

**ΕΝΣΩΜΑΤΩΣΗ ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑΣ ΤΗΣ
ΠΛΑΤΦΟΡΜΑΣ SPARK ΣΕ ΣΥΣΤΗΜΑ ON-LINE
ANALYTICAL PROCESSING**

Καδόγλου Κωνσταντίνος

Διπλωματική Εργασία

Επιβλέπων: Π. Βασιλειάδης

Ιωάννινα, Ιούλιος 2020



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

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

Ευχαριστίες

Ολοκληρώνοντας τη διπλωματική εργασία θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Π. Βασιλειάδη, για την καθοδήγηση και την υποστήριξή του καθ' όλη τη διάρκεια της εργασίας μου. Επίσης, θα ήθελα να ευχαριστήσω τους φίλους μου για την στήριξή τους. Τέλος θα ήθελα να ευχαριστήσω την οικογένειά μου για την μεγάλη υποστήριξη που μου προσφέρανε όλα αυτά τα χρόνια.

20 Ιουλίου 2020

Καδόγλου Κωνσταντίνος

Περίληψη στα ελληνικά

Το Apache Spark αποτελεί ένα state-of-the-art σύστημα για την γρήγορη ανάλυση δεδομένων σε συστάδες υπολογιστών. Στην παρούσα διπλωματική εργασία ενσωματώσαμε τη λειτουργικότητα του συστήματος Apache Spark στο σύστημα DelianCube Engine, ένα σύστημα On-Line Analytical Processing. Με τη χρήση του DelianCube Engine οργανώνουμε τα δεδομένα σε μορφή κύβου, μεταφράζουμε το ερώτημα OLAP που επιθυμούμε σε SQL query και εκτελούμε το ερώτημα σε ένα cluster με τη βοήθεια του Spark. Συγκεκριμένα από το Apache Spark χρησιμοποιήθηκε η βιβλιοθήκη Spark SQL που παρέχει. Στο σύστημα DelianCube Engine επανασχεδιάστηκε το γραφικό περιβάλλον με το οποίο αλληλοεπιδρά ο χρήστης, ώστε να μπορεί πλέον να επιλέξει εάν θα εκτελέσει το σύστημα με τη χρήση μίας βάσης δεδομένων RDBMS ή με τη χρήση του συστήματος Spark. Τέλος, έγιναν μετρήσεις ως προς τον χρόνο εκτέλεσης μεταξύ των δύο συστημάτων και αποδείχθηκε ότι σε μεγάλα δεδομένα το Spark είναι πολύ πιο γρήγορο από ένα τυπικό σύστημα RDBMS.

Λέξεις Κλειδιά: Spark, On-Line Analytical Processing

Abstract

Apache Spark is a state-of-the-art system for fast analyzing data in a cluster. In this thesis we have incorporated the functionality of Apache Spark on the DelianCube Engine, a system of On-Line Analytical Processing. Having integrated Spark, we took advantage of the Spark SQL library. With the use of DelianCube Engine we organize the data in cubes, we translate the OLAP query we want to run in a SQL query, and, we execute the query using Spark.

In the DelianCube Engine, we redesigned the graphical user interface via which the user interacts, having now the ability to choose whether he wants to run the system with the use of an RDBMS database or with the Spark system. Finally, measurements were made in terms of runtime between the two systems and it turned out that, in big data, Spark is much faster than a standard RDBMS system.

Keywords: Spark, On-Line Analytical Processing

Πίνακας περιεχομένων

Κεφάλαιο 1. Εισαγωγή	1
1.1 Αντικείμενο της διπλωματικής	1
1.2 Οργάνωση του τόμου	2
Κεφάλαιο 2. Περιγραφή Θέματος.....	5
2.1 Στόχος της εργασίας	5
2.2 Υπόβαθρο τεχνολογιών OLAP	7
2.2.1 <i>Ορολογία</i>	7
2.2.2 <i>Κύβος (Cube)</i>	8
2.2.3 <i>Διαστάσεις (Dimensions)</i>	8
2.2.4 <i>Ιεραρχία Διάστασης (Dimension Hierarchy)</i>	9
2.2.5 <i>Σχεσιακές Παραστάσεις Πολυδιάστατων Πινάκων</i>	9
2.2.6 <i>Λειτουργίες OLAP</i>	10
2.3 Υπόβαθρο για το Apache Spark.....	13
2.3.1 <i>Αρχιτεκτονικό υπόβαθρο του Spark</i>	13
2.3.2 <i>Λειτουργίες RDD</i>	17
2.3.3 <i>Spark SQL & DataSets</i>	19
2.3.4 <i>Spark Core & Cluster Manager</i>	22
Κεφάλαιο 3. Σχεδίαση & Υλοποίηση.....	25
3.1 Σχεδίαση και αρχιτεκτονική λογισμικού	25
3.1.1 <i>Μετατροπή σε Maven Java Project</i>	25
3.1.2 <i>Αλλαγές κώδικα</i>	28
3.1.3 <i>Περιγραφή του μοντέλου του λογισμικού</i>	34
3.2 Σχεδίαση και αποτελέσματα ελέγχου του λογισμικού	43
3.3 Λεπτομέρειες εγκατάστασης και υλοποίησης.....	47
Κεφάλαιο 4. Πειραματική Αξιολόγηση.....	55
4.1 Μεθοδολογία πειραματισμού	55
4.2 Αναλυτική παρουσίαση αποτελεσμάτων.....	58

Κεφάλαιο 5. Επίλογος	65
5.1 Σύνοψη και συμπεράσματα.....	65
5.2 Μελλοντικές επεκτάσεις	65

Κεφάλαιο 1. Εισαγωγή

Οι βάσεις δεδομένων είναι ένας κλάδος της τεχνολογίας, ο οποίος πλέον θεωρείται απαραίτητος σε οποιοδήποτε επιστημονικό και επιχειρηματικό τομέα. Με τον όρο βάση δεδομένων αναφερόμαστε σε μία οργανωμένη δομή δεδομένων από την οποία μπορούμε να εξάγουμε πληροφορίες μέσω κάποιας αίτησης. Οι σχεσιακές βάσεις δεδομένων εμφανίστηκαν το 1970 από τον Edgar Frank Codd, όταν ανακάλυψε το σχεσιακό μοντέλο των βάσεων δεδομένων, για το οποίο τιμήθηκε μια 10ετία αργότερα με το βραβείο Turing. Η επιστήμη των Β.Δ. θεωρείται ένας «ώριμος» κλάδος, σταθερός και ολοκληρωμένος για τον σκοπό που έχει δημιουργηθεί, δέχοντας όλα αυτά τα χρόνια βελτιώσεις, κρατώντας ωστόσο την βασική της λογική σταθερή.

1.1 Αντικείμενο της διπλωματικής

Μία από τις εξελίξεις που ακολούθησαν και βασίστηκαν πάνω στις σχεσιακές βάσεις δεδομένων είναι τα συστήματα Online analytical processing (OLAP). Ένα σύστημα OLAP είναι ένα σύστημα που προσπαθεί να βελτιώσει την ταχύτητα και το μέγεθος της εξαγωγής της πληροφορίας σε ερωτήματα πολυδιάστατων δεδομένων, προσφέροντας σημαντικές λειτουργίες σε έναν αναλυτή δεδομένων. Για το λόγο αυτό, το OLAP αποτελεί υποκατηγορία της επιστημονικής περιοχής του Business Intelligence (BI), η οποία περιλαμβάνει στρατηγικές και τεχνολογίες που χρησιμοποιούν οι επιχειρήσεις για την ανάλυση επιχειρησιακών δεδομένων.

Ένα άλλο σύστημα που έχει να κάνει με ανάλυση δεδομένων είναι το Apache Spark. Πιο συγκεκριμένα, το Apache Spark είναι ένα ενιαίο σύστημα για παράλληλη ανάλυση μεγάλων δεδομένων σε συστάδες υπολογιστών. Το Apache Spark προσφέρει, όπως και τα συστήματα OLAP, μια πληθώρα από εργαλεία και λειτουργίες σε έναν αναλυτή δεδομένων, ώστε να είναι η διαδικασία ανάλυσης των δεδομένων πιο εύχρηστη και διαδραστική.

Αναλύοντας παραπάνω τα δύο συστήματα, μπορούμε να πούμε ότι η παρούσα διπλωματική εργασία εντάσσεται στην βελτίωση της ανάλυσης μεγάλων δεδομένων. Το αντικείμενο, λοιπόν, της διπλωματικής εργασίας είναι η ενσωμάτωση του συστήματος Apache Spark στο υπάρχων project DelianCube Engine. Το DelianCube Engine είναι ένα project που σχεδιάστηκε από τον καθηγητή Παναγιώτη Βασιλειάδη (και δέχτηκε βελτιώσεις από προηγούμενες διπλωματικές εργασίες), το οποίο υλοποιεί ένα σύστημα OLAP και προσφέρει την εκτέλεση ενός αιτήματος SQL σε μία βάση δεδομένων τύπου RDBMS. Στόχος της διπλωματικής εργασίας είναι η χρήση του DelianCube Engine για την εξαγωγή αιτημάτων SQL τύπου OLAP, η περαιτέρω βελτιστοποίηση των SQL αιτημάτων και η εκτέλεσή τους μέσω του Apache Spark σε ένα cluster.

Παραπάνω αναφερθήκαμε στο σύστημα OLAP ως μια προέκταση των σχεσιακών βάσεων δεδομένων. Με την ενσωμάτωση του συστήματος Apache Spark αυτό που επιδιώκουμε είναι η βελτίωση του χρόνου εκτέλεσης των αιτημάτων σε σύγκριση με την κλασική τους εκτέλεση σε μία σχεσιακή βάση δεδομένων. Το ζήτημα αυτό είναι σημαντικό στην επιστήμη ανάλυσης δεδομένων, καθώς τα δεδομένα που παρέχονται για ανάλυση αυξάνονται εκθετικά με αποτέλεσμα την ολοένα και μεγαλύτερη καθυστέρηση της εξαγωγής των αιτημάτων. Σε ένα οικοσύστημα που απαιτείται γρήγορη λήψη αποφάσεων, η γρηγορότερη ανάλυση δεδομένων αποτελεί σημαντικό παράγοντα.

Η αξιολόγηση της συνεισφοράς της εργασίας έγινε με την καταμέτρηση των χρόνων εκτέλεσης SQL αιτημάτων μεταξύ της αρχικής προσέγγισης που πρόσφερε το DelianCube Engine, την εκτέλεση του αιτήματος μέσω ενός RDBMS, και την εκτέλεση σε ένα cluster που προσφέρει η ενσωμάτωση του Apache Spark. Οι συγκρίσεις και τα ακριβή αποτελέσματα που προκύψανε απέδειξαν ότι το Apache Spark μπορεί να αποτελέσει σημαντική προσθήκη για την βελτιστοποίηση των χρόνων εκτέλεσης στην ανάλυση μεγάλων δεδομένων.

1.2 Οργάνωση του τόμου

Παρακάτω θα αναλύσουμε την οργάνωση του τόμου που θα ακολουθήσει και για το τι διαπραγματεύεται σε κάθε κεφάλαιο.

Στο 2^o Κεφάλαιο αναφερόμαστε στον στόχο της διπλωματικής εργασίας, και αναλύουμε τις τεχνολογίες που θα χρησιμοποιήσουμε για την επίτευξή του. Θα αναφερθούμε αναλυτικά για την τεχνολογία On-Line Analytical Processing (OLAP) και για το Apache Spark.

Στο 3^o Κεφάλαιο περιγράφουμε, σε τεχνικό επίπεδο, τις αλλαγές και προσθήκες που κάναμε στο DelianCube Engine για να επιτευχθεί ο στόχος της εργασίας. Επίσης παρουσιάζονται τα Junit tests που διεξάχθηκαν για τον έλεγχο της σωστής λειτουργίας του συστήματος DelianCube Engine. Τέλος γίνεται μία αναφορά στον τρόπο εγκατάστασης του συστήματος ώστε να εκτελεστεί σε ένα cluster.

Στο 4^o Κεφάλαιο παρουσιάζουμε τις μετρήσεις και τα αποτελέσματα που έγιναν για την σύγκριση μεταξύ της χρήσης μίας βάσης δεδομένων RDBMS και του Spark.

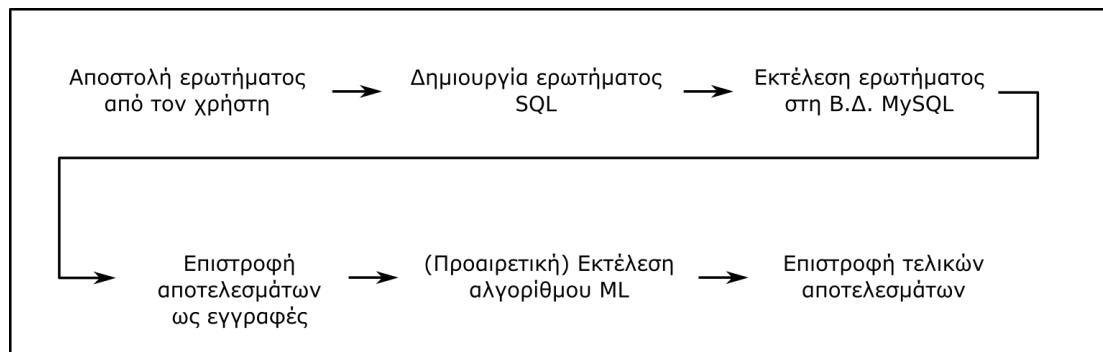
Στο 5^o Κεφάλαιο συνοψίζουμε τον στόχο και την επίτευξη της διπλωματικής εργασίας και παρέχουμε μερικές προτάσεις για μελλοντικές επεκτάσεις που θα μπορούσαν να υλοποιηθούν στο παρόν project.

Κεφάλαιο 2. Περιγραφή Θέματος

Στο Κεφάλαιο αυτό θα περιγράψουμε πιο τεχνικά τους στόχους της διπλωματικής εργασίας. Επιπλέον θα γίνει μια αναλυτική περιγραφή των δύο συστημάτων που θα χρησιμοποιήσουμε, OLAP και Apache Spark.

2.1 Στόχος της εργασίας

Στόχος της διπλωματικής εργασίας είναι η ενσωμάτωση λειτουργικότητας του συστήματος Apache Spark σε σύστημα On-Line Analytical Processing (OLAP). Το DelianCube Engine, του οποίου τον κώδικα θα επεκτείνουμε, είναι ένα πρόγραμμα με γραφικό περιβάλλον, γραμμένο στην γλώσσα προγραμματισμού Java. Το project έχει σχεδιαστεί ώστε να υλοποιεί τις λειτουργίες OLAP και να εκτελεί τα αιτήματα σε μία Β.Δ. MySQL. Μετά την εκτέλεση και την εξαγωγή των αποτελεσμάτων το πρόγραμμα DelianCube Engine, μπορεί επίσης να εφαρμόσει έναν από τους αλγόριθμους μηχανικής μάθησης που προσφέρει, και στη συνέχεια να εξάγει τα δεδομένα σε ένα νέο αρχείο. Τα τελικά αποτελέσματα συνοδεύονται ή επισημειώνονται από τα αποτελέσματα των ευφυών αλγορίθμων που εκτελέστηκαν σε αυτά. Στην περίπτωση που το πρόγραμμα εκτελεστεί μέσω του γραφικού περιβάλλοντος επιστρέφεται ένα νέο παράθυρο με τα αποτελέσματα. Στο Σχήμα 2.1.1 μπορούμε να δούμε μια απλή σχεδίαση με την παραπάνω ροή εκτέλεσης.



Σχήμα 2.1.1: Μια απλή αναπαράσταση της ροής εκτέλεσης του προγράμματος DelianCube Engine.

Σκοπός μας είναι να ενσωματώσουμε το Apache Spark και να εκμεταλλευτούμε λειτουργίες που αφορούν τη φόρτωση των δεδομένων στην βελτιωμένη αρχιτεκτονική δομή που προσφέρει το Apache Spark, τη διαμοίραση των δεδομένων σε ένα cluster, την βελτιστοποίηση του SQL ερωτήματος που παρέχει καθώς και την εκτέλεση του ερωτήματος SQL σε ένα cluster. Επιπλέον θα χρειαστεί να κάνουμε refactoring τον κώδικα του DelianCube Engine σε υψηλότερο επίπεδο αφαίρεσης για να προσθέσουμε το σύστημα Apache Spark, δίχως να χάσουμε την λειτουργικότητα μέσω RDBMS που προσφέρει ήδη, αλλά και για να είναι μελλοντικά ευκολότερη η προσθήκη επιπλέον συστημάτων.

Τέλος, θα αναφερθούμε σε μερικά σημεία που θεωρούμε ως αδυναμία στο υπάρχον σύστημα και πως αυτά θα βελτιωθούν με την προσθήκη του Apache Spark. Αρχικά η χρήση ενός RDBMS απαιτεί τις απαραίτητες εγκαταστάσεις ενός συστήματος RDBMS, η σχεδίασή του και η φόρτωση των δεδομένων του, μια διαδικασία που μπορεί να είναι χρονοβόρα αλλά και πολύπλοκη. Με την χρήση των Datasets, μια δομή που προσφέρει το Apache Spark, η διαδικασία αυτή απλοποιείται καθώς τα δεδομένα φορτώνονται κατά την εκτέλεση του προγράμματος, και μάλιστα πιο γρήγορα. Επίσης η χρήση του υπάρχοντος συστήματος εκτελεί ένα ερώτημα σε έναν μόνο κόμβο. Με τη χρήση του Apache Spark, ένα σύστημα που προσφέρει δυνατότητα clustering, αναμένουμε να δούμε βελτίωση στους χρόνους εκτέλεσης ερωτημάτων με μεγάλα δεδομένα.

2.2 Υπόβαθρο τεχνολογιών OLAP

Το 1970 ο Edgar Frank Codd ανακάλυψε το σχεσιακό μοντέλο των βάσεων δεδομένων, για το οποίο τιμήθηκε μια 10ετία αργότερα με το βραβείο Turing. Τη δεκαετία του 1990 με την επανάσταση των προσωπικών Η/Υ, η συσσώρευση δεδομένων άρχισε να γίνεται με όλο και μεγαλύτερους ρυθμούς. Προκειμένου λοιπόν να μπορέσουν να αξιοποιηθούν όλα τα νέα δεδομένα, δημιουργήθηκε ένα νέο μοντέλο, το πολυδιάστατο μοντέλο δεδομένων. Το μοντέλο αυτό έχει ως στόχο την ανάλυση των δεδομένων με τρόπους που είναι γρήγοροι, απλοί και οικείοι στους τελικούς χρήστες. Παρακάτω θα κάνουμε μια ανάλυση του μοντέλου των πολυδιάστατων δεδομένων και των όρων που το απαρτίζουν συγκεντρώνοντας πληροφορίες από τις πηγές [JPT10], [ORA19] και [WOC19].

2.2.1 Ορολογία

OLAP: Το OLAP είναι το ακρώνυμο του όρου On-Line Analytical Processing, όπου επικεντρώνεται στην ανάλυση δεδομένων. Είναι μια προσέγγιση που επιτρέπει στους τελικούς χρήστες να θέσουν εύκολα και να λάβουν μια γρήγορη απάντηση σε multi-dimensional analytical (MDA) ερωτήματα.

Business Intelligence: Με τον όρο Business Intelligence αναφερόμαστε συγκεντρωτικά σε τεχνικές που χρησιμοποιούνται με τη βοήθεια των τεχνικών OLAP και εξόρυξης δεδομένων προκειμένου να βελτιώσουν την αποδοτικότητα μιας επιχείρησης (περισσότερες πωλήσεις, αύξηση κέρδους, μείωση εξόδων κτλ.).

Data Warehouse: Είναι μια «αποθήκη» από ενσωματωμένα επιχειρηματικά δεδομένα. Περιέχει μια κεντρική συλλογή ομογενοποιημένων δεδομένων, τα οποία συλλέγονται αφενός από δεδομένα που προέρχονται και τα οποία ενσωματώνονται σε μία κοινή βάση δεδομένων και αφετέρου από δεδομένα εξωτερικών πηγών που επηρεάζουν την επιχείρηση.

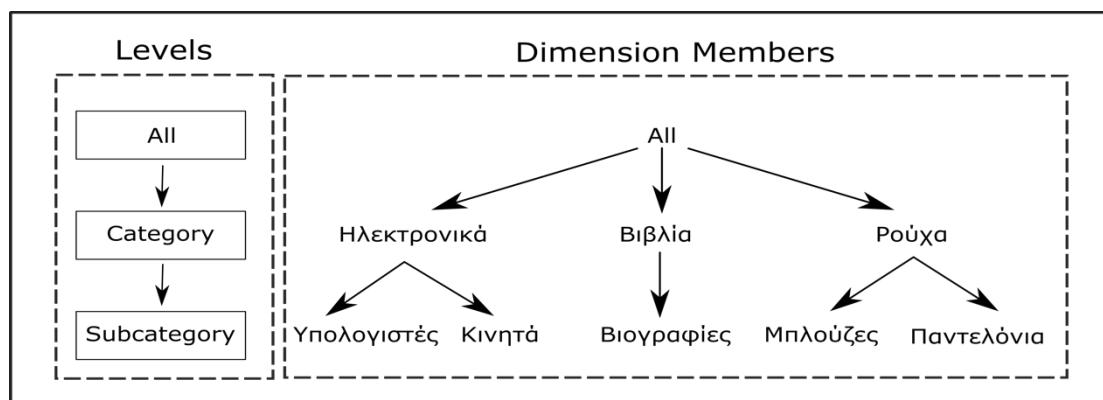
ETL: Είναι τα αρχικά των λέξεων Extract-Transform-Load και εκφράζουν την διαδικασία των τριών βημάτων για την φόρτωση δεδομένων σε ένα data warehouse. (i) Εξάγουμε δεδομένα από διαφορετικές πηγές, (ii) μετασχηματίζουμε τα δεδομένα σε μία κοινή μορφή και σημασιολογία και (iii) φορτώνουμε τα δεδομένα στο data warehouse.

2.2.2 Κύβος (Cube)

Ο υπερκύβος (hypercube) ή για την απλότητα κύβος (cube) είναι μια πολυδιάστατη δομή δεδομένων που χρησιμοποιείται για την αποθήκευση και ανάλυση δεδομένων. Μπορεί κανείς να σκεφτεί τους κύβους ως μια επέκταση των spreadsheets, έχοντας την δυνατότητα να περιλαμβάνουν πολλές διαστάσεις. Ακόμη, ο κύβος έχει ενσωματωμένη την ιδέα της ιεραρχίας που δεν υφίσταται στα spreadsheets. Ένας κύβος αποτελείται από μοναδικά αναγνωρισμένα **κελιά** (cells). Κάθε κελί περιέχει μια τιμή που καλείται **γεγονός** (fact), το οποίο καταγράφει την σημαντική πληροφορία που χρησιμοποιείται για την ανάλυση και εξαγωγή δεδομένων. Με την σειρά του, κάθε γεγονός περιέχει μία ή περισσότερες αριθμητικές τιμές που καλούνται **μετρήσεις** (measures).

