ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός και Υλοποίηση Υποσυστήματος Εκτέλεσης Ερωτημάτων για ένα Πρότυπο Σύστημα Εξατομικευμένης Διαχείρισης Βάσεων Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΚΩΝΣΤΑΝΤΙΝΟΥ ΚΑΡΟΖΟΥ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2012

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

Σχεδιασμός και Υλοποίηση Υποσυστήματος Εκτέλεσης Ερωτημάτων για ένα Πρότυπο Σύστημα Εξατομικευμένης Διαχείρισης Βάσεων Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΚΩΝΣΤΑΝΤΙΝΟΥ ΚΑΡΟΖΟΥ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την Δευτέρα, 8 Οκτωβρίου 2012.

.....	.....	.....
Τιμολέων Σελλής	Ιωάννης Βασιλείου	Γιάννης Σταύρακας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Ερευνητής Β' ΙΠΣΥ/Ε.Κ. "Αθηνά"

Αθήνα, Οκτώβριος 2012

.....

ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΑΡΟΖΟΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Πρωτίστως θα ήθελα να ευχαριστήσω τους γονείς μου για την στήριξη, υπομονή και αγάπη τους.
Ευχαριστώ επίσης τον κο Τίμο Σελλή για την καθοδήγησή του και για το ενδιαφέρον που έδειξε.

Περίληψη

Στα πλαίσια της αλματώδους ανάπτυξης του Διαδικτύου, του ηλεκτρονικού εμπορίου και των κοινωνικών δικτύων, καθίσταται επιτακτική η ανάγκη για συστήματα και τεχνικές που παρέχουν μοναδικές, εξατομικευμένες υπηρεσίες για χρήστες και επιχειρήσεις.

Τα συστήματα εξατομίκευσης προσαρμόζουν την λειτουργία τους στα ενδιαφέροντα και στις προτιμήσεις των χρηστών.

Στα περισσότερα από αυτά τα συστήματα, η εξατομίκευση επιτυγχάνεται στην εκάστοτε εφαρμογή βασιζόμενη σε αλγορίθμους οι οποίοι αναπτύσσονται ειδικά για την συγκεκριμένη εφαρμογή.

Ο στόχος της παρούσας διπλωματικής είναι η επέκταση του σχεσιακού μοντέλου βάσεων δεδομένων ώστε να ενσωματώνονται οι προτιμήσεις των χρηστών στο επίπεδο της διαχείρισης των δεδομένων. Σχεδιάστηκε και υλοποιήθηκε το σύστημα εξατομίκευσης PrefSQL, το οποίο επιτυγχάνει αυτή την ενσωμάτωση με την χρήση κάποιων νέων τελεστών.

Το PrefSQL αποτελείται από τα παρακάτω υποσυστήματα: το Υποσύστημα Διεπαφής (I), μέσω του οποίου ο χρήστης θέτει ερωτήματα στο σύστημα και λαμβάνει εξατομικευμένα αποτελέσματα, το Υποσύστημα Διαμόρφωσης – Βελτιστοποίησης Ερωτημάτων (II), το οποίο κάνει συντακτική ανάλυση του ερωτήματος, καταστρώνει το πλάνο εκτέλεσης και ενσωματώνει τις προτιμήσεις και το Υποσύστημα Εκτέλεσης Ερωτημάτων (III) το οποίο εκτελεί το διαμορφωμένο ερώτημα μέσω των νέων τελεστών.

Κατά την διάρκεια της διπλωματικής δόθηκε βάση στην εξεύρεση και υλοποίηση του βέλτιστου πλάνου εκτέλεσης PrefSQL ερωτημάτων. Προς αναζήτηση αυτού, έγιναν πολλές μετρήσεις, καταγράφηκαν χρόνοι εκτέλεσης διαφορετικών ερωτημάτων και εξήχθησαν συμπεράσματα.

Λέξεις Κλειδιά: preference, preference-aware συστήματα, preference-aware RDBMS, συστήματα συστάσεων, συστήματα εξατομίκευσης

Abstract

In the context of the exponential growth of the Web, e-business and social networks, there is a definite need for systems and techniques providing individual services to users and companies.

Personalized systems adopt their function to the users' interests and preferences.

In most of these systems, personalization is succeeded with algorithms developed especially for each occasion.

The main goal of this diploma thesis is to extend the relational database model in order to incorporate user's preferences in the database engine. We have designed and implemented a personalized system called PrefSQL which achieves to do this extension with the use of some new operators.

PrefSQL contains the following subsystems: the Interface Subsystem (I), through which the user poses queries and gets personalized results, the Query Plan Builder / Optimizer (II), which analyzes the query, builds the execution plan while at the same time incorporating the preferences and finally the Query Executor (III), which executes the formed query with the use of its new operators.

During our work, much attention has been given to finding and implementing the best execution plan for PrefSQL queries. In that direction, several kinds of queries where executed, the execution times of which were written down and studied so as to get to a conclusion.

Keywords: preference, preference-aware systems, preference-aware RDBMS, recommender systems, personalized systems

.

Περιεχόμενα

1 *Εισαγωγή*

1.1 Εξατομικευμένα Συστήματα Βάσεων Δεδομένων

Ο Παγκόσμιος Ιστός, στην σύγχρονη μορφή του, είναι μία τεράστια δεξαμενή ανομοιογενών δεδομένων. Η υπερπροσφορά πληροφοριών και υπηρεσιών δυσκολεύει τους χρήστες κατά την αναζήτηση των επιθυμητών σε αυτούς στοιχείων. Παράλληλα, η εξέλιξη των κοινωνικών δικτύων και του Web 2.0 σε συνδυασμό με τον αυξανόμενο όγκο συναλλαγών μέσω του διαδικτύου (π.χ. αγορές προϊόντων) παρέχουν άφθονα δεδομένα για τα χαρακτηριστικά, τα ενδιαφέροντα και τις καταναλωτικές συνήθειες των χρηστών. Για παράδειγμα, σε ένα κοινωνικό δίκτυο οι χρήστες συνηθίζεται να μοιράζονται πληροφορίες για τις αγαπημένες τους ταινίες, μουσική, βιβλία, ενώ οι χρήστες ενός διαδικτυακού καταστήματος αφήνουν ίχνη από τα προϊόντα που αγόρασαν ή εξέτασαν στο παρελθόν.

Τα δεδομένα αυτά είναι χρήσιμο να συλλεχθούν και να αξιοποιηθούν από μηχανές αναζήτησης, διαφημιστές κλπ. Ως εκ τούτου, γίνεται επιτακτική η ανάγκη για συστήματα και τεχνικές που να παρέχουν στοχευμένες εξατομικευμένες υπηρεσίες. Αυτό επιτυγχάνεται με τα συστήματα εξατομίκευσης.

Τα συστήματα εξατομίκευσης προσαρμόζουν την λειτουργία τους στις προτιμήσεις των χρηστών. Αποσκοπούν στην δημιουργία ενός προφίλ του χρήστη με βάση το οποίο του επιστρέφουν συστάσεις για αντικείμενα που φαίνεται να ικανοποιούν τις απαιτήσεις του. Αποτελούν αντικείμενο ενδιαφέροντος και έρευνας από το 1990, τα τελευταία χρόνια, όμως, γίνονται ολοένα και πιο δημοφιλή.

1.1.1 Εξατομίκευση στο Διαδίκτυο

Σε μια πλοήγηση στον Παγκόσμιο Ιστό συναντάμε πολλές ιστοσελίδες οι οποίες υποστηρίζουν εξατομίκευση των χρηστών και επιστρέφουν συστάσεις (recommendations), άλλοτε με περισσότερη και άλλοτε με λιγότερη επιτυχία. Υπάρχει μεγάλη ποικιλία στις μεθόδους που εφαρμόζονται για να εκμαιευθούν, αποθηκευτούν, επεξεργαστούν οι προτιμήσεις του χρήστη ώστε εντέλει να παραχθούν συστάσεις.

Κλασικότατο παράδειγμα εμπορικού συστήματος εξατομίκευσης – το οποίο υπήρξε και από τα πρώτα που εμφανίστηκαν στο διαδίκτυο – είναι το Amazon[1]. Πρόκειται για ένα online πολυκατάστημα (ηλεκτρονικά είδη, βιβλία κ.α.) το οποίο διαμορφώνει προτάσεις για την αγορά προϊόντων με βάση αυτά τα οποία ο χρήστης έχει αγοράσει ή επιλέξει σε συνδυασμό με τις επιλογές άλλων χρηστών με παρόμοια ενδιαφέροντα.

Άλλη επιτυχής εφαρμογή τέτοιου είδους υπηρεσιών σε ιστοσελίδες αποτελεί το γνωστό σε όλους μας Youtube[2], το οποίο προσφέρει την δυνατότητα αποθήκευσης, αναζήτησης και αναπαραγωγής video on demand. Με το που εισερχόμαστε στον συγκεκριμένο ιστότοπο, το σύστημα εντοπίζει τον κόμβο μέσω του οποίου γίνεται η σύνδεση και αρχικά προτείνει στον επισκέπτη βίντεο τα οποία είναι δημοφιλή σε χρήστες από την ίδια γεωγραφική θέση/χώρα. Σε περίπτωση σύνδεσης (Login) οι συστάσεις εμπλουτίζονται με βίντεο τα οποία φορτώθηκαν στα κανάλια στα οποία ο χρήστης έχει κάνει εγγραφή και με άλλα τα οποία είναι παραπλήσιου περιεχομένου με αυτά που έχει παρακολουθήσει στο παρελθόν.

Θα ήταν παράλειψη εάν δεν γινόταν αναφορά στα recommendation systems που αποσκοπούν στην σύσταση ταινιών. Η πιο ολοκληρωμένη βάση δεδομένων με ταινίες, σειρές και γενικότερα οποιοδήποτε άλλο οπτικοακουστικό υλικό έχει προβληθεί/ πρόκειται να προβληθεί στο άμεσο μέλλον σε κινηματογράφο, τηλεόραση, DVD, βιντεοκασέτα είναι η βάση του IMDb[3]. Ο χρήστης πλοηγείται, αναζητά ταινίες, ηθοποιούς, σκηνοθέτες κλπ απολαμβάνοντας την αξιοσημείωτα ποιοτική και γρήγορη ανταπόκριση του server παρά το γεγονός ότι οι αναζητήσεις γίνονται σε εκατομμύρια εγγραφές δεδομένων. Σε κάθε ταινία υπάρχει ένα “People who liked this also like:” κομμάτι το οποίο προκύπτει από την καταγραφή και επεξεργασία των ιχνών άλλων επισκεπτών. Το IMDb για να είναι σε θέση να κάνει προτάσεις στον εκάστοτε χρήστη του, του ζητά να βαθμολογήσει μερικές ταινίες ώστε να βγουν κάποια - σχετικώς – ασφαλή συμπεράσματα. Αυτά τα εμπλουτίζει με τα στοιχεία

[1] www.amazon.com

[2] www.youtube.com

[3] www.imdb.com

που προκύπτουν από τις ταινίες τις οποίες επιλέγει στην πορεία ώστε να σκιαγραφήσει ακόμα καλύτερα το προφίλ του.

Ο μέσος χρήστης (όπως είναι φυσικό) θα κρίνει την βαθμό επιτυχίας ή αποτυχίας ενός συστήματος εξατομίκευσης από το αποτέλεσμα. Εν προκειμένω εάν μιλάμε για ταινίες, η επιθυμία του ενδεχομένως θα είναι να λάβει προτάσεις ταινιών από το σύστημα που να τον ικανοποιούν την συγκεκριμένη χρονική στιγμή, στον χώρο στον οποίο βρίσκεται, με την διάθεση την οποία έχει. Όλα αυτά κατά προτίμηση χωρίς να μπει στην χρονοβόρα και κουραστική διαδικασία να «εξηγεί» στο σύστημα λεπτομερώς σε τι αρέσκεται. Όσο και αν κάτι τέτοιο αρχικά φαντάζει πολύ δύσκολο έως σχεδόν αδύνατο - από την πλευρά του μηχανικού λογισμικού – συναντάμε συστήματα στα οποία έως ένα βαθμό έχει επιτευχθεί. Το Jinni[4] είναι ίσως η καλύτερη μηχανή παραγωγής συστάσεων ταινιών. Προσφέρει την δυνατότητα πολλαπλών, πολυμορφικών αναζητήσεων βάσει απλών (πχ. είδος ταινίας, χρονική περίοδος) αλλά και σύνθετων (πχ διάθεση του χρήστη) κριτηρίων, καθώς και συνδυασμό όλων αυτών.

1.1.2 Κατηγοριοποίηση Συστημάτων Εξατομίκευσης

Καθίσταται φανερό το ότι οι πληροφορίες τις οποίες εκμεταλλεύονται τα συστήματα εξατομίκευσης ποικίλλουν. Κατά κανόνα, όσο πιο πολυμορφικά είναι τα data που αφουγκράζεται ένα τέτοιο σύστημα, τόσο περισσότερες δυνατότητες ορθής παραγωγής προτάσεων έχει. Δεν υφίστανται περιορισμοί ούτε στον όγκο των δεδομένων αυτών, αλλά ούτε και στο είδος τους. Στην πραγματικότητα πρόκειται για ένα υποσύνολο μιας άλλης γενικότερης έννοιας, που ονομάζεται context.

Η έννοια του context είναι σχετικά αφηρημένη. Περιλαμβάνει εν ολίγοις τα πάντα σχετικά με μια κατάσταση και τους χρήστες που την απαρτίζουν, όπως το περιβάλλον του ίδιου του χρήστη – δηλαδή η τοποθεσία που βρίσκεται, τα αντικείμενα γύρω του - το περιβάλλον και η φύση της υπολογιστικής συσκευής μέσω της οποίας γίνεται η σύνδεση – υπολογιστής laptop/ desktop, smart phone, tablet-PC -, εξωγενείς παράγοντες όπως οι καιρικές συνθήκες, εάν είναι νύχτα ή μέρα και βέβαια τα ενδιαφέροντα του χρήστη (στην παρούσα διπλωματική εστιάζουμε στο τελευταίο). Ακολουθεί μία απόπειρα αυστηρού ορισμού της έννοιας του context:

Ως context ορίζεται κάθε πληροφορία που μπορεί να χρησιμοποιηθεί για την περιγραφή της κατάστασης μια οντότητας, όπου οντότητα θεωρούνται τα άτομα, οι τοποθεσίες και τα αντικείμενα που θεωρούνται σχετικά με την αλληλεπίδραση ανθρώπου-εφαρμογής, του ανθρώπου και της εφαρμογής συμπεριλαμβανομένων.[DA00]

[4] www.jinni.com

Ως εκ τούτου γίνεται αντιληπτό ότι τα συστήματα εξατομίκευσης (preference-aware) αποτελούν μια κάπως εξειδικευμένη υποκατηγορία των context-aware συστημάτων, δίχως όμως την επιβολή περιοριστικών μέτρων καθότι είναι θεμιτό να συλλέξουν ως δεδομένα οποιοδήποτε στοιχείο υπάρχει στο context.

Είναι εύλογο, λόγω της πολυπλευρικότητας των preferences-aware systems, να υπάρχουν πολλών ειδών κατηγοριοποιήσεις.

A. Content-based vs Collaborative recommendations

Σύμφωνα με τα [AT05], [ASS+05] τα preferences-aware systems χωρίζονται σε Content-based, Collaborative και Hybrid, ανάλογα με το κατά πόσον βασίζουν τα αποτελέσματά τους σε στοιχεία που αφορούν τον ίδιο τον χρήστη ή άλλους χρήστες με παρεμφερή ενδιαφέροντα.

- Content-based recommendations

  Προτείνονται στον χρήστη αντικείμενα παρόμοια με αυτά που ο ίδιος έδειξε να προτιμά στο παρελθόν. Η content-based προσέγγιση έχει τις ρίζες της στο Information retrieval και Information filtering. Η βελτίωση που έχει γίνει πάνω σε αυτά είναι η χρήση του προφίλ χρήστη το οποίο περιέχει πληροφορίες σχετικά με τις προτιμήσεις και τις ανάγκες του.

  Η μέθοδος χρησιμοποιεί και το προφίλ του αντικειμένου που περιέχει στοιχεία που το χαρακτηρίζουν. Για παράδειγμα αυτά μπορεί να είναι τίτλος, λέξεις κλειδιά (keywords), σύντομη περιγραφή κα.

  Ο βαθμός της χρησιμότητας ενός αντικειμένου για έναν χρήστη προκύπτει από την εφαρμογή ενός αλγορίθμου (όπως ο cosine similarity) ο οποίος έχει 2 εισόδους: το προφίλ του χρήστη και αυτό του αντικειμένου.

- Collaborative filtering recommendations

  Αντίθετα με τις content-based μεθόδους, οι collaborative μέθοδοι επιχειρούν να προβλέψουν την χρησιμότητα των αντικειμένων για έναν συγκεκριμένο χρήστη [A] με βάση τα αντικείμενα που έχουν αξιολογηθεί παλαιότερα από «παρόμοιους» χρήστες, δηλαδή από χρήστες που έχουν βαθμολογήσει τα αντικείμενα με τρόπο παρεμφερή με τον χρήστη [A].

  Ο κάθε χρήστης εκπροσωπείται από ένα item vector όπου τα elements είναι τα διάφορα αντικείμενα (items) και οι τιμές των elements είναι η βαθμολογία (rating) του χρήστη για κάθε item.

Οι αλγόριθμοι που χρησιμοποιούνται ανήκουν σε 2 κατηγορίες:

1. Memory-based (heuristics)

   Η πρόβλεψη της βαθμολόγησης (rating) ενός χρήστη [B] για ένα αντικείμενο s, συνήθως υπολογίζεται από το σύνολο των ratings για το s αντικείμενο που έχουν δώσει κάποιοι άλλοι –συνήθως οι N περισσότερο όμοιοι με τον [B]-χρήστες (μέθοδος k-nearest neighbor). Η ομοιότητα μεταξύ δύο χρηστών υπολογίζεται με αλγορίθμους όπως: vector cosine, pearson correlation, ευκλείδια απόσταση κ.ά.

2. Model-based

   Χρησιμοποιούν ένα μοντέλο πιθανοτήτων προκειμένου να βρεθούν "patterns" με βάση πλασματικά (αλλά αληθοφανή) δεδομένα. Τα patterns αυτά χρησιμοποιούνται στην συνέχεια για να γίνουν προβλέψεις σε αληθινά δεδομένα.

- Υβριδικές προσεγγίσεις

  Με τις Content-based μεθόδους διατρέχεται ο κίνδυνος ο χρήστης να λαμβάνει προτάσεις για περιορισμένα αντικείμενα που είναι πολύ παρεμφερή με αυτά που επέλεξε στο παρελθόν. Από την άλλη μεριά, με την χρήση Collaborative filtering, υπάρχει το ενδεχόμενο να προτείνονται αντικείμενα με βάση τυχαία ευνοημένα γεγονότα, με αποτέλεσμα την αδυναμία του συστήματος για σωστά αποτελέσματα. Οι Υβριδικές προσεγγίσεις συνδυάζουν τις content-based με τις collaborative filtering και επιτυγχάνουν να συγκεράσουν τα πλεονεκτήματά τους και να εξαλείψουν –στο μέτρο του δυνατού- τα μειονεκτήματά τους.

Β. Ποιοτική vs Ποσοτική προσέγγιση

Μία άλλου είδους κατηγοριοποίηση εστιάζει στον τρόπο αναπαράστασης, έκφρασης των προτιμήσεων. Συναντάμε την ποιοτική και την ποσοτική προσέγγιση.

- Ποιοτική προσέγγιση (qualitative method)

  Στην ποιοτική προσέγγιση η κάθε προτίμηση εκφράζεται ως σύγκριση μεταξύ δύο ή περισσότερων επιλογών. Για παράδειγμα «Μου αρέσουν περισσότερο οι ταινίες δράσης από τις κομεντί». Η εφαρμογή στις εγγραφές της βάσης γίνεται συγκρίνοντας τις τιμές των αντίστοιχων attributes και δίνοντας προτεραιότητα σε αυτές που θεωρούνται προτιμητέες σύμφωνα με τα δηλωθέντα ενδιαφέροντα του χρήστη. Το σύστημα επιχειρεί επιπλέον να παράγει επαγόμενες προτιμήσεις συνδυάζοντας τις υπάρχουσες. Σε συνέχεια του προηγούμενου παραδείγματος, εάν δηλωθεί ότι

«Προτιμώ τις κομεντί από τα θρίλερ», το σύστημα θα εξάγει το συμπέρασμα ότι ο χρήστης αρέσκεται περισσότερο στις ταινίες δράσης από τα θρίλερ. Βέβαια, η αποτίμηση όλων των επαγόμενων προτιμήσεων, αλλά και η αναζήτηση των σχετικών με το δοθέν ερώτημα δεν είναι εύκολη, καθώς η εκφραστικότητα μιας τέτοιας προσέγγισης είναι απεριόριστη.

Τα εξατομικευμένα αποτελέσματα που προκύπτουν από αυτή την μέθοδο είναι ασφαλή, όμως δεν μπορεί να υπάρξει μία γενική κατάταξη των αντικειμένων εξαιτίας της φύσης της ίδιας της σύγκρισης.

- Ποσοτική προσέγγιση

  Αντίθετα, στην ποσοτική προσέγγιση η εκάστοτε προτίμηση εκφράζεται στις εγγραφές μέσω ενός score. Με βάση την τιμή ενός attribute, προσδιορίζεται το κατά πόσον ικανοποιείται πλήρως, μερικώς ή καθόλου και με την χρήση μιας scoring function λαμβάνει την αρμόζουσα βαθμολογία.

  Με αυτή την μέθοδο κάθε πλειάδα μπορεί να έχει πολλές διαφορετικές βαθμολογίες, μια για κάθε προτίμηση της οποίας ικανοποιεί την συνθήκη. Η τελική κατάταξη προκύπτει συναθροίζοντας τις βαθμολογίες. Με την καταγραφή ικανοποιητικού αριθμού προτιμήσεων του χρήστη, η συγκεκριμένη μέθοδος έχει την δυνατότητα να επιστρέψει σχετικώς επιτυχημένα αποτελέσματα στο σύνολο των εγγραφών – σε αντίθεση με την ποιοτική προσέγγιση η οποία εστιάζει μόνο στα υπό σύγκριση αντικείμενα -.

Γ. Plug-in approach vs preference-aware database

Ο συγκεκριμένος διαχωρισμός (ο οποίος περιγράφεται στο [Kou11]) έχει να κάνει με το κατά πόσον οι προτιμήσεις υπεισέρχονται στην σχεσιακή δομή της βάσης ή όχι.

- Plug-in (on-top)

  Οι plug-in προσεγγίσεις είναι εφαρμογές σε γλώσσα υψηλού επιπέδου οι οποίες παράγουν recommendations έξω από την database engine. Αυτό συνήθως γίνεται μεταφράζοντας το απλό SQL query σε preference-aware query και εκτελώντας το στο DBMS. Αναλυτικότερα, η διαδικασία περιλαμβάνει 3 βήματα:

  1) Query rewriting: Οι προτιμήσεις ενσωματώνονται στο δοθέν query σαν απλές SQL conditions, δημιουργώντας έτσι ένα νέο set από queries.

  2) Materialization: Εκτελούνται τα νέα queries.

  3) Aggregation: Τα επί μέρους αποτελέσματα συνδυάζονται σε μία βαθμολογημένη λίστα.

Άλλος τρόπος plug-in προσέγγισης είναι με την χρήση ειδικών αλγορίθμων εκτίμησης (evaluation algorithms). Οι συγκεκριμένοι εκτελούνται παίρνοντας σαν είσοδο το αποτέλεσμα του SQL query, το οποίο διαμορφώνουν ανάλογα με τις προτιμήσεις του χρήστη.

Σε γενικές γραμμές, οι plug-in μέθοδοι, εξαιτίας της φύσης τους να μην εμπλέκονται στον πυρήνα της βάσης, παρουσιάζουν αδυναμία στο να παράξουν ευέλικτες, πολύπλευρες συστάσεις.

- Preference-aware database (built-in)

  Από την άλλη μεριά, προτείνεται η επεξεργασία και αποτίμηση των συστάσεων να γίνεται μέσα στην βάση. Αυτό γίνεται με την χρήση preference-aware operators, οι οποίοι προσομοιώνουν τους κανονικούς SQL operators εμπλουτίζοντάς τους με στοιχεία που εκφράζουν τις προτιμήσεις του χρήστη. Κατ' αυτόν τον τρόπο η βάση «γνωρίζει» τα ενδιαφέροντα του χρήστη.

.

1.2 Αντικείμενο διπλωματικής

Αντικείμενο της παρούσας διπλωματικής είναι ο Σχεδιασμός και η Υλοποίηση ενός Υποσυστήματος Εκτέλεσης Ερωτημάτων για το Σύστημα Εξατομικευμένης Διαχείρισης Βάσεων Δεδομένων PrefSQL [AK11]. Πρόκειται για ένα Content-based, preference-aware database σύστημα με ποσοτική προσέγγιση.

Συνοψίζεται η συνεισφορά της διπλωματικής:

1. Μελετήσαμε την PrefSQL ως επέκταση της SQL.
2. Υλοποιήσαμε τους operators που περιγράφονται στην PrefSQL.
3. Εξετάσαμε και υλοποιήσαμε τα δυο αποδοτικότερα πλάνα εκτέλεσης.
4. Μελετήσαμε την συμπεριφορά τους και εξάγαμε συμπεράσματα.
5. Σχεδιάσαμε και υλοποιήσαμε ιστότοπο όπου ένας χρήστης πλοηγείται, δέχεται συστάσεις, υποβάλει ερωτήματα και παίρνει αποτελέσματα βαθμολογημένα σύμφωνα με τις προτιμήσεις του.

1.3 Οργάνωση κειμένου

Ο τόμος αποτελείται από 8 κεφάλαια που καλύπτουν πλήρως την ανάπτυξη της διπλωματικής εργασίας.

Στο **δεύτερο κεφάλαιο** γίνεται μια εκτενής αναφορά στην υπάρχουσα βιβλιογραφία σχετικά με τις έννοιες του preference και του preference-aware-databases, ενώ ταυτόχρονα περιγράφονται σχετικές με το θέμα εργασίες.

Στο τρίτο κεφάλαιο περιγράφεται το μοντέλο προτιμήσεων (Preference Model) στο οποίο βασίστηκε η υλοποίηση του PrefSQL συστήματος.

Στο **τέταρτο κεφάλαιο** γίνεται αναφορά στην αρχιτεκτονική του συνολικού συστήματος, και των επιμέρους υποσυστημάτων. Στη συνέχεια γίνεται αναλυτική παρουσίαση των λειτουργιών κάθε υποσυστήματος.

Στο **πέμπτο κεφάλαιο** γίνεται η περιγραφή της υλοποίησης του συστήματος. Περιγράφονται αναλυτικά οι PL/pgSQL operators και functions, η Διεπαφή χρήστη καθώς και η πλατφόρμα και τα προγραμματιστικά εργαλεία.

Το **έκτο κεφάλαιο** πραγματεύεται τον έλεγχο του συστήματος. Παρουσιάζεται ο έλεγχος ορθότητας των τελεστών προτίμησης, γίνεται σύγκριση των τελεστών προτίμησης, παρουσιάζονται μετρήσεις με διαφορετικά πλάνα εκτέλεσης ερωτημάτων και τέλος αναπτύσσεται ένα σενάριο μέσω του οποίου γίνεται ο έλεγχος της Διεπαφής του χρήστη.

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

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

2 *Σχετικές εργασίες*

Είναι γεγονός ότι τα εξατομικευμένα συστήματα έχουν απασχολήσει σε μεγάλο βαθμό την πανεπιστημιακή, ερευνητική κοινότητα παγκοσμίως, με αποτέλεσμα την πληθώρα προτάσεων μεθόδων σχεδιασμού και υλοποίησής τους. Προτού δημιουργήσουμε το PrefSQL μοντέλο, μελετήσαμε πολλές εξ' αυτών. Σκοπός μας ήταν να παραδειγματιστούμε από τα δυνατά τους σημεία και να καλύψουμε τις ελλείψεις και αδυναμίες τους.

Στο παρόν κεφάλαιο παρουσιάζουμε τρία συστήματα η λογική των οποίων είναι πιο κοντά σε αυτήν του PrefSQL και τα οποία είχαν την μεγαλύτερη επιρροή στον σχεδιασμό του.

2.1 FlexRecs

Το FlexRecs [KBG09] είναι ένα σύστημα το οποίο διαμορφώνει την διαδικασία παραγωγής συστάσεων πάνω από την βάση δεδομένων. Προσαρμόζεται εύκολα σε οποιαδήποτε τύπου –σχεσιακή- βάση και προσφέρει ευελιξία στις συστάσεις του. Η εξατομίκευση υλοποιείται με ένα παραμετροποιήσιμο workflow, το οποίο αποτελείται από τους γνωστούς, υπάρχοντες σχεσιακούς τελεστές σε συνδυασμό με κάποιους νέους.

Το workflow ουσιαστικά είναι μία αλληλουχία ενεργειών, η οποία ουσιαστικά εκφράζει τις προτιμήσεις του χρήστη. Αυτή παράγεται από το σύστημα έξω από το DBMS, για να μεταφραστεί ύστερα σε SQL εντολές και να εκτελεστεί εντέλει από την database engine.

Η εξαγωγή συμπερασμάτων βασίζεται στην σύγκριση μεταξύ των tuples. Για να μπορούν να γίνουν αποτελεσματικές και γρήγορες συγκρίσεις, εισάγεται η έννοια της extended σχέσης (επεκτεταμένη σχέση-δηλαδή πίνακας-). Μία extended σχέση έχει ένα attribute το οποίο εμπεριέχει πολλές τιμές για κάθε tuple, οι οποίες έχουν ληφθεί από μία άλλη σχέση. Ως εκ τούτου, για να δημιουργηθεί μία extended σχέση χρειάζονται δύο άλλες σχέσεις: Αυτή με την οποία θα έχει τις ίδιες οντότητες, η λεγόμενη base (βασική) σχέση και αυτή από την οποία θα

αντληθούν οι εξτρά πληροφορίες για τα tuples της πρώτης. Προς κατανόηση αυτών, παρατίθεται το παρακάτω παράδειγμα.

Departments(DepID, DepCode, Name)
Courses(CourseID, DepID, Title, Description, Units, Url)
CourseSched(CourseID, Year, Term, InstrID, Location,TimeSlot, Days)
Instructors(InstrID, Name, Url)
Students(SuID, Name, Class, GPA, Status)
StudentStudies(SuID, StudyPrgID)
StudyPrograms(StudyPrgID, ProgramName, Classification,DepID)
StudentHistory(SuID, CourseID, Year, Term, Grade, Rating)
Comments(SuID, CourseID, Year, Term, Text, Rating,Date)

Εικόνα 1: Μία σχεσιακή βάση

Η συγκεκριμένη σχεσιακή βάση αφορά φοιτητές οι οποίοι καλούνται να δηλώσουν μαθήματα. Το σύστημα επιχειρεί να τους βοηθήσει να επιλέξουν αυτά που πραγματικά θα τους ενδιέφεραν. Για να είναι να δυνατό να γίνουν συγκρίσεις με βάση τις προτιμήσεις του κάθε φοιτητή, πρώτο στάδιο είναι να εξαχθεί μία extended σχέση από την Students, η οποία συγκεντρώνει (σαν ένα view) όλα τα σχόλια κάθε φοιτητή σχετικά με μαθήματα που έχει παρακολουθήσει:

Students

SuID	Name	Class	Status
1	Paul Little	2009	H
2	John Doe	2010	N

Ext_Students

SuID	Name	Comments (CourseID, Rating, Date)		
1	Paul Little	C1	5	2 Feb 2008
		C2	6	3 Dec 2007
2	John Doe	C1	5	15 Mar 2007
		C2	6	12 Dec 2007
		C5	6.6	22 Jun 2007
		C7	7	22 Jun 2007

Εικόνα 2: Μία base και μία extended σχέση

Επιτρέπεται μόνο ένα επίπεδο τέτοιου είδους εμφωλιασμού δεδομένων και είναι είναι προϊόν ενός νέου operator. Ακολουθεί περιγραφή των νέων operators:

- *Extend (ε)* : Δημιουργεί μία extended σχέση από δύο άλλες οι οποίες έχουν ένα κοινό attribute για να γίνει join.

  $R_i \varepsilon R_j \rightarrow H\ R_i$ εμπεριέχει την R_j (η R_j είναι βασική σχέση).

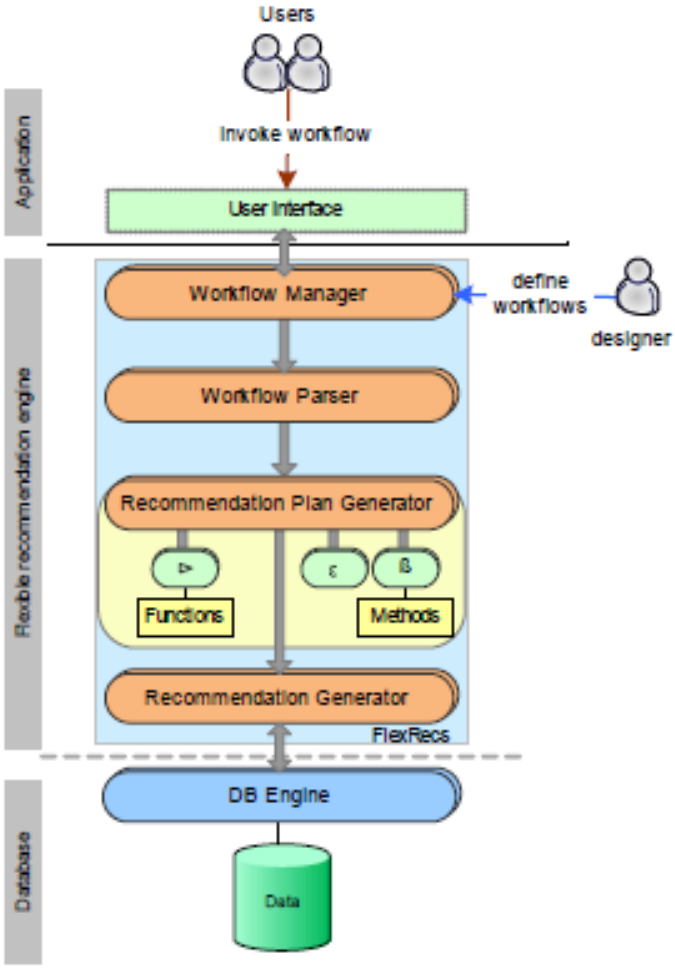
- *Recommend ($\triangleright$ cf,a)* : Παίρνει σαν είσοδο δύο σχέσεις, συγκρίνει τα tuples τους με την *cf* (comparison function –συνάρτηση σύγκρισης-), υπολογίζεται το τελικό score αυτών με την *a* και επιστρέφει το αποτέλεσμα.

$R_i \triangleright_{cf,a} R_j$ → Προκύπτουν τα tuples της R_i, επαυξημένα με το attribute score. Το score κάθε tuple προκύπτει ως εξής: Η *cf* συνάρτηση συγκρίνει ένα attribute της ίδιας της εγγραφής με όλες τις εγγραφές της R_j. Αναλόγως το αποτέλεσμα της σύγκρισης, η *a* συνάρτηση βαθμολογεί και βάζει την κατάλληλη τιμή στο score.

- *Blend* (β_M) : Συνδυάζει tuples που έχουν προκύψει από εκτελέσεις της Recommend, συναθροίζοντας τα score τους σε ένα bscore (blended score) attribute.

Με το κατάλληλο workflow (το οποίο θα αξιοποιεί τις λειτουργίες των τριών αυτών τελεστών, σε συνδυασμό με τους κλασικούς SQL τελεστές), μπορεί να εκφραστεί οποιαδήποτε προτίμηση – απλή ή σύνθετη - του χρήστη.

Η αρχιτεκτονική του FlexRecs συστήματος είναι διαμορφωμένη κατάλληλα ώστε το σύστημα να παράγει τα workflows έξω από την DB engine.



Εικόνα 3: Αρχιτεκτονική του FlexRecs

Η αρχιτεκτονική του συγκεκριμένου συστήματος είναι από τις πιο ολοκληρωμένες που έχουν σχεδιαστεί σε εξατομικευμένα συστήματα. Σε κάθε στάδιο εκτελούνται σαφώς ορισμένες διεργασίες, οι οποίες στο σύνολό τους επιτυγχάνουν ένα πλήρως εναρμονισμένο αποτέλεσμα.

Κατά την σχεδίαση του PrefSQL επηρεαστήκαμε σε υψηλό βαθμό από την αρχιτεκτονική του FlexRecs.

Το workflow αντιστοιχεί στο δικό μας πλάνο εκτέλεσης του ερωτήματος. Ενώ το workflow καταστρώνεται από τον designer του FlexRecs, στην PrefSQL το query plan καθορίζεται από το Υποσύστημα Διαμόρφωσης Ερωτημάτων.

Τέλος, όπως θα δούμε αργότερα, υλοποιήσαμε έναν operator που εκφέρει ομοιότητες με τον Recommend operator του FlexRecs.

2.2 Preference SQL

Το σύστημα Preference SQL [KEW11] ενσωματώνει στα σχεσιακά ερωτήματα (Select Statements) τις προτιμήσεις, με τρόπο παρόμοιο με αυτόν του PrefSQL. Ένα Preference SQL query έχει το εξής τρόπο σύνταξης:

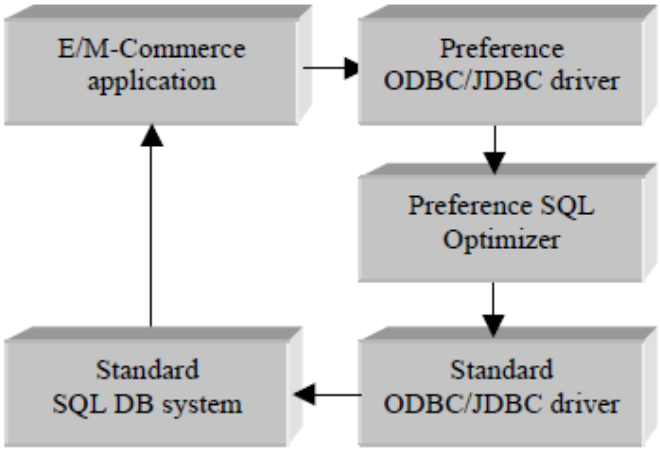
```
SELECT       ...  <selection>
FROM         ...  <table_reference>
WHERE        ...  <hard_conditions>
 PREFERRING  ...  <soft_conditions>
 GROUPING    ...  <attribute_list>
 BUT ONLY    ...  <but_only_condition>
GROUP BY     ...  <attribute_list>
HAVING       ...  <hard_conditions>
ORDER BY     ...  <attribute_list>
```

Εικόνα 4: Ορισμός ερωτήματος σε Preference SQL

Εισάγονται οι τελεστές *"PREFERING", "GROUPING" και "BUT ONLY"*.

- *PREFERRING:* Εφαρμόζεται το preference στα αποτελέσματα που προκύπτουν από την *WHERE* συνθήκη.
- *GROUPING*: Δηλώνονται τα attributes που αφορούν την preference.
- *BUT ONLY*: Για να εφαρμοστεί το preference σε μία εγγραφή, αυτή πρέπει να ικανοποιεί, πέρα από την *WHERE,* και την *BUT ONLY* συνθήκη.

Κατόπιν εκτέλεσης των παραπάνω, με την επιλογή συγκεκριμένου αλγορίθμου για την εξαγωγή των βέλτιστων αποτελεσμάτων, παρουσιάζονται στον χρήστη τα επικρατέστερα αποτελέσματα. Σε κάθε περίπτωση, το σύστημα γνωρίζει ποιό ήταν το χαρακτηριστικό μιας πλειάδας που την έκανε να επικρατήσει απέναντι σε κάποια άλλη.



Εικόνα 5: Αρχιτεκτονική του Preference SQL

Προκειμένου να ρυθμιστεί η χαλαρότητα του ερωτήματος ως προς τις προτιμήσεις του χρήστη, το Preference SQL χρησιμοποιεί soft conditions (*PREFFERING*) και hard conditions (*BUT ONLY*). Οι πρώτες είναι επιθυμητές τιμές, ενώ οι δεύτερες είναι απαραίτητες. Η PrefSQL, όπως θα δούμε παρακάτω, έχει σαν αντίστοιχες περιοριστικές λειτουργίες τις soft conditions οι οποίες ορίζονται στις preferences και τις hard conditions οι οποίες στο *"WHERE"* του select statement στο οποίο εφαρμόζονται οι preferences. Επιπροσθέτως, ο βαθμός χαλαρότητας, ελέγχεται από τον βαθμό εμπιστοσύνης. Αυτό είναι ένα μέτρο για το πόσο σίγουρο μπορεί να είναι το σύστημα ότι το αποτέλεσμα όντως ανταποκρίνεται στις προτιμήσεις του χρήστη. Το PrefSQL αντιλαμβάνεται πόσο ασφαλείς είναι οι επαγωγές που κάνει στο σύνολό τους, οπότε μπορεί να περιορίσει την απόκλιση των αποτελεσμάτων από το επιθυμητό, αλλά όχι να το κάνει σε συγκεκριμένα χαρακτηριστικά όπως το Preference SQL. Άρα λοιπόν, το PrefSQL υπολογίζει με μία αμιγώς ποσοτική προσέγγιση τα καλύτερα αποτελέσματα, ενώ το Preference SQL χρησιμοποιεί και κάποια κριτήρια που παραπέμπουν σε ποιοτική προσέγγιση.

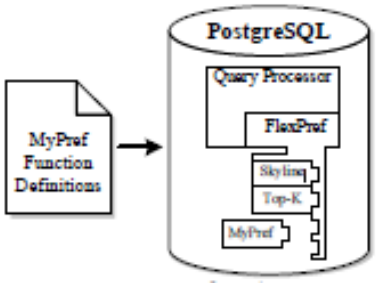
Τέλος, το Preference SQL τρέχει μετατρέποντας τα ερωτήματα με προτιμήσεις σε απλά σχεσιακά, ενώ το PrefSQL αλλάζει τον τρόπο με τον οποίο γίνεται η εκτέλεση του ερωτήματος στην βάση.

2.3 FlexPref

Το FlexPref [LMK10] είναι ένα παραμετροποιήσιμο σύστημα συστάσεων, το οποίο έχει εισχωρήσει μέσα στην execution engine της postgesql. Οι συναρτήσεις αποτίμησης προτιμήσεων έχουν οριστεί μέσα στο DBMS. Χρησιμοποιεί την λογική του κλαδέματος (pruning), σύμφωνα με την οποία κατά την διάρκεια της αποτίμησης των αποτελεσμάτων εάν

ένα tuple κρίνεται –λόγω απόκλισης τιμής από το επιθυμητό – ότι δεν έχει τις προδιαγραφές να αποτελέσει προϊόν σύστασης, απορρίπτεται εξ' αρχής.

Το σύστημα για να λειτουργήσει πρέπει πρώτα να οριστεί ένας αλγόριθμος αποτίμησης ερωτημάτων (πχ. skyline, top-k, k-dominance) που προτιμά ο χρήστης. Με τον ορισμό αυτό, ο αλγόριθμος περνάει «μέσα» στην database engine και γίνεται compile. Από την στιγμή που γίνει αυτό, αποτελεί κομμάτι του συστήματος και μπορεί να κληθεί οποτεδήποτε.



