ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Βελτιστοποίηση χειρισμού Μεγάλων Ενεργειακών Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

των

ΠΑΝΑΓΙΩΤΑΣ ΑΝΤΩΝΙΑΔΟΥ
ΛΑΜΠΡΟΥ ΣΕΚΛΙΖΙΩΤΗ

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2013

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

Βελτιστοποίηση χειρισμού Μεγάλων Ενεργειακών Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

των

ΠΑΝΑΓΙΩΤΑΣ ΑΝΤΩΝΙΑΔΟΥ
ΛΑΜΠΡΟΥ ΣΕΚΛΙΖΙΩΤΗ

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Δεκεμβρίου 2013.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
....	....	....
Ιωάννης Βασιλείου	Κωνσταντίνος Κοντογιάννης	Ιωάννης Σταύρακας
Καθηγητής Ε.Μ.Π.	Αναπληρωτης Καθηγητής Ε.Μ.Π.	Ερευνητής Β' ΙΠΣΥ/Ε.Κ. "Αθηνά"

Αθήνα, Δεκέμβριος 2013

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

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

Copyright © Παναγιώτα Αντωνιάδου, 2013
Copyright © Λάμπρος Σεκλιζιώτης, 2013
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τους συγγραφείς.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τους συγγραφείς και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας είναι η εφαρμογή διαφόρων τεχνικών βελτιστοποίησης σε μία βάση PostgreSQL πολύ μεγάλου όγκου δεδομένων. Η βάση αυτή ανήκει σε γνωστή εταιρεία που αναπτύσσει λογισμικό με σκοπό την παρακολούθηση φωτοβολταϊκών πάρκων. Η ροή των εισερχόμενων δεδομένων προς τη βάση είναι συνεχόμενη, αφού η εξόρυξη τους γίνεται με χρήση χρονοσειρών. Λόγω του μεγέθους της βάσης ο χρόνος απόκρισης είναι μεγάλος, με αποτέλεσμα οι χρήστες συχνά να δυσκολεύονται να αντλήσουν την πληροφορία που τους ενδιαφέρει. Αρχικός μας στόχος λοιπόν είναι να βρούμε τρόπους που θα κάνουν τα ερωτήματα πιο γρήγορα και τα δεδομένα πιο άμεσα διαχειρίσιμα. Στο πρώτο μέρος της εργασίας, θα περιγράψουμε την αρχική δομή της βάσης έτσι όπως μας δόθηκε και θα εξηγήσουμε τις αδυναμίες που εντοπίσαμε. Αφού περιγράψουμε την υπάρχουσα εφαρμογή, θα παραθέσουμε τις τεχνικές βελτιστοποίησης που εφαρμόσαμε και τα αποτελέσματα που πήραμε από κάθε πείραμα που κάναμε. Τα πειράματα που υλοποιήσαμε επιβεβαιώνουν την κακή κλιμάκωση των σχεσιακών βάσεων και για το λόγο αυτό στο δεύτερο μέρος της διπλωματικής θα ασχοληθούμε με το στήσιμο και την μελέτη μίας μη σχεσιακής NoSQL βάσης. Αφού κάνουμε αναλυτική περιγραφή των πλεονεκτημάτων και των μειονεκτημάτων μίας τέτοιας βάσης, θα παραθέσουμε τα αποτελέσματα της απόδοσής της πάνω στο σύστημα που έχουμε στη διάθεσή μας. Τέλος, με βάση τα αποτελέσματα από όλα τα πειράματα θα προτείνουμε αλλαγές που θα βελτιώσουν την απόδοση του συστήματος.

Λέξεις κλειδιά: <<σχεσιακές βάσεις, rdbms, big data, PostgreSQL, βελτιστοποίηση βάσεων δεδομένων, indexing, batch insertion, διαχωρισμός δεδομένων, μη σχεσιακές βάσεις, NoSQL, Cassandra, bulk loading>>

Abstract

The purpose of this diploma thesis is the application of optimization techniques on big data. The object of this study is a PostgreSQL database of approximately 1 terabyte, which came at our disposal from a well-known company. This company is building software for the monitoring of photovoltaic plants. Therefore, the data we are handling are measurements that describe each plant's performance. These data are generated by a timeseries mechanism that works continuously, so we always have new inserts in the database. At the same time, the users are applying queries on the database regarding present or past measurements. Our goal is to make this whole process more efficient by optimizing the mostly used queries and by changing the current structure of the database. In the first part of our thesis, we are going to test several optimization techniques on the relational model we were given and observe their results on the queries' performance. In the second part, we are going to examine the non-relational database field by studying the features of NoSQL database systems. In particular, we are transferring the database we had from PostgreSQL to Cassandra and compare basic operations such as backup and restore of a database dump or execute simple queries such as selecting and inserting a great amount of rows. Finally, considering the results from both parts we are going to form a proposal of changes on the database that will improve its performance.

Key words: <<relational databases, rdbms, big data, PostgreSQL, database optimization, indexing, batch insertion, partitioning, non-relational databases, NoSQL, Cassandra, bulk loading>>

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο ΙΠΣΥ/Ε.Κ. «Αθηνά» και αποτέλεσε μια πολύ καλή αφορμή να ασχοληθούμε με τη διαχείριση βάσεων μεγάλου όγκου δεδομένων. Στο σημείο αυτό θα θέλαμε να ευχαριστήσουμε τον καθηγητή κ. Ιωάννη Βασιλείου για την ευκαιρία που μας προσέφερε να ασχοληθούμε με ένα τέτοιο θέμα. Επίσης, θα θέλαμε να ευχαριστήσουμε ιδιαίτερα τον μεταδιδακτορικό ερευνητή κ. Μανώλη Τερροβίτη από το ΙΠΣΥ για την επίβλεψη της διπλωματικής και την συνεχή καθοδήγηση, καθώς και τον υποψήφιο διδάκτορα Θανάση Ζυγομήτρο, για την πολύτιμη βοήθεια που μας προσέφεραν, τις χρήσιμες συμβουλές τους και την γενικώς πολύ καλή συνεργασία που είχαμε καθ' όλη τη διάρκεια της διπλωματικής. Τέλος, ευχαριστούμε θερμά την εταιρεία inAccess για την παραχώρηση των δεδομένων που αποτέλεσαν το βασικό αντικείμενο μελέτης μας.

Πίνακας περιεχομένων

1 Εισαγωγή 21

1.1 Βάσεις μεγάλου όγκου δεδομένων 21

1.2 Αντικείμενο Διπλωματικής 24

1.3 Οργάνωση κειμένου 24

2 Σχεσιακές Βάσεις 27

2.1 Ιστορική Αναδρομή 27

2.2 Τι είναι ένα σύστημα RDBMS 28

2.3 Το Σχεσιακό Μοντέλο Δεδομένων 29

2.4 Ο Δωδεκάλογος του Dr.Codd για το Σχεσιακό Μοντέλο Βάσης Δεδομένων 30

2.5 ACID 31

2.6 Κλειδιά και αναφορική ακεραιότητα 31

2.7 Σχεσιακή Άλγεβρα 32

3 PostgreSQL 33

3.1 Τι είναι η PostgreSQL 33

3.2 Σύντομη ιστορική αναδρομή 33

3.2.1 Η Εργασία POSTGRES του πανεπιστημίου BERKELEY 33

3.2.2 Postgres95 35

3.2.3 PostgreSQL 36

3.3 Γενικά χαρακτηριστικά 36

3.4 Συμβατότητα 38

4 phpPgAdmin 39

4.1 Χαρακτηριστικά 39

5 Πρόβλημα 41

5.1 Ο πίνακας measurement_events 41

5.2 Ο πίνακας table_X 42

5.3 Λειτουργία 44

6 Βέλτιστος αρχικός σχεδιασμός συστήματος 47

6.1 Λογικός σχεδιασμός βάσης 47

6.2 Φυσικός σχεδιασμός βάσης 47

6.3 Απαιτούμενες προδιαγραφές Hardware 48

6.4 Άλλα θέματα tuning 48

7 Τεχνικές Βελτιστοποίησης 49

7.1 PostgreSQL Configuration 49

7.1.1 Memory Consumption 49

7.1.2 Write Ahead Log 49

7.1.3 Query Tuning 50

7.2 Indexing 50

7.2.1 Θεωρία 50

7.2.2 Πείραμα 51

7.2.2.1 Περιγραφή 51

7.2.2.2 Μετρήσεις 53

7.2.3 Λεπτομέρειες 54

7.3 Ενδιάμεσοι πίνακες 55

7.3.1 Θεωρία 55

7.3.2 Πείραμα 55

7.3.2.1 Περιγραφή 55

7.3.2.2 Μετρήσεις 59

7.3.3 Λεπτομέρειες 61

7.4 In-Memory Tables 64

7.4.1 Θεωρία 64

7.4.2 Πείραμα 65

7.4.2.1 Περιγραφή 65

7.4.2.2 Μετρήσεις 65

7.4.3 Λεπτομέρειες 67

7.5 Partitioning 68

7.5.1 Θεωρία 68

7.5.2 Πείραμα 68

7.5.2.1 Περιγραφή 68
7.5.2.2 Μετρήσεις 73
7.5.3 Λεπτομέρειες 76
7.6 Batch Insertion 76
7.6.1 Θεωρία 76
7.6.2 Πείραμα 77
7.6.2.1 Περιγραφή 77
7.6.2.2 Μετρήσεις 78
7.6.3 Λεπτομέρειες 79
7.7 Replication 79
7.8 Άλλες τεχνικές 80
7.8.1 Distributed caching layer 80
7.8.2 Denormalization 80
7.9 Πρόταση λύσης με εφαρμογή τεχνικών βελτιστοποίησης σε σχεσιακή βάση 81
8 Μη σχεσιακές Βάσεις Δεδομένων 85
8.1 Γενικά 85
8.2 Τύποι μη σχεσιακών ΒΔ. 86
8.2.1 Key-value Stores (Αποθήκες κλειδιών-τιμών) 86
8.2.2 Column-oriented Databases (Βάσεις δεδομένων προσανατολισμένες στη στήλη) 87
8.2.3 Document-based Stores (Αποθήκες βασισμένες στα έγγραφα) 87
8.2.4 Graph databases 87
8.3 Χαρακτηριστικά μη σχεσιακών συστημάτων ΒΔ. 88
8.3.1 Ανθεκτικότητα 88
8.3.2 Προσαρμοστικότητα 88
8.3.3 Διαχειρισιμότητα 88
8.3.4 Παράλληλη επεξεργασία 88
8.3.5 Διαθεσιμότητα 89
8.3.6 Υψηλή απόδοση 89
8.3.7 Αναπαραγωγή δεδομένων 89

8.3.8 Schema-less persistence – Δυναμικά Schemas 89
8.3.9 Κλιμάκωση 90
8.3.10 Ενσωματωμένο caching 90
8.3.11 Μικρότερο κόστος 91
8.4 Θεώρημα CAP 91
8.5 BASE 92
8.6 MapReduce 93
9 Cassandra 95
9.1 Γενικά 95
9.2 Ιστορικά 96
9.3 Χαρακτηριστικά 97
9.3.1 Κατανεμημένη και αποκεντρωμένη 97
9.3.2 Ελαστική κλιμάκωση 97
9.3.3 Υψηλή διαθεσιμότητα και ανοχή σε βλάβες 98
9.3.4 Υψηλή απόδοση 98
9.3.5 Schema-Free 99
9.3.6 Row-oriented 99
9.3.7 Ρυθμιζόμενη συνέπεια 99
9.3.7.1 Αυστηρή συνέπεια 100
9.3.7.2 Αιτιώδης συνέπεια 100
9.3.7.3 Ασθενής (τελική) συνέπεια 100
9.4 Βασικές τεχνικές διαφορές Cassandra και υπόλοιπων NoSQL και RDBMSs 101
9.5 Use cases της Cassandra 102
9.5.1 Εφαρμογές πολλών κόμβων 102
9.5.2 Πολλά writes, στατιστικά και ανάλυση 102
9.5.3 Γεωγραφική κατανομή 103
9.5.4 Εφαρμογές σε εξέλιξη 103
9.6 Μερικές εταιρείες που χρησιμοποιούν Cassandra 103
9.7 Εισαγωγή στο μοντέλο δεδομένων της Cassandra 104
9.8 Βασικές Λειτουργίες στην Cassandra 107

9.8.1 Writes στην Cassandra ... 107
9.8.2 Δοσοληψίες και έλεγχος συγχρονισμού ... 107
9.8.3 Inserts και Updates ... 108
9.8.4 Deletes ... 109
9.8.5 Reads στην Cassandra ... 110
9.8.6 Client Requests στην Cassandra ... 110
9.8.6.1 Write Requests ... 110
9.8.6.2 Multi-Data Center Write Requests ... 111
9.8.6.3 Read Requests ... 112
9.8.7 Αναπαραγωγή δεδομένων στην Cassandra ... 113
9.8.8 Διαχωρισμός δεδομένων στην Cassandra ... 116
9.8.8.1 Διανομή δεδομένων στο δαχτυλίδι (Ring) ... 116
9.8.8.2 Τύποι διαχωριστών ... 116
10 Πείραμα σε NoSQL σύστημα ... 119
10.1 Εγκατάσταση της Cassandra 1.2 ... 119
10.2 Εισαγωγή και ανάκτηση δεδομένων από αρχείο ... 120
10.2.1 Μέθοδος COPY ... 120
10.2.2 Μέθοδος bulk loading ... 121
10.2.3 Μετρήσεις ... 128
10.2.4 Λεπτομέρειες ... 130
10.3 Εισαγωγή και Επιλογή δεδομένων ... 130
10.4 Πρόταση συνδυαστικής λύσης με σχεσιασκή και μη σχεσιακή βάση ... 134
11 Συμπεράσματα ... 137
11.1 Σύνοψη ... 137
11.1.1 Απόδοση του υπάρχοντος συστήματος ... 137
11.1.2 Σύνοψη λύσης με εφαρμογή των τεχνικών βελτιστοποίησης ... 138
11.1.3 Λύση με συνδυασμό σχεσιακής και μη σχεσιακής βάσης ... 138
11.1.4 Σύγκριση ... 139
11.2 Επίλογος-Μελλοντικές επεκτάσεις ... 140
12 Βιβλιογραφία ... 141

Ευρετήριο Εικόνων

Εικόνα 1. Στατιστικά στοιχεία χρήσης big data σε όλους τους τομείς....23
Εικόνα 2. Χρήση big data σε όλο τον κόσμο....23
Εικόνα 3. Query Plan για ερώτημα που ελέγχει το check constraint των υποπινάκων....74
Εικόνα 4. Query Plan για ερώτημα που δεν ελέγχει το check constraint των υποπινάκων....75
Εικόνα 5. Θεώρημα CAP....92
Εικόνα 6. Σχήμα σχεσιακής βάσης....105
Εικόνα 7. Σχήμα μη σχεσιακής βάσης....106
Εικόνα 8. Αίτημα ενός client για γράψιμο σε single Data Center....111
Εικόνα 9. Αίτημα ενός client για γράψιμο σε multi Data Center....112
Εικόνα 10. Αίτημα ενός client για διάβασμα σε single Data Center....113
Εικόνα 11. Αναπαραγωγή δεδομένων με τη μέθοδο SimpleStrategy....114
Εικόνα 12. Αναπαραγωγή δεδομένων με τη μέθοδο NetworkStrategy....115
Εικόνα 13. Μεταφορά από αρχείο .csv στη βάση με bulk loading για πίνακα με ένα εκατομμύριο εγγραφές....128
Εικόνα 14. Μεταφορά από αρχείο .csv στη βάση με COPY για πίνακα με ένα εκατομμύριο εγγραφές....128
Εικόνα 15. Μεταφορά από αρχείο .csv στη βάση με bulk loading για πίνακα με ένα δισεκατομμύριο εγγραφές....129
Εικόνα 16. Μεταφορά από αρχείο .csv στη βάση με COPY για πίνακα με ένα δισεκατομμύριο εγγραφές....129
Εικόνα 17. Screenshot από την προσομοίωση των inserts στην Cassandra....131
Εικόνα 18. Screenshot από την εκτέλεση της εντολής select σε Cassandra....133

Ευρετήριο Πινάκων

Πίνακας 1. Τελεστές σχεσιακών δομών 32

Πίνακας 2. Οριακές τιμές της PostgreSQL 36

Πίνακας 3. Τύποι στην PostgreSQL 37

Πίνακας 4. Χαρακτηριστικά του πίνακα measurement_events 41

Πίνακας 5. Παράδειγμα εγγραφής του measurement_events 42

Πίνακας 6. Χαρακτηριστικά του βοηθητικού πίνακα table_X 42

Πίνακας 7. Query 1 με where-clause σε Indexed Column 51

Πίνακας 8. Query 2 με where-clause σε Unindexed Column 51

Πίνακας 9. Query για την εύρεση της τελευταίας μέτρησης για ένα συγκεκριμένο id 52

Πίνακας 10. Αποτελέσματα ερωτήσεων με κριτήριο αναζήτησης indexed και unindexed στήλες για 10000 ids. 53

Πίνακας 11. Αποτελέσματα της Συνάρτησης 1 σε πίνακα με σωστό και «χαλασμένο» index 53

Πίνακας 12. Εντολή για Inserting σε κανονικό indexed πίνακα από τον βοηθητικό για ένα εκατομμύριο γραμμές 59

Πίνακας 13. Αποτελέσματα μετρήσεων με χρήση και μη ενδιάμεσων πινάκων για υπολογισμό aggregates. 60

Πίνακας 14. Query για τη δημιουργία του πίνακα current_day 62

Πίνακας 15. Query για τη δημιουργία του πίνακα current_week 62

Πίνακας 16. Query για τη δημιουργία του πίνακα current_month 62

Πίνακας 17. script1.sql 64

Πίνακας 18. script2.sql 64

Πίνακας 19. script3.sql 64

Πίνακας 20. Αποτελέσματα μετρήσεων με χρήση ενδιάμεσων, in-memory πινάκων. 65

Πίνακας 21. Εντολή για τη δημιουργία του trigger στον πίνακα που έχει γίνει partitioning .. 72

Πίνακας 22. Χρόνοι εκτέλεσης για την εισαγωγή δεδομένων μίας μέρας 73

Πίνακας 23. Εντολή για δημιουργία index στον υποπίνακα της τρέχουσας μέρας 73

Πίνακας 24. Εντολή για τον υπολογισμό αθροίσματος και μέσου όρου για όλα τα ids για μία ημέρα 75

Πίνακας 25. Εντολή για εμφάνιση όλων των εγγραφών για ένα id για ένα μήνα 76

Πίνακας 26. Στατιστικά εισαγωγής δεδομένων στο αρχικό σύστημα 76

Πίνακας 27. Script σε C για την δημιουργία αρχείου .sql......78
Πίνακας 28. Αρχείο .sql με 1<=i<=10000......78
Πίνακας 29. Αποτελέσματα......78
Πίνακας 30. Εντολή για τη μεταφορά του πίνακα από τη βάση σε αρχείο .csv......121
Πίνακας 31. Εντολή για τη μεταφορά του πίνακα από το .csv στη βάση......121
Πίνακας 32. Εντολή δημιουργίας αρχείου .csv......122
Πίνακας 33. Εντολή δημιουργίας keyspace στην Cassandra......122
Πίνακας 34. Εντολή δημιουργίας column family στην Cassandra όμοιου με τον πίνακα measurement_events της PostgreSQL......123
Πίνακας 35. DataImport.java......126
Πίνακας 36. run.sh......127
Πίνακας 37. Αποτελέσματα......130
Πίνακας 38. Χρόνοι εκτέλεσης insert σε PostgreSQL και Cassandra......131
Πίνακας 39. Χρόνοι εκτέλεσης select σε PostgreSQL και Cassandra......133
Πίνακας 40. Συγκεντρωτικός πίνακας για τη σύγκριση των συστημάτων......139

Ευρετήριο Γραφημάτων

Γράφημα 1. Λειτουργία του συστήματος με ή χωρίς τη χρήση ενδιάμεσων πινάκων 61

Γράφημα 2. Λειτουργία του συστήματος με ή χωρίς τη χρήση ενδιάμεσων in memory πινάκων 66

Γράφημα 3. Λειτουργία του συστήματος με τη χρήση ενδιάμεσων πινάκων στο δίσκο και στη μνήμη RAM. 66

Γράφημα 4. Συγκεντρωτικό διάγραμμα σύγκρισης στο χρόνο εκτέλεσης των inserts 67

Γράφημα 5. Χρόνος εκτέλεσης 10000 inserts ανάλογα με τη συχνότητα των commits. 79

Γράφημα 6. Σύγκριση insert σε PostgreSQL και Cassandra..132

Γράφημα 7. Σύγκριση select σε PostgreSQL και Cassandra..133

Ευρετήριο Συναρτήσεων

Συνάρτηση 1 - Κώδικας plpgsql..52

Συνάρτηση 2 - Κώδικας plpgsql..56

Συνάρτηση 3 - Κώδικας plpgsql..57

Συνάρτηση 4. Κώδικας plpgsql 59

Συνάρτηση 5 - Κώδικας plpgsql..70

Συνάρτηση 6 - Κώδικας plpgsql..71

Συνάρτηση 7 - Κώδικας plpgsql..72

1 *Εισαγωγή*

1.1 Βάσεις μεγάλου όγκου δεδομένων

Οι προμηθευτές των γνωστών συστημάτων βάσεων και αποθηκών δεδομένων έλεγαν πως ο όρος big data απλά αναφέρεται σε βάσεις που περιέχουν πάρα πολύ μεγάλο αριθμό γραμμών (tuples) στους πίνακές τους. Το μέγεθός τους συνήθως είναι της τάξης ενός ή μερικών terabytes και οι γραμμές δεδομένων που είναι αποθηκευμένες μπορεί να φτάνουν έως και μερικά δισεκατομμύρια. Τα τελευταία χρόνια όμως ο ορισμός των big data έχει αλλάξει. Η εξέλιξη του Web έχει επαναπροσδιορίσει την ταχύτητα με την οποία κινείται η πληροφορία μέσα στα online συστήματα. Οι πελάτες των εταιρειών δεν καθορίζονται πλέον από γεωγραφικά κριτήρια, αφού το διαδίκτυο επιτρέπει άμεσες συναλλαγές διεθνούς εμβέλειας, με αποτέλεσμα ο αριθμός τους να αυξάνεται κατακόρυφα. Επίσης μεγάλη εξέλιξη έχει πραγματοποιηθεί και όσον αφορά τα είδη των δεδομένων που επεξεργάζονται και παρακολουθούνται μέσω του internet. Τέλος το χρονικό διάστημα μεταξύ της στιγμής που μπαίνουν τα δεδομένα σε κάποιο σύστημα και της στιγμής που μετατρέπονται σε πληροφορία που μπορεί να αναλυθεί και να χρησιμοποιηθεί θα πρέπει να ανταποκρίνεται στις νέες απαιτήσεις των χρηστών και να συμβαδίζει με τη γενικότερη μείωση της ταχύτητας που έχει επέλθει σε όλες τις λειτουργίες.

Μερικοί αναλυτές έχουν προσπαθήσει να κατηγοριοποιήσουν τις αλλαγές αυτές εξετάζοντας τα big data ως εξής:

- **Ταχύτητα**: πόσο γρήγορα έρχονται τα δεδομένα
- **Ποικιλία**: υπάρχει πληθώρα τύπων δεδομένων
- **Όγκος**: terabytes και petabytes δεδομένων
- **Πολυπλοκότητα**: μετακίνηση δεδομένων μεταξύ διαφορετικών πλατφορμών, χειρισμός τους μεταξύ πολλών servers σε διάφορα σημεία της γης

Λόγω των παραπάνω, έχουν δοθεί νέοι ορισμοί για τα big data, που στρέφονται κυρίως στις τεχνολογίες που χειρίζονται τέτοια δεδομένα.

Η IDC (International Data Corporation), κορυφαία εταιρεία παροχής υπηρεσιών σχετικά με τεχνολογίες πληροφορίας και τηλεπικοινωνίες, ορίζει τα big data ως εξής:

«Οι τεχνολογίες big data περιγράφουν μια νέα γενιά τεχνολογιών και αρχιτεκτονικών, που έχουν σχεδιαστεί για να εξάγουν γρήγορα και οικονομικά πληροφορία από τεράστιο όγκο και μεγάλη ποικιλία δεδομένων.»

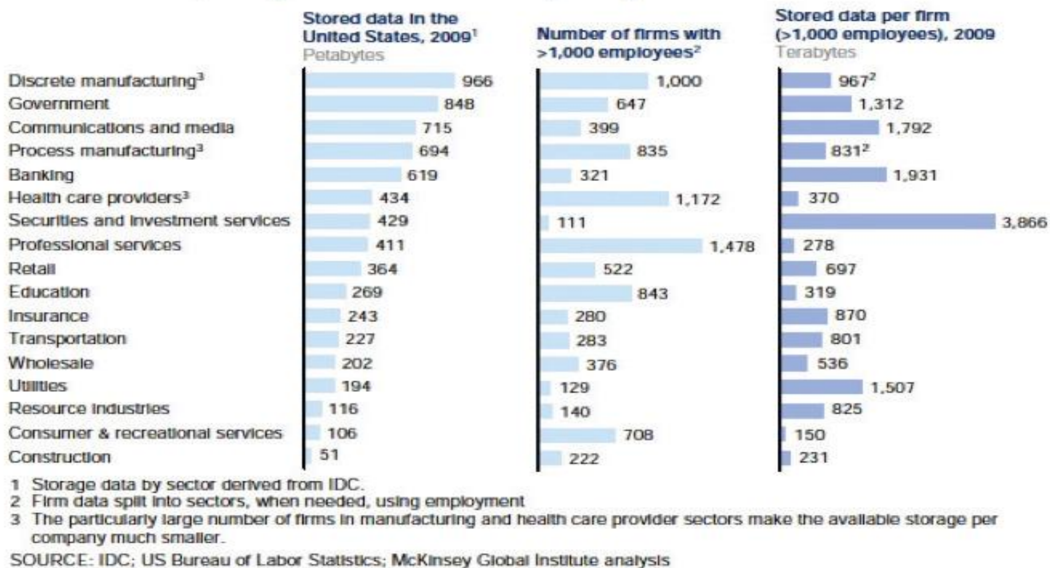
Ο Davig Kellogg, διευθύνων σύμβουλος της εταιρείας Host Analytics, δίνει έναν πιο απλό ορισμό, χαρακτηρίζοντάς τα ως «δεδομένα όγκου πολύ μεγάλου για να μπορούν να χειριστούν από τις υπάρχουσες τεχνολογίες».

Ίδιο ορισμό δίνει και η McKinsey & Co, πολυεθνική εταιρεία συμβούλων, λέγοντας πως «είναι σύνολα δεδομένων των οποίων το μέγεθος είναι πέρα από τις δυνατότητες των τυπικών εργαλείων και βάσεων για να καταγραφούν, να αποθηκευτούν και να αναλυθούν».

Μέρος της ανάγκης για νέες τεχνολογίες χειρισμού των big data έχει να κάνει με τη μορφή και όχι αποκλειστικά με το μέγεθος των δεδομένων που εισέρχονται στις online εφαρμογές. Χρειάζεται ένα πιο δυναμικό και ευέλικτο σύστημα, με δυνατότητα να χειρίζεται δομημένα, ημιδομημένα και αδόμητα δεδομένα.

Πολλοί πιστεύουν πως big data είναι κάτι που αφορά μόνο κολοσσούς όπως το Facebook και το Google. Υπάρχουν όμως έρευνες που αποδεικνύουν πως αυτό είναι λάθος. Για παράδειγμα, μία αναφορά της McKinsey βρήκε πως εταιρείες επενδύσεων με λιγότερους από 1000 εργαζόμενους κατά μέσο όρο κρατάνε αποθηκευμένα 3.8 petabytes δεδομένων και αυξάνονται κατά 40% κάθε χρόνο. Επίσης, σε 15 από τους 17 τομείς της βιομηχανίας στις ΗΠΑ τα αποθηκευμένα δεδομένα ανά εταιρεία ξεπερνούν τα 300 terabyte, και όλες οι εταιρείες σε όλους τους τομείς αποθηκεύουν τουλάχιστον 100 terabyte.

Η παρακάτω εικόνα παρουσιάζει αναλυτικά τα παραπάνω στατιστικά στοιχεία.



Εικόνα 1. Στατιστικά στοιχεία χρήσης big data σε όλους τους τομείς

Ο όγκος των δεδομένων αυτών δεν περιορίζεται μόνο σε αποθήκες δεδομένων όπου χρησιμοποιούνται μόνο για σκοπούς εντός της επιχείρησης. Αντιθέτως, υπάρχει και στα συστήματα βάσεων δεδομένων πραγματικού χρόνου, εξυπηρετώντας πρόσωπα εκτός της επιχείρησης, όπως για παράδειγμα πελάτες. Ο όγκος των δεδομένων αυτών συνεχίζει να αυξάνεται όσο η κάθε επιχείρηση γνωρίζει ολοένα και μεγαλύτερη επιτυχία.

Για παράδειγμα, το 2010 μόνο στις ΗΠΑ αποθηκεύτηκαν περισσότερα από 3,500 petabytes νέας πληροφορίας, όπως βλέπουμε και στην παρακάτω εικόνα. **[1]**

Εικόνα 2. Χρήση big data σε όλο τον κόσμο

1.2 Αντικείμενο Διπλωματικής

Στα πλαίσια της παρούσας διπλωματικής θα ασχοληθούμε με το χειρισμό των big data. Ο συνολικός όγκος δεδομένων που έχουμε στη διάθεσή μας είναι λίγο περισσότερο από 1 terabyte και ο μεγαλύτερος πίνακας της βάσης μας φτάνει τις 7.5 δισεκατομμύρια γραμμές.

Τα δεδομένα αυτά μας παραχωρήθηκαν από μία εταιρεία που είναι υπεύθυνη για το λογισμικό παρακολούθησης φωτοβολταϊκών πάρκων. Πιο συγκεκριμένα, τα πάρκα αυτά είναι συνδεδεμένα σε ένα μηχανισμό που με χρήση χρονοσειρών συλλέγει ενεργειακά δεδομένα. Με τον όρο χρονοσειρά, αναφερόμαστε σε μια σειρά από παρατηρήσεις (μετρήσεις τιμών στη συγκεκριμένη περίπτωση) που λαμβάνονται σε συγκεκριμένες χρονικές στιγμές που ισαπέχουν μεταξύ τους. **[2]** Τα δεδομένα που προκύπτουν από αυτή τη διαδικασία αποθηκεύονται στη βάση που μελετάμε.

Το βασικό πρόβλημα είναι ότι, πέρα από τον μεγάλο αριθμό των εισαγόμενων γραμμών, ταυτόχρονα γίνονται και ερωτήσεις πάνω στα δεδομένα, είτε από τους χρήστες-πελάτες είτε από την ίδια την εταιρεία. Εκτός από τη βάση είχαμε στη διάθεσή μας και τα αποτελέσματα του PgFouine, log analyzer της PostgreSQL, όπου επισημαίνονται τα queries που γίνονται πάνω στη βάση, η συχνότητα με την οποία εκτελούνται καθώς και άλλα σχετικά στατιστικά στοιχεία.

Στόχος μας είναι να αναλύσουμε την τρέχουσα λειτουργία του συστήματος, επισημαίνοντας τις αδυναμίες του. Στη συνέχεια θα πειραματιστούμε πάνω σε τεχνικές βελτιστοποίησης (database optimization) πάνω στη βάση, τόσο σε λογικό όσο και σε φυσικό επίπεδο. Από τα πειράματα αυτά θα προκύψει μία λύση με σχεσιακή βάση που θα καθιστά το σύστημα που μελετάμε πιο αποδοτικό. Έπειτα θα μελετήσουμε σε βάθος τις μη σχεσιακές βάσεις, τα χαρακτηριστικά τους, τη λειτουργία τους και τις δυνατότητές τους. Τέλος, θα καταλήξουμε σε μία συνδυαστική λύση, όπου το σχεσιακό με το μη σχεσιακό σύστημα θα συνυπάρχουν.

1.3 Οργάνωση κειμένου

Η παρούσα διπλωματική εργασία αποτελείται από 12 κεφάλαια.

Στο πρώτο κεφάλαιο γίνεται μία εισαγωγή και περιγράφεται το αντικείμενο της διπλωματικής, καθώς και τη οργάνωση του τόμου.

Στο δεύτερο κεφάλαιο κάνουμε μία σύντομη αναφορά στις σχεσιακές βάσεις και αναλύουμε τα βασικά χαρακτηριστικά τους.

Στα κεφάλαια 3 και 4 περιγράφουμε τα δύο εργαλεία που χρησιμοποιήσαμε για τη διαχείριση της βάσης μας, την PostgreSQL και το user interface phpPgAdmin.

Στο κεφάλαιο 5 γίνεται διεξοδική περιγραφή της βάσης που μελετάμε και αναλύονται οι αδυναμίες και τα προβλήματά της.

Στο κεφάλαιο 6 αναφερόμαστε σύντομα στον βέλτιστο αρχικό σχεδιασμό του συστήματος, ενώ στο κεφάλαιο 7 περιγράφουμε αναλυτικά τις τεχνικές βελτιστοποίησης που μπορούν να εφαρμοστούν σε ένα υπάρχον σύστημα. Για πολλές από αυτές τις τεχνικές παρουσιάζονται πειράματα και μετρήσεις που τεκμηριώνουν την θετική τους επίδραση στην απόδοση του συστήματος.

Στο κεφάλαιο 8 κάνουμε μία εισαγωγή στις μη σχεσιακές βάσεις και περιγράφουμε τα χαρακτηριστικά τους, ενώ στο κεφάλαιο 9 εξειδικεύουμε τη μελέτη αυτή στην Cassandra, ένα από τα πιο γνωστά NoSQL συστήματα, το οποίο και χρησιμοποιήσαμε.

Στο κεφάλαιο 10 παρουσιάζονται τα πειράματα που έγιναν μετά την μεταφορά της βάσης μας από την PostgreSQL σε Cassandra.

Στο κεφάλαιο 11 γίνεται ο τελικός σχηματισμός και η σύγκριση των προτεινόμενων λύσεων για βελτιστοποίηση του αρχικού συστήματος και παρουσιάζονται προτάσεις για μελλοντική έρευνα.

Τέλος, η βιβλιογραφία βρίσκεται στο κεφάλαιο 12.

2 *Σχεσιακές Βάσεις*

2.1 Ιστορική Αναδρομή

Η τεχνολογία των υπολογιστών έχει επιφέρει μια μόνιμη αλλαγή στο τρόπο με τον οποίο λειτουργούν οι επιχειρήσεις σε ολόκληρο τον κόσμο. Πληροφορίες που παλαιότερα βρίσκονταν αποθηκευμένες σε συρτάρια μπορούν πλέον να είναι προσβάσιμες άμεσα και ταχύτατα με το πάτημα ενός κουμπιού. Παραγγελίες που δίνονται από πελάτες σε άλλες χώρες μπορούν στιγμιαία να επεξεργαστούν στο εργοστάσιο κατασκευής.

Εκτός από την ανάπτυξη του σχεσιακού μοντέλου βάσης δεδομένων, δύο άλλες τεχνολογίες έχουν οδηγήσει στην ταχύτατη άνοδο των Συστημάτων Βάσεων Δεδομένων Πελάτη/Εξυπηρετητή (client/server datbase systems).

Η πρώτη σημαντική τεχνολογία ήταν ο προσωπικός υπολογιστής (Personal Computer, PC). Φθηνές, ευκολόχρηστες εφαρμογές όπως το Lotus 1-2-3 και το Word Perfect έδωσαν την δυνατότητα σε εργαζόμενους αλλά και χρήστες οικιακών υπολογιστών να δημιουργούν έγγραφα και να επεξεργάζονται δεδομένα εύκολα και με ακρίβεια. Οι χρήστες εξοικειώθηκαν με την συνεχή αναβάθμιση των συστημάτων τους σε ολοένα και πιο προηγμένα συστήματα, λόγω του ταχύτατου ρυθμού αλλαγών αλλά και της πτώσης των τιμών.

Η δεύτερη σημαντική τεχνολογία ήταν τα τοπικά δίκτυα (Local Area Network - LAN) και η ενσωμάτωση τους στα γραφεία των επιχειρήσεων σε όλο τον κόσμο. Παλαιότερα, οι χρήστες ήταν εξοικειωμένοι με συνδέσεις μέσω τερματικού (terminal) στα κεντρικά γραφεία των επιχειρήσεων τα οποία συνήθως ήταν εξοπλισμένα με υπολογιστές mainframe. Καθώς όμως τα έγγραφα των εφαρμογών γραφείου (επεξεργασία κειμένου, λογιστικά φύλλα κλπ) θα μπορούσαν να αποθηκευθούν σε ένα κεντρικό σημείο ώστε να είναι προσβάσιμα από οποιοδήποτε υπολογιστή που ήταν συνδεμένος στο δίκτυο, η χρήση των τοπικών δικτύων διαδόθηκε ραγδαία. Από την στιγμή που η εταιρεία υπολογιστών Apple κυκλοφόρησε τον Macintosh και εισήγαγε ένα νέο γραφικό περιβάλλον χρήσης (Graphical User Interface - GUI), οι προσωπικοί υπολογιστές δεν ήταν πλέον μόνο πανίσχυροι και φθηνοί, αλλά έγιναν και εύκολοι στην χρήση.

