Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Τεχνολογίας Λογισμικού

Αντικειμενοστραφές Περιβάλλον Διαχείρισης Ερωτημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΙΑΣΟΝΑ Β. ΑΝΥΦΑΝΤΑΚΗ

Επιβλέπων: **Κώστας Κοντογιάννης**
Αναπληρωτής Καθηγητής

ΑΘΗΝΑ, ΦΕΒΡΟΥΑΡΙΟΣ 2016

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Τεχνολογίας Λογισμικού

Αντικειμενοστραφές Περιβάλλον Διαχείρισης Ερωτημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
του

ΙΑΣΟΝΑ Β. ΑΝΥΦΑΝΤΑΚΗ

Επιβλέπων: **Κώστας Κοντογιάννης**
Αναπληρωτής Καθηγητής

Εγκρίθηκε από τριμελή εξεταστική επιτροπή την 17 Φεβρουαρίου 2016.

…….	…….	…….
Κώστας Κοντογιάννης	**Ιωάννης Βασιλείου**	**Γιώργος Στάμου**
Αναπληρωτής Καθηγητής	Καθηγητής	Επίκουρος Καθηγητής

ΑΘΗΝΑ, ΦΕΒΡΟΥΑΡΙΟΣ 2016

……..

Ιάσονας Β. Ανυφαντάκης

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

Ευχαριστίες

Θα ήθελα αρχικά να ευχαριστήσω τον επιβλέποντα μου, Αναπληρωτή Καθηγητή Κώστα Κοντογιάννη. Οι συμβουλές του και οι προτάσεις του έπαιξαν καθοριστικό ρόλο στη δομή της διπλωματικής. Ο ενθουσιασμός του και η υποστήριξή του αποτέλεσαν κινητήριο δύναμη για την ολοκλήρωση αυτής της διπλωματικής. Οι διαλέξεις του μαθήματος Τεχνολογίας Λογισμικού αποτέλεσαν σημαντική αφετηρία του έργου αυτού, καθώς με εισήγαγαν στη λογική του αντικειμενοστραφούς προγραμματισμού και της σωστής δομής που πρέπει να ακολουθεί κάθε εφαρμογή. Θα ήθελα να ευχαριστήσω ακόμα τον Άκη Χουρδάκη, συνάδελφο στη δουλειά, ο οποίος με τις γνώσεις του και την κατάρτισή του στον τομέα της πληροφορικής παρείχε οδηγίες και ιδέες κατά την ανάπτυξη της εφαρμογής που πραγματεύεται η διπλωματική αυτή. Οι συζητήσεις μας για ζητήματα υλοποιήσεων αποτέλεσαν μεγάλη έμπνευση για εμένα.
Στη συνέχεια θα ήθελα να ευχαριστήσω όλα τα μέλη της πανεπιστημιακής κοινότητας με τα οποία συναναστράφηκα αυτά τα χρόνια και με βοήθησαν να φέρω σε πέρας τις σπουδές μου. Θα ήθελα να ευχαριστήσω τους καθηγητές μου, που με εφοδίασαν με ένα δυνατό υπόβαθρο γνώσεων που θα με βοηθήσει στην καριέρα μου και στη ζωή μου. Ακόμα θα ήθελα να ευχαριστήσω όλους τους φίλους και φίλες που γνώρισα κατά τη διάρκεια των σπουδών μου. Οι μοναδικές προσωπικότητες τους έδωσαν χρώμα στη ζωή μου κατά τη φοιτητική μου σταδιοδρομία. Τους εύχομαι να είναι πάντα ευτυχισμένοι και να επιτύχουν σε κάθε πτυχή της ζωής τους.
Τέλος θα ήθελα να ευχαριστήσω την οικογένειά μου. Θέλω να ευχαριστήσω τους γονείς μου που με ανέθρεψαν με αξίες και ιδανικά και φρόντισαν να περάσω γεμάτα και ευχάριστα παιδικά και φοιτητικά χρόνια. Χάρης αυτούς μπόρεσα να αφοσιωθώ στις σπουδές μου και να αποχτήσω τα κατάλληλα εφόδια για να πορευτώ στη ζωή μου. Θα ήθελα να ευχαριστήσω ακόμα τα αδέρφια μου, που έδιναν ζωή στο σπίτι κάθε μέρα και βρίσκονταν πάντα εκεί για μένα. Τη διπλωματική την αφιερώνω στην οικογένειά μου γιατί χωρίς αυτήν δε θα βρισκόμουν σε αυτό το σημείο που βρίσκομαι τώρα.

Abstract

This thesis deals with the management of queries in an object oriented environment. The main objective is the establishment of a communication bridge between a relational model, that relational databases implement and the object oriented logic that many languages follow to date. The languages that are used are Java, as the object oriented language, and SQL, which manages relational databases. This project follows the conventions that are introduced from the archetype of the JPA. This archetype is being used from major successful projects, like Hibernate, which is one of the most known, and QueryDsl. Alienating from this archetype can limit a lot the capabilities and expansion of this application. It would also make it harder for users to become familiar with it, since they would have to learn a new archetype. The application consists of three main stages. The first one is the production of the model that represents the database and translates it to objects. The second stage is the use of said objects in order to produce queries and the last one is the processing of returned data and presenting them to the user. A lot of java technologies were used, like for example reflection of java classes and annotation processing, in order to expand the functionality of the java compiler. For the generation of the model a set of plug-ins were used, called EMF (Eclipse Modeling Framework). During this thesis there was extensive use of object oriented logic and as a result a deep understanding of this logic was achieved. In this document there will be a detailed analysis of the technologies that were used, basic aspects of SQL will be mentioned and finally the architecture of the application will be explained. This document also contains the code of most classes and files that were implemented, in order for the user to access them if needed. Finally examples of use cases are given for the best understanding of the application. In the last two sections of the document are given suggestions for the improvement of the application and also the bibliographic references of this document.

keywords
java, sql, object relational mapping (ORM), queries, framework, object oriented programming, relational databases

Περίληψη

Η διπλωματική αυτή πραγματεύεται τη διαχείριση ερωτημάτων σε αντικειμενοστραφές περιβάλλον. Γίνεται προσπάθεια για τη δημιουργία μιας γέφυρας μεταξύ του σχεσιακού μοντέλου που ακολουθούν οι σχεσιακές βάσεις δεδομένων και του αντικειμενοστραφούς περιβάλλοντος που ακολουθούν πολλές γλώσσες προγραμματισμού αυτή την περίοδο. Οι γλώσσες που χρησιμοποιούνται είναι η Java για το τμήμα του αντικειμενοστραφή προγραμματισμού και η σχεσιακή γλώσσα SQL. Η εργασία ακολουθεί τις συμβάσεις που έχουν εισαχθεί από το πρότυπο του JPA. Αυτό το πρότυπο ακολουθούν και άλλα έργα, από τα οποία ξεχωρίζουν το Hibernate κατά κύριο λόγο και το QueryDsl. Η αποξένωση από αυτό το πρότυπο περιορίζει πολύ την επεκτασιμότητα της εφαρμογής αυτής καθώς και θα καθιστούσε την κατανόηση από τους χρήστες της πιο δύσκολη, αφού θα έπρεπε να γίνει εκμάθηση του νέου προτύπου. Η εφαρμογή χωρίζεται σε τρία στάδια. Το πρώτο είναι η παραγωγή ενός μοντέλου που να μεταφράζει τη βάση δεδομένων σε αντικείμενα, το δεύτερο στάδιο είναι η χρήση των αντικειμένων αυτών για την παραγωγή ερωτημάτων και στο τελευταίο στάδιο γίνεται η επεξεργασία των δεδομένων και η επιστροφή τους στο χρήστη. Χρησιμοποιήθηκε ένα ευρύ φάσμα τεχνολογιών σε Java, όπως ενδοσκόπηση κλάσεων και επεξεργασία annotations για την επέκταση της λειτουργικότητας του μεταγλωττιστή της Java. Για τη δημιουργία του μοντέλου χρησιμοποιήθηκε η επέκταση EMF (Eclipse Modeling Framework). Κατά τη διάρκεια της εργασίας αυτής δόθηκε ιδιαίτερη έμφαση σε προγραμματισμό που ακολουθεί αντικειμενοστραφή λογική και αυτό είχε σαν αποτέλεσμα την σε βάθος κατανόηση αυτής της λογικής προγραμματισμού. Σε αυτό το έγγραφο θα εξηγηθούν αναλυτικά όλες οι τεχνολογίες που χρησιμοποιήθηκαν, βασικές έννοιες της SQL και η αρχιτεκτονική της εφαρμογής. Το κείμενο συνοδεύεται από τον κώδικα των περισσότερων κλάσεων και αρχείων που δημιουργήθηκαν, έτσι ώστε ο αναγνώστης να μπορέσει να ανατρέξει σε αυτόν αν χρειαστεί. Τέλος δίνονται παραδείγματα λειτουργίας της εφαρμογής για την καλύτερη κατανόησή της. Στις τελευταίες δύο ενότητες δίνονται μερικές προτάσεις για την επέκταση της εργασίας αυτής οι καθώς και οι βιβλιογραφικές αναφορές του κειμένου.

Λέξεις κλειδιά
java, sql, object relational mapping (ORM), framework, ερωτήματα, αντικειμενοστραφής προγραμματισμός, σχεσιακές βάσεις δεδομένων

ΠΕΡΙΕΧΟΜΕΝΑ

Ευχαριστίες **4**
Abstract **5**
Περίληψη **6**
ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ **9**
ΚΕΦΑΛΑΙΟ 1 ..... **10**
Εισαγωγή **10**
1.1 Motivation **10**
1.2 Περιγραφή Προβλήματος **10**
1.3 Σκιαγράφηση Λύσης **10**
1.4 Συνεισφορά **11**
1.5 Δομή **12**
ΚΕΦΑΛΑΙΟ 2 ..... **14**
Βιβλιογραφία **14**
2.1 Γνωστές υλοποιήσεις **14**
2.1.1 JPA 14
2.1.2 Hibernate 26
2.1.3 QueryDsl 30
2.2 Εργαλεία **33**
2.2.1 Eclipse Modeling Framework (EMF) 33
2.2.2 EMF Jet 42
2.2.3 JavaBeans 44
2.2.4 Reflection (ενδοσκόπηση) 45
2.2.5 Generics 46
2.2.6 Annotation Processing 46
2.3 SQL **47**
2.3.1 Εισαγωγή στις Βάσεις Δεδομένων 47
2.3.2 Σύστημα Διαχείρισης Βάσεων Δεδομένων 47
2.3.3 Σχεσιακό Μοντέλο Βάσης Δεδομένων 48
2.3.4 Πράξεις Ενημερώσεων 48
2.3.5 Εισαγωγή SQL 48
2.3.6 Βασικές ερωτήσεις SQL 49
2.3.7 Συνένωση Πινάκων 51
ΚΕΦΑΛΑΙΟ 3 ..... **53**
Αρχιτεκτονική **53**
3.1 Υλοποίηση **53**
3.1.1 Μοντέλο SQL 53
3.1.2 Βοηθητικά Αρχεία 63
3.1.3 Εξωτερικά αρχεία 63
3.2 Διαδικασία λειτουργίας **65**
3.2.1 Ορισμός Πινάκων 65
3.2.2 Δημιουργία Μεταδεδομένων 67
3.2.3 Δημιουργία Ερωτήματος 68

3.3 Υλοποίηση Κώδικα 79
3.3.1 Field 79
3.3.2 QueryGET 83
3.3.3 JOIN 101
3.3.4 Filters 112
3.3.5 Condition 118
3.3.6 Table 123
3.3.7 Operators 126
3.3.8 Bol 128
3.3.9 NaryNode 131
3.3.10 Factory 135
3.3.11 InitializeMetaData 142
3.3.12 SqlModelAnnotation 144
ΚΕΦΑΛΑΙΟ 4 147
Παραδείγματα Λειτουργίας 147
ΚΕΦΑΛΑΙΟ 5 155
Ζητήματα Υλοποίησης 155
5.1 Περιορισμοί Εφαρμογής 155
5.2 Μελλοντικές Υλοποιήσεις 155
ΚΕΦΑΛΑΙΟ 6 157
Βιβλιογραφικές Αναφορές 157

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

Figure 1: JPA architecture 15
Figure 2: JPA Class Associations 16
Figure 3: ORM Architecture 16
Figure 4: Hibernate Architecture (Simple) 26
Figure 5: Hibernate Architecture (Lite Form) 27
Figure 6: Hibernate Architecture (Full Cream Form) 27
Figure 7: Purchase Order and Item Model Demo 35
Figure 8: Connection between EMF and XML,UML,Java 36
Figure 9: UML of Ecore Metamodel 37
Figure 10: Structure of Demo Project 37
Figure 11: Basic Flow Diagram of EMF 40
Figure 12: JET Tutorial Project 43
Figure 13: JET Settings 43
Figure 14: Tutorial Project with JET functionality 44
Figure 15: UML Diagram of SQL Model 62
Figure 16: Sequence Diagram of application 77
Figure 17: Flow Diagram of application 78

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

1.1 Motivation

Ο αντικειμενοστραφής προγραμματισμός αποτελεί μια μέθοδο προγραμματισμού πολύ διαδεδομένη στη σύγχρονη εποχή για την ανάπτυξη πολλών εμπορικών εφαρμογών, μερικές εκ των οποίων είναι προγράμματα διαχείρισης συστημάτων, διαδικτυακές εφαρμογές, περιβάλλοντα αλληλεπίδρασης με χρήστες και γραφικά. Προκειμένου οι εφαρμογές να είναι πλήρεις συνήθως υλοποιούν και διαχειρίζονται κάποια βάση δεδομένων. Έτσι εξασφαλίζεται η σωστή συντήρηση και διαχείριση των δεδομένων τα οποία επεξεργάζεται η εφαρμογή. Η γνώση και η κατανόηση του αντικειμενοστραφούς προγραμματισμού μπορεί να προσφέρει πολλές δυνατότητες και να ανοίξει πολλούς δρόμους για κάθε προγραμματιστή. Στα πλαίσια αυτής της διπλωματικής γίνεται προσπάθεια καλύτερης κατανόησης του αντικειμενοστραφούς τρόπου προγραμματισμού, με τη χρήση της γλώσσας java και η εξοικείωση με τις πολλές δυνατότητες που προσφέρει. Επιπρόσθετα η διπλωματική αγγίζει και τον τομέα των βάσεων δεδομένων, σε μια προσπάθεια να παντρέψει το αντικειμενοστραφές και σχεσιακό μοντέλο δεδομένων.

1.2 Περιγραφή Προβλήματος

Κατά τη διάρκεια των μαθημάτων, μαθαίνουμε και χρησιμοποιούμε μεθόδους για να επικοινωνούμε με βάσεις δεδομένων. Συγκεκριμένα μαθαίνουμε τη σύνταξη και το λεξιλόγιο που χρειαζόμαστε, προκειμένου να θέτουμε ορθά ερωτήματα σε μια βάση δεδομένων, για τις ενέργειες CREATE,UPDATE,DELETE και SELECT.
Ο πιο συνηθισμένος τρόπος για να χρησιμοποιήσουμε αυτά τα ερωτήματα είναι μέσω άλλων γλωσσών προγραμματισμού, όπως java και JavaScript. Έτσι υπάρχει η δυνατότητα κατασκευής επικοινωνιακών προγραμμάτων, τα οποία ανάλογα με την είσοδό τους κάνουν τα κατάλληλα ερωτήματα σε μια βάση και ο χρήστης λαμβάνει τα αντίστοιχα αποτελέσματα.
Η πιο γνωστή μέθοδος που χρησιμοποιούμε στη σχολή είναι η δημιουργία στατικών ερωτημάτων σε μορφή συμβολοσειρών. Κάτι τέτοιο καλύπτει τις ανάγκες και λειτουργεί σωστά όταν έχουμε μικρό όγκο δεδομένων και μπορούμε να τον διαχειριστούμε με μερικά απλά ερωτήματα. Ωστόσο σε περίπτωση πιο πολύπλοκων συστημάτων αυτή η αντιμετώπιση δεν αποτελεί βιώσιμη λύση, γιατί αυξάνει πολύ την πολυπλοκότητα και μειώνει την επεκτασιμότητα του προγράμματος.
Στα πλαίσια αυτής της διπλωματικής γίνεται προσπάθεια να δοθεί λύση στο παραπάνω θέμα, χωρίς χρήση των ήδη υπάρχοντών εργαλείων που βρίσκονται στην αγορά.

1.3 Σκιαγράφηση Λύσης

Έργο αυτής της διπλωματικής είναι η υλοποίηση μιας επέκτασης του eclipse. Το eclipse είναι μια πλατφόρμα ανάπτυξης λογισμικού βασιζόμενη κυρίως σε java. Για την υλοποίηση της επέκτασης, εκτός από πληθώρα τεχνολογιών της java χρησιμοποιήθηκε και η επέκταση EMF (Eclipse Modeling Framework), η οποία παρέχει γρήγορο και καθαρό κώδικα για την ανάπτυξη μοντέλων.

Η διαδικασία που ακολουθείται για την επίλυση του παραπάνω θέματος χωρίζεται σε τρία βασικά στάδια. Το πρώτο στάδιο είναι η δημιουργία κάποιου μοντέλου της βάσης δεδομένων. Έτσι εξασφαλίζεται η μετάφραση της βάσης από το σχεσιακό μοντέλο, στο οποίο βρίσκεται, στο αντικειμενοστραφές. Για την ανάπτυξη του μοντέλου αξιοποιήθηκε η ομάδα επεκτάσεων που υλοποιούν το EMF. Βασικό ζήτημα που έπρεπε να αντιμετωπιστεί σε αυτό το στάδιο ήταν η ορθή δημιουργία του μοντέλου. Στην τελική υλοποίηση του μοντέλου δημιουργήθηκε μια κεντρική κλάση που αποτελεί τη ρίζα του ερωτήματος. Η κλάση αυτή καθορίζει το είδος του ερωτήματος (CREATE, UPDATE, DELETE, SELECT) και τον αρχικό πίνακα που χρησιμοποιείται. Επιπλέον πίνακες προστίθενται ως συνενώσεις πινάκων μέσω επιπλέων κλάσεων. Η κλάση που αντικατοπτρίζει τα φίλτρα του ερωτήματος συνδέεται μόνο με τη ρίζα. Πολλαπλά φίλτρα συνδέονται μεταξύ τους με κατάλληλες μεθόδους

Θα μπορούσε κάποιος να αναρωτηθεί γιατί τα φίλτρα δεν αντιστοιχίζονται απευθείας με την κλάση του πίνακα στον οποίο εφαρμόζονται. Αυτό συμβαίνει γιατί έτσι δε θα υπήρχε απευθείας ένωση μεταξύ των φίλτρων και κατά συνέπεια θα ήταν δύσκολη η εφαρμογή προτεραιότητας στα φίλτρα. Έτσι προτιμήθηκε να δημιουργηθεί ένα πλέγμα από φίλτρα τα οποία ακολουθώντας δεντρική δομή μπορούν εύκολα να ταξινομηθούν και στη συνέχεια να παρουσιαστούν στο ερώτημα. Επιπλέον η μορφή με την οποία ενώνονται τα φίλτρα στον κώδικα είναι πολύ πιο ευχάριστη στο μάτι του χρήστη και οδηγεί σε πολύ πιο καθαρό κώδικα.

Στο δεύτερο στάδιο γίνεται χρήση του παραπάνω μοντέλου προκειμένου να δημιουργηθεί ένα ορθό ερώτημα για τη βάση δεδομένων. Ιδιαίτερη προσοχή χρειάστηκε προκειμένου να υπάρξει τυπογραφική ασφάλεια στον κώδικα. Έγινε χρήση μεθόδων ενδοσκόπησης των κλάσεων και generics προκειμένου να εξασφαλιστεί η εγκυρότητα των δεδομένων που εισάγει ο χρήστης.

Στο τελευταίο στάδιο γίνεται επεξεργασία των δεδομένων που επιστρέφει η βάση και παρουσίαση τους στον χρήστη. Η επέκταση χρησιμοποιεί την κλάση του πίνακα που χρησιμοποιήθηκε στη ρίζα του ερωτήματος, και παράγοντας αντικείμενα της κλάσης αυτής, τα γεμίζει με τα κατάλληλα δεδομένα και τα παρουσιάζει στον χρήστη. Οι υπόλοιποι πίνακες είναι φωλιασμένοι είτε στην κλάση του πίνακα ρίζα, είτε στην κλάση κάποιου άλλου πίνακα, ανάλογα με τη συσχέτιση που έχουν με άλλους πίνακες στη βάση. Περισσότερες λεπτομέρειες θα δοθούν σε επόμενη ενότητα.

Σε αυτή την υλοποίηση έχουν γίνει κάποιες συμβάσεις. Ο χρήστης θεωρείται πως έχει κάποια απεικόνιση της βάσης δεδομένων που χρησιμοποιεί, η οποία έχει προκύψει είτε από εμπορικό κώδικα, όπως για παράδειγμα το JPA, είτε την έχει γράψει ο ίδιος ο χρήστης με το χέρι. Περισσότερες πληροφορίες για τη δημιουργία των μοντέλων δίνονται σε επόμενη ενότητα. Από εκεί και πέρα η επέκταση που αναπτύχθηκε σε αυτή τη διπλωματική αναλύει αυτό το μοντέλο και παράγει δεδομένα και μεθόδους τις οποίες μπορεί να χρησιμοποιήσει ο χρήστης για τη διαχείριση των ερωτημάτων. Η γλώσσα στην οποία θέτονται τα ερωτήματα είναι η SQL, μια από τις πιο γνωστές γλώσσες διαχείρισης σχεσιακών βάσεων δεδομένων.

1.4 Συνεισφορά

Σε αυτή τη διπλωματική γίνεται προσπάθεια δημιουργίας μιας καινούριας υλοποίησης του JPA. Κάθε υλοποίηση έχει τα δικά της χαρακτηριστικά και μπορεί να αποφασίσει να επεκτείνει την υπάρχουσα λειτουργικότητα του JPA ή να δημιουργήσει κάτι τελείως καινούριο βασιζομένη στα πρότυπα που επιβάλλονται από το JPA και είναι ευρέως γνωστά και χρησιμοποιημένα.

Από ακαδημαϊκή σκοπιά δίνει ένα εργαλείο που ενώνει τον αντικειμενοστραφή προγραμματισμό και τις σχεσιακές βάσεις δεδομένων. Αυτοί οι δύο τομείς χρησιμοποιούνται σε πάρα πολλά σενάρια μαζί και προσφέρουν πολλές δυνατότητες για ανάπτυξη λογισμικού.
Από άποψη υλοποίησης επιλύει ζητήματα EAGER/LAZY ανάκτησης δεδομένων, δίνοντας στο χρήστη τη δυνατότητα να επιλέξει ποια δεδομένα επιθυμεί σε κάθε ερώτημα και να μη λάβει όγκο πληροφορίας που δε χρειάζεται την εκάστοτε στιγμή. Ακόμα διαχειρίζεται πολλαπλή χρήση διαδοχικών ενώσεων (joins) πινάκων που μπορούν να φτάνουν μέχρι αορίστου βάθους. Τέλος προσφέρει μια διαπροσωπεία στον χρήστη με την οποία μπορεί εύκολα να διαχειριστεί τα ερωτήματά του.

1.5 Δομή

Αρχικά εξετάζονται υπάρχοντες υλοποιήσεις και ο γενικός τρόπος λειτουργίας τους. Η αναφορά ξεκινάει με το JPA που λειτουργεί και ως πρότυπο ανάπτυξης λογισμικού για υποστήριξη λειτουργικότητας ORM (Object Relational Mapping). Η πιο γνωστή υλοποίηση που ακολουθεί το πρότυπο JPA είναι το Hibernate, μια ολοκληρωμένη πλατφόρμα διαχείρισης ερωτημάτων. Τέλος γίνεται αναφορά σε μια ακόμα πλατφόρμα, το QueryDsl το οποίο έχει δυνατότητα να παράγει δικά του ερωτήματα, αλλά χρησιμοποιείται κυρίως για ανάπτυξη ερωτημάτων βασισμένα σε άλλες πλατφόρμες, όπως το Hibernate.

Στη συνέχεια εξετάζονται οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη αυτής της επέκτασης. Οι τεχνολογίες αυτές περιλαμβάνουν JavaBeans, Reflection, Annotation Processing, generics καθώς και την επέκταση EMF (Eclipse Modeling Framework) που χρησιμοποιήθηκε για την παρασκευή του μοντέλου. Η έκδοση της java που χρησιμοποιήθηκε είναι η 1.8 αλλά οι τεχνολογίες υποστηρίζονται και σε προηγούμενες εκδόσεις. Σε αυτό το σημείο γίνεται ανάλυση των προαναφερόμενων τεχνολογιών για να κατανοήσει ο αναγνώστης τον τρόπο λειτουργίας τους και τη χρησιμότητά τους.

Στο τέλος της δεύτερης ενότητας εξετάζεται η γλώσσα SQL. Εξετάζονται οι διάφορες λειτουργίες που υποστηρίζει η γλώσσα, καθώς και σημαντικά χαρακτηριστικά του σχεσιακού μοντέλου. Το μοντέλο που υλοποιείται λειτουργεί σύμφωνα με τους κανόνες που ακολουθούν οι σχεσιακές γλώσσες και κυρίως η SQL.

Στην τρίτη ενότητα αναλύεται η αρχιτεκτονική της εφαρμογής. Αρχικά περιγράφεται το μοντέλο της SQL που έχει υλοποιηθεί (μέσω του EMF), οι διάφορες μέθοδοι και τα χαρακτηριστικά τους. Εκτός από τις κλάσεις που υλοποιούν το μοντέλο, εξετάζονται και οι υπόλοιπες βοηθητικές κλάσεις που χρησιμοποιήθηκαν.

Στη συνέχεια παρουσιάζεται ο τρόπος λειτουργίας της επέκτασης. Η λειτουργία ξεκινάει από τον ορισμό των πινάκων από τον χρήστη. Στη συνέχεια εξετάζονται τα διάφορα μεταμοντέλα, πως παράγονται και που χρησιμεύουν. Έπειτα δίνεται η λογική με την οποία κατασκευάζεται το ερώτημα και ποιοι κανόνες και συμβάσεις οφείλουν να ακολουθούνται. Στο τέλος του τμήματος αυτού αναλύεται η επεξεργασία των δεδομένων και ο τρόπος που μπορεί να τα χειριστεί ο χρήστης.

Στο τελευταίο τμήμα της τρίτης ενότητας δίνεται ο κώδικας των περισσοτέρων κλάσεων που υλοποιούν το μοντέλο, καθώς και τα διάφορα βοηθητικά αρχεία. Οι κώδικες συνοδεύονται και από σχόλια για να βοηθήσουν τον αναγνώστη να κατανοήσει τον τρόπο λειτουργίας τους.

Στην τελευταία ενότητα δίνονται μερικά παραδείγματα λειτουργίας της ολοκληρωμένης εφαρμογής, ξεκινώντας από απλά σενάρια χρήσης και σταδιακά παρουσιάζονται πιο σύνθετα ερωτήματα.

ΚΕΦΑΛΑΙΟ 2

Βιβλιογραφία

2.1 Γνωστές υλοποιήσεις

Ανά τα χρόνια έχουν αναπτυχθεί διάφορα εργαλεία προκειμένου να δημιουργούνται με δυναμικό τρόπο τυπογραφικά ασφαλή ερωτήματα. Χρησιμοποιούν στοιχεία από το Java Persistence Api (JPA) και δύο γνωστά εργαλεία που χρησιμοποιούνται είναι τα Hibernate και Querydsl. Παρακάτω γίνεται μια αναφορά στα εργαλεία αυτά.

2.1.1 JPA

Το JPA παρέχει στους χρήστες έναν τρόπο για συσχέτιση μεταξύ ενός αντικειμενοστραφούς και ενός σχεσιακού μοντέλου για Java εφαρμογές. Αποτελείται από τέσσερα αντικείμενα:

- Το Java Persistence API
- Τη γλώσσα ερωτημάτων
- Το Java Persistence Criteria API
- Μεταδεδομένα για συσχέτιση μεταξύ Αντικειμένων/Σχέσεων

Βασικό γνώρισμα του JPA είναι τα αντικείμενα τύπου Entity, ή αλλιώς οντότητες. Οι οντότητες είναι ελαφρά αντικείμενα ορισμού, με δυνατότητα διατήρησης του περιεχομένου τους. Τυπικά, η οντότητα αναφέρεται σε έναν πίνακα μιας σχεσιακής βάσης δεδομένων και κάθε αντικείμενο της οντότητας αντιστοιχεί σε μια γραμμή του πίνακα. Το βασικό προγραμματιστικό στοιχείο μιας οντότητας είναι η ίδια η κλάση entity, αν και μπορεί να χρησιμοποιήσει και άλλες βοηθητικές κλάσεις. Η διατήρηση της κατάστασης του αντικειμένου αυτού επιτυγχάνεται είτε από διατηρούμενα πεδία είτε από διατηρούμενες ιδιότητες. Αυτά τα πεδία ή οι ιδιότητες χρησιμοποιούν annotations που συμβάλουν στη συσχέτιση μεταξύ αντικειμένων και σχέσεων. Με αυτό τον τρόπο αντιστοιχίζονται οι οντότητες μιας αντικειμενοστραφούς εφαρμογής με τα σχεσιακά δεδομένα μιας βάσης.

Αρχιτεκτονική του JPA

Παρακάτω δίνεται η αρχιτεκτονική επιπέδου κλάσεων του JPA:

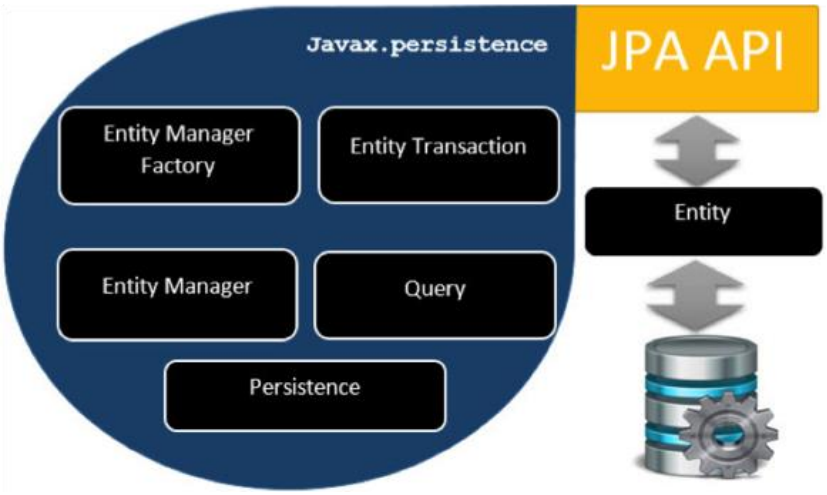

Figure 1: JPA architecture

Το τμήματα της αρχιτεκτονικής περιγράφονται παρακάτω:

- **EntityManagerFactory:** Αυτή είναι μια κλάση εργοστάσιο του EntityManager. Δημιουργεί και διαχειρίζεται πολλαπλά αντικείμενα από EntityManager.
- **EntityManager:** Είναι μια διαπροσωπεία η οποία διαχειρίζεται τις λειτουργίες διατήρησης της πληροφορίας των αντικείμενων. Λειτουργεί σαν εργοστάσιο για αντικείμενα ερωτημάτων (**Query**).
- **Entity:** Αυτά είναι τα διατηρούμενα αντικείμενα, που αποθηκεύονται ως εγγραφές στη βάση δεδομένων.
- **EntityTransaction:** Έχει ένα προς ένα σχέση με τον EntityManager. Για κάθε EntityManager οι λειτουργίες του συντηρούνται από μια κλάση τύπου EntityTransaction.
- **Persistence:** Αυτή η κλάση περιέχει στατικές μεθόδους για την παροχή αντικειμένου EntityManagerFactory.
- **Query:** Αυτή η διαπροσωπεία υλοποιείται από κάθε προμηθευτή JPA για την παροχή σχεσιακών αντικειμένων τα οποία πληρούν τα διάφορα criteria.

Στο παρακάτω διάγραμμα δίνονται οι συσχετίσεις μεταξύ των κλάσεων αυτών:

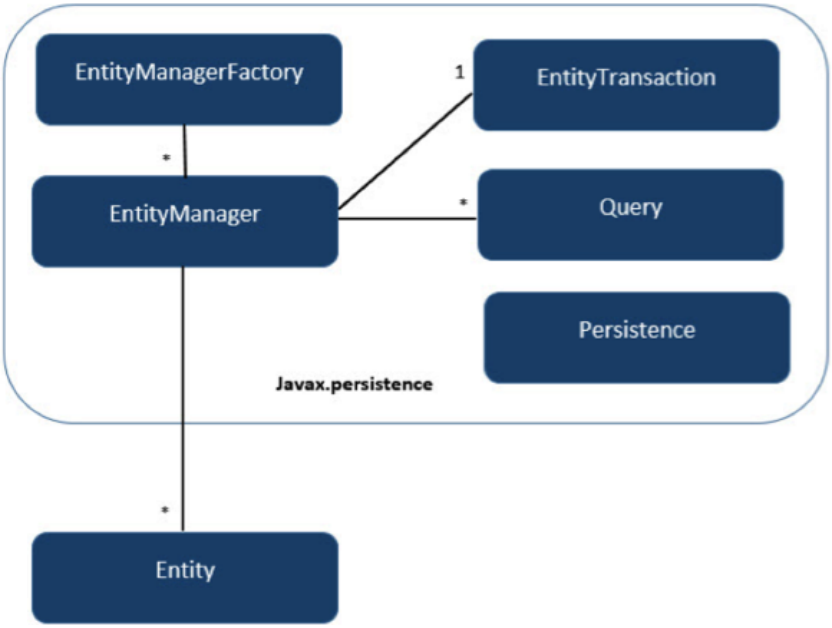

Figure 2: JPA Class Associations

Συσχέτιση Αντικειμένου-Σχέσης (Object Relational Mapping ORM)

Το βασικό χαρακτηριστικό του ORM είναι η συσχέτιση ή αλλιώς το δέσιμο ενός αντικειμένου και των αντίστοιχων δεδομένων του στη βάση δεδομένων. Κατά τη διάρκεια της συσχέτισης πρέπει να ληφθούν υπόψη τα δεδομένα, ο τύπος των δεδομένων και οι συσχετίσεις που έχουν οι πίνακες μεταξύ τους.
Παρακάτω δίνεται η αρχιτεκτονική του ORM σε ένα διάγραμμα.

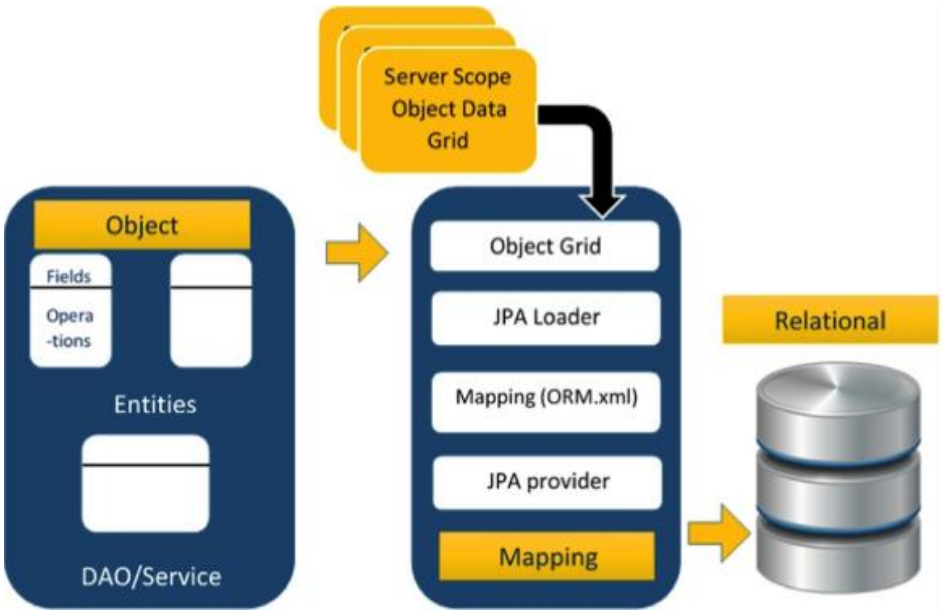

Figure 3: ORM Architecture

Στο παραπάνω διάγραμμα εξηγείται πως αποθηκεύονται δεδομένα αντικειμένων σε μια σχεσιακή βάση δεδομένων, σε τρία στάδια.

Στάδιο 1

Στο πρώτο στάδιο, ή αλλιώς και στάδιο αντικειμένων, περιέχονται κλάσεις POJO, διαπροσωπίες υπηρεσιών και κλάσεις. Αποτελεί το βασικό εμπορικό επίπεδο, το οποίο περιέχει λειτουργίες και ιδιότητες εμπορικής λογικής.

Στάδιο 2

Το δεύτερο στάδιο λέγεται και στάδιο συσχέτισης ή διατήρησης, και περιέχει παρόχους JPA (JPA providers), αρχεία συσχετίσεων (mapping files πχ ORM.xml), JPA Loaders και το πλέγμα αντικειμένων (object grid).

- **JPA Provider**: Το εμπορικό προϊόν που περιέχει λειτουργικότητα JPA. Παραδείγματα αυτών είναι το EclipseLing, Hibernate, Toplink κλπ.
- **Mapping File**: Το αρχείο συσχέτισης περιέχει ρυθμίσεις για συσχέτιση δεδομένων μεταξύ POJOs και δεδομένων σε μια σχεσιακή βάση.
- **JPA Loader**: Ο JPA Loader λειτουργεί ως μνήμη cache, που μπορεί να φορτώνει τοπικά δεδομένα πλέγματος. Λειτουργεί σαν ένα αντίγραφο της βάσης δεδομένων για την αλληλεπίδραση με κλάσεις υπηρεσιών για τα αρχεία POJO.
- **Object Grid**: Το πλέγμα αντικειμένου είναι μια προσωρινή τοποθεσία στην οποία μπορεί να αποθηκευτεί το αντίγραφο της βάσης. Όλα τα ερωτήματα που γίνονται στη βάση, πρώτα τροποποιούν το πλέγμα αντικειμένων και μόνο αφού παραδοθεί το πλέγμα επηρεάζεται η κυρίως βάση δεδομένων.

Στάδιο 3

Το τρίτο στάδιο αποτελεί το στάδιο των σχεσιακών δεδομένων. Περιέχει τα σχεσιακά δεδομένα που είναι λογικά συνδεδεμένα με το εμπορικό κομμάτι. Όπως εξηγήθηκε και παραπάνω, μόνο όταν το εμπορικό κομμάτι παραδώσει τα δεδομένα, αποθηκεύονται αυτά φυσικά στη βάση δεδομένων. Μέχρι τότε τα δεδομένα αποθηκεύονται στην cache με τη μορφή πλέγματος. Η ίδια διαδικασία εκτελείται και για την ανάκτηση δεδομένων.
Ο παραπάνω μηχανισμός ονομάζεται μηχανισμός συσχέτισης αντικειμένων και σχέσεων (object relational mapping).
Αξίζει να αναφερθεί το Mapping.xml το οποίο αναλαμβάνει να συσχετίσει τις κλάσεις (Οντότητες) με τους πίνακες της βάσης δεδομένων.

Αν για παράδειγμα η εφαρμογή έχει την παρακάτω Οντότητα:

```
public class Employee {

  private int eid;
  private String ename;
  private double salary;
  private String deg;

  public Employee(int eid, String ename, double salary, String deg) {
    super( );
    this.eid = eid;
```

```
    this.ename = ename;
    this.salary = salary;
    this.deg = deg;
  }

  public Employee( ) {
    super();
  }

  public int getEid( ) {
    return eid;
  }

  public void setEid(int eid) {
    this.eid = eid;
  }

  public String getEname( ) {
    return ename;
  }

  public void setEname(String ename) {
    this.ename = ename;
  }

  public double getSalary( ) {
    return salary;
  }

  public void setSalary(double salary) {
    this.salary = salary;
  }

  public String getDeg( ) {
    return deg;
  }

  public void setDeg(String deg) {
    this.deg = deg;
  }
}
```

Τότε το αντίστοιχο mapping.xml είναι το εξής:

```
<? xml version="1.0" encoding="UTF-8" ?>

<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
  http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
  version="1.0">

  <description> XML Mapping file</description>
```

```
  <entity class="Employee">
    <table name="EMPLOYEETABLE"/>
    <attributes>

      <id name="eid">
        <generated-value strategy="TABLE"/>
      </id>

      <basic name="ename">
        <column name="EMP_NAME" length="100"/>
      </basic>

      <basic name="salary">
      </basic>

      <basic name="deg">
      </basic>

    </attributes>
  </entity>

</entity-mappings>
```

Για μείωση του προγραμματιστικού φόρτου γίνεται χρήση annotations πάνω στην Οντότητα. Με τη χρήση των annotations αυτών, τα οποία εισάγονται πριν από τις κλάσεις, τις μεθόδους ή τα ορίσματα της οντότητας, παρέχεται πληροφορία για τη συσχέτιση μεταξύ Οντότητας και δεδομένων στη βάση.

