

**ΣΥΣΤΗΜΑ ΜΑΖΙΚΗΣ ΕΞΑΓΩΓΗΣ ΤΗΣ ΕΞΕΛΙΞΗΣ
ΣΧΗΜΑΤΩΝ ΣΧΕΣΙΑΚΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ**

Κωστούδας Σάββας

Διπλωματική Εργασία

Επιβλέπων: Π. Βασιλειάδης

Ιωάννινα, Ιούλιος 2020



**ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA**

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την οικογένεια μου η οποία με στήριζε σε όλη την διάρκεια των σπουδών μου και με βοήθησε να πραγματοποιήσω τα όνειρα μου. Στους καθηγητές μου που μου πρόσφεραν απλόχερα όλες τις απαραίτητες γνώσεις και τα εφόδια για το μέλλον. Στον κ. Βασιλειάδη που με την καθοδήγηση του δημιουργήθηκε η παρούσα Διπλωματική Εργασία.

Kostoudas Savvas

Department of Computer Science and Engineering

University of Ioannina, Greece

June 2020

System for the Mass Export of the Evolution of Databases Relational Objects

Supervisor: Panos Vassiliadis

Περίληψη στα ελληνικά

Όπως σε όλα τα συστήματα λογισμικού έτσι και οι βάσεις δεδομένων εξελίσσονται με το πέρασμα του χρόνου. Ο αντίκτυπος αυτής της εξέλιξης είναι καίριας σημασίας καθώς κάθε αλλαγή στο σχήμα μιας βάσης δεδομένων επηρεάζει την σημασιολογική εγκυρότητα όλων των εφαρμογών που την περιβάλλουν και απαιτούν εκ νέου συντήρηση προκειμένου να αρθούν τα λάθη από τον πηγαίο τους κώδικα. Σ' αυτή την πτυχιακή εργασία διεξάγουμε μια εμπεριστατωμένη, οργανωμένη και ελεγχόμενη μελέτη της εξέλιξης των σχημάτων βάσεων δεδομένων. Για τον σκοπό αυτό θα χρησιμοποιηθεί και θα επεκταθεί το εργαλείο EKATH το οποίο εκτιμά με μεγάλη ακρίβεια την εξέλιξη των πινάκων ενός σχήματος, τόσο σε γενικό και υψηλό επίπεδο αφαίρεσης δηλαδή οποιαδήποτε προσθήκη ή καταστροφή πινάκων, όσο και σε χαμηλό επίπεδο αφαίρεσης με την δημιουργία ή διαγραφή πεδίων των πινάκων. Παρόλα αυτά το εργαλείο αυτό παρουσιάζει συγκεκριμένες ελλείψεις τις οποίες προοδευτικά θα γίνει προσπάθεια κάλυψης τους στο πλαίσιο αυτής της πτυχιακής εργασίας.

Λέξεις Κλειδιά: εξέλιξη βάσεων δεδομένων, σχήματα βάσεων

Abstract

Like all software systems, databases are subject to evolution as time passes. The impact of this evolution is critical as every change to the schema database affects the syntactic correctness and the semantic validity of all the surrounding applications and as a result necessitates their maintenance in order to remove errors from their source code. In this Diploma Thesis a thorough, organized and controlled study is performed about the evolution of schemas of databases. In order to accomplish the above goal a tool by the name HECATE will be used and extended which can evaluate with high precision the evolution of the tables of a schema. This evaluation can happen in a higher and more abstract level such as the addition or the destruction of tables but also can happen in a lower and less abstract level such as the creation or the deletion of fields of a table. However, this useful tool has some shortcomings which will be progressively addressed in this Diploma thesis.

Keywords: databases evolution, database schema

Πίνακας περιεχομένων

Κεφάλαιο 1. Εισαγωγή.....	1
1.1 Αντικείμενο της διπλωματικής.....	1
1.2 Οργάνωση του τόμου	3
Κεφάλαιο 2. Περιγραφή Θέματος.....	5
2.1 Στόχος της εργασίας.....	5
2.2 Υπόβαθρο	6
2.2.1 Εργασίες Σχετικές με την Εξέλιξη Βάσεων Δεδομένων	6
2.2.2 Το Εργαλείο Εκάτη (Hecate)	7
2.2.3 Παραγωγή Μεταφραστών	8
Κεφάλαιο 3. Σχεδίαση & Υλοποίηση.....	14
3.1 Ορισμός προβλήματος και αλγόριθμοι επίλυσης	14
3.2 Σχεδίαση και αρχιτεκτονική λογισμικού.....	16
3.3 Σχεδίαση και αποτελέσματα ελέγχου του λογισμικού.....	24
3.4 Λεπτομέρειες εγκατάστασης και υλοποίησης	25
3.5 Επεκτασιμότητα του λογισμικού.....	25
Κεφάλαιο 4. Εργαλείο Insight.....	27
4.1 Το εργαλείο Insight	27
4.2 Σχεδίαση & Υλοποίηση	32
4.2.1 Αρχιτεκτονική πακέτων	32
4.2.2 Σχεδίαση κλάσεων και λειτουργιών	34
4.3 Εγχειρίδιο ορθής χρήσης (Insight Manual)	61
4.3.1 Κατανόηση των πληροφοριών	61
4.3.2 Φόρτωση log file	62
4.3.3 Επεξεργασία αρχείου sql	63
4.3.4 Αποθήκευση αρχείου sql.....	66
4.3.5 Λειτουργία Αναζήτησης (Find/Replace)	67
4.3.6 Παρεχόμενες λειτουργίες	68

4.3.7 Εκτύπωση αρχείου και διαθέσιμες επιλογές	69
4.3.8 Παρουσίαση αποτελεσμάτων αναζήτησης	70
Κεφάλαιο 5. Πειραματική αξιολόγηση	74
5.1 Μεθοδολογία Πειραματισμού	74
5.2 Αναλυτική παρουσίαση αποτελεσμάτων	75
Κεφάλαιο 6. Επίλογος	78
6.1 Σύνοψη και αποτελέσματα	78
6.2 Μελλοντικές επεκτάσεις	79

Κεφάλαιο 1. Εισαγωγή

Μια βάση δεδομένων αποτελεί μια συλλογή δεδομένων η οποία χαρακτηρίζεται από στοιχειώδη οργάνωση, ώστε να βοηθήσει στην μετέπειτα επερώτηση και αναζήτηση των δεδομένων από τις διάφορες εφαρμογές που στηρίζονται στην βάση αυτή. Όσο η πολυπλοκότητα των εφαρμογών αυξάνεται λόγω των ολοένα και περισσότερων προσφερόμενων λειτουργιών, τόσο και το μέγεθος της βάσης δεδομένων αυξάνεται παράλληλα. Ως αποτέλεσμα, η αύξηση μεγέθους επιφέρει μεταβολές στην δομή της βάσης, από τις οποίες μπορούν να προκύψουν ουσιώδη συμπεράσματα και παρατηρήσεις τα οποία μπορούν να χρησιμοποιηθούν για την εξομάλυνση των δυσκολιών που παρουσιάζονται κατά την συντήρηση της.

1.1 Αντικείμενο της διπλωματικής

Η βάση δεδομένων όντας στον πυρήνα των περισσότερων σύγχρονων εφαρμογών και συστημάτων, υπόκειται σε εξέλιξη προκειμένου να υποστηρίξει τις εφαρμογές αυτές. Ωστόσο, κάθε μεταβολή στο σχήμα μια βάσης δεδομένων μπορεί να επιφέρει πληθώρα επιπτώσεων στις υποστηριζόμενες εφαρμογές. Συγκεκριμένα, μια αφαίρεση πεδίου είναι πιθανό να προκαλέσει ολική αποτυχία ενός ερωτήματος (query) το οποίο μεταχειρίζεται την εν λόγω συσχέτιση, και ως αποτέλεσμα να οδηγήσει σε δυσεπίλυτο σφάλμα συντακτικής φύσεως.

Από την άλλη πλευρά, μια προσθήκη ενός πεδίου είναι πιθανό να περιέχει πληροφορία ζωτικής σημασίας η οποία όμως μπορεί να παραληφθεί από εφαρμογές που απλά δεν έχουν ενημερωθεί ότι εισήχθη η πληροφορία αυτή, προκαλώντας έτσι σφάλμα σημασιολογικής εγκυρότητας. Επομένως, όπως παρατηρείτε, οποιαδήποτε μεταβολή ανεξαρτήτου μεγέθους ή τύπου στο λογικό σχήμα της βάσης μπορεί να οδηγήσει στο

φαινόμενο “χιονοστιβάδας” καθώς από μία πολύ μικρή αιτία θα οδηγηθούμε σε κατάρρευση της εφαρμογής.

Το αντικείμενο αυτής της διπλωματικής είναι η υποβοήθηση της διαδικασίας αυτόματης εξαγωγής της ιστορίας και των αλλαγών ενός σχήματος μιας βάσης δεδομένων. Για τον σκοπό αυτό επεκτάθηκε ένα υπάρχον εργαλείο και κατασκευάστηκε ένα νέο, με σκοπό την υποβοήθηση της εξαγωγής μιας ακριβέστερης ιστορίας του σχήματος της βάσης.

Το εργαλείο που επεκτάθηκε ονομάζεται Εκάτη [Skou_13] και ο σκοπός του είναι να λαμβάνει ως είσοδο μια λίστα με τα αρχεία των διαφορετικών εκδοχών του σχήματος της βάσης δεδομένων, όπως προέκυψαν στην πορεία του χρόνου, και να παράγει ως έξοδο (α) τη λίστα με τις αλλαγές μεταξύ γειτονικών στο χρόνο εκδοχών του σχήματος, και κατά συνέπεια την πλήρη λίστα αλλαγών στην ιστορία του σχήματος της βάσης, (β) μετρήσεις σχετικά με το πότε και πώς άλλαξε το σχήμα σε κάθε μετάβαση από μία εκδοχή στην επόμενη, (γ) μετρήσεις για την εξελικτική συμπεριφορά των πινάκων του σχήματος. Επειδή κάθε εκδοχή της ιστορίας έρχεται με την μορφή Data Definition Language (DDL) αρχείου σε SQL, το εργαλείο περνά από ένα parser κάθε τέτοιο αρχείο και εντοπίζει τις εντολές δημιουργίας πινάκων. Η διαδικασία αυτή έχει προφανώς σφάλματα επεξεργασίας (parsing errors). Η πρώτη συνεισφορά της Διπλωματικής αυτής είναι η οργανωμένη αντιμετώπιση των parsing errors της Εκάτης, με την σύλληψη εξαιρέσεων που προέκυπταν στον κώδικα, και την καταγραφή των λαθών σε ένα log file με τρόπο που στη συνέχεια να επιτρέπει να τα εκμεταλλευόμαστε.

Η δεύτερη συνεισφορά της Διπλωματικής αυτής είναι η σχεδίαση και υλοποίηση ενός εργαλείου απεικόνισης των παραγόμενων σφαλμάτων ώστε να μπορεί ο αναλυτής να προχωρήσει ακόμα ένα βήμα στην κατανόηση των σφαλμάτων αλλά και να τα διαχωρίσει σε πιο σημαντικά και σε λιγότερα σημαντικά λάθη όπως είναι τα συντακτικά λάθη. Το εργαλείο αυτό ονομάζεται Insight (Ενόραση). Το εργαλείο Insight επιτρέπει την ομαδοποίηση των σφαλμάτων (καθώς πολλά από αυτά επαναλαμβάνονται σε πολλές εκδοχές του DDL αρχείου) ώστε να μπορεί ο αναλυτής να καταλάβει τα «μοναδικά» σημεία αποτυχίας του parsing. Επιπρόσθετα, το εργαλείο αυτό προσφέρει την δυνατότητα τροποποίησης των αρχείων sql που συμμετέχουν στο σχήμα της βάσης προσφέροντας με αυτόν τον τρόπο παραλληλισμό των ενεργειών, παρατήρηση του σφάλματος και ταυτόχρονα αντιμετώπιση του.

1.2 Οργάνωση του τόμου

Η συγκεκριμένη διπλωματική εργασία αποτελείται από 6 κεφάλαια τα οποία θα αναλυθούν στις επόμενες ενότητες.

Στο κεφάλαιο 2 περιγράφονται σχετικές εργασίες οι οποίες αποτελούν σημείο αναφοράς στην εξέλιξη και εξαγωγή σχημάτων βάσεων δεδομένων όπως αναλύονται σε [LMR+97] αλλά και σε [Skou13] με την παρουσίαση του εργαλείου Εκάτη. Επιπρόσθετα, στο κεφάλαιο 2 αναλύεται η λειτουργία του εργαλείου ANTLR στο οποίο έχει στηριχθεί το εργαλείο Εκάτη όπως επίσης αναλύεται η παραγωγή των μεταφραστών.

Στο κεφάλαιο 3 αναλύεται η σχεδίαση και υλοποίηση του πρώτου στόχου της συγκεκριμένης διπλωματικής ο οποίος αφορά την παραγωγή κατάλληλων μηνυμάτων σφάλματος που βοηθούν τον προγραμματιστή να κατανοήσει την αιτία των λαθών. Επίσης, συμπεριλαμβάνονται αποτελέσματα ελέγχου της επέκτασης του εργαλείου και εμπεριστατωμένη μελέτη για μελλοντικές επεκτάσεις και συντηρήσεις.

Στο κεφάλαιο 4 αναλύεται η σχεδίαση και υλοποίηση του εργαλείου Insight. Επιπρόσθετα, δίνεται ο πλήρης οδηγός χρήσης του εργαλείου Insight (Ενόραση) ώστε να επιδείξει την σωστή χρήση του εργαλείου.

Στο κεφάλαιο 5 περιέχεται πειραματική αξιολόγηση των εργαλείων Εκάτη αλλά και του Insight και παρουσιάζονται αναλυτικά τα αποτελέσματα.

Τέλος, στο κεφάλαιο 6 παραθέτονται τα συμπεράσματα των πειραμάτων και περιγράφονται οι μελλοντικές επεκτάσεις του λογισμικού.

Κεφάλαιο 2. Περιγραφή Θέματος

2.1 Στόχος της εργασίας

Η εξαγωγή σχεσιακών σχημάτων βάσεων δεδομένων με αναπαραστατικό τρόπο πάντα βοηθούσε στην καλύτερη κατανόηση και ανάλυση των βάσεων δεδομένων. Στόχος αυτής της διπλωματικής αποτελεί η ανακατασκευή και επέκταση του υπάρχοντος εργαλείου που ονομάζεται Hecate. Το εργαλείο αυτό είναι ικανό να αναπαραστήσει γραφικά την δομή μιας βάσης δεδομένων δηλαδή τους πίνακες, τις σχέσεις και τα πεδία από τα οποία αποτελείται καθώς επίσης και την μεταβολή στον χρόνο που υπέστη αυτή η βάση. Το εργαλείο αυτό προσφέρει και χρωματική επίδειξη των μεταβολών αυτών καθώς για προσθήκη σχέσεων, πινάκων ή πεδίων χρησιμοποιείται πράσινο χρώμα, για διαγραφή αντίστοιχα χρησιμοποιείται κόκκινο χρώμα και τέλος για μετονομασία ή αλλαγές(εντολή ALTER) χρησιμοποιείται το κίτρινο χρώμα. Ωστόσο, σε ορισμένες περιπτώσεις χρήσης του εργαλείου παρατηρήθηκαν προβλήματα τα οποία αποτελούν στόχο επίλυσης από αυτή την διπλωματική εργασία.

Στο πλαίσιο αυτό, έγιναν ενέργειες για την επίλυση των προβλημάτων που αφορούν την SQL εντολή 'ALTER' όμως μόνο στην περίπτωση που περιέχει την προσθήκη ξένου κλειδιού (Foreign Key). Στην περίπτωση αυτή το εργαλείο Hecate δεν μετέβαινε σε έγκυρες καταστάσεις με συνέπεια να μην παράγει τα αναμενόμενα αποτελέσματα και σε ορισμένες περιπτώσεις το εργαλείο παρουσίασε πλήρη αποσυντονισμό της λειτουργίας του με αποτέλεσμα να μην περατώσει την απαιτούμενη διεργασία (CRASH). Το εργαλείο Hecate σε μια πληθώρα άλλων περιπτώσεων εξακολουθούσε να παράγει μηνύματα σφαλμάτων όμως αυτές οι περιπτώσεις δεν το ωθούν σε πλήρη αποσυντονισμό της λειτουργίας του (CRASH). Αυτές οι περιπτώσεις εντάσσονται στην κατηγορία των

προβλημάτων όπου στα SQL αρχεία, οι εντολές (CREATE, ALTER, DROP..) παράχθηκαν με τέτοιο τρόπο ώστε να δημιουργούν λεκτικά προβλήματα στο εργαλείο Hecate.

Όλα τα προαναφερθέντα προβλήματα αποτελούν αντικείμενα της εδραιωμένης στοχοθεσίας τα οποία και θα απαντηθούν σε αυτή την διπλωματική εργασία.

2.2 Υπόβαθρο

2.2.1 Εργασίες Σχετικές Με την Εξέλιξη Βάσεων Δεδομένων

Μία από τις πρώτες εκτενείς αναλύσεις με θέμα την μέτρηση της εξέλιξης των σχημάτων και της επιρροής τους στις εφαρμογές που στηρίζονταν σε αυτά τα σχήματα πραγματοποιήθηκε από τον Sjoberg D. Ο συντάκτης της δουλειάς αυτής δημιούργησε ένα εργαλείο [Sjøb91] το οποίο υλοποιήθηκε πάνω από ένα υπάρχον σύστημα διαχείρισης της υγείας (HMS) όπου χρησιμοποιούσε μια σχεσιακή βάση δεδομένων και παρακολουθούσε την εξέλιξη της για ένα χρονικό διάστημα 18 μηνών. Πιο αναλυτικά, στα αποτελέσματα της έρευνας αυτής [Sjøb93], παρουσιάστηκε ότι στην δομή των σχημάτων σημειώθηκε αύξηση των σχέσεων κατά 139% και ως αποτέλεσμα αύξηση των πεδίων των σχέσεων κατά 274%. Οι επιπτώσεις μιας τόσο σημαντικής εξέλιξης της βάσης δεδομένων όπως και ο συγγραφέας του άρθρου αναφέρει, επιβεβαιώνει την ανάγκη εργαλείων διαχείρισης αλλαγών (Change Management Tools), τα οποία να μπορούν να διαδίδουν την μεταβολή στο σχήμα της βάσης στα επηρεαζόμενα πεδία διατηρώντας παράλληλα το σύστημα ενεργό και διαθέσιμο για χρήση.

Επιπλέον μία περίπτωση εκτεταμένης μελέτης, μεγαλύτερης κλίμακας αποτελεί η μελέτη επονομαζόμενη ως “Analysis of Schema Evolution for Databases in Open-Source Software”, η οποία δημοσιεύθηκε το έτος 2014. Ο στόχος της μελέτης αυτής ήταν να εξεταστεί εάν οι νόμοι του Lehman [LMR+97] που αφορούν την εξέλιξη λογισμικού μπορούν να εφαρμοστούν και στα σχήματα βάσεων δεδομένων. Το σύνολο αποτελείται συνολικά από οκτώ νόμους. Πρόκειται για ένα άρτια εδραιωμένο σύνολο από παρατηρήσεις που αποδεικνύουν τον τρόπο με τον οποίο εξελίσσονται τα συστήματα λογισμικού. Για την εξαγωγή των αποτελεσμάτων δημιουργήθηκε και χρησιμοποιήθηκε το εργαλείο Εκάτη (Hecate) όπως παρουσιάζεται στο [Skou13]. Στα αποτελέσματα που ανακαλύφθηκαν συγκαταλέγονται οι εξής παρατηρήσεις:

- οι βάσεις δεδομένων δεν εξελίσσονται και προσαρμόζονται συνεχώς αλλά αντιθέτως οι μεταλλάξεις λαμβάνουν χώρα σε συγκεκριμένα χρονικά διαστήματα.
- Όσον αφορά το μέγεθος του σχήματος βάσης δεδομένων αρχικά παρατηρείται ραγδαία αύξηση κυρίως στην αρχή ή μετά από μεγάλες πτώσεις του μεγέθους ενώ κατά κύριο λόγο παρατηρείται σταθερότητα στην τιμή του μεγέθους.
- Οι αλλαγές συχνά ακολουθούν το spike pattern δηλαδή συμβαίνουν μικρές αλλαγές σε μικρό χρονικό διάστημα και μετά δεν συμβαίνει καμία αλλαγή για ένα μεγάλο χρονικό διάστημα.

Τέλος, οι συγγραφείς της μελέτης αυτής καταλήγουν στο συμπέρασμα ότι η εξέλιξη των σχημάτων απεικονίζει την συμπεριφορά ενός συστήματος με ρυθμιζόμενη ανάδραση (Feedback-Regulated System), αφού υπακούει στον ανταγωνισμό μεταξύ της ανάγκης για επέκταση της πληροφοριακής χωρητικότητας ώστε να ικανοποιήσει τις ανάγκες των χρηστών, και της ανάγκης του ελέγχου της αταξινόμητης επέκτασης ώστε να απαιτείται όσο το δυνατόν λιγότερη συντήρηση του συστήματος.

2.2.2 Το Εργαλείο Εκάτη (Hecate)

Για την εξαγωγή των αποτελεσμάτων δημιουργήθηκε και χρησιμοποιήθηκε το εργαλείο Εκάτη (Hecate) όπως παρουσιάζεται στο [Skou13]. Το εργαλείο Εκάτη είναι ένα ανοικτό (open source) λογισμικό το οποίο αποδέχεται ως είσοδο αρχεία DDL τα οποία ουσιαστικά αποτελούν το ιστορικό (Log File) της βάσης, που χρονολογείται από την στιγμή της δημιουργίας της έως και την εξαγωγή του συγκεκριμένου αρχείου. Σε αυτά τα αρχεία καταγράφονται όλες οι μεταβολές στην υπό εξέταση βάση και ως αποτέλεσμα περιέχει όλες τις εκδόσεις της. Η κύρια λειτουργία του εργαλείου αυτού έγκειται στην σύγκριση και εξαγωγή διαφορών ανάμεσα στις δοθέντες εκδόσεις της βάσης. Προσφέρει πλούσια αναπαράσταση των αποτελεσμάτων με γραφικό τρόπο χρησιμοποιώντας διαθέσιμα χρώματα από την συλλογή της, όπως άλλωστε έχει αναφερθεί προηγουμένως. Ανάμεσα στις ικανότητες του εργαλείου, συγκαταλέγεται και η εξαγωγή βοηθητικών μετρικών όπως για παράδειγμα ο αριθμός των πεδίων που προστέθηκαν, διαγράφηκαν ή υπέστησαν μεταβολές. Ακόμα ανάμεσα στις μετρικές αυτές παρατηρούνται αναλυτικά για κάθε σχέση (πίνακα) ο αριθμός των πεδίων που υπέστησαν κάποια μεταβολή, οι ρυθμοί παραγωγής διαγραφής ή μεταβολής των πεδίων στο πέρασμα του χρόνου. Συγκεκριμένα, οι αλλαγές τις οποίες αντιλαμβάνεται το εργαλείο Εκάτη είναι οι ακόλουθες:

- Δημιουργία Πινάκων
- Διαγραφή Πινάκων
- Εισαγωγές Πεδίων
- Διαγραφές Πεδίων
- Αλλαγές πρωτευόντων Κλειδιών (Primary Keys)
- Μεταβολές στο πλήθος πεδίων ενός πίνακα
- Αλλαγές τύπων πεδίων

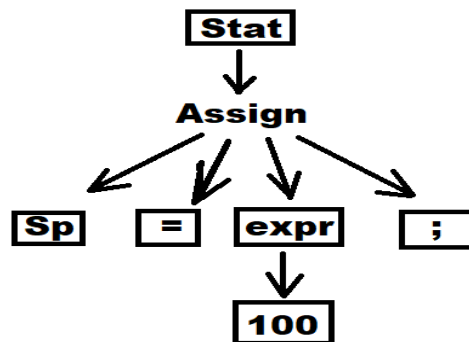
Τα datasets που συγκεντρώθηκαν και επεξεργάστηκαν από την Εκάτη για την εξαγωγή αποτελεσμάτων αντλήθηκαν από Συστήματα Διαχείρισης Περιεχομένου (CMS's), από online καταστήματα καθώς και επιστημονικές αποθήκες δεδομένων.

2.2.3 Παραγωγή Μεταφραστών

Σημαντικός παράγοντας στην δημιουργία του εργαλείου Εκάτη (Hecate) αποτέλεσε το εργαλείο ANTLR όπως περιγράφεται στο [Parr20] το οποίο δημιουργήθηκε από τον Terence Parr. Το ANTLR είναι ένα ισχυρό εργαλείο παραγωγής μεταφραστών (parsers), οι οποίοι χρησιμοποιούνται για ανάγνωση, επεξεργασία και εκτέλεση δομημένων αρχείων κειμένου ή δυαδικών αρχείων. Προμηθεύοντας μια γραμματική, το εργαλείο ANTLR όπως περιγράφεται και στο [GTom20] έχει την δυνατότητα να παράγει ένα μεταφραστή ο οποίος μπορεί να κατασκευάσει και να εξερευνήσει τα δέντρα που αντιστοιχούν στους κανόνες της προμηθευόμενης γραμματικής. Ως παραγόμενο δέντρο το οποίο ονομάζεται και AST (Abstract Syntax Tree), εννοείται μια αναπαράσταση της εισόδου η οποία έχει αναλυθεί σε tokens, δηλαδή στα συστατικά από τα οποία παράγεται. Η ιδέα αυτή παρουσιάζεται στο σχήμα 1:

Σχήμα 1. Αναπαράσταση Έκφρασης (Sp = 100;) σε AST Δέντρο [AntlrPTut]

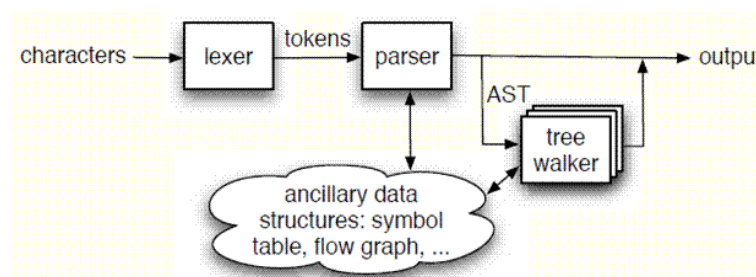
AST Tree for expression: Sp=100;



Στα φύλλα (τερματικοί κόμβοι) βρίσκονται τα tokens ενώ οι ενδιάμεσοι κόμβοι αποτελούν τους κανόνες της γλώσσας που δημιουργείται με χρήση του εργαλείου ANTLR.

Όσον αφορά το ANTLR, στα πλεονεκτήματα του συγκαταλέγεται η έννοια της παραγωγικότητας, δηλαδή δοθέντος μιας γραμματικής μπορεί να δημιουργήσει τους Parser, Letic Analyst (Λεκτικός Αναλυτής) και φυσικά τους Listener και Visitor οι οποίοι καθιστούν δυνατή την Ανάλυση, Αναζήτηση και Επεξεργασία των AST δέντρων που αντιστοιχούν στους κανόνες της γλώσσας. Επιπρόσθετα, τα παραχθέντα αυτά εργαλεία είναι ανοικτά σε μεταβολές που είτε αφορούν την ταχύτητα εκτέλεσης, είτε την παραγωγή πιο αναλυτικών και ειδικών μηνυμάτων σφάλματος, ώστε να ανταποκριθούν στις ανάγκες των χρηστών. Η διαδικασία χρήσης μεταγλώττισης και παραγωγής AST δέντρων χρησιμοποιώντας τα παραχθέντα εργαλεία παρουσιάζεται στο σχήμα 2:

Σχήμα 2. Input/Output της διαδικασίας μεταγλώττισης με χρήση εργαλείου ANTLR [Antlr_IO]



Ειδικότερα, η διαδικασία χρήσης του εργαλείου ANTLR επιτυγχάνεται χρησιμοποιώντας την ακόλουθη εντολή:

java -jar antlr-4.8-complete.jar DDL.g4

Στην σύνταξη της παραπάνω εντολής εντοπίζεται και το αρχείο DDL.g4. Στο αρχείο αυτό έχει συνταχθεί η γραμματική της γλώσσας που με βάση αυτή θα μεταγλωττιστούν τα αρχεία που δίνονται ως είσοδο στο εργαλείο Hecate. Στην γραμματική αυτή παρατάσσονται όλοι οι κανόνες της γλώσσας (rules), οι οποίοι θα χρησιμοποιηθούν από τον Parser προκειμένου να αναγνωριστούν οι ενέργειες του αρχείου εισόδου με τελικό σκοπό την παραγωγή των AST δέντρων. Στην παρακάτω αναπαράσταση σκιαγραφείται ένας από τους κανόνες της γλώσσας όπως ακριβώς χρησιμοποιείται από τον Parser:

Κανόνας της DDL γλώσσας που χρησιμοποιείται σε εργαλείο Hecate

alter_statement: ALTER TABLE table_name (ADD alter_constraint)+;

Η μετάφραση του παραπάνω κανόνα αναλύεται ως εξής: από τον Letic Analyst θα παραχθούν τα tokens ALTER και TABLE, επομένως ο Parser με την σειρά του θα μεταβεί στην κλάση ALTER_statementContext ώστε να προχωρήσει στην μετάφραση του κανόνα. Έπειτα από τον Letic Analyst θα παραχθεί το table_name το οποίο αφορά μια αλφαριθμητική αναπαράσταση του ονόματος του πίνακα που θα υποστεί μεταβολές. Ταυτόχρονα, ο Parser θα μεταφράσει αυτή την χρήσιμη πληροφορία. Με την ίδια διαδικασία θα παραχθεί το token ADD όπου είναι απαραίτητο σε κάθε μεταβολή του πίνακα το οποίο ακολουθείται από τον φωλιασμένο κανόνα alter_constraint ώστε να μεταφράσει ο Parser το περιεχόμενο του κανόνα. Το σύμβολο (+) στο τέλος του κανόνα αποτελεί ένα σύμβολο της γραμματικής και όχι κάποιο token της γλώσσας το οποίο μας προσδιορίζει ότι τα περιεχόμενα της παρένθεσης ενδέχεται να βρεθούν τουλάχιστον μία φορά μέσα σε αυτόν τον κανόνα.

Ως αποτέλεσμα της παραπάνω εντολής παράγονται επιπρόσθετα τα εξής αρχεία: DDLBaseListener.java, DDLBaseVisitor.java, DDLLexer.java, DDLParser.java καθώς και αρχεία που περιέχουν tokens δηλαδή τα DDL.tokens και DDLLexer.tokens. Στα αρχεία με κατάληξη tokens βρίσκονται όλα τα tokens που εντοπίστηκαν μέσα στην δοθείσα γραμματική. Επίσης για την εύκολη χρήση μέσα στα παραχθέντα .java αρχεία έχει αποδοθεί μια ακέραια τιμή στο καθένα token, που λειτουργεί ως αναγνωριστικός αριθμός (Id) έτσι ώστε να αναγνωρίζονται με ευκολία κατά τη διάρκεια της διάσχισης του AST δέντρου από τον μεταφραστή (parser).

Στην προαναφερθείσα λίστα αρχείων τα οποία παράγονται με την χρήση του εργαλείου ANTLR τα αρχεία DDLParser.java, DDLLexer.java και τα αρχεία που περιέχουν τα tokens

δεν τα μεταβάλλουμε μετά από τη δημιουργία τους. Αυτό συμβαίνει καθώς έχουν δημιουργηθεί για την συγκεκριμένη γραμματική που προμηθεύσαμε ως είσοδο. Αν επιθυμούμε κάποια αλλαγή στα παραπάνω αρχεία απαιτείται εκ νέου χρήση της εντολής ώστε να παράγουμε για ακόμα μία φορά τα προαναφερθέντα αρχεία. Στην απέναντι όχθη, τα αρχεία `DDLBaseListener.java`, `DDLBaseVisitor.java` είναι δυνατό να μεταβληθούνε όπως επιθυμεί ο διαχειριστής όμως για χάρι της ευκρίνειας και μειωμένης πολυπλοκότητας συνίσταται η επέκταση των κλάσεων αυτών από νέες κλάσεις έτσι ώστε οποιαδήποτε προσθήκη ή αλλαγή να βρίσκεται σε μία και μόνο κλάση.