Στην διάρκεια αυτής της εποχής των αλλαγών και της προόδου, ένας νέος τύπος συστήματος έκανε την εμφάνισή του. Λεγόταν σύστημα πελάτη/εξυπηρετητή (client/server) επειδή η επεξεργαστική απαίτηση μοιραζόταν πλέον μεταξύ του υπολογιστή-πελάτη και τους

εξυπηρετητή όπου βρίσκονταν το σύστημα βάσης δεδομένων. Η τεχνολογία αυτή ερχόταν να ανατρέψει την επικρατούσα αρχιτεκτονική mainframe, όπου όλη η επεξεργασία γινόταν σε κάποιον κεντρικό υπολογιστή mainframe που χρησιμοποιούνταν μέσω τερματικού (terminal), δηλαδή ενός πληκτρολογίου και μιας οθόνης χαρακτήρων.

Αυτός ο νέος τρόπος λειτουργίας των συστημάτων απαιτεί την χρήση νέων εργαλείων ανάπτυξης. Το περιβάλλον χρήσης είναι πλέον σχεδόν αποκλειστικά γραφικό, μέσω των λειτουργικών συστημάτων MS Windows, IBM OS/2, Apple Macintosh ή X-Window συστήματα UNIX. Με τη χρήση της SQL και μιας σύνδεσης δικτύου, οι εφαρμογές πλέον μπορούν να έχουν πρόσβαση στο σύστημα RDBMS που βρίσκεται εγκατεστημένο σε έναν απομακρυσμένο server. Η ολοένα και αυξανόμενη ισχύς των προσωπικών υπολογιστών επιτρέπει την χρήση δεδομένων τα οποία βρίσκονται καταχωρημένα σε ένα φθηνό, σε σύγκριση με τα mainframe, υπολογιστικό σύστημα. Επιπρόσθετα, με την αντικατάσταση του hardware του server μπορεί να αναβαθμιστεί η απόδοση ολόκληρου του συστήματος client/server χωρίς τα υπάρχει ανάγκη για την παραμικρή αλλαγή στους πελάτες-clients.

Με την εξέλιξη και την ολοένα μεγαλύτερη διάδοση του Internet και των νέων τεχνολογιών που έχουν αναπτυχθεί πάνω και γύρω από αυτό, οι βάσεις δεδομένων παίζουν πλέον κυρίαρχο ρόλο. Η εμφάνιση και η ανάπτυξη του ηλεκτρονικού εμπορίου και οι νέες συνθήκες ανταγωνισμού στην παγκόσμια αγορά δημιουργούν νέα πεδία εφαρμογών για τις βάσεις δεδομένων και αποτελούν κινητήριο μοχλό για την ενσωμάτωση ολοένα και περισσοτέρων δυνατοτήτων στα συστήματα RDBMS. **[3]**

2.2 Τι είναι ένα σύστημα RDBMS

Τα τελευταία χρόνια, τα Συστήματα Διαχείρισης Βάσεων Δεδομένων (DBMS-DataBase Management System) έχουν καθιερωθεί σαν το πρωταρχικό μέσο καταχώρησης δεδομένων για συστήματα πληροφοριών που κυμαίνονται από τα μεγαλύτερα τραπεζικά συστήματα συναλλαγών μέχρι μικροεφαρμογές για συστήματα προσωπικών υπολογιστών. Στην καρδιά των περισσοτέρων σημερινών πληροφοριακών συστημάτων υπάρχει ένα Σύστημα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων (RDBMS-Relational DataBase Management System). Τα συστήματα RDBMS είναι η κινητήρια δύναμη για συστήματα διαχείρισης πληροφοριών εδώ και περισσότερο από μια δεκαετία και εξακολουθούν να εξελίσσονται και να προσφέρουν όλο και πιο προηγμένα συστήματα αποθήκευσης, ανάκτησης και διανομής δεδομένων. Η μετεξέλιξη των συστημάτων αυτών έχει τροφοδοτήσει την ανάπτυξη διαφόρων προηγμένων τεχνολογιών όπως των συστημάτων πελάτη/διακομιστή (client/server), data

warehousing και on-line analytical processing (OLAP), τα οποία αποτελούν τον πυρήνα των σημερινών κορυφαίων πληροφοριακών συστημάτων.

Ας εξετάσουμε τα συστατικά του όρου RDBMS. Database (Βάση Δεδομένων) είναι μια ολοκληρωμένη συλλογή σχετικών μεταξύ τους δεδομένων. Με δεδομένο ένα συγκεκριμένο κομμάτι πληροφορίας, η δομή της Βάσης Δεδομένων επιτρέπει την ταχεία πρόσβαση σε πληροφορίες που συνδέονται με αυτό, όπως π.χ. ένας σπουδαστής και τα επιλεγμένα μαθήματα που παρακολουθεί ή όπως ένας υπάλληλος και οι υφιστάμενοι σ' αυτόν. Relational Database (Σχεσιακή Βάση Δεδομένων) είναι ένας τύπος Βάσης Δεδομένων που βασίζεται στο σχεσιακό μοντέλο, το οποίο περιγράφεται αναλυτικά στη επόμενη ενότητα. Τέλος RDBMS είναι το λογισμικό που διαχειρίζεται την σχεσιακή βάση δεδομένων. Τα συστήματα αυτά υπάρχουν σε πολλές μορφές, που κυμαίνονται από μονοχρηστικά (Single-user) συστήματα για PC μέχρι περίτεχνα, επιχειρησιακά συστήματα. Στο χώρο των RDBMS σήμερα τα πιο γνωστά από τα συστήματα που κυκλοφορούν είναι τα εξής: Oracle Database, DB2, Microsoft SQL Server, Sybase, Informix, MySQL, PostgreSQL. **[4]**

2.3 Το Σχεσιακό Μοντέλο Δεδομένων

Το σχεσιακό μοντέλο είναι σήμερα το βασικό μοντέλο δεδομένων για εμπορικές εφαρμογές επεξεργασίας δεδομένων. Έχει κερδίσει την πρωτεύουσα θέση, εξαιτίας της απλότητάς του, που διευκολύνει τη δουλειά των προγραμματιστών σε σύγκριση με άλλα μοντέλα δεδομένων. Μια σχεσιακή βάση δεδομένων αποτελείται από ένα σύνολο από πίνακες, καθένας εκ των οποίων έχει ένα μοναδικό όνομα. Πίνακας είναι μία δισδιάστατη δομή δεδομένων. Κάθε στήλη του πίνακα αφορά ένα συγκεκριμένο γνώρισμα (attribute). Μία γραμμή ενός πίνακα αντιπροσωπεύει μία σχέση ενός συνόλου από τιμές για τα αντίστοιχα γνωρίσματα. Αφού ένας πίνακας είναι ένα σύνολο από τέτοιες σχέσεις υπάρχει μια στενή σχέση μεταξύ ενός πίνακα και της μαθηματικής ιδέας της σχέσης (relation), από την οποία παίρνει το όνομά του το σχεσιακό μοντέλο δεδομένων. Δεν είναι απαραίτητο για μια σχέση να έχει γραμμές για να θεωρείται σχέση. Ακόμα και αν η σχέση δεν περιέχει δεδομένα, αυτή παραμένει ορισμένη με το σετ των ιδιοτήτων της. **[4]**

2.4 Ο Δωδεκάλογος του Dr.Codd για το Σχεσιακό Μοντέλο Βάσης Δεδομένων

Το Σχεσιακό Μοντέλο Βάσης Δεδομένων στηρίχθηκε σε μια εργασία του Dr. E.F.Codd από το 1970 με τίτλο «Ένα Σχεσιακό Μοντέλο Δεδομένων για Μεγάλες Κοινόχρηστες Τράπεζες Δεδομένων» (A Relational Model of Data for Large Shared Data Banks). Η γλώσσα SQL αναπτύχθηκε για να εξυπηρετήσει το πρότυπα του σχεσιακού μοντέλου βάσης δεδομένων.

Οι 13 κανόνες, που κατά περίεργο τρόπο ονομάζονται «Δωδεκάλογος του Codd», λένε τα εξής :

0. Ένα σύστημα για να χαρακτηρίζεται ως RDBMS πρέπει να είναι ικανό να διαχειρίζεται βάσεις δεδομένων αποκλειστικά μέσω των σχεσιακών δυνατοτήτων του.

1. Ο κανόνας της Πληροφορίας : Όλες οι πληροφορίες σε μια σχεσιακή βάση δεδομένων (συμπεριλαμβανομένων των ονομάτων των πινάκων και των στηλών τους) θα πρέπει να αναπαριστώνται άμεσα σαν τιμές σε πίνακες.

2. Εγγυημένη Πρόσβαση : Κάθε τιμή σε μια σχεσιακή βάση δεδομένων είναι εγγυημένο ότι θα είναι προσβάσιμη με τη χρήση του ονόματος του πίνακα, της τιμής του πρωτεύοντος κλειδιού και του ονόματος της στήλης.

3. Υποστήριξη της τιμής NULL : Το DBMS θα πρέπει να προσφέρει συστηματική υποστήριξη για τον χειρισμό των τιμών NULL ανεξάρτητα από τις προκαθορισμένες τιμές (default values) ή τους τύπους δεδομένων (data types).

4. Ενεργός, online σχεσιακός κατάλογος : Η περιγραφή της βάσης δεδομένων και των περιεχομένων της θα αναπαριστάται σε λογικό επίπεδο σαν πίνακες στους οποίους θα μπορούν να υποβληθούν ερωτήματα μέσω της γλώσσας της βάσης δεδομένων.

5. Περιεκτική γλώσσα δεδομένων : Τουλάχιστον μία από τις υποστηριζόμενες γλώσσες θα πρέπει να έχει αυστηρά καθορισμένη μορφή και να είναι περιεκτική. Θα πρέπει να υποστηρίζει Ορισμό Δεδομένων (Data Definition), χειρισμό δεδομένων (Data Manipulation), κανόνες ακεραιότητας (Interity Rules), εξουσιοδότηση (Authorization) και Συναλλαγές (Transactions).

6. Κανόνας ενημέρωσης όψεων : Όλες οι όψεις (Views) που είναι θεωρητικά ενημερώσιμες θα πρέπει να μπορούν να ενημερωθούν μέσω του συστήματος.

7. Εισαγωγή, ενημέρωση, διαγραφή σε επίπεδο set : Το DBMS θα πρέπει να υποστηρίζει όχι μόνο ανάκτηση σε επίπεδο set αλλά και εισαγωγή, ενημέρωση και διαγραφή.

8. Ανεξαρτησία των φυσικών δεδομένων : Η εφαρμογή δεν θα πρέπει να επηρεάζεται σε λογικό επίπεδο όταν γίνονται μεταβολές στις δομές αποθήκευσης.

9. Ανεξαρτησία των λογικών δεδομένων : Τα προγράμματα εφαρμογών δεν θα πρέπει να επηρεάζονται όταν γίνονται μεταβολές σε λογικό επίπεδο π.χ. σε πίνακες, στήλες, σειρές κλπ.

10. Ανεξαρτησία της ακεραιότητας : Η γλώσσα της βάσης δεδομένων θα πρέπει να είναι ικανή για τον καθορισμό των κανόνων ακεραιότητας. Οι κανόνες αυτοί θα πρέπει να βρίσκονται καταχωρημένοι στον κατάλογο της βάσης δεδομένων και δεν θα μπορούν να παρακαμφθούν.

11. Ανεξαρτησία της διανομής : Τα προγράμματα εφαρμογών δεν θα πρέπει να επηρεάζονται σε λογικό επίπεδο όταν τα δεδομένα διανέμονται για πρώτη φορά ή όταν επαναδιανέμονται.

12. Μη αντιστρεψιμότητα : Δεν θα πρέπει να είναι δυνατή η παράκαμψη των κανόνων ακεραιότητας που καθορίζονται από την γλώσσα της βάσης δεδομένων με την χρήση γλωσσών χαμηλότερου επιπέδου. **[5]**

2.5 ACID

Στην επιστήμη των υπολογιστών, το ACID (Atomicity/Ατομικότητα, Consistency/Συνέπεια, Isolation/Απομόνωση, Durability/Μονιμότητα) είναι ένα σύνολο ιδιοτήτων το οποίο εγγυάται ότι η συναλλαγές στην βάση δεδομένων λειτουργούν αξιόπιστα. Πιο αναλυτικά οι ιδιότητες αυτές εξασφαλίζουν ότι :

Ατομικότητα: είτε όλες οι πράξεις της δοσοληψίας επιτυγχάνουν, είτε όλες αποτυγχάνουν.

Συνέπεια: στο τέλος της δοσοληψίας, η βάση πρέπει να είναι σε συνεπή μορφή.

Απομόνωση: ακόμα κι αν τρέχουν πολλές δοσοληψίες ταυτόχρονα, κάθε δοσοληψία πρέπει να νομίζει ότι τρέχει μόνη της.

Μονιμότητα: αν η δοσοληψία επιτύχει, πρέπει το αποτέλεσμά της να επιβιώνει, ακόμα κι αν αποτύχει το σύστημα.

Το ACID είναι στην ουσία ένα σύστημα αξιολόγησης των βάσεων δεδομένων και των εφαρμογών που συνδέονται με αυτές. **[6]**

2.6 Κλειδιά και αναφορική ακεραιότητα

Οι ιδιότητες ομαδοποιούνται με βάση την εξάρτησή τους από ένα πρωτεύον κλειδί (primary key). Πρωτεύον κλειδί είναι μια ιδιότητα (ή μια ομάδα ιδιοτήτων) η οποία ταυτοποιεί μια γραμμή σε έναν πίνακα. Ένας πίνακας μπορεί να έχει το πολύ ένα πρωτεύον κλειδί. Επειδή οι

τιμές των κλειδιών αυτών χρησιμοποιούνται για την ταυτοποίηση της γραμμής δεν μπορούν να περιέχουν τιμή NULL.

Μπορεί να υπάρχουν και άλλες ιδιότητες σε μία σχέση των οποίων οι τιμές πρέπει να είναι επίσης μοναδικές. Σε αντίθεση όμως με το πρωτεύον κλειδί αυτές μπορούν να περιέχουν τιμή NULL. Στην πράξη, οι ιδιότητες αυτές, που ονομάζονται "μοναδικά κλειδιά" (unique keys) χρησιμοποιούνται για να αποκλείσουμε την πιθανότητα καταχώρησης δύο γραμμών με την ίδια τιμή για την ίδια ιδιότητα, και όχι για να ταυτ2οποιήσουμε την γραμμή.

Η σύνδεση μιας σχέσης με μια δεύτερη συνήθως απαιτεί μια κοινή ιδιότητα, η οποία υπάρχει και στις δύο σχέσεις. Οι κοινές ιδιότητες αυτές είναι συνήθως ένα πρωτεύον κλειδί (primary key) της μίας σχέσης και ένα ξένο κλειδί (foreign key) της δεύτερης. Ο κανόνας της αναφορικής ακεραιότητας (referential integrity) επιβάλλει ότι τιμές για ένα ξένο κλειδί της δεύτερης σχέσης αναφέρονται σε τιμές του πρωτεύοντος κλειδιού της πρώτης. **[4]**

2.7 Σχεσιακή Άλγεβρα

Το σχεσιακό μοντέλο ορίζει τις εργασίες που επιτρέπονται σε μία σχέση ή σε μια ομάδα σχέσεων. Υπάρχουν ατομικοί (unary) και δυαδικοί (binary) τελεστές, καθένας από τους οποίους δίνει σαν αποτέλεσμα μια άλλη σχέση. Ο παρακάτω πίνακας περιγράφει τους επτά τελεστές που χρησιμοποιούνται για τον χειρισμό των σχεσιακών δομών. Οι δυαδικοί τελεστές απαιτούν δύο σχέσεις για να λειτουργήσουν ενώ οι ατομικοί μία. **[4]**

Εργασία	Τύπος	Σχέση που προκύπτει
Union (Ένωση)	Binary	Επιστρέφεται συνδυασμός από τις δύο σχέσεις, αφού αποκλειστούν οι διπλές σειρές.
Intersection (Τομή)	Binary	Επιστρέφονται οι κοινές σειρές των δύο σχέσεων
Difference (Διαφορά)	Binary	Επιστρέφονται σειρές που υπάρχουν στην πρώτη σχέση αλλά όχι στην δεύτερη.
Projection (Προβολή)	Unary	Επιστρέφονται γραμμές που περιέχουν ορισμένες από τις ιδιότητες της αρχικής σχέσεις.
Selection (Επιλογή)	Unary	Επιστρέφονται σειρές από την αρχική σχέση οι οποίες ικανοποιούν ορισμένα κριτήρια επιλογής
Product (Γινόμενο)	Binary	Επιστρέφεται ακολουθία ιδιοτήτων από κάθε γραμμή της πρώτης σχέσης με κάθε γραμμή της δεύτερης σχέσης
Join (Σύνδεση)	Binary	Επιστρέφεται ακολουθία ιδιοτήτων γραμμών από την πρώτη σχέση με συνδυασμένες γραμμές από την δεύτερη σχέση.
Composition (Σύνθεση)	Binary	Συνδέει τις λειτουργίες σχεσιακής άλγεβρας σχηματίζοντας μία παράσταση σχεσιακής άλγεβρας
Rename (Μετονομασία)	Unary	Δίνει όνομα στα αποτελέσματα των παραστάσεων σχεσιακής άλγεβρας
Assignment (Εκχώρηση)	Unary	Εκχωρεί μέρη μιας παράστασης σχεσιακής άλγεβρας σε προσωρινές μεταβλητές.

Πίνακας 1. Τελεστές σχεσιακών δομών

3 *PostgreSQL*

3.1 Τι είναι η PostgreSQL

Η PostgreSQL αποτελεί μια ανοιχτού κώδικα σχεσιακή βάση δεδομένων με πολλές δυνατότητες. Η ανάπτυξη της ήδη διαρκεί πάνω από 20 χρόνια και βασίζεται σε μια αποδεδειγμένα καλή αρχιτεκτονική η οποία έχει δημιουργήσει μια ισχυρή αντίληψη των χρηστών της γύρω από την αξιοπιστία, την ακεραιότητα δεδομένων και την ορθή λειτουργία. Η PostgreSQL τρέχει σε όλα τα βασικά λειτουργικά συστήματα, περιλαμβάνοντας Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), και Windows. Είναι ACID συμβατή (ACID compliant), έχει ολοκληρωμένη υποστήριξη για foreign keys, joins, views, triggers, και stored procedures (σε διάφορες γλώσσες προγραμματισμού). Συμπεριλαμβάνει τα περισσότερα SQL92 και SQL99 data types, συμπεριλαμβανομένων INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, και TIMESTAMP. επίσης υποστηρίζει αποθήκευση binary large objects, όπως εικόνες, ήχοι ή video. Διαθέτει native programming interfaces για C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, κ.α. καθως και εξαιρετικό εγχειρίδιο χρήσης. Επίσης, ο πηγαίος κώδικας της PostgreSQL είναι διαθέσιμος κάτω από την πιο ελεύθερη open source άδεια: το BSD license. Αυτή η άδεια δίνει την δυνατότητα χρήσης, μετατροπής και διανομής της PostgreSQL σε οποιαδήποτε μορφή, ανοιχτού ή κλειστού κώδικα. Η PostgreSQL δεν είναι μόνο μια δυνατή βάση δεδομένων ικανή να τρέχει μέσα σε επιχειρήσεις, είναι μια πλατφόρμα ανάπτυξης πάνω στην οποία δύναται να γίνει ανάπτυξη in-house, web ή εμπορικών εφαρμογών τα οποία χρειάζονται RDBMS. **[7], [8]**

3.2 Σύντομη ιστορική αναδρομή

3.2.1 Η Εργασία POSTGRES του πανεπιστημίου BERKELEY

Η PostgreSQL προέρχεται από το πακέτο POSTGRES, το οποίο γράφτηκε στο Πανεπιστήμιο του Berkeley στην Καλιφόρνια των Η.Π.Α..Το σύνθετο πρόγραμμα της POSTGRES, που πραγματοποιήθηκε υπό τη βασική καθοδήγηση του καθηγητή Michael Stonebraker, χρηματοδοτήθηκε από την προηγμένη αντιπροσωπεία ερευνητικών προγραμμάτων

άμυνας(DARPA), το ερευνητικό γραφείο στρατού (ARO), το εθνικό ίδρυμα επιστήμης (NSF), αλλά και από τα ESL και INC. Η εφαρμογή POSTGRES άρχισε το 1986. Οι πρώτες ιδέες για το σύστημα παρουσιάστηκαν στο σχέδιο POSTGRES και ο καθορισμός του αρχικού δεδομενικού μοντέλου εμφανίστηκε στο μοντέλο δεδομένων POSTGRES. Το σχέδιο των κανόνων του συστήματος εκείνη την περίοδο, περιγράφηκε στο «Σχέδιο του συστήματος κανόνων POSTGRES». Η λογική και η αρχιτεκτονική του διευθυντή(manager) αποθήκευσης περιγράφονται λεπτομερώς στο «σχέδιο του συστήματος αποθήκευσης της POSTGRES». Η POSTGRES έχει υποβληθεί σε διάφορες σημαντικές επεκτάσεις από τότε. Το πρώτο «demoware» σύστημα κατέστη λειτουργικό το 1987 και παρουσιάστηκε στη διάσκεψη ACM-SIGMOD του 1988. Η έκδοση 1, που περιγράφηκε στην υλοποίηση της POSTGRES, κυκλοφόρησε σε μερικούς εξωτερικούς χρήστες τον Ιούνιο του 1989.

Ανταποκρινόμενο σε μια αρνητική κριτική του αρχικού συστήματος κανόνων, το σύστημα κανόνων της POSTGRES ξανασχεδιάστηκε σε ότι αφορά στους κανόνες, τις διαδικασίες, την εναποθήκευση και τις όψεις στα συστήματα βάσεων δεδομένων και η δεύτερη έκδοσή της κυκλοφόρησε τον Ιούνιο του 1990 με το νέο σύστημα κανόνων. Στην έκδοση 3 της POSTGRES, που πραγματοποιήθηκε το 1991, προστέθηκε η υποστήριξη πολλαπλών διευθυντών αποθήκευσης. Επιπροσθέτως, διαθέτει τώρα και ένα βελτιωμένο εκτελεστή επερωτήσεων, και ένα νέο σύστημα επανεγγράψιμων κανόνων. Ως επί το πλείστον, οι επόμενες εκδόσεις μέχρι και την Postgres95 εστίασαν στη μεταφερσιμότητα και την αξιοπιστία.

Η POSTGRES έχει χρησιμοποιηθεί για να υλοποιήσει πολλές διαφορετικές εφαρμογές έρευνας και παραγωγής. Σε αυτές περιλαμβάνονται: ένα οικονομικό σύστημα ανάλυσης δεδομένων, μια συσκευασία ελέγχου απόδοσης αεριωθούμενων μηχανών, μια ιατρική βάση δεδομένων πληροφοριών και διάφορα γεωγραφικά συστήματα πληροφοριών. Η POSTGRES έχει χρησιμοποιηθεί επίσης ως εκπαιδευτικό εργαλείο σε διάφορα πανεπιστήμια. Εν κατακλείδι, τεχνολογίες πληροφοριών Illustra (που συγχωνεύονται αργότερα σε Informix, το οποίο είναι κύριο τώρα από την IBM) πήραν τον κώδικα και τον εμπορευματοποίησαν. Στα τέλη του 1992, η POSTGRES αποτέλεσε τον πρωταρχικό διαχειριστή (manager) δεδομένων για το sequoia2000, ένα επιστημονικό πρόγραμμα υπολογισμού. Το μέγεθος της εξωτερικής κοινότητας χρηστών διπλασιάστηκε σχεδόν κατά τη διάρκεια του 1993. Έγινε όλο και περισσότερο προφανές ότι η συντήρηση του πρωτότυπου κώδικα και η υποστήριξή του απαιτεί μεγάλο χρονικό διάστημα, που θα έπρεπε να έχει αφιερωθεί στην έρευνα βάσεων δεδομένων. Σε μια προσπάθεια να μειωθεί αυτό το βάρος της υποστήριξης, το πρόγραμμα του Berkeley, POSTGRES, τελείωσε επίσημα με την έκδοση 4.2. **[7], [8]**

3.2.2 Postgres95

Το 1994, ο Andrew Yu και ο Jolly Chen πρόσθεσαν τον γλωσσικό διερμηνέα της SQL στην POSTGRES. Με ένα νέο όνομα, η Postgres95 απελευθερώθηκε στη συνέχεια στον Ιστό, ως απόγονος του αρχικού κώδικα POSTGRES. Στην Postgres95 ο κώδικας ήταν όλος σε Ansi C και μικρότερος στο μέγεθος κατά 25%. Πολλές εσωτερικές αλλαγές βελτίωσαν την απόδοση και τη συντηρησιμότητα του νέου αυτού λογισμικού. Η απελευθέρωση 1.0.x Postgres95 έτρεξε περίπου 30-50% γρηγορότερα στη συγκριτική μέτρηση επιδόσεων του Wisconsin σε σύγκριση με την έκδοση 4.2 της POSTGRES. Εκτός από κάποιες μικρές διορθώσεις, τα εξής ήταν οι σημαντικότερες επεκτάσεις:

- Η γλώσσα επερωτήσεων PostQUEL αντικαταστάθηκε από την SQL, που ενσωματώνεται στον εξυπηρετητή(server). Επερωτήσεις δεν υποστηρίχθηκαν μέχρι την εμφάνιση της PostgreSQL (που θα αναλυθεί παρακάτω), αλλά θα μπορούσαν να υλοποιηθούν στην Postgres95 με τις καθορισμένες από το χρήστη συναρτήσεις SQL. Οι αθροιστικές συναρτήσεις υλοποιήθηκαν από την αρχή. Επίσης προστέθηκε και το στοιχείο GROUP BY της εντολής SELECT των επερωτήσεων.
- Εκτός από το πρόγραμμα παρατηρητή (monitor), ένα νέο πρόγραμμα (psql) παρείχε τη δυνατότητα πραγματοποίησης διαλογικών επερωτήσεων SQL, οι οποίες χρησιμοποίησαν το GNU Readline.
- Μια νέα front-end βιβλιοθήκη (libpgtcl) καθιστούσε δυνατή την υποστήριξη πελατών Tcl. Ένας αντιπροσωπευτικός φλοιός, pgtclsh, παρείχε νέες εντολές Tcl για να διασυνδέσει τα προγράμματα Tcl με τον εξυπηρετητή της Postgres95.
- Η διεπαφή μεγάλο-αντικείμενο(large-object) εξετάστηκε λεπτομερώς. Τα μεγάλα αντικείμενα αντιστροφής ήταν ο μόνος μηχανισμός για την αποθήκευση μεγάλων αντικειμένων. Το σύστημα αρχείων αντιστροφής αφαιρέθηκε αργότερα.
- Το σε επίπεδο στιγμιότυπων σύστημα κανόνων αφαιρέθηκε. Κανόνες εξακολούθησαν να είναι διαθέσιμοι με τη μορφή επανεγγράψιμων κανόνων.
- Ένα σύντομο διδακτικό βοήθημα που κάνει μια εισαγωγή στα συνηθισμένα χαρακτηριστικά γνωρίσματα της SQL, καθώς επίσης και σε όλα εκείνα τα γνωρίσματα της Postgres95 που διανεμήθηκαν με τον πηγαίο κώδικα.
- GNU make (αντί του BSD make) χρησιμοποιήθηκε για το κτίσιμο (build). Επίσης, η Postgres95 θα μπορούσε να μεταγλωττιστεί με το GCC. **[7], [8]**

3.2.3 PostgreSQL

Το 1996 έγινε σαφές ότι η ονομασία Postgres95 δεν θα αντέξει με την πάροδο του χρόνου, γιαυτό και επιλέχθηκε η ονομασία PostgreSQL ώστε να αντικατοπτρίσει την σχέση ανάμεσα στην αρχική POSTGRES και τις πιο πρόσφατες εκδόσεις με συμβατότητα SQL. Την ίδια χρονιά ορίστηκε η αρίθμηση των εκδόσεων να ξεκινά από το νούμερο 6.0, έτσι ώστε να συνεχίζεται από την αρίθμηση που ξεκίνησε από το αρχικό σχέδιο POSTGRES του πανεπιστημίου Berkeley. Η έμφαση κατά τη διάρκεια της ανάπτυξης των εκδόσεων v1.0.x της Postgres95 ήταν στην σταθεροποίηση του backend. Με τη σειρά εκδόσεων v6.x της PostgreSQL η έμφαση μεταφέρθηκε από τον προσδιορισμό και την κατανόηση των υπαρχόντων προβλημάτων στο backend, στην προσπάθεια αύξησης των δυνατοτήτων. **[7], [8]**

3.3 Γενικά χαρακτηριστικά

Η PostgreSQL υλοποιεί εξεζητημένα χαρακτηριστικά όπως Multi-Version Concurrency Control (MVCC), point in time recovery, tablespaces, asynchronous replication, nested transactions (savepoints), online/hot backups, a sophisticated query planner/optimizer, write ahead logging for fault tolerance. Υποστηρίζει διεθνή σετ χαρακτήρων, κωδικοποίηση χαρακτήρων σε πολλά byte, Unicode καθώς και δυνατότητα ταξινόμησης δεδομένων ανεξάρτητα από το locale. Η PostgreSQL μπορεί να διαχειριστεί εύκολα μεγάλους αριθμούς ταυτόχρονων χρηστών καθώς και μεγάλο όγκο δεδομένων. Υπάρχουν ενεργές εγκαταστάσεις σε περιβάλλοντα παραγωγής που διαχειρίζονται πάνω από 4 terabytes δεδομένων.

Μερικές γενικές οριακές τιμές συμπεριλαμβάνονται στον παρακάτω πίνακα:

Limit	Value
Maximum Database Size	Unlimited
Maximum Table Size	32 TB
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows per Table	Unlimited
Maximum Columns per Table	250 - 1600 depending on column types
Maximum Indexes per Table	Unlimited

Πίνακας 2. Οριακές τιμές της PostgreSQL

Η PostgreSQL διαθέτει μια ευρεία ποικιλία τύπων δεδομένων. Ο χρήστης έχει την δυνατότητα να προσθέσει ένα νέο τύπο δεδομένων χρησιμοποιώντας την εντολή CREATE TYPE.

Κατηγορίες	Τύποι
Λογικοί και δυαδικοί (Boolean and binary types)	boolean, bool, bit(n), bit varying(n), varbit(n)
Χαρακτήρες (Character types)	character (n), char(n), character varying(n), varchar(n), text
Αριθμητικοί (Numeric types)	smallint, int2, integer, int, int4, bigint, int8, real, float4, double precision, float8, float, numeric(p,s), decimal(p,s), money, serial
Ημερομηνία και ώρα (Date and time types)	date, time, time with time zone, timestamp (includes time zone), interval
Γεωμετρικοί (Geometric types)	box, line, lseg, circle, path, point, polygon
Δικτυακοί (Network types)	cidr, inet, macaddr
Συστήματος (System types)	oid, xid

Πίνακας 3. Τύποι στην PostgreSQL

Η PostgreSQL απολαμβάνει αναγνώριση από τους χρήστες της και την βιομηχανία πληροφορικής, συμπεριλαμβανομένων των Linux New Media Award for Best Database System, και έχει υπάρξει 3 φορές νικήτρια στο Linux Journal Editors' Choice Award for best DBMS.

Η PostgreSQL είναι συνεπής με τις προδιαγραφές . Η υλοποίησή της είναι απολύτως σύμφωνη με τις προδιαγραφές ANSI-SQL 92/99. Έχει ολοκληρωμένη υποστήριξη για subqueries (συμπεριλαμβανομένων subselects μέσα από το FROM), read-committed και serializable transaction isolation levels. Η PostgreSQL αποτελεί ένα πλήρες σχεσιακό σύστημα που υποστηρίζει πολλαπλά σχήματα ανά database, ο κατάλογος (πληροφορίες σχετικά με τους πίνακες, στήλες, views κλπ) είναι διαθέσιμος διαμέσου του Information Schema όπως ορίζεται στο SQL standard.

Στα data integrity χαρακτηριστικά συμπεριλαμβάνονται: primary keys, foreign keys με υποστήριξη restricting και cascading updates/deletes, check constraints, unique constraints, και not null constraints.

Η PostgreSQL έχει αρκετά προηγμένα χαρακτηριστικά όπως: auto-increment columns μέσω sequences, LIMIT/OFFSET που επιτρέπουν την επιστροφή partial result sets. Όσον αφορά τα indexes υποστηρίζει compound, unique, partial, και functional indexes τα οποία μπορούν να χρησιμοποιήσουν οποιονδήποτε από τους B-tree, R-tree, hash, ή GiST αλγόριθμους.

Άλλα προηγμένα χαρακτηριστικά της PostgreSQL είναι: table inheritance, rules systems και database events .

Το **table inheritance** (κληρονομικότητα πινάκων) προσθέτει μια αντικειμενοστραφή διάσταση στην δημιουργία πινάκων, επιτρέποντας στους σχεδιαστές database να δημιουργούν νέους πίνακες από άλλους πίνακες χρησιμοποιώντας τους ως βάση. Ακόμα καλύτερα η PostgreSQL υποστηρίζει και μονή και πολλαπλή κληρονομικότητα με τον δικό της τρόπο.

Το **rules system**, που επίσης καλείται the query rewrite system, επιτρέπει στον σχεδιαστή βάσεων να δημιουργήσει κανόνες που ορίζουν συγκεκριμένες λειτουργίες για έναν πίνακα ή view, και να μετατρέπει δυναμικά λειτουργίες, την ώρα που εκτελούνται, σε άλλες εναλλακτικές.

Το **events system** αποτελεί ένα interprocess communication system στο οποίο μηνύματα και events μπορούν να μεταδοθούν μεταξύ πελατών (clients) χρησιμοποιώντας τις LISTEN και NOTIFY εντολές, επιτρέποντας από την απλή peer to peer επικοινωνία ως ένα εξελιγμένο συντονισμό βασισμένο σε database events. Εφόσον τα notifications μπορεί να προέρχονται από triggers και stored procedures, PostgreSQL clients μπορούν να επιβλέπουν λειτουργίες όπως: updates, inserts ή deletes πινάκων όταν αυτά γίνονται. **[7], [8]**

3.4 Συμβατότητα

Η PostgreSQL τρέχει stored procedures σε πολλές γλώσσες προγραμματισμού συμπεριλαμβανομένων Java, Perl, Python, Ruby, Tcl, C/C++, και της PL/pgSQL η οποία είναι παρόμοια με την PL/SQL της Oracle. Στην βασική βιβλιοθήκη συναρτήσεων της PostgreSQL συμπεριλαμβάνονται εκατοντάδες built-in συναρτήσεις οι οποίες καλύπτουν από βασικές μαθηματικές συναρτήσεις και διαχείριση Συμβολοσειρών ως κρυπτογραφία και Oracle compatibility. Triggers και stored procedures μπορούν να γράφουν σε c και να φορτωθούν μέσα στη βάση ως βιβλιοθήκη, επιτρέποντας μεγάλη ευελιξία στην επέκταση των δυνατοτήτων της βάσης. Παρομοίως η PostgreSQL περιλαμβάνει framework που επιτρέπει τον ορισμό και την δημιουργία custom data types καθώς και βοηθητικές συναρτήσεις και τελεστές (operators) που θα περιγράφουν την λειτουργία τους. Σαν αποτέλεσμα ένα πλήθος από εξελιγμένα data types έχουν δημιουργηθεί από γεωμετρικά και spatial δεδομένα ως διευθύνσεις δικτύων και ISBN/ISSN (International Standard Book Number/International Standard Serial Number), τα οποία μπορούν κατ' επιλογή να προστεθούν στο σύστημα.

Η PostgreSQL, όπως διαθέτει πολλές procedure languages, έτσι διαθέτει και πολλά library interfaces , επιτρέποντας πολλές γλώσσες προγραμματισμού είτε compiled είτε interpreted να επικοινωνούν με την PostgreSQL. Υπάρχουν interfaces για Java (JDBC), ODBC, Perl, Python, Ruby, C, C++, PHP, Lisp, Scheme, Qt, κ.α. **[7], [8]**

4 *phpPgAdmin*

Ως user interface της PostgreSQL χρησιμοποιήσαμε την διαδικτυακή εφαρμογή phpPgAdmin στην τελευταία της έκδοση 5.1.

4.1 Χαρακτηριστικά

- Ταυτόχρονη διαχείριση πολλών servers
- Συμβατότητα με τις εκδόσεις 8.4.x, 9.0.x, 9.1.x, 9.2.x της PostgreSQL
- Παροχή όλων των δυνατοτήτων σχετικά με:
    - Users & groups
    - Databases
    - Schemas
    - Tables, indexes, constraints, triggers, rules & privileges
    - Views, sequences & functions
    - Advanced objects
    - Reports
- Εύκολη διαχείριση δεδομένων:
    - Δυνατότητα για εμφάνιση όλων των δεδομένων(browse) σε tables, views & reports
    - Δυνατότητα απευθείας εκτέλεσης εντολών SQL select, insert, update, delete καθώς και SQL scripts.
- Εξαγωγή αρχείο ανάκτησης δεδομένων σε πολλαπλές μορφές : SQL, COPY, XML, XHTML, CSV, Tabbed, pg_dump
- Εισαγωγή δεδομένων επίσης με πολλαπλή συμβατότητα, μέσω SQL scripts, COPY data, XML, CSV, Tabbed
- Slony master-slave replication
- Διάθεση σε 27 γλώσσες χωρίς προβλήματα κωδικοποίησης και συμβατότητας μεταξύ δεδομένων και interface
- Εύκολη εγκατάσταση και εκμάθηση. **[9]**

5 *Πρόβλημα*

Στην ενότητα αυτή θα περιγράψουμε τη δομή και τη λειτουργία της βάσης που διαχειριζόμαστε, εστιάζοντας στις αδυναμίες της, τις οποίες θα προσπαθήσουμε να επιλύσουμε στη συνέχεια της διπλωματικής.