Εικόνα 6: Αρχιτεκτονική του FlexPref

Οι βασικές συναρτήσεις είναι τρεις:

- *PairwiseCompare(Object P, Object Q)*: Η συνάρτηση παίρνει σαν είσοδο τα αντικείμενα *P* και *Q*. Αρχικά ενημερώνει το score attribute του *P*. Κατόπιν, επιστρέφει 1 εάν το *Q* δεν δύναται –βάση score- να γίνει προτιμητέο, -1 εάν κάτι τέτοιο ισχύει για το *P* και 0 εάν δεν ισχύει τίποτα από τα δύο.
- *IsPreferredObject(Object P, PreferenceSet S):* Εισάγονται ένα αντικείμενο *P* και ένα πλήθος από προτιμητέα αντικείμενα *S*. Επιστρέφεται true εάν κρίνει ότι το *P* είναι και αυτό προτιμητέο και μπορεί να μπει στο *S*, διαφορετικά false.
- *AddPreferredToSet(Object P, PreferenceSet S):* Η συνάρτηση παίρνει σαν είσοδο ένα αντικείμενο *P* και ένα πλήθος προτιμητέων αντικειμένων *S*. Προσθέτει το *P* στο *S* και εάν αυτό είναι πλήρες διαγράφει κάποιο πρώην μέλος του.

Ομοίως με το Preference SQL, το FlexPref εισάγει νέους τελεστές με τους οποίους εμπλουτίζει την σύνταξη των SQL ερωτημάτων. Ένα select statement ορίζεται ως εξής:

```
Select [Select Clause]
From [Tables]
Where [Where Clause]
Preferring [Preference Attributes]
Using [method] With [Parameter]
Objectives [Objective]
```

Εικόνα 7: Ορισμός ερωτήματος σε FlexPref

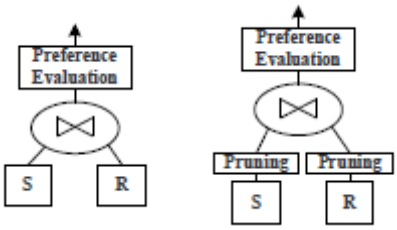
Εισάγονται οι τελεστές "*Preferring", "Using – With"*, *Objetives.*

- *Preferring*: Δηλώνονται τα attributes των σχέσεων τα οποία αποτελούν κριτήρια για προτιμήσεις.

- *Using – With*: Δηλώνεται ο αλγόριθμος αποτίμησης που θέλουμε να εκτελεστεί (εφόσον βέβαια έχει οριστεί στο σύστημα). Μετά το *"With"* δίνεται η παράμετρος του αλγορίθμου (εφόσον είναι απαραίτητη, πχ. Using Top-k WITH k=2).
- *Objectives*: Δίνονται οι ελάχιστες (ή μέγιστες ανάλογα με την φύση των προτιμήσεων) τιμές που επιθυμούμε να έχουν τα attributes που δηλώσαμε στο *Preferring*.

Αναλόγως με το αν θέλουμε να εξάγουμε προτιμήσεις από έναν μόνο table ή από περισσότερους – οπότε και θα χρειαστεί να γίνει join -, το σύστημα καλεί τον κατάλληλο αλγόριθμο εκτέλεσης των παραπάνω τριών συναρτήσεων με τις παραμέτρους που δόθηκαν.

Σε κάθε στάδιο, συγκρίνει τις τιμές των attributes που έχουν δηλωθεί στο *Preferring* με τις επιθυμητές τιμές *(Objectives)* και εάν δεν είναι μέσα στα όρια αυτών, απορρίπτει τις εγγραφές. Επιπλέον, προσπαθεί, εκμεταλλευόμενο την μεταβατική φύση των attributes – εφόσον αυτή υπάρχει -, να «κλαδέψει» εγγραφές αποφεύγοντας όσες περισσότερες συγκρίσεις είναι δυνατό.



Εικόνα 8:
Join χωρίς κλάδεμα – Join με κλάδεμα

Το κλάδεμα αποσκοπεί στην αποβολή των εγγραφών που δεν ικανοποιούν τις προτιμήσεις. Όταν αυτό γίνεται στα ενδιάμεσα στάδια εκτέλεσης, αποσκοπεί επιπλέον και στην βελτίωση της ταχύτητας των queries αφού το DBMS αποφορτίζεται από την επεξεργασία άχρηστου όγκου δεδομένων. Το κλάδεμα, στην παρούσα υλοποίησή του, αποτελεί μία μέθοδο ελέγχου της χαλαρότητας των ερωτημάτων.

Το μεγάλο πλεονέκτημα του FlexPref είναι η μεγάλη προσαρμοστικότητά του καθότι έχει την δυνατότητα να φορτώσει, εκτελέσει οποιονδήποτε αλγόριθμο αποτίμησης χωρίς καμία παρεμβολή του σχεδιαστή του συστήματος. Βεβαίως, αυτό γίνεται με ένα κόστος στον χρόνο εκτέλεσης. Γενικώς είναι πιο γρήγορο από τις plug-in μεθόδους, αλλά πιο αργό από τις built-in.

Το PrefSQL σύστημα, από την μεριά του, προσφέρει επίσης την δυνατότητα εκτέλεσης διαφορετικών αλγορίθμων αποτίμησης. Η βασική διαφορά μεταξύ των δύο συστημάτων έγκειται στο ότι το FlexPref είναι ουσιαστικά ένα πλαίσιο (framework) παραγωγής συστάσεων πάνω στο οποίο μπορούν να βασιστούν άλλα συστήματα (πχ το CareDB

[LM10]), ενώ αντιθέτως το PrefSQL είναι ένα ολοκληρωμένο σύστημα εξατομίκευσης (κάτι το οποίο φυσικά δεν το εμποδίζει να συνυπάρξει με άλλα υποσυστήματα).

3 Θεωρητικό υπόβαθρο

Στο παρόν κεφάλαιο παρουσιάζεται λεπτομερώς το μοντέλο προτιμήσεων (Preference Model) το οποίο ορίζεται στο [AK11], πάνω στο οποίο βασίζεται η ανάλυση, σχεδίαση και υλοποίηση του PrefSQL συστήματος.

Στο συγκεκριμένο μοντέλο εξατομίκευσης, οι προτιμήσεις εισχωρούν ως οντότητα μέσα στην βάση. Εφαρμόζοντας μία ποσοτική προσέγγιση, αποτυπώνεται στις εγγραφές το κατά πόσον αυτές ικανοποιούν ή όχι τις δηλωθείσες προτιμήσεις του χρήστη. Σκοπός είναι η μοντελοποίηση ορισμένων παραμέτρων που δεν είχαν υλοποιηθεί σε άλλα παρόμοια συστήματα και η βελτιστοποίηση της εκτέλεσης αυτών των ερωτημάτων.

Αναφερόμαστε πάντοτε σε σχεσιακή βάση δεδομένων.

3.1 Αξιολόγηση αντικειμένων και ορισμός προτίμησης (preference)

Έστω $R = \{A1,A2,\ldots,An\}$ μία σχέση και r μια πλειάδα αυτής. Θα συμβολίζεται με $r.Aj$ η τιμή του r για την ιδιότητα Aj. Επίσης, το πεδίο τιμών του Aj θα συμβολίζεται με $dom(Aj)$.

Ως συνάρτηση αξιολόγησης (scoring function) ορίζεται μια συνάρτηση $f: dom(A1)xdom(A2)x\ldots xdom(Ak) \rightarrow [0,1]$. Παραδείγματος χάριν, η συνάρτηση $fr(x1) = 0.1 * x1$ δέχεται τιμές από το 1 έως το 10 (τη βαθμολογία για κάποια ταινία), και επιστρέφει μια τιμή στο διάστημα [0,1]. Γενικώς τα ορίσματα των συναρτήσεων αξιολόγησης είναι attributes του σχεσιακού σχήματος.

Προκειμένου να εκφραστεί το κατά πόσον οι εγγραφές μιας σχέσης ικανοποιούν τις προτιμήσεις του χρήστη, εισάγονται δύο μεγέθη: i) το Score S, με $dom(S) = [0,1] \cup \{\perp\}$, το οποίο αποτυπώνει τον βαθμό εκπλήρωσης , ή μη, των προτιμήσεων και ii)το *Conf C*, με $dom(C) = \Re^+$, το οποίο εκφράζει την ασφάλεια , ή μη, της εκτίμησης του *Score* (κατά

πόσον είναι έμπιστη η πηγή/οι πηγές από όπου προέκυψε η εκτίμηση). Σαν πρώτο βήμα, προστίθενται σε όλες τις σχέσεις ως attributes τα *Score, Conf* και αρχικοποιούνται.

Μία προτίμηση (preference) p η οποία αφορά τις εγγραφές *r* μιας σχέσης *R*, ορίζεται ως μία τριάδα στοιχείων *(σφ, S, C).*

Το *σφ* συμβολίζει την συνθήκη η οποία απαιτείται να ικανοποιεί η τιμή ενός attribute μίας r_i εγγραφής ώστε να θεωρηθεί προτιμητέα.

Το *S* αντιπροσωπεύει την *Scoring function* την οποία εφαρμόζει το preference στις εγγραφές όπου ισχύει η *σφ* για να διαμορφώσει το Score τους.

Το *C* εκφράζει τον βαθμό εμπιστοσύνης της προτίμησης, δηλαδή το κατά πόσο είμαστε σίγουροι ότι το συγκεκριμένο preference εκφράζει τον χρήστη.

Προκειμένου να γίνει αντιληπτός ο τρόπος έκφρασης των ενδιαφερόντων μέσω των p παρατίθενται κάποια παραδείγματα. Έχουμε μία βάση δεδομένων με ταινίες, σκηνοθέτες, είδη ταινιών, ηθοποιούς, ratings, βραβεία:

MOVIES(m_id, title, year, duration, d id),
DIRECTORS(d_id, director), GENRES(m_id, genre),
ACTORS(a_id, actor), CAST(m_id, a id, role),
RATINGS(m_id, rating, votes), AWARDS(m_id, award, year)
Εικόνα 9: Σχεσιακή βάση με ταινίες

Στον χρήστη Alice αρέσει η ταινία “Million Dollar Baby”. Αυτό μπορεί να εκφραστεί με την *p1 preference*: $p_1[MOVIES] = (\sigma_{m\_id=m3}, 0.9, 1)$, όπου *m3* είναι η τιμή του *m_id* της συγκεκριμένης ταινίας. Δεν χρειάστηκε να χρησιμοποιηθεί συνάρτηση αξιολόγησης για την εξαγωγή του *score*,καθώς απλώς κρίθηκε ότι η ταινία αρέσει στον χρήστη σε βαθμό 9/10. Το *conf* ορίστηκε ίσο με 1. Όπως θα δούμε και παρακάτω, όλες σχεδόν οι προτιμήσεις θέτουν *conf*=1. Αυτό δεν είναι απαραίτητο, καθότι προβλέπεται να υπάρχει διαβάθμιση στην τιμή του *conf* ανάλογα με την εγκυρότητα της πληροφορίας που εκφράζεται. Κάτι τέτοιο, όμως, εξαρτάται από τον τρόπο απόκτησης την πληροφορίας από τον χρήστη. Επειδή στην παρούσα διπλωματική δεν ασχολούμαστε με το συγκεκριμένο θέμα, κάνουμε την σύμβαση ότι όλα τα preferences είναι το ίδιο αξιόπιστα και δίνουν *conf* ίσο με την μονάδα.

Ομοίως, το γεγονός ότι στην Alice δεν αρέσει το “Gran Torino” αποτυπώνεται από την *p2*: $p_2[MOVIES] = (\sigma_{\text{m\_id=m1}}, 0.1, 1)$, όπου *m1* το *m_id* του “Gran Torino”.

Παρατηρούμε ότι οι *p1, p2* αφορούν μία μόνο εγγραφή το καθένα, είναι δηλαδή ατομικές προτιμήσεις. Αντίθετα, για να δηλώσουμε ότι η Alice προτιμά τις δραματικές ταινίες, χρησιμοποιούμε την *p3* η οποία αναφέρεται σε πλήθος εγγραφών: $p_3[GENRES] = (\sigma_{genre='drama'}, 0.7, 1)$.

Η Alice αρέσκεται στο να βλέπει ταινίες σχετικώς πρόσφατες με διάρκεια περίπου 2 ώρες. Αυτό μεταφράζεται ως εξής:

$p_4[MOVIES] = (\sigma_{\text{year}>=2005}, 0.5 * \text{S}_\text{m}(year, 2012) + 0.5 * S_d(duration, 120), 1)$. Η $p4$ θα εφαρμοστεί στις ταινίες με έτος παραγωγής από το 2005 και μετά. Για την διαμόρφωση του score χρησιμοποιεί τις συναρτήσεις S_m, S_d που επιστρέφουν υψηλή βαθμολογία για πρόσφατες χρονολογίες και για διάρκεια κοντά στο επιθυμητό αντίστοιχα.

Εάν η χρήστης θελήσει να ευνοηθούν οι πρόσφατες ταινίες δράσης, τότε η *p5* θα οριστεί ως εξής: $p_5[MOVIES \bowtie GENRES] = (\sigma_{\text{year}>2005,\text{genre}='\text{action}'}, S_m(year, 2012), 1)$.

3.2 Ορισμός Τελεστών (operators)

Η επεκτεταμένη σχεσιακή άλγεβρα SQLPref συνδυάζει με αρμονία τις λειτουργίες των preferences με αυτές των τελεστών του DBMS. Περιγράφονται οι κλασικοί, ήδη υπάρχοντες τελεστές οι οποίοι έχουν υποστεί κάποιες μετατροπές ώστε να υποστηρίζουν την preference-aware λογική. Τέλος παρουσιάζεται ο βασικός τελεστής της SQLPref, ο τελεστής προτίμησης (Prefer operator).

- *Select,* $\sigma_\varphi(R)$ (τελεστής επιλογής): Όπως και στο απλό σχεσιακό μοντέλο, επιλέγει τις πλειάδες που ικανοποιούν τη συνθήκη φ, μόνο που αυτή μπορεί να περιλαμβάνει δύο επιπλέον κριτήρια, το *score* και το *conf.*
- *Project,* $\pi_{A1,\ldots,Ak}(R)$ (τελεστής προβολής): Όπως και στο απλό σχεσιακό μοντέλο, κρατάει τα επιλεγμένα attributes από τις πλειάδες στις οποίες εφαρμόζεται. Επιπλέον διατηρεί τα *score*, *conf.*
- *(Inner) Join, Ri* $\bowtie_{\varphi,F}$ *Rj* (τελεστής εσωτερικής συνένωσης): Προκύπτουν οι πλειάδες όπως θα προέκυπταν σε μια κανονική συνένωση, με *score* και *conf* που υπολογίζονται από την συνάθροιση των επί μέρους με την χρήση της συνάρτησης *F*.
- *Union, Ri* U_F *Rj* (τελεστής ένωσης): Σύμφωνα με την ίδια λογική, επιστρέφονται οι εγγραφές του Union με συναθροισμένα *score* και *conf.*
- *Intersection, Ri* $\cap_F$ *Rj* (τελεστής τομής): Ομοίως, ακολουθείται η ίδια λογική.
- *Prefer,* $\lambda_{p,F}(R)$ (τελεστής προτίμησης): Ο prefer εφαρμόζει μία προτίμηση σε μία σχέση. Υπολογίζονται νέες τιμές για τα *score*, *conf* των πλειάδων οι οποίες ικανοποιούν την φ συνθήκη της *p*. Οι νέες αυτές τιμές συναθροίζονται με αυτές που οι επιλεχθείσες εγγραφές ήδη έχουν. Τις πλειάδες που δεν περιλαμβάνονται στην συνθήκη φ, ο τελεστής τις αφήνει ανέπαφες.

Σε αντίθεση με άλλες προσεγγίσεις, ο Prefer operator επιστρέφει όλο τον αριθμό των εισερχόμενων σε αυτόν πλειάδων (είτε τους έχει αλλάξει τα *score*, *conf* είτε όχι). Αυτό το

χαρακτηριστικό του τον κάνει αρκετά ευέλικτο και συνεργάσιμο με τους υπόλοιπους τελεστές. Παρουσιάζονται τρεις βασικές του ιδιότητες τις οποίες θα εκμεταλλευτούμε για την εύρεση του πλάνου εκτέλεσης των queries:

1. Οι prefer και select τελεστές μεταξύ τους αντιμετατίθενται. Δηλαδή: $\sigma_\phi \lambda_{\mathrm{p}}(R) = \lambda_p \sigma_\phi(R)$.
2. Ο prefer τελεστής είναι αντιμεταθετικός με τον εαυτό του: $\lambda_{p1}(\lambda_{p2}(R)) = \lambda_{p2}\lambda_{p1}(R)$.
3. Ο prefer τελεστής παρουσιάζει αντιμεταθετικότητα και με τους τελεστές Union, Intersection, Join: $\lambda_{\mathrm{p}}(R \theta S) = \lambda_{\mathrm{p}}(R) \theta S$, όπου $\theta \in \{\bigcup, \bigcap, \bowtie\}$.

3.3 Ερωτήματα με προτιμήσεις (preference-aware queries)

3.3.1 Δυνατότητες Ερωτημάτων

Χρησιμοποιώντας τον τελεστή προτίμησης, καθώς και τους νέους ορισμούς για τους άλλους τελεστές, μπορούν τώρα να γραφτούν ερωτήματα με προτιμήσεις. Τα ερωτήματα αυτά επιστρέφουν τις εγγραφές που ανταποκρίνονται στις απαιτούμενες συνθήκες μαζί με τα *score*, *conf* που προκύπτουν από την εφαρμογή του prefer.

Οι τελεστές που περιγράφηκαν υποστηρίζουν την διαδοχική εκτέλεση. Υπάρχει, δηλαδή, η δυνατότητα κλήσης αλλεπάλληλων prefer, το καθένα από τα οποία παίρνει σαν είσοδο τις πλειάδες που επέστρεψε ο προηγούμενος κ.ο.κ. Χάρις στην ευελιξία των τελεστών προτίμησης και των υπολοίπων, μπορούμε να εκφράσουμε με PrefSQL οποιεσδήποτε προτιμήσεις του χρήστη, όσο μεγάλο και αν είναι το πλήθος τους, όσο σύνθετες και αν είναι.

Σε συνέχεια του παραδείγματος με την Alice, εάν επιζητά πρόσφατες δραματικές ταινίες με διάρκεια δύο ωρών, δηλαδή να εφαρμοστούν οι *p3, p4* προτιμήσεις της και να παραχθούν αποτελέσματα, το σύστημα θα διαμορφώσει το εξής query: $Q_{p3,p4\text{-}I}$: $\lambda_{p3}(GENRES) \bowtie \lambda_{p4}(MOVIES)$. Τα αποτελέσματα της εφαρμογής της *p3* στον GENRES γίνονται Join με αυτά της *p4* στην MOVIES, με τα score, conf των ταινιών οι οποίες είναι πρόσφατες δραματικές ταινίες να συναθροίζονται. Ως εκ τούτου, αυτές οι ταινίες θα αποκτήσουν *conf*=2.

Επιβάλλεται να αποσαφηνιστεί ότι θα επιστραφούν σαν αποτέλεσμα όλες οι εγγραφές (αρκεί βέβαια να υπάρχει αντιστοίχιση των εγγραφών στους δύο πίνακες, δηλαδή να μην συναντάται το φαινόμενο κάποια movie να μην έχει καταχωρημένο genre ή αντίστροφα, διότι σε τέτοια περίπτωση δεν θα επιστραφεί καθότι θα «κοπεί» από το join). Αυτές οι οποίες δεν θα

πληρούν καμία συνθήκη από τις συνθήκες των *p3, p4* απλώς θα έχουν στα *score, conf* τους τις by default τιμές *(score = ⊥, conf = 0).*

Υπάρχει η δυνατότητα να αποκλείσουμε εξαρχής πλειάδες οι οποίες δεν πρόκειται επουδενί να απασχολήσουν την χρήστη –κρίνοντας από τις προτιμήσεις του-, κάνοντας Select. Για να εμφανιστούν μόνο τα attributes που μας ενδιαφέρουν θα κάνουμε και Project. Με τις αλλαγές αυτές έχουμε:

$Q_{p3,p4\text{-}II}$: $\pi_{title,\ year,\ duration,genre}\{ \lambda_{p3}(GENRES) \bowtie \sigma_{duration>90\ OR\ duration<150} \lambda_{p4}(MOVIES) \}$ Κατ' αυτόν τον τρόπο δεν πρόκειται να επιστραφούν ταινίες με διάρκεια λιγότερη από 90 ή περισσότερη από 150 λεπτά.

Ενδεχομένως η Alice να επιθυμεί να προωθηθούν, εκτός από δραματικές ταινίες, και ντοκιμαντέρ. Η προστιθέμενη προτίμηση είναι η: $p_6[GENRES] = (\sigma_{genre='documentary'}, 0.7, 1)$.

Το ερώτημα διαμορφώνεται ως εξής:

$Q_{p3,p4,p6\text{-}I}$: $\pi_{title,\ year,\ duration,genre}\{ \lambda_{p6}(\lambda_{p3}(GENRES)) \bowtie \sigma_{duration>90\ OR\ duration<150} \lambda_{p4}(MOVIES) \}$.

Η πολυπλοκότητα των ερωτημάτων μπορεί να ανέβει σε σημείο όπου το ανθρώπινο μάτι να δυσκολεύεται εξαιτίας της πληθώρας παρενθέσεων και της αλληλουχίας τελεστών. Στην παρούσα φάση της διπλωματικής δεν κρίνεται σκόπιμο να προβούμε σε παραδείγματα υψηλής πολυπλοκότητας. Έτσι κι αλλιώς στα κεφάλαια της υλοποίησης και του ελέγχου παρατίθεται η εκτέλεση υπερ-ικανοποιητικού αριθμού τέτοιων queries.

3.3.2 Περιορισμοί Ερωτημάτων

Τα ερωτήματα σε PrefSQL έχουν κάποιους περιορισμούς οι οποίοι εύλογα προκύπτουν από τον ορισμό των τελεστών της. Ίσως η επισήμανσή τους να είναι περιττή, όμως εάν κάποιος μπει στην διαδικασία σχηματισμού σύνθετων queries με αλλεπάλληλες εμφωλευμένες εκτελέσεις του pref, πολλά join, union και intersection είναι εύκολο να γίνει λάθος.

Ο βασικός περιορισμός αφορά τον τελεστή προτίμησης. Είναι απαραίτητο η σχέση στην οποία αυτός εφαρμόζεται να περιέχει το/τα attribute/-s οι τιμές των οποίων καθορίζουν το κομμάτι της συνθήκης φ, καθώς και την τιμή του score που προκύπτει από την συνάρτηση αξιολόγησης –εάν αυτή υπάρχει- όταν εκείνη την παίρνει για παράμετρο.

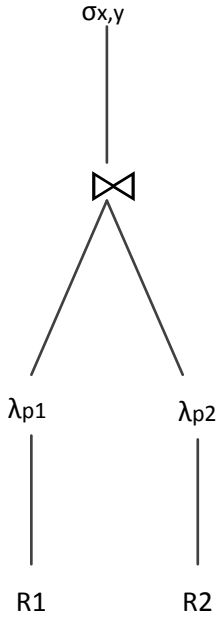
Ένας άλλος περιορισμός αφορά την παράλληλη χρήση των τελεστών της PrefSQL με αυτούς της κανονικής SQL. Ναι μεν υποστηρίζεται, όμως πρέπει να γίνεται με προσοχή, καθότι οι δυαδικοί απλοί τελεστές της SQL δεν υποστηρίζουν την συνάθροιση των score, conf και σε ενδεχόμενο συνδυασμό τελεστών υπάρχει περίπτωση να μην έχουμε τα σωστά αποτελέσματα.

Τέλος, αξίζει να επισημανθεί ότι όπως και οι κανονικοί τελεστές ένωσης και τομής, έτσι και οι αντίστοιχοι της PrefSQL παίρνουν σαν ορίσματα σχέσεις με ίδια attributes.

3.3.3 Διαμόρφωση, Βελτιστοποίηση Πλάνου Εκτέλεσης (Optimization)

Δοθέντος ενός query από τον χρήστη, το σύστημα καλείται να καταστρώσει το βέλτιστο πλάνο εκτέλεσής του, αλλάζοντας την αλληλουχία εκτέλεσης των τελεστών χωρίς, όμως, να επηρεάζει το τελικό αποτέλεσμα. Για την καλύτερη εποπτεία των ερωτημάτων, τα αναπαριστούμε με δέντρα εκτέλεσης.

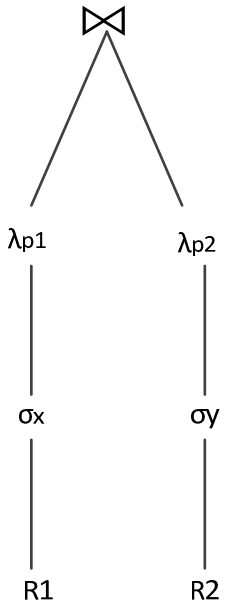
Ας υποθέσουμε ότι έχουμε τους tables R_1, R_2 και τις προτιμήσεις $p_1(\sigma_{\varphi 1}, S, C)$, $p_2(\sigma_{\varphi 2}, S, C)$ που τους αφορούν. Ο χρήστης επιθυμεί να επιστραφούν μόνο οι εγγραφές που ικανοποιούν τις υποθετικές συνθήκες *x* και *y*, πρέπει δηλαδή να γίνουν τα Select σ_x και σ_y στους R_1, R_2. Το πλάνο εκτέλεσης που εύλογα μπορεί να σκεφτούν οι περισσότεροι είναι:



Εικόνα 9: Plan-A

Τα *λp1*, *λp2* εφαρμόζονται στους *R1, R2*. Το αποτέλεσμα γίνεται join και τέλος επιδρά ο Select τελεστής με τους *x, y* περιορισμούς.

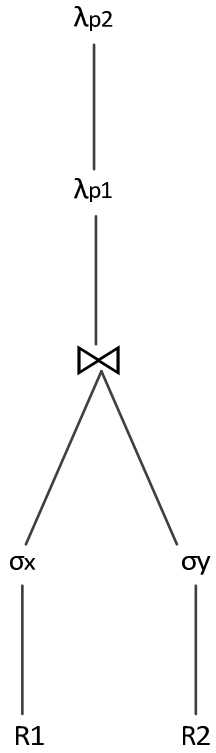
Για να γίνει πιο αποδοτικό το πλάνο εκτέλεσης, τα Select σ_x, σ_y πρέπει να κατέβουν στο κάτω μέρος του δέντρου, δηλαδή να εκτελεστούν στους R_1, R_2 και στο αποτέλεσμά τους να εφαρμοστούν οι prefer τελεστές. Κατ' αυτόν τον τρόπο αποφεύγεται η άσκοπη μεταφορά και επεξεργασία εγγραφών οι οποίες δεν συνεισφέρουν στο τελικό αποτέλεσμα:



Εικόνα 9: Plan-B

Το Plan-B, σαφέστατα είναι προτιμότερο του Plan-A, κάτι το οποίο επαληθεύεται και με πειράματα αργότερα στο κεφάλαιο 6.

Ένα θέμα που τίθεται υπό εξέταση είναι το κατά πόσον στην πράξη είναι αποδοτικότερο το να εφαρμόζουμε τους τελεστές προτίμησης στις επί μέρους σχέσεις και όχι στο αποτέλεσμα της συνάθροισης. Αυτό εξαρτάται, βασικά, από την σχέση των πλειάδων που υπεισέρχονται στον join operator με αυτές που εξέρχονται. Σε γενικές γραμμές, εάν αυτές που προκύπτουν από το join είναι περισσότερες από τις επί μέρους, τότε το Plan-B κρίνεται προτιμότερο. Σε διαφορετική περίπτωση προτιμάται το Plan-C:



Εικόνα 10: Plan-C

Αυτός ο κανόνας, όμως δεν επαληθεύεται πάντοτε καθότι δεν είναι καθολικός. Υπεισέρχονται διάφοροι παράγοντες που επηρεάζουν την βέλτιστη επιλογή μεταξύ Plan-B και Plan-C, όπως είναι η φύση του ερωτήματος, το ποσοστό των πλειάδων που επηρεάζονται από τις preferences (affected tuples). Στην παρούσα διπλωματική θα ασχοληθούμε διεξοδικά με την σύγκριση των Plan-B, Plan-C με εκτελέσεις πειραμάτων προκειμένου να εξάγουμε ασφαλή συμπεράσματα.

Είναι εμφανές ότι, εκμεταλλευόμενοι τις ιδιότητες 1, 2, 3 που περιγράφονται στο 3.2, έχουμε την δυνατότητα σχηματισμού και άλλων πλάνων εκτέλεσης πέρα των A, B, C. Καθότι, όμως, είναι αυταπόδεικτο ότι οποιεσδήποτε άλλες αλληλουχίες –που δεν αλλοιώνουν το τελικό αποτέλεσμα- των τελεστών, δεν βελτιώνουν την αποδοτικότητα των ερωτημάτων, δεν θα ασχοληθούμε με άλλα query plans.

4

Ανάλυση - Σχεδίαση Συστήματος

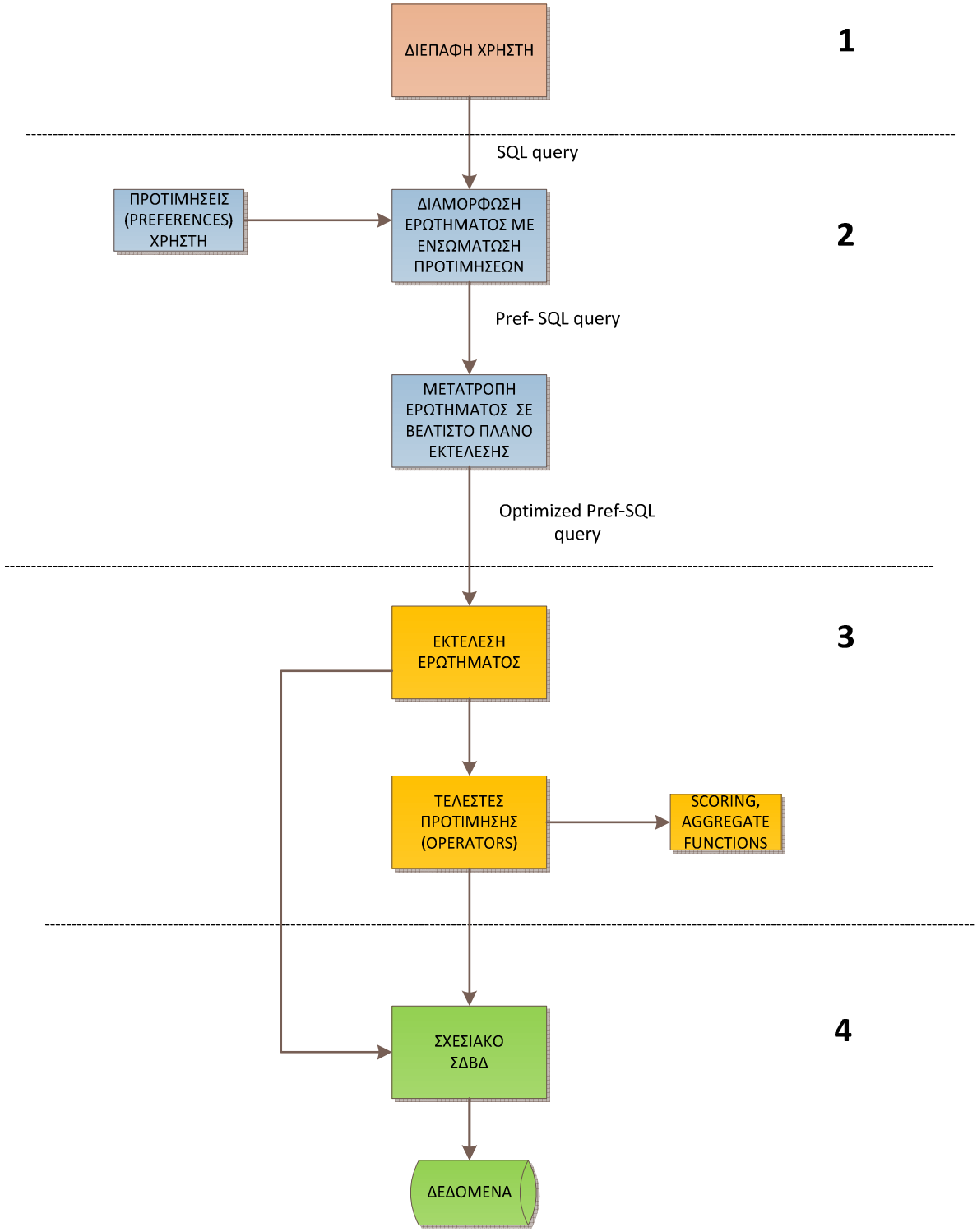
4.1 Γενική Αρχιτεκτονική του Συστήματος

Όπως φαίνεται από το Γενικό Διάγραμμα, το σύστημα αποτελείται από τρία υποσυστήματα.

Στο ανώτερο επίπεδο βρίσκεται το «Υποσύστημα Διεπαφής του χρήστη με το σύστημα» (ΥΠΟΣΥΣΤΗΜΑ Ι), ενώ ακολουθεί το «Υποσύστημα Διαμόρφωσης και Βελτιστοποίησης ερωτημάτων» (ΥΠΟΣΥΣΤΗΜΑ ΙΙ) το οποίο αναλύει το ερώτημα του χρήστη, ενσωματώνει σε αυτό τις καταγεγραμμένες στη βάση προτιμήσεις του και καταστρώνει το βέλτιστο σχέδιο εκτέλεσης του ερωτήματος.

Στο τρίτο επίπεδο βρίσκεται το «Υποσύστημα Εκτέλεσης των ερωτημάτων» (ΥΠΟΣΥΣΤΗΜΑ ΙΙΙ) όπου το ερώτημα εκτελείται μέσω της εκτέλεσης κάθε τελεστή.

Η ροή του συστήματος είναι «από πάνω προς τα κάτω» όταν τίθεται ένα ερώτημα από τον χρήστη. Τα αποτελέσματα του ερωτήματος αποστέλλονται στον χρήστη μέσω της διεπαφής.

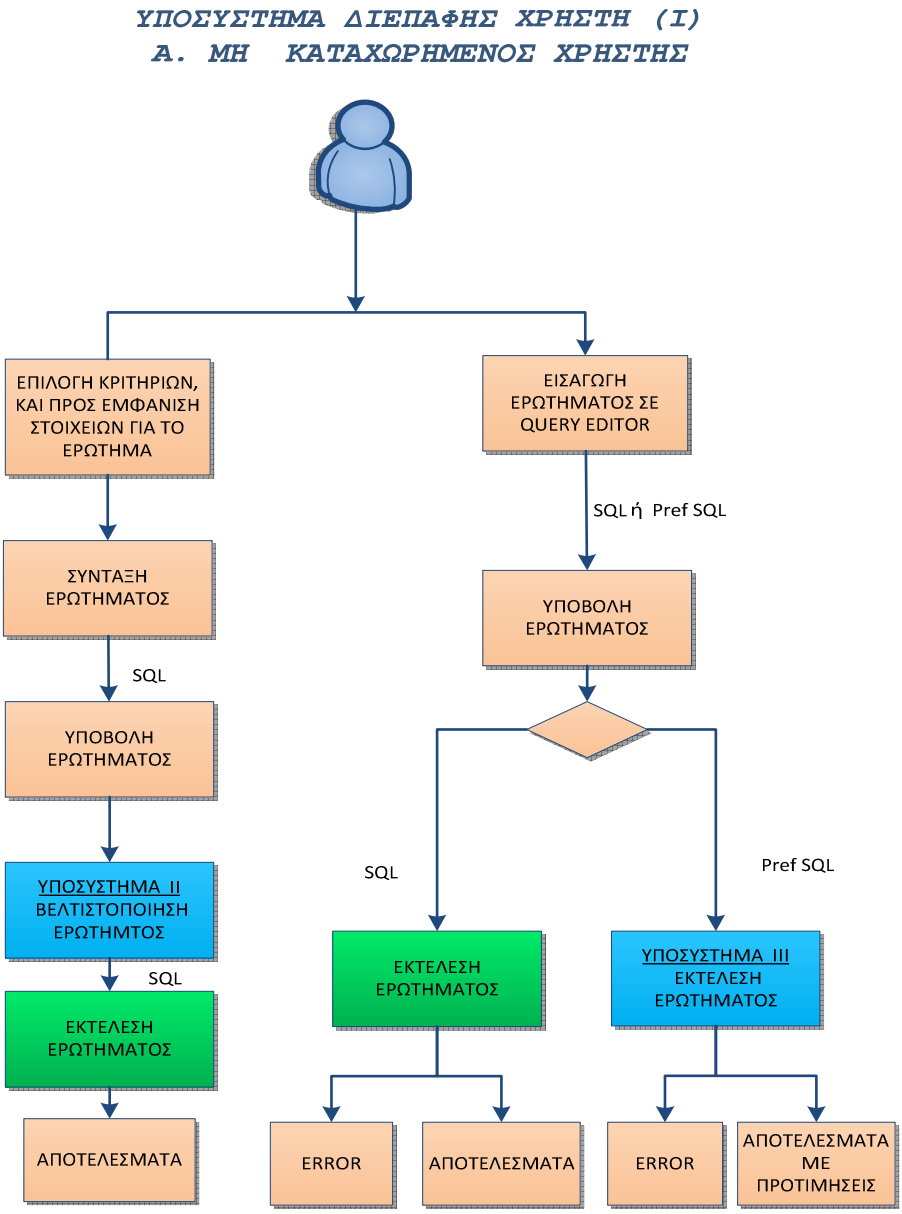


Εικόνα 11: Γενικό Διάγραμμα Συστήματος

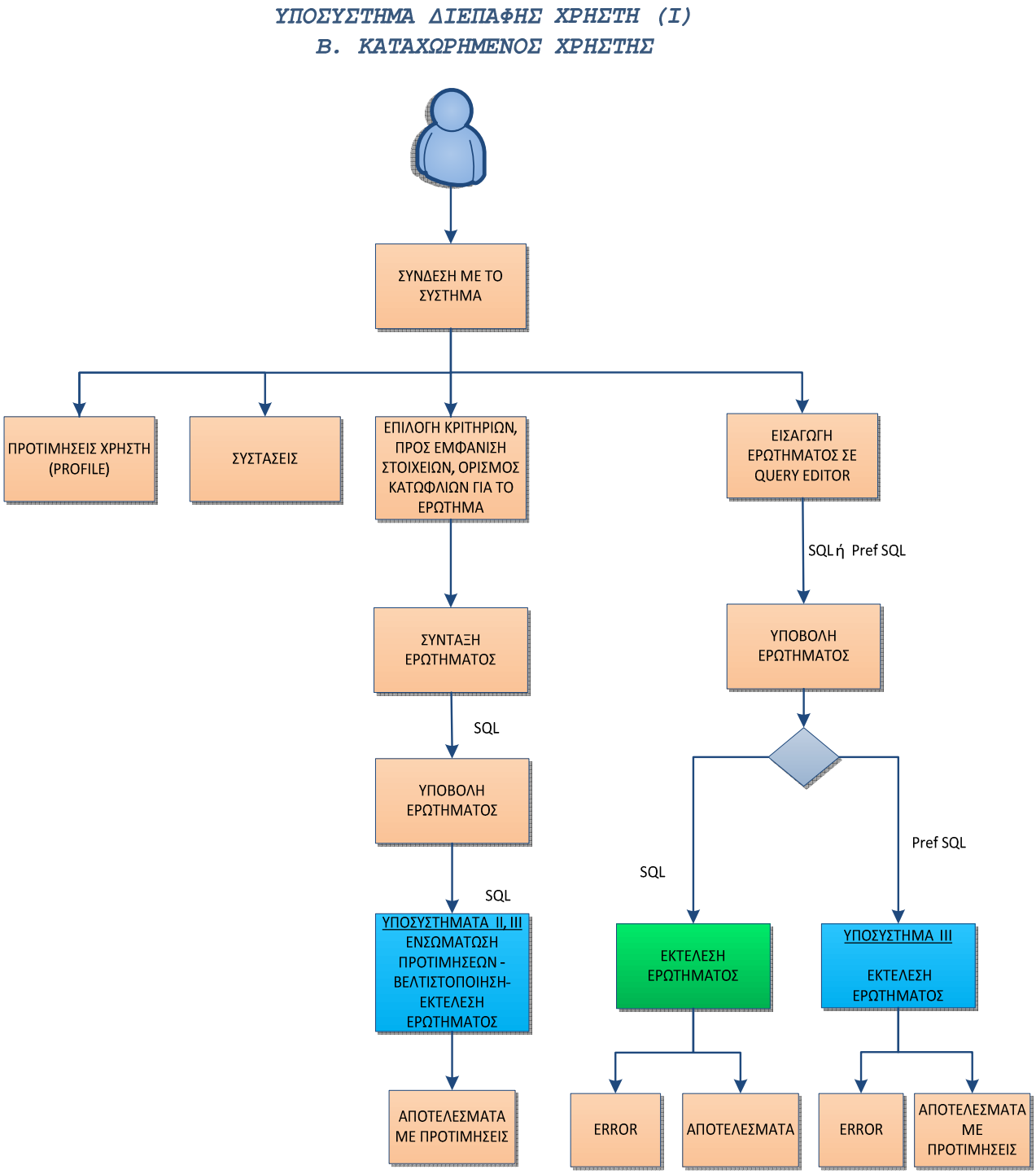
4.2 Αρχιτεκτονική Υποσυστήματος Διεπαφής του χρήστη (I)

Το υποσύστημα αποτελείται από το Client side και από το Server side. Το Client side είναι καταρχήν υπεύθυνο για την γραφική απεικόνιση των απαιτούμενων κριτηρίων προκειμένου να διαμορφωθεί ένα ερώτημα. Ο χρήστης αλληλεπιδρά με το σύστημα θέτοντας τα επιθυμητά κριτήρια για το ερώτημά του και στη συνέχεια το Server side αναλαμβάνει την σύνταξη του ερωτήματος σε SQL και την προώθησή του στο Υποσύστημα ΙΙ. Μετά την εκτέλεση του ερωτήματος, τα αποτελέσματα αποστέλλονται μέσω της Διεπαφής πίσω στον χρήστη.

Εκτός από την εκτέλεση ερωτημάτων, το Υποσύστημα Διεπαφής παρέχει συστάσεις σε έναν χρήστη, οι οποίες αντανακλούν τις καταχωρημένες προτιμήσεις του.



Εικόνα 12: Υποσύστημα Διεπαφής χρήστη - Α

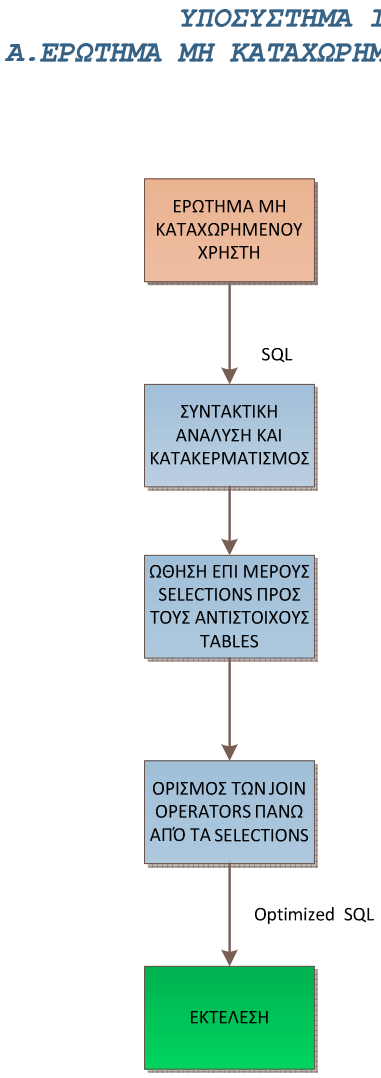


Εικόνα 13: Υποσύστημα Διεπαφής χρήστη – Β.