Έπειτα, το αρχείο `DDLParser` περιέχει τον λεκτικό αναλυτή του οποίου η λειτουργία είναι να διαχωρίζει την είσοδο (input) σε tokens και ταυτόχρονα να ελέγχει ότι τα tokens αυτά είναι αποδεκτά από την ορισθείσα γραμματική. Προχωρώντας, το αρχείο `DDLParser` περιέχει τις μεθόδους που είναι υπεύθυνες για την συντακτική και σημασιολογική μετάφραση του δοθέντος αρχείου. Επιπρόσθετα, είναι υπεύθυνο για τη παραγωγή του AST δέντρου, το οποίο δέντρο θα διασχιστεί είτε από το αρχείο `DDLListener` είτε από `DDLBaseVisitor`. Η διάσχιση του δέντρου και στις δύο περιπτώσεις πραγματοποιείται χρησιμοποιώντας τον αλγόριθμο της αναζήτησης πρώτα σε βάθος όπως φαίνεται στο σχήμα 3:

Σχήμα 3. Αλγόριθμος Αναζήτησης Πρώτα Σε Βάθος(DFS)

```
1 procedure DFS(G,v):
2   label v as explored
3   for all edges e in G.adjacentEdges(v) do
4     if edge e is unexplored then
5       w ← G.adjacentVertex(v,e)
6       if vertex w is unexplored then
7         label e as a discovery edge
8         recursively call DFS(G,w)
9     else
10      label e as a back edge
```

Δοθέντος δηλαδή, ενός γραφήματος G και ενός αρχικού κόμβου (ρίζα v) ο αλγόριθμος διασχίζει το AST δέντρο εξερευνώντας πρώτα τους κόμβους που βρίσκονται στο μεγαλύτερο βάθος στο δέντρο και μετά συνεχίζει με τους υπόλοιπους κόμβους. Επομένως, τα αρχεία `DDLBaseListener` και `DDLBaseVisitor` χρησιμοποιούνται για την εξερεύνηση του δέντρου που δημιουργείται από τους κανόνες της γλώσσας. Η ειδοποιός διαφορά μεταξύ των δύο αυτών αρχείων είναι ότι ο `DDLBaseVisitor` μπορεί να ελέγξει τον τρόπο με τον οποίο εισέρχονται οι κόμβοι στο AST δέντρο ή να συλλέξει πληροφορίες από έναν μεγάλο αριθμό αυτών των κόμβων. Οι παραπάνω ενέργειες δεν είναι διαθέσιμες στον `DDLBaseListener`.

Στο αρχείο DDLBaseListener περιέχονται μέθοδοι δύο κατηγοριών που εκτελούνται κατά τη διάρκεια της μετατροπής της μεταγλώττισης και δημιουργίας των AST δέντρων, οι μέθοδοι enter+statement name και οι μέθοδοι exit+statement name. Με την εύρεση ενός κανόνα κατά την διάρκεια της μεταγλώττισης ενεργοποιείται αυτόματα η ανάλογη μέθοδος enter+statement name του κανόνα, έτσι ώστε να εκτελέσει τις καθορισμένες ενέργειες. Μόλις τελειώσει ο κανόνας και εξεταστούν και όλοι οι εσωτερικοί κανόνες του (ονομάζονται και παιδιά του κανόνα), τότε καλείται η μέθοδος exit + statement name εκτελώντας αντίστοιχα τις καθορισμένες ενέργειες.

Στο σημείο αυτό αξίζει να αναφερθεί ότι από την πλευρά του προγραμματιστή μπορεί να ελέγξει την διαδικασία παραγωγής μηνυμάτων σφαλμάτων ή τυχόν εξαιρέσεων που μπορεί να συμβούν κατά την διάρκεια της μετάφρασης. Η εισαγωγή κατάλληλων μηνυμάτων σφαλμάτων μπορεί να γίνει μέσα στις μεθόδους enter + statement name ή exit + statement name αν δουλεύουμε με Listener τύπου διάσχισης των δέντρων ή μέσα στην μέθοδο visit + statement name αν δουλεύουμε με Visitor τύπο διάσχισης.

Επομένως η διαδικασία που απαιτείται ώστε να ελέγξει ο προγραμματιστής την εξαγωγή τόσο σφαλμάτων που συμβαίνουν κατά την διάρκεια της μεταγλώττισης όσο και της πληροφορίας από τους κανόνες ακολουθεί παρακάτω: απαιτείται από τον προγραμματιστή η δημιουργία μίας κλάσης που να επεκτείνει είτε την κλάση DDLBaseListener αν επιθυμεί έλεγχο μέσω enter και exit μεθόδων ή επέκταση της κλάσης DDLBaseVisitor αν επιθυμεί έλεγχο μέσω visit μεθόδου. Το περιεχόμενο των μεθόδων μπορεί να ρυθμιστεί καταλλήλως ώστε να ικανοποιήσει τις ανάγκες του προγραμματιστή. Παράδειγμα της επέκτασης αυτής απεικονίζεται στο σχήμα 4:

Σχήμα 4. SchemaLoader Class από HecateParser Class, Package parser

```
private static class SchemaLoader extends DDLBaseListener {
    private String fileName;

    public SchemaLoader(String fileName) {
        this.fileName = fileName;
    }

    public void enterStart (DDLParser.StartContext ctx) {
        s = new Schema();
    }
    public void exitStart (DDLParser.StartContext ctx) {
        processUnmached();
    }
}
```

Εν κατακλείδι, ο πυρήνας της λειτουργίας του εργαλείου ANTLR είναι ο ακόλουθος: το αρχείο είσοδος μετατρέπεται σε μια ροή από χαρακτήρες η οποία εισάγεται στον λεκτικό αναλυτή (DDLlexer) που με την σειρά του την μετασχηματίζει σε tokens έτσι ώστε τελικά να μεταφραστούν από τον parser (DDLParser).

Κεφάλαιο 3. Σχεδίαση & Υλοποίηση

3.1 Ορισμός προβλήματος και αλγόριθμοι επίλυσης

Η παραγωγή κατάλληλων μηνυμάτων σφάλματος ή προειδοποιήσεων είναι απαραίτητη προϋπόθεση για την σωστή λειτουργία μετάφρασης και διαδικασίας του parsing. Το εργαλείο Εκάτη (Hecate) μέχρι και προσφάτως διέθετε την προκαθορισμένη (default) παραγωγή σφαλμάτων όπως ακριβώς δημιουργείται μέσα στην κλάση DDLParser.java. Όμως τα παραγόμενα αυτά μηνύματα είναι πολύ γενικής φύσης με αποτέλεσμα πρώτον να μην είναι καθοδηγητικά για την εύρεση του λάθους και δεύτερον δεν διευκόλυναν το πρόβλημα της αμφισημίας καθώς αρκετές φορές ο parser εμφάνιζε το ίδιο μήνυμα λάθους για διαφορετικές περιπτώσεις.

Αυτό το πρόβλημα αποτελεί και τον πρώτο στόχο αυτής της διπλωματικής εργασίας, δηλαδή την παραγωγή κατάλληλων και πιο αναλυτικών μηνυμάτων σφάλματος έτσι ώστε να βοηθήσει τον διαχειριστή του προγράμματος να εντοπίσει και να διευθετήσει τα παραγόμενα σφάλματα και λάθη της εισόδου που παρείχε.

Για την υλοποίηση του παραπάνω στόχου αρχικά απαιτείται να γίνει μια ανάλυση η οποία διαχωρίζει τις κατηγορίες λαθών και σφαλμάτων έτσι ώστε να γίνει κατανοητός ο λόγος για τον οποίο έλαβαν χώρα όλες οι λειτουργίες που ακολουθούν.

Στην πρώτη κατηγορία σφαλμάτων εντάσσονται οποιαδήποτε αλφαριθμητικά λάθη ή λάθη σύνταξης τα οποία όμως έλαβαν χώρα μέσα σε κάποιον κανόνα που αναγνωρίστηκε από τον DDLParser. Αυτής της κατηγορίας τα λάθη μπορούν να διαχειριστούν από την κλάση Listener ή Visitor που έχει δημιουργηθεί για το project. Ιδιαίτερα στην Εκάτη, η εξέταση των δέντρων και των κανόνων υλοποιείται από τύπου Listener κλάση η οποία και είναι η κλάση HecateParser. Επομένως, για τον κάθε κανόνα που αναγνωρίζεται από τον parser η διαχείριση αυτής της κατηγορίας λαθών γίνεται διαμέσου των μεθόδων `enter + statement name` ή `exit + statement name`. Ως αποτέλεσμα, μέσα σε αυτές τις

μεθόδους εισάγονται οι ενέργειες που απαιτούνται για την διαχείριση των σφαλμάτων. Πιο πολλές πληροφορίες για την δομή τους παρουσιάζεται στην επόμενη ενότητα.

Συνεχίζοντας στην ανάλυση των λαθών, ως δεύτερη κατηγορία παρουσιάζονται τα λάθη και σφάλματα που αποτελούν και αυτά αλφαριθμητικά σφάλματα ή σφάλματα σύνταξης κατά την μεταγλώττιση της εισόδου. Η ειδοποιός διαφορά όμως με την προηγούμενη κατηγορία έγκειται στο γεγονός ότι τα λάθη αυτά αποτελούν και λάθη αναγνώρισης κανόνων. Με την ορισμό αυτό, εννοούμε τα λάθη που συνέβησαν εξαιτίας του γεγονότος ότι ο parser ή ο λεκτικός αναλυτής επιχείρησε να ταιριάσει τα tokens με κάποιον κανόνα ή keyword της γλώσσας (DDL) και απέτυχε. Ως αποτέλεσμα, τα σφάλματα αυτής της κατηγορίας δεν είναι εφικτό να τα διαχειριστούμε με τις κλάσεις Listener ή Visitor καθώς δεν αναγνωρίστηκε κάποιος κανόνας για να εισέλθουμε στην αντίστοιχη μέθοδο `enter + statement name` ή `visit + statement name`. Η ορθή αντιμετώπιση τους παρουσιάζεται αμέσως παρακάτω.

Για την υλοποίηση του παραπάνω στόχου αρχικά έπρεπε να καταργηθεί η προκαθορισμένη διαδικασία παραγωγής σφαλμάτων όπου αυτό πραγματοποιούνταν μέσω της `DDLParser.java` με την δημιουργία ενός exception τύπου `RecognitionException` με κάθε εμφάνιση λάθους στους κανόνες. Η προκαθορισμένη λειτουργία αυτή απεικονίζεται στο σχήμα 5:

Σχήμα 5. `RecognitionException`, `DDLParser` Class, Package `parser`

```
catch (RecognitionException re) {
    _localctx.exception = re;
    _errHandler.reportError(this, re);
    _errHandler.recover(this, re);
}
finally {
    exitRule();
}
return _localctx;
}
```

Επομένως, για την παραγωγή ειδικών μηνυμάτων σφάλματος, τα οποία είναι όπως ακριβώς τα επιθυμεί ο προγραμματιστής απαιτείται πρώτα η κατάργηση των default error Listeners της κλάσης και της προσθήκης δικών μας κλάσεων που να διαχειρίζονται τα λάθη μόλις αυτά παραχθούν. Η προσθήκη αυτής της κλάσης απαιτείται μόνο μέσα στην κλάση `HecateParser` και συγκεκριμένα στην μέθοδο `parse`. Η αλλαγή αυτή επιτελείται όπως παρουσιάζεται στο σχήμα 6:

Σχήμα 6. Error Listener Removal from method parse, HecateParser Class

```
DDLlexer      lexer = new DDLlexer(charStream);
// -----
lexer.removeErrorListeners();
lexer.addErrorListener(throwListener);
// -----

TokenStream    tokenStream = new CommonTokenStream(lexer);
DDLParser      parser = new DDLParser(tokenStream);
// -----
parser.removeErrorListeners();
parser.addErrorListener(throwListener);
// -----
```

Όπως παρατηρείτε, απαιτείται σε δύο περιπτώσεις η κατάργηση πρώτον στον λεκτικό αναλυτή (DDLlexer) και έπειτα στον DDLParser. Φυσικά, αφού αφαιρέσαμε την προκαθορισμένη διαδικασία παραγωγής σφαλμάτων τώρα απαιτείται να δημιουργήσουμε και να τοποθετήσουμε την καινούρια λειτουργία-κλάση που να εμφανίζει τα μηνύματα που επιθυμούμε εμείς, και όπως φαίνεται από την παραπάνω απεικόνιση αυτό γίνεται με την εντολή **addErrorListener(throwListener)**.

Το όρισμα στην παραπάνω εντολή και μέθοδο αποτελεί το αντικείμενο της νέας κλάσης που δημιουργούμε για την διαχείριση λαθών και εμφάνιση μηνυμάτων. Η δημιουργία αυτού του αντικειμένου παρουσιάζεται στο σχήμα 7. Ωστόσο η πλήρης λειτουργία και ανάλυση της κλάσης που διαχειρίζεται τα λάθη, δηλαδή της κλάσης ThrowingErrorListener class παρουσιάζεται σε επόμενη ενότητα.

Σχήμα 7. Creation of ThrowingErrorListener Class, method parse in HecateParser Class

```
HashMap<String, String> errorMap = fillMap(file.getName()); // this function creates and fills the error HashMap
ThrowingErrorListener throwListener = new ThrowingErrorListener(file.getName(),outputFilePath,errorMap);

try {
    charStream = new ANTLRFileStream(filePath);
} catch (IOException e) {
    e.printStackTrace();
    return null;
}
```

3.2 Σχεδίαση και αρχιτεκτονική λογισμικού

Αρχικά, για την υλοποίηση του στόχου που τέθηκε, απαραίτητη ήταν η δημιουργία μίας κλάσης ώστε να βοηθάει στην παραγωγή πληρέστερων μηνυμάτων σφάλματος αλλά και εκσφαλμάτωση του αρχείου εισόδου. Η κλάση αυτή ονομάστηκε ThrowingErrorListener

και δημιουργήθηκε όπως ακριβώς παρουσιάστηκε στην προηγούμενη ενότητα. Αξίζει να αναφερθεί ότι για την σωστή λειτουργία της κλάσης ThrowingErrorListener μεταξύ των ορισμάτων για την δημιουργία του αντικειμένου παρατηρείται και ένα HashMap τύπου <String,String>, το οποίο περιέχει αντιστοιχίες keyword σφαλμάτων που παρατηρήθηκαν με το αντίστοιχο μήνυμα σφάλματος που θα εμφανίζεται στον χρήστη σε κάθε περίπτωση. Η δημιουργία του HashMap παρουσιάζεται στο σχήμα 8:

Σχήμα 8. Δημιουργία και τοποθέτηση στοιχείων σε errorMap, method fillMap, HecateParser class

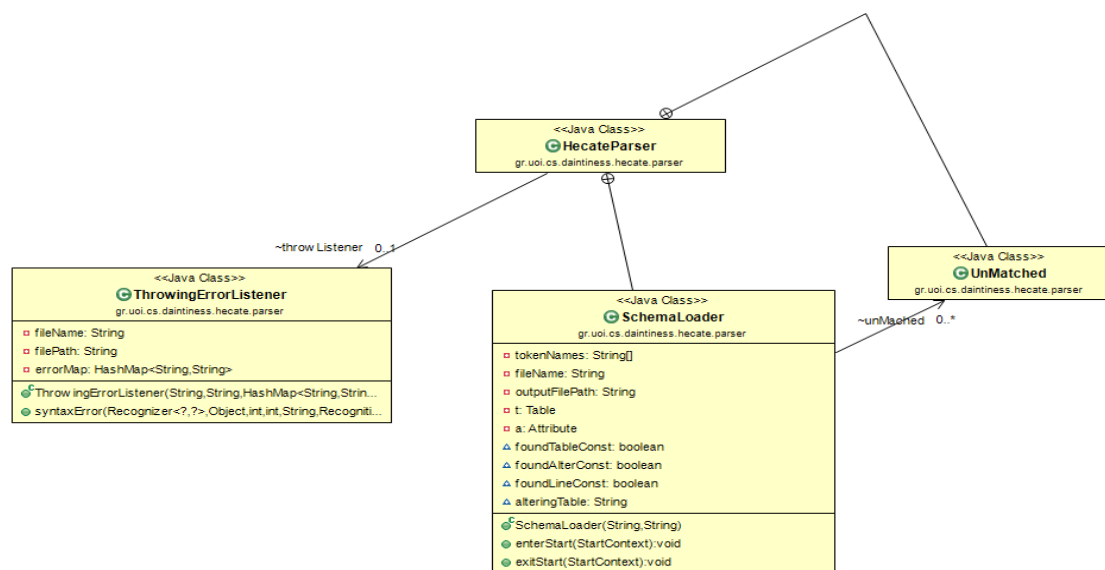
```
// this method fills the hashmap of errors with the appropriate text
public static HashMap<String,String> fillMap(String fileName){
    HashMap<String,String> errorMap = new HashMap<String,String>();
    String invalidComText = " Error: This command is not allowed in the Hecate Tool.";
    String createAlterText = " Error: Commands 'CREATE' and 'ALTER' do not accept this keyword.";
    String keywordText = " Error: This keyword is not allowed in the Hecate Tool.";
    String keywordMax = " Error: The keyword 'MAX' is not valid in the variable size definition.";
    String keywordDollar = " Error: The keyword '$' can only be used as a special character and not inside a command.";
    String keywordEqual = " Error: The keyword '=' is not allowed with 'DEFAULT' and 'SET' keywords inside a command.";
    String keywordBSize = " Error: The keyword 'KEY_BLOCK_SIZE' '+' is not valid. Perhaps you meant 'KEY_BSIZE' instead.";
    String keywordCom = " Error: The keyword ',' '+' is not allowed inside the command.";

    errorMap.put("SELECT",invalidComText);errorMap.put("SHOW",invalidComText);errorMap.put("BEGIN",invalidComText);errorMap.put("START",invalidComText);errorMap.put("REVOKE",invalidComText);errorMap.put("DO",invalidComText);errorMap.put("GRANT",invalidComText);errorMap.put("ANALYZE",invalidComText);errorMap.put("SEQUENCE",createAlterText);errorMap.put("TYPE",createAlterText);errorMap.put("EXTENSION",createAlterText);errorMap.put("ON",keywordText);errorMap.put("on",keywordText);errorMap.put("NONCLUSTERED",keywordText);errorMap.put("MODIFY",keywordText);errorMap.put("PROCEDURE",keywordText);errorMap.put("IDENTITY",keywordText);errorMap.put("MAX",keywordMax);errorMap.put("$",keywordDollar);errorMap.put("KEY_BLOCK_SIZE",keywordBSize);errorMap.put(",",keywordCom);

    return errorMap;
}
```

Έπειτα, η αλληλεπίδραση και οι σχέσεις της κλάσης ThrowingErrorListener με τις κλάσεις του parsing απεικονίζεται στο ακόλουθο UML Diagram του σχήματος 9.

Σχήμα 9. UML Diagram of ThrowingErrorListener Class and Rest of Parsing



Όπως, παρατηρούμε από το UML διάγραμμα στο σχήμα 9, UML Diagram of ThrowingErrorListener Class and Rest Of Parsing, η σχέση της ThrowingErrorListener Class αποτελεί μια απλή σχέση συσχέτισης με την κλάση HecateParser όπου και δημιουργείται. Από την σχέση αυτή κατανοούμε ότι για τη σωστή εκτέλεση και διεκπεραίωση της κλάσης HecateParser απαιτούνται από 0 έως 1 το πολύ αντικείμενα της κλάσης ThrowingErrorListener class.

Σε αυτό το σημείο αρχικά θα πρέπει να αναλυθεί σε βάθος η λειτουργία της κλάσης ThrowingErrorListener Class. Ο τρόπος με τον οποίο λειτουργεί η κλάση αυτή είναι ο εξής: διαμέσου του parser, την στιγμή που προσπαθεί να αναγνωρίσει και να αντιστοιχίσει το διαθέσιμο token που παράχθηκε από τον λεκτικό αναλυτή με κάποιον κανόνα ή keyword της γλώσσας DDL και αποτύχει, ως αποτέλεσμα, όπως είδαμε, πυροδοτείται μια εξαίρεση τύπου RecognitionException η οποία καλεί την μέθοδο **SyntaxError** της κλάσης ThrowingErrorListener.

Η μέθοδος SyntaxError της κλάσης ThrowingErrorListener αποδέχεται ως ορίσματα ιδιαίτερα χρήσιμα στοιχεία όπως είναι το OffendingSymbol το οποίο αποτελεί το token που ο parser δεν κατάφερε να αντιστοιχήσει σε κάποιον διαθέσιμο κανόνα ή keyword της γλώσσας, η γραμμή (int line) του αρχείου εισόδου στην οποία βρέθηκε το μη-αναγνωρισμένο token και τέλος ένα μήνυμα που παράγεται ως default από τον parser. Αναλυτικά ο ορισμός της μεθόδου SyntaxError παρουσιάζεται στο σχήμα 10:

Σχήμα 10. Syntax Error Method, ThrowingErrorListener Class, Package parser

```
// this method prints the messages for each error that arises.
@Override
public void syntaxError(Recognizer<?, > recognizer, Object offendingSymbol, int line, int charPositionInLine, String msg, RecognitionException e) {
    String errorMessage = ""; // in this temporary string will be stored the text of the error in order to be written in the file
    // open the file in order to write the error that has risen
```

Έπειτα, εξάγουμε το keyword αυτό από το όρισμα OffendingSymbol και το τοποθετούμε σε μια προσωρινή μεταβλητή με όνομα OffendingText. Στο σημείο αυτό πλέον απομένει να ελεγχθεί αν αυτό το keyword στο οποίο δημιουργήθηκε η RecognitionException βρίσκεται μέσα στο HashMap με τις περιπτώσεις λαθών που προβλέψαμε. Αν όντως υπάρχει, τότε παράγεται το αναλυτικό και καθοδηγητικό μήνυμα ώστε να βοηθήσει τον χρήστη να κατανοήσει τι ακριβώς λάθος παρουσιάστηκε. Αν παρουσιάστηκε περίπτωση σφάλματος για το οποίο δεν έχει προβλεφθεί μήνυμα τότε εμφανίζεται ένα πιο γενικής

φύσης μήνυμα το οποίο όμως πάλι βοηθά τον χρήστη να κατανοήσει ποιο και που βρίσκεται στο αρχείο εισόδου το λάθος. Η διαδικασία που προαναφέρθηκε παρουσιάζεται στο σχήμα 11:

Σχήμα 11. Procedure of Error Matching Using the Error HashMap, method SyntaxError, ThrowingErrorListener Class

```
// check if exists in error Map and if so write the appropriate warning message
if (errorMap.containsKey(offendingText)) {
    errorMessage = "File: "+fileName+" Line: "+line + errorMap.get(offendingText)+ " Keyword: '"+offendingText+"'";
}else {
    errorMessage = "File: "+fileName+" Line: "+line+" Error: The keyword '"+offendingText+"' is invalid. Keyword: '"+offendingText+"'";
}
```

Ως μηνύματα επομένως παράγονται τα εξής: αρχικά καταγράφεται το όνομα του αρχείου εισόδου (filename) και ακολουθεί η γραμμή στην οποία συνέβη το λάθος (line). Μετά τα δύο πρώτα στοιχεία ακολουθεί το μήνυμα της συγκεκριμένης περίπτωσης που προσφέρει μια εξήγηση του λάθους στον χρήστη, την οποία ολοκληρώνει το keyword που πυροδότησε την εξαίρεση (offendingText). Όπως αναφέρθηκε και προηγουμένως στην περίπτωση που δεν βρεθεί το συγκεκριμένο offendingText στο errorMap τότε παράγεται το πιο γενικό μήνυμα λάθους ώστε ακόμα και σε αυτή την περίπτωση να βοηθήσει τον χρήστη.

Τέλος, κάθε λάθος που καταγράφεται κάθε φορά από τον parser και την μέθοδο SyntaxError γράφεται στο αρχείο outputErrors.txt που δημιουργείται και αποθηκεύεται μαζί με όλα τα υπόλοιπα metrics μέσα στον φάκελο results που εξετάζουμε κάθε φορά. Η εξαγωγή των λαθών σε ένα αρχείο τύπου log file αποτελεί κομβικό σημείο της λειτουργίας της μεθόδου SyntaxError αλλά και γενικά του εργαλείου Εκάτη, καθώς με αυτόν τον τρόπο γίνεται ευκολότερη η παρακολούθηση και αντιμετώπιση των λαθών αλλά και η εξαγωγή μετέπειτα στατιστικών στοιχείων.

Όπως αναλύθηκε προηγουμένως, για την πλήρη παραγωγή των μηνυμάτων και στις δύο περιπτώσεις λαθών απαιτήθηκε: πρώτον, η παραγωγή της κλάσης ThrowingErrorListener, και, δεύτερον, η επέκταση της κλάσης SchemaLoader, η οποία αποτελεί τον Listener της Εκάτης που διασχίζει και επεκτείνει τους αναγνωρισμένους κανόνες. Στη συνέχεια, ακολουθεί η ανάλυση της επέκτασης που έλαβε χώρα στην κλάση SchemaLoader ώστε να παραχθούν συγκεκριμένα μηνύματα σφάλματος που αυτή την φορά συνέβησαν μέσα στους κανόνες οι οποίοι αναγνωρίστηκαν από τον parser.

Αρχικά, η κλάση SchemaLoader επεκτείνει την κλάση DDLBaseListener και επομένως διαθέτει ένα σύνολο μεθόδων enter + rule name και exit + rule name οι οποίες εκτελούν ενέργειες αφότου αναγνωριστεί και αντιστοιχηθεί ο εκάστοτε κανόνας της γλώσσας. Η προαναφερθείσα επέκταση της κλάσης αυτής για τον σκοπό της παραγωγής κατάλληλων μηνυμάτων σφάλματος έγκειται στο γεγονός της προσθήκης των μεθόδων αυτών έτσι ώστε όταν αναγνωριστεί ένας κανόνας της γλώσσας να παράγεται ένα στοχευμένο μήνυμα λάθους ανάλογα πάντα με την κάθε περίπτωση.

Παράδειγμα της προσθήκης αυτής ακολουθεί στο σχήμα 12:

Σχήμα 12. enterCreate_statement method, SchemaLoader Class in HecateParser Class

```
@Override
public void enterCreate_statement(@NotNull DDLParser.Create_statementContext ctx) {
    if (ctx.getText().contains("PROCEDURE")) {
        showError(fileName, ctx.start.getLine(), "Error: Command CREATE does not support action PROCEDURE, only {'TABLE', 'DATABASE', 'VIEW', 'TRIGGER', 'INDEX', 'pl_sql'}.");
    }
}
```

Ξεκινώντας από το όρισμα της μεθόδου enterCreate_statement το οποίο και είναι η γραμμή του αρχείου εισόδου στο οποίο αναγνωρίστηκε το keyword CREATE το οποίο αυτό σημαίνει ότι έχουμε ένα statement αρχικά τύπου ddl_statement όπως αναγράφεται σε γραμματική (DDL.g4) και μετά ο parser οδηγείται στον κανόνα create_statement. Ο κανόνας αυτός αφού αναγνωρίσει το keyword CREATE μετά περιμένει να εντοπίσει κάποιο από τα keywords ['database', 'table', 'index', 'view', 'trigger', 'pl_sql'] καθώς μόνο αυτά τα keywords υποστηρίζονται για τον κανόνα create. Ο κανόνας αυτός της γραμματικής παρουσιάζεται στο σχήμα 13.

Σχήμα 13. Κανόνας create_statement της γραμματικής DDL.g4, Package parser

```
create_statement : CREATE ( database | table | index | view | trigger |
pl_sql ) ;
```

Ως αποτέλεσμα, εάν μέσα σε αυτήν την γραμμή του create statement που μας δώσει ο parser ως το ctx όρισμα περιέχεται στην περίπτωση μας το keyword PROCEDURE, τότε αυτή η περίπτωση δεν είναι αποδεκτή από τον ορισμό του κανόνα και το γεγονός αυτό πυροδοτεί την παραγωγή μηνύματος λάθους που γράφεται στο log file των σφαλμάτων. Το ίδιο ισχύει και για παρόμοιες περιπτώσεις με εσφαλμένα keywords που δεν υποστηρίζονται από την γλώσσα.

Προχωρώντας στην ανάλυση της λειτουργίας της κλάσης SchemaLoader θα παρουσιαστεί αυτή την στιγμή ένα πιο πολύπλοκο παράδειγμα μεθόδου για τον χειρισμό του αναγνωρισμένου κανόνα. Στο παράδειγμα αυτό θα αναλύσουμε τον χειρισμό του κανόνα index ο οποίος αποτελεί παιδί του κανόνα create statement και εμφανίζεται στο σχήμα 14. Με τον όρο παιδί εννοείται το γεγονός ότι μετά την αναγνώριση του κανόνα create statement ο parser αναγνώρισε το keyword index και το αντιστοίχισε με τον συγκεκριμένο κανόνα. Επομένως για τον χειρισμό και την παραγωγή κατάλληλων μηνυμάτων σφάλματος δημιουργήθηκε η μέθοδος enterIndex(@NotNull DDLParser.IndexContext ctx) η οποία και παρουσιάζεται και αναλύεται στο σχήμα 15.

Σχήμα 14. Index Rule in Grammar, DDL.g4, Package parser

```
index
: ( ( UNIQUE | FULLTEXT | SPATIAL )? ) INDEX
  index_name index_type? ON table_name parNameList index_option?
;
```

Σχήμα 15. enterIndex method, SchemaLoader Class in HecateParser Class

```
@Override
public void enterIndex(@NotNull DDLParser.IndexContext ctx) {
    if (ctx.getText().contains("ifNotExists") || ctx.getText().contains("ifnotexists") || ctx.getText().contains("IFNOTEXISTS")) {
        showError(fileName, ctx.start.getLine(), "Error: Create Index statement does not support IF NOT EXISTS option.");
    }

    if (ctx.parNameList() != null) {
        String parNameFixed = ctx.parNameList().getText().substring(1, ctx.parNameList().getText().length()-1); // skip the parenthesis
        String[] parNameVariables = parNameFixed.split(",");

        for (int i = 0; i < parNameVariables.length; i++) {
            if (checkTokenList(tokenNames, parNameVariables[i].toUpperCase())) {
                showError(fileName, ctx.start.getLine(), "Error: Maybe missing quotes ( ` ` ) in command CREATE INDEX.");
            }
        }
    }

    if (ctx.getText().contains("NONCLUSTERED") || ctx.getText().contains("nonclustered")) {
        showError(fileName, ctx.start.getLine(), "Error: Create Index statement does not support 'NONCLUSTERED' option.");
    }

    // the below code checks the validity of the using option inside the create index command.
    if (ctx.getText().contains("USING")) {
        int usingIndex = ctx.getText().indexOf("USING");
        String usingText = ctx.getText().substring(usingIndex + 1, ctx.getText().length());
        checkUSINGvalidity(fileName, ctx.start.getLine(), usingText);
    }
}
```

Στην προηγούμενη απεικόνιση παρατηρείται αρχικά ότι όπως και προηγουμένως, αυτή την φορά δεν επιτρέπονται τα keywords IFNOTEXISTS και NONCLUSTERED καθώς δεν υποστηρίζονται από την γραμματική της Εκάτης για τον κανόνα index. Έπειτα, ελέγχεται η περίπτωση να έχουν παραληφθεί τα μονά εισαγωγικά (') στο όνομα του index καθώς σε ορισμένες περιπτώσεις το όνομα του index αποτελούσε και bound token word της γλώσσας δηλαδή ένα keyword που η γλώσσα το χρησιμοποιεί για την αναγνώριση

κάποιων κανόνων. Ως αποτέλεσμα, εισήχθη κατάλληλο μήνυμα που δημιουργείται σε αυτή την περίπτωση.

Τέλος, σε αυτή την μέθοδο για τον κανόνα αυτό ελέγχεται η περίπτωση της ύπαρξης του keyword USING. Σ' αυτή την περίπτωση χρησιμοποιείται η βοηθητική μέθοδος checkUSINGvalidity(), η οποία ελέγχει την συγγραφή της εντολής στο αρχείο εισόδου από το keyword USING έως και το τέλος της εντολής. Η λειτουργία της μεθόδου παρουσιάζεται στο σχήμα 16:

Σχήμα 16. checkUSINGvalidity Method, SchemaLoader Class in HecateParser Class

```
private void checkUSINGvalidity(String fileName,int lineNumber,String usingText) {  
    if (usingText.charAt(0) != '(' || usingText.contains("missing") || usingText.charAt(usingText.length() - 1) != ')') {  
        showError(fileName,lineNumber,"Error: USING option is not spelled correctly. The right syntax is this: (USING(BTREE|HASH))");  
    }  
}
```

Όπως παρατηρείται, αν υπάρχει το keyword USING στον κανόνα index, είναι αναγκαίο να γραφεί με ένα συγκεκριμένο τρόπο ώστε να γίνει αποδεκτό. Αυτό σημαίνει ότι αναγκαστικά πρέπει να ξεκινάει και να τελειώνει με σύμβολο παρένθεσης, να υπάρχει το keyword USING και κάποιο εκ των keywords ['BTREE','HASH']. Ως αποτέλεσμα, αν κάποιοι από τους προαναφερθέντες περιορισμούς δεν ικανοποιούνται, τότε παράγεται το αντίστοιχο μήνυμα λάθους που γράφεται και αυτό όπως και όλα τα άλλα στο log file.