2.2.3 Διαστάσεις (Dimensions)

Η διάσταση (dimension) είναι μια συλλογή από μοναδικές τιμές που προσδιορίζουν και κατηγοριοποιούν τα δεδομένα. Οι τιμές κάθε διάστασης ονομάζονται **dimension members** και καθεμία ανήκει σε ένα συγκεκριμένο **επίπεδο** (level). Στο Σχήμα 2.2.1 βλέπουμε ένα παράδειγμα της διάστασης 'Προϊόν'. Η διάσταση στο παράδειγμα περιέχει 3 επίπεδα με όνομα "All", "Category" και "Subcategory", ταξινομημένα από το ανώτερο στο κατώτερο επίπεδο. Οι τιμές των επιπέδων της ιεραρχίας "Προϊόν" κατανέμονται όπως φαίνεται στα δεξιά του Σχήματος 2.2.1. Π.χ. για το επίπεδο "Category" οι τιμές είναι "Ηλεκτρονικά", "Βιβλία" και "Ρούχα".



Σχήμα 2.2.1: Παράδειγμα σχήματος της διάστασης 'Προϊόν'

2.2.4 Ιεραρχία Διάστασης (Dimension Hierarchy)

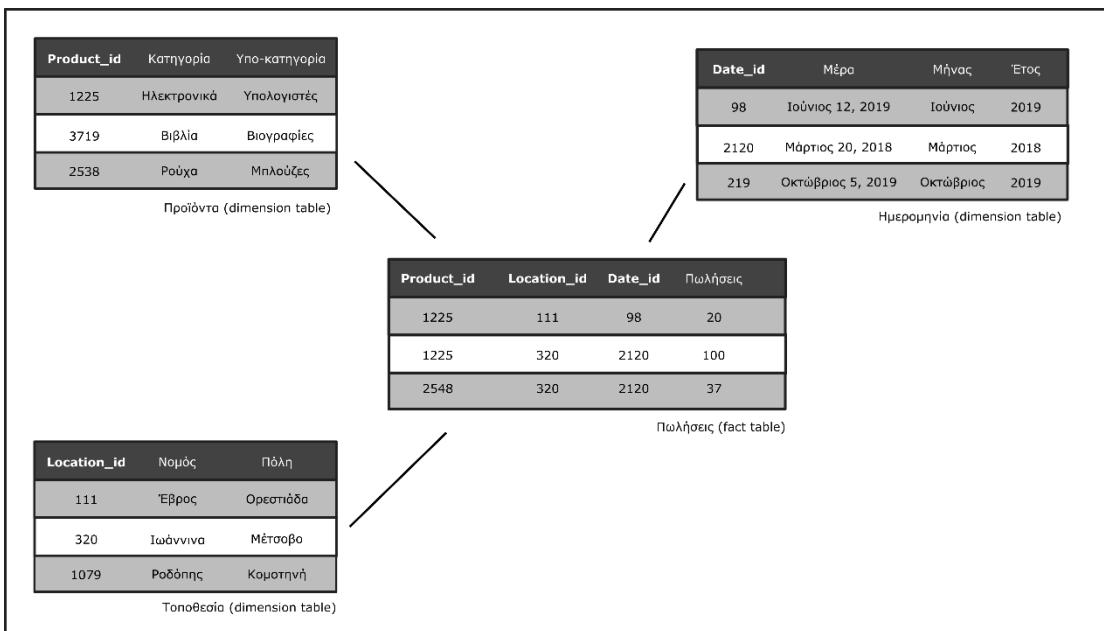
Οι διαστάσεις έχουν ενσωματωμένη την έννοια της **ιεραρχίας** (**dimension hierarchy**). Οι ιεραρχίες χρησιμοποιούνται από τον αναλυτή για την καλύτερη κατανόηση των δεδομένων, δίνοντάς του τη δυνατότητα να περιηγηθεί στα δεδομένα με τις λειτουργίες που θα αναλύσουμε στην ενότητα 2.2.6. Στο παράδειγμα της εικόνας 2.2.1 ωστόσο φαίνεται να υπάρχει μια ιεραρχία δένδρου, το οποίο δεν ισχύει πάντα. Στις περιπτώσεις λοιπόν όπου το επίπεδο του γονέα είναι ίδιο με αυτό του παιδιού θα πρέπει να χρησιμοποιηθεί μια αναπαράσταση που ονομάζεται **πίνακας γέφυρα** (**bridge table**). Ο πίνακας γέφυρα περιέχει πληροφορίες για κάθε μονοπάτι μεταξύ γονέα-παιδιού και αποτελείται από τις παρακάτω στήλες :

- **Ancestor:** Ένα ξένο κλειδί που αναφέρεται στο πρωτεύον κλειδί του πατέρα.
- **Descendant:** Ένα ξένο κλειδί που αναφέρεται στο πρωτεύον κλειδί του παιδιού.
- **Distance:** Ένας ακέραιος που καταγράφει την απόσταση μεταξύ του προγόνου και του απογόνου.
- **Bottom Flag:** Μια λογική τιμή που υποδεικνύει αν ο απόγονος ανήκει στο κατώτερο επίπεδο (δεν έχει απογόνους).
- **Top Flag:** Μια λογική τιμή που υποδεικνύει αν ο πρόγονος ανήκει στο ανώτατο επίπεδο (δεν έχει προγόνους).

2.2.5 Σχεσιακές Παραστάσεις Πολυδιάστατων Πινάκων

Έχοντας αναλύσει τις βασικές αρχές των πολυδιάστατων δεδομένων θα μιλήσουμε για το πως μπορούν αυτού του είδους τα δεδομένα να αποθηκευτούν σε μια σχεσιακή βάση. Υπάρχουν δύο τεχνικές που θα αναλύσουμε, το **σχήμα αστέρα** (star schema) και το **σχήμα χιονονιφάδας** (snowflake schema).

Το σχήμα αστέρα αποτελείται από έναν ή περισσότερους πίνακες γεγονότων και από έναν ή περισσότερους πίνακες διάστασης οι οποίοι σχετίζονται με ξένα κλειδιά. Ο **πίνακας γεγονότων** (**fact tables**) περιέχει τις μετρήσεις και ένα σύνθετο πρωτεύον κλειδί το οποίο αποτελείται από ξένα κλειδιά (ένα για κάθε πίνακα διάστασης). Ο **πίνακας διάστασης** (**dimension tables**) περιέχει όλες τις πληροφορίες για κάθε διάσταση, δηλαδή ένα πρωτεύον κλειδί, τα επίπεδα και τα attributes της διάστασης. Στο Σχήμα 2.2.2 βλέπουμε ένα παράδειγμα από ένα σχήμα αστέρα.



Σχήμα 2.2.2: Σχήμα αστέρα για κύβο πωλήσεων

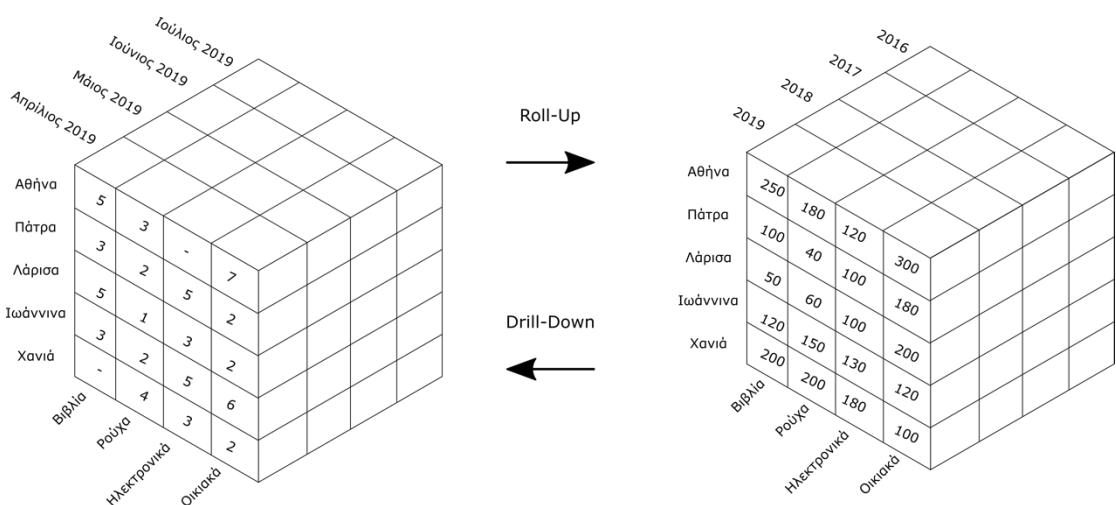
Το σχήμα χιονονιφάδας αποτελεί μια παρόμοια σχεδίαση, με τη διαφορά ότι προσπαθεί να μειώσει τον πλεονασμό διαχωρίζοντας τους πίνακες διάστασης σε επιμέρους πίνακες. Αυτό έχει ως αποτέλεσμα να αποφεύγεται η διπλοτυπία, κάνοντας ωστόσο την εκτέλεση ερωτήσεων σε μία βάση πιο περίπλοκη και χρονοβόρα καθώς χρειάζονται πολλά joins.

2.2.6 Λειτουργίες OLAP

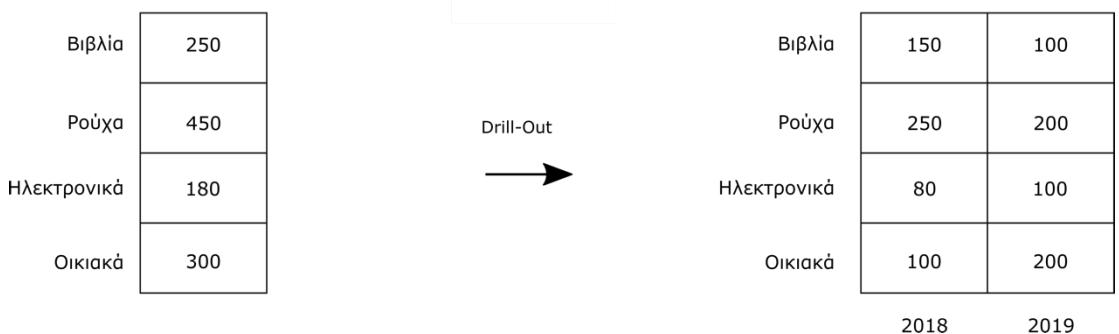
Ο σχεδιασμός της βάσης με κύβους προσφέρει στον αναλυτή λειτουργίες που διευκολύνουν την περιήγηση και την ευκολότερη κατανόηση των δεδομένων. Κάθε λειτουργία που θα περιγράψουμε εφαρμόζεται διαδραστικά σε έναν κύβο που επιθεωρεί ο χρήστης και παράγει έναν άλλον νέο κύβο, ανάλογα με την σημασιολογία της λειτουργίας. Παρακάτω θα αναφέρουμε τις πιο σημαντικές λειτουργίες και θα προβάλουμε την κάθε λειτουργία σε ένα σχήμα:

- **Roll-Up:** Η λειτουργία που επιτρέπει στο χρήστη να μεταβεί από ένα επίπεδο σε ένα ανώτερο επίπεδο. (Σχήμα 2.2.3)
- **Drill-Down:** Η λειτουργία που επιτρέπει στο χρήστη να μεταβεί από ένα επίπεδο σε ένα κατώτερο επίπεδο. (Σχήμα 2.2.3)
- **Drill-Out:** Η λειτουργία αυτή χρησιμοποιείται για την προσθήκη μιας επιπλέον διάστασης. (Σχήμα 2.2.4)
- **Slice:** Η διαδικασία όπου ο χρήστης επιλέγει μία συγκεκριμένη τιμή για μία διάσταση. (Σχήμα 2.2.5)
- **Dice:** Η λειτουργία αυτή επιτρέπει τον χρήστη να επιλέξει συγκεκριμένες τιμές για πολλές διαστάσεις. (Σχήμα 2.2.6)

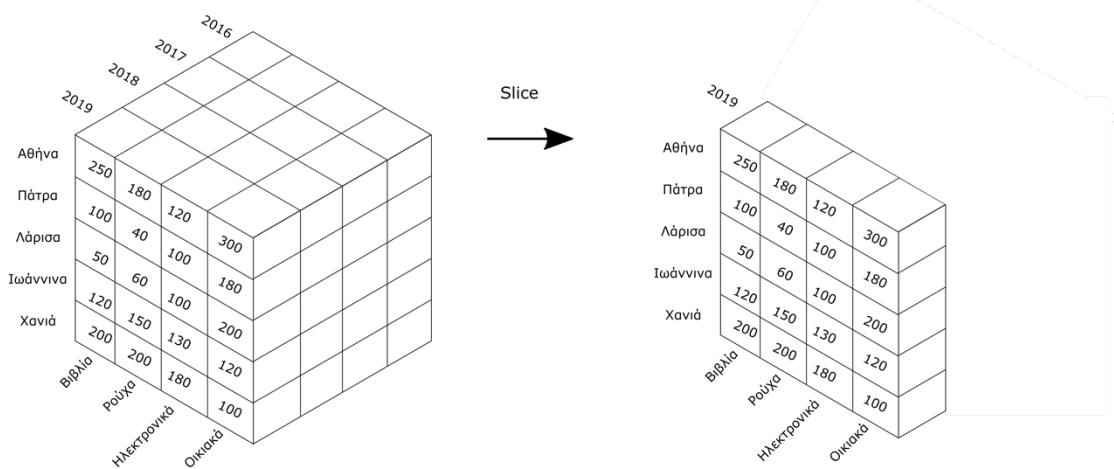
- **Pivot:** Η λειτουργία αυτή επιτρέπει στον αναλυτή να περιστρέψει τον κύβο ώστε να δει τα δεδομένα μιας άλλης πλευράς. (Σχήμα 2.2.7)



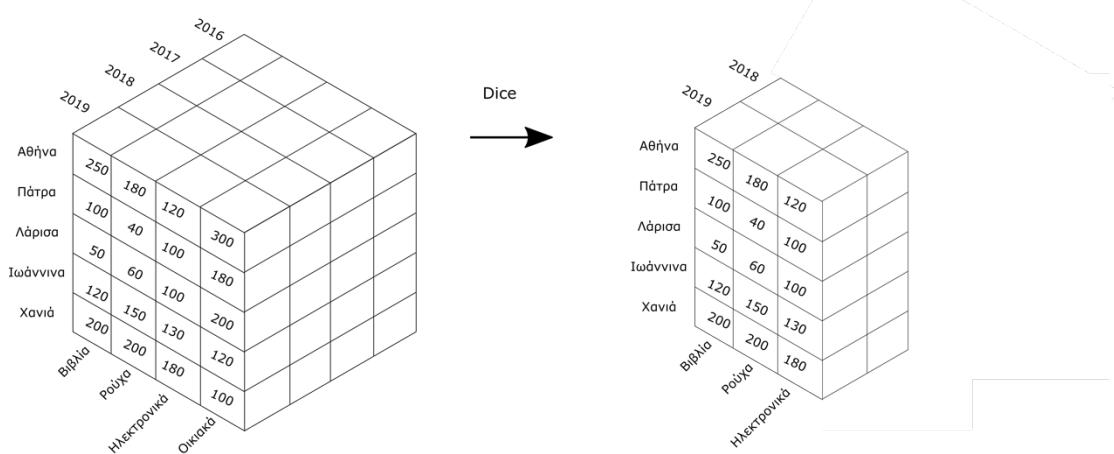
Σχήμα 2.2.3: Λειτουργίες Roll-Up και Drill-Down ως προς την διάσταση του 'Χρόνου'



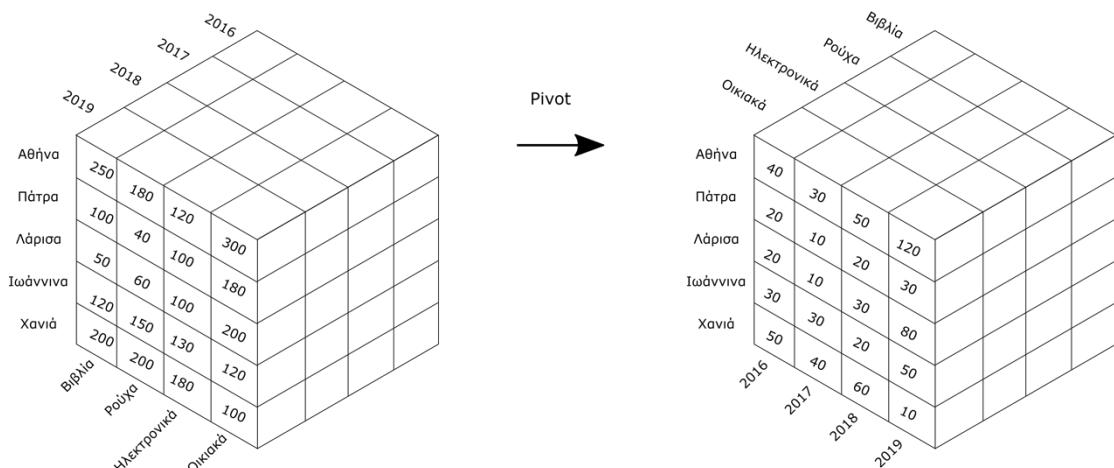
Σχήμα 2.2.4: Λειτουργία Drill-Out για να προστεθεί η διάσταση του 'Χρόνου'



Σχήμα 2.2.5: Λειτουργία slicing επιλέγοντας την συγκεκριμένη τιμή '2019' στη διάσταση του 'Χρόνου'



Σχήμα 2.2.6: Λειτουργία dicing επιλέγοντας τις τιμές '2018' και '2019' στη διάσταση του 'Χρόνου' και οι τιμές 'Βιβλία', 'Ρούχα' και 'Ηλεκτρονικά' στη διάσταση της 'Κατηγορία'



Σχήμα 2.2.7: Λειτουργία περιστροφή των διαστάσεων 'Χρόνος' και 'Κατηγορία'

2.3 Υπόβαθρο για το Apache Spark

Το **Apache Spark** [WAS20], είναι ένα ενιαίο σύστημα για παράλληλη ανάλυση μεγάλων δεδομένων σε συστάδες υπολογιστών. Δημιουργήθηκε το 2009 στο ερευνητικό τμήμα AMPLab του Πανεπιστημίου Μπέρκλεϋ της Καλιφόρνια, από τους Matei Zaharia, Mosharaf Chowdhury, Michael Franklin, Scott Shenker και Ion Stoici. Το 2013 παραδόθηκε στο ίδρυμα Apache Software Foundation, το 2014 κυκλοφόρησε η πρώτη επίσημη έκδοση Apache Spark 1.0 και το 2016 ακολούθησε η δεύτερη έκδοση, το Apache Spark 2.0.

2.3.1 Αρχιτεκτονικό υπόβαθρο του Spark

Το Apache Spark έχει ως αρχιτεκτονικό υπόβαθρο τα **Resilient Distributed Dataset (RDD)**. Συγκεκριμένα, όταν προγραμματίζουμε σε κάποια γλώσσα που υποστηρίζει το Spark, ένα RDD μας επιτρέπει να μεταχειριστούμε το σύνολο των δεδομένων ενός αρχείου σαν μια συλλογή δεδομένων διαμοιρασμένα σε συστάδες υπολογιστών. Επιπλέον το RDD προσφέρει **ανοχή σε σφάλματα**, η οποία επιτυγχάνεται με τη βοήθεια του **RDD lineage graph**, που εκφράζει ένα άκυκλο κατευθυνόμενο γράφημα (DAG), και επιτρέπει τον επανυπολογισμό ενός RDD σε περίπτωση που χαθεί ή καταστραφεί. Το Spark παρέχει ένα API σε τέσσερις (4) γλώσσες προγραμματισμού Scala, Java, Python και R το οποίο δίνει την δυνατότητα στον τελικό χρήστη να εκτελέσει δύο (2) κύριες λειτουργίες στα RDD, transformations και actions, τις οποίες θα αναλύσουμε στην Ενότητα 2.3.2. Το RDD αποτελεί τον πρώτο τύπο δεδομένων με τον οποίο σχεδιάστηκε το Spark, δεν αποτελεί όμως τον μοναδικό τύπο. Το Spark υποστηρίζει δύο (2) επιπλέον τύπους δεδομένων οι οποίοι αναπτύχθηκαν βασιζόμενοι στο RDD αλλά με επιπρόσθετες βελτιστοποιήσεις για την διευκόλυνση του τελικού χρήστη.

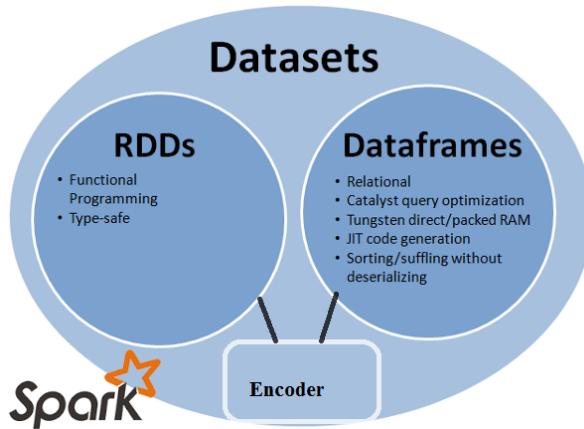
Η πρώτη “επέκταση” του RDD ήρθε με την έκδοση Spark 1.3 η οποία έφερε τα **DataFrames**. Τα DataFrames είναι, όπως και τα RDD, αμετάβλητες συλλογές διαμοιρασμένων αντικειμένων, με τη διαφορά ότι τα δεδομένα είναι οργανωμένα σε ονοματισμένες στήλες όπως θα ήταν μια δομή σε μία σχεσιακή βάση δεδομένων. Τα DataFrames έχουν δύο (2) μεγάλα πλεονεκτήματα ως προς τα RDD. (i) Το πρώτο πλεονέκτημα των DataFrames είναι ότι επιτρέπει στον τελικό χρήστη να εκτελέσει εντολές SQL είτε ως SQL query είτε ως συναρτήσεις αντίστοιχες των λειτουργιών ενός SQL query, χρησιμοποιώντας μάλιστα τον Catalyst Optimizer για βελτιστοποίηση της εκτέλεσης του query. (ii) Το δεύτερο πλεονέκτημα είναι ότι τα δεδομένα αποθηκεύονται

στη μνήμη σε δυαδική μορφή. Με αυτόν τον τρόπο ο τελικός χρήστης γλιτώνει χώρο στη μνήμη (Σχήμα 2.3.3, ισχύει και για τα DataSets), αποφεύγεται η χρήση του Garbage Collector της Java και δεν γίνεται χρήση του Java Serialization.

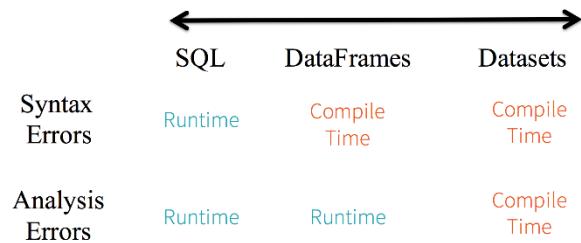
Στην γλώσσα προγραμματισμού που μας ενδιαφέρει, Java, ένα DataFrame εκφράζεται ως ένα *DataSet<Row>* αντικείμενο. Ένα DataFrame μπορεί να φορτώσει δεδομένα από αρχεία CSV, JSON, Parquet, από πίνακες Hive και από βάσεις δεδομένων που υποστηρίζουν JDBC σύνδεση, όπως MySQL, PostgreSQL κ.τ.λ..

