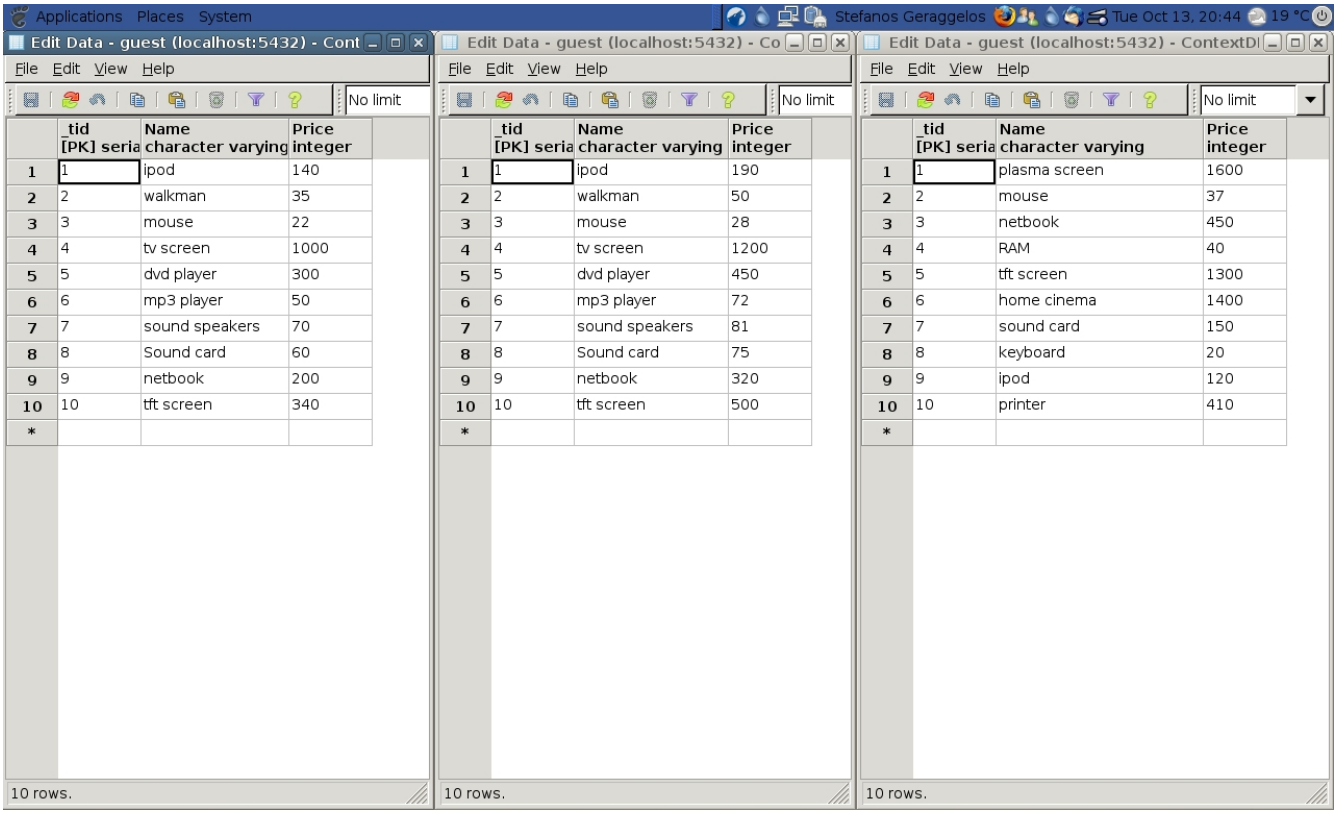
Παρατηρούμε από τον πίνακα του Σχήματος 5.20 ότι για το συγκεκριμένο ερώτημα η υλοποίηση B είναι πιο γρήγορη κατά ~37% από την υλοποίηση A σε ολόκληρο το ερώτημα, ενώ στην εκτέλεση του project είναι πιο γρήγορη κατά ~48% και στην εκτέλεση του select είναι πιο γρήγορη κατά ~52%.

5.2.5 Ερώτημα 5

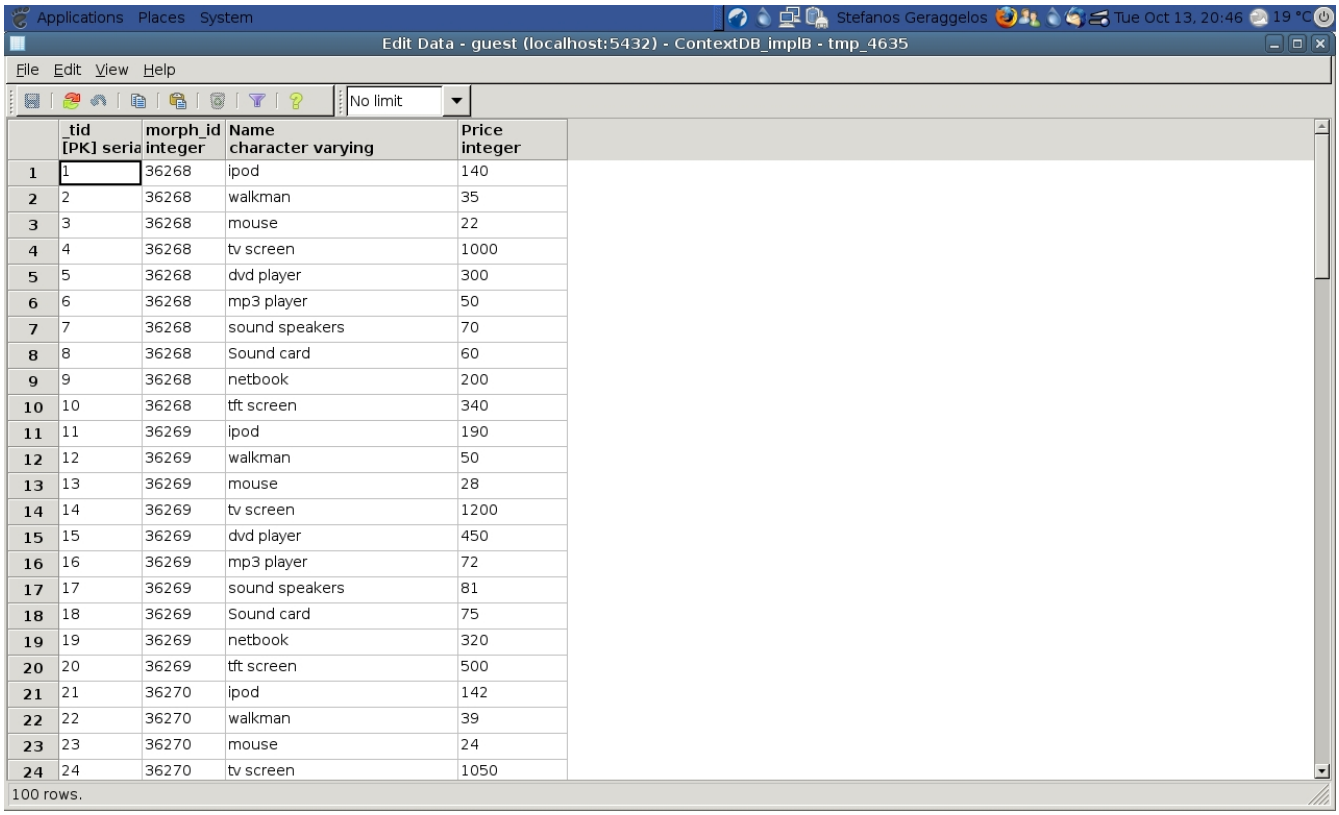
Το ερώτημα 5 αποτελείται από δύο τελεστές, έναν project και ένα select schema στο polymorph Product. Έτσι, το τελικό αποτέλεσμα παρουσιάζει τα ονόματα και τις τιμές εκείνων των προϊόντων, που είναι διαθέσιμα σε όλες τις χώρες εκτός από το Ηνωμένο Βασίλειο. Ακολουθούν τα Σχήματα 5.21-5.24, όπου παρουσιάζονται το πλάνο του ερωτήματος, τα αποτελέσματα εκτέλεσης αυτού και οι μέσοι χρόνοι αντίστοιχα. Ομοίως με το ερώτημα 3, στο Σχήμα 5.22 παρουσιάζονται τα αποτελέσματα της υλοποίησης A για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008> και <Retail, USA, 2008> και όχι για το context <Retail, UK, 2009>, που δεν υπάρχει στο αποτέλεσμα.



Σχήμα 5.21. Σχηματική απεικόνιση του πλάνου εκτέλεσης για το ερώτημα 5



Σχήμα 5.22. Αποτέλεσμα της εκτέλεσης του ερωτήματος 5 για την υλοποίηση A και για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008> και <Retail, USA, 2008> αντίστοιχα



	_tid [PK] seria	morph_id integer	Name character varying	Price integer
1	1	36268	ipod	140
2	2	36268	walkman	35
3	3	36268	mouse	22
4	4	36268	tv screen	1000
5	5	36268	dvd player	300
6	6	36268	mp3 player	50
7	7	36268	sound speakers	70
8	8	36268	Sound card	60
9	9	36268	netbook	200
10	10	36268	tft screen	340
11	11	36269	ipod	190
12	12	36269	walkman	50
13	13	36269	mouse	28
14	14	36269	tv screen	1200
15	15	36269	dvd player	450
16	16	36269	mp3 player	72
17	17	36269	sound speakers	81
18	18	36269	Sound card	75
19	19	36269	netbook	320
20	20	36269	tft screen	500
21	21	36270	ipod	142
22	22	36270	walkman	39
23	23	36270	mouse	24
24	24	36270	tv screen	1050

100 rows.

Σχήμα 5.23. Αποτέλεσμα της εκτέλεσης του ερωτήματος 5 για την υλοποίηση B

Ερώτημα/Τελεστής	Μέσος χρόνος εκτέλεσης για την υλοποίηση A (seconds)	Μέσος χρόνος εκτέλεσης για την υλοποίηση B (seconds)
Select Schema	0.495	0.316
Project	0.411	0.217
Ερώτημα 5	1.365	0.991

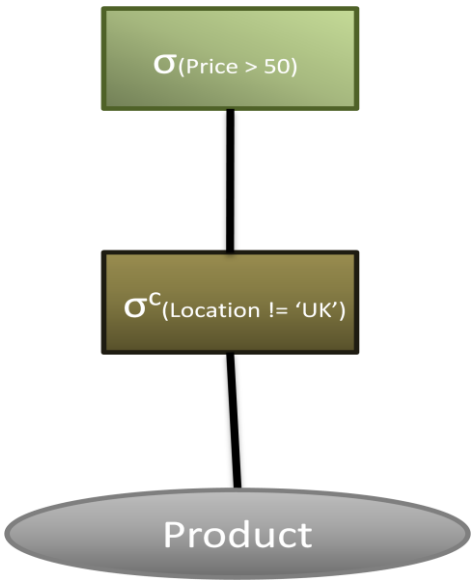
Σχήμα 5.24. Μέσοι χρόνοι εκτέλεσης για το ερώτημα 5

Παρατηρούμε από τον πίνακα του Σχήματος 5.24 ότι για το συγκεκριμένο ερώτημα η υλοποίηση B είναι πιο γρήγορη κατά ~27% από την υλοποίηση A σε ολόκληρο το ερώτημα, ενώ στην εκτέλεση του select schema είναι πιο γρήγορη κατά ~36% και στην εκτέλεση του project είναι πιο γρήγορη κατά ~47%.

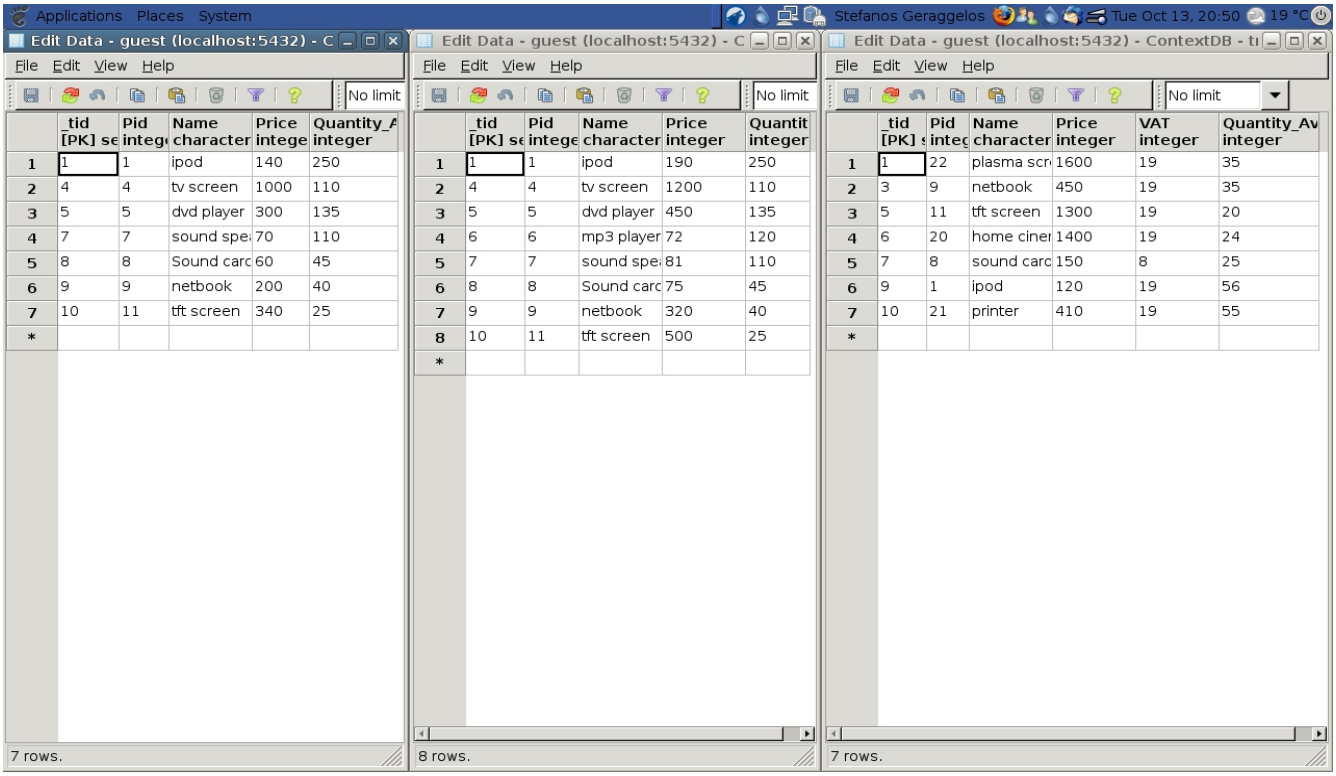
5.2.6 Ερώτημα 6

Το ερώτημα 6 αποτελείται από δύο τελεστές, ένα select και ένα select schema στο polymorph Product. Έτσι, το τελικό αποτέλεσμα παρουσιάζει εκείνα τα προϊόντα, που είναι διαθέσιμα σε όλες τις χώρες εκτός από το Ηνωμένο Βασίλειο και έχουν τιμές μεγαλύτερες από 50. Ακολουθούν τα Σχήματα 5.25-5.28, όπου παρουσιάζονται το πλάνο του ερωτήματος, τα αποτελέσματα εκτέλεσης αυτού και οι μέσοι χρόνοι αντίστοιχα. Ομοίως με το ερώτημα 3,