5.1 Ο πίνακας measurement_events

Η βάση μας αποτελείται από 64 πίνακες και έχει μέγεθος περίπου 1TB. Από τους πίνακες αυτούς ένας είναι αυτός που περιέχει την πλειοψηφία των δεδομένων, περίπου 7 δισεκατομμύρια γραμμές. Για το λόγο αυτό παρακάτω παρατίθεται αναλυτική περιγραφή των χαρακτηριστικών του.

Στήλη	Τύπος	Κλειδιά	NOT NULL
measurement_location_id	uuid		√
measurement_source_id	uuid	primary key	√
measurement_time	timestamp with time zone	primary key	√
event_reception_time	timestamp with time zone		√
measurement_value	character varying(200)		√
quality	character varying(500)		√
source_id	uuid		√
transducer_id	uuid		

Πίνακας 4. Χαρακτηριστικά του πίνακα measurement_events

Υπάρχουν 4 στήλες (measurement_source_id, measurement_location_id, source_id και transducer_id) που έχουν τύπο uuid, δηλαδή Universally Unique Identifiers. Είναι δηλαδή identifiers μεγέθους 128-bit που παράγονται από κάποιον αλγόριθμο έτσι ώστε να είναι σχεδόν αδύνατο να γίνει generate ίδιο από κάποιον άλλο, χρησιμοποιώντας τον αλγόριθμο αυτό. Άρα, για κατανεμημένα συστήματα (distributed systems), εγγυώνται καλύτερη μοναδικότητα από sequence generators (τα οποία θα είναι unique μόνο σε μία βάση). Η μορφή τους είναι μία σειρά από δεκαεξαδικά ψηφία, σε group των 8,4,4,4,12 ψηφίων χωρισμένα με παύλες. Αυτό έχει σαν σύνολο 32 ψηφία ή 128 bits. Υπάρχουν 2 στήλες (measurement_time, event_reception_time) τύπου timestamp with time zone, δηλαδή περιλαμβάνουν ημερομηνία, ώρα και ζώνη ώρας (με βάση το GMT). Όταν μία τέτοια ημερομηνία ζητείται να εμφανιστεί, γίνεται πάντα μετατροπή της από ώρα GMT στην τοπική ζώνη ώρας και εμφανίζεται ως τοπική ώρα. Έχει μέγεθος 8 bytes.

Υπάρχουν 2 στήλες (measurement_value, quality)τύπου character varying(n), που είναι στην ουσία το γνωστό varchar(n). Στις στήλες αυτές αποθηκεύονται κάποιες μετρήσεις σε επιστημονική μορφή (η συγκεκριμένη έκδοση PostgreSQL 8.4 που χρησιμοποιεί η εταιρεία, σε αντίθεση με τις νεότερες, δεν έχει ξεχωριστό τύπο γι αυτόν το σκοπό). Η μία στήλη λέει character varying(200) και η άλλη character varying(500).

Ως πρωτεύον κλειδί (primary key) της βάσης ορίζεται ο συνδυασμός των στηλών measurement_location_id και measurement_time, γεγονός το οποίο αποτελεί και τη μόνη λύση για μοναδικότητα. Ο τύπος του index είναι B-δέντρο.

Ένα παράδειγμα μιας εγγραφής στον πίνακα measurement_events φαίνεται παρακάτω :

measurement_location_id	measurement_source_id
9955273a-dc30-11e0-b12c-03163e944340	9c7af320-dc30-11c0-b07c-00163e944340

measurement_time	event_reception_time	measurement_value
2011-03-07 13:00:00+02	2011-03-08 01:31:20+02	244169616e+01

quality	source_id	transducer_id
100	5938360e-a1e1-11de-34e6-0013bbc9165c	NULL

Πίνακας 5. Παράδειγμα εγγραφής του measurement_events

5.2 Ο πίνακας table_X

Ο πίνακας table_X που θα αναφερθεί σε αρκετές συναρτήσεις στις παρακάτω ενότητες είναι ένας βοηθητικός πίνακας που δημιουργήσαμε εμείς. Σε αυτή την ενότητα θα αναλύσουμε τις ιδιότητές του και τις λειτουργίες που εξυπηρετεί η χρήση του.

Στήλη	Τύπος	Κλειδιά	NOT NULL
measurement_location_id	uuid		√
measurement_source_id	uuid	index	√
measurement_time	timestamp with time zone	index	√
event_reception_time	timestamp with time zone		√
measurement_value	character varying(200)		√
quality	character varying(500)		√
source_id	uuid		√
transducer_id	uuid		
aa	bigint	primary key	√

Πίνακας 6. Χαρακτηριστικά του βοηθητικού πίνακα table_X

Όπως αναφέραμε στην εισαγωγή, το σύστημα το οποίο εξετάζουμε είναι μία βάση δεδομένων η οποία τροφοδοτείται συνέχεια με νέα δεδομένα με αυτοματοποιημένο τρόπο. Ωστόσο, εμείς

έπρεπε να προσομοιώσουμε αυτή τη διαδικασία με χρήση βοηθητικών πινάκων οι οποίοι θα λειτουργούσαν ως πηγές δεδομένων. Ένα εμφανές μειονέκτημα της διαδικασίας αυτής ήταν η καθυστέρηση για εισαγωγή δεδομένων από τον πίνακα πηγή στον πίνακα προορισμού. Έπρεπε να επιλεχθούν κατάλληλα φίλτρα τα οποία θα εξασφάλιζαν μοναδικότητα των δεδομένων για αποφυγή λαθών για unique keys. Προφανώς μια απλή εντολή

```
INSERT INTO tableA
SELECT *
FROM tableB LIMIT N;
```

δεν εγγυάται καμία μοναδικότητα δεδομένων. Επίσης δεν ήταν δυνατή η χρήση κάποιου where-clause με χρήση του measurement_source_id λόγω του τύπου uuid που δεν συμβατός με κάποιο τελεστή διάταξης (<= , >=) που να επιστρέφει κάποιο συγκεκριμένο διάστημα τιμών. Επίσης, στο online σύστημα γνωρίζουμε ότι κάθε γραμμή δεδομένων που εισέρχεται στο σύστημα είναι πιο πρόσφατη χρονικά από κάθε καταχώρηση που ήδη υπάρχει και μάλιστα αναφέρεται σε μέτρηση που έγινε σε παροντική στιγμή. Το γεγονός αυτό, το συναντήσαμε ως πρόβλημα στην κατασκευή και διαχείριση των ενδιάμεσων πινάκων που θα δούμε παρακάτω οι οποίοι αναφέρονται σε συγκεκριμένες χρονικές περιόδους.

Για να αντιμετωπίσουμε όλα τα παραπάνω προβλήματα και για να έχουμε όσο το δυνατόν πιο πιστή απόδοση των inserts δημιουργήσαμε τον πίνακα table_X. Ο πίνακας αυτός έχει 9 στήλες, τις 8 ίδιες με τον measurement_events που περιγράφηκαν παραπάνω και μία ένατη στήλη που την ονομάσαμε aa και είναι τύπου BIG SERIAL. Ο τύπος αυτός είναι στην ουσία ένας ακέραιος 8 bytes (bigint) του οποίου η τιμή ορίζεται αυτόματα από το sequence: `nextval('table_X_aa_seq'::regclass)`. Έχοντας ορίσει την δέκατη στήλη ως primary key του πίνακα γνωρίζω με ένα απλό « `WHERE aa>= AND aa<=` » ποια ακριβώς δεδομένα παίρνω από τον πίνακά μου. Στον πίνακα table_X έχουμε βάλει όλα τα δεδομένα που αφορούν τον Ιούνιο 2011 αφού αυτός είναι ο μήνας που εξετάζουμε παρακάτω και μας χρησιμεύει να γνωρίζουμε σε ποιες ημερομηνίες κυμαίνονται τα δεδομένα μας, πράγμα που στο κανονικό σύστημα είναι αυτονόητο. Δηλαδή, γνωρίζουμε ότι κάθε γραμμή δεδομένων που εισέρχεται στο online σύστημα αφορά την τρέχουσα ημερομηνία. Διατηρούμε το index στις στήλες (measurement_source_id, measurement_time) όπως είχε ο measurement_events, γιατί στη συνέχεια θα εξετάσουμε και χρονικά υποσύνολα του πίνακα αυτού. Ο πίνακας αυτός έχει περίπου 250 εκατομμύρια εγγραφές και προήλθε από την εντολή

```
INSERT INTO table_X
SELECT *
FROM measurement_events
WHERE measurement_time >= '2011-06-01 00:00:00+03'
  AND measurement_time <= '2011-06-30 23:59:59+03'.
```

Οι πίνακες προορισμού έχουν επίσης την ίδια μορφή. Το γεγονός ότι έχουν τη στήλη aa ως πρωτεύον κλειδί μας βγάζει εντελώς από τον κίνδυνο του duplicate_key error. Διατηρούμε παρόλα αυτά και το index στις στήλες (measurement_source_id, measurement_time) όπως είχε ο measurement_events αλλά πλέον ο συνδυασμός δεν χρειάζεται να είναι μοναδικός. Η στήλη aa έχει προστεθεί σε όλους τους πίνακες που θα δούμε παρακάτω, γιατί διευκολύνει στην επιλογή μοναδικών συνόλων δεδομένων χωρίς τη χρήση της εντολής limit η οποία μας δίνει τις Ν πρώτες γραμμές του πίνακα αλλά για να πάμε στις επόμενες Ν πρέπει να χρησιμοποιήσουμε offset πράγμα που καθυστερεί σημαντικά τη διαδικασία μας λόγω του όγκου των δεδομένων.

5.3 Λειτουργία

Στην ενότητα αυτή θα περιγράψουμε τη λειτουργία του συστήματος καθώς και τις αδυναμίες του βάσει των οποίων επιλέχθηκαν και οι παρακάτω τεχνικές βελτιστοποίησης που εφαρμόσαμε.

Το βασικό πρόβλημα του συστήματος είναι, όπως είπαμε και στην αρχή, το μεγάλο μέγεθος του κεντρικού πίνακα της βάσης όπου αποθηκεύονται όλες οι μετρήσεις. Αυτός ο πίνακας δέχεται συνεχώς νέα δεδομένα από τα ενεργειακά πάρκα (writes), ενώ ταυτόχρονα εκτελούνται πάνω του ερωτήσεις σχετικά με το ιστορικό των μετρήσεων (reads). Το γεγονός ότι στον πίνακα αυτό γίνονται ερωτήσεις καθιστά απαραίτητη την παρουσία index σε κάποιες από τις στήλες του, στην περίπτωσή μας σε δύο από αυτές, measurement_source_id και measurement_time. Η ύπαρξη index σε συνδυασμό με το μέγεθος του πίνακα προκαλεί μεγάλη καθυστέρηση στο σύστημα κατά την εισαγωγή νέων δεδομένων. Το γεγονός ότι τα inserts είναι το μεγαλύτερο πρόβλημα της εταιρείας που μας παραχώρησε τα δεδομένα, επιβεβαιώνεται και από το log analyzer που είχαμε στη διάθεσή μας. Σύμφωνα με αυτό, τα πιο αργά queries του συστήματος είναι τα begin-insert-commit που τρέχουν σε κάθε εισαγωγή νέας γραμμής δεδομένων. Το κομμάτι της εισαγωγής που ρυθμίζεται από την εντολή insert επηρεάζεται από τους παράγοντες που αναφέραμε παραπάνω, δηλαδή από το μέγεθος του πίνακα και την ύπαρξη index σ' αυτόν. Οι εντολές begin και commit που οριοθετούν μία οποιαδήποτε ολοκληρωμένη δοσοληψία (transaction), εισάγουν καθυστέρηση στο σύστημα ανάλογη της συχνότητας που καλούνται και ανεξάρτητη από τη δομή της βάσης ή το μέγεθος των πινάκων. Για μεγαλύτερη αξιοπιστία, η εταιρεία έχει επιλέξει να «κλειδώνει» κάθε εισαγωγή νέας γραμμής δεδομένων με αναμενόμενες συνέπειες στο χρόνο εκτέλεσης των inserts.

Όπως προαναφέρθηκε, εκτός από τα inserts που γίνονται στο σύστημα, εκτελούνται ταυτόχρονα και διάφορα ερωτήματα. Τα ερωτήματα αυτά ανήκουν σε δύο κατηγορίες.

Η πρώτη κατηγορία περιλαμβάνει ερωτήματα που αφορούν το ιστορικό των μετρήσεων και ζητούν απλή εμφάνιση ορισμένων γραμμών που αφορούν συγκεκριμένη χρονική περίοδο. Σύμφωνα με το log analyzer συχνά είναι τα ερωτήματα που ζητούν όλες τις εγγραφές για μία ημέρα ή για ένα μήνα ενός συγκεκριμένου measurement_source_id.

```
SELECT measurement_time,
       measurement_source_id,
       measurement_value
FROM measurement_events
WHERE measurement_source_id = '2f081632-591a-11e1-ae8a-00163e944340'
  AND measurement_time >= '2012-01-01 00:00:01+02'
  AND measurement_time <= '2012-02-01 00:00:00+03'
ORDER BY measurement_time;
```

Λιγότερο συχνά εκτελείται το ερώτημα που ζητά την τελευταία καταχώρηση που έγινε στον πίνακα για ένα συγκεκριμένο measurement_source_id.

```
SELECT measurement_time,
       measurement_source_id,
       measurement_value
FROM measurement_events
WHERE measurement_source_id = '2f081632-591a-11e1-ae8a-00163e944340'
ORDER BY measurement_time DESC LIMIT 1;
```

Η δεύτερη κατηγορία περιλαμβάνει ερωτήματα που ζητούν τον υπολογισμό του μέσου όρου ή του αθροίσματος για τη στήλη measurement_value για συγκεκριμένες χρονικές περιόδους (π.χ. ένας μήνας) ανά measurement_source_id. Η παρούσα δομή του συστήματος δεν επιτρέπει τέτοιους υπολογισμούς με χρήση SQL εσωτερικά της βάσης διότι η διαδικασία είναι πολύ χρονοβόρα και πολλές φορές δεν φέρνει αποτέλεσμα, κυρίως όταν ζητάμε υπολογισμούς για περισσότερα από ένα ids. Για το λόγο αυτό όλη η διαδικασία τέτοιων υπολογισμών γίνεται εξωτερικά της βάσης με χρήση άλλων προγραμμάτων.

Τέλος, μία βασική αδυναμία του συστήματος είναι το ότι η ανάκτηση της βάσης από αρχείο (restore) είναι μία πολύ αργή διαδικασία.

6 *Βέλτιστος αρχικός σχεδιασμός συστήματος*

6.1 Λογικός σχεδιασμός βάσης

Για τη βέλτιστη λειτουργία ενός συστήματος βάσεων δεδομένων, είναι απαραίτητο να έχει γίνει πρώτα ένας προσεκτικός λογικός σχεδιασμός της βάσης. Είναι ίσως ο σημαντικότερος παράγοντας όσον αφορά την απόδοση του συστήματος. Αυτός που σχεδιάζει το σχήμα της βάσης, οφείλει να γνωρίζει άριστα το σκοπό για τον οποίο αυτή πρόκειται να χρησιμοποιηθεί. Βασισμένος στις ανάγκες που θα εξυπηρετεί η εφαρμογή και στους ανθρώπους που 8α τη χρησιμοποιούν, πρέπει να μοντελοποιήσει σωστά τις απαιτήσεις της και να σχεδιάσει με βέλτιστο τρόπο τις οντότητες που θα απαρτίζουν τη βάση, τα γνωρίσματα τους και τις σχέσεις μεταξύ τους.

6.2 Φυσικός σχεδιασμός βάσης

Μεγάλο ρόλο στην αποδοτικότητα ενός συστήματος βάσεων δεδομένων παίζει επίσης ο σχεδιασμός της βάσης σε φυσικό επίπεδο. Με τον όρο φυσικό επίπεδο αναφερόμαστε στο πως αναπαρίστανται στον υπολογιστή αυτά που δημιουργήσαμε στο λογικό σχεδιασμό. Καταρχάς πρέπει να επιλέξουμε τον τύπο των στηλών, των κλειδιών και των ευρετηρίων ώστε να έχουμε την αποδοτικότερη λύση. Για παράδειγμα αν κάποια οντότητα έχει ένα γνώρισμα 'id', οφείλουμε να γνωρίζουμε ως σχεδιαστές της βάσης αν αυτό είναι αριθμός ή αν περιέχει και άλλους χαρακτήρες, ώστε να ορίσουμε την αντίστοιχη στήλη ως int αντί για text. Όσον αφορά τα ευρετήρια, υπάρχουν διάφορα είδη (B-tree, Hash, GiST, GIN κ.α.) καθένα από τα οποία ανταποκρίνεται καλύτερα σε άλλου είδους ερωτήματα. Αν σε κάποια στήλη για παράδειγμα γίνονται range queries θα προτιμήσουμε B-tree index, ενώ αν σε κάποια άλλη γίνονται μόνο equality queries είναι καλύτερο το Hash index. Ακόμη, πρέπει να φροντίσουμε ώστε κάθε δοσοληψία να κάνει όσο το δυνατόν περισσότερες ενέργειες. Επίσης, πρέπει να δοθεί μεγάλη προσοχή στην επιλογή των στηλών που θα έχουν ευρετήρια. Τέτοιες στήλες συνήθως είναι αυτές που αποτελούν ξένα κλειδιά και αυτές που συμμετέχουν συχνά στα where clauses των ερωτημάτων. Αντιθέτως, πρέπει να αποφεύγονται τα ευρετήρια σε πίνακες

στους οποίους γίνονται πολύ συχνά inserts, updates και deletes καθώς και σε μικρούς πίνακες ή πίνακες με low-cardinality columns. Τέλος, όπου είναι εφικτό πρέπει να αποφεύγουμε τα joins.

6.3 Απαιτούμενες προδιαγραφές Hardware

Πέρα από τις παραπάνω ενέργειες που αφορούν αποκλειστικά το σχεδιασμό της βάσης, η καλή λειτουργία ενός συστήματος εξαρτάται και από το hardware.

Καταρχάς είναι πολύ σημαντικό να υπάρχει αρκετή RAM και η βάση να είναι ρυθμισμένη ώστε να την αξιοποιεί.

Εξίσου σημαντική είναι η χρήση όσο το δυνατόν γρηγορότερων δίσκων. Αυτό προφανώς βοηθάει στα inserts. Ιδιαίτερα όμως σε μεγάλες βάσεις, βοηθάει και στα selects καθώς τα ερωτήματα πολλές φορές δίνουν πληροφορία η οποία είναι πολύ μεγάλη για να αποθηκευτεί στη RAM και επομένως η ταχύτητα I/O του δίσκου καθίσταται αποφασιστικός παράγοντας.

Σημαντική επίσης είναι η χρήση ταχύτερης CPU και η ρύθμιση του server ώστε να χρησιμοποιεί το 100% της ταχύτητάς της.

Ακόμη, είναι σκόπιμο να έχουμε όσο το δυνατόν περισσότερους πυρήνες, ανάλογα πάντα και με το πόσα ταυτόχρονα ερωτήματα εκτελούνται στη βάση.

Τέλος αναφέρουμε πως παράγοντες όπως η L2 cache, οι κάρτες και τα καλώδια δικτύου, τα switches, αλλά και η θερμοκρασία και υγρασία του δωματίου του server μπορούν επίσης να αποτελέσουν bottleneck και επομένως χρειάζονται κι αυτά προσοχή. **[10]**

6.4 Άλλα θέματα tuning

- Τοποθέτηση tables και transaction log σε διαφορετικούς δίσκους για καλύτερη ισορροπία disk I/O και αποτροπή read queuing. Το βέλτιστο είναι πολύ μεγάλοι πίνακες να βρίσκονται σε ξεχωριστό tablespace και δίσκο.
- Χρήση μέγιστης δυνατής cache της βάσης για άμεση ανάκτηση των δεδομένων που έχουν ξαναχρησιμοποιηθεί.
- Defragmentation του συνόλου των αρχείων της βάσης, των πινάκων, των indexed στηλών.

7 *Τεχνικές Βελτιστοποίησης*

7.1 PostgreSQL Configuration

Είναι πολύ σημαντικό να αναφέρουμε πως η PostgreSQL είναι ένα εξαιρετικά προσαρμόσιμο (highly customizable) σύστημα ΒΔ. Συγκεκριμένα τα πάντα ελέγχονται από ένα και μόνο αρχείο, το postgresql.conf (default location σε debian: /etc/postgresql/(έκδοση)/main/postgresql.conf). Δε θα αναλύσουμε κάθε παράμετρο του αρχείου, αλλά θα σταθούμε σε μερικές τις οποίες αλλάξαμε και βελτίωσαν την απόδοση.

7.1.1 Memory Consumption

shared_buffers: Καθορίζει το μέγεθος της μνήμης που χρησιμοποιεί ο DB server για shared memory buffers. Αφού ο server έχει πάνω από 1GB RAM, είναι ασφαλές (με την έννοια ότι θα υπάρχει αρκετή RAM για τις υπόλοιπες λειτουργίες - OS κλπ) να θέσουμε την τιμή της παραμέτρου αυτής στο 25% της RAM. Παραπάνω δε χρειάζεται γιατί η PostgreSQL βασίζεται επίσης και στην cache του OS.

work_mem: Καθορίζει πόση μνήμη χρησιμοποιείται από sorts (ORDER BY, DISTINCT, MIN, MAX) και από hash tables (joins, aggregates, IN subqueries). Επομένως, όταν γίνονται συχνά (ή και ταυτόχρονα από πολλούς χρήστες) τέτοια queries, η τιμή αυτή πρέπει να είναι αρκετά μεγαλύτερη του default 1MB γιατί μόνο τόση μνήμη μπορεί να χρησιμοποιηθεί από τα queries πριν γράψουν τα data τους σε temporary files.

maintenance_work_mem: Καθορίζει πόση μνήμη έχουν στη διάθεσή τους λειτουργίες όπως VACUUM, CREATE INDEX, ALTER TABLE ADD FOREIGN KEY και άλλες λειτουργίες συντήρησης. Άρα, είναι πολύ σημαντικό η τιμή αυτή να είναι αρκετή (default=16MB) ώστε να ανταποκρίνεται στις ανάγκες μίας πολύ μεγάλης βάσης και να μην γίνουν τέτοιες λειτουργίες το bottleneck της απόδοσης. **[10], [11]**

7.1.2 Write Ahead Log

wal_buffers: Καθορίζει πόση shared memory χρησιμοποιείται για WAL data τα οποία δεν έχουν γραφτεί ακόμα στο δίσκο. Μερικά MB αντί για το default του 1/32 της τιμής της

παραμέτρου shared_buffers είναι απαραίτητα για καλύτερη απόδοση σε έναν busy server στον οποίο γίνονται πολλά commits από πολλούς clients.

fsync: Αν η τιμή της παραμέτρου αυτής είναι «on», ο PostgreSQL server θα προσπαθεί με τη χρήση μεθόδων όπως η αποστολή σημάτων fsync (ερώτηση αν είναι synchronized τα data του δίσκου με το WAL και τα temp files) να εξακριβώνει πως οποιαδήποτε αλλαγή γράφεται σίγουρα και στο δίσκο. Θέτοντας την τιμή στο «off», η διαδικασία αυτή δε γίνεται, με αποτέλεσμα να υπάρχει σημαντική βελτίωση σε queries. Είναι όμως αρκετά παρακινδυνευμένο αφού κανείς δεν εγγυάται πως η βάση θα ανανήψει μετά από ένα H/W ή OS crash ή restart του server και γι αυτό την αφήσαμε τελικά στο on.

synchronous_commit: Αν είναι «on», τότε επιστρέφεται success από κάποιο query στον client μόνο εφόσον έχουν γραφτεί τα WAL records στο δίσκο. Αν είναι «off», τότε δεν είναι απαραίτητο να έχουν γραφτεί τα WAL records στο δίσκο και επομένως μπορεί να υπάρξει κάποια καθυστέρηση μεταξύ του success που βλέπει ο client και του να είναι ασφαλή τα δεδομένα του transaction αυτού αν εκείνη τη στιγμή συμβεί κάποιο crash. Παρόλα αυτά, το να χαθούν ελάχιστα δεδομένα δεν είναι και πολύ σημαντικό αφού κερδίζουμε σε ταχύτητα. Αντιθέτως δηλαδή με το fsync off δεν υπάρχει κίνδυνος για inconsistency μετά από crash, αφού μόνο ελάχιστες transactions δε θα έχουν γράψει τα αποτελέσματά τους στο δίσκο. **[10], [11]**

7.1.3 Query Tuning

effective_cache_size: Καθορίζει τη μνήμη που έχουν στη διάθεση τους όλα τα υπόλοιπα queries. Μία καλή τιμή είναι τα 2/3 της ελεύθερης RAM (αντί του default 128MB), αφού τη μνήμη αυτή θα μοιράζονται όλα τα queries που τρέχουν ταυτόχρονα στον server. **[10], [11]**

7.2 Indexing

7.2.1 Θεωρία

Index, όπως καταλαβαίνουμε και από την αντίστοιχη ελληνική λέξη, είναι ένα ευρετήριο. Σε μία βάση δεδομένων δηλαδή, αν υπάρχει index σε κάποιο γνώρισμα μιας οντότητας δε χρειάζεται να διαβαστεί ολόκληρος ο πίνακας για να βρεθεί η ζητούμενη γραμμή. Δηλαδή όταν ζητηθεί μια τιμή ή εύρος τιμών, βρίσκει η βάση τη θέση των τιμών αυτών στον πίνακα και στη συνέχεια βρίσκει τις συγκεκριμένες γραμμές που απαντάνε το ερώτημα. Αν από την άλλη δεν υπήρχε το index, η βάση θα έκανε αναζήτηση σε ολόκληρο τον πίνακα, οπότε αν είχε πολλές γραμμές θα καθυστερούσε σημαντικά.

Η προσθήκη index είναι πολύ σημαντική για την βελτίωση της απόκρισης μιας βάσης δεδομένων, εφόσον βέβαια χρησιμοποιηθεί σωστά. Αν ένας πίνακας είναι πολύ μεγάλος ή αναμένουμε να γίνει, τότε είναι απαραίτητο να υπάρχει index στις στήλες του πίνακα αυτού με βάση τις οποίες γίνονται συχνά αναζητήσεις (αυτές που εμφανίζονται στο WHERE-clause). Ακόμη, index χρειάζεται και σε ξένα κλειδιά (foreign keys) τα οποία συνδέουν μεγάλους πίνακες μεταξύ τους. Παρόλα αυτά, χρειάζεται προσοχή στην χρήση index γιατί μπορεί να κάνουν τις αναγνώσεις πιο γρήγορες, αλλά προσθέτουν φόρτο εργασίας στα στάδια της εισαγωγής, ενημέρωσης και διαγραφής δεδομένων, ειδικά όταν πρόκειται για μεγάλους πίνακες.

Από τους διάφορους τύπους index που παρέχει η PostgreSQL, εμείς χρησιμοποιούμε τον τύπο B-tree. Το B-tree index είναι μία δενδροειδής δομή δεδομένων στην οποία κάθε κόμβος μπορεί να έχει πάνω από δύο παιδιά. Αποτελεί την καλύτερη επιλογή για πίνακες με πολλές εγγραφές αφού εξασφαλίζει λογαριθμικό χρόνο για αναζήτηση, επιλογή, εισαγωγή και διαγραφή δεδομένων. Παρόλα αυτά χρειάζεται ξαναχτίσιμο (rebuild) των indexes αν έχουν γίνει πολλές εισαγωγές, για να συνεχίσουν να είναι αποτελεσματικά. **[12]**

7.2.2 Πείραμα

7.2.2.1 Περιγραφή

Στην ενότητα αυτή θα περιγράψουμε τους τρόπους με τους οποίους διαπιστώσαμε κατά πόσο βελτιώνει την απόδοση της βάσης μας η αξιοποίηση των παραπάνω ιδιοτήτων των ευρετηρίων. Επιλέξαμε να τρέξουμε ισοδύναμα queries που περιέχουν στο where-clause στήλες με ή χωρίς index. Για παράδειγμα, ζητήσαμε την εμφάνιση 10000 γραμμών με βάση το measurement_source_id, που είναι indexed στήλη, και στη συνέχεια ζητήσαμε την ίδια ποσότητα δεδομένων με βάση το measurement_location_id που είναι μία unindexed στήλη.

```
SELECT *
FROM measurement_events
WHERE measurement_source_id = '31fc15bc-9c24-11e0-b25c-00163e944340'
LIMIT 10000;
```

Πίνακας 7. Query 1 με where-clause σε Indexed Column

```
SELECT *
FROM measurement_events
WHERE measurement_location_id= '20462ec0-dee2-11e0-25fe-00162e944340'
LIMIT 10000;
```

Πίνακας 8. Query 2 με where-clause σε Unindexed Column

Τα αποτελέσματα ήταν πολύ βελτιωμένα στην περίπτωση της αναζήτησης με index και παρουσιάζονται αναλυτικά στην ενότητα Μετρήσεις.

Επίσης μελετήσαμε κατά πόσο «χαλάει» η δομή του β-δέντρου του index μετά από πολλά inserts. Στην προκειμένη περίπτωση χρησιμοποιήσαμε δύο πίνακες με 10 εκατομμύρια γραμμές ο καθένας. Στον πρώτο, αφού κάναμε insert τα δεδομένα, φτιάξαμε από την αρχή το index. Στον δεύτερο κάναμε insert τα πρώτα 5 εκατομμύρια, φτιάξαμε το index και τέλος κάναμε insert άλλες 5 εκατομμύρια γραμμές. Σκοπός αυτού του δεύτερου πειράματος ήταν να δούμε κατά πόσο συμφέρει το να γίνεται rebuild το index, σε σύγκριση με την ενδεχόμενη καθυστέρηση των queries λόγω «χαλασμένου» index. Διαλέξαμε να χρησιμοποιήσουμε ένα βασικό query που βασίζεται στο index measurement_source_id όπως μας δόθηκε από το log analyzer PgFouine που έχουμε στη διάθεση μας από την εταιρεία, το οποίο φαίνεται παρακάτω:

```
SELECT measurement_time,
       measurement_source_id,
       measurement_value
FROM   ind_i   --ind_1   και   ind_2   είναι   οι   δύο   πίνακες   που
χρησιμοποιήθηκαν

WHERE measurement_location_id= '20464ec0-dee2-11e0-85fe-00163e944340'
ORDER BY measurement_time DESC LIMIT 1;
```

Πίνακας 9. Query για την εύρεση της τελευταίας μέτρησης για ένα συγκεκριμένο id

Για να είναι αντικειμενική η μέτρηση τρέξαμε το παραπάνω για όλα τα διαφορετικά ids που υπάρχουν στον κάθε πίνακα με τη βοήθεια της synartisi1 και βγάλαμε το μέσο όρο. Το αποτέλεσμα φαίνεται και αναλύεται επίσης στην παρακάτω ενότητα.

```
CREATE OR REPLACE FUNCTION synartisi1() RETURNS
SETOF record LANGUAGE plpgsql AS $$
DECLARE
   i uuid;
BEGIN
FOR i IN
  (SELECT DISTINCT(measurement_source_id)
   FROM ind_i)
LOOP
RETURN QUERY
SELECT measurement_time,
       measurement_source_id,
       measurement_value
FROM ind_i
WHERE measurement_source_id = i
ORDER BY measurement_time DESC LIMIT 1;
END LOOP;
END;$$;
```

Συνάρτηση 1 - Κώδικας plpgsql

Για την κλήση της παραπάνω συνάρτησης που είναι τύπου setof record χρησιμοποιήσαμε την εντολή

```
SELECT *
FROM synartisi1() AS a(measurement_time TIMESTAMP WITH time ZONE,measurement_source_id uuid, measurement_value character varying);
```

7.2.2.2 Μετρήσεις

Query 1	68,919.875 ms
Query 2	338,515.014 ms

Πίνακας 10. Αποτελέσματα ερωτήσεων με κριτήριο αναζήτησης indexed και unindexed στήλες για 10000 ids.

Από τον παραπάνω πίνακα φαίνεται ότι ένα query που χρησιμοποιεί το index στο where-clause κάνει κατά μέσο όρο 6.89 ms ανά id ενώ ένα που χρησιμοποιεί κάποια τυχαία στήλη για αναζήτηση κάνει σχεδόν πενταπλάσιο χρόνο (κατά μέσο όρο 33.9 ms ανά id). Είναι λοιπόν εμφανές ότι μας συμφέρει να έχουμε index στις στήλες τις οποίες χρησιμοποιούν τα queries.

Ωστόσο πρέπει η χρήση index να είναι λελογισμένη διότι στην περίπτωσή μας τόσο τα selects όσο και τα inserts «πέφτουν» στους ίδιους πίνακες, και μπορεί η χρήση index να βελτιώνει τα selects αλλά προκαλεί σημαντική καθυστέρηση στα inserts. Σε επόμενη ενότητα θα δούμε κατά πόσο συμφέρει να χωριστούν τα δεδομένα και να γίνονται οι δύο λειτουργίες σε διαφορετικούς πίνακες.

Το δεύτερο πείραμα που κάναμε όσον αφορά το indexing ήταν να δούμε αν μας συμφέρει η επιλογή του rebuild. Στον παρακάτω πίνακα φαίνονται τα αποτελέσματα της Συνάρτησης 1 όταν έτρεξε στους πίνακες ind_1 και ind_2.

ind_1	95,775.536 ms
ind_2	172,365.942 ms

Πίνακας 11. Αποτελέσματα της Συνάρτησης 1 σε πίνακα με σωστό και «χαλασμένο» index

Προφανώς όταν εκτελέσαμε το query μας στον πίνακα που το index ήταν φτιαγμένο πάνω σε όλα τα δεδομένα του πίνακα ήταν αισθητά γρηγορότερο. Χρειαστήκαμε τον διπλάσιο σχεδόν χρόνο για να τρέξουμε το ίδιο query στον πίνακα, του οποίο το index είχε επηρεαστεί από τα inserts. Στη συνέχεια κάναμε reindex τον πίνακα ind_2 και ξανατρέξαμε το query. Βρήκαμε ότι ο χρόνος πλέον μειώθηκε και είναι αντίστοιχος με τον χρόνο που έκανε για τον ind_1, στην προκειμένη περίπτωση 98,075.787 ms. Για να δούμε όμως κατά πόσο μας συμφέρει το rebuild του index χρονομετρήσαμε και την εντολή `REINDEX INDEX index_name;` και

πήραμε τον χρόνο 84,659.424 ms. Αν συνυπολογίσουμε τους χρόνους αυτούς παρατηρούμε ότι το αποτέλεσμα είναι το ίδιο με την εκτέλεση του query στον πίνακα με το «χαλασμένο» index. Συμπερασματικά, από το παραπάνω πείραμα βλέπουμε ότι το rebuild συμφέρει αν στη βάση μας έχουμε συχνότερα selects από ότι inserts. Σε αντίθετη περίπτωση όσες φορές και να κάνουμε rebuild χάνουμε χρόνο γιατί τα συνεχόμενα inserts στο σύστημά μας ακυρώνουν ουσιαστικά την βελτίωση που θα προσέφερε κάτι τέτοιο.

7.2.3 Λεπτομέρειες

Κατά τη δημιουργία των πινάκων ind_1 και ind_2 παρατηρήσαμε τα εξής χαρακτηριστικά σχετικά με την ταχύτητα των inserts σε indexed και unindexed πίνακες. Αρχικά δημιουργήσαμε τους πίνακες με τις εντολές

```
SELECT * INTO ind_1
FROM table_X
WHERE aa>=1
  AND aa<=10000000
```

και

```
SELECT * INTO ind_2
FROM table_X
WHERE aa>=1
  AND aa<=5000000
```

αντίστοιχα με σκοπό να δημιουργηθούν δύο πίνακες όμοιοι με τον measurement_events σε δομή οι οποίοι δεν περιέχουν κανένα index. Η διαδικασία αυτή έγινε πολύ γρήγορα λόγω απουσίας index από τον πίνακα(58,840.492 ms και 28,246.780 ms αντίστοιχα). Έπειτα κάναμε build index και στους δύο. Και τέλος κάναμε

```
INSERT INTO ind_2
SELECT *
FROM table_X
WHERE aa>=5000001
  AND aa<=10000000.
```

Η τελευταία αυτή εντολή έκανε περίπου 6.5 ώρες (23,379,563.332 ms) για να εκτελεστεί λόγω της ύπαρξης index στον πίνακα.