Με τη χρήση των annotations το παραπάνω POJO γράφεται ως εξής:

```
package com.tutorialspoint.eclipselink.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table
public class Employee {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)

  private int eid;
  private String ename;
  private double salary;
  private String deg;

  public Employee(int eid, String ename, double salary, String deg) {
```

```
    super( );
    this.eid = eid;
    this.ename = ename;
    this.salary = salary;
    this.deg = deg;
  }

  public Employee( ) {
    super();
  }

  public int getEid( ) {
    return eid;
  }

  public void setEid(int eid) {
    this.eid = eid;
  }

  public String getEname( ) {
    return ename;
  }

  public void setEname(String ename) {
    this.ename = ename;
  }

  public double getSalary( ) {
    return salary;
  }

  public void setSalary(double salary) {
    this.salary = salary;
  }

  public String getDeg( ) {
    return deg;
  }

  public void setDeg(String deg) {
    this.deg = deg;
  }

  @Override
  public String toString() {
    return "Employee [eid=" + eid + ", ename=" + ename + ", salary=" + salary + ", deg=" + deg + "]";
  }
}
```

Τα annotations είναι τα κομμάτια κώδικα που ξεκινάνε με ‘@’ όπως για παράδειγμα το @Override. Εκτός από τους πίνακες πρέπει να οριστεί και η βάση δεδομένων που χρησιμοποιείται στην εφαρμογή. Αυτό επιτυγχάνεται με ένα άλλο xml το persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

  <persistence-unit name="Eclipselink_JPA" transaction-type="RESOURCE_LOCAL">

    <class>com.tutorialspoint.eclipselink.entity.Employee</class>

    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/jpadb"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.password" value="root"/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <property name="eclipselink.logging.level" value="FINE"/>
      <property name="eclipselink.ddl-generation" value="create-tables"/>
    </properties>

  </persistence-unit>
</persistence>
```

Σε συνοχή με την παραπάνω Οντότητα Employees και το persistence.xml δίνεται και η κλάση CreateEmployee η οποία χρησιμοποιείται για τη δημιουργία ενός Employee

```
package com.tutorialspoint.eclipselink.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import com.tutorialspoint.eclipselink.entity.Employee;

public class CreateEmployee {

  public static void main( String[ ] args ) {

    EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Eclipselink_JPA" );

    EntityManager entitymanager = emfactory.createEntityManager( );
    entitymanager.getTransaction( ).begin( );

    Employee employee = new Employee( );
    employee.setEid( 1201 );
    employee.setEname( "Gopal" );
    employee.setSalary( 40000 );
    employee.setDeg( "Technical Manager" );

    entitymanager.persist( employee );
    entitymanager.getTransaction( ).commit( );

    entitymanager.close( );
    emfactory.close( );
```

```
  }
}
```

Σε αυτή την κλάση δημιουργείται ένα εργοστάσιο EntityManagerFactory, στη συνέχεια μέσω του εργοστασίου ένας EntityManager και τέλος ανοίγεται μια συναλλαγή. Δημιουργείται ένα αντικείμενο Employee, προστίθενται δεδομένα μέσα σε αυτό και στη συνέχεια αποθηκεύεται η νέα εγγραφή στη βάση. Με αντίστοιχες διαδικασίες μπορεί να γίνει και αφαίρεση και επεξεργασία των εγγραφών που υπάρχουν στη βάση.
Οι παραπάνω διαδικασία μπορεί να μην είναι πάντα αρκετά κομψή λύση, ειδικά αν ο χρήστης θέλει να λάβει και να επεξεργαστεί δεδομένα που προέρχονται από πολύπλοκα ερωτήματα. Για αυτό το λόγο είναι δυνατή η προσομοίωση ή και δημιουργία ερωτημάτων μέσω του JPA. Το JPA παρέχει τις ακόλουθες μεθόδους για να θέτει ο χρήστης ερωτήματα με οντότητες:

- **Java Persistence Query Language (JPQL)**, μια απλή, βασιζόμενη σε συμβολοσειρές, γλώσσα παρόμοια με την SQL, η οποία χρησιμοποιείται για να θέτει κανείς ερωτήματα στις οντότητες και τις σχέσεις τους.
- Τα **Criteria API** χρησιμοποιούνται για τη δημιουργία τυπογραφικά ασφαλών ερωτημάτων. Περιλαμβάνουν χρήση java APIs προκειμένου να θέτουν ερωτήματα στις οντότητες και τις σχέσεις τους.

Και οι δύο μέθοδοι έχουν πλεονεκτήματα και μειονεκτήματα.

JPQL

Τα JPQL ερωτήματα, χρειάζονται λίγες γραμμές για να συνταχθούν και είναι τυπικά πιο κατανοητά και ευκολοδιάβαστα από τα Criteria ερωτήματα. Οι προγραμματιστές που γράφουν σε SQL δεν αντιμετωπίζουν δυσκολία στην εκμάθηση της JPQL. Τα JPQL ερωτήματα μπορούν να προσδιοριστούν και στην κλάση της οντότητας χρησιμοποιώντας ένα java annotation ή στον deployment descriptor της εφαρμογής. Παρόλα αυτά τα ερωτήματα σε JPQL δεν είναι τυπογραφικά ασφαλή και χρειάζονται cast όταν ο entity manager επιστρέφει το αποτέλεσμα του ερωτήματος. Αυτό σημαίνει ότι σφάλματα τύπου type-casting δεν μπορούν να εντοπιστούν κατά τη διάρκεια μεταγλώττισης. Ακόμα τα JPQL ερωτήματα δεν υποστηρίζουν παραμέτρους τύπου Open-ended.

Ένα πολύ απλό παράδειγμα χρήσης του JPQL είναι το παρακάτω:

```
package com.tutorialspoint.eclipselink.service;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

public class ScalarandAggregateFunctions {
  public static void main( String[ ] args ) {

    EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Eclipselink_JPA" );
    EntityManager entitymanager = emfactory.createEntityManager();
```

```
    //Scalar function
    Query query = entitymanager.createQuery("Select UPPER(e.ename) from Employee e");
    List<String> list = query.getResultList();

    for(String e:list) {
      System.out.println("Employee NAME :"+e);
    }

    //Aggregate function
    Query query1 = entitymanager.createQuery("Select MAX(e.salary) from Employee e");
    Double result = (Double) query1.getSingleResult();
    System.out.println("Max Employee Salary :" + result);
  }
}
```

Παρατηρούμε πως η σύνταξη του ερωτήματος είναι παρόμοια με αυτή της SQL.

```
"Select UPPER(e.ename) from Employee e"
```

Όπως αναφέρθηκε παραπάνω ο συντακτικός έλεγχος του ερωτήματος δεν είναι εφικτός και αυτή η μέθοδος μπορεί να οδηγήσει σε λάθη κατά την διάρκεια εκτέλεσης της εφαρμογής.
Εκτός από αυτή τη μέθοδο μπορεί ο χρήστης να εισάγει στην Οντότητα κάποιο Ονομασμένο Ερώτημα (Named Query)

```
. . .
@Entity
@Table
@NamedQuery(query = "Select e from Employee e where e.eid = :id", name = "find employee by id")

public class Employee {
  @Id
. . .
```

Το ερώτημα αυτό εισάγεται ως όρισμα στο annotation @NamedQuery και συμβάλει στο διαχωρισμό μεταξύ των JPQL ερωτημάτων και τον POJO.

Criteria API

Τα ερωτήματα των Criteria επιτρέπουν στον χρήστη τον προσδιορισμό ερωτημάτων στο εμπορικό επίπεδο της εφαρμογής. Αν και αυτό είναι δυνατό και με χρήση JPQL δυναμικών ερωτημάτων, τα ερωτήματα Criteria προσδίδουν καλύτερη απόδοση γιατί τα JPQL ερωτήματα πρέπει να επεξεργάζονται κάθε φορά που καλούνται. Τα Criteria ερωτήματα είναι τυπογραφικά ασφαλή και για αυτό δε χρειάζονται casting, σε αντίθεση με τα JPQL ερωτήματα. Το Criteria API είναι ένα Java API και ως εκ ’τούτου ο χρήστης του δε χρειάζεται να μάθει ακόμα μία γλώσσα ερωτημάτων. Τα Criteria είναι τυπικά πιο λεπτομερή από τα JPQL ερωτήματα, και απαιτούν από το χρήστη τη δημιουργία αρκετών αντικειμένων και την επεξεργασία των αντικειμένων αυτών, πριν τα παραδώσουν στον entity manager.

Ένα παράδειγμα χρήσης των κριτηρίων είναι το εξής:

```
EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Eclipselink_JPA" );
EntityManager entitymanager = emfactory.createEntityManager( );
CriteriaBuilder criteriaBuilder = entitymanager.getCriteriaBuilder();
CriteriaQuery<Object> criteriaQuery = criteriaBuilder.createQuery();
Root<Employee> from = criteriaQuery.from(Employee.class);

//select all records
System.out.println("Select all records");
CriteriaQuery<Object> select = criteriaQuery.select(from);
TypedQuery<Object> typedQuery = entitymanager.createQuery(select);
List<Object> resultlist = typedQuery.getResultList();
```

Σε πρώτη ματιά φαίνεται πως έχει πολύ πιο πολύπλοκη σύνταξη από αυτή των JPQL. Όμως δεν έχει καθόλου συμβολοσειρές, κάτι που το καθιστά πολύ πιο ασφαλές και λιγότερο επιρρεπές σε λάθη, σε αντίθεση με το JPQL.
Παρακάτω δίνονται μερικά παραδείγματα χρήσης των criteria με μεταμοντέλα.

Έστω ότι υπάρχει μια οντότητα Pet. Η αρχικοποίηση ενός ερωτήματος κριτηρίων γίνεται ως εξής:

```
EntityManager em = ...;
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
```

Στο ίδιο αποτέλεσμα καταλήγουμε με τη χρήση μεταμοντέλων ως εξής:

```
EntityManager em = ...;
Metamodel m = em.getMetamodel();
EntityType<Pet> Pet_ = m.entity(Pet.class);
Root<Pet> pet = cq.from(Pet_);
```

Είναι εφικτή και η συνένωση μεταξύ πινάκων:

```
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Metamodel m = em.getMetamodel();
EntityType<Pet> Pet_ = m.entity(Pet.class);
EntityType<Owner> Owner_ = m.entity(Owner.class);

Root<Pet> pet = cq.from(Pet.class);
Join<Owner, Address> address = cq.join(Pet_.owners).join(Owner_.addresses);
```

Είναι εφικτή ακόμα και η χρήση αλυσιδωτών συνενώσεων πινάκων.
Είναι δυνατή η χρήση ενός select λεκτικού προκειμένου να επιλεγούν συγκεκριμένα πεδία του πίνακα.

```
CriteriaQuery<String> cq = cb.createQuery(String.class);
Metamodel m = em.getMetamodel();
EntityType<Pet> Pet_ = m.entity(Pet.class);

Root<Pet> pet = cq.from(Pet.class);
cq.select(pet.get(Pet_.name));
```

Τέλος δίνεται ακόμα ένα παράδειγμα με την εισαγωγή ενός φίλτρου με χρήση μεταμοντέλων

```
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Metamodel m = em.getMetamodel();
EntityType<Pet> Pet_ = m.entity(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
cq.where(cb.equal(pet.get(Pet_.name)), "Fido");
```

Στα παραπάνω παραδείγματα δε γίνεται καθόλου χρήση συμβολοσειρών, με εξαίρεση την εισαγωγή τιμών στα πεδία του πίνακα ή κάποια τιμή που χρησιμοποιείται σε ένα φίλτρο. Είναι δυνατή η αντιστοίχηση μεταξύ SQL και Criteria αλλά λόγο της χρήσης των μεθόδων η κατανόησή τους είναι πιο επίπονη σε αντίθεση με τα JPQL ερωτήματα.
Θα δοθούν ακόμα μερικά παραδείγματα για τα Criteria API και το JPQL. Σε αυτά τα παραδείγματα δείχνεται η χρήση των criteria χωρίς τα αντίστοιχα μεταμοντέλα, ωστόσο ένας προγραμματιστής έχει τη δυνατότητα να χρησιμοποιήσει όποια μέθοδο προτιμάει.

```
JPQL
TypedQuery<String> query = em.createQuery( "SELECT c.name FROM Country AS c", String.class);
List<String> results = query.getResultList();
Criteria
CriteriaQuery<Country> q = cb.createQuery(Country.class);
Root<Country> c = q.from(Country.class);

JPQL
TypedQuery<Object[]> query = em.createQuery( "SELECT c.name, c.capital.name FROM Country AS c",
bject[].class);

Criteria
CriteriaQuery<Object[]> q = cb.createQuery(Object[].class);
Root<Country> c = q.from(Country.class);
q.select(cb.array(c.get("name"), c.get("capital").get("name")));

JPQL
SELECT c, p.name FROM Country c LEFT OUTER JOIN c.capital p

Criteria
CriteriaQuery<Country> q = cb.createQuery(Country.class);
Root<Country> c = q.from(Country.class);
Join<Country> p = c.join("capital", JoinType.LEFT);
q.multiselect(c, p.get("name"));
```

2.1.2 Hibernate

Το Hibernate προσφέρει συσχετίσεις μεταξύ κλάσεων της Java και πίνακες βάσεων δεδομένων. Ακόμα προφέρει ερωτήματα δεδομένων και μεθόδους ανάκτησης δεδομένων, μειώνοντας το χρόνο που χρειάζεται η ανάπτυξη λογισμικού, αν αυτή γινόταν με χειροκίνητο χειρισμό μέσω SQL και JDBC.
Σκοπός του Hibernate είναι η απαλλαγή από συχνά προγραμματιστικά θέματα που αφορούν διατήρηση δεδομένων. Το Hibernate δεν αποτελεί την καλύτερη λύση για εφαρμογές που χρησιμοποιούν πολλές stored procedures, και ως εκ 'τούτου η εμπορική τους λογική βρίσκεται στη βάση. Η χρησιμότητά της αναδεικνύεται σε αντικειμενοστραφείς εφαρμογές και μοντέλα, που έχουν την εμπορική λογική τους στο βασισμένο σε Java μέσο επίπεδο (middle tier). Ωστόσο το Hibernate βοηθάει στην ενθυλάκωση πολυχρησιμοποιούμενου κώδικα SQL και στην επεξεργασία των δεδομένων που επιστρέφουν τα ερωτήματα.
Οι κλάσεις που χρησιμοποιούνται μπορεί να είναι απλά JavaBeans, η κλασσική μορφή των οποίων είναι ιδιωτικές μεταβλητές/ιδιότητες και μέθοδοι getters, setters για την επεξεργασία τους.

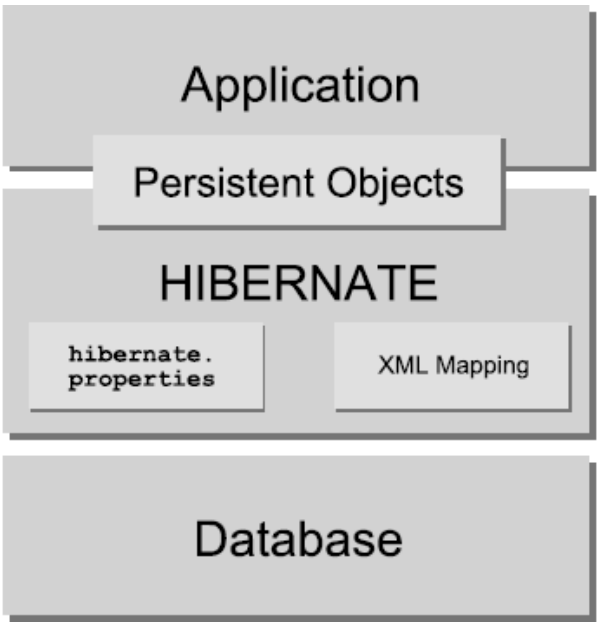

Figure 4: Hibernate Architecture (Simple)

Στο παραπάνω διάγραμμα παρουσιάζεται σε υψηλό επίπεδο η αρχιτεκτονική του Hibernate. Δείχνει τη λογική με την οποία προσφέρει persistent υπηρεσίες (και αντικείμενα) στην εφαρμογή. Παρακάτω παρουσιάζεται μια πιο αναλυτική αρχιτεκτονική. Χωρίζεται σε δύο τμήματα. Το "lite" το οποίο αφήνει την εφαρμογή να παρέχει το JDBC και να αναλάβει τις δικές της συναλλαγές και την "full cream " η οποία διαχωρίζει την εφαρμογή από τα JDBC/JTA APIs και αφήνει το Hibernate να αναλάβει τις λεπτομέρειες αυτές.

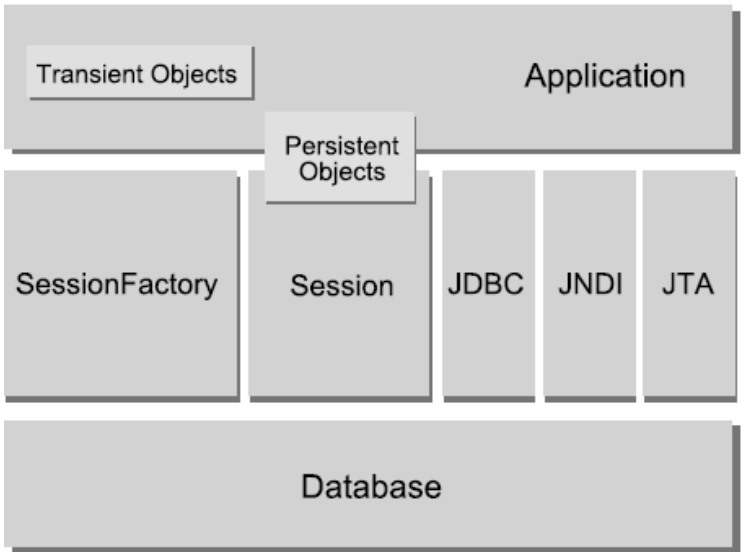

Figure 5: Hibernate Architecture (Lite Form)

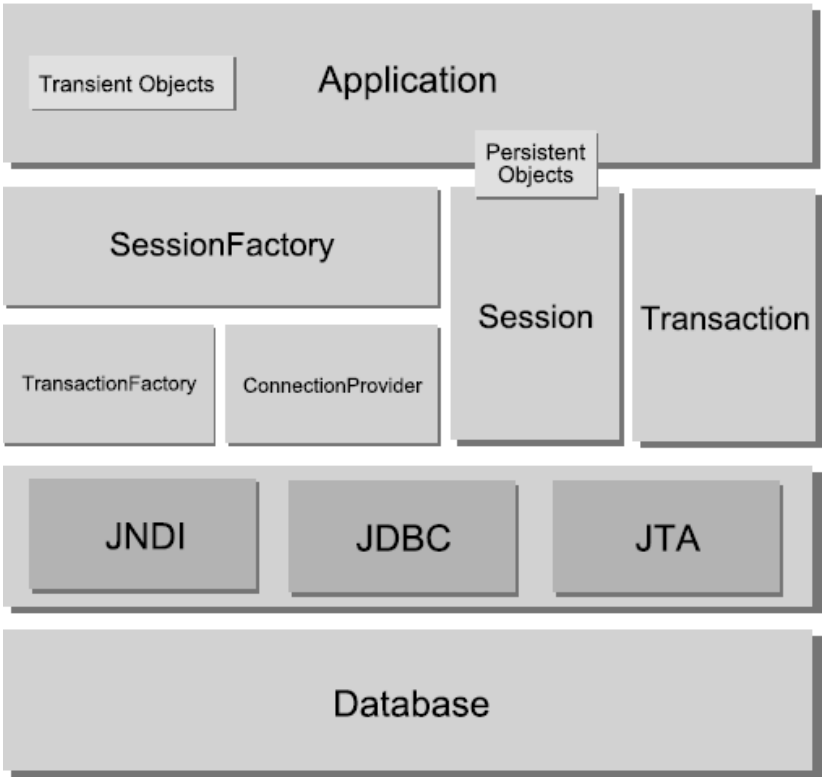

Figure 6: Hibernate Architecture (Full Cream Form)

Όπως και το JPA έτσι και το Hibernate έχει τη δυνατότητα επικοινωνίας με μια βάση είτε με χρήση criteria είτε με χρήση συμβολοσειρών. Σε περίπτωση που ο χρήστης θέλει να δουλέψει με συμβολοσειρές χρησιμοποιεί την HQL, μια γλώσσα παρόμοια με την JPQL. Όπως και η JPQL έτσι και η HQL έχει πολλά κοινά χαρακτηριστικά με την SQL και είναι εύκολη και γρήγορη στην εκμάθηση. Παρακάτω δίνονται παραδείγματα της HQL.

Αρχικά δίνεται ένα απλό ερώτημα τύπου SELECT.

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
List results = query.list();
```

Για πλήρες μονοπάτι της κλάσης Employee η συμβολοσειρά γράφεται ως εξής:

```
String hql = "FROM com.hibernatebook.criteria.Employee";
```

Είναι δυνατό να τεθεί και ψευδώνυμο στον πίνακα.

```
String hql = "FROM Employee AS E";
Query query = session.createQuery(hql);
List results = query.list();
```

Εναλλακτικά μπορεί να παραληφθεί το λεκτικό AS.

```
String hql = "FROM Employee E";
```

Για επιλογή συγκεκριμένων πεδίων του πίνακα εργαζόμαστε όπως παρακάτω:

```
String hql = "SELECT E.firstName FROM Employee E";
Query query = session.createQuery(hql);
List results = query.list();
```

Είναι δυνατή και η προσθήκη κάποιου φίλτρου:

```
String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";
Query query = session.createQuery(hql);
List results = query.list();
```

Είναι ακόμα εφικτή και η χρήση μεταβλητών:

```
String hql = "FROM Employee E WHERE E.id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("employee_id",10);
List results = query.list();
```

Ακόμα είναι δυνατή η χρήση εντολών UPDATE, DELETE και INSERT:

```
String hql = "UPDATE Employee set salary = :salary "  +  "WHERE id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("salary", 1000);
query.setParameter("employee_id", 10);
```

```
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

```
String hql = "DELETE FROM Employee "  + "WHERE id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("employee_id", 10);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

```
String hql = "INSERT INTO Employee(firstName, lastName, salary)"  +  "SELECT firstName, lastName, salary FROM
old_employee";
Query query = session.createQuery(hql);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

Το Hibernate προσφέρει τη δυνατότητα για χρήση criteria με δικές του μεθόδους. Τα criteria είναι πιο πολύπλοκος τρόπος δημιουργίας ενός ερωτήματος, όπως έχει εξηγηθεί και στην ανάλυση του JPA. Παρακάτω δίνονται παραδείγματα χρήσης του Hibernate Criteria.

Στο παρακάτω παράδειγμα δίνεται ένα απλό select του πίνακα Employee :

```
Criteria cr = session.createCriteria(Employee.class);
List results = cr.list();
```

Για την εισαγωγή κάποιου φίλτρου χρησιμοποιούνται Restrictions:

```
Criteria cr = session.createCriteria(Employee.class);
cr.add(Restrictions.eq("salary", 2000));
List results = cr.list();
```

Πολλαπλά φίλτρα εισάγονται με χρήση λογικών εκφράσεων:

```
Criteria cr = session.createCriteria(Employee.class);

Criterion salary = Restrictions.gt("salary", 2000);
Criterion name = Restrictions.ilike("firstNname","zara%");

// To get records matching with OR condistions
LogicalExpression orExp = Restrictions.or(salary, name);
cr.add( orExp );


// To get records matching with AND condistions
LogicalExpression andExp = Restrictions.and(salary, name);
cr.add( andExp );
```

```
List results = cr.list();
```

Στο επόμενο παράδειγμα προστίθεται και ταξινόμηση:

```
Criteria cr = session.createCriteria(Employee.class);
// To get records having salary more than 2000
cr.add(Restrictions.gt("salary", 2000));

// To sort records in descening order
crit.addOrder(Order.desc("salary"));

// To sort records in ascending order
crit.addOrder(Order.asc("salary"));

List results = cr.list();
```

Η δυνατότητα άντλησης πληροφορίας όπως αριθμό γραμμών, μέση τιμή κλπ. γίνεται μέσω projections. Ένα απλό παράδειγμα είναι το εξής:

```
Criteria cr = session.createCriteria(Employee.class);

// To get total row count.
cr.setProjection(Projections.rowCount());

// To get average of a property.
cr.setProjection(Projections.avg("salary"));

// To get distinct count of a property.
cr.setProjection(Projections.countDistinct("firstName"));
```

Εδώ ολοκληρώνεται η μικρή εισαγωγή στο Hibernate. Η διαφορά του JPA και του Hibernate είναι, ότι το JPA αποτελεί στην ουσία ένα πρότυπο για ανάπτυξη λογισμικού ORM λογικής, ενώ το Hibernate είναι μια υλοποίηση του προτύπου αυτού. Όπως φαίνεται το JPA μπορεί να χρησιμοποιηθεί αυτούσιο, αλλά το Hibernate απλοποιεί πολύ τη διαδικασία ανάπτυξης ερωτημάτων με αντικειμενοστραφής λογική, ειδικά όσον αφορά τη μέθοδο των criteria. Τέλος θα παρουσιάσουμε άλλη μια πλατφόρμα, το queryDsl.

2.1.3 QueryDsl

Το QueryDsl είναι μια πλατφόρμα που δίνει τη δυνατότητα για τη δημιουργία ερωτημάτων βασισμένα στην sql. Αντί να γίνεται γραφή ερωτημάτων με τη μορφή συμβολοσειρών ή η μετάφρασή τους σε XML αρχεία, μπορούν να κατασκευαστούν μέσω ενός API σαν το QueryDsl.

Σύμφωνα με το έγγραφο αναφοράς της πλατφόρμας τα πλεονεκτήματα χρήσης του API σε σχέση με απλές συμβολοσειρές είναι τα εξής:

- Αυτόματη συμπλήρωση κώδικα με τη χρήση κάποιου IDE
- Δεν επιτρέπονται σχεδόν καθόλου συντακτικά λάθος ερωτήματα
- Domain types και properties μπορούν να αναφερθούν με ασφάλεια

- Ανταποκρίνεται καλύτερα σε αλλαγές στα domain types.

Αυτή η πλατφόρμα δημιουργήθηκε με αρχικό σκοπό την ανάγκη για συντήρηση των ερωτημάτων τύπου HQL με τυπογραφικά ασφαλή τρόπο. Μη ασφαλής αναφορά σε domain types και ιδιότητες, μέσω απλών συμβολοσειρών ήταν ένας ακόμα λόγος.
Τα Backends που υποστηρίζει είναι HQL για hibernate, JPA, JDO, JDBC, Lucene, Hibernate Search, MongoDB, Collections και RDFBean.

Σε αυτή την παράγραφο θα ελέγξουμε τον τρόπο με τον οποίο το QueryDsl επεξεργάζεται ερωτήματα τύπου JPQL. Η γενική χρήση είναι η εξής:

Με χρήση ενός select ορίζει το τι θα επιστρέψει το ερώτημα. Με το from προστίθενται οι πόροι του ερωτήματος, δηλαδή οι πίνακες που χρησιμοποιούνται. Στη συνέχεια με κάποιο είδος συνένωσης, όπως innerJoin, join, leftJoin, rightjoin και on προστίθενται οι συνενώσεις μεταξύ των πινάκων. Για τις μεθόδους τύπου Join το πρώτο όρισμα είναι η πηγή της συνένωσης και το δεύτερο ο στόχος. Στη συνέχεια με where προστίθενται τα φίλτρα. Με το groupBy ορίζονται οι ομαδοποιήσεις που γίνονται, με having προστίθενται φίλτρα για τα ορίσματα groupBy και με OrderBy εισάγονται ταξινομήσεις. Τέλος με τα Limit, offset και restrict γίνεται η σελιδοποίηση του αποτελέσματος. Limit, για περιορισμό των αποτελεσμάτων, offset για το βήμα μεταξύ των γραμμών και restrict για τον καθορισμό των limit και offset με μία κλήση.

Ας υποθέσουμε πως έχουμε την ακόλουθη οντότητα:

```
@Entity
public class Customer {
	private String firstName;
	private String lastName;
	public String getFirstName() {
		return firstName;
	}
	public String getLastName() {
		return lastName;
	}
	public void setFirstName(String fn) {
		firstName = fn;
	}
	public void setLastName(String ln)[
		lastName = ln;
	}
}
```

Ο προεπιλεγμένος τρόπος για την αρχικοποίηση ενός αντικειμένου Customer είναι ο εξής:

```
QCustomer customer = QCustomer.customer;
```

Αν θέλει ο χρήστης να ορίσει τη δικιά του μεταβλητή customer μπορεί να κάνει και το εξής:

```
QCustomer customer = new QCustomer("myCustomer");
```

Για να γίνει χρήση του JPA API η αρχικοποίηση των αντικειμένων των ερωτημάτων έχουν την ακόλουθη μορφή:

```
// where entityManager is a JPA EntityManager
JPAQuery<?> query = new JPAQuery<Void>(entityManager);
```

Στην περίπτωση που γίνεται χρήση του Hibernate API εντολή αλλάζει ως εξής:

```
// where session is a Hibernate session
HibernateQuery<?> query = new HibernateQuery<Void>(session);
```

Στο επόμενο παράδειγμα επιστρέφεται ένας πελάτης με όνομα Bob:

```
QCustomer customer = QCustomer.customer;
Customer bob = queryFactory.selectFrom(customer)
.where(customer.firstName.eq("Bob"))
.fetchOne();
```

Στο παραπάνω παράδειγμα επιλέγεται το αντικείμενο customer και με ένα φίλτρο where απομονώνεται μια εγγραφή με όνομα Bob. Στη συνέχεια δίνει εντολή να φέρει μόνο μια εγγραφή. Για χρήση πολλαπλών πινάκων (πόρων) το ερώτημα γράφεται ως εξής:

```
QCustomer customer = QCustomer.customer;
QCompany company = QCompany.company;
query.from(customer, company);
```

και η χρήση πολλαπλών φίλτρων έχει αυτή τη μορφή:

```
queryFactory.selectFrom(customer)
.where(customer.firstName.eq("Bob"), customer.lastName.eq("Wilson"));
```

ή εναλλακτικά:

```
select customer from Customer as customer
where customer.firstName = "Bob" and customer.lastName = "Wilson"
```

Παράδειγμα χρήσης συνενώσεων δίνεται παρακάτω:

```
QCat cat = QCat.cat;
QCat mate = new QCat("mate");
QCat kitten = new QCat("kitten");
queryFactory.selectFrom(cat)
.innerJoin(cat.mate, mate)
.leftJoin(cat.kittens, kitten)
.fetch();
```

Το αντίστοιχο JPQL ερώτημα είναι το εξής:

```
select cat from Cat as cat
inner join cat.mate as mate
left outer join cat.kittens as kitten
```

Αν ο χρήστης θέλει να ταξινομήσει το αποτέλεσμα εργάζεται ως εξής:

```
QCustomer customer = QCustomer.customer;
queryFactory.selectFrom(customer)
.orderBy(customer.lastName.asc(), customer.firstName.desc())
.fetch();
JPQL
select customer from Customer as customer
order by customer.lastName asc, customer.firstName desc
```

και αν θελήσει να ομαδοποιήσει τα δεδομένα:

```
queryFactory.select(customer.lastName).from(customer)
.groupBy(customer.lastName)
.fetch();
JPQL
select customer.lastName
from Customer as customer
group by customer.lastName
```

Τέλος αναφέρεται και η δυνατότητα για σύνταξη ερωτημάτων διαγραφής και επεξεργασίας εγγραφών:

```
QCustomer customer = QCustomer.customer;
// delete all customers
queryFactory.delete(customer).execute();
// delete all customers with a level less than 3
queryFactory.delete(customer).where(customer.level.lt(3)).execute();
QCustomer customer = QCustomer.customer;
// rename customers named Bob to Bobby
queryFactory.update(customer).where(customer.name.eq("Bob"))
.set(customer.name, "Bobby")
.execute();
```

Εδώ ολοκληρώνεται η εισαγωγή σε μερικές επιτυχημένες υλοποιήσεις του προβλήματος. Κάθε μια υλοποίηση έχει τα δικά της χαρακτηριστικά παρόλο που όλες περιτριγυρίζονται από την ίδια κεντρική ιδέα, ανάπτυξη ερωτημάτων με αντικειμενοστραφή τρόπο. Στη συνέχεια εξετάζουμε τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής.

2.2 Εργαλεία

Σε αυτή την ενότητα παρουσιάζονται οι πιο σημαντικές τεχνολογίες που χρησιμοποιούνται.

2.2.1 Eclipse Modeling Framework (EMF)

Τα μοντέλα δεδομένων ή αλλιώς και domain models αναπαριστούν τα δεδομένα με τα οποία θέλει να εργαστεί κάποιος. Μια καλή και συνήθης τακτική είναι να φτιαχτεί το μοντέλο χωριστά από τη λογική της εφαρμογής και να παραχθεί κώδικας σύμφωνα με αυτό το μοντέλο για την εφαρμογή. Με λίγα λόγια χρειάζεται μια εφαρμογή που να έχει τη δυνατότητα να παράξει το μοντέλο ανεξάρτητα από την εφαρμογή.

Το EMF είναι μια ομάδα από plug-ins (επεκτάσεις) του eclipse που χρησιμοποιείται για την κατασκευή ενός μοντέλου δεδομένων και τη δημιουργία του απαραίτητου κώδικα για την αναπαράσταση του μοντέλου σε Java. Αρχικά κατασκευάζει ένα μεταμοντέλο του μοντέλου και μέσω του μεταμοντέλου κατασκευάζει το πραγματικό μοντέλο.

Μοντελοποίηση και προγραμματισμός

Το EMF έχει τη δυνατότητα να δημιουργήσει αποδοτικό κώδικα καθώς και να περιγράφει μοντέλα. Αντί να διαχωρίσει υψηλό επίπεδο προγραμματισμού/μοντελοποίησης και χαμηλό επίπεδο προγραμματισμού τα συνδυάζει.
Η μοντελοποίηση μιας εφαρμογής είναι σημαντική γιατί δίνει τη δυνατότητα περιγραφής της εφαρμογής πιο εύκολα από ότι ο κώδικας. Ακόμα προσφέρει έναν υψηλού επιπέδου τρόπο επικοινωνίας με την εφαρμογή και τροποποίησής της. Από την άλλη κάποιος που είναι εξοικειωμένος με χρήση μοντέλων μπορεί να βοηθηθεί μέσω του EMF για να κατανοήσει τη συσχέτιση μοντέλου και κώδικα προκειμένου να μπορέσει να επεξεργαστεί τον εκάστοτε κώδικα πιο αποτελεσματικά.

Java, XML και UML

Υπάρχουν τρεις βασικοί τρόποι για να περιγράψει κάποιος ένα μοντέλο δεδομένων. Μπορεί να το κάνει είτε με κώδικα σε Java απευθείας, είτε με κάποιο XML αρχείο ή τέλος με ένα διάγραμμα UML. Ένα απλό παράδειγμα σε κώδικα java είναι το παρακάτω:

```
public interface PurchaseOrder
{
	String getShipTo();
	void setShipTo(String value);

	String getBillTo();
	void setBillTo(String value);

	List getItems(); // List of Item
}
public interface Item
{
	String getProductName();
	void setProductName(String value);

	int getQuantity();
	void setQuantity(int value);

	float getPrice();
	void setPrice(float value);
}
```

Στο παραπάνω παράδειγμα υπάρχουν δύο οντότητες, η PurchaseOrder και το Item, οι οποίες διαχειρίζονται εντολές πωλήσεων και αντικείμενα των πωλήσεων αυτών. Το αντίστοιχο διάγραμμα σε UML είναι το εξής:

Figure 7: Purchase Order and Item Model Demo

Και αν θελήσει κάποιος να το περιγράψει με ένα XML αρχείο, τότε θα πάρει αυτή τη μορφή:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.example.com/SimplePO"
            xmlns:PO="http://www.example.com/SimplePO">

<xsd:complexType name="PurchaseOrder">
 <xsd:sequence>
            <xsd:element name="shipTo" type="xsd:string"/>
            <xsd:element name="billTo" type="xsd:string"/>
            <xsd:element name="items" type="PO:Item"
                        minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Item">
 <xsd:sequence>
            <xsd:element name="productName" type="xsd:string"/>
            <xsd:element name="quantity" type="xsd:int"/>
            <xsd:element name="price" type="xsd:float"/>
 </xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

Το EMF δίνει τη δυνατότητα στο χρήστη να εκφράσει το μοντέλο σε οποιαδήποτε μορφή θέλει και να δημιουργήσει αυτόματα τις άλλες δύο μορφές.

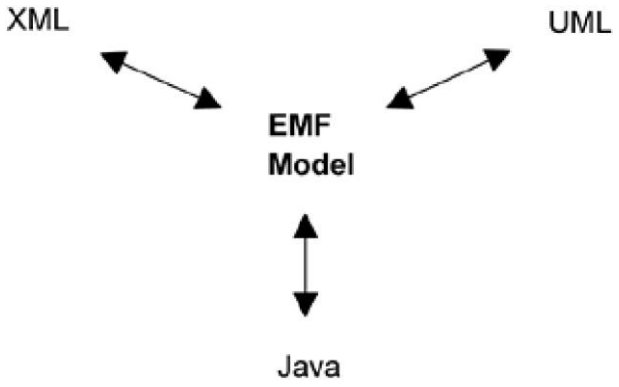

Figure 8: Connection between EMF and XML,UML,Java

Ορισμός Μοντέλου

Για την περιγραφή του μοντέλου χρησιμοποιούνται τα εξής στοιχεία:

- Ορισμοί κλάσεων για τα αντικείμενα PurchaseOrder και Item, ή στην περίπτωση της XML σύνθετοι τύποι ορισμού.
- Οι shipTo, billTo, productName, quantity, και price εκφράζονται ως μεταβλητές στη UML, ζευγάρια get()/set() στην Java και στην XML ως φωλιασμένα στοιχεία.
- Items, στη UML είναι μια αναφορά, στη Java Μια μέθοδος get() και στην XML ένα φωλιασμένο στοιχείο που δηλώνει αναφορά ενός πολύπλοκου τύπου.

Όπως φαίνεται το μοντέλο περιγράφεται χρησιμοποιώντας έννοιες πιο σύνθετες από απλές κλάσεις και μεθόδους. Οι Μεταβλητές (Attributes) για παράδειγμα αντιπροσωπεύουν ζευγάρια μεθόδων και έχουν και τη δυνατότητα να ενημερώνουν παρατηρητές (Observers) και να αποθηκεύονται ή να φορτώνονται από κάποιο persistent αποθηκευτικό μέσο (persistent storage). Οι αναφορές (references) είναι ακόμα πιο δυνατές καθώς μπορούν να είναι δύο κατευθύνσεων, και σε αυτή την περίπτωση διατηρείται η αναφορική ακεραιότητα του μοντέλου. Αυτές οι αναφορές μπορούν να διατηρούνται και μεταξύ πολλαπλών πόρων (εγγράφων πχ), όπου σε αυτή την περίπτωση γίνεται φόρτωση ανάλογα με τη ζήτηση που έχουν και ανάλυση τους μέσω proxies.

Προκειμένου όμως να οριστεί ένα μοντέλο που να φτιάχνει αυτά τα μοντέλα χρειάζεται κοινή ορολογία για να τα περιγράφει. Με λίγα λόγια χρειάζεται η ύπαρξη ενός μοντέλου που να περιγράφει τα EMF μοντέλα. Αυτό είναι ένα μεταμοντέλο.

Μεταμοντέλο Ecore

Το μεταμοντέλο που περιγράφει ένα EMF μοντέλο ονομάζεται Ecore. Το Ecore είναι ένα μοντέλο του EMF και κατά συνέπεια έχει ως μεταμοντέλο τον εαυτό του. Το Ecore είναι στην πραγματικότητα ένα απλοποιημένο υποσύνολο της UML. Παρακάτω δίνεται ένα απλοποιημένο υποσύνολο του Ecore μοντέλου.

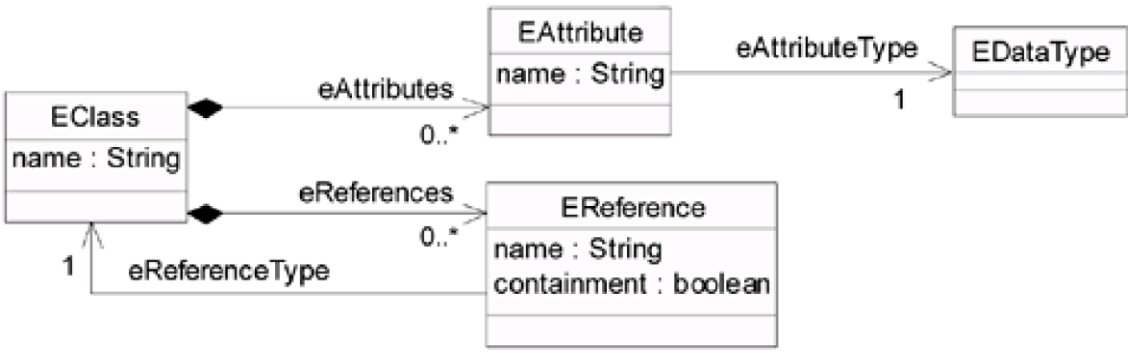

Figure 9: UML of Ecore Metamodel

Όπως φαίνεται υπάρχουν τρεις βασικές Ecore κλάσεις που αντιπροσωπεύουν το μοντέλο:

- **EClass** αντιστοιχεί σε μια μοντελοποιημένη κλάση. Περιέχει ένα όνομα, μηδέν ή περισσότερες μεταβλητές και μηδέν ή περισσότερες αναφορές.
- **EAttribute** αντιπροσωπεύει μια μοντελοποιημένη μεταβλητή. Οι μεταβλητές περιέχουν το όνομά τους και τον τύπο τους.
- **EReference** αντιστοιχεί σε μία συσχέτιση μεταξύ κλάσεων. Έχει ένα όνομα, μια σημαία Boolean για να δηλώνει αν είναι περιορισμός και μια αναφορά, η οποία είναι κάποια άλλη κλάση.
- **EDataType** αντιπροσωπεύει τον τύπο δεδομένων μιας μεταβλητής. Μπορεί να είναι πρωτόγονος τύπος ή κάποιο java αντικείμενο.

Με τις παραπάνω κλάσεις που προσδιορίζονται στο Ecore είναι δυνατή η περιγραφή της δομής μιας εφαρμογής.

Στο παράδειγμα που χρησιμοποιείται εδώ η δομή αυτή θα έχει τη μορφή:

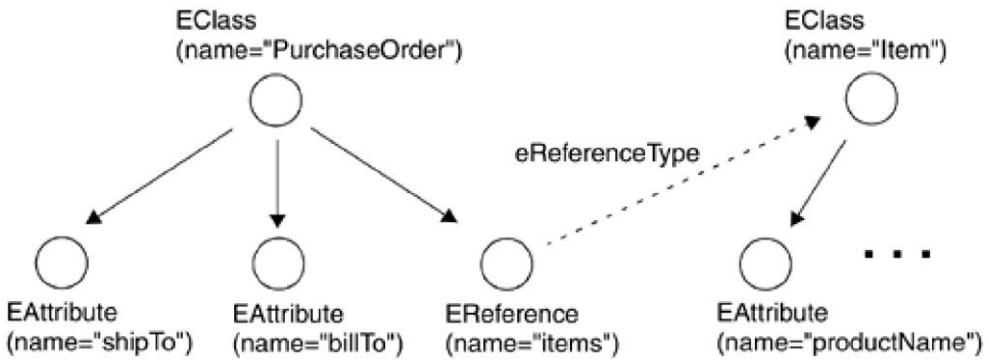

Figure 10: Structure of Demo Project

Δημιουργία και επεξεργασία του Μοντέλου

Αν ο χρήστης έχει διαπροσωπίες Java το EMF θα τις εξετάσει και θα δημιουργήσει το μοντέλο. Αν υπάρχει κάποιο XML αρχείο, τότε θα δημιουργηθεί από αυτό. Στην περίπτωση που κάποιος δουλεύει με ένα UML υπάρχουν τρείς πιθανότητες.

1. **Απευθείας Επεξεργασία Ecore**
2. **Εισαγωγή από UML**
3. **Εξαγωγή από UML**

Η πρώτη επιλογή είναι η ποιο επιθυμητή. Με αυτήν δεν υπάρχει βήμα Εισαγωγής/Εξαγωγής στη διαδικασία της υλοποίησης. Απλά γίνεται επεξεργασία και παραγωγή του μοντέλου. Ακόμα δε χρειάζεται να ανησυχούμε για το αν το μοντέλο είναι συντονισμένο με το έμφυτο μοντέλο του εργαλείου που χρησιμοποιούμε. Οι άλλες δύο μέθοδοι χρειάζονται καινούρια εισαγωγή ή εξαγωγή κάθε φορά που αλλάζει το μοντέλο. Τα πλεονεκτήματα των άλλων δύο μεθόδων όμως είναι, το ότι μπορεί να χρησιμοποιηθεί το εκάστοτε εργαλείο UML για περισσότερες εργασίες εκτός από μοντελοποίηση με το EMF.

Σειριοποίηση με XMI

Το XMI ή αλλιώς XML Metadata Interchange είναι μια μέθοδος σειριοποίησης μεταδεδομένων, όπως είναι και το Ecore. Εκτός από τον κώδικα σε Java για την αναπαράσταση του μοντέλου, όλες οι άλλες μορφές είναι προαιρετικές. Σε περίπτωση που γινόταν χρήση που κώδικα για την αναπαράσταση του μοντέλου, θα χρειαζόταν να γίνει ενδοσκοπική ανάλυση στον κώδικα κάθε φορά που θέλαμε να αναπαράγουμε το μοντέλο, κάτι το οποίο δεν είναι αποδοτικό.
Για αυτό το λόγο το XMI είναι μια λογική επιλογή για την κανονική μορφή του Ecore. Το πρόβλημα είναι πως κάθε εργαλείο UML έχει τη δικιά της persistent μορφή μοντέλου. Ένα αρχείο XMI αποτελεί την 'Standard' XML σειριοποίηση ακριβώς των μεταδεδομένων που χρησιμοποιεί το EMF.

Αν το παράδειγμα που χρησιμοποιούμε σειριοποιηθεί με XMI δείχνει κάπως έτσι:

```
<?xml version="1.0" encoding="ASCII"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
        name="po" nsURI="http:///com/example/po.ecore"
        nsPrefix="com.example.po">
<eClassifiers xsi:type="ecore:EClass" name="PurchaseOrder">
 <eReferences name="items"
   eType="#//Item" upperBound="-1" containment="true"/>
 <eAttributes name="shipTo"
   eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EString"/>
<eAttributes name="billTo"
   eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EString"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="Item">
<eAttributes name="productName"
  eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EString"/>
<eAttributes name="quantity"
  eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
<eAttributes name="price"
```

```
  eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EFloat"/>
</eClassifiers>
</ecore:EPackage>
```

Annotation για το EMF

Όπως ειπώθηκε παραπάνω όταν το μοντέλο δημιουργείται από διαπροσωπίες Java το EMF τις εξετάζει και θα παράγει τα στοιχεία του μοντέλου. Όμως δε θεωρεί πως όλες οι διαπροσωπίες και μέθοδοι αποτελούν στοιχεία του μοντέλου. Ο λόγος για αυτό είναι το ότι η γεννήτρια του EMF υποστηρίζει συνένωση κώδικα. Παράγει κώδικα που επιβάλλεται να συνδυάζεται με κώδικα που έχει γράψει ο χρήστης.
Προκειμένου μια μέθοδος να θεωρηθεί πως ανήκει στο μοντέλο πρέπει να περιέχει στα Javadoc σχόλια το annotation @model.