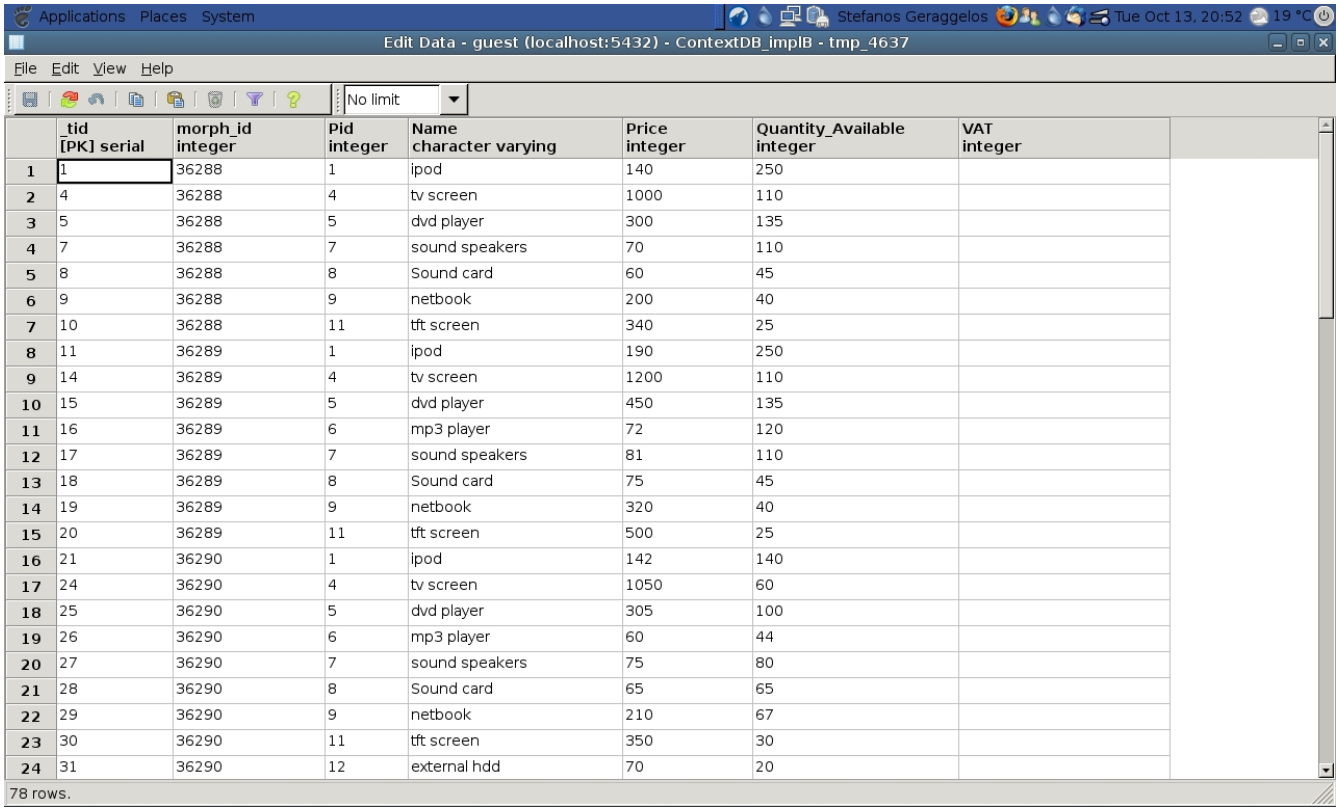
στο Σχήμα 5.26 παρουσιάζονται τα αποτελέσματα της υλοποίησης A για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008> και <Retail, USA, 2008> και όχι για το context <Retail, UK, 2009>, που δεν υπάρχει στο αποτέλεσμα.



Σχήμα 5.25. Σχηματική απεικόνιση του πλάνου εκτέλεσης για το ερώτημα 6



Σχήμα 5.26. Αποτέλεσμα της εκτέλεσης του ερωτήματος 6 για την υλοποίηση A και για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008> και <Retail, USA, 2008> αντίστοιχα



	_tid [PK] serial	morph_id integer	Pid integer	Name character varying	Price integer	Quantity_Available integer	VAT integer
1	1	36288	1	ipod	140	250	
2	4	36288	4	tv screen	1000	110	
3	5	36288	5	dvd player	300	135	
4	7	36288	7	sound speakers	70	110	
5	8	36288	8	Sound card	60	45	
6	9	36288	9	netbook	200	40	
7	10	36288	11	tft screen	340	25	
8	11	36289	1	ipod	190	250	
9	14	36289	4	tv screen	1200	110	
10	15	36289	5	dvd player	450	135	
11	16	36289	6	mp3 player	72	120	
12	17	36289	7	sound speakers	81	110	
13	18	36289	8	Sound card	75	45	
14	19	36289	9	netbook	320	40	
15	20	36289	11	tft screen	500	25	
16	21	36290	1	ipod	142	140	
17	24	36290	4	tv screen	1050	60	
18	25	36290	5	dvd player	305	100	
19	26	36290	6	mp3 player	60	44	
20	27	36290	7	sound speakers	75	80	
21	28	36290	8	Sound card	65	65	
22	29	36290	9	netbook	210	67	
23	30	36290	11	tft screen	350	30	
24	31	36290	12	external hdd	70	20	

78 rows.

Σχήμα 5.27. Αποτέλεσμα της εκτέλεσης του ερωτήματος 6 για την υλοποίηση B

Ερώτημα/Τελεστής	Μέσος χρόνος εκτέλεσης για την υλοποίηση A (seconds)	Μέσος χρόνος εκτέλεσης για την υλοποίηση B (seconds)
Select Schema	0.511	0.315
Select	0.487	0.320
Ερώτημα 6	1.420	1.062

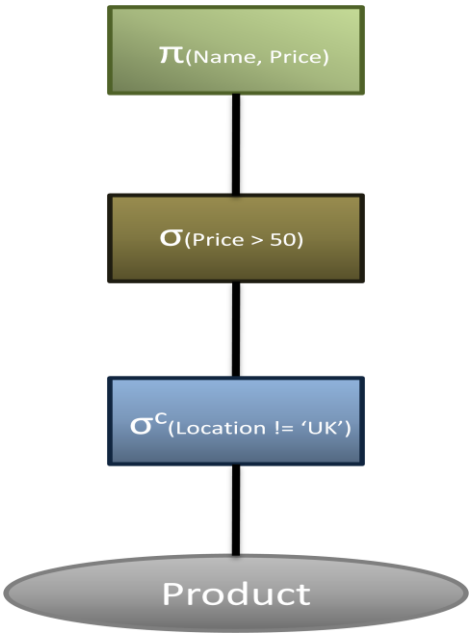
Σχήμα 5.28. Μέσοι χρόνοι εκτέλεσης για το ερώτημα 5

Παρατηρούμε από τον πίνακα του Σχήματος 5.28 ότι για το συγκεκριμένο ερώτημα η υλοποίηση B είναι πιο γρήγορη κατά ~25% από την υλοποίηση A σε ολόκληρο το ερώτημα, ενώ στην εκτέλεση του select schema είναι πιο γρήγορη κατά ~38% και στην εκτέλεση του select είναι πιο γρήγορη κατά ~34%.

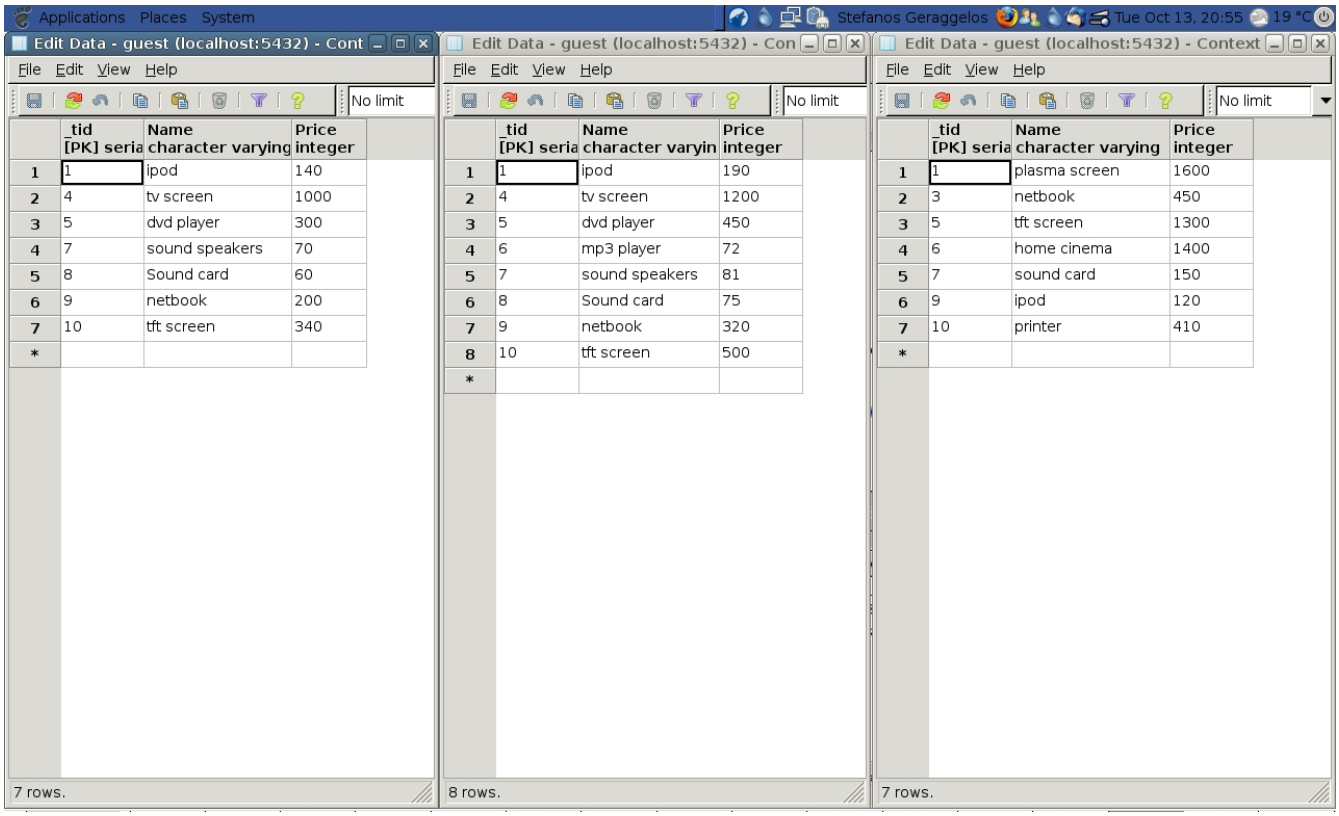
5.2.7 Ερώτημα 7

Το ερώτημα 7 αποτελείται από την εκτέλεση των ακόλουθων τελεστών, ενός project, ενός select και ενός select schema στο polymorph Product. Έτσι, το τελικό αποτέλεσμα παρουσιάζει τα ονόματα και τις τιμές εκείνων των προϊόντων, που είναι διαθέσιμα σε όλες τις χώρες εκτός από το Ηνωμένο Βασίλειο και έχουν τιμές μεγαλύτερες από 50. Ακολουθούν τα Σχήματα 5.29-5.32, όπου παρουσιάζονται το πλάνο του ερωτήματος, τα αποτελέσματα