7.3 Ενδιάμεσοι πίνακες

7.3.1 Θεωρία

Με τον όρο ενδιάμεσοι πίνακες αναφερόμαστε σε πίνακες που δημιουργήσαμε στη βάση για να αποθηκεύουμε δεδομένα πάνω στα οποία ο χρήστης εκτελεί συχνά ερωτήματα και τα οποία απαιτούν κάποια μορφή υπολογισμού όπως για παράδειγμα μέσοι όροι, αθροίσματα, μέγιστα και ελάχιστα στοιχεία κ.ά. Οι υπολογισμοί αυτοί γίνονται με ομαδοποίηση σύμφωνα με κάποια στήλη κλειδί (group by - clause). Έτσι ο κάθε πίνακας αποθηκεύει το αποτέλεσμα της εκτέλεσης του ζητούμενου υπολογισμού και δεν χρειάζεται κάθε φορά να ξοδεύεται πολύτιμος χρόνος για επανεκτίμηση του ερωτήματος, πόσο μάλλον όταν αυτό αναφέρεται σε έναν τεράστιο πίνακα όπως ο measurement_events. Αυτή η τεχνική φαίνεται πολλά υποσχόμενη όμως κρύβει ένα μεγάλο μειονέκτημα. Όπως θα φανεί και στις συναρτήσεις που θα παρουσιάσουμε στο πειραματικό κομμάτι της ενότητας αυτής για τη δημιουργία των ενδιάμεσων πινάκων απαιτείται με κάθε insert που γίνεται στον κύριο πίνακα να πραγματοποιείται και ένα insert/update στον αντίστοιχο ενδιάμεσο πίνακα. Έτσι δεν αρκεί μόνο να συγκρίνουμε ένα απλό select στον ενδιάμεσο με ένα select aggregate στον κύριο, αλλά πρέπει να συνυπολογιστεί το κόστος των inserts/updates που απαιτούνται για να αποφανθούμε αν η τεχνική αυτή όντως βελτιστοποιεί τη απόδοση του συστήματος ή προκαλεί επιπρόσθετη καθυστέρηση.

7.3.2 Πείραμα

7.3.2.1 Περιγραφή

Αρχικά θα περιγράψουμε τους ενδιάμεσους πίνακες που θεωρήσαμε σκόπιμο να δημιουργήσουμε βάσει των αναγκών της βάσης μας.

- **current_day**: Πίνακας με τέσσερις στήλες measurement_source_id (uuid): primary key, sum (real), avg(real), count(integer). Αποθηκεύει το άθροισμα και το μέσο όρο της στήλης measurement_value του πίνακα measurement_events ανά measurement_source_id για την τελευταία ημέρα. Η στήλη count είναι μία βοηθητική στήλη η οποία μετράει το πόσες εγγραφές συνυπολογίστηκαν στην τιμή του sum και του avg και χρησιμεύει στον «προγραμματιστικό» υπολογισμό των νέων τιμών κάνοντας χρήση των παλιών χωρίς να χρειαστεί να ξαναγίνει υπολογισμός από τον πίνακα measurement_events με sql query.

  Ο πίνακας αυτός δημιουργήθηκε με χρήση της synartisi2. Η ίδια συνάρτηση περιλαμβάνει και την δυνατότητα του update για κάθε καινούριο insert. Στην ουσία

καλούμε την synartisi2 για όλα τα στοιχεία του measurement_events την πρώτη φορά που θα «γεμίσουμε» τον πίνακα current_day και στην συνέχεια την καλούμε για κάθε καινούριο insert που γίνεται στη βάση ώστε να κρατάμε τον πίνακά μας ενημερωμένο. Παρακάτω παρουσιάζεται ο κώδικας της βασικής αυτής συνάρτησης και αναλύεται η λειτουργία της.

```
CREATE OR REPLACE FUNCTION synartisi2(a uuid, b uuid, c
timestamp with time zone, d timestamp with time zone, e
character varying, f character varying, g uuid, h uuid) RETURNS
void
    LANGUAGE plpgsql
    AS $$BEGIN
    LOOP

        INSERT INTO measurement_events
        VALUES (a,b,c,d,e,f,g,h);

        UPDATE current_day
        SET
              sum= ((SELECT sum
                     FROM current_day
                     WHERE measurement_source_id=b)
                     + (e::real)),
              count = ((SELECT count
                        FROM current_day
                        WHERE measurement_source_id=b)
                        + 1)
        WHERE measurement_source_id=b ;

        UPDATE current_day
        SET avg = ((SELECT sum
                    FROM current_day
                    WHERE  measurement_source_id=b)
                    /
                   (SELECT count
                    FROM current_day
                    WHERE measurement_source_id=b))
        WHERE measurement_source_id=b ;

        IF found THEN
            RETURN;
        END IF;

        BEGIN
            INSERT INTO current_day
            VALUES (b, e::real, e::real, 1);
            RETURN;
        EXCEPTION WHEN unique_violation THEN
        END;

    END LOOP;
END;$$;
```

Συνάρτηση 2 - Κώδικας plpgsql

Λειτουργία Συνάρτησης 2 :

Κλήση κατά την εισαγωγή νέας γραμμής στη βάση: Η synartisi2 έχει ακριβώς τη μορφή που δίνεται στον παραπάνω πίνακα. Παίρνει ως είσοδο 8 ορίσματα (measurement_location_id, measurement_source_id, measurement_time, event_reception_time, measurement_value, quality, source_id, transducer_id) τα οποία αρχικά εισάγει στον πίνακα measurement_events. Στη συνέχεια, αν υπάρχει ήδη το συγκεκριμένο measurement_source_id στον πίνακα current_day, τότε γίνεται update της τιμής της στήλης sum σύμφωνα με τον «προγραμματιστικό» τύπο sum_new =sum_old + measurement_value και η στήλη count αυξάνεται κατά μία μονάδα με όμοιο τρόπο count_new=count_old+1. Στη συνέχεια γίνεται update της τιμής της στήλης avg με υπολογισμό του τύπου avg_new=sum_new/count_new. Σε αντίθετη περίπτωση αν το measurement_source_id που εμφανίζεται ως όρισμα δεν υπάρχει στον πίνακα current_day τότε προστίθεται σ' αυτόν μία νέα γραμμή με sum=avg=measurement_value και count=1. Σημειώνεται ότι τα παραπάνω sum_old, sum_new, count_old, count_new, avg_new αποτελούν καταχρηστικές ονομασίες για καλύτερη κατανόηση του κώδικα από τον αναγνώστη και δεν αντιστοιχούν σε πραγματικές μεταβλητές της συνάρτησης.

Κλήση κατά την δημιουργία του πίνακα current_day: Για την δημιουργία του πίνακα current_day αν υποθέσουμε ότι δεν τον είχαμε εξ' αρχής στη βάση μας, τρέχουμε την βοηθητική συνάρτηση synartisi3 με σκοπό να συμπεριλάβουμε στους υπολογισμούς μας τα inserts που «χάσαμε».

```
CREATE OR REPLACE FUNCTION synartisi3() RETURNS void LANGUAGE
plpgsql AS $$DECLARE a uuid; b uuid; c TIMESTAMP WITH time
ZONE; d TIMESTAMP WITH time ZONE; e character varying; f
character varying; g uuid; h uuid; BEGIN
FOR a, b, c, d, e, f, g, h IN
  (SELECT measurement_location_id,
          measurement_source_id,
          measurement_time,
          event_reception_time,
          measurement_value,
          quality,
          source_id,
          transducer_id
   FROM measurement_events
   WHERE measurement_time>=CURRENT_DATE)
LOOP
PERFORM synartisi2(a,b,c,d,e,f,g,h);
END LOOP;
END;$$;
```

Συνάρτηση 3 - Κώδικας plpgsql

Λειτουργία Συνάρτησης 3 :

Η synartisi3 διατρέχει τον πίνακα measurement_events και για όσες γραμμές δεδομένων αφορούν την τρέχουσα ημερομηνία καλεί την «τροποποιημένη» synartisi2 (παραλείποντας το πρώτο insert) και έτσι δημιουργείται ο πίνακας current_day.

Πίνακας current_day – Αρχικοποίηση :

Σκοπός του πίνακα αυτού είναι να κρατάει μεγέθη που αφορούν μόνο την τρέχουσα ημέρα. Επομένως, πρέπει να γίνεται άδειασμα του πίνακα κάθε φορά που αλλάζει η ημέρα. Θα περιγράψουμε έναν τρόπο ρύθμισης του συστήματός μας στην ενότητα Λεπτομέρειες για αυτοματοποίηση αυτής της διαδικασίας.

- **current_week**: Πίνακας με τέσσερις στήλες measurement_source_id (uuid): primary key, sum (real), avg(real) , count(integer). Αποθηκεύει το άθροισμα και το μέσο όρο της στήλης measurement_value του πίνακα measurement_events ανά measurement_source_id για την τελευταία εβδομάδα. Η μόνη διαφορά με τον πίνακα current_day είναι το χρονικό διάστημα που αντιπροσωπεύουν τα αποθηκευμένα μεγέθη. Και εδώ χρησιμοποιήσαμε τις συναρτήσεις 2 και 3 για τη δημιουργία και την ενημέρωση του πίνακα με την μόνη διαφορά ότι η συνθήκη του where-clause στην synartisi3 γίνεται measurement_time >= (select date_trunc('week', current_date)) που επιστέφει την ημερομηνία της Δευτέρας της τρέχουσας βδομάδας. Για την αρχικοποίηση αυτού του πίνακα, πρέπει να γίνεται άδειασμα κάθε φορά που αλλάζει η εβδομάδα.
- **current_month**: Πίνακας με τέσσερις στήλες measurement_source_id (uuid) : primary key , sum (real), avg(real) , count(integer). Αποθηκεύει το άθροισμα και το μέσο όρο της στήλης measurement_value του πίνακα measurement_events ανά measurement_source_id για τον τελευταίο μήνα. Η μόνη διαφορά με τους παραπάνω πίνακες είναι το χρονικό διάστημα που αντιπροσωπεύουν τα αποθηκευμένα μεγέθη. Και εδώ χρησιμοποιήσαμε τις συναρτήσεις 2 και 3 για τη δημιουργία και την ενημέρωση του πίνακα με την μόνη διαφορά ότι η συνθήκη του where-clause στην synartisi3 γίνεται measurement_time >= (select date_trunc('month', current_date) που επιστέφει την ημερομηνία της πρώτης μέρας του τρέχοντος μήνα.

Παραπάνω περιγράψαμε τις συναρτήσεις που τρέχουμε για να κάνουμε insert/update χρησιμοποιώντας τους ενδιάμεσους πίνακες. Στο σημείο αυτό θα δείξουμε με ποιες συναρτήσεις ή απλά queries αντιστοιχίζεται η παραπάνω διαδικασία με το υπάρχον σύστημα

για να καταλήξουμε τελικά στη σύγκριση των δύο μεθόδων στην παρακάτω ενότητα μιλώντας πια με αριθμητικά δεδομένα.

```
INSERT INTO table_Y
SELECT *
FROM table_X
WHERE aa>=1
  AND aa<= 1000000;
```

Πίνακας 12. Εντολή για Inserting σε κανονικό indexed πίνακα από τον βοηθητικό για ένα εκατομμύριο γραμμές

Όλοι οι πίνακες προορισμού δεδομένων για τους οποίους έγιναν οι παρακάτω μετρήσεις είναι μεγέθους 90 εκατομμυρίων γραμμών.

Ταυτόχρονα με τα inserts εκτελούνταν και η παρακάτω συνάρτηση, synartisi4, που μας δίνει ως αποτέλεσμα τα sum και avg της στήλης measurement_value για N συγκεκριμένα measurement_source_id για ένα μήνα.

```
CREATE OR REPLACE FUNCTION synartisi4() RETURNS
SETOF record LANGUAGE plpgsql AS $$DECLARE DECLARE i integer; x uuid;
ret RECORD; BEGIN
FOR i IN 1..N LOOP
SELECT measurement_source_id INTO x
FROM table_X
WHERE aa=i;
  SELECT x,
         AVG(measurement_value::real),
         SUM(measurement_value::real)
  FROM table_X WHERE measurement_source_id=x INTO ret; RETURN NEXT
ret;
END LOOP;
RETURN;
END;$$;
```

Συνάρτηση 4. Κώδικας plpgsql

Εδώ για παράδειγμα χρησιμοποιούμε το aa για να πάρουμε ξεχωριστά ids. Αν δεν είχαμε τη στήλη aa θα έπρεπε να κάνουμε `SELECT DISTINCT(measurement_source_id) FROM table_X LIMIT N` στη συνθήκη του IN.

7.3.2.2 Μετρήσεις

Αφού περιγράψαμε αναλυτικά τη λειτουργία των ενδιάμεσων πινάκων που δημιουργήσαμε στη βάση, θα παρουσιάσουμε παρακάτω τα αποτελέσματα που προέκυψαν από την εκτέλεση

των παραπάνω συναρτήσεων, καθώς και τις ακριβείς ιδιότητες του περιβάλλοντος στο οποίο πραγματοποιήθηκαν οι μετρήσεις αυτές για να έχουμε όσο το δυνατό πιο πιστή αναπαράσταση του συστήματος.

Αρχικά να πούμε ότι όλη η φιλοσοφία των ενδιάμεσων πινάκων σαν τεχνική βελτιστοποίησης είναι ότι θέλουμε να ξεχωρίσουμε τα reads και τα writes που γίνονται στον ίδιο πίνακα. Αυτό γίνεται διότι οι δύο λειτουργίες έχουν ακριβώς αντίστροφη συμπεριφορά σε συγκεκριμένες δομές της βάσης. Για παράδειγμα ένας πίνακας που δεν περιέχει καθόλου index είναι ιδανικός για γρήγορα inserts, ενώ ένας πίνακας με index σε όλες τις στήλες μπορεί να απαντήσει σε οποιοδήποτε query με οποιοδήποτε κριτήριο αναζήτησης πολύ γρήγορα. Αντίστροφα κάτι τέτοιο θα ήταν καταστροφικό αν θέλαμε να εισάγουμε δεδομένα στον δεύτερο πίνακα ή αν εκτελούσαμε κάποιο query στον πρώτο.

Με αυτή τη λογική κάναμε την παρακάτω σύγκριση. Εκτελέσαμε ένα εκατομμύριο inserts σε έναν πίνακα με index ενώ παράλληλα ζητήσαμε τον υπολογισμό μέσου όρου και αθροίσματος μίας στήλης του ίδιου πίνακα. Σε αντιδιαστολή με αυτό, εκτελέσαμε ένα εκατομμύριο inserts σε έναν πίνακα χωρίς καθόλου index περνώντας τα ίδια δεδομένα των inserts μέσω της synartisi2 που παρουσιάσαμε παραπάνω, στον αντίστοιχο ενδιάμεσο πίνακα. Ταυτόχρονα ζητήσαμε να εμφανιστούν ο μέσος όρος και το άθροισμα για κάποιο συγκεκριμένο id μέσω του ενδιάμεσου πίνακα. Τα δεδομένα αφορούν τον ενδιάμεσο πίνακα που καλύπτει το διάστημα ενός μήνα.

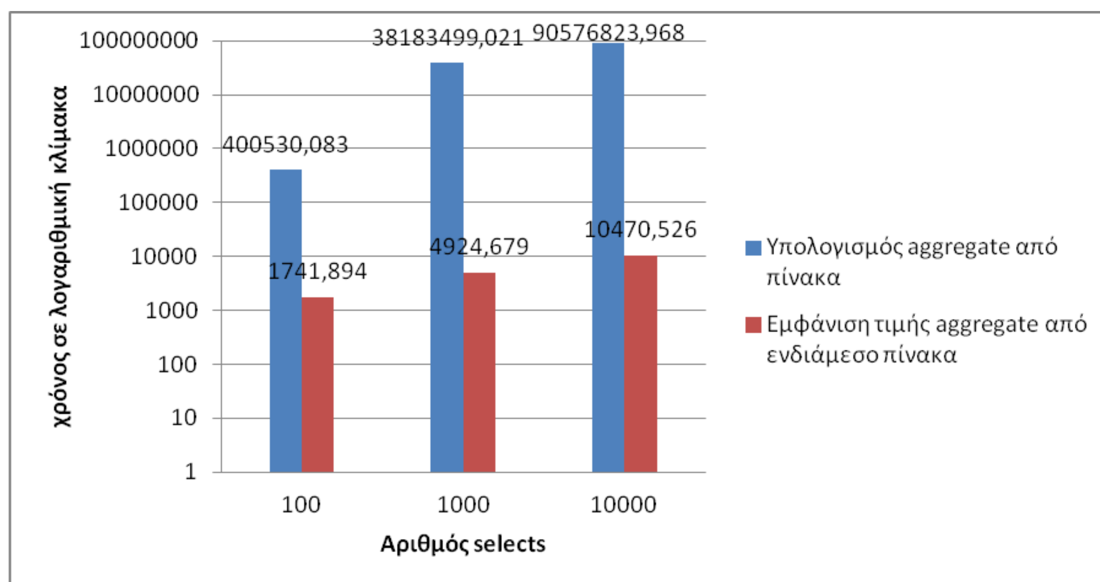
Η παραπάνω διαδικασία πραγματοποιήθηκε για ένα εκατομμύριο inserts συνολικό όγκο δεδομένων σε πίνακα 90 εκατομμυρίων γραμμών και 100,1000 και 10000 select queries και τα αποτελέσματα φαίνονται στον πίνακα 13.

sel/ins	insert	Σύστημα (ms)	Πείραμα (ms)	select	Σύστημα (ms)	Πείραμα (ms)
1/10000	1000000	5,934,889.114	6,938,690.484	100	400,530.083	1,741.894
1/1000	1000000	5,834,643.709	7,384,776.299	1000	38,183,499.021	4,924.679
1/100	1000000	6,860,758.277	6,650,363.200	10000	90,576,823.968	10,470.526

Πίνακας 13. Αποτελέσματα μετρήσεων με χρήση και μη ενδιάμεσων πινάκων για υπολογισμό aggregates.

Παρατηρούμε ότι το ένα εκατομμύριο inserts ή inserts/updates πραγματοποιείται και στις δύο περιπτώσεις σε πολύ κοντινούς χρόνους. Αντίθετα μεγάλη διαφοροποίηση παρατηρείται στα selects. Να σημειωθεί ότι για 10000 selects η τιμή που φαίνεται στον πίνακα είναι η βίαιη διακοπή του query το οποίο δεν ολοκληρώθηκε ποτέ. Επίσης τρέξαμε στον ενδιάμεσο πίνακα την ίδια συνάρτηση και για 100000 selects, που εκτελέστηκαν σε 31,941.584 ms. Από το παραπάνω πείραμα γίνεται φανερό ότι όσο συχνότερα είναι τα selects τόσο περισσότερο μας συμφέρει να φτιάξουμε στη βάση μας ενδιάμεσους πίνακες. Σε περίπτωση που όλα τα ερωτήματα του χρήστη αναφέρονται σε ενδιάμεσους πίνακες που καλύπτουν διάφορες

ανάγκες η τεχνική που περιγράψαμε είναι αποδοτική. Αντίθετα αν οι ανάγκες του συστήματος απαιτούν πρόσβαση στα συνολικά δεδομένα τότε η απουσία index από τον πίνακα που γίνονται όλα τα inserts αποτελεί σημαντικό πρόβλημα. Παρακάτω παρουσιάζεται η γραφική αναπαράσταση των αποτελεσμάτων σε σύγκριση με την κανονική λειτουργία του συστήματος χωρίς τη χρήση ενδιάμεσων πινάκων.



Γράφημα 1. Λειτουργία του συστήματος με ή χωρίς τη χρήση ενδιάμεσων πινάκων

7.3.3 Λεπτομέρειες

Για την δημιουργία των παραπάνω πινάκων είναι εμφανές ότι δεν χρειάζεται να τρέξουμε την synartisi3. Κάτι τέτοιο έγινε πολύ πιο απλά με τις παρακάτω εντολές. Η συνάρτηση αυτή απλά μας χρησίμευσε ως επαλήθευση, για να επιβεβαιώσουμε ότι παίρνουμε τα σωστά αποτελέσματα από την synartisi2. Επίσης, το σύστημα μας δεν είναι online χρησιμοποιούμε απλά ένα μέρος αποθηκευμένων δεδομένων που μας έχει παραχωρήσει η εταιρεία επομένως οι συναρτήσεις μας δεν λειτουργούν για CURRENT_TIME του συστήματος. Για να προσομοιώσουμε την παραπάνω πειραματική διαδικασία χρησιμοποιήσαμε τα χρονικά διαστήματα που φαίνονται στα queries δημιουργίας των πινάκων.

current_day:

```
INSERT INTO current_day
SELECT measurement_source_id,
       SUM(measurement_value::real),
       AVG(measurement_value::real),
       COUNT(measurement_source_id)
FROM measurement_events
WHERE measurement_time >= '2011-06-24 00:00:00+03'
  AND measurement_time<='2011-06-24 23:59:59+03'
GROUP BY measurement_source_id;
```

Πίνακας 14. Query για τη δημιουργία του πίνακα current_day

current_week:

```
INSERT INTO current_week
SELECT measurement_source_id,
       SUM(measurement_value::real),
       AVG(measurement_value::real),
       COUNT(measurement_source_id)
FROM measurement_events
WHERE measurement_time >= '2011-06-20 00:00:00+03'
  AND measurement_time<='2011-06-26 23:59:59+03'
GROUP BY measurement_source_id;
```

Πίνακας 15. Query για τη δημιουργία του πίνακα current_week

current_month:

```
INSERT INTO current_month
SELECT measurement_source_id,
       SUM(measurement_value::real),
       AVG(measurement_value::real),
       COUNT(measurement_source_id)
FROM measurement_events
WHERE measurement_time >= '2011-06-01 00:00:00+03'
  AND measurement_time<='2011-06-30 23:59:59+03'
GROUP BY measurement_source_id;
```

Πίνακας 16. Query για τη δημιουργία του πίνακα current_month

Επίσης, παραπάνω αναφέραμε ότι στους ενδιάμεσους πίνακες πρέπει να γίνεται αρχικοποίηση. Το εργαλείο που προτείνουμε να χρησιμοποιείται για το σκοπό αυτό είναι το Crontab. **[13]** Είναι ένα πρόγραμμα φτιαγμένο για unix-like λειτουργικά συστήματα που χρησιμοποιείται για τον προγραμματισμό εργασιών που θέλουμε να εκτελούνται περιοδικά. Για να δημιουργήσουμε μια τέτοια εργασία, γράφουμε την εντολή sudo crontab -e. Με την

εντολή αυτή μπαίνουμε στον προεπιλεγμένο editor όπου μπορούμε να γράψουμε την πρώτη εργασία. Αν ξαναγράψουμε στο μέλλον την εντολή αυτή, θα έχει την εργασία αυτή και μπορούμε να προσθέσουμε κι άλλες στις επόμενες γραμμές.

Μία γραμμή του αρχείου έχει ως εξής:

```
* * * * * /execute/this/script.sql
```

Όπως βλέπουμε υπάρχουν 5 αστέρια. Το καθένα από αυτά αντιπροσωπεύει αντίστοιχα:

-λεπτό (από 0 ως 59)

-ώρα (από 0 ως 23)

-μέρα του μήνα (από 1 ως 31)

-μήνα (από 1 ως 12)

-μέρα της εβδομάδας (από 0 ως 6, με 0 την Κυριακή)

Όταν κάποια παράμετρος είναι αστέρι, η εντολή θα εκτελείται σε κάθε τιμή που μπορεί να πάρει η παράμετρος αυτή. Δηλαδή όταν είναι όλες αστέρια η εντολή εκτελείται κάθε λεπτό.

Υπάρχουν επίσης μερικές μεταβλητές που μπορεί να χρησιμοποιηθούν αντί για τα παραπάνω:

`@reboot` Run once, at startup

`@yearly` Run once a year `0 0 1 1 *`

`@annually` (same as @yearly)

`@monthly` Run once a month `0 0 1 * *`

`@weekly` Run once a week `0 0 * * 0`

`@daily` Run once a day `0 0 * * *`

`@midnight` (same as @daily)

`@hourly` Run once an hour `0 * * * *`

Στην περίπτωσή μας λοιπόν μπορούμε να έχουμε το εξής αρχείο χρονοπρογραμματισμού:

```
@daily /DIRECTORY/script1.sql
@weekly /DIRECTORY/script2.sql
@monthly /DIRECTORY/script3.sql
```

Αν τώρα θέλουμε να αποθηκεύουμε το αποτέλεσμα των script σε κάποιο logfile ώστε να ξέρουμε αν έχει συμβεί κάποιο λάθος, προσθέτουμε τα εξής μετά το τέλος κάθε εντολής:

```
>> /var/log/script1_output.log 2>&1
>> /var/log/script2_output.log 2>&1
>> /var/log/script3_output.log 2>&1
```

Με αυτόν τον τρόπο λέμε στα linux να γράψουν το STDOUT στα αρχεία αυτά (με το σύμβολο >>) και να εμφανίσουν το STDERR (2) όπου εμφανίζεται και το STDOUT (1).

Ο κώδικας των script φαίνεται παρακάτω.

```
psql mydb -p 5432 -c "TRUNCATE current_day; "
```

Πίνακας 17. script1.sql

```
psql mydb -p 5432 -c "TRUNCATE current_week; "
```

Πίνακας 18. script2.sql

```
psql mydb -p 5432 -c "TRUNCATE current_month; "
```

Πίνακας 19. script3.sql

7.4 In-Memory Tables

7.4.1 Θεωρία

Τα in-memory tables είναι στην ουσία ενδιάμεσοι πινάκες όμοιοι με αυτούς που περιγράψαμε στο προηγούμενο κεφάλαιο με την μόνη διαφορά ότι είναι αποθηκευμένοι στη RAM για αποφυγή του χρόνου επικοινωνίας δίσκου-μνήμης και μείωση φόρτου εργασίας του δίσκου, πράγμα που βελτιώνει σημαντικά το χρόνο εκτέλεσης των ερωτημάτων.

Ακολουθούν αναλυτικά τα βήματα που ακολουθήσαμε για να φτιάξουμε τους πίνακες αυτούς στη μνήμη RAM **[14]**:

1. Για να δημιουργήσουμε ένα filesystem στην μνήμη RAM χρησιμοποιούμε την εντολή:

   ```
   # mount -t ramfs none /mnt/ramfs
   ```

   Όλα τα αρχεία αποθηκεύονται στη RAM. Έχουμε δικαίωμα για read και write. Δεν υπάρχει κάποιος περιορισμός για το πόσο χώρο θα χρησιμοποιήσουμε- το filesystem αυξομειώνεται για να χωρέσει όλα τα αρχεία που περιλαμβάνει. Σε περίπτωση επανεκκίνησης του συστήματος το filesystem χάνει όλα του τα περιεχόμενα.

2. Δημιουργούμε το directory όπου θα αποθηκεύσουμε το tablespace, το οποίο πρέπει να βρίσκεται στο ramfs filesystem.

   ```
   # mkdir /mnt/ramfs/pgdata
   # chown postgres:postgres /mnt/ramfs/pgdata
   # chmod go-rwx /mnt/ramfs/pgdata
   ```

3. Για τη δημιουργία του tablespace και την παραχώρηση των δικαιωμάτων του στον χρήστη της PostgreSQL που χρησιμοποιούμε, τρέξαμε τις παρακάτω εντολές αφού συνδεθήκαμε πρώτα στη βάση μας με psql mydb :

```
mydb=# CREATE TABLESPACE tablespace_name LOCATION
'/mnt/ramfs/pgdata';
mydb=# GRANT CREATE ON TABLESPACE tablespace_name TO user_name;
```

7.4.2 Πείραμα

7.4.2.1 Περιγραφή

Χρησιμοποιήθηκαν ακριβώς οι ίδιες συναρτήσεις που περιγράφηκαν στην ενότητα Ενδιάμεσοι Πίνακες.

7.4.2.2 Μετρήσεις

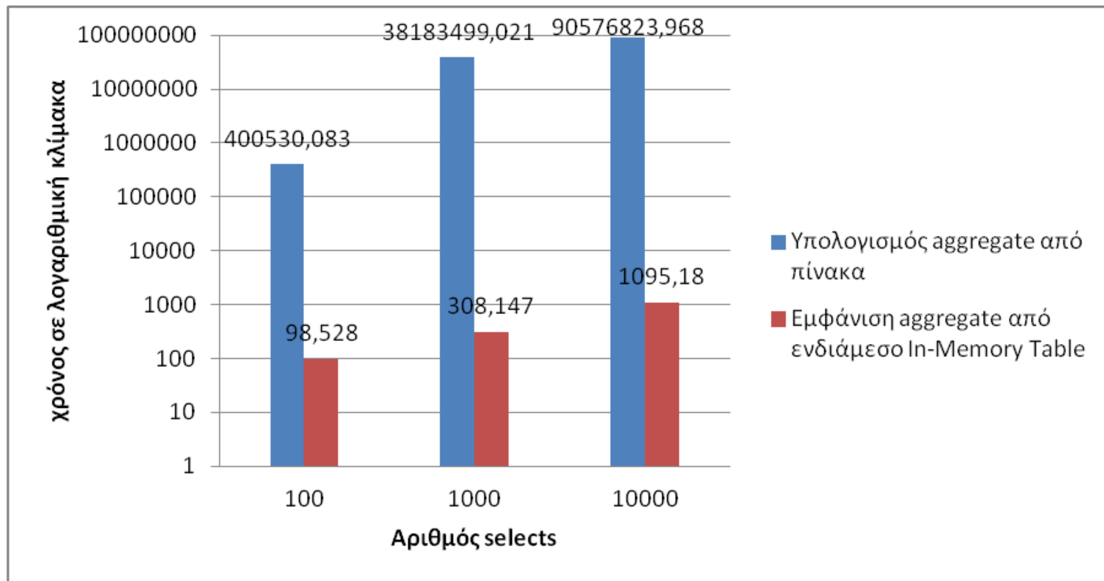
Πραγματοποιήθηκαν οι ίδιες ακριβώς μετρήσεις με την ενότητα Ενδιάμεσοι Πίνακες, οποίες παρουσιάζονται στον παρακάτω πίνακα.

sel/ins	insert	in memory	select	in memory
1-10000	1000000	2,524,853.806 ms	100	98.528 ms
1-1000	1000000	3,379,966.464 ms	1000	308.147 ms
1-100	1000000	3,271,187.782 ms	10000	1,095.180 ms
1-10	1000000	2,870,683.677 ms	100000	10,400.629 ms

Πίνακας 20. Αποτελέσματα μετρήσεων με χρήση ενδιάμεσων, in-memory πινάκων.

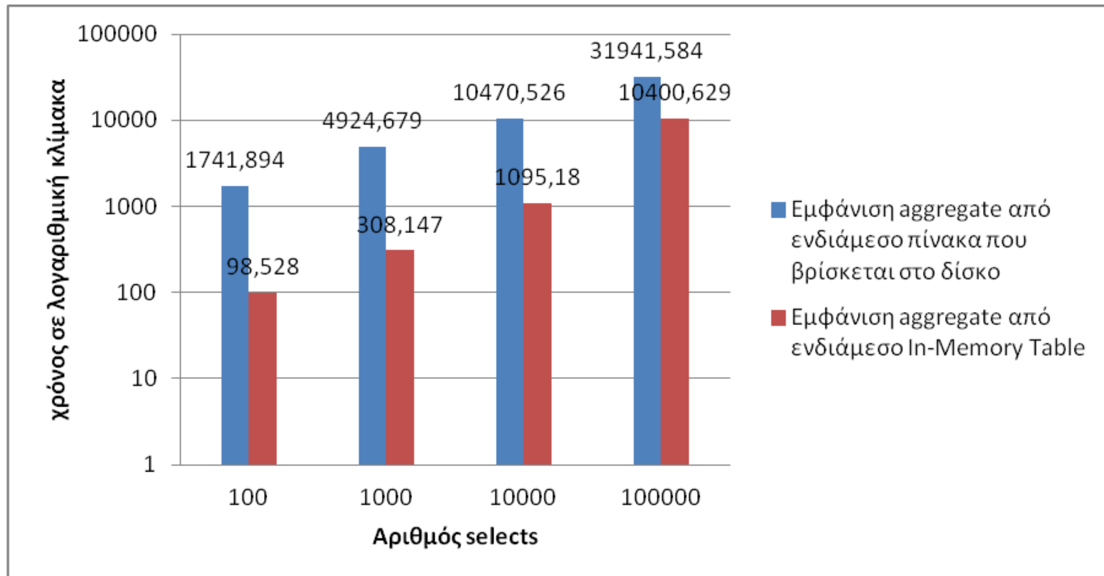
Από τα αποτελέσματα που πήραμε αξίζει να μείνουμε στην μεγάλη ταχύτητα με την οποία βλέπουμε ότι γίνονται τα inserts/updates στον αντίστοιχο ενδιάμεσο πίνακα που βρίσκεται στην μνήμη RAM. Πρόκειται για ακριβώς μισό χρόνο από αυτόν που έκαναν τα Inserts είτε στον κανονικό πίνακα του συστήματος είτε στον ενδιάμεσο πίνακα που ήταν αποθηκευμένος στο δίσκο. Όπως είδαμε στην προηγούμενη ενότητα δεν λάβαμε καθόλου υπόψη μας το χρόνο που έκαναν τα inserts και επικεντρωθήκαμε στην σύγκριση των selects. Εδώ παρόλο που και τα selects γίνονται γρηγορότερα, το ουσιαστικό κέρδος προκύπτει από την τεράστια διαφορά που παρατηρείται στα inserts.

Παρακάτω παρουσιάζεται η γραφική αναπαράσταση των αποτελεσμάτων σε σύγκριση με την κανονική λειτουργία τους συστήματος χωρίς τη χρήση ενδιάμεσων πινάκων.



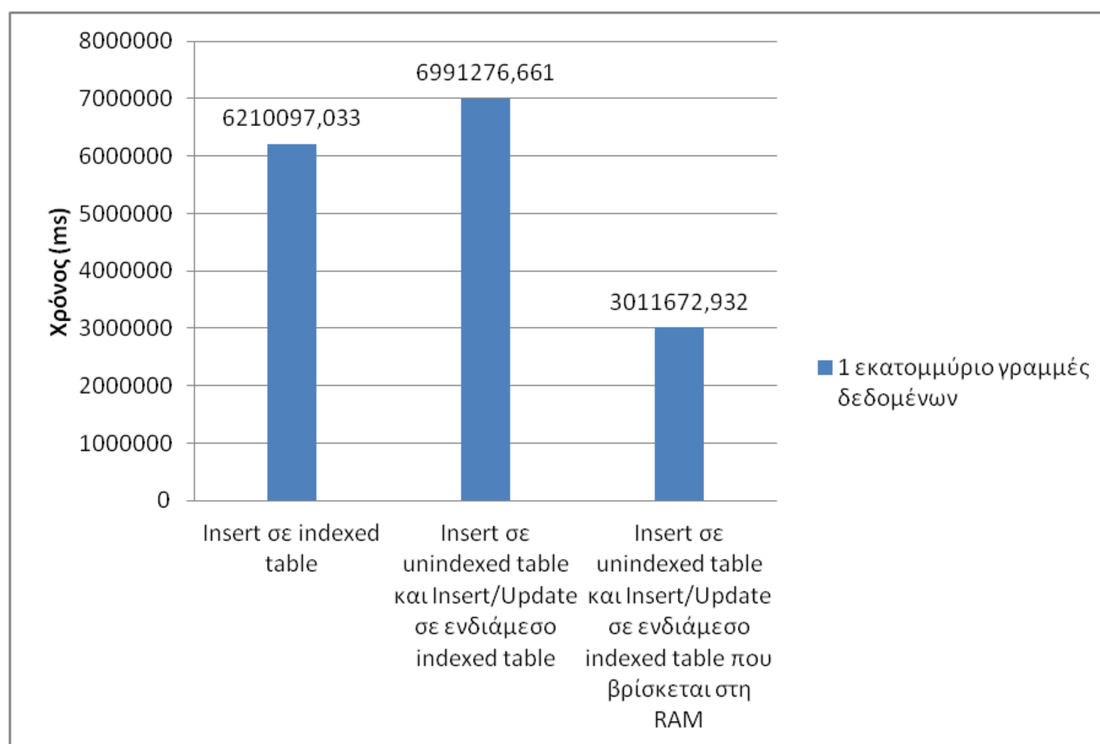
Γράφημα 2. Λειτουργία του συστήματος με ή χωρίς τη χρήση ενδιάμεσων in memory πινάκων

Παρακάτω παρουσιάζεται η γραφική σύγκριση των ενδιάμεσων πινάκων που βρίσκονται στη RAM με αυτούς που βρίσκονται στο δίσκο.



Γράφημα 3. Λειτουργία του συστήματος με τη χρήση ενδιάμεσων πινάκων στο δίσκο και στη μνήμη RAM.

Παρακάτω παρουσιάζουμε γραφικά τη διαφορά εκτέλεσης των inserts για τις τρεις μεθόδους που εξετάσαμε παραπάνω, σύστημα, ενδιάμεσοι πίνακες στο δίσκο και ενδιάμεσοι πίνακες στη RAM.



Γράφημα 4. Συγκεντρωτικό διάγραμμα σύγκρισης στο χρόνο εκτέλεσης των inserts

7.4.3 Λεπτομέρειες

Βασικό μειονέκτημα της μεθόδου αυτής είναι ότι σε ενδεχόμενο restart του server ή κατάρρευση του συστήματος το tablespace χάνεται και κατ' επέκταση όλα τα δεδομένα που είναι αποθηκευμένα σ'αυτό. Στη συνέχεια θα πρέπει να ξαναδημιουργηθεί από την αρχή και να φορτωθούν ξανά τα δεδομένα. Για άμεση ανάκτηση του συστήματος θα πρέπει να τρέχουμε τις παρακάτω εντολές ανά κάποια inserts (π.χ. ένα εκατομμύριο) ώστε να δημιουργείται ένα αντίγραφο ασφαλείας.

```
DROP TABLE table_backup; --για να σβηστεί το προηγούμενο backup
SELECT * INTO table_ backup FROM in_memory_table;
```

Οι εντολές αυτές έχουν χρόνο εκτέλεσης περίπου 2 δευτερόλεπτα(2,132.021 ms) που είναι μηδαμινός χρόνος και δεν εμποδίζει τις υπόλοιπες διαδικασίες.