Η δεύτερη “επέκταση” του RDD ήρθε με την έκδοση Spark 1.6 η οποία έφερε τα **DataSets**. Ο τύπος DataSet δημιουργήθηκε με σκοπό να προσφέρει έναν συνδυασμό από τα θετικά των RDDs και DataFrames (Σχήμα 2.3.1). Μάλιστα στην έκδοση Spark 2.0 τα DataFrames ενοποιήθηκαν με τα DataSets, και πλέον είναι κοινά αποδεκτό στην κοινότητα του Apache Spark [DFB20] ότι τα DataSets είναι η πιο εύκολη και βέλτιστη επιλογή για την λειτουργία του Spark. Τα DataSets προσφέρουν compile time safety όπως τα RDD, χρησιμοποιούν τον Catalyst Optimizer όπως τα DataFrames και βελτιώνουν την διαχείριση μνήμης, που είδαμε στα DataFrames, με την χρήση εξιδεικευμένων Encoders. Στην ουσία σειριοποιεί τα δεδομένα σε bytes, σε μορφή που επιτρέπει στο Spark να εκτελεί λειτουργίες όπως φιλτράρισμα, ταξινόμηση και hashing χωρίς να χρειάζεται να γίνει πρώτα αποσειριοποίηση των bytes σε μορφή αντικειμένου. Να επισημάνουμε ότι τα DataFrames και DataSets τελικώς εκτελούν τις λειτουργίες σε RDD δεδομένα με τη διαφορά ότι ο τελικός χρήστης δεν χρειάζεται να αναφέρει στον κώδικα του τα RDD. Τα RDD δημιουργούνται διαφανώς στον χρήστη και εκτελούν τις λειτουργίες του χρήστη στο τελικό στάδιο όταν έχουν γίνει όλες οι βελτιστοποιήσεις και έχει βγει το πλάνο εκτέλεσης. Τα DataSets, όπως και τα RDD, ελέγχουν για συντακτικά σφάλματα και σφάλματα ανάλυσης κατά την διάρκεια του compile που βοηθάει ακόμη περισσότερο τον τελικό χρήστη (Σχήμα 2.3.2). Τέλος να αναφέρουμε ότι τα DataSets είναι μέρος της βιβλιοθήκης Spark SQL που συμπεριλαμβάνει το Spark. Περισσότερες λεπτομέρειες σχετικά με το Spark SQL και την εκτέλεση των DataSets σε κώδικα θα δούμε παρακάτω στην Ενότητα 2.3.4.

Εν κατακλείδι έχοντας αναλύσει όλους τους τύπους δεδομένων του Spark, είναι ξεκάθαρο γιατί το DataSet είναι η καλύτερη επιλογή για έναν τελικό χρήστη και γιατί και εμείς επιλέξαμε να χρησιμοποιήσουμε τα DataSets στην υλοποίηση της εργασίας μας. Στο Σχήμα 2.3.4 φτιάξαμε έναν πίνακα με τις τρεις δομές για μία σύντομη σύγκριση όσων αναφέραμε παραπάνω.

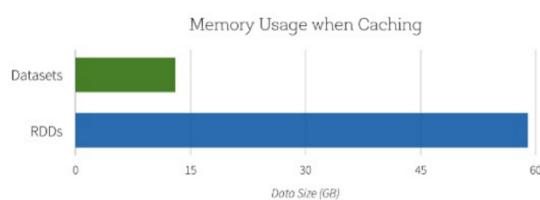


Σχήμα 2.3.1: Σχηματική αναπαράσταση των συνόλων των δομών δεδομένων του Spark
 (πηγή : <https://stackoverflow.com/questions/31508083/difference-between-dataframe-dataset-and-rdd-in-spark>)



Σχήμα 2.3.2: Χρόνος ελέγχου για σφάλματα
 (πηγή : <https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html>)

Space Efficiency



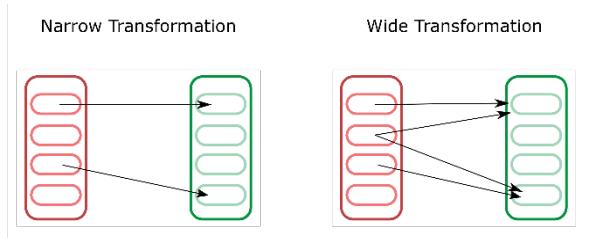
Σχήμα 2.3.3: Σύγκριση memory usage μεταξύ RDDs και DataSets/DataFrames
 (πηγή : <https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html>)

	RDD	DataFrame	DataSet
Immutability	✓	✓	✓
Schema	✗	✓	✓
Plan Optimizer	✗	Catalyst Optimizer	Catalyst Optimizer
Memory Optimizer	✗	Project Tungsten	Spark Encoders
Level	Low	High (build upon RDD)	High (DataFrame Extension)
Typed	✓	✗	✓
SQL Support	✗	✓	✓
Syntax Error	Compile Time	Compile Time	Compile Time
Analysis Error	Compile Time	Runtime	Compile Time

Σχήμα 2.3.4: Πίνακας σύγκρισης των RDD, DataFrame και DataSet

2.3.2 Λειτουργίες RDD

Τα RDD εκτελούν δύο λειτουργίες, **transformations** και **actions**. Οι λειτουργίες transformations είναι υπεύθυνες για την επεξεργασία των RDD, δημιουργώντας νέα RDD. Τα **transformations** δεν εκτελούνται άμεσα, αλλά δημιουργούν ένα lineage των γεγονότων το οποίο είναι ένα πλάνο εκτέλεσης που εκφράζεται ως ένα κατευθυνόμενο άκυκλο γράφημα (DAG) ανάμεσα στους προγόνους RDD που θα χρησιμοποιηθούν για την δημιουργία του νέου RDD. Τα transformations χαρακτηρίζονται από **lazy evaluation**, δηλαδή δεν εκτελούνται άμεσα, και για να εκτελεστούν πρέπει να κληθεί μία λειτουργία action. Τα transformations χωρίζονται σε δύο υποκατηγορίες, στα **narrow transformations** και τα **wide transformations**. Στα narrow transformations όλα τα αντικείμενα που χρειάζονται για να υπολογιστεί ένα RDD σε ένα partition, βρίσκονται σε ένα RDD σε κάποιο συγκεκριμένο partition. Ενώ τα wide transformations είναι αυτά που για τον υπολογισμό ενός RDD, θα χρειαστούν τα δεδομένα από τουλάχιστον δύο προηγούμενα RDD (Σχήμα 2.3.5). Στο Παράδειγμα 2.3.2 φορτώνουμε δεδομένα σε ένα DataSet, από ένα csv αρχείο, και εκτελούμε το filter transformation, το οποίο φιλτράρει τα δεδομένα.



Σχήμα 2.3.5: Narrow Transformation & Wide Transformation

Παράδειγμα 2.3.2: Παράδειγμα του filter transformation με τρεις διαφορετικούς τρόπους.

```
// Load Data
Dataset<Row> dataset = spark.read().option("header", true).csv("...");

// Filter using Expression
Dataset<Row> results = dataset.filter("product = 'Smartphone' AND year >= 2017");

// Filter using Lambda Function
Dataset<Row> results = dataset.filter( row ->
    row.getAs("product").equals("Smartphone") && row.getAs("year") >= 2017);

// Filter using Columns
Dataset<Row> results =
    dataset.filter(col("product").equalTo("Smartphone").and(col("year").geq(2017)));
```

Η δεύτερη κατηγορία λειτουργιών ενός RDD είναι τα **actions**, τα οποία μας επιτρέπουν να πάρουμε ένα τελικό αποτέλεσμα σε πραγματική μορφή DataSet από τα RDD. Είναι, όπως θα δούμε παρακάτω, η διαδικασία αποστολής δεδομένων από τους *executors* στον *driver* (Βλ. Ενότητα 2.3.3). Στο Παράδειγμα 2.3.3 αρχικά φιλτράρουμε τα δεδομένα όπως στο προηγούμενο παράδειγμα και στη συνέχεια εκτελούμε το count action, το οποίο επιστρέφει τον αριθμό των δεδομένων του DataSet.

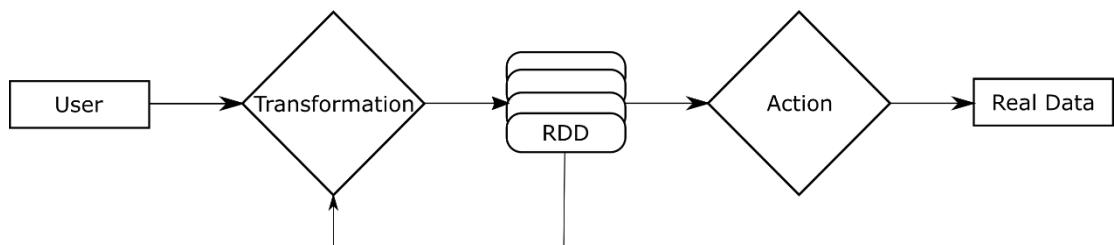
Στο Σχήμα 2.3.6 αναπαρίσταται ο κύκλος εκτέλεσης των transformations και actions. Για περισσότερες πληροφορίες σχετικά με τις λειτουργίες σχετικά με τα transformations και actions, όπως ποιες εντολές περιέχουν και πώς εκτελούνται μπορείτε να επισκεφτείτε το επίσημο documentation του Apache Spark [ASR19].

Παράδειγμα 2.3.3: Παράδειγμα του count action.

```
// Load Data
Dataset<Row> dataset = spark.read()
    .option("header", true).csv("...");

// Filter using Expression
Dataset<Row> results = dataset.filter("product = 'Smartphone' AND year >= 2017");

// Action count
int counter = results.count();
```



Σχήμα 2.3.6: Γραφική αναπαράσταση της διαδικασίας των Transformations και Action

Τέλος να αναφέρουμε ότι μια από τις προκαθορισμένες ρυθμίσεις του Spark είναι να αποθηκεύει τα RDD στη μνήμη cache, που είναι ένας από τους λόγους που η εκτέλεσή τους είναι πολύ γρηγορότερη σε σύγκριση με προηγούμενες δομές δεδομένων που χρησιμοποιούνταν π.χ. από το MapReduce. Επιπλέον αν ο χρήστης επιθυμεί να μην χάσει ένα RDD ακόμη και όταν η μνήμη cache είναι πλήρης, μπορεί να χρησιμοποιήσει τη μέθοδο **persist**.

2.3.3 Spark SQL & DataSets

Όπως φαίνεται στο Σχήμα 2.3.8 πάνω από το Spark Core βρίσκονται τέσσερις (4) βιβλιοθήκες που παρέχει το Spark, η Spark SQL, η Spark MLlib, η Spark GraphX και η Spark Streaming. Περισσότερες πληροφορίες για την κάθε βιβλιοθήκη μπορείτε να δείτε στο επίσημο Documentation του Apache Spark [ASR19]. Στη δική μας εργασία, η βιβλιοθήκη που θα μας απασχολήσει, είναι η Spark SQL και σε αυτήν την ενότητα θα την αναλύσουμε σε βάθος.

Η Spark SQL προσφέρει μια πιο λεπτομερή και δομημένη μορφή στα δεδομένα για τον τελικό χρήστη. Όπως αναφέραμε και παραπάνω, είναι η βιβλιοθήκη που υποστηρίζει και παρέχει τις βελτιστοποιήσεις, όπως τη χρήση του Catalyst Optimizer και την δυνατότητα ανάγνωσης δεδομένων από διάφορες πηγές δεδομένων, στα DataFrames και DataSets. Επιπλέον, η Spark SQL δίνει την δυνατότητα σε μία εφαρμογή Spark να εκτελέσει SQL queries, πάνω σε φορτωμένα RDD. Έχοντας, λοιπόν, ολοκληρώσει πλήρως την αναφορά μας στο Spark και τα DataSets, μπορούμε να δούμε μερικές εντολές σε κώδικα, για βασικές λειτουργίες των Datasets.

Παράδειγμα 2.3.4: Παράδειγμα του SparkSession.

Η Spark SQL κατά την εκκίνηση απαιτεί την δημιουργία ενός SparkSession, το οποίο είναι μια επέκταση του SparkContext και προτιμάτε από την έκδοση Spark 2.0 και μετά.

```
import org.apache.spark.sql.SparkSession;

SparkSession spark = SparkSession
    .builder()
    .appName("Spark Session")
    .master("local[*]")
    .config("spark.sql.warehouse.dir", "file:///c:/tmp/")
    .getOrCreate();
```

Έστω το αρχείο products.csv :

```
product_id,name,price,subcategory,category
0,product53,205.45,subcategory4,category1
1,product79,53.49,subcategory2,category3
2,product37,233.85,subcategory5,category3
3,product87,288.04,subcategory4,category8
4,product100,34.42,subcategory5,category7
5,product46,259.11,subcategory2,category2
6,product70,172.78,subcategory1,category2
7,product60,89.11,subcategory5,category4
8,product45,147.91,subcategory4,category3
9,product36,135.11,subcategory3,category1
10,product25,113.65,subcategory5,category3
```

Παράδειγμα 2.3.5: Παράδειγμα δημιουργίας ενός Data Frame.

Για να δημιουργήσουμε ένα DataFrame, στην ουσία στην Java ένα DataSet<Row>, από ένα csv αρχείο, εκτελούμε την συνάρτηση read():

```
Dataset<Row> df = spark.read()
    .format("csv")
    .option("sep", ",")
    .option("inferSchema", "true")
    .option("header", "true")
    .load("../products.csv");
```

Μετά την συνάρτηση read() καλούμε πάντα την συνάρτηση format(...), που περιγράφει την μορφή του αρχείου εισόδου.

Στην συνέχεια μπορούμε να έχουμε προαιρετικές ρυθμίσεις με την συνάρτηση option(...). Στο παράδειγμά μας έχουμε ορίσει τρεις προαιρετικές ρυθμίσεις

- option("sep", ",") : Ορίζει το διαχωριστικό σύμβολο των τιμών του csv, σε εμάς είναι με κόμμα ",".
- option("inferSchema", "true") : Δημιουργεί αυτόματα το σχήμα των δεδομένων. Χρειάζεται να διατρέξει δεύτερη φορά τα δεδομένα.
- option("header", "true") : Ενημερώνει ότι η πρώτη γραμμή είναι τα ονόματα των στηλών και αγνοείται στην φόρτωση των δεδομένων.

Τέλος η συνάρτηση load(...) παίρνει ως όρισμα το path του αρχείου που θέλουμε να διαβάσουμε.

Με την συνάρτηση show() μπορούμε να δούμε το περιεχόμενο ενός DataSet.

```
df.show();
```

product_id	name	price	subcategory	category
0	product53	205.45	subcategory4	category1
1	product79	53.49	subcategory2	category3
2	product37	233.85	subcategory5	category3
3	product87	288.04	subcategory4	category8
4	product100	34.42	subcategory5	category7
5	product46	259.11	subcategory2	category2
6	product70	172.78	subcategory1	category2
7	product60	89.11	subcategory5	category4
8	product45	147.91	subcategory4	category3
9	product36	135.11	subcategory3	category1
10	product25	113.65	subcategory5	category3

Και με την συνάρτηση printSchema() μπορούμε να δούμε το σχήμα που δημιουργησε το Spark

```

df.printSchema();

// root
// |-- product_id: integer (nullable = true)
// |-- name: string (nullable = true)
// |-- price: double (nullable = true)
// |-- subcategory: string (nullable = true)
// |-- category: string (nullable = true)

```

Παράδειγμα 2.3.6:

Παράδειγμα εκτέλεσης SQL query σε ένα Data Frame.

Έχοντας φορτώσει τα DataSets με τα οποία θέλουμε να δουλέψουμε, χάρη στο Spark SQL μπορούμε να εκτελέσουμε ένα SQL query :

```

df.createOrReplaceTempView("products");

Dataset<Row> sqlDF = spark.sql("SELECT * FROM products WHERE category =
'category1');

sqlDF.show();
// +-----+-----+-----+-----+
// |product_id|      name|  price| subcategory| category|
// +-----+-----+-----+-----+
// |          0|product53|205.45|subcategory4|category1|
// |          9|product36|135.11|subcategory3|category1|
// +-----+-----+-----+-----+

spark.close();

```

Η συνάρτηση `createOrReplaceTempView(...)` αποθηκεύει προσωρινά τον πίνακα στη μνήμη, δηλαδή μέχρι να τερματιστεί το `SparkSession` που την δημιούργησε, και μπορεί ο τελικός χρήστης να τρέξει μια ερώτηση SQL σε αυτό. Υπάρχει και η συνάρτηση `createGlobalTempView(...)`, που χρησιμοποιείται όταν τρέχουν πολλά `SparkSession` ταυτόχρονα, και αποθηκεύει τα δεδομένα μέχρι να τερματιστούν όλα τα `SparkSession`. Στη συνέχεια χρησιμοποιούμε το `SparkSession` με την συνάρτηση `sql("query_here")` για να εκτελέσουμε ένα κανονικό SQL query και βλέπουμε ότι μας επέστρεψε ένα `DataSet` με το αποτέλεσμα. Τέλος όταν θέλουμε να τερματίσουμε μία `Spark` εφαρμογή εκτελούμε την συνάρτηση `close()`.

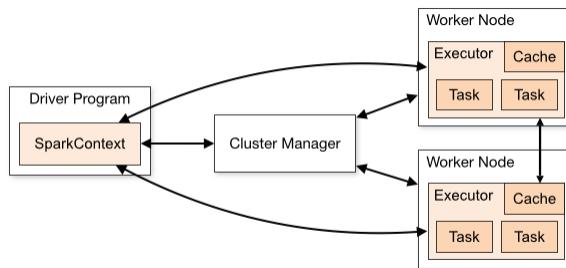
2.3.4 Spark Core & Cluster Manager

Όπως προαναφέραμε το Apache Spark είναι σχεδιασμένο για να εκτελεί παράλληλους υπολογισμούς σε συστάδες υπολογιστών. Βασικό θεμέλιο για να επιτευχθεί αυτό είναι το Spark Core. Το **Spark Core** είναι υπεύθυνο για την κατανομή των εργασιών σε ένα cluster, για τη δημιουργία του πλάνου εκτέλεσης και για τις βασικές I/O λειτουργίες. Η επικοινωνία του Spark Core με ένα Spark cluster γίνεται μέσω του *SparkContext*, το οποίο αντιπροσωπεύει τον κόμβο επικοινωνίας τους. Το Spark περιέχει τον δικό του cluster manager, τον Standalone, και υποστηρίζει και τρεις δημοφιλείς, τον Apache Mesos, τον Hadoop YARN και τον Kubernetes. Το Spark είναι σχεδιασμένο με την αρχιτεκτονική master/slave (αφέντη/σκλάβου). Περιέχει έναν κεντρικό κόμβο, τον driver, ο οποίος επικοινωνεί με τους executors (Σχήμα 2.3.7). Ο **driver** είναι η διεργασία όπου τρέχει η main συνάρτηση. Όταν ο χρήστης ζητήσει να εκτελεστούν κάποιες λειτουργίες σε RDD, ο driver θα μετατρέψει πρώτα το πρόγραμμα σε tasks και στη συνέχεια θα βγάλει το πλάνο εκτέλεσης των tasks στους executors. Οι **executors** είναι οι κόμβοι που εκτελούνται τα tasks και επιστρέφουν το αποτέλεσμά τους στον driver, μαζί με μία αναφορά επιτυχίας ή αποτυχίας. Επίσης στους executors γίνεται η αποθήκευση των RDD στη μνήμη. Στο Σχήμα 2.3.8 βλέπουμε την αρχιτεκτονική σχεδίαση ενός ολοκληρωμένου συστήματος Spark. Το Spark δεν έχει κάποιο συγκεκριμένο σύστημα αποθήκευσης δεδομένων, αλλά χρησιμοποιεί όλα τα δημοφιλή συστήματα αποθήκευσης, όπως το HDFS, Cloud και RDBMS/NOSQL. Η πιο σύνηθες και βέλτιστη επιλογή είναι το HDFS. Το HDFS λειτουργεί, έχοντας έναν κεντρικό κόμβο και τρεις τουλάχιστον κόμβους με τριπλά αντίγραφα των δεδομένων. Με αυτόν τον τρόπο επιτυγχάνεται η ανοχή σε σφάλματα που έχει το Spark, καθώς δεν χάνονται τα δεδομένα λόγω της αρχιτεκτονικής του HDFS και το Spark μπορεί να επανυπολογίσει ένα RDD σε περίπτωση που ένας κόμβος καταρρεύσει. Τέλος θα δώσουμε ένα παράδειγμα μιας ροής εκτέλεσης :

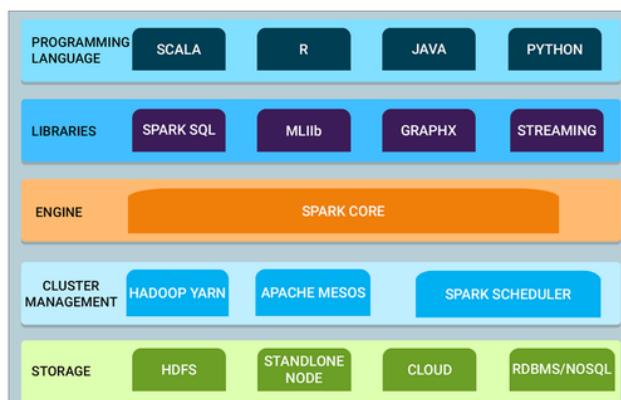
1. Μια εφαρμογή δημιουργεί ένα *SparkContext* και αρχίζει η επικοινωνία της εφαρμογής Spark με το cluster. Αυτός ο κόμβος θεωρείται ο driver.
2. Ο driver ζητάει από τον cluster manager να δημιουργήσει τους executors.
3. Ο cluster manager δημιουργεί τους executors.
4. Ο χρήστης ζητάει από την εφαρμογή να εκτελέσει actions και transformations.
5. Ο driver μετατρέπει τις λειτουργίες σε tasks και βγάζει ένα πλάνο εκτέλεσης των tasks για τους executors.
6. Οι executors εκτελούν τις λειτουργίες και επιστρέφουν τα αποτελέσματα. Να σημειωθεί εδώ ότι σε περίπτωση αποτυχίας ενός executor (crash) το Spark, όπως

αναφέραμε, είναι ανθεκτικό σε σφάλματα και θα διαμοιράσει το task στους ενεργούς executors.

7. Ο driver συλλέγει τα αποτελέσματα και επιστρέφει το τελικό αποτέλεσμα στον χρήστη.