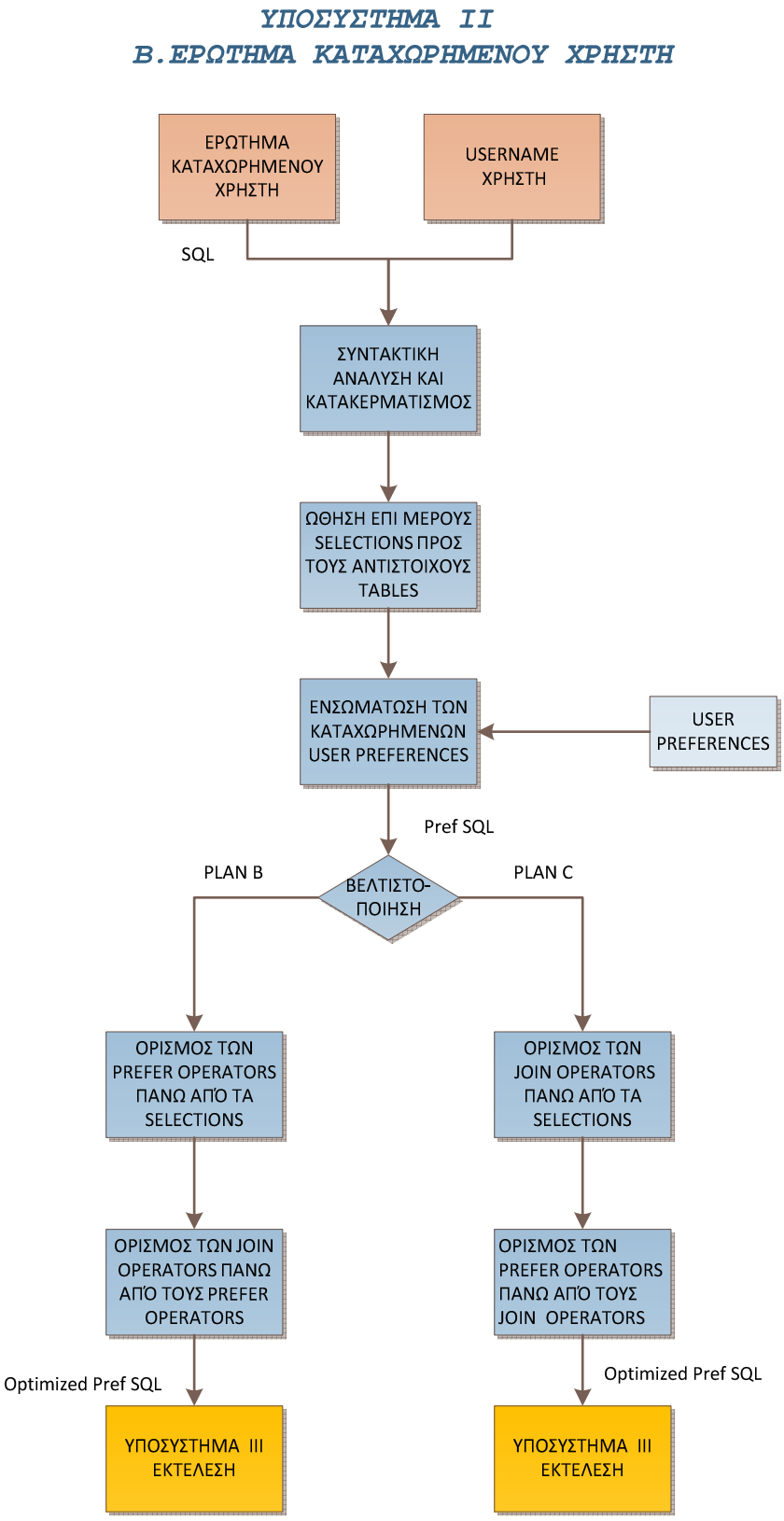
4.3 Αρχιτεκτονική Υποσυστήματος Διαμόρφωσης - Βελτιστοποίησης Ερωτημάτων (II)

Το Υποσύστημα αυτό παραλαμβάνει ένα SQL ερώτημα από το Υποσύστημα I, το αναλύει συντακτικά, το κατακερματίζει και προσπαθεί να καταστρώσει το βέλτιστο πλάνο εκτέλεσης για το ερώτημα και συγχρόνως να ενσωματώσει τις καταχωρημένες στη βάση δεδομένων προτιμήσεις του χρήστη στο ερώτημα. Προκύπτει ένα Optimized PrefSQL ερώτημα, το οποίο προωθείται στο Υποσύστημα III για εκτέλεση.

Η υλοποίηση έχει γίνει με PL/pg SQL δηλαδή στο επίπεδο της βάσης.



Εικόνα 14: Υποσύστημα Διαμόρφωσης – Βελτιστοποίησης – Α.



Εικόνα 15: Υποσύστημα Διαμόρφωσης – Βελτιστοποίησης – Β.

4.4 Αρχιτεκτονική Υποσυστήματος Εκτέλεσης Ερωτημάτων (III)

Το Υποσύστημα ΙΙΙ παραλαμβάνει ένα ερώτημα σε μορφή Optimized Pref SQL και το εκτελεί, εκτελώντας πρώτα του Τελεστές Προτίμησης (Operators) που περιέχει το ερώτημα. Η υλοποίηση των operators έχει γίνει εξολοκλήρου με PL/pg SQL. Χωρίζονται σε δυο κατηγορίες : α) Prefer Operators οι οποίοι εφαρμόζουν τα preferences στα δεδομένα δηλαδή αξιολογούν και βαθμολογούν τα δεδομένα και β) Binary Operators (Join, Union, Intersect) οι οποίοι λειτουργούν όπως οι αντίστοιχοι operators της SQL, με την διαφορά ότι επιπλέον διαχειρίζονται τα βαθμολογημένα δεδομένα εφαρμόζοντας συναθροιστικές συναρτήσεις.

4.5 Αποτύπωση στα δεδομένα – Αποθήκευση – Διαχείριση των Προτιμήσεων (preferences)

Οι προτιμήσεις των χρηστών εκφράζονται αναφορικά με το περιεχόμενο των πινάκων της βάσης δεδομένων. Μια προτίμηση *(preference*) ορίζεται από τον πίνακα στα δεδομένα του οποίου αναφέρεται, τις προϋποθέσεις που πρέπει να ικανοποιούνται από τα δεδομένα, το κριτήριο αξιολόγησης για απόδοση βαθμού προτίμησης στα δεδομένα και τον δείκτη βεβαιότητας (εμπιστοσύνης) της συγκεκριμένης προτίμησης.

Τα δεδομένα, αφού αξιολογηθούν από ένα preference, θα πρέπει να ενημερωθούν για τον βαθμό προτίμησης ο οποίος τους αποδόθηκε καθώς και για το πόσο έμπιστο είναι το preference που τα αξιολόγησε. Εάν στα ίδια δεδομένα επιδράσουν και άλλα preferences, η ενημέρωση θα διαμορφωθεί συνολικά συναθροίζοντας τις επί μέρους επιδράσεις. Η γνώση αυτή από τα δεδομένα, δηλαδή αν έχουν κριθεί αρεστά και κατά πόσο, θα πρέπει να διατηρηθεί έως ότου ολοκληρωθεί η διαδικασία κατάταξής τους σε μια διατεταγμένη λίστα.

Προκειμένου λοιπόν να αποτυπωθούν στα δεδομένα ο βαθμός προτίμησης και ο βαθμός εμπιστοσύνης όπως έχουν προκύψει από τις επιδράσεις διαφόρων preferences, εμπλουτίζουμε τα δεδομένα τα οποία προσφέρονται για αξιολόγηση από preferences, με τα attributes *score, conf*.

Η αποτύπωση - γνώση αυτή αφού είναι προσωρινή, δεν καταγράφεται βέβαια στους πίνακες της βάσης δεδομένων αλλά αποτυπώνεται όποτε απαιτηθεί σε ενδιάμεσες προσωρινές relations με δεδομένα της βάσης οι οποίες παράγονται από τους operators.

Οι πίνακες της βάσης όπου αποθηκεύονται τα preferences για κάθε χρήστη είναι:
user_preferences (*id_p, username, id)* και *preferences (id, name, preftable, preffield,, condition, functionid, confidence).*

Η function *get_db_preference(id)* διαβάζει από την βάση ένα συγκεκριμένο preference και το επιστρέφει ως *preferences_type* type*,* προκειμένου να γίνει η διαχείριση από τους postgresql operators και τις functions του συστήματος.

Ένα preference περιέχει τις εξής πληροφορίες:

- *id* : Μοναδικό αναγνωριστικό
- *name* : Περιγράφει συνοπτικά το preference
- *preftable* : Ο table της βάσης στα δεδομένα του οποίου απευθύνεται το preference.
- *preffield* : Το attribute του *preftable* την τιμή του οποίου παίρνει η scoring function *(function_id)* για να υπολογίσει τον βαθμό προτίμησης που θα αποδοθεί στο εκάστοτε tuple.
- *condition* : Καθορίζει ποιά tuples θα αξιολογηθούν και θα βαθμολογηθούν από το preference. Περιέχει συνθήκες που αναφέρονται σε attributes του *preftable*.
- *functionid* : Η scoring function που θα εφαρμοστεί στο *preffield* προκειμένου να προκύψει η βαθμολογία του tuple.
- *confidence* : Δείκτης εμπιστοσύνης που σχετίζεται με το preference

Η αξιολόγηση ενός set από tuples μιας relation από ένα preference, γίνεται από τον Prefer Operator. Η βαθμολόγηση και ο βαθμός εμπιστοσύνης ενός tuple από το preference καταγράφονται στα columns *score, conf* της relation αντίστοιχα, τα οποία αν δεν υπάρχουν προστίθενται.

Εάν το tuple έχει αξιολογηθεί ήδη από άλλα preferences, κατά την εφαρμογή του νέου preference γίνεται συνάθροιση (aggregate function) των *score* και *conf* του tuple με τις τιμές που αποδίδει το preference. Ένα preference εφαρμόζεται σε ένα set από tuples μιας relation υπό την προϋπόθεση ότι το *scoring field* του preference και τα attributes που συμμετέχουν στο *condition* του preference, περιέχονται στα attributes αυτού του set.

Εξ' ορισμού ένα preference αναφέρεται και εφαρμόζεται σε έναν συγκεκριμένο table της βάσης, τον *preftab*le και το *preffield* και τα attributes του *condition* είναι attributes του *preftable*. Υπάρχει όμως η δυνατότητα ένα preference να εφαρμοστεί σε μια relation πχ. αποτέλεσμα join ή union operator, αρκεί να ισχύει η παραπάνω προϋπόθεση.

Ακολουθεί ένα screenshot από τον PgAdmin με user preferences.

```
SELECT UP.USERNAME, UP.ID, P.NAME, P.PREFTABLE,P.PREFFIELD,P.CONDITION, P.FUNCTIONID, P.CONFIDENCE FROM USER_PREFERENCES UP,PREFERENCES P
WHERE UP.ID=P.ID ORDER BY USERNAME ,PREFTABLE
```

Output pane

Data Output | Explain | Messages | History

	username character varying(40)	id integer	name character varying(40)	preftable character varying(40)	preffield character varying(40)	condition character varying(140)	functionid integer	confidence real
9	aleka	46	voted movies	ratings	votes	votes>15000	22	1
10	iro	8	Action movies	genres	genre	genre LIKE '%Action%'	3	1
11	iro	37	murder	keywords	keyword	keyword LIKE '%murder%'	6	1
12	iro	4	Recent movies	movies	prod year	prod year >= 2000	2	1
13	iro	17	rated movies	ratings	rating	rating>8	24	1
14	john	16	English movies	countries	country	country='UK'	4	1
15	john	45	French movies	countries	country	country='France'	4	1
16	john	12	American movies	countries	country	country='USA'	4	1
17	john	26	sex	keywords	keyword	keyword LIKE '%sex%'	6	1
18	john	38	love	keywords	keyword	keyword LIKE '%love%'	6	1
19	john	2	police	keywords	keyword	keyword LIKE '%police%'	6	1
20	john	4	Recent movies	movies	prod year	prod year >= 2000	2	1
21	john	47	very recent movies	movies	prod year	prod year >= 2010	2	1
22	john	7	less recent movies	movies	prod year	prod year >= 1980 and prod year<2000	2	1
23	john	41	actor : Leonardo DiCaprio	movies2actors	actorid	actorid=265595	4	1
24	john	40	actor : Tom Hanks	movies2actors	actorid	actorid=425970	4	1
25	john	39	actor : George Clooney	movies2actors	actorid	actorid=196730	4	1
26	john	43	director : Woody Allen	movies2directors	directorid	directorid=3128	4	1
27	kostis	16	English movies	countries	country	country='UK'	4	1
28	kostis	8	Action movies	genres	genre	genre LIKE '%Action%'	3	1
29	kostis	2	police	keywords	keyword	keyword LIKE '%police%'	6	1
30	kostis	4	Recent movies	movies	prod year	prod year >= 2000	2	1
31	kostis	42	director : Martin Scorsese	movies2directors	directorid	directorid=154456	4	1
32	lena	12	American movies	countries	country	country='USA'	4	1
33	lena	3	Comedies	genres	genre	genre LIKE '%Comedy%'	3	1
34	lena	38	love	keywords	keyword	keyword LIKE '%love%'	6	1
35	lena	19	actor : Meryl Streep	movies2actors	actorid	actorid=1760182	4	1

Εικόνα 16: Προτιμήσεις χρηστών αποθηκευμένες στους πίνακες preferences και user_preferences

4.6 Λεπτομέρειες Υποσυστημάτων

Στην ενότητα αυτή παρατίθενται οι λεπτομέρειες σχεδίασης και υλοποίησης για τα Υποσυστήματα που αποτελούν το Γενικό Σύστημα.

4.6.1 Υποσύστημα Διεπαφής του χρήστη (I)

Η ανάπτυξη της διαδικτυακής αυτής διεπαφής στοχεύει στην εφαρμογή και παρουσίαση της υλοποίησης του συστήματος «Εκτέλεση ερωτημάτων χρήστη σε Πρότυπο Σύστημα Εξατομικευμένης Διαχείρισης Βάσεων Δεδομένων». Η ανάπτυξη έγινε σε πλατφόρμα NetBeans με χρήση τεχνολογιών JSP, HTML, CSS, Javascript.

Έγινε προσπάθεια ώστε αφενός μεν να παρέχεται η δυνατότητα διατύπωσης ερωτημάτων από έναν έμπειρο χρήστη σε Pref SQL μέσω query editor και εκτέλεσής τους –όπως δηλαδή γίνεται στον PgAdmin-, αφετέρου δε ένας μη εξειδικευμένος χρήστης να έχει την δυνατότητα να διαμορφώνει ένα ερώτημα σε γραφικό περιβάλλον μέσω κριτηρίων από λίστες τιμών επιλέγοντας τα στοιχεία που επιθυμεί να εμφανιστούν στα αποτελέσματα και τέλος ορίζοντας τα thresholds για τα score και conf, δηλώνοντας δηλαδή ουσιαστικά το εύρος των επιθυμητών αποτελεσμάτων.

Η εφαρμογή δεν υποχρεώνει τον χρήστη να συνδεθεί στο σύστημα. Εάν ο χρήστης δεν κάνει Login, αντιμετωπίζεται ως Unregistered User. Σε αυτή την περίπτωση έχει την δυνατότητα να υποβάλει ερωτήματα μέσω του γραφικού περιβάλλοντος και να λάβει τα αντίστοιχα αποτελέσματα, τα οποία όμως –όπως είναι φυσικό- δεν είναι εξατομικευμένα (Επισημαίνουμε ότι η λειτουργία του query editor , δεν επηρεάζεται από το αν ο χρήστης έχει συνδεθεί ή όχι).

Α. Μη Συνδεδεμένος Χρήστης – Διαθέσιμες λειτουργίες

Όπως φαίνεται στην Εικόνα 12, ο χρήστης ο οποίος επιλέγει να μην συνδεθεί στο σύστημα, ή συνδέεται μεν αλλά δεν υπάρχει καταχωρημένο προφίλ προτιμήσεων για αυτόν, υποβάλει ερωτήματα μέσω του γραφικού περιβάλλοντος ορίζοντας τις συνθήκες που θέλει να ικανοποιούνται καθώς και τα προς εμφάνιση στοιχεία. Το ερώτημα συντάσσεται με βάση τις επιλογές του χρήστη από το υποσύστημα σε SQL και προωθείται στο Υποσύστημα ΙΙ.Α για να διαμορφωθεί το βέλτιστο πλάνο εκτέλεσης. Το optimized SQL query που προκύπτει εκτελείται στην Database engine.

Το αποτέλεσμα του ερωτήματος εμφανίζεται στο γραφικό περιβάλλον σε πίνακα, στον οποίο προσφέρεται δυνατότητα πλοήγησης καθώς και επιλογής της γραμμής (στην περίπτωσή μας ταινίας) ώστε να λάβει περαιτέρω πληροφορίες σχετικά με την επιλογή του.

Επιπλέον, ο μη καταχωρημένος χρήστης μπορεί να υποβάλει ερώτημα σε απλή SQL ή σε PrefSQL στον query editor –αρκεί να γνωρίζει την σύνταξή τους- και να λάβει το ζητούμενο αποτέλεσμα.

Β. Συνδεδεμένος Χρήστης – Διαθέσιμες λειτουργίες

Όπως φαίνεται στην Εικόνα 13 οι λειτουργίες που διαθέτει το Υποσύστημα Διεπαφής, εφόσον έχει γίνει Login και υφίσταται κάποιο προφίλ προτιμήσεων στη βάση δηλαδή υπάρχουν καταχωρημένες προτιμήσεις (preferences) που αφορούν τον χρήστη, είναι:

- Συστάσεις (*Recommend*)

  Το Server Side καλεί την προς εκτέλεση διαδικασία *make_query* (pl/pgsql) η οποία συλλέγει από την βάση τις καταχωρημένες προτιμήσεις του συνδεδεμένου χρήστη, «χτίζει» ένα σύνθετο PrefSQL ερώτημα για την εξαγωγή των items (ταινίες) που συγκεντρώνουν τις προτιμήσεις του και στην συνέχεια το εκτελεί χρησιμοποιώντας τις *pref_gl10* και *join_gl3* συναρτήσεις. Τα αποτελέσματα αποθηκεύονται σε table με όνομα το username του χρήστη. Επειδή ο χρόνος εκτέλεσης της παραπάνω διαδικασίας είναι μεγάλος, αφού εφαρμόζονται οι προτιμήσεις του χρήστη σε ολόκληρους tables και στην συνέχεια εκτελούνται χρονοβόρα join με συναθροίσεις σε μεγάλο όγκο δεδομένων, κατέστη αναγκαίο η *make_query* να μην καλείται από το υποσύστημα της Διεπαφής. Αντ' αυτού, καλείται από τον Διαχειριστή της βάσης όποτε προστίθεται/διαγράφεται μία προτίμηση του συγκεκριμένου χρήστη στον/από τον table *user_preferences*.

  Κατ' αυτόν τον τρόπο διατίθενται άμεσα οι συστάσεις από την βάση όπου έχουν αποθηκευτεί τα αποτελέσματα της τελευταίας εκτέλεσης της αναφερόμενης διαδικασίας.

- Προτιμήσεις Χρήστη (*MyPrefs*)

  Παρουσιάζεται το περιεχόμενο του table *preferences* που αφορά τον συνδεδεμένο χρήστη. Εμφανίζονται όλες οι πληροφορίες που περιγράφονται στο 4.5.

- Υποβολή ερωτήματος (*Home*)

  Είναι η αρχική και κύρια σελίδα της Διεπαφής. Περιέχει λίστες τιμών με κριτήρια-παραμέτρους για το προς υποβολή ερώτημα. Ο χρήστης επιλέγει μέσω αυτών τις συνθήκες που επιθυμεί να πληρούνται. Κάθε λίστα τιμών αναφέρεται σε κάποιο attribute ενός table της βάσης δεδομένων. Ο χρήστης επιλέγει επίσης εάν θέλει να εμφανίζεται το κάθε attribute στα αποτελέσματα. Έχει δε την δυνατότητα να επιλέξει ένα attribute για εμφάνιση, χωρίς να συμπεριλάβει στο ερώτημα κάποια συνθήκη αναφορικά με αυτό.

  Τέλος, τα επιστραφέντα στοιχεία περιορίζονται με τον καθορισμό κατωφλίων στα *score* ή/και *conf*.

  Με την υποβολή των παραμέτρων από τον χρήστη (submit) το υποσύστημα συντάσσει το ερώτημα σε SQL. Στα Projections συμπεριλαμβάνονται τα attributes που συμμετέχουν στις επιλεχθείσες συνθήκες καθώς και εκείνα που έχουν επιλεγεί για εμφάνιση. Έτσι, εξασφαλίζεται η αποστολή όλων των στοιχείων προς αξιολόγηση που χρειάζονται για την ορθή εκτέλεση του εξατομικευμένου ερωτήματος και την εφαρμογή των προτιμήσεων στα δεδομένα.

Τα βασικά στοιχεία για κάθε ταινία (*movieid, title*) συμπεριλαμβάνονται πάντοτε (η εμφάνισή τους ή μη δεν ελέγχεται από τον χρήστη καθότι είναι δεδομένη). Επειδή θεωρείται δεδομένο ότι μεταξύ των επί μέρους Selections θα τεθεί ο τελεστής JOIN, κατά την σύνταξη του SQL query δεν συμπεριλαμβάνονται τα join conditions.

Το ερώτημα υποβάλλεται από το Υποσύστημα Διεπαφής στο Υποσύστημα II.B για ενσωμάτωση προτιμήσεων, βελτιστοποίηση και εκτέλεση.

Τα αποτελέσματα εμφανίζονται στο γραφικό περιβάλλον σε πίνακα. Είναι αξιολογημένα και βαθμολογημένα (*score,conf*) με βάση τις δηλωθείσες προτιμήσεις του χρήστη και ταξινομημένα ως προς *score*conf descending*. Ο χρήστης δύναται να επιλέξει κάποιο συγκεκριμένο στοιχείο για να λάβει περαιτέρω πληροφορίες.

- Εισαγωγή ερωτήματος σε query editor (*Advanced*)

  Ο χρήστης εισάγει ένα SQL ή PrefSQL ερώτημα στον query editor και είναι αποκλειστικά υπεύθυνος για την ορθότητα της σύνταξης. Έχει την δυνατότητα να αποθηκεύσει ένα ερώτημα είτε είναι σωστό είτε όχι (table *queries*), να διαγράψει ένα ερώτημα από τον table *queries* και να εκτελέσει είτε ένα αποθηκευμένο query είτε ένα νέο. Προς διευκόλυνσή του, ο πίνακας *queries* βρίσκεται στην διάθεσή του με δυνατότητα φόρτωσης ενός query στον editor προς επεξεργασία ή/και εκτέλεση.

  Εάν το ερώτημα είναι σε απλή SQL, προωθείται προς εκτέλεση στην Database Engine και εφόσον είναι συντακτικά ορθό, τα αποτελέσματα εμφανίζονται σε πίνακα με δυνατότητα επί μέρους πληροφοριών για το εκάστοτε στοιχείο με ένα μόνο κλικ.

  Εάν το ερώτημα είναι σε PrefSQL, η εκτέλεση ανατίθεται στο Υποσύστημα III (Operators), για να φτάσουν τα αποτελέσματα ομοίως σε μορφή πίνακα.

  Και στις δύο περιπτώσεις, σε περίπτωση λάθους στο query, εμφανίζεται (pop up) μήνυμα λάθους.

4.6.2 Υποσύστημα Διαμόρφωσης – Βελτιστοποίησης Ερωτημάτων (II)

Η ανάπτυξη του υποσυστήματος έγινε αποκλειστικά με PL/pg SQL. Λαμβάνει υποχρεωτικά ως είσοδο ένα ερώτημα σε SQL και προαιρετικά το username ενός χρήστη. Παραδίδει στην έξοδο το ερώτημα σε μορφή Optimized SQL ή Optimized PrefSQL το οποίο στη συνέχεια εκτελείται.

Το υποσύστημα έχει σχεδιαστεί και υλοποιηθεί με στόχο την βελτιστοποίηση του χρόνου εκτέλεσης ενός ερωτήματος το οποίο εκφράζει τα κριτήρια - επιθυμίες του χρήστη που θα απεικονίζονται στο αποτέλεσμα. Δεν διαχειρίζεται οποιοδήποτε SQL ερώτημα όπως π.χ. με Union ή Group by τελεστές ή εμφωλευμένα άλλα ερωτήματα. Ένα ερώτημα το οποίο

συγκεντρώνει κριτήρια - επιθυμίες αποτελείται από *Projections (Select r1.a, r1.b, r2.c .....), Relations (From r1, r2,....), Selections (Where r1.a=... and r2.c=.....)* και join conditions. Τα join conditions επειδή θεωρούνται δεδομένα δεν χρειάζεται να συμπεριλαμβάνονται στο ερώτημα της εισόδου. Αντί αυτών, στα Projection attributes του ερωτήματος εισόδου πρέπει να αναφέρονται τα join attributes των relations (χωρίς αυτό να σημαίνει ότι θα εμφανίζονται όλα στο αποτέλεσμα).

Στην περίπτωση του ανώνυμου χρήστη, η επεξεργασία περιορίζεται στην κατασκευή του βέλτιστου πλάνου εκτέλεσης του SQL ερωτήματος, ενώ για τον καταγεγραμμένο στο σύστημα χρήστη, διεξάγεται επί πλέον η ενσωμάτωση των προτιμήσεών του (με βάση το profile του) στο ερώτημα.

Α. Επεξεργασία ερωτήματος μη καταχωρημένου χρήστη (Είσοδος: SQL ερώτημα)

Όπως φαίνεται (γενικά) στην Εικόνα 14, οι ενέργειες του υποσυστήματος στην περίπτωση αυτή είναι:

- Συντακτική ανάλυση του ερωτήματος, κατακερματισμός και απομόνωση των Relations, των επί μέρους Selections και των Projection attributes.
- "Χτίσιμο" του βελτιστοποιημένου ερωτήματος με την παρακάτω διαδικασία:
  - Για κάθε Relation ώθηση των Projection attributes (αν υπάρχουν) και των Selections (αν υπάρχουν) προς την Relation που εξετάζεται με την βοήθεια του alias name της Relation και διαμόρφωση των επί μέρους queries.
  - Συνένωση των επί μέρους queries με την τοποθέτηση του *Join_gl2_exp (****)* Operator ο οποίος προορίζεται κυρίως για την διαχείριση αξιολογημένων από preferences δεδομένων, λειτουργεί όμως εξίσου καλά και για τα μη αξιολογημένα.

Β. Επεξεργασία ερωτήματος καταχωρημένου χρήστη (Είσοδος: SQL ερώτημα και username χρήστη)

Οι διαδικασίες φαίνονται συνοπτικά στην Εικόνα 15.

- Συντακτική ανάλυση του ερωτήματος, κατακερματισμός και απομόνωση των Relations, των επί μέρους Selections και των Projection attributes.

- "Χτίσιμο" του βελτιστοποιημένου ερωτήματος με την παρακάτω διαδικασία:
  - Για κάθε Relation ώθηση των Projection attributes (αν υπάρχουν) και των Selections (αν υπάρχουν) προς την Relation που εξετάζεται με την βοήθεια του alias name της Relation και διαμόρφωση των επί μέρους queries.
  - Για κάθε Relation ενσωμάτωση των προτιμήσεων του χρήστη με δύο διαφορετικές προσεγγίσεις μέσω του *Pref_gl1_exp (***)* Operator ή μέσω του *Pref_gl2_exp (****)* Operator.
  - Συνένωση των επί μέρους queries με την τοποθέτηση του *Join_gl1_exp (&^&)* Operator.

Σχέδιο εκτέλεσης του ερωτήματος

Ο τρόπος ενσωμάτωσης των προτιμήσεων του χρήστη στο ερώτημα εξαρτάται από το πλάνο εκτέλεσης του ερωτήματος. Τα πιθανά πλάνα εκτέλεσης που μπορούν να οδηγήσουν στην επίτευξη του βέλτιστου χρόνου εκτέλεσης του ερωτήματος είναι δύο:

PLAN - B

Οι Prefer Operators ορίζονται πάνω από τα επί μέρους Selections - Projections και ο καθένας ωθείται προς την Relation για την οποία προορίζεται. Εξ ορισμού ένα preference το οποίο επιδρά σε μια relation, χρησιμοποιεί ως scoring field ένα attribute αυτής της relation, καθώς επίσης στο condition του preference αναφέρονται attributes της συγκεκριμένης relation. Επομένως η ώθηση ενός prefer operator προς μια relation, είναι πάντα εφικτή.

Οι Join Operators ορίζονται πάνω από τους Prefer Operators:

$(\lambda_{p1} (\lambda_{p2} (\sigma_{\varphi 1} (R_1))))$ $\bowtie$ $(\lambda_{p3} (\lambda_{p4} (\sigma_{\varphi 2} (R_2))))$ $\bowtie$ $(\lambda_{p5} (\sigma_{\varphi 3} (R_3)))$

PLAN - C

Οι Join Operators ορίζονται πάνω από τα Projections - Selections. Οι Prefer Operators ορίζονται όλοι μαζί στο ενδιάμεσο αποτέλεσμα δηλαδή πάνω από τους Join Operators:

$\lambda_{p5} (\lambda_{p4} (\lambda_{p3} (\lambda_{p2} (\lambda_{p1} (\sigma_{\varphi 1} (R_1) \bowtie \sigma_{\varphi 2} (R_2) \bowtie \sigma_{\varphi 3} (R_3))))))$

Όπως αναφέρεται στο [*AK11*], το Plan-B οδηγεί στην μείωση του μεγέθους του input στους Prefer operators υποθέτοντας ότι το μέγεθος του αποτελέσματος των Join operators $[\sigma_{\varphi 1} (R_1) \bowtie \sigma_{\varphi 2} (R_2) \bowtie \sigma_{\varphi 3} (R_3)]$ θα είναι μεγαλύτερο από τα μεγέθη των $\sigma_{\varphi 1} (R_1)$, $\sigma_{\varphi 2} (R_2)$, $\sigma_{\varphi 3} (R_3)$.

Παράδειγμα 1

Έστω ότι ένας χρήστης ζητάει όλες τις ταινίες μετά το 1980 με το είδος τους και τους ηθοποιούς που παίζουν σε αυτές.

Το σύστημα γνωρίζει δύο προτιμήσεις του χρήστη : Του αρέσουν οι πρόσφατες ταινίες (από το 2000 και μετά) και έχει αδυναμία στις ταινίες δράσης.

Το ερώτημα εκφράζεται σε Pref SQL ως εξής:

PLAN-B

SELECT

*($$Select movieid, title, prod_year, score, conf from movies where prod_yea r> 1980$$ **** get_db_preference(4))*

&^&

*($$Select movieid, genre, score, conf from genres$$ **** get_db_preference(8))*

&^&

($$Select movieid, actorid, score, conf from movies2actors$$)

PLAN-C

SELECT

((($$Select movieid, title, prod_year, score, conf from movies where prod_year> 1980$$)

&^&

($$Select movieid, genre, score, conf from genres$$)

&^&

($$Select movieid, actorid, score, conf from movies2actors$$))

***** get_db_preference(4)) **** get_db_preference(8)*

Τα μεγέθη των relations του ερωτήματος είναι:

R1: Select ....from movies where prod_year > 1980 773.642 tuples.

R2: Select.....from genres 999.616 tuples

R3: Select.....from movies2actors 13.145.520 tuples

Το μέγεθος του $R1 \bowtie R2 \bowtie R3$ είναι: 6.546.235 tuples

Ο χρόνος εκτέλεσης του ερωτήματος στο Plan-B είναι 163 sec ενώ στο Plan-C είναι 294 sec.

Διαπιστώνουμε ότι το βέλτιστο πλάνο εκτέλεσης είναι το Plan-B, γεγονός το οποίο εξηγείται απόλυτα αν παρατηρήσουμε ότι:

Στο Plan-B ο πρώτος Prefer operator έχει input μεγέθους 773.642 tuples και ο δεύτερος έχει input μεγέθους 999.616 tuples.

Στο Plan-C και οι δύο Prefer operators έχουν input μεγέθους 6.546.325 tuples.

Παράδειγμα 2

Έστω ότι ο ίδιος χρήστης ζητάει όλες τις Αμερικάνικες ταινίες μετά το 1980 με το είδος τους.

Το ερώτημα εκφράζεται σε Pref - SQL ως εξής:

PLAN-B

```
SELECT
( $$Select movieid, title, prod_year, score, conf from movies where prod_yea r> 1980$$
  **** get_db_preference(4) )
 &^&
 ( $$Select movieid, genre, score, conf from genres$$ **** get_db_preference(8) )
 &^&
 ( $$Select movieid, country, score, conf from countries where country='USA'$$)
```

PLAN-C

```
SELECT
( ( ( $$Select movieid, title, prod_year, score, conf from movies where
    prod_year> 1980$$ )
 &^&
 ($$Select movieid, genre, score, conf from genres$$ )
  &^&
 ( $$Select movieid, country, score, conf from countries where country='USA'$$) )
  **** get_db_preference(4) ) **** get_db_preference(8)
```

Τα μεγέθη των relations του ερωτήματος είναι:

R1: Select ....from movies where prod_year > 1980	773.642 tuples.
R2: Select.....from genres	999.616 tuples
R3: Select.....from countries where country='USA'	326.992 tuples
Το μέγεθος του $R1 \bowtie R2 \bowtie R3$ είναι:	281.903 tuples

Παρατηρούμε ότι είναι μικρότερο από όλες τις relations του ερωτήματος.

Ο χρόνος εκτέλεσης του ερωτήματος στο Plan-B είναι 35 sec ενώ στο Plan-C είναι 24 sec.

Εδώ, το βέλτιστο πλάνο εκτέλεσης είναι το Plan-C γεγονός που εξηγείται αφού:

Στο Plan-B ο πρώτος Prefer operator έχει input μεγέθους 773.642 tuples και ο δεύτερος έχει input μεγέθους 999.616 tuples.

Στο Plan-C και οι δύο Prefer operators έχουν input μεγέθους 326.992 tuples.

Μετά από πολλαπλές εκτελέσεις ερωτημάτων και καταγραφή των χρόνων εκτέλεσης και των μεγεθών, προέκυψε το συμπέρασμα ότι:

Στα περισσότερα ερωτήματα ενός χρήστη στα πλαίσια μιας εφαρμογής συστάσεων, δηλώνονται διάφορα κριτήρια τα οποία περιορίζουν το αποτέλεσμα της αναζήτησης έτσι ώστε είναι δύσκολο το μέγεθος του αποτελέσματος (join product) να είναι μεγαλύτερο από τις επί μέρους relations του ερωτήματος, άρα σε τέτοιου είδους ερωτήματα είναι προτιμότερο να εφαρμόζεται το Plan-C. Αν όμως το ερώτημα περιέχει στοιχεία (attributes) από tables για τους οποίους δεν υπάρχει συνθήκη περιορισμού, δηλαδή ουσιαστικά ζητείται να συμμετάσχουν ολόκληροι αυτοί οι tables στο ερώτημα, τότε είναι πιθανό να είναι καταλληλότερο το Plan-B.

Ενσωμάτωση Προτιμήσεων χρήστη στο ερώτημα

Γίνεται ανάκτηση από τους tables της βάσης *user_preferences* και *preferences* όλων των preferences του χρήστη που απευθύνονται (*preference preftable*) στις relations του ερωτήματος. Στα projection του ερωτήματος πρέπει να υπάρχουν όλα τα attributes που χρειάζεται ένα preference δηλ. το *scoring field* και τα attributes που συγκροτούν το *condition* του preference. Εάν το ερώτημα έχει υποβληθεί από την Διεπαφή/Home Page το υποσύστημα έχει φροντίσει γι αυτό. Εάν όμως έχει υποβληθεί από την Διεπαφή/query editor ή από τον PostgreSQL-PgAdmin ή από το PostgreSQL-psql πρέπει ο συνθέτων το ερώτημα να εξασφαλίσει όλες τις προϋποθέσεις για την εκτέλεση του ερωτήματος.

Τα preferences ενσωματώνονται στο ερώτημα από τον Prefer operator είτε πάνω από τα αντίστοιχα Selections (Plan-B) είτε πάνω από τους Join operators (Plan-C). Χρησιμοποιείται η βοηθητική PL/PgSQL function *get_db_preference(p_id)*.

Είναι προφανές ότι η εφαρμογή ενός preference σε μια relation, δεν διαφοροποιεί το μέγεθος της relation. Ο Prefer Operator επιλέγει τα tuples τα οποία ικανοποιούν το *condition* του preference και εφαρμόζοντας την *scoring function* του preference, αποδίδει τιμές στα attributes *score* και *conf* του tuple, συναθροίζοντας τις προηγούμενες τιμές τους (αν υπάρχουν) με τις δικές του. Με τον τρόπο αυτό, προκύπτει η αξιολόγηση - βαθμολόγηση του tuple από πολλά preferences. Η output relation του Prefer Operator περιέχει τις affected και

τα μη affected από το preference tuples της input relation και ωθείται ως input relation σε έναν άλλο Prefer Operator.

Έχουν υλοποιηθεί δύο Prefer Operators που λειτουργούν με αυτόν τον τρόπο οι *Pref_gl1_exp* και *Pref_gl2_exp.*

Ο *Pref_gl2_exp* operator, δημιουργεί μια νέα relation από την input relation με διαμορφωμένα τα *score, conf* των affected tuples, την οποία επιστρέφει. Η χρήση του ενδείκνυται στις περιπτώσεις κατά τις οποίες το ποσοστό των affected από το preference tuples είναι σχετικά μεγάλο.

Ο *Pref_gl1_exp* operator, δημιουργεί μια νέα relation από την input relation μόνο την πρώτη φορά που εκτελείται (1ο Prefer). Στις επόμενες διαδοχικές εκτελέσεις, ενημερώνει αυτήν την ίδια relation. Ενδείκνυται στις περιπτώσεις μεγάλων relations στις οποίες ένα preference επηρεάζει λίγα tuples.

Παράδειγμα 1

Έστω ότι ένας χρήστης έχει προτίμηση στις πρόσφατες ταινίες αλλά και στις πολύ παλιές ταινίες πριν το1960. Και τα δυο preferences εφαρμόζονται στον table της βάσης *movies*.

Το μέγεθος του table *movies* είναι 1.573.104 tuples. Το πρώτο preference επιδρά σε 463.617 tuples, ενώ το δεύτερο σε 200.540 tuples.

Pref_gl1_exp Operator (***)

*Select ('movies' *** get_db_preference(4)) *** get_db_preference(9)*

1ο Prefer : Χρόνος: 2,9 sec (Ενέργεια : Create table...) + 22,37 sec (Ενέργεια : Update table...)

2ο Prefer : Χρόνος: 12,03 sec (Ενέργεια : Update table...)

Σύνολο χρόνου εκτέλεσης : **37,30 sec**

Pref_gl2_exp Operator (****)

*Select ('movies' **** get_db_preference(4)) **** get_db_preference(9)*

1ο Prefer : Χρόνος: 14,79 sec (Ενέργεια : Create table...)

2ο Prefer : Χρόνος: 9,05 sec (Ενέργεια : Create table...)

Σύνολο χρόνου εκτέλεσης : **23,84 sec**

Παράδειγμα 2

Έστω ότι ο ίδιος χρήστης έχει προτίμηση στις ταινίες όπου παίζει ο ηθοποιός Leonardo DiCaprio ή ο ηθοποιός Brad Pitt. Τα δύο preferences θα εφαρμοστούν στον table της βάσης *movies2actors.*

Το μέγεθος του table *movies2actors* είναι 13.145.520 tuples. Το πρώτο preference επιδρά σε 387 tuples ενώ το δεύτερο σε 241 tuples.

Pref_gl1_exp Operator (***)

*Select ('movies2actors' *** get_db_preference(41)) *** get_db_preference(30)*

1ο Prefer : Χρόνος: 14,99 sec (Ενέργεια: Create table...) + 1,69 sec (Ενέργεια: Update table...)

2ο Prefer : Χρόνος: 1,68 sec (Ενέργεια : Update table...)

Σύνολο χρόνου εκτέλεσης : **18,36 sec**

Pref_gl2_exp Operator (****)

*Select ('movies2actors' **** get_db_preference(41)) **** get_db_preference(30)*

1ο Prefer : Χρόνος: 45,91 sec (Ενέργεια : Create table...)

2ο Prefer : Χρόνος: 37,16 sec (Ενέργεια : Create table...)

Σύνολο χρόνου εκτέλεσης : **83,07 sec**

Επέκταση των σχεδίων εκτέλεσης ερωτημάτων

Σύμφωνα με τα σχέδια εκτέλεσης Plan-B και Plan-C , σε ένα ερώτημα ενσωματώνονται preferences τα οποία προορίζονται να επιδράσουν σε έναν αριθμό από tuples μόνο των relations που περιλαμβάνονται στο ερώτημα.

Σε μια εφαρμογή συστάσεων όμως, θα ήταν επιθυμητό να γίνεται αξιολόγηση και βαθμολόγηση του αποτελέσματος ενός ερωτήματος που θέτει ένας χρήστης, από όλες τις καταχωρημένες προτιμήσεις του και όχι μόνο από τις άμεσες με το ερώτημα.

Παράδειγμα

Οι γνωστές στο σύστημα προτιμήσεις ενός χρήστη είναι:

1. Του αρέσουν οι Αμερικάνικες ταινίες. (12)
2. Προτιμάει τις Κωμωδίες (3)
3. Του αρέσουν οι ταινίες με keyword "love" (38)
4. Είναι ικανοποιημένος όταν μια ταινία έχει λάβει πάνω από 15000 ψήφους (46)
5. Έχει αδυναμία στους ηθοποιούς Meryl Streep (19) και George Clooney (39)

Ο χρήστης αυτός θέτει ένα ερώτημα στο σύστημα ζητώντας: α)Κωμωδίες, β)Αμερικάνικες ταινίες, γ)Να έχουν keyword : "sex"

Το σύστημα κατασκευάζει το ερώτημα (Plan-C) :

SELECT

(((($$Select movieid, title, prod_year, score, conf from movies$$)

&^&

($$Select movieid, genre, score, conf from genres where genre like '%Comedy%'$$)

&^&

($$Select movieid, country, score, conf from countries where country='USA'$$)

&^&

($$Select movieid, keyword, score, conf from keywords where keyword='sex'$$))

***** get_db_preference(3)) **** get_db_preference(12)) **** get_db_preference(38)*

Το αποτέλεσμα του ερωτήματος θα αξιολογηθεί και θα βαθμολογηθεί από τα preferences : 3, 12, 38 επειδή αυτά αναφέρονται στις relations genres, countries, keywords οι οποίες υπάρχουν στο ερώτημα.