7.5 Partitioning

7.5.1 Θεωρία

Η τεχνική του partitioning υπαγορεύει το διαχωρισμό ενός μεγάλου πίνακα σύμφωνα με συγκεκριμένα κριτήρια για καλύτερη διαχείριση μεγάλου όγκου δεδομένων. Παρακάτω περιγράφουμε τις κατηγορίες partitioning που υπάρχουν.

Το **Horizontal partitioning** είναι ο διαχωρισμός ενός πίνακα σε υποπίνακες ίδιας δομής (στήλες,indexes,constraints) βάσει κάποιας λογικής ιδιότητας που περιγράφει τα δεδομένα. Έτσι ομαδοποιούμε τα δεδομένα σε μικρότερους πίνακες και οι ερωτήσεις πλέον που αφορούν τα συγκεκριμένα δεδομένα «πέφτουν» πάνω στους μικρότερους πίνακες και δεν είναι επιβαρυμένες με κάποιο σύνθετο where-clause που καθυστερεί το χρόνο εκτέλεσης.

Το **Database Shard** είναι ένα horizontal partitioning της βάσης με τη διαφορά ότι οι υποπίνακες που προκύπτουν βρίσκονται σε διαφορετικούς servers πράγμα που βελτιώνει κι άλλο την απόδοση του συστήματος.

Στο **Vertical partitioning** δημιουργούμε ένα νέο πίνακα με λιγότερες στήλες από τον αρχικό κρατώντας αυτές που χρησιμοποιούνται συχνότερα στα queries και αποθηκεύουμε τις υπόλοιπες σε κάποιον άλλο πίνακα. Με τον τρόπο αυτό αποφεύγουμε την περιττή πληροφορία που αυξάνει το χρόνο εκτέλεσης. **[15]**

Στην PostgreSQL μπορούν να επιτευχθούν οι εξής δύο τύποι partitioning:

Διαχωρισμός διαστημάτων (range partitioning): Ο πίνακας διαχωρίζεται ανά διαστήματα που ορίζονται από μια στήλη-κλειδί ή μία ομάδα στηλών, χωρίς να υπάρχει επικάλυψή τους. Για παράδειγμα κάποιος μπορεί να διαχωρίσει έναν πίνακα με βάση χρονικά διαστήματα (ημερομηνίες), ή διαστήματα γνωρισμάτων για συγκεκριμένα αντικείμενα.

Διαχωρισμός λίστας (list partitioning): Ο πίνακας διαχωρίζεται ορίζοντας ρητά ποιες τιμές εμφανίζονται σε κάθε υποπίνακα. **[16]**

7.5.2 Πείραμα

7.5.2.1 Περιγραφή

Στην περίπτωση μας θεωρήσαμε ότι θα ήταν σκόπιμο να εφαρμόσουμε horizontal partitioning στον πίνακα measurement_events χωρίζοντας τις μετρήσεις σύμφωνα με τη στήλη measurement_time. Κάθε υποπίνακας θα περιέχει τα δεδομένα που αφορούν ένα εικοσιτετράωρο. Η απόφαση αυτή προέκυψε από το γεγονός ότι τα δεδομένα που γίνονται insert στη βάση είναι πάντα «μεγαλύτερης» ημερομηνίας από τα ήδη υπάρχοντα δεδομένα

στον πίνακα. Έτσι στην ουσία τα inserts γίνονται μόνο στον τελευταίο partition του πίνακα. Το παραπάνω σχέδιο υλοποιήθηκε με τη βοήθεια ορισμένων συναρτήσεων που θα περιγραφούν παρακάτω.

```
CREATE OR REPLACE FUNCTION synartisi5() RETURNS void
    LANGUAGE plpgsql
    AS $$DECLARE
        archiveTables text[];
        masterTable text;
        archiveTable text;
        rec abstime;
        tablename text;
        start_time date;
        end_time date;
        create_line text;
        alter_line text;
        index_line text;
        insert_line text;
        truncate_line text;
BEGIN
        -- Table to create partitions for
        masterTable:='measurement_events';
        FOR rec IN
       (SELECT (EXTRACT(epoch FROM
date_trunc('day',measurement_time::abstime)::int::abstime) AS DAY
        FROM ONLY measurement_events
        GROUP BY DAY
        ORDER BY DAY) LOOP
        SELECT to_char(rec,'ddmmyyyy') INTO tablename;
        SELECT rec::date INTO start_time;
        SELECT (rec + INTERVAL '1 day')::date INTO end_time;
        -- Child table
        archiveTable:=masterTable || '_' || tablename;
        -- Checking if table is already created
        BEGIN
        EXECUTE 'SELECT * FROM ONLY ' || archiveTable || ' LIMIT 1';
        EXCEPTION WHEN UNDEFINED_TABLE THEN
        create_line:='CREATE TABLE ' || archiveTable || '(check (
measurement_time >=''' || start_time || ''' and measurement_time <'''
|| end_time || '''),like ' || masterTable || ' including defaults
including storage) with oids';
```

```
        alter_line:='ALTER TABLE ' || archiveTable || ' inherit ' ||
masterTable || '';
        index_line:='CREATE INDEX ' || archiveTable || '_1 on ' ||
archiveTable || '(measurement_source_id,measurement_time)';
        RAISE NOTICE '--- DAY %', rec;
        RAISE NOTICE '--- % Create: %', masterTable, create_line;
        EXECUTE create_line;
        RAISE NOTICE '--- % Alter: %', masterTable, alter_line;
        EXECUTE alter_line;
        RAISE NOTICE '--- % Index: %', masterTable, index_line;
        EXECUTE index_line;
        END;
        insert_line:='INSERT INTO ' || archiveTable || ' SELECT *
FROM ONLY ' || masterTable || ' WHERE measurement_time >=''' ||
start_time || ''' AND measurement_time <''' || end_time||'''';
        RAISE NOTICE '--- % Insert: %', masterTable, insert_line;
        EXECUTE insert_line;
        END LOOP;
        -- Truncating master table
        truncate_line:='TRUNCATE TABLE ONLY ' || masterTable;
        RAISE NOTICE '--- % Truncate: %', masterTable, truncate_line;
        EXECUTE truncate_line;
END$$;
```

Συνάρτηση 5 - Κώδικας plpgsql

Λειτουργία Συνάρτησης 5 :

Η synartisi5 δημιουργεί τα partitions ανά μέρα στον πίνακα measurement_events. Με χρήση κατάλληλων time functions που υπάρχουν έτοιμες στην PostgreSQL και απαραίτητων μετατροπών στους τύπους των δεδομένων αυτοματοποιήσαμε πλήρως τη διαδικασία δημιουργίας partitions με βάση τη στήλη measurement_time. Πιο αναλυτικά, για κάθε διαφορετική ημερομηνία που εμφανίζεται στη στήλη αυτή δημιουργήσαμε έναν ξεχωριστό πίνακα με όνομα που συντίθεται από το όνομα του αρχικού πίνακα, τον χαρακτήρα '_' και ένα οκταψήφιο νούμερο που υποδηλώνει την ακριβή ημερομηνία που καλύπτει ο συγκεκριμένος πίνακας, για παράδειγμα το partition measurement_events_14062011 περιέχει όλες τι γραμμές για τις οποίες το measurement_time έχει την ημερομηνία 14-06-2011. Ο πίνακας αυτός «κληρονομεί» όλες τις ιδιότητες του αρχικού πίνακα. Δημιουργούμε το κατάλληλο index που στην περίπτωσή μας είναι ο συνδυασμός (measurement_source_id, measurement_time). Στη συνέχεια, για κάθε ξεχωριστή ημερομηνία που βρήκαμε, εισάγουμε τα κατάλληλα δεδομένα στο αντίστοιχο partition. Επαναλαμβάνουμε την ίδια διαδικασία για όλες τις ημερομηνίες. Στο τέλος αδειάζουμε τον αρχικό πίνακα. Καλώντας την synartisi5

δημιουργούνται τα partitions των δεδομένων που είναι ήδη καταχωρημένα σε ένα πίνακα. Για τη δημιουργία μελλοντικών partitions που περιμένουμε ότι θα χρειαστούν αφού το σύστημά μας είναι online, συνεχώς τροφοδοτούμενο από νέα δεδομένα μπορούμε να χρησιμοποιήσουμε την synartisi6 που παρουσιάζεται παρακάτω και με αλλαγή στον αριθμό N να δημιουργήσουμε όσα «παρακάτω» partitions θέλουμε, θεωρώντας πάντα ότι έχουμε ήδη δημιουργήσει όλα τα partitions μέχρι και την τρέχουσα ημέρα.

```
CREATE FUNCTION synartisi6() RETURNS void
    LANGUAGE plpgsql
    AS $$
DECLARE
    i integer;
    query1 text;
    query2 text;
    query3 text;
    tablename text;
    index_tablename text;
    alter_tablename text;
    next_partition text;
    current_check date;
    next_check date;

BEGIN
   FOR i IN 1..N LOOP
     query1:= 'SELECT EXTRACT(epoch FROM
date_trunc(''day'',CURRENT_TIMESTAMP + INTERVAL '''||i||'
day''))::int::abstime::date';
     query2:= 'SELECT EXTRACT(epoch FROM
date_trunc(''day'',CURRENT_TIMESTAMP + INTERVAL '''||i+1||'
day''))::int::abstime::date';
     query3:= 'SELECT to_char(CURRENT_TIMESTAMP + INTERVAL '''||i||'
day'', ''ddmmyyyy'')';
     EXECUTE query1 INTO current_check;
     EXECUTE query2 INTO next_check;
     EXECUTE query3 INTO next_partition;
     tablename:='create table measurement_events_'|| next_partition
||'(check ( measurement_time::date >= ''' || current_check || ''' and
measurement_time::date < ''' || next_check || '''),like ind_2
including defaults including storage) with oids';
     alter_tablename:='alter table  measurement_events_' ||
next_partition || ' inherit measurement_events';
     index_tablename:='create index measurement_events_' ||
next_partition || '_1 on measurement_events_' || next_partition ||
'(measurement_source_id,measurement_time)';
     EXECUTE tablename;
     EXECUTE alter_tablename;
     EXECUTE index_tablename;
   END LOOP;
END$$;
```

Συνάρτηση 6 - Κώδικας plpgsql

Η παραπάνω συνάρτηση δεν μας χρησίμευσε στο πείραμά μας διότι εμείς δεν έχουμε μελλοντικά inserts αφού δουλεύουμε σε σταθερό όγκο δεδομένων, όμως την γράψαμε και τη δοκιμάσαμε για να έχουμε μία συνολική εικόνα και να κάνουμε μία πιο ολοκληρωμένη προσέγγιση της πραγματικής λειτουργίας του συστήματος.

Για να ολοκληρώσουμε την διαδικασία του partitioning δεν αρκεί μόνο η δημιουργία των υποπινάκων, πρέπει να ορίσουμε έναν κανόνα που θα στέλνει τα inserts στο αντίστοιχο partition χωρίς να τα βάζει στον αρχικό πίνακα. Η παραπάνω λειτουργία υλοποιήθηκε με τη βοήθεια ενός trigger πάνω στον πίνακα measurement_events το οποίο κάθε φορά που εκτελείται η εντολή INSERT στον πίνακα αυτόν θα εκτελεί την synartisi7 που θα εισάγει την γραμμή δεδομένων στον αντίστοιχο υποπίνακα. Ακολουθούν η εντολή δημιουργίας του trigger καθώς και ο κώδικας της synartisi7.

```
CREATE TRIGGER insert_measurement_events_trigger
BEFORE
INSERT ON measurement_events
FOR EACH ROW EXECUTE PROCEDURE synartisi6();
```

Πίνακας 21. Εντολή για τη δημιουργία του trigger στον πίνακα που έχει γίνει partitioning

```
CREATE OR REPLACE FUNCTION synartisi7() RETURNS trigger
    LANGUAGE plpgsql
    AS $$
        DECLARE
                insert_sql text;
         BEGIN
                 insert_sql:= 'INSERT INTO measurement_events_' ||
to_char((NEW.measurement_time)::date,'ddmmyyyy') || '
(measurement_location_id, measurement_source_id, measurement_time,
event_reception_time, measurement_value, quality, source_id,
transducer_id,aa) VALUES ' || '(''' || NEW.measurement_location_id ||
''',''' || NEW.measurement_source_id || ''',''' ||
NEW.measurement_time || ''',''' || NEW.event_reception_time || ''','
|| NEW.measurement_value ||',' || NEW.quality || ',''' ||
NEW.source_id || ''',''' || NEW.transducer_id || ''')';
                EXECUTE insert_sql;
        RETURN NULL;
        END$$;
```

Συνάρτηση 7 - Κώδικας plpgsql

Εφαρμόζοντας την τεχνική του partitioning στον πίνακα measurement_events θα μετρήσουμε την ώρα που κάνουν να εισαχθούν τα δεδομένα μίας ολόκληρης μέρας. Θα αναλύσουμε τις διαφορές πριν και μετά το partitioning καθώς και το κατά πόσο επηρεάζει το indexing του πίνακα και στις δύο περιπτώσεις. Οι παρατηρήσεις και τα αποτελέσματα των παραπάνω παρουσιάζονται στην ενότητα Μετρήσεις.

7.5.2.2 *Μετρήσεις*

Αρχικά εισάγαμε τα δεδομένα μίας μέρας στον measurement_events χωρίς να έχουμε αλλάξει κάτι από την αρχική του μορφή και με index στον συνδυασμό στηλών (measurement_source_id, measurement_time). Για τα inserts χρησιμοποιήθηκε η παρακάτω εντολή :

```
INSERT INTO measurement_events
SELECT *
FROM day;
```

όπου day ένας βοηθητικός πίνακας που περιέχει τα δεδομένα μίας ολόκληρης μέρας, δηλαδή περίπου 8 εκατομμύρια γραμμές.

Έπειτα, μετά την εφαρμογή των συναρτήσεων για το partitioning που περιγράψαμε παραπάνω, εισάγαμε τα ίδια δεδομένα στον measurement_events χρησιμοποιώντας ακριβώς την ίδια εντολή με αποτέλεσμα να μπουν τα δεδομένα σε ένα εντελώς κενό partition του πίνακα, αφού πρόκειται για μία ολόκληρη μέρα, το οποίο είχε ίδιο index με τον αρχικό πίνακα. Τέλος επαναλάβαμε την τελευταία μέτρηση έχοντας σβήσει το index από το συγκεκριμένο partition του πίνακα. Στον παρακάτω πίνακα φαίνονται τα αποτελέσματα.

Μέθοδος	**Χρόνος εκτέλεσης**
Χωρίς Partitioning με Index	49,747,483.011 ms
Με partitioning με Index	30,348,488.101 ms
Με partitioning χωρίς index στον υποπίνακα προορισμού	1,341,839.738 ms

Πίνακας 22. Χρόνοι εκτέλεσης για την εισαγωγή δεδομένων μίας μέρας

Από τον παραπάνω πίνακα φαίνεται ξεκάθαρα ότι τα inserts επιταχύνονται σημαντικά όταν χρησιμοποιούμε έναν διαχωρισμένο πίνακα και κυρίως όταν το partition table προορισμού δεν έχει index. Όταν περάσει η τρέχουσα μέρα κάνουμε build το index του συγκεκριμένου partition με την εντολή

```
CREATE INDEX measurement_events_ddmmyyyy_1 ON
measurement_events_ddmmyyyy USING btree (measurement_source_id,
measurement_time) ;
```

Πίνακας 23. Εντολή για δημιουργία index στον υποπίνακα της τρέχουσας μέρας

η οποία εκτελείται σε 114,475.209 ms για τα περίπου 8 εκατομμύρια γραμμές που αντιστοιχούν στα δεδομένα μίας μέρας. Το γεγονός ότι κάθε γραμμή δεδομένων που αποθηκεύεται στη βάση ακολουθεί χρονολογική σειρά μας επιτρέπει εφαρμόσουμε την παραπάνω διαδικασία, γιατί μας εξασφαλίζει ότι όλα τα δεδομένα που εισέρχονται στο σύστημα σε μία ημέρα αφορούν την ίδια την ημέρα και επομένως εισάγονται στο partition που φέρει την τρέχουσα ημερομηνία.

Επίσης θα θέλαμε να εξετάσουμε κατά πόσο το partitioning επιταχύνει τα queries που έχουν στο where-clause τους τη στήλη measurement_time. Να σημειωθεί σε αυτό το σημείο ότι για να εκμεταλλευτούμε τις ιδιότητες του partitioning ως προς τα selects έπρεπε να επέμβουμε στο αρχείο postgresql.conf της PostgreSQL, όπου αλλάξαμε την παράμετρο constraint_exclusion από off σε on. Έτσι κάθε query που θα εκτελείται πάνω σε κάποιο πίνακα με partition θα εξετάζει τα constraints των υποπινάκων και θα επιλέγει με αυτόν τον τρόπο μόνο τα υποσύνολα δεδομένων, όπου η συνθήκη του where-clause συμπίπτει με αυτή του check του αντίστοιχου υποπίνακα. Με τον τρόπο αυτό παρακάμπτεται μεγάλο ποσοστό του όγκου δεδομένων που δεν αφορούν το query και υπό άλλες συνθήκες θα καθυστερούσαν κατά πολύ την εκτέλεσή του. Αυτή η διαδικασία θα τεκμηριωθεί ακριβώς παρακάτω και από την εντολή EXPLAIN της PostgreSQL η οποία περιγράφει τα βήματα που γίνονται για να αποτιμηθεί μία εντολή.

```
set constraint_exclusion=on
```

QUERY PLAN
HashAggregate (cost=428031.02..428033.52 rows=200 width=27)
-> Append (cost=0.00..287955.14 rows=8004336 width=27)
-> Seq Scan on ind_2 (cost=0.00..0.00 rows=1 width=30)
Filter: ((measurement_time >= '2011-06-14 00:00:00+03'::timestamp with time zone) AND (measurement_time < '2011-06-15 00:00:00+03'::timestamp with time zone))
-> Seq Scan on ind_2_14062011 ind_2 (cost=0.00..287955.14 rows=8004335 width=27)
Filter: ((measurement_time >= '2011-06-14 00:00:00+03'::timestamp with time zone) AND (measurement_time < '2011-06-15 00:00:00+03'::timestamp with time zone))

6 row(s)

Total runtime: 33.169 ms

SQL executed.

Εικόνα 3. Query Plan για ερώτημα που ελέγχει το check constraint των υποπινάκων

```
set constraint_exlusion=off
```

QUERY PLAN
HashAggregate (cost=726420.44..726422.94 rows=200 width=27)
-> Append (cost=0.00..586344.44 rows=8004343 width=27)
-> Seq Scan on ind_2 (cost=0.00..0.00 rows=1 width=30)
Filter: ((measurement_time >= '2011-06-14 00:00:00+03'::timestamp with time zone) AND (measurement_time < '2011-06-15 00:00:00+03'::timestamp with time zone))
-> Seq Scan on ind_2_14062011 ind_2 (cost=0.00..287955.14 rows=8004335 width=27)
Filter: ((measurement_time >= '2011-06-14 00:00:00+03'::timestamp with time zone) AND (measurement_time < '2011-06-15 00:00:00+03'::timestamp with time zone))
-> Index Scan using ind_2_18062011_1 on ind_2_18062011 ind_2 (cost=0.00..42848.44 rows=1 width=30)
Index Cond: ((measurement_time >= '2011-06-14 00:00:00+03'::timestamp with time zone) AND (measurement_time < '2011-06-15 00:00:00+03'::timestamp with time zone))

...

QUERY PLAN
Index Cond: ((measurement_time >= '2011-06-14 00:00:00+03'::timestamp with time zone) AND (measurement_time < '2011-06-15 00:00:00+03'::timestamp with time zone))
-> Index Scan using ind_2_02102013_1 on ind_2_02102013 ind_2 (cost=0.00..8.76 rows=1 width=434)
Index Cond: ((measurement_time >= '2011-06-14 00:00:00+03'::timestamp with time zone) AND (measurement_time < '2011-06-15 00:00:00+03'::timestamp with time zone))
-> Index Scan using ind_2_03102013_1 on ind_2_03102013 ind_2 (cost=0.00..8.76 rows=1 width=434)
Index Cond: ((measurement_time >= '2011-06-14 00:00:00+03'::timestamp with time zone) AND (measurement_time < '2011-06-15 00:00:00+03'::timestamp with time zone))

20 row(s)

Total runtime: 2.755 ms

SQL executed.

Εικόνα 4. Query Plan για ερώτημα που δεν ελέγχει το check constraint των υποπινάκων

```
SELECT measurement_source_id,
       SUM (measurement_value::real), AVG(measurement_value::real)
FROM table_i
WHERE measurement_time>= '2011-06-14'
  AND measurement_time < '2011-06-15'
  AND measurement_source_id = '2dbd3128-e66d-11de-aafa-00163efee9fd';
```

Πίνακας 24. Εντολή για τον υπολογισμό αθροίσματος και μέσου όρου για όλα τα ids για μία ημέρα

Πίνακας χωρίς partitioning : 2,168,001.315 ms

Πίνακας με partitioning: 49,210.544 ms

```
SELECT *
FROM table_i
WHERE measurement_time>= '2011-06-14'
  AND measurement_time < '2011-06-15'
  AND measurement_source_id='2dbd3128-e66d-11de-aafa-00163efee9fd';
```

Πίνακας 25. Εντολή για εμφάνιση όλων των εγγραφών για ένα id για ένα μήνα

Πίνακας χωρίς partitioning: 1,979,715.206 ms

Πίνακας με partitioning: 37,290.688 ms

7.5.3 Λεπτομέρειες

Το σύστημά μας, όπως έχουμε ήδη αναφέρει δεν είναι online, όπως το πραγματικό σύστημα της εταιρείας, επομένως απομονώσαμε τα δεδομένα που αφορούν μία ολόκληρη ημέρα στον πίνακα day. Στη συνέχεια τα αφαιρέσαμε από τον measurement_events ώστε να μην έχουμε διπλές εγγραφές που θα προκαλέσουν duplicate_key errors. Τέλος τα κάναμε ξανά insert στον αρχικό πίνακα για να χρονομετρήσουμε τη διαδικασία.

Για την σύγκριση των select δημιουργήσαμε δύο ίδιους πίνακες με 20 εκατομμύρια εγγραφές. Ο ένας έχει διαχωριστεί σε υποπίνακες σύμφωνα με τις παραπάνω συναρτήσεις ενώ ο άλλος όχι.

7.6 Batch Insertion

7.6.1 Θεωρία

Στο σύστημά μας γίνονται περίπου 134 inserts το δευτερόλεπτο τα οποία συνοδεύονται από 134 begin και 134 commit σύμφωνα με τα αποτελέσματα που του log analyzer PgFouine. Συγκεκριμένα για διάστημα 01:06:36” μας δόθηκαν οι εξής μετρήσεις :

total duration	times executed	Query
23m9s	532844	**COMMIT;**
1m51s	532844	**INSERT INTO** measurement_events (measurment_location_id, measurement_source_id, measurement_time, event_reception_time, measurement_value, quality, source_id, transducer_id) **VALUES** ('', '', '', 0.0e+0, 0.0e+0, '', NULL);
16.7s	532844	**BEGIN;**

Πίνακας 26. Στατιστικά εισαγωγής δεδομένων στο αρχικό σύστημα

Φαίνεται ξεκάθαρα ότι για να μην χαθεί καμία πληροφορία καλούνται συνεχόμενα transactions της μορφής begin/insert/commit. Κάτι τέτοιο είναι πολύ «ακριβό» για την απόδοση του συστήματος. Η τεχνική του batch insertion αναφέρεται στην πραγματοποίηση περισσότερων inserts ανά commit. Εφόσον γίνονται πολλές χιλιάδες inserts το λεπτό στη βάση μας (περίπου 8000), το να χαθούν μερικά από αυτά δεν αποτελεί μεγάλο πρόβλημα αν αυτό επιφέρει σημαντική βελτίωση στην ταχύτητα εκτέλεσης. Όπως προαναφέραμε για να γίνει ένα insert πρέπει να κληθεί ένα transaction με αποτέλεσμα στα 100 inserts για παράδειγμα να έχουμε και 100 κλήσεις. Βλέποντας ότι κάτι τέτοιο είναι πολύ χρονοβόρο εφαρμόσαμε την εν λόγω τεχνική βελτιστοποίησης, θεωρώντας ότι η απώλεια λίγων γραμμών δεδομένων θα επηρεάσει ελάχιστα την αξιοπιστία της εφαρμογής.

7.6.2 Πείραμα

7.6.2.1 Περιγραφή

Παραπάνω είδαμε ότι πραγματοποιούνται 134 inserts το δευτερόλεπτο. Εφαρμόζοντας το batch insertion θα δούμε κατά πόσο μπορεί να βελτιωθεί αυτό το νούμερο. Για την υλοποίηση της παραπάνω τεχνικής δημιουργήσαμε ένα script σε γλώσσα C όπως αυτό που φαίνεται στον Πίνακα 27 το οποίο δημιουργούσε με τη σειρά του ένα .sql αρχείο όπως φαίνεται στον Πίνακα 28. Με την εκτέλεση του .sql αρχείου προέκυψαν τα αποτελέσματα του Πίνακα 29 για συνολικό όγκο δεδομένων 10000 γραμμών.

```
#include<stdio.h>
int main()
{
int i,j;
for (i = 1; i <= n; i++)
/* m*n=10000 το n είναι ο αριθμός των commit και το m ο αριθμός inserts/commit.*/
{
  for (j =1; j <= m ; j ++)
      {
      printf ("BEGIN;\n");
      printf ("INSERT INTO measurement_events SELECT measurement_location_id, measurement_source_id, measurement_time, event_reception_time, measurement_value, quality, source_id, transducer_id FROM tableX WHERE aa=%d;\n",i);
/*aa είναι η στήλη-φίλτρο που μας επιτρέπει να έχουμε μοναδικά
```

```
Inserts
Και το tableY βρίσκεται στη RAM για γρηγορότερη ανάκτηση των δεδομένων*/
      }
  printf ("COMMIT;\n");
}
return 0;
}
```

Πίνακας 27. Script σε C για την δημιουργία αρχείου .sql

```
BEGIN;
INSERT INTO tableX
SELECT *
FROM tableY
WHERE aa=i;
COMMIT;
```

Πίνακας 28. Αρχείο .sql με 1<=i<=10000

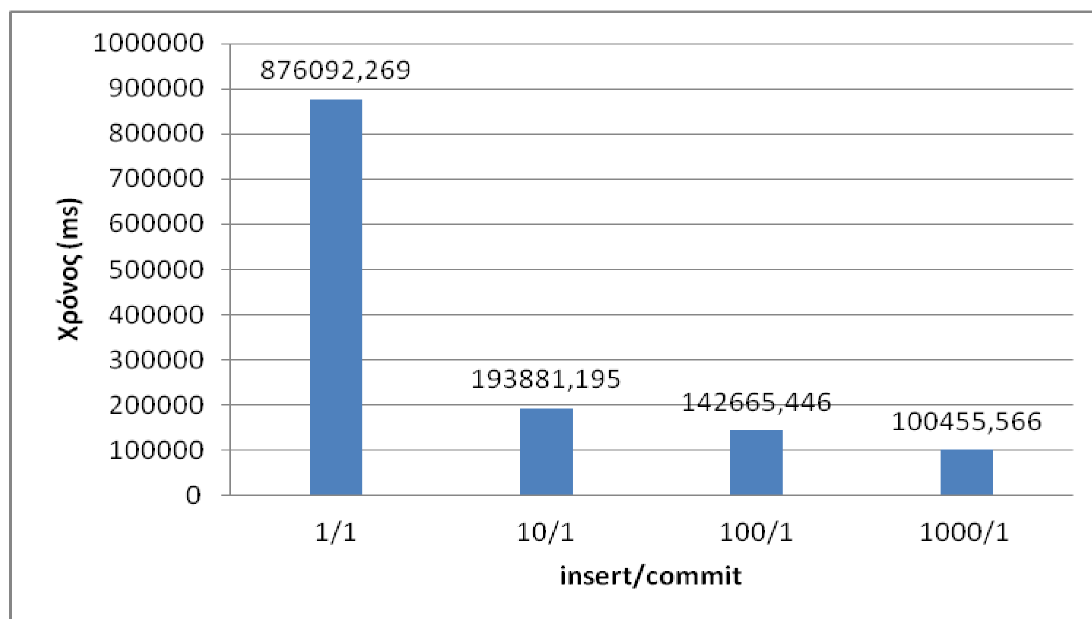
7.6.2.2 Μετρήσεις

inserts/commits	n	m	total runtime (ms)
1/1	10000	1	876,092.269
10/1	1000	10	193,881.195
100/1	100	100	142,665.446
1000/1	10	1000	100,455.566

Πίνακας 29. Αποτελέσματα

Παρατηρώντας τον πίνακα των αποτελεσμάτων βλέπουμε ότι, όπως ήταν αναμενόμενο, όσο λιγότερα commits πραγματοποιούνται τόσο μικραίνει ο χρόνος εκτέλεσης του αρχείου μας. Συμπερασματικά η λύση που φαίνεται να συνδυάζει βελτίωση της απόδοσης του συστήματος και αξιοπιστία ως προς τον χρήστη είναι το να πραγματοποιείται ένα commit ανά 10 inserts.

Παρακάτω παρουσιάζονται και γραφικά τα αποτελέσματα.



Γράφημα 5. Χρόνος εκτέλεσης 10000 inserts ανάλογα με τη συχνότητα των commits.

7.6.3 Λεπτομέρειες

Τα αποτελέσματα που φαίνονται στους παραπάνω πίνακες προέκυψαν ως μέσος όρος πολλών εκτελέσεων του ίδιου αρχείου για μεγαλύτερη αξιοπιστία. Ο πίνακας προορισμού των inserts περιείχε 90 εκατομμύρια γραμμές. Για την δημιουργία αρχείου με παραπάνω δεδομένα μας έβγαζε error η PostgreSQL για duplicate key, παρόλο που το αρχείο μας εμφανώς φτιάχνει μοναδικά inserts επομένως για το λόγο αυτό εξετάσαμε το batch insertion σε όγκο μόνο 10000 γραμμών δεδομένων. Παρόλα αυτά η διαφορά είναι εμφανής.

7.7 Replication

Ιστορικά, σημειώνουμε πως η ομάδα των developers της PostgreSQL θεωρούσε πως μία ενσωματωμένη τεχνολογία replication και clustering θα ήταν πέρα από το σκοπό του κυρίου project. Κι αυτό για να αφήσουν χώρο σε άλλες λύσεις/projects να δημιουργηθούν διότι κάθε μία θα ήταν περισσότερο ή λιγότερο καλή σε κάποιο feature οπότε θα υπήρχε ανταγωνισμός και επομένως συνεχής βελτίωση. Τελικά όμως είδαν πως το γεγονός αυτό μείωνε την αποδοχή της PostgreSQL σε μεγάλο βαθμό αφού τα συμπληρωματικά λογισμικά που αναπτύχθηκαν για replication και clustering ήταν πολύπλοκα στην εγκατάσταση και χρήση, οπότε δε βόλευαν για απλές περιπτώσεις. Έτσι, φρόντισαν η PostgreSQL 9.0 να περιλαμβάνει ένα απλό και αξιόπιστο σύστημα για replication που όμως σε καμία περίπτωση δε θα αντικαθιστούσε τις λύσεις που έχουν δημιουργηθεί για τον σκοπό αυτό. Επειδή το σύστημα

της εταιρείας χρησιμοποιεί PostgreSQL 8.4, ψάξαμε τα προγράμματα που χρειάζονται για εφαρμογή replication/clustering/connection pooling τεχνικών και πιστεύουμε πως το καλύτερο γι αυτό που θέλουμε (σύμφωνα με τα features τους και γνώμες διαφόρων που τα έχουν χρησιμοποιήσει) είναι το pgpool-II. Με το εργαλείο αυτό εκτός του ότι δεν υπάρχει φόβος αν πάθει κάτι η βάση μας (αφού υπάρχουν αντίγραφα σε άλλους servers τους οποίους διαχειρίζεται), υπάρχει η δυνατότητα διανομής των SELECT queries μεταξύ των servers με αποτέλεσμα τη βελτίωση της συνολικής απόδοσης του συστήματος. Γίνεται δηλαδή μία εξισορρόπηση του φόρτου εργασίας όταν τρέχουν πολλά query ταυτόχρονα. Επίσης, υποστηρίζεται και η εκτέλεση των queries παράλληλα. Αυτό σημαίνει πως -εφόσον τα data υπάρχουν σε περισσότερους από έναν servers- το query μπορεί να εκτελεστεί ταυτόχρονα σε όλους τους servers και να συλλέξει συμπληρωματικά data. Έτσι μειώνεται ο συνολικός χρόνος εκτέλεσης. Η τεχνική του replication χρησιμοποιείται κυρίως για δεδομένα μεγάλης κλίμακας. **[17]**

Για την αποφυγή χρήσης κάποιου επιπλέον εργαλείου, όπως αυτό που περιγράψαμε παραπάνω, προτείνουμε αναβάθμιση του συστήματος σε PostgreSQL 9.2 που είναι η τελευταία δοκιμασμένη έκδοση. Η PostgreSQL 8.4 δεν μπορεί να γίνει update σε 9.2. Πρέπει να απεγκατασταθεί πλήρως και να εγκατασταθεί εκ νέου και συνεπώς η διαδικασία θα κοστίσει χρονικά τόσο όσο χρειάζονται τα δεδομένα να φορτωθούν στην νέα βάση. **[18]**

7.8 Άλλες τεχνικές

7.8.1 Distributed caching layer

Όταν ένα σύστημα μεγαλώνει, είναι σκόπιμο να υιοθετήσουμε ένα στρώμα κατανεμημένης κρυφής μνήμης. Αυτό κάνει στην ουσία ότι κάνει η κρυφή μνήμη, αλλά διανέμει τη δουλειά μεταξύ πολλών servers. Η λογική όψη όμως παραμένει αυτή μίας ενιαίας κρυφής μνήμης. Έχουμε επομένως μία προσωρινή αποθήκη για μέρος των δεδομένων μας, κατανεμημένη στους διάφορους servers. Αυτό χρησιμεύει ιδιαίτερα στην περίπτωση που τα δεδομένα που αποθηκεύουμε στο στρώμα αυτό δε τα χρειαζόμαστε για πάντα, αλλά μόνο προσωρινά (ώρες, μέρες, βδομάδες). Υπάρχουν πολλές εμπορικές επιλογές για cache distributing, όπως το memcached, το EHCache, το Oracle Coherence και άλλα. **[10]**

7.8.2 Denormalization

Αποκανονικοποίηση είναι η διαδικασία κατά την οποία αναπαράγουμε μέρος των δεδομένων ώστε να ανταποκρίνονται καλύτερα στα ερωτήματα που τα αφορούν. Για παράδειγμα

προσθέτουμε στήλες σε πίνακες οι οποίες υπάρχουν ήδη σε άλλους πίνακες (πλεονασμός) με σκοπό να αποφύγουμε joins που κάνουν κάποιο ερώτημα απαγορευτικά αργό. **[10]**

7.9 Πρόταση λύσης με εφαρμογή τεχνικών βελτιστοποίησης σε σχεσιακή βάση

Η λύση που προτείνουμε προέκυψε από την εφαρμογή τεχνικών βελτιστοποίησης στο υπάρχον σύστημα.

Για τη βελτίωση του χρόνου εισαγωγής των δεδομένων επιλέξαμε να κάνουμε partitioning by date range, δηλαδή χωρίσαμε τον πίνακα measurement_events σε υποπίνακες που αντιπροσωπεύουν τα δεδομένα μίας μέρας σύμφωνα με τη στήλη measurement_time. Κάθε υποπίνακας κληρονομεί όλες τις ιδιότητες του αρχικού πίνακα. Όπως αναφέρθηκε και στην εισαγωγή, τα δεδομένα συλλέγονται από τα φωτοβολταϊκά πάρκα με χρήση χρονοσειρών, πράγμα που μας εξασφαλίζει ότι κάθε νέα εισαγωγή είναι νεότερη χρονικά από όλες τις προηγούμενες καταχωρήσεις και όλες οι εισαγωγές νέων γραμμών σε μία μέρα αφορούν την ίδια την μέρα. Αυτό μας εξασφαλίζει ότι όλα τα inserts γίνονται σε ένα μόνο partition του πίνακα και συγκεκριμένα σε αυτό που αφορά την τρέχουσα ημέρα. Έτσι κάνοντας το partitioning αντιμετωπίζουμε στην ουσία το πρώτο πρόβλημα που ήταν ο μεγάλος όγκος δεδομένων του πίνακα προορισμού. Συγκρίνοντας τους χρόνους εκτέλεσης της εισαγωγής δεδομένων για μία ολόκληρη μέρα, πριν και μετά το partitioning, παρατηρήσαμε μείωση του χρόνου κατά 40%.