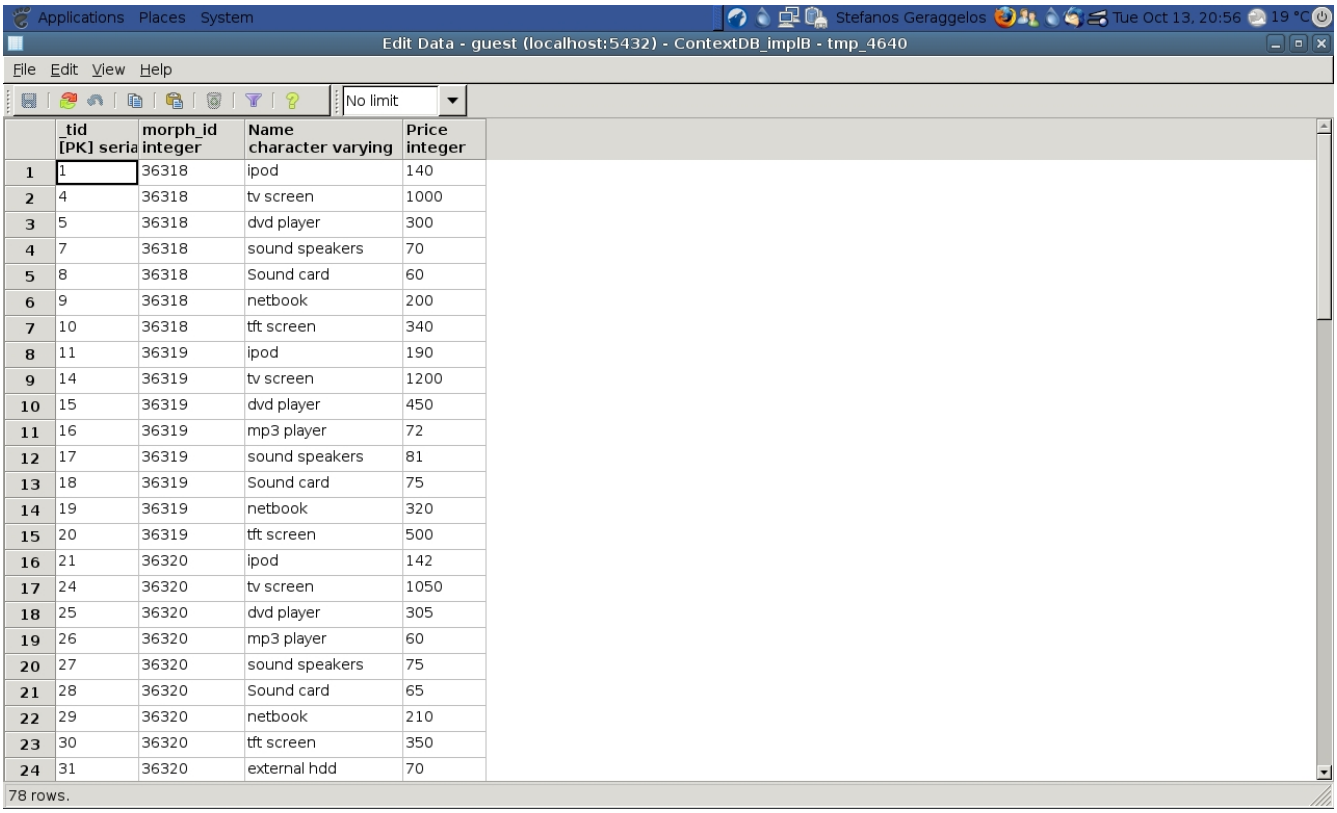
εκτέλεσης αυτού και οι μέσοι χρόνοι αντίστοιχα. Ομοίως με το ερώτημα 3, στο Σχήμα 5.30 παρουσιάζονται τα αποτελέσματα της υλοποίησης A για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008> και <Retail, USA, 2008> και όχι για το context <Retail, UK, 2009>, που δεν υπάρχει στο αποτέλεσμα.



Σχήμα 5.29. Σχηματική απεικόνιση του πλάνου εκτέλεσης για το ερώτημα 7



Σχήμα 5.30. Αποτέλεσμα της εκτέλεσης του ερωτήματος 6 για την υλοποίηση A και για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008> και <Retail, USA, 2008> αντίστοιχα



Σχήμα 5.31. Αποτέλεσμα της εκτέλεσης του ερωτήματος 7 για την υλοποίηση B

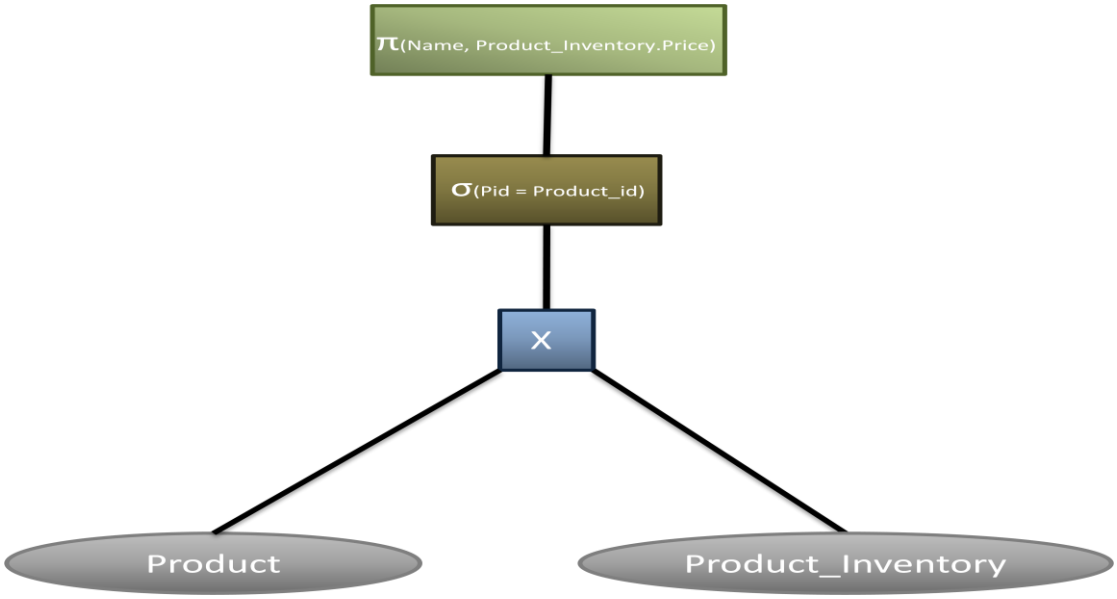
Ερώτημα/Τελεστής	Μέσος χρόνος εκτέλεσης για την υλοποίηση A (seconds)	Μέσος χρόνος εκτέλεσης για την υλοποίηση B (seconds)
Select Schema	0.512	0.309
Select	0.501	0.307
Project	0.461	0.218
Ερώτημα 7	2.039	1.394

Σχήμα 5.32. Μέσοι χρόνοι εκτέλεσης για το ερώτημα 7

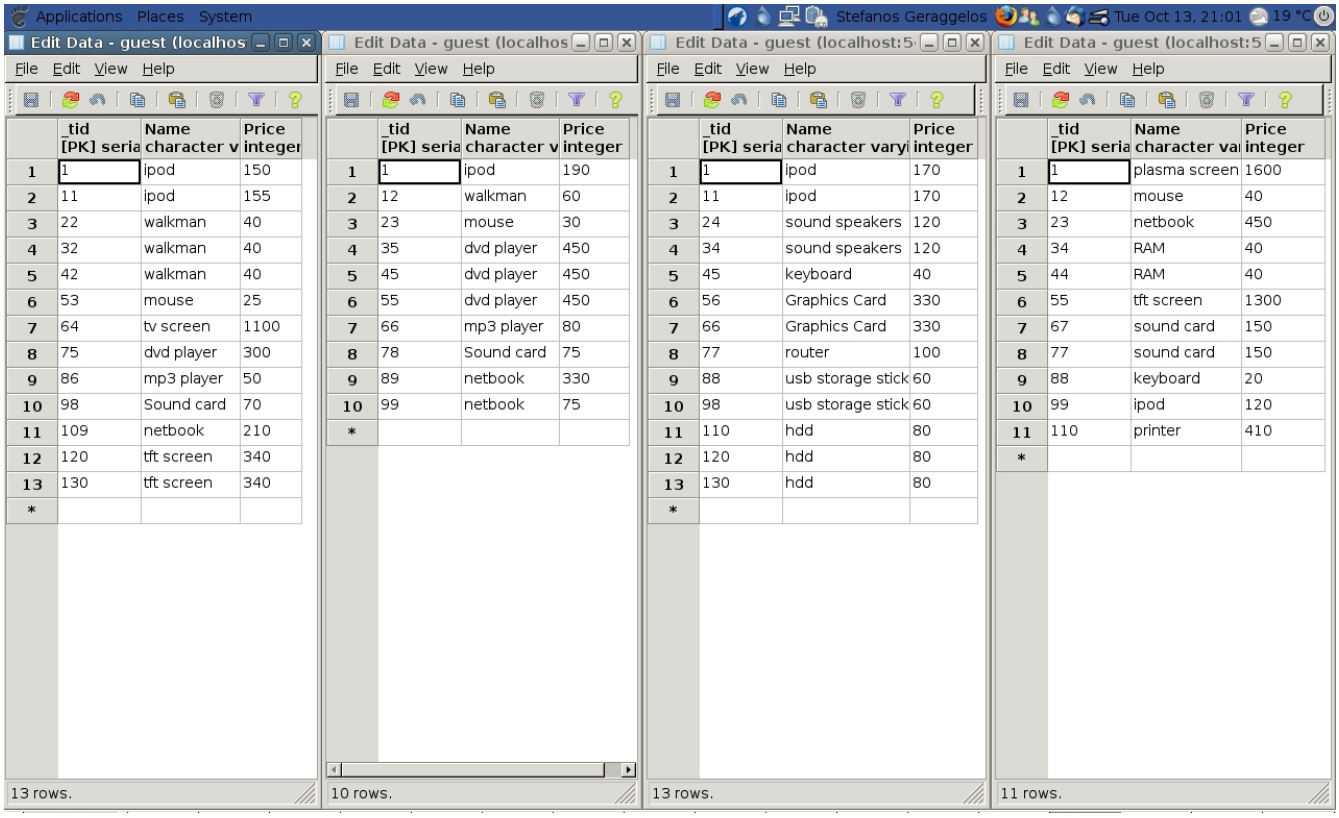
Παρατηρούμε από τον πίνακα του Σχήματος 5.32 ότι για το συγκεκριμένο ερώτημα η υλοποίηση B είναι πιο γρήγορη κατά ~32% από την υλοποίηση A σε ολόκληρο το ερώτημα, ενώ στην εκτέλεση του select schema είναι πιο γρήγορη κατά ~40%, στην εκτέλεση του select είναι πιο γρήγορη κατά ~39% και στην εκτέλεση του project είναι πιο γρήγορη κατά ~53%.

5.2.8 Ερώτημα 8

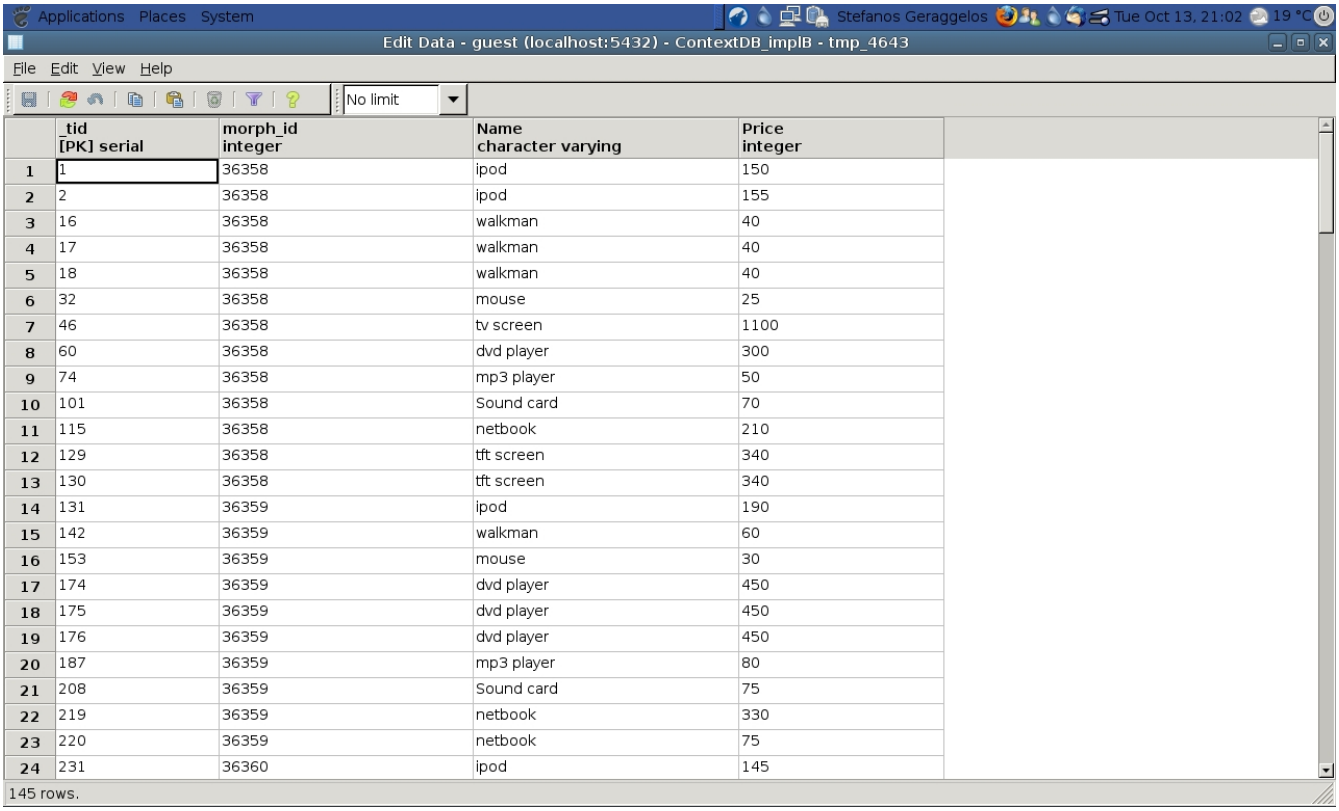
Το ερώτημα 8 αποτελείται από την εκτέλεση των ακόλουθων τελεστών, ενός project, ενός select και ενός καρτεσιανού γινομένου στα polymorphs Product και Product_Inventory. Έτσι, ουσιαστικά πρόκειται για ένα join σε αυτά τα δύο polymorphs και μία προβολή κάποιων attributes. Το ερώτημα, δηλαδή, επιστρέφει τα ονόματα και τις ακριβείς τιμές που πωλήθηκαν συγκεκριμένα προϊόντα. Υπενθυμίζουμε, εδώ, ότι το polymorph Product_Inventory περιέχει πληροφορίες για τις πωλήσεις συγκεκριμένων προϊόντων σε συγκεκριμένα καταστήματα. Ακολουθούν τα Σχήματα 5.33-5.37, όπου παρουσιάζονται το πλάνο του ερωτήματος, τα αποτελέσματα εκτέλεσης αυτού, το output στο terminal και οι μέσοι χρόνοι αντίστοιχα. Το Σχήμα 5.36 περιέχει το output στο terminal για την υλοποίηση A του ερωτήματος 8. Τέλος, στο Σχήμα 5.34 παρουσιάζονται τα αποτελέσματα της υλοποίησης A για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008>, <Retail, UK, 2009> και <Retail, USA, 2008>.