Θα ήταν όμως επιθυμητό, εάν στα αποτελέσματα υπάρχουν ταινίες με περισσότερες από 15000 ψήφους, ή ταινίες στις οποίες παίζουν η Meryl Streep ή ο George Clooney, αυτές οι ταινίες να πάρουν επιπλέον βαθμολόγηση.

Για το λόγο αυτό, έχει υλοποιηθεί μια επέκταση της διαδικασίας αξιολόγησης - βαθμολόγησης των αποτελεσμάτων ενός ερωτήματος από τις καταχωρημένες προτιμήσεις του χρήστη, οι οποίες κανονικά δεν θα συμμετείχαν στην διαβάθμιση και προβολή των αποτελεσμάτων του ερωτήματος.

Η επέκταση αυτή είναι στην κρίση του χρήστη αν θα την εφαρμόσει στην αναζήτησή του ή όχι.

Για τις ανάγκες της επέκτασης αυτής, έχουν υλοποιηθεί δύο ακόμα operators: Ο *Pref_gl10* Operator *(@@@@@)* και ο *Join_gl2* Operator *(####).*

Ο *Pref_gl10* Operator εφαρμόζοντας ένα preference σε μια relation, επιστρέφει μόνο τις affected από το preference tuples και βέβαια τις affected από προηγούμενα preferences tuples με συναθροιστική βαθμολόγηση στις κοινές ταινίες.

Ο *Join_gl2* Operator εκτελεί LEFT OUTER JOIN αντί για INNER JOIN με σκοπό να μην διαφοροποιήσει αριθμητικά τον αριστερό operand (που είναι το αποτέλεσμα του ερωτήματος με την κανονική αξιολόγηση - βαθμολόγηση) παρά μόνο να ενημερώσει τα *score, conf* σε κάποια tuples. Αυτό είναι αναγκαίο δεδομένου ότι -σε αντίθεση με τον *Join_gl1_exp* operator - ο δεξιός operand του *Join_gl2* (θα είναι το output του *Pref_gl10* operator) θα έχει περιορισμένο αριθμό tuples που αντιπροσωπεύουν τα επί πλέον ενδιαφέροντα του χρήστη και μόνο.

Παράδειγμα

Έστω ότι η relation *pr23_select* περιέχει το αποτέλεσμα του ερωτήματος του προηγούμενου παραδείγματος. Με την εκτέλεση του :

SELECT

$$pr23_select$$ #### ($$ratings$$ @@@@@ get_db_preference(46)) ####

(($$movies2actors$$ get_db_preference(19)) @@@@@ get_db_preference(39))

προκύπτει η relation *j22_yyy* η οποία αποτελεί προέκταση της *pr23_select* και είναι εμπλουτισμένη με τα επί πλέον attributes για *ratings* και *movies2actors*. Για τα επί πλέον αυτά attributes, υπάρχουν τιμές μόνο στα tuples που αφορούν ταινίες με ψήφους πάνω από 15000 ή ταινίες με την Meryl Streep ή τον George Clooney.

Ο χρόνος εκτέλεσης (του παραδείγματος) είναι 0,7 sec.

4.6.3 Υποσύστημα εκτέλεσης ερωτημάτων (III)

Η εκτέλεση των ερωτημάτων Pref-SQL πραγματοποιείται με την εκτέλεση των Τελεστών Προτίμησης (Operators) σε χαμηλότερο επίπεδο.

Οι operators οι οποίοι έχουν υλοποιηθεί με Pl/PgSQL, αποτελούν επέκταση της PostgreSQL με στόχο την αποτύπωση των preferences στα δεδομένα και την διαχείρισή τους.

Λαμβάνουν στην είσοδο μια ή δυο relations, δημιουργούν και επιστρέφουν μια νέα relation.

Επειδή η PostgreSQL function δεν έχει την δυνατότητα να επιστρέψει *TABLE* όταν δεν περιγράφεται το σχήμα του στην δήλωση της function, αναγκαστικά όλοι οι operators επιστρέφουν το όνομα της relation που δημιούργησαν, αντί για την ίδια την relation.

Με την γενική τακτική να λαμβάνεται και στην είσοδο των operators το όνομα της/των input relation/relations, επιτυγχάνονται οι διαδοχικές κλήσεις όλων των operators.

Έχει εφαρμοστεί μια πολιτική στα ονόματα των ενδιάμεσων relations που δημιουργούνται από τους operators: Κάθε κατηγορία (Prefer operators, Join operators, Union operator, Intersect operator) ορίζει ένα αρχικό γράμμα για το όνομα της relation που επιστρέφει, το οποίο ακολουθείται από έναν αύξοντα αριθμό (1 ή 2 ή 3 ψηφία). Το όνομα που επιλέγεται είναι το επόμενο διαθέσιμο αφού αποκλειστούν οι υπάρχουσες relations.

Για να σβήνονται οι relations αυτές από την βάση, εκτελείται η PL/pgSQL function *droptables* σε ανύποπτο χρόνο.

Στο σημείο αυτό κρίνεται σκόπιμο να αναφερθεί ότι η αρχική πρόθεση ήταν να ορίζονται οι ενδιάμεσες relations οι οποίες δημιουργούνται από τους operators, ως temporary tables ώστε να μην υφίστανται πλέον μετά το τέλος του session. Δυστυχώς όμως διαπιστώθηκε ότι στο θέμα αυτό η PostgreSQL δεν συμπεριφέρεται σωστά. Δημιουργούνται λανθασμένες εγγραφές στους Catalog tables : *pg_class, pg_attribute, pg_type,* η συνύπαρξη δε πολλαπλών sessions είναι απαγορευτική.

Παρακάτω παρουσιάζεται η συνοπτική περιγραφή των πέντε βασικών PrefSQL Operators:

1. ***pref_gl2_exp (operator : ****)***

   Δημιουργεί από την input relation μια νέα relation στην οποία εφαρμόζει ένα preference.

2. ***pref_gl1_exp (operator : ***)***

   Όταν στην είσοδο υπάρχει πίνακας της βάσης, ή Select Statement, δημιουργεί μια νέα relation στην οποία στη συνέχεια εφαρμόζει ένα preference με update (1ο Prefer). Όταν η input relation δεν είναι πίνακας της βάσης ή Select Statement (όχι 1ο Prefer), εφαρμόζει στην ίδια την input relation το preference με update.

3. ***join_gl1_exp (operator : &^&)***

   Δημιουργεί μια relation από το inner join των input relations εφαρμόζοντας συνάρτηση συνάθροισης στις τιμές των attributes *score, conf.*

4. ***union_gl1_exp (operator : &&&)***

   Δημιουργεί μια relation από το full join των input relations εφαρμόζοντας συνάρτηση συνάθροισης στις τιμές των attributes *score,conf*. Στη συνέχεια με κατάλληλες επεμβάσεις την μετατρέπει σε σωστή μορφή και την επιστρέφει.

5. ***intersection_gl1_exp (operator : ^^^)***

   Δημιουργεί μια relation από το inner join των input relations εφαρμόζοντας συνάρτηση συνάθροισης στις τιμές των attributes *score,conf*. Στη συνέχεια με κατάλληλες επεμβάσεις την μετατρέπει σε σωστή μορφή και την επιστρέφει.

5 Υλοποίηση

5.1 Περιγραφή Operators

5.1.1 pref_gl2_exp (operator : ****)

Είναι ο βασικός Prefer Operator της Pref SQL. Εκτελείται σε ικανοποιητικό χρόνο όταν διαχειρίζεται preferences που επιδρούν σε μεγάλο αριθμό tuples.

Operand αριστερά : (input relation) Select Statement ή

Όνομα DbTable (table Β.Δ) ή

Όνομα table : pr999_DbTable (επιστρέφεται από άλλο pref_gl),

pr999_select (επιστρέφεται από άλλο pref_gl),

j999_yyy (επιστρέφεται από join_gl

u999_yyy (επιστρέφεται από union_gl

i999_yyy (επιστρέφεται από intersection_gl)

Operand δεξιά : Αντικείμενο Preference (*pref_id, pref_description, prefer_table, scoring_field, condition, scoring_function_id, confidence*). →*p1*

Επιστρέφει: Το όνομα του table τον οποίο δημιουργεί και εφαρμόζει το Preference. Συμπεριλαμβάνει όλα τα tuples του αρχικού input table (ή του Select Statement) είτε έχουν επηρεαστεί από το *condition* του Preference είτε όχι.
Τύπος: *pr999_DbTable ή pr999_select.*

Εάν αριστερά έχει Select Statement το εκτελεί και δημιουργεί τον table *temp_sel*. Αυτός είναι ο preftable.
Εάν αριστερά έχει όνομα DbTable ή pr999_...., j999_yyy, u999_yyy, i999_yyy αυτός είναι ο preftable.
Προσθέτει στον preftable τα attributes *score, conf* αν δεν υπάρχουν.
Ανακτά από τους catalog tables *pg_class, pg_attribute* τα attributes του preftable και τα εισάγει στο *array attr1[].*
"Χτίζει" το query δημιουργίας της προς επιστροφή relation με τις παρακάτω ενέργειες:

A. **"Χτίζει" επί μέρους query με τα tuples του preftable που επηρεάζονται από το *condition* του preference:**

A.1 Τοποθετείται αρχικά στο προς κατασκευή query το αρχικό *SELECT:*

myquery3:='SELECT ';

A.2 Τοποθετούνται τα attributes του preftable από το *attr1[]* array:

myquery3:=myquery3||preftable||'.'||attr1[i]||',';

A.3 Τοποθετούνται τα attributes *score, conf.* Το *score* διαμορφώνεται από το αποτέλεσμα της *scoring_function* του preference πάνω στο *scoring_field* του preference το οποίο οπωσδήποτε υπάρχει ανάμεσα στα ανακτηθέντα attributes. Στο *conf* attribute ανατίθεται το *confidence* του preference. Στα attributes *score, conf* εφαρμόζεται συνάρτηση συνάθροισης των προηγούμενων τιμών με τις τιμές που αποδίδονται τώρα από το preference:

myquery3:=

myquery3||'f_avg_s('||s_function||'('||p1.scoring_field||'),'||p1.conf||',score,conf) as score, f_avg_c('||s_function||'('||p1.scoring_field||'),'||p1.conf||',score,conf) as conf ';

A.4 Τοποθετούνται τα keywords *FROM* και *WHERE*:

myquery3:=myquery3||' FROM '||preftable;

myquery3:=myquery3||' WHERE '||p1.condition;

Β. **"Χτίζει" επί μέρους query με τα tuples του preftable που ΔΕΝ επηρεάζονται από το *condition* του preference:**

Β.1 Τοποθετείται αρχικά στο προς κατασκευή query το αρχικό *SELECT:*

myquery4:='SELECT ';

και στη συνέχεια:

myquery4:=myquery4||' FROM '||preftable;

myquery4:=myquery4||' WHERE NOT ('||p1.condition||')';

Β.2 Τοποθετούνται τα attributes του preftable από το *attr1[]* array:

myquery4:=myquery4||preftable||'.'||attr1[i]||',';

Γ. Αφού βρει το όνομα της relation που θα δημιουργήσει, ενώνει τα δύο επί μέρους queries με UNION και εκτελεί το συνολικό query:

EXECUTE 'CREATE TABLE '||create_table||' AS (('||myquery3||') UNION ('||myquery4||'))';

Παράδειγμα 1 :

*SELECT 'ratings' **** get_db_preference(17) **** get_db_preference(18)*

Επιστρέφει : *pr02_ratings*

Παράδειγμα 2 :

*SELECT 'Select movieid ,rating ,votes ,score,conf FROM ratings WHERE rating >= 9' **** get_db_preference(17) **** get_db_preference(18)*

Επιστρέφει : *pr36_select*

*5.1.2 pref_gl1_exp (operator : ***)*

Είναι ο δεύτερος Prefer Operator της Pref SQL. Υπερτερεί του pref_gl2_exp όταν διαχειρίζεται preferences που εφαρμόζονται σε μεγάλες relations και επιδρούν σε μικρό αριθμό tuples.

Operand αριστερά : (input relation) Select Statement ή

Όνομα DbTable (table Β.Δ) ή

Όνομα table : pr999_DbTable (επιστρέφεται από άλλο pref_gl),

pr999_select (επιστρέφεται από άλλο pref_gl),
j999_yyy (επιστρέφεται από join_gl),
u999_yyy (επιστρέφεται από union_gl),
i999_yyy (επιστρέφεται από intersection_gl)

Operand δεξιά : Αντικείμενο Preference *(pref_id, pref_description, prefer_table, scoring_field, condition, scoring_function_id, confidence).* →*p1*

Επιστρέφει: Το όνομα του table τον οποίο δημιουργεί και εφαρμόζει το Preference. Συμπεριλαμβάνει όλα τα tuples του αρχικού input table (ή του Select Statement) είτε έχουν επηρεαστεί από το *condition* του Preference είτε όχι.
Τύπος: pr999_DbTable ή pr999_select.

Εάν αριστερά έχει Select Statement (1ο prefer), το εκτελεί και δημιουργεί τον table pr999_select στον οποίο προσθέτει τα attributes *score, conf*. Αυτός είναι ο preftable.
Εάν αριστερά έχει όνομα DbTable (1ο prefer), δημιουργεί από αυτόν (από όλα τα tuples) τον table pr999_DbTable. Αυτός είναι ο *preftable.*
Εάν αριστερά έχει όνομα table pr999_....., u999_yyy, j999_yyy, i999_yyy (όχι 1ο prefer) **δεν δημιουργεί κανέναν table**, ο ίδιος ο input-operand table είναι ο *preftable.*
Εφαρμόζει με απλό **update** το Preference στον *preftable* διαμορφώνοντας τις columns *score* και *con*f των tuples του *preftable* τα οποία επηρεάζονται από το *condition* του Preference ως εξής:
Για κάθε tuple, η τιμή του score attribute προκύπτει από την συνάθροιση της προϋπάρχουσας τιμής του, με την τιμή που επιστρέφει η *scoring function* του Preference εφαρμοζόμενη στο *scoring_field του* Preference, το οποίο οπωσδήποτε υπάρχει ανάμεσα στα attributes του *preftable.*
Για κάθε tuple, η τιμή του *conf* attribute προκύπτει από την συνάθροιση της προϋπάρχουσας τιμής του, με την τιμή του *confidence* του preference:

```
EXECUTE 'UPDATE '||preftable||' SET
score=f_avg_s('||s_function||'('||p1.scoring_field||'),'||p1.conf||',score,conf),'||
'conf =f_avg_c('||s_function||'('||p1.scoring_field||'),'||p1.conf||',score,conf) '||   'WHERE'
||p1.condition;
```

Παράδειγμα

```
SELECT 'movies2directors' *** get_db_preference(33) *** get_db_preference(34)
```

Επιστρέφει : *pr04_movies2directors*

5.1.3 join_gl1_exp (operator : &^&)

Operands αριστερά-δεξιά :	Select Statement ή	
	Όνομα table :	pr999_DbTable (επιστρέφεται από pref_gl), pr999_select (επιστρέφεται από pref_gl), j999_yyy (επιστρέφεται από άλλο join_gl), u999_yyy (επιστρέφεται από union_gl), i999_yyy (επιστρέφεται από intersection_gl)

Επιστρέφει :	Το όνομα του table ο οποίος δημιουργήθηκε από το join. Τύπος: j999_yyy
Προϋπόθεση :	Οι operands να συμπεριλαμβάνουν κοινό attribute (primary / foreign key). Δεν είναι απαραίτητο να περιέχουν και οι δύο operands τα attributes score, conf.

Εάν αριστερά έχει Select Statement, το εκτελεί και δημιουργεί τον ενδιάμεσο table xxx. Αυτός είναι ο table1.
Εάν δεξιά έχει Select Statement, το εκτελεί και δημιουργεί τον ενδιάμεσο table xxx1. Αυτός είναι ο table2.
Εάν αριστερά έχει όνομα table , αυτός είναι ο table1.
Εάν δεξιά έχει όνομα table, αυτός είναι ο table2.
Από τους catalog tables *pg_class, pg_attribute* ανακτά τα attributes των table1, table2 και τα τοποθετεί σε arrays. Στη συνέχεια εντοπίζει το κοινό attribute (το πρώτο στη σειρά) των table1, table2 το οποίο λαμβάνεται ως join-attribute. Αυτό είναι το id της βασικής οντότητας που στην περίπτωσή μας είναι το movieid. Η function επιστρέφει λάθος αν δεν ικανοποιείται αυτή η προϋπόθεση.
Σχηματίζει το query για το join εφαρμόζοντας συνάρτηση συνάθροισης στις τιμές των attributes score, conf (εφόσον υπάρχουν και στους δύο tables) και INNER JOIN operator.
Εκτελεί το query και δημιουργεί τον table j999_yyy το όνομα του οποίου επιστρέφει.
Εάν οι input relations δεν έχουν αξιολογηθεί από preferences, η function λειτουργεί ως απλός SQL join operator.

Παράδειγμα 1

SELECT

*(('movies' **** get_db_preference(9)) **&^&** ('genres' **** get_db_preference(3))) **&^&** ('ratings' **** get_db_preference(18))*

Επιστρέφει *: j29_yyy*

Παράδειγμα 2

SELECT

*((($$Select movieid ,title ,prod_year ,score,conf FROM movies WHERE mtype='film' and prod_year>=2010$$ **** get_db_preference(7)) **** get_db_preference(4)) **** get_db_preference(47)) **&^&***

*((($$Select movieid ,country ,score,conf FROM countries WHERE country='USA'$$ **** get_db_preference(16)) **** get_db_preference(12)) **** get_db_preference(45))*

Επιστρέφει *: j31_yyy*

5.1.4 union_gl1_exp (operator : &&&)

Operands αριστερά-δεξιά :	Select Statement ή	
	Όνομα table :	pr999_..... (επιστρέφεται από pref_gl), u999_yyy (επιστρέφεται από union_gl), j999_yyy (επιστρέφεται από άλλο join_gl) i999_yyy (επιστρέφεται από intersection_gl)
Επιστρέφει :		Το όνομα του table ο οποίος δημιουργήθηκε από το union. Τύπος: u999_yyy
Προϋπόθεση :		Οι operands να έχουν το ίδιο structure και το πρώτο στη σειρά να είναι primary / foreign key.

Εάν αριστερά έχει Select Statement, το εκτελεί και δημιουργεί τον ενδιάμεσο table xxx στον οποίο προσθέτει τα attributes *score, conf*. Αυτός είναι ο table1.
Εάν δεξιά έχει Select Statement, το εκτελεί και δημιουργεί τον ενδιάμεσο table xxx1 στον οποίο προσθέτει τα attributes *score, conf*. Αυτός είναι ο table2.
Εάν αριστερά έχει όνομα table, αυτός είναι ο table1.
Εάν δεξιά έχει όνομα table, αυτός είναι ο table2.
Από τους catalog tables *pg_class, pg_attributes* βρίσκει τα attributes των table1, table2, τα τοποθετεί σε arrays και ελέγχει αν οι δύο tables έχουν τα ίδια attributes στην ίδια σειρά. Αν αυτό δεν ισχύει, επιστρέφει λάθος. Στη συνέχεια εντοπίζει το κοινό attribute (το πρώτο στη σειρά) των table1, table2 με βάση το οποίο θα γίνει συνάθροιση των *score,conf*. Αυτό είναι το id της βασικής οντότητας που στην περίπτωσή μας είναι το *movieid*. Η function επιστρέφει λάθος αν δεν ικανοποιείται αυτή η προϋπόθεση. Τέλος αν δεν έχουν και οι δύο tables attributes *score,conf* (δεν υπάρχει λόγος να καλείται η function) επιστρέφει λάθος.
Σχηματίζει το query για το union εφαρμόζοντας FULL JOIN operator προκειμένου να εντοπίσει τα αντίστοιχα tuples στους δύο tables με ίδια τιμή στο κοινό attribute. Κατά τον σχηματισμό του query, εφαρμόζει συνάρτηση συνάθροισης στις τιμές των attributes *score, conf*.
Εκτελεί το query και δημιουργεί τον table u999_yyy ο οποίος όμως ως αποτέλεσμα join δεν έχει τη σωστή μορφή.
Με τις κατάλληλες επεμβάσεις (updates και alter table) ο table u999_yyy έρχεται στην σωστή μορφή και επιστρέφεται.
Παράδειγμα

SELECT

*('select * from ratings where votes>10000' *** get_db_preference(17))* ***&&&*** *('select * from ratings where rating>6' *** get_db_preference(46))*

Επιστρέφει : *u01_yyy*

5.1.5 intersection_gl1_exp (operator : ^^^)

Operands αριστερά-δεξιά :	Select Statement ή	
	Όνομα table :	pr999_..... (επιστρέφεται από pref_gl),
		u999_yyy (επιστρέφεται από άλλο union_gl),
		j999_yyy (επιστρέφεται από join_gl)

Επιστρέφει: Το όνομα του table ο οποίος δημιουργήθηκε από το intersection.

Τύπος: i999_yyy.

Εάν αριστερά έχει Select Statement, το εκτελεί και δημιουργεί τον ενδιάμεσο table xxx στον οποίο προσθέτει τα attributes score, conf. Αυτός είναι ο table1.
Εάν δεξιά έχει Select Statement, το εκτελεί και δημιουργεί τον ενδιάμεσο table xxx1 στον οποίο προσθέτει τα attributes *score, conf*. Αυτός είναι ο table2.
Εάν αριστερά έχει όνομα table , αυτός είναι ο table1.
Εάν δεξιά έχει όνομα table , αυτός είναι ο table2.
Από τους catalog tables βρίσκει τα attributes των table1, table2, τα τοποθετεί σε arrays και ελέγχει αν οι δύο tables έχουν τα ίδια attributes στην ίδια σειρά. Αν αυτό δεν ισχύει, επιστρέφει λάθος. Στη συνέχεια εντοπίζει το κοινό attribute (το πρώτο στη σειρά) των table1, table2 με βάση το οποίο θα γίνει συνάθροιση των *score,conf*. Αυτό είναι το id της βασικής οντότητας που στην περίπτωσή μας είναι το *movieid*. Η function επιστρέφει λάθος αν δεν ικανοποιείται αυτή η προϋπόθεση. Τέλος αν δεν έχουν και οι δύο tables attributes *score,conf* (δεν υπάρχει λόγος να καλείται η function) επιστρέφει λάθος.
Σχηματίζει το query για το union εφαρμόζοντας INNER JOIN operator προκειμένου να εντοπίσει τα αντίστοιχα tuples στους δύο tables με ίδια τιμή στο κοινό attribute. Κατά τον σχηματισμό του query, εφαρμόζει συνάρτηση συνάθροισης στις τιμές των attributes *score, conf*.
Εκτελεί το query και δημιουργεί τον table i999_yyy ο οποίος όμως ως αποτέλεσμα join δεν έχει τη σωστή μορφή.
Με τις κατάλληλες επεμβάσεις (updates και alter table) ο table i999_yyy έρχεται στην σωστή μορφή και επιστρέφεται.

Παράδειγμα

SELECT

*('select * from ratings where votes>10000' *** get_db_preference(17))* **^^^** *('select * from ratings where rating>6' *** get_db_preference(46))*

Επιστρέφει : *i01_yyy*

5.2 Περιγραφή PL/Pg Functions

5.2.1 make_query_plan_B

Μετατρέπει το SQL ερώτημα σε Pref SQL ερώτημα εφαρμόζοντας το σχέδιο εκτέλεσης Plan-B και το δίνει για εκτέλεση.

Είσοδος : Username χρήστη, SQL ερώτημα

Επιστρέφει : Το όνομα της relation η οποία περιέχει τα αποτελέσματα του ερωτήματος.

Συντακτική ανάλυση του SQL ερωτήματος και κατακερματισμός.

A. Εντοπίζονται οι θέσεις των keywords *FROM, WHERE, ORDER BY*. Από αυτά μόνο το *FROM* είναι βέβαιο ότι υπάρχει οπωσδήποτε.

Εντοπίζονται οι tables (FROM ...) του ερωτήματος και τα alias names των tables και εισάγονται σε αντίστοιχα arrays.

Εντοπίζονται τα Projection attributes (SELECT ...) και τα alias names των tables στους οποίους ανήκουν τα attributes και εισάγονται και αυτά σε αντίστοιχα arrays.

Εάν υπάρχει condition στο ερώτημα, εντοπίζονται οι επί μέρους συνθήκες (selections), τα attributes των selections και τα alias names των tables στους οποίους ανήκουν τα attributes. Τα selections και τα alias names εισάγονται σε αντίστοιχα arrays.

"Χτίσιμο" του *Optimized Pref SQL query*

B. Τοποθετείται αρχικά στο προς κατασκευή query το αρχικό *SELECT:*

myquery0:='SELECT ';

Γ. **Για κάθε table του array των tables (tables[i] : εξεταζόμενος table)**

Γ.1 Εάν δεν πρόκειται για τον πρώτο εξεταζόμενο table, τοποθετείται ο join_gl1_exp (&^&) operator στο προς κατασκευή query*:*

myquery0 :=myquery0||' &^& ';

Γ.2 Εντοπίζονται από το array των Projection attributes με τη βοήθεια του αντίστοιχου array των alias names, τα attributes που ανήκουν στον tables[i] και παρατάσσονται στο query μετά από το keyword *Select*. Στο τέλος προστίθενται τα attributes *score, conf*:

myquery0:=myquery0||'$$Select '||select_attr[j]||' ';

myquery0:=myquery0||','||select_attr[j]||' ';

myquery0 := myquery0||',score,conf';

Γ.3 Τοποθετείται το keyword *FROM* ακολουθούμενο από τον tables[i].

myquery0:=myquery0||' FROM '||tables[i];

Γ.4 Ανακτώνται από τους tables της βάσης *user_preferences* και *preferences* τα preferences του χρήστη που έχουν ως *preftable* τον tables[i] και τοποθετούνται σε array.

Γ.5 **Για κάθε preference του array** *(vid_pref* **):**

Γ.5.1 Εάν πρόκειται για το 1ο Preference για τον tables[i]:

Εντοπίζονται από το array των Selections με την βοήθεια του αντίστοιχου array των alias names, οι συνθήκες για τον tables[i] και τοποθετούνται στο υπό κατασκευή query, πίσω από το keyword *WHERE:*

selection_where:='WHERE ';

selection_where:=selection_where||' and '||cond_sf[j];

Αφού ολοκληρωθεί το selection για τον tables[i], τοποθετείται το εξεταζόμενο preference (*vid_pref)* με τον *pref_gl2_exp* operator ή με τον *pref_gl1_exp* operator.

*myquery0 := myquery0||' '||selection_where||'$$ **** get_db_preference('||vid_pref||')) '*

Γ.5.2 Εάν δεν πρόκειται για το 1ο Preference:

Τοποθετείται το εξεταζόμενο preference με τον *pref_gl2_exp* ή με τον *pref_gl1_exp* operator.

Γ.6 Εάν για τον tables[i] δεν υπάρχουν καταχωρημένα preferences, εντοπίζονται από το array των Selections με την βοήθεια του αντίστοιχου array των alias names, οι συνθήκες για τον tables[i] και τοποθετούνται στο υπό κατασκευή query, πίσω από το keyword *WHERE.*

Δ. Το query το οποίο "χτίστηκε", εκτελείται και επιστρέφεται το όνομα της relation η οποία δημιουργήθηκε από το τελευταίο join_gl1_exp που εκτελέστηκε.
Το Optimized Pref SQL query που κατασκευάστηκε, τυπώνεται στο output.

Παράδειγμα

select make_query_plan_B*('john' ,'SELECT m.movieid,m.title,m.prod_year ,r.movieid, r.rating, r.votes , c.movieid, c.country ,l.movieid, l.language FROM movies m ,ratings r , countries c , language l WHERE m.mtype="film" and m.prod_year>=2010 and r.rating >= 9.5 and c.country="USA" and l.language="English" ')*

Κατασκευάζεται το PrefSQL query:

SELECT

*((($$Select movieid ,title ,prod_year ,score,conf FROM movies WHERE mtype='film' and prod_year>=2010$$ **** get_db_preference(7)) **** get_db_preference(4)) **** get_db_preference(47)) &^&*

($$Select movieid ,rating ,votes ,score,conf FROM ratings WHERE rating >= 9$$) &^&

*((($$Select movieid ,country ,score,conf FROM countries WHERE country='USA'$$ **** get_db_preference(16)) **** get_db_preference(12)) **** get_db_preference(45)) &^&*

($$Select movieid ,language ,score,conf FROM language WHERE language='English'$$)

Επιστρέφεται η relation *: j18_yyy*

5.2.2 make_query_plan_C

Μετατρέπει το SQL ερώτημα σε Pref SQL ερώτημα εφαρμόζοντας το σχέδιο εκτέλεσης Plan-C και το δίνει για εκτέλεση.

Είσοδος : Username χρήστη, SQL ερώτημα

Επιστρέφει : Το όνομα της relation η οποία περιέχει τα αποτελέσματα του ερωτήματος.

Συντακτική ανάλυση του SQL ερωτήματος και κατακερματισμός.

A. Εντοπίζονται οι θέσεις των keywords *FROM, WHERE, ORDER BY*. Από αυτά μόνο το *FROM* είναι βέβαιο ότι υπάρχει οπωσδήποτε.

Εντοπίζονται οι tables (FROM ...) του ερωτήματος και τα alias names των tables και εισάγονται σε αντίστοιχα arrays.

Εντοπίζονται τα Projection attributes (SELECT ...) και τα alias names των tables στους οποίους ανήκουν τα attributes και εισάγονται και αυτά σε αντίστοιχα arrays.

Εάν υπάρχει condition στο ερώτημα, εντοπίζονται οι επί μέρους συνθήκες (selections), τα attributes των selections και τα alias names των tables στους οποίους ανήκουν τα attributes. Τα selections και τα alias names εισάγονται σε αντίστοιχα arrays.

"Χτίσιμο" του *Optimized Pref SQL query*

B. **Για κάθε table του array των tables (*tables[i]* : εξεταζόμενος table)**

B.1 Εάν δεν πρόκειται για τον πρώτο εξεταζόμενο table, τοποθετείται ο *join_gl1_exp* (&^&) operator στο προς κατασκευή query:

myquery0 :=myquery0||' &^& ';

B.2 Εντοπίζονται από το array των Projection attributes με τη βοήθεια του αντίστοιχου array των alias names, τα attributes που ανήκουν στον *tables[i]* και παρατάσσονται στο query μετά από το keyword *Select*. Στο τέλος προστίθενται τα attributes *score, conf*:

myquery0:=myquery0||'$$Select '||select_attr[j]||' ';

myquery0:=myquery0||','||select_attr[j]||' ';

myquery0 := myquery0||',score,conf';

B.3 Τοποθετείται το keyword *FROM* ακολουθούμενο από τον *tables[i]:*

myquery0:=myquery0||' FROM '||tables[i];

B.4 Εντοπίζονται από το array των Selections με την βοήθεια του αντίστοιχου array των alias names, οι συνθήκες για τον *tables[i]* και τοποθετούνται στο υπό κατασκευή query, πίσω από το keyword *WHERE:*

selection_where:='WHERE ';

selection_where:=selection_where||' and '||cond_sf[j];

myquery0 := myquery0||' '||selection_where||'$$)';

Γ. **Για κάθε table του array των tables (*tables[i]* : εξεταζόμενος table)**

Γ.1 Ανακτώνται από τους tables της βάσης *user_preferences* και *preferences* τα preferences του χρήστη που έχουν ως *preftable* τον *tables[i]* και τοποθετούνται σε array.

Τα preferences και οι *pref_gl2_exp* operators θα συνταχθούν όλοι μαζί στο τέλος του query.

Γ.2 **Για κάθε preference του array (*pref_id*):**

Γ.2.1 Τοποθετείται στο query το εξεταζόμενο preference με τον *pref_gl2_exp* operator ή με τον *pref_gl1_exp* operator:

*myquery0 :=myquery0||') **** get_db_preference('||vid_pref||') ';*

Δ. Τοποθετείται στην αρχή του προς κατασκευή query το αρχικό *SELECT*. Δεν τοποθετήθηκε στην αρχή της function διότι έπρεπε να τοποθετηθούν οι ανάλογες με τα preferences παρενθέσεις που ακολουθούν το αρχικό SELECT:

myquery0:='SELECT '||myquery0;

Ε. Το query το οποίο "χτίστηκε", εκτελείται και επιστρέφεται το όνομα της relation η οποία δημιουργήθηκε από το τελευταίο pref_gl2_exp που εκτελέστηκε.

Το Optimized Pref SQL query που κατασκευάστηκε, τυπώνεται στο output.

Παράδειγμα

select make_query_plan_C*('john' ,'SELECT m.movieid,m.title,m.prod_year ,r.movieid, r.rating, r.votes , c.movieid, c.country ,l.movieid, l.language FROM movies m ,ratings r , countries c , language l WHERE m.mtype="film" and m.prod_year>=2010 and r.rating >= 9.5 and c.country="USA" and l.language="English" ')*

Κατασκευάζεται το PrefSQL query:

SELECT

(((((((($$Select movieid ,title ,prod_year ,score,conf FROM movies where mtype='film' and prod_year>=2010 and prod_year<2019$$) &^&

($$Select movieid ,rating ,votes ,score,conf FROM ratings where rating >= 9$$) &^&

($$Select movieid ,country ,score,conf FROM countries where country='USA'$$) &^&

*($$Select movieid ,language ,score,conf FROM language where language='English'$$)) **** get_db_preference(7)) **** get_db_preference(4)) **** get_db_preference(47)) **** get_db_preference(16)) **** get_db_preference(12)) **** get_db_preference(45)*

Επιστρέφεται η relation *: pr16_select*

5.2.3 pref_gl10 (operator : @@@@@*)*

Operand αριστερά : (input relation)	Όνομα dbTable (table Β.Δ) ή Όνομα table: pr999_dbTable (επιστρέφεται από άλλο pref_gl)
Operand δεξιά :	Αντικείμενο Preference *(pref_id, pref_description, prefer_table, scoring_field, condition, scoring_function_id, confidence*). →p1
Επιστρέφει:	Το όνομα του table (preftable) τον οποίο δημιουργεί και εφαρμόζει το Preference. Συμπεριλαμβάνει μόνο τα tuples του αρχικού table που έχουν επηρεαστεί από το *condition* του preference. Τύπος: pr999_dbTable.

Εάν αριστερά έχει όνομα dbTable (1^{ο} prefer), δημιουργεί τον table pr999_dbTable από τα tuples

του dBTable που επηρεάζονται από το Condition του Preference. Αυτός είναι ο preftable.
Εάν αριστερά έχει όνομα table pr999_ dbTable (όχι 1ο prefer), τότε από το όνομα του input table (pr999_dbTable) συμπεραίνει το όνομα του πρωτογενούς table της βάσης (dbTable) από τον οποίο προήλθε ο pr999_DbTable και δημιουργεί τον ενδιάμεσο table kkk από τα tuples του dbTable που επηρεάζονται από το *Condition* του preference. Αυτός είναι ο preftable δηλαδή ο table που θα εφαρμοστεί το p1 preference:

*EXECUTE 'CREATE TABLE '||create_table||' AS SELECT * FROM '||input_text||' WHERE ' ||p1.condition;*

Εφαρμόζει με απλό **update** το p1 preference στον preftable όπως η *pref_gl1_exp:*

EXECUTE 'UPDATE '||preftable||' SET
score=f_avg_s('||s_function||'('||p1.scoring_field||'),'||p1.conf||',score,conf),'||
'conf =f_avg_c('||s_function||'('||p1.scoring_field||'),'||p1.conf||',score,conf) ';.

Εάν πρόκειται για 1ο prefer, ο προς επιστροφή table είναι έτοιμος.
Εάν δεν πρόκειται για 1ο prefer, ο προς επιστροφή table (create_table) δημιουργείται από το *union_gl10* του table kkk με τον input table (input_text):
myquery1:='SELECT union_gl10($$'||input_text||'$$,$$kkk$$,$$'||create_table||'$$)';
Στον ενδιάμεσο table kkk, έχει εφαρμοστεί το p1 preference, ενώ στον *input_text* table έχουν εφαρμοστεί ήδη τα προηγούμενα preferences. Η function *union_gl10* συναθροίζει τις τιμές των *score, conf* των tables που υπεισέρχονται σε αυτήν ώστε τα tuples του table που επιστρέφεται να έχουν αξιολογηθεί από όλα τα μέχρι τώρα preferences της ακολουθίας. Δεν περιέχονται τα tuples τα οποία δεν έχουν αξιολογηθεί από κανένα preference της ακολουθίας.

Εφαρμόζει με ανορθόδοξο τρόπο το Preference και επιστρέφει **μόνο τα αξιολογημένα tuples** από ένα ή περισσότερα preferences.

Παράδειγμα
SELECT 'keywords' @@@@@ get_db_preference(26) @@@@@ get_db_preference(37) @@@@@ get_db_preference(2)

Επιστρέφει : pr31_keywords (μόνο affected tuples)

5.2.4 join_gl2 (operator : ####)

Operands αριστερά-δεξιά : Select Statement ή
Όνομα table : pr999_DbTable (επιστρέφεται από pref_gl),
pr999_select (επιστρέφεται από pref_gl),

j999_yyy (επιστρέφεται από άλλο join_gl),
u999_yyy (επιστρέφεται από union_gl),
i999_yyy (επιστρέφεται από intersection_gl)

Επιστρέφει : Το όνομα του table ο οποίος δημιουργήθηκε από το join.
Τύπος: j999_yyy

Προϋπόθεση : Οι operands να συμπεριλαμβάνουν κοινό attribute (primary / foreign key). Δεν είναι απαραίτητο να περιέχουν και οι δύο operands τα attributes score, conf.

Είναι ακριβώς ίδια με την *join_gl1_exp* με τη διαφορά ότι εκτελεί LEFT OUTER JOIN ώστε να μην διαφοροποιηθεί αριθμητικά το περιεχόμενο του αριστερού operand-table π.χ. από έναν δεξιό operand table με αξιολογημένα μόνο tuples, ο οποίος έχει προκύψει από την *pref_gl10* (η οποία επιστρέφει αξιολογημένες μόνο τα tuples που έχουν επηρεαστεί από το condition του preference).

Παράδειγμα

SELECT

$$pr36_select$$ #### (($$keywords$$ @@@@@ get_db_preference(26)) @@@@@ get_db_preference(38))

Επιστρέφει : *j25_yyy*

5.2.5 join_gl3 (operator : ####@)

Operands αριστερά-δεξιά — Όνομα table : pr999_DbTable (επιστρέφεται από pref_gl),
j999_yyy (επιστρέφεται από άλλο join_gl),
u999_yyy (επιστρέφεται από union_gl),
i999_yyy (επιστρέφεται από intersection_gl)

Επιστρέφει : Το όνομα του table ο οποίος δημιουργήθηκε από το join.
Τύπος: j99_yyy

Προϋποθέσεις : Οι operands να συμπεριλαμβάνουν κοινό

attribute (primary / foreign key).

Και οι δυο tables περιέχουν attributes score, conf.

Διαχειρίζεται μόνο τα tuples τα οποία έχουν βαθμολογηθεί από preferences.

Από τον αριστερό table δημιουργεί τον ενδιάμεσο table xxx από τα tuples του αριστερού table που έχουν αξιολογηθεί από preferences**.** Αυτός είναι ο table1.

Από τον δεξιό table δημιουργεί τον ενδιάμεσο table xxx1 από τα tuples του δεξιού table που έχουν αξιολογηθεί από preferences. Αυτός είναι ο table2.

Από τους catalog tables βρίσκει τα attributes των table1, table2 και τα τοποθετεί σε arrays. Στη συνέχεια εντοπίζει το κοινό attribute (το πρώτο στη σειρά) των table1, table2 το οποίο λαμβάνεται ως join-attribute. Αυτό είναι το id της βασικής οντότητας που στην περίπτωσή μας είναι το movieid. Η function επιστρέφει λάθος αν δεν ικανοποιείται αυτή η προϋπόθεση.

Σχηματίζει το query για το join εφαρμόζοντας συνάρτηση συνάθροισης στις τιμές των attributes score, conf και FULL JOIN operator.

Εκτελεί το query και δημιουργεί τον table j99_yyy το όνομα του οποίου επιστρέφει μετά από update επεμβάσεις ώστε να έρθει σε σωστή μορφή.

Με την εφαρμογή του full join επιτυγχάνεται να συμπεριλαμβάνονται στο αποτέλεσμα και τα tuples τα οποία υπάρχουν μόνο στον ένα από τους δυο tables.

Χρησιμοποιείται μόνο για την εξαγωγή συστάσεων με βάση τα καταχωρημένα preferences ενός χρήστη.

Παράδειγμα

SELECT ($$language$$ @@@@@ get_db_preference(15)) ####@ ($$genres$$ @@@@@ get_db_preference(8))

Επιστρέφει : *j32_yyy* (όλα τα tuples affected)

5.2.6 make_query_plan_B_add

Μετατρέπει το SQL ερώτημα σε Pref SQL ερώτημα με σχέδιο εκτέλεσης Plan-B, στη συνέχεια ενσωματώνει όλες τις προτιμήσεις του χρήστη και το δίνει για εκτέλεση.

Είσοδος : Username χρήστη, SQL ερώτημα

Επιστρέφει : Το όνομα της relation η οποία περιέχει τα αποτελέσματα του ερωτήματος.

Διεργασίες make_query_plan_B

Α. Όπως ακριβώς γίνεται στην *make_query_plan_B* με τις ίδιες διαδικασίες, δημιουργείται η relation *j99_yyy* ή *j999_yyy* με τα αποτελέσματα του ερωτήματος αξιολογημένα από τις καταχωρημένες προτιμήσεις του χρήστη που είναι συναφείς με τα κριτήρια του ερωτήματος.

Εμπλουτισμός του αποτελέσματος με πρόσθετες προτιμήσεις

Β. Ανακτώνται με cursor τα preftable και ο αριθμός των preferences για κάθε preftable από τους tables *user_preferences, preferences* (που αφορούν τον χρήστη) *group by* preftable.

→ *vpreftable / prefs_on_same_table*

Γ. **Για κάθε set: *vpreftable / prefs_on_same_table* που εξετάζονται:**

Γ.1 Εξετάζεται αν ο vpreftable υπάρχει ανάμεσα στους tables του array tables[] που περιέχει τους tables του ερωτήματος για τους οποίους τα καταχωρημένα preferences του χρήστη, έχουν ήδη αξιολογήσει το αποτέλεσμα του ερωτήματος.

Γ.2 Εάν διαπιστωθεί ότι δεν υπάρχει, ανακτώνται με δεύτερο cursor τα καταχωρημένα preferences του χρήστη που έχουν preftable = vpreftable.

Γ.3 **Για κάθε *preference (vid_pref)* που εξετάζεται :**

Γ.3.1 Εάν πρόκειται για το 1ο vid_pref που εξετάζεται, "χτίζεται" πρόταση που περιλαμβάνει τον vpreftable. ακολουθούμενο από το preference vid_pref εφαρμοζόμενο στον vpreftable με τον *pref_gl10* operator. δηλαδή:

myquery2 := myquery2||'$$'||vpreftable||'$$ @@@@@ get_db_preference('||vid_pref||'))'.

Εάν δεν πρόκειται για το 1ο vid_pref που εξετάζεται, προστίθεται στην πρόταση μόνο το preference vid_pref εφαρμοζόμενο στον vpreftable με τον *pref_gl10 operator*. δηλαδή:

myquery2:=myquery2||' @@@@@ get_db_preference('||vid_pref||'))'.