```
/**
* @model
*/
public interface PurchaseOrder
{
        /**
        * @model
        */
        String getShipTo();
        /**
        * @model
        */
        String getBillTo();
        /**
        * @model type="Item" containment="true"
        */
        List getItems();
}
```

Σε αυτή την περίπτωση το @model αναγνωρίζει τη διαπροσωπεία PurchaseOrder ως μια κλάση του μοντέλου με δύο μεταβλητές, την shipTo, και billTo και μια αναφορά, την items. Και οι δύο μεταβλητές έχουν όλες τις πληροφορίες τους διαθέσιμες με ενδοσκοπική αναζήτηση της Java, όπου σε αυτή την περίπτωση είναι μεταβλητές τύπου String.

Υπάρχουν και μερικές μη προκαθορισμένες πληροφορίες του μοντέλου που χρειάζονται για την αναφορά items. Επειδή η αναφορά είναι πολλαπλότητας many, κάτι που το αντιλαμβανόμαστε από το γεγονός πως επιστρέφει μια λίστα, πρέπει να προσδιορίσουμε και τον τύπο του στόχου της αναφοράς. Ακόμα πρέπει να ορίσουμε το containment αληθές για να προσδιορίσουμε το ότι οι εντολές πωλήσεων είναι δοχείο για τα items και να τα σειριοποιήσει ως παιδιά.

Οι μέθοδοι setShipTo() και setBillTo() δε χρειάζεται να εγκλειστούν με κάποιο annotation. Λόγω του ότι οι μέθοδοι get περιέχουν το annotation, δε χρειάζεται και στις set. Όταν προσδιοριστούν οι μεταβλητές, οι μέθοδοι set θα παραχθούν και προστεθούν στη διαπροσωπεία, αν δεν είναι ακόμα εκεί.

Συνοπτικά έχουμε τα εξής για το EMF:

- Το Ecore και η σειριοποίηση XMI είναι το επίκεντρο του EMF
- Ένα βασικό μοντέλο μπορεί να δημιουργηθεί από τουλάχιστον τρεις πηγές: UML Μοντέλο, XML Schema ή διαπροσωπίες Java με annotations.
- Κώδικας υλοποίησης σε Java και προαιρετικά άλλες μορφές του μοντέλου μπορούν να δημιουργηθούν από το βασικό μοντέλο.

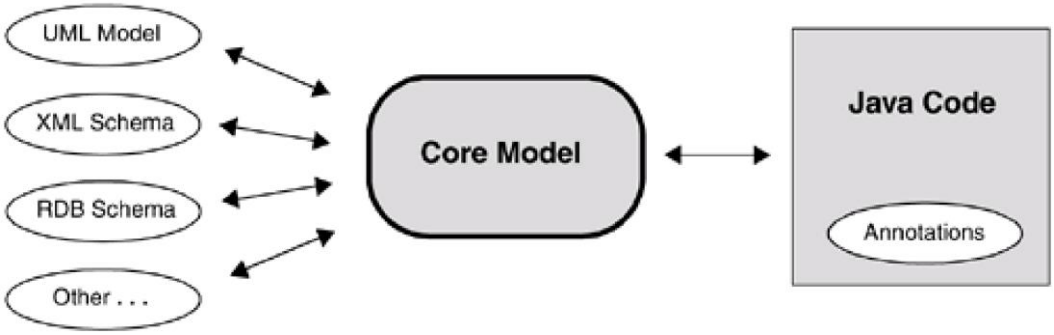

Figure 11: Basic Flow Diagram of EMF

Παραγωγή Κώδικα

Η Ecore κλάση (η οποία είναι μια EClass) αντιστοιχεί σε δύο πράγματα στην Java, μια διαπροσωπεία και την αντίστοιχή της υλοποίηση. Για παράδειγμα η EClass για τα PurchaseOrder αντιστοιχεί στη διαπροσωπεία,

```
public interface PurchaseOrder ...
```

και την αντίστοιχη κλάση υλοποίησης

```
public class PurchaseOrderImpl extends ... implements PurchaseOrder {
```

Ο διαχωρισμός μεταξύ διαπροσωπείας και υλοποίησης είναι μια σχεδιαστική απόφαση του EMF. Ο λόγος που χρησιμοποιείται είναι επειδή ένα τέτοιο μοτίβο πρέπει να ακολουθείται από οποιοδήποτε καλό API μοντελοποίησης. Αυτό το μοτίβο ακολουθείται από το DOM και σε μεγάλο βαθμό και από το Eclipse. Ακόμα είναι ένα απαραίτητο μοτίβο, προκειμένου να υποστηρίζεται πολλαπλή κληρονομικότητα στην Java.
Το επόμενο που παρατηρείται είναι πως η δημιουργημένη διαπροσωπεία επεκτείνει έμμεσα ή άμεσα τη βασική διαπροσωπεία EObject.

```
public interface PurchaseOrder extends EObject {
```

Το EObject είναι το αντίστοιχο java.lang.Object και αποτελεί τη βάση για όλα τα αντικείμενα του μοντέλου. Επέκταση από το EObject εισάγει τρεις βασικές συμπεριφορές:

1. **eClass** επιστρέφει το μετα-αντικείμενο του αντικειμένου (μια EClass).
2. **eContainer()** και **eResource()** επιστρέφουν το εμπεριεχόμενο αντικείμενο και πόρο του αντικειμένου.

3. **eSet(), eGet(), elsSet(), και eUnset()** παρέχουν ένα API για την πρόσβαση στα αντικείμενα.

Εκτός από τα παραπάνω το EObject επεκτείνει ένα ακόμα interface:

```
public interface EObject extends Notifier {
```

Η διαπροσωπεία του Notifier είναι αρκετά μικρή, αλλά εισάγει ένα σημαντικό χαρακτηριστικό σε κάθε αντικείμενο του μοντέλου, ειδοποιήσεις για αλλαγές στο μοντέλο, όπως στο Μοτίβο Σχεδιασμού Παρατηρητών. Όπως η διατηρησιμότητα των αντικειμένων, έτσι και η δυνατότητα ειδοποιήσεων είναι ένα σημαντικό χαρακτηριστικό ενός EMF αντικειμένου.
Όσον αφορά τις παραγμένες μεθόδους, η ακριβής πατέντα για κάθε χαρακτηριστικό της υλοποίησης (μεταβλητή ή αναφορά για παράδειγμα) εξαρτάται από τον τύπο και άλλες ιδιότητες που έχουν εισαχθεί από τον χρήστη. Γενικά όμως η μορφή των μεθόδων είναι αυτή που θα περίμενε ο κάθε χρήστης.

Η μέθοδος get() για παράδειγμα υλοποιείται ως εξής:

```
public String getShipTo() {
        return shipTo;
}
```

Η μέθοδος set() δίνει τιμή στην αντίστοιχη μεταβλητή αλλά ταυτόχρονα στέλνει ειδοποιήσεις και σε οποιονδήποτε παρατηρητή που ενδιαφέρεται για την αλλαγή της κατάστασης της μεταβλητής:

```
public void setShipTo(String newShipTo) {
        String oldShipTo = shipTo;
        shipTo = newShipTo;
        if (eNotificationRequired())
                eNotify(new ENotificationImpl(this,
                        Notification.SET,
                        POPackage.PURCHASE_ORDER__SHIP_TO,
                        oldShipTo, shipTo));
}
```

Σε περίπτωση που δεν υπάρχουν παρατηρητές η κλήση της ακριβής eNotify αποφεύγεται με τον φρουρό eNotificationRequired.

Σε γενικές γραμμές ο κώδικας διατηρείται απλός, αποδοτικός και καθαρός. Το EMF δεν είναι βαρύ και κατά συνέπεια ούτε και τα αντικείμενα που δημιουργούνται από/για το μοντέλο. Η βασική ιδέα είναι πως το EMF παράγει κώδικα που θα έγραφε ο χρήστης από μόνος του, αν τον έγραφε με το χέρι.
Πρέπει να αναφερθούν και άλλες δύο κλάσεις που παράγονται για το μοντέλο: ένα εργοστάσιο (factory) και ένα πακέτο (package). Το παραγόμενο εργοστάσιο περιλαμβάνει μια μέθοδο δημιουργίας για κάθε κλάση του μοντέλου. Ο προγραμματισμός με EMF προτρέπει τη χρήση εργοστασίων για την παρασκευή των αντικειμένων, αλλά δεν το επιβάλει.

Η προτεινόμενη χρήση είναι η εξής:

```
PurchaseOrder aPurchaseOrder = POFactory.eINSTANCE.createPurchaseOrder();
```

Το παραγόμενο πακέτο παρέχει βολικά όργανα εκτίμησης για όλα τα Ecore μεταδεδομένα για το μοντέλο.

Λοιπά παραγόμενα Δεδομένα

Εκτός από τις διαπροσωπείες και τις κλάσεις που αναφέρθηκαν παραπάνω η μηχανή του EMF μπορεί να παράγει και τα εξής:

1. Το σκελετό μιας κλάσης εργοστασίου προσαρμογέα (adapter factory) για το μοντέλο. Αυτή η κλάση μπορεί να χρησιμοποιηθεί για την υλοποίηση εργοστασίων προσαρμογής, που χρειάζονται για τη δημιουργία ειδικών προσαρμογέων. Παραδείγματα αυτού είναι PurchaseOrderAdapter για τα PurchaseOrders, ItemAdapter για τα Items κλπ.
2. Μια κλάση τύπου switch που υλοποιεί έναν μηχανισμό τύπου “switch” ανάλογα με τον τύπο του αντικειμένου. Η κλάση του εργοστασίου προσαρμογής που περιεγράφηκε παραπάνω χρησιμοποιεί αυτή την κλάση εναλλαγής στην υλοποίησή του.
3. Ένα μανιφέστο της επέκτασης (plug-in manifest) προκειμένου το μοντέλο να χρησιμοποιηθεί ως επέκταση του Eclipse.
4. Ένα XML σχήμα για το μοντέλο.

Επαναδημιουργία και Συνένωση

Το EMF παράγει αρχεία που αποτελούν συνδυασμό από παραγμένα κομμάτια κώδικα και κώδικα γραμμένο με το χέρι. Ο χρήστης στις περισσότερες περιπτώσεις επεξεργάζεται τον κώδικα και εισάγει κλάσεις και μεταβλητές. Σε κάθε περίπτωση έχει τη δυνατότητα να αναδημιουργήσει το μοντέλο και οι αλλαγές να διατηρηθούν.
Το EMF χρησιμοποιεί το annotation @generated στα σχόλια των δημιουργημένων διαπροσωπειών, κλάσεων, μεθόδων και πεδίων για τον προσδιορισμό των στοιχείων που δημιουργούνται.

Για παράδειγμα η getShipTo() έχει τη μορφή:

```
/**
* @generated
*/
public String getShipTo() { ...
```

Κάθε μέθοδος και στοιχείο γενικά που δεν περιέχει το @generated δεν λαμβάνεται υπόψη από το EMF και τη δημιουργία κώδικα.
Με τα παραπάνω ολοκληρώνεται μια μικρή εισαγωγή στο EMF. Υπάρχουν ακόμα αρκετά θέματα και αντικείμενα που πραγματεύεται το EMF αλλά δεν έχουν χρησιμοποιηθεί στην παρούσα διπλωματική. Η χρήση του EMF περιορίστηκε στη δημιουργία του μοντέλου, αλλά δεν έγινε πλήρης αξιοποίηση όλων των δυνατοτήτων του.

2.2.2 EMF Jet

Το Eclipse Modeling Framework (EMF) περιέχει δύο εργαλεία για την παραγωγή κώδικα, το Jet (Java Emitter Templates) και το JMerge (Java Merge). Το Jet χρησιμοποιεί σύνταξη παρόμοια με αυτή του JSP και δίνει ευκολία στη δημιουργία προτύπων που εκφράζουν τον κώδικα της εκάστοτε

εφαρμογής. Το Jet είναι γενική μηχανή που μπορεί να χρησιμοποιηθεί για την παραγωγή SQL, XML, πηγαίου κώδικα Java και άλλα κείμενα.
Το Jet χρησιμοποιήθηκε για να παραχθεί ο κώδικας των μεταμοντέλων αυτής της επέκτασης. Παρακάτω δίνεται ένα παράδειγμα χρήσης του jet για να εξοικειωθεί ο αναγνώστης με τη λειτουργία του.
Ένα jet πρότυπο είναι ένα αρχείο κειμένου που τελειώνει με το jet. Για παράδειγμα ένα .javajet αρχείο δημιουργεί ένα αρχείο java ενώ ένα .xmljet δημιουργεί ένα αρχείο xml.
Αφού ο χρήστης δημιουργήσει την εφαρμογή του, μέσω του Package Explorer ή του Hierarchy επιλέγει:

```
New > Other... > Java Emitter Templates > Convert Projects to JET Projects
```

και μετατρέπει την εφαρμογή σε μια jet εφαρμογή. Ο wizard προσθέτει λειτουργικότητα jet στην εφαρμογή του. Ακόμα εισάγεται ένας JET builder στην εφαρμογή, που αυτόματα μεταφράζει κάθε αρχείο στον φάκελο που ονομάζεται “templates” και τελειώνει με jet σε μια κλάση java.

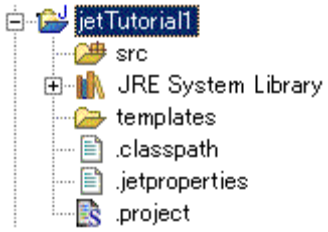

Figure 12: JET Tutorial Project

Αν πατήσει πάνω στην εφαρμογή και με δεξί κλικ πάει στην επιλογή ιδιότητες και JET Settings μπορεί να αλλάξει ρυθμίσεις, όπως ποιος φάκελος περιέχει τα πρότυπα jet και σε ποιον φάκελο θα αποθηκευτούν τα παραγόμενα αρχεία.

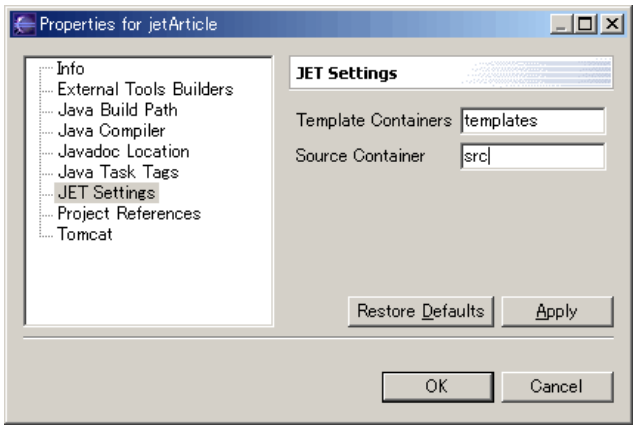

Figure 13: JET Settings

Στη συνέχεια μέσα στον φάκελο templates, η όποιον φάκελο έχει επιλέξει ο χρήστης για τη δημιουργία του κώδικα φτιάχνει ένα καινούριο αρχείο helloworld.txtjet. Αν εμφανίσει κάποιο μήνυμα λάθους αγνοείται σε πρώτη φάση γιατί πολύ απλά δεν έχουμε εισάγει τίποτα μέσα στο νέο αρχείο, αλλά ο builder πάει να το μεταφράσει με το που δημιουργήθηκε.

Μέσα στο νέο αρχείο προσθέτουμε το:

```
<%@ jet package="hello" class="HelloWorldTemplate" %>

 Hello, world!
```

Με την αποθήκευση του αρχείου γίνεται απευθείας μετάφραση από τον builder και δημιουργείται το αρχείο HelloWorldTemplate.java :

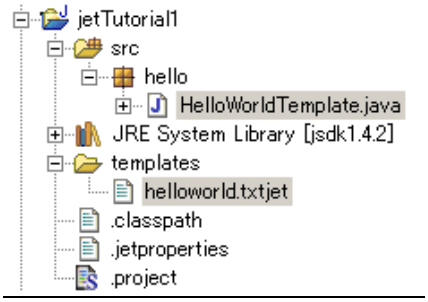

Figure 14: Tutorial Project with JET functionality

Στη συνέχεια μπορούμε να δούμε το αποτέλεσμα που εκτυπώνει η παραπάνω κλάση αν καλέσουμε τις εντολές:

```
HelloWorldTemplate helloworld = new HelloWorldTemplate();
String result = helloworld.generate(null);
System.out.println(result);
```

Με αυτό το απλό παράδειγμα γίνεται μια εισαγωγή στον αναγνώστη για το EMF JET. Περισσότερες λεπτομέρειες δίνονται σε επόμενη ενότητα, όπου και εξηγείται η χρήση του JET για την παραγωγή των μεταμοντέλων.

2.2.3 JavaBeans

Ο γενικός ορισμός ενός JavaBean είναι ο εξής:

« Ένα JavaBean είναι ένα επαναχρησιμοποιούμενο στοιχείο από κώδικα το οποίο μπορεί να χειραγωγείται οπτικά σε ένα κατασκευαστικό εργαλείο.»

Υπάρχουν πολλά είδη από JavaBeans. Αυτά περιλαμβάνουν JavaBeans που αποτελούν απλά GUI (Graphic User Interface) στοιχεία, όπως για παράδειγμα ένα κουμπί. Σε άλλες περιπτώσεις μπορούν να αποτελούν εξελιγμένα οπτικά στοιχεία όπως database viewers. Τέλος κάποια άλλα JavaBeans δεν έχουν δικιά τους μορφή GUI αλλά μπορούν να συντεθούν οπτικά με τη χρήση κάποιου κατασκευαστή (builder tool). Σε αυτή την εφαρμογή ασχολούμαστε με απλά JavaBeans που δεν έχουν γραφική απεικόνιση και για αυτό δεν θα μπούμε σε λεπτομέρειες για γραφικά στοιχεία και κατασκευαστές JavaBeans, αλλά θα δοθεί μια αναφορά στη σύσταση ενός JavaBean.

Τα πιο σημαντικά χαρακτηριστικά που περιέχει ένα JavaBean είναι ένα σύνολο από ιδιότητες που το χαρακτηρίζουν, ένα σύνολο από μεθόδους που επιτρέπουν άλλα στοιχεία να το καλέσουν και ένα σύνολο από γεγονότα που πυροδοτεί.

Οι ιδιότητες αυτές είναι στην ουσία τα χαρακτηριστικά του JavaBean, στα οποία έχουν πρόσβαση άλλα κομμάτια κώδικα μέσω των μεθόδων του JavaBean. Συνήθως έχουν ιδιωτική εμβέλεια και ο πιο ορθός τρόπος επεξεργασίας τους είναι μέσω των μεθόδων του JavaBean.

Οι μέθοδοι είναι απλές συναρτήσεις της Java, με δημόσια εμβέλεια συνήθως, προκειμένου να τις καλούν άλλα στοιχεία ή κομμάτια κώδικα. Για κάθε ιδιότητα που ορίζεται στο JavaBean ορίζονται και οι αντίστοιχες μεθόδοι getters, setters της ιδιότητας αυτής προκειμένου να μπορέσουν να την επεξεργαστούν άλλα τμήματα κώδικα.

Τα γεγονότα είναι η μέθοδος που χρησιμοποιείται για να ειδοποιεί ένα στοιχείο τα υπόλοιπα στοιχεία της εφαρμογής για κάποια αλλαγή που έγινε. Στα πλαίσια της εφαρμογής αυτής μέθοδοι υλοποίησης των γεγονότων δεν είναι απαραίτητοι, οπότε και δεν θα αναφερθούν περαιτέρω.

Τα JavaBeans με τα οποία θα ασχοληθούμε δεν διαφέρουν από μια κλάση της Java πολύ, απλά ακολουθούν κάποιες συμβάσεις. Παραδείγματα από JavaBeans (αναφέρονται και ως οντότητες) δίνονται σε πολλές ενότητες του εγγράφου.

2.2.4 Reflection (ενδοσκόπηση)

Ενδοσκόπηση είναι η ικανότητα ενός προγράμματος να ελέγχει τον εαυτό του και το λογισμικό του περιβάλλον και να αλλάζει τη συμπεριφορά του ανάλογα με το αποτέλεσμα του ελέγχου. Ο βασικός αλγόριθμος λειτουργίας της ενδοσκόπησης είναι κάπως έτσι:

1. Έλεγχος προγράμματος για τη δομή και τα δεδομένα του.
2. Λήψη αποφάσεων ανάλογα με τα ευρήματα του ελέγχου.
3. Αλλαγή της συμπεριφοράς, της δομής ή των δεδομένων του προγράμματος ανάλογα με τις αποφάσεις που πάρθηκαν.

Προκειμένου να εξεταστεί το πρόγραμμα πρέπει να έχει μια αναπαράσταση του ίδιου του του εαυτού. Αυτή η πληροφορία ονομάζεται μεταδεδομένο. Στον αντικειμενοστραφή κόσμο τα μεταδεδομένα οργανώνονται σε αντικείμενα, τα οποία ονομάζουμε μετα-αντικείμενα. Η αυτό-εξέταση κατά τη διάρκεια λειτουργίας του προγράμματος ονομάζεται ενδοσκόπηση.

Σε γενικές γραμμές υπάρχουν τρεις τεχνικές με τις οποίες μπορεί ένα API ενδοσκόπησης να αλλάξει τη συμπεριφορά του προγράμματος:

- Απευθείας τροποποίηση των μετα-αντικειμένων
- Λειτουργίες για χρήση μεταδεδομένων (όπως δυναμική επίκληση μεθόδων)
- Ενδοσκόπηση, κατά την οποία ο κώδικας μπορεί να μεσολαβήσει σε πολλά στάδια της εκτέλεσης του προγράμματος

Αν ο χρήστης δεν είναι προσεκτικός μπορεί να προκύψουν προβλήματα στα εξής πεδία

- Ασφάλεια
- Πολυπλοκότητα κώδικα
- Απόδοση εκτέλεσης

Για θέματα ασφάλειας η Java είναι σχεδιασμένη ορθά και τα API ενδοσκόπησης έχουν αρκετούς περιορισμούς, έτσι ώστε η ασφάλεια ενός προγράμματος να μπορεί να ελεγχθεί με ευκολία. Η

πολυπλοκότητα κώδικα αποτελεί πρόβλημα για κάποιον ερασιτέχνη. Γνωρίζοντας πότε να χρησιμοποιηθεί ενδοσκόπηση και πότε όχι μπορεί να αποφευχθεί αχρείαστος πολύπλοκος κώδικας. Τέλος η αποδοτικότητα ενός προγράμματος είναι θέμα εμπειρίας και όπως και με την πολυπλοκότητα του κώδικα, σωστή χρήση του API και σωστή εκτίμηση της απόδοσης κατά το σχεδιασμό του κώδικα μπορεί να ικανοποιήσει τις απαιτήσεις επιδόσεων.

2.2.5 Generics

Τα generics επιτρέπουν αφαιρετικότητα πάνω σε τύπους. Ένα παράδειγμα προσδιορισμού τύπων χωρίς generics είναι το εξής:

```
List myIntList = new LinkedList(); // 1
myIntList.add(new Integer(0)); // 2
Integer x = (Integer) myIntList.iterator().next(); // 3
```

Ο προγραμματιστής γνωρίζει γενικά ποιοι τύποι δεδομένων χρησιμοποιούνται για κάθε δομή δεδομένων, αλλά είναι αναγκαία η χρήση κάποιου cast. Ο μεταγλωττιστής εξασφαλίζει την επιστροφή ενός αντικειμένου, αλλά για να εξασφαλιστεί η τυπογραφική ασφάλεια χρειάζεται ένα cast. Αυτό το cast μπορεί να δημιουργήσει σύγχυση όμως καθώς και λάθη κατά την εκτέλεση του προγράμματος, λόγο προγραμματιστικών σφαλμάτων. Τα generics επιλύουν το πρόβλημα αυτό δίνοντας τη δυνατότητα στον προγραμματιστή να δώσει συγκεκριμένο τύπο δεδομένων στις δομές δεδομένων του. Αυτό παρουσιάζεται στο παρακάτω παράδειγμα:

```
List<Integer> myIntList = new LinkedList<Integer>(); // 1'
myIntList.add(new Integer(0)); //2'
Integer x = myIntList.iterator().next(); // 3'
```

Με αυτή την προσέγγιση ο μεταγλωττιστής μπορεί να ελέγξει την ορθότητα των τύπων κατά το χρόνο μεταγλώττισης. Εδώ ο κώδικας αποσαφηνίζει πως η λίστα myIntList είναι μια λίστα με Integers. Με cast σε αντίθετη περίπτωση γίνονται φανερές οι σκέψεις του προγραμματιστή σε συγκεκριμένο σημείο στον κώδικα. Το κέρδος των generics είναι μεγαλύτερη αξιοπιστία και αναγνωσιμότητα.

2.2.6 Annotation Processing

Αυτή η τεχνολογία επιτρέπει τη δημιουργία και επεξεργασία κώδικα, καθώς και εισαγωγή ελέγχου σφαλμάτων, ειδοποιήσεων και σημειώσεων στην εφαρμογή. Αυτό γίνεται με την επεξεργασία των annotations που βρίσκονται σε μια εφαρμογή.
Παρόλη την ευελιξία που προσφέρει ένας annotation processor έχει περιορισμούς στο βαθμό που ελέγχει την Java, αφού δεν κάνει αλλαγές στη γλώσσα αυτή καθ' αυτή. Συγκεκριμένα το annotation processing αποτελεί ένα τμήμα του μεταγλωττιστή της Java και δεν μεταβάλει τον πυρήνα την γλώσσας.

Πιο συγκεκριμένα ο μεταγλωττιστής της Java εκτελεί τα παρακάτω βήματα:

1. **Ανάλυση (parse)** : διάβασμα του συνόλου των *.java αρχείων πηγαίου κώδικα και αντιστοίχηση της ακολουθία συμβόλων που προκύπτει σε AST-κόμβους.
2. **Εισαγωγή**: εισαγωγή συμβόλων για τους ορισμούς στον πίνακα συμβόλων.

3. **Επεξεργασία annotation (process annotation)**: Αν ζητηθεί, επεξεργασία των annotations που έχουν βρεθεί στις μονάδες μεταγλώττισης.
4. **Αποσαφηνισμός**: Χαρακτηρισμό του συντακτικού δέντρου. Αυτό το βήμα περιλαμβάνει επίλυση ονομάτων, έλεγχο τύπων και αναδίπλωση σταθερών.
5. **Ροή**: Εκτελεί ανάλυση ροής στα δέντρα του προηγούμενου βήματος. Αυτό περιλαμβάνει έλεγχο για αναθέσεις και εμβέλεια.
6. **Απλοποίηση (desugar)**: Επανεγγραφή του AST με σκοπό την απομάκρυνση συντακτικής ζάχαρης.
7. **Παραγωγή**: παραγωγή των πηγαίων αρχείων ή των αρχείων κλάσεων.

Η χρήση ου annotation processing θα γίνει πιο κατανοητή στο επόμενο κεφάλαιο.

2.3 SQL

2.3.1 Εισαγωγή στις Βάσεις Δεδομένων

Μια βάση δεδομένων αποτελεί μια συλλογή από σχετιζόμενα δεδομένα , δηλαδή γεγονότα που μπορούν να καταγραφούν και έχουν κάποια υπονοούμενη σημασία. Η βάση έχει τις ακόλουθες ιδιότητες.

- Αναπαριστά κάποια άποψη του πραγματικού κόσμου, τον μικρόκοσμο ή το πεδίο αναφορά. Οι αλλαγές στο πεδίο αναφοράς αντανακλώνται στη βάση δεδομένων.
- Είναι λογικά συνεκτική συλλογή δεδομένων, με κάποια εγγενή σημασία. Μια τυχαία διευθέτηση δεδομένων δεν είναι ορθό να αναφέρεται ως βάση δεδομένων.
- Η σχεδίαση, δημιουργία και χρήση της έχει συγκεκριμένο σκοπό. Προορίζεται για συγκεκριμένες ομάδες χρηστών (ανάλογα με η βάση πάντα) και για προκαθορισμένες εφαρμογές, για τις οποίες αυτοί οι χρήστες ενδιαφέρονται.

2.3.2 Σύστημα Διαχείρισης Βάσεων Δεδομένων

Ένα Σύστημα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ) είναι μια συλλογή προγραμμάτων που επιτρέπουν στους χρήστες να δημιουργούν και να συντηρούν μια βάση δεδομένων. Είναι ένα γενικής χρήσης σύστημα λογισμικού που διευκολύνει τις διαδικασίες ορισμού, κατασκευής, χειρισμού και διαμοιρασμού βάσεων δεδομένων για διάφορες εφαρμογές. Ο ορισμός περιλαμβάνει την προδιαγραφή των τύπων, των δομών και των περιορισμών των δεδομένων που θα αποθηκευτούν στη βάση. Ο ορισμός αυτός αποθηκεύεται ως σύνολο μεταδεδομένων, με τη μορφή ενός καταλόγου ή λεξικού στη βάση δεδομένων. Κατασκευή είναι η διαδικασία αποθήκευσης των ίδιων δεδομένων σε ένα μέσο αποθήκευσης που ελέγχεται από το ΣΔΒΔ. Ο χειρισμός μιας βάσης δεδομένων περιλαμβάνει λειτουργίες όπως υποβολή ερωτημάτων προς τη βάση για ανάκτηση συγκεκριμένων δεδομένων, ενημέρωση της βάσης, ώστε να αντανακλά το μικρόκοσμο και παραγωγή αναφορών από τα δεδομένα. Τέλος ο διαμοιρασμός επιτρέπει σε πολλούς χρήστες και προγράμματα να έχουν ταυτόχρονη προσπέλαση στη βάση δεδομένων.

Ένα πρόγραμμα εφαρμογής προσπελαύνει μια βάση δεδομένων στέλνοντας στο ΣΔΒΔ ερωτήσεις ή αιτήματα για τα δεδομένα. Μια ερώτηση τυπικά προκαλεί την ανάκτηση των δεδομένων. Μια δοσοληψία μπορεί να προκαλεί ανάγνωση και εγγραφή δεδομένων στη βάση.

2.3.3 Σχεσιακό Μοντέλο Βάσης Δεδομένων

Το σχεσιακό μοντέλο περιγράφει τη βάση δεδομένων ως μια συλλογή από σχέσεις. Η κάθε σχέση μπορεί να αντιμετωπιστεί ως ένας πίνακας. Όταν μια σχέση αντιμετωπίζεται ως ένας πίνακας τιμών, κάθε γραμμή στον πίνακα παριστάνει μια συλλογή από τιμές δεδομένων που συσχετίζονται. Σε αυτό το μοντέλο κάθε γραμμή σε έναν πίνακα παριστάνει ένα γεγονός που τυπικά αντιστοιχεί σε μια οντότητα ή μια συσχέτιση του πραγματικού κόσμου. Το όνομα του πίνακα και των στηλών του, χρησιμοποιούνται βοηθητικά προκειμένου να ερμηνευθεί η σημασία των τιμών σε κάθε γραμμή του πίνακα. Προφανώς οι τιμές των δεδομένων σε κάθε στήλη είναι του ίδιου τύπου. Τυπικά η γραμμή λέγεται πλειάδα, η επικεφαλίδα μιας στήλης γνώρισμα και ο πίνακας σχέση.
Τυπικά μια βάση δεδομένων αποτελείται από πολλές σχέσεις και οι πλειάδες των σχέσεων αυτών σχετίζονται με διαφορετικούς τρόπους. Γενικά υπάρχουν πολλοί περιορισμοί στις πραγματικές τιμές μιας κατάστασης βάσης δεδομένων. Οι περιορισμοί αυτοί προέρχονται από κανόνες του μικρόκοσμου που παριστάνει η βάση δεδομένων.

Οι περιορισμοί χωρίζονται στις εξής κατηγορίες:

- Ενδογενείς στο μοντέλο δεδομένων
- Περιορισμοί που εμφανίζονται στο σχήμα.. Μπορούν να εκφραστούν άμεσα στα σχήματα του μοντέλου δεδομένων.
- Σημασιολογικοί περιορισμοί. Δεν μπορούν να εκφραστούν άμεσα στα σχήματα του μοντέλου δεδομένων και πρέπει να επιβληθούν από τα προγράμματα εφαρμογών.

2.3.4 Πράξεις Ενημερώσεων

Οι πράξεις του σχεσιακού μοντέλου διαχωρίζονται σε ανακτήσεις και ενημερώσεις. Βασική μέθοδος για την εκτέλεση των πράξεων είναι μέσω ερωτήσεων στη βάση δεδομένων. Ο χρήστης σχηματίζει μια ερώτηση που προσδιορίζει δεδομένα που τον ενδιαφέρουν και σχηματίζει μια νέα σχέση εφαρμόζοντας σχεσιακούς τελεστές για την ανάκτηση αυτών των δεδομένων.
Υπάρχουν τρεις βασικές πράξεις για ενημερώσεις στις σχέσεις: η εισαγωγή, η διαγραφή και η τροποποίηση. Με την εισαγωγή, εισάγουμε νέες πλειάδες σε μια σχέση. Η διαγραφή χρησιμοποιείται για διαγραφή πλειάδων και η ενημέρωση χρησιμοποιείται για αλλαγή τιμών στις πλειάδες. Τέλος υπάρχει μια πράξη ανάκτησης δεδομένων, η επιλογή, με την οποία επιλέγονται και παρουσιάζονται τα δεδομένα του ερωτήματος. Όποτε γίνεται χρήση των ενημερώσεων ή της επιλογής, δεν πρέπει να παραβιάζονται οι περιορισμοί.

2.3.5 Εισαγωγή SQL

Η SQL (Structured Query Language) είναι πλήρης γλώσσα βάσεων δεδομένων. Διαθέτει εντολές για τον ορισμό δεδομένων, ερωτήσεις και ενημερώσεις. Είναι ταυτόχρονα Γλώσσα Ορισμού Δεδομένων και Γλώσσα Χειρισμού Δεδομένων. Χρησιμοποιεί τους όρους πίνακα, γραμμή και στήλη για τις έννοιες της σχέσης, της πλειάδας και του γνωρίσματος αντίστοιχα.

Οι βασικοί τύποι δεδομένων που είναι διαθέσιμοι για τα ορίσματα περιλαμβάνουν αριθμητικούς τύπους, συμβολοσειρές χαρακτήρων, σειρές δυαδικών αριθμών, λογικές τιμές, ημερομηνίες και ώρα.

- Οι αριθμητικοί τύποι περιλαμβάνουν ακεραίους (integer, smallint) και πραγματικούς αριθμούς (float, real, double)
- Οι τύποι δεδομένων συμβολοσειρών χαρακτήρων μπορούν να είναι σταθερού μεγέθους (char) ή μεταβλητού μεγέθους (varchar)
- Οι τύποι δεδομένων σειρών δυαδικών ψηφίων είναι είτε σταθερού μεγέθους (bit) είτε μεταβλητού (bit varying).
- λογικός τύπος δεδομένων περιλαμβάνει τις κλασσικές τιμές true και false. Ακόμα λόγο της πιθανότητας ύπαρξης της τιμής Null στο πεδίο η sql περιλαμβάνει και την τιμή unknown.
- τύπος δεδομένων date έχει δέκα θέσεις και τα στοιχεία του είναι year, month και day, συνήθως στη μορφή yyyy-mm-dd. Ο τύπος δεδομένων Time περιέχει τουλάχιστον 8 θέσεις με στοιχεία hour, minute και second, συνήθως στη μορφή hh:mm:ss.
- τύπος χρονοσήμου (timestamp) περιλαμβάνει τα πεδία date και time καθώς και έξι θέσεις για κλάσματα δευτερολέπτου και έναν προαιρετικό προσδιορισμό with time zone
- Ένας άλλος τύπος δεδομένων σχετικός με το date, time και timestamp είναι ο τύπος δεδομένων Interval. Ο τύπος αυτός προσδιορίζει ένα διάστημα (interval), δηλαδή μια τιμή που μπορεί να χρησιμοποιηθεί για την αύξηση ή τη μείωση μιας απόλυτης τιμής ημερομηνίας, ώρας ή χρονοσήμου. Τα διαστήματα προσδιορίζονται είτε ως διαστήματα ετών/μηνών (YEAR/MONTH) είτε ως διαστήματα ημερών/ωρών(DAY/TIME)
- Η μορφή των DATE, TIME, TIMESTAMP μπορούν να θεωρηθούν ένας ειδικός τύπος συμβολοσειρών. Επομένως μπορούν να χρησιμοποιηθούν σε συγκρίσεις συμβολοσειρών, αφού μετατραπούν σε ισοδύναμες συμβολοσειρές.

2.3.6 Βασικές ερωτήσεις SQL

Η βασική εντολή για την ανάκτηση δεδομένων στη SQL είναι η εντολή SELECT.
Η βασική μορφή της εντολής SELECT, που λέγεται και απεικόνιση ή μπλοκ SELECT-FROM-WHERE έχει τη μορφή:

```
SELECT <λίστα γνωρισμάτων>
FROM <λίστα πινάκων>
WHERE <συνθήκες>
```

Οι λογικοί τελεστές σύγκρισης για σύγκριση τιμών γνωρισμάτων και για λεκτικούς περιορισμούς είναι = (ίσο), < (μικρότερο), <= (μικρότερο ή ίσο), >= (μεγαλύτερο ή ίσο) και <> (διάφορο)

Ένα ερώτημα μπορεί να περιέχει περισσότερες από μία συνθήκες οι οποίες συνδέονται μεταξύ τους με τους λογικούς τελεστές AND ή/και OR

Η SQL επιτρέπει τη χρήση δύο ή περισσοτέρων γνωρισμάτων με το ίδιο όνομα, με την προϋπόθεση ότι ανήκουν σε διαφορετικές σχέσεις (πίνακες). Σε περίπτωση που έχουμε διαφορετικά γνωρίσματα με το ίδιο όνομα μπορούμε να τα συμπεριλάβουμε στο ίδιο ερώτημα με την εξής μορφή:
<πίνακας>.<γνώρισμα>

Δηλαδή το όνομα του πίνακα, ακολουθούμενο από τελεία, ακολουθούμενη από το όνομα του γνωρίσματος.

```
SELECT <πίνακας>.<γνώρισμα>, <πίνακας>.<γνώρισμα>
FROM <πίνακας>
WHERE <συνθήκη> OR/AND <συνθήκη>
```

Αν στο ερώτημα χρησιμοποιούνται όλα τα γνωρίσματα της σχέσης, αντί να γραφτούν όλα δίπλα από το λεκτικό SELECT μπορούν να αντικατασταθούν από το σύμβολο “*”.
Στο ερώτημα δεν είναι αναγκαίο να υπάρχουν συνθήκες. Σε αυτή την περίπτωση το λεκτικό WHERE παραλείπεται. Αυτές οι δύο λειτουργίες παρουσιάζονται μαζεμένες στο παρακάτω παράδειγμα

```
SELECT *
FROM <πίνακας>
```

Προκειμένου να απαλείψουμε διπλότυπες πλειάδες χρησιμοποιούμε το λεκτικό DISTINCT και πιο συγκεκριμένα SELECT DISTINCT. Αν θέλουμε τις διπλότυπες πλειάδες χρησιμοποιούμε τα λεκτικά SELECT ALL. Η χρήση μόνο του λεκτικού SELECT είναι ισοδύναμη του SELECT ALL.

Η SQL υποστηρίζει πράξεις συνόλων. Υπάρχουν πράξεις για ένωση (UNION) διαφορά (EXCEPT) και τομή συνόλων (INTERSECT).ΟΙ σχέσεις που προκύπτουν ως αποτέλεσμα των πράξεων αυτών είναι σύνολα πλειάδων. Οι πράξεις αυτές εμφανίζονται σε σχέσεις συμβατές ως προς την ένωση και πρέπει να εξασφαλιστεί ότι οι δύο σχέσεις στις οποίες εφαρμόζεται η πράξη έχουν ίδια γνωρίσματα και ότι τα γνωρίσματα αυτά εμφανίζονται και στις δύο σχέσεις με την ίδια σειρά.

Μια ακόμα δυνατότητα της SQL είναι η διάταξη των πλειάδων που περιέχονται στο αποτέλεσμα μιας ερώτησης σύμφωνα με τις τιμές ενός ή περισσοτέρων γνωρισμάτων χρησιμοποιώντας το λεκτικό ORDER BY.

Για την τροποποίηση μιας βάσης δεδομένων η SQL χρησιμοποιεί τρείς εντολές, τις INSERT, UPDATE και DELETE.

Η απλούστερη εντολή της INSERT χρησιμοποιείται για την προσθήκη μιας πλειάδας σε μια σχέση. Πρέπει να προσδιοριστεί το όνομα της σχέσης καθώς και μια λίστα τιμών για την πλειάδα. Οι τιμές πρέπει να παρατίθενται με την ίδια σειρά που προσδιορίζονται τα γνωρίσματα της σχέσης.

```
INSERT INTO <πίνακας>
VALUES <τιμές γνωρισμάτων>
```

Σε μια άλλη μορφή μπορούν να οριστούν τα γνωρίσματα που θέλει ο χρήστης να συμπληρώσει με τις αντίστοιχες τιμές τους. Όσα γνωρίσματα δεν αναφερθούν συμπληρώνονται με NULL ή με τις προκαθορισμένες τους τιμές. Προσοχή πρέπει να δοθεί στους περιορισμούς των γνωρισμάτων.

```
INSERT INTO <πίνακας> (<γνώρισμα>, <γνώρισμα>… <γνώρισμα>)
VALUES (<τιμές γνωρισμάτων>)
```

Η εντολή DELETE διαγράφει πλειάδες από μια σχέση. Περιέχει μια πρόταση WHERE όμοια με αυτή της εντολής SELECT. Οι πλειάδες διαγράφονται ρητά από έναν μόνο πίνακα κάθε φορά. Ανάλογα με τη συνθήκη της εντολής WHERE μπορούν να διαγραφούν καμία, μία ή περισσότερες πλειάδες από τη σχέση. Παράλειψη της εντολής WHERE οδηγεί στη διαγραφή όλων των πλειάδων της σχέσης. Ωστόσο η σχέση παραμένει στη βάση δεδομένων.