Σχήμα 5.33. Σχηματική απεικόνιση του πλάνου εκτέλεσης για το ερώτημα 8



Σχήμα 5.34. Αποτέλεσμα της εκτέλεσης του ερωτήματος 8 για την υλοποίηση A και για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008>, <Retail, UK, 2009> και <Retail, USA, 2008> αντίστοιχα



	_tid [PK] serial	morph_id integer	Name character varying	Price integer
1	1	36358	ipod	150
2	2	36358	ipod	155
3	16	36358	walkman	40
4	17	36358	walkman	40
5	18	36358	walkman	40
6	32	36358	mouse	25
7	46	36358	tv screen	1100
8	60	36358	dvd player	300
9	74	36358	mp3 player	50
10	101	36358	Sound card	70
11	115	36358	netbook	210
12	129	36358	tft screen	340
13	130	36358	tft screen	340
14	131	36359	ipod	190
15	142	36359	walkman	60
16	153	36359	mouse	30
17	174	36359	dvd player	450
18	175	36359	dvd player	450
19	176	36359	dvd player	450
20	187	36359	mp3 player	80
21	208	36359	Sound card	75
22	219	36359	netbook	330
23	220	36359	netbook	75
24	231	36360	ipod	145

145 rows.

Σχήμα 5.35. Αποτέλεσμα της εκτέλεσης του ερωτήματος 8 για την υλοποίηση B

```
                              TESTING execute().............

TIME OF CARTESIAN PRODUCT OPERATOR EXECUTION:   1.536 seconds

TIME OF SELECT OPERATOR EXECUTION:  1.191 seconds

TIME OF PROJECT OPERATOR EXECUTION: 0.620 seconds

                              .............execute()---->OK

TOTAL TIME OF PLAN8 EXECUTION (attempt No.1):   3.967 seconds


                              TESTING execute().............

TIME OF CARTESIAN PRODUCT OPERATOR EXECUTION:   1.008 seconds

TIME OF SELECT OPERATOR EXECUTION:  0.905 seconds

TIME OF PROJECT OPERATOR EXECUTION: 0.615 seconds

                              .............execute()---->OK

TOTAL TIME OF PLAN8 EXECUTION (attempt No.2):   3.150 seconds
```

```
                              TESTING execute().............

TIME OF CARTESIAN PRODUCT OPERATOR EXECUTION:   1.028 seconds
TIME OF SELECT OPERATOR EXECUTION:  0.938 seconds
TIME OF PROJECT OPERATOR EXECUTION: 0.643 seconds
                              .............execute()---->OK

TOTAL TIME OF PLAN8 EXECUTION (attempt No.3):   3.228 seconds

                              TESTING execute().............

TIME OF CARTESIAN PRODUCT OPERATOR EXECUTION:   1.077 seconds
TIME OF SELECT OPERATOR EXECUTION:  0.937 seconds
TIME OF PROJECT OPERATOR EXECUTION: 0.620 seconds
                              .............execute()---->OK

TOTAL TIME OF PLAN8 EXECUTION (attempt No.4):   3.288 seconds

                              TESTING execute().............

TIME OF CARTESIAN PRODUCT OPERATOR EXECUTION:   1.019 seconds
TIME OF SELECT OPERATOR EXECUTION:  0.966 seconds
TIME OF PROJECT OPERATOR EXECUTION: 0.637 seconds
                              .............execute()---->OK

TOTAL TIME OF PLAN8 EXECUTION (attempt No.5):   3.251 seconds

                              TESTING execute().............

TIME OF CARTESIAN PRODUCT OPERATOR EXECUTION:   1.061 seconds
TIME OF SELECT OPERATOR EXECUTION:  1.036 seconds
TIME OF PROJECT OPERATOR EXECUTION: 0.662 seconds
                              .............execute()---->OK
```

```
TOTAL TIME OF PLAN8 EXECUTION (attempt No.6):   3.396 seconds

                              TESTING execute().............

TIME OF CARTESIAN PRODUCT OPERATOR EXECUTION:   1.048 seconds
TIME OF SELECT OPERATOR EXECUTION:  0.977 seconds
TIME OF PROJECT OPERATOR EXECUTION: 0.689 seconds
                              .............execute()---->OK

TOTAL TIME OF PLAN8 EXECUTION (attempt No.7):   3.369 seconds

                              TESTING execute().............

TIME OF CARTESIAN PRODUCT OPERATOR EXECUTION:   1.075 seconds
TIME OF SELECT OPERATOR EXECUTION:  0.968 seconds
TIME OF PROJECT OPERATOR EXECUTION: 0.650 seconds
                              .............execute()---->OK

TOTAL TIME OF PLAN8 EXECUTION (attempt No.8):   3.350 seconds

                              TESTING execute().............

TIME OF CARTESIAN PRODUCT OPERATOR EXECUTION:   1.096 seconds
TIME OF SELECT OPERATOR EXECUTION:  1.069 seconds
TIME OF PROJECT OPERATOR EXECUTION: 0.756 seconds
                              .............execute()---->OK

TOTAL TIME OF PLAN8 EXECUTION (attempt No.9):   3.577 seconds

                              TESTING execute().............

TIME OF CARTESIAN PRODUCT OPERATOR EXECUTION:   1.098 seconds
TIME OF SELECT OPERATOR EXECUTION:  0.990 seconds
TIME OF PROJECT OPERATOR EXECUTION: 1.469 seconds
```

```
                              .............execute()---->OK

TOTAL TIME OF PLAN8 EXECUTION (attempt No.10):  4.178 seconds


AVERAGE TOTAL TIME OF PLAN8 EXECUTION:    3.475 seconds


*********************************************
* PLAN8 completed as expected!!!              *
*********************************************
```

Σχήμα 5.36 Output εκτέλεσης του ερωτήματος 8 για την υλοποίηση A

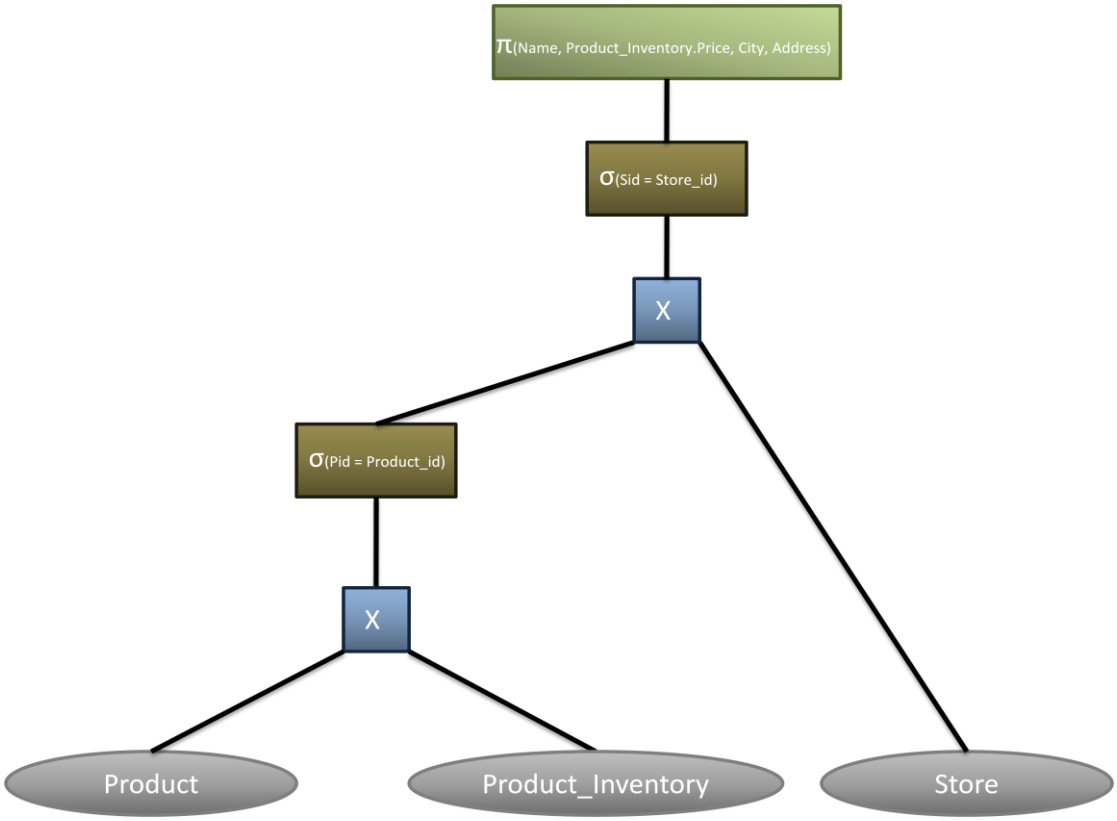
Ερώτημα/Τελεστής	Μέσος χρόνος εκτέλεσης για την υλοποίηση A (seconds)	Μέσος χρόνος εκτέλεσης για την υλοποίηση B (seconds)
Cartesian Product	1.105	0.824
Select	0.998	0.717
Project	0.736	0.372
Ερώτημα 8	3.475	2.546

Σχήμα 5.37. Μέσοι χρόνοι εκτέλεσης για το ερώτημα 8

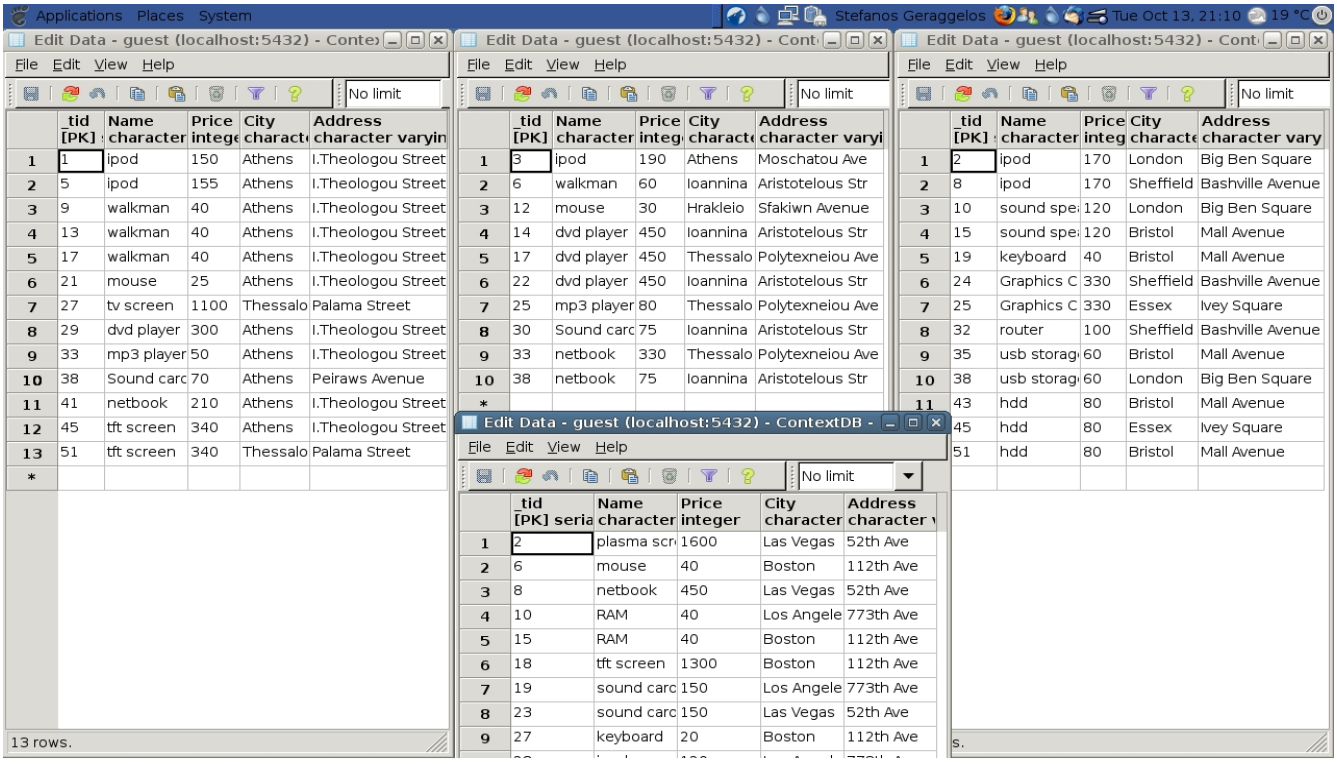
Παρατηρούμε από τον πίνακα του Σχήματος 5.37 ότι για το συγκεκριμένο ερώτημα η υλοποίηση B είναι πιο γρήγορη κατά ~27% από την υλοποίηση A σε ολόκληρο το ερώτημα, ενώ στην εκτέλεση του cartesian product είναι πιο γρήγορη κατά ~25%, στην εκτέλεση του select είναι πιο γρήγορη κατά ~28% και στην εκτέλεση του project είναι πιο γρήγορη κατά ~49%.