Σημειώνεται ότι ο *pref_gl10* operator επιστρέφει μόνο τα affected από το preference που εφαρμόζει tuples και τα affected από προηγούμενα preferences tuples.

Γ.4 Εάν το set vpreftable / prefs είναι το πρώτο που εξετάζεται, "χτίζεται" το τελικό query από το αποτέλεσμα των διαδικασιών "*make_query_plan_B*" που είναι ο returned_table, από τον join_gl2 operator και από την Γ.3.1 πρόταση:

myquery1:= 'SELECT $$'||returned_table||'$$ #### '||myquery2.

Εάν το set vpreftable / prefs δεν είναι το πρώτο που εξετάζεται, το τελικό query συμπληρώνεται , από *τον join_gl2* operator και από την Γ.3.1 πρόταση:

myquery1:= myquery1||' #### '||myquery2;

Σημειώνεται ότι ο join_gl2 operator εκτελεί LEFT OUTER JOIN.

Δ. Το query (*myquery1*) το οποίο "χτίστηκε", εκτελείται και επιστρέφεται το όνομα της relation η οποία δημιουργήθηκε από το τελευταίο *join_gl2* που εκτελέστηκε.
Το συμπληρωματικό Pref SQL query που κατασκευάστηκε, τυπώνεται στο output.

Επεμβάσεις στο τελικό αποτέλεσμα

Ε. Για την ενσωμάτωση των υπολοίπων προτιμήσεων του χρήστη στο ερώτημα, χρειάστηκε να εκτελεστεί join operator μεταξύ του αποτελέσματος της make_query_plan_B και tables της βάσης με ενσωματωμένα preferences. Στο τελικό αποτέλεσμα υπάρχουν λοιπόν απρόσμενα attributes πιθανά άχρηστες πληροφορίες οι οποίες με τις κατάλληλες επεμβάσεις διαγράφονται.

5.2.7 make_query_plan_C_add

Μετατρέπει το SQL ερώτημα σε Pref SQL ερώτημα με σχέδιο εκτέλεσης Plan-C, στη συνέχεια ενσωματώνει όλες τις προτιμήσεις του χρήστη και το δίνει για εκτέλεση.

Είσοδος : Username χρήστη, SQL ερώτημα

Επιστρέφει : Το όνομα της relation η οποία περιέχει τα αποτελέσματα του ερωτήματος.

Διεργασίες make_query_plan_C

Α. Όπως ακριβώς γίνεται στην make_query_plan_C με τις ίδιες διαδικασίες, δημιουργείται η relation pr99_select ή pr999_select με τα αποτελέσματα του ερωτήματος αξιολογημένα από τις καταχωρημένες προτιμήσεις του χρήστη που είναι συναφείς με τα κριτήρια του ερωτήματος.

Εμπλουτισμός του αποτελέσματος με πρόσθετες προτιμήσεις

Β. Όπως ακριβώς γίνεται στην make_query_plan_B_add.

Επεμβάσεις στο τελικό αποτέλεσμα

Γ. Όπως ακριβώς γίνεται στην make_query_plan_B_add.

Παράδειγμα

select make_query_plan_C_add*('maria','SELECT m.movieid, m.title, m.prod_year, g.movieid,g.genre,c.movieid,c.country FROM movies m ,genres g, countries c WHERE m.prod_year>=1980 and c.country="USA" ')*

Κατασκευάζεται το ακόλουθο query (όπως ακριβώς με την make_query_plan_B)

και δημιουργείται η relation: ***pr64_select***

SELECT

(((((((($$Select movieid ,title ,prod_year ,score,conf FROM movies WHERE

prod_year>=1980$$) &^&

($$Select movieid ,genre ,score,conf FROM genres$$) &^&

($$Select movieid ,country ,score,conf FROM countries WHERE country='USA'$$))

***** get_db_preference(9)) **** get_db_preference(47)) **** get_db_preference(6))*

***** get_db_preference(3)) **** get_db_preference(16)) **** get_db_preference(5)*

Στη συνέχεια κατασκευάζεται και εκτελείται το ακόλουθο query και παράγεται το τελικό αποτέλεσμα:

SELECT $$pr64_select$$ #### (($$keywords$$ @@@@@ get_db_preference(26))

@@@@@ get_db_preference(28))

Επιστρέφεται η relation *: j40_yyy*

5.2.8 make_query_plan_B_no_prefs

Βελτιστοποιεί το SQL ερώτημα.

Είσοδος : Username χρήστη, SQL ερώτημα

Επιστρέφει : Το όνομα της relation η οποία περιέχει τα αποτελέσματα του ερωτήματος.

Συντακτική ανάλυση του SQL ερωτήματος και κατακερματισμός.

A. Εντοπίζονται οι θέσεις των keywords *FROM, WHERE, ORDER BY*. Από αυτά μόνο το *FROM* είναι βέβαιο ότι υπάρχει οπωσδήποτε.

Εντοπίζονται οι tables (FROM ...) του ερωτήματος και τα alias names των tables και εισάγονται σε αντίστοιχα arrays.

Εντοπίζονται τα Projection attributes (SELECT ...) και τα alias names των tables στους οποίους ανήκουν τα attributes και εισάγονται και αυτά σε αντίστοιχα arrays.

Εάν υπάρχει condition στο ερώτημα, εντοπίζονται οι επί μέρους συνθήκες (selections), τα attributes των selections και τα alias names των tables στους οποίους ανήκουν τα attributes. Τα selections και τα alias names εισάγονται σε αντίστοιχα arrays.

"Χτίσιμο" του *Optimized SQL query*

Β. Τοποθετείται αρχικά στο προς κατασκευή query το αρχικό *SELECT:*

myquery0:='SELECT ';

Γ. **Για κάθε table του array των tables (*tables[i]* : εξεταζόμενος table)**

Γ.1 Εάν δεν πρόκειται για τον πρώτο εξεταζόμενο table, τοποθετείται ο *join_gl1_exp* (&^&) operator στο προς κατασκευή query*:*

myquery0 :=myquery0||' &^& ';

Γ.2 Εντοπίζονται από το array των Projection attributes με τη βοήθεια του αντίστοιχου array των alias names, τα attributes που ανήκουν στον *tables[i]* και παρατάσσονται στο query μετά από το keyword *Select*.

myquery0:=myquery0||'$$Select '||select_attr[j]||' ';

myquery0:=myquery0||','||select_attr[j]||' ';

Γ.3 Τοποθετείται το keyword *FROM* ακολουθούμενο από τον tables[i].

myquery0:=myquery0||' FROM '||tables[i];

Γ.4 Εντοπίζονται από το array των Selections με την βοήθεια του αντίστοιχου array των alias names, οι συνθήκες για τον tables[i] και τοποθετούνται στο υπό κατασκευή query, πίσω από το keyword *WHERE:*

selection_where:='WHERE ';

selection_where:=selection_where||' and '||cond_sf[j];

Δ. Το query το οποίο "χτίστηκε", εκτελείται και επιστρέφεται το όνομα της relation η οποία δημιουργήθηκε από το τελευταίο *join_gl1_exp* που εκτελέστηκε.
Το Optimized SQL query που κατασκευάστηκε, τυπώνεται στο output.

Παράδειγμα

select make_query_plan_B_no_prefs*('SELECT m.movieid, m.title,m.prod_year, g.movieid,g.genre ,mp.movieid,mp.mpaa FROM movies m ,genres g ,mpaa mp WHERE m.mtype="film" and m.prod_year >=2000 and g.genre like "%Action%" and mp.mpaa="PG"')*

Κατασκευάζεται το query:

SELECT
($$Select movieid ,title ,prod_year FROM movies where mtype='film' and prod_year >=2000$$) &^&
($$Select movieid ,genre FROM genres where genre like '%Action%'$$) &^&
($$Select movieid ,mpaa FROM mpaa where mpaa='PG'$$)

Επιστρέφεται η relation *: j42_yyy*

5.2.9 make_query

Συλλέγει από τους tables preferences και user_preferences τα καταγεγραμμένα preferences του χρήστη και χτίζει και εκτελεί ένα σύνθετο query για την εξαγωγή των προτιμήσεων που θα προταθούν στον χρήστη.

Είσοδος : Username χρήστη

Επιστρέφει : Το όνομα της relation (το οποίο ταυτίζεται με το Username του χρήστη) η οποία περιέχει τα αποτελέσματα του ερωτήματος .

A. Τοποθετείται το αρχικό SELECT για το "χτίσιμο" του query.

myquery0 := 'select ';

B. Ανακτώνται με cursor τα *preftable* και ο αριθμός των preferences για κάθε *preftable* από τους tables *user_preferences*, *preferences* (που αφορούν τον χρήστη) group by *preftable.*

→ *vpreftable* / *prefs_on_same_table*

Γ. **Για κάθε set: *vpreftable / prefs_on_same_table* που εξετάζονται:**

Γ.1 Εάν δεν εξετάζεται το πρώτο *vpreftable*, τοποθετείται ο *join_gl3* operator ο οποίος επιστρέφει μόνο τα tuples που επηρεάζονται από το preference:

myquery0 :=myquery0||' ####@ '

Γ.2 Ανακτώνται με δεύτερο cursor τα καταχωρημένα preferences του χρήστη που έχουν *preftable = vpreftable.*

Γ.3 **Για κάθε preference (vid_pref) με preftable = vpreftable :**

Γ.3.1 Εάν πρόκειται για το 1ο *vid_pref* που εξετάζεται, "χτίζεται" πρόταση που περιλαμβάνει τον *vpreftable.* ακολουθούμενο από το preference *vid_pref* εφαρμοζόμενο στον *vpreftable* με τον *pref_gl10* operator. δηλαδή:

myquery0 := myquery0||'$$'||vpreftable||'$$ @@@@@ get_db_preference('||vid_pref||'))'.

Εάν δεν πρόκειται για το 1ο *vid_pref* που εξετάζεται, προστίθεται στην πρόταση μόνο το preference *vid_pref* εφαρμοζόμενο στον *vpreftable* με τον *pref_gl10* operator. δηλαδή:

myquery0:=myquery0||' @@@@@ get_db_preference('||vid_pref||'))'.

Σημειώνεται ότι ο *pref_gl10* operator επιστρέφει μόνο τα affected από το preference που εφαρμόζει tuples και τα affected από προηγούμενα

preferences tuples.

Δ. Το query (*myquery0)* το οποίο "χτίστηκε", εκτελείται και επιστρέφεται το όνομα της relation η οποία δημιουργήθηκε από το τελευταίο *join_gl3* που εκτελέστηκε.

Ε. Επειδή το αποτέλεσμα της make_query χρησιμοποιείται αποκλειστικά προκειμένου να γίνουν συστάσεις στον χρήστη στο Υποσύστημα Διεπαφής, επιλέγονται τα top 500 tuples με εφαρμογή του κριτηρίου: score*conf για επιστροφή στον χρήστη, στα οποία μάλιστα γίνονται τροποποιήσεις και αναδιατάξεις ώστε να προκύψει μια κατανοητή και πλήρης εικόνα κατάλληλη για recommendations.

Παράδειγμα

select make_query('kostis')

Κατασκευάζεται το query:

SELECT

($$genres$$ @@@@@ get_db_preference(8)) ####@ ($$movies$$ @@@@@ get_db_preference(4)) ####@ ($$keywords$$ @@@@@ get_db_preference(2)) ####@ ($$countries$$ @@@@@ get_db_preference(16)) ####@ ($$movies2directors$$ @@@@@ get_db_preference(42))

Επιστρέφεται η relation *: kostis* (500 tuples)

Η function πρέπει να εκτελείται κάθε φορά που προστίθεται ένα νέο preference του Χρήστη στον table user_preferences. Η Διεπαφή Χρήστη απλά ανακτά το περιεχόμενο της relation με το Username του χρήστη.

5.2.10 Scoring functions

- ***f_2(x: numeric)***

  Επιστρέφει : y = round(x/2013,2)

- ***f_3(x: text)***

  Επιστρέφει : 0,9

- ***f_4(x: text)***

  Επιστρέφει : 0,8

- ***f_6(x: text)***

  Επιστρέφει : 0,6

- ***f_22(x: numeric)***

  Επιστρέφει : y = round(x/max_votes,2); όπου max_votes = 400.000

- ***f_23(x: numeric)***

  Επιστρέφει : y:= round(min_year/x,2); όπου min_year = 1888

- ***f_24(x: numeric)***

  Επιστρέφει : y:= round(x/max_rank,2); όπου max_rank = 10

5.2.11 Βοηθητικές functions

- ***get_db_preference (id_pref : integer)***

  Ανακτά όλα τα στοιχεία του *id_pref* preference από τον table *preferences* και επιστρέφει *preferences_type* αντικείμενο.

- ***union_gl10 (stmt1, stmt2, returned_table_name)***

  Δημιουργεί έναν table με προκαθορισμένο όνομα (input παράμετρος) με το αποτέλεσμα ο οποίος επιστρέφεται. **Καλείται μόνο από την *pref_g10* function – operator** της οποίας αποτελεί μέρος. Επειδή για το όνομα του παραγόμενου table υπεύθυνη είναι η pref_gl10, είναι αναγκαστικό η union_gl10 να παίρνει παραμετρικά το όνομα του επιστρεφόμενου table.

- ***f_avg_s (s1 numeric, c1 numeric, s2 numeric, c2 numeric)***

  Aggregate function, επιστρέφει : (s1 * c1 + s2 * c2) / (c1 + c2)

- ***f_avg_c (s1 numeric, c1 numeric, s2 numeric, c2 numeric)***

  Aggregate function, επιστρέφει : c1 + c2

- ***droptables()***

  Σβήνει τις ενδιάμεσες relations από την βάση.

5.3 Περιγραφή Διεπαφής Χρήστη

Η εφαρμογή αποτελείται από τις εξής JSP σελίδες:

- index.jsp (Home Page, Ερωτήματα χρήστη)
- login.jsp
- masterMenu.jsp (Υλοποιεί το menu, γίνεται include από όλες τις άλλες σελίδες)
- my_preferences.jsp (Profile χρήστη - προτιμήσεις)
- recommend.jsp (Συστάσεις συστήματος προς τον χρήστη)
- moviedetails.jsp
- advanced.jsp (Query editor)

Για να εμφανίζονται σωστά στο χρήστη οι JSP σελίδες, χρησιμοποιούνται stylesheets (CSS) τα οποία καθορίζουν την εμφάνιση και την θέση των αντικειμένων πάνω στις σελίδες.

Για την λειτουργικότητα των σελίδων (στο Client side) τρέχουν τα javascript functions. Κυρίως εξυπηρετούν την προβολή μηνυμάτων λάθους προς τον χρήστη είτε προέρχονται από τον server είτε οφείλονται σε λανθασμένα input του χρήστη.

Τέλος στα JSP τρέχει κώδικας java για να κάνει requests προς τον server καθώς και για να διαχειριστεί τις απαντήσεις από τον server.

5.4 Πλατφόρμες και προγραμματιστικά εργαλεία

Η ανάπτυξη έγινε σε πλατφόρμα Windows 7 v.6.1 64-bit , 8.0 GB Ram και επεξεργαστή Intel Core i7-3610 QM, στα 2.30 GHz.
Η υλοποίηση του συστήματος - στο μεγαλύτερο μέρος - έγινε με την SQL Procedural language PL/pgSQL και ο database server είναι PostgreSQL 9.0.3.

Για την υλοποίηση και τις δοκιμές των PL/pgSQL operators και functions καθώς και για την διεξαγωγή των πειραμάτων, έγινε χρήση του PgAdmin III της PostgreSQL.
Για την ανάπτυξη της διαδικτυακής εφαρμογής χρησιμοποιήθηκε η Java 1.7.0, HTML 5.0, CSS 3.0 και Javascript. Η επικοινωνία με τον server της PostgreSql έγινε με τον postgresql-9.1-902.jdbc driver. Το deployment έγινε με τον Apache Tomcat 7.0.
Ως περιβάλλον ανάπτυξης χρησιμοποιήθηκε το NetBeans IDE 7.1.2.

6 Έλεγχος

Στο παρόν κεφάλαιο παρουσιάζεται ο έλεγχος των βασικών λειτουργιών του συστήματος. Αυτός αφορά τους τελεστές, τα πλάνα εκτέλεσης και το υποσύστημα διεπαφής χρήστη. Όσον αφορά τις διαφορετικές υλοποιήσεις του prefer operator και του optimized query plan, εκτός από τον έλεγχο ορθότητας, διεξάγονται συγκριτικές εκτελέσεις τους προς εξαγωγή συμπερασμάτων.

6.1 PrefSQL τελεστές

6.1.1 Έλεγχος ορθότητας

Παρατίθενται χαρακτηριστικά παραδείγματα εκτέλεσης των *pref_gl1_exp (***), pref_gl2_exp (****), join_gl1_exp(&^&), union_gl1_exp (&&&), intersection_gl1_exp (^^^)* και εξετάζεται η ορθότητά των αποτελεσμάτων τους.

6.1.1.1 Ερώτημα 1

Δίνεται το εξής ερώτημα σε PrefSQL, το οποίο χρησιμοποιεί τον *** τελεστή *(pref_gl1_exp):*

*select ('select movieid, rating, votes, score, conf from ratings where votes>=10000 and rating>=5' *** get_db_preference(17)) *** get_db_preference(46)*

Ζητείται από το σύστημα να εμφανιστούν τα attributes *movieid, rating, votes, score, conf* των ταινιών που έχουν *votes* περισσότερες από 10000 και *rating* μεγαλύτερο του 5. Σε αυτές τις πλειάδες εφαρμόζονται τα *preferences* με id 17 και 46:

	id integer	name character varying(40)	preftable character varying(40)	preffield character varying(40)	condition character varying(140)	functionid integer	confidence real
1	17	rated movies	ratings	rating	rating>8	24	1
2	46	voted movies	ratings	votes	votes>15000	22	1

Εικόνα 19: Επιλεχθείσες προτιμήσεις στο Ερώτημα 1

Το *preference* 17, όπως φαίνεται στην Εικόνα 19, βαθμολογεί τις ταινίες με rating μεγαλύτερο του 8, καλώντας την *f_24*. Η *f_24*, όπως περιγράφεται στο 5.2.10, παίρνει σαν είσοδο το *rating* της κάθε πλειάδας και επιστρέφει *score* με την τιμή του *rating* αναχθείσα από την κλίμακα [0-10] στην κλίμακα [0-1].

Το *preference* 46, από την μεριά του ευνοεί τις ταινίες με *votes* περισσότερες των 15000, τις οποίες αξιολογεί με την f_22. Η f_22 παίρνει σαν παράμετρο την τιμή των *votes* και όσο περισσότερες είναι αυτές, τόσο μεγαλύτερο *score* αποδίδει.

Από την εκτέλεση του Ερωτήματος 1 προκύπτουν 2729 εγγραφές, εκ των οποίων οι 1845 έχουν *conf=1* (δηλαδή ικανοποιούν την μία από τις δύο προτιμήσεις) και οι 279 έχουν *conf=2* (ικανοποιούν και τις δύο). Το ερώτημα εκτελέσθηκε σε 406 ms.

Παρατίθεται ένα μέρος των αποτελεσμάτων του query από τον pgAdmin:

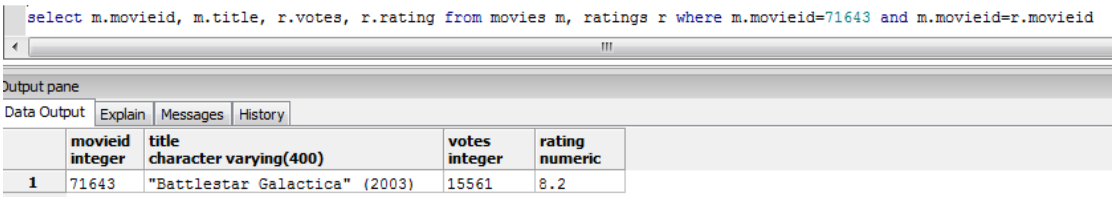
	movieid integer	rating numeric	votes integer	score numeric(3,2)	conf numeric(6,2
268	1047756	8.1	15850	0.43	2.00
269	71643	8.2	15561	0.43	2.00
270	1405716	8.2	16102	0.43	2.00
271	1227403	8.2	16454	0.43	2.00
272	1553646	8.1	18093	0.43	2.00
273	1220458	8.1	18537	0.43	2.00
274	1201769	8.1	18578	0.43	2.00
275	1441394	8.1	21807	0.43	2.00
276	1243637	8.1	21957	0.43	2.00
277	1444987	8.2	16240	0.43	2.00
278	1000635	8.1	16960	0.43	2.00
279	954863	8.1	19932	0.43	2.00
280	563378	9.8	12192	0.98	1.00
281	543958	9.3	11578	0.93	1.00
282	786651	9.3	14275	0.93	1.00
283	183838	9.2	12388	0.92	1.00
284	61751	9.2	11291	0.92	1.00
285	295255	9.0	13308	0.90	1.00
286	453382	8.9	12396	0.89	1.00
287	207995	8.8	12354	0.88	1.00
288	977165	8.8	11382	0.88	1.00
289	668196	8.8	12221	0.88	1.00
290	580896	8.8	12685	0.88	1.00
291	875339	8.7	11801	0.87	1.00
292	668801	8.6	13575	0.86	1.00
293	45764	8.5	12796	0.85	1.00
294	519452	8.5	14594	0.85	1.00
295	274489	8.5	12470	0.85	1.00
296	458764	8.4	13813	0.84	1.00
297	93627	8.4	14953	0.84	1.00
298	1194198	8.3	12949	0.83	1.00

Εικόνα 20: Αποτελέσματα Ερωτήματος 1

Προς επαλήθευση των ζητούμενων, όλες ο επιστραφείσες εγγραφές ικανοποιούν την συνθήκη του Select statement (*votes>=10000 and rating>=5)*. Επίσης, παρατηρούμε ότι όλες οι ταινίες με *conf=2* -ανεξάρτητα από το κατά πόσον έχουν υψηλό score ή όχι- πληρούν τα κριτήρια και των δύο προτιμήσεων, ενώ αυτές με *conf=1* πληρούν μόνο της μιας εκ των δύο.

Η συνολική διαδικασία εκτέλεσης είναι η ακόλουθη: Πρώτα εκτελείται η *pref_gl1_exp* με παραμέτρους το πρώτο Select statement και το *preference 17*. Η *pref_gl1_exp* δημιουργεί τον table *pr01_select* όπου αρχικά εκχωρεί το αποτέλεσμα του Select. Όσες εγγραφές πληρούν τις συνθήκη του *preference* τις ενημερώνει με τα διαμορφωμένα *score, conf*. Κατόπιν, εκτελείται η επόμενη κλήση της *pref_gl1_exp* παίρνοντας παραμετρικά τον *pr01_select* και το *preference 46*. Όσες εγγραφές πληρούν την συνθήκη του, τις ενημερώνει με τις καινούργιες τιμές των *score, conf*. Το αποτέλεσμα του Ερωτήματος 1, εντέλει, βρίσκεται στον table *pr01_select*.

Επιλέγουμε μία τυχαία πλειάδα για να εξετάσουμε την ορθότητα των *score, conf* που διαμορφώθηκαν. Ας επιλέξουμε την ταινία με *movieid = 71643*.



Εικόνα 21: movieid = 71643

Η συγκεκριμένη ταινία έχει *votes=15561(>15000) και rating=8.2(>8).* Επομένως, ορθώς έχει conf=2 εφόσον πληροί τις απαιτήσεις και των δύο προτιμήσεων. Εάν εκτελέσουμε την συνάρτηση αξιολόγησης f_24 για την συγκεκριμένη τιμή του rating έχουμε:

$$f\_24(8.2) = \frac{8.2}{10} = 0.82$$

Η f_22 για τις δεδομένες votes επιστρέφει:

$$f\_22(15561) = \frac{15561}{400000} \approx 0.04$$

Τέλος, συναθροίζουμε τις δύο αυτές τιμές με την f_avg_s χρησιμοποιώντας για βάρη τα conf τους:

$$f\_avg\_s(0.82,1,0.04,1) = \frac{0.82*1+0.04*1}{1+1} = 0.43$$

Όπως φαίνεται και στην Εικόνα 20, η ίδια τιμή έχει προκύψει και από το σύστημα.

6.1.1.2 Ερώτημα 2

Εκτελούμε το ίδιο ερώτημα με την διαφορά ότι αντί για την χρήση του *** τελεστή, επιλέγουμε τον **** (*pref_gl2_exp*):

*select ('select movieid, rating, votes, score, conf from ratings where votes>=10000 and rating>=5' **** get_db_preference(17)) **** get_db_preference(46)*

Τα ζητούμενα είναι, προφανώς, τα ίδια, όπως και τα δύο χρησιμοποιούμενα preferences. Η εκτέλεση φέρνει τα ίδια ακριβώς αποτελέσματα με πριν, σε χρόνο 265 ms.

Η διαδικασία εκτέλεσης είναι η εξής: Πρώτα εκτελείται η *pref_gl2_exp* με παραμέτρους το Select statement και το *preference 17*. Η *pref_gl2_exp* δημιουργεί τον βοηθητικό πίνακα *temp_sel* όπου εκχωρεί το αποτέλεσμα του Select statement. Έπειτα δημιουργεί τον table *pr02_select* (ο pr01_select υπάρχει από την προηγούμενη εκτέλεση καθώς δεν έτρεξε η *droptables()*) στον οποίο καταχωρεί όλες τις εγγραφές του temp_sel(διαμορφωμένες κατά *preference 17* και μη). Διαγράφεται *ο temp_sel*. Κατόπιν, εκτελείται η επόμενη κλήση της *pref_gl2_exp,* παίρνοντας παραμετρικά τον *pr02_select* και το *preference 46*. Αυτή, με την σειρά της, δημιουργεί νέο table εν ονόματι *pr03_select* όπου εκχωρεί όλες τις εγγραφές του *pr02_select* (διαμορφωμένες κατά *preference 46* και μη). Το αποτέλεσμα του Ερωτήματος 2 βρίσκεται στον *pr03_select.*

6.1.1.3 Ερώτημα 3

Εφόσον δείξαμε ότι και οι δύο prefer τελεστές λειτουργούν σωστά, πάμε να εξετάσουμε τον &^& (*join_gl1_exp*), δίνοντας το ακόλουθο ερώτημα:

*select ('select movieid, title, prod_year from movies where prod_year>1980' **** get_db_preference(4))*

*&^& ('select movieid,genre from genres where genre!=$$Thriller$$' **** get_db_preference(3))*

*&^& ('select movieid,keyword from keywords where keyword like $$%family%$$ or keyword like $$%love%$$' **** get_db_preference(28))*

Σε αυτό το ερώτημα επιθυμούμε να συνδυάσουμε τρεις προτιμήσεις, η κάθε μία από τις οποίες αφορά select statement διαφορετικού table.

Επιλέγονται οι ταινίες που έχουν χρονολογία παραγωγής μετά το 1980. Σε αυτές εφαρμόζεται το *preference 4*, το οποίο πριμοδοτεί όσες είναι από το 2000 και μετά. Έπειτα επιλέγονται όλα τα είδη ταινιών εκτός από τα θρίλερ. Από αυτά βαθμολογούνται οι κωμωδίες με την *preference 3*. Τέλος, επιλέγονται οι ταινίες που έχουν στις λέξεις κλειδιά τους την λέξη "family" ή την λέξη "love" και πριμοδοτούνται από την *preference 28 αυτά με "family"*. Για να βρεθούν τα κοινά tuples η join_gl1_exp εκτελείται δύο φορές.

	id integer	name character varying	preftable character varyi	preffield character varyi	condition character varying(140)	functionid integer	confidence real
1	3	Comedies	genres	genre	genre LIKE '%Comedy%'	3	1
2	4	Recent movies	movies	prod_year	prod_year >= 2000	2	1
3	28	family	keywords	keyword	keyword LIKE '%family%'	6	1

Εικόνα 22: Επιλεχθείσες προτιμήσεις στο Ερώτημα 3

Οι συναρτήσεις αξιολόγησης *f_3, f_6* που καλούνται από τις *preferences 3, 28* δεν παίρνουν παράμετρο. Απλώς επιστρέφουν τιμή 0.9, 0.6 αντίστοιχα. Η *f_2* της *preference 2,* από την άλλη, επιστρέφει την τιμή του έτους παραγωγής διαιρεμένη με την maximum που θεωρείται το 2013.

Από την εκτέλεση του Ερωτήματος 3 προκύπτουν 46334 εγγραφές, εκ των οποίων 1416 έχουν conf=3, 12495 έχουν conf=2 και 23344 έχουν conf=1. Χρειάστηκαν 31771ms για να εκτελεστεί το ερώτημα. Παρατίθεται μέρος των αποτελεσμάτων από τον pgAdmin:

	movieid integer	title character varying(400)	prod_year integer	genre character v	keyword character varying(60)	score numeric	conf numeric
1399	941640	A Thin Line Between Coke an	2002	Comedy	family-quarrel	0.830	3.00
1400	1022130	Chatter (2002)	2002	Comedy	family-gathering	0.830	3.00
1401	969970	Arthur's Perfect Christmas	2000	Comedy	family-relationships	0.830	3.00
1402	970187	Aruku, hito (2001)	2001	Comedy	family-reunion	0.830	3.00
1403	970187	Aruku, hito (2001)	2001	Comedy	family-relationships	0.830	3.00
1404	1022676	Chekhovskie motivy (2002)	2002	Comedy	family-relationships	0.830	3.00
1405	987794	Bend It Like Beckham (2002)	2002	Comedy	family-traditions	0.830	3.00
1406	1023017	Cherish (2002)	2002	Comedy	kids-and-family	0.830	3.00
1407	1036087	Corky Romano (2001)	2001	Comedy	family-portrait	0.830	3.00
1408	987794	Bend It Like Beckham (2002)	2002	Comedy	family-values	0.830	3.00
1409	1023806	Chicken Run (2000)	2000	Comedy	kids-and-family	0.830	3.00
1410	1023807	Chicken Run (2000) (VG)	2000	Comedy	kids-and-family	0.830	3.00
1411	973357	Atlético San Pancho (2001)	2001	Comedy	kids-and-family	0.830	3.00
1412	1025007	Chiquititas: Rincón de luz	2001	Comedy	kids-and-family	0.830	3.00
1413	1064153	Dibu 3 (2002)	2002	Comedy	kids-and-family	0.830	3.00
1414	987794	Bend It Like Beckham (2002)	2002	Comedy	family-disapproval	0.830	3.00
1415	974882	Austin Powers in Goldmember	2002	Comedy	lost-family	0.830	3.00
1416	985055	Beautiful (2000)	2000	Comedy	family-secret	0.830	3.00
1417	1429161	Sylvia's Baklava (2006)	2006	Comedy	love	0.950	2.00
1418	1027035	Ciao tesoro (2008)	2008	Comedy	love	0.950	2.00
1419	1317039	Okul (2004)	2004	Comedy	love	0.950	2.00
1420	947373	Adam (2009/I)	2009	Comedy	character-says-i-love-you	0.950	2.00
1421	1394775	Shake, Rattle & Roll 9 (200	2007	Comedy	unrequited-love	0.950	2.00
1422	947360	Adam & Steve (2005)	2005	Comedy	love	0.950	2.00
1423	1394775	Shake, Rattle & Roll 9 (200	2007	Comedy	love-charm	0.950	2.00
1424	988537	Bernard and Doris (2006)	2006	Comedy	love	0.950	2.00
1425	1324850	Oté (2008)	2008	Comedy	love	0.950	2.00
1426	1027896	Circumstances (2008)	2008	Comedy	love	0.950	2.00
1427	1403697	Sitzriesen an Stehimbissen	2005	Comedy	first-love	0.950	2.00
1428	948358	Adventureland (2009)	2009	Comedy	romantic-love	0.950	2.00
1429	1394902	Shall We Dance (2004)	2004	Comedy	love	0.950	2.00

Εικόνα 23: Αποτελέσματα Ερωτήματος 3

Προς επαλήθευση των ζητούμενων, όλες ο επιστραφείσες εγγραφές ικανοποιούν τις συνθήκες των Select statements (*prod_year>1980, genre!='Thriller, keyword like '%family%' or keyword like '%love%'*). Επίσης, παρατηρούμε ότι όλες οι ταινίες με *conf=3* πληρούν τα κριτήρια και των τριών προτιμήσεων, αυτές με *conf=2* πληρούν δύο από τις τρεις και αυτές με *conf=1* πληρούν μία.

Η συνολική διαδικασία εκτέλεσης είναι η ακόλουθη: Εκτελείται η *pref_gl2_exp* με παραμέτρους το πρώτο Select statement και το *preference 4*. Αυτό εφαρμόζεται στις επιλεχθείσες πλειάδες με τον ίδιο τρόπο που περιγράφηκε, για να παραχθεί ο table *pr04_select*. Επιπλέον, εκτελείται *η pref_gl2_exp* με παραμέτρους το δεύτερο Select statement και το *preference 3*, για να παραχθεί ο *pr05_select*. Έπειτα, τρέχει *η join_gl1_exp* με παραμέτρους τους δημιουργηθείσες πίνακες *pr04_select* και *pr05_select*. Η *join_gl1_exp* βρίσκει το join attribute που είναι το movieid, φτιάχνει τον *j01_yyy* όπου εκχωρεί το αποτέλεσμα του inner join των δύο πινάκων συναθροίζοντας τα score, conf τους. Εν συνεχεία, καλείται η *pref_gl2_exp* με παραμέτρους το τρίτο Select statement και το *preference 20*, και δημιουργείται ο *pr06_select*. Αυτός, μαζί με τον *j01_yyy* αποτελούν τους operands της δεύτερης κλίσης της *join_gl1_exp*. Η *join_gl1_exp* κάνει inner join, συναθροίζει και επιστρέφει τα τελικά αποτελέσματα στον table *j02_yyy*.

Για να επαληθεύσουμε την ορθότητα των διαδικασιών, θα εξετάσουμε μία τυχαία πλειάδα: Αυτή με *movieid = 987794*:

```
select m.movieid, m.title, m.prod_year, g.genre, k.keyword from movies m, genres g, keywords k
where m.movieid=987794 and m.movieid=g.movieid and m.movieid=k.movieid
```

Output pane

Data Output | Explain | Messages | History

	movieid integer	title character varying(400)	prod_year integer	genre character varying(50)	keyword character varying(60)
94	987794	Bend It Like Beckham (2002)	2002	Sport	goalkeeper
95	987794	Bend It Like Beckham (2002)	2002	Romance	goalkeeper
96	987794	Bend It Like Beckham (2002)	2002	Drama	goalkeeper
97	987794	Bend It Like Beckham (2002)	2002	Comedy	women's-rights
98	987794	Bend It Like Beckham (2002)	2002	Sport	women's-rights
99	987794	Bend It Like Beckham (2002)	2002	Romance	women's-rights
100	987794	Bend It Like Beckham (2002)	2002	Drama	women's-rights
101	987794	Bend It Like Beckham (2002)	2002	Comedy	european-asian
102	987794	Bend It Like Beckham (2002)	2002	Sport	european-asian
103	987794	Bend It Like Beckham (2002)	2002	Romance	european-asian
104	987794	Bend It Like Beckham (2002)	2002	Drama	european-asian
105	987794	Bend It Like Beckham (2002)	2002	Comedy	family-relationships
106	987794	Bend It Like Beckham (2002)	2002	Sport	family-relationships
107	987794	Bend It Like Beckham (2002)	2002	Romance	family-relationships
108	987794	Bend It Like Beckham (2002)	2002	Drama	family-relationships
109	987794	Bend It Like Beckham (2002)	2002	Comedy	kicking

Εικόνα 24: movieid = 987794

Από το αποτέλεσμα του (SQL) join των τριών πινάκων, προκύπτουν 620 εγγραφές. Αυτό οφείλεται στο ότι ο η σχέση του movies με τον genres και τον keywords είναι 1:N (άρα ο genres με τον keywords είναι N:N). Μία ταινία έχει παραπάνω από ένα είδος και επιπλέον έχει πολλές λέξεις-κλειδιά.

Για να επαληθεύσουμε το αποτέλεσμα της ταινίας "Bend It Like Beckham" που εμφανίζεται στην σειρά 1405 της Εικόνας 23, θα επεξεργαστούμε τα -αντίστοιχα- στοιχεία της σειράς 105 της Εικόνας 24. Η συγκεκριμένη εγγραφή έχει *prod_year=2002 (>=2000), genre='Comedy'*

και keyword='family-relationships' (like '%family%'). Ικανοποιεί, λοιπόν, τις συνθήκες και των τριών preferences, οπότε δικαίως της έχει αποδοθεί *conf=3*.

Η συνάρτηση αξιολόγησης f_2 της προσδίδει τιμή:

$$f\_2(2002) = \frac{2002}{2012} \approx 1.00$$

Από την μεριά της, η f_3 την βαθμολογεί με f_3() = 0.90

Συναθροίζουμε τις δύο αυτές τιμές με βάρη τα conf:

$$f\_avg\_s(1.00,1,0.90,1) = \frac{1.00*1+0.90*1}{1+1} = 0.95$$

το conf αυτής της νέας εκτίμησης είναι προφανώς: $f\_avg\_c(1.00,1,0.90,1) = 1+1 = 2$

Η f_6 επιστρέφει βαθμό f_6() = 0.60, ο οποίος συναθροίζεται με τον προηγούμενο:

$$f\_avg\_s(0.95,2,0.60,1) = \frac{0.95*2+0.6*1}{2+1} = 0.83$$

Η τιμή είναι η ίδια με αυτή που υπολογίστηκε.

6.1.1.4 Ερώτημα 4

Αποδειχθείσας της σωστής λειτουργίας των prefer και join τελεστών, εξετάζουμε τον *&&& (union_gl1_exp)*. Δίνεται το εξής ερώτημα:

select

*('select c.movieid, c.country, r.rating from countries c, ratings r where c.country="USA" and c.movieid=r.movieid' *** get_db_preference(17))*

&&&

*('select c.movieid, c.country, r.rating from countries c, ratings r where c.country="UK" and c.movieid=r.movieid' *** get_db_preference(17))*

Ζητούνται οι ταινίες που είναι αμερικάνικης ή βρετανικής παραγωγής (ή και τα δύο). Σε αυτές εφαρμόζεται το *preference 17* το οποίο πριμοδοτεί όσες έχουν *rating* μεγαλύτερο του 8. Η *union_gl1_exp* καλείται μία φορά για να συνενώσει τα επιμέρους αποτελέσματα.

	id integer	name character varying(40)	preftable character varying(40)	preffield character varying(40)	condition character varying(140)	functionid integer	confidence real
1	17	rated movies	ratings	rating	rating>8	24	1

Εικόνα 25: Επιλεχθείσα προτίμηση στο Ερώτημα 4

Το *preference 17* καλεί την συνάρτηση αξιολόγησης *f_24*.

Από την εκτέλεση του Ερωτήματος 4 προκύπτουν 152764 εγγραφές, 290 εκ των οποίων έχουν conf=2 και 23170 έχουν conf=1. Χρειάστηκαν 5321ms για να εκτελεστεί. Παρατίθεται μέρος των αποτελεσμάτων από τον pgAdmin:

	movieid integer	country character	rating numeric	score numeric	conf numeric
279	703689	USA	8.1	0.810	2.00
280	1449961	USA	8.1	0.810	2.00
281	707683	USA	8.1	0.810	2.00
282	1458298	USA	8.1	0.810	2.00
283	707872	USA	8.1	0.810	2.00
284	1476437	USA	8.1	0.810	2.00
285	985521	USA	8.1	0.810	2.00
286	1495985	USA	8.1	0.810	2.00
287	988069	USA	8.1	0.810	2.00
288	1497209	USA	8.1	0.810	2.00
289	1515193	USA	8.1	0.810	2.00
290	1523592	USA	8.1	0.810	2.00
291	304281	USA	10.0	1.000	1.00
292	1007853	USA	10.0	1.000	1.00
293	1004140	USA	10.0	1.000	1.00
294	1079394	USA	10.0	1.000	1.00
295	292578	USA	10.0	1.000	1.00
296	1061762	USA	10.0	1.000	1.00
297	307660	USA	10.0	1.000	1.00
298	1531382	USA	10.0	1.000	1.00
299	358863	USA	10.0	1.000	1.00
300	1076772	USA	10.0	1.000	1.00
301	367195	USA	10.0	1.000	1.00
302	1073080	USA	10.0	1.000	1.00
303	417024	USA	10.0	1.000	1.00
304	1064247	USA	10.0	1.000	1.00
305	420385	USA	10.0	1.000	1.00
306	1096089	USA	10.0	1.000	1.00
307	240180	USA	10.0	1.000	1.00
308	1076079	USA	10.0	1.000	1.00

Εικόνα 26: Αποτελέσματα Ερωτήματος 4

Προς επαλήθευση των συνθηκών στα δύο Select statements, όλες οι εγγραφές έχουν *country = USA ή UK*. Παρατηρούμε ότι σε όλα τα tuples με *conf=2*, η αναγραφόμενη χώρα είναι η USA, ενώ θα περιμέναμε να δούμε USA και UK, καθότι αυτές πληρούν και τις δύο συνθήκες. Το γεγονός αυτό δεν οφείλεται σε κάποιο λάθος, αλλά στην λειτουργία της *union_gl1_exp* η οποία συνενώνει τις εγγραφές των δύο παραμέτρων της. Για κάθε ζεύγος ίδιων πλειάδων με ίδιο *movieid*, η *union_gl1_exp* επιστρέφει τις τιμές των attributes του πρώτου operand και τις aggregated τιμές των *score, conf*. Εξάλλου παρόμοια είναι και η λειτουργία της κανονικής SQL union συνάρτησης (την οποία η *union_gl1_exp* προσομοιώνει) με την διαφορά ότι εκείνη για να θεωρήσει δύο πλειάδες ίδιες, απαιτεί να είναι ίδιες οι τιμές όλων των attributes.

Η ακολουθούμενη διαδικασία είναι η εξής: Εφαρμόζεται το *preference 17* στις ταινίες που επιλέγονται από το πρώτο Select statement και δημιουργείται ο πίνακας *pr07_select*. Ομοίως διαμορφώνεται ο *pr08_select*. Εκτελείται ο *union_gl1_exp* operator (&&&) με παραμέτρους αυτούς τους δύο πίνακες. Κάνει aggregate τις όποιες κοινές εγγραφές αυτών και επιστρέφει το αποτέλεσμα μέσω του *u01_yyy*.

Επιλέγουμε μία τυχαία εγγραφή, για την οποία θα εκτελέσουμε βήμα-βήμα τις διαδικασίες αξιολόγησής της. Η εγγραφή αυτή έχει *movieid=707683*.

```
select m.movieid, c.country, r.rating from movies m, countries c, ratings r
where m.movieid=707683 and m.movieid=c.movieid and m.movieid=r.movieid
```

Output pane

Data Output | Explain | Messages | History

	movieid integer	country character varying(	rating numeric
1	707683	UK	8.1
2	707683	Germany	8.1
3	707683	USA	8.1

Εικόνα 27: movieid = 707683

Η συγκεκριμένη ταινία έχει τρεις εγγραφές στον πίνακα countries, καθότι είναι συμπαραγωγή Μ.Βρετανίας, Γερμανίας και Αμερικής. Οι εγγραφές, βέβαια, που επελέγησαν από το Ερώτημα 4 είναι η πρώτη και η τρίτη.