```
DELETE FROM <πίνακας>
WHERE <συνθήκη>
```

Η εντολή UPDATE χρησιμοποιείται για την ενημέρωση γνωρισμάτων σε μία ή περισσότερες πλειάδες μιας σχέσης. Με την εντολή WHERE είναι δυνατή η τροποποίηση καμίας, μίας ή και περισσότερων πλειάδων ταυτόχρονα. Η εντολή SET χρησιμοποιείται για την αλλαγή των τιμών των γνωρισμάτων.

```
UPDATE <πίνακας>
SET <γνώρισμα>=<τιμή>… <γνώρισμα>=<τιμή>
WHERE <συνθήκη>
```

2.3.7 Συνένωση Πινάκων

Η έννοια του πίνακα συνένωσης ενσωματώθηκε στην SQL ώστε να επιτρέπει στους χρήστες να προσδιορίζουν έναν πίνακα που προκύπτει από μια πράξη συνένωσης στην πρόταση FROM μιας ερώτησης.

```
SELECT <γνωρίσματα>
FROM <πίνακας-κ> JOIN <πίνακας-λ> ON <γνώρισμα-κ> = <γνώρισμα-λ>
WHERE <συνθήκη>
```

Ο προκαθορισμένος τύπος συνένωσης είναι η εσωτερική συνένωση (INNER JOIN), στην οποία μια πλειάδα συμπεριλαμβάνεται στο τελικό αποτέλεσμα που επιστρέφει το ερώτημα, μόνο αν υπάρχει μια πλειάδα που να της ταιριάζει στην άλλη σχέση. Για να εμφανιστούν όλες οι πλειάδες πρέπει να γίνει κάποιου τύπου εξωτερική συνένωση (OUTER JOIN). Κατηγορίες εξωτερικών συνενώσεων είναι οι αριστερή εξωτερική συνένωση (LEFT OUTER JOIN), στην οποία συμπεριλαμβάνονται όλες οι πλειάδες της αριστερής σχέσης και όσες πλειάδες της δεξιάς σχέσης έχουν κοινό γνώρισμα με κάποιο γνώρισμα της αριστερής σχέσης, δεξιά εξωτερική συνένωση (RIGHT OUTER JOIN), στην οποία περιλαμβάνονται όλες οι πλειάδες της δεξιάς σχέσεις και όσες πλειάδες της αριστερής έχουν κοινό γνώρισμα με αυτό της δεξιάς σχέσης και τέλος πλήρης εξωτερική συνένωση (FULL OUTER JOIN), στην οποία περιέχονται όλες οι πλειάδες και της δεξιάς και της αριστερής σχέσης. Σε περίπτωση που κάποια πλειάδα της μιας σχέσης δεν αντιστοιχεί σε κάποια πλειάδα της δεύτερης σχέσης, παίρνει την τιμή NULL στα αντίστοιχα γνωρίσματα. Τέλος το λεκτικό CROSS JOIN δημιουργεί ένα καρτεσιανό γινόμενο μεταξύ των σχέσεων.

Ακόμα είναι δυνατή η συνένωση περισσότερων από δύο σχέσεων. Αυτό επιτυγχάνεται με διαδοχικές φωλιασμένες συνενώσεις. Σε τέτοιες περιπτώσεις πρέπει να δοθεί ιδιαίτερη προσοχή στη μέθοδο συνένωσης, καθώς κακός χειρισμός μπορεί να οδηγήσει σε πολύ μεγάλο όγκο δεδομένων.

Τέλος, προκειμένου να χρησιμοποιηθούν γνωρίσματα από διαφορετικές σχέσεις με ίδιο όνομα, είναι δυνατό να μετονομαστούν αυτά τα γνωρίσματα στο ερώτημα. Αυτό γίνεται με την εντολή AS. Εκτός από γνωρίσματα είναι δυνατή και η μετονομασία πινάκων. Η μετονομασία αυτή δεν αλλάζει το όνομα του γνωρίσματος ή της σχέσης στη βάση αλλά στο ερώτημα. Πρακτικά δίνεται ένα ψευδώνυμο στη σχέση ή στο γνώρισμα

```
SELECT <γνώρισμα> AS <ψευδώνυμο>
FROM <πίνακα> AS <ψευδώνυμο>
WHERE <συνθήκη>
```

Σε αυτό το σημείο ολοκληρώνεται το δεύτερο κεφάλαιο. Πλέον ο αναγνώστης έχει μια βασική ιδέα για το πώς λειτουργούν οι πάροχοι ORM λειτουργικότητας, αρκετές τεχνολογίες της java και απλές γλώσσες πάνω στις σχεσιακές βάσεις δεδομένων και την SQL. Στο επόμενο κεφάλαιο θα παρουσιαστεί η αρχιτεκτονική της εφαρμογής καθώς και τον τρόπο που συνδυάζονται οι παραπάνω τεχνολογίες.

ΚΕΦΑΛΑΙΟ 3

Αρχιτεκτονική

Σε αυτή την ενότητα παρουσιάζεται η αρχιτεκτονική της εφαρμογής. Αρχικά εξετάζεται το μοντέλο της SQL και τα διάφορα αρχεία που χρησιμοποιούνται για την υποστήριξη του μοντέλου αυτού. Στη συνέχεια παρουσιάζεται η ροή της λειτουργίας του κώδικα και τα διάφορα στάδια από τα οποία πρέπει να περάσει κάποια εφαρμογή που χρησιμοποιεί αυτή την επέκταση. Τέλος δίνονται οι κώδικες που έχουν γραφτεί για να έχει ο αναγνώστης τη δυνατότητα να εξετάσει πιο αναλυτικά την κάθε κλάση του μοντέλου και των βοηθητικών αρχείων.

3.1 Υλοποίηση

3.1.1 Μοντέλο SQL

3.1.1.1 Field

Ορισμός : Field<T, TableInstance extends Table>

Περιγραφή: Αντιστοιχεί σε ένα πεδίο του πίνακα. Το generic T αντιστοιχεί στον τύπο του πεδίου, στο οποίο αναφέρεται. Το generic TableInstance αντιστοιχεί στον πίνακα, στον οποίο βρίσκεται το πεδίο.

Μεταβλητές
Public

- -

Private

- -

Protected

- alias
  - Τα ψευδώνυμο του πεδίου
- fieldName
  - Το πραγματικό όνομα του πεδίου

Συναρτήσεις
Constructor

- FieldImpl()
  - Απλός constructor του πεδίου

Public

- getAlias()
  - getter της μεταβλητής alias
- setAlias(String)
  - setter της μεταβλητής alias
- getFieldName()
  - getter Της μεταβλητής fieldName
- setFieldName(String)
  - setter της μεταβλητής fieldName

Private

- -

Protected

- -

3.1.1.2 QueryGET

Ορισμός : QueryGET<TableInstance extends Table>

Περιγραφή: Αυτό το αντικείμενο αντιστοιχεί σε μια εντολή τύπου SELECT.

Μεταβλητές

Public:

- -

Private:

- joinsOfTables:
  - Απεικονίζονται οι συσχετίσεις μεταξύ πινάκων συνδεδεμένων μέσω joins.
- resultMap:
  - Πίνακας κατακερματισμού, στον οποίο στον οποίο αποθηκεύεται το τελικό αποτέλεσμα

Protected:

- filter: Αντικείμενο τύπου Filters<TableInstance>, στο οποίο αποθηκεύονται τα φίλτρα που εφαρμόζουμε πάνω στο ερώτημα
- join: Λίστα τύπου JOIN<Tableinstance,?,?>. Αποθηκεύονται τα JOIN που αντιστοιχούν σε αυτό τον πίνακα
- rootAlias: String, στο οποίο αποθηκεύεται το ψευδώνυμο του πίνακα
- rootInstance: String, στο οποίο αποθηκεύεται η κλάση του πίνακα
- rootTable: String, στο οποίο αποθηκεύεται το πραγματικό όνομα του πίνακα
- selectedFields: λίστα τύπου Field στην οποία αποθηκεύονται τα πεδία του πίνακα που πρέπει να επιστραφούν από το ερώτημα.
- union: Λίστα από αντικείμενα τύπου QueryGET. Αντιστοιχούν σε διαδοχικά select Πάνω στο ερώτημα.

Συναρτήσεις

Constructors:

- QueryGETImpl() : Απλός constructor. Χρήση του δε συνίσταται
- QueryGETImpl(Class<TableInstance> tableInstance) : Constructor. Δέχεται ως όρισμα μια κλάση που αντιστοιχεί στον πίνακα εκφρασμένο σε μορφή αντικειμένου.

Public:

- generate(Class<?>) : Κατασκευή του ερωτήματος, αποστολή στη βάση και επεξεργασία αποτελεσμάτων. Επιστέφει μια λίστα από αντικείμενα τύπου clazz
- getFilter(): getter της μεταβλητής Filter
- setFilter(EList<Filters>): setter της μεταβλητής Filter
- getJoin(): getter της μεταβλητής Join
- getRootAlias(): getter της μεταβλητής rootAlias
- setRootAlias(String): setter της μεταβλητής rootAlias
- getRootTable(): getter της μεταβλητής rootTable
- getSelectedFields(): getter της μεταβλητής seletedFields
- getUnion(): getter της μεταβλητής union

Private:

- generateKey(ResultSet, JOIN<?,?,?>,String):
    - Κατασκευή κλειδιού για τη μεταβλητή ResultMap
- joinFieldUnwrap(StringBuffer, JOIN<?,?,?>)
    - Εισαγωγή των πεδίων ενός Joined πίνακα στο τελικό ερώτημα
- joinTableUnwrap(StringBuffer, JOIN<?,?,?>)
    - Εισαγωγή των joined πινάκων στο ερώτημα
- resolveFilter(StringBuffer, Filters<?>,String)
    - Εισαγωγή ενός φίλτρου στο ερώτημα
- resolveJoin(Class<?>,JOIN<?,?,?>,ResultSet,Map<String,List<JOIN<?,?,?>>>)
    - Εμβάθυνση σε ένα εσωτερικό JOIN
- createObject(Class<?>,ResultSet, JOIN<?,?,?>)
    - Δημιουργία τελικού αντικειμένου πίνακα
- Combine(Class<?>,String,Map<String,List<JOIN<?,?,?>>>,JOIN<?,?,?>,Resultset)
    - Συνένωση συσχετισμένων αντικειμένων

Protected:

- -

3.1.1.3 JOIN

Ορισμός: JOIN<InitialTable extends Table, JoinedTable extends Table, Type>

Περιγραφή: Σε αυτό το αντικείμενο αποθηκεύεται πληροφορία για έναν συνδεδεμένο πίνακα Πρέπει να οριστούν ο αρχικός πίνακας, ο πίνακας που συνδέεται με τον αρχικό και ο τύπος του πεδίου πάνω στον οποίο γίνεται η σύνδεση.

Μεταβλητές

Public

- -

Private

- resultMap
    - Πίνακας κατακερματισμού, στον οποίο αποθηκεύονται τα ενδιάμεσα αποτελέσματα για τα joins

Protected

- initialAlias
    - Ψευδώνυμο αρχικού πίνακα
- initialTable
    - Πραγματικό όνομα αρχικού πίνακα
- joinedAlias
    - Ψευδώνυμο συνδεδεμένου πίνακα
- joinedTable
    - Πραγματικό όνομα συνδεδεμένου πίνακα
- joinMore
    - Λίστα με επιπλέον JOIN αντικείμενα. Ο joinedTable σε αυτά τα JOIN θεωρείται initialTable
- onFieldInitial
    - Το πεδίο του αρχικού πίνακα πάνω στο οποίο θα γίνει το join
- onFieldJoined
    - Το πεδίο του συνδεδεμένου πίνακα πάνω στο οποίο θα γίνει το join
- joinedTableFields

- Τα επιπλέον πεδία που θα μπουν στο ερώτημα και ανήκουν στον joinedTable

Συναρτήσεις

Constructor

- JOINImpl()
  - Απλός constructor του αντικειμένου JOIN. Δε συνίσταται η χρήση αυτού του constructor
- JOINImpl(Class<InitialTable>,Class<JoinedTable>)
  - Σύνθετος constructor του αντικειμένου JOIN. Δέχεται τις κλάσεις των πινάκων και αρχικοποιεί τα πεδία initialTable, joinedTable και resultMap. Το αντικείμενο πρέπει να αρχικοποιείται με αυτόν τον constructor

Public

- getInitialAlias()
  - getter του πεδίου initialAlias
- setInitialAlias()
  - setter του πεδίου initialAlias
- getInitialTable()
  - getter του πεδίου initialTable. Δεν υπάρχει setter για αυτό το πεδίο. Δεν πρέπει να δίνεται η δυνατότητα στον χρήστη να αλλάζει το πραγματικό όνομα ενός πίνακα
- getJoinedAlias()
  - getter του πεδίου joinedAlias
- setJoinedAlias()
  - setter του πεδίου joinedAlias
- getJoinedTable
  - getter του πεδίου joinedTable. Δεν υπάρχει setter για αυτό το πεδίο. Δεν πρέπει να δίνεται η δυνατότητα στον χρήστη να αλλάζει το πραγματικό όνομα ενός πίνακα
- getJoinedTableFields()
  - getter της λίστας joinedTableFields. Ο χρήστης μπορεί να εισάγει στοιχεία στη λίστα χρησιμοποιώντας μεθόδους του αντικειμένου List αφού χρησιμοποιήσει τον getter
- getJoinMore()
  - getter της λίστας joinMore. Ο χρήστης μπορεί να εισάγει στοιχεία στη λίστα χρησιμοποιώντας μεθόδους του αντικειμένου List αφού χρησιμοποιήσει τον getter
- getOnFieldInitial()
  - getter του πεδίου του αρχικού πίνακα πάνω στο οποίο θα γίνει το join
- setOnFieldInitial()
  - setter του πεδίου του αρχικού πίνακα πάνω στον οποίο θα γίνει το Join
- getOnFieldJoined()
  - getter του πεδίου του συνδεδεμένου πίνακα πάνω στο οποίο θα γίνει το join
- setOnFieldJoined()
  - setter του πεδίου του συνδεδεμένου πίνακα πάνω στο οποίο θα γίνει το join
- getResultMap
  - getter του resultMap
- setResultMap
  - setter του resultMap

Private

- -

Protected

- -

3.1.1.4 Filters

Ορισμός: Filters<TableInstance extends Table>

Περιγραφή Αυτό το αντικείμενο αντιστοιχεί σε ένα φίλτρο τύπου WHERE. Εισάγεται σε ένα αντικείμενο QueryGET. Είναι δυνατή η εισαγωγή πολλαπλών φίλτρων

Μεταβλητές

Public

- -

Private

- andFilter
  - αντικείμενο τύπου Filter<?>. περιέχει ένα φίλτρο που συνδέεται με το προηγούμενό του με λογικό AND
- orFilter
  - αντικείμενο τύπου Filter<?>. περιέχει ένα φίλτρο που συνδέεται με το προηγούμενό του με λογικό OR
- chainedFilters
  - Λίστα τύπου NaryNode. Χρησιμοποιείται για την αποθήκευση διαδοχικών φίλτρων που έχουν ως πατέρα το τρέχων αντικείμενο Filter

Protected

- Condition
  - Αντικείμενο τύπου condition. Η συνθήκη που εκφράζει αυτό το αντικείμενο Filter
- fromJOIN
  - Αντικείμενο τύπου JOIN<?,TableInstance, ?>. Περιέχει τα στοιχεία του joined πίνακα που χρησιμοποιείται.

Συναρτήσεις

Constructor

- FiltersImpl()
  - Απλός constructor. Δε συνίσταται η χρήση του. Αν χρησιμοποιηθεί αυτός ο constructor τα υπόλοιπα δεδομένα πρέπει να εισαχθούν με το χέρι
- FiltersImpl(Field<?,?>, Operators, Object, JOIN<?,TableInstance,?>)
  - Κατασκευάζει ένα αντικείμενο τύπου Filters, με όλα τα δεδομένα που χρειάζεται. Αν η μεταβλητή JOIN είναι null θεωρείται ως πίνακας αυτός που έχει εισαχθεί στο queryGET

Public

- and(Filters<?>)
  - προσθέτει ένα αντικείμενο NaryNode στη λίστα chainedFilters. Θέτει το πεδίο bl του NaryNode σε AND
- or(Filters<?>)
  - προσθέτει ένα αντικείμενο NaryNode στη λίστα chainedFilters. Θέτει το πεδίο bl του NaryNode σε OR
- getAndFilter()
  - getter της μεταβλητής andFilter
- getOrFilter()
  - getter της μεταβλητής orFilter
- getChainedFilters()
  - getter της μεταβλητής chainedFilters
- getFromJOIN()

    - getter της μεταβλητής fromJOIN
- setFromJOIN(JOIN<?,TableInstance,?>)
    - setter της μεταβλητής fromJOIN
- getCondition()
    - getter της μεταβλητής condition
- setCondition(Condition<?,TableInstance>)
    - setter της μεταβλητής condition

Private

- -

Protected

- **-**

3.1.1.5 Condition

Ορισμός: Condition<T extends JavaObject, TableInstance extends Table>

Περιγραφή : Αυτό το αντικείμενο αποτελεί τη συνθήκη ενός φίλτρου

Μεταβλητές

Public

- -

Private

- -

Protected

- field
    - Πεδίο τύπου Field<T,TableInstance>. Το πεδίο πάνω στο οποίο εφαρμόζεται η συνθήκη
- operator
    - Πεδίο τύπου Operators. Ο τελεστής που χρησιμοποιείται
- other
    - Πεδίο τύπου T. Η τιμή του field.

Συναρτήσεις

Constructor

- ConditionImpl()
    - Απλός constructor για αυτό το αντικείμενο

Public

- getField()
    - getter του πεδίου field
- setField(Field<T,TableInstance>)
    - setter του πεδίου field
- getOperator()
    - getter του πεδίου operator
- setOperator(Operators)
    - setter του πεδίου operator
- getOther()
    - getter του πεδίου other
- setOther(T)
    - setter του πεδίου other

Private

- -

Protected

- -

3.1.1.6 Table

Ορισμός: Table

Περιγραφή: Βοηθητικό αντικείμενο. Επέκταση ενός αντικειμένου πίνακα. Η εφαρμογή χρησιμοποιεί το Table προκειμένου να αναγνωρίζει τα αντικείμενα που μπορεί να χρησιμοποιήσει.
Επεκτείνει το generic τύπου TableInstance κατά κύριο λόγο για να εξασφαλίσει το typesafety των δεδομένων του χρήστη.

Μεταβλητές
Public
- -

Private
- -

Protected
- tableEntity
  - Μεταβλητή τύπου Class. Αντιστοιχεί στην κλάση του αντικειμένου ενός πίνακα
- tableName
  - Μεταβλητή τύπου String. Αντιστοιχεί στο όνομα του πίνακα

Συναρτήσεις
Constructor
- TableImpl()
  - Απλός constructor του αντικειμένου

Public
- getTableEntity()
  - getter της μεταβλητής tableEntity
- setTableEntity(Class)
  - setter της μεταβλητής tableEntity
- getTableName()
  - getter της μεταβλητής tableName
- setTableName(String)
  - setter Της μεταβλητής tableName

Private
- -

Protected
- **-**

3.1.1.7 Bol

Ορισμός: Bol

Περιγραφή: Αντικείμενο που περιέχει enumerations. Χρησιμοποιείται για να προσδιορίσει αν δύο διαδοχικά φίλτρα συνδέονται με AND ή OR

Μεταβλητές
Public

- -

Private

- literal
- name
- value

Protected

- -

Συναρτήσεις

Constructor

- -

Public

- get(int)
  - επιστρέφει τον κατάλληλο operator ανάλογα με το value
- get(String)
  - επιστρέφει τον κατάλληλο operator ανάλογα με το String
- getByName(String)
  - επιστρέφει τον κατάλληλο operator ανάλογα με το name
- getLiteral()
  - getter για το literal
- getName()
  - getter για το name
- getValue()
  - getter για το value

Private

- -

Protected

- -

3.1.1.8 NaryNode

Ορισμός: NaryNode

Περιγραφή: Αντικείμενο που περιέχει ένα αντικείμενο Filter και ένα αντικείμενο bol. Βοηθάει στην υλοποίηση πολλαπλών διαδοχικών φίλτρων

Μεταβλητές

Public

- -

Private

- -

Protected

- bl
  - μεταβλητή τύπου bol. Καθορίζει αν το φίλτρο συνδέεται με το προηγούμενό του μέσω AND ή OR τελεστή
- value
  - μεταβλητή τύπου Filters<?>. περιέχει τα στοιχεία του φίλτρου

Συναρτήσεις

Constructor

- NaryNodeImpl()
    - Απλός constructor. Κάνει Instantiate ένα αντικείμενο τύπου NaryNode

Public

- getBl()
    - getter της μεταβλητής bl
- setBl(bol)
    - setter της μεταβλητής bl
- getValue()
    - getter της μεταβλητής value
- setValue(Filters<?>)
    - setter της μεταβλητής value

Private

- -

Protected

- -

3.1.1.9 UML διάγραμμα

Οι παραπάνω κλάσεις και ο συσχετισμός μεταξύ τους περιγράφονται μέσω του διαγράμματος UML που δίνεται παρακάτω:

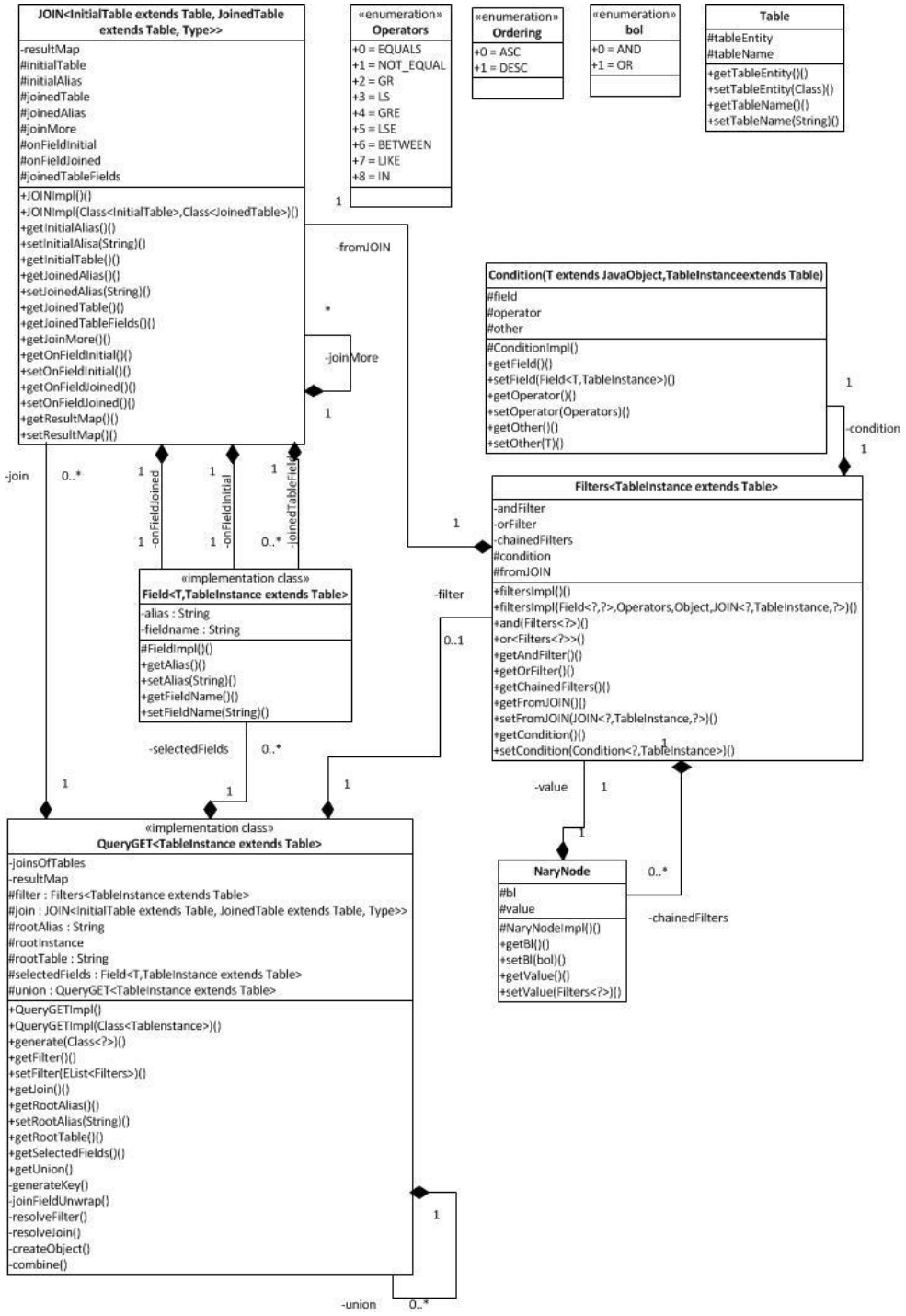

Figure 15: UML Diagram of SQL Model

3.1.2 Βοηθητικά Αρχεία

3.1.2.1 InitializeMetaData

Περιγραφή: Αυτό το αρχείο χρησιμοποιείται για την αρχικοποίηση των μετα-δεδομένων που υπάρχουν στην εφαρμογή. Γίνεται χρήση της μεθόδου initializeFromPackage(String scannedPackage).

Μέθοδοι

Public

- initializeFromPackage(String)
  - Αρχικοποίηση δεδομένων που βρίσκονται μέσα στο package με όνομα String.

Private

- find(File, String)

Εύρεση όλων των κλάσεων που περιέχονται στο μονοπάτι String και το αρχείο File

3.1.3 Εξωτερικά αρχεία

Προκειμένου να μπορέσει η επέκταση να χρησιμοποιήσει annotation processing χρειάζεται μια εξωτερική βιβλιοθήκη η οποία περιέχει τον κώδικα που εκτελείται κατά τη διάρκεια της μεταγλώττισης. Αυτός ο κώδικας βρίσκεται στο jar sqlProcessor (sqlModelAnnotation).

sqlModelAnnotation
- **src**
  - **gencode.Tables.Annotation**
    - MetaSource.java
  - **Tables.Annotation**
    - AnnotationProcessor.java
    - GenerateMetaData.java
    - InitMetaData.java
    - MetaStorage.java
- **JRE System Library**
- **META-INF**
  - **Services**
    - javax.annotation.processing.Processor
- **templates**
  - metagen.javajet

Παραπάνω παρουσιάζεται η δομή του sqlProcessor.
Το MetaSource δημιουργείται αυτόματα από το emf-jet και χρησιμοποιείται για την δημιουργία των μεταδεδομένων των οντοτήτων του χρήστη. Το AnnotationProcessor είναι το αρχείο που χρησιμοποιείται από τον μεταγλωττιστή προκειμένου να επεξεργαστεί τις οντότητες και μέσω αυτών και του MetaSource παράγει τα μεταδεδομένα. Τα GenerateMetadata και InitMetaData είναι άδεια Annotations και χρησιμοποιούνται για να υποδείξουν ποιες οντότητες θα επεξεργαστούν από τις μεθόδους των κλάσεων InitializeMetaData και AnnotationProcessor.

Τέλος η κλάση MetaStorage είναι ένα Javabean που περιέχει πληροφορίες των οντοτήτων για τη δημιουργία των μεταδεδομένων.

3.1.3.1 AnnotationProcessor

Περιγραφή: Κλάση που περιέχει εντολές για δημιουργία μεταδεδομένων κατά το κτίσιμο της εφαρμογής

Μεταβλητές

Public

- -

Private

- filer
  - μεταβλητή τύπου filer. Χρησιμοποιείται από το annotation Processing Για δημιουργία καινούργιων αρχείων

Protected

- -

Συναρτήσεις

Constructor

- -

Public

- Init(ProcessingEnvironment)
  - Συνάρτηση αρχικοποίησης. Καλείται κατά τη διάρκεια του annotation Processing και σε αυτή τη συνάρτηση γίνεται αρχικοποίηση της μεταβλητής Filer
- Process(Set, RoundEnvironment)
  - Καλείται κατά τη διάρκεια του annotation Processing και αναλαμβάνει την επεξεργασία των οντοτήτων και τη δημιουργίας των μεταδεδομένων

Private

- primitiveToObjectConverter(String)
  - Παίρνει ένα String με το όνομα κάποιου πρωτόγονου τύπου (int, char κλπ) επιστρέφει ένα String με το αντίστοιχο αντικείμενο (java.lang.Integer, java.lang.Character κλπ)

Protected

- -

3.1.3.2 MetaStorage

Περιγραφή: Κλάση στην οποία αποθηκεύονται πληροφορίες για τις οντότητες.

Μεταβλητές

Public

- -

Private

- className
  - Μεταβλητή τύπου String. Περιέχει το όνομα της οντότητας που επεξεργάζεται
- metaClassName
  - Μεταβλητή τύπου String. Περιέχει το όνομα της κλάσης των μεταδεδομένων που θα δημιουργηθούν για την οντότητα με className
- fieldMapping
  - Χάρτης τύπου <String, String>.περιέχει τα πεδία της οντότητας className. Το κλειδί είναι το όνομα του πεδίου και η τιμή είναι ο τύπος του πεδίου.

Protected

- -

Συναρτήσεις

Constructor

- MetaStorage. Απλός constructor Που αρχικοποιεί τις μεταβλητές.

Public

- getClassName()
  - getter της μεταβλητής className
- setClassName(String)
  - setter της μεταβλητής className
- getFieldMapping()
  - getter της μεταβλητής fieldMapping
- setFieldMapping(Map<String,String>)
  - setter της μεταβλητής fieldMapping
- getMetaClassName()
  - getter της μεταβλητής metaClassName
- setMetaClassName(String)
  - setter Της μεταβλητής metaClassName

Private

- -

Protected

- -

3.2 Διαδικασία λειτουργίας

3.2.1 Ορισμός Πινάκων

Οι πίνακες πρέπει να οριστούν από τον χρήστη, ανάλογα με τη βάση δεδομένων που χρησιμοποιεί. Ένας πίνακας έχει τη μορφή ενός POJO (Plain Old Java Object). Η χρήση των JavaBeans ή των Entities (σε περίπτωση που ο χρήστης χρησιμοποιεί JPA) είναι η πιο συνήθης μέθοδος για αντιστοίχηση ενός πίνακα με κάποιο αντικείμενο σε Java.

Η επέκταση λειτουργεί ανεξάρτητα από άλλα frameworks (όπως το Hibernate), με αποτέλεσμα να μπορεί να χρησιμοποιηθεί σε συνδυασμό με αυτά. Είτε δοθεί ένα απλό JavaBean είτε κάποιο Entity (το οποίο στην ουσία είναι JavaBean με μεγαλύτερη λειτουργικότητα), μπορεί με ελάχιστες τροποποιήσεις στο JavaBean να χρησιμοποιηθεί από τον χρήστη αυτή εδώ η επέκταση, χωρίς να θυσιάστει η λειτουργικότητα που προσφέρουν άλλες επεκτάσεις όπως το JPA και το Hibernate.

Στη συνέχεια παρουσιάζεται ένα παράδειγμα ενός JavaBean που χρησιμοποιεί ένας χρήστης:

```
package sqlmodel_v2.entitiesTest;
import java.io.Serializable;
import Tables.Annotation.GenerateMetadata;
import java.util.List;

@GenerateMetadata
public class Person implements Serializable {
	private static final long serialVersionUID = 1L;
	private int id;
	private String name;
	private List<Hobby> hobbies;
	private int hobby_id;
```

```
	public Person() {
	}
	public int getHobby_id() {
		return hobby_id;
	}
	public void setHobby_id(int hobby_id) {
		this.hobby_id = hobby_id;
	}
	public int getId() {
		return this.id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return this.name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public List<Hobby> getHobbies() {
		return this.hobbies;
	}
	public void setHobbies(List<Hobby> hobbies) {
		this.hobbies = hobbies;
	}
	public Hobby addHobby(Hobby hobby) {
		getHobbies().add(hobby);
		hobby.setPerson(this);
		return hobby;
	}
	public Hobby removeHobby(Hobby hobby) {
		getHobbies().remove(hobby);
		hobby.setPerson(null);
		return hobby;
	}
}
```

Η κλάση Person

Όπως παρατηρούμε στο παραπάνω παράδειγμα έχουμε τέσσερεις private μεταβλητές (εξαιρούμε το **private static final long *serialVersionUID*** = 1L;). Αυτές οι μεταβλητές είναι τα ορίσματα. Για κάθε όρισμα έχουμε τους αντίστοιχους getters και setters. Η μέθοδος Person()είναι ο constructor του JavaBean και χρησιμοποιείται στη δημιουργία ενός αντικειμένου της κλάσης Person.

Προκειμένου η εφαρμογή αυτή να αναγνωρίσει την κλάση Person (και οποιαδήποτε άλλη κλάση) χρησιμοποιείται το annotation @GenerateMetadata.

Αυτή η κλάση αντιστοιχεί πλήρως σε έναν πίνακα στη βάση που λέγεται Person και πιο συγκεκριμένα σε μια γραμμή του πίνακα αυτού. Τα πεδία του πίνακα αυτού είναι στην πραγματικότητα τρία, το id, name και hobby_id. Η μεταβλητή ***serialVersionUID*** αγνοείται για την ώρα. Τέλος έχουμε μια λίστα

private List<Hobby> hobbies. Η λίστα αυτή αντιπροσωπεύει μια αναφορά σε έναν άλλον πίνακα, στη συγκεκριμένη περίπτωση σε έναν πίνακα που ονομάζεται Hobby. Το πραγματικό πεδίο που βλέπει κάποιος στη βάση είναι το hobby_id. Εφόσον ο χρήστης θελήσει μαζί με μια εγγραφή Person να λάβει και τα hobbies της εγγραφής περιμένει μια λίστα από αντικείμενα τύπου Hobby. Τη μεταβλητή hobby_id τη χρειάζεται η εφαρμογή για την κατασκευή του ερωτήματος.

Κάτι πολύ σημαντικό που πρέπει να αναφερθεί είναι ότι το όνομα της κλάσης (Person στο παράδειγμά αυτό) πρέπει να αντιστοιχεί στο όνομα του αντίστοιχου πίνακα στη βάση. Ομοίως τα ονόματα των μεταβλητών (id, name, hobby_id έχουν το ίδιο όνομα με τα αντίστοιχα πεδία που πίνακα Person)

3.2.2 Δημιουργία Μεταδεδομένων

Τα μεταδεδομένα αποτελούν πληροφορία και δεδομένα που χρειάζεται η επέκταση προκειμένου να λειτουργήσει σωστά. Συνήθως παράγονται δυναμικά ανάλογα με τα δεδομένα που έχει εισάγει ο χρήστης, χωρίς επιπλέον προγραμματιστική επιβάρυνση από τη μεριά του.

Η δημιουργία των μεταδεδομένων γίνεται κατά τη διάρκεια του στησίματος (build) μιας εφαρμογής. Η επέκταση που αναπτύσσεται εδώ χρειάζεται μεταδεδομένα που παράγονται από τις κλάσεις πινάκων που έχει δημιουργήσει ο χρήστης στην εφαρμογή του. Αυτές οι κλάσεις πρέπει να έχουν σημανθεί με το annotation @GenerateMetadata όπως εξηγήθηκε στην προηγούμενη ενότητα. Η δημιουργία των μεταδεδομένων βασίζεται στην ικανότητα του eclipse να κάνει επεξεργασία των annotations (annotation processing).

Περιγραφή κώδικα

Σε αυτή την παράγραφο θα αναλυθεί ο κώδικας που συμμετέχει στο annotation Processing. Ο σκοπός αυτής της επεξεργασίας είναι η δημιουργία των μεταδεδομένων. Όπως ειπώθηκε παραπάνω, ο χρήστης πρέπει να ορίσει στις κλάσεις-πίνακες το annotation @GenerateMetadata. Στο κώδικα του annotation processor έχουμε το αντίστοιχο annotation:
@SupportedAnnotationTypes(value= {"Tables.Annotation.GenerateMetadata"})
Αυτό ειδοποιεί την κλάση, προκειμένου να επεξεργαστεί τμήματα κώδικα, όπως άλλες κλάσεις για παράδειγμα, που περιέχουν το annotation GenerateMetadata.
Οι αρχικοποιήσεις της κλάσης γίνονται σε μια κλάση init. Η επεξεργασία γίνεται στην κλάση process. Αρχικά ελέγχονται όλα τα στοιχεία που βρίσκονται στο Project της εφαρμογής τα οποία περιέχουν το annotation GenerateMetadata.

Για αυτές τις κλάσεις φτιάχνεται ένα προσωρινό αντικείμενο αποθήκευσης δεδομένων, το MetaStorage και για κάθε πεδίο τους αποθηκεύεται η τιμή και ο τύπο της. Τέλος αποθηκεύεται το όνομα της κλάσης και το όνομα της κλάσης των μεταδεδομένων που θα δημιουργηθούν. Στο τελευταίο βήμα της επεξεργασίας αυτής παράγεται ένα αρχείο σύμφωνα με ένα πρότυπο και εισάγονται τα δεδομένα.

Προκειμένου να δημιουργηθούν τα αρχεία των μεταδεδομένων, γίνεται η χρήση ενός προτύπου. Τη διαδικασία της δημιουργίας αυτής την πετυχαίνουμε με ένα εργαλείο του EMF, το Jet. Το πρότυπο αυτό ονομάζεται metagen.javajet και βρίσκεται μέσα στο sqlProcessor.jar. Όταν χτίστηκε το sqlProcessor δημιούργησε ένα java αρχείο, το MetaSource.java, με χρήση του προτύπου. Στη συνέχεια καλείται το MetaSource για να παράγει τον κώδικα, ο οποίος στη συνέχεια αποθηκεύεται σε ένα JavaFileObject, με τη βοήθεια του Filer του annotation processor.

Παρακάτω παρουσιάζεται ένα παράδειγμα μεταδεδομένων που δημιουργείται από την παραπάνω διαδικασία και αντιστοιχεί στην κλάση Person :

```
package meta;

import Tables.Field;
import Tables.impl.TableImpl;
import Tables.Annotation.InitMetaData;


@InitMetaData
public class _Person extends TableImpl{

	public final String tableName = "Person";

	public static volatile Field<java.lang.Long,_Person> serialVersionUID ;

	public static volatile Field<java.util.List<sqlmodel_v2.entitiesTest.Hobby>,_Person> hobbies ;

	public static volatile Field<java.lang.String,_Person> name ;

	public static volatile Field<java.lang.Integer,_Person> id ;

	public static volatile Field<java.lang.Integer,_Person> hobby_id ;

}
```

Εδώ παρατηρείται ένα ακόμα annotation, το @InitMetaData. Τα δεδομένα αυτά είναι άδεια στην αρχή της εφαρμογής. Με το annotation αυτό τρέχει αντίστοιχος κώδικας και αρχικοποιεί τα δεδομένα των μεταβλητών αυτών που είναι τύπου Field. Περισσότερη ανάλυση θα γίνει σε επόμενη παράγραφο.

3.2.3 Δημιουργία Ερωτήματος

3.2.3.1 Αρχικοποίηση

Πριν χρησιμοποιηθούν τα αντικείμενα της επέκτασης ο χρήστης πρέπει να τα αρχικοποιήσει. Τα αντικείμενα που αρχικοποιούνται είναι τα πεδία Field στα μεταδεδομένα. Θεωρητικά θα μπορούσε και ο χρήστης να δώσει τιμές στα αντικείμενα αυτά αλλά για λόγους τυπογραφικής ασφάλειας το κάνει η επέκταση αυτό. Για κάθε πεδίο τα δεδομένα που θέλουμε να αρχικοποιήσουμε είναι το όνομα του πεδίου καθώς και ένα ψευδώνυμο για το πεδίο αυτό.
Η επέκταση στηρίζεται σε ψευδώνυμα για την αποφυγή κοινών ονομάτων πεδίων σε ένα ερώτημα. Ακόμα το όνομα του πεδίου πρέπει να είναι το ίδιο με αυτό του αντίστοιχου πεδίου, του πίνακα της βάσης. Για αυτό το λόγο αρχικοποιείται από την επέκταση και δεν δίνεται η δυνατότητα στον χρήστη να το αλλάξει. Από την άλλη ο χρήστης έχει τη δυνατότητα να αλλάξει το ψευδώνυμο του πεδίου. Σε αυτή την περίπτωση πρέπει να προσέξει έτσι ώστε να μην προκύψει κάποιο κοινό όνομα μεταξύ των πεδίων.

Για να αρχικοποιήσει τα δεδομένα ο χρήστης πρέπει να καλέσει την εντολή:

```
List<Class<?>> rs = InitializeMetaData.initializeFromPackage("meta");
```

Αυτή η εντολή βρίσκεται στο πακέτο «Tables.util.InitializeMetaData» της επέκτασης. Η συμβολοσειρά “meta” αναφέρεται στο πακέτο meta που βρίσκεται στην εφαρμογή και περιέχει τα μεταδεδομένα. Ο χρήστης μπορεί να ορίσει το πακέτο στο οποίο αποθηκεύονται τα μεταδεδομένα. Δεν είναι αποδοτικό να ελεγχθούν όλα τα πακέτα προγραμματιστικά.
Η συνάρτηση *initializeFromPackage* ελέγχει όλα τα αρχεία μέσα στο πακέτο και όσα έχουν το annotation InitMetaData ενημερώνονται.

Ακόμα χρησιμοποιείται η συνάρτηση find η οποία έχει σαν σκοπό την αναδρομική διείσδυση στους υποφακέλους του πακέτου για τον εντοπισμό όλων των κλάσεων. Στο τέλος έχουμε τη λίστα classes με όλες τις κλάσεις που βρίσκονται μέσα στο πακέτο.

Στη συνέχεια με την εντολή:

```
Annotation A = annot.getAnnotation(InitMetaData.class);
```

Ελέγχουμε ποιες κλάσεις είναι σημασμένες με το annotation InitMetaData και για αυτές τις κλάσεις χρησιμοποιούνται μέθοδοι ενδοσκόπησης (reflection) για την αρχικοποίηση των πεδίων τους.

3.2.3.2 Δημιουργία Αντικειμένων

Αφού ο χρήστης έχει φτιάξει τα JavaBeans που συσχετίζουν αντικείμενα με πίνακες στη βάση και έχει χτίσει την εφαρμογή, έχει στη διάθεσή του τα μεταδεδομένα. Στη συνέχεια μπορεί να δημιουργήσει αντικείμενα και να χτίσει το ερώτημα που θέλει να θέσει στη βάση.

Το μοντέλο των ερωτημάτων έχει χτιστεί με τη βοήθεια του EMF. Για αυτό η δημιουργία των αντικειμένων γίνεται από ένα Factory, το TablesFactory. Το Factory αυτό περιλαμβάνει μεθόδους κατασκευής για κάθε μη-αφηρημένη κλάση του μοντέλου.

Ρίζα του Ερωτήματος

Κάθε ερώτημα ξεκινάει από τον ορισμό του είδους του ερωτήματος. Υπάρχουν τέσσερα είδη ερωτημάτων. Αυτά είναι τα GET,PUT,POST,DELETE και αντιστοιχούν στα λεκτικά SELECT,INSERT, UPDATE και DELETE αντίστοιχα. Σε αυτή την έκδοση έχει υλοποιηθεί η ανάπτυξη ερωτημάτων τύπου GET.

Για την κατασκευή ενός ερωτήματος τύπου GET χρειάζεται να κληθεί το TablesFactory ως εξής:

```
QueryGET<_Person> qg2 = TablesFactory.eINSTANCE.createQueryGET(_Person.class);
```

Με αυτή την εντολή καλείται το TablesFactory, αρχικοποιείται με τις μεθόδους του EMF και στη συνέχεια καλεί τον constructor του αντικειμένου QueryGET. Εκτός του constructor όλες οι άλλες δημόσιες μέθοδοι γίνονται διαθέσιμες στον χρήστη μέσω του interface QueryGET.java ενώ η υλοποίησή τους γίνεται στο αρχείο QueryGETImpl.java. Αυτή η τακτική ακολουθείται για όλα τα αντικείμενα.

Το αντικείμενο QueryGET αποτελεί τη ρίζα ενός ερωτήματος τύπου GET. Σε αυτό το αντικείμενο περιέχονται ορίσματα όπως τα filter, join, rootAlias, selectedFields και union στα οποία αποθηκεύονται άλλα αντικείμενα του μοντέλου και αποτελούν βασικά συστατικά ενός ερωτήματος.

Όπως φαίνεται από το παραπάνω κομμάτι κώδικα το QueryGET που δημιουργείται είναι τύπου _Person. Αυτό είναι το μεταδεδομένο του πίνακα Person που χρησιμοποιείται ως τώρα για παράδειγμα. Το ερώτημα που δημιουργείται έχει ως ρίζα τον πίνακα Person. Ως εκ τούτου τα δεδομένα που εισάγονται από εδώ και πέρα στο ερώτημα, περιορίζονται σε αυτό το πίνακα.

Είναι δυνατό να αλλάξει το ψευδώνυμο του πίνακα με την εντολή:

```
qg2.setRootAlias("people");
```

Ο χρήστης πρέπει να ορίσει τα πεδία που θέλει να του επιστραφούν από το ερώτημα. Αυτό το πετυχαίνει με την εντολή:

```
qg2.getSelectedFields().add(_Person.name);
```

Όπως φαίνεται από την παραπάνω εντολή θα επιστραφεί η στήλη του πίνακα που ονομάζεται «name». Ο χρήστης δεν έχει τη δυνατότητα να εισάγει πεδίο που δεν ανήκει στον πίνακα Person γιατί η επέκταση κάνει εκτενή χρήση από generics για λόγους τυπογραφικής ασφάλειας. Αν για παράδειγμα ο χρήστης γράψει:

```
qg2.getSelectedFields().add(_Hobby.name);
```

τότε το Eclipse IDE θα εμφανίσει σφάλμα και το πρόγραμμα δεν θα μεταγλωττιστεί. Λόγο του Eclipse IDE γίνεται εκτενής χρήση αυτόματης συμπλήρωσης κώδικα, κάτι που διευκολύνει πολύ τον χρήστη στην δημιουργία των ερωτημάτων.

Προσθήκη Επιπλέον Πινάκων

Ο χρήστης έχει τη δυνατότητα να προσθέσει επιπλέον πίνακες στο ερώτημα ως ενώσεις μεταξύ πινάκων (join).