Σχήμα 2.3.7: Αρχιτεκτονική σχεδίαση των Driver, Executors & Cluster Manager (πηγή: <https://spark.apache.org/docs/latest/cluster-overview.html>)



Σχήμα 2.3.8: Η αρχιτεκτονική δομή ενός ολοκληρωμένου συστήματος Apache Spark (πηγή: <https://towardsdatascience.com/a-beginners-guide-to-apache-spark-ff301cb4cd92>)

Κεφάλαιο 3. Σχεδίαση & Υλοποίηση

3.1 Σχεδίαση και αρχιτεκτονική λογισμικού

Στην ενότητα αυτή θα περιγράψουμε αναλυτικά την αρχιτεκτονική δομή και σχεδίαση του λογισμικού. Αρχικά θα μιλήσουμε για την μετατροπή του Java Project σε ένα Maven Java Project, πως γίνεται, τι όφελος έχει η χρήση του Maven και ποιες σχετικές ρυθμίσεις έγιναν στο δικό μας λογισμικό. Στη συνέχεια θα αναλύσουμε όλες τις αλλαγές που έγιναν σε επίπεδο κώδικα στις ήδη υπάρχουσες κλάσεις του λογισμικού καθώς και τις νέες προσθήκες που έγιναν. Έπειτα θα περιγράψουμε το μοντέλο του λογισμικού παρουσιάζοντας UML Class Diagrams και τέλος θα γίνει μια περιγραφή σημαντικών Sequence Diagrams.

3.1.1 Μετατροπή σε Maven Java Project

Το Maven [MVN20] είναι ένα build automation εργαλείο, που χρησιμοποιείται κυρίως σε Java projects. Σκοπός ενός build automation εργαλείου είναι η αυτοματοποίηση βασικών διεργασιών σε ένα project όπως το compile, το packaging και το testing. Επιπλέον ένας σημαντικός σκοπός του Maven είναι να γνωρίζει και να κατεβάζει αυτόματα όλα τα dependencies που χρησιμοποιεί ένα project. Αυτό έχει ως στόχο να διευκολύνει την ομαλή λειτουργία ενός project κατά την διάρκεια της μεταφοράς σε άλλον υπολογιστή ή στον συντονισμό μιας ομάδας να χρησιμοποιεί όμοιες εκδόσεις σε ένα ομαδικό project. Παρακάτω θα αναλύσουμε πως μετατρέψαμε το λογισμικό του DelianCube Engine σε Maven Project και θα μιλήσουμε για τις λειτουργίες που εμείς το χρησιμοποιήσαμε.

Λόγω της δημοφιλίας του Maven, τα γνωστά IDEs, όπως είναι το Eclipse που χρησιμοποιούμε, παρέχει μια εύκολη μετατροπή ενός Java Project σε Maven Java Project. Στην περίπτωσή μας αυτό έγινε ανοίγοντας το project στο Eclipse, πατώντας δεξί κλικ πάνω στο project και επιλέγοντας “Configure -> Convert to Maven Project”. Στο παράθυρο που ανοίγει μας ζητούνται τρία ορίσματα που απαιτεί ένα Maven Project. Τα ορίσματα αυτά είναι :

- **groupid** : Η μεταβλητή αυτή αναγνωρίζει μοναδικά το κάθε project από τα υπόλοιπα και θα πρέπει να είναι της μορφής “com.example.project”.

- **artifactid** : Η μεταβλητή αυτή ορίζει το όνομα του εξαγόμενου αρχείου jar που θα δημιουργηθεί αργότερα.
- **version** : Η μεταβλητή αυτή εκφράζει την έκδοση που θα αναγράφεται στο εξαγόμενο αρχείο jar.

Στην δική μας περίπτωση τα ορίσματα αυτά έχουν ως εξής :

```
groupid = com.DelianCubeEngine
artifactid = DelianCubeEngine
version = 0.0.1-SNAPSHOT
```

Μόλις το project μετατραπεί σε Maven Project θα γίνει μια αναδόμηση των φακέλων του project, θα εμφανιστεί δύο νέοι φάκελοι bin και target και ένα νέο αρχείο “pom.xml”. Ο φάκελος bin είναι ο φάκελος εξαγωγής των αρχείων ενός Java Project, ο οποίος υπήρχε και πριν την μετατροπή σε Maven Project, ως κρυμμένος φάκελος. Ο φάκελος αυτός αποτελούσε έναν κρυμμένο φάκελο του Java Project για την αποθήκευση των εξαγόμενων αρχείων. Σε ένα Maven Project ο ρόλος του φακέλου bin αντικαθίσταται από τον φάκελο target. Επομένως ο φάκελος bin μπορεί να διαγραφεί. Ο φάκελος target είναι ο φάκελος που εξάγονται όλα τα αρχεία όταν θα ζητηθεί η διαδικασία build του project. Τέλος το “pom.xml” που δημιουργήθηκε είναι πολύ σημαντικό αρχείο και αποτελεί την βάση για την όλη λειτουργία του Maven. Η ονομασία του αποτελεί το ακρώνυμο του όρου “Project Object Model”. Στο αρχείο pom περιλαμβάνονται οι ρυθμίσεις του project, οι ρυθμίσεις για την διαδικασία του build καθώς και όλα τα dependencies που χρησιμοποιεί το project.

Στο δικό μας project, μία από τις λειτουργικότητες του pom αρχείου που θα χρησιμοποιήσουμε είναι να αυτοματοποιήσουμε την διαδικασία εγκατάστασης των κατάλληλων dependencies. Η δομή ενός dependency, για παράδειγμα για την βιβλιοθήκη junit, είναι ως εξής :

```
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.12</version>
<scope>test</scope>
</dependency>
...
```

```
</dependencies>
```

Παρατηρούμε ότι για κάθε dependency αρκεί να ορίσουμε τις μεταβλητές που αναφέραμε και παραπάνω, δηλαδή τις μεταβλητές που ο συνδυασμός τους εκφράζει μοναδικά ένα project. Ένα dependency μπορεί να έχει και επιπλέον ορίσματα, τα οποία μπορείτε να δείτε αναλυτικότερα στο επίσημο documentation της Maven. Αυτό που είναι σημαντικό για εμάς και θα αναφέρουμε είναι το πεδίο scope. Το scope θέτει κάποια όρια στην χρήση του κάθε dependency. Στο Σχήμα 3.1.1 βλέπουμε τις πέντε παραμέτρους που μπορεί να πάρει το πεδίο scope.

Πεδίο	Περιγραφή
compile	Είναι η default τιμή για κάθε dependency. Τα compile dependencies είναι διαθέσιμα σε όλα τα classpaths.
provided	Είναι παρόμοιο με το πεδίο compile, αλλά υποδεικνύει ότι θα τα παραδώσει το JDK κατά την διάρκεια εκτέλεσης.
runtime	Το πεδίο αυτό υποδηλώνει ότι το dependency δεν απαιτείται κατά την διάρκεια του compilation, αλλά μόνο κατά την διάρκεια της εκτέλεσης.
test	Το πεδίο αυτό υποδηλώνει ότι το dependency δεν απαιτείται για την ομαλή λειτουργία ενός project, αλλά μόνο κατά την διάρκεια του compile και του runtime.
system	Είναι παρόμοιο με το πεδίο provided, με την διαφορά ότι πρέπει να δώσει ο χρήστης το jar του dependency.

Σχήμα 3.1.1 : Τα πέντε πεδία του πεδίου scope σε ένα dependency.

Στην παρακάτω λίστα αναφέρονται τα dependencies που χρησιμοποιήθηκαν στο δικό μας project :

1. Junit - version : 4.12
2. Spark Core - version : 2.4.3
3. Spark SQL - version : 2.4.3
4. Hadoop HDFS - version : 3.2.0
5. Antlr - version : 3.5.2
6. Commons-io - version : 2.6
7. Commons-math3 - version : 3.6.1
8. MySQL-Connector-Java - version : 5.1.48
9. JFoenix - version : 8.0.8
10. Commons-lang - version : 2.6

Τέλος αναφέραμε παραπάνω ότι το Maven Project αναδομεί τους φακέλους ενός project. Το Maven έχει μια κοινή δομή σε όλα τα Maven Project με σκοπό την ευκολότερη κατανόηση ενός project από όλους τους χρήστες της. Μετατρέποντας το DelianCube Engine σε Maven Project υιοθετήσαμε και την δομή του. Στο Σχήμα 3.1.2 μπορεί κανείς να δει τους βασικούς φακέλους που θα μας χρησιμεύσουν στη δική μας υλοποίηση.

Folder	Description
src/main/java	Application/Library sources
src/main/resources	Application/Library resources
src/test/java	Test sources
src/test/resources	Test resources

Σχήμα 3.1.2: Η δομή των φακέλων σε ένα project Maven.

3.1.2 Αλλαγές κώδικα

Στη συνέχεια θα αναφερθούμε στις αλλαγές και προσθήκες που κάναμε στον κώδικα για την βελτίωση της υπάρχοντος δομής του project καθώς και την επέκταση για την ενσωμάτωση του Spark. Μπορείτε να δείτε συνοπτικά τις αλλαγές στον Πίνακα 3.2.1.

src/main/java -> client -> gui -> application

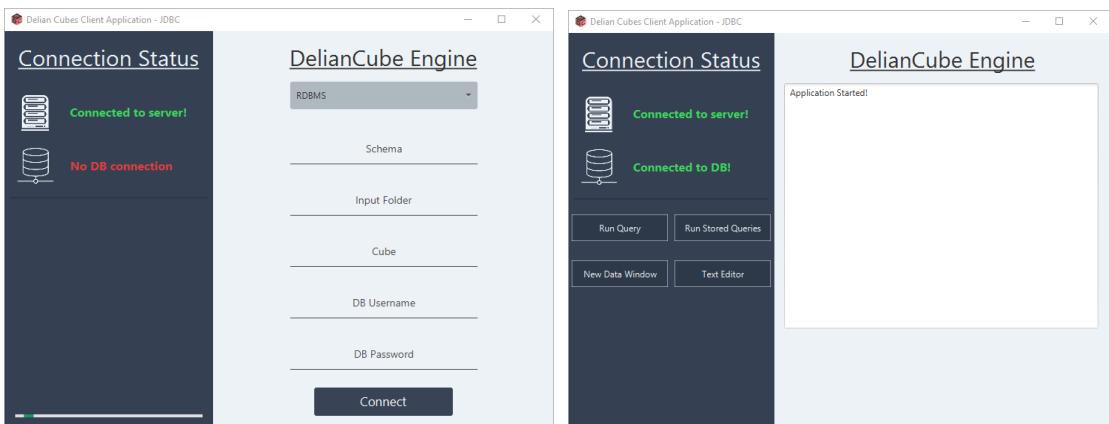
- **MainApp** : Η κλάση MainApp εκτελούσε την αρχικοποίηση του IMainEngine. Πλέον μεταφέρθηκε η αρμοδιότητα στον controller, οπότε στέλνει κατά την εκτέλεση του constructor της MainAppController, ένα αντικείμενο “IMainEngine service” και ένα “String serverStatus” που μας επιτρέπει να εμφανίζουμε στο gui εάν έχει βρεθεί η σύνδεση με τον Server.

src/main/java -> client -> gui -> controllers

- **MainAppController** : Η κλάση MainAppController όπως αναφέραμε πλέον δημιουργεί αυτή τη σύνδεση με τον Server. Αυτή η μεταφορά της αρμοδιότητας έγινε καθώς ο χρήστης πλέον μπορεί να επιλέξει τον τύπο σύνδεσης (RDBMS ή Spark) και η δήλωση των μεταβλητών (username, password, schemaName, cubeName, inputFolder) γίνεται δια-δραστικά από το γραφικό περιβάλλον. Η υλοποίηση των παραπάνω βρίσκεται στην συνάρτηση “connectDB()”. Επίσης έχει προστεθεί η συνάρτηση initialize η οποία ελέγχει την σύνδεση με τον Server κατά την εκκίνηση του Client, ώστε να βγάλει το αντίστοιχο μήνυμα στο GUI, και σε περίπτωση που έχει υπάρξει προηγούμενη επιτυχής σύνδεση με τα στοιχεία που δόθηκαν από τον χρήστη να συμπληρώσει αυτόματα τα πεδία στην επόμενη συνεδρία. Τα δεδομένα αυτά αποθηκεύονται στο αρχείο “latestDB.txt”.

src/main/java -> client -> gui -> views

- **MainApp.fxml** : Ως προς το γραφικό περιβάλλον έγινε ανασχεδιασμός του κεντρικού παραθύρου. Όπως αναφέραμε, ο χρήστης πλέον μπορεί να δώσει τα στοιχεία της σύνδεσης μέσω του παραθύρου, όπου προηγουμένως γινόταν hard-coded. Μπορούμε να δούμε το νέο γραφικό περιβάλλον στο Σχήμα 3.2.2. Για τον σχεδιασμό χρησιμοποιήθηκε το Scene Builder και η βιβλιοθήκη γραφικών JPhoenix.



Σχήμα 3.2.2: Κεντρικό παράθυρο εφαρμογής DelianCube Engine. Αριστερά είναι το αρχικό menu, ενώ δεξιά είναι το κεντρικό παράθυρο αφού γίνει σύνδεση.

Σε αυτό το σημείο θα ήταν καλό να αναλύσουμε και την δομή των στοιχείων που δίνονται και πως εκφράζονται στο κώδικα μας, καθώς τα χρησιμοποιούμε στην αρχικοποίηση των κύβων παρακάτω στον κώδικα.

Τα δεδομένα που παίρνουμε από τον χρήστη τα αποθηκεύουμε σε ένα `HashMap<String, String>` με όνομα “`userInputList`”. Η δομή του `HashMap` είναι ως εξής :

- **username** : Περιέχει το `username` για την ΒΔ.
- **password** : Περιέχει το `password` για την ΒΔ.
- **schemaName** : Περιέχει το όνομα του σχήματος που θα χρησιμοποιήσουμε από την ΒΔ.
- **cubeName** : Περιέχει το όνομα του κύβου που θέλουμε να χρησιμοποιήσουμε
- **inputFolder** : Περιέχει το του φακέλου που θέλουμε να χρησιμοποιήσουμε μέσα από τον φάκελο `InputFiles`.

Όπως βλέπουμε παραπάνω τα πεδία `username` και `password` χρησιμοποιούνται για τη σύνδεση με την Βάση Δεδομένων, οπότε στην περίπτωση που ο χρήστης επιλέξει σύνδεση μέσω του Spark τα δύο αυτά πεδία δεν χρειάζονται και αφαιρούνται και από το γραφικό περιβάλλον.

[src/main/java -> cubemanager -> connection](#)

Για την επιλογή σύνδεσης (RDBMS ή Spark) δημιουργήσαμε μία Factory κλάση που δημιουργεί ένα αντικείμενο τύπου `Connection`. Σε αυτό το πακέτο, λοιπόν, είναι όλες οι κώδικας που είναι υπεύθυνος για τις διαφορετικές συνδέσεις. Ο παρακάτω κώδικας,

συγκεκριμένα η κλάση RDBMSConnection, αποτελούν και refactoring στον προηγούμενο κώδικα.

- **Connection** : Η κλάση Connection είναι μια abstract κλάση, η οποία περιέχει τις όμοιες συναρτήσεις που χρειάζονται για όλες τις συνδέσεις. Είναι το γενικό αντικείμενο που χρησιμοποιείται από το υπόλοιπο πρόγραμμα και μπορεί να αποτελέσει αντικείμενο είτε RDBMSConnection είτε SparkConnection.
- **ConnectionFactory** : Η κλάση αυτή αποτελεί το Factory της σύνδεσης, η οποία διαβάζει τον τύπο σύνδεσης από την μεταβλητή “typeOfConnection”, δημιουργεί την αντίστοιχη σύνδεση και την επιστρέφει στο σημείο που κλήθηκε.
- **RDBMSConnection** : Ήταν η κλάση Database η οποία έγινε refactoring εξολοκλήρου και υλοποιεί την σύνδεση με τη Βάση Δεδομένων (MySQL). Επιπλέον, στην κλάση RDBMSConnection γίνεται η δημιουργία του κύβου, μαζί με την παραγωγή των πινάκων στον κύβο, όπως αυτοί υπάρχουν στη Βάση Δεδομένων. Τέλος η κλάση είναι επίσης υπεύθυνη για την εκτέλεση των queries μέσω της συνάρτησης “executeSql” και επιστρέφει ένα Result.
- **SparkConnection** : Είναι μία αντίστοιχη υλοποίηση της RDBMSConnection, για τη σύνδεση με Spark, αντί για βάση δεδομένων. Δημιουργεί επίσης τους πίνακες τον κύβο και τους πίνακές του, αυτή τη φορά ωστόσο διαβάζοντας αρχεία csv, θα αναλύσουμε παρακάτω τα συγκεκριμένα αρχεία. Τέλος δημιουργεί έναν SparkManager ο οποίος τρέχει το κυρίως πρόγραμμα του Spark.
- **SparkManager** : Αποτελεί το κυρίως μέρος του Spark για την εκτέλεσή του. Γίνεται η αρχικοποίηση του Spark, η ανάγνωση των αρχείων και η δημιουργία των DataSets. Επίσης περιέχει την συνάρτηση “executeQueryToSpark” η οποία τρέχει τα queries και επιστρέφει τα αποτελέσματα μέσω της “generateResult”.

src/main/java -> cubemanager -> cubebase

- **CubeBase** : Η κλάση CubeBase είναι αυτή που δημιουργεί το αντικείμενο Connection για την κεντρική κλάση “SimpleQueryProcessorEngine”. Παλαιότερα η διαδικασία αυτή γινόταν στην κλάση “Database”, το οποίο έπρεπε να τροποποιηθεί καθώς πλέον μπορούμε να έχουμε δύο (και στη συνέχεια ίσως και περισσότερους) τύπους σύνδεσης.

src/main/java -> cubemanager -> starschema

- **Database** : Έχει διαγραφεί και οι αρμοδιότητές της έχουν μεταφερθεί στην RDBMSConnection.
- **Table** : Προστέθηκε μία επιπλέον συνάρτηση “setAttribute”. Υπήρχε ήδη για το κομμάτι του RDBMS και υλοποιήθηκε για την ανάγνωση των attributes των

δεδομένων του Spark. Στην περίπτωση του RDBMS διαβάζουμε τα attributes του κάθε πίνακα από τη Βάση Δεδομένων. Στο Spark χρειάστηκε να σημειώσουμε τα attributes σε αρχεία csv για τον κάθε “πίνακα”. Αναλυτική αναφορά για την δομή των αρχείων αυτών γίνεται στην ενότητα «Αρχικοποίηση του Spark», και σχηματικά φαίνεται στο Σχήμα 3.4.1.

src/main/java -> cubemanager

- **CubeManager** : Μεταφέρθηκαν δύο συναρτήσεις από την SimpleQueryProcessorEngine, η “parseFile”, διατηρώντας το όνομα, και η “constructDimension”, η οποία έχει μετονομαστεί σε “constructCube”.

src/main/java -> client -> naiveJavaClient

- **NaiveJavaClient** : Η κλάση NaiveJavaClient είναι υπεύθυνη για την εκτέλεση του συστήματος σε τερματικό περιβάλλον. Προηγουμένως η είσοδος των δεδομένων ήταν hard-coded στο αρχείο. Πλέον η αλληλοεπίδραση γίνεται μέσω του αρχείου naive.ini (Περισσότερες πληροφορίες στην Ενότητα 3.3).

Σύνοψη αλλαγών κώδικα

Connection [Νέο αρχείο]	Abstract κλάση για τη σύνδεση
ConnectionFactory [Νέο αρχείο]	Factory κλάση για τη σύνδεση
CubeBase [Τροποποιήθηκε]	Αρχικοποίηση σύνδεσης
CubeManager [Τροποποιήθηκε]	Υλοποίηση refactoring
Database [Αφαιρέθηκε]	-
MainApp [Τροποποιήθηκε]	Μεταφορά κώδικα σύνδεσης με το IMainEngine
MainAppController [Τροποποιήθηκε]	Υλοποίηση σύνδεσης του χρήστη από το γραφικό περιβάλλον (προηγουμένως ήταν hard-coded) και προσθήκη επιλογής RDBMS ή Spark
MainApp.fxml [Τροποποιήθηκε]	Ανασχεδιασμός γραφικού περιβάλλοντος
NaiveJavaClient [Τροποποιήθηκε]	Τροποποίηση ώστε να γίνεται η είσοδος των δεδομένων από αρχείο.
RDBMSConnection [Νέο αρχείο]	Σύνδεση και παροχή λειτουργικότητας σε σύστημα RDBMS
SimpleQueryProcessorEngine [Τροποποιήθηκε]	Τροποποιήσεις ώστε να γίνεται δυνατή η λειτουργία του συστήματος με Spark.
SparkConnection [Νέο αρχείο]	Σύνδεση και παροχή λειτουργικότητας σε σύστημα Spark
SparkManager [Νέο αρχείο]	Κώδικας εκτέλεσης του Spark.
Table [Τροποποιήθηκε]	Υλοποίηση κύβων για τα συστήματα RDBMS και Spark

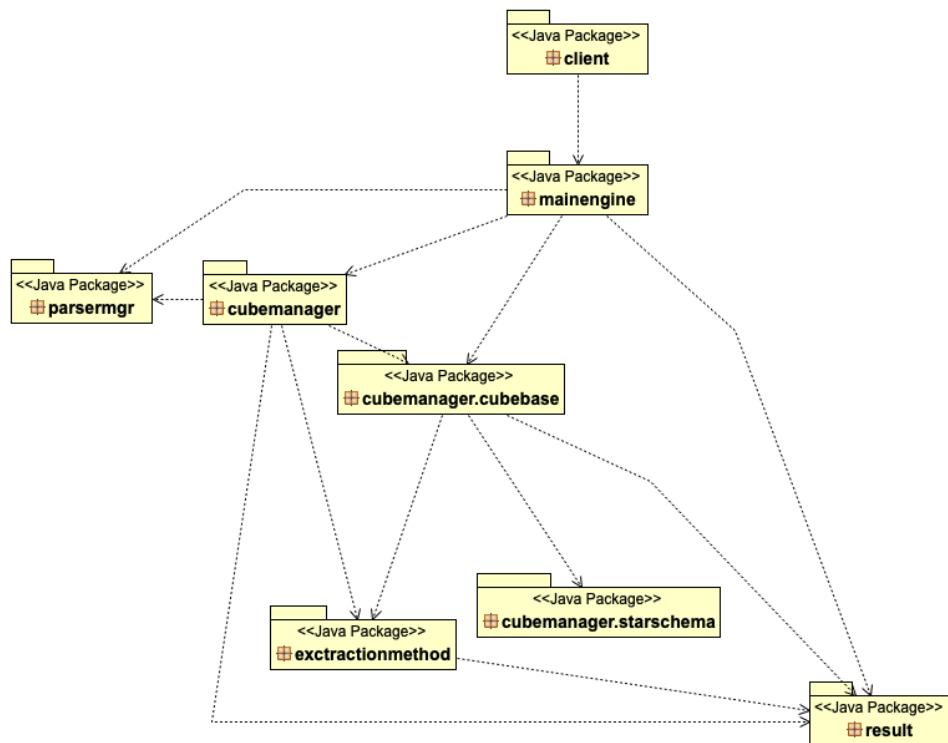
Πίνακας 3.2.1: Συνοπτικά οι αλλαγές που έγιναν.