Σε αυτό το σημείο αξίζει να αναλυθεί ακόμα μια περίπτωση της επέκτασης της κλάσης SchemaLoader. Η ειδοποιός διαφορά της περίπτωσης αυτής με όλες τις προαναφερθέντες είναι ότι το εργαλείο Hecate παρουσίαζε πλήρη αποσυντονισμό της λειτουργίας του. Ο λόγος λοιπόν γίνεται για τον κανόνα constraint ο οποίος αποτελεί παιδί της alter_constraint ο οποίος με τη σειρά του αποτελεί παιδί του κανόνα alter_statement. Ο κανόνας αυτός συνθέτεται από πολλές περιπτώσεις, όμως η περίπτωση η οποία προκαλούσε τον αποσυντονισμό της λειτουργίας της Εκάτης αποτέλεσε η περίπτωση εισαγωγής ξένου κλειδιού(Foreign Key). Η δομή του κανόνα παρουσιάζεται στο σχήμα 17.

Σχήμα 17. Constraint Rule in Grammar, DDL.g4, Package parser constraint

```
CONSTRAINT? constr_name? FOREIGN KEY index_name? parNameList  
reference_definition
```


Σε αυτή την περίπτωση η λειτουργία της Εκάτης ήταν εσφαλμένη. Οι αιτίες που προκαλούσαν την κατάσταση αυτή ήταν δύο. Αρχικά, για τα εισαγωγικά διαφόρων τύπων δηλαδή [' , " , `] στο όνομα του πίνακα δεν είχε προβλεφθεί διαχείριση και αντιμετώπιση. Ως αποτέλεσμα οι εντολές του σχήματος 18,

Σχήμα 18. Εντολές ανάθεσης ονόματος πινάκων

```
orTable = s.getTables().get(alteringTable);

reTable = s.getTables().get(reTableName);
```

προσπαθούσαν να τοποθετήσουν τα ονόματα των πινάκων στις μεταβλητές orTable και reTable. Επομένως, λόγω των εισαγωγικών που υπήρχαν στα ονόματα των πινάκων δεν μπορούσαν να βρεθούν μέσα στο σύνολο των πινάκων και ως αποτέλεσμα το εργαλείο Εκάτη αποτύγχανε να λειτουργήσει λόγω εξαίρεσης τύπου Null Pointer Exception.

Για την αντιμετώπιση του προαναφερθέντος προβλήματος εντάχθηκαν δύο μέθοδοι των οποίων η δομή και χρήση παρουσιάζεται στα σχήματα 19 και 20.

Σχήμα 19. enterForeign method, SchemaLoader Class in HecateParser Class

```
// this if statement checks if there is any keyword between the alter table command and the table name
if (!checkEquality(s.getTables().keySet(),removeQuotes(alteringTable))) {
    String context = (ctx.getRuleContext().getParent()).getParent().getText();
    String contextTail = (ctx.getRuleContext().getParent()).getParent().getText();
    context = context.substring(0, context.indexOf(alteringTable));
    context+=(ctx.getRuleContext().getParent()).getParent().getText().
        substring(contextTail.indexOf(alteringTable)+alteringTable.length(),contextTail.length());
    alteringTable = getTableNameFromRelation(context,s.getTables().keySet());
}

orTable = s.getTables().get(removeQuotes(alteringTable));
reTable = s.getTables().get(removeQuotes(reTableName));
```

Σχήμα 20. getTableNameFromRelation and checkEquality methods, SchemaLoader Class

```
// method to handle alter table command with foreign key
private String getTableNameFromRelation(String context, Set<String> tablesSet) {
    String result = "";
    String con = "ALERTABLE";

    if (context.contains("ALERTABLE") && context.contains("ADDCONSTRAINT")) {
        result = context.substring(context.indexOf("ALERTABLE")+con.length(),context.indexOf("ADDCONSTRAINT"));
    }else if (context.contains("altertable") && context.contains("constraint")) {
        result = context.substring(context.indexOf("altertable")+con.length(),context.indexOf("addconstraint"));
    }else if (context.contains("altertable") && context.contains("ADDCONSTRAINT")) {
        result = context.substring(context.indexOf("altertable")+con.length(),context.indexOf("ADDCONSTRAINT"));
    }else if (context.contains("ALERTABLE") && context.contains("constraint")) {
        result = context.substring(context.indexOf("ALERTABLE")+con.length(),context.indexOf("addconstraint"));
    }

    if (checkEquality(tablesSet,removeQuotes(result))) {
        return result;
    }else { // this is not suppose to happen
        System.out.println("Error: Command ALTER TABLE is not parsed correctly!");
        System.exit(1);
    }
    return "ERROR";
}

// this method checks if the table name is actually a table name!
private static boolean checkEquality(Set<String> tablesSet, String tableName) {
    boolean flag = false;
    for (String table : tablesSet) {
        if (table.equals(tableName)) {
            flag = true;
            break;
        }
    }
    return flag;
}
```


Η αντιμετώπιση του προβλήματος αναλύεται ως εξής: αρχικά μέσα στην μέθοδο `enterForeign` ελέγχουμε αν ο πίνακας `alteringTable` υπάρχει μέσα στο `set` των πινάκων αφαιρώντας προηγουμένως τα εισαγωγικά. Αυτό γίνεται με την εντολή `(!checkEquality(s.getTables().keySet(),removeQuotes(alteringTable)))`. Αν η συνθήκη αυτή αποτιμηθεί ως συνολικά ψευδής αυτό σημαίνει ότι το όνομα του πίνακα βρέθηκε στο σύνολο πινάκων και συνεχίζουμε με την εντολή `ALTER`. Αν όμως η εντολή αυτή αποτιμηθεί συνολικά αληθής τότε το όνομα του πίνακα δεν βρέθηκε και απαιτείται αντιμετώπιση για ακόμη μια φορά.

Σε αυτή την περίπτωση καλείται να χρησιμοποιηθεί η μέθοδος `getTableNameFromRelation`. Η λειτουργία της μεθόδου αυτής έγκειται στο γεγονός ότι εισάγει ως όνομα του πίνακα ότι υπάρχει στο `alter-statement` μεταξύ των keywords `ALTERTABLE` και `ADDCONSTRAINT`. Αφαιρώντας για ακόμη μια φορά ότι υπάρχουντα εισαγωγικά, ελέγχεται αν το όνομα του πίνακα υπάρχει στο `set` των πινάκων και αν αληθεύει αυτό τότε εισάγεται το όνομα του πίνακα στην αντίστοιχη μεταβλητή. Αν η παραπάνω συνθήκη αποτιμηθεί ως ψευδής τότε ο πίνακας αυτός δεν υπάρχει, δηλαδή δεν έχει δημιουργηθεί στην βάση, και επομένως παράγεται το κατάλληλο μήνυμα λάθους.

3.3 Σχεδίαση και αποτελέσματα ελέγχου του λογισμικού

Ο έλεγχος ορθής λειτουργίας των συστημάτων λογισμικού καταλαμβάνει σύμφωνα με έρευνές το 35-50% της συνολικού απαιτούμενου χρόνου για την παραγωγή του λογισμικού.

Αρχικά, ο έλεγχος ορθής λειτουργίας του εργαλείου Εκάτη αποτελείται από δύο tests μέσω των οποίων διαφαίνεται η σωστή λειτουργία του εργαλείου καθώς αυτά αποτελούν τους βασικούς πυλώνες του. Πρώτο test αφορά τον έλεγχο της λειτουργίας εξαγωγής των διαφορών μεταξύ δύο εκδόσεων της βάσης δεδομένων. Την λειτουργία αυτή προσφέρει η κλάση `SqlDifferenceExtractor` η οποία καλείται μέσω της κλάσης `HecateBackEndEngine`. Στην κλάση του test επομένως, εξετάζουμε έναν φάκελο με

εκδόσεις μιας βάσης δεδομένων ώστε να αποφανθούμε εάν η ενέργεια αυτή επιτελείται επιτυχημένα. Ο έλεγχος ορθής λειτουργίας αυτής της κλάσης αποτελεί άκρας σημασίας για το εργαλείο διότι με βάση την κλάση αυτή πραγματοποιείται και η εξέταση ενός ολόκληρου φακέλου εκδόσεων της βάσης.

Έπειτα, δεύτερο test για το εργαλείο Εκάτη αποτελεί ο έλεγχος της λειτουργίας της κλάσης `HecateBackEndEngine`. Η κλάση αυτή προμηθεύει όλες τις λειτουργίες του εργαλείου Εκάτη, επομένως ο έλεγχος της αποτελεί σημείο αναφοράς του testing. Ξεκινώντας το testing διαγράφουμε οτιδήποτε παλαιά αρχεία αποτελεσμάτων υπάρχουν ώστε το testing να είναι ανεξάρτητο από προηγούμενα αποτελέσματα. Αφού ολοκληρωθεί η ενέργεια αυτή με επιτυχία, έπειτα ξεκινά ο έλεγχος της συγκεκριμένης περίπτωσης, όπου και θα εξετάσουμε έναν ολόκληρο φάκελο με εκδόσεις μιας βάσης δεδομένων. Στη συνέχεια, συγκρίνουμε τα περιεχόμενα τεσσάρων χαρακτηριστικών αρχείων στα οποία περιέχονται τα αποτελέσματα της σύγκρισης των εκδόσεων. Τα αρχεία αυτά είναι τα `metrics.csv`, `table_stats.csv`, `tables_DetailedStats.tsv` και `SchemaHeartBeat.tsv`. Εάν η σύγκριση αυτή στεφθεί με επιτυχία τότε ο συνολικός έλεγχος της κλάσης αυτής έχει ολοκληρωθεί και το testing σταματά.

3.4 Λεπτομέρειες εγκατάστασης και υλοποίησης

Όσον αφορά τα εργαλεία υλοποίησης, το εργαλείο Εκάτη χτίστηκε στην πλατφόρμα Eclipse χρησιμοποιώντας το `jdk-14`. Ωστόσο απαιτούνται και ορισμένες βιβλιοθήκες ώστε να χρησιμοποιηθεί με ασφάλεια το εργαλείο Εκάτη. Όλες οι απαραίτητες αυτές βιβλιοθήκες στις οποίες συγκαταλέγονται η `antlr-runtime-4.4.jar` και η `commons-io-2.6.jar` της Apache περιέχονται στον φάκελο `libs` του εργαλείου.

3.5 Επεκτασιμότητα του λογισμικού

Η επεκτασιμότητα του λογισμικού αποτελεί έναν από τους σημαντικότερους πυλώνες της δημιουργίας ενός συστήματος λογισμικού ή εργαλείου όπως η Εκάτη καθώς απαιτείται η εύκολη μετατροπή του κώδικα σε μελλοντικές επεκτάσεις ή συντηρήσεις. Βασιζόμενοι σ' αυτό το σκεπτικό οργανώθηκε η παρακάτω λίστα η οποία απεικονίζει σημεία του κώδικα στα οποία η προσθήκη κώδικα μπορεί να επιτευχθεί χωρίς δυσκολία.

Σε αυτή την λίστα απεικονίζονται περιπτώσεις στον κώδικα στις οποίες σε μελλοντικές επεκτάσεις η προσθήκη επιπλέον περιπτώσεων είναι εύκολο να επιτευχθεί. Αναλυτικά:

- Μέθοδος `SyntaxError` που χρησιμοποιεί το `HashMap` με τις περιπτώσεις λαθών και αντίστοιχων μηνυμάτων. Η μέθοδος **`fillMap`** της κλάσης `HecateParser`, η οποία δημιουργεί το `HashMap`, μπορεί να επεκταθεί για επιπλέον περιπτώσεις με την μόνη προϋπόθεση την εισαγωγή του keyword λάθους και το αντίστοιχο μήνυμα που θα καταγράφεται στο log file.
- Η κλάση `SchemaLoader` θα απαιτηθεί να συντηρηθεί στο μέλλον εξαιτίας ίσως της δημιουργίας συστημάτων DBMS τα οποία εξάγουν με λίγο διαφορετικό τρόπο διάφορες εντολές. Η μεταβλητότητα αυτή θα πρέπει να διαχειριστεί ώστε το εργαλείο `Hecate` να λειτουργεί χωρίς προβλήματα. Για να συντηρηθεί η κλάση αυτή αρχικά θα απαιτηθεί η προσθήκη μεθόδων `enter + statement name` ή `exit + statement name` που να αντιμετωπίζουν τα προβλήματα με τους κανόνες που αναγνωρίζονται από τον parser και υπάρχουν στην γραμματική αλλά με συντακτικά λάθη στην δομή τους.
- Σημείο του κώδικα στο οποίο είναι πιθανό να απαιτηθεί συντήρηση ή επέκταση αποτελεί η μέθοδος `enterForeign`. Η μέθοδος αυτή διαχειρίζεται την περίπτωση της εισαγωγής ξένου κλειδιού σε πίνακα. Σημείο συντήρησης αποτελεί η προαναφερθείσα διαδικασία με την οποία εντοπίζεται το όνομα πίνακα καθώς πρώτον είναι πιθανό να υπάρχουν διαφορετικά σύμβολα από τα εισαγωγικά στο όνομα του πίνακα. Επιπρόσθετα, είναι πιθανό το όνομα του πίνακα να βρίσκεται ανάμεσα σε διαφορετικά keywords από τα υπάρχοντα ('ALTERTABLE', 'ADDCONSTRAINT') και ως αποτέλεσμα χρήζει μελλοντικής συντήρησης ή επέκτασης.

Κεφάλαιο 4. Εργαλείο Insight

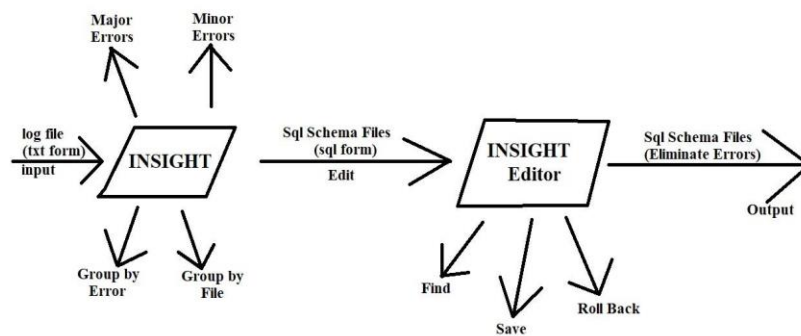
4.1 Το εργαλείο Insight

Η δεύτερη συνεισφορά της Διπλωματικής αυτής είναι η σχεδίαση και υλοποίηση ενός εργαλείου απεικόνισης των παραγόμενων σφαλμάτων ώστε να μπορεί ο αναλυτής να προχωρήσει ακόμα ένα βήμα στην κατανόηση των σφαλμάτων αλλά και να τα διαχωρίσει σε πιο σημαντικά και σε λιγότερα σημαντικά λάθη όπως είναι τα συντακτικά λάθη. Το εργαλείο αυτό ονομάζεται Insight (Ενόραση). Το εργαλείο Insight επιτρέπει την ομαδοποίηση των σφαλμάτων (καθώς πολλά από αυτά επαναλαμβάνονται σε πολλές εκδοχές του DDL αρχείου) ώστε να μπορεί ο αναλυτής να καταλάβει τα «μοναδικά» σημεία αποτυχίας του parsing. Επιπρόσθετα, το εργαλείο αυτό προσφέρει την δυνατότητα τροποποίησης των αρχείων sql που συμμετέχουν στο σχήμα της βάσης προσφέροντας με αυτόν τον τρόπο, παραλληλισμό των ενεργειών, παρατήρηση του σφάλματος και ταυτόχρονα αντιμετώπιση του.

Ξεκινώντας, το εργαλείο Insight αναγνωρίζει τα λάθη μετάφρασης τα οποία έχουν αποθηκευτεί σε ένα αρχείο κειμένου τύπου txt. Το εργαλείο Hecate παράγει αυτό το αρχείο και το ονοματίζει ως **outputErrors.txt**. Έπειτα, το εργαλείο Insight διαθέτει ένα σύνολο από ενέργειες όπως είναι η ομαδοποίηση των σφαλμάτων είτε ως κοινό σφάλμα στο ίδιο αρχείο είτε ως κοινό σφάλμα μεταξύ διαφορετικών αρχείων. Επιπρόσθετα, προσφέρει δυνατότητα εύρεσης λέξεων ή σφαλμάτων μέσα στο log file που απεικονίζεται. Σε επόμενο στάδιο, δίνεται η δυνατότητα της επεξεργασίας των sql αρχείων που περιέχονται στο σχήμα που εξετάζουμε ώστε να αντιμετωπιστούν άμεσα τα λάθη που συνέβησαν. Την δυνατότητα αυτή προμηθεύει ένας editor που έχει υλοποιηθεί και ενσωματωθεί στο εργαλείο Insight, ο οποίος λαμβάνει ως είσοδο sql τύπου αρχεία με

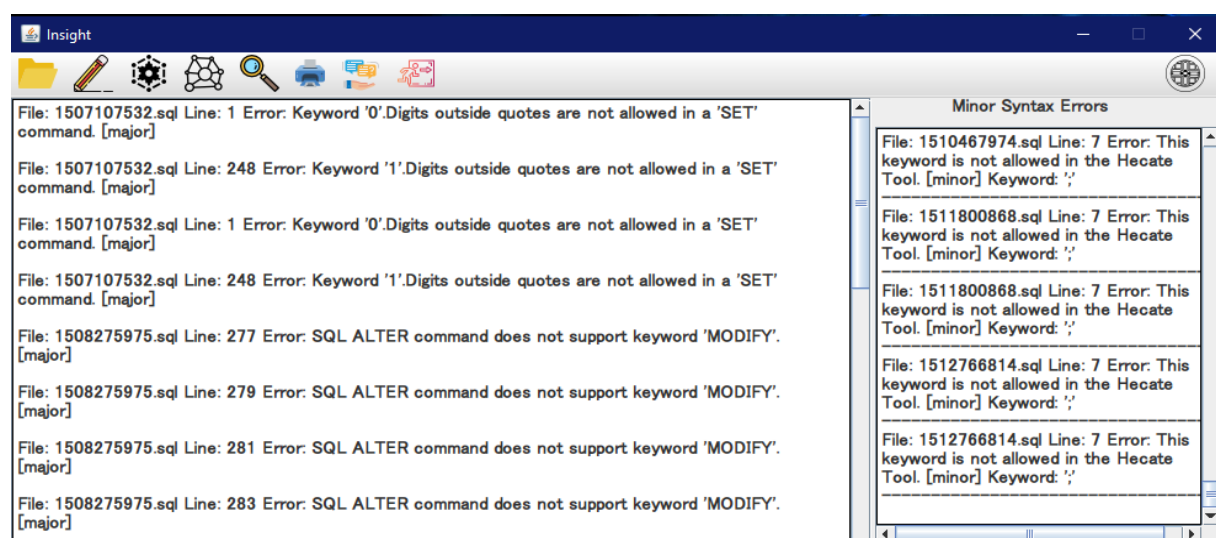
απώτερο στόχο την εκσφαλμάτωση τους ώστε να μεταγλωττιστούν επιτυχώς αργότερα από την Εκάτη. Η ροή των προαναφερθέντων ενεργειών απεικονίζεται στο σχήμα 21.

Σχήμα 21. Ακολουθία ενεργειών στο εργαλείο Insight



Αρχικά, βασικός πυλώνας του εργαλείου Insight είναι η ικανότητα του να απεικονίζει τα σφάλματα που συνέβησαν στην μετάφραση των sql αρχείων από το εργαλείο Εκάτη. Το εργαλείο Insight διαθέτει την δυνατότητα διαχωρισμού των σφαλμάτων σε σημαντικά (major) σφάλματα και σε λιγότερο σημαντικά (minor). Στα σημαντικά σφάλματα εντάσσονται σφάλματα όπως, μη υποστηριζόμενες εντολές SQL καθώς η υποστηριζόμενη γλώσσα είναι η DDL (Data Definition Language), εσφαλμένη λειτουργία υποστηριζόμενων εντολών όπως για παράδειγμα πολλαπλά ADD CONSTRAINT σε εντολή ALTER TABLE ενώ η γλώσσα επιτρέπει μόνο ένα την φορά, και άλλα σφάλματα. Στα λιγότερο σημαντικά, εντάσσονται σφάλματα συντακτικής φύσεως. Ο διαχωρισμός αυτός αποτυπώνεται στο σχήμα 22.

Σχήμα 22. Major and minor faults representation at Insight

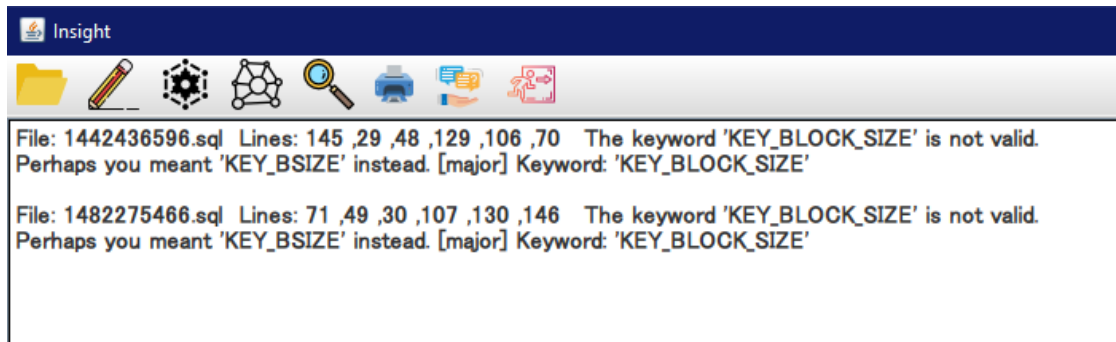


Σε αυτό το σημείο θα ακολουθήσει λίστα με όλες τις ενέργειες που διαθέτει το εργαλείο Insight αλλά και τα εικονίδια από τα οποία διατελούμε τις ενέργειες αυτές. Αναλυτικά:

- Load log file: η ενέργεια εκτελείται από το πιο αριστερό εικονίδιο διαλέγοντας την επιλογή Load Log File ή πληκτρολογώντας τον συνδυασμό των πλήκτρων Ctrl + o. Έπειτα, ανοίγει ένα νέο παράθυρο από όπου ο αναλυτής θα διαλέξει το log file και στη συνέχεια τα περιεχόμενα του αρχείου απεικονίζονται στη Insight με τον προκαθορισμένο τρόπο.
- Edit Sql File: η ενέργεια αυτή εκτελείται από το δεύτερο εικονίδιο διαλέγοντας την επιλογή Edit Sql File ή πληκτρολογώντας τον συνδυασμό πλήκτρων Ctrl + e. Έπειτα, με δεξί κλικ επάνω στο επιλεγθέν αρχείο ανοίγει ένα menu το οποίο προσφέρει την επιλογή Edit. Εναλλακτικά, είναι εφικτή η επεξεργασία sql αρχείου και από την λίστα αρχείων που παρουσιάζεται κάτω δεξιά στο εργαλείο. Με τον ίδιο τρόπο, στο επιλεγθέν αρχείο δεξί κλικ ώστε να ανοίξει ο editor για την επεξεργασία του sql αρχείου.
- Group by Error/ Restore initial log file contents: οι ενέργειες αυτές προσφέρονται από το ίδιο menu το οποίο απεικονίζεται από το τρίτο εικονίδιο. Η ενέργεια Group by error ομαδοποιεί τα στιγμιότυπα του ίδιου σφάλματος στο ίδιο αρχείο. Ως αποτέλεσμα, εμφανίζει το αρχείο που βρίσκεται το λάθος το οποίο ακολουθούν οι γραμμές που βρίσκεται το λάθος και τέλος το μήνυμα του λάθους. Η ενέργεια αυτή καλείται με τον συνδυασμό των πλήκτρων alt + e ή φυσικά διαλέγοντας την επιλογή Group by error από το αντίστοιχο menu. Επιπρόσθετα στο ίδιο menu, προμηθεύεται η δυνατότητα της επαναφοράς στα αρχικά περιεχόμενα του log file που απεικονίστηκαν, διαλέγοντας την επιλογή από το

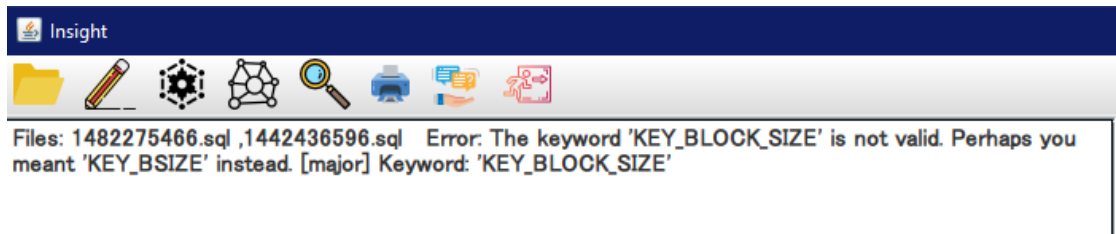
ίδιο menu ή πληκτρολογώντας τον συνδυασμό πλήκτρων Ctrl + r. Αποτέλεσμα αυτού του είδους της ομαδοποίησης απεικονίζεται στο σχήμα 23.

Σχήμα 23. Group by Error



- Group by file: η δεύτερη προμηθευόμενη ενέργεια ομαδοποίησης μπορεί να εκτελεστεί πληκτρολογώντας τον συνδυασμό πλήκτρων alt + f ή διαλέγοντας την επιλογή από το τέταρτο menu. Η ενέργεια αυτή ομαδοποιεί τα κοινά σφάλματα που λαμβάνουν χώρα σε ξεχωριστά αρχεία. Ως, αποτέλεσμα, θα εμφανιστούν τα αρχεία στα οποία βρίσκονται τα σφάλματα χωρισμένα με κόμμα και μετά θα ακολουθήσει το μήνυμα λάθους. Αποτέλεσμα αυτού του είδους της ομαδοποίησης απεικονίζεται στο σχήμα 24.

Σχήμα 24. Group by file

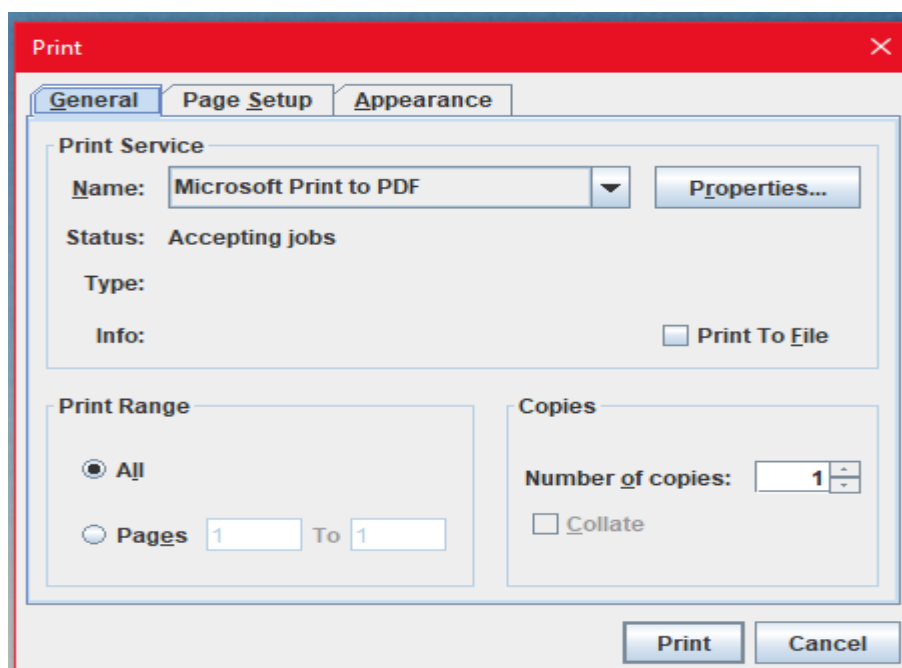


- Find Function: το εργαλείο Insight υποστηρίζει ικανότητα εύρεσης οποιασδήποτε λέξης μέσα στο log file. Η ενέργεια αυτή ενεργοποιείται είτε με τον συνδυασμό των πλήκτρων Ctrl + f είτε διαλέγοντας την επιλογή Find/Search από το πέμπτο menu. Επιπρόσθετα, δίνονται αρκετές επιλογές στα είδη της αναζήτησης που μπορεί να επιτευχθεί στα οποία συγκαταλέγονται αναζήτηση σε ολόκληρο ή στο μισό log file, case sensitive αναζήτηση το οποίο σημαίνει ότι μπορεί να ανακαλύψει επιτυχώς οποιαδήποτε μορφή της λέξης που αναζητείται (όλα τα γράμματα κεφαλαία, όλα τα γράμματα πεζά) και τέλος την επιλογή whole word όπου ανακαλύπτει επιτυχώς μόνον τα στιγμιότυπα της λέξης στα οποία η λέξη δεν αποτελεί μέρος μια μεγαλύτερης λέξης. Η find function θα παρουσιαστεί

αναλυτικά στο εγχειρίδιο χρήσης. Η λειτουργία find function υφίσταται και στον editor, στον οποίο επιτρέπει και πιο πολλές ενέργειες.

- **Print Log file:** Η ενέργεια αυτή μπορεί να επιτελεστεί πληκτρολογώντας Ctrl + p ή διαλέγοντας την επιλογή Print Log file από το έκτο menu. Όπως αναφέρει και το όνομα της, επιτελεί την εκτύπωση του αρχείου σε πραγματικό εκτυπωτή αν τέτοιος εκτυπωτής είναι συνδεδεμένος διαφορετικά το εξάγει σε αρχείο τύπου pdf. Το παράθυρο που προμηθεύει όλες τις διαθέσιμες δυνατότητες για εκτύπωση απεικονίζεται στο σχήμα 25.

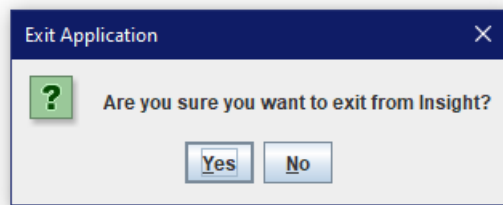
Σχήμα 25. Print Window



- **Service and Help Menu:** το εργαλείο Insight παρέχει ένα menu το οποίο περιλαμβάνει όλες τις απαραίτητες πληροφορίες για την Insight εφαρμογή. Στις πληροφορίες αυτές συγκαταλέγονται πληροφορίες για όλα τα εργαλεία αλλά και ο ορθός τρόπος χρήσης τους, όλες οι διαθέσιμες συντομεύσεις πληκτρολογίου (keyboard shortcuts) για πιο προχωρημένους χρήστες της εφαρμογής αλλά και γενικές πληροφορίες για το εργαλείο. Το service and help menu μπορεί να εμφανιστεί κάνοντας κλικ στο έβδομο κατά σειρά menu στην μπάρα με τα εικονίδια.
- **Exit Option:** φυσικά, το εργαλείο Insight παρέχει εύκολο και σαφή τρόπο διαφυγής από την εφαρμογή απλά διαλέγοντας το όγδοο και τελευταίο menu από την μπάρα εικονιδίων ή εναλλακτικά πατώντας το πλήκτρο **Esc**. Ένα

διαγνωστικό παράθυρο ελέγχου θα εμφανιστεί ζητώντας τελική επιβεβαίωση για τερματισμό της εφαρμογής. Το παράθυρο αυτό απεικονίζεται στο σχήμα 26.