```
JOIN<_Person,_Hobby,Integer> join = TablesFactory.eINSTANCE.createJOIN(_Person.class, _Hobby.class);
```

Για την αρχικοποίηση ενός αντικειμένου JOIN ο χρήστης χρειάζεται να ορίσει τον αρχικό πίνακα, τον πίνακα που θέλει να συνδεθεί με τον αρχικό και τον τύπο του πεδίου που ενώνει τους δύο πίνακες στη βάση. Στη συγκεκριμένη περίπτωση ο αρχικός πίνακας είναι ο _Person, ο συνδεόμενος είναι ο _Hobby και ο τύπος του πεδίου που γίνεται η σύνδεση είναι Integer.

Στη συνέχεια μπορεί να ορίσει το ψευδώνυμο του πίνακα αν το επιθυμεί. Είναι απαραίτητο να ορίσει τα πεδία στα οποία γίνεται η ένωση όμως.

```
join.setJoinedAlias("hobby");
join.setOnFieldInitial(_Person.hobby_id);
join.setOnFieldJoined(_Hobby.id);
```

Το IDE ελέγχει αν οι μεταβλητές onFieldInitial και onFieldJoined είναι ίδιου τύπου. Σε αυτή την περίπτωση αν η μια μεταβλητή είναι τύπου String και η άλλη είναι τύπου Integer θα εμφανιστεί μήνυμα λάθους. Ακόμα η μεταβλητή onFieldInitial μπορεί να ανήκει μόνο στον πίνακα Person και η μεταβλητή onFieldJoined μπορεί να ανήκει μόνο στον πίνακα _Hobby. Για την αρχικοποίηση των αντικειμένων τύπου JOIN ο χρήστης συμβουλεύεται να χρησιμοποιήσει τα μεταδεδομένα που προκύπτουν στα προηγούμενα βήματα. Σε αντίθετη περίπτωση η επέκταση δεν θα λειτουργήσει ορθά.

Αφού έχει δημιουργηθεί το αντικείμενο JOIN πρέπει να συνδεθεί με τον πίνακα του αντικειμένου QueryGET. Αυτό επιτυγχάνεται με την εντολή:

```
qg2.getJoin().add(join);
```

Στην παραπάνω λίστα αποθηκεύονται όλοι οι πίνακες που ενώνονται με τον αρχικό. Ιδιαίτερο ενδιαφέρον παρουσιάζει η δυνατότητα της επέκτασης να χειρίζεται φωλιασμένες ενώσεις πινάκων. Αν έχουμε δύο αντικείμενα JOIN και θέλουμε να τα συνδέσουμε λειτουργούμε ως εξής:

```
JOIN<_Hobby,_Type,Integer> nestedjoin = TablesFactory.eINSTANCE.createJOIN(_Hobby.class, _Type.class);
nestedjoin.setJoinedAlias("type");
nestedjoin.setOnFieldInitial(_Hobby.type_id);
nestedjoin.setOnFieldJoined(_Type.id);
nestedjoin.getJoinedTableFields().add(_Type.name);
join.getJoinMore().add(nestedjoin);
```

Σε αυτή την περίπτωση το ρόλο του αρχικού πίνακα τον έχει ο πίνακας που αντιστοιχεί στο εξωτερικό JOIN ενώ του συνδεδεμένου ο πίνακας του αντικειμένου nestedjoin.

Δημιουργία φίλτρων

Τέλος ο χρήστης μπορεί να δημιουργήσει αντικείμενα φίλτρων που ονομάζονται Filters. Αυτά τα αντικείμενα περιέχουν πληροφορία για περιορισμούς που ενδεχομένως να θέλει ο χρήστης να εισάγει στο ερώτημα. Παρακάτω δίνεται ένα παράδειγμα απλού φίλτρου:

```
Filters<_Person> whereroot =
TablesFactory.eINSTANCE.createFilters(_Person.name,Operators.EQUALS,"jason",null);
```

Στο κάθε φίλτρο πρέπει να ορίζεται ο πίνακας στον οποίο πρέπει να εφαρμοστεί. Στη συγκεκριμένη περίπτωση ο πίνακας είναι ο Person. Στη συνέχεια πρέπει να ορίσει στον κατασκευαστή του αντικειμένου το πεδίο του πίνακα στο οποίο εφαρμόζεται το φίλτρο, η πράξη που θα πρέπει να γίνει και η αντίστοιχη τιμή που πρέπει να πάρει το πεδίο αυτό. Τα τρία τελευταία ορίσματα αποθηκεύονται σε ένα αντικείμενο τύπου condition και στη συνέχεια στο φίλτρο. Τέλος αν ο πίνακας ανήκει σε κάποιο JOIN αντικείμενο, πρέπει να προστεθεί το JOIN αυτό στο τελευταίο όρισμα όπωςδείχνεται στο παρακάτω παράδειγμα.

```
Filters<_Hobby> wherejasonhb = TablesFactory.eINSTANCE.createFilters(_Hobby.name, Operators.EQUALS,
"chess",join);
```

Μέσω του enumeration Operators είναι δυνατή η επισκόπηση των πράξεων που είναι δυνατό να γίνουν από την SQL.

Ο χρήστης μπορεί να αρχικοποιήσει το αντικείμενο Filters χωρίς να ορίσει αμέσως κάποια συνθήκη και να την εισάγει προγραμματιστικά σε μεταγενέστερο σημείο στον κώδικά του μέσω ενός αντικειμένου condition, αλλά αυτό δυσκολεύει την ανάγνωση του κώδικα και αυξάνει το φόρτο προγραμματισμού, οπότε και δε συνίσταται.

Αφού δημιουργηθούν τα αντικείμενα Filters, αν το πλήθος τους είναι μεγαλύτερο από ένα πρέπει να συνδεθούν. Ο χρήστης επιλέγει τη σειρά με την οποία θα γίνει η σύνδεση των φίλτρων καθώς και το αν τα φίλτρα είναι φωλιασμένα σε κάποιο άλλο φίλτρο. Αν και η διαδικασία είναι απλή, χρειάζεται προσοχή έτσι ώστε οι συσχετίσεις μεταξύ των φίλτρων να έχουν νόημα. Παρακάτω δίνεται ένα παράδειγμα φωλιασμού φίλτρων:

```
whereroot.and(wherejasonhb).or(whereeli).or(wheremarios);
```

Τέλος το φίλτρο πρέπει να εισαχθεί στο αντικείμενο του αρχικού πίνακα (queryGET)

```
qg2.setFilter(whereroot);
```

Με την ανάλυση των αντικειμένων που αντιστοιχούν στα φίλτρα ολοκληρώνεται η διαδικασία επεξεργασίας των αντικειμένων.

3.2.3.3 Εκτέλεση Ερωτήματος

Αφού έχουν δημιουργηθεί τα αντικείμενα με την ιεραρχία που παρουσιάζεται παραπάνω πρέπει να παραχθεί το ερώτημα με τη μορφή μιας συμβολοσειράς και να γίνει η επικοινωνία με τη βάση δεδομένων. Ο χρήστης αρκεί να καλέσει την εντολή:

```
List<?> returned = qg2.generate(Person.class);
```

Μόνη προϋπόθεση είναι να εισάγει την κλάση του πραγματικού αντικειμένου που εκφράζει τον αρχικό πίνακα. Δεν πρέπει να γίνεται σύγχυση με το μεταμοντέλο _Person. Η επέκταση χρειάζεται να γνωρίζει τι είδους αντικείμενο περιμένει να επιστραφεί στον χρήστη. Στο παραπάνω παράδειγμα ο χρήστης περιμένει μια λίστα από αντικείμενα τύπου Person.

Σε αυτή την έκδοση η λίστα είναι στην πραγματικότητα τύπου Object και χρειάζεται να γίνει cast σε μια λίστα τύπου Person:

```
for(Person prs: (List<Person>) returned){
```

3.2.3.3.1 Generate

Στη συνέχεια θα γίνει ανάλυση της μεθόδου generate. Η μέθοδος generate αποτελείται από τρία στάδια:

- Κατασκευή ερωτήματος
- Επικοινωνία με Βάση

- Επεξεργασία αποτελέσματος

3.2.3.3.2 Κατασκευή Ερωτήματος

Η κατασκευή θα γίνει μέσω ενός StringBuffer. Τα Strings κανονικά είναι αμετάβλητα και μόνο για ανάγνωση. Αυτό έχει ως αποτέλεσμα η επεξεργασία ενός String να είναι δαπανηρή διαδικασία. Εν αντιθέση ένας StringBuffer έχει τη δυνατότητα τροποποίησης της συμβολοσειράς του και αποτελεί τον προτεινόμενο τρόπο επεξεργασίας συμβολοσειρών .
Σε αυτό το στάδιο αρχικοποιείται ένα αντικείμενο τύπου StringBuffer και με διαδοχικές κλήσεις της μεθόδου append του StringBuffer εμπλουτίζεται το ερώτημα.

```
StringBuffer query = new StringBuffer("SELECT ");
```

Στη συνέχεια αναλύονται τα πεδία, ξεκινώντας από τα πεδία του αρχικού πίνακα. Το κάθε πεδίο εμφανίζεται με τη μορφή:

<ΠΙΝΑΚΑΣ>.<ΟΝΟΜΑ ΠΕΔΙΟΥ> AS <ΨΕΥΔΩΝΥΜΟ ΠΕΔΙΟΥ>.

Αν ο πίνακας έχει ψευδώνυμο στη θέση <ΠΙΝΑΚΑΣ> εμφανίζεται το ψευδώνυμο και σε αντίθετη περίπτωση το όνομα του πίνακα. Στο <ΟΝΟΜΑ ΠΕΔΙΟΥ> εμφανίζεται το όνομα του πεδίου και στο <ΨΕΥΔΩΝΥΜΟ ΠΕΔΙΟΥ> το ψευδώνυμο του πεδίου. Αυτό είναι σημαντικό γιατί δεν επιτρέπεται να υπάρχουν πεδία χωρίς ψευδώνυμα. Όπως αναφέρθηκε και στο τμήμα της αρχικοποίησης στοιχείων κάθε πεδίο έχει ψευδώνυμο. Ο χρήστης έχει τη δυνατότητα να το τροποποιήσει, αν το επιθυμεί, αλλά δεν πρέπει να παραμείνει κενό. Για ακόμα μεγαλύτερη ασφάλεια στο ψευδώνυμο που έχει οριστεί προστίθεται και ο κώδικας κατακερματισμού του αντικειμένου (QueryGET ή JOIN) στο οποίο ανήκει ο πίνακας.
Η ίδια διαδικασία εκτελείται στη συνέχεια για όλα τα αντικείμενα join που υπάρχουν. Κάθε αντικείμενο QueryGET ή JOIN έχει μια λίστα από Joins που αποτελούν τα παιδιά του. Αν αυτή η ιεραρχία θεωρηθεί ως ένα δέντρο, ακολουθείται μια αναζήτηση κατά βάθος με χρήση της μεθόδου joinFieldUnwrap. Με κάθε κλήση της μεθόδου αυτής γίνεται διερεύνηση στο δέντρο κατά μία βαθμίδα πιο κάτω.
Μετά την ανάλυση των πεδίων πρέπει να προστεθούν οι πίνακες που χρησιμοποιούνται. Αρχικά προστίθεται το λεκτικό FROM:

```
query.append(" FROM ");
```

Η διαδικασία είναι παρόμοια με αυτή που χρησιμοποιήθηκε για την εισαγωγή. Αν ο πίνακας είναι η ρίζα του ερωτήματος, τότε εμφανίζεται στη μορφή:

<ΠΙΝΑΚΑΣ> AS <ΨΕΥΔΩΝΥΜΟ ΠΙΝΑΚΑ>

Ενώ σε αντίθετη περίπτωση, που αποτελεί σύνδεση μέσω λεκτικού join εμφανίζεται με τη μορφή:

JOIN <ΠΙΝΑΚΑΣ> AS <ΨΕΥΔΩΝΥΜΟ ΠΙΝΑΚΑ> ON <ΠΡΩΤΟ ΠΕΔΙΟ>.<ΠΙΝΑΚΑΣ ΡΙΖΑ> = <ΔΕΥΤΕΡΟ ΠΕΔΙΟ>.<ΠΙΝΑΚΑΣ>

Σε κάθε περίπτωση το λεκτικό AS <ΨΕΥΔΩΝΥΜΟ ΠΙΝΑΚΑ> παραλείπεται εάν δεν έχει δοθεί ψευδώνυμο στον πίνακα. Το <ΠΡΩΤΟ ΠΕΔΙΟ> είναι το πεδίο του πίνακα ρίζα πάνω στο οποίο

γίνεται η πράξη της συνένωσης, ενώ το <ΔΕΥΤΕΡΟ ΠΕΔΙΟ> είναι το πεδίο του συνδεόμενου πίνακα πάνω στο οποίο γίνεται η συνένωση.

Το τελευταίο δομικό στοιχείο του ερωτήματος που πρέπει να προστεθεί είναι τα φίλτρα του, εφόσον και αυτά υπάρχουν. Όλα τα φίλτρα του ερωτήματος είναι αποθηκευμένα στο αντικείμενο QueryGET και πιο συγκεκριμένα φωλιασμένα σε ένα αντικείμενο Filters<…>.
Η πρόσβαση στα φίλτρα γίνεται μέσω της συνάρτησης resolveFilter(…). Αυτή η συνάρτηση κάθε φορά που καλείται προσθέτει στο ερώτημα το όνομα του φίλτρου, την πράξη που εκτελείται και την τιμή με την οποία εκτελείται η πράξη. Στη συνέχεια καλεί αναδρομικά τον εαυτό της για όλα υπόλοιπα φίλτρα.
Ανάλογα με τη θέση στην οποία έχει εισαχθεί κάποιο φίλτρο μπορεί να θεωρηθεί φωλιασμένο σε κάποιο προηγούμενό του. Φωλιασμένα φίλτρα περιβάλλονται από παρενθέσεις για την καλύτερη κατανόησή τους από τον χρήστη.

Σε αυτό το σημείο ολοκληρώνεται η κατασκευή του ερωτήματος. Στη συνέχεια θα γίνει η σύνδεση με τη βάση.

3.2.3.3.3 Επικοινωνία με τη Βάση

Αυτό αποτελεί ένα πολύ μικρό τμήμα της συνάρτησης και περιλαμβάνει απλά την επικοινωνία με τη βάση. Ο χρήστης ορίζει το όνομα της βάσης, το όνομα του χρήστη και τον κατάλληλο κωδικό, ανοίγει μια σύνδεση και εκτελεί το ερώτημα.

```
Connection con = DriverManager.getConnection(host, uName, uPass);
Statement stmt = con.createStatement();
```

Στη συνέχεια απομένει το τελευταίο στάδιο της generate() που είναι και η επεξεργασία των δεδομένων που επιστρέφονται από το παραπάνω ερώτημα.

3.2.3.3.4 Επεξεργασία Αποτελέσματος

Κατά την επεξεργασία των δεδομένων, το αποτέλεσμα του ερωτήματος του προηγούμενου σταδίου αναλύεται και τοποθετείται σε αντικείμενα που προέρχονται από τις κλάσεις που χρησιμοποιεί ο χρήστης για να υλοποιήσει την ORM λογική της εφαρμογής του, δηλαδή τις οντότητες που έχει ορίσει ο χρήστης.

Κάθε αντικείμενο QueryGET ή JOIN έχει έναν πίνακα κατακερματισμού που ονομάζεται resultMap. Σε αυτό το πίνακα κατακερματισμού αποθηκεύονται τα αντικείμενα που αντιστοιχούν στους πίνακες της βάσης δεδομένων.
Για κάθε γραμμή του ResultSet (αποτέλεσμα που επιστρέφει η βάση) γίνεται απομόνωση των πεδίων του κάθε πίνακα, αποθήκευση των τιμών στο αντίστοιχο αντικείμενο-οντότητα και στο τέλος συνένωση των αντικειμένων.

Αρχικά δημιουργείται το κλειδί του αντικειμένου:

```
String key = generateKey(resset,null,null);
```

Αυτή η συνάρτηση παίρνει ως παραμέτρους τη γραμμή του αποτελέσματος ResultSet που εξετάζεται, το αντικείμενο του πίνακα στον οποίο αναφερόμαστε και το προηγούμενο κλειδί αν υπάρχει.
Βάση των πεδίων του πίνακα και το προηγούμενο κλειδί κατασκευάζεται το νέο κλειδί για το resultMap.

Αν το κλειδί βρίσκεται στο resultMap ήδη σημαίνει πως το ίδιο αντικείμενο έχει ξαναεμφανιστεί στο παρελθόν και δε χρειάζεται να δημιουργηθεί. Σε αντίθετη περίπτωση καλείται η συνάρτηση:

```
Object obj = createObject(clazz, resset,null);
```

Στη συνέχεια ελέγχονται όλοι οι συνδεδεμένοι πίνακες με τη συνάρτηση:

```
resolveJoin(clazz, join, resset, this.joinsOfTables,key);
```

Αυτή η συνάρτηση κρατάει το κλειδί που έχει δημιουργηθεί ως τώρα έτσι ώστε να παράγει νέα κλειδιά για το καινούριο αντικείμενο join.

Αφού έχουν παραχθεί τα αντικείμενα και βρίσκονται στα αντίστοιχα resultMaps των αντικειμένων JOIN και QueryGET γίνεται η ένωση των αντικειμένων.

Για τη συνένωση των αντικειμένων καλείται η συνάρτηση combiner:

```
combiner(clazz);
```

Αυτή η συνάρτηση καθώς και μια υποσυνάρτηση, η inner_combiner καλούνται αναδρομικά και ξεκινόντας από το queryGET και χρησιμοποιόντας το hashMap joinsOfTables ως αναφορά βρίσκουμε τους πίνακες JOIN που δεν περιέχουν άλλα JOINS. Αυτοί οι πίνακες θεωρούνται φύλλα του δέντρου των πινάκων. Τα πεδία του resultMap, το οποίο βρίσκεται μέσα στα φύλλα αυτά, τοποθετούνται μέσα στα πεδία του resultMap του πατέρα του φύλλου. Αφού κάποιο JOIN έχει επεξεργαστεί όλα τα παιδιά του λειτουργεί ως φύλλο για τον πατέρα του και επαναλαμβάνεται η διαδικασία που περιγράφηκε για τα φύλλα. Όταν επεξεργαστούν και όλα τα παιδιά του πίνακα που ανήκει στο QueryGET έχει τελειώσει και η επεξεργασία του ResultSet.

Στην ουσία η επεξεργασία των δεδομένων αποτελείται από δύο βασικά στάδια. Χρησιμοποιούντας το ResultSet με top-down επεξεργασία του δέντρου που ορίζεται από το joinsOfTables δημιουργούνται τα αντικείμετα οντοτήτων και αποθηκεύονται στα resultMaps των JOINς και του QueryGET. Στη συνέχεια με bottom-up λογικήτα resultMaps που βρίσκονται στα διάφορα αντικείμενα ενόνονται σε ένα τελικά resultMap που βρίσκεται στο QueryGET.

3.2.3.4 Επεξεργασία Δεδομένων

Όταν επιστρέψει η generate το αποτέλεσμά της, ο χρήστης έχει στη διάθεσή του μια λίστα από αντικείμενα πινάκων. Πριν μπορέσει να το επεξεργαστεί πρέπει να το κάνει cast σε μια λίστα στην οποία πρέπει να ορίσει για τύπο, τον τύπο του πίνακα ρίζα:

```
(List<Person>) returned
```

3.2.3.5 Διάγραμμα ακολουθίας

Η παραπάνω διαδικασία, από την εισαγωγή των οντοτήτων, έως και τη λήψη του τελικού αποτελέσματος περιγράφεται και με το διάγραμμα ακολουθίας που παρουσιάζεται παρακάτω. Σε αυτό το διάγραμμα λαμβάνουν ενέργειες οι χρήστες(users), η εφαρμογή (application), το εργοστάσιο που παράγει τα αντικείμενα της επέκτασης (model factory), ένα σύστημα συσχέτισης αντικειμένων και σχέσεων (ORM), όπως παράδειγμα το JPA, μια βάση δεδομένων (database) και ο μεταγλωττιστής (compiler).
Ο χρήστης αρχικά ζητάει τις οντότητές του μέσω του ORM, στη συνέχεια κατασκευάζει τα μεταδεδομένα και τέλος τα αρχικοποιεί και δημιουργεί το ερώτημα στη βάση. Η πιο ορθή χρήση της επέκτασης είναι με την εφαρμογή της ακολουθίας που παρουσιάζεται στο διάγραμμα. Στο τμήμα που δημιουργεί το ερώτημα, μετά την εντολή Initialize metadata, μπορεί να ακολουθήσει διαφορετική σειρά στη δημιουργία των αντικειμένων, αλλά η προτεινόμενη είναι και η πιο ασφαλής, καθώς δίνει στον κώδικα μια λογική συνοχή.

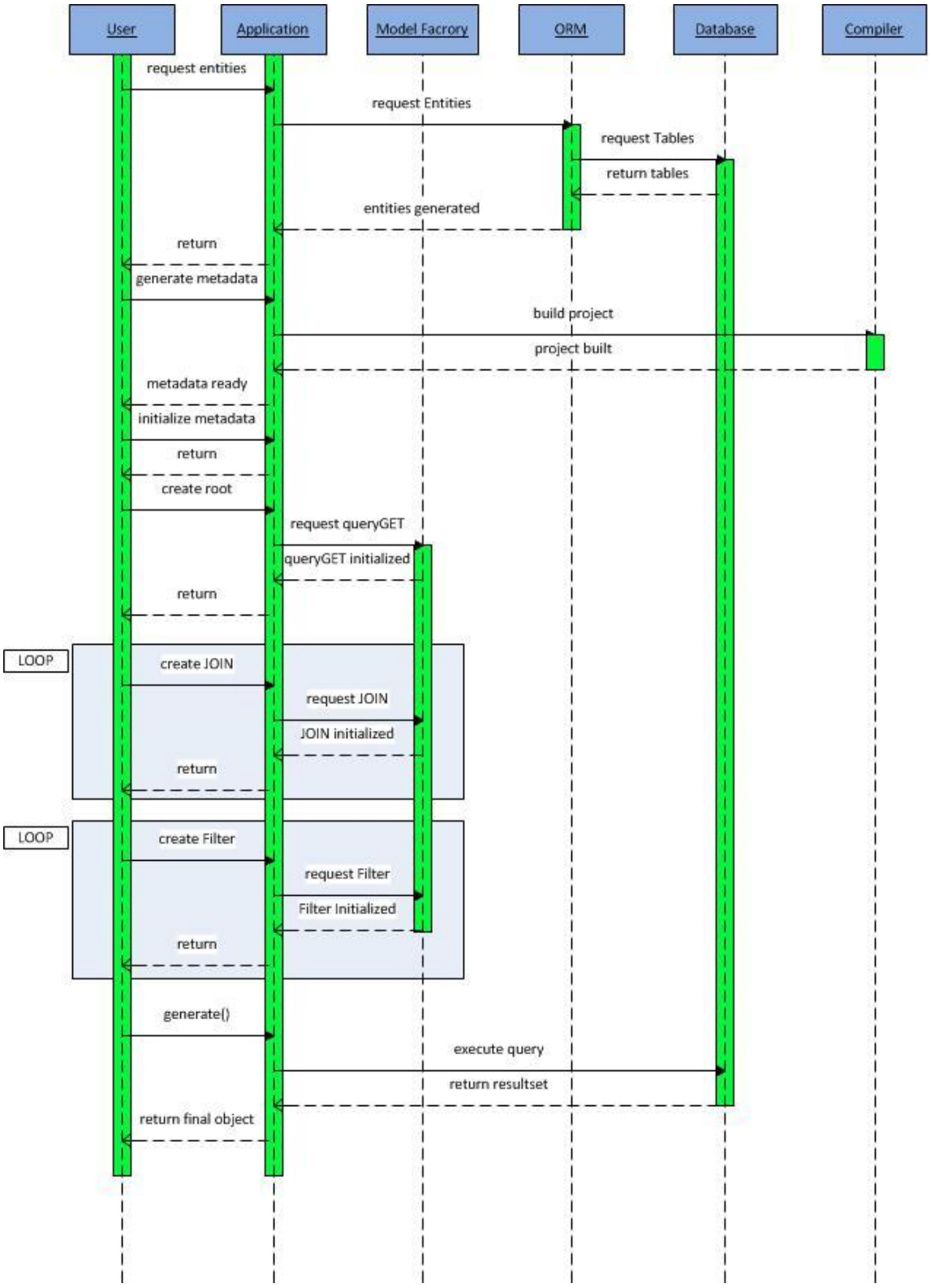

Figure 16: Sequence Diagram of application

3.2.3.6 Διάγραμμα ροής ερωτήματος

Εκτός από το διάγραμμα ακολουθίας δίνεται και ένα διάγραμμα ροής για την ορθή κατασκευή του ερωτήματος. Ο χρήστης αρχικά οφείλει να αρχικοποιήσει τα μετα-δεδομένα του, στη συνέχεια να ορίσει μια ρίζα-πίνακα από την οποία θα εκκινεί το ερώτημα, μετά να ορίσει τις συνενώσεις πινάκων και τέλος να θέσει τους περιορισμούς.

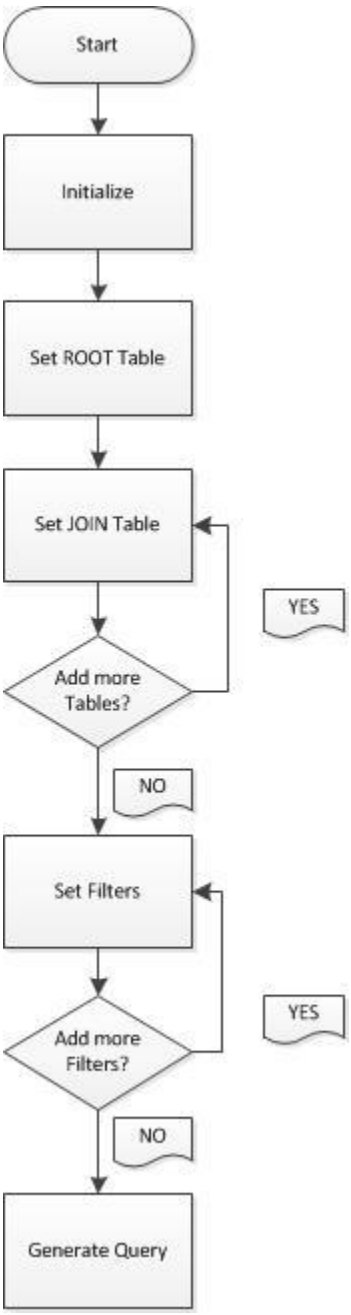

Figure 17: Flow Diagram of application

3.3 Υλοποίηση Κώδικα

Σε αυτή την παράγραφο παρατίθεται ο κώδικας των διαφόρων τμημάτων:

3.3.1 Field

```
package Tables;

import org.eclipse.emf.ecore.EObject;

/**
 * <!-- begin-user-doc -->
 * Describes a column of a table in the database
 * <!-- end-user-doc -->
 * @see Tables.TablesPackage#getField()
 * @model
 * @generated
 */
public interface Field<T, TableInstance extends Table> extends EObject {
		/**
		 * Returns the value of the '<em><b>Field Name</b></em>' attribute.
		 * @return the value of the '<em>Field Name</em>' attribute.
		 * @see #setFieldName(String)
		 * @see Tables.TablesPackage#getField_FieldName()
		 * @model
		 * @generated
		 */
		String getFieldName();

		/**
		 * Sets the value of the '{@link Tables.Field#getFieldName <em>Field Name</em>}' attribute.
		 * @param value the new value of the '<em>Field Name</em>' attribute.
		 * @see #getFieldName()
		 * @generated
		 */
		void setFieldName(String value);

		/**
		 * Returns the value of the '<em><b>Alias</b></em>' attribute.
		 * @return the value of the '<em>Alias</em>' attribute.
		 * @see #setAlias(String)
		 * @see Tables.TablesPackage#getField_Alias()
		 * @model
		 * @generated
		 */
		String getAlias();

		/**
		 * Sets the value of the '{@link Tables.Field#getAlias <em>Alias</em>}' attribute.
		 * @param value the new value of the '<em>Alias</em>' attribute.
		 * @see #getAlias()
		 * @generated
		 */
		void setAlias(String value);

} // Field
```

Field Interface

```
package Tables.impl;

import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.impl.ENotificationImpl;
```

```
import org.eclipse.emf.ecore.impl.MinimalEObjectImpl;
import Tables.Field;
import Tables.Table;
import Tables.TablesPackage;

/**
 * <!-- begin-user-doc -->
 * An implementation of the model object '<em><b>Field</b></em>'.
 * <!-- end-user-doc -->
 * @generated
 */
public class FieldImpl<T, TableInstance extends Table> extends MinimalEObjectImpl.Container implements Field<T, TableInstance> {
		/**
		 * The default value of the '{@link #getFieldName() <em>Field Name</em>}' attribute.
		 * @generated
		 * @ordered
		 */
		protected static final String FIELD_NAME_EDEFAULT = null;

		/**
		 * The cached value of the '{@link #getFieldName() <em>Field Name</em>}' attribute.
		 * @generated
		 * @ordered
		 */
		protected String fieldName = FIELD_NAME_EDEFAULT;

		/**
		 * The default value of the '{@link #getAlias() <em>Alias</em>}' attribute.
		 * @generated
		 * @ordered
		 */
		protected static final String ALIAS_EDEFAULT = null;

		/**
		 * The cached value of the '{@link #getAlias() <em>Alias</em>}' attribute.
		 * @generated
		 * @ordered
		 */
		protected String alias = ALIAS_EDEFAULT;

		/**
		 * <!-- begin-user-doc -->
		 * Simple constructor for the Field object
		 * <!-- end-user-doc -->
		 * @generated NOT
		 */
		public FieldImpl() {
			super();
		}

		/**
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @generated
		 */
		@Override
		protected EClass eStaticClass() {
			return TablesPackage.Literals.FIELD;
		}

		/**
		 * <!-- begin-user-doc -->
		 * returns the fieldName
		 * <!-- end-user-doc -->
		 * @generated
		 */
		public String getFieldName() {
```

```
                return fieldName;
        }

        /**
         * <!-- begin-user-doc -->
         * Updates the 81ieldname
         * <!—end-user-doc →
         * @generated
         */
        public void setFieldName(String newFieldName) {
                String oldFieldName = 81ieldname;
                81ieldname = newFieldName;
                if (eNotificationRequired())
                        eNotify(new EnotificationImpl(this, Notification.SET, TablesPackage.FIELD__FIELD_NAME,
oldFieldName, 81ieldname));
        }

        /**
         * <!—begin-user-doc →
         * <!—end-user-doc →
         * @generated
         */
        public String getAlias() {
                return alias;
        }

        /**
         * <!—begin-user-doc →
         * <!—end-user-doc →
         * @generated
         */
        public void setAlias(String newAlias) {
                String oldAlias = alias;
                alias = newAlias;
                if (eNotificationRequired())
                        eNotify(new EnotificationImpl(this, Notification.SET, TablesPackage.FIELD__ALIAS, oldAlias, alias));
        }

        /**
         * <!—begin-user-doc →
         * <!—end-user-doc →
         * @generated
         */
        @Override
        public Object eGet(int featureID, 81ieldna resolve, 81ieldna coreType) {
                switch (featureID) {
                        case TablesPackage.FIELD__FIELD_NAME:
                                return getFieldName();
                        case TablesPackage.FIELD__ALIAS:
                                return getAlias();
                }
                return super.eGet(featureID, resolve, coreType);
        }

        /**
         * <!—begin-user-doc →
         * <!—end-user-doc →
         * @generated
         */
        @SuppressWarnings("unchecked")
        @Override
        public void eSet(int featureID, Object newValue) {
                switch (featureID) {
                        case TablesPackage.FIELD__FIELD_NAME:
                                setFieldName((String)newValue);
                                return;
                        case TablesPackage.FIELD__ALIAS:
```

```
                                        setAlias((String)newValue);
                                        return;
                        }
                        super.eSet(featureID, newValue);
        }

        /**
         * <!—begin-user-doc →
         * <!—end-user-doc →
         * @generated
         */
        @Override
        public void eUnset(int featureID) {
                switch (featureID) {
                        case TablesPackage.FIELD__FIELD_NAME:
                                setFieldName(FIELD_NAME_EDEFAULT);
                                return;
                        case TablesPackage.FIELD__ALIAS:
                                setAlias(ALIAS_EDEFAULT);
                                return;
                }
                super.eUnset(featureID);
        }

        /**
         * <!—begin-user-doc →
         * <!—end-user-doc →
         * @generated
         */
        @Override
        public 82ieldna eIsSet(int featureID) {
                switch (featureID) {
                        case TablesPackage.FIELD__FIELD_NAME:
                                return FIELD_NAME_EDEFAULT == null ? 82ieldname != null :
!FIELD_NAME_EDEFAULT.equals(82ieldname);
                        case TablesPackage.FIELD__ALIAS:
                                return ALIAS_EDEFAULT == null ? alias != null : !ALIAS_EDEFAULT.equals(alias);
                }
                return super.eIsSet(featureID);
        }

        /**
         * <!—begin-user-doc →
         * <!—end-user-doc →
         * @generated
         */
        @Override
        public String toString() {
                if (eIsProxy()) return super.toString();

                StringBuffer result = new StringBuffer(super.toString());
                result.append(" (FieldName: ");
                result.append(82ieldname);
                result.append(", alias: ");
                result.append(alias);
                result.append(')');
                return result.toString();
        }

} //FieldImpl
```

Field Implementation

3.3.2 QueryGET

```
/**
 */
package Tables;

import java.util.List;

import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.ecore.EObject;

/**
 * <!-- begin-user-doc -->
 * A representation of the model object '<em><b>Query GET</b></em>'.
 * <!-- end-user-doc -->
 * @see Tables.TablesPackage#getQueryGET()
 * @model
 * @generated
 */
public interface QueryGET<TableInstance extends Table> extends EObject {
		/**
		 * Returns the value of the '<em><b>Selected Fields</b></em>' reference list.
		 * The list contents are of type {@link Tables.Field}&lt;?, TableInstance>.
		 * @return the value of the '<em>Selected Fields</em>' reference list.
		 * @model
		 * @generated
		 */
		EList<Field<?, TableInstance>> getSelectedFields();

		/**
		 * Returns the value of the '<em><b>Join</b></em>' reference list.
		 * The list contents are of type {@link Tables.JOIN}&lt;TableInstance, ?, ?>.
		 * @return the value of the '<em>Join</em>' reference list.
		 * @model
		 * @generated
		 */
		EList<JOIN<TableInstance, ?, ?>> getJoin();

		/**
		 * Returns the value of the '<em><b>Filter</b></em>' reference.
		 * <!-- begin-user-doc -->
		 * @return the value of the '<em>Filter</em>' reference.
		 * @model
		 * @generated
		 */
		Filters<TableInstance> getFilter();

		/**
		 * Sets the value of the '{@link Tables.QueryGET#getFilter <em>Filter</em>}' reference.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @param value the new value of the '<em>Filter</em>' reference.
		 * @generated
		 */
		void setFilter(Filters<TableInstance> value);

		/**
		 * Returns the value of the '<em><b>Root Table</b></em>' attribute.
		 * @return the value of the '<em>Root Table</em>' attribute.
		 * @model unsettable="true" required="true" transient="true" changeable="false" derived="true"
		 * @generated
		 */
		String getRootTable();

		/**
		 * Returns whether the value of the '{@link Tables.QueryGET#getRootTable <em>Root Table</em>}' attribute is set.
```

```
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @return whether the value of the '<em>Root Table</em>' attribute is set.
	 * @generated
	 */
	boolean isSetRootTable();

	/**
	 * Returns the value of the '<em><b>Root Alias</b></em>' attribute.
	 * @return the value of the '<em>Root Alias</em>' attribute.
	 * @see #setRootAlias(String)
	 * @model
	 * @generated
	 */
	String getRootAlias();

	/**
	 * Sets the value of the '{@link Tables.QueryGET#getRootAlias <em>Root Alias</em>}' attribute.
	 * @param value the new value of the '<em>Root Alias</em>' attribute.
	 * @generated
	 */
	void setRootAlias(String value);

	/**
	 * @model joinRequired="true" tableRequired="true" fieldRequired="true"
	 * @generated
	 */
	void addJoin(JOIN<TableInstance, ?, ?> join, TableInstance table, Field<?, TableInstance> field);

	/**
	 * @model filtersRequired="true" filtersMany="false"
	 * @generated
	 */
	void setFilter(EList<Filters> filters);

	/**
	 * @model clazzRequired="true"
	 * @generated NOT
	 */
	List<Object> generate(Class<?> clazz);

	/**
	 * Returns the value of the '<em><b>Union</b></em>' containment reference list.
	 * The list contents are of type {@link Tables.QueryGET}&lt;?>.
	 * @return the value of the '<em>Union</em>' containment reference list.
	 * @model containment="true"
	 * @generated
	 */
	EList<QueryGET<?>> getUnion();
} // QueryGET
```

QueryGET Interface

```
/**
 */
package Tables.impl;


import java.lang.reflect.ParameterizedType;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
```

```
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.common.notify.NotificationChain;
import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;
import org.eclipse.emf.ecore.impl.ENotificationImpl;
import org.eclipse.emf.ecore.impl.MinimalEObjectImpl;
import org.eclipse.emf.ecore.util.EObjectContainmentEList;
import org.eclipse.emf.ecore.util.EObjectResolvingEList;
import org.eclipse.emf.ecore.util.InternalEList;

import Tables.Field;
import Tables.Filters;
import Tables.JOIN;
import Tables.NaryNode;
import Tables.QueryGET;
import Tables.Table;
import Tables.TablesPackage;

/**
 * <!-- begin-user-doc -->
 * An implementation of the model object '<em><b>Query GET</b></em>'.
 * <!-- end-user-doc -->
 * <p>
 * The following features are implemented:
 * <ul>
 *   <li>{@link Tables.impl.QueryGETImpl#getUnion <em>Union</em>}</li>
 *   <li>{@link Tables.impl.QueryGETImpl#getSelectedFields <em>Selected Fields</em>}</li>
 *   <li>{@link Tables.impl.QueryGETImpl#getRootTable <em>Root Table</em>}</li>
 *   <li>{@link Tables.impl.QueryGETImpl#getRootAlias <em>Root Alias</em>}</li>
 * </ul>
 * </p>
 *
 * @generated
 */
public class QueryGETImpl<TableInstance extends Table> extends MinimalEObjectImpl.Container implements
QueryGET<TableInstance> {
		/**
		 * The cached value of the '{@link #getUnion() <em>Union</em>}' containment reference list.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @see #getUnion()
		 * @generated
		 * @ordered
		 */
		protected EList<QueryGET<?>> union;

		/**
		 * The cached value of the '{@link #getSelectedFields() <em>Selected Fields</em>}' reference list.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @see #getSelectedFields()
		 * @generated
		 * @ordered
		 */
		protected EList<Field<?, TableInstance>> selectedFields;

		/**
		 * The cached value of the '{@link #getJoin() <em>Join</em>}' reference list.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @see #getJoin()
		 * @generated
```

```
     * @ordered
     */
    protected EList<JOIN<TableInstance, ?, ?>> join;

    /**
     * The cached value of the '{@link #getFilter() <em>Filter</em>}' reference.
     * <!-- begin-user-doc -->
     * <!-- end-user-doc -->
     * @see #getFilter()
     * @generated
     * @ordered
     */
    protected Filters<TableInstance> filter;

    /**
     * The default value of the '{@link #getRootTable() <em>Root Table</em>}' attribute.
     * <!-- begin-user-doc -->
     * <!-- end-user-doc -->
     * @see #getRootTable()
     * @generated
     * @ordered
     */
    protected static final String ROOT_TABLE_EDEFAULT = null;

    /**
     * The cached value of the '{@link #getRootTable() <em>Root Table</em>}' attribute.
     * <!-- begin-user-doc -->
     * <!-- end-user-doc -->
     * @see #getRootTable()
     * @generated
     * @ordered
     */
    protected String rootTable = ROOT_TABLE_EDEFAULT;

    protected Class<TableInstance> rootInstance;

    /**
     * This is true if the Root Table attribute has been set.
     * <!-- begin-user-doc -->
     * <!-- end-user-doc -->
     * @generated
     * @ordered
     */
    protected boolean rootTableESet;