5.2.9 Ερώτημα 9

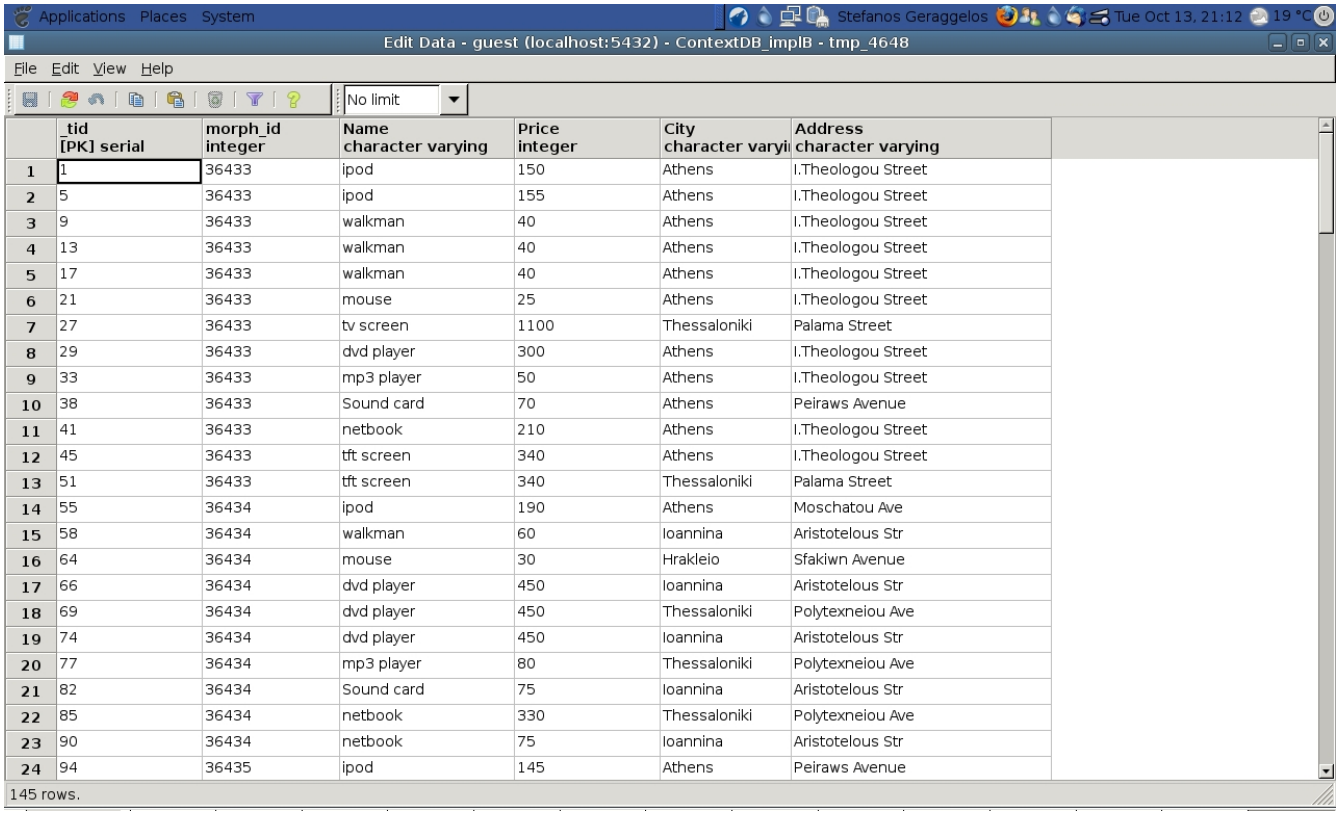
Το ερώτημα 9 αποτελείται από την εκτέλεση ενός join (καρτεσιανό + select) μεταξύ των polymorphs Product και Product_Inventory, ενός join (καρτεσιανό + select) μεταξύ του αποτελέσματος και του polymorph Store και ενός project σε αυτό το αποτέλεσμα. Το ερώτημα, δηλαδή επιστρέφει τα συγκεκριμένα προϊόντα (όνομα και ακριβής τιμή πώλησης) και τα συγκεκριμένα καταστήματα (πόλη και διεύθυνση) στα οποία πωλήθηκαν Ακολουθούν τα Σχήματα 5.38-5.41, όπου παρουσιάζονται το πλάνο του ερωτήματος, τα αποτελέσματα εκτέλεσης αυτού και οι μέσοι χρόνοι αντίστοιχα.



Σχήμα 5.38. Σχηματική απεικόνιση του πλάνου εκτέλεσης για το ερώτημα 9



Σχήμα 5.39. Αποτέλεσμα της εκτέλεσης του ερωτήματος 9 για την υλοποίηση Α και για τα contexts <Wholesale, Greece, 2008>, <Retail, Greece, 2008>, <Retail, UK, 2009> και <Retail, USA, 2008> αντίστοιχα



	_tid [PK] serial	morph_id integer	Name character varying	Price integer	City character varyi	Address character varying
1	1	36433	ipod	150	Athens	I.Theologou Street
2	5	36433	ipod	155	Athens	I.Theologou Street
3	9	36433	walkman	40	Athens	I.Theologou Street
4	13	36433	walkman	40	Athens	I.Theologou Street
5	17	36433	walkman	40	Athens	I.Theologou Street
6	21	36433	mouse	25	Athens	I.Theologou Street
7	27	36433	tv screen	1100	Thessaloniki	Palama Street
8	29	36433	dvd player	300	Athens	I.Theologou Street
9	33	36433	mp3 player	50	Athens	I.Theologou Street
10	38	36433	Sound card	70	Athens	Peiraws Avenue
11	41	36433	netbook	210	Athens	I.Theologou Street
12	45	36433	tft screen	340	Athens	I.Theologou Street
13	51	36433	tft screen	340	Thessaloniki	Palama Street
14	55	36434	ipod	190	Athens	Moschatou Ave
15	58	36434	walkman	60	Ioannina	Aristotelous Str
16	64	36434	mouse	30	Hrakleio	Sfakiwn Avenue
17	66	36434	dvd player	450	Ioannina	Aristotelous Str
18	69	36434	dvd player	450	Thessaloniki	Polytexneiou Ave
19	74	36434	dvd player	450	Ioannina	Aristotelous Str
20	77	36434	mp3 player	80	Thessaloniki	Polytexneiou Ave
21	82	36434	Sound card	75	Ioannina	Aristotelous Str
22	85	36434	netbook	330	Thessaloniki	Polytexneiou Ave
23	90	36434	netbook	75	Ioannina	Aristotelous Str
24	94	36435	ipod	145	Athens	Peiraws Avenue

145 rows.

Σχήμα 5.40. Αποτέλεσμα της εκτέλεσης του ερωτήματος 9 για την υλοποίηση B

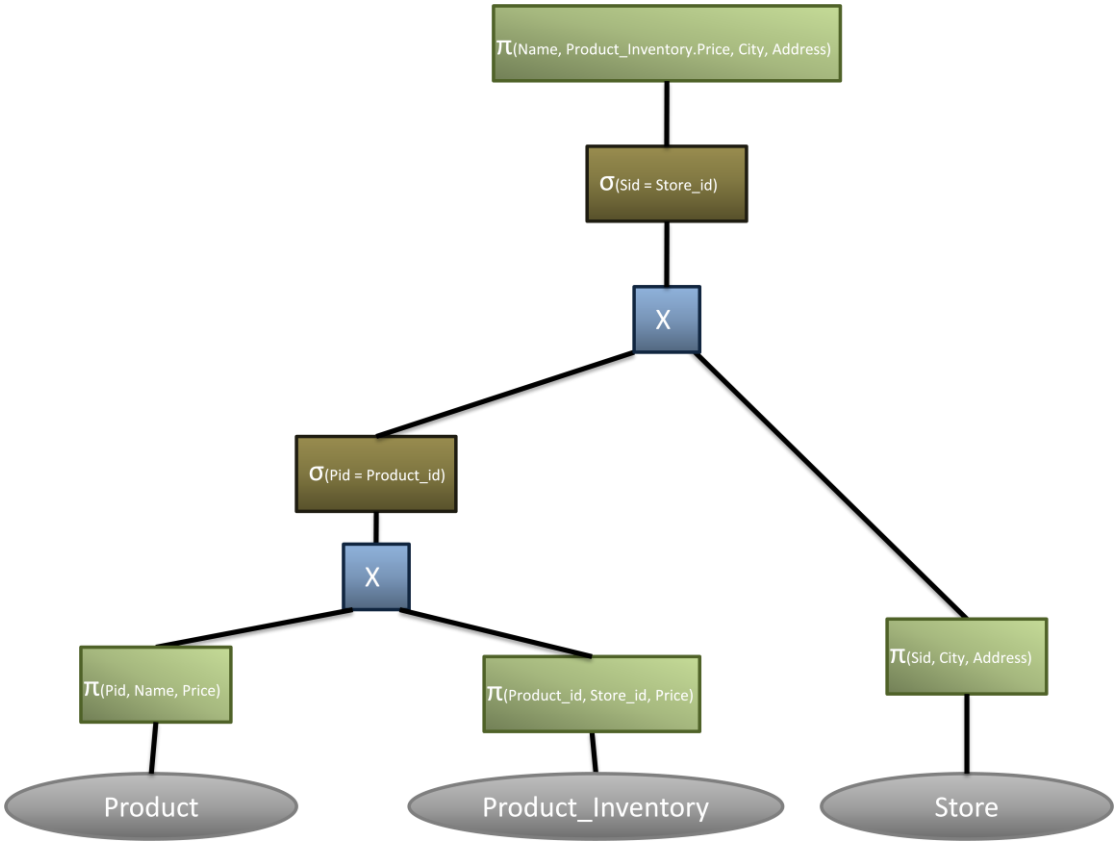
Ερώτημα/Τελεστής	Μέσος χρόνος εκτέλεσης για την υλοποίηση A (seconds)	Μέσος χρόνος εκτέλεσης για την υλοποίηση B (seconds)
Cartesian Product(1)	1.108	0.907
Select(1)	1.031	0.781
Cartesian Product(2)	1.470	1.323
Select(2)	1.279	1.190
Project	0.805	0.551
Ερώτημα 9	6.657	5.702

Σχήμα 5.41. Μέσοι χρόνοι εκτέλεσης για το ερώτημα 9

Παρατηρούμε από τον πίνακα του Σχήματος 5.41 ότι για το συγκεκριμένο ερώτημα η υλοποίηση B είναι πιο γρήγορη κατά ~14% από την υλοποίηση A σε ολόκληρο το ερώτημα, στην εκτέλεση του πρώτου cartesian product είναι πιο γρήγορη κατά ~18%, στην εκτέλεση του πρώτου select είναι πιο γρήγορη κατά ~24%, στην εκτέλεση του δεύτερου cartesian product είναι πιο γρήγορη κατά 10%, στην εκτέλεση του δεύτερου select είναι πιο γρήγορη κατά ~7% και στην εκτέλεση του project είναι πιο γρήγορη κατά ~32%.

5.2.10 Ερώτημα 10

Το ερώτημα 10 επιστρέφει τα ίδια αποτελέσματα με αυτά που επιστρέφει το ερώτημα 9 κατά την εκτέλεσή του. Για αυτό το λόγο δεν παραθέτουμε screenshot με τα αποτελέσματα. Η μόνη διαφορά είναι στη σειρά των τελεστών στο πλάνο εκτέλεσης. Έτσι, στο συγκεκριμένο ερώτημα προσθέσαμε κάποια projects, τα οποία προηγούνται των δύο καρτεσιανών γινομένων. Αυτό παρουσιάζεται στο Σχήμα 5.42. Στο Σχήμα 5.43 φαίνονται οι μέσοι χρόνοι εκτέλεσης για τις δύο υλοποιήσεις.