Επιπλέον καθυστέρηση για την εισαγωγή δεδομένων ήταν η ύπαρξη index, γι’ αυτό αφαιρέσαμε το index από τον πίνακα όπου γίνονταν οι εισαγωγές και το επαναφέραμε στο τέλος της μέρας, όταν πια οι εισαγωγές θα μεταφέρονταν στον επόμενο υποπίνακα. Αυτή η κίνηση είχε σαν αποτέλεσμα 97% μείωση στο χρόνο εκτέλεσης των εισαγωγών για μία μέρα. Τα δεδομένα που αντιστοιχούν σ’ αυτό το χρονικό διάστημα είναι περίπου 8 εκατομμύρια γραμμές, συνεπώς η μη ύπαρξη index δεν επιτρέπει στους χρήστες πρόσβαση στα δεδομένα αυτά πριν το πέρας της μέρας. Εξαρτάται λοιπόν από τις απαιτήσεις των πελατών της εταιρείας κατά πόσο θα πρέπει να υπάρχει index στον υποπίνακα της τρέχουσας ημέρας.

Οι παραπάνω κινήσεις βελτιώνουν την απόδοση της εντολής insert, όμως όπως προαναφέραμε η εταιρεία έχει επιλέξει κάθε εισαγωγή γραμμής να συνοδεύεται από μία δοσοληψία begin-commit. Οι μετρήσεις μας έδειξαν ότι κάτι τέτοιο είναι πολύ χρονοβόρο, πράγμα που φαίνεται ξεκάθαρα και από το log analyzer που δείχνει ότι η εντολή commit είναι μακράν η πιο αργή εντολή του συστήματος. Μία μικρή μείωση στη συχνότητα των commit

από 1 προς 1 σε 1 προς 10 αποτελεί τεράστια βελτίωση στην απόδοση του συστήματος αφού όπως μετρήσαμε στην ενότητα του batch insertion ο χρόνος εκτέλεσης της δοσοληψίας begin-insert-commit μειώνεται κατά 78%. Η αλλαγή αυτή μειώνει ελάχιστα την αξιοπιστία που υπόσχεται η εταιρεία αφού το ποσό των δεδομένων που θα χαθεί σε περίπτωση κατάρρευσης του συστήματος είναι υπερβολικά μικρό σε σχέση με το συνολικό όγκο.

Η απόφασή μας για partitioning by range στον κεντρικό πίνακα έχει θετικά αποτελέσματα και για τα ερωτήματα που έχουν κάποιο περιορισμό στο where-clause με βάση τη στήλη measurement_time. Κατά την εκτέλεση ενός τέτοιου ερωτήματος στον κεντρικό πίνακα γίνεται έλεγχος των check constraints των υποπινάκων με αποτέλεσμα να λαμβάνονται υπόψη στην αποτίμησή του μόνο οι υποπίνακες των οποίων το check συμπίπτει με το ζητούμενο διάστημα. Με τον τρόπο αυτό παρακάμπτεται μεγάλος όγκος δεδομένων και συνεπώς βελτιώνεται σημαντικά η απόδοση του ερωτήματος. Για παράδειγμα, στην περίπτωσή μας ο κεντρικός πίνακας measurement_events περιέχει δεδομένα για 932 διαφορετικές μέρες άρα μετά το partitioning θα προκύψουν 932 υποπίνακες. Συνεπώς, ένα ερώτημα που ζητάει τα δεδομένα ενός μήνα ενός id θα ψάξει μόνο σε 30 υποπίνακες ενώ αντίστοιχα για τα δεδομένα μίας μέρας για ένα id μόνο σε ένα. Το κέρδος που έχουμε σ’ αυτή την περίπτωση χάρη στο partitioning είναι μεγάλο.

Ένα άλλο πρόβλημα που έπρεπε να λύσουμε ήταν τα ερωτήματα που ζητούν τον υπολογισμό του μέσου όρου ή του αθροίσματος για τη στήλη measurement_value για συγκεκριμένες χρονικές περιόδους (π.χ. ένας μήνας) ανά measurement_source_id. Η παρούσα δομή του συστήματος δεν επιτρέπει τέτοιους υπολογισμούς με χρήση SQL εσωτερικά της βάσης διότι η διαδικασία είναι πολύ χρονοβόρα και πολλές φορές δεν φέρνει αποτέλεσμα, κυρίως όταν ζητάμε υπολογισμούς για περισσότερα από ένα ids. Για το λόγο αυτό όλη η διαδικασία τέτοιων υπολογισμών γίνεται εξωτερικά της βάσης με χρήση άλλων προγραμμάτων. Εμείς προσπαθήσαμε να κάνουμε εφικτούς αυτούς τους υπολογισμούς μέσα από την PostgreSQL με όσο το δυνατόν καλύτερες αποδόσεις. Μετά από μελέτη του συστήματος καταλήξαμε στο ότι μία ικανοποιητική λύση είναι η χρήση ενδιάμεσων πινάκων. Οι πίνακες αυτοί θα αποθηκεύουν τον μέσο όρο και το άθροισμα του measurement_value ανά μήνα για κάθε measurement_source_id, εκ των οποίων μόνο ο τρέχων μήνας θα είναι αποθηκευμένος στη RAM (in memory), ενώ όλοι οι υπόλοιποι μετά την ολοκλήρωσή τους θα αποθηκεύονται σε έναν πίνακα του δίσκου που θα κρατάει το ιστορικό. Στον πίνακα αυτόν θα έχει γίνει partitioning ανά μήνα. Η διαδικασία αυτή μας εξασφαλίζει γρήγορα inserts/updates αφού ο μόνος πίνακας στον οποίο γίνονται βρίσκεται στη μνήμη RAM. Επίσης τα selects για οποιοδήποτε μήνα γίνονται σε ελάχιστο χρόνο αφού ο κάθε υποπίνακας περιέχει το πολύ 605.239 εγγραφές όσα είναι όλα τα διαφορετικά measurement_source_ids που υπάρχουν στη πίνακα measurement_events. Ο αριθμός αυτός είναι το ανώτερο θεωρητικό όριο για κάθε

υποπίνακα, αλλά για τα δεδομένα που έχουμε παρατηρήσαμε ότι εμφανίζονται περίπου 150.000 διαφορετικά measurement_source_ids άρα τόσες θα είναι και οι εγγραφές σε κάθε υποπίνακα. Και στις δύο περιπτώσεις το μέγεθος του πίνακα είναι πάρα πολύ μικρό άρα οποιαδήποτε εντολή select θα εκτελείται άμεσα. Γνωρίζουμε ότι σε περίπτωση κατάρρευσης του συστήματος τα δεδομένα που βρίσκονται αποθηκευμένα στη μνήμη RAM χάνονται. Επομένως θα αποθηκεύουμε αντίγραφο ασφαλείας στο δίσκο ανά ένα εκατομμύριο inserts για εύκολη ανάκτηση.

Συνοψίζουμε την πρόταση μας στις παρακάτω αλλαγές :

- Partitioning στον κεντρικό πίνακα measurement_events ανά μέρα βάσει της στήλης measurement_time, χωρίς index μόνο στο partition στο οποίο γίνονται τα writes. Δημιουργία του index που λείπει μετά το πέρας της μέρας.
- Ενδιάμεσοι πίνακες που θα αποθηκεύουν τον μέσο όρο και το άθροισμα του measurement_value ανά μήνα για κάθε measurement_source_id, εκ των οποίων μόνο ο τρέχων μήνας θα είναι in memory ενώ όλοι οι υπόλοιποι μετά την ολοκλήρωσή τους θα αποθηκεύονται σε έναν πίνακα του δίσκου που θα κρατάει το ιστορικό. Στον πίνακα αυτόν θα έχει γίνει partitioning ανά μήνα. Επίσης θα αποθηκεύεται αντίγραφο ασφαλείας του in memory πίνακα ανά ένα εκατομμύριο insert.
- Αύξηση των insert ανά commit από 1 προς 1 σε 10 προς 1.

8 *Μη σχεσιακές Βάσεις Δεδομένων*

8.1 Γενικά

Για πολλά χρόνια, το σχεσιακό μοντέλο δεδομένων ήταν αυτό που επικρατούσε σε όλους τους τομείς της πληροφορικής. Στο διαδίκτυο, στα πληροφοριακά συστήματα, στους προσωπικούς υπολογιστές. Λόγω της απήχησης αυτού του μοντέλου αναπαράστασης των δεδομένων, των πληροφοριών και των σχέσεων μεταξύ τους, οι σχεσιακές βάσεις δεδομένων ήταν στην ουσία μονόδρομος. Τα τελευταία χρόνια όμως, λόγω της ταχύτατης αύξησης των χρηστών του διαδικτύου, του ρυθμού κυκλοφορίας των πληροφοριών και τελικά του cloud computing, οι επιστήμονες του χώρου και οι προγραμματιστές συνειδητοποίησαν πως οι σχεσιακές βάσεις δεδομένων δεν ήταν κατάλληλες πια.

Ξεκίνησαν να αντιμετωπίζουν προβλήματα κλιμάκωσης όταν οι σχεσιακές εφαρμογές είχαν επιτυχία και αυξανόταν η χρήση τους. Τα joins, που είναι αναπόφευκτα σε οποιαδήποτε, ακόμα και μικρού μεγέθους, κανονικοποιημένη σχεσιακή βάση, προκαλούν μεγάλες καθυστερήσεις. Ακόμη, η εξασφάλιση της συνέπειας (consistency), που είναι βασική αρχή για τις σχεσιακές βάσεις, δεν επιτρέπει ταυτόχρονα reads και writes στη βάση, πράγμα που απαιτεί κλείδωμα μέρους της βάσης σε κάθε περίπτωση, με αποτέλεσμα τα αντίστοιχα δεδομένα να μην είναι διαθέσιμα στους χρήστες. Αυτό μπορεί να γίνει αφόρητο υπό την πίεση μεγάλου φόρτου δεδομένων, αφού τα locks θα έχουν σαν αποτέλεσμα οι χρήστες να «ανταγωνίζονται» για τα δεδομένα και να κάνουν ουρά περιμένοντας τη σειρά τους να διαβάσουν ή να γράψουν κάτι.

Για την αντιμετώπιση όλων αυτών των προβλημάτων, εφαρμόστηκαν διάφορες τεχνικές βελτιστοποίησης όπως αυτές που περιγράψαμε αναλυτικά στο πρώτο μέρος της διπλωματικής που αφορούσε τις σχεσιακές βάσεις. Όμως, η διαδικασία συνεχούς βελτιστοποίησης, ειδικά όταν αφορά εφαρμογές μεγάλου όγκου δεδομένων (big data), είχε ως αποτέλεσμα να χαθεί ο σχεσιακός χαρακτήρας της βάσης και να παραβιαστούν οι κανόνες του Dr. Codd. Έτσι άρχισαν να ερευνούν εναλλακτικούς τρόπους αποθήκευσης δεδομένων. Αυτή τους η αναζήτηση είχε ως αποτέλεσμα τη δημιουργία μη σχεσιακών συστημάτων βάσεων δεδομένων τα οποία θα μπορούσαν να ανταποκριθούν καλύτερα σε κατανεμημένο

περιβάλλον. Έτσι, το 2009 ξεκίνησε η χρήση του όρου NoSQL (Not only SQL) για την αναφορά σε αυτόν το νέο, μη σχεσιακό τρόπο αποθήκευσης διαδικτυακών πληροφοριών. Η φιλοσοφία των NoSQL βάσεων άρχισε σταδιακά να κερδίζει όλο και περισσότερους οπαδούς στο διαδίκτυο και μάλιστα εταιρείες-κολοσσοί όπως οι: Google, Amazon, Facebook, Twitter, Digg, Reddit, Linkedin, Sourceforge, Bing χρησιμοποιούν μη σχεσιακές βάσεις τώρα πια. Ο όγκος των δεδομένων που διαχειρίζονται τέτοιες εταιρείες είναι της τάξης πολλών petabyte και επομένως θα ήταν αδύνατη η κλιμάκωσή τους με κάποιο σχεσιακό σύστημα ΒΔ. Κι αυτό γιατί οι σχεσιακές ΒΔ συναντούν περιορισμούς όσον αφορά την αντιμετώπιση προβλημάτων όπως είναι η εξόρυξη δεδομένων, το Web 2.0, το cloud computing, τα μη γραμμικά σε εκτέλεση ερωτήματα κα. Αντιθέτως, ο κατανεμημένος μηχανισμός που προσφέρουν οι μη σχεσιακές ΒΔ καθιστά τις εφαρμογές κάθετα και οριζόντια κλιμακώσιμες και δίνει τη δυνατότητα εκμετάλλευσης πολλαπλών υπολογιστικών συστημάτων με πολυπύρηνους επεξεργαστές οπότε καθιστά εφικτή την αντιμετώπιση των προαναφερθέντων προβλημάτων. Επίσης, ορισμένα hot use-cases όπως αραιά δεδομένα, συνεργατική επεξεργασία, κοινωνικός γράφος, αρχεία καταγραφής τα μη σχεσιακά συστήματα τα καλύπτουν καλύτερα από τα σχεσιακά. Συμπερασματικά, έχει ήδη ξεκινήσει η εξάπλωση των μη σχεσιακών συστημάτων σε διαδικτυακές εφαρμογές και αναμένεται να κατακλύσουν το Διαδίκτυο. Ήδη μεγάλες εταιρίες έχουν μεταφέρει όλες τους τις εφαρμογές που χρησιμοποιούσαν σχεσιακές βάσεις δεδομένων σε μη σχεσιακές. Αναμένεται να ακολουθήσουν κι οι υπόλοιπες εταιρίες, ούτως ώστε να εκμεταλλευθούν και αυτές την κλιμάκωση που προσφέρουν οι μη σχεσιακές βάσεις δεδομένων και την εκμετάλλευση περισσότερων υπολογιστικών πόρων.

Στο χώρο των NoSql βάσεων αυτή την στιγμή τα πιο γνωστά συστήματα που κυκλοφορούν είναι: Dynamo, SimpleDB, BigTable, HBase, CouchDB, MongoDB, Voldemort, Cassandra. **[19], [20]**

8.2 Τύποι μη σχεσιακών ΒΔ.

8.2.1 Key-value Stores (Αποθήκες κλειδιών-τιμών)

Αποτελούν αποθήκες δεδομένων οι οποίες λειτουργούν με την χρήση ενός hashtable, στον οποίο αποθηκεύονται μοναδικά κλειδιά και δείκτες για τα δεδομένα του κάθε αντικειμένου. Τέτοιου τύπου αποθήκες δεδομένων προσφέρουν υπηρεσίες για την διαχείριση δεδομένων χωρίς τον ορισμό συγκεκριμένου σχήματος. Οι αντιστοιχίσεις των κλειδιών σε δείκτες συνδυάζονται με μηχανισμούς caching για την μεγιστοποίηση της απόδοσης. Τα δεδομένα αποθηκεύονται ως ζευγάρια κλειδιών-τιμών, ούτως ώστε οι τιμές να είναι ευρετηριασμένες από τα αντίστοιχα κλειδιά. Αυτά τα συστήματα μπορούν να αποθηκεύουν δομημένη και μη

δομημένη πληροφορία. Ένα παράδειγμα τέτοιου συστήματος είναι τα Dynamo, SimpleDB, Voldemort, Scalaris. **[20]**

8.2.2 Column-oriented Databases (Βάσεις δεδομένων προσανατολισμένες στη στήλη)

Δημιουργήθηκαν με σκοπό την αποθήκευση και την επεξεργασία μεγάλης ποσότητας δεδομένων, τα οποία είναι κατανεμημένα σε πολλούς διαφορετικούς υπολογιστικούς κόμβους. Υπάρχουν και στην περίπτωση αυτή κλειδιά το οποία δείχνουν σε διαφορετικά σύνολα στηλών. Οι γραμμές διαχωρίζονται από κλειδιά και οι στήλες χωρίζονται σε οικογένειες. Αυτοί οι τύποι των βάσεων δεδομένων περιέχουν μία επεκτάσιμη στήλη από πολύ κοντινά συσχετιζόμενη πληροφορία, παρά σύνολα πληροφοριών σε μία αυστηρή δομή πινάκων από στήλες και γραμμές, όπως μπορούν να βρεθούν στις σχεσιακές βάσεις δεδομένων. Παραδείγματα τέτοιων συστημάτων βάσεων δεδομένων είναι τα BigTable, Cassandra, Hyperbase και HBase. **[20]**

8.2.3 Document-based Stores (Αποθήκες βασισμένες στα έγγραφα)

Είναι παρόμοια με τα key – value stores . Το μοντέλο ουσιαστικά διαχειρίζεται εκδόσεις αρχείων, τα οποία αποτελούν συλλογές από κλειδιά άλλων αρχείων. Η πληροφορία αποθηκεύεται και οργανώνεται ως μία συλλογή από δεδομένα. Οι χρήστες επιτρέπεται να προσθέτουν οποιοδήποτε αριθμό πεδίων οποιουδήποτε μεγέθους στο έγγραφο. Τα συστήματα αυτά τείνουν τα αποθηκεύουν τα έγγραφα σύμφωνα με το πρότυπο JSON. Παράδειγμα τέτοιων συστημάτων είναι τα Riak, CouchDB και MongoDB. **[20]**

8.2.4 Graph databases

Βασίζονται στην θεωρία των γράφων για την κατασκευή συστημάτων με κόμβους, οι οποίοι τοποθετούνται ανάλογα με τις σχέσεις που προκύπτουν μεταξύ των δεδομένων. Χρησιμοποιείται ένα εύκαμπτο μοντέλο γραφημάτων, για την εύκολη κλιμάκωση του συστήματος. Το data model των ΒΔ αυτών είναι μη σχεσιακό, schema-less και σχεδιασμένο για εύκολο partitioning. Παράδειγμα τέτοιων συστημάτων είναι τα Neo4j, AllegroGraph και GraphDB. **[20]**

8.3 Χαρακτηριστικά μη σχεσιακών συστημάτων ΒΔ.

8.3.1 Ανθεκτικότητα

Όπως και στις σχεσιακές βάσεις, έτσι και στις NoSQL βάσεις όταν κάτι γραφτεί στο δίσκο είμαστε σίγουροι πως δε θα χαθεί. Η διαφορά από τις σχεσιακές βάσεις βέβαια είναι πως στα κατανεμημένα συστήματα κάτι τέτοιο σημαίνει πως η πληροφορία έχει γραφτεί εκ προεπιλογής σε παραπάνω από έναν δίσκο και επομένως δε θα χαθεί ακόμα και σε περίπτωση αποτυχίας ενός δίσκου ή υπολογιστή. **[20], [21]**

8.3.2 Προσαρμοστικότητα

Οι NoSQL βάσεις είναι ευέλικτες σχετικά με τα δεδομένα που αποθηκεύουν. Δεν υπάρχει περιορισμός στον τύπο των δεδομένων που μπορούν να αποθηκευτούν, ενώ στις SQL βάσεις ο τύπος έχει καθοριστεί κατά τη δημιουργία τους. Ακόμη, οι NoSQL βάσεις μπορούν να αντιμετωπίσουν με ευκολία πιο πολύπλοκες και προηγμένες δομές δεδομένων. Οι περισσότερες από τις υπάρχουσες NoSQL βάσεις, παρέχουν πλούσιες δομές δεδομένων και εργασιών σε λίστες, σύνολα, ταξινομημένα σύνολα και κλειδιά κατακερματισμού που κάνουν πράγματα όπως εξισορρόπηση φορτίου και message-queuing απλούστατα στην υλοποίηση. **[20], [21]**

8.3.3 Διαχειρισιμότητα

Η ύπαρξη ενός άριστα εκπαιδευμένου, με πολλή εμπειρία και γνώση, διαχειριστή βάσεων δεδομένων είναι απαραίτητη για τη συντήρηση ενός υψηλού επιπέδου RDBMS. Είναι άρρηκτα συνδεδεμένος με όλες τις λειτουργίες που αφορούν τη βάση, δηλαδή με το σχεδιασμό, την εγκατάσταση και τη συνεχή παρακολούθηση και ρύθμιση της. Οι μη σχεσιακές βάσεις από τα θεμέλιά τους έχουν σχεδιαστεί για να χρειάζονται λιγότερη διαχείριση. Η δυνατότητα αυτόματης επισκευής, η διανομή δεδομένων, το απλούστερο μοντέλο δεδομένων είναι όλα χαρακτηριστικά που συντελούν στη μείωση των απαιτήσεων σε θέματα ρύθμισης και διαχείρισης. **[20], [21]**

8.3.4 Παράλληλη επεξεργασία

Χάρη στις δυνατότητες που προσφέρουν τα συστήματα NoSQL, όπως εύκολη αναπαραγωγή και διαμοιρασμός δεδομένων, καθίσταται εξίσου εύκολη και η εκμετάλλευση των φυσικών πόρων του συμπλέγματος όπου λειτουργεί η βάση. Από τη στιγμή που τα δεδομένα βρίσκονται σε διαφορετικούς δίσκους, υπολογιστές ή ακόμα και δίκτυα, τα ερωτήματα

μπορούν χωρίς καθόλου επιπλέον κώδικα να εκτελούνται παράλληλα και να δίνουν άμεσα απαντήσεις. **[20], [21]**

8.3.5 Διαθεσιμότητα

Όπως θα ήταν αναμενόμενο, δε συμφέρει καθόλου κάποια web ή mobile based επιχείρηση (π.χ. facebook) να είναι πεσμένη. Θέλει δηλαδή να υπάρχει όσο το δυνατόν λιγότερο downtime. Χάρη πάλι στο γεγονός ότι οι NoSQL βάσεις είναι κατανεμημένες, οι ενημερώσεις λογισμικού, αναβαθμίσεις υλικού αλλά και τυχόν αποτυχίες υλικού δε σημαίνουν πως θα πέσει η εφαρμογή, αφού υπάρχει σε πολλαπλούς servers. Αντιθέτως, αν η εφαρμογή βασίζεται σε μία σχεσιακή βάση, τα παραπάνω ισοδυναμούν με δυσκολίες και downtime. **[20], [21]**

8.3.6 Υψηλή απόδοση

Εκτός από το γεγονός ότι με την προσθήκη υπολογιστών στο cluster έχουμε ήδη περισσότερη υπολογιστική ισχύ και άρα καλύτερη απόδοση, η ίδια η αρχιτεκτονική των NoSQL εργαλείων κάνει τις βάσεις αυτές πιο αποδοτικές. Αν μία σχεσιακή βάση είχε μερικές εκατοντάδες χιλιάδες πίνακες, η επεξεργασία των δεδομένων θα δημιουργούσε πάρα πολλά locks και θα υποβάθμιζε την απόδοση. Λόγω όμως της ασθενέστερης συνέπειας που χαρακτηρίζει τα μοντέλα δεδομένων των NoSQL βάσεων, δίνεται βάρος στην αποτελεσματικότητα αντί της συνοχής και επομένως δεν υπάρχει πρόβλημα όσο μεγάλη και να είναι η βάση. **[20], [21]**

8.3.7 Αναπαραγωγή δεδομένων

Τα δεδομένα διανέμονται μεταξύ πολλών κόμβων με πολύ εύκολο τρόπο. Σε όλα τα NoSQL συστήματα κάτι τέτοιο ρυθμίζεται πολύ απλά βάζοντας σε ένα .conf αρχείο την ip ή το domain name των servers στους οποίους θέλουμε να αναπαράγονται τα δεδομένα. Η αυτοματοποίηση της αναπαραγωγής έχει ως αποτέλεσμα την υψηλή διαθεσιμότητα των δεδομένων και επομένως την εύκολη αποκατάσταση από καταστροφή τους, χωρίς να εμπλέκονται ξεχωριστές εφαρμογές για το σκοπό αυτό. Ακόμη, χάρη σε αυτό το χαρακτηριστικό, η προσθήκη ή αφαίρεση υλικού (υπολογιστές, servers κλπ) γίνεται χωρίς downtime. **[20], [21]**

8.3.8 Schema-less persistence – Δυναμικά Schemas

Στις σχεσιακές βάσεις είναι απαραίτητο το σχήμα να είναι ορισμένο πριν αρχίσουμε να προσθέτουμε δεδομένα. Για παράδειγμα, αν θέλουμε να κρατάμε σε κάποια SQL βάση

δεδομένα όπως όνομα, επώνυμο και τηλέφωνο πελάτη, είναι απαραίτητο η βάση αυτή να το γνωρίζει από πριν.

Τώρα όμως που ο ρυθμός αύξησης των δυνατοτήτων ιστοσελίδων/εφαρμογών είναι ταχύτατος, κάθε φορά που ολοκληρώνεται κάποιο νέο χαρακτηριστικό, είναι πολύ πιθανό να χρειάζεται αλλαγή και το σχήμα της βάσης. Αν θέλω δηλαδή στη βάση του παραδείγματος να ξεκινήσω να κρατάω και κάτι παραπάνω για τους πελάτες, πρέπει να αλλάξω το σχήμα και να προσθέσω στήλη και ίσως να φτιάξω επιπλέον indexes/foreign keys κλπ. Αν η βάση είναι μεγάλη, η διαδικασία αυτή θα πάρει πολλή ώρα οπότε η βάση δε θα λειτουργεί για το διάστημα αυτό. Επίσης, με ένα σχεσιακό σύστημα, δεν υπάρχει τρόπος να αντιμετωπίσουμε δεδομένα που είναι αδόμητα ή άγνωστα από πριν. Επομένως, το φλέγον θέμα της διαχείρισης των αλλαγών (change management) είναι μεγάλος πονοκέφαλος σε ένα μεγάλο RDBMS.

Αντιθέτως, οι NoSQL βάσεις είναι πολύ ευέλικτες και έχουν χαλαρούς (ή και ανύπαρκτους) περιορισμούς όσον αφορά το μοντέλο δεδομένων. Είναι σχεδιασμένες να επιτρέπουν την εισαγωγή δεδομένων χωρίς την ύπαρξη προκαθορισμένου σχήματος. Έτσι, οι σημαντικές αλλαγές που χρειάζονται γίνονται σε πραγματικό χρόνο, χωρίς την ανησυχία για διακοπή της λειτουργίας της βάσης. Επομένως η ανάπτυξη καθίσταται γρηγορότερη, η ενσωμάτωση κώδικα πιο αξιόπιστη και χρειάζεται λιγότερη ώρα ενασχόλησης του διαχειριστή της βάσης. **[20], [21]**

8.3.9 Κλιμάκωση

Είναι ίσως το πιο σημαντικό χαρακτηριστικό των NoSQL βάσεων. Όταν οι επισκέψεις σε ένα site γίνονται μερικές εκατοντάδες ή και χιλιάδες το δευτερόλεπτο, μία σχεσιακή βάση θα έπρεπε να είναι διαμοιρασμένη και αναπαραγμένη σε μεγάλο βαθμό για να καταφέρει να ανταποκριθεί. Μία NoSQL βάση όμως, λόγω της κατανεμημένης της φύσης, το μόνο που χρειάζεται για σωστή κλιμάκωση, είναι προσθήκη υπολογιστών στο σύμπλεγμα (cluster). Δε χρειάζεται ούτε σπατάλη του χρόνου του διαχειριστή της βάσης, ούτε πολύπλοκος κώδικας για partitioning και replication. Ιδιαίτερα με τις νέες τεχνολογίες του cloud και των virtual machines, η κλιμάκωση είναι ιδιαίτερα εύκολη και φθηνή λύση. **[20], [21]**

8.3.10 Ενσωματωμένο caching

Οι περισσότερες NoSQL βάσεις παρέχουν τεχνολογίες ενσωματωμένου caching. Τα συχνά χρησιμοποιούμενα δεδομένα αποθηκεύονται στη μνήμη όσο το δυνατόν περισσότερο και επομένως η επικοινωνία με το δίσκο ελαχιστοποιείται. Έτσι δεν υπάρχει η ανάγκη για ξεχωριστό επίπεδο caching, ενώ στις περισσότερες SQL βάσεις χρειάζεται ξεχωριστή υποδομή για να επιτευχθεί κάτι τέτοιο. **[20], [21]**

8.3.11 Μικρότερο κόστος

Στις NoSQL βάσεις χρησιμοποιούνται φθηνοί, απλοί servers που μπορεί να βρίσκονται και στο cloud ή να είναι εικονικοί. Από την άλλη, τα RDBMS βασίζονται σε ακριβούς, ιδιωτικούς servers και συστήματα αποθήκευσης. Το κόστος gb ή δοσοληψία ανά δευτερόλεπτο είναι επομένως πολύ χαμηλότερο σε ένα σύστημα NoSQL από ένα σύστημα SQL. **[20], [21]**

8.4 Θεώρημα CAP

Σύμφωνα με το θεώρημα CAP **[22]**, ή αλλιώς θεώρημα του Brewer, είναι αδύνατο για ένα κατανεμημένο υπολογιστικό σύστημα να παρέχει ταυτόχρονα και τα τρία από τα παρακάτω χαρακτηριστικά:

Consistency (Συνέπεια): όλοι οι κόμβοι βλέπουν τα ίδια δεδομένα μία συγκεκριμένη χρονική στιγμή.

Availability (Διαθεσιμότητα): εγγύηση πως για κάθε λειτουργία που ζητείται να εκτελεστεί υπάρχει ανταπόκριση, ακόμα κι αν αυτό είναι μήνυμα λάθους.

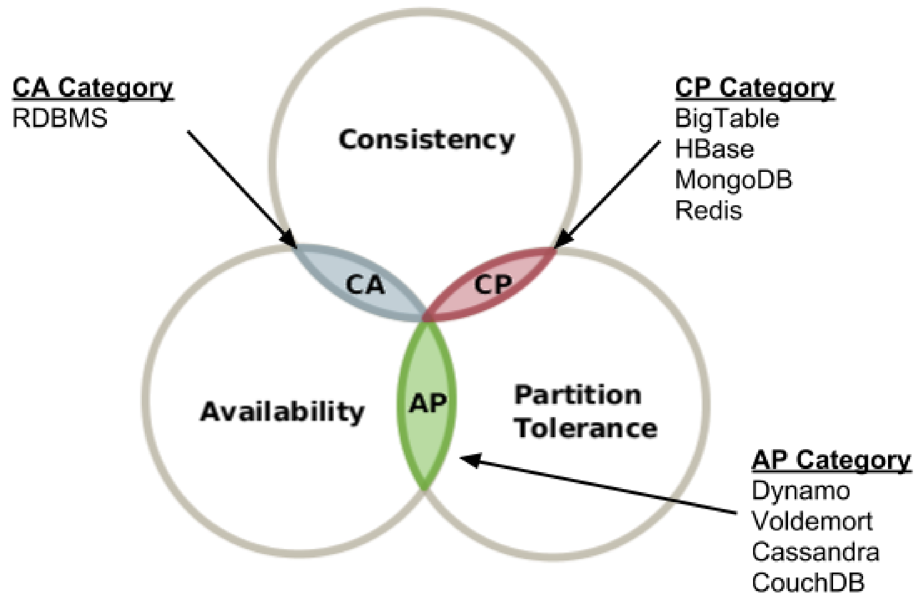
Partition Tolerance (Ανοχή τμημάτων): το σύστημα συνεχίζει να λειτουργεί παρά την απώλεια τμημάτων του.

Η συνέπεια μπορεί να περιγραφεί από την εξής πρόταση: «Όλοι οι πελάτες έχουν πάντα την ίδια όψη για τα δεδομένα». Η διαθεσιμότητα μπορεί να περιγραφεί από την εξής πρόταση: «Κάθε πελάτης μπορεί πάντα να διαβάζει και να γράφει στη βάση δεδομένων». Τέλος, η ανοχή τμημάτων μπορεί να περιγραφεί από την εξής πρόταση: «Το σύστημα δουλεύει καλά ανεξαρτήτως των φυσικών δικτυακών τμημάτων». Για κάθε σύστημα βάσης δεδομένων, πρέπει να επιλεγούν δύο μόνο από αυτά τα χαρακτηριστικά. Τα υπάρχουσα συστήματα σχεσιακών βάσεων δεδομένων επιλέγουν τη συνέπεια και τη διαθεσιμότητα, αλλά δε μπορούν να έχουν την ανοχή τμημάτων. Οπότε, οι μη σχεσιακές βάσεις δεδομένων μπορούν να έχουν την ανοχή τμημάτων, αλλά πρέπει να θυσιάσουν είτε τη διαθεσιμότητα είτε τη συνέπεια. Για να μπορεί να λειτουργήσει μία μη σχεσιακή βάση δεδομένων παίρνει όσα χρειάζεται για να λειτουργήσει.

CA: Μικρά τοπικά δίκτυα ή πολύ μικρά clusters, με μικρό partitioning στα δεδομένα. Εγγύηση για μεγάλο availability και για consistency. Λάθος λειτουργία στο σύστημα, όταν κλιμακωθεί

CP: Δεδομένα μη προσβάσιμα συνεχώς, αλλά έχουμε συνέπεια και αντοχή στην κλιμάκωση

AP: Δεδομένα συνεχώς διαθέσιμα με κίνδυνο να μην είναι ενημερωμένα. Eventual Consistency



Εικόνα 5. Θεώρημα CAP

8.5 BASE

Το ACID είναι το μοντέλο που παρέχει συνέπεια στις βάσεις δεδομένων, γι' αυτό και αποτελεί βασική αρχή των σχεσιακών βάσεων, αλλά πως μπορούμε να έχουμε υψηλότερη απόδοση και κλιμάκωση, που είναι ο στόχος των μη σχεσιακών βάσεων δεδομένων;

Μία απάντηση είναι το BASE, η αντιδιαμετρική άποψη του ACID (ονομάστηκε έτσι γιατί base-βάση είναι το αντίθετο του acid-οξέως). Η λέξη BASE βγαίνει ως εξής: Basically Available, Soft state, Eventually consistent.

Basically Available: Ακόμα κι αν υπάρξει κάποιο πρόβλημα σε κάποιον κόμβο και μερικά δεδομένα δεν είναι διαθέσιμα, το υπόλοιπο στρώμα δεδομένων παραμένει διαθέσιμο και σε λειτουργία.

Soft-State: Η κατάσταση του συστήματος δεν αλλάζει μέσω transactions. Μπορεί να αλλάξει δηλαδή, αλλά όχι απαραίτητα με την παρεμβολή κάποιου χρήστη, αλλά λόγω της τελικής συνέπειας.

Eventually Consistent: Η συνέπεια θα επέλθει με το χρόνο σε όλους τους κόμβους του συστήματος, αλλά κάποια δεδομένη χρονική στιγμή μπορεί κάποιος κόμβος να μην έχει ενημερωθεί ακόμα.

Τα παραπάνω χαρακτηριστικά καθιστούν τα συστήματα που ακολουθούν τη μεθοδολογία αυτή αποδοτικότερα και πιο κλιμακώσιμα από τα συστήματα που ακολουθούν το ACID. **[23]**

8.6 MapReduce

Μία άλλη έννοια που είναι συνδεδεμένη με τις NoSQL βάσεις δεδομένων είναι το MapReduce.

Τα τελευταία χρόνια στον τομέα των βάσεων δεδομένων, καλούνται τα συστήματα να επεξεργαστούν τεράστια σύνολα δεδομένων, τα οποία είναι κατανεμημένα σε διαφορετικούς servers. Μάλιστα συνήθως το υλικό που χρησιμοποιείται είναι μέτριας ποιότητας (ώστε να είναι οικονομική η αγορά και συντήρησή του) με αποτέλεσμα να συμβαίνουν συχνά βλάβες. Έτσι δεν υπήρχε κάποιος αποδοτικός τρόπος επεξεργασίας των δεδομένων αυτών, μέχρι που βρέθηκε το μοντέλο προγραμματισμού MapReduce, που αφορά την γρήγορη και παράλληλη επεξεργασία και παραγωγή μεγάλου όγκου δεδομένων.

Η φιλοσοφία πίσω από το MapReduce είναι η εξής: οποιοδήποτε πρόβλημα σπάει σε δύο φάσεις, τη Map και τη Reduce. Κατά τη Map, μη αλληλεπικαλυπτόμενα κομμάτια από δεδομένα εισόδου (εγγραφές <key,value>) ανατίθενται σε διαφορετικές διεργασίες (τους mappers) που βγάζουν ένα σύνολο από ενδιάμεσα αποτελέσματα (πάλι της μορφής <key,value>). Κατά τη Reduce, τα δεδομένα της φάσης Map τροφοδοτούνται σε έναν - συνήθως μικρότερο- αριθμό διεργασιών (τους reducers) που συνοψίζουν τα αποτελέσματα εισόδου σε μικρότερο αριθμό <key,value> εγγραφών. Υπάρχει δηλαδή ένας master ο οποίος είναι υπεύθυνος για το χρονοπρογραμματισμό των εργασιών και αναθέτει συγκεκριμένες εργασίες (και άρα συγκεκριμένα κομμάτια δεδομένων) σε διάφορους mappers και στη συνέχεια reducers. Όταν ένας εργάτης (mapper ή reducer) ολοκληρώσει την εργασία που έχει ανατεθεί, ενημερώνει τον master. Όταν όλοι οι εργάτες ενημερώσουν τον master, τότε αυτός επιστρέφει το αποτέλεσμα της λειτουργίας στον client. **[24]**

9 *Cassandra*

Εξετάζοντας τις επιλογές NoSQL βάσεων που υπάρχουν, καταλήξαμε ότι η επιλογή μας θα έπρεπε να είναι μία Column-oriented βάση, αφού η φιλοσοφία που κρύβεται πίσω από αυτές είναι ότι οι στήλες χωρίζονται σε οικογένειες στηλών (που είναι στην ουσία οι γνωστοί πίνακες των σχεσιακών βάσεων) και οι γραμμές χωρίζονται σύμφωνα με κάποιο κλειδί (που είναι στην ουσία το πρωτεύον κλειδί των σχεσιακών βάσεων). Στην κατηγορία αυτή ήμασταν μεταξύ των εξής δύο συστημάτων: Cassandra και HBase. Και οι δύο είναι απόγονοι του BigTable της Google, με την Cassandra να έχει πάρει λιγότερα χαρακτηριστικά καθώς ήταν επηρεασμένη και από το Dynamo της Amazon. Τελικά αποφασίσαμε να χρησιμοποιήσουμε την Cassandra κυρίως γιατί προσφέρει τη γλώσσα CQL (Cassandra Query Language) της οποίας οι εντολές μοιάζουν πολύ με τις εντολές της SQL. Άλλες παράμετροι που επίσης έπαιξαν ρόλο στην επιλογή της Cassandra ήταν το γεγονός ότι έχει πολύ αναλυτικό τεκμηρίωση (documentation) αλλά και το γεγονός ότι υπάρχει μία εμπορική διανομή της -την οποία τελικά χρησιμοποιήσαμε- από την εταιρεία DataStax με πολλά χρήσιμα εργαλεία και καλύτερη τεχνική υποστήριξη (support). Η χρήση της διανομής αυτής έγινε μετά από συνεννόηση με το τμήμα εξυπηρέτησης πελατών της εταιρείας DataStax στην Ευρώπη και επιβεβαίωση πως η χρήση δε θα είναι εμπορική αλλά καθαρά ερευνητική. **[25]**

9.1 Γενικά

Η Cassandra είναι ένα κατανεμημένο σύστημα βάσεων δεδομένων ανοιχτού κώδικα που έχει σχεδιαστεί για την αποθήκευση και διαχείριση πολύ μεγάλων ποσοτήτων δεδομένων μοιρασμένων σε πολλούς servers. Μπορεί να χρησιμοποιηθεί και ως real-time operational data store for online transactional applications και ως a read-intensive database for large-scale business intelligence (BI) systems.