Σχήμα 26. Exit confirmation window



4.2 Σχεδίαση και Υλοποίηση

4.2.1 Αρχιτεκτονική πακέτων

Στο πλαίσιο της σχεδίασης και υλοποίησης δημιουργήθηκαν κλάσεις και πακέτα κλάσεων γύρω από έξι βασικούς πυλώνες στους οποίους στηρίχθηκε όλο το εργαλείο Insight. Αρχικά πυλώνα πρώτο, αποτελεί ένα πακέτο στο οποίο συμπεριλαμβάνονται κλάσεις που δημιουργούν και διαχειρίζονται το βασικό παράθυρο του εργαλείου. Το πακέτο αυτό ονομάζεται `guimainconsole`. Δεύτερος πυλώνας στον οποίο στηρίζεται το εργαλείο Insight είναι το πακέτο `guitexteditor`. Οι κλάσεις του πακέτου αυτού δημιουργούν το κεντρικό παράθυρο του ενσωματωμένου editor που λαμβάνει χώρα στο εργαλείο αλλά και επιπρόσθετα παράθυρα όπως το παράθυρο που προσφέρει την λειτουργία αναζήτησης και αντικατάστασης (`find/replace`) και τέλος το παράθυρο που προσφέρει τα αρχεία `sql` που συμμετέχουν στο σχήμα ώστε να επιλεγθούν για μετέπειτα επεξεργασία.

Ωστόσο, για να δημιουργηθούν όλα τα προαναφερθέντα παράθυρα απαιτούν ορισμένες μεθόδους και λειτουργίες τις οποίες προσφέρουν κλάσεις του πακέτου `guicommon`. Το

πακέτο αυτό αποτελεί τον τρίτο πυλώνα στον οποίο στηρίζεται το εργαλείο Insight καθώς προσφέρει αναγκαίες μεθόδους για την δημιουργία των βασικών παραθύρων του εργαλείου. Επιπρόσθετα, κλάσεις του πακέτου αυτού προσφέρουν styling στο εργαλείο για πιο όμορφη παρουσίαση στον χρήστη. Παράδειγμα αυτού του styling αποτελούν το χρώμα και το σχήμα των buttons στο παράθυρο της λειτουργίας αναζήτησης και αντικατάστασης.

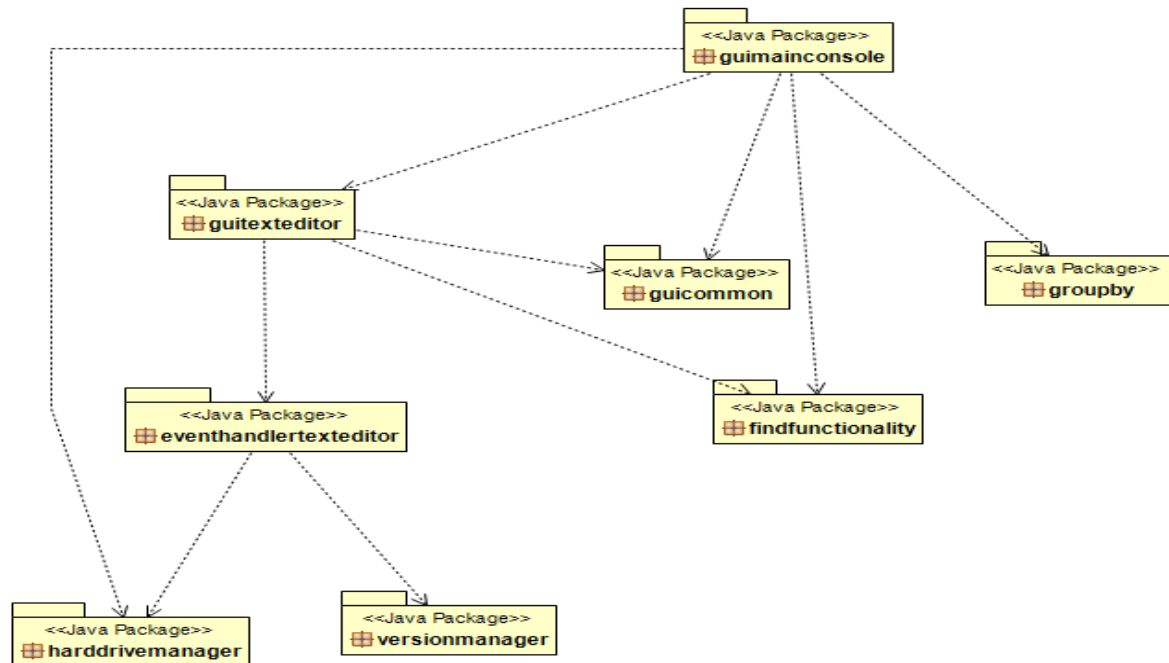
Έπειτα, σημαντικό ρόλο διαδραματίζουν οι κλάσεις των πακέτων `findfunctionality` και `groupby` καθώς προμηθεύουν το εργαλείο με τις λειτουργίες αναζήτησης/αντικατάστασης και ομαδοποίησης αντίστοιχα. Τα πακέτα αποτελούν τον τέταρτο πυλώνα του εργαλείου και έχουν σχέση συσχέτισης με τα πακέτα `guimainconsole` και `guitexteditor`. Ωστόσο, το πακέτο `groupby` έχει σχέση συσχέτισης μόνο με το πακέτο `guimainconsole` καθώς η λειτουργία `group by` χρησιμοποιείται μόνο στο εργαλείο Insight και όχι στον ενσωματωμένο editor.

Προχωρώντας, πέμπτος πυλώνας στον οποίο στηρίζεται το εργαλείο Insight είναι το πακέτο `eventhandlertexteditor`. Το πακέτο αυτό προμηθεύει τον χειρισμό όλων των γεγονότων που συμβαίνουν κατά την διάρκεια της επεξεργασίας των sql αρχείων από τον editor. Γεγονότα όπως αποθήκευση του αρχείου, επαναφορά γραμμένων στοιχείων (`undo typing`), ολική εκκαθάριση του text panel του editor και έξοδος από τον editor μπορούν να διαχειριστούν με ασφάλεια από τις κλάσεις του πακέτου `eventhandlertexteditor`. Το στοιχείο αυτό καθιστά το πακέτο αυτό ως ένα από τα πακέτα ζωτικής σημασίας για το εργαλείο, καθώς η διαχείριση των γεγονότων του editor προσδίδει ένα αίσθημα ασφάλειας και σιγουριάς ότι η ροή ελέγχου στο εργαλείο Ενόραση ακολουθεί την προκαθορισμένη πορεία, όπως άλλωστε αποτυπώνεται στο σχήμα 20.

Τέλος, έκτος και τελευταίος πυλώνας στον οποίο βασίζεται το εργαλείο Ενόραση αποτελούν τα πακέτα κλάσεων `harddrivemanager` και `versionmanager`. Το πακέτο `harddrivemanager` προσφέρει τόσο την λειτουργία της αποθήκευσης των περιεχομένων του text panel στο αρχείο sql όσο και την λειτουργία της φόρτωσης των επιλεγμένων αρχείων. Το πακέτο `versionmanager` προσφέρει την λειτουργία της τοπικής αποθήκευσης των περιεχομένων του text panel του editor, στην οποία εδράζονται η αποθήκευση των περιεχομένων στο αρχείο αλλά η και επαναφορά γραμμένων στοιχείων (`roll back`).

Όλα τα προαναφερθέντα πακέτα και οι μεταξύ τους συσχετίσεις που αναλύθηκαν σε προηγούμενες παραγράφους αποτυπώνονται στο σχήμα 27.

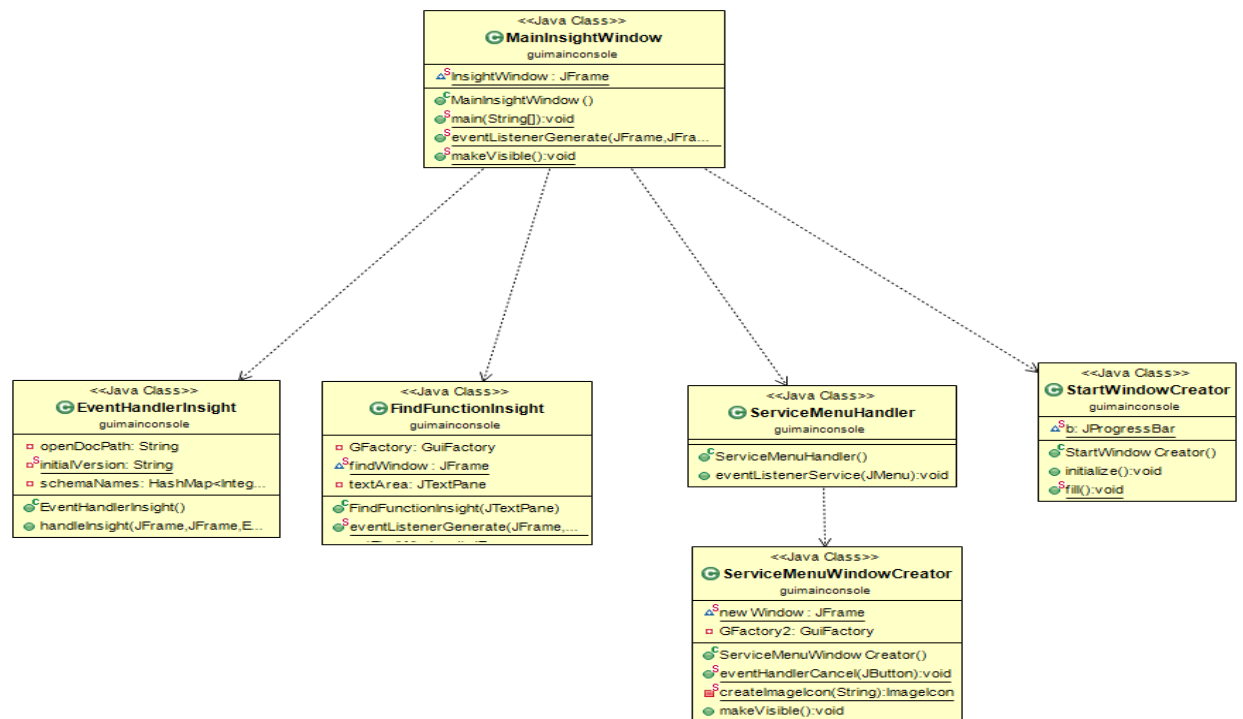
Σχήμα 27. Πακέτα και συσχετίσεις του εργαλείου Ενόραση



4.2.2 Σχεδίαση κλάσεων και λειτουργιών

Όπως ακριβώς περιεγράφηκαν τα πακέτα του εργαλείου Insight, ακριβώς με την ίδια ακολουθία θα περιγράφουμε και οι κλάσεις και λειτουργίες του κάθε πακέτου. Αρχικά, το πακέτο `guimainconsole` απαρτίζεται από έξι κλάσεις όπου η ανατιθέμενη λειτουργία τους είναι η δημιουργία του κεντρικού παραθύρου της εφαρμογής αλλά και χρήσιμων βοηθητικών παραθύρων όπως είναι το αρχικό παράθυρο που εμφανίζεται στο χρήστη, το παράθυρο που επιτελεί την λειτουργία αναζήτησης λέξεων αλλά και το παράθυρο του `service` και πληροφοριών. Ωστόσο, το πακέτο αυτό διαθέτει και μία κλάση της οποίας η λειτουργία είναι ο χειρισμός και αντιμετώπιση των γεγονότων που θα λάβουν χώρα κατά την διάρκεια της χρήσης του εργαλείου. Η κλάση αυτή ονομάζεται `eventHandlerInsight` και διαθέτει μία μέθοδο η οποία είναι υπεύθυνη για τον χειρισμό των γεγονότων. Η δομή και οι σχέσεις μεταξύ των κλάσεων εντός του πακέτου `guimainconsole` παρουσιάζονται στο σχήμα 28.

Σχήμα 28. Πακέτο guimainconsole και κλάσεις



Όπως παρουσιάζεται και στο σχήμα 28, για την δημιουργία και εμφάνιση του παραθύρου service and information απαιτείται ένα αντικείμενο της κλάσης ServiceMenuHandler. Ο σκοπός του αντικειμένου αυτού είναι αφενός να δημιουργήσει το αντικείμενο κλάσης ServiceMenuWindowCreator και αφετέρου να το εμφανίσει ώστε να καταστεί διαθέσιμο στον χρήστη. Έπειτα, το κεντρικό παράθυρο της εφαρμογής δημιουργείται από την κλάση MainInsightWindow. Η κλάση αυτή περιέχει και την μέθοδο main του εργαλείου η οποία σηματοδοτεί την έναρξη λειτουργίας του εργαλείου με την δημιουργία και παρουσίαση του του κεντρικού παραθύρου. Όλες οι υπόλοιπες κλάσεις του πακέτου καλούνται από την κλάση MainInsightWindow το οποίο άλλωστε φανερώνεται και από το UML διάγραμμα του πακέτου.

Έπειτα, εντός του πακέτου εντοπίζεται η κλάση StartWindowCreator. Ο σκοπός ύπαρξης της κλάσης αυτής είναι ένας και μοναδικός, η δημιουργία και παρουσίαση του πρώτου παραθύρου στον χρήστη το οποίο αποτελείται από το logo της εφαρμογής και μία μπάρα πρόοδου (progress bar). Η μπάρα προόδου στο αρχικό παράθυρο έχει ρυθμιστεί στα τρία δευτερόλεπτα δίνοντας στον χρήστη αρκετό χρόνο να προετοιμαστεί για την χρήση του εργαλείου.

Σημείο αναφοράς του πακέτου αυτού αποτελεί η κλάση `EventHandlerInsight`. Η σημασία της κλάσης αυτής είναι ανάλογη της σημασίας της κλάσης `MainInsightWindow`, καθώς προσφέρει τον χειρισμό όλων των γεγονότων που θα συμβούν κατά την διάρκεια χρήσης του εργαλείου `Insight`. Στη συνέχεια, ακολουθούν ορισμένα παραδείγματα χειρισμού γεγονότων όπως για παράδειγμα ο χειρισμός του `load` γεγονότος, της εξόδου από το εργαλείο, την επιλογή και επεξεργασία των `sql` αρχείων και τέλος την ομαδοποίηση των περιεχόμενων σύμφωνα με κάποια από τις διαθέσιμες επιλογές.

Αναλυτικά θα παρουσιαστούν τα προαναφερθέντα γεγονότα και ο αντίστοιχός χειρισμός τους:

- **Load log file:** αρχικά για την εκπλήρωση της φόρτωσης του `log file` στην κεντρικό `text panel`, στο δευτερεύον `text panel` αλλά και των `sql` αρχείων του σχήματος, ορίζονται και αρχικοποιούνται τα αντικείμενα `ldFile` της κλάσης `FileLoaderController`, `bufReader` της κλάσης `BufferedReader` αλλά και ορισμένες `arraylist` που κρίνονται αναγκαίες. Η αρχικοποίηση αυτή απεικονίζεται στο σχήμα 29.

Σχήμα 29. Αρχικοποίηση αναγκαιών αντικειμένων για φόρτωση `log file`

```
// -----  
schemaNames = new HashMap<Integer, String>();  
FileLoaderController ldFile = new FileLoaderController();  
  
// necessary object for creating grouping handlers  
GroupFactory groupFactory = new GroupFactory();  
GroupByFileHandler groupByFileHandler = (GroupByFileHandler) groupFactory.createGroupHandlers("File");  
GroupErrorHandler groupErrorHandler = (GroupErrorHandler) groupFactory.createGroupHandlers("Error");  
  
// filling schema Names into a JList -----  
BufferedReader bufReader = new BufferedReader(new StringReader(ldFile.load()));  
openDocPath = ldFile.getName(); // this variable contains the path of the chosen log file  
  
String line;  
int schNumber = 1; // counting the schema sql files  
ArrayList<String> minorContents = new ArrayList<String>();  
ArrayList<String> majorContents = new ArrayList<String>();  
minorContents = ldFile.getMinorContents();
```

Έπειτα, δοθέντος αυτών των αντικειμένων και δομών δεδομένων επιτελείται η φόρτωση των περιεχομένων του `log file` ξεχωρίζοντας τα ονόματα των `sql` αρχείων και τοποθέτηση τους στο `HashMap schemaNames`, η φόρτωση γραμμή – γραμμή των σημαντικών (`major`) σφαλμάτων στην κεντρικό `text panel` χρησιμοποιώντας την `arraylist majorContents` και αντίστοιχα επιτελείται και η φόρτωση των λιγότερο σημαντικών (`minor`) σφαλμάτων στο δευτερεύον δεξιά

text panel του κεντρικού παραθύρου. Ο κώδικας που εκτελεί όλα τα παραπάνω αναφορικά με την φόρτωση αναπαρίσταται στο σχήμα 30.

Σχήμα 30. Load log file Coding

```
try {
    while( (line = bufReader.readLine()) != null )
    {

        int fileIndex = line.indexOf("File:");
        int lineIndex = line.indexOf("Line:");

        if (fileIndex != -1 && lineIndex != -1) {
            String schemaName = line.substring(fileIndex + ("File:").length() + 1, lineIndex);

            boolean flag = false;
            for (int name: schemaNames.keySet()){
                String value = schemaNames.get(name);
                if (value.equals(schemaName)) {
                    flag = true;
                    break;
                }
            }

            if (!flag) {
                schemaNames.put(schNumber, schemaName);
                schNumber ++;
            }

            flag = false;

            if (line.contains("major")) {
                majorContents.add(line);
            }
        }
    }
} catch (IOException e1) {
    e1.printStackTrace();
}

DefaultListModel<String> listModel = new DefaultListModel<>();

for (int name: schemaNames.keySet()){
    String value = schemaNames.get(name);
    listModel.addElement(value);
}

schemaList.setModel(listModel);
// -----

// -----
StyledDocument styledDoc = mainTArea.getStyledDocument();
mainTArea.setText("");

for (String lineMajor : majorContents) {
    try {
        styledDoc.insertString(styledDoc.getLength(),lineMajor + "\n\n", null);
    }catch (Exception ex) {
        System.out.println(ex);
    }
}

initialVersion = mainTArea.getText(); // contents of initial version after load
StyledDocument styledDocSecond = secondaryArea.getStyledDocument();

for (String lineMinor : minorContents) {
    try {
        styledDocSecond.insertString(styledDocSecond.getLength(),lineMinor + "\n", null);
        styledDocSecond.insertString(styledDocSecond.getLength(),"-----" + "\n", null);
    }catch (Exception ex) {
        System.out.println(ex);
    }
}

// -----
```

- Exit from Insight tool: αρχικά, για να μεταβούμε στην κατάσταση εξόδου στο εργαλείο Insight αρκεί να πιάσουμε το πλήκτρο Escape ή να διαλέξουμε το τελευταίο εικονίδιο από την μπάρα μενού. Ο χειρισμός του γεγονότος εξόδου υλοποιείται σε πρώτο στάδιο ζητώντας τελική άδεια από τον χρήστη και αν η απάντηση είναι καταφατική τότε αφενός αποκρύπτει το κεντρικό παράθυρο από τον χρήστη και αφετέρου καλείται η μέθοδος exit του συστήματος. Ο κώδικας που υλοποιεί την λειτουργία αυτή απεικονίζεται στο σχήμα 31.

Σχήμα 31. Exiting from Insight Coding

```
btnCancel.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (JOptionPane.showConfirmDialog(null, "Are you sure you want to exit the Insight application?", "Exit Insight" , 0) == JOptionPane.YES_OPTION){
            InsightWindow.setVisible(false);
            System.exit(0);
        }
    }
});
```

- Select and edit a Sql file: όπως έχει προαναφερθεί για την επιλογή και επεξεργασία ενός sql αρχείου υφίστανται δύο διαφορετικοί τρόποι στο εργαλείο. Ο πρώτος ξεκινά από την λίστα αρχείων που βρίσκεται στο δεξί κάτω μέρος του κεντρικού παραθύρου, και εκτελώντας δεξί κλικ επάνω στο επιλεγθέν sql αρχείο ο editor εμφανίζεται με τα περιεχόμενα του sql αρχείου ήδη φορτωμένα. Επομένως, από τα παραπάνω κατανοούμε ότι με δεξί κλικ στο επιλεγθέν sql αρχείο θα εμφανίζεται ένα pop up menu με τις διαθέσιμες επιλογές και αν πατηθεί το πλήκτρο Edit του μενού αυτού τότε να ανοίγει ο editor. Ο κώδικας που εκτελεί την παραπάνω διαδικασία παρουσιάζεται στο σχήμα 32.

Σχήμα 32. Α' τρόπος επεξεργασίας sql αρχείου

```
// handle right click on a file in the schema list, show pop up menu
schemaList.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        if (SwingUtilities.isRightMouseButton(e) && e.getClickCount() == 1) { // right click on any file and the editor opens

            // create the pop up menu
            JPopupMenu popupMenu = new JPopupMenu();
            JMenuItem btnEditItem = new JMenuItem("Edit File");
            JMenuItem btnExitItem = new JMenuItem("Cancel");
            popupMenu.add(btnEditItem);
            popupMenu.addSeparator();
            popupMenu.add(btnExitItem);
            popupMenu.show(e.getComponent(), e.getX(), e.getY());
            String selected = schemaList.getSelectedValue();

            // if this gets selected then the editor opens up with the selected file loaded
            btnEditItem.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    MainEditorWindow editor = new MainEditorWindow(selected,openDocPath);
                }
            });
        }
    }
});
```

Ο δεύτερος τρόπος που λαμβάνει χώρα στο εργαλείο παρουσιάζεται στο σχήμα 33.

Σχήμα 33. Β' τρόπος επεξεργασίας sql αρχείου

```
// handle edit button by showing list with the schema files
btnEdit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        edw.setSchemaNames(schemaNames);
        edw.setSchemaPath(openDocPath);
        edw.makeVisible();
        edw.eventListenerGenerate(edw.getEditWindow(), openDocPath, edw.getNames(), edw.getCancelB());
    }
});
```

Για την υλοποίηση του στόχου φόρτωσης των sql αρχείων σε αυτή την περίπτωση αρχικά καλείται η μέθοδος `setSchemaNames` η οποία τοποθετεί τα ονόματα των sql αρχείων στην λίστα που εμφανίζεται, έπειτα ακολουθεί η εκτέλεση της μεθόδου `setSchemaPath` όπου τοποθετεί το μονοπάτι που βρίσκονται το log file και τα sql αρχεία, στο πεδίο `schemaPath` της κλάσης `EditWindow` και τέλος εμφανίζεται το παράθυρο με την λίστα των αρχείων ακολουθούμενο από την κλήση της μεθόδου `eventListenerGenerate` για τον χειρισμό των γεγονότων του `EditWindow`.

- **Grouping by Error or by File:** στο εργαλείο *Insight* προσφέρονται δύο ειδών ομαδοποιήσεις των περιεχομένων του log file, ομαδοποίηση με γνώμονα τα όμοια σφάλματα που περιέχονται στο ίδιο αρχείο ή με γνώμονα τα όμοια σφάλματα που υπάρχουν σε διαφορετικά αρχεία. Με την επιλογή της κάθε είδους ομαδοποίησης καλείται η αντίστοιχη μέθοδος που χειρίζεται το γεγονός αυτό. Το κάλεσμα των μεθόδων που χειρίζονται τις ομαδοποιήσεις αναπαρίσταται στο σχήμα 34.

Σχήμα 34. Group by callings

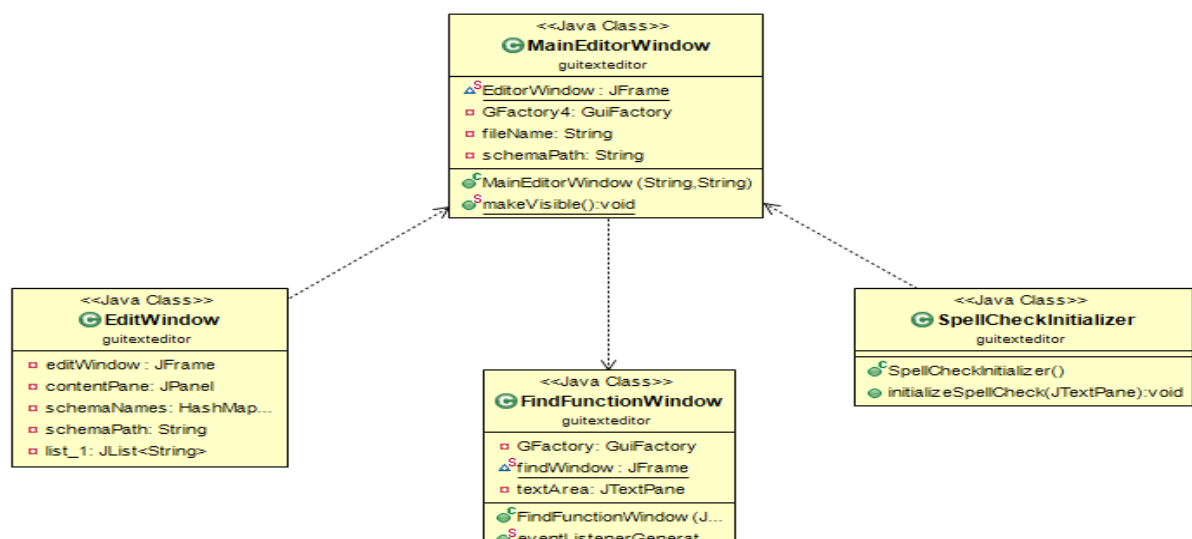
```
// grouping by Error
groupError.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        groupErrorHandler.writeGroupOutput(mainTArea, groupErrorHandler.handleGroupError(initialVersion));
    }
});

// group by file
groupFile.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        groupByFileHandler.writeGroupOutput(mainTArea, groupByFileHandler.handleGroupError(initialVersion));
    }
});
```


Τελευταία κλάση του πακέτου αυτού είναι η κλάση `FindFunctionInsight`. Όπως αναφέρει και ο τίτλος της κλάσης, ο σκοπός ύπαρξης της κλάσης αυτής είναι η δημιουργία του παραθύρου αναζήτησης λέξεων. Το παράθυρο αυτό έχει ορισμένες διαφορές από το παράθυρο αναζήτησης/αντικατάστασης του editor καθώς στο εργαλείο Insight δεν επιτρέπονται αντικαταστάσεις ή διαγραφές λέξεων. Οι ενέργειες αυτές επιτρέπονται μόνο μέσα στον editor. Παρόλο που το παράθυρο αναζήτησης δημιουργείται μέσα στην κλάση αυτή, ο χειρισμός των γεγονότων που θα συμβούν κατά την διάρκεια της αναζήτησης λέξεων βρίσκεται μέσα στο πακέτο `findfunctionality`.

Έπειτα, επόμενο πακέτο το οποίο χρήζει ανάλυσης είναι το πακέτο `guitexteditor`. Το πακέτο αυτό είναι υπεύθυνο για την δημιουργία του κεντρικού παραθύρου του editor καθώς και των βοηθητικών παραθύρων στα οποία συγκαταλέγονται το παράθυρο `edit` που προσφέρει τα ονόματα των αρχείων του σχήματος της βάσης τα οποία είναι διαθέσιμα για επεξεργασία, και το παράθυρο `findFunctionWindow` το οποίο διεκπεραιώνει την διεργασία αναζήτησης/αντικατάστασης λέξεων στον editor. Η δομή και οι σχέσεις μεταξύ των κλάσεων απεικονίζονται στο σχήμα 35.

Σχήμα 35. Πακέτο `guitexteditor` και κλάσεις



Ωστόσο, στο πακέτο αυτό δεν υφίστανται μόνο κλάσεις που δημιουργούν παράθυρα. Ο λόγος γίνεται για την κλάση `SpellCheckInitializer`. Η κλάση αυτή είναι υπεύθυνη για την αρχικοποίηση της λειτουργίας συντακτικού ελέγχου των λέξεων που πληκτρολογούνται στον editor. Ο συντακτικός έλεγχος βασίζεται στο αγγλικό λεξιλόγιο το οποίο δίνεται ως

όρισμα στην μέθοδο registerDictionaries. Η δομή της κλάσης αυτής παρουσιάζεται στο σχήμα 36.

Σχήμα 36. Δομή της κλάσης SpellCheckInitializer

```
public class SpellCheckInitializer {  
    public void initializeSpellCheck(JTextPane textArea) {  
        SpellChecker.setUserDictionaryProvider(new FileUserDictionary());  
        SpellChecker.registerDictionaries(MainEditorWindow.class.getResource("/dictionary"), "en");  
        SpellChecker.register(textArea);  
        SpellCheckerOptions sco=new SpellCheckerOptions();  
        sco.setCaseSensitive(true);  
        sco.setSuggestionsLimitMenu(15);  
  
        JPopupMenu popup = SpellChecker.createCheckerPopup(sco);  
        textArea.setComponentPopupMenu(popup);  
    }  
}
```

Η ακολουθία των ενεργειών για την αρχικοποίηση και χρήση του ελεγκτή συντακτικής ορθότητας αναλύεται ως εξής: αρχικά η μέθοδος initializeSpellCheck λαμβάνει ως όρισμα το κεντρικό text panel του editor στο οποίο θα δρα ο ελεγκτής ορθότητας. Ακολουθεί, η προσθήκη του λεξικού με βάση το οποίο γίνεται εφικτός ο έλεγχος της κάθε λέξης του editor. Επιπρόσθετα, ενεργοποιούνται ορισμένες επιλογές όπως ο περιορισμός των εναλλακτικών προσφερόμενων λέξεων για αντικατάσταση κάποιας λανθασμένης λέξης σε δεκαπέντε επιλογές και η λειτουργία Case Sensitive. Τέλος, με το πάτημα του δεξί κλικ επάνω σε οποιαδήποτε συντακτικά λανθασμένη λέξη εμφανίζεται ένα pop-up menu, το οποίο προμηθεύει ενέργειες όπως τις επιλογές για αντικατάσταση της λανθασμένης λέξης και την εισαγωγή στο λεξικό (Add to dictionary).

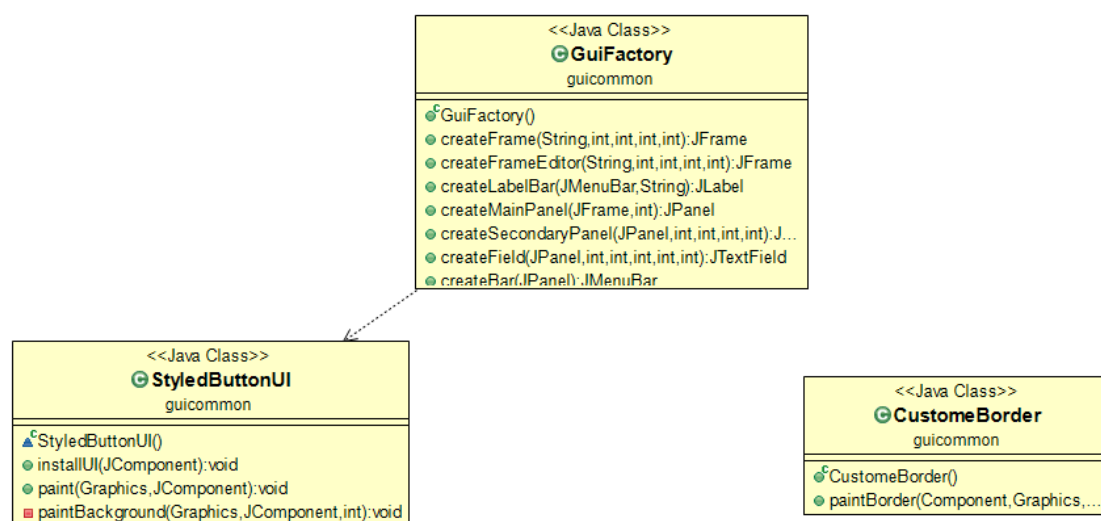
Το κεντρικό παράθυρο του editor δημιουργείται από την κλάση MainEditorWindow. Η κλάση αυτή δημιουργεί όλα τα αλλά αντικείμενα τα οποία χρειάζονται για την ολοκλήρωση της επεξεργασίας sql αρχείων. Επιπρόσθετα, η κλάση αυτή διαθέτει μια μέθοδο με όνομα makeVisible η οποία είναι υπεύθυνη για την εμφάνιση του παραθύρου του editor μόλις αυτό δημιουργηθεί.

Τέλος, το πακέτο guitexteditor συμπληρώνει η κλάση FindFunctionWindow. Η κλάση αυτή δημιουργεί το παράθυρο για την λειτουργία αναζήτησης/ αντικατάστασης και διαγραφής. Το παράθυρο αυτό, όπως έχει προαναφερθεί είναι ελαφρώς διαφορετικό από το παράθυρο αναζήτησης του Insight εργαλείου καθώς τώρα στον editor προσφέρονται οι λειτουργίες αντικατάστασης και διαγραφής των στιγμιότυπων της αναζητούμενης λέξης. Όπως και προηγουμένως, παρόλο που δημιουργείται το παράθυρο αναζήτησης και αντικατάστασης, τα γεγονότα που θα συμβούν κατά την διάρκεια

χρήσης του θα διαχειριστούν από τις κλάσεις του πακέτου findfunctionality, το οποίο θα αναλυθεί στη συνέχεια.