3.1.3 Περιγραφή του μοντέλου του λογισμικού

Έχοντας ολοκληρώσει όλες τις αλλαγές στον κώδικα μας, θα παρουσιάσουμε τα UML Class και Package Diagrams και θα αναλύσουμε τις σχέσεις των αντίστοιχων κλάσεων και πακέτων. Όλα τα σχέδια που παρουσιάζονται αποτελούν τα τελικά (νέα) σχέδια ύστερα από την υλοποίηση της εργασίας.

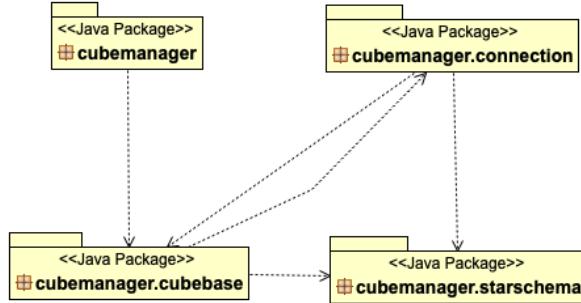
3.1.3.1 UML Package Diagram

To Package Diagram αναπαριστά την αλληλεπίδραση κατά την διάρκεια της λειτουργίας μεταξύ των πακέτων στο λογισμικό μας. Όπως φαίνεται στο Σχήμα 3.2.3.1, η διαδικασία εκκίνησης του προγράμματος θα είναι από τον client. Το πακέτο client αλληλεπιδρά με το πακέτο mainengine για την εκκίνηση του Server-Client συστήματος. Η mainengine φαίνεται πως είναι το πακέτο το οποίο συντονίζει όλους τους επιμέρους μηχανισμούς του DelianCube Engine. Ο συντονισμός αυτός αφορά τα τρία κύρια μέρη-πακέτα, του parsemgr, του result και του cubemanager.



Σχήμα 3.2.3.1 : UML Package Diagram του project

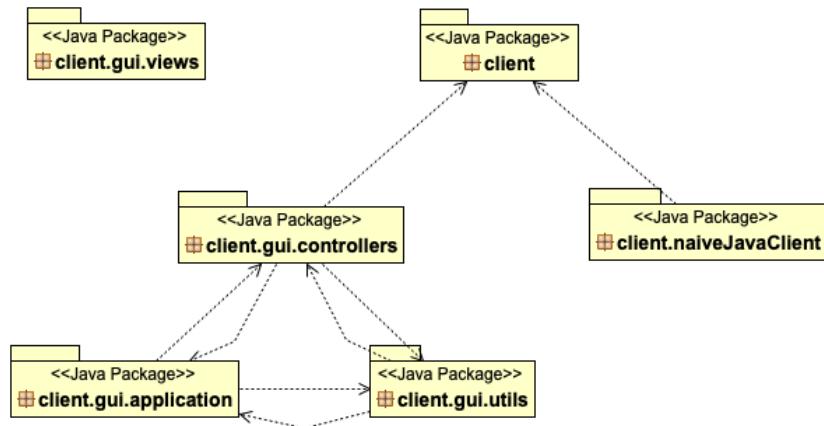
Με τη σειρά του το cubemanager είναι το κεντρικό πακέτο σε ότι αφορά τον συντονισμού του συστήματος OLAP. Στο Σχήμα 3.2.3.2 μπορούμε να δούμε απομονωμένα το πακέτο cubemanager με τα τρία υπό-πακέτα που περιέχει.



Σχήμα 3.2.3.2 : UML Package Diagram του πακέτου cubemanager.

3.1.3.2 UML Class Diagram του πακέτου client

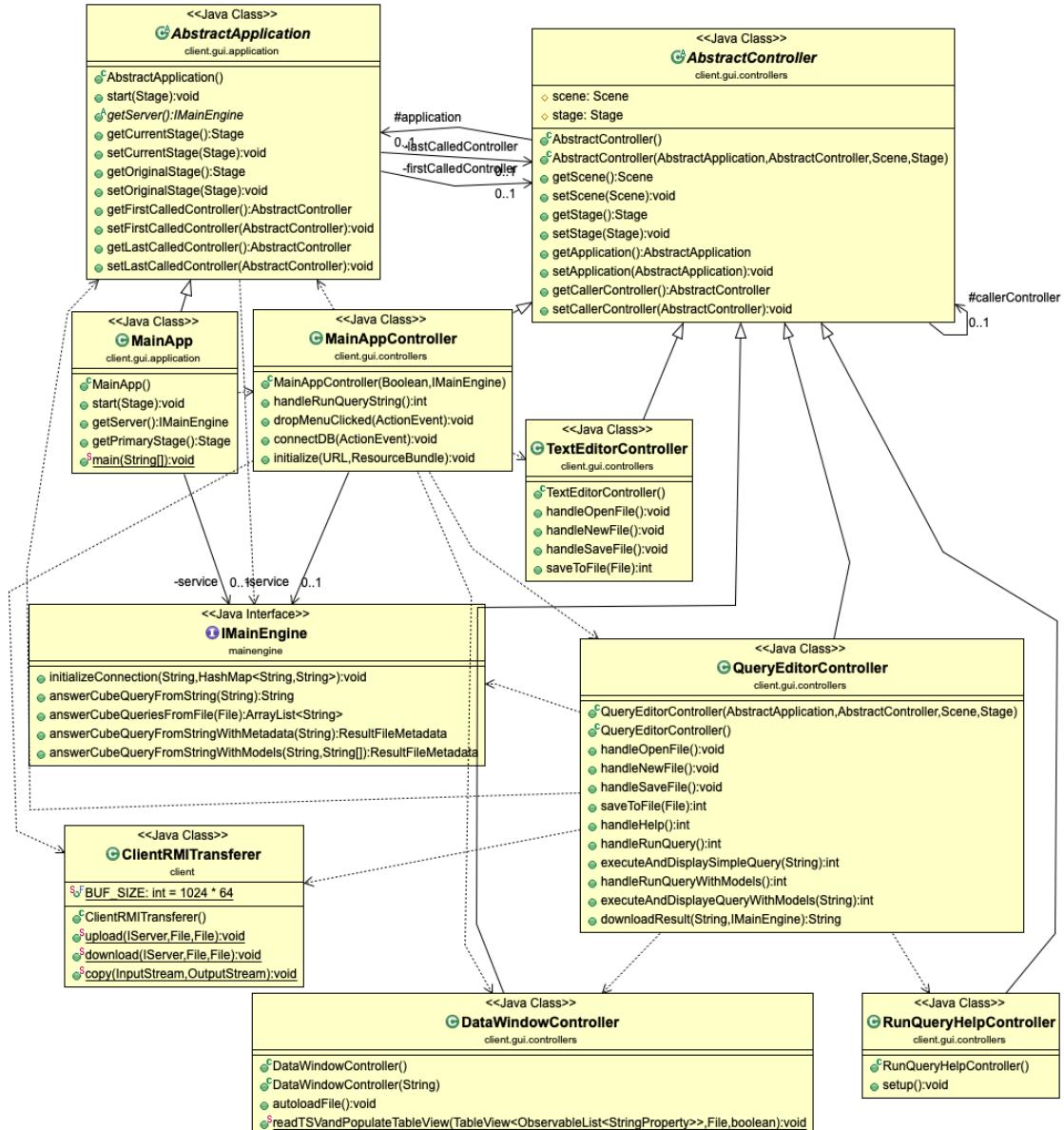
Το πακέτο client, όπως υποδεικνύει το όνομά του, αφορά την λειτουργία του Client. Περιλαμβάνει δύο πακέτα, το gui το οποίο περιέχει τον κώδικα για την εκτέλεση του Client σε γραφικό περιβάλλον, και του naiveJavaClient το οποίο περιέχει τον κώδικα για την εκτέλεση του Client μέσω του τερματικού. Το πακέτο views δεν αποτελεί κώδικα java αλλά fxml αρχεία που υλοποιούν το γραφικό περιβάλλον, οπότε δεν έχει κάποια σύνδεση με τα υπόλοιπα πακέτα. Στο Σχήμα 3.2.3.3 βλέπουμε το UML Package Diagram για το πακέτο client.



Σχήμα 3.2.3.3 : UML Package Diagram του πακέτου client.

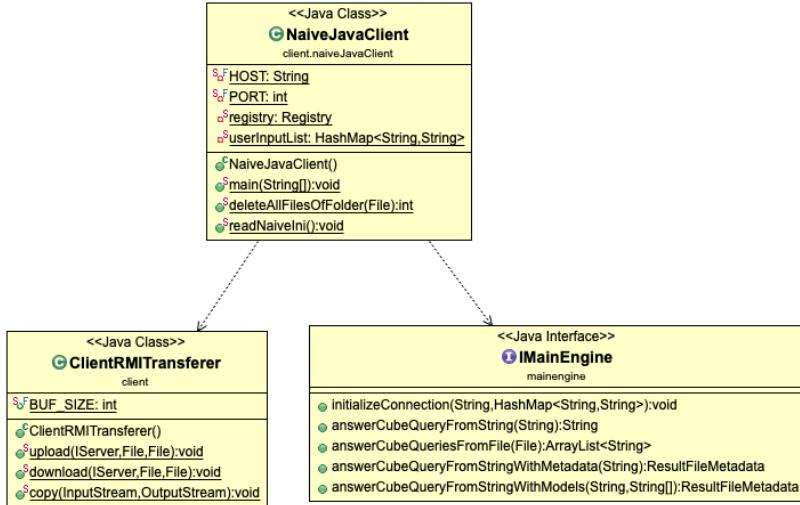
Για τις συναρτήσεις τις οποίες περιέχει το πακέτο gui μπορούμε να δούμε το Σχήμα 3.2.3.4. Στο σχήμα αυτό βλέπουμε πως η MainApp κλάση αποτελεί επέκταση της AbstractApplication κλάσης, η οποία με τη σειρά της αποτελεί επέκταση της κλάσης

Application της βιβλιοθήκης JavaFX. Η MainApp περιέχει την main κλάση και είναι το αρχείο που απαιτείται για την εκτέλεση του Client με γραφικό περιβάλλον. Αντίστοιχη συνάρτηση της AbstractApplication, αποτελεί η AbstractController για τις κλάσεις που εμφανίζουν ένα νέο παράθυρο στον χρήστη. Τέλος παρατηρούμε ότι η MainApp περιέχει ένα αντικείμενο IMainEngine, το οποίο αποστέλλει στον controller της, MainAppController, και απαιτείται για την σύνδεση επικοινωνίας του Client με τον Server.



Σχήμα 3.2.3.4 : UML Class Diagram του πακέτου client.gui.

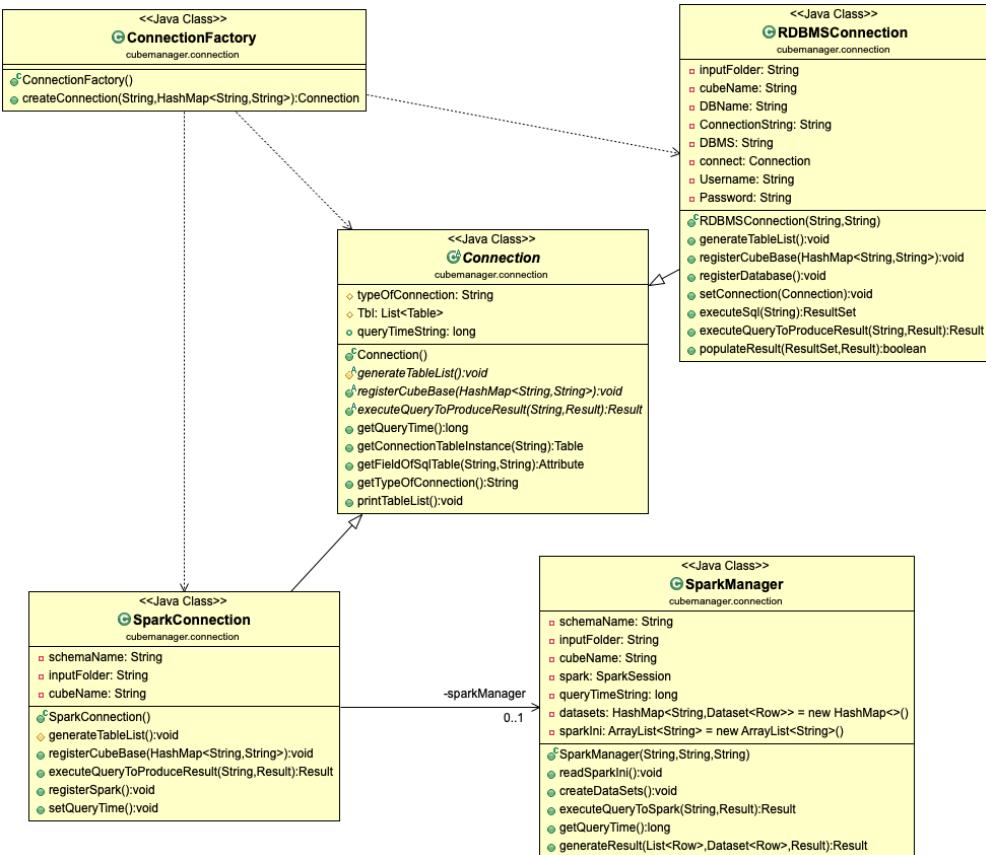
Τέλος, παρέχεται το αντίστοιχο Σχήμα 3.2.3.5 για το υπό-πακέτο naiveJavaClient.



Σχήμα 3.2.3.5 : UML Class Diagram του πακέτου client.naiveJavaClient.

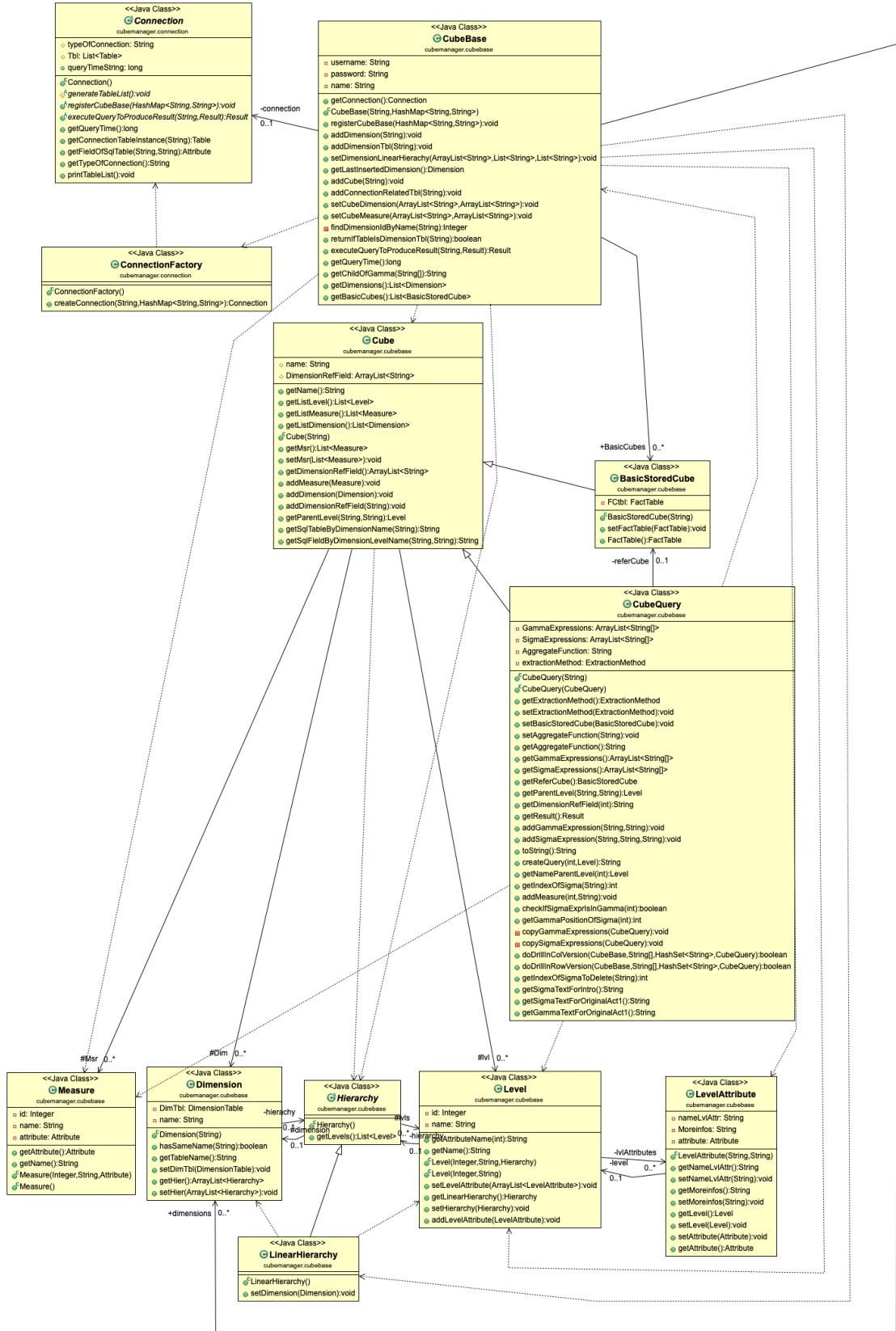
3.1.3.3 UML Class Diagram του πακέτου cubemanager

Όπως αναφέραμε στο Σχήμα 3.2.3.2, το πακέτο cubemanager είναι υπεύθυνο για την υλοποίηση του συστήματος OLAP. Το πακέτο cubemanager περιέχει την κλάση `CubeManager`, η οποία είναι υπεύθυνη για την αρχικοποίηση της σύνδεσης με τον τύπο του συστήματος που επιλέγει ο χρήστης, RDBMS ή Spark. Επιπλέον, στο πακέτο cubemanager, είναι υπεύθυνο για την φόρτωση του κύβου στο σύστημα DelianCube Engine. Για την υλοποίηση των παραπάνω, το πακέτο cubemanager περιέχει τρία υπό-πακέτα, το `connection`, το `cubebase` και το `starschema`. Το πακέτο `connection` περιέχει την υλοποίηση της σύνδεσης με τον αντίστοιχο τύπο. Η υλοποίηση έγινε εφαρμόζοντας την αρχιτεκτονική δομή Factory της Java. Τον ρόλο του Factory αναλαμβάνει η κλάση `ConnectionFactory`. Κατά την εκκίνηση του Client, δημιουργείται ένα αφηρημένο αντικείμενο τύπου `Connection`, το οποίο η κλάση `ConnectionFactory` μετατρέπει στο κατάλληλο αντικείμενο, `RDBMSConnection` για σύνδεση με σχεσιακή βάση δεδομένων ή `SparkConnection` για σύνδεση με το σύστημα Apache Spark. Στο Σχήμα 3.2.3.6 μπορείτε να δείτε το UML Class Diagram του πακέτου `connection`.

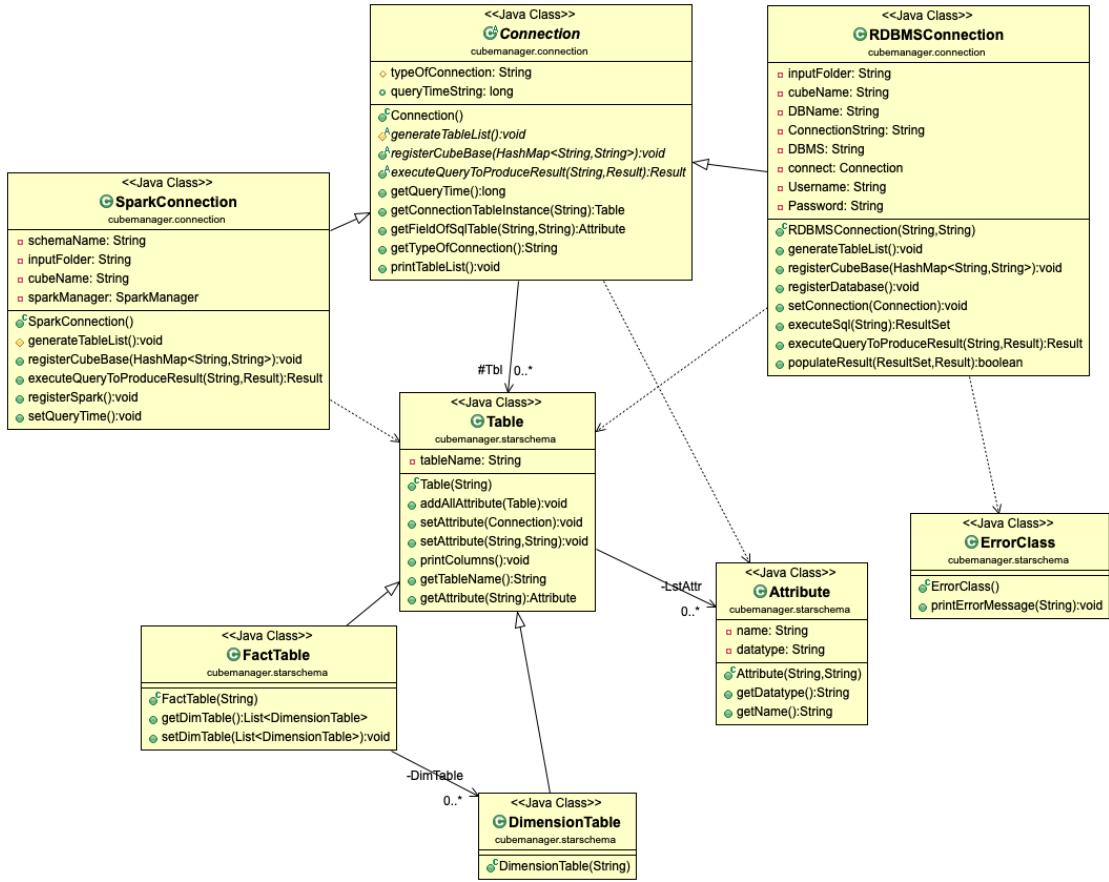


Σχήμα 3.2.3.6 : UML Class Diagram του πακέτου cubemanager.connection.

Τα άλλα δύο πακέτα, cubebase και starschema, υλοποιούν την δημιουργία του κύβου στο σύστημα. Αποτελούν την σχεδίαση μιας εικονικής μορφής των δεδομένων σε κύβο, ο οποίος κύβος χρησιμοποιείται για την εξαγωγή του κατάλληλου ερωτήματος SQL αργότερα. Στο Σχήμα 3.2.3.7 και Σχήμα 3.2.3.8 βλέπουμε τα UML Class Diagrams του πακέτου cubebase και starschema αντίστοιχα.