Σχήμα 5.42. Σχηματική απεικόνιση του πλάνου εκτέλεσης για το ερώτημα 10

Ερώτημα/Τελεστής	Μέσος χρόνος εκτέλεσης για την υλοποίηση A (seconds)	Μέσος χρόνος εκτέλεσης για την υλοποίηση B (seconds)
Project(1)	0.720	0.459
Project(2)	0.673	0.332
Cartesian Product(1)	1.040	0.686
Select(1)	0.925	0.594

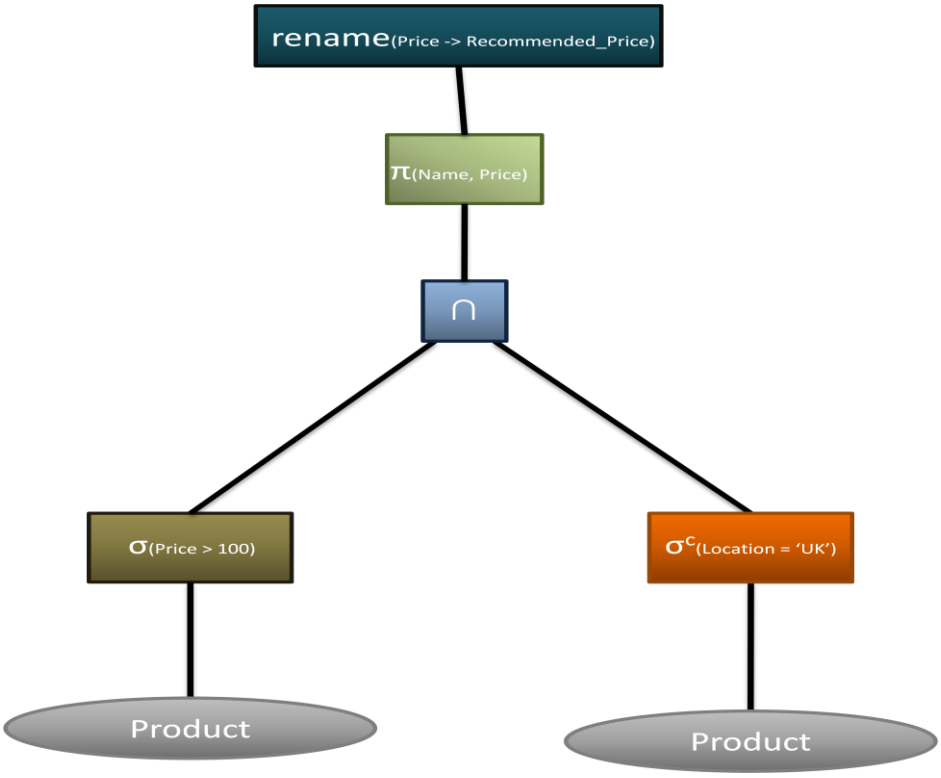
Project(3)	0.769	0.383
Cartesian Product(2)	1.218	0.902
Select(2)	1.093	0.810
Project(4)	0.841	0.541
Ερώτημα 10	8.679	6.098

Σχήμα 5.43. Μέσοι χρόνοι εκτέλεσης για το ερώτημα 10

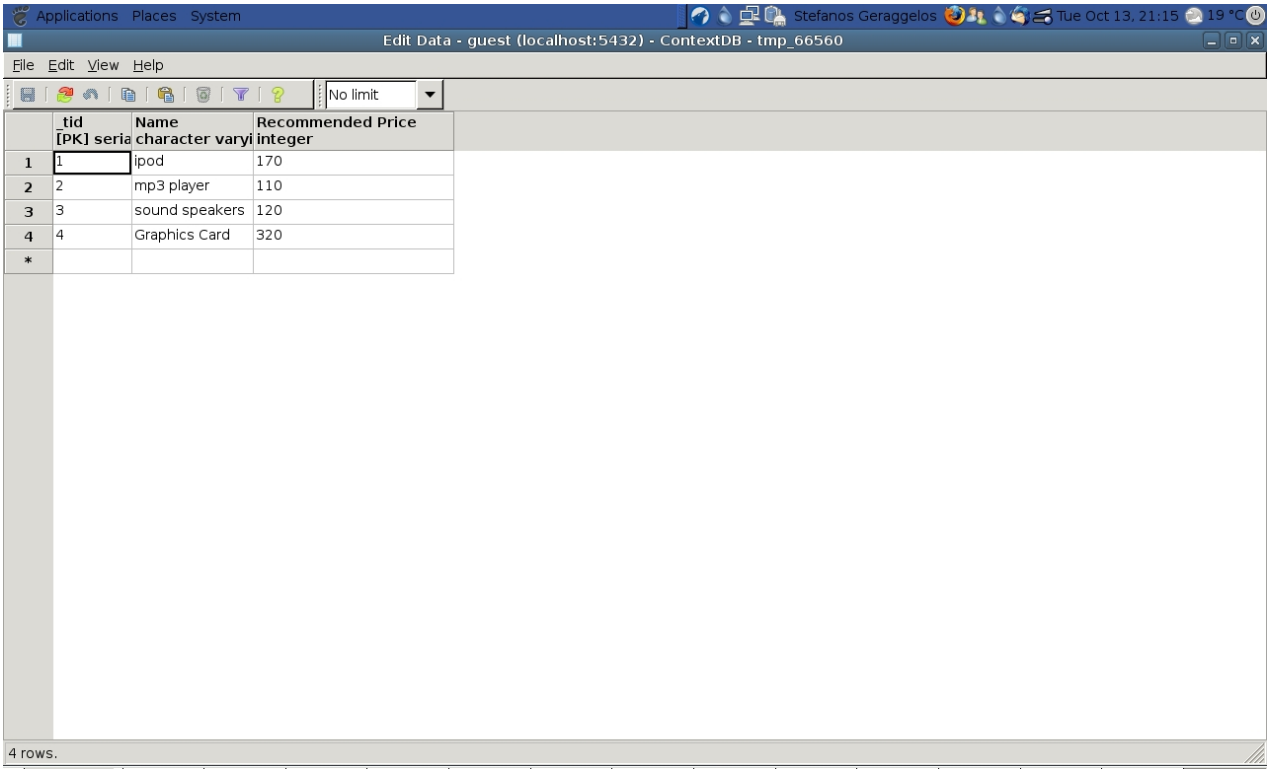
Παρατηρούμε από τον πίνακα του Σχήματος 5.43 ότι για το συγκεκριμένο ερώτημα η υλοποίηση Β είναι πιο γρήγορη κατά ~30% από την υλοποίηση Α σε ολόκληρο το ερώτημα, στην εκτέλεση του πρώτου project είναι πιο γρήγορη κατά ~36%, στην εκτέλεση του δεύτερου project είναι πιο γρήγορη κατά ~51%, στην εκτέλεση του πρώτου cartesian product είναι πιο γρήγορη κατά ~34%, στην εκτέλεση του πρώτου select είναι πιο γρήγορη κατά ~36%, στην εκτέλεση του τρίτου project είναι πιο γρήγορη κατά ~50%, στην εκτέλεση του δεύτερου cartesian product είναι πιο γρήγορη κατά ~26%, στην εκτέλεση του δεύτερου select είναι πιο γρήγορη κατά ~26% και στην εκτέλεση του τέταρτου project είναι πιο γρήγορη κατά ~36%. Γενικά, το ερώτημα 10 είναι πιο από αργό το ερώτημα 9 και για τις δύο υλοποιήσεις.

5.2.11 Ερώτημα 11

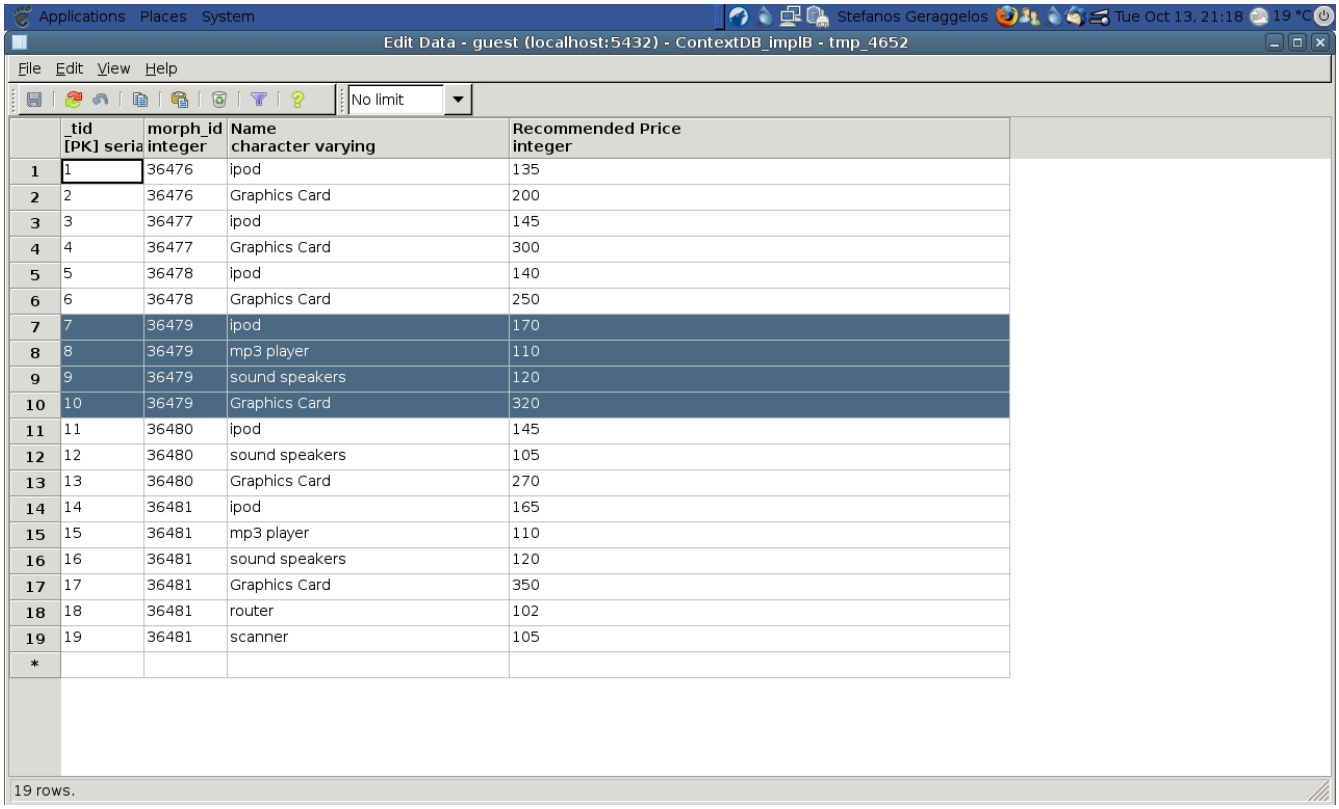
Το ερώτημα 11 επιστρέφει το όνομα και την τιμή (μετονομασμένη) εκείνων των προϊόντων που έχουν τιμή μεγαλύτερη του 100 και είναι διαθέσιμα στο Ηνωμένο Βασίλειο. Γίνεται χρήση, μεταξύ άλλων, και του τελεστή τομής. Το συγκεκριμένο ερώτημα θα μπορούσε να εκτελεστεί πιο αποτελεσματικά με διαφορετικά πλάνα εκτέλεσης. Παρ' όλα αυτά, χρησιμοποιήσαμε τον τελεστή τομής, προκειμένου να μετρήσουμε το χρόνο εκτέλεσης αυτού σε ένα πολύπλοκο ερώτημα. Ακολουθούν τα Σχήματα 5.44-5.47, όπου παρουσιάζονται το πλάνο του ερωτήματος, τα αποτελέσματα εκτέλεσης αυτού, και οι μέσοι χρόνοι αντίστοιχα. Στο Σχήμα 5.45 παρουσιάζονται τα αποτελέσματα της υλοποίησης Α για το context <Retail, UK, 2009>. Στο Σχήμα 5.46 οι tuples που αφορούν το context <Retail, UK, 2009> είναι σκιασμένες.



Σχήμα 5.44. Σχηματική απεικόνιση του πλάνου εκτέλεσης για το ερώτημα 11



Σχήμα 5.45. Αποτέλεσμα της εκτέλεσης του ερωτήματος 11 για την υλοποίηση Α για το context <Retail, UK, 2009>



	_tid [PK] seria	morph_id integer	Name character varying	Recommended Price integer
1	1	36476	ipod	135
2	2	36476	Graphics Card	200
3	3	36477	ipod	145
4	4	36477	Graphics Card	300
5	5	36478	ipod	140
6	6	36478	Graphics Card	250
7	7	36479	ipod	170
8	8	36479	mp3 player	110
9	9	36479	sound speakers	120
10	10	36479	Graphics Card	320
11	11	36480	ipod	145
12	12	36480	sound speakers	105
13	13	36480	Graphics Card	270
14	14	36481	ipod	165
15	15	36481	mp3 player	110
16	16	36481	sound speakers	120
17	17	36481	Graphics Card	350
18	18	36481	router	102
19	19	36481	scanner	105
*				

Σχήμα 5.46. Αποτέλεσμα της εκτέλεσης του ερωτήματος 11 για την υλοποίηση B (σκιασμένες είναι οι tuples που αφορούν στο context <Retail , UK, 2009>)