Ωστόσο τα πακέτα guimainconsole και guitexteditor διαθέτουν κοινά στοιχεία και μεθόδους κυρίως όσον αφορά την δημιουργία παραθύρων. Τα στοιχεία αυτά εξήχθησαν σε κλάσεις του πακέτου guicommon. Η δομή και οι κλάσεις του πακέτου απεικονίζονται στο σχήμα 37.

Σχήμα 37. Πακέτο guicommon και κλάσεις



Το πακέτο αυτό αποτελείται από τρεις κλάσεις, την guiFactory, την StyledButtonUi και την CustomeBorder. Η κλάση guiFactory προμηθεύει το εργαλείο Insight με μεθόδους ζωτικής σημασίας για την δημιουργία των παραθύρων όπως για παράδειγμα οι μέθοδοι δημιουργίας των frames, των menus, των text panels αλλά και μεθόδων που εισάγουν εικονίδια στα βασικά μενού. Επιπρόσθετα, περιέχει μεθόδους οι οποίες δημιουργούν πιο πολύπλοκα παράθυρα που περιέχουν κάθετες και οριζόντιες γραμμές διαχωρισμού των στοιχείων του παραθύρου όπως για παράδειγμα το παράθυρο service and information.

Σε αυτό το σημείο θα αναλυθούν ορισμένες βασικές μέθοδοι της κλάσης guiFactory, οι οποίες συνεργάζονται μεταξύ τους με σκοπό την παραγωγή των στοιχείων που συμμετέχουν στα παράθυρα του εργαλείου. Στο σχήμα 38 απεικονίζονται δύο βασικές μέθοδοι, η μέθοδος createBar η οποία είναι υπεύθυνη για την δημιουργία της μπάρας μενού που χρησιμοποιείται και στο κεντρικό παράθυρο του Insight αλλά και στον

ενσωματωμένο editor. Η μέθοδος αυτή δημιουργεί την μπάρα μενού και έπειτα την ενσωματώνει στο κεντρικό panel. Δεύτερη μέθοδος που απεικονίζεται στο σχήμα 38 είναι η μέθοδος createMenu. Η μέθοδος έχει επωμιστεί την δημιουργία των μενού ολόκληρης της εφαρμογής. Η μέθοδος δημιουργεί το μενού, του προσθέτει κατάλληλο tooltip text ώστε να προσφέρει ενημερωτικό μήνυμα της λειτουργίας που προσφέρει το κάθε μενού και τέλος του τοποθετεί το κατάλληλο εικονίδιο διαμέσου της μεθόδου setImage. Η ανάλυση της μεθόδου setImage έπεται.

Σχήμα 38. Μέθοδοι createBar και createMenu

```
public JMenuBar createBar(JPanel mainPanel){
    JMenuBar bar = new JMenuBar();
    bar.setBounds(0,0, 795, 23);
    bar.setSize(795, 23);
    mainPanel.add(bar);
    return bar;
}

public JMenu createMenu(JMenuBar bar,String text,String ImageName){
    JMenu menu = new JMenu();
    menu.setToolTipText(text);
    setImage(menu,ImageName);
    bar.add(menu);
    return menu;
}
```

Στην κλάση guiFactory, εντοπίζεται και η μέθοδος setImage. Όπως προδίδει ο τίτλος της μεθόδου, ο ρόλος της είναι η προσθήκη εικονιδίων στα διάφορα menu που χρησιμοποιούνται στο εργαλείο. Η μέθοδος λαμβάνει ως ορίσματα το menu που θα τοποθετηθεί το εικονίδιο και το μονοπάτι του εικονιδίου αυτού. Η δομή της παρουσιάζεται στο σχήμα 39.

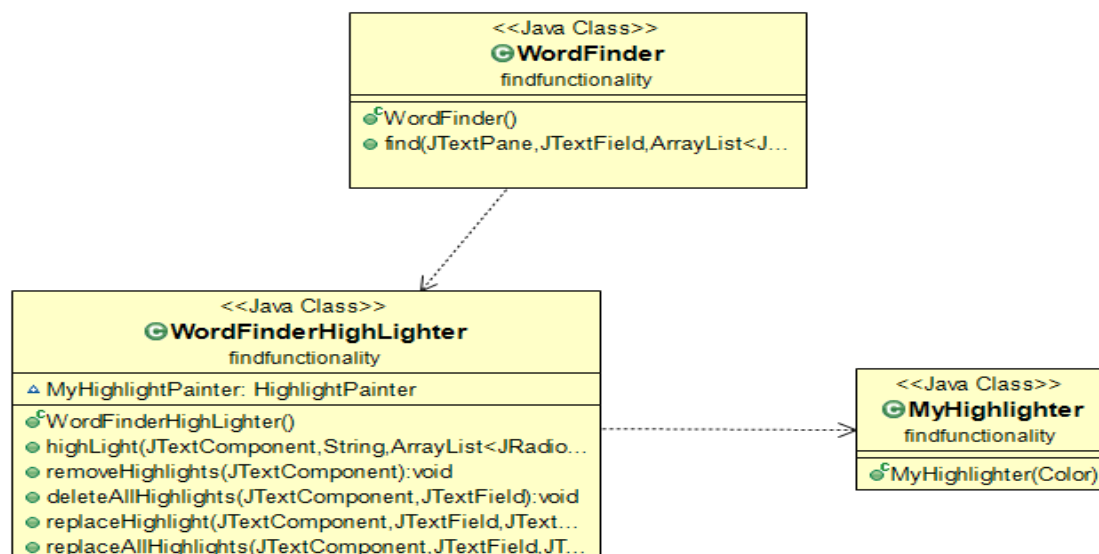
Σχήμα 39. Η μέθοδος setImage

```
public void setImage(JMenu menu,String ImageName){
    try {
        Image img = ImageIO.read(getClass().getResource(ImageName));
        menu.setIcon(new ImageIcon(img));
    } catch (Exception ex) {
        System.out.println(ex);
    }
}
```

Το πακέτο guicommon συμπληρώνουν οι κλάσεις StyledButtonUI και CustomeBorder. Ο ρόλος που διαδραματίζει η κλάση StyledButtonUI στην εφαρμογή είναι να προσδώσει επιπρόσθετο styling στα κουμπιά του παραθύρου findfunction ώστε ως αποτέλεσμα να είναι αισθητικά πιο όμορφα. Άλλωστε η δράση της μεθόδου αυτής παρατηρείται από τα παράθυρα FindFunctionWindow και FindFunctionInsight όπου και βρίσκονται κουμπιά στρογγυλού σχήματος και χρώματος RGB(31,190,214). Τέλος, η κλάση CustomeBorder κατασκευάζει τα αντίστοιχα όρια των κουμπιών ώστε να διαθέτουν ίδιο χρώμα και σχήμα με τα κουμπιά των find παραθύρων.

Το πακέτο που θα αναλυθεί στη συνέχεια είναι το πακέτο findfunctionality που προμηθεύει την λειτουργία αναζήτησης, αντικατάστασης και διαγραφής των στιγμιότυπων των αναζητούμενων λέξεων. Το πακέτο αυτό αποτελείται από τρεις κλάσεις οι οποίες ονομάζονται WordFindHighLighter, WordFinder και MyHighlighter. Η εσωτερική δομή του πακέτου καθώς και οι σχέσεις μεταξύ των κλάσεων παρουσιάζονται στο σχήμα 40.

Σχήμα 40. Πακέτο findfunctionality και κλάσεις



Το πακέτο αυτό όπως αποτυπώνεται και από το UML διάγραμμα του σχήματος 40 αποτελείται από τρεις κλάσεις. Αρχικά, στην κλάση WordFinder έχει ανατεθεί ανεπίσημα ένας ρόλος factory. Με τον όρο factory class εννοείται μια κλάση η οποία δημιουργεί αντικείμενα που θα χρειαστούν για την λειτουργία του συστήματος. Η δημιουργία αυτών των αντικειμένων έχει εξαχθεί από την κλάση EventHandlerInsight ώστε να βελτιωθεί ο έλεγχος σωστής δημιουργίας των αντικειμένων. Η κλάση WordFinder αφού

δημιουργήσει το αντικείμενο της κλάσης WordFindHighLighter, καλεί την μέθοδο highlight η οποία είναι η βασική μέθοδος της κλάσης αυτής.

Έπειτα, η κλάση MyHighlighter επεκτείνει την κλάση DefaultHighlightPainter του πακέτου DefaultHighlighter με μοναδικό σκοπό την αρχικοποίηση του highlighting με χρώμα κόκκινο. Η κλάση αυτή συσχετίζεται με την κλάση WordFindHighLighter, αφού πριν την έναρξη οποιουδήποτε highlighting η κλάση αρχικοποιεί το κόκκινο χρώμα ως το χρώμα με το οποίο θα παρουσιάζονται τα ευρεθέντα στιγμιότυπα των αναζητούμενων λέξεων.

Τέλος, η πιο σημαντική κλάση του πακέτου είναι η κλάση WordFindHighLighter. Η κλάση αυτή παρέχει στο εργαλείο Insight όλες τις λειτουργίες που συσχετίζονται με τα find function windows, όπως η αντικατάσταση ή διαγραφή ενός ή όλων των στιγμιότυπων των αναζητούμενων λέξεων, η εύρεση φυσικά των λέξεων και τέλος η μέθοδος που αφαιρεί όλα τα προηγούμενα highlights προτού αναζητήσουμε την επόμενη λέξη.

Σε αυτό το σημείο θα παρουσιαστεί η δομή ορισμένων βασικών μεθόδων της κλάσης WordFindHighLighter. Αναλυτικά:

- 1) Μέθοδος removeHighlights: η συγκεκριμένη μέθοδος έχει επωμιστεί την λειτουργία της αφαίρεσης όλων των προηγούμενων στιγμιότυπων της λέξης που αναζητήθηκε ώστε να μην εμφανιστούν και τα αποτελέσματα αναζήτησης που αφορούν προηγούμενες ευρέσεις λέξεων. Η δομή της μεθόδου αυτής παρουσιάζεται στο σχήμα 41.

Σχήμα 41. Μέθοδος removeHighlights

```
/**
 * This method removes all the highlighted words before every new find.
 * @param Tcomp
 */
public void removeHighlights(JTextComponent Tcomp) {
    Highlighter hilite = Tcomp.getHighlighter();
    Highlighter.Highlight[] hilites = hilite.getHighlights();

    for (int i=0;i<hilites.length;i++) {
        if (hilites[i].getPainter() instanceof MyHighlighter) {
            hilite.removeHighlight(hilites[i]);
        }
    }
}
```

- 2) Μέθοδος `deleteAllHighlights`: όπως αναφέρει και το όνομα της, η μέθοδος αυτή είναι υπεύθυνη για την διαγραφή όλων των στιγμιότυπων της αναζητούμενης λέξης από το text panel του editor. Δοθέντος, των ορισμάτων του text panel και του text field που περιέχει την λέξη που αναζήτησε ο χρήστης η μέθοδος αυτή θα αφαιρέσει όλα τα στιγμιότυπα της λέξης. Ο τρόπος με τον οποίο αυτό επιτελείται έγκειται στο γεγονός ότι ουσιαστικά η διαγραφή λέξεων μπορεί να επιτευχθεί με την αντικατάστασή τους με το κενό σύμβολο. Η ιδέα αυτή καθώς και η υλοποίηση της απεικονίζεται στο σχήμα 42.

Σχήμα 42. Μέθοδος `deleteAllHighlights`

```
/**
 * This method deletes all the instances of the searched word inside the text panel of
 * the editor.
 * @param Tcomp
 * @param field
 */
public void deleteAllHighlights(JTextComponent Tcomp, JTextField field) {

    Highlighter hilite = Tcomp.getHighlighter();
    Highlighter.Highlight[] hilites = hilite.getHighlights();
    String contents = Tcomp.getText();
    String result="";

    for (int i=0;i<hilites.length;i++) {
        if (hilites[i].getPainter() instanceof MyHighlighter) {
            if (contents.contains(field.getText())) {
                result = contents.replace(field.getText(), "");
            }
        }
    }
    Tcomp.setText(result);
}
```

- 3) Μέθοδος `replaceAllHighlights`: η μέθοδος αυτή καθιστά εφικτή την αντικατάσταση όλων των στιγμιότυπων της λέξης που αναζητείται. Η μέθοδος λαμβάνει ως ορίσματα το text panel του editor, text field με την λέξη που αναζητούμε και τέλος text field το οποίο περιέχει την λέξη με την οποία θα αντικαταστήσουμε όλα τα ευρεθέντα στιγμιότυπα. Η δομή της μεθόδου αυτής απεικονίζεται στο σχήμα 43.

Σχήμα 43. Μέθοδος `replaceAllHighlights`

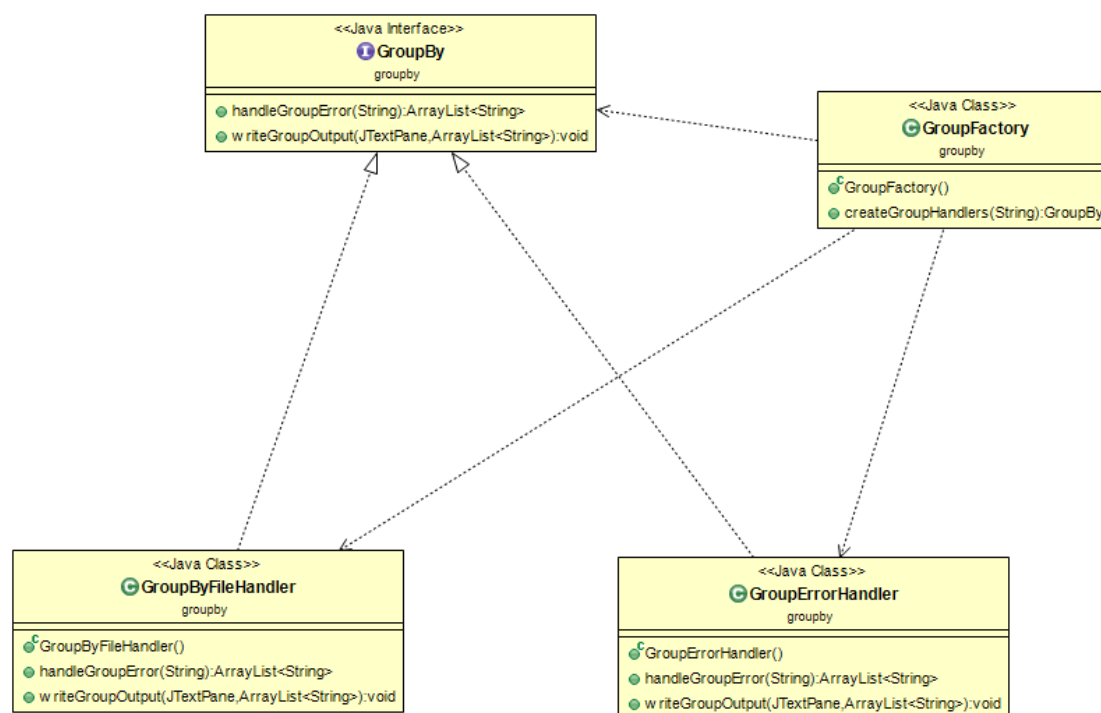
```
/**
 * This method replaces all the instances of the searched word with the input that the user gave
 * inside the Replace With text field.
 * @param Tcomp
 * @param fieldFind
 * @param fieldReplace
 */
public void replaceAllHighlights(JTextComponent Tcomp, JTextField fieldFind, JTextField fieldReplace) {

    Highlighter hilite = Tcomp.getHighlighter();
    Highlighter.Highlight[] hilites = hilite.getHighlights();
    String contents = Tcomp.getText();
    String result="";

    for (int i=0;i<hilites.length;i++) {
        if (hilites[i].getPainter() instanceof MyHighlighter) {
            if (contents.contains(fieldFind.getText())) {
                result = contents.replace(fieldFind.getText(), fieldReplace.getText());
            }
        }
    }
    Tcomp.setText(result);
}
```

Το πακέτο που θα αναλυθεί στη συνέχεια είναι το πακέτο groupby το οποίο προσφέρει στο εργαλείο Insight τις επιλογές ομαδοποίησης είτε με βάση τα όμοια λάθη στο ίδιο αρχείο είτε με βάση τα ίδια λάθη τα οποία όμως εντοπίζονται σε διαφορετικά αρχεία. Το πακέτο αυτό αποτελείται από τέσσερις κλάσεις οι οποίες προμηθεύουν τις απαραίτητες πληροφορίες. Στο πακέτο αυτό λαμβάνει χώρα και μία κλάση η οποία κατασκευάζει ένα interface το οποίο ορίζει τις απαραίτητες μεθόδους που θα πρέπει να διαθέτουν οι κλάσεις του groupby που το υλοποιούν. Η εσωτερική δομή του πακέτου και οι σχέσεις μεταξύ των κλάσεων παρουσιάζονται στο σχήμα 44.

Σχήμα 44. Πακέτο groupby και κλάσεις



Ξεκινώντας από την κλάση GroupBy η οποία εφοδιάζει το interface που υλοποιούν οι κλάσεις GroupByFileHandler και GroupErrorHandler ,η κλάση αυτή ορίζει την λειτουργικότητα που θα πρέπει να προσφέρεται από τις δύο προαναφερθείσες κλάσεις. Το interface της κλάσης απαιτεί την πραγμάτωση δύο μεθόδων, της handleGroupError και της writeGroupOutput. Η πρώτη μέθοδος ανάλογα με την κλάση που την υλοποιεί επιστρέφει μια arraylist η οποία περιέχει τα ομαδοποιημένα περιεχόμενα του log file κατά γραμμή. Επιπρόσθετα, η λειτουργία της μεθόδου writeGroupOutput είναι να γράψει όλα τα αποτελέσματα των ομαδοποιήσεων στο κεντρικό text panel του editor. Περισσότερες πληροφορίες για τις μεθόδους αυτές θα δοθούν σε επομένη παράγραφο. Η δομή της κλάσης που προσφέρει το interface απεικονίζεται στο σχήμα 45.

Σχήμα 45. Δομή της κλάσης GroupBy

```
public interface GroupBy {  
    public ArrayList<String> handleGroupError(String contents);  
    public void writeGroupOutput(JTextArea textArea, ArrayList<String> groupLines);  
}
```

Για να δημιουργηθούν όλα τα αναγκαία αντικείμενα για την λειτουργία της ομαδοποίησης, έχει οριστεί η κλάση GroupFactory που εφοδιάζει ακριβώς αυτό. Η κλάση αυτή διαθέτει μόνο μία μέθοδο η οποία δημιουργεί τα αντικείμενα των κλάσεων ομαδοποίησης. Λαμβάνει ως όρισμα μία παράμετρο αλφαριθμητικού τύπου με βάση την οποία επιτυγχάνεται η δημιουργία των αντικειμένων ως εξής: αν τα περιεχόμενα της παραμέτρου είναι ίσα με την λέξη “Error” τότε δημιουργείται αντικείμενο της κλάσης GroupErrorHandler ενώ αν τα περιεχόμενα της παραμέτρου είναι ίσα με την λέξη “File” τότε δημιουργείται αντικείμενο της κλάσης GroupByFileHandler. Σε διαφορετική περίπτωση από τις δύο προαναφερθείσες η μέθοδος επιστρέφει κενό ως ένδειξη λάθους. Η δομή της μεθόδου αλλά και της κλάσης αναπαρίσταται στο σχήμα 46.

Σχήμα 46. Δομή της κλάσης GroupFactory

```
public class GroupFactory {  
  
    /**  
     * This method creates the two group by objects. If the handlerType is equal  
     * to 'Error' then a GroupErrorHandler is created or if the handlerType is  
     * equal to 'File' then a GroupFileHandler is created.  
     * @param handlerType  
     * @return  
     */  
  
    public GroupBy createGroupHandlers(String handlerType) {  
  
        if (handlerType.equals("Error")) {  
            return new GroupErrorHandler();  
        } else if (handlerType.equals("File")) {  
            return new GroupByFileHandler();  
        } else { // this is something went wrong  
            return null;  
        }  
    }  
}
```

Σε αυτό το σημείο θα αναλυθούν οι δύο κλάσεις που υλοποιούν τα δύο είδη ομαδοποιήσεων. Αρχικά, η κλάση GroupErrorHandler ομαδοποιεί τα ίδια σφάλματα που λαμβάνουν χώρα στο ίδιο αρχείο. Ο τρόπος με τον οποίο επιτυγχάνεται αυτό εδράζεται στο γεγονός ότι είναι αναγκαίο να ελέγχουμε τα περιεχόμενα του log file για το καθένα σφάλμα και κατά πόσο το σφάλμα αυτό παρουσιάζεται περισσότερες από μία φορές στο

log file. Για την εξοικονόμηση χρόνου και επαναλήψεων, με κάθε εύρεση λάθους, η γραμμή που περιέχει το λάθος μαρκάρεται ώστε να μην επανελεγχθεί άσκοπα. Η δομή της μεθόδου αυτής παρουσιάζεται στο σχήμα 47.

Σχήμα 47. Δομή μεθόδου handleGroupError

```
try {
    while( (line = bufReader.readLine()) != null ){
        errorWordIndex = line.indexOf("Error: ");

        if (errorWordIndex != -1) {
            lineIndex = line.indexOf("Line:");
            String fileName = line.substring("File: ".length(),lineIndex);
            errorCount ++;
            errorMatches.put(line,errorCount);
            linesProcessed.put(line,"processed");
            String errorMessageString = line.substring(errorWordIndex + ("Error: ".length())); // this contains the error message

            try {
                while ((line2 = bufReader2.readLine()) != null ) {

                    if (line2.contains(errorMessageString) && line2.contains(fileName) && !errorMatches.containsKey(line2) && !linesProcessed.containsKey(line2)) {
                        errorCount++;
                        errorMatches.put(line2,errorCount);
                        linesProcessed.put(line2,"processed");
                    }
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

Όσον αφορά το αποτέλεσμα της ομαδοποίησης στην περίπτωση αυτή όπως έχει παρουσιαστεί ήδη η γραμμή θα περιέχει το όνομα του αρχείου ακολουθούμενο από τις γραμμές χωρισμένες με κόμμα στις οποίες βρίσκεται το κοινό λάθος και τέλος το μήνυμα του σφάλματος. Η λειτουργία αυτή υλοποιείται από τον κώδικα ο οποίος απεικονίζεται στο σχήμα 48.

Σχήμα 48. Group Error Results Coding

```
int indexLine = -1,indexError = -1;

if (errorMatches.size() > 1) {
    String allLines = "";
    errorCheck.put(line, true);

    for (String match : errorMatches.keySet()) {
        errorCheck.put(match, true);
        indexLine = match.indexOf("Line: ");
        indexError = match.indexOf("Error: ");
        allLines += match.substring(indexLine + "Line: ".length(),indexError)+",";
    }
    StringBuilder builder = new StringBuilder(allLines);
    builder.setCharAt(allLines.length() - 1, ' ');

    finalTextAreaContents.add("File: " + fileName + " Lines: " + builder + " " + errorMessageString);
}else { // this case catches a one time show error

    HashMap.Entry<String,Integer> entry = errorMatches.entrySet().iterator().next();

    if (!finalTextAreaContents.containsKey(entry.getKey()) && !errorCheck.containsKey(line)) {
        finalTextAreaContents.add(entry.getKey());
    }
}
```

Τελευταίο συστατικό που συμπληρώνει την κλάση GroupErrorHandler είναι η μέθοδος writeGroupOutput. Η λειτουργία της μεθόδου αυτής είναι να γράψει όλα τα αποτελέσματα της ομαδοποίησης στο κεντρικό text panel του editor. Για να γίνει εφικτό αυτό, αρχικά αφαιρεί όλα τα περιεχόμενα του text panel και έπειτα γραμμή προς γραμμή γράφει τα περιεχόμενα της arraylist στο κεντρικό text panel. Ωστόσο, η μέθοδος αυτή χρησιμοποιείται και από την κλάση GroupByFileHandler με τον ίδιο ακριβώς τρόπο. Η δομή της προβάλλεται στο σχήμα 49.

Σχήμα 49. Δομή μεθόδου writeGroupOutput

```
/**
 * This method writes to the main text panel the groups that were assembled
 * by the method handleGroupError
 */
public void writeGroupOutput(JTextPane textArea, ArrayList<String> grouplines) {
    // now setting the gathered text into the main text panel
    textArea.setText("");
    StyledDocument styledDoc = textArea.getStyledDocument();

    for (String groupLine : grouplines) {
        try {
            styledDoc.insertString(styledDoc.getLength(), groupLine + "\n\n", null);
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
}
```

Έπειτα, η κλάση GroupByFileHandler προμηθεύει την λειτουργία της ομαδοποίησης των κοινών σφαλμάτων σε διαφορετικά αρχεία. Για να το πετύχει αυτό χρησιμοποιεί την μέθοδο handleGroupError με ανάλογο τρόπο με την κλάση GroupErrorHandler και τέλος γράφει τα αποτελέσματα στο text panel διαμέσου της μεθόδου writeGroupOutput.

Αρχικά, η μέθοδος handleGroupError είναι υπεύθυνη για ολόκληρη την διαδικασία. Η ειδοποιός διαφορά με την προηγούμενη κλάση είναι ότι τώρα αναζητούμε να βρούμε το κοινό λάθος ανάμεσα σε διαφορετικά αρχεία. Επομένως, διατηρούμε ένα HashMap με όνομα errorMatches στο οποίο βάζουμε όλες τις γραμμές που περιέχουν το αναζητούμενο λάθος. Όπως και στην προηγούμενη κλάση μαρκάρουμε τις γραμμές που έχουμε ελέγξει ώστε να εξοικονομηθεί χρόνος και άσκοπες επαναλήψεις κώδικα. Οι προαναφερθείσες ενέργειες παρουσιάζονται στο σχήμα 50.

Σχήμα 50. Μέθοδος handleGroupError κλάσης GroupByFileHandler

```
try {
    while( (line = bufReader.readLine()) != null ){
        errorWordIndex = line.indexOf("Error: ");

        if (errorWordIndex != -1) {
            lineIndex = line.indexOf("Line:");
            errorCount ++;
            errorMatches.put(line,errorCount);
            linesProcessed.put(line,"processed");
            String errorMessageString = line.substring(errorWordIndex + ("Error: ".length())); // this contains the error message

            try {
                while ((line2 = bufReader2.readLine()) != null ) {

                    if (line2.contains(errorMessageString) && !errorMatches.containsKey(line2) && !linesProcessed.containsKey(line2)) {
                        errorCount++;
                        errorMatches.put(line2,errorCount);
                        linesProcessed.put(line2,"processed");
                    }
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

Τέλος, όπως και προηγουμένως παράγουμε την arraylist που θα περιέχονται τα αποτελέσματα της ομαδοποίησης ως εξής: η γραμμή θα ξεκινά με την λέξη Files και θα ακολουθείται από όλα τα ονόματα των αρχείων στα οποία έχει εντοπιστεί το σφάλμα χωρισμένα με κόμμα, και τέλος ακολουθεί το μήνυμα του λάθους. Στην περίπτωση όπου κάποιο σφάλμα βρίσκεται μόνο σε ένα αρχείο τότε απλά τοποθετείται στην arraylist η γραμμή του log file αυτούσια. Η λειτουργία αυτή παρουσιάζεται στο σχήμα 51.

Σχήμα 51. Κώδικας αποθήκευσης ομαδοποιημένων δεδομένων σε arraylist

```
// Found an error across many files
if (errorMatches.size() > 1) {
    String allFiles = "";
    int indexFile= -1, indexLine = -1;
    errorCheck.put(line, true);

    for (String match : errorMatches.keySet()) {
        errorCheck.put(match, true);
        indexFile = match.indexOf("File: ");
        indexLine = match.indexOf("Line: ");

        if (!allFiles.contains(match.substring(indexFile + "File: ".length(),indexLine))) {
            allFiles += match.substring(indexFile + "File: ".length(),indexLine)+",";
        }
    }

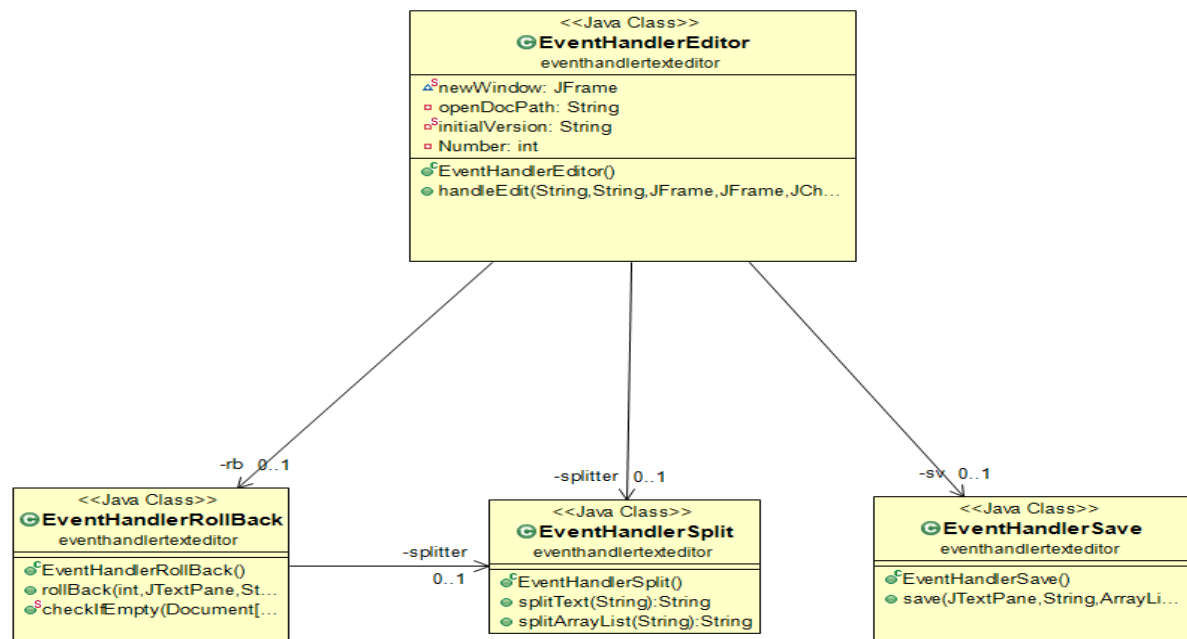
    StringBuilder builder = new StringBuilder(allFiles);
    builder.setCharAt(allFiles.length() - 1, ' ');
    finalTextAreaContents.add("Files: " + builder + " " + "Error: " + errorMessageString);
} else { // this case catches a one time show error

    HashMap.Entry<String,Integer> entry = errorMatches.entrySet().iterator().next();

    if (!finalTextAreaContents.contains(entry.getKey()) && !errorCheck.containsKey(line)) {
        finalTextAreaContents.add(entry.getKey());
    }
}
```

Όπως έχει ήδη παρουσιαστεί, το πακέτο `guitexteditor` είναι υπεύθυνο για την δημιουργία των βασικών παραθύρων του editor όπως είναι το κεντρικό παράθυρο και το παράθυρο αναζήτησης/αντικατάστασης. Ωστόσο, σε αυτά τα παράθυρα θα συμβούν γεγονότα τα οποία πυροδοτούνται για την επιτέλεση των απαραίτητων λειτουργιών. Την διαχείριση των γεγονότων αυτών προσφέρουν οι κλάσεις του πακέτου `eventhandlertexteditor` το οποίο και θα αναλυθεί στη συνέχεια. Η εσωτερική διάρθρωση του πακέτου σκιαγραφείται στο σχήμα 52.

Σχήμα 52. Πακέτο `eventhandlertexteditor` και κλάσεις



Το πακέτο αυτό αποτελείται από τέσσερις κλάσεις όπως αποτυπώνεται και στο σχήμα 52. Η κλάση `EventHandlerEditor` διαχειρίζεται όλη την ροή των γεγονότων του editor η οποία όπως φαίνεται καλεί όλες τις άλλες κλάσεις του πακέτου για την διαχείριση κάποιων γεγονότων. Η κλάση αυτή διαθέτει μία μέθοδο που είναι υπεύθυνη για την διαχείριση των συμβάντων του editor όπου ονομάζεται `handleEdit`. Στην συνέχεια θα παρουσιαστεί αναλυτικά ο τρόπος με τον οποίο η κλάση αυτή χειρίζεται γεγονότα όπως αποθήκευση sql αρχείου, φόρτωση των περιεχομένων του αρχείου, επαναφορά προηγούμενων γραμμένων στοιχείων κειμένου (Undo Typing), πλήρης εκκαθάριση του text panel του editor, εκτύπωση και τέλος την ασφαλή έξοδο από τον editor δίνοντας την επιλογή στον χρήστη να αποθηκεύσει τα περιεχόμενα του text panel στο αρχείο που επεξεργάζεται εκείνη την στιγμή.