Έχοντας δημιουργηθεί αρχικά για το Facebook, η Cassandra είναι σχεδιασμένη να έχει συμμετρικούς, peer-to-peer κόμβους αντί για master/named κόμβους, έτσι ώστε να είναι σίγουρο πως θα αποφευχθεί η ύπαρξη single point of failure (SPoF). Η Cassandra διαμοιράζει αυτόματα τα δεδομένα μεταξύ των κόμβων που ανήκουν στο cluster της βάσης, αλλά ο διαχειριστής του συστήματος είναι αυτός που έχει τη δυνατότητα να αποφασίσει ποια δεδομένα θα αναπαραχθούν και πόσα αντίγραφα των δεδομένων θα τηρούνται κατανεμημένα ανά τον κόσμο. **[1], [26], [27]**

9.2 Ιστορικά

Η Cassandra έγινε ανοιχτού κώδικα από το Facebook τον Ιούλιο του 2008. Η αρχική της έκδοση γράφτηκε κυρίως από έναν πρώην υπάλληλο της Amazon κι έναν της Microsoft. Ήταν έντονα επηρεασμένη από το Dynamo, την πρωτοποριακή κλειδί-τιμή (key-value) βάση δεδομένων της Amazon. Η Cassandra εφαρμόζει το μοντέλο αναπαραγωγής δεδομένων του Dynamo, χωρίς single point of failure, αλλά προσθέτει και ένα πιο ισχυρό μοντέλο οικογένειας στηλών.

Η Cassandra έγινε δεκτή στην Apache Incubator, και από τη στιγμή που βγήκε, τον Μάρτιο του 2010, ήταν μια πραγματική επιτυχία ανοιχτού κώδικα, με commiters από το Rackspace, το Digg, το Twitter και άλλες μεγάλες εταιρείες που ενώ δεν ήθελαν να γράψουν τη δική τους βάση δεδομένων από το μηδέν, μαζί χτίσανε κάτι σημαντικό.

Σήμερα, η Cassandra έχει εξελιχθεί από το αρχικό σύστημα που χρησιμοποιούνταν για την αναζήτηση στα inbox messages του Facebook. Έχει γίνει «ο νικητής στην απόδοση της επεξεργασίας συναλλαγών», και δικαίως φημίζεται για την αξιοπιστία της και την απόδοσή της σε δεδομένα μεγάλης κλίμακας.

Με τον καιρό η Cassandra άρχισε να ωριμάζει και να τραβάει πιο mainstream χρήστες. Έτσι, κατέστη σαφές πως υπήρχε η ανάγκη για εμπορική υποστήριξη. Για το λόγο αυτό ιδρύθηκε τον Απρίλιο του 2010 η Riptano, που βοήθησε στην προώθηση της Cassandra και υιοθέτησή της για ακόμα περισσότερα είδη χρήσης. Η εταιρεία Riptano αργότερα μετονομάστηκε σε DataStax γιατί όπως λέει ο συνιδρυτής και διευθύνων σύμβουλος, Matt Pfeil, «θέλαμε το όνομά μας να αντικατοπτρίζει με ακρίβεια τι είμαστε, τι κάνουμε και τι πρεσβεύουμε».

Το όνομα Cassandra είναι εμπνευσμένο από τη μάντισσα Κασσάνδρα της Ελληνικής μυθολογίας, την οποία είχε καταραστεί ο θεός Απόλλωνας να μην πιστεύει κανείς τις προβλέψεις της για το μέλλον. Ίσως να έχει ονομαστεί έτσι ως ένα αστείο για την Oracle (που έχει ονομαστεί έτσι από το Μαντείο των Δελφών - Oracle at Delphi). **[26]**

9.3 Χαρακτηριστικά

9.3.1 Κατανεμημένη και αποκεντρωμένη

Η Cassandra είναι κατανεμημένη, πράγμα που σημαίνει πως είναι ικανή να λειτουργεί σε πολλαπλά μηχανήματα ενώ εμφανίζεται στους χρήστες ως ένα ενιαίο σύνολο. Στην πραγματικότητα, δεν υπάρχει πολύ νόημα στο να τρέχει η Cassandra σε έναν μόνο κόμβο. Γρήγορα συνειδητοποιεί κανείς πως χρειάζονται περισσότερες από μία μηχανές για να καταλάβουμε πόσο πολύ μπορεί να μας ωφελήσει η Cassandra. Μεγάλο μέρος του σχεδιασμού της και της βάσης του κώδικά της, είναι ειδικά κατασκευασμένο όχι μόνο για να δουλεύει σε πολλά διαφορετικά μηχανήματα, αλλά και για να βελτιστοποιεί την απόδοση σε πολλαπλά κέντρα δεδομένων διάσπαρτα γεωγραφικά. Μπορεί με βεβαιότητα κάποιος χρήστης να γράψει κάτι απ' οποιοδήποτε σημείο του συμπλέγματος και η Cassandra θα το ενσωματώσει. Σε άλλα συστήματα (όπως η MySQL ή το BigTable), όταν τα δεδομένα αυξάνονται, κάποιοι κόμβοι πρέπει να οριστούν ως masters ώστε να οργανώνουν τους υπόλοιπους κόμβους που έχουν οριστεί ως slaves. Η Cassandra ωστόσο είναι αποκεντρωμένη, το οποίο σημαίνει πως όλοι οι κόμβοι είναι ίδιοι. Δεν υφίσταται δηλαδή κόμβος στην Cassandra ο οποίος να εκτελεί λειτουργίες οργάνωσης διαφορετικές από άλλον. Αντιθέτως, η Cassandra διαθέτει ένα πρωτόκολλο peer-to-peer και χρησιμοποιεί το λεγόμενο 'gossip' για να διατηρήσει και να κρατήσει συγχρονισμένη τη λίστα με τους κόμβους που είναι ζωντανοί ή νεκροί. Το γεγονός ότι η Cassandra είναι αποκεντρωμένη, σημαίνει πως δεν έχει single point of failure. Όλοι οι κόμβοι σε ένα σύμπλεγμα λειτουργούν ακριβώς το ίδιο. Αυτό είναι γνωστό και ως server symmetry. Με λίγα λόγια, επειδή η Cassandra είναι κατανεμημένη και αποκεντρωμένη, δεν υπάρχει single point of failure και επομένως υπάρχει υψηλή διαθεσιμότητα των δεδομένων. **[26], [27]**

9.3.2 Ελαστική κλιμάκωση

Κλιμάκωση είναι το χαρακτηριστικό εκείνο ενός συστήματος, το οποίο επιτρέπει την εξυπηρέτηση μεγαλύτερου αριθμού αιτήσεων με μικρή ή καθόλου υποβάθμιση της απόδοσής του. Υπάρχουν δύο τρόποι να επιτευχθεί αυτό: η κάθετη κλιμάκωση που συνεπάγεται την προσθήκη hardware μεγαλύτερης χωρητικότητας και μνήμης στο υπάρχον μηχάνημα και η οριζόντια, που σημαίνει πρόσθεση περισσότερων μηχανημάτων καθένα από τα οποία έχει ένα μέρος ή το σύνολο των δεδομένων έτσι ώστε κανένα να μην χρειάζεται να φέρει όλο το βάρος των αιτήσεων μόνο του. Στην περίπτωση αυτή όμως, το λογισμικό πρέπει να διαθέτει έναν εσωτερικό μηχανισμό για να διατηρεί τα δεδομένα συγχρονισμένα μεταξύ των κόμβων του συμπλέγματος.

Ο όρος ελαστική κλιμάκωση αναφέρεται σε μια ιδιαίτερη ιδιότητα της οριζόντιας κλιμάκωσης. Σημαίνει ότι το σύμπλεγμά μας μπορεί να κάνει scale up ή scale down απρόσκοπτα. Για να γίνει αυτό, το σύμπλεγμα πρέπει να είναι σε θέση να δεχτεί νέους κόμβους που μπορούν να αρχίσουν να συμμετέχουν παίρνοντας ένα αντίγραφο μερικών ή όλων των δεδομένων χωρίς σημαντική διακοπή ή αναδιάρθρωση του συμπλέγματος. Δε χρειάζεται δηλαδή να επανεκκινήσουμε την εφαρμογή ή να αλλάξουμε τα ερωτήματα. Απλά προσθέτουμε ένα νέο μηχάνημα και η Cassandra θα το βρει και θα το βάλει αμέσως σε λειτουργία. **[26], [27]**

9.3.3 Υψηλή διαθεσιμότητα και ανοχή σε βλάβες

Η διαθεσιμότητα ενός συστήματος, μετράται σύμφωνα με τη δυνατότητά του να εκπληρώνει τις αιτήσεις. Οι υπολογιστές όμως μπορεί να πάθουν οτιδήποτε, από βλάβη hardware ως διακοπή δικτύου. Υπάρχουν βέβαια πολύ εξελιγμένα μηχανήματα (απαγορευτικά ακριβά βέβαια) που μπορούν να μετριάσουν πολλές από τις πιθανές βλάβες, αλλά ο καθένας μας μπορεί να βγάλει κατά λάθος κάποιο καλώδιο ethernet, με καταστροφικά αποτελέσματα σε ένα κέντρο δεδομένων. Έτσι, για να είναι ένα σύστημα υψηλά διαθέσιμο, θα πρέπει να περιλαμβάνει πολλούς δικτυωμένους υπολογιστές και το λογισμικό που τρέχει θα πρέπει να μπορεί να λειτουργήσει στο σύμπλεγμα των υπολογιστών αυτών και να έχει κάποια λειτουργία για αναγνώριση βλαβών σε κόμβους και αποτυχίας τους να ανταπεξέλθουν στις αιτήσεις.

Η Cassandra είναι υψηλά διαθέσιμη. Μπορούμε να αντικαταστήσουμε κόμβους ενός συμπλέγματος με βλάβη χωρίς να υπάρξει downtime και μπορούμε να μεταφέρουμε τα δεδομένα σε πολλά κέντρα δεδομένων ώστε να αποτραπεί η διακοπή λειτουργίας ακόμα και στην περίπτωση μεγάλης καταστροφής (πυρκαγιά ή πλημμύρα για παράδειγμα) σε ένα από αυτά. **[26], [27]**

9.3.4 Υψηλή απόδοση

Η Cassandra είναι ειδικά σχεδιασμένη για να επωφελείται πλήρως από μηχανήματα πολλών επεξεργαστών και να τρέχει σε πολλά τέτοια μηχανήματα μεταξύ πολλών κέντρων δεδομένων. Είναι αποδεδειγμένο πως ανταποκρίνεται εξαιρετικά κάτω από βαρύ φορτίο. Επίσης, μπορεί να παράγει πολύ μεγάλο αριθμό writes per second. Όσο αυξάνονται οι servers στο σύστημά μας, μπορούμε να διατηρήσουμε όλες τις επιθυμητές ιδιότητες της Cassandra χωρίς να θυσιάσουμε απόδοση. **[26], [27]**

9.3.5 Schema-Free

Η Cassandra απαιτεί από μας να ορίσουμε έναν χώρο που ονομάζεται keyspace και περιέχει τις οικογένειες στηλών, κάτι σαν πίνακες στις σχεσιακές βάσεις. Από κει και πέρα, δεν υπάρχει κάτι άλλο που να παραπέμπει σε schema. Οι πίνακες δεδομένων είναι αραιοί, έτσι ώστε να μπορούμε να προσθέτουμε στοιχεία χρησιμοποιώντας μόνο τις στήλες που θέλουμε. Δεν υπάρχει κανένας λόγος να καθορίζουμε από πριν τις στήλες. Μπορούμε να προσθέτουμε τυχόν νέες στήλες στην πορεία. Αντί δηλαδή να χτίζουμε ένα μοντέλο δεδομένων από πριν χρησιμοποιώντας ακριβά εργαλεία data-modeling, και να γράφουμε ερωτήματα με πολύπλοκα joins, με την Cassandra επικεντρωνόμαστε πρώτα στα ερωτήματα που μας ενδιαφέρει να απαντήσουμε και μετά χτίζουμε τα δεδομένα μας γύρω τους. **[26], [27]**

9.3.6 Row-oriented

Οι δομές δεδομένων της Cassandra αντιπροσωπεύονται με αραιά πολυδιάστατα hashtables. Με τον όρο αραιά, εννοούμε πως για μία δεδομένη σειρά μπορούμε να έχουμε μία ή περισσότερες στήλες, χωρίς να χρειάζεται κάθε σειρά να έχει τον ίδιο αριθμό στηλών με τις υπόλοιπες (όπως συμβαίνει στο σχεσιακό μοντέλο). Κάθε σειρά έχει ένα μοναδικό κλειδί το οποίο καθιστά τα δεδομένα του προσβάσιμα. Ακόμη, με τον τρόπο που αποθηκεύονται τα δεδομένα στην Cassandra, δε χρειάζεται να αποφασίζουμε εκ των προτέρων πως ακριβώς πρέπει να είναι τα δεδομένα μας ή τι πεδία θα μας χρειαστούν. Αυτό είναι πολύ χρήσιμο αν συμβαίνουν συχνές αλλαγές στα δεδομένα που αποθηκεύουμε. Όπως αναφέρθηκε και πιο πάνω, στην Cassandra πρώτα σκεφτόμαστε τα ερωτήματα για τα οποία θέλουμε απαντήσεις και μετά φτιάχνουμε τα δεδομένα οργανωμένα έτσι ώστε να τις δίνουν. **[26], [27]**

9.3.7 Ρυθμιζόμενη συνέπεια

Συνέπεια σημαίνει ουσιαστικά ότι το διάβασμα θα επιστρέφει πάντα την πιο πρόσφατη τιμή. Ας πάρουμε για παράδειγμα δύο πελάτες ενός e-shop που θέλουν να αγοράσουν το ίδιο εμπόρευμα. Αν το συγκεκριμένο εμπόρευμα είναι το τελευταίο του είδους του και ο ένας πελάτης το αγοράσει λίγο πριν τον άλλο, ο δεύτερος πρέπει να πληροφορηθεί ότι το εμπόρευμα αυτό δεν είναι πια διαθέσιμο για αγορά. Αυτό συμβαίνει εγγυημένα όταν υπάρχει συνέπεια μεταξύ των κόμβων που περιέχουν τα δεδομένα. Όπως θα δούμε όμως στη συνέχεια, η κλιμάκωση των εφαρμογών μας πολλές φορές προϋποθέτει ορισμένους συμβιβασμούς μεταξύ της συνέπειας των δεδομένων, της διαθεσιμότητας των κόμβων και της ανοχής διαχωρισμού.

Η Cassandra συχνά χαρακτηρίζεται «τελικά συνεπής», που είναι λίγο παραπλανητικό. Όντως ανταλλάσσει συνέπεια για να επιτύχει συνολική διαθεσιμότητα, αλλά θα ήταν πιο σωστός ο

χαρακτηρισμός της ως «ρυθμιζόμενα συνεπής». Αυτό σημαίνει πως μας δίνει τη δυνατότητα να ρυθμίσουμε οι ίδιοι το επίπεδο συνέπειας που επιθυμούμε. Η «τελική συνέπεια» είναι ένα από τα πολλά μοντέλα συνέπειας που είναι διαθέσιμα. **[26], [27]**

9.3.7.1 Αυστηρή συνέπεια

Μερικές φορές ονομάζεται και «διαδοχική συνέπεια» και είναι το πιο ψηλό επίπεδο συνέπειας. Απαιτεί ότι κάθε ανάγνωση κάποιου πεδίου θα επιστρέφει πάντα την πιο πρόσφατη τιμή του. Αυτό ακούγεται τέλειο, αλλά στην πραγματικότητα δεν είναι πάντα εύκολο. Αν η βάση μας βρίσκεται σε ένα μηχάνημα τότε φυσικά δεν αποτελεί πρόβλημα η συνέπεια έτσι κι αλλιώς. Σε ένα σύστημα όμως το οποίο λειτουργεί σε πολλά γεωγραφικά διασκορπισμένα κέντρα δεδομένων, το παραπάνω καθίσταται δύσκολο. Για την επίτευξή του απαιτείται ένα παγκόσμιο ρολόι ικανό να αποτυπώνει την ώρα (timestamping) για όλες τις λειτουργίες που εκτελούνται, ανεξάρτητα από την τοποθεσία των δεδομένων ή τον χρήστη που τα ζητά ή πόσες υπηρεσίες χρειάζονται για να καθορίσουν την απόκριση. Η Cassandra παρέχει τη δυνατότητα αυτή, χωρίς βέβαια να εγγυάται πως τα επίπεδα απόδοσης και διαθεσιμότητας των κόμβων θα είναι και τα υψηλότερα.

9.3.7.2 Αιτιώδης συνέπεια

Είναι μια ελαφρώς ασθενέστερη μορφή συνέπειας. Δεν χρησιμοποιεί το ρολόι που αναφέρθηκε προηγουμένως, το οποίο συγχρονίζει μεν όλες τις λειτουργίες αλλά δημιουργεί συμφόρηση. Αντί λοιπόν να στηρίζεται σε χρονοσφραγίδες, βασίζεται σε μια πιο σημασιολογική προσέγγιση. Προσπαθεί να καθορίσει την αιτία των γεγονότων για να δημιουργήσει κάποια συνέπεια στη σειρά τους. Αυτό σημαίνει πως τα writes που πιθανώς σχετίζονται μεταξύ τους, πρέπει να διαβάζονται με τη σειρά. Αν δύο διαφορετικές, άσχετες μεταξύ τους λειτουργίες ξαφνικά γράψουν στο ίδιο πεδίο, τότε τα writes αυτά θεωρούμε πως δεν σχετίζονται. Αν όμως το ένα write συμβαίνει μετά το άλλο, μπορούμε να συμπεράνουμε ότι σχετίζονται. Χαρακτηρίζεται επομένως 'αιτιώδης συνέπεια' γιατί οι αιτιώδεις εγγραφές πρέπει να διαβάζονται σειριακά.

9.3.7.3 Ασθενής (τελική) συνέπεια

Αυτή η μορφή συνέπειας σημαίνει πως τελικά όλες οι ενημερώσεις θα διαδοθούν σε όλα τα αντίγραφα σε ένα κατανεμημένο σύστημα, αλλά αυτό θα πάρει πιθανώς κάποιο χρόνο. Τελικά όμως όλα τα αντίγραφα θα είναι συνεπή. Η συνέπεια αυτή γίνεται πολύ ελκυστική αν σκεφτεί κανείς τι φόρτος απαιτείται για την επίτευξη των προηγούμενων, ισχυρότερων μορφών συνέπειας.

Η Cassandra προσφέρει δύο ρυθμίσεις σχετικά με την συνέπεια: τον συντελεστή αναπαραγωγής (replication factor) και το επίπεδο συνέπειας (consistency level).

Με την πρώτη ρύθμιση μπορούμε να αποφασίσουμε πόσο θέλουμε να 'πληρώσουμε σε απόδοση' για να κερδίσουμε σε συνέπεια. Με τον συντελεστή αναπαραγωγής ορίζουμε σε πόσους κόμβους του συστήματος θέλουμε να υπάρχουν τα δεδομένα μας. Με το επίπεδο συνέπειας ορίζουμε πόσα replicas του συμπλέγματος πρέπει να αναγνωρίσουν ένα write ή να ανταποκριθούν σε ένα read ώστε να θεωρηθεί επιτυχής η συγκεκριμένη λειτουργία. Επομένως, αν θέλουμε, μπορούμε να θέσουμε το επίπεδο συνέπειας ίσο με τον συντελεστή αναπαραγωγής, αποκτώντας καλύτερη συνέπεια στο κόστος όμως λειτουργιών κλειδώματος που περιμένουν να ενημερωθούν πρώτα όλοι οι κόμβοι και να δηλώσουν επιτυχή τη λειτουργία πριν επιστρέψει αποτέλεσμα η εφαρμογή. Αυτό βέβαια δεν προτείνεται για λόγους προφανείς (επηρεάζει την απόδοση, δε βοηθά στη διαθεσιμότητα και γενικότερα έρχεται σε αντίθεση με τους λόγους που θα 'θελε κανείς να χρησιμοποιήσει την Cassandra εξ αρχής). Αν όμως θέσουμε το επίπεδο συνέπειας σε τιμή μικρότερη του συντελεστή αναπαραγωγής, οι ενημερώσεις θα θεωρούνται επιτυχείς ακόμα και αν κάποιο κόμβοι είναι εκτός λειτουργίας (πετυχαίνοντας έτσι υψηλότερη διαθεσιμότητα).

9.4 Βασικές τεχνικές διαφορές Cassandra και υπόλοιπων NoSQL και RDBMSs

Καταρχάς, η αρχιτεκτονική με βάση την οποία έχει αναπτυχθεί η Cassandra είναι τέτοια ώστε να μπορεί να χειριστεί petabytes πληροφορίας και χιλιάδες ταυτόχρονους χρήστες κάθε δευτερόλεπτο με την ίδια ευκολία που χειρίζεται και μικρότερο όγκο δεδομένων και κίνησης.

Ακόμη, η επικοινωνία μεταξύ των κόμβων γίνεται με βάση το πρωτόκολλο peer-to-peer, σε αντίθεση με τα περισσότερα συστήματα στα οποία υπάρχει η φιλοσοφία master/slave. Το πρωτόκολλο αυτό αποτρέπει την ύπαρξη single point of failure (SPOF) για οποιαδήποτε διαδικασία και λειτουργία της βάσης.

Η αναπαραγωγή των δεδομένων είναι πολύ εύκολη διαδικασία. Με απλό τρόπο μπορούμε να προσθέσουμε κόμβους στο σύμπλεγμα και η Cassandra θα τους ενσωματώσει και θα τους χρησιμοποιήσει άμεσα.

Επίσης, είναι δυνατό να προστεθεί χωρητικότητα (για επιπλέον αντίγραφα των δεδομένων για παράδειγμα) online. Αυτό σημαίνει πως μπορούμε όποτε κρίνουμε αναγαίο να αυξάνουμε την απόδοση των reads και writes χωρίς να χρειάζεται να πέσει το σύστημά μας.

Η συνέπεια στην Cassandra είναι, όπως έχουμε αναφέρει, ρυθμιζόμενη. Αυτό σημαίνει πως μπορούμε αν θέλουμε να έχουμε την ανθεκτικότητα και προστασία ενός RDBMS, διατηρώντας όμως την επιλογή για πιο χαλαρή συνέπεια όταν το επιτρέπει η εφαρμογή.

Δεν υπάρχει ανάγκη να έχει η βάση μας προκαθορισμένο σχήμα. Το σχήμα στην Cassandra είναι ευέλικτο και δυναμικό ώστε να βολεύει στη διαχείριση big data.

Επιπλέον, η Cassandra προσφέρει τη δυνατότητα συμπίεσης των αρχείων, που φτάνει στη μείωση ακατέργαστων δεδομένων μέχρι και 80%.

Τέλος, η Cassandra μπορεί να λειτουργήσει και ως βάση δεδομένων στο νέφος (cloud DB). Αυτό σημαίνει πως υπάρχει η δυνατότητα αποθήκευσης και προσπέλασης των δεδομένων ανεξαρτήτου δικτύων, προσφέροντας έτσι τοπική ανεξαρτησία στην εφαρμογή. **[1]**

9.5 Use cases της Cassandra

9.5.1 Εφαρμογές πολλών κόμβων

Στην Cassandra έχει γίνει πολύ προσεκτική σχεδίαση όσον αφορά την υψηλή διαθεσιμότητα, τη ρυθμιζόμενη συνέπεια, το πρωτόκολλο peer-to-peer και την εύκολη κλιμάκωση. Καμία όμως από τις ιδιότητες αυτές δεν έχει νόημα σε σύστημα ενός μόνο κόμβου, αφού δε θα μπορούμε να τις αξιοποιήσουμε πλήρως. Πρέπει επομένως να κάνουμε κάποιους υπολογισμούς. Πρέπει να σκεφτούμε την αναμενόμενη κίνηση, τις ανάγκες για απόδοση και άλλες παραμέτρους. Δεν υπάρχει κάποιος απλός κανόνας για να μας βοηθήσει να πάρουμε την απόφαση. Αν πιστεύουμε πως μπορούμε να ικανοποιούμε την κίνηση με μερικές σχεσιακές βάσεις, ίσως πρέπει να επιλέξουμε αυτές. Αν από την άλλη πιστεύουμε πως χρειαζόμαστε από μερικούς έως πολλούς κόμβους για να λειτουργήσει ικανοποιητικά η εφαρμογή μας, η Cassandra θα είναι μάλλον αυτό που θέλουμε. **[26]**

9.5.2 Πολλά writes, στατιστικά και ανάλυση

Η Cassandra έχει βελτιστοποιηθεί για άριστη απόδοση σε writes. Πολλές από τις πρώτες χρήσεις της Cassandra περιλάμβαναν την αποθήκευση της δραστηριότητας χρηστών, χρήση κοινωνικών δικτύων, προτάσεις/σχόλια και στατιστικά της εφαρμογής. Τέτοιες περιπτώσεις περιλαμβάνουν πολύ γράψιμο και λιγότερες λειτουργίες ανάγνωσης. Επίσης οι ενημερώσεις μπορεί να μην είναι ισόποσα κατανεμημένες στο χρόνο και να συμβαίνουν ξαφνικές περίοδοι αιχμής. Η ικανότητα χειρισμού πολλαπλών clients ταυτόχρονα και γενικότερα μεγάλου όγκου εγγραφών είναι ένα από τα κύρια χαρακτηριστικά της Cassandra. **[26]**

9.5.3 Γεωγραφική κατανομή

Η Cassandra παρέχει υποστήριξη για τη γεωγραφική κατανομή των δεδομένων. Μπορούμε εύκολα να τη ρυθμίσουμε ώστε να αναπαράγει τα δεδομένα μεταξύ πολλών κέντρων δεδομένων. Αν για παράδειγμα έχουμε μία εφαρμογή παγκόσμιου επιπέδου και θα είχαμε όφελος όσον αφορά την απόδοση αν τα δεδομένα ήταν κοντά στο χρήστη, τότε η Cassandra θα μας ταίριαζε εξαιρετικά. **[26]**

9.5.4 Εφαρμογές σε εξέλιξη

Αν βρισκόμαστε σε πρωταρχικό στάδιο αλλά η εφαρμογή μας εξελίσσεται ραγδαία, η Cassandra θα μπορούσε να μας ταιριάζει λόγω του schema-free μοντέλου δεδομένων της. Το χαρακτηριστικό αυτό καθιστά εύκολο το να ανανεώνουμε τη βάση μας χωρίς κόπο όσο αυξάνονται οι ανάγκες της εφαρμογής. **[26]**

9.6 Μερικές εταιρείες που χρησιμοποιούν Cassandra

Το Twitter χρησιμοποιεί την Cassandra για ανάλυση πραγματικού χρόνου, για geolocation και θέσεις δεδομένων ενδιαφέροντος, καθώς και για την εξόρυξη δεδομένων σχετικά με τους χρήστες.

Το Mahalo τη χρησιμοποιεί ως το βασικό τρόπο αποθήκευσης των δεδομένων του.

Το Facebook τη χρησιμοποιεί για τον φάκελο εισερχομένων μηνυμάτων.

Το Digg τη χρησιμοποιεί ως τη βασικό τρόπο αποθήκευσης των δεδομένων του.

Το Rackspace τη χρησιμοποιεί για τις υπηρεσίες νέφους του, για έλεγχο και καταγραφή γεγονότων.

Το Reddit τη χρησιμοποιεί ως μόνιμη cache.

Το Ooyala τη χρησιμοποιεί για την αποθήκευση και εξυπηρέτηση analytics πραγματικού χρόνου για βίντεο.

Το SimpleGeo τη χρησιμοποιεί ως τη βασικό τρόπο αποθήκευσης για δεδομένα πραγματικού χρόνου που αφορούν τοποθεσίες.

Το OneSpot τη χρησιμοποιεί για ένα υποσύνολο του συνολικού όγκου των δεδομένων του.

Η Cassandra χρησιμοποιείται κι από άλλες γνωστές εταιρείες όπως η Cisco και η Platform64. Έχει αρχίσει επίσης να χρησιμοποιείται και στο Comcast και bee.tv για streaming τηλεόρασης σε διαδίκτυο και κινητά. Από τα παραπάνω καταλαβαίνουμε πως όλο και περισσότερες επιχειρήσεις βρίσκουν τα use cases της Cassandra ελκυστικά και την

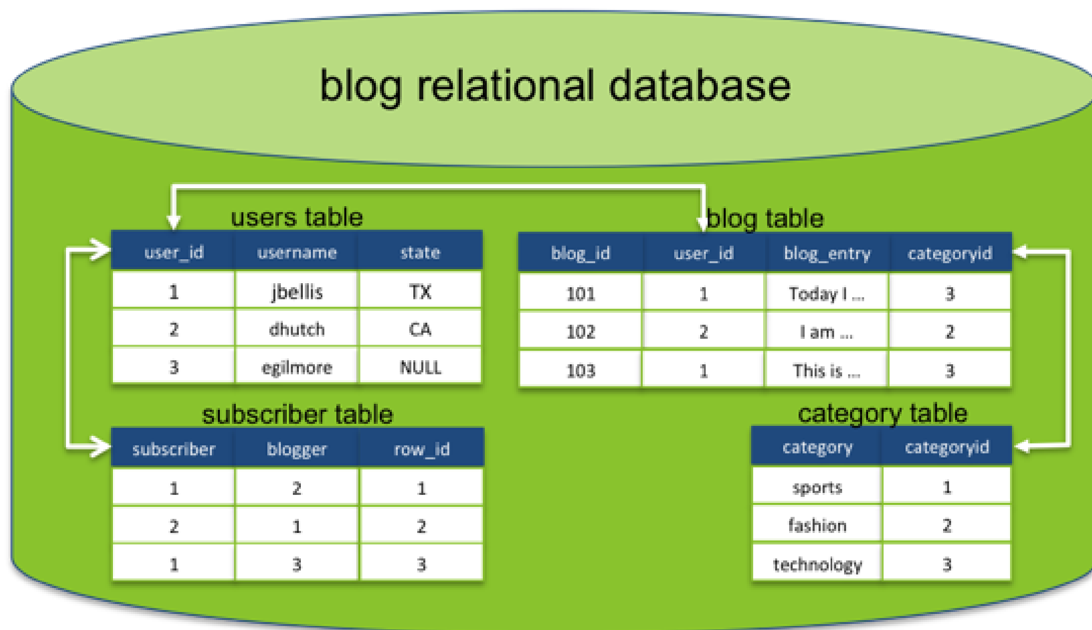
υιοθετούν. Η μεγαλύτερη γνωστή εγκατάσταση της Cassandra είναι στο Facebook, όπου είναι αποθηκευμένα πάνω από 200 TB δεδομένων σε περισσότερα από 100 μηχανήματα. Τη στιγμή που γράφεται η διπλωματική αυτή ολοένα και περισσότερες εταιρείες δοκιμάζουν την Cassandra για χρήση σε περιβάλλοντα παραγωγής. **[26]**

9.7 Εισαγωγή στο μοντέλο δεδομένων της Cassandra

Αντίθετα απ' ότι έχουμε συνηθίσει στις σχεσιακές βάσεις δεδομένων, με την Cassandra δε χρειάζεται να γνωρίζουμε εκ των προτέρων όλες τις στήλες που χρειάζεται η εφαρμογή μας, αφού δε χρειάζεται κάθε γραμμή να έχει τον ίδιο αριθμό στηλών με τις υπόλοιπες. Οι επιπλέον στήλες και τα μεταδεδομένα τους μπορούν να προστεθούν στην εφαρμογή μας εφόσον χρειάζονται χωρίς να επιβαρυνθεί το σύστημά μας με downtime.

Αν και είναι φυσιολογικό να θέλουμε να συγκρίνουμε τη δομή του μοντέλου δεδομένων της Cassandra με αυτή μιας σχεσιακής βάσης, κάτι τέτοιο είναι αρκετά δύσκολο αφού είναι πολύ διαφορετικές. Στις σχεσιακές βάσεις τα δεδομένα αποθηκεύονται σε πίνακες. Σε μία εφαρμογή τέτοιοι πίνακες συνήθως υπάρχουν πολλοί και μάλιστα σχετίζονται μεταξύ τους. Έτσι, τα δεδομένα κανονικοποιούνται ώστε να μειώνονται οι περιττές καταχωρήσεις και οι πίνακες συνδέονται με κλειδιά για να ικανοποιήσουν τα ερωτήματα.

Ας σκεφτούμε για παράδειγμα μία απλή εφαρμογή που επιτρέπει στους χρήστες να δημιουργούν καταχωρήσεις σε ένα blog. Οι καταχωρήσεις αυτές κατηγοριοποιούνται με βάση τη θεματική τους περιοχή (αθλητικά, μόδα, τεχνολογία κλπ). Οι χρήστες έχουν επίσης τη δυνατότητα να εγγράφονται στα blogs των άλλων χρηστών. Ας υποθέσουμε ότι το πρωτεύον κλειδί κάθε χρήστη στον πίνακα «Χρήστες» είναι το id του, η οποία είναι και ξένο κλειδί στους πίνακες «Blog» και «Συνδρομητές». Αντίστοιχα, το id της κατηγορίας είναι πρωτεύον κλειδί στον πίνακα «Κατηγορία» και ξένο κλειδί στον πίνακα «Blog_entries». Χρησιμοποιώντας αυτό το σχεσιακό μοντέλο, για να απαντηθούν ερωτήματα όπως «δείξε μου όλες τις καταχωρίσεις του τάδε χρήστη που αφορούν μόδα» ή «δείξε μου ποιοί χρήστες έχουν εγγραφεί στο blog μου», πρέπει να εκτελεσθούν joins σε διάφορους πίνακες.