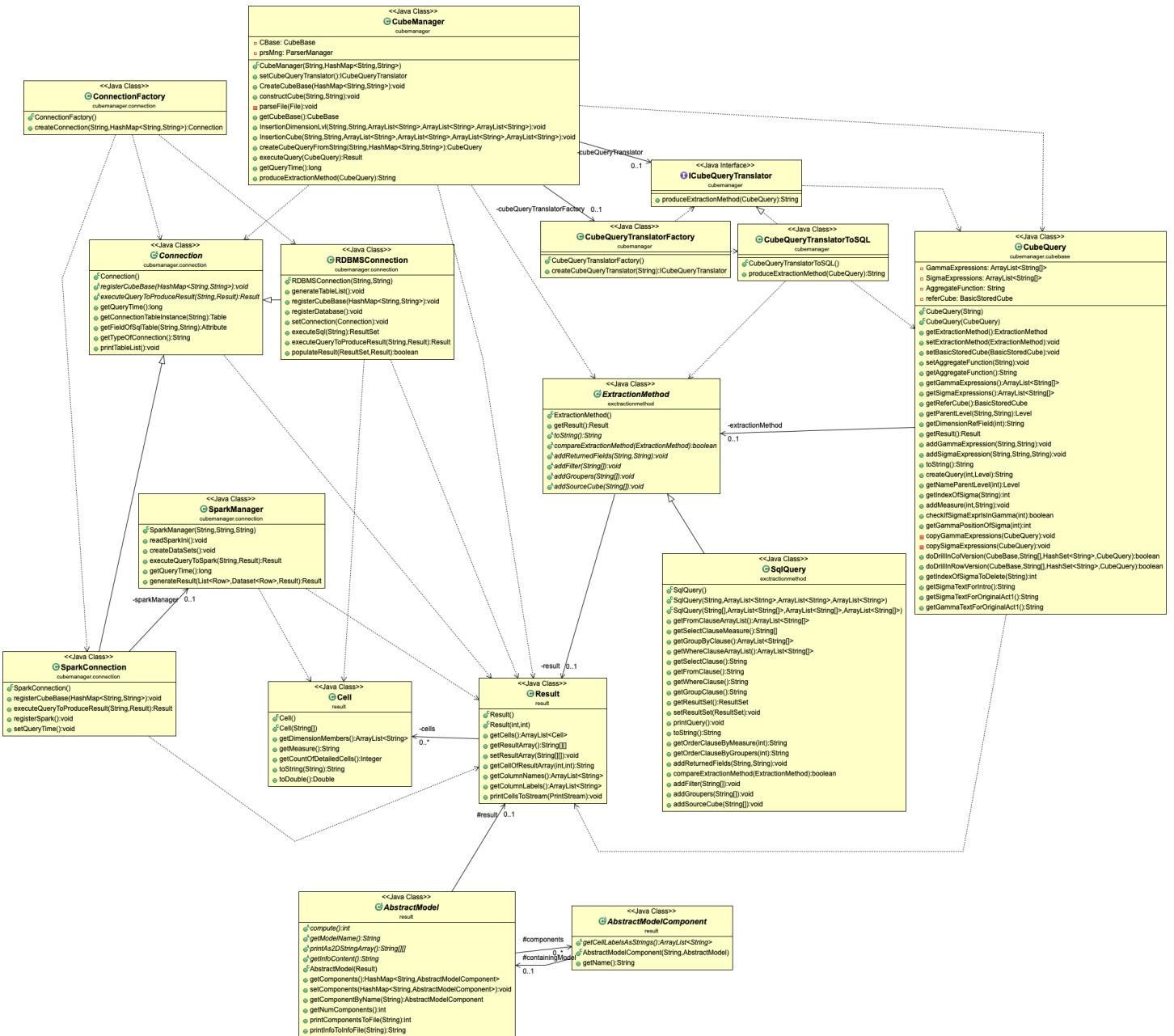


Σχήμα 3.2.3.7 : UML Class Diagram του πακέτου `cubemanager.cubebase`.



Σχήμα 3.2.3.8 : UML Class Diagram του πακέτου cubemanager.starschema.

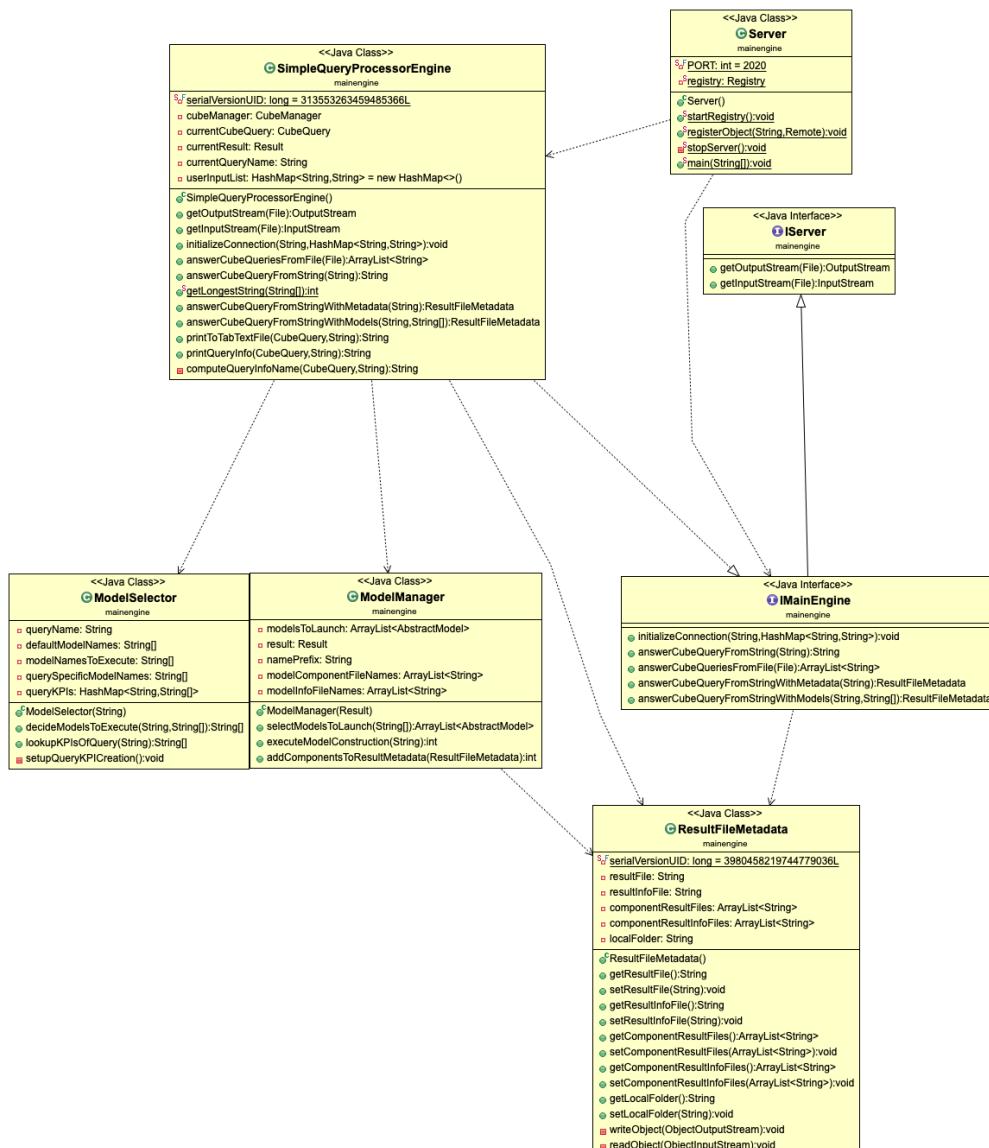
Ολοκληρώνοντας την αναφορά στο πακέτο `cubemanager` παρατίθεται στο Σχήμα 3.2.3.9 το UML Class Diagram ολόκληρου του πακέτου `cubemanager`, στο οποίο μπορούμε να δούμε την όλη σύνδεση της κλάσης `CubeManager` με τις υπόλοιπες κλάσεις του πακέτου.



Σχήμα 3.2.3.9 : Ολοκληρωμένο UML Class Diagram του πακέτου cubemanager.

3.1.3.4 UML Class Diagram του πακέτου mainengine

Το πακέτο mainengine περιέχει τις δύο λειτουργίες για την επικοινωνία του όλου συστήματος Client-Server. Αρχικά περιέχει την κλάση Server, η οποία πρέπει να εκτελείται πάντα πρώτη κατά την εκκίνηση του DelianCube Engine. Η κλάση Server δημιουργεί μία νέα σύνδεση σε μία θύρα (default θύρα 2020) και δημιουργεί ένα νέο αντικείμενο SimpleQueryProcessorEngine. Η κλάση SimpleQueryProcessorEngine αποτελεί τη δεύτερη σημαντική λειτουργία του πακέτου mainengine. Είναι το αντικείμενο το οποίο, όταν εκκινήσει ο Client και βρει επιτυχώς τη σύνδεση με τον Server, θα δοθεί στον Client και θα επιτρέπει την λειτουργία του συστήματος από τον χρήστη. Στο Σχήμα 3.2.3.10 μπορείτε να δείτε το UML Class Diagram του πακέτου mainengine.



Σχήμα 3.2.3.10 : UML Class Diagram του πακέτου mainEngine.

3.2 Σχεδίαση και αποτελέσματα ελέγχου του λογισμικού

Στο τέλος της ενσωμάτωσης του νέου συστήματος ελέγχαμε εάν η υλοποίηση του Spark έγινε σωστά με τη χρήση του Junit 4. Τα tests ελέγχουν εάν τα αποτελέσματα πριν το refactoring παρέμειναν σωστά μετά τις αλλαγές που κάναμε στον κώδικα. Επιπλέον ελέγχουν εάν το Spark παράγει τα ίδια αποτελέσματα με το RDBMS.

Για τον έλεγχο δημιουργήσαμε δύο Junit Suite Tests, ένα για τον έλεγχο της λειτουργικότητας του RDBMS και ένα για το Spark, το RDBMSSuiteTest και το SparkSuiteTest αντίστοιχα. Και τα δύο αυτά tests εκτελούν τους ίδιους ελέγχους, χρησιμοποιώντας το αντίστοιχο σύστημα. Στον Πίνακα 3.3.1 μπορούμε να δούμε αναλυτικά τα tests που εκτελούν τα δύο suites.

RDBMSSuiteTest	SparkSuiteTest
RDBMSInitializeTestsFail	SparkInitializeTestsFail
RDBMSAnswerQueryFromFileSuccess	SparkAnswerQueryFromFileSuccess
RDBMSAnswerQueryFromStringSuccess	SparkAnswerQueryFromStringSuccess
RDBMSAnswerQueryFromStringWithMetadataSuccess	SparkAnswerQueryFromStringWithMetadataSuccess
RDBMSAnswerQueryFromStringWithModelsSuccess	SparkAnswerQueryFromStringWithModelsSuccess

Πίνακας 3.3.1 : Τα Junit Tests που εκτελεί η κάθε Junit Suite.

Στο πρώτο test γίνεται ο έλεγχος ορθής λειτουργίας του initialization του προγράμματος. Εκτελεί τέσσερα tests με λανθασμένες αρχικές μεταβλητές και αναμένει την επιστροφή του κατάλληλου σφάλματος. Στον Πίνακα 3.3.2 βλέπουμε τα tests που εκτελούν το Junit Tests “InitializeTestsFail” με τα αναμενόμενα σφάλματα.

Method	Expected
typeOfConnectionIllegalArgumentException	IllegalArgumentException
cubeNameNullPointerException	NullPointerException
inputFolderNullPointerException	NullPointerException
schemaNameIOException	IndexOutOfBoundsException

Πίνακας 3.3.2 : Οι συναρτήσεις του “SparkInitializeTestsFail”.

Η πρώτη συνάρτηση “typeOfConnectionIllegalArgumentException” εκτελεί το initialization του Client, δίνοντας λάθος όρισμα στην μεταβλητή “typeOfConnection”, η οποία πρέπει να έχει όρισμα “RDBMS” ή “Spark”. Το αναμενόμενο σφάλμα σε αυτήν την περίπτωση είναι “IllegalArgumentException”.

Η δεύτερη συνάρτηση “cubeNameNullPoinerException” εκτελεί το initialization του Client, δίνοντας λάθος όρισμα στην μεταβλητή “cubeName”, η οποία πρέπει να περιέχει ένα σωστό όνομα ενός κύβου, τον οποίο παρέχουμε για αρχικοποίηση. Το αναμενόμενο σφάλμα σε αυτήν την περίπτωση είναι “NullPoinerException” καθώς δεν μπορεί να βρει το σωστό αρχείο και θα παραδώσει κενό αρχείο παρακάτω.

Η τρίτη συνάρτηση “inputFolderNullPoinerException” εκτελεί το initialization του Client, δίνοντας λάθος όρισμα στην μεταβλητή “inputFolder”, η οποία πρέπει να περιέχει ένα σωστό όνομα του φακέλου που περιέχει τα δεδομένα για αρχικοποίηση. Το αναμενόμενο σφάλμα σε αυτήν την περίπτωση είναι “NullPoinerException” καθώς δεν μπορεί να βρει το σωστό φάκελο για να την αρχικοποίηση του συστήματος.

Η τέταρτη συνάρτηση “schemaNameIOException” εκτελεί το initialization του Client, δίνοντας λάθος όρισμα στην μεταβλητή “schemaName”, η οποία πρέπει να περιέχει ένα σωστό όνομα ενός σχήματος δεδομένων. Το αναμενόμενο σφάλμα σε αυτήν την περίπτωση είναι “IndexOutOfBoundsException” καθώς δεν μπορεί να βρει το σωστό φάκελο για την αρχικοποίηση του συστήματος.

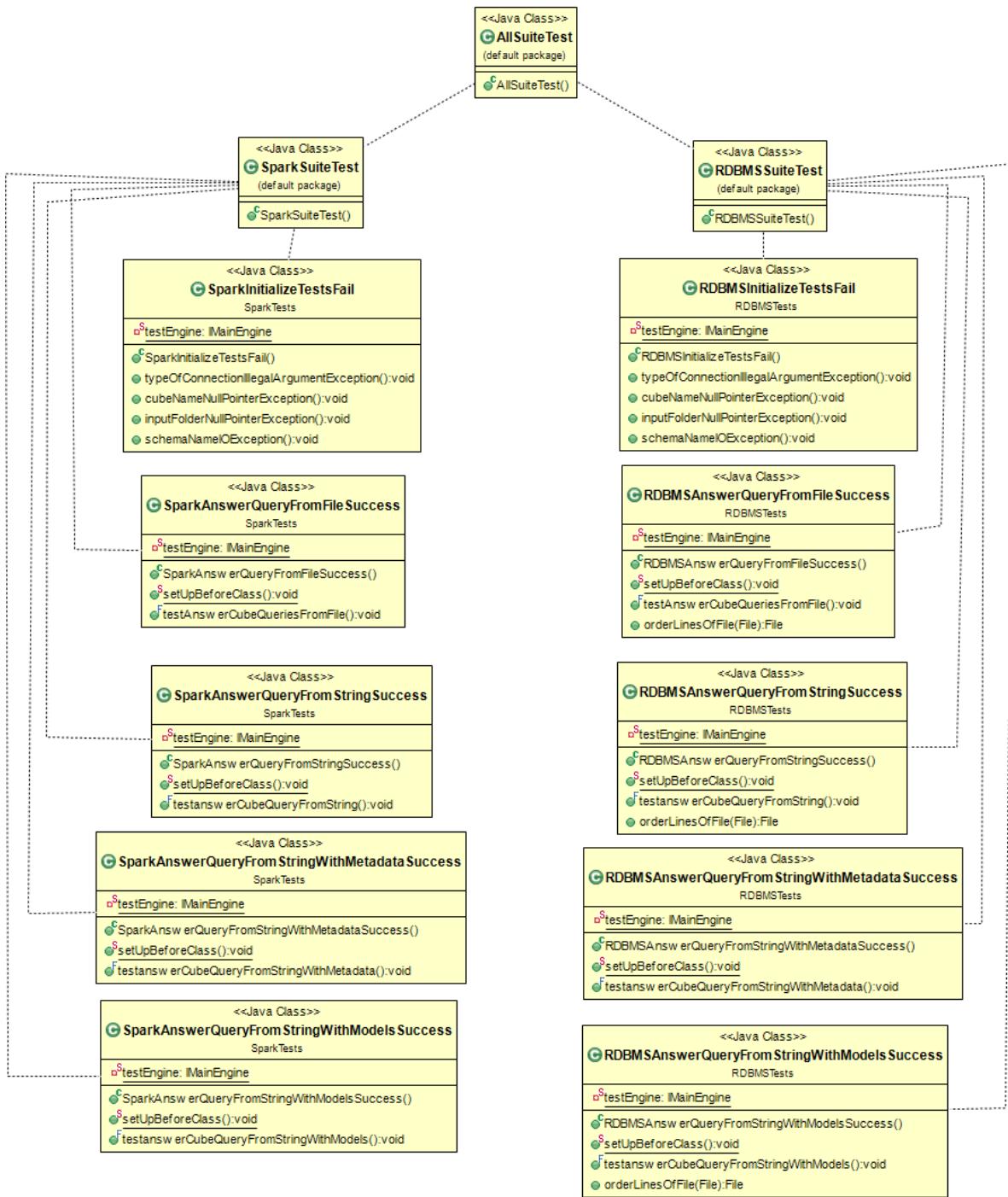
Όμοιο είναι και το περιεχόμενο του Junit Test “RDBMSInitializationTestsFail”.

Τα επόμενα Junit Tests ελέγχουν την ορθή εξαγωγή των αποτελεσμάτων από ερωτήματα. Για τον έλεγχο αυτό τρέξαμε αρχικά τα ερωτήματα στο πρόγραμμα DelianCube Engine το οποίο εξαγάγει τα ερωτήματα σε μορφή SQL. Στη συνέχεια τρέξαμε το ερώτημα SQL στη βάση δεδομένων MySQL και αποθηκεύσαμε τα αποτελέσματα σε ένα αρχείο. Έπειτα ελέγχουμε εάν τα αποτελέσματα που επιστρέφει το DelianCube Engine, στα δύο συστήματα RDBMS και Spark, είναι όμοια με αυτά που εξαγάγαμε από την MySQL. Τα ερωτήματα που τρέξαμε παρέχονται στο αρχείο “src/test/resources/InputFiles/10K-products/spark_sales_test_queries.txt” και “rdbms_sales_test_queries” για το Spark και RDBMS αντίστοιχα, τα οποία περιέχουν εφτά ερωτήματα. Κάθε ερώτημα εκτελείται εξάγεται το αποτέλεσμα και συγκρίνεται με τα αρχεία στον φάκελο “src/test/resources/OutputFiles/10K-products/SparkFiles/*”. Για τον έλεγχο του αποτελέσματος όλα τα αρχεία ταξινομούνται πριν τον έλεγχο καθώς παρατηρήθηκε διαφορετικός τρόπος ταξινόμησης μεταξύ των συστημάτων.

Αναλυτικά τον έλεγχο τα επόμενα Junit Tests είναι :

- Τα Junit Tests “AnswerQueryFromFileSuccess” ελέγχουν την εκτέλεση ερωτημάτων από την ανάγνωση αρχείων με ερωτήματα.
- Τα Junit Tests “AnswerQueryFromStringSuccess” ελέγχουν την εκτέλεση ερωτημάτων από την ανάγνωση ερωτημάτων από μία μεταβλητή “String”.
- Τα Junit Tests “AnswerQueryFromStringWithMetadataSuccess” ελέγχουν την σωστή εξαγωγή των Metadata από την εκτέλεση ενός ερωτήματος από μια μεταβλητή “String”.
- Τέλος τα Junit Tests “AnswerQueryFromStringWithModelsSuccess” ελέγχουν την σωστή εξαγωγή των αποτελεσμάτων για όλα τα μοντέλα μηχανικής μάθησης που προσφέρει το DelianCube Engine.

Έχοντας ολοκληρώσει τον έλεγχο της ορθότητας της υλοποίησης στο σύστημα, στο Σχήμα 3.3.1 παρουσιάζεται το UML Class Diagram των Junit Tests.



Εχίμα 3.3.1 : To UML Class Diagram των Junit Tests.

3.3 Λεπτομέρειες εγκατάστασης και υλοποίησης

Για την υλοποίηση, του συστήματος που υλοποιήσαμε, χρειάζονται δύο εγκαταστάσεις. Η πρώτη είναι η υλοποίηση της εγκατάστασης του Apache Spark στο cluster και η δεύτερη είναι η προετοιμασία των δεδομένων με την κατάλληλη δομή ώστε να μπορεί το σύστημά μας να φορτώσει επιτυχώς τα δεδομένα στο cluster.

Εγκατάσταση του Apache Spark στο cluster

Για την σύνδεση στο cluster, ανοίγουμε ένα τερματικό και εκτελούμε την εντολή :

```
ssh [username]@gatepc73.cs.uoi.gr -p[port]
```

Με τη παραπάνω εντολή θα συνδεθούμε στον master node του cluster. Στη συνέχεια θα συνδεθούμε στον node01, ο οποίος θα αποτελεί τον master node για το δικό μας σύστημα.

**Στο αρχείο /etc/hosts ορίζουμε τους κόμβους

Για να τρέξουμε το spark μέσω προγράμματος σε Java, που είναι η περίπτωσή μας, θα πρέπει να εγκαταστήσουμε πρώτα μερικά πακέτα για την εγκατάσταση των πακέτων Java που θα χρειαστούμε.

```
sudo apt-get install python-software-properties  
sudo add-apt-repository ppa:webupd8team/java  
sudo apt-get update  
sudo apt-get install oracle-java8-installer
```

Ολοκληρώνοντας τις παραπάνω εγκαταστάσεις, μπορούμε να ελέγξουμε αν έγινε επιτυχώς η εγκατάσταση εκτελώντας :

```
java -version
```

Στη συνέχεια θα πρέπει να ρυθμίσουμε το ssh. Σε περίπτωση που δεν είναι εγκατεστημένο στο σύστημά μας θα πρέπει να τρέξουμε την εντολή :

```
sudo apt-get install openssh-server openssh-client
```

Δημιουργούμε ένα νέο κλειδί ssh :

```
ssh-keygen -t rsa -P ""
```

Και αντιγράφουμε το περιεχόμενο του .ssh/id_rsa.pub (από τον μάστερ) στο αρχείο .ssh/authorized_keys (του κάθε σκλάβου). Τέλος για να ελέγξουμε αν έγινε επιτυχής η μεταφορά του κλειδιού ssh σε κάθε κόμβο σκλάβο προσπαθούμε να συνδεθούμε στον κάθε κόμβο μέσω "ssh nodeXX".

Το επόμενο βήμα που έχουμε να κάνουμε είναι να εγκαταστήσουμε το Spark στο cluster.

Στον master node τρέχουμε την εντολή :

```
wget http://www-us.apache.org/dist/spark/spark-2.3.0/spark-2.3.0-bin-hadoop2.7.tgz
```

Εξάγουμε το περιεχόμενο του αρχείου :

```
tar xvf spark-2.3.0-bin-hadoop2.7.tgz
```

Μεταφέρουμε το αρχείο στον φάκελο local :

```
sudo mv spark-2.3.0-bin-hadoop2.7 /usr/local/spark
```

Προσθέτουμε την τοποθεσία του spark στο bashrc, ανοίγοντας το αρχείο :

```
sudo nano ~/.bashrc
```

Και γράφοντας στο τέλος του αρχείου :

```
export PATH = $PATH:/usr/local/spark/bin
```

Αποθηκεύουμε το αρχείο και ανανεώνουμε το bashrc :

```
source ~/.bashrc
```

Δομή των αρχείων και ρυθμίσεις των initialization αρχείων

1. project_path.ini

Για την αρχικοποίηση του συστήματος θα πρέπει να ρυθμίσουμε τα αρχεία “ini” στο project. Η πρώτη μεταβλητή μου πρέπει να ορίσουμε είναι στο αρχείο **project_path.ini**. Το αρχείο αυτό περιέχει στην 1^η σειρά το path ως τον αρχικό φάκελο. Τέλος το αρχείο θα πρέπει να βρίσκεται στην τοποθεσία από την οποία θα εκτελέσουμε το project.