Σε αυτό το σημείο θα παρουσιαστούν βασικά γεγονότα που συμβαίνουν στον editor τα οποία διαχειρίζονται από την κλάση EventHandlerEditor. Αναλυτικά:

- 1) Load Sql file: με την επιλογή για επεξεργασία ενός sql αρχείου του σχήματος, το παράθυρο του editor ανοίγει με τα περιεχόμενα του αρχείου ήδη φορτωμένα. Η λειτουργία αυτή γίνεται εφικτή από τον κώδικα που παρουσιάζεται στο σχήμα 53.

Σχήμα 53. Load sql file coding

```
if (fileName != "" && schemaPath != "") {  
  
    File f = new File(schemaPath + File.separator + "schemas" + File.separator + fileName);  
    FileReader reader = null;  
    try {  
        reader = new FileReader(f);  
    } catch (FileNotFoundException e1) {  
        System.out.println("File was not found. Incorrect file Path");  
        System.exit(-1);  
    }  
    BufferedReader br = new BufferedReader(reader);  
  
    try {  
        textArea.read(br, null);  
    }  
}
```

Η διαδικασία φόρτωσης των περιεχομένων του sql αρχείου ολοκληρώνεται με την εμφάνιση ενός κατάλληλου μηνύματος που ειδοποιεί για ορθή φόρτωση επιδεικνύοντας το όνομα του sql αρχείου. Παράλληλα, τοποθετούνται σε μία μεταβλητή αλφαριθμητικού τύπου τα αρχικά περιεχόμενα του αρχείου για μετέπειτα χρήση. Οι προαναφερθέντες ενέργειες παρουσιάζονται στο σχήμα 54.

Σχήμα 54. Ολοκλήρωση διαδικασίας φόρτωσης sql αρχείου

```
try {  
    br.close();  
} catch (IOException e1) {  
    e1.printStackTrace();  
}  
textArea.requestFocus();  
JOptionPane.showMessageDialog(null, fileName, "Directory of chosen sql document", JOptionPane.INFORMATION_MESSAGE);  
labelN.setText("initialVersion = " + textArea.getText()); // contents of initial version after load  
openDocPath = schemaPath + File.separator + "schemas" + File.separator + fileName;  
} else {  
    // something error happened if the control came here, so print error and exit safely  
    JOptionPane.showMessageDialog(null, "Filepath or Filename is invalid", "Invalid Directory", JOptionPane.ERROR_MESSAGE);  
    System.exit(-1);  
}
```

- 2) Save Sql file: η αποθήκευση ενός sql αρχείου αποτελεί μια διαδικασία η οποία ξεκινά με την τοποθέτηση των περιεχομένων του text panel του editor σε μια arraylist. Έπειτα, ένα αντικείμενο της κλάσης Document, η οποία βρίσκεται στο πακέτο versionmanager, δημιουργείται που έχει σκοπό την προσωρινή

αποθήκευση των περιεχομένων του text panel του editor τα οποία απαιτούνται για την λειτουργία της επαναφοράς γραμμένων στοιχείων κειμένου. Ακολουθώντας, καλείται η μέθοδος save της κλάσης eventHandlerSave ώστε να προχωρήσει η διαδικασία αποθήκευσης. Τέλος, αμέσως μετά την ολοκλήρωση της αποθήκευσης του αρχείου εμφανίζεται ένα μήνυμα που ενημερώνει τον χρήστη για την επιτυχημένη αποθήκευση του sql αρχείου. Όλες οι προαναφερθέντες ενέργειες, παρουσιάζονται στο σχήμα 55.

Σχήμα 55. Διαχείριση γεγονότος αποθήκευσης αρχείου

```
// event handling for the save button
buttonSave.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        if (checkBox1.isSelected()){ // volatile storage
            ArrayList<String> list = new ArrayList<String>();
            list.add(textArea.getText());
            Document doc1 = new Document(Number, "savvas", "21-5-2020", list, splitter.splitText(ldFile.getName())+"Log"+Number+".txt");
            vt1.putVersion(doc1);
            Number++;
            sv.save(textArea, openDocPath, list);
            JOptionPane.showMessageDialog(null, "Current document saved", "Saving Action Successfull", JOptionPane.INFORMATION_MESSAGE);
        }else{
            JOptionPane.showMessageDialog(null, "Select the volatile checkbox in order to save your file.", "No storage type selected", JOptionPane.INFORMATION_MESSAGE);
        }
    }
});
```

- 3) Undo Typing: η επαναφορά προηγούμενων γραμμένων στοιχείων κειμένου αποτελεί μία από τις λειτουργίες του editor. Αρχικά η διαδικασία συλλέγει όλες τις εκδόσεις του αρχείου που επεξεργάζεται εκείνη την δεδομένη χρονική στιγμή. Σε επόμενο στάδιο, εφόσον η επιλογή για αποθήκευση είναι ενεργοποιημένη και ο αριθμός των εκδόσεων του αρχείου είναι μεγαλύτερος του μηδενός τότε καλείται η μέθοδος rollback της κλάσης EventHandlerRollback, θέτοντας τα περιεχόμενα του text panel ως τα περιεχόμενα της προηγούμενης έκδοσης του αρχείου. Αν ο αριθμός εκδόσεων του αρχείου είναι ίσος με το μηδέν τότε ως περιεχόμενα φορτώνονται στο text panel τα αρχικά περιεχόμενα του αρχείου, εκείνα δηλαδή που αποθηκευτήκαν με την φόρτωση. Οι ενέργειες αυτές επιτελούνται από τον κώδικα του σχήματος 56.

Σχήμα 56. Rollback Action

```
// event handling for the roll back button in order to roll back to a previous version
buttonRol.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        Document[] docList = vt1.getEntireHistory(splitter.splitText(ldFile.getName())); // a list with all the volatile strategy version
        if (checkBox1.isSelected() && Number > 0){
            textArea.setText(rb.rollback(Number, textArea, splitter.splitText(ldFile.getName()), checkBox1, docList));
            Number--;
        }else if (checkBox1.isSelected() && Number == 0) {
            JOptionPane.showMessageDialog(null, "Initial contents were loaded.", "Undo Manager", JOptionPane.INFORMATION_MESSAGE);
            textArea.setText(initialVersion);
        }
    }
});
```

- 4) Πλήρης εκκαθάριση text panel: εφόσον το επιθυμεί ο χρήστης, ο editor προσφέρει δυνατότητα πλήρης εκκαθάρισης της text panel απλά με το πάτημα ενός κουμπιού. Ωστόσο, ένα παράθυρο εμφανίζεται ζητώντας από τον χρήστη τελική άδεια για την εκτέλεση της εκκαθάρισης. Ο κώδικας που επιτελεί την ενέργεια παρουσιάζεται στο σχήμα 57.

Σχήμα 57. Κώδικας πλήρους εκκαθάρισης text panel

```
// event handling for clear the entire doc
buttonClear.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        if (JOptionPane.showConfirmDialog(null, "Are you sure you want to clear the entire file?", "", 0) == JOptionPane.YES_OPTION){
            textArea.setText(null);
        }
    }
});
```

- 5) Ασφαλής έξοδος και αποθήκευση: ο editor προσφέρει δυνατότητα εξόδου από την επεξεργασία του αρχείου. Ωστόσο, για την ασφάλεια των περιεχομένων των sql αρχείων προσφέρεται η δυνατότητα αποθήκευσης των αλλαγών πριν από την έξοδο. Προϋποθέτει την επιλογή για αποθήκευση να είναι ενεργοποιημένη και εφόσον στεφθεί με επιτυχία η διαδικασία της αποθήκευσης τότε η ροή ελέγχου μεταφέρεται πίσω στο εργαλείο Insight και ο editor κλείνει.

Έπειτα, επόμενη κλάση του πακέτου eventhandlertexteditor είναι η EventHandlerRollback. Όπως έχει ήδη αναφερθεί η κλάση αυτή επιτελεί τον στόχο τη επαναφοράς γραμμένων στοιχείων κειμένου. Διαθέτει μία μέθοδο η οποία είναι υπεύθυνη για ολόκληρη την διαδικασία και μια βοηθητική μέθοδο που ελέγχει εάν η λίστα με τις εκδόσεις του αρχείου είναι κενή παράγοντας σε αυτή την περίπτωση κατάλληλο διαγνωστικό μήνυμα ελέγχου. Σε αντίθετη περίπτωση, η μέθοδος θα φορτώσει τα περιεχόμενα προηγούμενης έκδοσης του αρχείου στο text panel του editor. Η δομή της μεθόδου παρουσιάζεται στο σχήμα 58.

Σχήμα 58. Δομή μεθόδου rollback της κλάσης EventHandlerRollback

```
public String rollback(int Number, JTextPane textArea, String name, JCheckBoxMenuItem checkBox1, Document[] VolatileVersionsList){
    if (isEmpty(VolatileVersionsList)){
        JOptionPane.showMessageDialog(null, "You must save a version first in order to load it.", "You haven't store any versions yet.", JOptionPane.INFORMATION_MESSAGE);
    }else if(checkBox1.isSelected() && Number > 0){ // volatile storage
        Number--;
        return splitter.splitArrayList(VolatileVersionsList[Number].getContents());
    }
    return "";
}
```


Στο πακέτο αυτό λαμβάνει χώρα και μια κλάση η οποία διαθέτει βοηθητικό ρόλο. Η κλάση αυτή ονομάζεται `EventHandlerSplit` και η χρησιμότητα της εμφανίζεται ως μέρος της διαδικασίας `rollback` όπου αφαιρεί τις αγκύλες της `arraylist` ώστε να επιστρέψει μονάχα τα χρήσιμα περιεχόμενα του αρχείου. Δεύτερη εφαρμογή μεθόδου της κλάσης εντοπίζεται στην κλάση `EventHandlerEditor` όπου και χρησιμοποιείται η μέθοδος `splitText` για να κρατήσουμε μόνο το όνομα του `sql` αρχείου από όλο το υπόλοιπο μονοπάτι του αρχείου. Η δομή της κλάσης σκιαγραφείται στο σχήμα 59.

Σχήμα 59. Δομή κλάσης `EventHandlerSplit`

```
public class EventHandlerSplit {  
  
    public String splitText(String text){  
        String[] arrayS = text.split("[.]");  
        return arrayS[0];  
    }  
  
    public String splitArrayList(String text){  
        String[] arrayS = text.split("\\[");  
        String[] result = arrayS[1].split("\\]");  
        return result[0];  
    }  
}
```

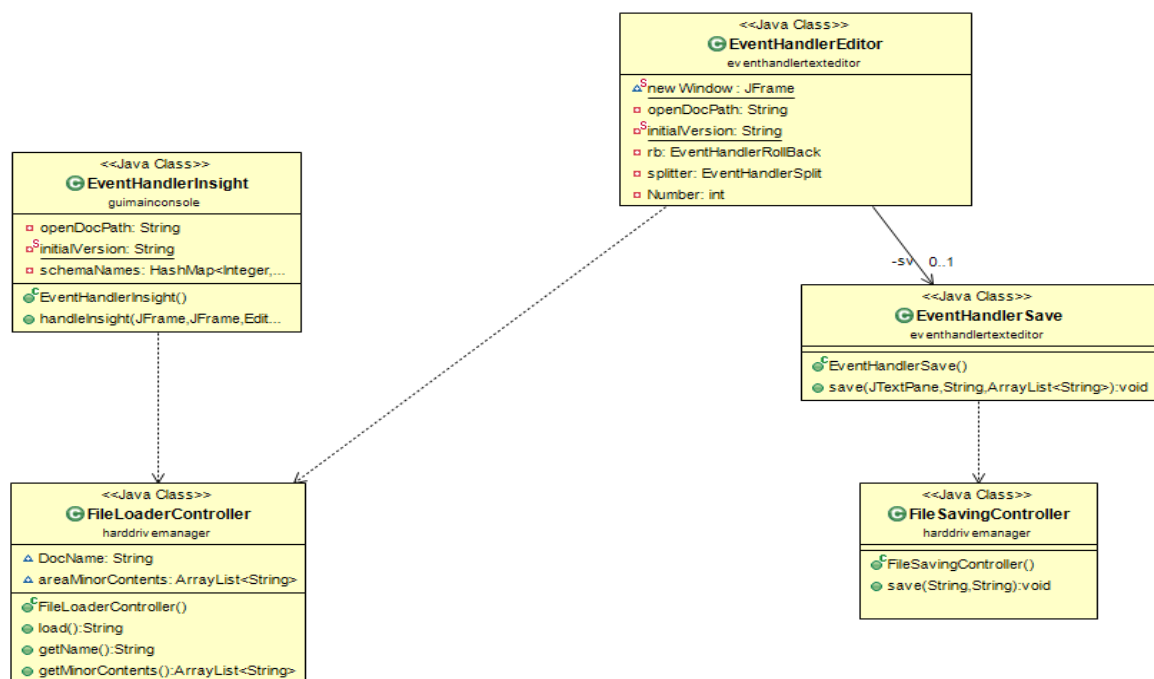
Τελευταία κλάση του πακέτου `eventhandlertexteditor` είναι η `EventHandlerSave`. Ο ρόλος της κλάσης αυτής αποτελεί η προσθήκη ενός ελέγχου για την ομαλή λειτουργία της διαδικασίας αποθήκευσης στην περίπτωση όπου το `text panel` του `editor` είναι κενό και ο χρήστης επιθυμεί αποθήκευση. Μόλις η έκβαση του ελέγχου αποτιμηθεί αληθής, τότε η μέθοδος κατασκευάζει ένα αντικείμενο της κλάσης `FileSavingController` όπου και ολοκληρώνεται η διαδικασία αποθήκευσης του αρχείου. Η δομή της μεθόδου `save` παρουσιάζεται στο σχήμα 60.

Σχήμα 60. Δομή μεθόδου `save` της κλάσης `EventHandlerSave`

```
public class EventHandlerSave {  
  
    public void save(JTextPane textArea,String openDocPath,ArrayList<String> list){  
  
        FileSavingController svFile = new FileSavingController();  
  
        // if text area is empty just set it to "" in order to save  
        if (textArea.getText().trim().length() == 0){  
            textArea.setText(" ");  
        }  
  
        svFile.save(openDocPath, textArea.getText());  
    }  
}
```

Το πακέτο το οποίο θα αναλυθεί στη συνέχεια είναι αυτό που προμηθεύει στο εργαλείο Insight και στον editor τις λειτουργίες φόρτωσης και αποθήκευσης των αρχείων. Το πακέτο ονομάζεται `harddrivemanager` και απαρτίζεται από δύο κλάσεις, την `FileSavingController` και την `FileLoaderController`. Όπως προδίδουν τα ονόματα των κλάσεων η πρώτη κλάση θεμελιώνει την ιδιότητα αποθήκευσης των αρχείων ενώ η δεύτερη, την ιδιότητα φόρτωσης των log και sql αρχείων. Η εσωτερική δομή του πακέτου αυτού φανερώνεται στο σχήμα 61. Ωστόσο στο σχήμα αυτό προστέθηκαν και ορισμένες κλάσεις του πακέτου `eventhandlertexteditor` ώστε να γνωστοποιηθούν με παραστατικό τρόπο και οι αλληλεπιδράσεις μεταξύ των πακέτων σε επίπεδο κλάσεων.

Σχήμα 61. Πακέτο `harddrivemanager` και αλληλεπιδράσεις με κλάσεις άλλων πακέτων



Όπως έχει αναλυθεί ήδη και εντοπίζεται και στο σχήμα 60, η κλάση `FileLoaderController` καλείται και χρησιμοποιείται από τις κλάσεις `EventHandlerInsight` και `EventHandlerEditor` ενώ η κλάση χρησιμοποιείται μόνο από την κλάση `EventHandlerSave` η οποία ανήκει στο πακέτο διαχείρισης των γεγονότων του editor. Αυτό συμβαίνει καθώς το κεντρικό εργαλείο δεν επιτρέπει μετάλλαξη των log files, μόνο οπτικοποίηση και ομαδοποίηση των περιεχομένων τους.

Ξεκινώντας την ανάλυση της εσωτερικής δομής των κλάσεων του πακέτου αυτού, αρχικά η κλάση `FileSavingController` αποτελεί μια κοινή διαδικασία αποθήκευσης περιεχομένων σε αρχείο. Ξεκινάει με τον έλεγχο ότι το όρισμα της μεθόδου `save` είναι όντως αρχείο `sql` και αν αποτιμηθεί αληθής η συνθήκη δημιουργείται ένα αντικείμενο της κλάσης `FileWriter` το οποίο γράφει όλα τα περιεχόμενα του `text panel` στο επιθυμητό `sql` αρχείο.

Έπειτα, η κλάση `FileLoaderController` προμηθεύει την λειτουργία της φόρτωσης των `log files`. Η βασική μέθοδος της κλάσης είναι η `load`. Η μέθοδος ξεκινά με την δημιουργία αντικειμένων ζωτικής σημασίας για την διαδικασία φόρτωσης τα οποία είναι των κλάσεων `FileReader` και `BufferedReader`. Έπειτα, η μέθοδος διαχωρίζει τα περιεχόμενα του `log file` με γνώμονα τις ετικέτες `major` και `minor` που έχουν χρησιμοποιηθεί για την κατηγοριοποίηση των σφαλμάτων, τοποθετώντας τα σε δύο `arraylists` για μετέπειτα χρήση. Τέλος, η κλάση διαθέτει δύο μεθόδους οι οποίες επιστρέφουν χρήσιμες πληροφορίες στις οποίες εντάσσονται το μονοπάτι του αρχείου που μόλις φορτώθηκε αλλά και την `arraylist` με τα λιγότερο σημαντικά σφάλματα (`minor`). Η δομή της μεθόδου απεικονίζεται στο σχήμα 62.

Σχήμα 62. Δομή `load` μεθόδου της κλάσης `FileLoaderController`

```
public String load(){
    JFileChooser openFileChooser = new JFileChooser();
    String openDocPath="";

    JTextPane textArea = new JTextPane();
    FileNameExtensionFilter xmlfilter = new FileNameExtensionFilter( "txt files (*.txt)", "txt");
    openFileChooser.setFileFilter(xmlfilter);
    //openFileChooser.setCurrentDirectory(new java.io.File("."));
    openFileChooser.setDialogTitle("Please select the log file.");

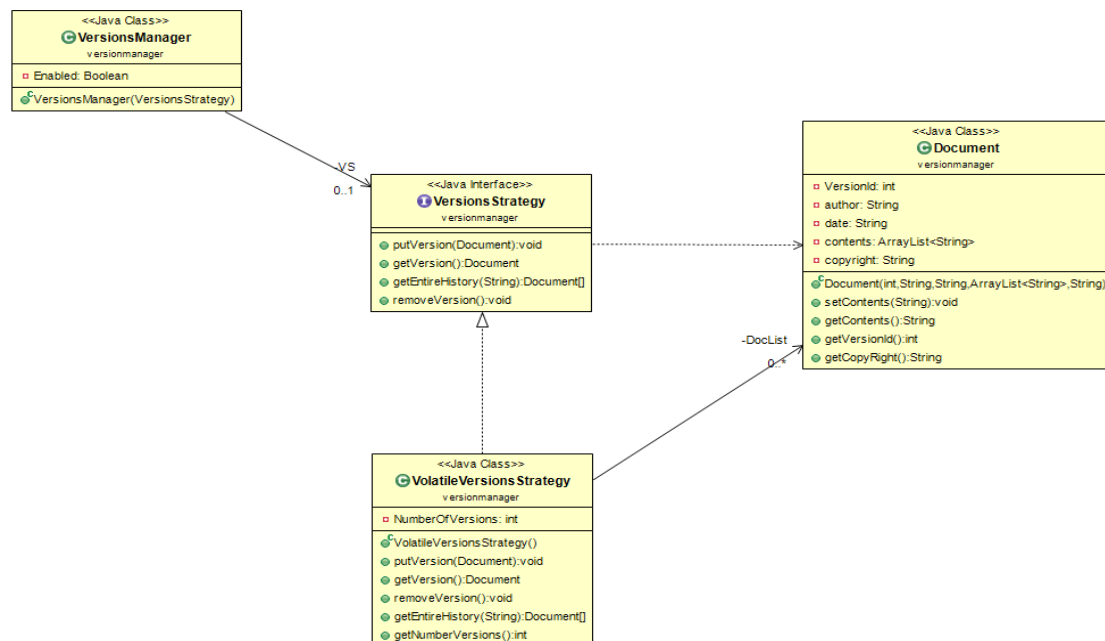
    if(openFileChooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
        openDocPath = openFileChooser.getSelectedFile().toString();
        FileReader reader = null;
        try {
            reader = new FileReader(openDocPath);
        } catch (FileNotFoundException e1) {
            e1.printStackTrace();
        }
        BufferedReader br = new BufferedReader(reader);
        File fileParent = new File(openFileChooser.getSelectedFile().getParent().toString()).getParentFile();
        DocName = fileParent.getAbsolutePath();

        try {
            textArea.read(br, null);
        } catch (IOException e1) {
            e1.printStackTrace();
        }

        try {
            br.close();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        textArea.requestFocus();
        JOptionPane.showMessageDialog(null, openDocPath, "Directory of chosen log file", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Τέλος, το πακέτο που θα αναλυθεί στη συνέχεια είναι το versionmanager. Η συμβολή του πακέτου αυτού στο εργαλείο είναι η προσφορά της τοπικής αποθήκευσης των περιεχομένων του text panel του editor για να υλοποιηθούν με βάση αυτό στη συνέχεια οι λειτουργίες αποθήκευσης και επαναφοράς σε προηγούμενα γραμμένα στοιχεία κειμένου. Η εσωτερική δομή του πακέτου αλλά και οι συσχετίσεις μεταξύ των κλάσεων που το στελεχώνουν απεικονίζονται στο σχήμα 63.

Σχήμα 63. Πακέτο versionmanager και κλάσεις



Όπως παρατηρούμε από το Uml διάγραμμα του σχήματος 63, η κλάση VersionStrategy θέτει ένα interface το οποίο θα πρέπει να υλοποιείται από τις κλάσεις που συμμετέχουν ως μορφές αποθήκευσης. Στο παρόν εργαλείο έχει δημιουργηθεί και χρησιμοποιείται η τοπική αποθήκευση ως τύπου Document των περιεχομένων του text panel, η οποία μπορεί να χειριστεί από την κλάση VolatileVersionsStrategy. Το πακέτο αυτό σε μελλοντικές επεκτάσεις μπορεί να επεκταθεί με τις κατάλληλες κλάσεις και μεθόδους ώστε να υποστηρίξει και άλλου είδους αποθήκευση. Περισσότερες πληροφορίες για την επέκταση αυτή θα παρουσιαστούν στο κεφάλαιο 6.

Έπειτα, τα περιεχόμενα του text panel του editor αποθηκεύονται τοπικά ως αντικείμενα της κλάσης Document. Για την δημιουργία ενός αντικειμένου Document απαιτούνται η παροχή πληροφοριών στις οποίες συγκαταλέγονται ο αριθμός της έκδοσης του Document (VersionId), όνομα συγγραφέα, ημερομηνία εγγραφής, copyright το οποίο

εκδίδεται ως ένα συνενωμένο text του ονόματος του αρχείου που επεξεργάζεται, του αριθμού έκδοσης (VersionId) και της λέξης log. Επιπρόσθετα, η κλάση αυτή διαθέτει μεθόδους οι οποίες επιστρέφουν τα στοιχεία της κλάσης.

Σε αυτό το σημείο θα αναλυθεί η κλάση που υλοποιεί την τοπική αποθήκευση, δηλαδή η κλάση VolatileVersionsStrategy. Όπως ορίζει το interface, η κλάση θα πρέπει να υλοποιεί τέσσερις μεθόδους τις putVersion, getVersion, getEntireHistory και removeVersion. Ακολουθεί η περιγραφή της λειτουργίας καθεμιάς μεθόδου. Αναλυτικά:

- 1) putVersion: όπως αναφέρει και το όνομα της, η μέθοδος αυτή τοποθετεί την κάθε έκδοση των περιεχομένων στην λίστα τύπου Document, στην οποία διατηρούνται όλες οι εκδόσεις του αρχείου. Το εργαλείο υποστηρίζει τριακόσιες εκδόσεις του αρχείου.
- 2) getVersion: η μέθοδος αυτή επιστρέφει την έκδοση του αρχείου που αποθηκεύτηκε τελευταία.
- 3) getEntireHistory: ίσως η πιο σημαντική μέθοδος της κλάσης. Επιστρέφει ολόκληρη την λίστα με τις εκδόσεις του αρχείου ώστε να μπορούμε μετέπειτα να πραγματοποιήσουμε την ενέργεια επαναφορά προηγούμενων γραμμένων στοιχείων κειμένου (Undo Typing).
- 4) removeVersion: η μέθοδος αυτή, με την προϋπόθεση ότι ο συνολικός αριθμός των εκδόσεων είναι μεγαλύτερος του μηδενός, αφαιρεί την τελευταία έκδοση που αποθηκεύτηκε στην λίστα των εκδόσεων. Η δομή της μεθόδου αυτής παρουσιάζεται στο σχήμα 64.

Σχήμα 64. Δομή μεθόδου removeVersion

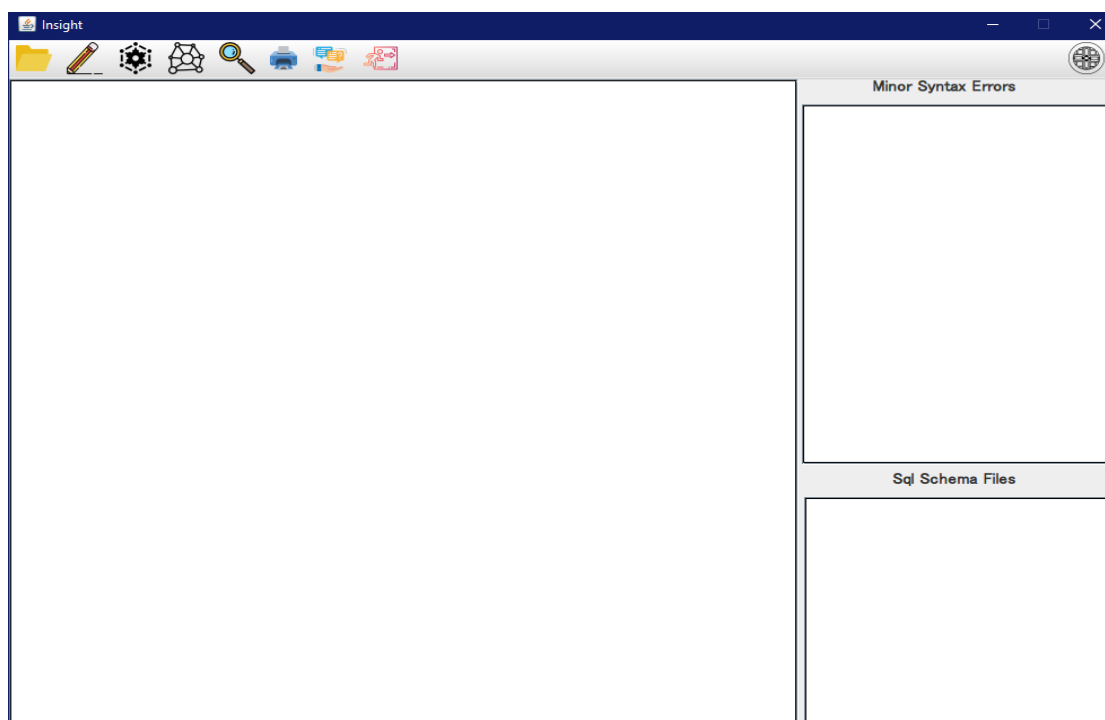
```
public void removeVersion() {  
    if (NumberOfVersions == 0){  
        System.out.println("Initial empty version of the document has reached");  
    }else{  
        Doclist[NumberOfVersions] = null;  
        NumberOfVersions--;  
        System.out.println("Current version has removed from history");  
    }  
}
```

4.3 Εγχειρίδιο ορθής χρήσης (Insight Manual)

4.3.1 Κατανόηση των πληροφοριών

Αρχικά, σε αυτή την ενότητα θα λάβει χώρα μια παρουσίαση του κεντρικού παραθύρου με στόχο την καλύτερη κατανόηση των πληροφοριών και των εργαλείων που παρέχονται στο εργαλείο Insight (Ενόραση). Όπως παρατηρούμε στο σχήμα 15, το κεντρικό παράθυρο αποτελείται από την μπάρα μενού η οποία εντοπίζεται στην κορυφή του παραθύρου και προσφέρει όλα τα εργαλεία της εφαρμογής. Έπειτα, εντοπίζονται δύο panel τα οποία θα στεγάσουν τα errors της μετάφρασης. Αριστερά και κεντρικά βρίσκεται το κύριο panel όπου απεικονίζονται τα σημαντικά λάθη (major errors) και στο δεξί panel θα βρίσκονται τα δευτερεύοντα λάθη (minor errors). Τέλος, το κεντρικό παράθυρο συμπληρώνει μια λίστα αρχείων με όνομα Sql Schema List όπου θα περιλαμβάνει τα αρχεία sql που συμμετέχουν στο σχήμα και όπως αναφέρει και ο τίτλος, στα αρχεία αυτά βρίσκονται τα σφάλματα που δημιουργήθηκαν στο στάδιο της μετάφρασης. Όλα τα παραπάνω στοιχεία παρουσιάζονται στο σχήμα 65.

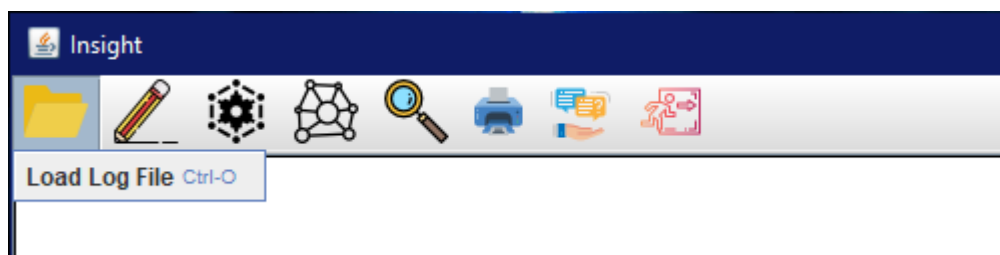
Σχήμα 65. Insight Main Window



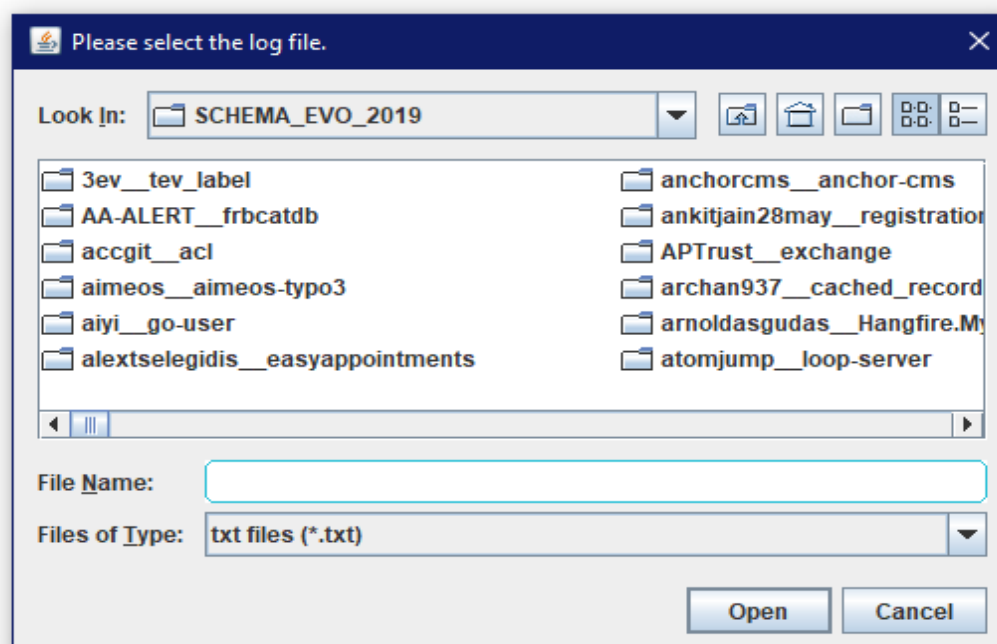
4.3.2 Φόρτωση Log File

Προκειμένου να εξετάσει ο αναλυτής τα λάθη της μετάφρασης των sql αρχείων, θα πρέπει να φορτώσει το log file στην Insight. Ο στόχος αυτός μπορεί να επιτευχθεί πατώντας το εικονίδιο φάκελος, το οποίο είναι το πρώτο εικονίδιο της μπάρας μενού. Αφού πατήσει ο αναλυτής το μενού εκείνο ένα παράθυρο θα εμφανιστεί προκειμένου να βρει και να φορτώσει το κατάλληλο log file. Εναλλακτικά, ο στόχος αυτός μπορεί να επιτευχθεί πληκτρολογώντας τον συνδυασμό των πλήκτρων Ctrl + ο ώστε να εμφανιστεί γρήγορα το παράθυρο επιλογής log file. Το log file μπορεί να βρίσκεται οπουδήποτε στο σύστημα. Η διαδικασία αυτή αναπαρίσταται στα σχήματα 66 και 67.

Σχήμα 66. Insight Load Log File Part 1



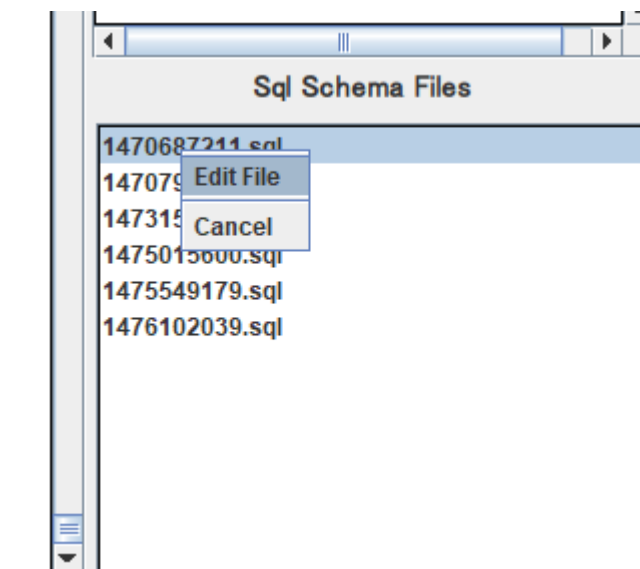
Σχήμα 67. Insight Load Log File Part 2



4.3.3 Επεξεργασία Sql αρχείου

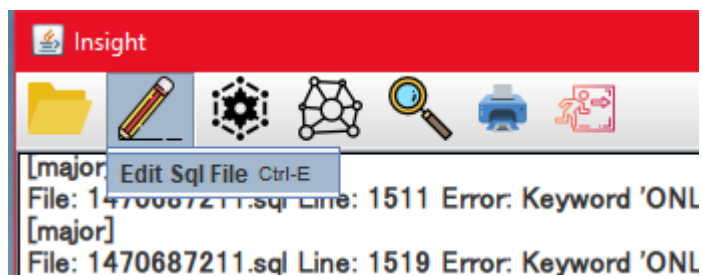
Στο εργαλείο Insight, προμηθεύονται δύο τρόποι προκειμένου να φορτώσουμε και να επεξεργαστούμε sql αρχεία, στα οποία βρίσκονται τα λάθη μετάφρασης. Ο πρώτος τρόπος είναι να διαλέξουμε το επιθυμητό αρχείο από την λίστα sql αρχείων κάτω δεξιά στο κεντρικό παράθυρο και έπειτα δεξί κλικ επάνω του ώστε να ανοίξει ένα μικρό μενού. Το μενού αυτό περιέχει δύο ενέργειες, την ενέργεια Edit, η οποία αν επιλεγθεί μας οδηγεί στην επεξεργασία του διαδεχθέντος αρχείου από τον Editor και την ενέργεια Cancel η οποία κλείνει το μικρό μενού. Η ακολουθία ενεργειών που περιεγράφηκε απεικονίζεται στο σχήμα 68.

Σχήμα 68. Editing a Sql File

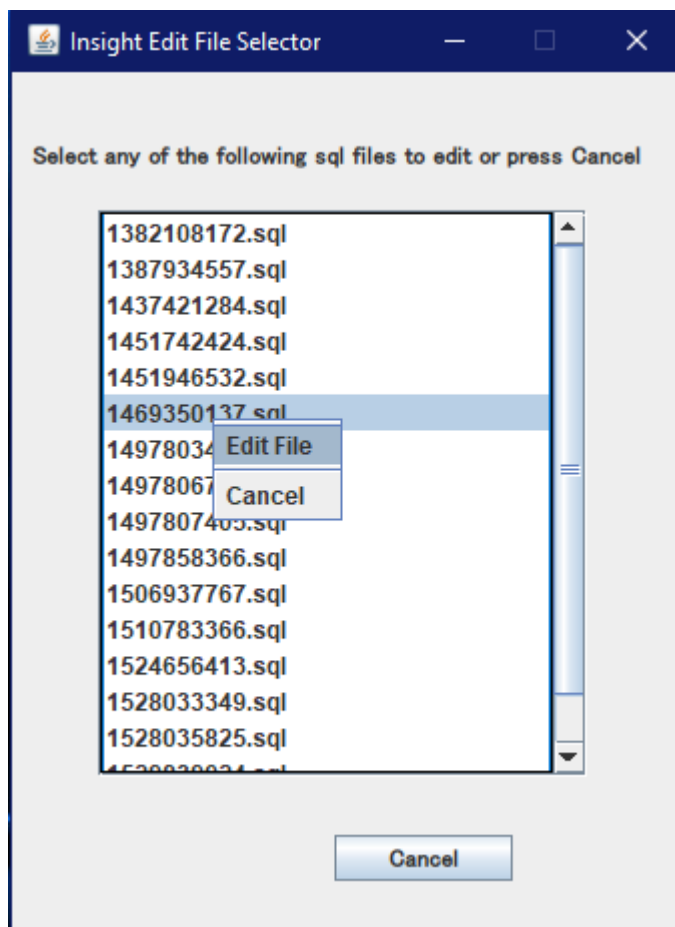


Ο δεύτερος τρόπος για να ανοίξουμε και να επεξεργαστούμε ένα sql αρχείο του σχήματος μπορεί να γίνει πατώντας το εικονίδιο pencil, το οποίο είναι το δεύτερο εικονίδιο στην μπάρα μενού και ως αποτέλεσμα θα εμφανιστεί ένα παράθυρο το οποίο θα περιέχει την ίδια λίστα αρχείων sql που εμφανίζεται και στο δεξί άκρο του κεντρικού παραθύρου. Για ακόμη μια φορά, θα επεξεργαστούμε το αρχείο sql κάνοντας δεξί κλικ επάνω στο επιλεγθέν αρχείο. Η διαδικασία που περιεγράφηκε, αναπαρίσταται στα σχήματα 69 και 70.

Σχήμα 69. Second way of editing a sql file



Σχήμα 70. Opening a sql file for editing

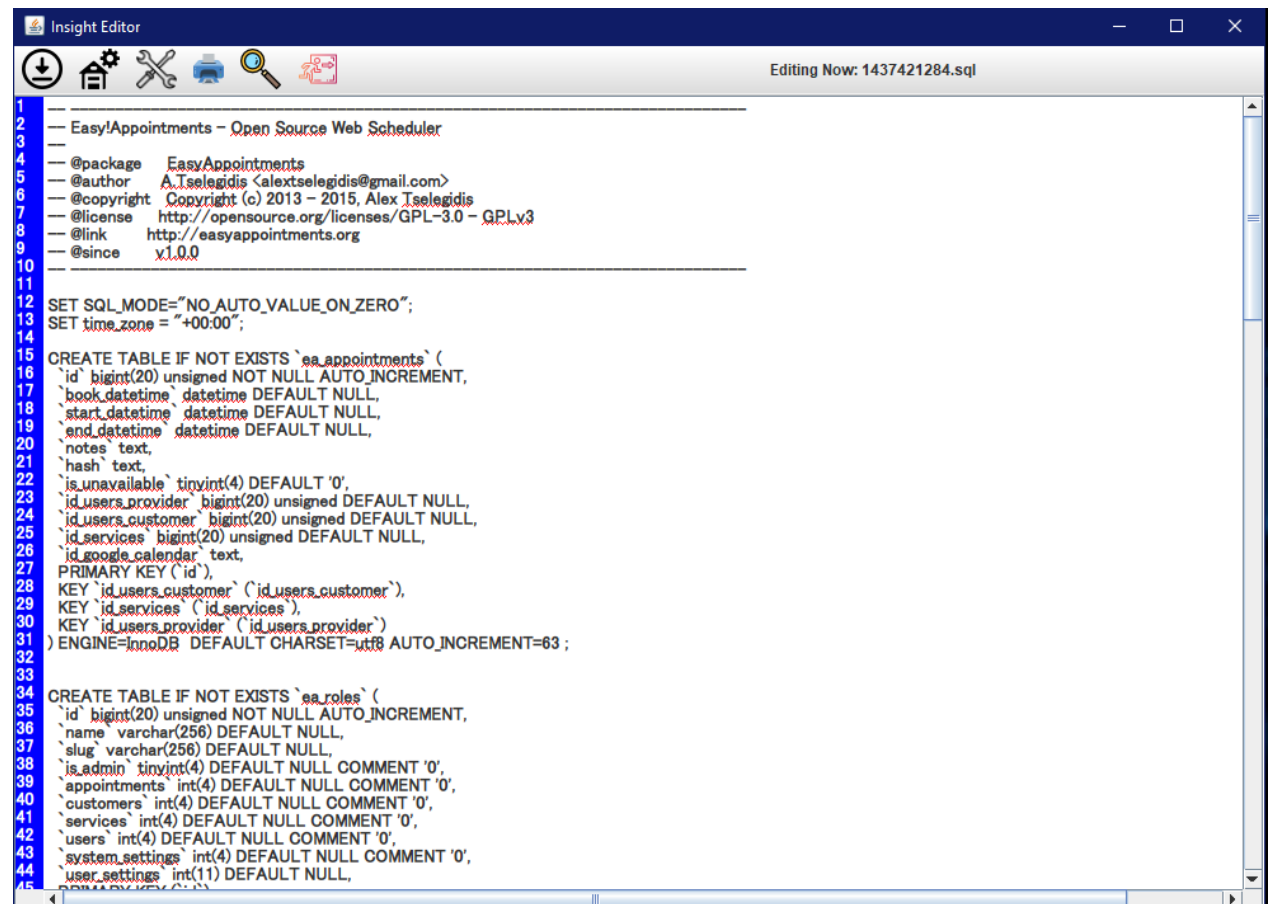


Το κουμπί Cancel στο κάτω μέρος του παραθύρου παρέχει σαφή έξοδο διαφυγής από την ενέργεια Edit και αφού πατηθεί το κουμπί αυτό το παράθυρο Edit παύει να είναι εμφανές.

Μετά την επιβεβαίωση της επιλογής Edit, ο editor θα εμφανιστεί και τα περιεχόμενα του επιλεγθέντος sql αρχείου θα είναι αρχικά φορτωμένα και θα εμφανιστούν στην κεντρική περιοχή κειμένου. Ο editor υποστηρίζει λειτουργίες αποθήκευσης του αρχείου, επαναφοράς προηγούμενων στοιχείων κειμένου (Undo Typing), λειτουργία

εύρεσης/αντικατάστασης λέξεων η οποία είναι πιο εκτεταμένη από αυτή του Insight εργαλείου καθώς προσφέρει επιλογές αντικατάστασης, αντικατάστασης και διαγραφής όλων των στιγμιότυπων της λέξης. Ο editor παρουσιάζεται στο σχήμα 71.

Σχήμα 71. Editor Main Window



The screenshot shows the 'Insight Editor' window. The title bar includes the application name and standard window controls. Below the title bar is a toolbar with icons for file operations (download, save, open, print, search, etc.). The main area is a code editor displaying SQL code. The code includes package information for 'Easy!Appointments' and two SQL statements: 'CREATE TABLE IF NOT EXISTS `ea_appointments`' and 'CREATE TABLE IF NOT EXISTS `ea_roles`'. The code is syntax-highlighted, with keywords in blue, identifiers in black, and string literals in red. Line numbers are visible on the left side of the editor.