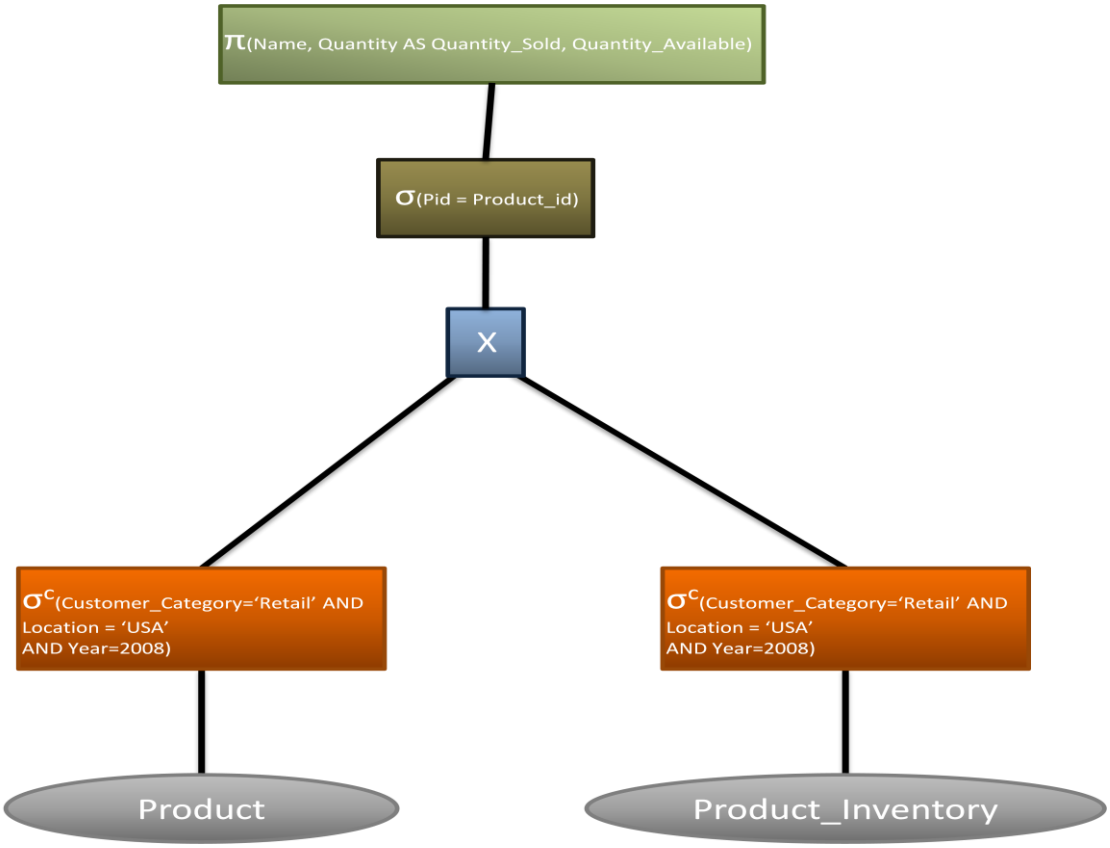
Ερώτημα/Τελεστής	Μέσος χρόνος εκτέλεσης για την υλοποίηση A (seconds)	Μέσος χρόνος εκτέλεσης για την υλοποίηση B (seconds)
Select Schema	0.270	0.176
Select	0.813	0.431
Intersection	0.288	0.233
Project	0.178	0.124
Rename	0.016	0.013
Ερώτημα 11	2.532	1.934

Σχήμα 5.47. Μέσοι χρόνοι εκτέλεσης για το ερώτημα 11

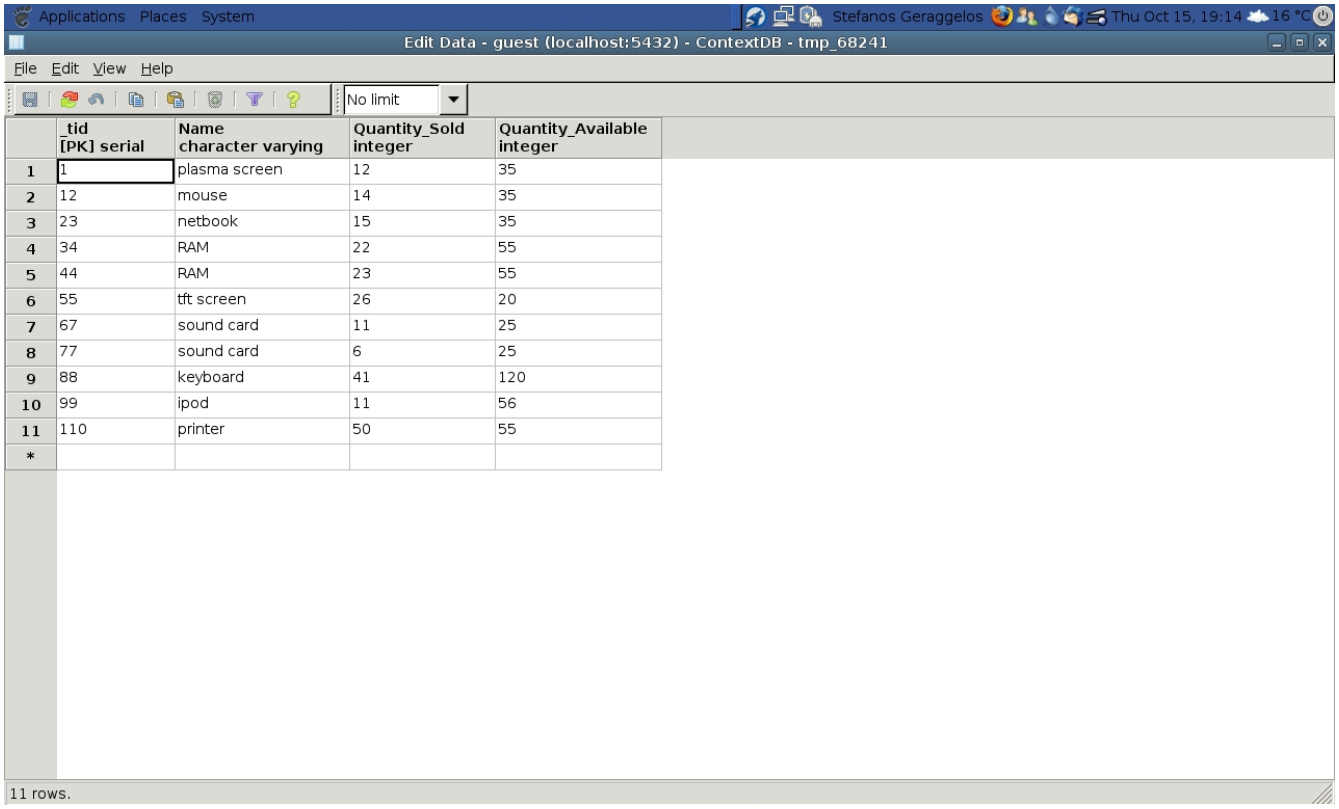
Παρατηρούμε από τον πίνακα του Σχήματος 5.47 ότι για το συγκεκριμένο ερώτημα η υλοποίηση B είναι πιο γρήγορη κατά ~24% από την υλοποίηση A σε ολόκληρο το ερώτημα, στην εκτέλεση του select schema είναι πιο γρήγορη κατά ~35%, στην εκτέλεση του select είναι πιο γρήγορη κατά ~47%, στην εκτέλεση του intersection είναι πιο γρήγορη κατά ~19%, στην εκτέλεση του project είναι πιο γρήγορη κατά ~30% και στην εκτέλεση του rename είναι πιο γρήγορη κατά ~19%.

5.2.12 Ερώτημα 12

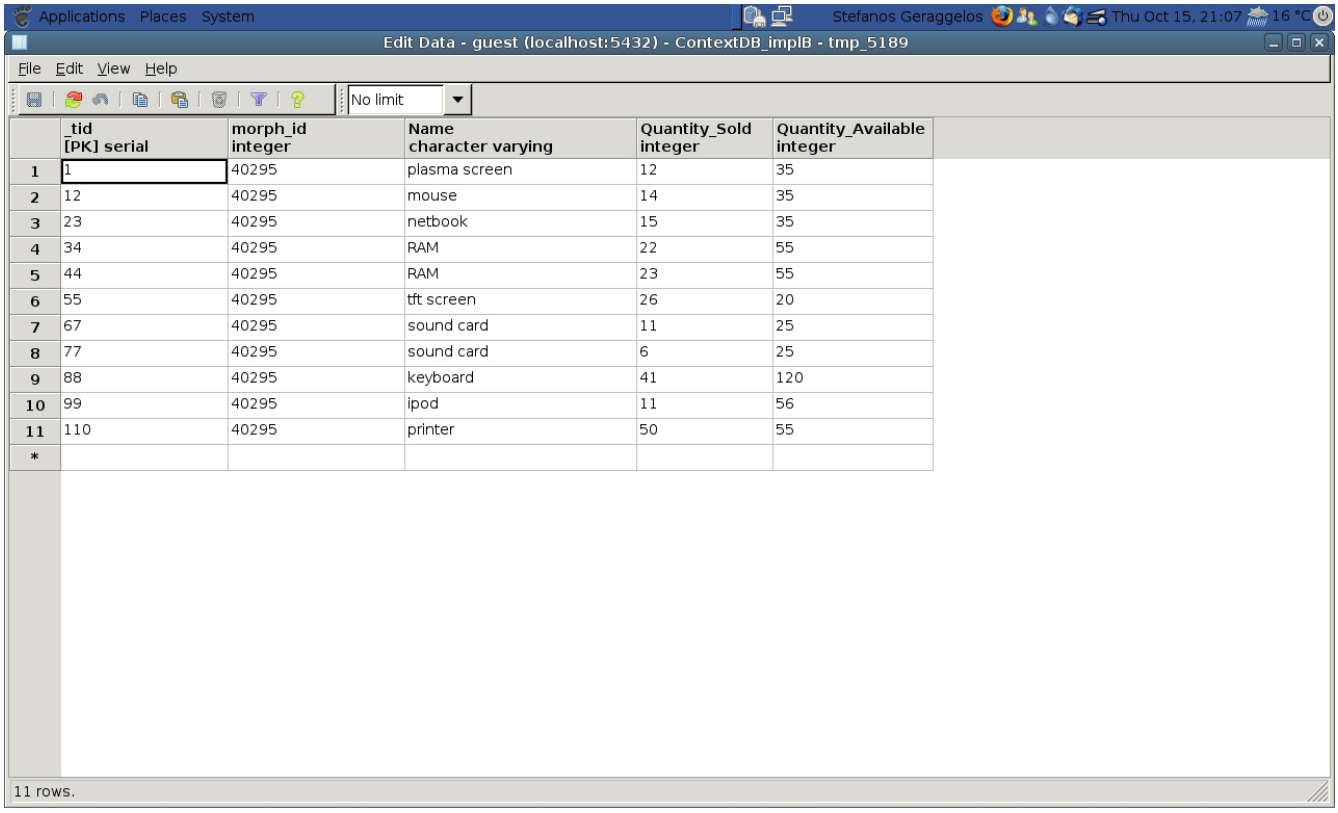
Το ερώτημα 12 πραγματοποιεί ένα natural join ανάμεσα στα polymorphs Product και Product_Inventory, αφού πρώτα επιστρέψει εκείνα τα προϊόντα που είναι διαθέσιμα στις Ηνωμένες Πολιτείες Αμερικής, το 2008 και έχουν λιανική τιμή στο πρώτο επίπεδο. Το ερώτημα, δηλαδή, επιστρέφει τα ονόματα και τις ακριβείς τιμές που πωλήθηκαν συγκεκριμένα προϊόντα. Η μόνη διαφορά που έχει με το ερώτημα 8 είναι ότι πριν το καρτεσιανό γινόμενο προηγούνται δύο τελεστές select schema, προκειμένου οι επόμενοι τελεστές να πραγματοποιηθούν σε ένα morph. Πραγματοποιούμε αυτό το ερώτημα, ώστε να συγκρίνουμε τις δύο υλοποιήσεις στην περίπτωση όπου κάθε polymorph, το οποίο αναφέρεται στο ερώτημα, περιέχει ένα ακριβώς morph. Σε αυτήν την περίπτωση αναμένεται η υλοποίηση A να είναι πιο γρήγορη από την υλοποίηση B, καθώς πλέον κατά την υλοποίηση A τα επιμέρους ερωτήματα πραγματοποιούνται σε ένα πίνακα. Ακολουθούν τα Σχήματα 5.48-5.51, όπου παρουσιάζονται το πλάνο του ερωτήματος, τα αποτελέσματα εκτέλεσης αυτού, και οι μέσοι χρόνοι αντίστοιχα. Στο Σχήμα 5.49 παρουσιάζονται τα αποτελέσματα της υλοποίησης A για το context <Retail, USA, 2008>, το οποίο είναι και το μοναδικό morph του αποτελέσματος. Ομοίως, στο Σχήμα 5.50 παρουσιάζονται τα αποτελέσματα της υλοποίησης B για το context <Retail, USA, 2008>.



Σχήμα 5.48. Σχηματική απεικόνιση του πλάνου εκτέλεσης για το ερώτημα 12



Σχήμα 5.49. Αποτέλεσμα της εκτέλεσης του ερωτήματος 12 για την υλοποίηση A (το αποτέλεσμα περιλαμβάνει μόνο το morph για το context <Retail, USA, 2008>



Σχήμα 5.50. Αποτέλεσμα της εκτέλεσης του ερωτήματος 12 για την υλοποίηση A (το αποτέλεσμα περιλαμβάνει μόνο το morph για το context <Retail, USA, 2008>

Ερώτημα/Τελεστής	Μέσος χρόνος εκτέλεσης για την υλοποίηση A (seconds)	Μέσος χρόνος εκτέλεσης για την υλοποίηση B (seconds)
Select Schema(1)	0.080	0.079
Select Schema(2)	0.062	0.062
Cartesian Product	0.106	0.125
Select	0.090	0.106
Project	0.066	0.074
Ερώτημα 12	1.328	1.387

Σχήμα 5.51. Μέσοι χρόνοι εκτέλεσης για το ερώτημα 12

Παρατηρούμε από τον πίνακα του Σχήματος 5.51 ότι για το συγκεκριμένο ερώτημα η υλοποίηση Α είναι πιο γρήγορη κατά ~4% από την υλοποίηση Β σε ολόκληρο το ερώτημα, στην εκτέλεση του πρώτου select schema η υλοποίηση Β είναι πιο γρήγορη από την υλοποίηση Α κατά ~1%, στην εκτέλεση του δεύτερου select schema δεν υπάρχει διαφορά ως προς την απόδοση των δύο υλοποιήσεων, στην εκτέλεση του cartesian product η υλοποίηση Α είναι πιο γρήγορη κατά ~15% από την υλοποίηση Β, στην εκτέλεση του select είναι πιο γρήγορη κατά ~15% και στην εκτέλεση του project είναι πιο γρήγορη κατά ~11%.