    /**
     * The default value of the '{@link #getRootAlias() <em>Root Alias</em>}' attribute.
     * <!-- begin-user-doc -->
     * <!-- end-user-doc -->
     * @see #getRootAlias()
     * @generated
     * @ordered
     */
    protected static final String ROOT_ALIAS_EDEFAULT = null;

    /**
     * The cached value of the '{@link #getRootAlias() <em>Root Alias</em>}' attribute.
     * <!-- begin-user-doc -->
     * <!-- end-user-doc -->
     * @see #getRootAlias()
     * @generated
     * @ordered
     */
    protected String rootAlias = ROOT_ALIAS_EDEFAULT;

    /**
     * The tablesToAlias creates a mapping between
```

```
	 * the actual name of the Table and its
	 * coresponding alias
	 */
	private Map<String,String> tablesToAlias;
	/**
	 * resultMap is the final map that will be filled
	 * with data returned from the database querying
	 */
	private Map<String,Object> resultMap;
	/**
	 * Mapping of the joins between the Tables
	 * Gives information on which Table is joined
	 * with which other Table
	 * key(String)		-> name of the Table
	 * value(List)		->the tables that are joined with the key
	 *					through the join objects
	 */
	private Map<String,List<JOIN<?,?,?>>> joinsOfTables;
	/**
	 * <!-- begin-user-doc -->
	 * Simple constructor for the QueryGET
	 * Invoked through the TableFactory
	 * Mostly for syntactic sugar. Use is not
	 * recommended
	 * <!-- end-user-doc -->
	 * @generated
	 */
	protected QueryGETImpl() {
		super();
	}

	/**
	 * <!-- begin-user-doc -->
	 * Constructor for QueryGET
	 * Invoked through the TablesFactory
	 * Appropriate for use instead of the simple
	 * constructor
	 * <!-- end-user-doc -->
	 * @generated NOT
	 */
	protected QueryGETImpl(Class<TableInstance> tableInstance) {
		super();
		this.rootTable = tableInstance.getSimpleName().substring(1);
		this.rootInstance = tableInstance;

	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	@Override
	protected EClass eStaticClass() {
		return TablesPackage.Literals.QUERY_GET;
	}


	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public EList<Field<?, TableInstance>> getSelectedFields() {
		if (selectedFields == null) {
			selectedFields = new EObjectResolvingEList<Field<?, TableInstance>>(Field.class, this,
TablesPackage.QUERY_GET__SELECTED_FIELDS);
```

```
                }
                return selectedFields;
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        public EList<JOIN<TableInstance, ?, ?>> getJoin() {
                if (join == null) {
                        join = new EObjectResolvingEList<JOIN<TableInstance, ?, ?>>(JOIN.class, this,
TablesPackage.QUERY_GET__JOIN);
                }
                return join;
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @SuppressWarnings("unchecked")
        public Filters<TableInstance> getFilter() {
                if (filter != null && filter.eIsProxy()) {
                        InternalEObject oldFilter = (InternalEObject)filter;
                        filter = (Filters<TableInstance>)eResolveProxy(oldFilter);
                        if (filter != oldFilter) {
                                if (eNotificationRequired())
                                        eNotify(new ENotificationImpl(this, Notification.RESOLVE,
TablesPackage.QUERY_GET__FILTER, oldFilter, filter));
                        }
                }
                return filter;
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        public Filters<TableInstance> basicGetFilter() {
                return filter;
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        public void setFilter(Filters<TableInstance> newFilter) {
                Filters<TableInstance> oldFilter = filter;
                filter = newFilter;
                if (eNotificationRequired())
                        eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.QUERY_GET__FILTER, oldFilter,
filter));
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        public String getRootTable() {
                return rootTable;
        }
```

```
	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public boolean isSetRootTable() {
		return rootTableESet;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public String getRootAlias() {
		return rootAlias;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public void setRootAlias(String newRootAlias) {
		String oldRootAlias = rootAlias;
		rootAlias = newRootAlias;
		if (eNotificationRequired())
			eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.QUERY_GET__ROOT_ALIAS,
oldRootAlias, rootAlias));
	}


	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public EList<QueryGET<?>> getUnion() {
		if (union == null) {
			union = new EObjectContainmentEList<QueryGET<?>>(QueryGET.class, this,
TablesPackage.QUERY_GET__UNION);
		}
		return union;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	@Override
	public NotificationChain eInverseRemove(InternalEObject otherEnd, int featureID, NotificationChain msgs) {
		switch (featureID) {
			case TablesPackage.QUERY_GET__UNION:
				return ((InternalEList<?>)getUnion()).basicRemove(otherEnd, msgs);
		}
		return super.eInverseRemove(otherEnd, featureID, msgs);
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	@Override
	public Object eGet(int featureID, boolean resolve, boolean coreType) {
		switch (featureID) {
```

```
                        case TablesPackage.QUERY_GET__UNION:
                                return getUnion();
                        case TablesPackage.QUERY_GET__SELECTED_FIELDS:
                                return getSelectedFields();
                        case TablesPackage.QUERY_GET__JOIN:
                                return getJoin();
                        case TablesPackage.QUERY_GET__FILTER:
                                if (resolve) return getFilter();
                                return basicGetFilter();
                        case TablesPackage.QUERY_GET__ROOT_TABLE:
                                return getRootTable();
                        case TablesPackage.QUERY_GET__ROOT_ALIAS:
                                return getRootAlias();
                }
                return super.eGet(featureID, resolve, coreType);
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @SuppressWarnings("unchecked")
        @Override
        public void eSet(int featureID, Object newValue) {
                switch (featureID) {
                        case TablesPackage.QUERY_GET__UNION:
                                getUnion().clear();
                                getUnion().addAll((Collection<? extends QueryGET<?>>)newValue);
                                return;
                        case TablesPackage.QUERY_GET__SELECTED_FIELDS:
                                getSelectedFields().clear();
                                getSelectedFields().addAll((Collection<? extends Field<?, TableInstance>>)newValue);
                                return;
                        case TablesPackage.QUERY_GET__JOIN:
                                getJoin().clear();
                                getJoin().addAll((Collection<? extends JOIN<TableInstance, ?, ?>>)newValue);
                                return;
                        case TablesPackage.QUERY_GET__FILTER:
                                setFilter((Filters<TableInstance>)newValue);
                                return;
                        case TablesPackage.QUERY_GET__ROOT_ALIAS:
                                setRootAlias((String)newValue);
                                return;
                }
                super.eSet(featureID, newValue);
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @Override
        public void eUnset(int featureID) {
                switch (featureID) {
                        case TablesPackage.QUERY_GET__UNION:
                                getUnion().clear();
                                return;
                        case TablesPackage.QUERY_GET__SELECTED_FIELDS:
                                getSelectedFields().clear();
                                return;
                        case TablesPackage.QUERY_GET__JOIN:
                                getJoin().clear();
                                return;
                        case TablesPackage.QUERY_GET__FILTER:
                                setFilter((Filters<TableInstance>)null);
```

```
                                        return;
                                case TablesPackage.QUERY_GET__ROOT_ALIAS:
                                        setRootAlias(ROOT_ALIAS_EDEFAULT);
                                        return;
                        }
                        super.eUnset(featureID);
                }

                /**
                 * <!-- begin-user-doc -->
                 * <!-- end-user-doc -->
                 * @generated
                 */
                @Override
                public boolean eIsSet(int featureID) {
                        switch (featureID) {
                                case TablesPackage.QUERY_GET__UNION:
                                        return union != null && !union.isEmpty();
                                case TablesPackage.QUERY_GET__SELECTED_FIELDS:
                                        return selectedFields != null && !selectedFields.isEmpty();
                                case TablesPackage.QUERY_GET__JOIN:
                                        return join != null && !join.isEmpty();
                                case TablesPackage.QUERY_GET__FILTER:
                                        return filter != null;
                                case TablesPackage.QUERY_GET__ROOT_TABLE:
                                        return isSetRootTable();
                                case TablesPackage.QUERY_GET__ROOT_ALIAS:
                                        return ROOT_ALIAS_EDEFAULT == null ? rootAlias != null :
!ROOT_ALIAS_EDEFAULT.equals(rootAlias);
                        }
                        return super.eIsSet(featureID);
                }

                /**
                 * <!-- begin-user-doc -->
                 * <!-- end-user-doc -->
                 * @generated
                 */
                @Override
                public String toString() {
                        if (eIsProxy()) return super.toString();

                        StringBuffer result = new StringBuffer(super.toString());
                        result.append(" (RootTable: ");
                        if (rootTableESet) result.append(rootTable); else result.append("<unset>");
                        result.append(", rootAlias: ");
                        result.append(rootAlias);
                        result.append(')');
                        return result.toString();
                }

                /**
                 * <!-- begin-user-doc -->
                 * The following operations occur
                 *                      -> generation of the query
                 *                      -> connection with the database and return of the
                 *                        resultset
                 *                      -> format data into object
                 * <!-- end-user-doc -->
                 * @generated NOT
                 */
                public List<Object> generate(Class<?> clazz) {

                        /*Query GEneration phase*/
                        tablesToAlias = new HashMap<String,String>();
                        /*Initialize the joinsOfTables*/
                        this.joinsOfTables = new HashMap<String,List<JOIN<?,?,?>>>();
```

```
/*Start with select clause*/
StringBuffer query = new StringBuffer("SELECT ");
/*Display select for the fields of the main table */
	for(Field<?,?> rootField:this.selectedFields){
		if(!(this.selectedFields.indexOf(rootField)==0)){
			query.append(", ");
		}
		else{
			query.append(" ");
		}
		if(this.rootAlias.equals(null)){
			query.append(this.rootTable+".");
		}
		else{
			query.append(this.rootAlias+".");
		}
		query.append(rootField.getFieldName()+" AS ");
		query.append(rootField.getAlias()+this.hashCode());
	}
/*If there are multiple tables,
 * loop through the joins and get the joined fields*/
if(this.join!=null){
	/*get all the joined tables of the root*/
	for(JOIN<?,?,?> join:this.join){
		/*Display their fields*/
		/*The general form of appending a field is:
		 *<Table>.<Field> AS <Field Alias>
		 */
		for(Field<?,?> joinfield:join.getJoinedTableFields()){
			/* Use the alias of the joined table if it is not null.
			 * If the alias is null use the name of the table instead
			 */
			if(join.getJoinedAlias().equals(null)){
				query.append(" ,"+join.getJoinedTable()+".");
			}
			else{
				query.append(" ,"+join.getJoinedAlias()+".");
			}
			/* Append the name of each field along with its alias
			 * The alias of the field MUST NOT be null
			 */
			query.append(joinfield.getFieldName()+" AS ");
			query.append(joinfield.getAlias()+join.hashCode());
		}
		query = joinFieldUnwrap(query,join);
	}
}
/*add FROM clause, first for Table, and then for the joins*/
query.append(" FROM ");
/*Append the name of the root table in the query*/
query.append(this.rootTable);

/*if the root Table has an alias append it*/
if(!this.rootAlias.equals(null)){
	tablesToAlias.put(this.rootTable, this.rootAlias);
	query.append(" AS "+this.rootAlias);
}
/*In case you have multiple tables add them through the join fields*/
/*The general form of appending a field is:
 *JOIN <Table> AS <Table Alias> ON <Root Table>.<Initial Field> = <Joined Table>.<Joined Field>
 */
if(this.join!=null){
	for(JOIN<?,?,?> join:this.join){
		/*First of all update the hashMap joinsOfTables*/
		if(!this.joinsOfTables.containsKey(join.getInitialTable())){
			List<JOIN<?,?,?>> joinList = new ArrayList<JOIN<?,?,?>>();
			joinList.add(join);
```

```
                                this.joinsOfTables.put(join.getInitialTable(), joinList);
                        }
                        else{
                                List<JOIN<?,?,?>> temp = this.joinsOfTables.get(join.getInitialTable());
                                temp.add(join);
                                this.joinsOfTables.put(join.getInitialTable(), temp);
                        }
                        /* Append the name of the joined table along with the keyword JOIN */
                        query.append(" JOIN "+ join.getJoinedTable());
                        /*If the table has an alias, append it*/
                        if(!join.getJoinedAlias().equals(null)){
                                tablesToAlias.put(join.getJoinedTable(), join.getJoinedAlias());
                                query.append(" AS "+join.getJoinedAlias());
                        }
                        /*Define on which field the join happens*/
                        query.append(" ON ");
                        /*Append the root table or alias*/
                        if(!this.rootAlias.equals(null)){
                                query.append(this.rootAlias+".");
                        }
                        else{
                                query.append(this.rootTable+".");
                        }
                        /*Append the field name of the root.*/
                        query.append(join.getOnFieldInitial().getFieldName()+"=");
                        /*Append the joined table name or alias*/
                        if(!join.getJoinedAlias().equals(null)){
                                query.append(join.getJoinedAlias()+".");
                        }
                        else{
                                query.append(join.getJoinedTable()+".");
                        }
                        /*Append the joined table field*/
                        query.append(join.getOnFieldJoined().getFieldName());
                        /*Repeat the above process with the joined Table as the root*/
                        query = joinTableUnwrap(query,join);
                }
        }
        /*Add WHERE clause */
        if(this.getFilter()!=null){
                query.append(" WHERE ");

                /*Add the filter for the root and/or the joined tables*/
                if(this.getFilter().getFromJOIN()==null){
                        query = resolveFilter(query,this.getFilter(),this.rootAlias);
                }
                else{
                        query = resolveFilter(query,this.getFilter(),this.getFilter().getFromJOIN().getJoinedAlias());
                }

        }
        System.out.println("Query Generated: "+query);

        /*connect with database and bring the results*/
        try{
                /*Transaction Phase*/
                /*connect with the database*/
                String host = "jdbc:mysql://localhost:3306/sqltest1";
                String uName = "root";
                String uPass = "root";

                Connection con = DriverManager.getConnection(host, uName, uPass);

                Statement stmt = con.createStatement();

                /*execute the query to get the resultset back*/
                ResultSet resset = stmt.executeQuery( query.toString() );
```

```
/*Processing Phase*/
/*Initialize the resultMap*/
this.resultMap = new HashMap<String,Object>();

/*Loop through the resultsets*/
/*Objects will be generated with Top-Down logic*/
while(resset.next()){
	/*generate the key of the table
	 * (contains the values of the fields in a string)
	 */
	String key = generateKey(resset,null,null);
	/*If the key exist go and resolve the joins of this table with
	 * the use of joinsOfTables
	 */
	if(resultMap.containsKey(key)){
		/*resolve the other joins*/
		if(this.joinsOfTables.get(this.rootTable)!=null){
			for(JOIN<?,?,?> join:this.joinsOfTables.get(this.rootTable)){
				try {
					resolveJoin(clazz,join, resset,
this.joinsOfTables,key);
				} catch (ClassNotFoundException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}
			}
		}
	}
	/*if the key does not exist, create an object, put it in resultMam
	 *and then resolve the joins of the tables
	 */
	else{
		try {
			/*create the object*/
			Object obj = createObject(clazz, resset,null);
			this.resultMap.put(key, obj);
			/*resolve joins*/
			if(this.joinsOfTables.get(this.rootTable)!=null){
				for(JOIN<?,?,?> join:this.joinsOfTables.get(this.rootTable)){
					try {
						resolveJoin(clazz, join, resset,
this.joinsOfTables,key);
					} catch (ClassNotFoundException e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					}
				}
			}
		} catch (IllegalArgumentException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (NoSuchFieldException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (SecurityException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
}

/*Combine the objects into the resultMap*/
combiner(clazz);

Iterator it = this.resultMap.entrySet().iterator();
List<Object> finalList = new ArrayList<Object>();
```

```
                while (it.hasNext()) {
                        Map.Entry pair = (Map.Entry)it.next();
                        finalList.add(pair.getValue());
                }
                return finalList;
            }
            catch(SQLException err){
                        System.out.println(err.getMessage());
            }


            return null;
    }
    /**
     *
     * @param query
     * @param join
     * @return
     */
    private StringBuffer joinFieldUnwrap(StringBuffer query, JOIN<?,?,?> join){
            for(JOIN<?,?,?> nextjoin:join.getJoinMore()){
                    if(nextjoin!=null){
                            for(Field<?,?> joinfield:nextjoin.getJoinedTableFields()){
                                    if(nextjoin.getJoinedAlias().equals(null)){
                                            query.append(" ,"+nextjoin.getJoinedTable()+".");
                                    }
                                    else{
                                            query.append(" ,"+nextjoin.getJoinedAlias()+".");
                                    }
                                    query.append(joinfield.getFieldName()+" AS ");
                                    query.append(joinfield.getAlias()+nextjoin.hashCode());
                            }
                            query = joinFieldUnwrap(query,nextjoin);
                    }
            }
            return query;
    }
    /**
     *
     * @param query
     * @param join
     * @return
     */
    private StringBuffer joinTableUnwrap(StringBuffer query, JOIN<?,?,?> join){
            /*loop through all the tables that are joined with the root
             *The "join" object is considered the root
             */
            for(JOIN<?,?,?> nextjoin : join.getJoinMore()){
                    /*First of all update the hashMap joinsOfTables*/
                    if(!this.joinsOfTables.containsKey(nextjoin.getInitialTable())){
                            List<JOIN<?,?,?>> joinList = new ArrayList<JOIN<?,?,?>>();
                            joinList.add(nextjoin);
                            this.joinsOfTables.put(nextjoin.getInitialTable(), joinList);
                    }
                    else{
                            List<JOIN<?,?,?>> temp = this.joinsOfTables.get(nextjoin.getInitialTable());
                            temp.add(nextjoin);
                            this.joinsOfTables.put(nextjoin.getInitialTable(), temp);
                    }
                    /*repeat the process described in the generate sector of field manipulation*/
                    if(nextjoin!=null){
                            query.append(" JOIN "+ nextjoin.getJoinedTable());
                            if(!nextjoin.getJoinedAlias().equals(null)){
                                    tablesToAlias.put(nextjoin.getJoinedTable(), nextjoin.getJoinedAlias());
                                    query.append(" AS "+nextjoin.getJoinedAlias());
                            }
                            query.append(" ON ");
```

```
                                        if(nextjoin.getInitialAlias()!=null){
                                                query.append(nextjoin.getInitialAlias()+".");
                                        }
                                        else{
                                                query.append(nextjoin.getInitialTable()+".");
                                        }
                                        query.append(nextjoin.getOnFieldInitial().getFieldName()+"=");
                                        if(nextjoin.getJoinedAlias()!=null){
                                                query.append(nextjoin.getJoinedAlias()+".");
                                        }
                                        else{
                                                query.append(nextjoin.getJoinedTable()+".");
                                        }
                                        query.append(nextjoin.getOnFieldJoined().getFieldName());
                                        query = joinTableUnwrap(query,nextjoin);
                                }
                        }
                        return query;
        }

        private StringBuffer joinWhereUnwrap(StringBuffer query, JOIN<?,?,?> join){
                        if(join.getWhere()!=null){
                                query = resolveFilter(query,join.getWhere(),join.getJoinedAlias());
                        }
                        for(JOIN<?,?,?> innerjoin:join.getJoinMore()){
                                query = joinWhereUnwrap(query,innerjoin);
                        }
                        return query;
        }
        /**
         * Analyze of the filter, for query creation
         * @param query
         * @param filter
         * @param tableAlias
         * @return
         */
        private StringBuffer resolveFilter(StringBuffer query, Filters<?> filter, String tableAlias){
                        query.append("(");
                        /*append the fieldname*/
                        query.append(tableAlias+"."+filter.getCondition().getField().getFieldName()+" ");
                        /*append the operator*/
                        switch(filter.getCondition().getOperator().getLiteral()){
                                case "EQUALS":
                                        query.append("= ");
                                        break;
                                case "NOT_EQUAL":
                                        query.append("!= ");
                                        break;
                                case "GR":
                                        query.append("> ");
                                        break;
                                case "LS":
                                        query.append("< ");
                                        break;
                                case "GRE":
                                        query.append(">= ");
                                        break;
                                case "LSE":
                                        query.append("<= ");
                                        break;
                                case "BETWEEN":
                                        query.append(filter.getCondition().getOperator().getLiteral()+" ");
                                        break;
                                case "LIKE":
                                        query.append(filter.getCondition().getOperator().getLiteral()+" ");
                                        break;
                                case "IN":
```

```
                                        query.append(filter.getCondition().getOperator().getLiteral()+" ");
                                        break;
                        }
                        /*append the value*/
                        query.append("'"+filter.getCondition().getOther()+"'");

                        /*go to the inner filters*/
                        for(NaryNode treelist:filter.getChainedFilters()){
                                /*find type of the filter*/
                                switch(treelist.getBl()){
                                        case AND:
                                                query.append(" AND ");
                                                /*Resolve inner filters*/
                                                if(treelist.getValue().getFromJOIN()==null){
                                                        query = resolveFilter(query,treelist.getValue(),this.rootAlias);
                                                }
                                                else{
                                                        query =
resolveFilter(query,treelist.getValue(),treelist.getValue().getFromJOIN().getJoinedAlias());
                                                }
                                                query.append(" ");
                                                break;
                                        case OR:
                                                query.append(" OR ");
                                                /*Resolve inner filters*/
                                                if(treelist.getValue().getFromJOIN()==null){
                                                        query = resolveFilter(query,treelist.getValue(),this.rootAlias);
                                                }
                                                else{
                                                        query =
resolveFilter(query,treelist.getValue(),treelist.getValue().getFromJOIN().getJoinedAlias());
                                                }
                                                query.append(" ");
                                                break;
                                }
                        }
                        query.append(")");
                        return query;
                }
                /**
                 * Generate a key for the resultMap of the root and the joins
                 * The key consists of the values of the fields
                 * @param resset
                 * @param join
                 * @param prevKey
                 * @return
                 * @throws SQLException
                 */
                private String generateKey(ResultSet resset, JOIN<?,?,?> join, String prevKey) throws SQLException{
                        StringBuffer key = new StringBuffer();
                        /*root case*/
                        if(join==null){
                                if(!this.getSelectedFields().isEmpty()){
                                        for(Field<?,?> fld:this.getSelectedFields()){
                                                key.append(resset.getObject(fld.getAlias()+this.hashCode()));
                                        }
                                }
                                else{
                                        key.append(resset.hashCode());
                                }
                        }
                        /*join case*/
                        else{
                                key.append(prevKey);
                                if(!join.getJoinedTableFields().isEmpty()){
                                        for(Field<?,?> fld:join.getJoinedTableFields()){
                                                key.append(resset.getObject(fld.getAlias()+join.hashCode()));
```

```
                                        }
                                }
                                else{
                                        key.append(resset.hashCode());
                                }
                        }
                        key.append("-");
//                      System.out.println(key);
                        return key.toString();
                }
                /**
                 * Create a new object for the resultMap
                 * @param clazz : class of the object produced from the ORM
                 * @param resset
                 * @param join
                 * @return
                 * @throws SQLException
                 * @throws IllegalArgumentException
                 * @throws NoSuchFieldException
                 * @throws SecurityException
                 */
                private Object createObject(Class<?> clazz, ResultSet resset, JOIN<?,?,?> join) throws SQLException,
IllegalArgumentException, NoSuchFieldException, SecurityException{
                        /*Create object for the root*/
                        if(join==null){
                                try {
                                        /*Make a new instance of the class*/
                                        Object obj = clazz.newInstance();
                                        /* iterate through the fields of the root
                                         * and fill the object
                                         */
                                        for(Field<?, ?> fld:this.selectedFields){
                                                java.lang.reflect.Field tempfld =
obj.getClass().getDeclaredField(fld.getFieldName());
                                                tempfld.setAccessible(true);
                                                tempfld.set(obj, resset.getObject(fld.getAlias()+this.hashCode()));
                                                tempfld.setAccessible(false);
                                        }
//                                      System.out.println("Object created for "+this.rootTable);
                                        return obj;
                                } catch (InstantiationException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                        return null;
                                } catch (IllegalAccessException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                        return null;
                                }
                        }
                        /*Create object for the joins*/
                        else{
                                try {
                                        /*create a new instance of the class*/
                                        Object obj = clazz.newInstance();
                                        /*iterate through the fields of the root
                                         * and fill the object
                                         * */
                                        for(Field<?, ?> fld:join.getJoinedTableFields()){
                                                java.lang.reflect.Field tempfld =
obj.getClass().getDeclaredField(fld.getFieldName());
                                                tempfld.setAccessible(true);
                                                tempfld.set(obj, resset.getObject(fld.getAlias()+join.hashCode()));
                                                tempfld.setAccessible(false);
                                        }
//                                      System.out.println("Object created for "+join.getJoinedTable());
                                        return obj;
```

```
                                } catch (InstantiationException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                        return null;
                                } catch (IllegalAccessException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                        return null;
                                }

                        }
                }
                /**
                 * Analyze a join.
                 * Generate its key from the parent of the join.
                 * Create an object for the resultMap that exists in the JOIN object.
                 * Get the inner joins of this JOIN object and resolve them the same way
                 * @param clazz
                 * @param join
                 * @param resset
                 * @param joinsOfTables
                 * @param prevKey
                 * @throws SQLException
                 * @throws ClassNotFoundException
                 */
                private void resolveJoin(Class<?> clazz,JOIN<?,?,?> join, ResultSet resset,Map<String,List<JOIN<?,?,?>>> joinsOfTables, String
prevKey) throws SQLException, ClassNotFoundException{
                        /*generate the key join*/
                        String key = generateKey(resset,join,prevKey);
                        /*if the key exist draw the object, else create the object*/
                        if(join.getResultMap().containsKey(key)){
                                /*call resolve joins*/
                                if(joinsOfTables.get(join.getJoinedTable())!=null){
                                        for(JOIN<?,?,?> innerjoin:joinsOfTables.get(join.getJoinedTable())){
                                                resolveJoin(clazz,innerjoin, resset, joinsOfTables,key);
                                        }
                                }
                        }
                        else{
                                /*create it and put it in the result map*/
                                Class<?> joinClass = Class.forName(clazz.getPackage().getName()+"."+join.getJoinedTable());
                                try {
                                        Object obj = createObject(joinClass, resset,join);
                                        join.getResultMap().put(key, obj);
                                        /*resolve the other joins*/
                                        if(joinsOfTables.get(join.getJoinedTable())!=null){
                                                for(JOIN<?,?,?> innerjoin:joinsOfTables.get(join.getJoinedTable())){
                                                        resolveJoin(clazz,innerjoin, resset, joinsOfTables,key);
                                                }
                                        }
                                } catch (IllegalArgumentException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                } catch (NoSuchFieldException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                } catch (SecurityException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                        }

                }

                /* Combines the objects in the resultMaps, Starting from the root table
                 * the method combiner_inner is called recursively in order to resolve the joins
                 * With the joinsOfTables we can see the hierarchy in which the Tables are accessed.
```

```
	 * It implements a tree which we access with a DFS.
	 * The Lists of the Entities are filled Bottom-Up.
	 * */
	private void combiner(Class<?> clazz){
		/*Enter only if you have joined tables*/
		if(this.joinsOfTables.get(this.rootTable)!=null){
			/*Loop through all the JOIN objects in your list*/
			for(JOIN<?,?,?> join:this.joinsOfTables.get(this.rootTable)){
				/* Call combiner_inner to resolve each join first.
				 * The logic is to access the tables as a DFS tree of tables, combining the
				 * "leaf" Tables first
				 * */
				combiner_inner(clazz,join);
				/*Find all the children in the join HashMap. Put them in the appropriate list*/
				/*Iterate through the resultMap of the child JOIN and insert them in the apropriate List*/
				Iterator it = join.getResultMap().entrySet().iterator();
				while(it.hasNext()){
					/*Tokenize the key in depth*/
					Map.Entry pair = (Map.Entry)it.next();
					/* Get the parent of the child.
					 * The parent is calculated from the key
					 * */
					String parentKey = getItToken(pair.getKey().toString(),"-");

					/*find the fields of the parent*/
					for(java.lang.reflect.Field fld :
this.resultMap.get(parentKey).getClass().getDeclaredFields()){
						/*get only the fields that are Lists*/
						if(fld.getType().getSimpleName().equals("List")){
							/*Check if the list is of the appropriate type*/
							ParameterizedType party =
(ParameterizedType)fld.getGenericType();

		if(party.getActualTypeArguments()[0].getTypeName().equals(clazz.getPackage().getName()+"."+join.getJoinedTable())){
							try {
								/*update the list of the parent*/
							fld.setAccessible(true);
							List templist = (List) fld.get(this.resultMap.get(parentKey));
							if(templist==null){
								templist = new ArrayList();
							}
							templist.add(pair.getValue());

		fld.set(this.resultMap.get(parentKey), templist);
								fld.setAccessible(false);
							} catch (IllegalArgumentException e) {
								// TODO Auto-generated catch block
								e.printStackTrace();
							} catch (IllegalAccessException e) {
								// TODO Auto-generated catch block
								e.printStackTrace();
							}
						}
					}
				}

			}
		}
	}

	/* Combiner for joins and their inner joins.
	 * It works recursively.
	 * Logic is similar with combiner.
	 * */
	private void combiner_inner(Class<?> clazz, JOIN<?,?,?> join){
		if(joinsOfTables.get(join.getJoinedTable())!=null){
```

```
                                for(JOIN<?,?,?> innerjoin:joinsOfTables.get(join.getJoinedTable())){
                                        combiner_inner(clazz,innerjoin);

                                        Iterator it = innerjoin.getResultMap().entrySet().iterator();
                                        while(it.hasNext()){

                                                Map.Entry pair = (Map.Entry)it.next();
                                                String parentKey = getItToken(pair.getKey().toString(),"-");

                                        for(java.lang.reflect.Field fld : join.getResultMap().get(parentKey).getClass().getDeclaredFields()){
                                                if(fld.getType().getSimpleName().equals("List")){
                                                        ParameterizedType party = (ParameterizedType)fld.getGenericType();

                        if(party.getActualTypeArguments()[0].getTypeName().equals(clazz.getPackage().getName()+"."+innerjoin.getJoinedTable())){
                                                                        try {
                                                                                fld.setAccessible(true);
                                                                                List templist = (List)
fld.get(join.getResultMap().get(parentKey));
                                                                                if(templist==null){
                                                                                        templist = new ArrayList();
                                                                                }
                                                                                //System.out.println("Templist Size: "+templist.size());
                                                                                templist.add(pair.getValue());

                        fld.set(join.getResultMap().get(parentKey), templist);
                                                                                        fld.setAccessible(false);
                                                                                } catch (IllegalArgumentException e) {
                                                                                        // TODO Auto-generated catch block
                                                                                        e.printStackTrace();
                                                                                } catch (IllegalAccessException e) {
                                                                                        // TODO Auto-generated catch block
                                                                                        e.printStackTrace();
                                                                                }

                                                }
                                        }
                                }
                                        }
                                }
                        }
                        return ;
                }

                /*Returns the parent of the entity*/
                private String getItToken(String strValue, String splitter )
                {
                  String[] strArray = strValue.split("-");
                  String lastToken = strArray[strArray.length -1];
                  String parentKey = strValue.substring(0, strValue.length()-lastToken.length()-1);
                  return parentKey;

                }

} //QueryGETImpl
```

QueryGET Implementation

3.3.3 JOIN

```
/**
 */
package Tables;

import java.util.Map;
```

```
import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.ecore.EObject;

/**
 * <!-- begin-user-doc -->
 * A representation of the model object '<em><b>JOIN</b></em>'.
 * <!-- end-user-doc -->
 * @model
 * @generated
 */
public interface JOIN<InitialTable extends Table, JoinedTable extends Table, Type> extends EObject {
		/**
		 * Returns the value of the '<em><b>On Field Initial</b></em>' reference.
		 * @return the value of the '<em>On Field Initial</em>' reference.
		 * @model required="true" derived="true"
		 * @generated
		 */
		Field<Type, InitialTable> getOnFieldInitial();

		/**
		 * Sets the value of the '{@link Tables.JOIN#getOnFieldInitial <em>On Field Initial</em>}' reference.
		 * @param value the new value of the '<em>On Field Initial</em>' reference.
		 * @see #getOnFieldInitial()
		 * @generated
		 */
		void setOnFieldInitial(Field<Type, InitialTable> value);

		/**
		 * Returns the value of the '<em><b>On Field Joined</b></em>' reference.
		 * @return the value of the '<em>On Field Joined</em>' reference.
		 * @model required="true" derived="true"
		 * @generated
		 */
		Field<Type, JoinedTable> getOnFieldJoined();

		/**
		 * Sets the value of the '{@link Tables.JOIN#getOnFieldJoined <em>On Field Joined</em>}' reference.

		 * @param value the new value of the '<em>On Field Joined</em>' reference.
		 * @generated
		 */
		void setOnFieldJoined(Field<Type, JoinedTable> value);

		/**
		 * Returns the value of the '<em><b>Joined Table Fields</b></em>' reference list.
		 * The list contents are of type {@link Tables.Field}&lt;?, JoinedTable>.
		 * @return the value of the '<em>Joined Table Fields</em>' reference list.
		 * @model
		 * @generated
		 */
		EList<Field<?, JoinedTable>> getJoinedTableFields();

		/**
		 * Returns the value of the '<em><b>Where</b></em>' reference.
		 * @return the value of the '<em>Where</em>' reference.
		 * @model
		 * @generated
		 */
		Filters<JoinedTable> getWhere();

		/**
		 * Sets the value of the '{@link Tables.JOIN#getWhere <em>Where</em>}' reference.
		 * @param value the new value of the '<em>Where</em>' reference.
		 * @generated
		 */
		void setWhere(Filters<JoinedTable> value);
```

```
/**
 * Returns the value of the '<em><b>Initial Table</b></em>' attribute.
 * @return the value of the '<em>Initial Table</em>' attribute.
 * @generated
 */
String getInitialTable();

/**
 * Returns whether the value of the '{@link Tables.JOIN#getInitialTable <em>Initial Table</em>}' attribute is set.
 * @return whether the value of the '<em>Initial Table</em>' attribute is set.
 * @generated
 */
boolean isSetInitialTable();

/**
 * Returns the value of the '<em><b>Joined Table</b></em>' attribute.
 * @return the value of the '<em>Joined Table</em>' attribute.
 * @model unsettable="true" required="true" transient="true" changeable="false" derived="true"
 * @generated
 */
String getJoinedTable();

/**
 * Returns whether the value of the '{@link Tables.JOIN#getJoinedTable <em>Joined Table</em>}' attribute is set.
 * @return whether the value of the '<em>Joined Table</em>' attribute is set.
 * @generated
 */
boolean isSetJoinedTable();

/**
 * Returns the value of the '<em><b>Joined Alias</b></em>' attribute.
 * @return the value of the '<em>Joined Alias</em>' attribute.
 * @model
 * @generated
 */
String getJoinedAlias();

/**
 * Sets the value of the '{@link Tables.JOIN#getJoinedAlias <em>Joined Alias</em>}' attribute.
 * @param value the new value of the '<em>Joined Alias</em>' attribute.
 * @generated
 */
void setJoinedAlias(String value);

/**
 * Returns the value of the '<em><b>Initial Alias</b></em>' attribute.
 * @return the value of the '<em>Initial Alias</em>' attribute.
 * @model
 * @generated
 */
String getInitialAlias();

/**
 * Sets the value of the '{@link Tables.JOIN#getInitialAlias <em>Initial Alias</em>}' attribute.
 * @param value the new value of the '<em>Initial Alias</em>' attribute.
 * @generated
 */
void setInitialAlias(String value);

/**
 * Returns the value of the '<em><b>Join More</b></em>' reference list.
 * The list contents are of type {@link Tables.JOIN}&lt;?, ?, ?>.
 * @return the value of the '<em>Join More</em>' reference list.
 * @model
 * @generated
 */
EList<JOIN<?, ?, ?>> getJoinMore();
```

```
		Map<String, Object> getResultMap();
		void setResultMap(Map<String, Object> resultMap);
} // JOIN
```

JOIN Interface

```
/**
 */
package Tables.impl;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;
import org.eclipse.emf.ecore.impl.ENotificationImpl;
import org.eclipse.emf.ecore.impl.MinimalEObjectImpl;
import org.eclipse.emf.ecore.util.EObjectResolvingEList;

import Tables.Field;
import Tables.Filters;
import Tables.JOIN;
import Tables.Table;
import Tables.TablesPackage;

/**
 * <!-- begin-user-doc -->
 * An implementation of the model object '<em><b>JOIN</b></em>'.
 * <!-- end-user-doc -->
 * @generated
 */
public class JOINImpl<InitialTable extends Table, JoinedTable extends Table, Type> extends MinimalEObjectImpl.Container implements
JOIN<InitialTable, JoinedTable, Type> {
		/**
		 * The cached value of the '{@link #getOnFieldInitial() <em>On Field Initial</em>}' reference.
		 * @generated
		 * @ordered
		 */
		protected Field<Type, InitialTable> onFieldInitial;

		/**
		 * The cached value of the '{@link #getOnFieldJoined() <em>On Field Joined</em>}' reference.
		 * @generated
		 * @ordered
		 */
		protected Field<Type, JoinedTable> onFieldJoined;

		/**
		 * The cached value of the '{@link #getJoinedTableFields() <em>Joined Table Fields</em>}' reference list.
		 * @generated
		 * @ordered
		 */
		protected EList<Field<?, JoinedTable>> joinedTableFields;

		/**
		 * The cached value of the '{@link #getWhere() <em>Where</em>}' reference.
		 * @generated
		 * @ordered
		 */
		protected Filters<JoinedTable> where;

		/**
		 * The default value of the '{@link #getInitialTable() <em>Initial Table</em>}' attribute.
```

```
	 * @generated
	 * @ordered
	 */
	protected static final String INITIAL_TABLE_EDEFAULT = null;

	/**
	 * The cached value of the '{@link #getInitialTable() <em>Initial Table</em>}' attribute.
	 * @generated
	 * @ordered
	 */
	protected String initialTable = INITIAL_TABLE_EDEFAULT;

	/**
	 * This is true if the Initial Table attribute has been set.
	 * @generated
	 * @ordered
	 */
	protected boolean initialTableESet;

	/**
	 * The default value of the '{@link #getJoinedTable() <em>Joined Table</em>}' attribute.
	 * @generated
	 * @ordered
	 */
	protected static final String JOINED_TABLE_EDEFAULT = null;

	/**
	 * The cached value of the '{@link #getJoinedTable() <em>Joined Table</em>}' attribute.
	 * @generated
	 * @ordered
	 */
	protected String joinedTable = JOINED_TABLE_EDEFAULT;

	/**
	 * This is true if the Joined Table attribute has been set.
	 * @generated
	 * @ordered
	 */
	protected boolean joinedTableESet;

	/**
	 * The default value of the '{@link #getJoinedAlias() <em>Joined Alias</em>}' attribute.
	 * @generated
	 * @ordered
	 */
	protected static final String JOINED_ALIAS_EDEFAULT = null;

	/**
	 * The cached value of the '{@link #getJoinedAlias() <em>Joined Alias</em>}' attribute.
	 * @generated
	 * @ordered
	 */
	protected String joinedAlias = JOINED_ALIAS_EDEFAULT;

	/**
	 * The default value of the '{@link #getInitialAlias() <em>Initial Alias</em>}' attribute.
	 * @generated
	 * @ordered
	 */
	protected static final String INITIAL_ALIAS_EDEFAULT = null;

	/**
	 * The cached value of the '{@link #getInitialAlias() <em>Initial Alias</em>}' attribute.
	 * @generated
	 * @ordered
	 */
	protected String initialAlias = INITIAL_ALIAS_EDEFAULT;
```

```
/**
 * The cached value of the '{@link #getJoinMore() <em>Join More</em>}' reference list.
 * @generated
 * @ordered
 */
protected EList<JOIN<?, ?, ?>> joinMore;

/**
 * Here are stored the final data from a JOIN object.
 * key is the values returned from the joined table
 * value is the object that represents the table in the database
 * The objects are combined in lists (or sets in the future)
 * and finally stored in the resultMap of the root table
 */
private Map<String,Object> resultMap;

public Map<String, Object> getResultMap() {
		return resultMap;
}

public void setResultMap(Map<String, Object> resultMap) {
		this.resultMap = resultMap;
}

/**
 * <!-- begin-user-doc -->
 * Simple constructor for JOIN object
 * Its use is not recommended for the application
 * Mostly for syntactic sugar
 * <!-- end-user-doc -->
 * @generated
 */
protected JOINImpl() {
		super();
}

/**
 * <!-- begin-user-doc -->
 * Actual constructor that must be used
 * Takes as arguments classes of main Table
 * and a Joined table for initialization and typesafe processes
 * <!-- end-user-doc -->
 * @generated NOT
 */
protected JOINImpl(Class<InitialTable> initialTable,Class<JoinedTable>joinedTable) {
		super();
		this.initialTable = initialTable.getSimpleName().substring(1);
		this.joinedTable = joinedTable.getSimpleName().substring(1);
		this.resultMap = new HashMap<String,Object>();
}


/**
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated
 */
@Override
protected EClass eStaticClass() {
		return TablesPackage.Literals.JOIN;
}

/**
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated
```

```
	 */
	@SuppressWarnings("unchecked")
	public Field<Type, InitialTable> getOnFieldInitial() {
		if (onFieldInitial != null && onFieldInitial.eIsProxy()) {
			InternalEObject oldOnFieldInitial = (InternalEObject)onFieldInitial;
			onFieldInitial = (Field<Type, InitialTable>)eResolveProxy(oldOnFieldInitial);
			if (onFieldInitial != oldOnFieldInitial) {
				if (eNotificationRequired())
					eNotify(new ENotificationImpl(this, Notification.RESOLVE,
TablesPackage.JOIN__ON_FIELD_INITIAL, oldOnFieldInitial, onFieldInitial));
			}
		}
		return onFieldInitial;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public Field<Type, InitialTable> basicGetOnFieldInitial() {
		return onFieldInitial;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public void setOnFieldInitial(Field<Type, InitialTable> newOnFieldInitial) {
		Field<Type, InitialTable> oldOnFieldInitial = onFieldInitial;
		onFieldInitial = newOnFieldInitial;
		if (eNotificationRequired())
			eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.JOIN__ON_FIELD_INITIAL,
oldOnFieldInitial, onFieldInitial));
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	@SuppressWarnings("unchecked")
	public Field<Type, JoinedTable> getOnFieldJoined() {
		if (onFieldJoined != null && onFieldJoined.eIsProxy()) {
			InternalEObject oldOnFieldJoined = (InternalEObject)onFieldJoined;
			onFieldJoined = (Field<Type, JoinedTable>)eResolveProxy(oldOnFieldJoined);
			if (onFieldJoined != oldOnFieldJoined) {
				if (eNotificationRequired())
					eNotify(new ENotificationImpl(this, Notification.RESOLVE,
TablesPackage.JOIN__ON_FIELD_JOINED, oldOnFieldJoined, onFieldJoined));
			}
		}
		return onFieldJoined;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public Field<Type, JoinedTable> basicGetOnFieldJoined() {
		return onFieldJoined;
	}