Η καθεμία από αυτές τις δύο εγγραφές βαθμολογείται από την *f_24* ως εξής:

$$f\_24(8.1) = \frac{8.1}{10} = 0.81$$

Η συνάθροιση μέσω της *f_avg_*s απλώς επιστρέφει την ίδια τιμή. Πράγματι, η ταινία με *movieid=707683* έχει βαθμολογηθεί, όπως βλέπουμε στην Εικόνα 26, με 0,81.

6.1.1.5 Ερώτημα 5

Εξετάζουμε την ορθότητα του ^^^ τελεστή (intersection_gl1_exp). Εκτελούμε την ίδια ερώτηση με πριν, με την διαφορά ότι εναλλάσσουμε τον *union operator* με τον *intersect operator:*

select

*('select c.movieid, c.country, r.rating from countries c, ratings r where c.country="USA" and c.movieid=r.movieid' *** get_db_preference(17))*

^^^

*('select c.movieid, c.country, r.rating from countries c, ratings r where c.country="UK" and c.movieid=r.movieid' *** get_db_preference(17))*

Τα Select statements και το preference είναι τα ίδια με το Ερώτημα 4. Το προκύπτον αποτέλεσμα είναι υποσύνολο του αποτελέσματος του Ερωτήματος 4, καθότι περιλαμβάνει μόνο τις κοινές εγγραφές των δύο Select statements. Επιστρέφονται 2121 εγγραφές σε 2902 ms. Από αυτές, 290 έχουν *conf=2* (όπως και πριν), ενώ δεν υπάρχει καμία με *conf=1* (γεγονός που οφείλεται στην εφαρμογή του ίδιου preference).

Η αξιολόγηση των αποτελεσμάτων γίνεται με τον ίδιο τρόπο με το Ερώτημα 4.

6.1.2 Σύγκριση pref_gl1_exp, pref_gl2_exp

Όπως αναφέρθηκε στο 3, ο *Pref_gl2_exp* operator έχει μικρότερο χρόνο εκτέλεσης από τον *Pref_gl1_exp* όταν το ποσοστό των tuples μιας relation που επηρεάζονται από το preference είναι μεγάλο. Αντίθετα στις περιπτώσεις όπου επηρεάζεται από το preference ένα μικρό ποσοστό των tuples της relation, ο *Pref_gl1_exp* κρίνεται καταλληλότερος ειδικά όταν το μέγεθος της relation είναι μεγάλο.

Ο *Pref_gl1_exp* όταν το preference που διαχειρίζεται είναι το πρώτο μιας ακολουθίας από preferences σε μια relation, κάνει *create* έναν table αντίγραφο της input relation και στη συνέχεια ενημερώνει (update) τα attributes *score, conf* των tuples αυτού του table που εμπίπτουν στο condition του preference. Οι επόμενοι *Pref_gl1_exp operators,* απλώς ενημερώνουν συναθροιστικά τον ίδιο αυτόν table. Ο χρόνος που απαιτείται για το αρχικό *create table* (ακόμα και όταν το μέγεθος της input relation είναι μεγάλο) , είναι μικρός αφού γίνεται μόνο αντιγραφή της input relation, σε αντίθεση με τα updates που εκτελούνται μετά, από τον αρχικό *Pref_gl1_exp* και από τους επόμενους τα οποία απαιτούν μεγαλύτερο χρόνο. Αν λοιπόν οι tuples που επηρεάζονται από τα preferences είναι σχετικά λίγα, ο *Pref_gl1_exp* αποδεικνύεται πιο αποδοτικός.

Ο *Pref_gl2_exp* κάνει πάντα τις ίδιες ενέργειες. Κάνει create έναν table από την input relation διαμορφώνοντας κατευθείαν τις τιμές των attributes *score , conf* στις "affected" tuples. Ο χρόνος που απαιτείται για αυτό το *create table* είναι πολύ μεγαλύτερος από το *create table* του *Pref_gl1_exp,* αλλά στο σύνολο αποδεικνύεται αποδοτικότερος όταν μεγάλο ποσοστό των tuples της input relation εμπίπτει στο condition του preference.

Ακολουθούν ενδεικτικές μετρήσεις:

1. Relation *movies* (1.573.104 rows) και ένα preference:

   Pref_gl1_exp

   *select 'movies' *** get_db_preference(4)* → 23,9 sec

   - ✓ Χρόνος : 2,85 sec : create table + 21,05 sec : update
   - ✓ Affected tuples: 463.617

Pref_gl2_exp

*select 'movies' **** get_db_preference(4)* → 14,71 sec

- ✓ Χρόνος : 14,71 sec : create table
- ✓ Affected tuples: 463.617

Σημειώνεται ότι όταν η input relation στους *Pref_gl1_exp , Pref_gl2_exp* προκύπτει από Select Statement το οποίο εκτελείται στην αρχή, οι χρόνοι εκτέλεσης και των δύο operators αυξάνονται:

*select 'select * from movies' *** get_db_preference(4)* → *29,8 sec*

*select 'select * from movies' **** get_db_preference(4)* →*19,2 sec*

2. Relation *movies* (1.573.104 rows) και δύο preferences:

Pref_gl1_exp

*select ('movies' *** get_db_preference(4)) *** get_db_preference(9)* → 37,30 sec

1ο preference:

- ✓ Χρόνος : 2,9 sec : create table + 22,37 sec : update
- ✓ Affected tuples : 463.617

2ο preference:

- ✓ Χρόνος : 12,03 sec : update
- ✓ Affected tuples : 200.540

Pref_gl2_exp

*select ('movies' **** get_db_preference(4)) *** get_db_preference(9)* → 23,84 sec

1ο preference:

- ✓ Χρόνος : 14,79 sec : create table
- ✓ Affected tuples : 463.617

2ο preference:

- ✓ Χρόνος : 9,05 sec : create table
- ✓ Affected tuples : 200.540

3. Relation *movies* (1.573.104 rows) και τρία preferences:

Pref_gl1_exp

*select (('movies' *** get_db_preference(4)) *** get_db_preference(9)) *** get_db_preference(7)* → 52,53 sec

1ο preference :

- ✓ Χρόνος : 2,82 sec : create table + 21,17 sec update.
- ✓ Affected tuples : 463.617

2ο preference :

- ✓ Χρόνος : 11,79 sec : update
- ✓ Affected tuples : 200.540

3ο preference :

- ✓ Χρόνος : 16,75 sec : update
- ✓ Affected tuples : 319.135

Pref_gl2_exp

*select (('movies' **** get_db_preference(4)) *** get_db_preference(9)) *** get_db_preference(7)* → 34,41 sec

1ο preference :

- ✓ Χρόνος : 14,30 sec : create table
- ✓ Affected tuples : 463.617 tuples

2ο preference :

- ✓ Χρόνος : 9,34 sec : create table
- ✓ Affected tuples : 200.540

3ο preference :

- ✓ Χρόνος : 10,77 sec : create table
- ✓ Affected tuples : 319.135

4. Relation movies2actors (13.145.520 tuples) και ένα preference :

Pref_gl1_exp

*select 'movies2actors' *** get_db_preference(41)* → 18,05 sec

- ✓ Χρόνος : 16,35 sec : create table + 1,70 sec : update table
- ✓ Affected tuples : 241

Pref_gl2_exp

*select 'movies2actors' **** get_db_preference(41)* → 43,98 sec

- ✓ Χρόνος : 43,98 sec : create table
- ✓ Affected tuples : 241

5. Relation movies2actors (13.145.520 tuples) και δύο preferences :

Pref_gl1_exp

*select ('movies2actors' *** get_db_preference(41)) *** get_db_preference(30)* → 18,36 sec

1ο preference:

- ✓ Χρόνος : 14,99 sec : create table + 1,69 sec : update
- ✓ Affected tuples : 241

2ο preference:

- ✓ Χρόνος : 1,68 sec : update
- ✓ Affected tuples : 387

Pref_gl2_exp

*select ('movies2actors' **** get_db_preference(41)) *** get_db_preference(30)* → 83,07 sec

1ο preference:

- ✓ Χρόνος : 45,91 sec : create table
- ✓ Affected tuples : 241

2ο preference:

- ✓ Χρόνος : 37,16 sec : create table
- ✓ Affected tuples : 387

6. Relation movies2actors (13.145.520 tuples) και τέσσερα preferences :

Pref_gl1_exp

*select ((('movies2actors' *** get_db_preference(41)) *** get_db_preference(30)) *** get_db_preference(14)) *** get_db_preference(35)* → 22,20 sec

1ο preference :

- ✓ Χρόνος : 15,44 sec : create table + 1,68 sec : update
- ✓ Affected tuples : 241

2ο preference :

- ✓ Χρόνος : 1,72 sec : update
- ✓ Affected tuples : 387

3ο preference :

- ✓ Χρόνος : 1,67 sec : update
- ✓ Affected tuples : 271

4ο preference :

- ✓ Χρόνος : 1,69 sec : update
- ✓ Affected tuples : 281

Pref_gl2_exp

*select (('movies2actors' **** get_db_preference(..)) *** get_db_preference(..)) *** get_db_preference(..)* → 152,40 sec

1ο preference :

- ✓ Χρόνος : 46,03 sec : create table
- ✓ Affected tuples : 241

2ο preference :

- ✓ Χρόνος : 34,50 sec : create table
- ✓ Affected tuples : 387

3ο preference :

- ✓ Χρόνος : 35,27 sec : create table
- ✓ Affected tuples : 271

4ο preference :

- ✓ Χρόνος : 6,60 sec : create table
- ✓ Affected tuples : 281

Οι μετρήσεις αποδεικνύουν ότι ο *Pref_gl2_exp* operator είναι απαγορευτικός για εφαρμογή πολύ εξειδικευμένων preferences (π.χ. ο ηθοποιός τάδε) που έχουν απήχηση σε μικρό ποσοστό tuples της input relation, ειδικά δε όταν εφαρμόζονται πολλά preferences στην relation.

6.2 Πλάνα εκτέλεσης PrefSQL ερωτημάτων

Από τα προηγούμενα κεφάλαια προκύπτει ότι τα επικρατέστερα πλάνα εκτέλεσης preference-aware ερωτημάτων είναι τα Plan-B, Plan-C (Εικόνες 9, 10 αντίστοιχα). Τα κριτήρια τα οποία καθορίζουν ποιό είναι το βέλτιστο εξ΄αυτών σε κάθε περίπτωση, δεν είναι ιδιαιτέρα ευδιάκριτα. Στο παρόν κεφάλαιο εκτελούμε ερωτήματα ακολουθώντας αυτά τα δύο πλάνα, συγκρίνουμε τον χρόνο εκτέλεσης και εξάγουμε συμπεράσματα.

6.2.1 Ερωτήματα με σταθερές συνθήκες επιλογής (SELECT)

Εκτελούμε κάποια ερωτήματα με τα Plan-B, Plan-C κρατώντας σταθερές τις συνθήκες επιλογής τους (Select Statements). Ο παράγοντας που αλλάζουμε είναι οι προτιμήσεις που εφαρμόζονται. Σε κάθε εκτέλεση αυξάνουμε τον αριθμό των εφαρμοζόμενων προτιμήσεων.

6.2.1.1 Ερώτημα Q1 (2 σχέσεις)

Q1: SELECT m.movieid,m.title,m.prod_year,g.movieid,g.genre FROM movies m,genres g WHERE m.prod_year>=1980 and m.movieid=g.movieid

Ο πίνακας genres έχει 999.616 εγγραφές, ενώ το επιλεχθέν υποσύνολο του movies έχει 782.752 εγγραφές..

1η εκτέλεση: Η μία σχέση με 1 preference

Εφαρμόζουμε το preference 3 (genre LIKE ‘%Comedy%’) στο genres.

Η Plan-B εκτέλεση είναι η εξής:

*SELECT (‘SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980’) &^& ((‘SELECT movieid,genre,score,conf FROM genres’) **** get_db_preference(3))*

Χρειάστηκαν 20.218 msec. Το αποτέλεσμα έχει 625.761 εγγραφές, εκ των οποίων οι 69.105 έχουν διαμορφωμένα score, conf (11%).

Η Plan-C εκτέλεση είναι η εξής:

*SELECT ((‘SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980’) &^& (‘SELECT movieid,genre,score,conf FROM genres’)) **** get_db_preference(3)*

Χρειάστηκαν 16.786 msec. Το αποτέλεσμα, προφανώς, είναι το ίδιο.

2η εκτέλεση: Κάθε σχέση από 1 preference

Εφαρμόζουμε το preference 4 (prod_year>=2000) στο υποσύνολο του movies και το preference 8 (genre LIKE '%Action%') στο genres.

Η Plan-B εκτέλεση είναι η εξής:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') **** get_db_preference(4)) &^& (('SELECT movieid,genre,score,conf FROM genres') **** get_db_preference(8))*

Χρειάστηκαν 26.582 msec. Το αποτέλεσμα έχει 625.761 εγγραφές, εκ των οποίων διαμορφωμένα score, conf έχουν οι 424.926 (68%).

Η Plan-C εκτέλεση είναι η εξής:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') &^& ('SELECT movieid,genre,score,conf FROM genres')) **** get_db_preference(4) **** get_db_preference(8)*

Χρειάστηκαν 25.006 msec. Το αποτέλεσμα, προφανώς, είναι το ίδιο.

3η εκτέλεση: Κάθε σχέση από 2 preferences

Εφαρμόζουμε τα preferences 3, 6 (genre like '%Musical%') στο genres και τα preferences 9 (prod_year<1960), 47 (prod_year>=2010) στο υποσύνολο movies.

Η Plan-B εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') **** get_db_preference(9) **** get_db_preference(47)) &^& (('SELECT movieid,genre,score,conf FROM genres') **** get_db_preference(3) **** get_db_preference(6))*

Χρόνος 24.835 msec. Παράχθηκαν 625.761 εγγραφές, εκ των οποίων βαθμολογημένες είναι οι 94.035 (15%).

Plan-C εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') &^& ('SELECT movieid,genre,score,conf FROM genres')) **** get_db_preference(9) **** get_db_preference(47) **** get_db_preference(6) **** get_db_preference(3)*

Χρόνος 24.523 msec.

4η εκτέλεση: Κάθε σχέση από 3 preferences

Εφαρμόζουμε τα preferences 8 (genre LIKE '%Action%'), 48 (genre LIKE '%Documentary%'), 49 (genre LIKE '%Thriller%') στον genres και τα 4, 7 (prod_year>=1980), 47 στο υποσύνολο movies.

Plan-B εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') **** get_db_preference(4) **** get_db_preference(7) **** get_db_preference(47)) &^& (('SELECT movieid,genre,score,conf FROM genres') **** get_db_preference(8) **** get_db_preference(48) **** get_db_preference(49))*

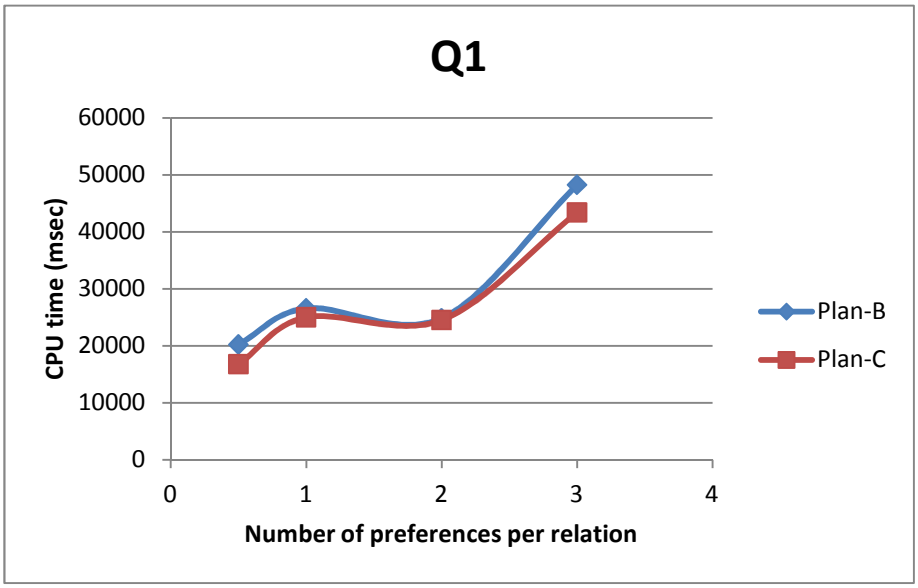
Χρόνος 48.251 msec. Προέκυψαν 625.761 εγγραφές, εκ των οποίων βαθμολογημένες όλες (100%).

Plan-C εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') &^& ('SELECT movieid,genre,score,conf FROM genres')) **** get_db_preference(8) **** get_db_preference(48) **** get_db_preference(49) **** get_db_preference(4) **** get_db_preference(7) **** get_db_preference(47)*

Χρόνος εκτέλεσης 43.400 msec.

Από τα παραπάνω, σχεδιάζουμε το διάγραμμα για το ερώτημα Q1 με οριζόντιο άξονα-x τις εφαρμοζόμενες προτιμήσεις ανά σχέση και κατακόρυφο άξονα-y τον χρόνο εκτέλεσης:



Εικόνα 28: Σύγκριση Plan-B, Plan-C για το Q1

Εκ του αποτελέσματος, για το ερώτημα Q1 το Plan-C φαίνεται να είναι αποδοτικότερο έναντι του Plan-B.

6.2.1.2 Ερώτημα Q2 (3 σχέσεις)

Q2: SELECT m.movieid,m.title,m.prod_year,g.movieid,g.genre,c.movieid,c.country FROM movies m,genres g,countries c WHERE m.prod_year>=1980 and c.country='USA' and m.movieid=g.movieid and m.movieid=c.movieid

Ο genres έχει 999.616 εγγραφές, το υποσύνολο του movies έχει 782.752 και το υποσύνολο του countries έχει 326.992 .

1η εκτέλεση: Οι δύο σχέσεις από 1 preference

Εφαρμόζουμε το preference 3 στον genres και το preference 12 (country='USA') στο υποσύνολο του countries.

Plan-B εκτέλεση:

*SELECT ('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') &^& (('SELECT movieid,genre,score,conf FROM genres') **** get_db_preference(3)) &^& (('SELECT movieid,country,score,conf FROM countries WHERE country=''USA''') **** get_db_preference(12))*

Χρόνος 29.296 msec. Προέκυψαν 283.770 εγραφές και βαθμολογήθηκαν όλες (100%).

Plan-C εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') &^& ('SELECT movieid,genre,score,conf FROM genres') &^& ('SELECT movieid,country,score,conf FROM countries WHERE country=''USA''')) **** get_db_preference(3) **** get_db_preference(12)*

Χρόνος 25.771 msec.

2η εκτέλεση: Κάθε σχέση από 1 preference

Εφαρμόζουμε το preference 8 στο genres, το 4 στο υποσύνολο των movies και το 16 (country='UK') στο υποσύνολο του countries.

Plan-B εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') **** get_db_preference(4)) &^& (('SELECT movieid,genre,score,conf FROM genres') **** get_db_preference(8)) &^& (('SELECT movieid,country,score,conf FROM countries WHERE country=''USA''') **** get_db_preference(16))*

Χρόνος 34.695 msec. Προέκυψαν 283.770 εγγραφές, εκ των οποίων οι 211.101 αξιολογημένες (74%).

Plan-C εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') &^& ('SELECT movieid,genre,score,conf FROM genres') &^& ('SELECT movieid,country,score,conf FROM countries WHERE country=''USA''')) **** get_db_preference(8) **** get_db_preference(4) **** get_db_preference(16)*

Χρόνος 26.271 msec.

3η εκτέλεση: Κάθε σχέση από 2 preferences

Εφαρμόζουμε τα preferences 3, 6 στο genres, τα 9, 47 στο υποσύνολο movies και τα 5 (country='Italy'), 16 στο υποσύνολο countries.

Plan-B εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') **** get_db_preference(9) *** get_db_preference(47)) &^& (('SELECT movieid,genre,score,conf FROM genres') **** get_db_preference(3) **** get_db_preference(6)) &^& (('SELECT movieid,country,score,conf FROM countries WHERE country=''USA''') **** get_db_preference(5) **** get_db_preference(16))*

Χρόνος εκτέλεσης 32.183 msec. Προέκυψαν 283.770 εγγραφές, από τις οποίες οι 44.191 είναι βαθμολογημένες (16%).

Plan-C εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') &^& ('SELECT movieid,genre,score,conf FROM genres') &^& ('SELECT movieid,country,score,conf FROM countries WHERE country=''USA''')) **** get_db_preference(3) **** get_db_preference(6) **** get_db_preference(9) **** get_db_preference(47) **** get_db_preference(5) **** get_db_preference(16)*

Χρόνος εκτέλεσης 25.881 msec.

4η εκτέλεση: Κάθε σχέση από 3 preferences

Εφαρμόζουμε τα preferences 8, 48, 49 στον genres, τα preferences 4, 7, 47 στο υποσύνολο του movies, τα preferences 12, 16, 45 (country='France') στο υποσύνολο των countries.

Plan-B εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980') **** get_db_preference(4) **** get_db_preference(7) **** get_db_preference(47)) &^& (('SELECT movieid,genre,score,conf FROM genres') **** get_db_preference(8) **** get_db_preference(48) **** get_db_preference(49)) &^& (('SELECT movieid,country,score,conf FROM countries WHERE country=''USA''') **** get_db_preference(12) **** get_db_preference(16) **** get_db_preference(45))*

Χρόνος 62.775 msec. Το αποτέλεσμα έχει 283.770 εγγραφές, εκ των οποίων οι βαθμολογημένες είναι όλες (100%).

Plan-C εκτέλεση:

*SELECT ((‘SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980’) &^& (‘SELECT movieid,genre,score,conf FROM genres’) &^& (‘SELECT movieid,country,score,conf FROM countries WHERE country=’’USA’’’)) **** get_db_preference(8) **** get_db_preference(48) **** get_db_preference(49) **** get_db_preference(4) **** get_db_preference(7) **** get_db_preference(47) **** get_db_preference(12) **** get_db_preference(16) **** get_db_preference(45)*

Χρόνος 40.295 msec.

5η εκτέλεση: Κάθε σχέση από 4 preferences

Εφαρμόζουμε τα preferences 3, 8, 48, 49 στον genres, τα 4, 7, 9, 47 στο υποσύνολο του movies και τα 5, 12, 16, 45 στο υποσύνολο του countries.

Plan-B εκτέλεση:

*SELECT ((‘SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980’) **** get_db_preference(5) **** get_db_preference(12) **** get_db_preference(16) **** get_db_preference(45)) &^& ((‘SELECT movieid,genre,score,conf FROM genres’) **** get)db_preference(3) **** get_db_preference(8) **** get_db_preference(48) **** get_db_preference(49)) &^& ((‘SELECT movieid,country,score,conf FROM countries WHERE country=’’USA’’’) **** get_db_preference(5) **** get_db_preference(12) **** get_db_preference(16) **** get_db_preference(45))*

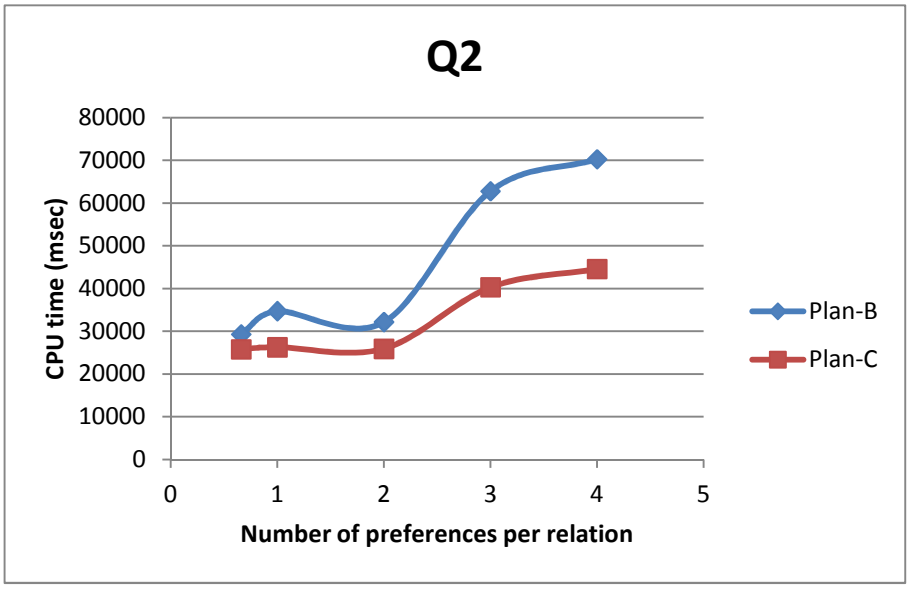
Χρόνος 70.247 msec. Προκύπτουν 283.770 εγγραφές, όλες με διαμορφωμένα score, conf.

Plan-C εκτέλεση:

*SELECT ((‘SELECT movieid,title,prod_year,score,conf FROM movies WHERE prod_year>=1980’) &^& (‘SELECT movieid,genre,score,conf FROM genres’) &^& (‘SELECT movieid,country,score,conf FROM countries WHERE country=’’USA’’’)) **** get_db_preference(3) **** get_db_preference(8) **** get_db_preference(48) **** get_db_preference(49) **** get_db_preference(4) **** get_db_preference(7) **** get_db_preference(9) **** get_db_preference(47) **** get_db_preference(5) **** get_db_preference(12) **** get_db_preference(16) **** get_db_preference(45)*

Χρόνος 44.569 msec.

Από αυτές τις μετρήσεις προκύπτει το διάγραμμα του Q2 ερωτήματος:



Εικόνα 29: Σύγκριση Plan-B, Plan-C για το Q2

Το Plan-C αναδεικνύεται πάλι αποδοτικότερο. Μάλιστα, ο χρόνος που εξοικονομεί έναντι του Plan-B είναι σημαντικής τάξης μεγέθους.

6.2.1.3 Ερώτημα Q3 (4 σχέσεις)

Q3: SELECT m.movieid, m.title, m.prod_year, g.movieid, g.genre, c.movieid, c.country, k.movieid, k.keyword FROM movies m, genres g, countries c, keywords k WHERE m.prod_year>=1980 and c.country='USA'

Το υποσύνολο του movies έχει 782.752 εγγραφές, αυτό του countries έχει 326.992, ο πίνακας genres έχει 999.616, ενώ ο keywords έχει 2.883.220 εγγραφές.

<u>1η εκτέλεση: Οι τρεις σχέσεις από 1 preference</u>

Εφαρμόζουμε το preference 3 στον genres, το 12 στο υποσύνολο του countries και το 38 (keyword LIKE '%love'%') στον keywords.

Plan-B εκτέλεση:

*SELECT ('SELECT movieid, title, prod_year, score, conf FROM movies WHERE prod_year>=1980') &^& (('SELECT movieid, genre, score, conf FROM genres') **** get_db_preference(3)) &^& (('SELECT movieid, country, score, conf FROM countries WHERE country=''USA''') **** get_db_preference(12)) &^& (('SELECT movieid, keyword, score, conf FROM keywords') **** get_db_preference(38))*

Χρόνος εκτέλεσης 75.816 msec. Το αποτέλεσμα έχει 2.194.686 εγγραφές, όλες βαθμολογημένες (100%).

Plan-C εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year, score, conf FROM movies WHERE prod_year>=1980') &^& ('SELECT movieid, genre, score, conf FROM genres') &^& ('SELECT movieid, country, score, conf FROM countries WHERE country=''USA''') &^& ('SELECT movieid, keyword, score, conf FROM keywords')) **** get_db_preference(3) **** get_db_preference(12) **** get_db_preference(38)*

Χρόνος εκτέλεσης 155.298 msec.

2η εκτέλεση: Κάθε σχέση από 1 preference

Εφαρμόζουμε το preference 8 στον genres, το 16 στο υποσύνολο του countries, το 4 στο υποσύνολο του movies και το 2 (keyword LIKE '%police%') στον keywords.

Plan-B εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year, score, conf FROM movies WHERE prod_year>=1980') **** get_db_preference(4)) &^& (('SELECT movieid, genre, score, conf FROM genres') **** get_db_preference(8)) &^& (('SELECT movieid, country, score, conf FROM countries WHERE country=''USA''') **** get_db_preference(16)) &^& (('SELECT movieid, keyword, score, conf FROM keywords') **** get_db_preference(2))*

Χρειάστηκε χρόνος 76.877 msec. Το αποτέλεσμα αποτελείται από 2.194.686 εγγραφές, από τις οποίες έχουν βαθμολογηθεί οι 1.368.079 (62%).

Plan-C εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year, score, conf FROM movies WHERE prod_year>=1980') &^& ('SELECT movieid, genre, score, conf FROM genres') &^& ('SELECT movieid, country, score, conf FROM countries WHERE country=''USA''') &^& ('SELECT movieid, keyword, score, conf FROM keywords')) **** get_db_preference(2) **** get_db_preference(4) **** get_db_preference(8) **** get_db_preference(16)*

Χρειάστηκαν 159.869 msec.

3η εκτέλεση: Κάθε σχέση από 2 preferences

Εφαρμόζουμε τα preferences 3, 6 στον genres, τα 5, 16 στο υποσύνολο countries, τα 9, 47 στο υποσύνολο movies και τα 26 (keyword LIKE '%sex%'), 28 (keyword LIKE '%family%') στον keywords.

Plan-B εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year, score, conf FROM movies WHERE prod_year>=1980') **** get_db_preference(9) **** get_db_preference(47)) &^&*

*((‘SELECT movieid, genre, score, conf FROM genres’) **** get_db_preference(3) **** get_db_preference(6)) &^& ((‘SELECT movieid, country, score, conf FROM countries WHERE country=’’USA’’’) **** get_db_preference(5) **** get_db_preference(16)) &^& ((‘SELECT movieid, keyword, score, conf FROM keywords’) **** get_db_preference(26) **** get_db_preference(28))*

Χρόνος 84.147 msec. Επιστράφηκαν 2.194.686 εγγραφές, εκ των οποίων οι 373.231 (17%) με διαμορφωμένα score, conf.

Plan-C εκτέλεση:

*SELECT ((‘SELECT movieid, title, prod_year, score, conf FROM movies WHERE prod_year>=1980’) &^& (‘SELECT movieid, genre, score, conf FROM genres’) &^& (‘SELECT movieid, country, score, conf FROM countries WHERE country=’’USA’’’) &^& (‘SELECT movieid, keyword, score, conf FROM keywords’)) **** get_db_preference(3) **** get_db_preference(6) **** get_db_preference(5) **** get_db_preference(16) **** get_db_preference(9) **** get_db_preference(47)*

Χρόνος 225.249 msec.

<u>4η εκτέλεση: Κάθε σχέση από 3 preferences</u>

Εφαρμόζουμε τα preferences 8, 48, 49 στον genres, τα 12, 16, 45 στο υποσύνολο countries, τα 2, 27(keyword LIKE ‘%hardcore%’), 37(keyword LIKE ‘%murder%’) στον keywords και τα 4, 7, 47 στο υποσύνολο του movies.

Plan-B εκτέλεση:

*SELECT ((‘SELECT movieid, title, prod_year, score, conf FROM movies WHERE prod_year>=1980’) **** get_db_preference(4) **** get_db_preference(7) **** get_db_preference(47)) &^& ((‘SELECT movieid, genre, score, conf FROM genres’) **** get_db_preference(8) **** get_db_preference(48) **** get_db_preference(49)) &^& ((‘SELECT movieid, country, score, conf FROM countries WHERE country=’’USA’’’) **** get_db_preference(12) **** get_db_preference(16) **** get_db_preference(45)) &^& ((‘SELECT movieid, keyword, score, conf FROM keywords’) **** get_db_preference(2) **** get_db_preference(27) **** get_db_preference(37))*

Χρόνος 137.359 msec. Επιστρέφονται 2.194.686 εγγραφές, όλες με διαμορφωμένα score, conf.

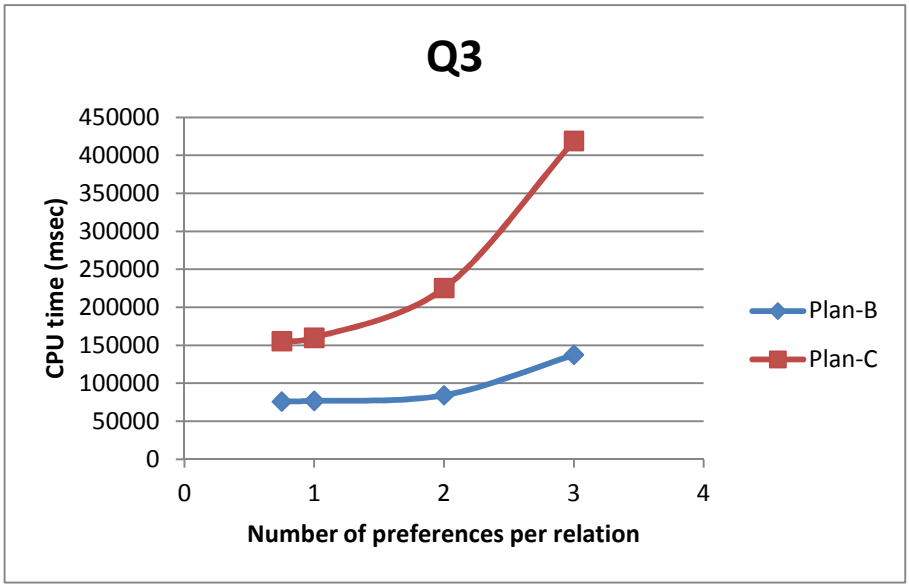
Plan-C εκτέλεση:

*SELECT ((‘SELECT movieid, title, prod_year, score, conf FROM movies WHERE prod_year>=1980’) &^& (‘SELECT movieid, genre, score, conf FROM genres’) &^& (‘SELECT movieid, country, score, conf FROM countries WHERE country=’’USA’’’) &^& (‘SELECT movieid, keyword, score, conf FROM keywords’)) **** get_db_preference(8)*

***** get_db_preference(48) **** get_db_preference(49) **** get_db_preference(12) **** get_db_preference(16) **** get_db_preference(45) **** get_db_preference(2) **** get_db_preference(27) **** get_db_preference(37) **** get_db_preference(4) **** get_db_preference(7) **** get_db_preference(47)*

Χρόνος 418.767 msec.

Ακολουθεί το διάγραμμα για το Q3:



Εικόνα 30: Σύγκριση Plan-B, Plan-C για το Q3

Σε αντίθεση με τα Q1, Q2, στο Q3 καταλληλότερο πλάνο εκτέλεσης –με χαρακτηριστική διαφορά- κρίνεται το Plan-B.

6.2.1.4 Ερώτημα Q4 (2 σχέσεις)

Q4: SELECT m.movieid,m.title,m.prod_year,g.movieid,g.genre FROM movies m,genres g WHERE m.movieid=g.movieid

Το ερώτημα Q4 είναι σαν το Q1, με την διαφορά ότι αντλεί πληροφορίες από ολόκληρο τον πίνακα movies και όχι από υποσύνολό του. Ο πίνακας movies έχει 1.573.104 εγγραφές, ενώ ο genres 999.616 .

<u>1η εκτέλεση: Η μία σχέση με 1 preference</u>

Εφαρμόζουμε το preference 3 στον genres.

Plan-B εκτέλεση:

*SELECT ('SELECT movieid,title,prod_year,score,conf FROM movies') &^& (('SELECT movieid,genre,score,conf FROM genres') **** get_db_preference(3))*

Χρόνος 39.094 msec. Το αποτέλεσμα αποτελείται από 997.050 εγγραφές, οι 126.878 (13%) εκ των οποίων είναι αξιολογημένες.

Plan-C εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies') &^& ('SELECT movieid,genre,score,conf FROM genres')) **** get_db_preference(3)*

Χρόνος 29.562 msec.

2η εκτέλεση: Κάθε σχέση από 1 preference

Εφαρμόζουμε το preference 8 στον genres και το preference 4 στον movies.

Plan-B εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies') **** get_db_preference(4)) &^& (('SELECT movieid,genre,score,conf FROM genres') **** get_db_preference(8))*

Χρόνος εκτέλεσης 42.089 msec. Επιστρέφονται 997.050 εγγραφές, εκ των οποίων οι 432.609 (43%) είναι βαθμολογημένες.

Plan-C εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies') &^& ('SELECT movieid,genre,score,conf FROM genres')) **** get_db_preference(4) **** get_db_preference(8)*

Χρόνος 36.770 msec.

3η εκτέλεση: Κάθε σχέση από 2 preferences

Εφαρμόζουμε τα preferences 3, 6 στον πίνακα genres και τα 9, 47 στον movies.

Plan-B εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies') **** get_db_preference(9) **** get_db_preference(47)) &^& (('SELECT movieid,genre,score,conf FROM genres') **** get_db_preference(3) **** get_db_preference(6))*

Απαιτείται χρόνος 42.759 msec. Επιστρέφονται 997.050 εγγραφές, εκ των οποίων οι 359.818 (36%) με βαθμολογημένα score, conf.

Plan-C εκτέλεση:

*SELECT (('SELECT movieid,title,prod_year,score,conf FROM movies') &^& ('SELECT movieid,genre,score,conf FROM genres')) **** get_db_preference(4) **** get_db_preference(6) **** get_db_preference(9) **** get_db_preference(47)*

Χρόνος 42.198 msec.

4η εκτέλεση: Κάθε σχέση από 3 preferences

Εφαρμόζονται τα preferences 8, 48, 49 στον genres και τα 4, 7, 47 στον movies.

Plan-B εκτέλεση:

*SELECT ((‘SELECT movieid,title,prod_year,score,conf FROM movies’) **** get_db_preference(4) **** get_db_preference(7) **** get_db_preference(47)) &^& ((‘SELECT movieid,genre,score,conf FROM genres’) **** get_db_preference(8) **** get_db_preference(48) **** get_db_preference(49))*

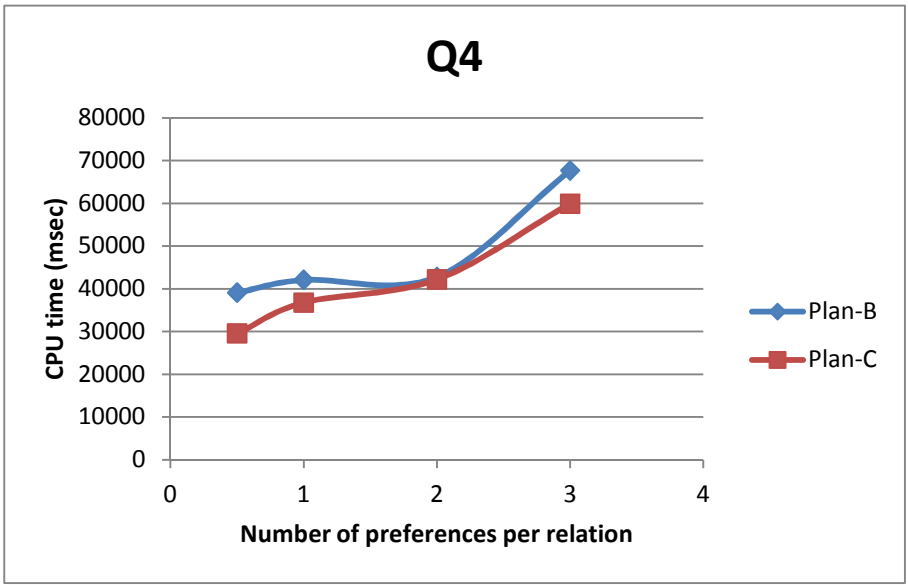
Χρόνος εκτέλεσης 67.673 msec. Επιστρέφονται 997.050 εγγραφές, οι 669.926 (67%) από τις οποίες είναι αξιολογημένες.

Plan-C εκτέλεση:

*SELECT ((‘SELECT movieid,title,prod_year,score,conf FROM movies’) &^& (‘SELECT movieid,genre,score,conf FROM genres’)) **** get_db_preference(4) **** get_db_preference(6) **** get_db_preference(9) **** get_db_preference(47)*

Χρόνος 59.889 msec.

Προκύπτει το εξής διάγραμμα για το ερώτημα Q4:



Εικόνα 31: Συγκριση Plan-B, Plan-C για το Q4

Στο ερώτημα Q4 παρατηρείται ανάλογη συμπεριφορά των πλάνων εκτέλεσης με τα Q1, Q2, Q3, δηλαδή το Plan-C αναδεικνύεται καταλληλότερο.

6.2.1.5 Ερώτημα Q5 (3 σχέσεις)

Q5: SELECT m.movieid, m.title, m.prod_year, g.movieid, g.genre, c.movieid ,c.country FROM movies m, genres g, countries

Ο πίνακας movies έχει 1.573.104 εγγραφές, ο genres 999.616 και ο countries 823.505 .

1η εκτέλεση: Οι δύο σχέσεις από 1 preference

Εφαρμόζουμε το preference 3 στον genres και το preference 12 στον countries.

Plan-B εκτέλεση:

*SELECT ('SELECT movieid, title, prod_year FROM movies') &^& (('SELECT movieid, genre FROM genres') **** get_db_preference(3)) &^& (('SELECT movieid, country FROM countries') **** get_db_preference(12))*

Χρόνος 51.573 msec. Επιστρέφονται 1.047.147 εγγραφές, εκ των οποίων οι 541.897 (52%) βαθμολογημένες.

Plan-C εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year FROM movies') &^& ('SELECT movieid, genre FROM genres') &^& ('SELECT movieid, country FROM countries')) **** get_db_preference(3) **** get_db_preference(12)*

Χρόνος εκτέλεσης 58.703 msec.

2η εκτέλεση: Κάθε σχέση από 1 preference

Εφαρμόζουμε το preference 8 στον genres, το 16 στον countries, το 4 στον movies.

Plan-B εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year FROM movies') **** get_db_preference(4)) &^& (('SELECT movieid, genre FROM genres') **** get_db_preference(8)) &^& (('SELECT movieid, country FROM countries') **** get_db_preference(16))*

Χρόνος 56.659 msec. Επιστρέφονται 1.047.147 εγγραφές, εκ των οποίων οι 498.039 βαθμολογημένες.

Plan-C εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year FROM movies') &^& ('SELECT movieid, genre FROM genres') &^& ('SELECT movieid, country FROM countries')) **** get_db_preference(4) **** get_db_preference(8) **** get_db_preference(16)*

Χρόνος 57.831 msec.

3η εκτέλεση: Κάθε σχέση από 2 preferences

Εφαρμόζονται τα preferences 3, 6 στον genres, τα 5, 16 στον countries και τα 9, 47 στον movies.

Plan-B εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year FROM movies') **** get_db_preference(9) **** get_db_preference(47)) &^& (('SELECT movieid, genre FROM genres') ****

*get_db_preference(3) **** get_db_preference(6)) &^& (('SELECT movieid, country FROM countries') **** get_db_preference(5) **** get_db_preference(16))*

Χρόνος 63.976 msec. Το αποτέλεσμα αποτελείται από 1.047.147 εγγραφές, 430.243 (41%) εκ των οποίων αξιολογημένες.

Plan-C εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year FROM movies') &^& ('SELECT movieid, genre FROM genres') &^& ('SELECT movieid, country FROM countries')) **** get_db_preference(3) **** get_db_preference(6) **** get_db_preference(5) **** get_db_preference(16) **** get_db_preference(9) **** get_db_preference(47)*

Χρόνος εκτέλεσης 71.862 msec.

4η εκτέλεση: Κάθε σχέση από 3 preferences

Εφαρμόζονται τα preferences 8, 48, 49 στον genres, τα 12, 16, 45 στον countries και τα 4, 7, 47 στον movies.