5.3 Συμπεράσματα

Από την εκτέλεση των προαναφερθέντων ερωτημάτων στη context-aware βάση δεδομένων που περιγράψαμε προκύπτουν διάφορα συμπεράσματα. Συγκεκριμένα, παρατηρούμε από τα αποτελέσματα ότι στα ερωτήματα όπου παράγονται περισσότερα του ενός morphs ανά polymorph η υλοποίηση Β είναι κατά πολύ γρηγορότερη από την υλοποίηση Α. Κάτι τέτοιο ήταν αναμενόμενο καθώς τελικά κατά την υλοποίηση Α πραγματοποιείται η εκάστοτε πράξη σε τόσους πίνακες όσα είναι τα morphs, ενώ κατά την υλοποίηση Β η πράξη πραγματοποιείται σε ένα μόνο πίνακα. Έτσι, γίνεται σαφής ο λόγος για τον οποίο η υλοποίηση Β είναι τόσο αποδοτικότερη από την υλοποίηση Α.

Πρέπει, βέβαια, να αναφέρουμε ότι για βάσεις δεδομένων στις οποίες τα δεδομένα (tuples) αυξάνονται δραματικά (για δεδομένο αριθμό morphs ανά polymorph), αναμένεται να υπάρξει κάποιο σημείο στο οποίο η υλοποίηση Α θα είναι πιο αποδοτική από την υλοποίηση Β, καθώς ο αριθμός των tuples θα επιβαρύνει περισσότερο την υλοποίηση Β. Πειράματα με τέτοιο όγκο

δεδομένων ανήκουν στις μελλοντικές επεκτάσεις του συστήματος, που περιγράφονται στο κεφάλαιο 6.

Τέλος, στους τελεστές που εκτελούνται και στον πίνακα του context (πχ select schema), η επιβάρυνση για την υλοποίηση Α προκύπτει από τις DDL/DML εντολές που δημιουργούν τους νέους προσωρινούς πίνακες και εισάγουν δεδομένα σε αυτούς. Για παράδειγμα, η εκτέλεση ενός select schema και στις δύο υλοποιήσεις γίνεται μεν σε ένα μόνο πίνακα (αυτόν του context), αλλά το αποτέλεσμα (τα morphs που προκύπτουν) πρέπει να αντιγραφεί στους καινούριους προσωρινούς πίνακες, όπου για την υλοποίηση Α είναι πάντα περισσότεροι.

Η μοναδική περίπτωση στα πειράματα που εκτελέστηκαν, όπου η υλοποίηση Α είναι οριακά πιο γρήγορη από την υλοποίηση Β είναι όταν μετά από ένα select schema προκύπτει ένα μόνο morph και έτσι και στις δύο υλοποιήσεις οι επόμενες πράξεις πραγματοποιούνται σε ένα πίνακα.

6

Επίλογος

Στο κεφάλαιο αυτό συνοψίζουμε τα αποτελέσματα της διπλωματικής εργασίας (ενότητα 6.1) και παραθέτουμε κάποιες πιθανές μελλοντικές επεκτάσεις (ενότητες 6.2) του συστήματος.

6.1 Σύνοψη και συμπεράσματα

Στην παρούσα διπλωματική υλοποιήσαμε κάποια υποσυστήματα του Context-Aware Συστήματος Διαχείρισης Βάσεων Δεδομένων. Συγκεκριμένα, τα υποσυστήματα που υλοποιήθηκαν ήταν το Υποσύστημα Εκτέλεσης Πλάνων, το Υποσύστημα Εκτέλεσης Τελεστών και το Υποσύστημα Αποθήκευσης στο σχεσιακό ΣΔΒΔ.

Οι επιμέρους μονάδες που υλοποιήθηκαν ήταν:

- το "Υποσύστημα **Αποθήκευσης και Διεπαφής με το Σχεσιακό ΣΔΒΔ**", με το οποίο το σύστημα επικοινωνεί κάνοντας χρήση του σχεσιακού ΣΔΒΔ,
- το "Υποσύστημα του **Καταλόγου** (Catalogue)", το οποίο είναι υπεύθυνο για τη διαχείριση των μετα-δεδομένων (meta-data) που αφορούν α) το context και β) τα polymorphs/morphs,
- το "Υποσύστημα **Διαχείρισης Δεδομένων** (Data)", που χειρίζονται τις εκάστοτε εισαγωγές/διαγραφές/ενημερώσεις δεδομένων που, επίσης, αφορούν α) το context και β) τα polymorphs/morphs,

- το "Υποσύστημα **Εκτέλεσης Τελεστών** (Operators)", το οποίο είναι υπεύθυνο για την εκτέλεση όλων των context-aware τελεστών, που έχουν οριστεί για να υποστηρίζεται το νέο αυτό context-aware μοντέλο.
- και το "Υποσύστημα **Εκτέλεσης Πλάνων** (Query Executor)", το οποίο αναλαμβάνει την εκτέλεση ενός context-aware ερωτήματος με τη μορφή ενός πλάνου.

Έτσι, με την ολοκλήρωση του συστήματος είναι δυνατή η εισαγωγή ενός context-aware ερωτήματος, η εκτέλεσή του και η αποθήκευση των αποτελεσμάτων στο σχεσιακό ΣΔΒΔ.

Επίσης, υλοποιήθηκαν δύο διαφορετικές υλοποιήσεις που αφορούν στην αποθήκευση των context-aware δεδομένων στο σχεσιακό ΣΔΒΔ. Τέλος, εκτελέστηκαν πειράματα, με τη μορφή ερωτημάτων σε μία δεδομένη context-aware βάση δεδομένων για τις δύο αυτές υλοποιήσεις. Σε αυτά τα πειράματα μετρήθηκαν οι χρόνοι εκτέλεσης του εκάστοτε ερωτήματος για κάθε μία από τις υλοποιήσεις (Α, Β).

Το συμπέρασμα από την εκτέλεση των πειραμάτων αυτών είναι ότι γενικά η υλοποίηση Β είναι πιο αποδοτική στις βάσεις δεδομένων και στα ερωτήματα που αφορούν περισσότερα morphs ανά polymorph. Η υλοποίηση Α είναι οριακά πιο γρήγορη σε βάσεις δεδομένων και σε ερωτήματα (πχ μετά από κάποιο select schema) που αφορούν ένα (ή λίγο περισσότερα) morphs ανά polymorph. Στη γενική περίπτωση, δηλαδή, μπορούμε να υποστηρίξουμε ότι η υλοποίηση Β είναι προτιμητέα σε σχέση με την υλοποίηση Α. Παρ' όλα αυτά, μπορούν να πραγματοποιηθούν και άλλου είδους πειράματα, τα οποία περιγράφονται στην επόμενη ενότητα, τα αποτελέσματα των οποίων να διαφοροποιήσουν τα συμπεράσματα για την απόδοση των διαφορετικών υλοποιήσεων.

6.2 Μελλοντικές επεκτάσεις

Το σύστημα μπορεί να επεκταθεί σε πρώτη φάση πραγματοποιώντας την υλοποίηση Γ (βλ. ενότητα 3.3.3). Επίσης, στην περίπτωση αυτή είναι αναγκαίο τα αντίστοιχα πειράματα, που πραγματοποιήθηκαν και περιγράφηκαν στο κεφάλαιο 5, να εκτελεσθούν και για την υλοποίηση Γ. Οι μετρήσεις των αποτελεσμάτων και η σύγκρισή τους με τα αποτελέσματα των υλοποιήσεων Α και Β μπορεί να καταδείξει την αποδοτικότερη υλοποίηση από τις τρεις, ανά περίπτωση χρήσης.

Μια άλλη διερεύνηση με στόχο την καλύτερη αποτίμηση της απόδοσης του συστήματος είναι η υλοποίηση ελέγχων πλήρως παραμετροποιήσιμων ως προς α) το πλήθος των context instances, β) το πλήθος των context attributes, γ) το πλήθος των polymorphs, δ) το πλήθος των morphs ανά polymorph, ε) το πλήθος των morph attributes και στ) το πλήθος των tuples ανά morph. Η κρίσιμη παράμετρος του αριθμού των tuples ανά morph θα πρέπει να κινείται

σε ένα μεγάλο εύρος, προκειμένου να ελεγχθεί το σύστημα σε πραγματικά μεγάλο όγκο δεδομένων. Με τη δημιουργία υπέρογκων και τυχαίων (random) δεδομένων τα αποτελέσματα της εκτέλεσης πειραμάτων ενδέχεται να διαφοροποιηθούν δραματικά. Αυτά τα πειράματα θα εκτελεστούν και στις τρεις, πλέον, υλοποιήσεις, ώστε να εξαχθούν πιο ολοκληρωμένα και ασφαλή συμπεράσματα.

Επίσης, το σύστημα που υλοποιήσαμε μπορεί να επεκταθεί με την προσθήκη λειτουργιών, οι οποίες θα κινούνται προς τη βελτιστοποίηση του, μειώνοντας τους χρόνους εκτέλεσης ερωτημάτων. Μία τέτοια βελτιστοποίηση μπορεί να είναι η εκτέλεση ερωτημάτων στη μνήμη, χωρίς να χρειάζεται τα αποτελέσματα να αποθηκεύονται στο δίσκο. Μία άλλη εργασία προς αυτή την κατεύθυνση είναι η υλοποίηση της αποθήκευσης δεδομένων απευθείας στο δίσκο, χωρίς τη μεσολάβηση του σχεσιακού ΣΔΒΔ, που χρησιμοποιήσαμε.

Τέλος, καθώς ο απώτερος στόχος είναι η υλοποίηση ενός ολοκληρωμένου και πλήρους λειτουργικού Context-Aware ΣΔΒΔ, το επόμενο βήμα είναι ο σχεδιασμός και η υλοποίηση των υπόλοιπων υποσυστημάτων, που το απαρτίζουν, από το Γραφικό Περιβάλλον της διεπαφής του χρήστη με το σύστημα μέχρι και το Βελτιστοποιητή Πλάνων. Έτσι, μετά την πλήρη υλοποίηση του Context-Aware ΣΔΒΔ, θα παρέχεται η δυνατότητα στο χρήστη να χρησιμοποιήσει το σύστημα σε όλη την έκτασή του, δημιουργώντας context-aware βάσεις δεδομένων μέσω γραφικών εργαλείων και εκτελώντας αντίστοιχα ερωτήματα. Όντας στο σημείο αυτό, θα πρέπει να εκτελεστούν τα προαναφερθέντα πειράματα στο συνολικό σύστημα, ώστε να αποκτήσουμε σαφέστερη εικόνα για τη λειτουργία του και την απόδοσή του.

7

Βιβλιογραφία

[ABS00] Serge Abiteboul, Peter Buneman, Dan Suciu,
Data on the Web, From Relations to Semistructured Data and XML, Morgan Kaufmann Publishers San Francisco, California 2000

[DA00] Anind K. Dey and Gregory D. Abowd,
Towards a Better Understanding of Context and Context-Awareness,
Georgia Institute of Technology, Atlanta, GA

[DBN09] The Debian Operating System 5.0,
http://www.debian.org/

[EAM09] Hicham G. Elmongui, Walid G. Aref, Mohamed F. Mokbel,
Chameleon: Context-Awareness inside DBMSs,
Purdue University and University of Minnesota

[ECL] Eclipse Platform for C++, Version: 3.4.2,
http://www.eclipse.org/platform

[FAJ04] Ling Feng, Peter M.G. Apers and Willem Jonker,
Towards Context-Aware Data Management for Ambient Intelligence,
Dept. of Computer Science, University of Twente

[GPP] g++ compiler (gcc version 4.3.2)

[HBS02] Held A., Buchholz S. and Schill A.,
Modeling of context information for pervasive computing applications,
In Proceedings of SCI 2002/ISAS 2002 (2002)

[HIR03] Henricksen K., Indulska J. and Rakotonirainy A.,
Generating Context Management Infrastructure from High-Level Context Models,
In Industrial Track Proceedings of the 4th International Conference on Mobile Data Management (MDM2003) (Melbourne/Australia, January 2003)

[PSQ] The open source object-relational database system PostgreSQL 8.3.8,
http://www.postgresql.org/

[PQX] The libpqxx library (version: 3.0.2),
http://pqxx.org/development/libpqxx

[RS09] Ioannis N. Roussos and Timos Sellis,
A Model for Context Aware Relational Databases,
School of Electrical and Computer Engineering, National Technical University of Athens,
Institute for the Management of Information Systems (R.C. “Athena”) , 2009

[SG01] Yannis Stavrakas, Manolis Gergatsoulis,
Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web, Athens 2001

[SKS04] Silberschatz, Korth and Sudarshan,
Συστήματα Βάσεων Δεδομένων,
Εκδόσεις Μ. Γκιούρδας, Έκδοση 4η

[SPV05] Kostas Stefanidis, Evaggelia Pitoura and Panos Vassiliadis,
A Context-Aware Preference Database System,
Department of Computer Science, University of Ioannina, Greece, 2009

[SP04] Thomas Strang and Claudia Linnhoff-Popien,
A Context Modeling Survey,
Institute for Communications and Navigation, Wessling/Oberpfaffenhofen, Institute for Informatics, Munich, Germany

[Sta02] Yannis Stavrakas,
Multidimensional Query Language, Technical Report,
Knowledge and Database Systems Laboratory,
National Technical University, Athens 2002

[Sta03] Yannis Stavrakas,
Multidimensional Semistructured Data,
Representing and Querying Context-Dependent
Multifaceted Information on the Web, PhD thesis,
Knowledge and Database Systems Laboratory,
National Technical University, June 2003

[STL94] Standard Template Library Programmer's Guide
http://www.sgi.com/tech/stl/, Hewlett-Packard Company 1994

[Str97] B. Stroustrup,
The C++ Programming Language (3rd edition),
Addison Wesley Longman, Reading, MA. 1997

[WAP09] WAPFORUM. User Agent Profile (UAProf),
http://www.wapforum.org

[W3C07] W3C. Composite Capabilities / Preferences Profile (CC/PP),
http://www.w3.org/Mobile/CCPP