```
1  -- Easy!Appointments - Open Source Web Scheduler
2
3
4  -- @package EasyAppointments
5  -- @author A.Iselegidis <alexselegidis@gmail.com>
6  -- @copyright Copyright (c) 2013 - 2015, Alex Iselegidis
7  -- @license http://opensource.org/licenses/GPL-3.0 - GPLv3
8  -- @link http://easyappointments.org
9  -- @since v1.0.0
10
11
12 SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
13 SET time_zone = "+00:00";
14
15 CREATE TABLE IF NOT EXISTS `ea_appointments` (
16   `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
17   `book_datetime` datetime DEFAULT NULL,
18   `start_datetime` datetime DEFAULT NULL,
19   `end_datetime` datetime DEFAULT NULL,
20   `notes` text,
21   `hash` text,
22   `is_unavailable` tinyint(4) DEFAULT '0',
23   `id_users_provider` bigint(20) unsigned DEFAULT NULL,
24   `id_users_customer` bigint(20) unsigned DEFAULT NULL,
25   `id_services` bigint(20) unsigned DEFAULT NULL,
26   `id_google_calendar` text,
27   PRIMARY KEY (`id`),
28   KEY `id_users_customer` (`id_users_customer`),
29   KEY `id_services` (`id_services`),
30   KEY `id_users_provider` (`id_users_provider`)
31 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=63 ;
32
33
34 CREATE TABLE IF NOT EXISTS `ea_roles` (
35   `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
36   `name` varchar(256) DEFAULT NULL,
37   `slug` varchar(256) DEFAULT NULL,
38   `is_admin` tinyint(4) DEFAULT NULL COMMENT '0',
39   `appointments` int(4) DEFAULT NULL COMMENT '0',
40   `customers` int(4) DEFAULT NULL COMMENT '0',
41   `services` int(4) DEFAULT NULL COMMENT '0',
42   `users` int(4) DEFAULT NULL COMMENT '0',
43   `system_settings` int(4) DEFAULT NULL COMMENT '0',
44   `user_settings` int(11) DEFAULT NULL,
45   PRIMARY KEY (`id`)
```

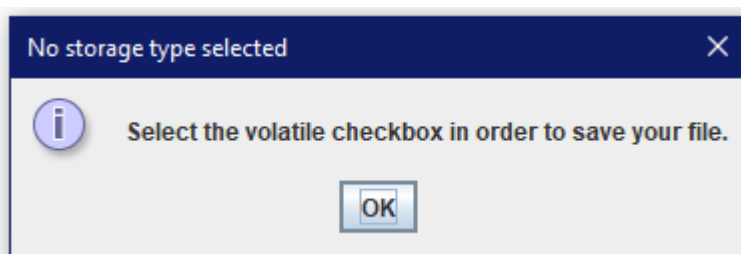
Στην κορυφή του παραθύρου υπάρχει μια μπάρα μενού η οποία περιέχει όλα τα προαναφερθέντα εργαλεία. Ο editor ακόμα προσφέρει δυνατότητα εκτύπωσης του sql αρχείου σε συνδεδεμένο εκτυπωτή αλλιώς υπάρχει η δυνατότητα εξαγωγής του αρχείου σε μορφή pdf. Οι επιλογές εκτύπωσης και εξαγωγής θα αναλυθούν σε επομένη ενότητα. Επιπρόσθετα στον editor, λαμβάνει χώρα μια κάθετη γραμμή μπλε χρώματος η οποία επιδεικνύει τον αριθμό γραμμής στο αρχείο ώστε ο αναλυτής να διακρίνει με ευκολία σε ποια γραμμή υπάρχει το λάθος μετάφρασης. Όπως μπορούμε να διακρίνουμε, στην μπάρα μενού στο δεξί μέρος υπάρχει μια ετικέτα η οποία περιέχει το όνομα του sql αρχείου που επεξεργάζεται αυτή την στιγμή. Τέλος, ο editor διαθέτει την ικανότητα του συντακτικού ελέγχου των λέξεων (Spellcheck) που έχουν γραφτεί με σημείο αναφοράς το αγγλικό λεξιλόγιο. Οι λανθασμένες λέξεις απεικονίζονται με κόκκινη υπογράμμιση.

4.3.4 Αποθήκευση Sql αρχείου

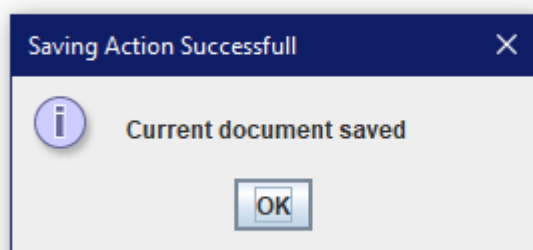
Για να αποθηκεύσουμε τα περιεχόμενα της περιοχής κειμένου του editor στο αντίστοιχο sql αρχείο είναι αναγκαίο να διατελέσουμε δύο ενέργειες. Πρώτον θα πρέπει να ενεργοποιήσουμε την ικανότητα αποθήκευσης το οποίο μπορεί να επιτευχθεί κάνοντας κλικ στο checkbox με όνομα volatile που βρίσκεται στο δεύτερο εικονίδιο της μπάρας μενού. Αν δεν ενεργοποιήσουμε την ικανότητα αποθήκευσης τότε ένα μήνυμα θα εμφανιστεί αναφέροντάς ότι το αρχείο δεν μπορεί να αποθηκευτεί. Η ικανότητα αποθήκευσης ενεργοποιείται και με την πληκτρολόγηση του συνδυασμού πλήκτρων Ctrl + 1.

Δεύτερον, για να αποθηκεύσουμε θα πρέπει απλά να κάνουμε κλικ στην επιλογή που θα εμφανιστεί στο πρώτο μενού και το αρχείο θα αποθηκευτεί επιτυχώς. Εναλλακτικά, όπως όλοι οι σύγχρονοι editors μπορούν να αποθηκεύσουν με την πληκτρολόγηση του συνδυασμού Ctrl + s έτσι και ο editor του εργαλείου Insight αποθηκεύει το αρχείο και με αυτόν τον τρόπο. Η ακολουθία ενεργειών για την αποθήκευση του αρχείου αναπαρίσταται στο σχήματα 72 και 73.

Σχήμα 72. Saving ability not enabled



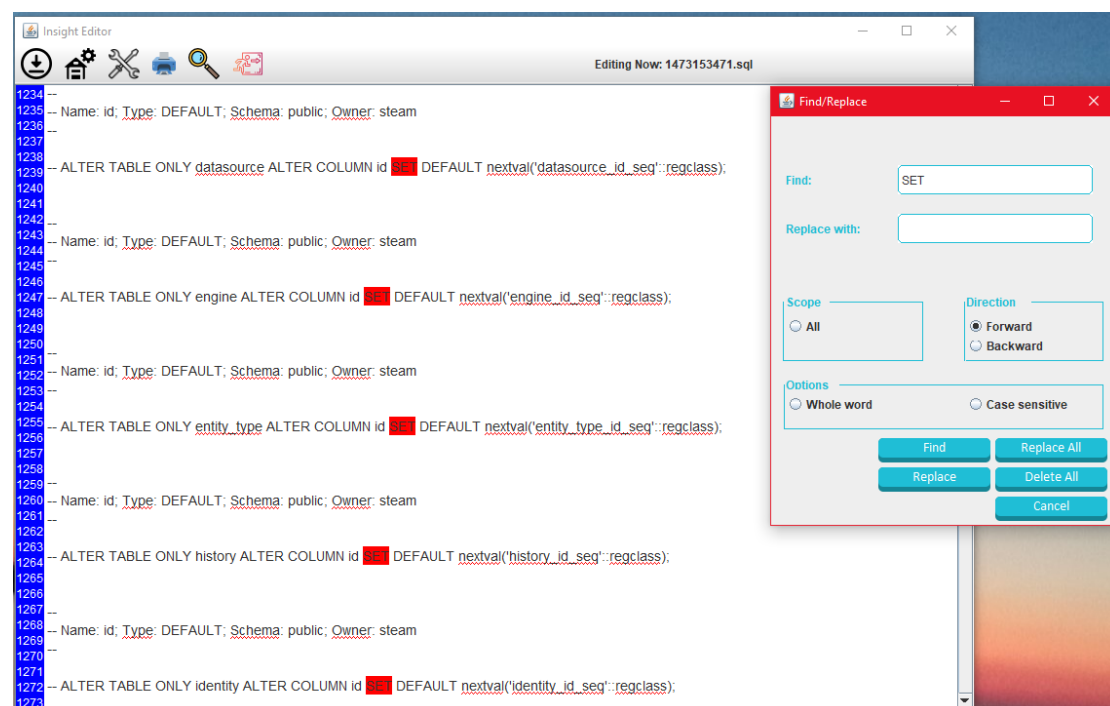
Σχήμα 73. Saving complete



4.3.5 Λειτουργία Αναζήτησης (Find/Replace)

Στον editor υφίσταται λειτουργία αναζήτησης η οποία προσφέρει μια πληθώρα από επιλογές ώστε ο αναλυτής να βρει με επιτυχία οποιαδήποτε λέξη. Για να εμφανιστεί το παράθυρο αναζήτησης μπορούμε είτε να το διαλέξουμε από το εικονίδιο με τον μεγεθυντικό φακό είτε πληκτρολογώντας Ctrl + f. Η εμβέλεια της αναζήτησης μπορεί να είναι είτε ολόκληρο το sql αρχείο διαλέγοντας την επιλογή All στο κουτί Scope είτε το μισό αρχείο διαλέγοντας οποιαδήποτε από τις επιλογές στο κουτί Direction. Οι επιλογές αυτές αναπαρίστανται στο σχήμα 74.

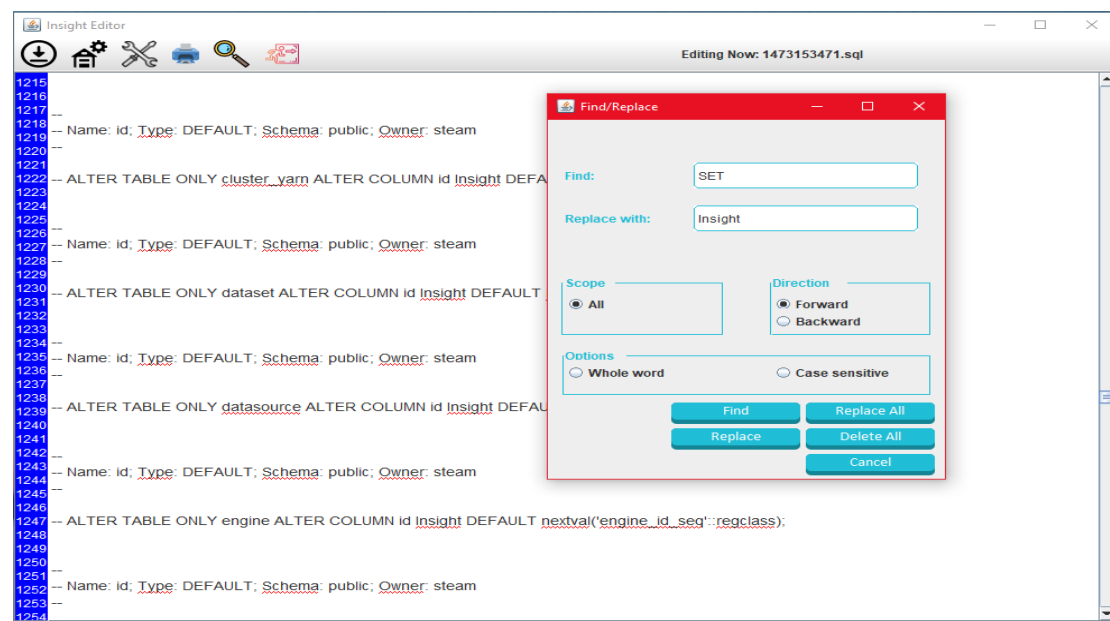
Σχήμα 74. Finding a word in a sql file



Όπως απεικονίζεται, η λειτουργία αναζήτησης προσφέρει τις δυνατότητες αναζήτησης μια λέξης ως αυτόνομη λέξη και όχι υποσύνολο μιας άλλης λέξης και αυτό μπορεί να επιτευχθεί κάνοντας κλικ στο radio button Whole Word. Επιπρόσθετα, το radio button Case Sensitive επιτρέπει την εύρεση κάθε είδους στιγμιότυπου της λέξης που αναζητείται. Τέλος, το παράθυρο αναζήτησης προσφέρει λειτουργίες αντικατάστασης όλων και διαγραφής όλων των στιγμιότυπων. Για να επιτευχθούν οι στόχοι αυτοί θα πρέπει πρώτα, να πληκτρολογήσουμε στην περιοχή με ετικέτα Find την λέξη που θέλουμε να αναζητήσουμε και έπειτα να πατήσουμε το button Find προκειμένου να βρούμε όλα τα στιγμιότυπα. Ακολούθως, αν επιθυμούμε την διαγραφή όλων των στιγμιότυπων διαλέγουμε το button Delete All και ο στόχος επιτυγχάνεται. Αν επιθυμούμε

αντικατάσταση ενός ή όλων των στιγμιότυπων της λέξης αρχικά πληκτρολογούμε την λέξη με την οποία θα αντικαταστήσουμε την παρούσα λέξη στο πεδίο με ετικέτα Replace With. Έπειτα, διαλέγουμε όποιο button επιθυμούμε μεταξύ των Replace ή Replace All. Το κουμπί Cancel κλείνει το παράθυρο και την λειτουργία εύρεσης λέξεων. Παράδειγμα της αντικατάστασης όλων των στιγμιότυπων της λέξης SET με την λέξη Insight παρουσιάζεται στο σχήμα 75.

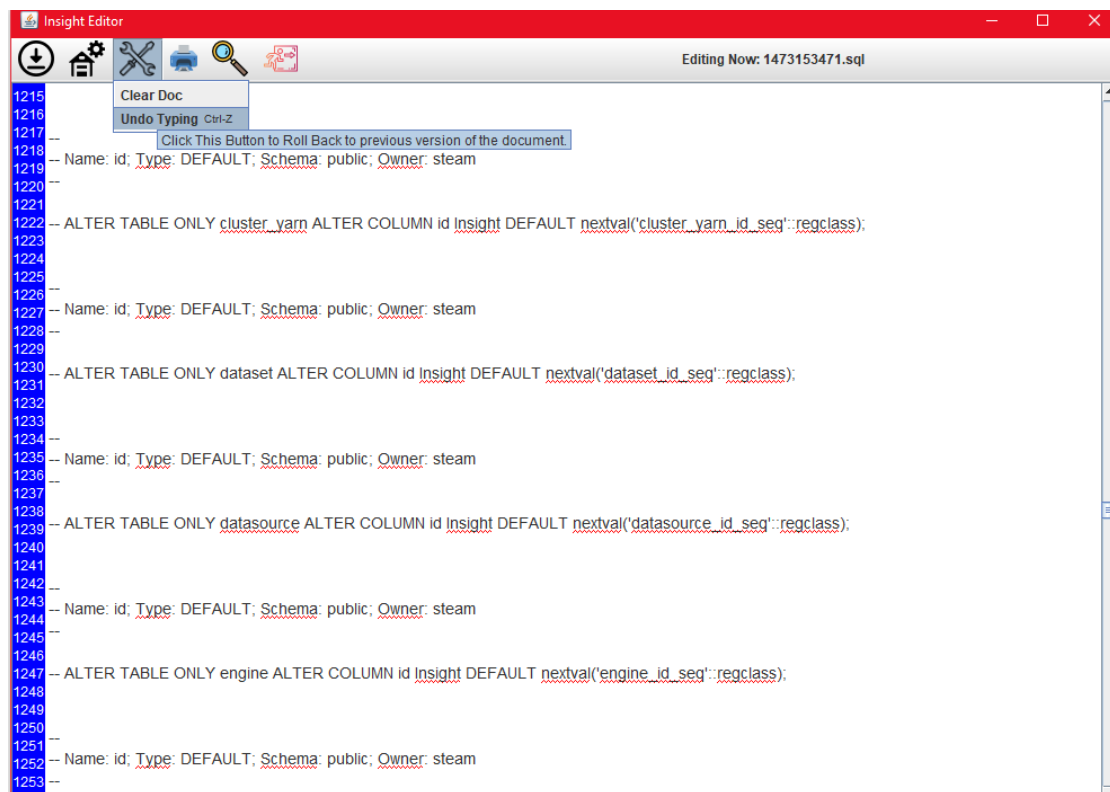
Σχήμα 75. Replace All Example



4.3.6 Παρεχόμενες Λειτουργίες

Ο editor του εργαλείου Insight διαθέτει την ικανότητα της επαναφοράς γραμμένων στοιχείων κειμένου και να αντικαταστήσουν τα παρόν στοιχεία κειμένου. Η ενέργεια αυτή μπορεί να εκτελεστεί είτε κάνοντας κλικ στο μενού με το εικονίδιο εργαλεία (τρίτο εικονίδιο της μπάρας μενού) και επιλέγοντας το αντικείμενο Undo Typing. Εναλλακτικά, ο editor διαθέτει την συντόμευση για την λειτουργία αυτή πληκτρολογώντας Ctrl + z. Επιπρόσθετα, ο editor διαθέτει λειτουργία ολικής εκκαθάρισης της περιοχής κειμένου απλά διαλέγοντας το αντικείμενο Clear Doc από το μενού με εικονίδιο εργαλεία. Για την ολική εκκαθάριση θα εμφανιστεί παράθυρο που θα ζητά τελική επιβεβαίωση. Οι δύο αυτές ενέργειες παρουσιάζονται στο σχήμα 76.

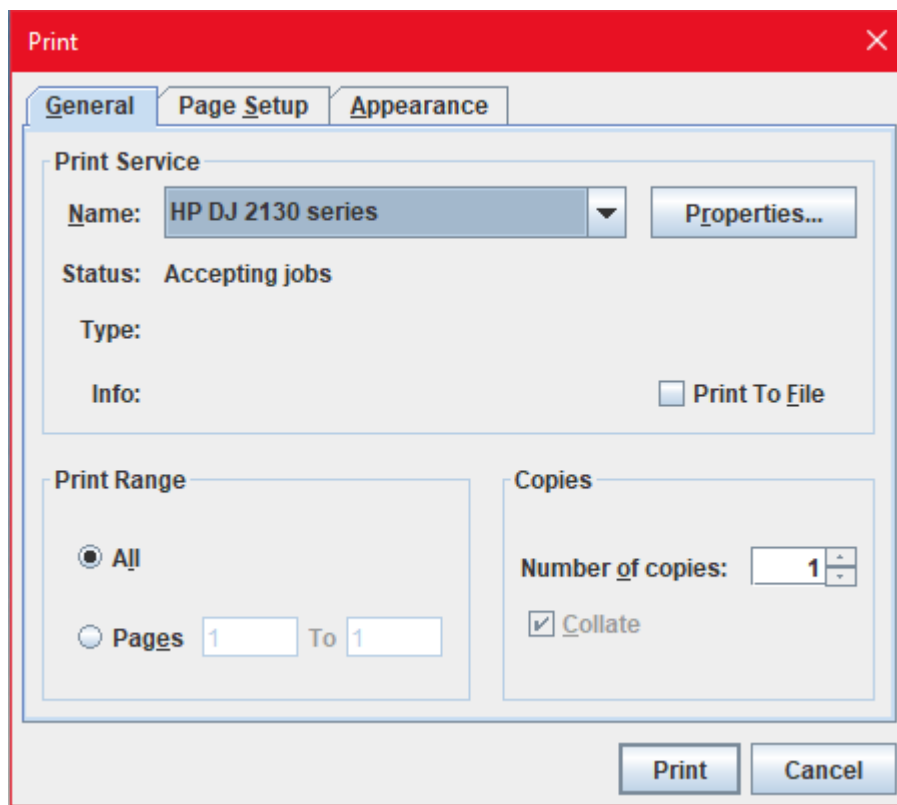
Σχήμα 76. Undo Typing and Clear Doc Tools



4.3.7 Εκτύπωση αρχείου και διαθέσιμες επιλογές

Το παράθυρο εκτύπωσης προσφέρει ευρεία ποικιλία επιλογών όσον αφορά το αποτέλεσμα της εκτύπωσης. Αυτό σημαίνει πως αν δεν διαθέτουμε συνδεδεμένο εκτυπωτή δεν αποτελεί πρόβλημα καθώς υπάρχει η δυνατότητα εξαγωγής των περιεχομένων του αρχείου log ή sql σε μορφή PDF και να αποθηκευτεί οπουδήποτε στο σύστημα. Η επιλογή αυτή ονομάζεται Microsoft Print to PDF. Στην αντίθετη περίπτωση, όπου διαθέτουμε εκτυπωτή τότε το εργαλείο Insight θα ανιχνεύσει τον συνδεδεμένο εκτυπωτή και θα εμφανίσει τα στοιχεία του στην περιοχή Name του print παραθύρου. Παράδειγμα εκτύπωσης αρχείου με συνδεδεμένο εκτυπωτή παρουσιάζεται στο σχήμα 77.

Σχήμα 77. Print window

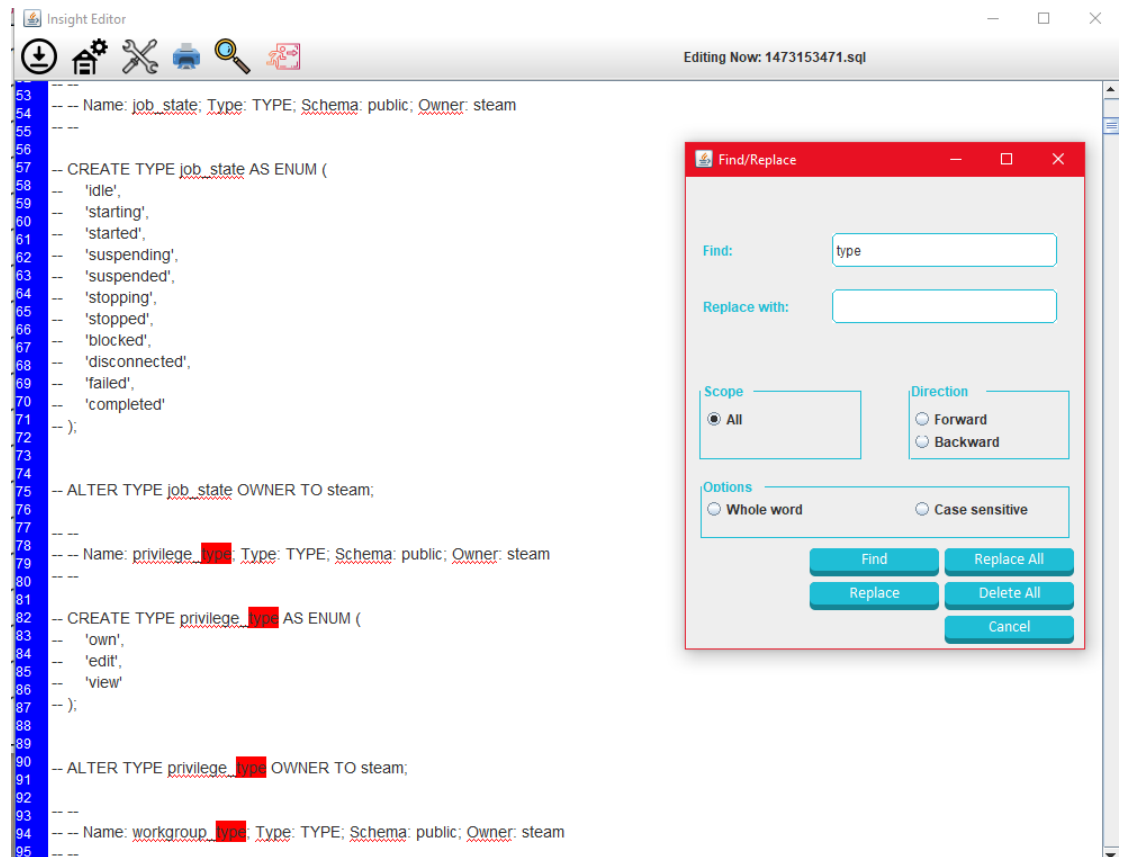


4.3.8 Παρουσίαση αποτελεσμάτων αναζήτησης

Στην ενότητα αυτή θα παρουσιαστούν ορισμένα παραδείγματα χρήσης της λειτουργίας αναζήτησης με σκοπό την επίδειξη των δυνατοτήτων που προσφέρει η λειτουργία αναζήτησης τόσο στο Insight εργαλείο όσο και στον editor. Οι παρακάτω περιπτώσεις αφορούν την αναζήτηση της λέξης 'type' αλλά η μία περίπτωση από την άλλη διαφέρει στην επιλογή των εργαλείων τα οποία παράγουν και διαφορετικά αποτελέσματα.

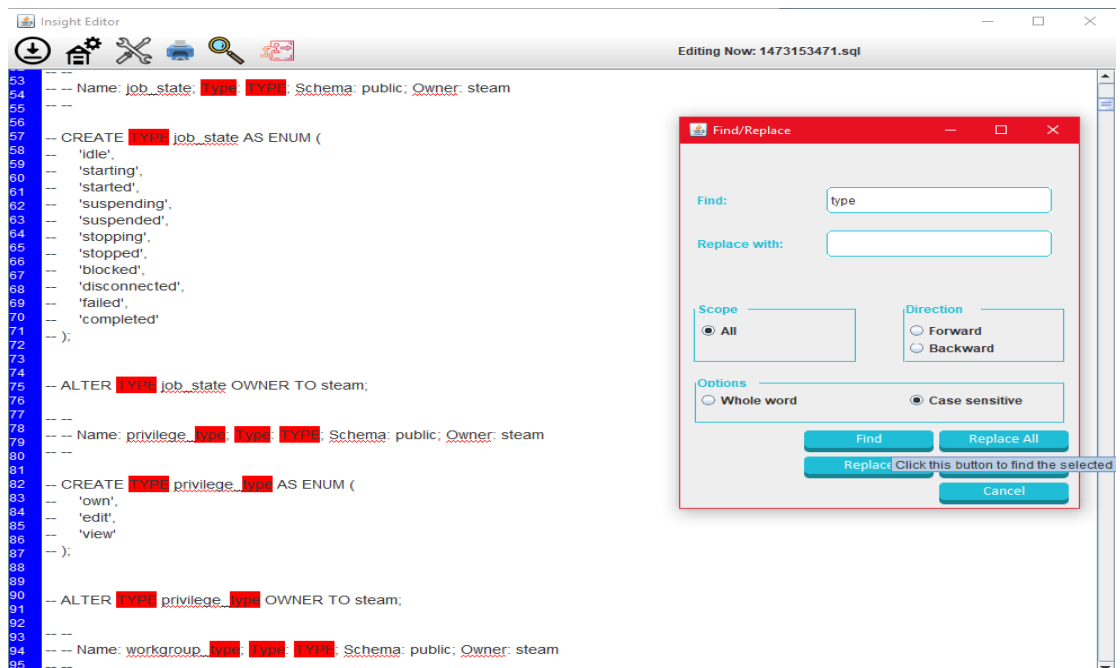
1. Πρώτη περίπτωση: απλή αναζήτηση της λέξης 'type' με εμβέλεια όλο το sql αρχείο. Επομένως επιλέγουμε το radio button All στο box Scope. Η περίπτωση αυτή αποτελεί απλή περίπτωση αναζήτησης καθώς δεν συνδυάζει παραπάνω από ένα εργαλεία για μια πιο επιτυχημένη αναζήτηση. Σε αυτή την περίπτωση όπως γίνεται αντιληπτό ότι η αναζήτηση θα βρει την λέξη και ως υποσύνολο άλλης λέξης. Το αποτέλεσμα παρουσιάζεται στο σχήμα 78.

Σχήμα 78. Search of keyword 'type' with scope All



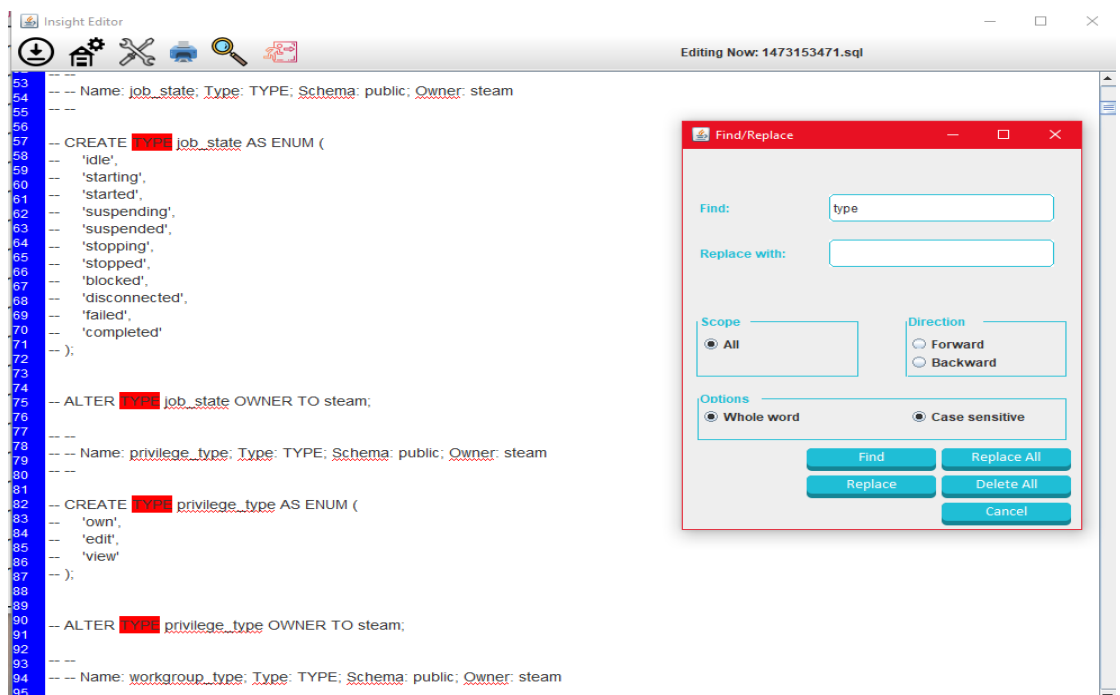
2. Δεύτερη περίπτωση: η περίπτωση αυτή αποτελεί πιο αποδοτική περίπτωση αναζήτησης καθώς σε αυτή θα συνδυαστούν πιο πολλά εργαλεία όπως το εργαλείο Case Sensitive, το οποίο επιτρέπει να βρεθούν όλα τα στιγμιότυπα της λέξης που αναζητούμε όποια μορφή και αν διαθέτουν. Αυτό σημαίνει ότι τώρα μπορούν να βρεθούν όλοι οι δυνατοί συνδυασμοί με τους οποίους είναι δυνατό να γραφεί η λέξη 'type' δηλαδή 'TYPE' ή 'Type'. Τα αποτελέσματα της αναζήτησης παρουσιάζονται στο σχήμα 79.

Σχήμα 79. Search of keyword 'type' with case Sensitive option enabled



3. Τρίτη περίπτωση: αποτελεί την πιο αποδοτική περίπτωση αναζήτησης καθώς τώρα θα συνδυαστούν τα εργαλεία Case Sensitive και Whole word το οποίο επιτρέπει να βρεθούν μόνο τα στιγμιότυπα της αναζητούμενης λέξης τα οποία αποτελούν αυτόνομες λέξεις και όχι υποσύνολα άλλων λέξεων. Το αποτέλεσμα της αναζήτησης αναπαρίστανται στο σχήμα 80.

Σχήμα 80. Search of keyword 'type' with Case Sensitive and Whole word enabled



Κεφάλαιο 5. Πειραματική Αξιολόγηση

5.1 Μεθοδολογία Πειραματισμού

Όσον αφορά τον πειραματισμό, μετρήθηκαν οι απαιτούμενοι χρόνοι για την φόρτωση των σχημάτων της βάσης για το εργαλείο Εκάτη και αντίστοιχα για το εργαλείο Insight μετρήθηκαν οι απαιτούμενοι χρόνοι για την ορθή φόρτωση των περιεχομένων των log files που παράχθηκαν για τα παραπάνω σχήματα.

Τα υπό εξέταση σχήματα είναι τα εξής;

- 1) h2oai_steam το οποίο αποτελείται από δέκα sql αρχεία.
- 2) azzlack_Sentinel.OAuth το οποίο αποτελείται από τρία sql αρχεία.
- 3) blabla1337_skf-flask το οποίο αποτελείται από 45 sql αρχεία.
- 4) intelliants_subrion το οποίο αποτελείται από 266 sql αρχεία.
- 5) webnuts_post_json το οποίο αποτελείται από 3 sql αρχεία.

Στη συνέχεια παρουσιάζεται ένας πίνακας στο σχήμα 5.1 με όλα τα αναλυτικά αποτελέσματα που προέκυψαν από το πείραμα. Μεταξύ των αποτελεσμάτων εντοπίζονται ο συνολικός αριθμός αρχείων στο σχήμα, οι συνολικοί χρόνοι φόρτωσης των αρχείων και οι συνολικοί αριθμοί γραμμών των αρχείων.