Plan-B εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year FROM movies') **** get_db_preference(4) **** get_db_preference(7) **** get_db_preference(47)) &^& (('SELECT movieid, genre FROM genres') **** get_db_preference(8) **** get_db_preference(48) **** get_db_preference(49)) &^& (('SELECT movieid, country FROM countries') **** get_db_preference(12) **** get_db_preference(16) **** get_db_preference(45))*

Χρόνος 91.448 msec. Επιστρέφονται 1.047.147 εγγραφές, από τις οποίες οι 928.388 (89%) είναι βαθμολογημένες.

Plan-C εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year FROM movies') &^& ('SELECT movieid, genre FROM genres') &^& ('SELECT movieid, country FROM countries')) **** get_db_preference(8) **** get_db_preference(48) **** get_db_preference(49) **** get_db_preference(12) **** get_db_preference(16) **** get_db_preference(45) **** get_db_preference(4) **** get_db_preference(7) **** get_db_preference(47)*

Χρόνος 101.352 msec.

5η εκτέλεση: Κάθε σχέση από 4 preferences

Εφαρμόζονται τα preferences 3, 8, 48, 49 στον genres, τα preferences 5, 12, 16, 45 στον countries και τα preferences 4, 7, 9, 47 στον movies.

Plan-B εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year FROM movies') **** get_db_preference(4) **** get_db_preference(7) **** get_db_preference(9) **** get_db_preference(47)) &^& (('SELECT movieid, genre FROM genres') **** get_db_preference(3) **** get_db_preference(8) **** get_db_preference(48) **** get_db_preference(49)) &^& (('SELECT movieid, country FROM countries') **** get_db_preference(5) **** get_db_preference(12) **** get_db_preference(16) **** get_db_preference(45))*

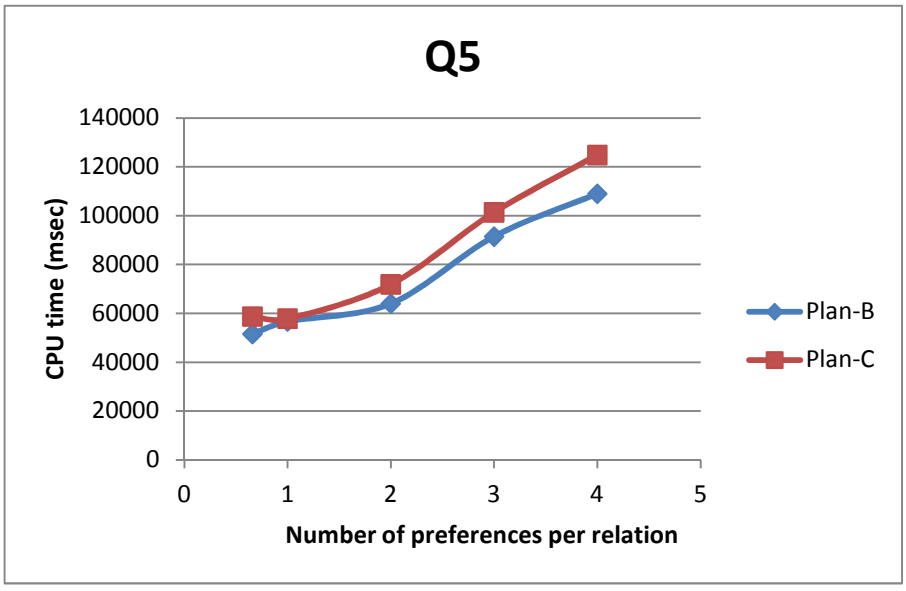
Χρόνος 108.997 msec. Επιστρέφονται 1.047.147 εγγραφές, εκ των οποίων οι 989.755 (95%) έχουν διαμορφωμένα score, conf.

Plan-C εκτέλεση:

*SELECT (('SELECT movieid, title, prod_year FROM movies') &^& ('SELECT movieid, genre FROM genres') &^& ('SELECT movieid, country FROM countries')) **** get_db_preference(3) **** get_db_preference(8) **** get_db_preference(48) **** get_db_preference(49) **** get_db_preference(5) **** get_db_preference(12) **** get_db_preference(16) **** get_db_preference(45) **** get_db_preference(4) **** get_db_preference(7) **** get_db_preference(9) **** get_db_preference(47)*

Χρόνος 124.832 msec.

Από αυτές τις μετρήσεις, παράγεται το εξής διάγραμμα για το ερώτημα Q5:



Εικόνα 32: Σύγκριση των Plan-B, Plan-C για το Q5

Διαπιστώνουμε ότι στο Q5 το βέλτιστο πλάνο εκτέλεσης είναι το Plan-B όπως και στο ερώτημα Q3, με την διαφορά ότι εδώ υπερέχει σχεδόν οριακά.

Από τις παραπάνω μετρήσεις εξάγεται το εξής ασφαλές (και λογικό) συμπέρασμα: Για ένα δεδομένο SQL query όσο και αν αυξομειώσουμε τα preferences που εφαρμόζονται σε αυτό –

ώστε να το μετατρέψουμε σε PrefSQL query - , η σχετική αποτελεσματικότητα των Plan-B, Plan-C δεν αλλάζει.

Στον πίνακα της Εικόνας 33 συνοψίζουμε τα συνολικά μεγέθη των σχέσεων και των αποτελεσμάτων των ερωτημάτων Q1 έως Q5 :

	Q1	Q2	Q3	Q4	Q5
Number of Relations:	2	3	4	2	3
Input rows:	782752, 999616	999616, 782752, 326992	782752, 326992, 999616, 2883220	999616, 1573104	1573104, 999616, 823505
Total Output rows:	625761	283770	2194686	997050	1047147

Εικόνα 33: Συνολικές εγγραφές ερωτημάτων Q1 έως Q5

Ανεξαρτήτως του πλήθους και της φύσης των εφαρμοζόμενων προτιμήσεων, στα Q1, Q2, Q4 βέλτιστο πλάνο εκτέλεσης αναδεικνύεται πάντοτε το Plan-C, ενώ στα Q3, Q5 το Plan-B. Για την ουσιαστικότερη μελέτη των παραγόντων που καθορίζουν το αποδοτικότερο πλάνο εκτέλεσης, ακολουθούν οι μετρήσεις της υποενότητας 6.2.2 .

6.2.2 Ερωτήματα με σταθερές προτιμήσεις

Η διαφορά μεταξύ των Plan-B και Plan-C έγκειται στο πότε επιδρούν οι προτιμήσεις στις σχέσεις με την εκτέλεση των pref τελεστών. Στο Plan-B (Εικόνα 9) αυτό γίνεται αμέσως μετά την εκτέλεση του τελεστή επιλογής (Select), ενώ στο Plan-C (Εικόνα 10) αναβάλλεται για μετά τον τελεστή συνάθροισης (Join). Για κάθε query, βέλτιστο αναδεικνύεται το πλάνο κατά το οποίο η εκτέλεση των pref γίνεται για τις λιγότερες εγγραφές. Για παράδειγμα, στο ερώτημα Q1, ο συνολικός αριθμός εγγραφών για τις οποίες εκτελείται η pref_gl2_exp, είναι μεγαλύτερος εάν κληθεί πριν το Join απ' ότι εάν κληθεί μετά, γι αυτό και η Plan-C, εν προκειμένω, είναι αποδοτικότερη.

Το κόστος εκτέλεσης K των pref τελεστών σε μία σχέση είναι ανάλογο του αριθμού εκτέλεσής του n_i σε αυτή (δηλαδή του αριθμού των προτιμήσεων που την αφορούν) επί και του πλήθους των εγγραφών της r_i: $K = n_i * r_i$

Προκειμένου να συγκρίνουμε το κόστος εκτέλεσης των προτιμήσεων πριν και μετά το Join, εισάγουμε το αδιάστατο μέγεθος «ΛΟΓΟΣ», το οποίο είναι απλά ο λόγος του K της κλήσης των pref πριν το Join (κατά Plan-B) προς το K της κλήσης των pref μετά το Join (κατά Plan-C): $\text{ΛΟΓΟΣ} = \frac{\sum (n_i * r_i)}{\sum (n_i) * r_{join-product}}$,όπου n_i το πλήθος των preferences που εφαρμόζεται στην σχέση R_i ,

r_i το πλήθος των εγγραφών της R_i και

$r_{join\text{-}product}$ το πλήθος των εγγραφών του join product

Στα πειράματα που ακολουθούν, μελετάται η σχέση που διέπει τον ΛΟΓΟ με την επίδοση των Plan-B, Plan-C.

Εν αντιθέσει με το πλήθος των εγγραφών των join operands, το πλήθος των εγγραφών του join product, προφανώς, δεν είναι άμεσα ελέγξιμο. Εξαρτάται από τις εισερχόμενες στο join εγγραφές και την συσχέτισή τους. Ως εκ τούτου, ο ΛΟΓΟΣ δεν είναι άμεσα ελέγξιμο μέγεθος. Στο καθένα από τα επόμενα ερωτήματα, με δεδομένα τα εφαρμοζόμενα preferences στις σχέσεις του, διαφοροποιούμε τις συνθήκες των select statements έτσι ώστε να αλλάζουμε άμεσα το πλήθος και την συσχέτιση των join operands εγγραφών και να διαμορφώνουμε έμμεσα την τιμή του ΛΟΓΟΥ.

Δεδομένου ότι ο ΛΟΓΟΣ είναι ένα συγκριτικό, αδιάστατο μέγεθος το οποίο εκφράζει μία αναλογία καταστάσεων η οποία αντιπροσωπεύει άπειρο δυνατό συνδυασμό πραγματικών καταστάσεων, δεν θα ήταν ορθό να σχεδιαστεί διάγραμμα του πραγματικού χρόνου εκτέλεσης συναρτήσει του ΛΟΓΟΥ. Αντί αυτού, στον άξονα y παρουσιάζεται ο αντίστοιχος λόγος των πραγματικών χρόνων εκτέλεσης t_B/t_C. Εξάλλου, το αντικείμενο ενδιαφέροντός μας είναι καθαρά το πότε και σε ποιο βαθμό το ένα πλάνο εκτέλεσης είναι αποδοτικότερο έναντι του άλλου.

Όλες οι μετρήσεις των PrefSQL queries στην παρούσα υποενότητα βασίζονται στο εξής SQL query:

Q: SELECT m.movieid, m.title, m.prod_year, k.movieid, k.keyword FROM movies m, keywords k

6.2.2.1 2 σχέσεις – από 1 preference

Τα preferences που εφαρμόζονται στο Q είναι το 4 (prod_year>=2000) και το 2 (keyword LIKE '%police%').

Τα εφαρμόζουμε κατά Plan-B, Plan-C. Οι αντίστοιχοι χρόνοι είναι 66.362 msec και 94.380 msec, οπότε t_B/t_C=0,703 . Οι movies εγγραφές είναι στο σύνολό τους 1.573.104, οι keywords εγγραφές 2.883.220 και αυτές του join product 2.883.220 επίσης. Άρα ΛΟΓΟΣ = 0,773 .

Στις επόμενες μετρήσεις, το στοιχείο που θα διαφοροποιείται θα είναι η συνθήκη επιλογής για τα tuples από τον table movies. Συγκεκριμένα θα γίνεται ολοένα και πιο αυστηρή, απαιτώντας σε κάθε μέτρηση ταινίες με πιο πρόσφατο έτος παραγωγής από την

προηγούμενη. Ο υπολογισμός του t_B/t_C και του ΛΟΓΟΥ γίνεται με τον ίδιο τρόπο. Ακολουθούν οι μετρήσεις.

2 TABLES - ΑΡΟ 1 PREF Ο ΚΑ8ΕΝΑΣ		movies_SELECT <-- preference 4 & keywords <-- preference 2		
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k				
movies (*) -->	1573104	rows		
keywords (*)-->	2883220	rows		
join product -->	2883220	rows		
ΛΟΓΟΣ =	0,772803324			
PLAN-B :	66362	ms		
PLAN-C:	94380	ms	tB/tC =	0,703136
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1960				
movies (>1960) -->	962056	rows		
keywords (*)-->	2883220	rows		
join product -->	2170533	rows		
ΛΟΓΟΣ =	0,885790725			
PLAN-B :	58094	ms		
PLAN-C:	79514	ms	tB/tC =	0,730613
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1970				
movies (>1970) -->	870813	rows		
keywords (*)-->	2883220	rows		
join product -->	2015407	rows		
ΛΟΓΟΣ =	0,931333721			
PLAN-B :	53961	ms		
PLAN-C:	75083	ms	tB/tC =	0,718685
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1975				
movies (>1975) -->	817950	rows		
keywords (*)-->	2883220	rows		
join product -->	1903281	rows		
ΛΟΓΟΣ =	0,972313074			
PLAN-B :	54116	ms		
PLAN-C:	73024	ms	tB/tC =	0,741071
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1980				
movies (>1980) -->	773642	rows		
keywords (*)-->	2883220	rows		
join product -->	1796276	rows		
ΛΟΓΟΣ =	1,017900924			
PLAN-B :	53087	ms		
PLAN-C:	69373	ms	tB/tC =	0,76524
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1985				
movies (>1985) -->	717486	rows		

keywords (*)-->	2883220	rows		
join product -->	1662386	rows		
ΛΟΓΟΣ =	1,08299336			
PLAN-B :	51730	ms		
PLAN-C:	68079	ms	tB/tC =	0,759853

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1990

movies (>1990) -->	650583	rows		
keywords (*)-->	2883220	rows		
join product -->	1496566	rows		
ΛΟΓΟΣ =	1,180637205			
PLAN-B :	49655	ms		
PLAN-C:	64600	ms	tB/tC =	0,768653

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1995

movies (>1995) -->	562464	rows		
keywords (*)-->	2883220	rows		
join product -->	1286205	rows		
ΛΟΓΟΣ =	1,339476989			
PLAN-B :	46410	ms		
PLAN-C:	59452	ms	tB/tC =	0,78063

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2000

movies (>2000) -->	432734	rows		
keywords (*)-->	2883220	rows		
join product -->	969979	rows		
ΛΟΓΟΣ =	1,709291644			
PLAN-B :	41730	ms		
PLAN-C:	48688	ms	tB/tC =	0,85709

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2005

movies (>2005) -->	200335	rows		
keywords (*)-->	2883220	rows		
join product -->	467888	rows		
ΛΟΓΟΣ =	3,295184959			
PLAN-B :	31808	ms		
PLAN-C:	29406	ms	tB/tC =	1,081684

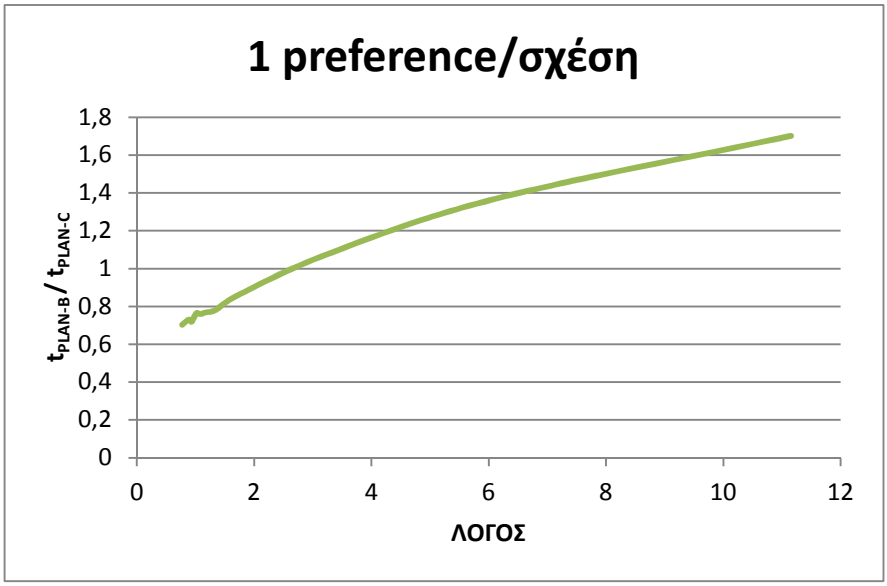
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2007

movies (>2007) -->	110764	rows		
keywords (*)-->	2883220	rows		
join product -->	244005	rows		
ΛΟΓΟΣ =	6,135087396			
PLAN-B :	24648	ms		
PLAN-C:	17986	ms	tB/tC =	1,370399

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2008

movies (>2008) -->	64765	rows		
keywords (*)-->	2883220	rows		
join product -->	132165	rows		
ΛΟΓΟΣ =	11,15266901			
PLAN-B :	22776	ms		
PLAN-C:	13391	ms	tB/tC =	1,700844
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2009				
movies (>2009) -->	15217	rows		
keywords (*)-->	2883220	rows		
join product -->	27232	rows		
ΛΟΓΟΣ =	53,21748311			
PLAN-B :	17270	ms		
PLAN-C:	6037	ms	tB/tC =	2,860692
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2007				
movies (>2010) -->	3885	rows		
keywords (*)-->	2883220	rows		
join product -->	7379	rows		
ΛΟΓΟΣ =	195,6298279			
PLAN-B :	15335	ms		
PLAN-C:	4399	ms	tB/tC =	3,48602

Παρατηρούμε ότι στις πρώτες μετρήσεις, το Plan-B είναι σαφέστατα αποδοτικότερο του Plan-C. Στη συνέχεια, όμως, όσο αυξάνεται ο ΛΟΓΟΣ, η διαφορά μεταξύ τους ψαλιδίζεται και εντέλει το Plan-C αναδεικνύεται καταλληλότερο στις τελευταίες πέντε μετρήσεις. Ακολουθεί το διάγραμμα για τα q1 ερωτήματα:



Εικόνα 34: 2 σχέσεις, από 1 προτίμηση στην καθεμία

Το σημείο στο οποίο γίνεται η αλλαγή καταλληλότητας των δύο πλάνων είναι το σημείο στο οποίο το διάγραμμα τέμνει την οριζόντια γραμμή x=1.

6.2.2.2 2 σχέσεις – από 2 preferences

Ομοίως με πριν, βασιζόμαστε στο SQL ερώτημα Q και αντλούμε δεδομένα από τους πίνακες movies, keywords. Σε αυτούς εφαρμόζονται τα preferences 9 (prod_year<1960), 47 (prod_year>=2010), 26 (keyword LIKE '%sex%') και 28 (keyword LIKE '%family%').

Ακολουθούν οι μετρήσεις των εκτελέσεων.

		movies_SELECT <-- preference 9, 47		
2 TABLES - ΑΠΟ 2 PREF Ο ΚΑΘΕΝΑΣ		& keywords <-- preference 26, 28		
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k				
movies (*) -->	1573104	rows		
keywords (*)-->	2883220	rows		
join product -->	2883220	rows		
ΛΟΓΟΣ =	0,772803324			
PLAN-B :	80262	ms		
PLAN-C:	124192	ms	tB/tC =	0,646274
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1960				
movies (>1960) -->	962056	rows		
keywords (*)-->	2883220	rows		
join product -->	2170533	rows		
ΛΟΓΟΣ =	0,885790725			
PLAN-B :	63180	ms		
PLAN-C:	85083	ms	tB/tC =	0,742569
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1970				
movies (>1970) -->	870813	rows		
keywords (*)-->	2883220	rows		
join product -->	2015407	rows		
ΛΟΓΟΣ =	0,931333721			

PLAN-B : 60482 ms

PLAN-C: 80122 ms tB/tC = 0,754874

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1975

movies (>1975) --> 817950 rows

keywords (*)--> 2883220 rows

join product --> 1903281 rows

ΛΟΓΟΣ = 0,972313074

PLAN-B : 59764 ms

PLAN-C: 76737 ms tB/tC = 0,778816

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1980

movies (>1980) --> 773642 rows

keywords (*)--> 2883220 rows

join product --> 1796276 rows

ΛΟΓΟΣ = 1,017900924

PLAN-B : 63336 ms

PLAN-C: 72634 ms tB/tC = 0,871988

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1985

movies (>1985) --> 717486 rows

keywords (*)--> 2883220 rows

join product --> 1662386 rows

ΛΟΓΟΣ = 1,08299336

PLAN-B : 55178 ms

PLAN-C: 70013 ms tB/tC = 0,788111

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1990

movies (>1990) --> 650583 rows

keywords (*)--> 2883220 rows

join product --> 1496566 rows

ΛΟΓΟΣ = 1,180637205

PLAN-B : 53102 ms

PLAN-C: 63009 ms tB/tC = 0,842768

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1995

movies (>1995) --> 562464 rows

keywords (*)--> 2883220 rows

join product --> 1286205 rows

ΛΟΓΟΣ = 1,339476989

PLAN-B : 50263 ms

PLAN-C: 55708 ms tB/tC = 0,902258

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2000

movies (>2000) --> 432734 rows

keywords (*)--> 2883220 rows

join product --> 969979 rows

ΛΟΓΟΣ = 1,709291644

PLAN-B : 45864 ms

PLAN-C: 43930 ms tB/tC = 1,044025

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2005

movies (>2005) --> 200335 rows

keywords (*)--> 2883220 rows

join product --> 467888 rows

ΛΟΓΟΣ = 3,295184959

PLAN-B : 39547 ms

PLAN-C: 25210 ms tB/tC = 1,568703

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2007

movies (>2007) --> 110764 rows

keywords (*)--> 2883220 rows

join product --> 244005 rows

ΛΟΓΟΣ = 6,135087396

PLAN-B : 34929 ms

PLAN-C: 16458 ms tB/tC = 2,122311

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2008

movies (>2008) --> 64765 rows

keywords (*)--> 2883220 rows

join product --> 132165 rows

ΛΟΓΟΣ = 11,15266901

PLAN-B : 32617 ms

PLAN-C: 12699 ms tB/tC = 2,56847

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2009

movies (>2009) --> 15217 rows

keywords (*)--> 2883220 rows

join product --> 27232 rows

ΛΟΓΟΣ = 53,21748311

PLAN-B : 28906 ms

PLAN-C: 5569 ms tB/tC = 5,190519

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2010

movies (>2010) --> 3885 rows

keywords (*)--> 2883220 rows

join product --> 7379 rows

ΛΟΓΟΣ = 195,6298279

PLAN-B : 27752 ms

PLAN-C: 4695 ms tB/tC = 5,910969

2 TABLES - ΑΠΟ 3 PREF Ο ΚΑΘΕΝΑΣ		movies_SELECT <-- preference 4, 7, 47 & keywords <-- preference 2, 27, 37		
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k				
movies (*) -->	1573104	rows		
keywords (*)-->	2883220	rows		
join product -->	2883220	rows		
ΛΟΓΟΣ =	0,772803324			
PLAN-B :	113911	ms		
PLAN-C:	216919	ms	tB/tC =	0,525132
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1960				
movies (>1960) -->	962056	rows		
keywords (*)-->	2883220	rows		
join product -->	2170533	rows		
ΛΟΓΟΣ =	0,885790725			
PLAN-B :	95269	ms		
PLAN-C:	153473	ms	tB/tC =	0,620754
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1970				
movies (>1970) -->	870813	rows		
keywords (*)-->	2883220	rows		
join product -->	2015407	rows		
ΛΟΓΟΣ =	0,931333721			
PLAN-B :	89732	ms		
PLAN-C:	136488	ms	tB/tC =	0,657435
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1975				
movies (>1975) -->	817950	rows		
keywords (*)-->	2883220	rows		
join product -->	1903281	rows		
ΛΟΓΟΣ =	0,972313074			
PLAN-B :	89996	ms		
PLAN-C:	136672	ms	tB/tC =	0,658482
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1980				
movies (>1980) -->	773642	rows		
keywords (*)-->	2883220	rows		
join product -->	1796276	rows		
ΛΟΓΟΣ =	1,017900924			
PLAN-B :	84677	ms		
PLAN-C:	126626	ms	tB/tC =	0,668717
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1985				
movies (>1985) -->	717486	rows		
keywords (*)-->	2883220	rows		
join product -->	1662386	rows		

ΛΟΓΟΣ =	1,08299336			
PLAN-B :	85021	ms		
PLAN-C:	122085	ms	tB/tC =	0,696408

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1990

movies (>1990) -->	650583	rows		
keywords (*)-->	2883220	rows		
join product -->	1496566	rows		
ΛΟΓΟΣ =	1,180637205			
PLAN-B :	80684	ms		
PLAN-C:	110261	ms	tB/tC =	0,731755

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1995

movies (>1995) -->	562464	rows		
keywords (*)-->	2883220	rows		
join product -->	1286205	rows		
ΛΟΓΟΣ =	1,339476989			
PLAN-B :	74256	ms		
PLAN-C:	99014	ms	tB/tC =	0,749955

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2000

movies (>2000) -->	432734	rows		
keywords (*)-->	2883220	rows		
join product -->	969979	rows		
ΛΟΓΟΣ =	1,709291644			
PLAN-B :	68968	ms		
PLAN-C:	71714	ms	tB/tC =	0,961709

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2005

movies (>2005) -->	200335	rows		
keywords (*)-->	2883220	rows		
join product -->	467888	rows		
ΛΟΓΟΣ =	3,295184959			
PLAN-B :	50575	ms		
PLAN-C:	39156	ms	tB/tC =	1,291628

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2007

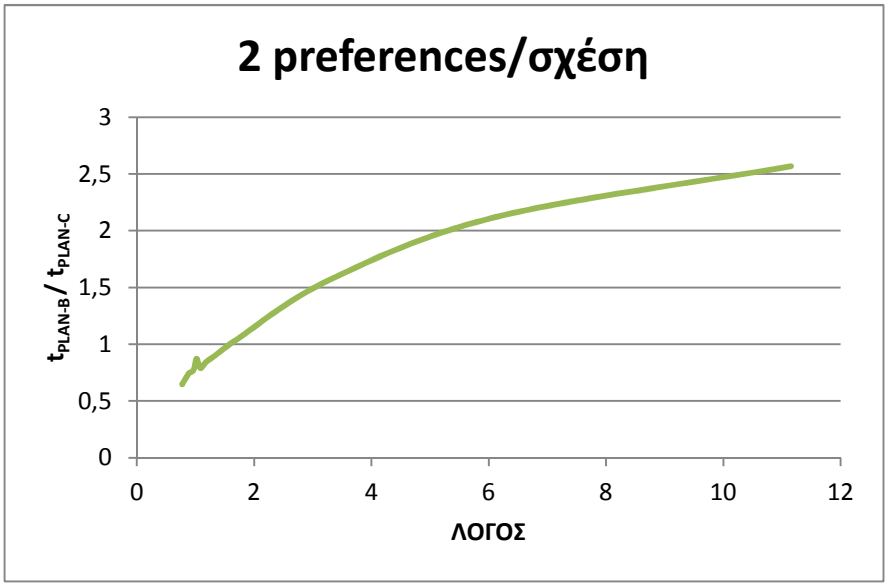
movies (>2007) -->	110764	rows		
keywords (*)-->	2883220	rows		
join product -->	244005	rows		
ΛΟΓΟΣ =	6,135087396			
PLAN-B :	45537	ms		
PLAN-C:	24133	ms	tB/tC =	1,886918

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2008

movies (>2008) -->	64765	rows		
keywords (*)-->	2883220	rows		

join product -->	132165	rows		
ΛΟΓΟΣ =	11,15266901			
PLAN-B :	43325	ms		
PLAN-C:	16848	ms	tB/tC =	2,571522
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2009				
movies (>2009) -->	15217	rows		
keywords (*)-->	2883220	rows		
join product -->	27232	rows		
ΛΟΓΟΣ =	53,21748311			
PLAN-B :	37862	ms		
PLAN-C:	6258	ms	tB/tC =	6,050176
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2007				
movies (>2010) -->	3885	rows		
keywords (*)-->	2883220	rows		
join product -->	7379	rows		
ΛΟΓΟΣ =	195,6298279			
PLAN-B :	35022	ms		
PLAN-C:	4556	ms	tB/tC =	7,687006

Ακολουθεί το αντίστοιχο διάγραμμα:



Εικόνα 35: 2 σχέσεις, από 2 προτιμήσεις στην καθεμία

6.2.2.3 2 σχέσεις – από 3 preferences

Παίρνουμε πάλι τις ίδιες μετρήσεις, με την διαφορά ότι στον πίνακα movies εφαρμόζονται τα preferences 4 (prod_year>=2000), 7 (prod_year>=1980), 47 (prod_year>=2010) και στον keywords τα 2 (keyword LIKE '%police%'), 27 (keyword LIKE '%hardcore%'), 37 (keyword LIKE '%murder%').

Παρατίθενται τα αποτελέσματα:

2 TABLES - ΑΠΟ 3 PREF Ο ΚΑΘΕΝΑΣ		movies_SELECT <-- preference 4, 7, 47 & keywords <-- preference 2, 27, 37		
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k				
movies (*) -->	1573104	rows		
keywords (*)-->	2883220	rows		
join product -->	2883220	rows		
ΛΟΓΟΣ =	0,772803324			
PLAN-B :	113911	ms		
PLAN-C:	216919	ms	tB/tC =	0,525132
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1960				
movies (>1960) -->	962056	rows		
keywords (*)-->	2883220	rows		
join product -->	2170533	rows		
ΛΟΓΟΣ =	0,885790725			
PLAN-B :	95269	ms		
PLAN-C:	153473	ms	tB/tC =	0,620754
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1970				
movies (>1970) -->	870813	rows		
keywords (*)-->	2883220	rows		
join product -->	2015407	rows		
ΛΟΓΟΣ =	0,931333721			
PLAN-B :	89732	ms		
PLAN-C:	136488	ms	tB/tC =	0,657435
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1975				
movies (>1975) -->	817950	rows		
keywords (*)-->	2883220	rows		
join product -->	1903281	rows		
ΛΟΓΟΣ =	0,972313074			
PLAN-B :	89996	ms		
PLAN-C:	136672	ms	tB/tC =	0,658482
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1980				
movies (>1980) -->	773642	rows		
keywords (*)-->	2883220	rows		
join product -->	1796276	rows		
ΛΟΓΟΣ =	1,017900924			
PLAN-B :	84677	ms		
PLAN-C:	126626	ms	tB/tC =	0,668717
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1985				
movies (>1985) -->	717486	rows		
keywords (*)-->	2883220	rows		

join product -->	1662386	rows		
ΛΟΓΟΣ =	1,08299336			
PLAN-B :	85021	ms		
PLAN-C:	122085	ms	tB/tC =	0,696408

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1990

movies (>1990) -->	650583	rows		
keywords (*)-->	2883220	rows		
join product -->	1496566	rows		
ΛΟΓΟΣ =	1,180637205			
PLAN-B :	80684	ms		
PLAN-C:	110261	ms	tB/tC =	0,731755

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>1995

movies (>1995) -->	562464	rows		
keywords (*)-->	2883220	rows		
join product -->	1286205	rows		
ΛΟΓΟΣ =	1,339476989			
PLAN-B :	74256	ms		
PLAN-C:	99014	ms	tB/tC =	0,749955

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2000

movies (>2000) -->	432734	rows		
keywords (*)-->	2883220	rows		
join product -->	969979	rows		
ΛΟΓΟΣ =	1,709291644			
PLAN-B :	68968	ms		
PLAN-C:	71714	ms	tB/tC =	0,961709

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2005

movies (>2005) -->	200335	rows		
keywords (*)-->	2883220	rows		
join product -->	467888	rows		
ΛΟΓΟΣ =	3,295184959			
PLAN-B :	50575	ms		
PLAN-C:	39156	ms	tB/tC =	1,291628

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2007

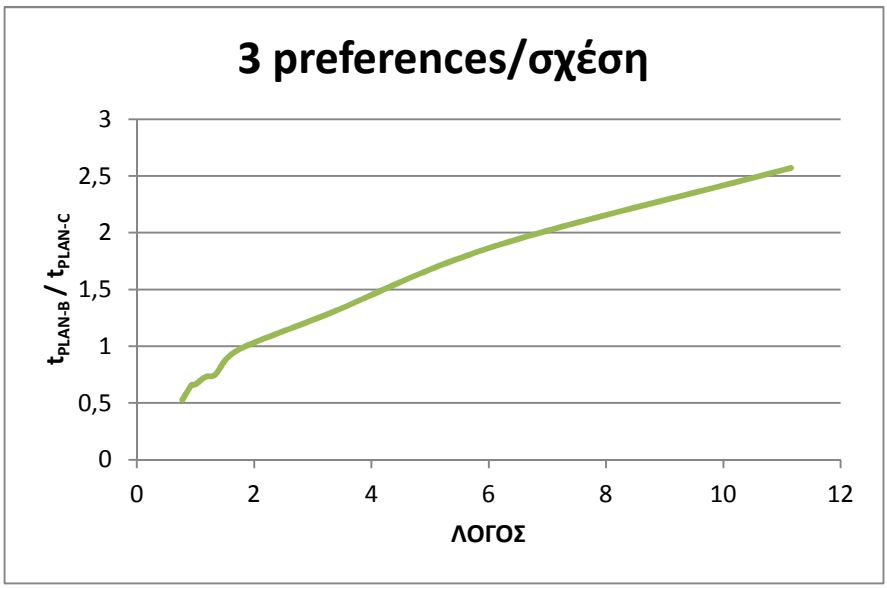
movies (>2007) -->	110764	rows		
keywords (*)-->	2883220	rows		
join product -->	244005	rows		
ΛΟΓΟΣ =	6,135087396			
PLAN-B :	45537	ms		
PLAN-C:	24133	ms	tB/tC =	1,886918

SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2008

movies (>2008) -->	64765	rows		

keywords (*)-->	2883220	rows		
join product -->	132165	rows		
ΛΟΓΟΣ =	11,15266901			
PLAN-B :	43325	ms		
PLAN-C:	16848	ms	tB/tC =	2,571522
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2009				
movies (>2009) -->	15217	rows		
keywords (*)-->	2883220	rows		
join product -->	27232	rows		
ΛΟΓΟΣ =	53,21748311			
PLAN-B :	37862	ms		
PLAN-C:	6258	ms	tB/tC =	6,050176
SELECT m.movieid,m.title,m.prod_year,k.movieid,k.keyword FROM movies m,keywords k WHERE m.prod_year>2007				
movies (>2010) -->	3885	rows		
keywords (*)-->	2883220	rows		
join product -->	7379	rows		
ΛΟΓΟΣ =	195,6298279			
PLAN-B :	35022	ms		
PLAN-C:	4556	ms	tB/tC =	7,687006

Οι μετρήσεις συνοψίζονται στο διάγραμμα:



Εικόνα 36: 3 σχέσεις, από 3 προτιμήσεις στην καθεμία

Η αναμενόμενη συμπεριφορά των δύο εξεταζόμενων πλάνων εκτέλεσης θα ήταν η Plan-B να υπερέχει για τιμές του ΛΟΓΟΥ μικρότερες της μονάδας και η Plan-C για μεγαλύτερες αυτής. Τα πειράματα, όμως, δείχνουν ότι η μετάβαση της καταλληλότητας από Plan-B σε Plan-C δεν γίνεται για τιμές του ΛΟΓΟΥ γύρω από το 1, αλλά γύρω από το 1,7 με 1,8 . Η εφαρμογή των

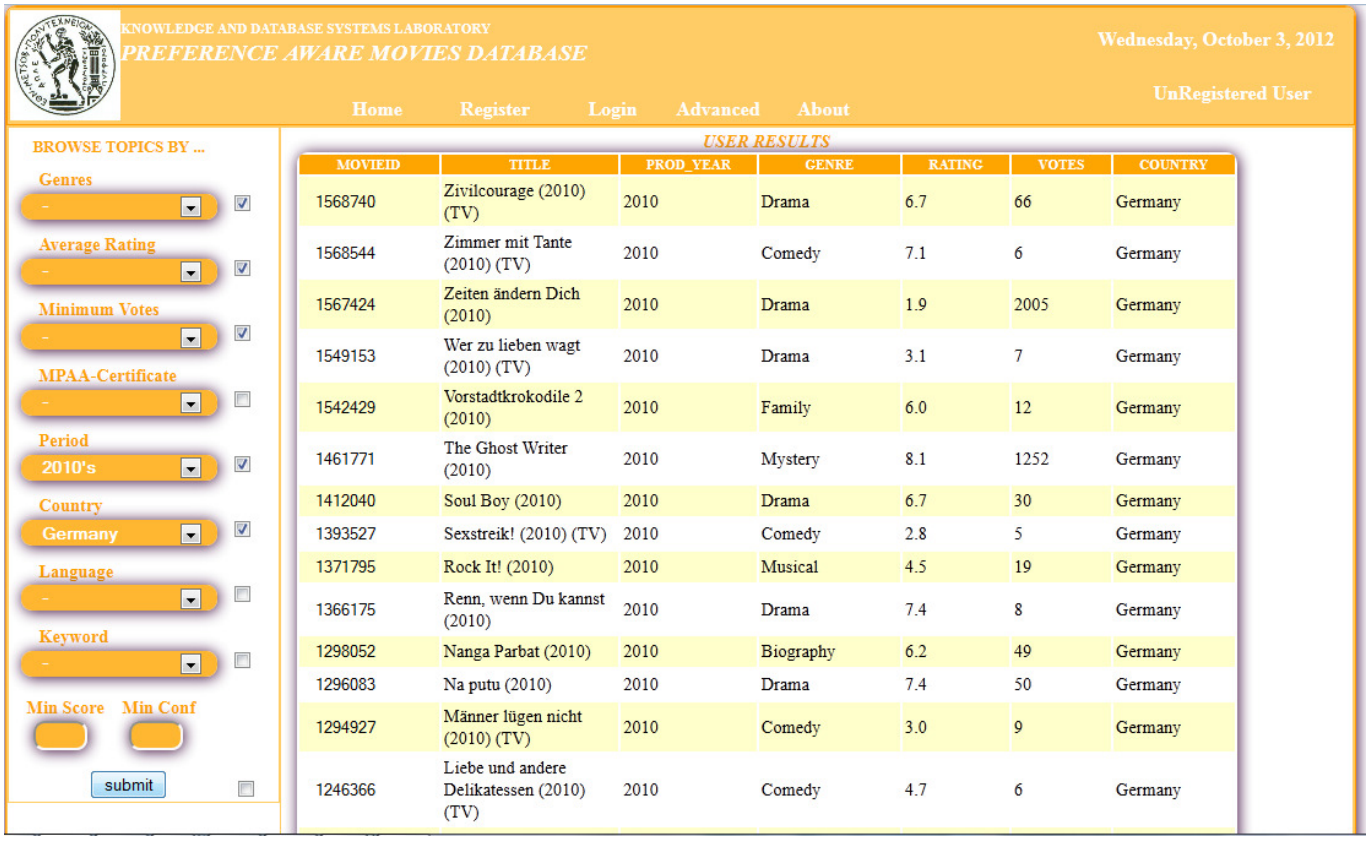
προτιμήσεων στην κάθε μία εγγραφή του join product έχει ένα ελάχιστα μεγαλύτερο κόστος από την εφαρμογή στις επιμέρους σχέσεις, το οποίο προκαλεί το συγκεκριμένο φαινόμενο.

Στην πράξη, όταν θέλουμε να επιλέξουμε το βέλτιστο πλάνο εκτέλεσης, η διαπίστωση αυτή δεν μας είναι ιδιαιτέρως χρήσιμη, καθότι δεν γνωρίζουμε από πριν το μέγεθος του join product ώστε να υπολογίσουμε τον ΛΟΓΟ και να πράξουμε αναλόγως. Μπορούμε όμως να εφαρμόσουμε την γενική διαπίστωση ότι το Plan-B ενδείκνυται για μεγάλο όγκο δεδομένων ενώ το Plan-C είναι πιο αποδοτικό για εξειδικευμένα ερωτήματα και μικρό σχετικά μέγεθος αποτελέσματος.

6.3 Διεπαφή χρήστη

6.3.1 Ερώτημα μη καταχωρημένου χρήστη

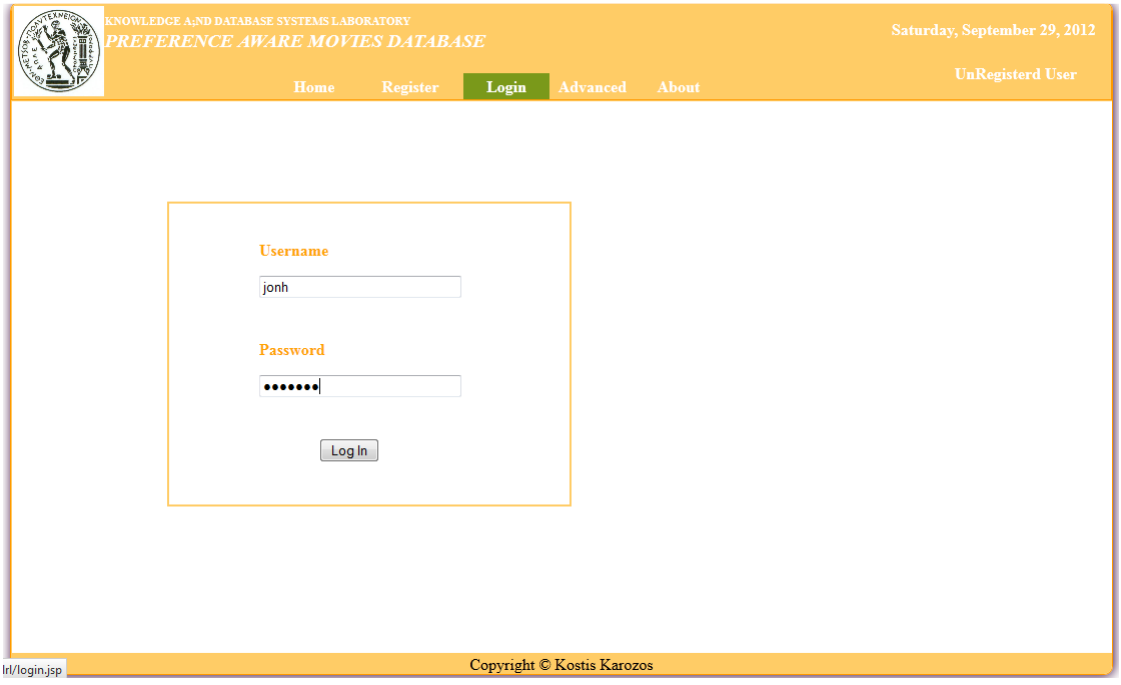
Ο χρήστης εισέρχεται στην *Home* Page και χωρίς να κάνει *Login*, ζητάει από το σύστημα να του εμφανίσει πρόσφατες Γερμανικές ταινίες. Επί πλέον ζητάει να πληροφορηθεί για το είδος της κάθε ταινίας καθώς και για τα ratings και τις ψήφους που έχει λάβει η κάθε μία:



Εικόνα 37: Ερώτημα UnRegistered User

6.3.2 Login χρήστη

Στη συνέχεια αποφασίζει να κάνει *Login*:

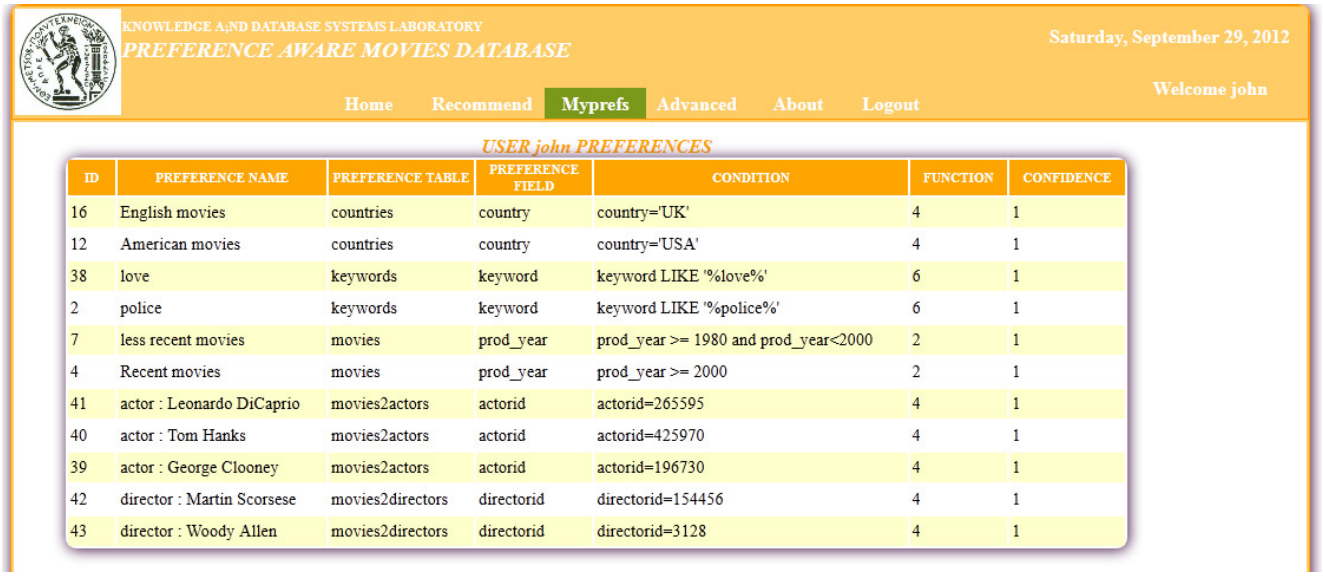


Εικόνα 37: Login χρήστη

και εισέρχεται πλέον στην *Home* Page ως καταχωρημένος User : john.