Για παράδειγμα έστω ότι έχουμε το project στον φάκελο :

```
C:\Users\admin\Desktop\JavaProject\DelianCubeEngine (1)
```

Βρισκόμαστε στην τοποθεσία :

```
C:\Users\admin\Desktop (2)
```

Θα πρέπει στην τοποθεσία (2) να έχουμε το αρχείο **project_path.ini** :

```
C:\Users\admin\Desktop\project_path.ini
```

Και στην πρώτη σειρά στο αρχείο “**project_path.ini**” να περιέχει το path (1) στο οποίο βρίσκεται το project.

2. naive.ini

Στο αρχείο **naive.ini** έχουμε τις ρυθμίσεις για την εκτέλεση του προγράμματος μέσω της κλάσης “**NaiveJavaClient**”. Η κλάση “**NaiveJavaClient**” αφορά την εκτέλεση του προγράμματος στο τερματικό, χωρίς δηλαδή την χρήση του GUI. Στο αρχείο αυτό θα πρέπει να ορίσουμε τις εξής μεταβλητές :

- schemaName = Το όνομα της ΒΔ που θέλουμε να χρησιμοποιήσουμε.
- cubeName = Το όνομα του κύβου που θέλουμε να χρησιμοποιήσουμε.
- inputFolder = Το όνομα του φακέλου της ΒΔ που χρησιμοποιούμε, συνήθως έχει το ίδιο όνομα με τη ΒΔ,
- username = Το όνομα του χρήστη στη ΒΔ. (Default = root)
- password = Τον κωδικό του χρήστη στη ΒΔ. (Default = password)
- queryFile = Το όνομα του αρχείου που περιέχει το query που θέλουμε να εκτελέσουμε.

- connectionType = Το είδος της σύνδεσης. (RDBMS ή Spark)
- runModels = Όρισμα εάν θέλουμε να εκτελέσουμε στα αποτελέσματα του query και κάποιο επιπλέον μοντέλο (YES ή NO).

Για παράδειγμα θέτουμε το αρχείο ως εξής :

```
schemaName = 100K-products
cubeName = sales
inputFolder = 100K-products
username = root
password = password
queryFile = Sales_Queries.txt
connectionType = Spark
runModels = No
```

Τέλος η τοποθεσία του αρχείου θα πρέπει να είναι στον αρχικό φάκελο του project :

```
C:\Users\admin\Desktop\JavaProject\DelianCubeEngine\naive.ini
```

3. Δημιουργία δεδομένων

Για την δημιουργία dummy data έχει γραφτεί ένα script σε Python. Ο χρήστης μπορεί να μεταφέρει το python αρχείο «products_script.py» στον φάκελο Data του κάθε σχήματος ΒΔ και να το εκτελέσει.

4. Το αρχείο του κύβου

Για την φόρτωση ενός κύβου στο DelianCube Engine και την ικανότητα εξαγωγής του SQL ερωτήματος θα πρέπει να ορίσουμε ένα αρχείο που να εκφράζει τη σχέση του fact table με τους dimension tables του κύβου. Στο αρχείο [cube_name].ini ορίζουμε τον κάθε dimension table ως εξής :

```

CREATE DIMENSION [dimension_table_name]_dim
RELATED SQL_TABLE [dimension_table_name]
LIST OF LEVEL {
    [dimension_table_name].[id_variable_name] AS lvl0,
    [dimension_table_name].[variable_name_of_1st_level] AS lvl1,
    [dimension_table_name].[variable_name_of_2nd_level] AS lvl2,
    ...
    ...
    [dimension_table_name].[variable_name_of_last_level] AS lvl[N]
}
HIERARCHY lvl0>lvl1>lvl2>...>...>lvl[N];

```

Και τέλος ορίζουμε τον fact table ως εξής :

```

CREATE CUBE [cube_table_name]_cube
RELATED SQL_TABLE [cube_table_name]
MEASURES [measure_name] AS [cube_table_name].[variable_name_to_measure]
REFERENCES DIMENSION
    [1st_dimension_table_name]_dim AT
    [cube_table_name].[foreign_key_to_dimension_table],
    [2nd_dimension_table_name]_dim AT
    [cube_table_name].[foreign_key_to_dimension_table],
    ...
    ...
    [Nth_dimension_table_name]_dim AT
    [cube_table_name].[foreign_key_to_dimension_table]

```

Το αρχείο πρέπει να έχει το όνομα του κύβου και πρέπει να βρίσκεται μέσα στον φάκελο του οποίου τα δεδομένα εκφράζει. Ο κύβος παραμένει όμοιος και για τα δύο συστήματα και περιγράφεται από το ίδιο ini αρχείο. Στην περίπτωση της αποκλειστικής χρήσης του συστήματος με το Spark, το “dimension_table_name” και “cube_table_name” είναι το όνομα του αρχείου csv που θέλουμε να χρησιμοποιήσουμε. Για τις τιμές “variable_name_of_n_level”, ορίζονται με τα attributes που ορίζουμε. Τα attributes αναλύονται παρακάτω στο Spark Initialization.

5. MySQL Initialization

Για την βάση δεδομένων, ο χρήστης μπορεί να βρει το σχήμα της ΒΔ καθώς και τις εντολές SQL για την φόρτωση των δεδομένων στο φάκελο “DB_Schema”. Παρέχεται σχήμα και κώδικας φόρτωσης δεδομένων για το λειτουργικό σύστημα Windows και Linux.

6. Spark Initialization

Για την αρχικοποίηση του Spark υπάρχει ένα αρχείο "spark.ini" μέσα σε κάθε φάκελο δεδομένων. Το αρχείο έχει τα εξής πεδία :

- Hadoop Home Directory (Default 'c:/hadoop') = Είναι ο φάκελος του Hadoop. Το default όρισμα είναι το c:/hadoop εάν δεν έχει γίνει κάποια αλλαγή σχετική με το Hadoop.
- Number Of Nodes (Default 'local[*]', to run at home) = Η μεταβλητή αυτή αφορά την εκτέλεση του Spark σε έναν υπολογιστή. Η default τιμή "local[*]" ορίζει να εκτελεστεί το Spark σε όλους τους πυρήνες του τοπικού υπολογιστή.
- Spark SQL Warehouse Directory (Default 'file:///c:/tmp/') = Η μεταβλητή ορίζει την τοποθεσία του Spark Warehouse. Αφορά μόνο την εκτέλεση σε τοπικό υπολογιστή και σε περίπτωση που δεν υπάρχουν αλλαγές σε αυτό το σύστημα η default τιμή είναι "file:///c:/tmp/".
- CSV Seperator Symbol = Ορίζουμε το σύμβολο με το οποίο διαχωρίζονται οι τιμές στα csv αρχεία μας. Στην δική μας περίπτωση η τιμή αυτή παίρνει το όρισμα ",".
- Cluster (Yes or No) = Η τιμή ενημερώνει το σύστημα εάν θέλουμε να τρέξουμε το σύστημα σε cluster. Κατά την εκτέλεση σε έναν τοπικό υπολογιστή ορίζουμε την τιμή ως "NO". Στην περίπτωση του cluster ορίζουμε την τιμή αυτή ως "YES".

Ένα παράδειγμα ενός ολοκληρωμένου αρχείου θα ήταν :

```
"Hadoop Home Directory (Default 'c:/hadoop')" = c:/hadoop  
"Number Of Nodes (Default 'local[*]', to run at home)" = local[*]  
"Spark SQL Warehouse Directory (Default 'file:///c:/tmp/')" =  
file:///c:/tmp/  
"CSV Seperator Symbol" = ,  
"Cluster (Yes or No)" = NO
```

To Apache Spark, αν και μπορεί να εγκαθιδρύσει επικοινωνία μέσω JDBC σε έναν RDBMS, δεν απαιτεί απαραίτητα την ύπαρξη του RDBMS για την ανάγνωση των δεδομένων. Στην δική μας περίπτωση επιλέξαμε να κάνουμε την ανάγνωση των δεδομένων από τα ήδη υπάρχοντα αρχεία csv που δημιουργήσαμε παραπάνω. Το Spark διαβάζει τα αρχεία τα μετατρέπει σε εικονικούς πίνακες και διαμοιράζει τα δεδομένα σε μορφή, πλέον, RDD σε όλους τους σκλάβους-κόμβους. Για τον λόγο αυτό στην αρχικοποίηση του Spark,

προκειμένου να μπορεί το σύστημα να μετατρέψει τα δεδομένα των αρχείων csv σε εικονικούς πίνακες, θα πρέπει να ορίσουμε επιπλέον αρχεία. Τα αρχεία αυτά εκφράζουν το είδος των μεταβλητών, όπως θα ήταν σε έναν σχεσιακό πίνακα δεδομένων και τα ονομάζουμε “attributes files”. Η τοποθεσία των αρχείων πρέπει να είναι μέσα σε έναν φάκελο μετονομασμένο με το όνομα του κύβου που θέλουμε να εξετάσουμε. Για κάθε αρχείο στον φάκελο Data πρέπει να υπάρχει και το αντίστοιχο αρχείο attribute στον φάκελο του κύβου. Η παραπάνω διαδικασία γίνεται για την δημιουργία των κύβων χωρίς τη χρήση μιας βάσης δεδομένων. Όσον αφορά το Spark φορτώνει τα δεδομένα στη RAM δημιουργώντας εικονικούς πίνακες, αυτόματα με την ανάγνωση των αρχείων, εφόσον έχουμε ορίσει αυτή τη λειτουργία στον κώδικα, ενεργοποιώντας την επιλογή “inferSchema”.

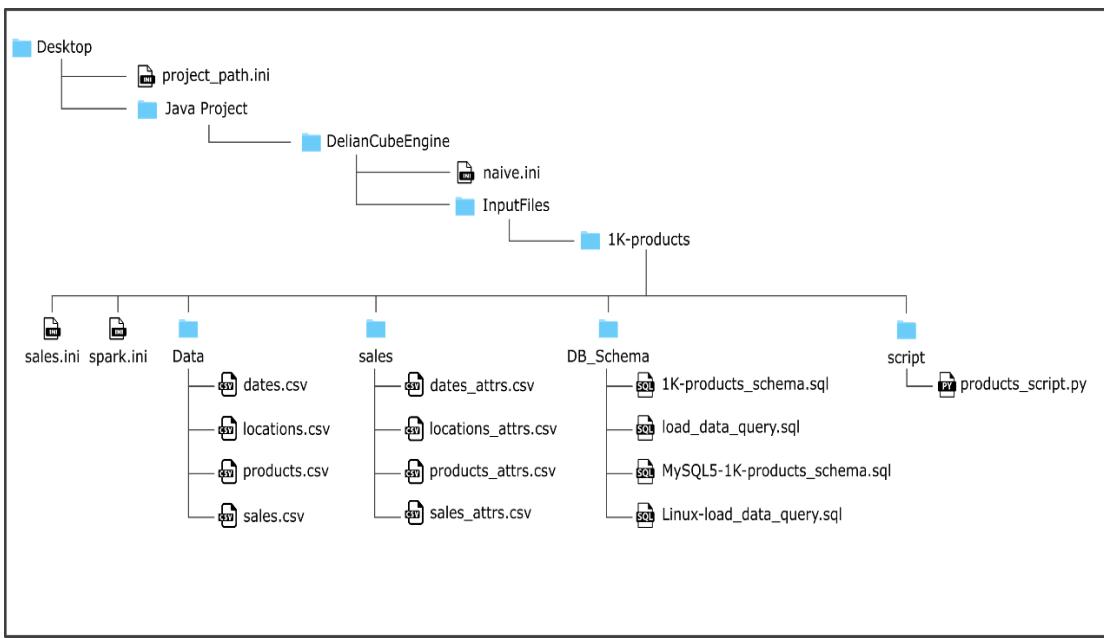
Παρακάτω δίνεται ένα παράδειγμα για την μορφή των attributes files. Έστω το αρχείο locations.csv στον φάκελο Data :

```
location_id,INT,city,VARCHAR,state,VARCHAR,ALL,VARCHAR
0,city8,state7
1,city10,state7
2,city1,state4
3,city6,state7
4,city1,state3
5,city5,state9
...
```

Το αντίστοιχο αρχείο attributes θα έχει την ονομασία “locations_attr.csv” και τη δομή :

```
location_id,INT,city,VARCHAR,state,VARCHAR,ALL,VARCHAR
```

Με την ολοκλήρωση και της αρχικοποίησης του Spark ολοκληρώθηκε η όλη διαδικασία που απαιτείται ώστε να τρέξε το πρόγραμμα. Στο Σχήμα 3.4.1 μπορείτε να δείτε σχηματικά την όλη δομή των αρχείων.



Σχήμα 3.4.1: Η δομή των αρχείων που περιγράψαμε στην αρχικοποίηση του συστήματος.

Κεφάλαιο 4. Πειραματική Αξιολόγηση

Στην ενότητα αυτή παρουσιάζουμε αναλυτικά την πειραματική αξιολόγηση της μεθόδου μας.

4.1 Μεθοδολογία πειραματισμού

Αφού υλοποιήσαμε την ενσωμάτωση του Apache Spark στο DelianCube Engine κάναμε μετρήσεις σχετικά με την ταχύτητα εκτέλεσης των SQL query μεταξύ του συστήματος RDBMS και Apache Spark. Παρακάτω δίνονται όλες οι πληροφορίες για τον τρόπο εκτέλεσης των πειραμάτων.

1. SQL Query

Χρησιμοποιήθηκε το ερώτημα :

```
CubeName:sales  
Name:CubeQuerySales  
AggrFunc:Avg  
Measure:sales  
Gamma:products_dim_LVL0,locations_dim_LVL0  
Sigma:dates_dim_LVL3='2'
```

Το οποίο παράγει το παρακάτω ερώτημα σε SQL μορφή :

```
SELECT products.product_id, locations.location_id,Avg(sales) as measure,COUNT(*) as  
countOfDetailedCells  
FROM sales, dates, products, locations  
WHERE sales.date_id = dates.date_id AND dates.month = '2' AND sales.product_id =  
products.product_id AND sales.location_id = locations.location_id  
GROUP BY products.product_id, locations.location_id  
ORDER BY measure ASC
```

2. Μέγεθος δεδομένων

Δημιουργήσαμε 5 διαφορετικά μεγέθη των αρχείων products, locations, dates και sales με τον παρακάτω αριθμό δεδομένων για το καθένα :

	Lines Of Data
Configuration 1	1.000 (1K)
Configuration 2	10.000 (10K)
Configuration 3	100.000 (100K)
Configuration 4	1.000.000 (1M)
Configuration 5	10.000.000 (10M)

3. Συστήματα εκτέλεσης του κώδικα

Η εκτέλεση των δεδομένων έγινε στο σύστημα :

- RDBMS, το οποίο διαβάζει τα αρχεία από μία Β.Δ. MySQL. Κατά τη χρήση του MySQL χρησιμοποιήθηκαν indexes B-Tree.
- Apache Spark χρησιμοποιώντας από 1 έως και 10 κόμβους, το οποίο διαβάζει τα δεδομένα από τα αρχεία csv και τα φορτώνει στην μνήμη RAM σε εικονικούς πίνακες σε μορφή DataSets.

4. Hardware

Η εκτέλεση των μετρήσεων έγινε στο cluster της σχολής, όπου η εκτέλεση του RDBMS έγινε σε έναν κόμβο, ενώ του Spark από 1 έως και 10 κόμβους. Τα χαρακτηριστικά του cluster είναι :

- Κάθε κόμβος του cluster αποτελείται από ένα Sun Fire X4100 Server με 4 CPUs (Dual Core AMD Opteron στα 2193 MHz κάθε CPU).
- Κάθε κόμβος έχει 12GB Ram.

Για την εκτέλεσή μας χρησιμοποιήθηκαν 3GB RAM σε κάθε κόμβο. Εξετάστηκε και η χρήση με παραπάνω μνήμη και δεν έχει αλλαγή στην ταχύτητα εκτέλεσης.

5. Αριθμός Επαναλήψεων

Για κάθε συνδυασμό των παραπάνω, εκτελέσαμε το ερώτημα από 10 φορές. Αυτό, π.χ. σημαίνει ότι εκτελέσαμε το query σε σύστημα RDBMS, με μέγεθος 1000 εγγραφές 10 φορές. Ομοίως για τα άλλα μεγέθη αρχείων. Έπειτα εκτελέσαμε το ερώτημα στο Spark ομοίως με όλους τους συνδυασμούς 10 φορές καθώς και για όλα τα διαφορετικά πλήθη των κόμβων, από 1 έως 10 κόμβους-σκλάβους.

6. Κώδικας Εκτέλεσης των μετρήσεων

Ο κώδικας εκτέλεσης των επαναλήψεων μπορεί να βρεθεί στο αρχείο “src/main/java/client/naiveJavaClient/NaiveJavaClientRunAllTimes”. Το αρχείο αυτό είναι μια υλοποίηση της NaiveJavaClient η οποία τρέχει αυτοματοποιημένα τις 10 επαναλήψεις για πολλά αρχεία αρχικοποίησης naive.ini. Τα αρχεία αρχικοποίησης naive μπορούν να βρεθούν στον φάκελο “naiveTestFiles”.

Για την εξαγωγή των χρόνων εκτέλεσης έγινε μια προσθήκη στην κλάση “SimpleQueryProcessorEngine”, στη συνάρτηση “answerCubeQueryFromString” που μπορεί να ενεργοποιηθεί/απενεργοποιηθεί και έχει πρακτικότητα μόνο στην περίπτωση που θέλουμε να κάνουμε μετρήσεις, δηλαδή να εκτελέσουμε πολλαπλά ερωτήματα. Για κάθε συνδυασμό εκτέλεσης δημιουργεί ένα αρχείο csv στον φάκελο “OutputFiles/script_times”, όπου το κάθε αρχείο περιέχει τις 10 μετρήσεις του κάθε συνδυασμού. Τα εξαγόμενα δεδομένα είναι τα εξής :

- **model** : Το όνομα του μοντέλου που εκτελέστηκε.
- **total_result** : Το πλήθος των επιστρεφόμενων αποτελεσμάτων.
- **query_time** : Ο χρόνος εκτέλεσης του query.
- **result_time** : Ο χρόνος δημιουργίας του Result με τα επιστρεφόμενα δεδομένα.
- **execution_time** : Το άθροισμα του χρόνου query_time και result_time.
- **output_time** : Ο χρόνος εξαγωγής των δεδομένων.
- **total_time** : Ο συνολικός χρόνος εκτέλεσης.

4.2 Αναλυτική παρουσίαση αποτελεσμάτων

Στο σημείο αυτό θα παρουσιάσουμε τις μετρήσεις μας. Η ανάλυση των δεδομένων και οι γραφικές παραστάσεις που αναπαρίστανται παρακάτω μπορούν να βρεθούν στον αρχείο “OutputFiles/script_times/Time_plots.ipynb”. Για κάθε σχήμα που παρουσιάζεται θα γίνεται και η αντίστοιχη ανάλυση των αποτελεσμάτων της.

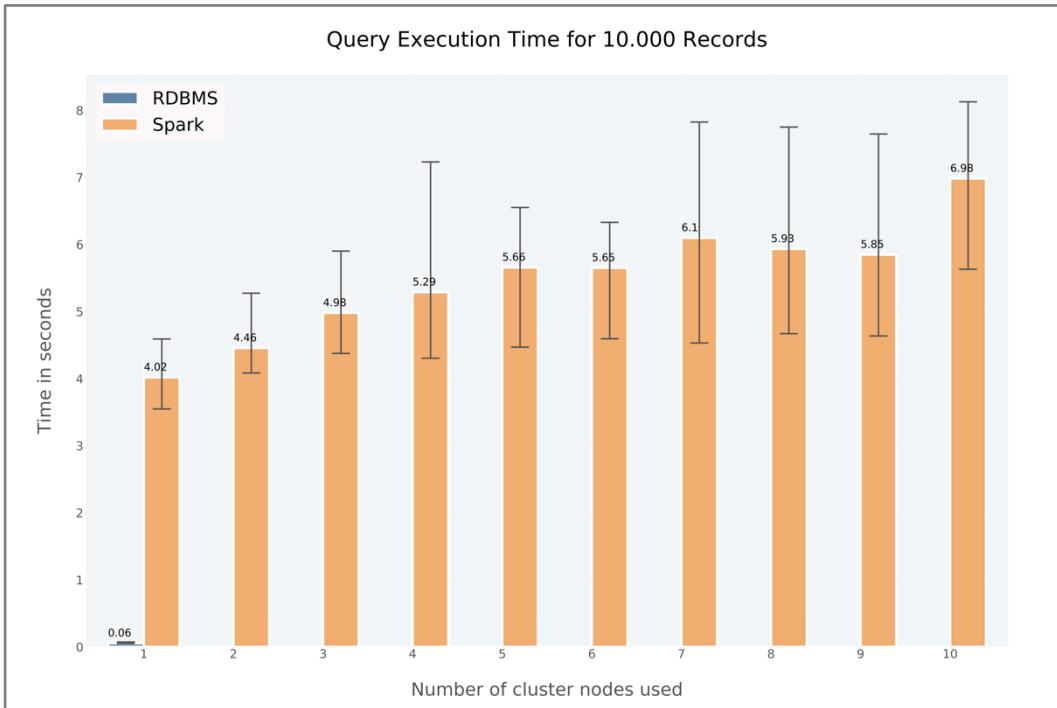
Στα παρακάτω σχήματα βλέπουμε τον χρόνο εκτέλεσης του συστήματος RDBMS και του Apache Spark με 1 έως 10 κόμβους-σκλάβους σε όλα τα μεγέθη δεδομένων.



Σχήμα 4.2.1 : Μέσος χρόνος εκτέλεσης του SQL query, σε δευτερόλεπτα, ως συνάρτηση του συστήματος που εκτελέστηκε. Οι αποκλίσεις υποδεικνύουν την τιμή min και max, σε κάθε σύστημα, στις 10 επαναλήψεις που έγιναν. Αφορά τα δεδομένα για το Configuration 1.

Στο Σχήμα 4.2.1 βλέπουμε ότι το σύστημα RDBMS για την ανάλυση πολύ μικρού μεγέθους δεδομένων, για το Configuration 1, είναι πολύ πιο αποτελεσματικό. Συγκεκριμένα το RDBMS έκανε μηδαμινό χρόνο εκτέλεσης του ερωτήματος. Αντίθετα το Spark απαιτεί την μετατροπή των δεδομένων σε RDD, στη συνέχεια την διαμοίραση των δεδομένων από τον κόμβο-αφέντη στους κόμβους-σκλάβους. Έπειτα την επαναποστολή των αποτελεσμάτων του ερωτήματος από τους κόμβους-σκλάβους στον κόμβο-αφέντη και μετατροπή των RDD σε αναγνώσιμη μορφή της Java. Η διαδικασία αυτή είναι πιο

χρονοβόρα από την εκτέλεση του ερωτήματος και γι' αυτό έχουμε σχετικά σταθερούς χρόνους όσους κόμβους και να επιλέξουμε.