	/**
	 * <!-- begin-user-doc -->
```

```
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public void setOnFieldJoined(Field<Type, JoinedTable> newOnFieldJoined) {
		Field<Type, JoinedTable> oldOnFieldJoined = onFieldJoined;
		onFieldJoined = newOnFieldJoined;
		if (eNotificationRequired())
			eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.JOIN__ON_FIELD_JOINED,
oldOnFieldJoined, onFieldJoined));
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public EList<Field<?, JoinedTable>> getJoinedTableFields() {
		if (joinedTableFields == null) {
			joinedTableFields = new EObjectResolvingEList<Field<?, JoinedTable>>(Field.class, this,
TablesPackage.JOIN__JOINED_TABLE_FIELDS);
		}
		return joinedTableFields;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	@SuppressWarnings("unchecked")
	public Filters<JoinedTable> getWhere() {
		if (where != null && where.eIsProxy()) {
			InternalEObject oldWhere = (InternalEObject)where;
			where = (Filters<JoinedTable>)eResolveProxy(oldWhere);
			if (where != oldWhere) {
				if (eNotificationRequired())
					eNotify(new ENotificationImpl(this, Notification.RESOLVE,
TablesPackage.JOIN__WHERE, oldWhere, where));
			}
		}
		return where;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public Filters<JoinedTable> basicGetWhere() {
		return where;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public void setWhere(Filters<JoinedTable> newWhere) {
		Filters<JoinedTable> oldWhere = where;
		where = newWhere;
		if (eNotificationRequired())
			eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.JOIN__WHERE, oldWhere, where));
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
```

```
	 * @generated
	 */
	@SuppressWarnings("unchecked")
	public String getInitialTable() {
		return initialTable;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public boolean isSetInitialTable() {
		return initialTableESet;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	@SuppressWarnings("unchecked")
	public String getJoinedTable() {
		return joinedTable;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public boolean isSetJoinedTable() {
		return joinedTableESet;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public String getJoinedAlias() {
		return joinedAlias;
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public void setJoinedAlias(String newJoinedAlias) {
		String oldJoinedAlias = joinedAlias;
		joinedAlias = newJoinedAlias;
		if (eNotificationRequired())
			eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.JOIN__JOINED_ALIAS,
oldJoinedAlias, joinedAlias));
	}

	/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @generated
	 */
	public String getInitialAlias() {
		return initialAlias;
	}

	/**
```

```
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @generated
		 */
		public void setInitialAlias(String newInitialAlias) {
			String oldInitialAlias = initialAlias;
			initialAlias = newInitialAlias;
			if (eNotificationRequired())
				eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.JOIN__INITIAL_ALIAS,
oldInitialAlias, initialAlias));
		}

		/**
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @generated
		 */
		public EList<JOIN<?, ?, ?>> getJoinMore() {
			if (joinMore == null) {
				joinMore = new EObjectResolvingEList<JOIN<?, ?, ?>>(JOIN.class, this,
TablesPackage.JOIN__JOIN_MORE);
			}
			return joinMore;
		}

		/**
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @generated
		 */
		@Override
		public Object eGet(int featureID, boolean resolve, boolean coreType) {
			switch (featureID) {
				case TablesPackage.JOIN__ON_FIELD_INITIAL:
					if (resolve) return getOnFieldInitial();
					return basicGetOnFieldInitial();
				case TablesPackage.JOIN__ON_FIELD_JOINED:
					if (resolve) return getOnFieldJoined();
					return basicGetOnFieldJoined();
				case TablesPackage.JOIN__JOINED_TABLE_FIELDS:
					return getJoinedTableFields();
				case TablesPackage.JOIN__WHERE:
					if (resolve) return getWhere();
					return basicGetWhere();
				case TablesPackage.JOIN__INITIAL_TABLE:
					return getInitialTable();
				case TablesPackage.JOIN__JOINED_TABLE:
					return getJoinedTable();
				case TablesPackage.JOIN__JOINED_ALIAS:
					return getJoinedAlias();
				case TablesPackage.JOIN__INITIAL_ALIAS:
					return getInitialAlias();
				case TablesPackage.JOIN__JOIN_MORE:
					return getJoinMore();
			}
			return super.eGet(featureID, resolve, coreType);
		}

		/**
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @generated
		 */
		@SuppressWarnings("unchecked")
		@Override
		public void eSet(int featureID, Object newValue) {
			switch (featureID) {
```

```
                        case TablesPackage.JOIN__ON_FIELD_INITIAL:
                                setOnFieldInitial((Field<Type, InitialTable>)newValue);
                                return;
                        case TablesPackage.JOIN__ON_FIELD_JOINED:
                                setOnFieldJoined((Field<Type, JoinedTable>)newValue);
                                return;
                        case TablesPackage.JOIN__JOINED_TABLE_FIELDS:
                                getJoinedTableFields().clear();
                                getJoinedTableFields().addAll((Collection<? extends Field<?, JoinedTable>>)newValue);
                                return;
                        case TablesPackage.JOIN__WHERE:
                                setWhere((Filters<JoinedTable>)newValue);
                                return;
                        case TablesPackage.JOIN__JOINED_ALIAS:
                                setJoinedAlias((String)newValue);
                                return;
                        case TablesPackage.JOIN__INITIAL_ALIAS:
                                setInitialAlias((String)newValue);
                                return;
                        case TablesPackage.JOIN__JOIN_MORE:
                                getJoinMore().clear();
                                getJoinMore().addAll((Collection<? extends JOIN<?, ?, ?>>)newValue);
                                return;
                }
                super.eSet(featureID, newValue);
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @Override
        public void eUnset(int featureID) {
                switch (featureID) {
                        case TablesPackage.JOIN__ON_FIELD_INITIAL:
                                setOnFieldInitial((Field<Type, InitialTable>)null);
                                return;
                        case TablesPackage.JOIN__ON_FIELD_JOINED:
                                setOnFieldJoined((Field<Type, JoinedTable>)null);
                                return;
                        case TablesPackage.JOIN__JOINED_TABLE_FIELDS:
                                getJoinedTableFields().clear();
                                return;
                        case TablesPackage.JOIN__WHERE:
                                setWhere((Filters<JoinedTable>)null);
                                return;
                        case TablesPackage.JOIN__JOINED_ALIAS:
                                setJoinedAlias(JOINED_ALIAS_EDEFAULT);
                                return;
                        case TablesPackage.JOIN__INITIAL_ALIAS:
                                setInitialAlias(INITIAL_ALIAS_EDEFAULT);
                                return;
                        case TablesPackage.JOIN__JOIN_MORE:
                                getJoinMore().clear();
                                return;
                }
                super.eUnset(featureID);
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @Override
        public boolean eIsSet(int featureID) {
```

```
                switch (featureID) {
                        case TablesPackage.JOIN__ON_FIELD_INITIAL:
                                return onFieldInitial != null;
                        case TablesPackage.JOIN__ON_FIELD_JOINED:
                                return onFieldJoined != null;
                        case TablesPackage.JOIN__JOINED_TABLE_FIELDS:
                                return joinedTableFields != null && !joinedTableFields.isEmpty();
                        case TablesPackage.JOIN__WHERE:
                                return where != null;
                        case TablesPackage.JOIN__INITIAL_TABLE:
                                return isSetInitialTable();
                        case TablesPackage.JOIN__JOINED_TABLE:
                                return isSetJoinedTable();
                        case TablesPackage.JOIN__JOINED_ALIAS:
                                return JOINED_ALIAS_EDEFAULT == null ? joinedAlias != null :
!JOINED_ALIAS_EDEFAULT.equals(joinedAlias);
                        case TablesPackage.JOIN__INITIAL_ALIAS:
                                return INITIAL_ALIAS_EDEFAULT == null ? initialAlias != null :
!INITIAL_ALIAS_EDEFAULT.equals(initialAlias);
                        case TablesPackage.JOIN__JOIN_MORE:
                                return joinMore != null && !joinMore.isEmpty();
                }
                return super.eIsSet(featureID);
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @Override
        public String toString() {
                if (eIsProxy()) return super.toString();

                StringBuffer result = new StringBuffer(super.toString());
                result.append(" (InitialTable: ");
                if (initialTableESet) result.append(initialTable); else result.append("<unset>");
                result.append(", JoinedTable: ");
                if (joinedTableESet) result.append(joinedTable); else result.append("<unset>");
                result.append(", joinedAlias: ");
                result.append(joinedAlias);
                result.append(", initialAlias: ");
                result.append(initialAlias);
                result.append(')');
                return result.toString();
        }

} //JOINImpl
```

JOIN Implementation

3.3.4 Filters

```
/**
 */
package Tables;

import java.util.List;

import org.eclipse.emf.ecore.EObject;

/**
 * <!-- begin-user-doc -->
 * A representation of the model object '<em><b>Filters</b></em>'.
 * <!-- end-user-doc -->
```

```
 * @model
 * @generated
 */
public interface Filters<TableInstance extends Table> extends EObject {
		/**
		 * Returns the value of the '<em><b>Condition</b></em>' reference.
		 * @return the value of the '<em>Condition</em>' reference.
		 * @model required="true"
		 * @generated
		 */
		Condition<?, TableInstance> getCondition();

		/**
		 * Sets the value of the '{@link Tables.Filters#getCondition <em>Condition</em>}' reference.
		 * @param value the new value of the '<em>Condition</em>' reference.
		 * @generated
		 */
		void setCondition(Condition<?, TableInstance> value);

		/**
		 * Returns the value of the '<em><b>From JOIN</b></em>' reference.
		 * @return the value of the '<em>From JOIN</em>' reference.
		 * @model
		 * @generated
		 */
		JOIN<?, ?, ?> getFromJOIN();

		/**
		 * Sets the value of the '{@link Tables.Filters#getFromJOIN <em>From JOIN</em>}' reference.
		 * @param value the new value of the '<em>From JOIN</em>' reference.
		 * @generated
		 */
		void setFromJOIN(JOIN<?, TableInstance, ?> value);

		/**
		 * @model conditionRequired="true"
		 * @generated
		 */
		Filters<?> and(Filters<?> condition);

		/**
		 * @model conditionRequired="true"
		 * @generated
		 */
		Filters<?> or(Filters<?> condition);

		List<NaryNode> getChainedFilters();

} // Filters
```

Filters Interface

```
/**
 */
package Tables.impl;

import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;
import java.util.List;

import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;
import org.eclipse.emf.ecore.impl.ENotificationImpl;
import org.eclipse.emf.ecore.impl.MinimalEObjectImpl;
```

```
import Tables.Condition;
import Tables.Field;
import Tables.Filters;
import Tables.JOIN;
import Tables.NaryNode;
import Tables.Operators;
import Tables.Table;
import Tables.TablesFactory;
import Tables.TablesPackage;
import Tables.bol;

/**
 * <!-- begin-user-doc -->
 * An implementation of the model object '<em><b>Filters</b></em>'.
 * <!-- end-user-doc -->
 * @generated
 */
public class FiltersImpl<TableInstance extends Table> extends MinimalEObjectImpl.Container implements Filters<TableInstance> {
		/**
		 * The cached value of the '{@link #getCondition() <em>Condition</em>}' reference.
		 * @generated
		 * @ordered
		 */
		protected Condition<?, TableInstance> condition;

		/**
		 * The cached value of the '{@link #getFromJOIN() <em>From JOIN</em>}' reference.
		 * @generated
		 * @ordered
		 */
		protected JOIN<?, TableInstance, ?> fromJOIN;

		private Filters<?> andFilter;
		private Filters<?> orFilter;
		private List<NaryNode> chainedFilters;

		/**
		 * <!-- begin-user-doc -->
		 * Constructor for Filters object.
		 * Mostly used for syntactic sugar
		 * Can be used in the application but makes
		 * query construction more difficult
		 * since these is need for manual initialization of
		 * the fields
		 * <!-- end-user-doc -->
		 * @generated NOT
		 */
		protected FiltersImpl() {
			super();
			this.chainedFilters = new ArrayList<NaryNode>();
		}

		/**
		 * <!-- begin-user-doc -->
		 * Constructor for Filters object
		 * Use of this constructor is preferred since
		 * it automatically initialize the data of the filter
		 * <!-- end-user-doc -->
		 * @generated NOT
		 */
		@SuppressWarnings({ "unchecked", "rawtypes" })
		protected FiltersImpl(Field<?,?> field, Operators op, Object other,JOIN<?,TableInstance,?> fromJOIN) {
			super();
			Condition cond = TablesFactory.eINSTANCE.createCondition();
			cond.setField(field);
			cond.setOperator(op);
```

```
                cond.setOther(other);
                this.condition = cond;
                this.chainedFilters = new ArrayList<NaryNode>();
                this.fromJOIN = fromJOIN;
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @Override
        protected EClass eStaticClass() {
                return TablesPackage.Literals.FILTERS;
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @SuppressWarnings("unchecked")
        public Condition<?, TableInstance> getCondition() {
                if (condition != null && condition.eIsProxy()) {
                        InternalEObject oldCondition = (InternalEObject)condition;
                        condition = (Condition<?, TableInstance>)eResolveProxy(oldCondition);
                        if (condition != oldCondition) {
                                if (eNotificationRequired())
                                        eNotify(new ENotificationImpl(this, Notification.RESOLVE,
TablesPackage.FILTERS__CONDITION, oldCondition, condition));
                        }
                }
                return condition;
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        public Condition<?, TableInstance> basicGetCondition() {
                return condition;
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        public void setCondition(Condition<?, TableInstance> newCondition) {
                Condition<?, TableInstance> oldCondition = condition;
                condition = newCondition;
                if (eNotificationRequired())
                        eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.FILTERS__CONDITION,
oldCondition, condition));
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @SuppressWarnings("unchecked")
        public JOIN<?, TableInstance, ?> getFromJOIN() {
                if (fromJOIN != null && fromJOIN.eIsProxy()) {
                        InternalEObject oldFromJOIN = (InternalEObject)fromJOIN;
                        fromJOIN = (JOIN<?, TableInstance, ?>)eResolveProxy(oldFromJOIN);
```

```
                                        if (fromJOIN != oldFromJOIN) {
                                                if (eNotificationRequired())
                                                        eNotify(new ENotificationImpl(this, Notification.RESOLVE,
TablesPackage.FILTERS__FROM_JOIN, oldFromJOIN, fromJOIN));
                                        }
                        }
                        return fromJOIN;
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        public JOIN<?, TableInstance, ?> basicGetFromJOIN() {
                        return fromJOIN;
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        public void setFromJOIN(JOIN<?, TableInstance, ?> newFromJOIN) {
                        JOIN<?, ?, ?> oldFromJOIN = fromJOIN;
                        fromJOIN = newFromJOIN;
                        if (eNotificationRequired())
                                eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.FILTERS__FROM_JOIN,
oldFromJOIN, fromJOIN));
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated NOT
         */
        public Filters<?> and(Filters<?> condition) {
                        this.andFilter = condition;
                        NaryNode node = TablesFactory.eINSTANCE.createNaryNode();
                        node.setBl(bol.AND);
                        node.setValue(condition);
                        chainedFilters.add(node);
                        return this;
        }

        public Filters<?> getAndFilter(){
                        return this.andFilter;
        }

        public Filters<?> getOrFilter(){
                        return this.orFilter;
        }

        public List<NaryNode> getChainedFilters(){
                        return chainedFilters;
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated NOT
         */
        public Filters<?> or(Filters<?> condition) {
                        this.orFilter = condition;
                        NaryNode node = TablesFactory.eINSTANCE.createNaryNode();
                        node.setBl(bol.OR);
                        node.setValue(condition);
```

```
                chainedFilters.add(node);
                return this;
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @Override
        public Object eGet(int featureID, boolean resolve, boolean coreType) {
                switch (featureID) {
                        case TablesPackage.FILTERS__CONDITION:
                                if (resolve) return getCondition();
                                return basicGetCondition();
                        case TablesPackage.FILTERS__FROM_JOIN:
                                if (resolve) return getFromJOIN();
                                return basicGetFromJOIN();
                }
                return super.eGet(featureID, resolve, coreType);
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @SuppressWarnings("unchecked")
        @Override
        public void eSet(int featureID, Object newValue) {
                switch (featureID) {
                        case TablesPackage.FILTERS__CONDITION:
                                setCondition((Condition<?, TableInstance>)newValue);
                                return;
                        case TablesPackage.FILTERS__FROM_JOIN:
                                setFromJOIN((JOIN<?, TableInstance, ?>)newValue);
                                return;
                }
                super.eSet(featureID, newValue);
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @Override
        public void eUnset(int featureID) {
                switch (featureID) {
                        case TablesPackage.FILTERS__CONDITION:
                                setCondition((Condition<?, TableInstance>)null);
                                return;
                        case TablesPackage.FILTERS__FROM_JOIN:
                                setFromJOIN((JOIN<?, TableInstance, ?>)null);
                                return;
                }
                super.eUnset(featureID);
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @Override
        public boolean eIsSet(int featureID) {
                switch (featureID) {
```

```
                                case TablesPackage.FILTERS__CONDITION:
                                        return condition != null;
                                case TablesPackage.FILTERS__FROM_JOIN:
                                        return fromJOIN != null;
                        }
                        return super.eIsSet(featureID);
        }

        /**
         * <!-- begin-user-doc -->
         * <!-- end-user-doc -->
         * @generated
         */
        @Override
        @SuppressWarnings("unchecked")
        public Object eInvoke(int operationID, EList<?> arguments) throws InvocationTargetException {
                switch (operationID) {
                        case TablesPackage.FILTERS___AND__FILTERS:
                                return and((Filters<?>)arguments.get(0));
                        case TablesPackage.FILTERS___OR__FILTERS:
                                return or((Filters<?>)arguments.get(0));
                }
                return super.eInvoke(operationID, arguments);
        }

} //FiltersImpl
```

Filters Implementation

3.3.5 Condition

```
/**
 */
package Tables;

import org.eclipse.emf.ecore.EObject;

/**
 * <!-- begin-user-doc -->
 * A representation of the model object '<em><b>Condition</b></em>'.
 * @model TBounds="org.eclipse.emf.ecore.EJavaObject"
 * @generated
 */
public interface Condition<T extends Object, TableInstance extends Table> extends EObject {
        /**
         * Returns the value of the '<em><b>Field</b></em>' reference.
         * @return the value of the '<em>Field</em>' reference.
         * @model required="true"
         * @generated
         */
        Field<T, TableInstance> getField();

        /**
         * Sets the value of the '{@link Tables.Condition#getField <em>Field</em>}' reference.
         * @param value the new value of the '<em>Field</em>' reference.
         * @generated
         */
        void setField(Field<T, TableInstance> value);

        /**
         * Returns the value of the '<em><b>Operator</b></em>' attribute.
         * The literals are from the enumeration {@link Tables.Operators}.
         * @return the value of the '<em>Operator</em>' attribute.
         * @model required="true"
         * @generated
```

```
		 */
		Operators getOperator();

		/**
		 * Sets the value of the '{@link Tables.Condition#getOperator <em>Operator</em>}' attribute.
		 * @param value the new value of the '<em>Operator</em>' attribute.
		 * @generated
		 */
		void setOperator(Operators value);

		/**
		 * Returns the value of the '<em><b>Other</b></em>' reference.
		 * @return the value of the '<em>Other</em>' reference.
		 * @model kind="reference" required="true"
		 * @generated
		 */
		T getOther();

		/**
		 * Sets the value of the '{@link Tables.Condition#getOther <em>Other</em>}' reference.
		 * @param value the new value of the '<em>Other</em>' reference.
		 * @generated
		 */
		void setOther(T value);

} // Condition
```

Condition Interface

```
/**
 */
package Tables.impl;

import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;
import org.eclipse.emf.ecore.impl.ENotificationImpl;
import org.eclipse.emf.ecore.impl.MinimalEObjectImpl;
import Tables.Condition;
import Tables.Field;
import Tables.Operators;
import Tables.Table;
import Tables.TablesPackage;

/**
 * <!-- begin-user-doc -->
 * An implementation of the model object '<em><b>Condition</b></em>'.
 * <!-- end-user-doc -->
 * @generated
 */
public class ConditionImpl<T extends Object, TableInstance extends Table> extends MinimalEObjectImpl.Container implements
Condition<T, TableInstance> {
		/**
		 * The cached value of the '{@link #getField() <em>Field</em>}' reference.
		 * @generated
		 * @ordered
		 */
		protected Field<T, TableInstance> field;

		/**
		 * The default value of the '{@link #getOperator() <em>Operator</em>}' attribute.
		 * @generated
		 * @ordered
		 */
		protected static final Operators OPERATOR_EDEFAULT = Operators.EQUALS;
```

```
	/**
	 * The cached value of the '{@link #getOperator() <em>Operator</em>}' attribute.
	 * @generated
	 * @ordered
	 */
	protected Operators operator = OPERATOR_EDEFAULT;

	/**
	 * The cached value of the '{@link #getOther() <em>Other</em>}' reference.
	 * @generated
	 * @ordered
	 */
	protected T other;

	/**
	 * @generated
	 */
	protected ConditionImpl() {
		super();
	}

	/**
	 * @generated
	 */
	@Override
	protected EClass eStaticClass() {
		return TablesPackage.Literals.CONDITION;
	}

	/**
	 * @generated
	 */
	@SuppressWarnings("unchecked")
	public Field<T, TableInstance> getField() {
		if (field != null && field.eIsProxy()) {
			InternalEObject oldField = (InternalEObject)field;
			field = (Field<T, TableInstance>)eResolveProxy(oldField);
			if (field != oldField) {
				if (eNotificationRequired())
					eNotify(new ENotificationImpl(this, Notification.RESOLVE,
TablesPackage.CONDITION__FIELD, oldField, field));
			}
		}
		return field;
	}

	/**
	 * @generated
	 */
	public Field<T, TableInstance> basicGetField() {
		return field;
	}

	/**
	 * @generated
	 */
	public void setField(Field<T, TableInstance> newField) {
		Field<T, TableInstance> oldField = field;
		field = newField;
		if (eNotificationRequired())
			eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.CONDITION__FIELD, oldField,
field));
	}

	/**
	 * @generated
	 */
```

```
	public Operators getOperator() {
		return operator;
	}

	/**
	 * @generated
	 */
	public void setOperator(Operators newOperator) {
		Operators oldOperator = operator;
		operator = newOperator == null ? OPERATOR_EDEFAULT : newOperator;
		if (eNotificationRequired())
			eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.CONDITION__OPERATOR,
oldOperator, operator));
	}

	/**
	 * @generated NOT
	 */
	@SuppressWarnings("unchecked")
	public T getOther() {
		return other;
	}

	/**
	 * @generated
	 */
	public T basicGetOther() {
		return other;
	}

	/**
	 * @generated
	 */
	public void setOther(T newOther) {
		T oldOther = other;
		other = newOther;
		if (eNotificationRequired())
			eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.CONDITION__OTHER, oldOther,
other));
	}

	/**
	 * @generated
	 */
	@Override
	public Object eGet(int featureID, boolean resolve, boolean coreType) {
		switch (featureID) {
			case TablesPackage.CONDITION__FIELD:
				if (resolve) return getField();
				return basicGetField();
			case TablesPackage.CONDITION__OPERATOR:
				return getOperator();
			case TablesPackage.CONDITION__OTHER:
				if (resolve) return getOther();
				return basicGetOther();
		}
		return super.eGet(featureID, resolve, coreType);
	}

	/**
	 * @generated
	 */
	@SuppressWarnings("unchecked")
	@Override
	public void eSet(int featureID, Object newValue) {
		switch (featureID) {
			case TablesPackage.CONDITION__FIELD:
```

```
                                        setField((Field<T, TableInstance>)newValue);
                                        return;
                                case TablesPackage.CONDITION__OPERATOR:
                                        setOperator((Operators)newValue);
                                        return;
                                case TablesPackage.CONDITION__OTHER:
                                        setOther((T)newValue);
                                        return;
                        }
                        super.eSet(featureID, newValue);
                }

                /**
                 * @generated
                 */
                @Override
                public void eUnset(int featureID) {
                        switch (featureID) {
                                case TablesPackage.CONDITION__FIELD:
                                        setField((Field<T, TableInstance>)null);
                                        return;
                                case TablesPackage.CONDITION__OPERATOR:
                                        setOperator(OPERATOR_EDEFAULT);
                                        return;
                                case TablesPackage.CONDITION__OTHER:
                                        setOther((T)null);
                                        return;
                        }
                        super.eUnset(featureID);
                }

                /**
                 * @generated
                 */
                @Override
                public boolean eIsSet(int featureID) {
                        switch (featureID) {
                                case TablesPackage.CONDITION__FIELD:
                                        return field != null;
                                case TablesPackage.CONDITION__OPERATOR:
                                        return operator != OPERATOR_EDEFAULT;
                                case TablesPackage.CONDITION__OTHER:
                                        return other != null;
                        }
                        return super.eIsSet(featureID);
                }

                /**
                 * @generated
                 */
                @Override
                public String toString() {
                        if (eIsProxy()) return super.toString();

                        StringBuffer result = new StringBuffer(super.toString());
                        result.append(" (operator: ");
                        result.append(operator);
                        result.append(')');
                        return result.toString();
                }

} //ConditionImpl
```

Condition Implementation

3.3.6 Table

```
/**
 */
package Tables;

import org.eclipse.emf.ecore.EObject;

/**
 * <!-- begin-user-doc -->
 * A representation of the model object '<em><b>Table</b></em>'.
 * <!-- end-user-doc -->
 * @model
 * @generated
 */
public interface Table extends EObject {
		/**
		 * Returns the value of the '<em><b>Table Name</b></em>' attribute.
		 * @return the value of the '<em>Table Name</em>' attribute.
		 * @model
		 * @generated
		 */
		String getTableName();

		/**
		 * Sets the value of the '{@link Tables.Table#getTableName <em>Table Name</em>}' attribute.
		 * @param value the new value of the '<em>Table Name</em>' attribute.
		 * @generated
		 */
		void setTableName(String value);

		/**
		 * Returns the value of the '<em><b>Table Entity</b></em>' attribute.
		 * @return the value of the '<em>Table Entity</em>' attribute.
		 * @model required="true"
		 * @generated
		 */
		Class getTableEntity();

		/**
		 * Sets the value of the '{@link Tables.Table#getTableEntity <em>Table Entity</em>}' attribute.
		 * @param value the new value of the '<em>Table Entity</em>' attribute.
		 * @generated
		 */
		void setTableEntity(Class value);

} // Table
```

Table Interface

```
/**
 */
package Tables.impl;

import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.impl.ENotificationImpl;
import org.eclipse.emf.ecore.impl.MinimalEObjectImpl;
import Tables.Table;
import Tables.TablesPackage;

/**
 * <!-- begin-user-doc -->
 * An implementation of the model object '<em><b>Table</b></em>'.
 * <!-- end-user-doc -->
```

```
 * @generated
 */
public class TableImpl extends MinimalEObjectImpl.Container implements Table {
	/**
	 * The default value of the '{@link #getTableName() <em>Table Name</em>}' attribute.
	 * @generated
	 * @ordered
	 */
	protected static final String TABLE_NAME_EDEFAULT = null;

	/**
	 * The cached value of the '{@link #getTableName() <em>Table Name</em>}' attribute.
	 * @generated
	 * @ordered
	 */
	protected String tableName = TABLE_NAME_EDEFAULT;

	/**
	 * The cached value of the '{@link #getTableEntity() <em>Table Entity</em>}' attribute.
	 * @generated
	 * @ordered
	 */
	protected Class tableEntity;

	/**
	 * @generated NOT
	 */
	protected TableImpl() {
		super();
	}

	/**
	 * @generated
	 */
	@Override
	protected EClass eStaticClass() {
		return TablesPackage.Literals.TABLE;
	}

	/**
	 * @generated
	 */
	public String getTableName() {
		return tableName;
	}

	/**
	 * @generated
	 */
	public void setTableName(String newTableName) {
		String oldTableName = tableName;
		tableName = newTableName;
		if (eNotificationRequired())
			eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.TABLE__TABLE_NAME,
oldTableName, tableName));
	}

	/**
	 * @generated
	 */
	public Class getTableEntity() {
		return tableEntity;
	}

	/**
	 * @generated
	 */
```

```
	public void setTableEntity(Class newTableEntity) {
		Class oldTableEntity = tableEntity;
		tableEntity = newTableEntity;
		if (eNotificationRequired())
			eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.TABLE__TABLE_ENTITY,
oldTableEntity, tableEntity));
	}

	/**
	 * @generated
	 */
	@Override
	public Object eGet(int featureID, boolean resolve, boolean coreType) {
		switch (featureID) {
			case TablesPackage.TABLE__TABLE_NAME:
				return getTableName();
			case TablesPackage.TABLE__TABLE_ENTITY:
				return getTableEntity();
		}
		return super.eGet(featureID, resolve, coreType);
	}

	/**
	 * @generated
	 */
	@SuppressWarnings("unchecked")
	@Override
	public void eSet(int featureID, Object newValue) {
		switch (featureID) {
			case TablesPackage.TABLE__TABLE_NAME:
				setTableName((String)newValue);
				return;
			case TablesPackage.TABLE__TABLE_ENTITY:
				setTableEntity((Class)newValue);
				return;
		}
		super.eSet(featureID, newValue);
	}

	/**
	 * @generated
	 */
	@Override
	public void eUnset(int featureID) {
		switch (featureID) {
			case TablesPackage.TABLE__TABLE_NAME:
				setTableName(TABLE_NAME_EDEFAULT);
				return;
			case TablesPackage.TABLE__TABLE_ENTITY:
				setTableEntity((Class)null);
				return;
		}
		super.eUnset(featureID);
	}

	/**
	 * @generated
	 */
	@Override
	public boolean eIsSet(int featureID) {
		switch (featureID) {
			case TablesPackage.TABLE__TABLE_NAME:
				return TABLE_NAME_EDEFAULT == null ? tableName != null :
!TABLE_NAME_EDEFAULT.equals(tableName);
			case TablesPackage.TABLE__TABLE_ENTITY:
				return tableEntity != null;
		}
```

```
                return super.eIsSet(featureID);
        }

        /**
         * @generated
         */
        @Override
        public String toString() {
                if (eIsProxy()) return super.toString();

                StringBuffer result = new StringBuffer(super.toString());
                result.append(" (TableName: ");
                result.append(tableName);
                result.append(", tableEntity: ");
                result.append(tableEntity);
                result.append(')');
                return result.toString();
        }

} //TableImpl
```

Table Implementation

3.3.7 Operators

```
/**
 */
package Tables;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

import org.eclipse.emf.common.util.Enumerator;

/**
 * <!-- begin-user-doc -->
 * A representation of the literals of the enumeration '<em><b>Ordering</b></em>',
 * and utility methods for working with them.
 * <!-- end-user-doc -->
 * @model
 * @generated
 */
public enum Ordering implements Enumerator {
        /**
         * The '<em><b>ASC</b></em>' literal object.
         * @generated
         * @ordered
         */
        ASC(0, "ASC", "ASC"),

        /**
         * The '<em><b>DESC</b></em>' literal object.
         * @generated
         * @ordered
         */
        DESC(1, "DESC", "DESC");

        /**
         * The '<em><b>ASC</b></em>' literal value.
         * @model
         * @generated
         * @ordered
         */
        public static final int ASC_VALUE = 0;
```

```
/**
 * The '<em><b>DESC</b></em>' literal value.
 * @model
 * @generated
 * @ordered
 */
public static final int DESC_VALUE = 1;

/**
 * An array of all the '<em><b>Ordering</b></em>' enumerators.
 * @generated
 */
private static final Ordering[] VALUES_ARRAY =
        new Ordering[] {
                ASC,
                DESC,
        };

/**
 * A public read-only list of all the '<em><b>Ordering</b></em>' enumerators.
 * @generated
 */
public static final List<Ordering> VALUES = Collections.unmodifiableList(Arrays.asList(VALUES_ARRAY));

/**
 * Returns the '<em><b>Ordering</b></em>' literal with the specified literal value.
 * @generated
 */
public static Ordering get(String literal) {
        for (int i = 0; i < VALUES_ARRAY.length; ++i) {
                Ordering result = VALUES_ARRAY[i];
                if (result.toString().equals(literal)) {
                        return result;
                }
        }
        return null;
}

/**
 * Returns the '<em><b>Ordering</b></em>' literal with the specified name.
 * @generated
 */
public static Ordering getByName(String name) {
        for (int i = 0; i < VALUES_ARRAY.length; ++i) {
                Ordering result = VALUES_ARRAY[i];
                if (result.getName().equals(name)) {
                        return result;
                }
        }
        return null;
}

/**
 * Returns the '<em><b>Ordering</b></em>' literal with the specified integer value.
 * @generated
 */
public static Ordering get(int value) {
        switch (value) {
                case ASC_VALUE: return ASC;
                case DESC_VALUE: return DESC;
        }
        return null;
}

/**
 * @generated
```

```
		 */
		private final int value;

		/**
		 * @generated
		 */
		private final String name;

		/**
		 * @generated
		 */
		private final String literal;

		/**
		 * Only this class can construct instances.
		 * @generated
		 */
		private Ordering(int value, String name, String literal) {
			this.value = value;
			this.name = name;
			this.literal = literal;
		}

		/**
		 * @generated
		 */
		public int getValue() {
		 return value;
		}

		/**
		 * @generated
		 */
		public String getName() {
		 return name;
		}

		/**
		 * @generated
		 */
		public String getLiteral() {
		 return literal;
		}

		/**
		 * Returns the literal value of the enumerator, which is its string representation.
		 * @generated
		 */
		@Override
		public String toString() {
			return literal;
		}

} //Ordering
```

Ordering class

3.3.8 Bol

```
/**
 */
package Tables;

import java.util.Arrays;
import java.util.Collections;
```

```
import java.util.List;

import org.eclipse.emf.common.util.Enumerator;

/**
 * <!-- begin-user-doc -->
 * A representation of the literals of the enumeration '<em><b>bol</b></em>',
 * and utility methods for working with them.
 * <!-- end-user-doc -->
 * @model
 * @generated
 */
public enum bol implements Enumerator {
	/**
	 * The '<em><b>AND</b></em>' literal object.
	 * @generated
	 * @ordered
	 */
	AND(0, "AND", "AND"),

	/**
	 * The '<em><b>OR</b></em>' literal object.
	 * @generated
	 * @ordered
	 */
	OR(1, "OR", "OR");

	/**
	 * The '<em><b>AND</b></em>' literal value.
	 * @model
	 * @generated
	 * @ordered
	 */
	public static final int AND_VALUE = 0;

	/**
	 * The '<em><b>OR</b></em>' literal value.
	 * @model
	 * @generated
	 * @ordered
	 */
	public static final int OR_VALUE = 1;

	/**
	 * An array of all the '<em><b>bol</b></em>' enumerators.
	 * @generated
	 */
	private static final bol[] VALUES_ARRAY =
		new bol[] {
			AND,
			OR,
		};

	/**
	 * A public read-only list of all the '<em><b>bol</b></em>' enumerators.
	 * @generated
	 */
	public static final List<bol> VALUES = Collections.unmodifiableList(Arrays.asList(VALUES_ARRAY));

	/**
	 * Returns the '<em><b>bol</b></em>' literal with the specified literal value.
	 * @generated
	 */
	public static bol get(String literal) {
		for (int i = 0; i < VALUES_ARRAY.length; ++i) {
			bol result = VALUES_ARRAY[i];
			if (result.toString().equals(literal)) {
```

```
                                        return result;
                                }
                        }
                        return null;
        }

        /**
         * Returns the '<em><b>bol</b></em>' literal with the specified name.
         * @generated
         */
        public static bol getByName(String name) {
                        for (int i = 0; i < VALUES_ARRAY.length; ++i) {
                                bol result = VALUES_ARRAY[i];
                                if (result.getName().equals(name)) {
                                        return result;
                                }
                        }
                        return null;
        }

        /**
         * Returns the '<em><b>bol</b></em>' literal with the specified integer value.
         * @generated
         */
        public static bol get(int value) {
                        switch (value) {
                                case AND_VALUE: return AND;
                                case OR_VALUE: return OR;
                        }
                        return null;
        }

        /**
         * @generated
         */
        private final int value;

        /**
         * @generated
         */
        private final String name;

        /**
         * @generated
         */
        private final String literal;

        /**
         * Only this class can construct instances.
         * @generated
         */
        private bol(int value, String name, String literal) {
                        this.value = value;
                        this.name = name;
                        this.literal = literal;
        }

        /**
         * @generated
         */
        public int getValue() {
         return value;
        }

        /**
         * @generated
         */
```

```
		public String getName() {
		 return name;
		}

		/**
		 * @generated
		 */
		public String getLiteral() {
		 return literal;
		}

		/**
		 * Returns the literal value of the enumerator, which is its string representation.
		 * @generated
		 */
		@Override
		public String toString() {
			return literal;
		}

} //bol
```

Bol class

3.3.9 NaryNode

```
/**
 */
package Tables;

import org.eclipse.emf.ecore.EObject;

/**
 * <!-- begin-user-doc -->
 * A representation of the model object '<em><b>Nary Node</b></em>'.
 * <!-- end-user-doc -->
 * @model
 * @generated
 */
public interface NaryNode extends EObject {
		/**
		 * Returns the value of the '<em><b>Bl</b></em>' attribute.
		 * The literals are from the enumeration {@link Tables.bol}.
		 * @return the value of the '<em>Bl</em>' attribute.
		 * @model required="true"
		 * @generated
		 */
		bol getBl();

		/**
		 * Sets the value of the '{@link Tables.NaryNode#getBl <em>Bl</em>}' attribute.
		 * @param value the new value of the '<em>Bl</em>' attribute.
		 * @generated
		 */
		void setBl(bol value);

		/**
		 * Returns the value of the '<em><b>Value</b></em>' reference.
		 * @return the value of the '<em>Value</em>' reference.
		 * @model required="true"
		 * @generated
		 */
		Filters<?> getValue();

		/**
```

```
		 * Sets the value of the '{@link Tables.NaryNode#getValue <em>Value</em>}' reference.
		 * @param value the new value of the '<em>Value</em>' reference.
		 * @generated
		 */
		void setValue(Filters<?> value);

} // NaryNode
```

NaryNode Interface

```
/**
 */
package Tables.impl;

import Tables.Filters;
import Tables.NaryNode;
import Tables.TablesPackage;
import Tables.bol;

import org.eclipse.emf.common.notify.Notification;

import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;

import org.eclipse.emf.ecore.impl.ENotificationImpl;
import org.eclipse.emf.ecore.impl.MinimalEObjectImpl;

/**
 * <!-- begin-user-doc -->
 * An implementation of the model object '<em><b>Nary Node</b></em>'.
 * <!-- end-user-doc -->
 * @generated
 */
public class NaryNodeImpl extends MinimalEObjectImpl.Container implements NaryNode {
		/**
		 * The default value of the '{@link #getBl() <em>Bl</em>}' attribute.
		 * @generated
		 * @ordered
		 */
		protected static final bol BL_EDEFAULT = bol.AND;

		/**
		 * The cached value of the '{@link #getBl() <em>Bl</em>}' attribute.
		 * @generated
		 * @ordered
		 */
		protected bol bl = BL_EDEFAULT;

		/**
		 * The cached value of the '{@link #getValue() <em>Value</em>}' reference.
		 * @generated
		 * @ordered
		 */
		protected Filters<?> value;

		/**
		 * @generated
		 */
		protected NaryNodeImpl() {
			super();
		}

		/**
		 * @generated
```

```
		 */
		@Override
		protected EClass eStaticClass() {
			return TablesPackage.Literals.NARY_NODE;
		}

		/**
		 * @generated
		 */
		public bol getBl() {
			return bl;
		}

		/**
		 * @generated
		 */
		public void setBl(bol newBl) {
			bol oldBl = bl;
			bl = newBl == null ? BL_EDEFAULT : newBl;
			if (eNotificationRequired())
				eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.NARY_NODE__BL, oldBl, bl));
		}

		/**
		 * @generated
		 */
		public Filters<?> getValue() {
			if (value != null && value.eIsProxy()) {
				InternalEObject oldValue = (InternalEObject)value;
				value = (Filters<?>)eResolveProxy(oldValue);
				if (value != oldValue) {
					if (eNotificationRequired())
						eNotify(new ENotificationImpl(this, Notification.RESOLVE,
TablesPackage.NARY_NODE__VALUE, oldValue, value));
				}
			}
			return value;
		}

		/**
		 * @generated
		 */
		public Filters<?> basicGetValue() {
			return value;
		}

		/**
		 * @generated
		 */
		public void setValue(Filters<?> newValue) {
			Filters<?> oldValue = value;
			value = newValue;
			if (eNotificationRequired())
				eNotify(new ENotificationImpl(this, Notification.SET, TablesPackage.NARY_NODE__VALUE, oldValue,
value));
		}

		/**
		 * @generated
		 */
		@Override
		public Object eGet(int featureID, boolean resolve, boolean coreType) {
			switch (featureID) {
				case TablesPackage.NARY_NODE__BL:
					return getBl();
				case TablesPackage.NARY_NODE__VALUE:
					if (resolve) return getValue();
```

```
                                        return basicGetValue();
                        }
                        return super.eGet(featureID, resolve, coreType);
        }

        /**
         * @generated
         */
        @Override
        public void eSet(int featureID, Object newValue) {
                        switch (featureID) {
                                        case TablesPackage.NARY_NODE__BL:
                                                        setBl((bol)newValue);
                                                        return;
                                        case TablesPackage.NARY_NODE__VALUE:
                                                        setValue((Filters<?>)newValue);
                                                        return;
                        }
                        super.eSet(featureID, newValue);
        }

        /**
         * @generated
         */
        @Override
        public void eUnset(int featureID) {
                        switch (featureID) {
                                        case TablesPackage.NARY_NODE__BL:
                                                        setBl(BL_EDEFAULT);
                                                        return;
                                        case TablesPackage.NARY_NODE__VALUE:
                                                        setValue((Filters<?>)null);
                                                        return;
                        }
                        super.eUnset(featureID);
        }

        /**
         * @generated
         */
        @Override
        public boolean eIsSet(int featureID) {
                        switch (featureID) {
                                        case TablesPackage.NARY_NODE__BL:
                                                        return bl != BL_EDEFAULT;
                                        case TablesPackage.NARY_NODE__VALUE:
                                                        return value != null;
                        }
                        return super.eIsSet(featureID);
        }

        /**
         * @generated
         */
        @Override
        public String toString() {
                        if (eIsProxy()) return super.toString();

                        StringBuffer result = new StringBuffer(super.toString());
                        result.append(" (bl: ");
                        result.append(bl);
                        result.append(')');
                        return result.toString();
        }

} //NaryNodeImpl
```

NaryNode Implementation

3.3.10 Factory

```
/**
 */
package Tables;

import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.EFactory;
import org.eclipse.emf.ecore.EObject;

/**
 * <!-- begin-user-doc -->
 * The <b>Factory</b> for the model.
 * It provides a create method for each non-abstract class of the model.
 * <!-- end-user-doc -->
 * @see Tables.TablesPackage
 * @generated
 */
public interface TablesFactory extends EFactory {
	/**
	 * The singleton instance of the factory.
	 * @generated
	 */
	TablesFactory eINSTANCE = Tables.impl.TablesFactoryImpl.init();

	/**
	 * Returns a new object of class '<em>Table</em>'.