6.3.3 Καταχωρημένες προτιμήσεις χρήστη

Ο χρήστης john πληροφορείται για το Profile του *(MyPrefs),* βλέπει δηλαδή τις καταχωρημένες γι αυτόν προτιμήσεις από το σύστημα:

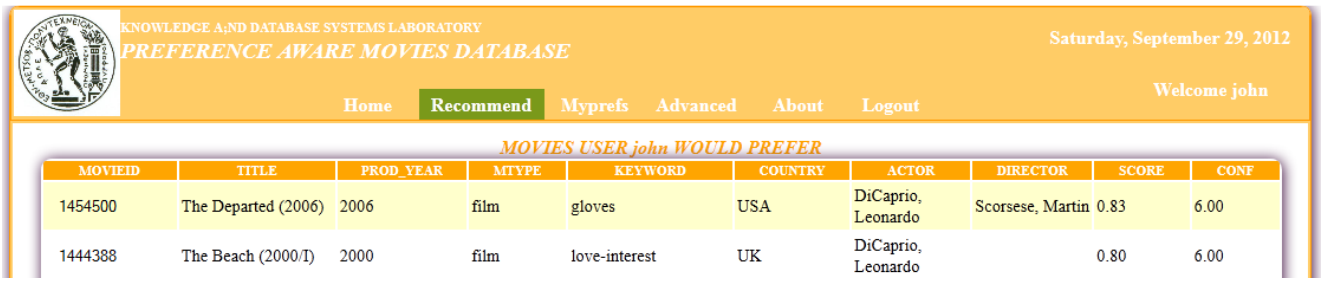


ID	PREFERENCE NAME	PREFERENCE TABLE	PREFERENCE FIELD	CONDITION	FUNCTION	CONFIDENCE
16	English movies	countries	country	country='UK'	4	1
12	American movies	countries	country	country='USA'	4	1
38	love	keywords	keyword	keyword LIKE '%love%'	6	1
2	police	keywords	keyword	keyword LIKE '%police%'	6	1
7	less recent movies	movies	prod_year	prod_year >= 1980 and prod_year<2000	2	1
4	Recent movies	movies	prod_year	prod_year >= 2000	2	1
41	actor : Leonardo DiCaprio	movies2actors	actorid	actorid=265595	4	1
40	actor : Tom Hanks	movies2actors	actorid	actorid=425970	4	1
39	actor : George Clooney	movies2actors	actorid	actorid=196730	4	1
42	director : Martin Scorsese	movies2directors	directorid	directorid=154456	4	1
43	director : Woody Allen	movies2directors	directorid	directorid=3128	4	1

Εικόνα 38: Καταχωρημένες προτιμήσεις χρήστη

6.3.4 Συστάσεις προς τον χρήστη

Ο χρήστης john χωρίς ακόμα να υποβάλει ερώτημα, θέλει να δει τι έχει να του προτείνει το σύστημα *(Recommend)* με βάση τις καταχωρημένες προτιμήσεις του:



MOVIEID	TITLE	PROD_YEAR	MTYPE	KEYWORD	COUNTRY	ACTOR	DIRECTOR	SCORE	CONF
1454500	The Departed (2006)	2006	film	gloves	USA	DiCaprio, Leonardo	Scorsese, Martin	0.83	6.00
1444388	The Beach (2000/I)	2000	film	love-interest	UK	DiCaprio, Leonardo		0.80	6.00

1018056	Catch Me If You Can (2002)	2002	film	police	USA	DiCaprio, Leonardo		0.88	5.00
1019045	Celebrity (1998)	1998	film	love	USA	DiCaprio, Leonardo	Allen, Woody	0.88	5.00
1125904	Gangs of New York (2002)	2002	film	police-officer	USA	DiCaprio, Leonardo	Scorsese, Martin	0.88	5.00
1443321	The Aviator (2004)	2004	film	love	USA	DiCaprio, Leonardo	Scorsese, Martin	0.88	5.00
966474	Anything Else (2003)	2003	film	love-at-first-sight	UK		Allen, Woody	0.84	5.00
1034343	Confessions of a Dangerous Mind (2002)	2002	film	love	UK	Clooney, George		0.84	5.00
1135589	Good Night, and Good Luck. (2005)	2005	film	russian-secret-police	UK	Clooney, George		0.84	5.00
1270976	Match Point (2005)	2005	film	outdoor-love-scene	UK		Allen, Woody	0.84	5.00
1312106	O Brother, Where Art Thou? (2000)	2000	film	police	UK	Clooney, George		0.84	5.00
1017478	Cassandra's Dream (2007)	2007	film		UK		Allen, Woody	0.90	4.00
1033932	Concert for George (2003)	2003	film		UK	Hanks, Tom		0.90	4.00
1106249	Fantastic Mr. Fox (2009)	2009	film		UK	Clooney, George		0.90	4.00
1178076	Inception (2010)	2010	film		UK	DiCaprio, Leonardo		0.90	4.00
1306400	No Direction Home: Bob Dylan (2005)	2005	film		UK		Scorsese, Martin	0.90	4.00
1386534	Scoop (2006)	2006	film		UK		Allen, Woody	0.90	4.00
1397693	Shine a Light (2008)	2008	film		UK		Scorsese, Martin	0.90	4.00
1449773	The Celluloid Closet (1995)	1995	film		UK	Hanks, Tom		0.90	4.00
1476182	The Man in the Iron Mask (1998/I)	1998	film		UK	DiCaprio, Leonardo		0.90	4.00

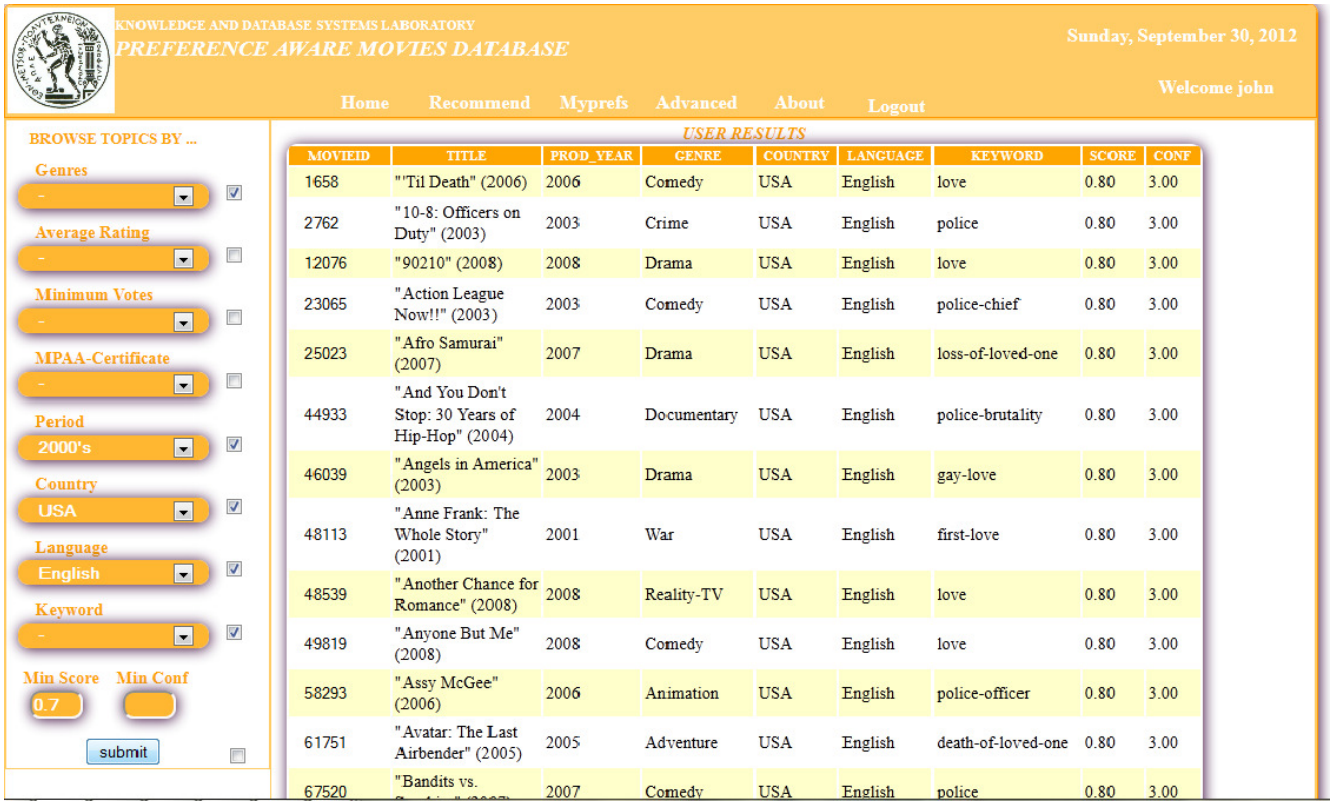
Εικόνα 39: Συστάσεις συστήματος προς τον χρήστη

Παρατηρώντας τρία τμήματα των αποτελεσμάτων διαπιστώνουμε ότι οι συστάσεις προς τον χρήστη john, αντικατοπτρίζουν όλες τις προτιμήσεις του που βλέπουμε στο *MyPrefs*. Δηλαδή: Ταινίες μετά το 2000, αλλά και μεταξύ 1980 και 2000, Αμερικάνικες ή Αγγλικές, κάποιες που περιέχουν τις λέξεις "police" ή "love" , κάποιες που έχουν σκηνοθετηθεί από τον Martin Scorcese ή τον Woody Allen και πολλές στις οποίες παίζουν οι ηθοποιοί Leonardo DiCaprio, Tom Hanks και George Clooney. Οι συστάσεις περιλαμβάνουν 500 ταινίες και η σειρά με την οποία παρουσιάζονται, καθορίζεται από : *score* conf desc*.

6.3.5 Ερώτημα καταχωρημένου χρήστη

Ο χρήστης john επιθυμεί να ζητήσει ταινίες με συγκεκριμένα κριτήρια. Το σύστημα θα αξιολογήσει και θα βαθμολογήσει τα αποτελέσματα της αναζήτησής του με βάση τις προτιμήσεις του.

Ο john μέσω της *Home* Page υποβάλλει το ερώτημά του.



Εικόνα 40: Υποβολή Ερωτήματος από καταχωρημένο χρήστη

Ζητάει από το σύστημα να του φέρει ταινίες οι οποίες πληρούν τις παρακάτω προϋποθέσεις:

1. *Period: 2000 - 2009*
2. *Country : USA*
3. *Language : English*

Επί πλέον δηλώνει ότι επιθυμεί η βαθμολογία *(score)* των ταινιών με βάση τις καταχωρημένες προτιμήσεις του να μην είναι κάτω από 0,7.

Κάνοντας *check* στα *checkboxes: genres, period, country, language, keyword*, καθορίζει ότι αυτά τα attributes θα συμπεριλαμβάνονται στα αποτελέσματα.

Το σύστημα αξιολογεί και βαθμολογεί τις ταινίες οι οποίες προκύπτουν από την εκτέλεση του ερωτήματος του john, με βάση τις σχέσεις που υπάρχουν στο ερώτημα που στην προκειμένη περίπτωση είναι: *genres, movies, countries, language, keywords.* Οι ταινίες που φαίνονται στο screenshot έχουν βαθμολογηθεί από τρία preferences του john : *"American movies", keyword : "love" ή "police", "recent movies"* και ως εκ τούτου έχουν *conf=3*.

Η τιμή του *score* προκύπτει ως εξής:

Η *scoring function f_4 (scoring field: country)* επιστρέφει 0,8.

Η *scoring function f_6 (scoring field: keyword)* επιστρέφει 0,6.

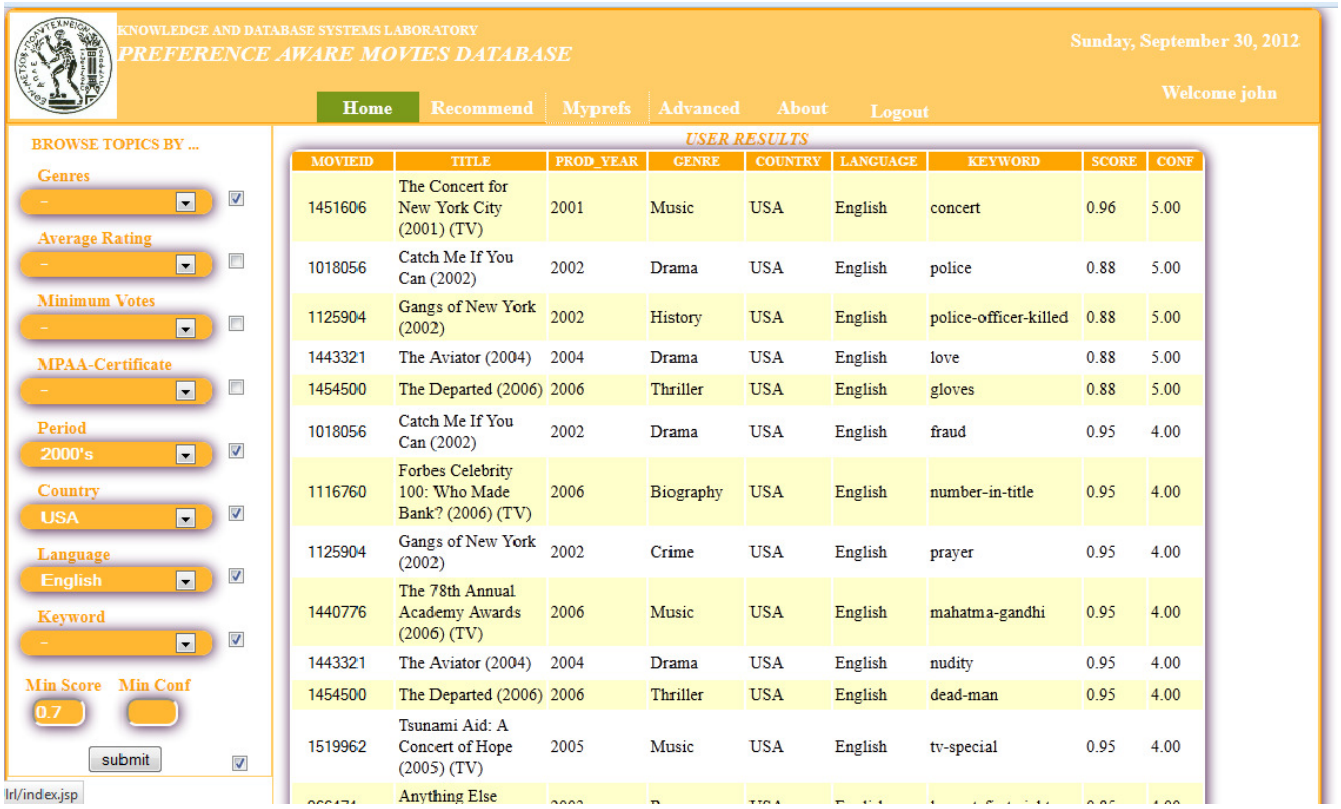
Η *scoring function f_2 (scoring field: prod_year)* επιστρέφει *(prod_year / 2013)* με στογγύλευση στα δύο δεκαδικά ψηφία.

Οι τιμές των *score, conf* που αποδίδουν τα *preferences* σε κάθε ταινία συναθροίζονται (functions *f_avg_s και f_avg_c)* κατά την εκτέλεση του *pref_gl2_exp operator,* επομένως : 0,8 + 0,6 + (0,99 ή 1,00) / 3 = 0,80.

Στο σημείο αυτό κρίνεται σκόπιμο να αναφερθεί ότι το ερώτημα εκτελέστηκε από το υποσύστημα με σχέδιο βέλτιστης εκτέλεσης Plan_B. Κριτήριο γι αυτήν την απόφαση στάθηκε το γεγονός ότι στο ερώτημα συμμετέχουν δυο tables *(genres, keywords)* για τους οποίους δεν έχει οριστεί κανένας περιορισμός, άρα το παραγόμενο join product των relations που συμμετέχουν στο ερώτημα θα είναι αρκετά μεγάλο, επομένως είναι πιο πιθανό να επιτευχθεί καλύτερος χρόνος αν κάθε preference εφαρμοστεί στην αντίστοιχη relation (Plan_B), παρά αν όλα τα preferences εφαρμοστούν στο join product (Plan_C).

Πράγματι το συγκεκριμένο ερώτημα με εκτέλεση Plan_B έχει χρόνο 66.362 ms, ενώ με εκτέλεση Plan_C έχει χρόνο 143.708 ms.

Στη συνέχεια ο χρήστης, κάνοντας ένα check στο *checkbox* δίπλα στο *Submit button*, υποβάλει πάλι το ίδιο ερώτημα με τη διαφορά ότι τώρα θέλει να πάρει τα αποτελέσματα αξιολογημένα και βαθμολογημένα από όλα τα καταγεγραμμένα γι αυτόν preferences και όχι μόνο από τα σχετικά με το ερώτημα preferences.



MOVIEID	TITLE	PROD_YEAR	GENRE	COUNTRY	LANGUAGE	KEYWORD	SCORE	CONF
1451606	The Concert for New York City (2001) (TV)	2001	Music	USA	English	concert	0.96	5.00
1018056	Catch Me If You Can (2002)	2002	Drama	USA	English	police	0.88	5.00
1125904	Gangs of New York (2002)	2002	History	USA	English	police-officer-killed	0.88	5.00
1443321	The Aviator (2004)	2004	Drama	USA	English	love	0.88	5.00
1454500	The Departed (2006)	2006	Thriller	USA	English	gloves	0.88	5.00
1018056	Catch Me If You Can (2002)	2002	Drama	USA	English	fraud	0.95	4.00
1116760	Forbes Celebrity 100: Who Made Bank? (2006) (TV)	2006	Biography	USA	English	number-in-title	0.95	4.00
1125904	Gangs of New York (2002)	2002	Crime	USA	English	prayer	0.95	4.00
1440776	The 78th Annual Academy Awards (2006) (TV)	2006	Music	USA	English	mahatma-gandhi	0.95	4.00
1443321	The Aviator (2004)	2004	Drama	USA	English	nudity	0.95	4.00
1454500	The Departed (2006)	2006	Thriller	USA	English	dead-man	0.95	4.00
1519962	Tsunami Aid: A Concert of Hope (2005) (TV)	2005	Music	USA	English	tv-special	0.95	4.00
	Anything Else							

Εικόνα 41 : Ερώτημα χρήστη αξιολογημένο από όλα τα preferences

Παρατηρούμε ότι η εικόνα των αποτελεσμάτων είναι τελείως διαφορετική. Στη αξιολόγηση των αποτελεσμάτων του ερωτήματος συμμετείχαν εκτός από τα προηγούμενα preferences, και τα 39, 40, 41 preferences που δηλώνουν αρέσκεια για τους ηθοποιούς George Clooney, Tom Hanks, Leonardo DiCaprio αντίστοιχα και τα 42, 43 που δηλώνουν αρέσκεια για τους σκηνοθέτες Martin Scorsese, Woody Allen.

Επειδή στο interface δεν εμφανίζονται τα επιπλέον attributes, προκειμένου να δικαιολογήσουμε τις τιμές των *score, conf* καθότι τα εμφανιζόμενα attributes καθορίζονται αυστηρά από τον χρήστη, παραθέτουμε μέρος του αποτελέσματος του ερωτήματος από τον PgAdmin:

utput pane

Data Output | Explain | Messages | History

	movieid integer	title character varying(400)	prod_year integer	genre character va	country character va	language character vary	keyword character varying(60)	actorid integer	directorid integer	score numeric	conf numeric
1	1451606	The Concert for New York City (20	2001	Music	USA	English	new-york	265595	3128	0.96	5.00
2	1018056	Catch Me If You Can (2002)	2002	Crime	USA	English	police	265595		0.88	5.00
3	1125904	Gangs of New York (2002)	2002	Crime	USA	English	police-officer	265595	154456	0.88	5.00
4	1443321	The Aviator (2004)	2004	Drama	USA	English	love	265595	154456	0.88	5.00
5	1454500	The Departed (2006)	2006	Crime	USA	English	police-surveillanc	265595	154456	0.88	5.00
6	1018056	Catch Me If You Can (2002)	2002	Crime	USA	English	1960s	265595		0.95	4.00
7	1116760	Forbes Celebrity 100: Who Made Ba	2006	Biography	USA	English	number-in-title	265595		0.95	4.00
8	1125904	Gangs of New York (2002)	2002	History	USA	English	punched-in-the-fac	265595	154456	0.95	4.00
9	1440776	The 78th Annual Academy Awards (2	2006	Music	USA	English	mahatma-gandhi	425970		0.95	4.00
10	1443321	The Aviator (2004)	2004	Drama	USA	English	country-estate	265595	154456	0.95	4.00
11	1454500	The Departed (2006)	2006	Crime	USA	English	undercover	265595	154456	0.95	4.00
12	1519962	Tsunami Aid: A Concert of Hope (2	2005	Music	USA	English	tv-special	265595		0.95	4.00
13	966474	Anything Else (2003)	2003	Comedy	USA	English	love-at-first-sigh		3128	0.85	4.00
14	1000098	Body of Lies (2008)	2008	Drama	USA	English	falling-in-love	265595		0.85	4.00
15	1016861	Cars (2006)	2006	Animation	USA	English	police-chase	425970		0.85	4.00

Εικόνα 42: Αποτέλεσμα ερωτήματος από τον PgAdmin

Η πρώτη ταινία 1451606 (η οποία είναι μάλλον μουσική εκδήλωση) έχει βαθμολογηθεί από πέντε preferences (conf=5) :

1. *"American movies",*
2. *"recent movies"*
3. *"actor: Leonardo DiCaprio"*
4. *"director: Woody Allen"*
5. *"director: Martin Scorsese"*

Η επίδραση του τελευταίου preference δεν είναι οφθαλμοφανής, διότι πρόκειται για συνάθροιση δυο preferences στο ίδιο attribute (scoring field), και η ταινία εμφανίζεται μια φορά στα αποτελέσματα. Η απόδειξη υπάρχει στον table της βάσης *movies2directors* όπου στην εν λόγω ταινία εμφανίζονται και οι δύο αναφερόμενοι σκηνοθέτες.

Η δεύτερη ταινία 1018056 έχει βαθμολογηθεί από πέντε preferences (conf=5) :

1. *"American movies",*
2. *"recent movies"*
3. *Keyword : "police"*
4. *"actor: Leonardo DiCaprio"*

5. *"actor: Tom Hanks"*

Η επίδραση του τελευταίου preference και εδώ δεν είναι οφθαλμοφανής, διότι πρόκειται για συνάθροιση δυο preferences στο ίδιο attribute (scoring field), και η ταινία εμφανίζεται μια φορά στα αποτελέσματα. Η απόδειξη υπάρχει στον table της βάσης *movies2actors* όπου στην εν λόγω ταινία παίζουν και οι δύο αναφερόμενοι ηθοποιοί.

Η τρίτη ταινία 1125904 έχει κι αυτή βαθμολογηθεί από πέντε preferences τα οποία είναι όλα οφθαλμοφανή.

Ο χρήστης με click σε μια ταινία έχει τη δυνατότητα να δει λεπτομέρειες της ταινίας :

Εικόνα 43: Λεπτομέρειες ταινίας

6.3.6 Υποβολή ερωτήματος σε query editor

Αν ο χρήστης έχει τις απαιτούμενες γνώσεις *(Advanced user)* το interface του δίνει την δυνατότητα να υποβάλει ερωτήματα στο σύστημα γράφοντάς τα σε query editor.

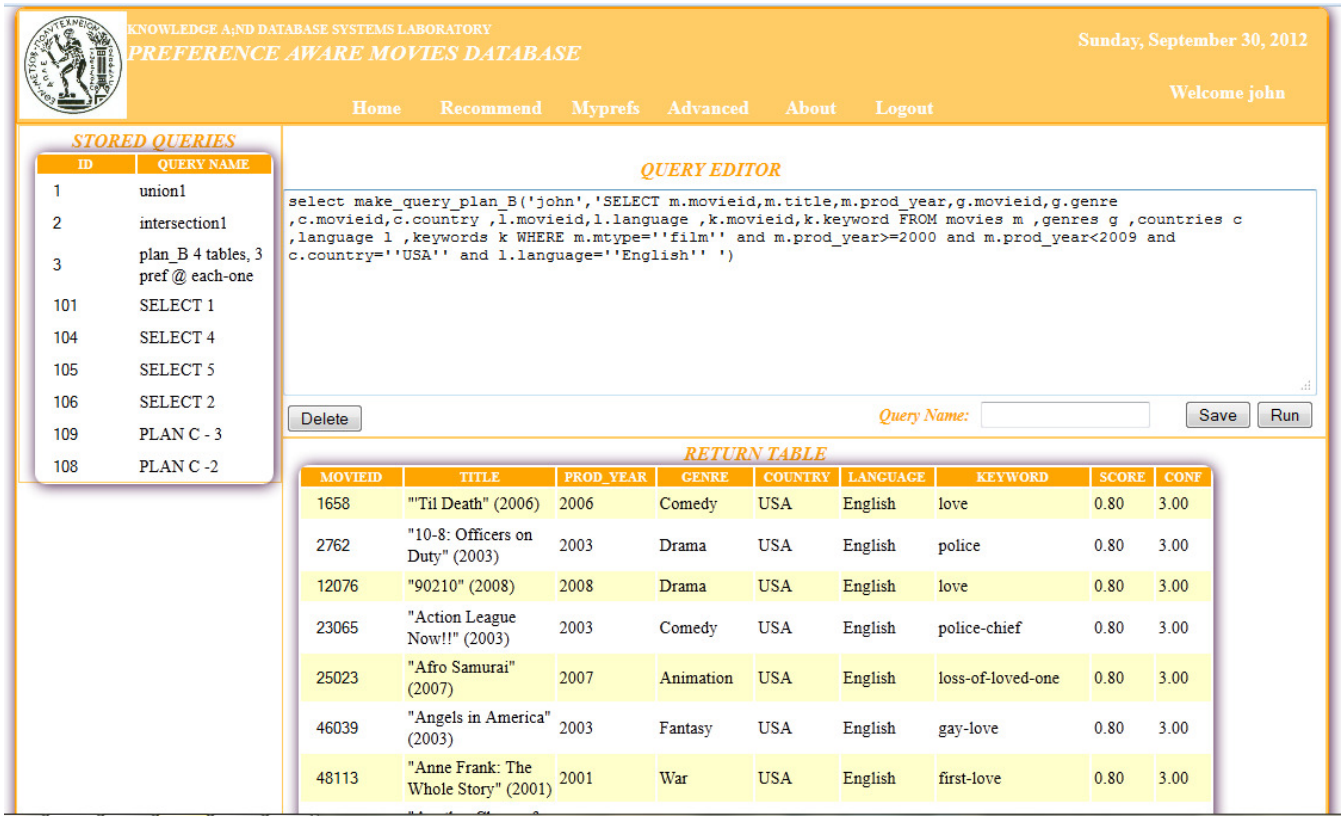
Μπορεί να υποβάλει οποιοδήποτε SQL ή PrefSQL ερώτημα ή να εκτελέσει οποιαδήποτε function. Σε περίπτωση συντακτικού λάθους θα λάβει μήνυμα από τον Server. Μπορεί να αποθηκεύσει το ερώτημά του (είτε είναι σωστό είτε είναι λάθος) για να το επεξεργαστεί και να το εκτελέσει κάποια άλλη φορά, όπως επίσης μπορεί να σβήσει ένα αποθηκευμένο ερώτημα.

Ο χρήστης john θέλει να εισάγει και να εκτελέσει στον query editor το ερώτημα το οποίο προηγουμένως δημιουργήθηκε με τις δικές του επιλογές και δόθηκε για εκτέλεση από το interface.

Επιστρατεύει δύο διαφορετικούς τρόπους:

1ος τρόπος : Καλεί την function *make_query_plan_B* η οποία παραλαμβάνει το SQL ερώτημα σε μη βέλτιστη μορφή και το username του χρήστη, μετατρέπει το ερώτημα σε βέλτιστη μορφή Plan_B και το δίνει για εκτέλεση, επιστρέφει δε μια relation με τα αποτελέσματα:

select make_query_plan_B*('john','SELECT m.movieid, m.title, m.prod_year, g.movieid, g.genre, c.movieid, c.country ,l.movieid, l.language ,k.movieid, k.keyword FROM movies m ,genres g, countries c ,language l ,keywords k WHERE m.prod_year*>=2000 *and m.prod_year<2009 and c.country="USA" and l.language="English" ')*



Εικόνα 44: Query Editor - Run PL/Pg function

Με τον τρόπο αυτό, ο χρήστης john έτρεξε μια function του συστήματος στον query editor.

Κανονικά θα έπρεπε να επιστραφεί το όνομα της relation που δημιουργήθηκε η οποία στην προκειμένη περίπτωση είναι η : *j618_yyy,* και όχι το περιεχόμενό της, αλλά για να απαλλάξει τον χρήστη από πρόσθετο κόπο, το interface εκτελεί το *Select ....from j618_yyy* και επιστρέφει τα αποτελέσματα του ερωτήματος.

2ος τρόπος: Παίρνει ερώτημα σε μορφή PrefSQL όπως το μετέτρεψε η *make_query_plan_B* (ή το συντάσσει μόνος του) και το εκτελεί:

SELECT (($$Select movieid ,title ,prod_year ,score,conf FROM movies
*WHERE prod_year>=2000 and prod_year<2009$$ **** get_db_preference(7))*
***** get_db_preference(4)) &^&*
($$Select movieid ,genre ,score,conf FROM genres$$) &^&
(($$Select movieid ,country ,score,conf FROM countries WHERE country='USA'$$
***** get_db_preference(16)) **** get_db_preference(12)) &^&*
($$Select movieid ,language ,score,conf FROM language WHERE language='English'$$) &^& (($$Select movieid ,keyword ,score,conf FROM keywords$$)
***** get_db_preference(38) **** get_db_preference(2))*

Εικόνα 45: Query Editor - Run PrefSQL query

Με τον τρόπο αυτό, ο χρήστης john έτρεξε ένα PrefSQL ερώτημα στον query editor.

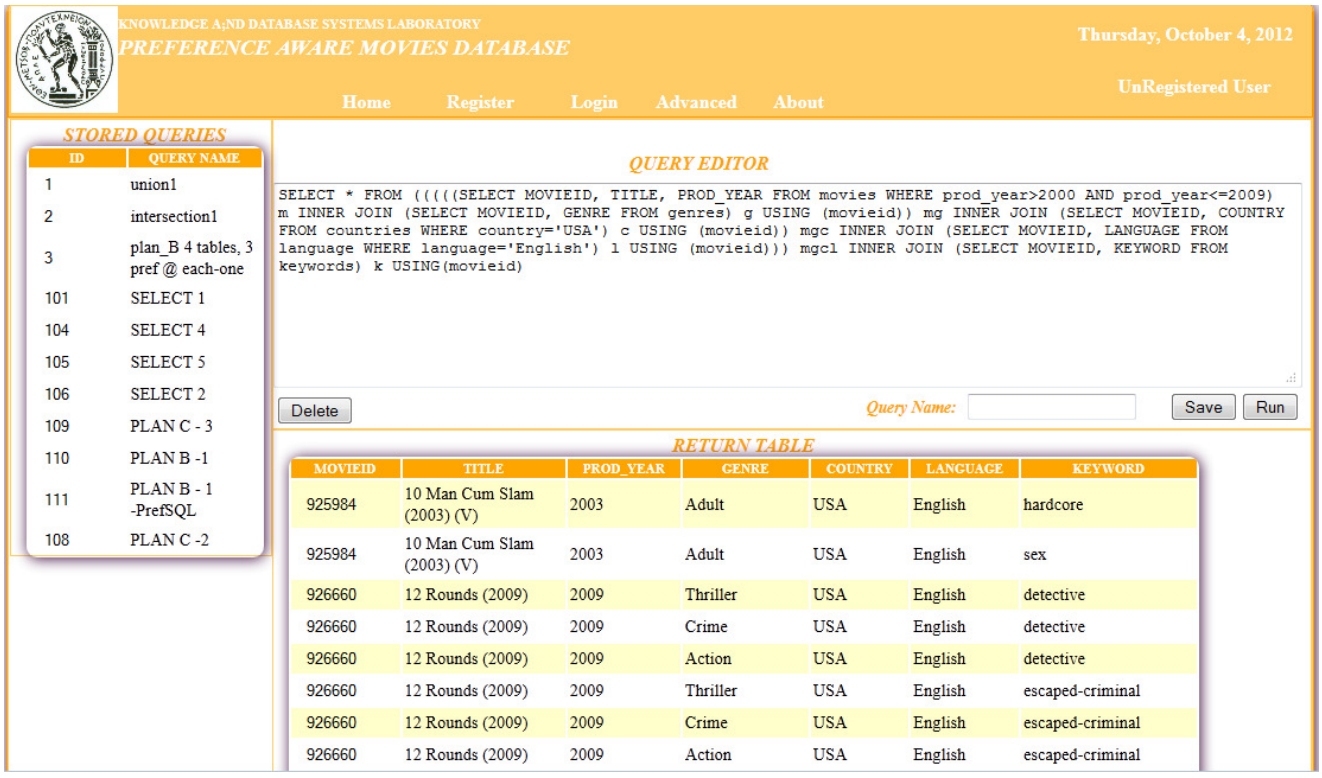
Και εδώ θα έπρεπε να επιστραφεί το όνομα της relation που δημιουργήθηκε, αλλά όπως είπαμε χάριν ευκολίας του χρήστη επιστρέφεται κατευθείαν το αποτέλεσμα του ερωτήματος.

Και με τους δύο αυτούς τρόπους (στην πραγματικότητα ο δεύτερος αποτελεί στάδιο του πρώτου), το αποτέλεσμα του PrefSQL ερωτήματος καταχωρείται σε πίνακα τύπου j999_yyy (εν προκειμένω *j618_yyy*). Ο query editor διευκολύνει τον χρήστη εκτελώντας το εξής:

SELECT distinct on (movieid,score,conf) MOVIEID, TITLE, PROD_YEAR, GENRE, COUNTRY, LANGUAGE, KEYWORD, round(SCORE,2) as SCORE, CONF FROM ***j618_yyy*** *ORDER BY CONF DESC, SCORE DESC LIMIT 1000*

Ο χρήστης έχει την δυνατότητα μέσω του Query Editor να εκτελέσει και ένα οποιοδήποτε απλό SQL ερώτημα. Ο john συντάσει το προηγούμενο ερώτημα χωρίς την ενσωμάτωση προτιμήσεων:

*SELECT * FROM (((((SELECT MOVIEID, TITLE, PROD_YEAR FROM movies WHERE prod_year>2000 AND prod_year<=2009) m INNER JOIN (SELECT MOVIEID, GENRE FROM genres) g USING (movieid)) mg INNER JOIN (SELECT MOVIEID, COUNTRY FROM countries WHERE country='USA') c USING (movieid)) mgc INNER JOIN (SELECT MOVIEID, LANGUAGE FROM language WHERE language='English') l USING (movieid))) mgcl INNER JOIN (SELECT MOVIEID, KEYWORD FROM keywords) k USING(movieid)*



Εικόνα 46: Query Editor - Run SQL query

7 Επίλογος

7.1 Σύνοψη και συμπεράσματα

Στην παρούσα διπλωματική εργασία επεκτείναμε το σχεσιακό μοντέλο βάσεων δεδομένων και υλοποιήσαμε την γλώσσα PrefSQL με την οποία γίνεται δυνατή η ενσωμάτωση των προτιμήσεων των χρηστών στα δεδομένα. Για τον σκοπό αυτό:

Υλοποιήσαμε πέντε νέους τελεστές : α) Δύο τελεστές (prefer) οι οποίοι αξιολογούν και βαθμολογούν (score) τα δεδομένα σύμφωνα με μια προτίμηση και στη συνέχεια καθιστούν τα δεδομένα ενήμερα για τα *score, conf* που τους απέδωσαν. β) Τρεις binary τελεστές join, intersect, union οι οποίοι κατά την εκτέλεσή τους διαχειρίζονται κατάλληλα τα αξιολογημένα από τους prefer τελεστές δεδομένα και συναθροίζουν τα score αυτών των δεδομένων που αναφέρονται στην ίδια οντότητα. Με τον τρόπο αυτό η αξιολόγηση είναι ολοκληρωμένη και προσφέρεται για εξαγωγή συστάσεων.

Παρά τους περιορισμούς που επιβάλει η PL/pgSQL, πραγματοποιούμε διαδοχικές κλήσεις όλων των operators.

Διαμορφώσαμε τα δύο αποδοτικότερα πλάνα εκτέλεσης PrefSQL ερωτημάτων και πραγματοποιήσαμε πειράματα προκειμένου να διαπιστώσουμε ποιό εξ' αυτών είναι βέλτιστο και υπό ποιές συνθήκες.

Αναπτύξαμε (με PL/pgSQL) εφαρμογή η οποία αναλύει ένα SQL ερώτημα, συντάσσει το αντίστοιχο PrefSQL ερώτημα σε βέλτιστη μορφή και το εκτελεί.

Αναπτύξαμε διαδικτυακή εφαρμογή βασιζόμενη στην PrefSQL, με την οποία οι χρήστες θέτουν ερωτήματα και παίρνουν απαντήσεις διατεταγμένες σύμφωνα με τις καταγεγραμμένες προτιμήσεις τους. Για τις ανάγκες της εφαρμογής υλοποιήσαμε τρεις ακόμα νέους τελεστές. Το ερώτημα εκτελείται με εκείνο το πλάνο εκτέλεσης που κρίνεται βέλτιστο ανάλογα με το ερώτημα το οποίο τίθεται.

7.2 Μελλοντικές επεκτάσεις

Στο επίπεδο του Υποσυστήματος Διεπαφής (I), μια μελλοντική επέκταση είναι να γίνεται καταχώρηση νέων προτιμήσεων του χρήστη. Αυτό μπορεί να συμβεί άμεσα αλλά και έμμεσα. Άμεσος τρόπος είναι να δηλώνει ο ίδιος ο χρήστης τα ενδιαφέροντά του ή να βαθμολογεί αντικείμενα και το σύστημα να αναπαράγει προτιμήσεις με βάση τις βαθμολογήσεις και τα χαρακτηριστικά των αντικειμένων αυτών. Έμμεσος τρόπος είναι το σύστημα να μαθαίνει, συμπεραίνει από μόνο του προτιμήσεις επεξεργαζόμενο τα αντικείμενα τα οποία έχουν επιλεγεί από τον χρήστη (self-learning).

Όσον αφορά το Υποσύστημα Διαμόρφωσης Ερωτημάτων (II), μία πραγματικά χρήσιμη επέκταση είναι η περαιτέρω μελέτη των συμπεριφορών των Plan-B, Plan-C και η ουσιαστική εκμετάλλευση των πορισμάτων ώστε το σύστημα να κάνει ένα ολοκληρωμένο και αποτελεσματικό optimization.

Επιπλέον, κρίνεται σκόπιμη η βελτίωση των συναρτήσεων *make_query_plan_B, make_query_plan_C* ώστε να αντιμετωπίζουν όλο το φάσμα των SQL ερωτημάτων που δέχονται ως input χωρίς την ύπαρξη περιορισμών.

Στο κατώτερο επίπεδο του PrefSQL συστήματος, αυτό του Υποσυστήματος Εκτέλεσης Ερωτημάτων (III), θα ήταν σαφώς προτιμότερο οι τελεστές να επιστρέφουν την ίδια την σχέση και όχι το όνομά της. Η υλοποίηση αυτή δεν κατέστη δυνατή λόγω της αδυναμίας της PL/pgSQL να επιστρέφει table όταν το σχήμα αυτού δεν υπάρχει στην δήλωση της function.

Μία επιπλέον βελτίωση είναι το preference να μην περιορίζεται σε attributes μίας μόνο σχέσης.

8 Βιβλιογραφία

[ASS+05] *Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen and Alexander Tuzhilin,*
Incorporating Contexual Information in Recommender Systems Using a Multidimensional Approach,
ACM Transactions on Information Systems, vol 23, No. 1, pages 103-145, 2005

[AK11] *Anastasios Arvanitis, Georgia Koutrika,*
Towards Preference-aware Relational Databases,
Data Engineering (ICDE), IEEE 28th International Conference, 2012

[AT05] *Gediminas Adpmavicious, Alexander Tuzhilin,*
Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions,
IEEE Transactions on knowledge and data engineering, Vol.17, No 6, 2005

[DA00] *Anind K. Dey and Gregory D. Abowd,*
Towards a Better Understanding of Context and Context-Awareness,
Georgia Institute of Technology, Atlanta, 1999

[KBG09] *Georgia Koutrika, Bercovitz B. and Garcia-Molina H,*
FlexRecs: Expressing and Combining Flexible Recommendations,
ACM SIGMOD Conference, pages 745-757, 2009

[KEW11] *Werner Kießling, Markus Endres, Florian Wenzel,*
The Preference SQL System – An Overview,
University of Augsburg, Institute for Computer Science, 2011

[Kou11] *Georgia Koutrika,*
Personalized DBMS: an Elephant in Disguise or a Chameleon?,
IBM Almaden Research Center, USA, 2011

[LM10] *Justin J.Levandoski, Mohamed F.Mokbel,*
CareDB: A Context and Preference-Aware Location-Based Database System,
Department of Computer Science and Engineering, University of Minnesota, Minneapolis, 2010

[LMK10] *Justin J.Levandoski, Mohamed F.Mokbel, Mohamed E.Khalefa,*
FlexPref: A Framework for Extensible Preference Evaluation in Database

Systems,
Department of Computer Science and Engineering, University of Minnesota, Minneapolis, 2010