Breakdown of Running Sql Database Schemas both in Hecate and Insight tools

	Hecate		Insight		Total (#) lines of log file
	Total(## sql files	Total Time(sec)	Total (## lines	Total Time (sec)	
h2oai_steam	10	2,75	17.079	2,51	2.162
Azzlack_Sentinel.OAuth	3	1,32	145	1,63	122
blabla1337_skf-flask	45	8,1	28.139	13,70	37.682
intelliants_subrion	266	25,31	672.471	60,89	91.002
webnuts_post_json	3	0,87	881	0,91	121

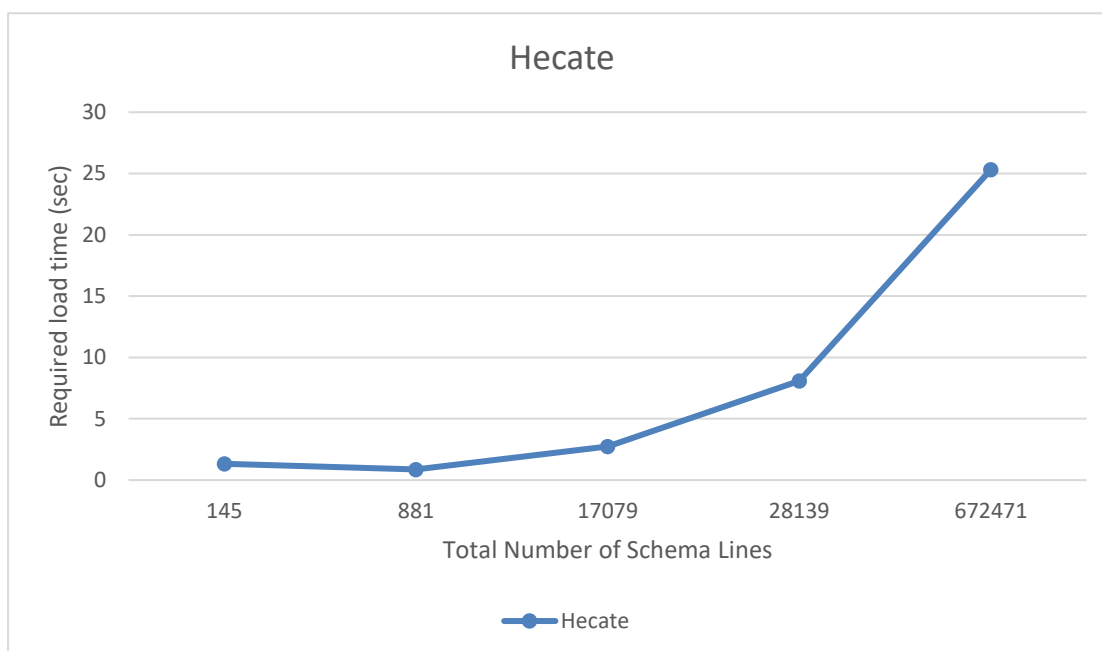
Πίνακας 5.1 Περιέχονται τα αποτελέσματα για πέντε σχήματα τα οποία εξετάστηκαν και στα δύο εργαλεία ως προς το χρόνο που απαιτείται για την πλήρη φόρτωση τους.

5.2 Αναλυτική Παρουσίαση αποτελεσμάτων

Σε αυτή την ενότητα θα παρουσιαστούν γραφήματα που συσχετίζουν τον χρόνο που απαιτείται για την φόρτωση των σχημάτων βάσεων, με το μέγεθος των σχημάτων αυτών ώστε να κατανοηθεί σε βάθος αυτή η συσχέτιση.

Όσον αφορά τα σχήματα βάσεων, δύο σχήματα περιείχαν πολλά sql αρχεία που ως αποτέλεσμα είχαν την αύξηση του απαιτούμενου χρόνου φόρτωσης. Μάλιστα, στο σχήμα `intelliants_subrion` ο χρόνος φόρτωσης και επεξεργασίας στο εργαλείο Εκάτη αναρριχήθηκε στα 25.31 δευτερόλεπτα και αντίστοιχα για το εργαλείο Insight πλησίασε το ένα λεπτό. Ωστόσο, για τα υπόλοιπα σχήματα βάσεων τα οποία κατά κανόνα δεν ξεπερνούσαν τον αριθμό των δέκα αρχείων, ο απαιτούμενος χρόνος δεν ξεπερνούσε τα τρία δευτερόλεπτα.

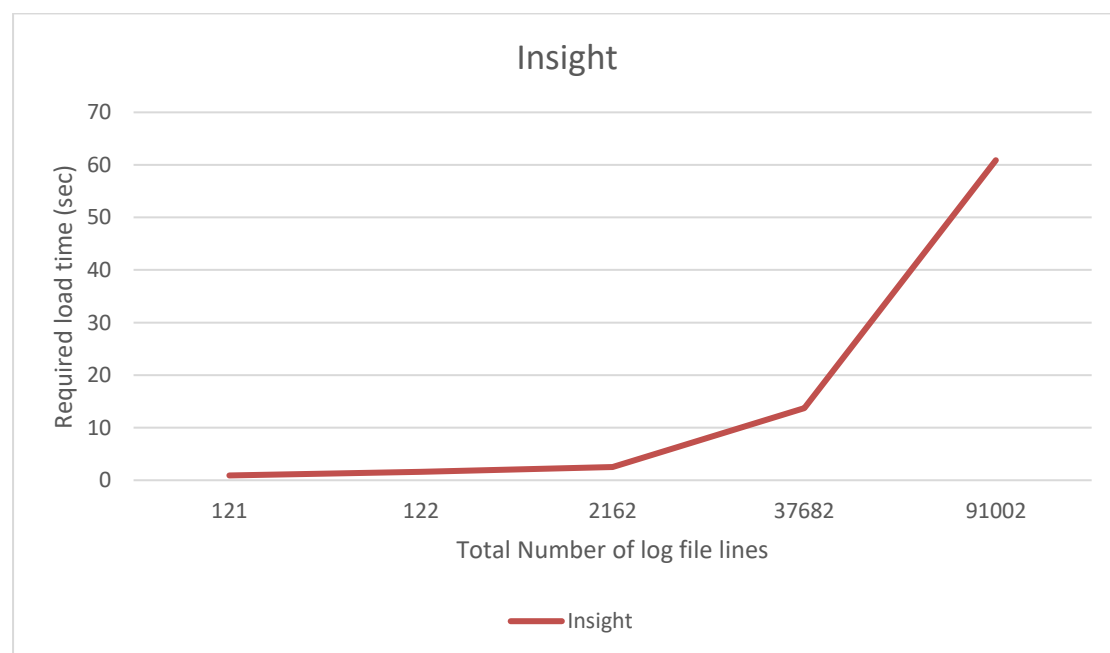
Στη συνέχεια στο σχήμα 5.2 παρουσιάζεται η αντιστοιχία απαιτούμενου χρόνου και μεγέθους των σχημάτων για το εργαλείο Εκάτη.



Σχήμα 5.2 Περιέχονται τα αποτελέσματα για πέντε σχήματα τα οποία εξετάστηκαν στο εργαλείο Εκάτη ως προς το χρόνο που απαιτείται για την πλήρη φόρτωση τους.

Στο σχήμα 5.2 απεικονίζεται ο χρόνος που απαιτείται για την πλήρη φόρτωση των σχημάτων βάσεων δεδομένων. Τον y-άξονα καταλαμβάνει ο απαιτούμενος χρόνος και η μονάδα μέτρησης του είναι τα δευτερόλεπτα. Το x-άξονα καταλαμβάνει ο συνολικός αριθμός γραμμών του κάθε σχήματος καθώς αυτή η μονάδα προσφέρει καλύτερα και πιο πραγματικά αποτελέσματα από την μονάδα συνολικός αριθμός αρχείων του σχήματος, καθώς ένα σχήμα ενδέχεται να έχει πολλά αρχεία αλλά με λίγες γραμμές και άρα ως αποτέλεσμα να απαιτήσει πιο λίγο χρόνο από ένα σχήμα με λίγα αλλά μεγάλα sql αρχεία. Τέλος, για το σχήμα `intelliants_subrion`, το οποίο περιέχει 266 sql αρχεία των οποίων οι γραμμές αθροίζονται σε 672.471, ο συνολικός απαιτούμενος χρόνος αυξάνεται και σκαρφαλώνει στα 25,31 δευτερόλεπτα.

Στη συνέχεια στο σχήμα 5.3 παρουσιάζεται η αντιστοιχία απαιτούμενου χρόνου και μεγέθους των log files για το εργαλείο Insight.



Στο σχήμα 5.3 απεικονίζεται ο συνολικός χρόνος που απαιτείται για να φορτωθούν τα περιεχόμενα των log files στα text panels του εργαλείου Insight. Όπως παρουσιάζεται στο σχήμα, με την αύξηση των γραμμών του log file ο απαιτούμενος χρόνος δεν αυξάνεται ραγδαία. Ωστόσο, στις δύο τελευταίες περιπτώσεις όπου το log file περιέχει 37.682 και 91.002 γραμμές, ο απαιτούμενος χρόνος αυξάνεται, καθώς ο αριθμός των λαθών που πρέπει να διαχωριστούν στις κατηγορίες σημαντικά και λιγότερο σημαντικά, είναι πολύ μεγάλος, απαιτώντας με αυτό τον τρόπο παραπάνω χρόνο φόρτωσης. Το τελευταίο

σχήμα όπου απαιτεί και τον περισσότερο χρόνο φόρτωσης αποτελεί μια ειδική περίπτωση, καθώς ανάμεσα σε έναν μεγάλο αριθμό από σχήματα που εξετάστηκαν είναι το μοναδικό που ο αριθμός των sql αρχείων ξεπερνάει το 100, καθώς διαθέτει 266 αρχεία. Τέλος, κατά μέσο όρο, λαμβάνοντας υπόψιν ότι εξετάστηκαν 100 σχήματα με μέγεθος αρχείων που κυμαίνονταν από 2 αρχεία έως και 266, μετρήθηκε ως **1.6** δευτερόλεπτα για το εργαλείο Εκάτη και αντίστοιχα **1.42** δευτερόλεπτα για το εργαλείο Insight.

Κεφάλαιο 6. Επίλογος

6.1 Σύνοψη και συμπεράσματα

Το αντικείμενο αυτής της διπλωματικής ήταν η υποβοήθηση της διαδικασίας αυτόματης εξαγωγής της ιστορίας και των αλλαγών ενός σχήματος μιας βάσης δεδομένων. Για τον σκοπό αυτό επεκτάθηκε ένα υπάρχον εργαλείο και κατασκευάστηκε ένα νέο, με σκοπό την υποβοήθηση της εξαγωγής μιας ακριβέστερης ιστορίας του σχήματος της βάσης.

Το εργαλείο που επεκτάθηκε ονομάζεται Εκάτη [Sku_13] και ο σκοπός του είναι να λαμβάνει ως είσοδο μια λίστα με τα αρχεία των διαφορετικών εκδοχών του σχήματος της βάσης δεδομένων, όπως προέκυψαν στην πορεία του χρόνου, και να παράγει ως έξοδο (α) τη λίστα με τις αλλαγές μεταξύ γειτονικών στο χρόνο εκδοχών του σχήματος, και κατά συνέπεια την πλήρη λίστα αλλαγών στην ιστορία του σχήματος της βάσης, (β) μετρήσεις σχετικά με το πότε και πώς άλλαξε το σχήμα σε κάθε μετάβαση από μία εκδοχή στην επόμενη, (γ) μετρήσεις για την εξελικτική συμπεριφορά των πινάκων του σχήματος. Επειδή κάθε εκδοχή της ιστορίας ερχόταν με την μορφή Data Definition Language (DDL) αρχείου σε SQL, το εργαλείο περνούσε από ένα parser κάθε τέτοιο αρχείο και εντόπιζε τις εντολές δημιουργίας πινάκων. Η διαδικασία αυτή προφανώς είχε σφάλματα επεξεργασίας (parsing errors). Η πρώτη συνεισφορά της Διπλωματικής αυτής ήταν η οργανωμένη αντιμετώπιση των parsing errors της Εκάτης, με την σύλληψη εξαιρέσεων που προέκυπταν στον κώδικα, και την καταγραφή των λαθών σε ένα log file με τρόπο που στη συνέχεια να επιτρέπει να τα εκμεταλλευόμαστε.

Η δεύτερη συνεισφορά της Διπλωματικής αυτής ήταν η σχεδίαση και υλοποίηση ενός εργαλείου απεικόνισης των παραγόμενων σφαλμάτων ώστε να μπορεί ο αναλυτής να προχωρήσει ακόμα ένα βήμα στην κατανόηση των σφαλμάτων αλλά και να τα διαχωρίσει σε πιο σημαντικά και σε λιγότερα σημαντικά λάθη όπως είναι τα συντακτικά λάθη. Το εργαλείο αυτό ονομάστηκε Insight (Ενόραση). Το εργαλείο Insight επιτρέπει την ομαδοποίηση των σφαλμάτων (καθώς πολλά από αυτά επαναλαμβάνονται σε πολλές εκδοχές του DDL αρχείου) ώστε να μπορεί ο αναλυτής να καταλάβει τα «μοναδικά» σημεία αποτυχίας του parsing. Επιπρόσθετα, το εργαλείο αυτό προσφέρει την δυνατότητα τροποποίησης των αρχείων sql που συμμετέχουν στο σχήμα της βάσης

προσφέροντας με αυτόν τον τρόπο παραλληλισμό των ενεργειών, παρατήρηση του σφάλματος και ταυτόχρονα αντιμετώπιση του.

Από τα αποτελέσματα των πειραμάτων που πραγματοποιήθηκαν στην συγκεκριμένη εργασία προκύπτουν τα παρακάτω:

- 1) Με την αύξηση κατά τάξεων του μεγέθους της εισόδου, τα εργαλεία Εκάτη και Ενόραση δεν παρουσιάζουν σημαντική αύξηση στον χρόνο φόρτωσης της πληροφορίας. Ωστόσο, για πολύ μεγάλα σχήματα παρατηρείται σημαντικός απαιτούμενος χρόνος φόρτωσης.
- 2) Ο παραλληλισμός των ενεργειών απεικόνιση του σφάλματος από το εργαλείο Insight και άμεση αντιμετώπιση του αποτελεί μια από τις μεγαλύτερες αρετές του εργαλείου καθώς μειώνει κατά πολύ την έγκαιρη διόρθωση των σφαλμάτων που λαμβάνουν χώρα στα sql αρχεία της βάσης.

6.2 Μελλοντικές επεκτάσεις

Σε αυτό το σημείο θα παρουσιαστεί μια λίστα με μελλοντικές επεκτάσεις των εργαλείων Εκάτη και Ενόραση ώστε να προσφέρεται ένα πιο ευρύ σύνολο λειτουργιών και εργαλείων που ως αποτέλεσμα θα βοηθήσει σε μεγαλύτερο βαθμό τον αναλυτή βάσεων δεδομένων. Αναλυτικά:

- 1) Επί του παρόντος, στο εργαλείο Insight προσφέρονται δύο είδη ομαδοποιήσεων των περιεχόμενων του log file, η ομαδοποίηση με βάση το ίδιο σφάλμα που εντοπίζεται στο ίδιο αρχείο και η ομαδοποίηση με βάση το ίδιο σφάλμα το οποίο εντοπίζεται και σε διαφορετικά αρχεία sql. Ωστόσο, το εργαλείο αυτό μπορεί να επεκταθεί εφοδιάζοντας τον αναλυτή και με άλλα είδη εργαλείων ομαδοποίησης στα οποία συγκαταλέγονται ο διαμερισμός των σφαλμάτων σε ομάδες που έχουν δημιουργηθεί εκ των προτέρων και η απεικόνιση τους. Οι ομάδες αυτές θα μπορούσαν να αφορούν κατηγορίες εντολών της γλώσσας DDL όπως για παράδειγμα CREATE TABLE, ALTER TABLE ή DROP STATEMENT.
- 2) Ταξινόμηση των ομαδοποιημένων σφαλμάτων κατά φθίνουσα σειρά με βάση τον αριθμό απεικονίσεων του καθενός σφάλματος. Πρώτο στοιχείο δηλαδή της σειράς ταξινόμησης θα τοποθετείται εκείνο το σφάλμα το οποίο εντοπίζεται τις περισσότερες φορές στα αρχεία του σχήματος της βάσης, θέτοντας το ως το πιο σημαντικό.

- 3) Δημιουργία ενός επιλογέα στοιχείων ο οποίος θα ενεργεί επάνω στο log file επιτρέποντας σε πρώτο στάδιο την επιλογή κάποιου σφάλματος από τον αναλυτή, και σε δεύτερο στάδιο την εύρεση του σφάλματος αυτού στα αρχεία του σχήματος και αυτόματη αντιμετώπιση του με την προσθήκη των συμβόλων σχολίων πριν από την γραμμή του σφάλματος, μετατρέποντας με αυτό τον τρόπο τα σφάλματα ως σχόλια. Η αντιμετώπιση αυτή θα βοηθήσει τον αναλυτή σε μετέπειτα επεξεργασία των sql αρχείων να εντοπίσει με μεγαλύτερη ευκολία τα σημεία στα οποία είναι αναγκαίο να παρέμβει, διορθώνοντας κατά αυτόν τον τρόπο το κάθε λάθος.
- 4) Επέκταση της γραμματικής του εργαλείου Εκάτη ώστε να λάβει υπόψιν και πολλές άλλες λειτουργίες και εντολές που στην παρούσα στιγμή δεν μπορούν να εκτελεστούν από το εργαλείο Εκάτη και ως αποτέλεσμα παράγονται σφάλματα.
- 5) Προσθήκη επιπλέον λειτουργίας στο ενσωματωμένο editor η οποία θα επιτελεί αυτόματο εντοπισμό των σημαντικών εντολών της DDL γλώσσας και θα προτρέπει σε επισήμανση τους με διαφορετικό χρώμα ανάλογα με το είδος της εντολής. Η λειτουργία αυτή θα βοηθήσει τον αναλυτή στον διαχωρισμό των εντολών της γλώσσας από τα υπόλοιπα περιεχόμενα του sql αρχείου.

Βιβλιογραφία

- [LMR+97] Meir M. Lehman, Juan F. Ramil, Paul Wernick, Dewayne E. Perry, Wladyslaw M. Turski. Metrics and Laws of Software Evolution – The Nineties View. 4th IEEE International Software Metrics Symposium (METRICS 1997), November 5-7, 1997.
- [Sjøb91] Dag Sjøberg, The Thesaurus – A Tool for Meta Data Management, Technical Report FIDE/91/6, ESPRIT Basic Research Action, Project Number 3070 --- FIDE, February 1991.
- [Sjøb93] Dag Sjøberg. “Quantifying schema evolution.” Information and Software Technology 35.1 (1993): 35-44.
- [Skou13] Ιωάννης Σκουλής. Ανάλυση της εξέλιξης σχήματος για βάσεις δεδομένων σε λογισμικό ανοικτού κώδικα. Μεταπτυχιακή Εργασία εξειδίκευσης. Πανεπιστήμιο Ιωαννίνων, Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής, pp: 1-124, Σεπτέμβριος 2013.
- [Antlr_IO] Input/Output της διαδικασίας μεταγλώττισης με χρήση εργαλείου ANTLR. Lexical Analysis, Parsing, Terence Par, Ιούλιος 2010
- [AntlrPTut] Αναλυτικό Tutorial (Antlr v4 with Terrence Parr) με θέμα την χρήση και επεξήγηση της λειτουργίας του εργαλείου ANTLR από τον ίδιο το δημιουργό. Terence Parr, Φεβρουάριος 2013
- [GTom20] G.Tomassetti ANTLR Tutorial. Available at <https://tomassetti.me/antlr-mega-tutorial/> . Last checked 2020/05/29
- [Parr20] T.Parr. ANTLR. Available at [antr.org](http://antlr.org). Last checked 2020/05/29