	 * @return a new object of class '<em>Table</em>'.
	 * @generated
	 */
	Table createTable();

	/**
	 * Returns a new object of class '<em>Field</em>'.
	 * @return a new object of class '<em>Field</em>'.
	 * @generated
	 */
	<T, TableInstance extends Table> Field<T, TableInstance> createField();

	/**
	 * Returns a new object of class '<em>Field</em>'.
	 * @return a new object of class '<em>Field</em>'.
	 * @generated NOT
	 */
	<F extends EClass> Field createFieldCustom();

	/**
	 * Returns a new object of class '<em>Values</em>'.
	 * @return a new object of class '<em>Values</em>'.
	 * @generated
	 */
	Values createValues();

	/**
	 * Returns a new object of class '<em>Query GET</em>'.
	 * @return a new object of class '<em>Query GET</em>'.
	 * @generated
	 */
	<TableInstance extends Table> QueryGET<TableInstance> createQueryGET();
```

```
		/**
		 * Returns a new object of class '<em>FROM</em>'.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @return a new object of class '<em>FROM</em>'.
		 * @generated NOT
		 */
		<TableInstance extends Table> QueryGET<TableInstance> createQueryGET(Class<TableInstance> tableInstance);


		/**
		 * Returns a new object of class '<em>JOIN</em>'.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @return a new object of class '<em>JOIN</em>'.
		 * @generated
		 */
		<InitialTable extends Table, JoinedTable extends Table, Type> JOIN<InitialTable, JoinedTable, Type> createJOIN();


		/**
		 * Returns a new object of class '<em>Filters</em>'.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @return a new object of class '<em>Filters</em>'.
		 * @generated
		 */
		<TableInstance extends Table> Filters<TableInstance> createFilters();

		/**
		 * Returns a new object of class '<em>Filters</em>'.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @return a new object of class '<em>Filters</em>'.
		 * @generated NOT
		 */
		<TableInstance extends Table> Filters<TableInstance> createFilters(Field<?,?> field, Operators op, Object other,
JOIN<?,TableInstance,?> join);

		/**
		 * Returns a new object of class '<em>Condition</em>'.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @return a new object of class '<em>Condition</em>'.
		 * @generated
		 */
		<T extends Object, TableInstance extends Table> Condition<T, TableInstance> createCondition();

		/**
		 * Returns a new object of class '<em>Order By</em>'.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @return a new object of class '<em>Order By</em>'.
		 * @generated
		 */
		OrderBy createOrderBy();

		/**
		 * Returns a new object of class '<em>Nary Node</em>'.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @return a new object of class '<em>Nary Node</em>'.
		 * @generated
		 */
		NaryNode createNaryNode();

		/**
```

```
		 * Returns a new object of class '<em>JOIN</em>'.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @return a new object of class '<em>JOIN</em>'.
		 * @generated NOT
		 */
		<InitialTable extends Table, JoinedTable extends Table,Type> JOIN<InitialTable, JoinedTable,Type>
createJOIN(Class<InitialTable> initialTable,Class<JoinedTable> joinedTable);

		/**
		 * Returns the package supported by this factory.
		 * <!-- begin-user-doc -->
		 * <!-- end-user-doc -->
		 * @return the package supported by this factory.
		 * @generated
		 */
		TablesPackage getTablesPackage();

} //TablesFactory
```

Factory Interface

```
/**
 */
package Tables.impl;

import Tables.*;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.EdataType;
import org.eclipse.emf.ecore.Eobject;
import org.eclipse.emf.ecore.Epackage;
import org.eclipse.emf.ecore.impl.EfactoryImpl;
import org.eclipse.emf.ecore.plugin.EcorePlugin;
import Tables.Condition;
import Tables.Database;
import Tables.FROM;
import Tables.Field;
import Tables.Filters;
import Tables.JOIN;
import Tables.Operators;
import Tables.OrderBy;
import Tables.Ordering;
import Tables.QueryDELETE;
import Tables.QueryGET;
import Tables.QueryPOST;
import Tables.QueryPUT;
import Tables.Simple_Query;
import Tables.Table;
import Tables.TablesFactory;
import Tables.TablesPackage;
import Tables.Values;

/**
 * <!—begin-user-doc →
 * An implementation of the model <b>Factory</b>.
 * <!—end-user-doc →
 * @generated
 */
public class TablesFactoryImpl extends EfactoryImpl implements TablesFactory {

		private Condition condition;
		/**
		 * Creates the default factory implementation.
		 * <!—begin-user-doc →
```

```
		 * <!—end-user-doc →
		 * @generated
		 */
		public static TablesFactory init() {
			try {
				TablesFactory theTablesFactory =
(TablesFactory)Epackage.Registry.INSTANCE.getEFactory(TablesPackage.eNS_URI);
				if (theTablesFactory != null) {
					return theTablesFactory;
				}
			}
			catch (Exception exception) {
				EcorePlugin.INSTANCE.log(exception);
			}
			return new TablesFactoryImpl();
		}

		/**
		 * Creates an instance of the factory.
		 * <!—begin-user-doc →
		 * <!—end-user-doc →
		 * @generated
		 */
		public TablesFactoryImpl() {
			super();
		}

		/**
		 * <!—begin-user-doc →
		 * <!—end-user-doc →
		 * @generated
		 */
		@Override
		public Eobject create(Eclass eClass) {
			switch (eClass.getClassifierID()) {
				case TablesPackage.DATABASE: return createDatabase();
				case TablesPackage.TABLE: return createTable();
				case TablesPackage.FIELD: return createField();
				case TablesPackage.VALUES: return createValues();
				case TablesPackage.QUERY_GET: return createQueryGET();
				case TablesPackage.QUERY_POST: return createQueryPOST();
				case TablesPackage.QUERY_PUT: return createQueryPUT();
				case TablesPackage.QUERY_DELETE: return createQueryDELETE();
				case TablesPackage.SIMPLE_QUERY: return createSimple_Query();
				case TablesPackage.FROM: return createFROM();
				case TablesPackage.JOIN: return createJOIN();
				case TablesPackage.FILTERS: return createFilters();
				case TablesPackage.CONDITION: return createCondition();
				case TablesPackage.ORDER_BY: return createOrderBy();
				case TablesPackage.NARY_NODE: return createNaryNode();
				default:
					throw new IllegalArgumentException("The class '" + eClass.getName() + "' is not a valid
classifier");
			}
		}

		/**
		 * <!—begin-user-doc →
		 * <!—end-user-doc →
		 * @generated
		 */
		@Override
		public Object createFromString(EdataType eDataType, String initialValue) {
			switch (eDataType.getClassifierID()) {
				case TablesPackage.OPERATORS:
					return createOperatorsFromString(eDataType, initialValue);
				case TablesPackage.ORDERING:
```

```
                                        return createOrderingFromString(eDataType, initialValue);
                                case TablesPackage.BOL:
                                        return createbolFromString(eDataType, initialValue);
                                default:
                                        throw new IllegalArgumentException("The datatype '" + eDataType.getName() + "' is not a
valid classifier");
                        }
                }

                /**
                 * <!—begin-user-doc →
                 * <!—end-user-doc →
                 * @generated
                 */
                @Override
                public String convertToString(EdataType eDataType, Object instanceValue) {
                        switch (eDataType.getClassifierID()) {
                                case TablesPackage.OPERATORS:
                                        return convertOperatorsToString(eDataType, instanceValue);
                                case TablesPackage.ORDERING:
                                        return convertOrderingToString(eDataType, instanceValue);
                                case TablesPackage.BOL:
                                        return convertbolToString(eDataType, instanceValue);
                                default:
                                        throw new IllegalArgumentException("The datatype '" + eDataType.getName() + "' is not a
valid classifier");
                        }
                }

                /**
                 * <!—begin-user-doc →
                 * <!—end-user-doc →
                 * @generated
                 */
                public Table createTable() {
                        TableImpl table = new TableImpl();
                        return table;
                }

                /**
                 * <!—begin-user-doc →
                 * <!—end-user-doc →
                 * @generated
                 */
                public <T, TableInstance extends Table> Field<T, TableInstance> createField() {
                        FieldImpl<T, TableInstance> field = new FieldImpl<T, TableInstance>();
                        return field;
                }

                /**
                 * <!—begin-user-doc →
                 * <!—end-user-doc →
                 * @generated NOT
                 */
                public <F extends Eclass> Field createFieldCustom() {
                        FieldImpl field = new FieldImpl();
                        return field;
                }

                /**
                 * <!—begin-user-doc →
                 * <!—end-user-doc →
                 * @generated
                 */
                public <TableInstance extends Table> QueryGET<TableInstance> createQueryGET() {
                        QueryGETImpl<TableInstance> queryGET = new QueryGETImpl<TableInstance>();
                        return queryGET;
```

```
	}

	/**
	 * <!—begin-user-doc →
	 * <!—end-user-doc →
	 * @generated NOT
	 */
	public <TableInstance extends Table> QueryGET<TableInstance> createQueryGET(Class<TableInstance> tableInstance) {
		QueryGETImpl<TableInstance> queryGET = new QueryGETImpl<TableInstance>(tableInstance);
		return queryGET;
	}

	/**
	 * <!—begin-user-doc →
	 * <!—end-user-doc →
	 * @generated
	 */
	public <InitialTable extends Table, JoinedTable extends Table, Type> JOIN<InitialTable, JoinedTable, Type> createJOIN() {
		JOINImpl<InitialTable, JoinedTable, Type> join = new JOINImpl<InitialTable, JoinedTable, Type>();
		return join;
	}

	/**
	 * <!—begin-user-doc →
	 * <!—end-user-doc →
	 * @generated
	 */
	public <TableInstance extends Table> Filters<TableInstance> createFilters() {
		FiltersImpl<TableInstance> filters = new FiltersImpl<TableInstance>();
		return filters;
	}

	/**
	 * <!—begin-user-doc →
	 * <!—end-user-doc →
	 * @generated
	 */
	public <T extends Object, TableInstance extends Table> Condition<T, TableInstance> createCondition() {
		ConditionImpl<T, TableInstance> condition = new ConditionImpl<T, TableInstance>();
		return condition;
	}

	/**
	 * <!—begin-user-doc →
	 * <!—end-user-doc →
	 * @generated NOT
	 */
	public <TableInstance extends Table> Filters<TableInstance> createFilters(Field<?,?> field, Operators op, Object other,
JOIN<?,TableInstance,?> join) {
		FiltersImpl<TableInstance> filters = new FiltersImpl<TableInstance>(field,op,other,join);
		return filters;
	}

	/**
	 * <!—begin-user-doc →
	 * <!—end-user-doc →
	 * @generated
	 */
	public OrderBy createOrderBy() {
		OrderByImpl 140rderBy = new OrderByImpl();
		return 140rderBy;
	}

	/**
	 * <!—begin-user-doc →
	 * <!—end-user-doc →
	 * @generated
```

```
		 */
		public NaryNode createNaryNode() {
			NaryNodeImpl naryNode = new NaryNodeImpl();
			return naryNode;
		}

		/**
		 * <!—begin-user-doc →
		 * <!—end-user-doc →
		 * @generated
		 */
		public Operators createOperatorsFromString(EdataType eDataType, String initialValue) {
			Operators result = Operators.get(initialValue);
			if (result == null) throw new IllegalArgumentException("The value '" + initialValue + "' is not a valid enumerator of
'" + eDataType.getName() + "'");
			return result;
		}

		/**
		 * <!—begin-user-doc →
		 * <!—end-user-doc →
		 * @generated
		 */
		public String convertOperatorsToString(EdataType eDataType, Object instanceValue) {
			return instanceValue == null ? null : instanceValue.toString();
		}

		/**
		 * <!—begin-user-doc →
		 * <!—end-user-doc →
		 * @generated
		 */
		public Ordering createOrderingFromString(EdataType eDataType, String initialValue) {
			Ordering result = Ordering.get(initialValue);
			if (result == null) throw new IllegalArgumentException("The value '" + initialValue + "' is not a valid enumerator of
'" + eDataType.getName() + "'");
			return result;
		}

		/**
		 * <!—begin-user-doc →
		 * <!—end-user-doc →
		 * @generated
		 */
		public String convertOrderingToString(EdataType eDataType, Object instanceValue) {
			return instanceValue == null ? null : instanceValue.toString();
		}

		/**
		 * <!—begin-user-doc →
		 * <!—end-user-doc →
		 * @generated
		 */
		public bol createbolFromString(EdataType eDataType, String initialValue) {
			bol result = bol.get(initialValue);
			if (result == null) throw new IllegalArgumentException("The value '" + initialValue + "' is not a valid enumerator of
'" + eDataType.getName() + "'");
			return result;
		}

		/**
		 * <!—begin-user-doc →
		 * <!—end-user-doc →
		 * @generated
		 */
		public String convertbolToString(EdataType eDataType, Object instanceValue) {
			return instanceValue == null ? null : instanceValue.toString();
```

```
		}

		/**
		 * <!—begin-user-doc →
		 * <!—end-user-doc →
		 * @generated NOT
		 */
		public <InitialTable extends Table, JoinedTable extends Table,Type> JOIN<InitialTable, JoinedTable,Type>
createJOIN(Class<InitialTable> initialTable,Class<JoinedTable> joinedTable) {
			JOINImpl<InitialTable,JoinedTable,Type> join = new
JOINImpl<InitialTable,JoinedTable,Type>(initialTable,joinedTable);
			return join;
		}

		/**
		 * <!—begin-user-doc →
		 * <!—end-user-doc →
		 * @generated
		 */
		public TablesPackage getTablesPackage() {
			return (TablesPackage)getEPackage();
		}

		/**
		 * <!—begin-user-doc →
		 * <!—end-user-doc →
		 * @deprecated
		 * @generated
		 */
		@Deprecated
		public static TablesPackage getPackage() {
			return TablesPackage.eINSTANCE;
		}

} //TablesFactoryImpl
```

Factory Implementation

3.3.11 InitializeMetaData

```
package Tables.util;

import java.io.File;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.lang.annotation.Annotation;
import java.lang.reflect.Method;
import java.net.URL;
import java.net.URLDecoder;
import java.security.CodeSource;
import java.security.ProtectionDomain;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.List;

import Tables.Field;
import Tables.Annotation.InitMetaData;
import Tables.impl.FieldImpl;


public class InitializeMetaData {

		private static final char DOT = '.';
```

```
  private static final char SLASH = '/';
  private static final String CLASS_SUFFIX = ".class";
  private static final String BAD_PACKAGE_ERROR = "Unable to get resources from path '%s'. Are you sure the package '%s' exists?";



            public static List<Class<?>> initializeFromPackage(String scannedPackage) throws IllegalArgumentException,
IllegalAccessException{
                        /*Create a path out of the given scannedPackage*/
                        String scannedPath = scannedPackage.replace(DOT, SLASH);
                        /*Get the class loader of the scannedPath*/
    URL scannedUrl = Thread.currentThread().getContextClassLoader().getResource(scannedPath);
    if (scannedUrl == null) {
      throw new IllegalArgumentException(String.format(BAD_PACKAGE_ERROR, scannedPath, scannedPackage));
    }
    /*Create a file*/
    File scannedDir = new File(scannedUrl.getFile());
    List<Class<?>> classes = new ArrayList<Class<?>>();
    /*get the list of files contained in the scannedDir and put them in the list with type. Class<?>*/
    for (File file : scannedDir.listFiles()) {
      classes.addAll(find(file, scannedPackage));
    }

    /*We now have all the classes of the package "scannedPackage"*/
    for(Class<?> annot:classes){
            /*See if the class has the annotation InitMetaData*/
            Annotation A = annot.getAnnotation(InitMetaData.class);
            if(A!=null){
                        /*Using reflection get all the fields of that class*/
                        java.lang.reflect.Field[] fields = annot.getDeclaredFields();
                        for(java.lang.reflect.Field fld:fields){
                                    /*Check if the type of the field is "Field"*/
                                    if(fld.getType().getSimpleName().equals("Field")){
                                                /*Initialize the Field with default data*/
                                                Field fldd = new FieldImpl();
                                                fldd.setAlias("_"+fld.getName().toString());
                                                fldd.setFieldName(fld.getName().toString());
                                                fld.set(null, fldd);
                                    }
                        }
            }
    }

    return classes;
            }

            /*find all the classes in the package*/
            private static List<Class<?>> find(File file, String scannedPackage) {
    List<Class<?>> classes = new ArrayList<Class<?>>();
    String resource = scannedPackage + DOT + file.getName();
    if (file.isDirectory()) {
      for (File child : file.listFiles()) {
        classes.addAll(find(child, resource));
      }
    } else if (resource.endsWith(CLASS_SUFFIX)) {
      int endIndex = resource.length() - CLASS_SUFFIX.length();
      String className = resource.substring(0, endIndex);
      try {
        classes.add(Class.forName(className));
      } catch (ClassNotFoundException ignore) {
      }
    }
    return classes;
  }

}
```

InitializeMetaData.java

3.3.12 SqlModelAnnotation

3.3.12.1 AnnotationProcessor

```
package Tables.Annotation;

import gencode.Tables.Annotation.MetaSource;

import java.io.File;
import java.io.IOException;
import java.lang.annotation.Annotation;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLClassLoader;
import java.util.List;
import java.util.Set;

import javax.annotation.processing.AbstractProcessor;
import javax.annotation.processing.Filer;
import javax.annotation.processing.Messager;
import javax.annotation.processing.ProcessingEnvironment;
import javax.annotation.processing.RoundEnvironment;
import javax.annotation.processing.SupportedAnnotationTypes;
import javax.annotation.processing.SupportedSourceVersion;
import javax.lang.model.element.AnnotationMirror;
import javax.lang.model.element.Element;
import javax.lang.model.type.TypeMirror;
import javax.lang.model.SourceVersion;
import javax.persistence.JoinColumn;
import javax.tools.Diagnostic.Kind;
import javax.tools.JavaFileObject;

@SupportedAnnotationTypes(value= {"Tables.Annotation.GenerateMetadata"})
@SupportedSourceVersion(SourceVersion.RELEASE_8)
public class AnnotationProcessor extends AbstractProcessor{

	private Filer filer;
	private Messager messager;

	@Override
	public void init(ProcessingEnvironment env) {
		filer = env.getFiler();
		messager = env.getMessager();
	}

	@Override
	public boolean process(@SuppressWarnings("rawtypes") Set elements, RoundEnvironment env) {

		for (Element element : env.getElementsAnnotatedWith(GenerateMetadata.class)){

			MetaStorage stored = new MetaStorage();

			for(Element littleElements:element.getEnclosedElements()){
				if(littleElements.getKind().isField()){
					TypeMirror type =  littleElements.asType();
					String objectType = null;
					objectType = primitiveToObjectConverter(type.toString());
				stored.getFieldMapping().put(littleElements.toString(), objectType);
					}
				}
```

```
                                        stored.setClassName(element.getSimpleName().toString());
                                        stored.setMetaClassName("_"+element.getSimpleName().toString());
                                        String className ="_"+element.getSimpleName().toString();

                                        JavaFileObject file = null;

                                        MetaSource met = new MetaSource();

                                        try {
                                                file = filer.createSourceFile(
                                                                "meta/" + className,
                                                                element);
                                                file.openWriter().append(met.generate(stored)).close();
                                        } catch (IOException e) {
                                                e.printStackTrace();
                                        }
                                }
                        return true;
                }

                private String primitiveToObjectConverter(String primitive){
                        String object;

                        switch(primitive){
                                case "int": object = "java.lang.Integer";
                                                                break;
                                case "short": object = "java.lang.Short";
                                                                break;
                                case "long": object = "java.lang.Long";
                                                                break;
                                case "byte": object = "java.lang.Byte";
                                                                break;
                                case "float": object = "java.lang.Float";
                                                                break;
                                case "double": object = "java.lang.Double";
                                                                break;
                                case "char": object = "java.lang.Character";
                                                                break;
                                case "boolean": object = "java.lang.Boolean";
                                                                break;
                                default:    object = primitive;
                        }

                        return object;

                }

}
```

AnnotationProcessor.java

3.3.12.2 MetaStorage

```
package Tables.Annotation;

import java.util.HashMap;
import java.util.Map;

public class MetaStorage {

                private String className;
                private String metaClassName;
                private Map<String,String> fieldMapping;
```

```
	public MetaStorage() {
		super();
		this.className = new String();
		this.fieldMapping = new HashMap<String,String>();
		this.metaClassName = new String();
	}

	public String getClassName() {
		return className;
	}
	public void setClassName(String className) {
		this.className = className;
	}

	public Map<String,String> getFieldMapping() {
		return fieldMapping;
	}
	public void setFieldMapping(Map<String,String> fieldMapping) {
		this.fieldMapping = fieldMapping;
	}

	public String getMetaClassName() {
		return metaClassName;
	}

	public void setMetaClassName(String metaClassName) {
		this.metaClassName = metaClassName;
	}
}
```

MetaStorage.java

3.3.12.3 metagen

```
<%@ jet package="gencode.Tables.Annotation"
class="MetaSource" imports ="java.util.Iterator java.util.Map Tables.Annotation.MetaStorage"
%>
<% MetaStorage stored = (MetaStorage) argument; %>
package meta;

import Tables.Field;
import Tables.impl.TableImpl;
import Tables.Annotation.InitMetaData;


@InitMetaData
public class <%=stored.getMetaClassName()%> extends TableImpl{

	public final String tableName = "<%=stored.getClassName()%>";
	<%
	Iterator iter = stored.getFieldMapping().entrySet().iterator();
	while(iter.hasNext()){
		Map.Entry pair = (Map.Entry)iter.next();
	%>
	public static volatile Field<<%=pair.getValue()%>,<%=stored.getMetaClassName()%>> <%=pair.getKey()%> ;
	<%
		iter.remove();
	}
	%>
}
```

Metagen.javajet

ΚΕΦΑΛΑΙΟ 4

Παραδείγματα Λειτουργίας

Στην τελευταία ενότητα θα δοθούν μερικά παραδείγματα λειτουργίας της επέκτασης.

Οι πίνακες που χρησιμοποιούνται στα παραδείγματα ονομάζονται Person, Hobby και Type και περιέχουν τα εξής δεδομένα:

id	name	hobby_id
1	jason	3
2	jason	2
3	elisabeth	1
4	marios	5
5	alexander	4

Person Table

id	name	type_id
1	reading	3
2	chess	1
3	video_games	1
4	watch_tv	4
5	workout	2

Hobby Table

id	name
1	Game
2	Fitness
3	mind
4	leisure

Type Table

Απλό SELECT

Στην αρχή δίνεται ένα παράδειγμα στο οποίο επιλέγεται ο πίνακας Person και επιστρέφει το πεδίο name όλων των εγγραφών του πίνακα:

```
/*Simple test, return all persons from a table*/
public void simpleTest() throws IllegalArgumentException, IllegalAccessException{
        List<Class<?>> rs = InitializeMetaData.initializeFromPackage("meta");

        QueryGET<_Person> queryGET = TablesFactory.eINSTANCE.createQueryGET(_Person.class);

        queryGET.setRootAlias("people");
        queryGET.getSelectedFields().add(_Person.name);
```

```
		List<?> returned = queryGET.generate(Person.class);

		System.out.println("Final results");
		System.out.println("=========================");
		for(Person prs: (List<Person>) returned){
			System.out.println("Name: "+prs.getName());
		}
}
```

Και η αντίστοιχη έξοδος είναι η εξής:

```
Query Generated: SELECT  people.name AS _name1876631416 FROM Person AS people
Final results
=========================
Name: marios
Name: alexander
Name: jason
Name: elisabeth
```

Σε αυτό το παράδειγμα, μετά την αρχικοποίηση:

```
List<Class<?>> rs = InitializeMetaData.initializeFromPackage("meta");
```

Καλούμε το Factory για να δημιουργήσουμε μια ρίζα-πίνακα:

```
QueryGET<_Person> queryGET = TablesFactory.eINSTANCE.createQueryGET(_Person.class);
```

Στη συνέχεια θέτουμε ένα ψευδώνυμο στον πίνακα και ζητάμε την επιστροφή του πεδίου name:

```
queryGET.setRootAlias("people");
queryGET.getSelectedFields().add(_Person.name);
```

Τέλος καλούμε την generate(…) και παρουσιάζουμε τα δεδομένα:

```
List<?> returned = queryGET.generate(Person.class);
```

Απλό SELECT με πολλαπλά πεδία

```
/*Simple test, return all persons from a table and their ids*/
private void simpleSELECT() throws IllegalArgumentException, IllegalAccessException{
		List<Class<?>> rs = InitializeMetaData.initializeFromPackage("meta");

		QueryGET<_Person> queryGET = TablesFactory.eINSTANCE.createQueryGET(_Person.class);

		queryGET.setRootAlias("people");
		queryGET.getSelectedFields().add(_Person.id);
		queryGET.getSelectedFields().add(_Person.name);

		List<?> returned = queryGET.generate(Person.class);

		System.out.println("Final results");
```

```
		System.out.println("=========================");
		for(Person prs: (List<Person>) returned){
			System.out.println("Name: "+prs.getName()+" with id: "+prs.getId());
		}
}
```

Το αποτέλεσμα είναι το εξής:

```
Query Generated: SELECT  people.id AS _id692342133, people.name AS _name692342133 FROM Person AS
people
Final results
=========================
Name: marios with id: 4
Name: elisabeth with id: 3
Name: alexander with id: 5
Name: jason with id: 1
Name: jason with id: 2
```

Προφανώς σε αυτή την περίπτωση έχουμε δύο εγγραφές jason αφού ο πίνακας δεν έχει ως primary key το πεδίο name και μπορεί κάποιος να εισάγει ίδιες τιμές. Όμως στο προηγούμενο παράδειγμα παρατηρήσαμε πως επιστράφηκε μόνο μια εγγραφή Jason!
Αυτό συνέβη γιατί ζητήθηκε μόνο το πεδίο name, οπότε και επιστράφηκαν μοναδικές εγγραφές με το πεδίο name. Το να επιστραφούν πολλαπλές εγγραφές με ίδιο περιεχόμενο εμποδίζεται. Στο δεύτερο παράδειγμα όμως που ζητείται και το id, το συνολικό περιεχόμενο κάθε εγγραφής αλλάζει. Τώρα περιέχεται και το πεδίο name και το πεδίο id, οπότε το συνολικό περιεχόμενο των εγγραφών θα είναι διαφορετικό. Επειδή ζητήσαμε το id, το οποίο σε αυτή την περίπτωση είναι και το primary key του πίνακα θα επιστραφούν όλες οι εγγραφές του πίνακα Person.

Απλό SELECT με φίλτρο WHERE

Στο τρίτο παράδειγμα εφαρμόζουμε στο ερώτημα ένα φίλτρο για να φέρουμε μόνο τις εγγραφές των οποίων το πεδίο name ισούται με “jason”

```
/*Return the person named "jason" */
private void simpleSELECTwithWHERE() throws IllegalArgumentException, IllegalAccessException{
		List<Class<?>> rs = InitializeMetaData.initializeFromPackage("meta");

		QueryGET<_Person> queryGET = TablesFactory.eINSTANCE.createQueryGET(_Person.class);

		queryGET.setRootAlias("people");
		queryGET.getSelectedFields().add(_Person.name);

		Filters<_Person> whereroot =
TablesFactory.eINSTANCE.createFilters(_Person.name,Operators.EQUALS,"jason",null);

		queryGET.setFilter(whereroot);

		List<?> returned = queryGET.generate(Person.class);

		System.out.println("Final results");
		System.out.println("=========================");
		for(Person prs: (List<Person>) returned){
			System.out.println("Name: "+prs.getName());
```

```
		}
}
```

Η έξοδος της παράπανω συνάρτησης είναι η εξής:

```
Query Generated: SELECT  people.name AS _name692342133 FROM Person AS people WHERE (people.name
= 'jason')
Final results
==========================
Name: jason
```

Η διαφορά σε σχέση με το προηγούμενο ερώτημα είναι οι γραμμές:

```
Filters<_Person> whereroot =
TablesFactory.eINSTANCE.createFilters(_Person.name,Operators.EQUALS,"jason",null);
queryGET.setFilter(whereroot);
```

Σε αυτή την περίπτωση εφαρμόζουμε ένα απλό φίλτρο ισότητας. Όπως και στα προηγούμενα παραδείγματα αν προσθέταμε στα επιστρεφόμενα πεδία και το πεδίο Id θα μας επέστρεφε και τις δύο εγγραφές jason.

SELECT με συνένωση ενός πίνακα

Σε αυτό το παράδειγμα γίνεται συνένωση δύο πινάκων, των Person και Hobby.

```
/*Select with a simple join association*/
private void sELECTwithSimpleJOIN() throws IllegalArgumentException, IllegalAccessException{
		List<Class<?>> rs = InitializeMetaData.initializeFromPackage("meta");

		QueryGET<_Person> queryGET = TablesFactory.eINSTANCE.createQueryGET(_Person.class);

		queryGET.setRootAlias("people");
		queryGET.getSelectedFields().add(_Person.name);

		JOIN<_Person,_Hobby,Integer> join = TablesFactory.eINSTANCE.createJOIN(_Person.class, _Hobby.class);
		join.setJoinedAlias("hobby");
		join.setOnFieldInitial(_Person.hobby_id);
		join.setOnFieldJoined(_Hobby.id);
		join.getJoinedTableFields().add(_Hobby.name);

		queryGET.getJoin().add(join);

		List<?> returned = queryGET.generate(Person.class);

		System.out.println("Final results");
		System.out.println("==========================");
		for(Person prs: (List<Person>) returned){
			System.out.println("Name: "+prs.getName());
			for(Hobby hobs: prs.getHobbies()){
				System.out.println("   Hobby: "+hobs.getName());
			}
		}
}
```

Και η έξοδος είναι η εξής:

```
Query Generated: SELECT  people.name AS _name692342133 ,hobby.name AS _name578866604 FROM
Person AS people JOIN Hobby AS hobby ON people.hobby_id=hobby.id
Final results
=========================
Name: marios
  Hobby: workout
Name: alexander
  Hobby: watch_tv
Name: elisabeth
  Hobby: reading
Name: jason
  Hobby: chess
  Hobby: video_games
```

Όπως φαίνεται και στο ερώτημα που δημιουργείται, γίνεται απλό JOIN μεταξύ των πινάκων. Η generate() επιστρέφει μια λίστα αντικειμένων τύπου Person. Επειδή ο πίνακας Person έχει συσχέτιση με τον πίνακα Hobbies, περιέχει στην οντότητά του μια λίστα από αντικείμενα Hobbies, η οποία συμπληρώνεται μέσα στην generate().

SELECT με διαδοχικά φωλιασμένα JOINS

Σε αυτό το παράδειγμα προσθέτουμε έναν ακόμα πίνακα, τον Type, ο οποίος περιέχει την κατηγορία στην οποία ανήκει το κάθε Hobby. Ο πίνακας Hobby λοιπόν έχει μια συσχέτιση με τον Type. Έχουμε λοιπόν μια αλυσίδα από ενώσεις πινάκων, ξεκινώντας από τον Person, πηγαίνοντας στον Hobby και στη συνέχεια από τον Hobby στον Type.

```
/*Select with a nested join association*/
private void sELECTwithNestedJOIN() throws IllegalArgumentException, IllegalAccessException{
		List<Class<?>> rs = InitializeMetaData.initializeFromPackage("meta");

		QueryGET<_Person> queryGET = TablesFactory.eINSTANCE.createQueryGET(_Person.class);

		queryGET.setRootAlias("people");
		queryGET.getSelectedFields().add(_Person.name);

		JOIN<_Person,_Hobby,Integer> join = TablesFactory.eINSTANCE.createJOIN(_Person.class, _Hobby.class);
		join.setJoinedAlias("hobby");
		join.setOnFieldInitial(_Person.hobby_id);
		join.setOnFieldJoined(_Hobby.id);
		join.getJoinedTableFields().add(_Hobby.name);

		queryGET.getJoin().add(join);

		JOIN<_Hobby,_Type,Integer> nestedjoin = TablesFactory.eINSTANCE.createJOIN(_Hobby.class, _Type.class);
		nestedjoin.setJoinedAlias("type");
		nestedjoin.setOnFieldInitial(_Hobby.type_id);
		nestedjoin.setOnFieldJoined(_Type.id);
		nestedjoin.getJoinedTableFields().add(_Type.name);
		join.getJoinMore().add(nestedjoin);
```

```
        List<?> returned = queryGET.generate(Person.class);

        System.out.println("Final results");
        System.out.println("=========================");
        for(Person prs: (List<Person>) returned){
                System.out.println("Name: "+prs.getName());
                for(Hobby hobs: prs.getHobbies()){
                        System.out.println("   Hobby: "+hobs.getName());
                        for(sqlmodel_v2.entitiesTest.Type typ:hobs.getType()){
                                System.out.println("     Type: "+typ.getName());
                        }
                }
        }
}
```

Η αντίστοιχη έξοδος είναι η εξής:

```
Query Generated: SELECT  people.name AS _name692342133 ,hobby.name AS _name578866604 ,type.name AS _name353842779 FROM Person AS people JOIN Hobby AS hobby ON people.hobby_id=hobby.id JOIN Type AS type ON Hobby.type_id=type.id
Final results
=========================
Name: marios
  Hobby: workout
    Type: fittness
Name: alexander
  Hobby: watch_tv
    Type: leisure
Name: elisabeth
  Hobby: reading
    Type: mind
Name: jason
  Hobby: chess
    Type: game
  Hobby: video_games
    Type: game
```

SELECT, JOIN και WHERE

Τέλος θα δώσουμε ένα παράδειγμα που να συνδυάζει όλα τα παραπάνω:

```
private void test1() throws IllegalArgumentException, IllegalAccessException{
        List<Class<?>> rs = InitializeMetaData.initializeFromPackage("meta");

        QueryGET<_Person> qg2 = TablesFactory.eINSTANCE.createQueryGET(_Person.class);

        qg2.setRootAlias("people");
        qg2.getSelectedFields().add(_Person.name);

        JOIN<_Person,_Hobby,Integer> join = TablesFactory.eINSTANCE.createJOIN(_Person.class, _Hobby.class);
        join.setJoinedAlias("hobby");
        join.setOnFieldInitial(_Person.hobby_id);
        join.setOnFieldJoined(_Hobby.id);
```

```
            join.getJoinedTableFields().add(_Hobby.name);
            qg2.getJoin().add(join);

            JOIN<_Hobby,_Type,Integer> join2 =    TablesFactory.eINSTANCE.createJOIN(_Hobby.class, _Type.class);
            join2.setJoinedAlias("type");
            join2.setOnFieldInitial(_Hobby.type_id);
            join2.setOnFieldJoined(_Type.id);
            join2.getJoinedTableFields().add(_Type.name);
            join.getJoinMore().add(join2);

            //create some filters
            Filters<_Person> whereroot =
TablesFactory.eINSTANCE.createFilters(_Person.name,Operators.EQUALS,"jason",null);
            Filters<_Person> whereeli =
TablesFactory.eINSTANCE.createFilters(_Person.name,Operators.EQUALS,"elisabeth",null);
            Filters<_Person> wheremarios =
TablesFactory.eINSTANCE.createFilters(_Person.name,Operators.EQUALS,"marios",null);
            Filters<_Hobby> wherejasonhb = TablesFactory.eINSTANCE.createFilters(_Hobby.name, Operators.EQUALS,
"chess",join);

            whereroot.and(wherejasonhb).or(whereeli).or(wheremarios);

            qg2.setFilter(whereroot);

            List<?> returned = qg2.generate(Person.class);

            System.out.println("Final results");
            System.out.println("==========================");
            for(Person prs: (List<Person>) returned){
                        System.out.println("Name: "+prs.getName());
                        for(Hobby hobs: prs.getHobbies()){
                                    System.out.println("   Hobby: "+hobs.getName());
                                    for(sqlmodel_v2.entitiesTest.Type typ:hobs.getType()){
                                                System.out.println("     Type: "+typ.getName());
                                    }
                        }
                        System.out.println("------------------------");
            }
}
```

Και η έξοδος είναι η παρακάτω:

```
Query Generated: SELECT  people.name AS _name353842779 ,hobby.name AS _name1338823963
,type.name AS _name1156060786 FROM Person AS people JOIN Hobby AS hobby ON
people.hobby_id=hobby.id JOIN Type AS type ON Hobby.type_id=type.id WHERE (people.name = 'jason' AND
(hobby.name = 'chess')  OR (people.name = 'elisabeth')  OR (people.name = 'marios') )
Final results
==========================
Name: marios
  Hobby: workout
    Type: fittness
------------------------
Name: elisabeth
  Hobby: reading
    Type: mind
------------------------
Name: jason
```

```
  Hobby: chess
    Type: game
------------------------
```

Παρατηρήσεις

- Στα παραπάνω παραδείγματα δεν ορίστηκαν ψευδώνυμα για τα πεδία. Η επέκταση θέτει τα δικά της ψευδώνυμα αυτόματα μέσω της αρχικοποίησης των μεταδεδομένων. Ο χρήστης έχει τη δυνατότητα να τα μεταβάλει. Όμως επειδή η επέκταση επιστρέφει αντικείμενα που δε βασίζονται σε ψευδώνυμα κάτι τέτοιο είναι περιττό.

Με τα παραπάνω παραδείγματα δίνεται η βασική ιδέα της λειτουργίας την επέκτασης αυτής και καλύπτει τα πιο σημαντικά χαρακτηριστικά της.

ΚΕΦΑΛΑΙΟ 5

Ζητήματα Υλοποίησης

5.1 Περιορισμοί Εφαρμογής

Σε αυτή την έκδοση της εφαρμογής έχουν ήδη υλοποιηθεί σημαντικές λειτουργίες που αφορούν την δημιουργία ερωτημάτων. Ωστόσο υπάρχουν ακόμα θέματα που δεν έχουν λυθεί.
Βασικός περιορισμός είναι η έλλειψη υποστήριξης των ερωτημάτων ενημερόσεων CREATE, UPDATE, DELETE. Ακόμα δεν υποστηρίζεται η επιλογή του είδους συνένωσης που εφαρμόζεται. Για την ώρα υποστηρίζεται το απλό join. Τέλος πρέπει να προστεθούν και άλλες λειτουργίες της SQL, όπως για παράδειγμα η δυνατότητα εύρεσης μέσης τιμής και πλήθους εγγραφών.
Σε θέματα υλοποίησης υποστηρίζεται η αλυσιδωτή εισαγωγή φωλιασμένων joins. Αυτό επιτυγχάνεται με ένα δέντρο πινάκων. Σε αυτό το δέντρο δεν υποστηρίζονατι κύκλοι όμως. Αν για κάποιο λόγο υπάρξει κύκλος στο δέντρο τότε θα οδηγηθεί η εφαρμογή σε ατέρμονο loop. Αυτό το ενδεχόμενο είναι σπάνιο όμως, αφού ένας πίνακας δεν ανήκει αποκλειστικά σε ένα αντικείμενο JOIN ή στο queryGET και το δέντρο περιέχει τα αντικείμενα JOIN και όχι τους πίνακες αυτούς καθ' αυτούς. Ωστόσο πρέπει να αναφερθεί το ενδεχόμενο αυτό. Εκτως των άλλων πρέπει να αναφερθεί πως ο χρήστης οφείλει να ορίσει τα ψευδώνυμα στους πίνακές του, αν το κρίνει απαραίτητο.
Η εφαρμογή λειτουργεί καλά για μικρό πλήθος πινάκων. Αν προστεθούν πολλοί πίνακες μέσω joins, και δεδομένου πως ένα απλό join υπολογίζει το καρτεσιανό γινόμενο των πινάκων, μπορεί να υπάρξει πρόβλημα λόγο πολύ μεγάλου όγκου δεδομένων. Ακόμα χρησιμοποιούνται hashMaps εκτενώς που θεωρητικά προσφέρουν ταχύτητα στην ανάκτηση δεδομένων, αλλά κοστίζουν πολύ σε μνήμη.
Τέλος, όσον αφορά τις σχεσιακές γλώσσες υποστηρίζεται μόνο η SQL. Απαιτείται επέκταση της εφαρμογής για υποστήριξη περισσότερων σχεσιακών γλωσσών. Αυτό αν και σημαντικό βήμα για την επέκταση της εφαρμογής, πρέπει να είναι ένα από τα τελευταία θέματα που πρέπει να επιλυθούν γιατί είναι πιο σημαντική η σταθερότητα της εφαρμογής, έστω και στην SQL μόνο και στη συνέχεια η επέκτασή της και σε άλλες σχεσιακές γλώσσες.

5.2 Μελλοντικές Υλοποιήσεις

Παρακάτω παρουσιάζονται προτάσεις για μελλοντική ανάπτυξη λογισμικού πάνω σε αυτή την εφαρμογή.

1. Επέκταση της εφαρμογής προκειμένου να υποστηρίζει και ερωτήματα τύπου CREATE, UPDATE και DELETE.
2. Προσθήκη επιπλέων λειτουργιών που υποστηρίζει η SQL, όπως για παράδειγμα τη δυνατότητα να μετράει (COUNT) τις εγγραφές, να βρίσκει το μέσο όρο τιμών (AVERAGE) να υποστηρίζει ένωση πινάκων (UNION) και να δίνεται επιλογή στον χρήστη αν θέλει μοναδικές εγγραφές (DISTINCT).
3. Επέκταση της εφαρμογής για υποστήριξη επιπλέων γλωσσών διαχείρισης σχεσιακών βάσεων δεδομένων.
4. Βελτιστοποίηση κώδικα για επιπλέον απλοποίηση και αυτοματισμό στη διαδικασία δημιουργίας ερωτημάτων.

5. Επανεγγραφής κώδικα σε σημεία που κρίνεται αναγκαίο, για επίτευξη καλύτερης πολυπλοκότητας και επίδοσης συστήματος.

ΚΕΦΑΛΑΙΟ 6

Βιβλιογραφικές Αναφορές

[1] Bowersox, K. (2014, May 17). *To Thought What is JPA? How does it differ from Hibernate?* Ανάκτηση από blog-tothought: https://blog-tothought.rhcloud.com//post/2

[2] Bracha, G. (February 13,2004). *Generics in the Java Programming Language.*

[3] David Erni, A. K. (March 2008). The Hacker's Guide to Javac.

[4] Frank Budinsky, D. S. (August 11, 2003). *Eclipse Modeling Framework: A Developer's Guide.* Addison Wesley.

[5] Ira R. Forman, N. F. (n.d.). *Java Reflection In Action.* Manning Publications Co.

[6] Ltd, A. (2003, 2004). *JET Tutorial Part 1 (Introduction to JET)*. Ανάκτηση από eclipse: https://eclipse.org/articles/Article-JET/jet_tutorial1.html

[7] ObjectDB. (n.d.). *Using the Java Persistence API (JPA)*. Ανάκτηση από objectdb: http://www.objectdb.com/java/jpa/persistence

[8] Oracle Corporation and/or its affiliates. (2010). *Chapter 22 Creating Queries Using The Criteria API (The Java EE 6 Tutorial, Volume I)*. Ανάκτηση από docs.oracle: https://docs.oracle.com/cd/E19226-01/820-7627/gjitv/index.html

[9] Oracle Corporation and/or its affiliates. (2013, January). *The Java EE 6 Tutorial.* Ανάκτηση από docs.oracle: http://docs.oracle.com/javaee/6/tutorial/doc/

[10] R. Elmasri, S. N. (n.d.). *Θεμελειώσεις Αρχές Συστημάτων Βάσεων Δεδομένων* (6 εκδ.). (Μ. Χατζόπουλος, Μεταφρ.)

[11] Silberschatz, K. S. (n.d.). *Συστήματα Βάσεων Δεδομένων Η Πλήρης Θεωρία των Βάσεων Δεδομένων* (4 εκδ.).

[12] The Hibernate Team, T. J. (2016, 1 13). *Hibernate User Guide.* Ανάκτηση από docs.jboss: http://docs.jboss.org/hibernate/orm/5.0/userGuide/en-US/html_single/

[13] Timo Westkämper, S. S. (2007-2015). *Querydsl Reference Guide.* Ανάκτηση από querydsl: http://www.querydsl.com/static/querydsl/4.0.4/reference/html_single/

[14] tutorialspoint. (2016). *Hibernate Query Language*. Ανάκτηση από tutorialspoint: http://www.tutorialspoint.com/hibernate/hibernate_query_language.htm

[15] tutorialspoint. (2016). *JPA Tutorial*. Ανάκτηση από tutorialspoint: http://www.tutorialspoint.com/jpa/index.htm