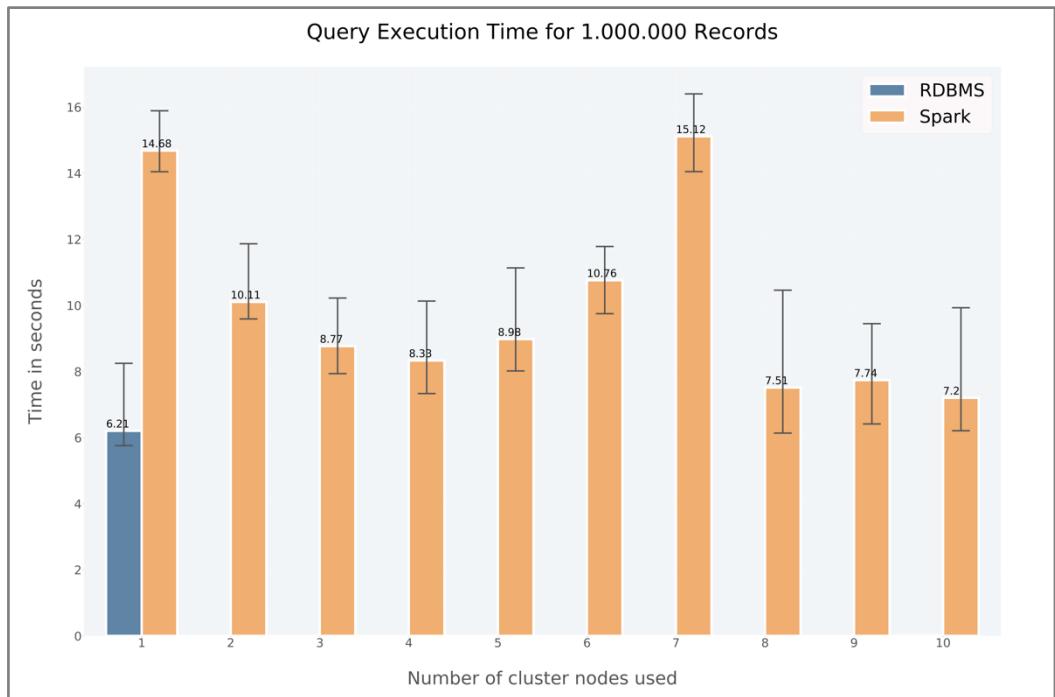
Σχήμα 4.2.2 : Μέσος χρόνος εκτέλεσης του SQL query, σε δευτερόλεπτα, ως συνάρτηση του συστήματος που εκτελέστηκε. Οι αποκλίσεις υποδεικνύουν την τιμή min και max, σε κάθε σύστημα, στις 10 επαναλήψεις που έγιναν. Αφορά τα δεδομένα για το Configuration 2.

Τα ίδια αποτελέσματα παρατηρούμε και στο Σχήμα 4.2.2, όπου παρουσιάζονται οι μετρήσεις στα δεδομένα για το Configuration 2. Επίσης βλέπουμε ότι η αύξηση των κόμβων επιφέρει και αύξηση στον χρόνο εκτέλεσης. Αυτό οφείλεται, όπως αναλύσαμε και παραπάνω, στην καθυστέρηση αποστολής-λήψης των δεδομένων. Η αύξηση των κόμβων αυξάνει και την διαδικασία αυτή.



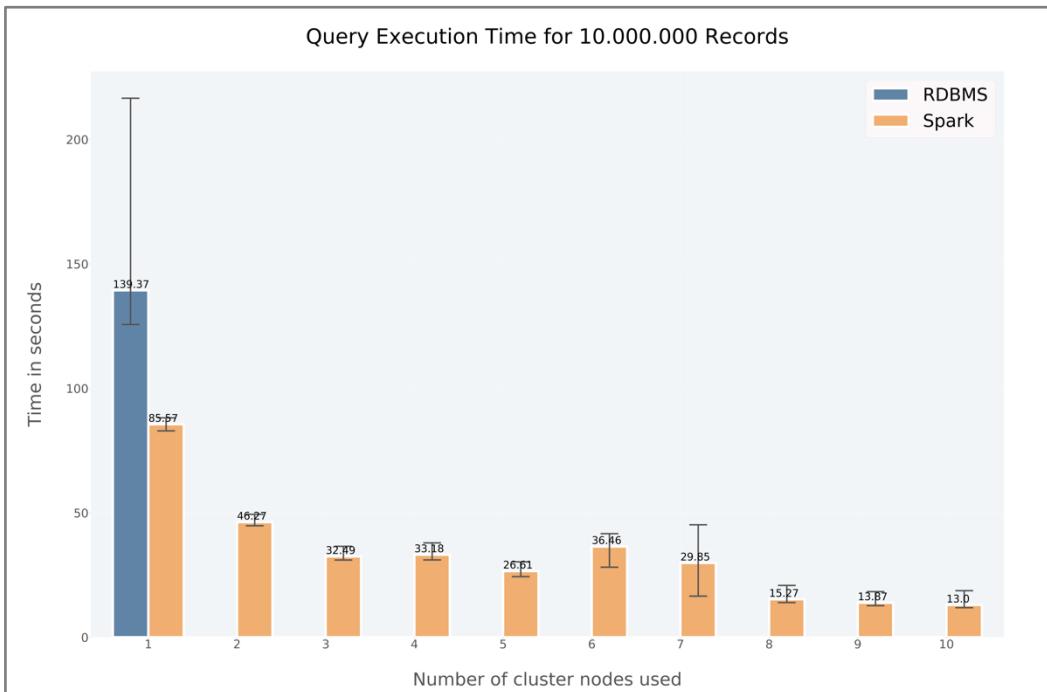
Σχήμα 4.2.3 : Μέσος χρόνος εκτέλεσης του SQL query, σε δευτερόλεπτα, ως συνάρτηση του συστήματος που εκτελέστηκε. Οι αποκλίσεις υποδεικνύουν την τιμή min και max, σε κάθε σύστημα, στις 10 επαναλήψεις που έγιναν. Αφορά τα δεδομένα για το Configuration 3.

Αντίστοιχα αποτελέσματα παρατηρούνται και στα δεδομένα για το Configuration 3 στο Σχήμα 4.2.3.



Σχήμα 4.2.4 : Μέσος χρόνος εκτέλεσης του SQL query, σε δευτερόλεπτα, ως συνάρτηση του συστήματος που εκτελέστηκε. Οι αποκλίσεις υποδεικνύουν την τιμή min και max, σε κάθε σύστημα, στις 10 επαναλήψεις που έγιναν. Αφορά τα δεδομένα για το Configuration 4.

Στο Σχήμα 4.2.4 όπου παρουσιάζουμε τα αποτελέσματα στα δεδομένα για το Configuration 4, παρατηρούμε ότι πλέον το σύστημα RDBMS απαιτεί κατά μέσο όρο 6.21 δευτερόλεπτα, τη στιγμή που το σύστημα Apache Spark παραμένει σταθερό στους χρόνους του, κυρίως στους 10 κόμβους, στα 7,2 δευτερόλεπτα. Επιπλέον φαίνεται μια αύξηση του χρόνου εκτέλεσης του ερωτήματος SQL στις μετρήσεις με τους 6 και 7 κόμβους. Αυτό μπορεί να οφείλεται στο ότι οι κόμβοι που προστέθηκαν, είχαν κάποια απασχόληση από μια άλλη διεργασία και να υπήρχε καθυστέρηση στην εκτέλεση του ερωτήματος στους κόμβους αυτούς. Η καθυστέρηση των κόμβων αυτών καθυστερεί και τον ολικό χρόνο εξαγωγής του αποτελέσματος καθώς οι υπόλοιποι κόμβοι θα πρέπει να περιμένουν την εξαγωγή του αποτελέσματος από όλους του κόμβους. Ωστόσο με την εισαγωγή του 8^{ου} κόμβου και μετά βλέπουμε ότι ο χρόνος μειώνεται πάλι. Αυτό μπορεί να οφείλεται στο ότι οι κόμβοι που καθυστερούσαν την διεργασία πλέον δεν εκτελούν κάποια άλλη διεργασία που ενδεχομένως εκτελούσαν πριν, είτε στο ότι φόρτο εργασίας μοιράζεται και οι κόμβοι 6 και 7 είναι πλέον σε θέση να εκτελέσουν γρηγορότερα το ερώτημα παρά την ύπαρξη μιας άλλης διεργασίας.



Σχήμα 4.2.5 : Μέσος χρόνος εκτέλεσης του SQL query, σε δευτερόλεπτα, ως συνάρτηση του συστήματος που εκτελέστηκε. Οι αποκλίσεις υποδεικνύουν την τιμή min και max, σε κάθε σύστημα, στις 10 επαναλήψεις που έγιναν. Αφορά τα δεδομένα για το Configuration 5.

Τέλος στο Σχήμα 4.2.5 βλέπουμε τους χρόνους εκτέλεσης στα δεδομένα για το Configuration 5. Όπως φαίνεται στο σχήμα για μεγάλα δεδομένα ο χρόνος του συστήματος RDBMS είναι πολύ μεγάλος ενώ ο χρόνος του Apache Spark είναι ολοένα και πιο γρήγορος με την εισαγωγή και νέου σκλάβου, με το σύστημα Spark να τρέχει 10

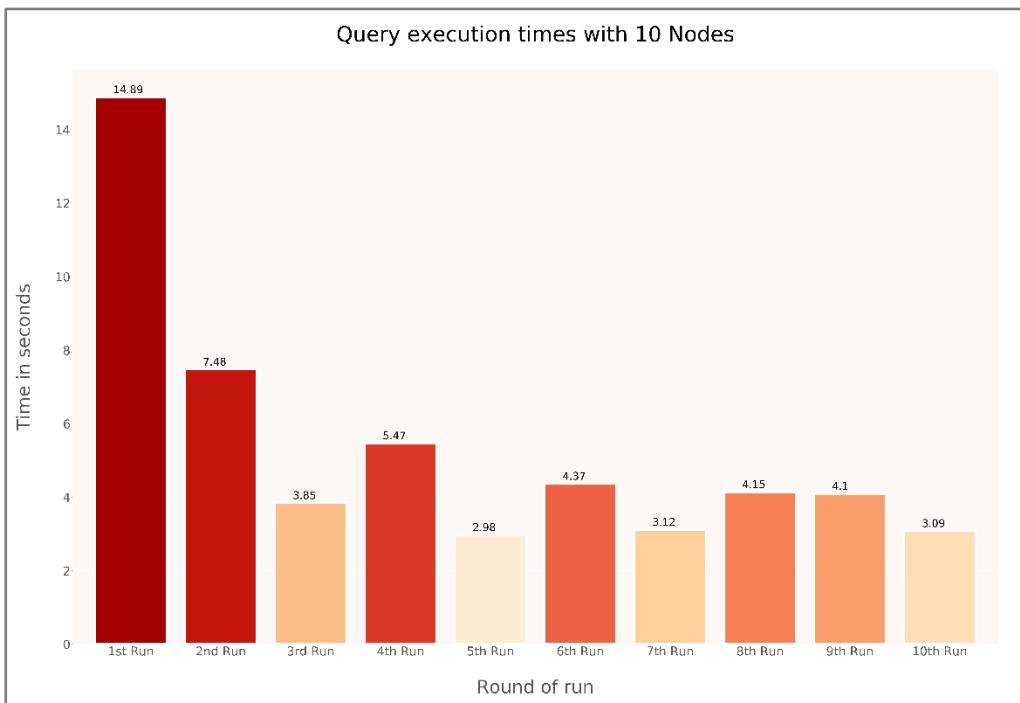
φορές πιο γρήγορα από το RDBMS κατά μέσω όρο. Παρατηρείται, όπως και στο προηγούμενο σχήμα, μια αύξηση με την συμμετοχή του κόμβου 6 και 7 για τους ίδιους λόγους που αναλύσαμε και παραπάνω. Επίσης παρατηρούμε ότι ακόμη και με έναν κόμβο το Spark είναι και πάλι πιο γρήγορο από το RDBMS. Αυτό οφείλεται στις επιπλέον βελτιστοποιήσεις που κάνει το Spark στο παρασκήνιο στο ερώτημα SQL πριν την εκτέλεσή του.

Στον Πίνακα 4.2.1 βλέπουμε τα ποσοστά απόδοσης των παραπάνω αποτελεσμάτων. Ο υπολογισμός του ποσοστού γίνεται με τη συνάρτηση « $f(x,y)=(x-y)/x*100$ » αν το σύστημα RDBMS είναι πιο γρήγορο, ή « $f(x,y)=(y-x)/y*100$ » αν το σύστημα Spark είναι πιο γρήγορο, όπου x είναι ο μέσος χρόνος εκτέλεσης του RDBMS και y ο μέσος χρόνος εκτέλεσης του Spark.

Data	1 Node	2 Node	3 Node	4 Node	5 Node	6 Node	7 Node	8 Node	9 Node	10 Node
1K	-52775%	-42206%	-42968%	-43964%	-49610%	-56484%	-55361%	-56728%	-59107%	-60023%
10K	-7167%	-7959%	-8903%	-9461%	-10131%	-10120%	-10928%	-10629%	-10476%	-12528%
100K	-849%	-952%	-1001%	-1068%	-1119%	-1194%	-1217%	-1327%	-1406%	-1397%
1M	-136%	-62%	-41%	-34%	-44%	-73%	-143%	-21%	-24%	-16%
10M	+62%	+201%	+328%	+320%	+423%	+282%	+366%	+812%	+904%	+971%

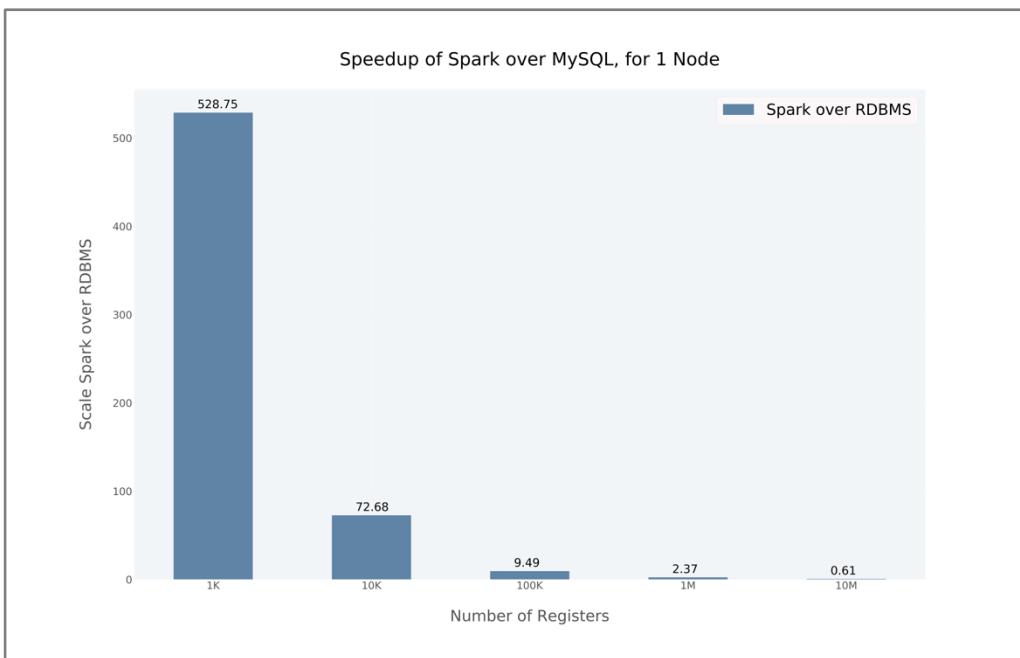
Πίνακας 4.2.1 : Ποσοστό απόδοσης του συστήματος Spark συγκριτικά με το σύστημα RDBMS.

Από τις παραπάνω γραφικές παραστάσεις βλέπουμε επίσης ότι το Apache Spark έχει πολλά δευτερόλεπτα απόκλισης στους χρόνους του, στις 10 επαναλήψεις που γίνονται σε κάθε σετ δεδομένων. Για παράδειγμα όπως φαίνεται στο Σχήμα 4.2.1 στο σετ δεδομένων για το Configuration 1, ο μέσος χρόνος εκτέλεσης του Spark με 10 κόμβους είναι τα 5.35 δευτερόλεπτα, ενώ φαίνεται η χειρότερη περίπτωση να έκανε περίπου 14 δευτερόλεπτα και η βέλτιστη περίπτωση 3 περίπου δευτερόλεπτα. Αυτό οφείλεται λόγω του ότι κάποιο ερώτημα μπορεί να έχει επαναληφθεί και να έχει αποθηκευτεί το αποτέλεσμά του στην cache. Στο Σχήμα 4.2.6 φαίνεται ο χρόνος εκτέλεσης σε σχέση σε ποια εκτέλεση βρισκόμαστε.



Σχήμα 4.2.6 : Ο χρόνος εκτέλεσης του ερωτήματος για κάθε εκτέλεση στο σύστημα Spark με 10 κόμβους και τα δεδομένα 1K.Ο χρωματισμός είναι ταξινομημένος αντίστοιχα με τον χρόνο εκτέλεσης, με το σκούρο χρώμα να εμφανίζεται η μεγαλύτερη καθυστέρηση.

Τέλος στο Σχήμα 4.2.7 συγκρίνουμε τους χρόνους του συστήματος Spark σε σύγκριση με το σύστημα RDBMS. Ο υπολογισμός του χρόνου γίνεται με τον τύπο **[Time of Spark for Config x]/[Time of RDBMS for Config x]**, για τη χρήση ενός node, όπου το Config είναι το πλήθος των εγγραφών.



Κεφάλαιο 5. Επίλογος

Στην ενότητα αυτή συνοψίζουμε τη συνεισφορά και τα αποτελέσματα της εργασίας και παραθέτουμε σκέψεις για μελλοντικές επεκτάσεις της.

5.1 Σύνοψη και συμπεράσματα

Στόχος της διπλωματικής εργασίας ήταν να επεκτείνουμε το υπάρχον σύστημα Delian Cube Engine ώστε να ενσωματωθεί η λειτουργικότητα του συστήματος Apache Spark. Η ενσωμάτωση ήταν επιτυχής με τις κατάλληλες αλλαγές και προσθήκες στον υπάρχων κώδικα του συστήματος Delian Cube Engine. Η εφαρμογή παρέχει γραφικό περιβάλλον με το οποίο ο χρήστης μπορεί να αλληλοεπιδράσει με το σύστημα Delian Cube Engine. Συνοψίζοντας το σύστημα Delian Cube Engine πλέον είναι σε θέση να προσφέρει την ανάλυση δεδομένων είτε με τη χρήση RDBMS είτε σε συστάδα υπολογιστών με το σύστημα Spark. Για την αξιολόγηση της βελτίωσης της ταχύτητας που προσφέρει το Spark σε σχέση με το RDBMS έγιναν μετρήσεις στο cluster της σχολής. Τα συμπεράσματα μας είναι ότι :

- Το Spark προσφέρει μια πιο απλοποιημένη λύση στο τελικό χρήστη, όσον αφορά την μη ανάγκη μεταφόρτωσης των δεδομένων σε ένα σύστημα RDBMS.
- Η χρήση ενός cluster κάνει την ανάλυση μεγάλων δεδομένων πολύ πιο γρήγορη σε σχέση με μία βάση δεδομένων τύπου RDBMS.

5.2 Μελλοντικές επεκτάσεις

Ολοκληρώνοντας αυτή τη διπλωματική εργασία θα αναφερθούμε σε μελλοντικές επεκτάσεις που θα μπορούσαν να υλοποιηθούν στο παρόν project.

Στην παραπάνω εργασία ενσωματώσαμε μία από τις τέσσερις βιβλιοθήκες που προσφέρει το Spark, την Spark SQL. Μία σημαντική επέκταση θα είναι να ενσωματώσουμε την βιβλιοθήκη Spark MLlib, η οποία περιέχει αλγορίθμους μηχανικής μάθησης που μπορούν να εκτελεστούν στα δεδομένα που παράγουμε. Οι αλγόριθμοι που περιέχει η MLlib συμπεριλαμβάνουν clustering, regression, decision tree, recommendation, clustering και άλλες.

Μια επιπλέον επέκταση θα μπορούσε να γίνει στο υπάρχον σύστημα ως προς τη χρήση διαφορετικών cluster managers και συστήματα αποθήκευσης των δεδομένων. Το Apache Spark μπορεί να χρησιμοποιήσει και άλλους cluster managers όπως τον Kubernetes ή τον Apache Mesos και να συγκρίνουμε την αποτελεσματικότητά τους με τον Standalone Cluster Manager που παρέχει το Spark ως προεπιλογή. Επιπλέον μπορεί να γίνει χρήση διαφορετικών συστημάτων αποθήκευσης δεδομένων όπως το Hadoop, το Apache Cassandra, Apache Hive και άλλα.

Τέλος ως προς το γραφικό περιβάλλον που παρέχεται στον τελικό χρήστη θα μπορούσε να γίνει μια προσπάθεια μετατροπής του υπάρχοντος project σε ένα δια-δραστικό web-application. Η υλοποίηση θα μπορούσε να γίνει μια μία βιβλιοθήκη front-end όπως το React και τη χρήση ως back-end ένα σύστημα όπως το Spring Boot.

Βιβλιογραφία

- [ASR19] RDD Programming Guide – Spark 2.4.4 Documentation (2019, November 7). Retrieved from
<https://spark.apache.org/docs/latest/rdd-programming-guide.html>
- [ASS19] Spark SQL and DataFrames – Spark 2.4.4 Documentation (2019, November 7). Retrieved from
<https://spark.apache.org/docs/latest/sql-programming-guide.html>
- [DFA19] Apache Spark RDD vs DataFrame vs Dataset – DataFlair (2019, November 7). Retrieved from <https://data-flair.training/blogs/apache-spark-rdd-vs-dataframe-vs-dataset/>
- [DFB20] A Tale Of Three Apache Spark APIs – DataFrailr (2020, July 15). Retrieved from <https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html>
- [JPT10] Christian S. Jensen, Torben Back Pedersen, Christian Thomsen. Multidimensional Databases and Data Warehousing. Morgan and Claypool Publishers, pp. 1-24, 2010.
- [MVN20] Apache Maven Project – Maven (2020, July 15). Retrieved from
<https://maven.apache.org>
- [ORA19] The Multidimensional Data Model – Oracle (2019, November 7). Retrieved from
https://docs.oracle.com/cd/B13789_01/olap.101/b10333/multimodel.htm
- [WOC19] Wikipedia – OLAP Cube (2019, November 7). Retrieved from
https://en.wikipedia.org/wiki/OLAP_cube
- [WAS20] Wikipedia – Apache Spark (2020, July 15). Retrieved from
https://en.wikipedia.org/wiki/Apache_Spark