Εικόνα 6. Σχήμα σχεσιακής βάσης

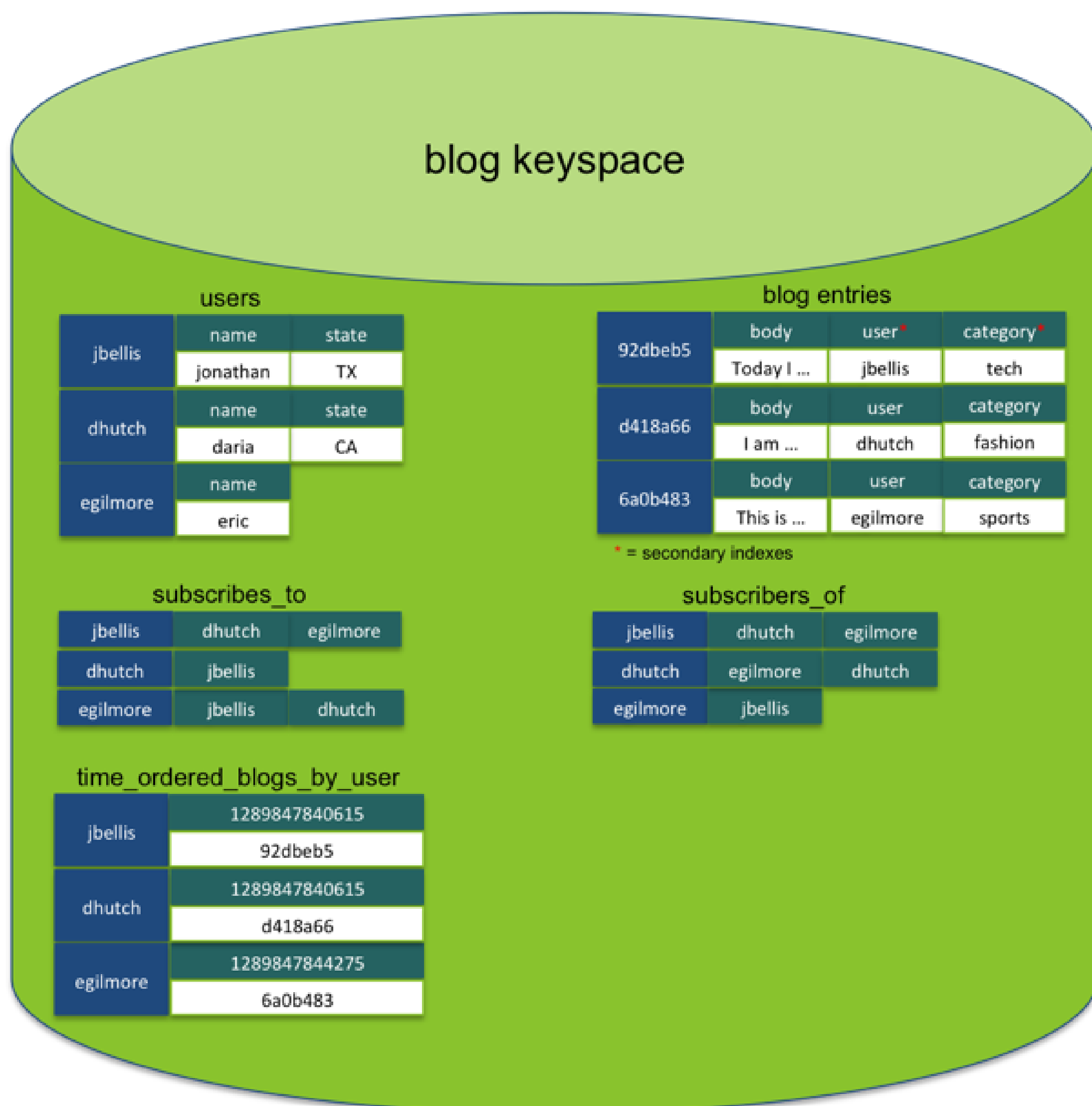
Στην Cassandra, ο χώρος που περιέχει τα δεδομένα μας ονομάζεται keyspace. Είναι το αντίστοιχο του σχήματος (database schema) σε μία σχεσιακή βάση. Μέσα στο keyspace βρίσκονται οι οικογένειες στηλών (column families) που είναι το αντίστοιχο των πινάκων. Κάθε οικογένεια περιλαμβάνει ένα σύνολο σχετιζόμενων στηλών και προσδιορίζεται από ένα κλειδί που παρέχεται από την εφαρμογή. Όπως προαναφέρθηκε, δεν είναι απαραίτητο κάθε γραμμή μιας οικογένειας να έχει τον ίδιο αριθμό στηλών. Παρά το γεγονός ότι οι οικογένειες στηλών είναι πολύ ευέλικτες, στην πράξη δεν είναι εντελώς schema-less. Υπάρχουν δύο τυπικά σχεδιαστικά πρότυπα οικογενειών στηλών στην Cassandra: οι στατικές και οι δυναμικές οικογένειες στηλών.

Η στατική οικογένεια χρησιμοποιεί ένα σχετικά στατικό σύνολο ονομάτων στηλών και είναι παρόμοια με τον πίνακα των σχεσιακών βάσεων. Η δυναμική οικογένεια εκμεταλλεύεται την ικανότητα της Cassandra να χρησιμοποιεί ορισμένα από την εφαρμογή μας, αυθαίρετα ονόματα στηλών για την αποθήκευση των δεδομένων. Μας επιτρέπει να υπολογίσουμε εκ των προτέρων τα αποτελέσματα που αναμένουμε και να τα αποθηκεύσουμε σε μία γραμμή ώστε η ανάκτηση των δεδομένων να είναι αποτελεσματικότερη. Στην ουσία, κάθε γραμμή μιας δυναμικής οικογένειας στηλών είναι ένα στιγμιότυπο της πληροφορίας που αποτελεί απάντηση σε ένα δεδομένο ερώτημα. Κάτι σαν όψη (view) στις σχεσιακές βάσεις. Παράδειγμα μιας τέτοιας οικογένειας θα ήταν μία η οποία κρατά τους χρήστες που εγγράφονται στο blog κάποιου άλλου χρήστη.

Η Cassandra δεν επιβάλλει σχέσεις μεταξύ των οικογενειών στηλών όπως κάνουν οι σχεσιακές βάσεις μεταξύ των πινάκων: δεν υπάρχουν ξένα κλειδιά και δεν υποστηρίζεται η ένωση οικογενειών την ώρα εκτέλεσης του ερωτήματος. Κάθε οικογένεια στηλών έχει ένα

αυτοτελές σύνολο στηλών που προορίζεται για την ικανοποίηση συγκεκριμένων ερωτήσεων της εφαρμογής μας.

Ας πάμε πάλι στο παράδειγμα της εφαρμογής blog που είδαμε προηγουμένως από τη σκοπιά των σχεσιακών βάσεων. Μπορούμε πάλι να έχουμε μία οικογένεια στηλών που κρατάει τα στοιχεία των χρηστών, αντίστοιχη με τον πίνακα «Χρήστες». Όμοια και για τον πίνακα «Blog_entries». Στη συνέχεια μπορούν να προστεθούν οικογένειες στηλών ανάλογα με το τι ερωτήματα καλείται να απαντήσει η εφαρμογή μας. Αν για παράδειγμα θέλω απαντήσεις σε ερωτήματα όπως «δείξε μου όλες τις καταχωρίσεις του τάδε χρήστη που αφορούν μόδα» ή «δείξε μου ποιοί χρήστες έχουν εγγραφεί στο blog μου», θα προσθέσω τις αντίστοιχες οικογένειες στηλών, όπως φαίνεται στο σχήμα. Προφανώς, για να γίνει κάτι τέτοιο συνήθως χρειάζεται αποκανονικοποίηση των δεδομένων. **[28]**



Εικόνα 7. Σχήμα μη σχεσιακής βάσης

9.8 Βασικές Λειτουργίες στην Cassandra

9.8.1 Writes στην Cassandra

Η Cassandra είναι βελτιστοποιημένη για πολύ γρήγορο και υψηλής διαθεσιμότητας γράψιμο δεδομένων. Στις σχεσιακές βάσεις, τυπικά σχηματίζουμε τους πίνακες με τέτοιο τρόπο ώστε να κρατάνε την επικάλυψη των δεδομένων στο ελάχιστο. Τα διάφορα κομμάτια πληροφορίας που χρειάζονται για να ικανοποιήσουν κάποιο ερώτημα είναι αποθηκευμένα σε διάφορους σχετιζόμενους πίνακες οι οποίοι τηρούν μία προκαθορισμένη δομή. Όμως, επειδή είναι έτσι αποθηκευμένα τα δεδομένα σε μια σχεσιακή βάση, το γράψιμο τους είναι ακριβό αφού ο server πρέπει να κάνει παραπάνω δουλειά για να είναι σίγουρος για την ακεραιότητα των δεδομένων μεταξύ των διαφόρων σχετιζομένων πινάκων. Επομένως οι σχεσιακές βάσεις δεν έχουν τόσο καλή απόδοση όσον αφορά τα writes.

Αντιθέτως, η Cassandra είναι βελτιστοποιημένη για μέγιστη απόδοση όσον αφορά τα writes. Τα writes γράφονται αρχικά σε ένα commit log (για μονιμότητα) και στη συνέχεια σε μία in-memory δομή πινάκων που ονομάζεται memtable. Ένα write θεωρείται πετυχημένο όταν γραφτεί στο commit log και στο memtable, οπότε υπάρχει ελάχιστος φόρτος I/0 όσον αφορά το δίσκο αφού όλα συμβαίνουν στη μνήμη. Τα writes στοιβάζονται λοιπόν στη μνήμη και γράφονται περιοδικά στον δίσκο σε μία ανθεκτική δομή πινάκων που ονομάζεται SSTable (sorted string table). Τα memtables και τα SSTables διατηρούνται ανά column family. Τα memtables οργανώνονται σε ταξινομημένη σειρά με βάση το key κάθε γραμμής και μεταφέρουν τα δεδομένα τους στα SSTables σειριακά.

Τα SSTables είναι αμετάβλητα (δεν ξαναγράφεται κάτι σε ένα SSTable αφού αυτό γίνει flush). Αυτό σημαίνει πως μία γραμμή αποθηκεύεται τυπικά σε πολλαπλά αρχεία SSTable. Κατά το χρόνο ανάγνωσης, μία γραμμή πρέπει να συνδυαστεί από όλα τα SSTables στο δίσκο (καθώς και από τα memtables που δεν έχουν ακόμα γίνει flush) για να προκύψουν τα ζητούμενα δεδομένα. Για τη βελτιστοποίηση αυτής της διαδικασίας-παζλ, η Cassandra χρησιμοποιεί μια in-memory δομή που ονομάζεται 'bloom filter'. Σε κάθε SSTable αντιστοιχίζεται ένα bloom filter το οποίο χρησιμοποιείται για να ελέγξει αν υπάρχει το ζητούμενο κλειδί σειράς στο συγκεκριμένο SSTable πριν γίνει οποιαδήποτε αναζήτηση στο δίσκο. **[28]**

9.8.2 Δοσοληψίες και έλεγχος συγχρονισμού

Σε αντίθεση με σχεσιακές βάσεις δεδομένων, η Cassandra δεν προσφέρει συναλλαγές πλήρως συμβατές με το πρότυπο ACID. Δεν υπάρχουν κλειδώματα ή εξαρτήσεις μεταξύ δοσοληψιών κατά την ταυτόχρονη ενημέρωση πολλών σειρών ή οικογενειών στηλών.

Η Cassandra ανταλλάσσει τις ιδιότητες ατομικότητα (Atomicity) και απομόνωση (Isolation) για υψηλή διαθεσιμότητα και ταχύτατη απόδοση διαβάσματος και εγγραφής (reads και writes). Στην Cassandra, μία εγγραφή είναι ατομική σε επίπεδο γραμμής, που σημαίνει πως εισαγωγή ή ενημέρωση στηλών για ένα συγκεκριμένο κλειδί σειράς αντιμετωπίζεται ως μία λειτουργία εγγραφής. Η Cassandra δεν υποστηρίζει δοσοληψίες με την έννοια της ομαδοποίησης πολλών ενημερώσεων γραμμών σε μία όλα-ή-τίποτα λειτουργία, ούτε κάνει roll back όταν κάποιο write πετυχαίνει σε ένα από τα αντίγραφα αλλά αποτυχαίνει σε άλλα. Είναι πιθανό να έχουμε κάποια αναφορά σφάλματος εγγραφής στον client, αλλά παρόλα αυτά να έχει γίνει η εγγραφή σε κάποιο αντίγραφο.

Για παράδειγμα, αν χρησιμοποιούμε το επίπεδο συνοχής εγγραφών(write consistency level, ρύθμιση της Cassandra) QUORUM με συντελεστή (replication factor) 3, η Cassandra θα στείλει την εγγραφή σε 2 αντίγραφα. Αν το write αποτύχει σε ένα από τα 2 αντίγραφα αλλά επιτύχει στο άλλο, η Cassandra θα αναφέρει αποτυχία εγγραφής στον client αλλά το write δεν γίνεται αυτομάτως roll back στο άλλο αντίγραφο.

Η Cassandra χρησιμοποιεί χρονοσφραγίδες (timestamps) για αν καθορίσει την πιο πρόσφατη ενημέρωση σε μια στήλη. Οι χρονοσφραγίδες αυτές παρέχονται από την εφαρμογή. Το τελευταίο timestamp κερδίζει πάντοτε όταν ζητά πληροφορίες, οπότε αν υπάρχουν πολλαπλές ενημερώσεις ίδιων στηλών μιας σειράς ταυτόχρονα, θα επικρατήσει τελικά η πιο πρόσφατη.

Οι εγγραφές στην Cassandra είναι ανθεκτικές. Όλες οι εγγραφές σε έναν κόμβο καταγράφονται τόσο στη μνήμη όσο και στο commit log πριν αναγνωριστούν ως επιτυχείς. Αν συμβεί κάποιο crash ή οποιαδήποτε αποτυχία του server πριν γίνουν flush τα memory tables στον δίσκο, τότε κατά την επανεκκίνηση του server αναπαράγεται το commit log για την ανάκτηση των χαμένων writes. **[28]**

9.8.3 Inserts και Updates

Στην Cassandra μπορεί να εισαχθεί ταυτόχρονα οποιοσδήποτε αριθμός στηλών. Κατά την εισαγωγή ή ενημέρωση στηλών σε μια οικογένεια στηλών, η εφαρμογή (client application) είναι αυτή που καθορίζει το κλειδί σειράς για να προσδιοριστεί πια αρχεία πρέπει να ενημερωθούν. Το κλειδί σειράς είναι παρόμοιο με το πρωτεύον κλειδί στο ότι πρέπει να είναι μοναδικό για κάθε σειρά μίας οικογένειας στηλών. Παρόλα αυτά, σε αντίθεση με ένα πρωτεύον κλειδί, εισάγοντας ένα ίδιο κλειδί σειράς δε θα οδηγήσει σε παραβίαση του περιορισμού μοναδικότητας (Use constraint violation) αλλά θα αντιμετωπιστεί στην ουσία σαν UPSERT (που σημαίνει update if found, insert if not found). Δηλαδή αν υπάρχουν οι συγκεκριμένες στήλες σε αυτή τη γραμμή θα τις ενημερώσει, ενώ αν δεν υπάρχουν θα τις εισάγει.

Επανεγγραφή γίνεται μόνο στην περίπτωση που η χρονοσφραγίδα της νέας έκδοσης της στήλης είναι πιο πρόσφατη από την υπάρχουσα, επομένως χρειάζονται ακριβείς χρονοσφραγίδες αν είναι συχνές οι ενημερώσεις. Οι χρονοσφραγίδες παρέχονται από τον client, επομένως πρέπει τα ρολόγια όλων των clients να είναι συγχρονισμένα (χρησιμοποιώντας π.χ. NTP - network time protocol). **[28]**

9.8.4 Deletes

Κατά τη διαγραφή κάποιας γραμμής ή στήλης στην Cassandra, υπάρχουν μερικά πράγματα που πρέπει να γνωρίζει κανείς που είναι διαφορετικά απ' ότι θα περιμέναμε σε μια σχεσιακή βάση.

Τα διαγραμμένα δεδομένα δε σβήνονται αμέσως από το δίσκο. Όταν εισάγουμε δεδομένα στην Cassandra, αυτά υπάρχουν και στα SSTables πριν μπουν στο δίσκο. Όταν γραφτεί ένα SSTable, όπως έχουμε προαναφέρει, είναι αμετάβλητο (δεν αλλάζει από επόμενες DML εργασίες- όπως inserts, updates, deletes). Αυτό σημαίνει πως μια διαγραμμένη στήλη δεν απομακρύνεται αμέσως. Αντίθετα, τοποθετείται ένας δείκτης στην στήλη αυτή που ονομάζεται 'tombstone' για να δείξει το νέο status της. Στήλες οι οποίες είναι επισημασμένες με tombstone συνεχίζουν να υπάρχουν για ένα προκαθορισμένο χρονικό διάστημα (η οποία καθορίζεται από την παράμετρο gc_grace_seconds που αντιστοιχεί σε κάθε οικογένεια στηλών), και διαγράφονται οριστικά αφού λήξει το διάστημα αυτό.

Σήμανση μιας διαγραμμένης στήλης με tombstone εξασφαλίζει ότι ένα αντίγραφο το οποίο βρίσκεται σε κόμβο που για οποιοδήποτε λόγο ήταν πεσμένος τη στιγμή της διαγραφής, θα λάβει τελικά την πληροφορία της διαγραφής όταν επανέλθει σε λειτουργία. Παρόλα αυτά, αν ο κόμβος είναι εκτός λειτουργίας για περισσότερο χρόνο απ' όσο καθορίζει η παράμετρος gc_grace_seconds, τότε πιθανότατα ο κόμβος αυτός να χάσει τη διαγραφή εντελώς και να αναπαραχθούν τα δεδομένα όταν επανέλθει. Για να αποφύγουμε μία τέτοια περίπτωση επανεμφάνισης διαγραμμένων στοιχείων, ο διαχειριστής πρέπει να τρέχει τακτικά επισκευή κόμβων σε κάθε κόμβο του συμπλέγματος (η προεπιλογή είναι κάθε 10 μέρες, ανάλογα με το πόσο συχνά γίνονται deletes και πόσο συχνά έχουμε αποτυχίες κόμβων η παράμετρος αυτή μπορεί να αυξηθεί ή να μειωθεί).

Το κλειδί σειράς μιας διαγραμμένης σειράς μπορεί να εμφανίζεται ακόμα σε αποτελέσματα ερωτημάτων εύρους. Όταν διαγράφουμε μια σειρά στην Cassandra, όλες οι στήλες της σειράς αυτής σηματοδοτούνται με tombstone. Μέχρι να εκκαθαριστούν τα tombstone αυτά, υπάρχει ένα άδειο κλειδί γραμμής (μια γραμμή η οποία δεν περιέχει στήλες). Επομένως αν η εφαρμογή μας εκτελεί ερωτήματα εύρους στις γραμμές κάποιας οικογένειας στηλών, ίσως πρέπει να φιλτράρουμε τα αποτελέσματα ώστε να μην επιστρέφονται κλειδιά σειρών με κενές στήλες. **[28]**

9.8.5 Reads στην Cassandra

Όπως έχουμε προαναφέρει, όταν φτάνει σε έναν κόμβο αίτημα για ανάγνωση, η γραμμή που θα επιστραφεί θα προκύψει ως συνδυασμός τόσο από τα SSTables στον κόμβο αυτό που περιέχουν στήλες που αφορούν τη γραμμή που μας ενδιαφέρει, όσο και από memtables που δεν έχουν γίνει ακόμα flush. Για το σκοπό αυτό υπάρχει και το 'bloom-filter' που έχει περιγραφεί παραπάνω. Ως αποτέλεσμα, η Cassandra είναι πολύ αποδοτική συγκριτικά με άλλα συστήματα αποθήκευσης όσον αφορά τα reads, ακόμα και σε βαρύ φόρτο εργασίας. Όπως και με κάθε βάση δεδομένων, η ανάγνωση είναι ταχύτερη όταν τα πιο περιζήτητα δεδομένα βρίσκονται στη μνήμη. Παρά το γεγονός ότι όλα τα σύγχρονα συστήματα αποθήκευσης βασίζονται σε κάποια μορφή προσωρινής αποθήκευσης (caching) για να είναι γρηγορότερη η πρόσβαση σε συχνά ζητούμενα δεδομένα, δεν αποδίδουν όλα τόσο καλά όταν υπερβαίνεται η χωρητικότητα της προσωρινής μνήμης και χρειάζεται επικοινωνία με το δίσκο. Η απόδοση των αναγνώσεων στην Cassandra επωφελείται από το ενσωματωμένο caching της αλλά δεν πέφτει και δραματικά όταν απαιτούνται τυχαία reads στον δίσκο. Όταν αρχίζει να αυξάνεται η δραστηριότητα του δίσκου λόγω αυξημένου φόρτου αναγνώσεων, μπορούμε εύκολα να το διορθώσουμε με την προσθήκη περισσότερων κόμβων στο σύμπλεγμα. Για γραμμές στις οποίες γίνονται συχνά αναγνώσεις, η Cassandra έχει μια ενσωματωμένη cachce κλειδιών. Και γενικότερα έχει πολλές επιλογές για βελτιστοποίηση της απόδοσης με ενσωματωμένες λειτουργίες caching. **[28]**

9.8.6 Client Requests στην Cassandra

Όλοι οι κόμβοι σε κάποιο σύμπλεγμα της Cassandra είναι peers. Για να ικανοποιηθεί ένα αίτημα εγγραφής ή ανάγνωσης ενός client μπορεί να χρησιμοποιηθεί οποιοσδήποτε κόμβος του συμπλέγματος. Από τη στιγμή που κάποιος client συνδεθεί με κάποιον κόμβο, ο κόμβος αυτός λειτουργεί ως συντονιστής για τα αιτήματα του client αυτού.

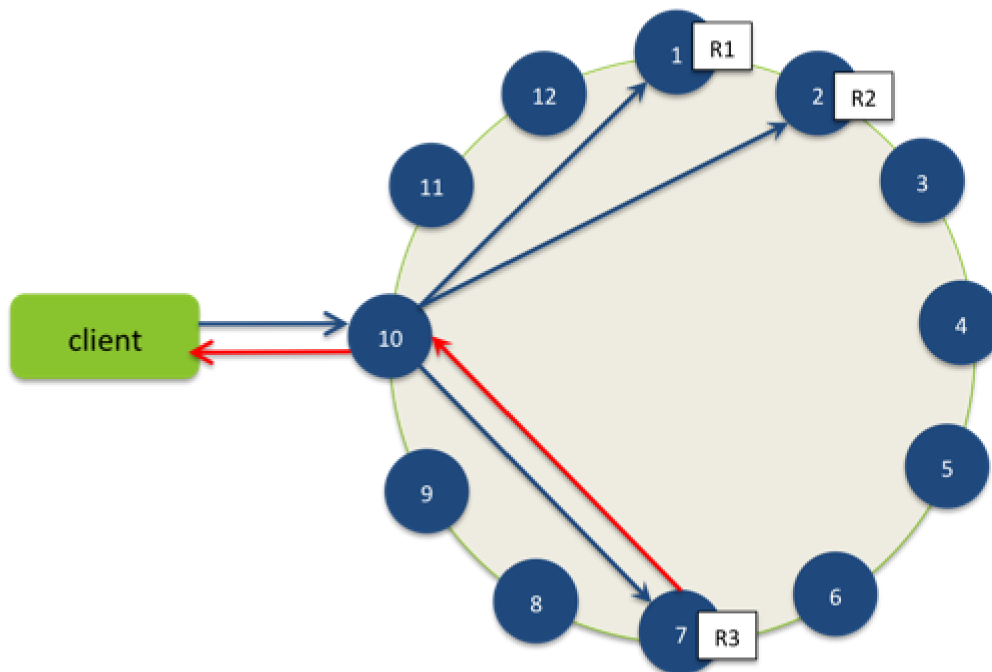
Η δουλειά ενός κόμβου-συντονιστή είναι να λειτουργεί ως proxy μεταξύ της εφαρμογής (του client) και των κόμβων στους οποίους βρίσκονται οι πληροφορίες που ζητούνται. Ο συντονιστής αποφασίζει ποιοι κόμβοι του δαχτυλιδιού πρέπει να λάβουν το αίτημα με βάση τη στρατηγική αναπαραγωγής και διαμέρισης που ακολουθείται (replica and partitioner strategy). **[28]**

9.8.6.1 Write Requests

Όσον αφορά τις εγγραφές, ο συντονιστής τις στέλνει σε όλα τα αντίγραφα (replicas) που έχουν τη γραμμή που πρόκειται να γραφτεί. Εφόσον όλοι οι κόμβοι στους οποίους βρίσκονται αντίγραφα λειτουργούν και είναι διαθέσιμοι, θα λάβουν το write ανεξάρτητα από

το επίπεδο συνέπειας (consistency level) που είναι ορισμένο στον client. Το επίπεδο συνέπειας εγγραφής είναι η παράμετρος που καθορίζει πόσοι κόμβοι πρέπει να ανταποκριθούν με επιτυχία ώστε να θεωρηθεί επιτυχής μια εγγραφή.

Για παράδειγμα, σε ένα σύμπλεγμα μονού κέντρου δεδομένων (single data server cluster) που αποτελείται από 10 κόμβους και έχει συντελεστή αναπαραγωγής 3, μία εισερχόμενη εγγραφή θα πάει σε 3 κόμβους. Αν το επίπεδο συνέπειας εγγραφής είναι ONE, τότε τη στιγμή που θα ολοκληρωθεί η εγγραφή σε έναν από τους 3 κόμβους, ο κόμβος αυτός θα στείλει μήνυμα στον συντονιστή ο οποίος με τη σειρά του θα το στείλει πίσω στον client. Επίπεδο συνέπειας ONE σημαίνει πως υπάρχει περίπτωση δύο από τους τρείς κόμβους να χάσουν το write αν τύχει να έχουν κάποιο πρόβλημα τη στιγμή που έγινε το αίτημα. Παρόλο που θα το έχουν χάσει όμως, τελικά θα ξαναυπάρξει συνέπεια μέσω κάποιο από τους ενσωματωμένους μηχανισμούς της Cassandra (hinted handoff, read repair, anti-entropy node repair). Γι αυτό και λέμε ότι στην Cassandra -και στις άλλες NoSQL βάσεις- έχουμε eventual consistency.

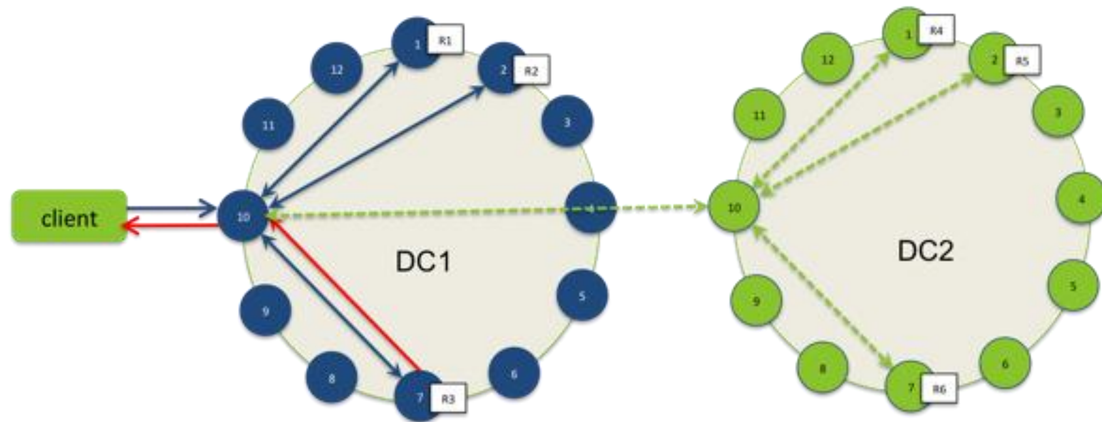


Εικόνα 8. Αίτημα ενός client για γράψιμο σε single Data Center

9.8.6.2 Multi-Data Center Write Requests

Αν η εφαρμογή μας είναι μεγαλύτερη και χρειάζονται περισσότερα του ενός κέντρα δεδομένων, η διαδικασία των αιτημάτων εγγραφών έχει ως εξής: η Cassandra επιλέγει έναν κόμβο-συντονιστή σε κάθε κέντρο για να χειρίζεται τα αιτήματα προς τους κόμβους όπου βρίσκονται τα αντίγραφα στο κέντρο αυτό. Όταν ο client επικοινωνεί με τον κόμβο-συντονιστή που του έχει αντιστοιχηθεί, αυτός απλά προωθεί το αίτημα εγγραφής σε έναν κόμβο σε κάθε κέντρο.

Αν χρησιμοποιείται επίπεδο συνέπειας ONE ή LOCAL_QUORUM, αρκεί να ανταποκριθούν οι κόμβοι του ίδιου κέντρου με τον συντονιστή για να θεωρηθεί το αίτημα εγγραφής επιτυχές. Φυσικά, όπως αναφέρθηκε και προηγουμένως, οι ενσωματωμένοι μηχανισμοί της Cassandra θα φροντίσουν ώστε να ενημερωθούν και τα άλλα κέντρα για την αλλαγή. Με αυτόν τον τρόπο δεν επηρεάζεται ο χρόνος ανταπόκρισης στα ερωτήματα λόγω γεωγραφικής καθυστέρησης (latency).



Εικόνα 9. Αίτημα ενός client για γράψιμο σε multi Data Center

9.8.6.3 Read Requests

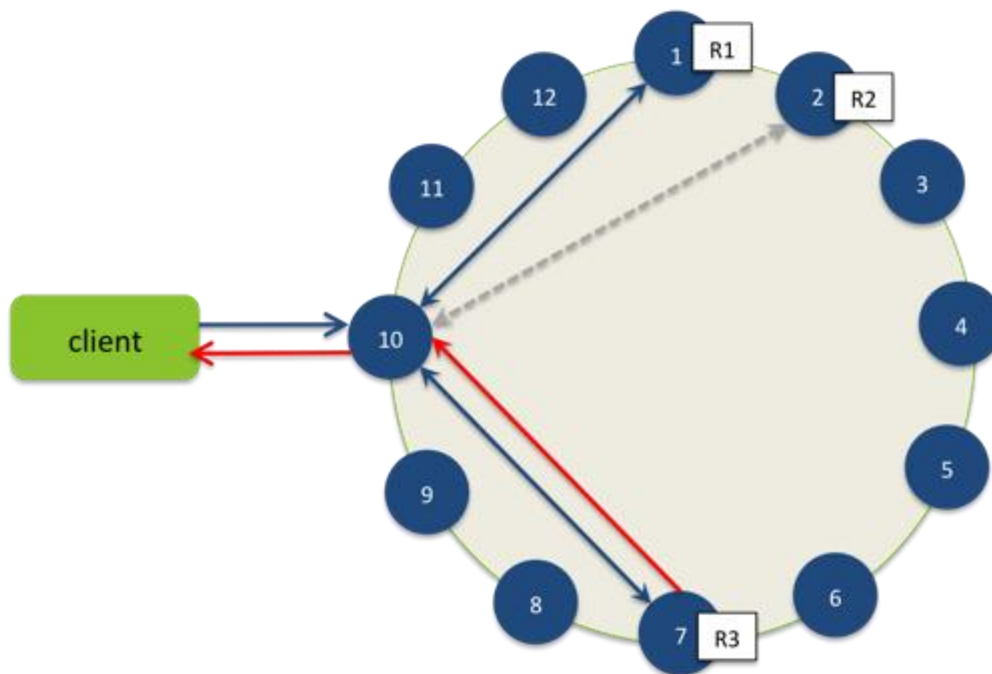
Όσον αφορά τις αναγνώσεις, υπάρχουν δύο είδη αιτήσεων ανάγνωσης που μπορεί να στείλει ο συντονιστής σε κάποιο αντίγραφο. Ένα άμεσο αίτημα ανάγνωσης κι ένα έμμεσο. Ο αριθμός των αντιγράφων της βάσης με τα οποία έρχεται σε επαφή ο συντονιστής με άμεσο αίτημα καθορίζεται από το επίπεδο συνέπειας που ορίζει ο client. Τα έμμεσα αιτήματα στέλνονται σε αντίγραφα τα οποία δεν έλαβαν άμεσο αίτημα. Τα αιτήματα αυτά απλά εξασφαλίζουν ότι η γραμμή που εξετάζεται είναι συνεπής σε όλα τα αντίγραφα.

Έτσι, ο συντονιστής πρώτα επικοινωνεί με τα αντίγραφα που καθορίζονται από το επίπεδο συνέπειας. Στέλνει τις αιτήσεις σε αυτά που ανταποκρίνονται αμέσως. Οι κόμβοι με τους οποίους έχει έρθει σε επαφή ο συντονιστής απαντάνε κατευθείαν με τα ζητούμενα δεδομένα. Αν έχει γίνει αίτημα σε πολλούς κόμβους, τότε οι γραμμές κάθε αντιγράφου συγκρίνονται μεταξύ τους για να επαληθευτεί η συνέπειά τους. Αν υπάρχει ασυνέπεια, τότε συμβαίνουν τα εξής:

Ανεξάρτητα από το πως έχει ρυθμιστεί η παράμετρος read_repair_chance, γίνεται μια επισκευή αναγνώσεων στα δεδομένα. Ο συντονιστής χρησιμοποιεί το αντίγραφο που έχει τα πιο πρόσφατα δεδομένα (με βάση τη χρονοσφραγίδα) για να διαβιβάσει το αποτέλεσμα στον client. Στο παρασκήνιο, ο συντονιστής συγκρίνει τα δεδομένα από όλα τα υπόλοιπα αντίγραφα στα οποία υπάρχει η συγκεκριμένη γραμμή. Αν παρατηρηθεί κάποια ασυνέπεια,

τότε ο συντονιστής ενημερώνει τα out-of-date αντίγραφα ώστε να έχουν ίδια δεδομένα με το πιο πρόσφατο.

Η διαδικασία αυτή είναι στην ουσία επιδιόρθωση μέσω ανάγνωσης και είναι ενεργοποιημένη από προεπιλογή. Για παράδειγμα, σε ένα σύμπλεγμα με συντελεστή αναπαραγωγής 3 και επίπεδο συνέπειας QUORUM, 2 στα 3 αντίγραφα που έχουν τη γραμμή που εξετάζεται πρέπει να εκπληρώσουν το αίτημα ανάγνωσης. Αν τα τρία αυτά αντίγραφα έχουν διαφορετικές εκδόσεις της γραμμής, τότε θα επέστρεφε το αποτέλεσμα μόνο αυτό με την πιο πρόσφατη έκδοση. Στο παρασκήνιο, τα άλλα αντίγραφα ελέγχονται για συνέπεια με βάση αυτό και ενημερώνονται κατάλληλα.



Εικόνα 10. Αίτημα ενός client για διάβασμα σε single Data Center

9.8.7 Αναπαραγωγή δεδομένων στην Cassandra

Αναπαραγωγή (replication) είναι η διαδικασία αποθήκευσης αντιγράφων δεδομένων σε πολλαπλούς κόμβους για να εξασφαλίζεται αξιοπιστία και ανοχή σφαλμάτων.

Η Cassandra αποθηκεύει τα αντίγραφα (που τα ονομάζει replicas) κάθε σειράς βασισμένα στο κλειδί της σειράς. Όταν δημιουργούμε ένα keyspace, ορίζουμε και τη μέθοδο με βάση την οποία αποθηκεύονται τα αντίγραφα (replica placement strategy). Εκτός από τον ορισμό του αριθμού των αντιγράφων, η μέθοδος αυτή καθορίζει τον τρόπο κατανομής των αντιγράφων μεταξύ των κόμβων στο σύμπλεγμα (cluster), ανάλογα και με την τοπολογία του cluster.

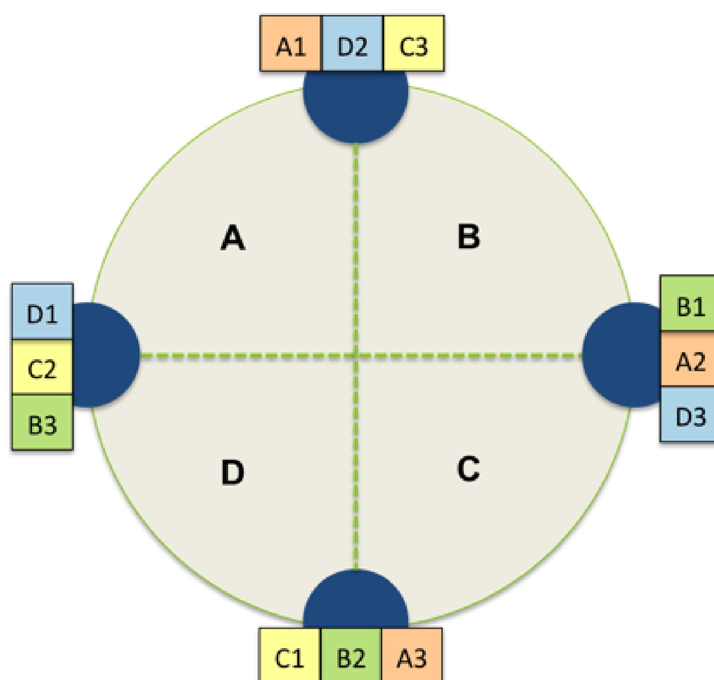
Ο συνολικός αριθμός των αντιγράφων στο cluster ονομάζεται replication factor. Αν η τιμή της παραμέτρου αυτής είναι 1, τότε θα υπάρχει μόνο μία φορά κάθε γραμμή στον κόμβο. Αν είναι 2, τότε θα υπάρχουν 2 αντίγραφα, σε διαφορετικούς κόμβους το καθένα. Επίσης, δεν

υπάρχει η λογική master/slave στην Cassandra. Όλα τα αντίγραφα είναι εξίσου σημαντικά. Και φυσικά ενώ δεν έχει νόημα η τιμή του replication factor να ξεπερνά τον αριθμό των κόμβων του cluster, είναι δυνατό να θέσουμε πρώτα την παράμετρο και μετά να αυξήσουμε τον αριθμό των κόμβων.

Οι μέθοδοι αποθήκευσης των αντιγράφων είναι 2:

SimpleStrategy: Χρησιμοποιείται για clusters που βρίσκονται σε μοναδικό data center. Είναι η προεπιλεγμένη μέθοδος όταν χρησιμοποιούμε το Cassandra CLI, ενώ όταν χρησιμοποιούμε το CQL πρέπει να ορίζουμε απαραίτητα κάποια μέθοδο (δεν υπάρχει προεπιλεγμένη). Η μέθοδος αυτή τοποθετεί το πρώτο αντίγραφο σε κάποιον κόμβο που αποφασίζεται από τον διαχωριστή (partitioner). Τα επόμενα αντίγραφα τοποθετούνται στους επόμενους κόμβους στο δαχτυλίδι δεξιόστροφα, χωρίς να έχει σημασία η φυσική τοποθεσία του data center που ανήκει το καθένα.

Στην εικόνα που ακολουθεί βλέπουμε τρία αντίγραφα τριών σειρών, τοποθετημένα σε τέσσερις κόμβους:



Εικόνα 11. Αναπαραγωγή δεδομένων με τη μέθοδο SimpleStrategy

NetworkStrategy: Αυτή η μέθοδος είναι καλό να χρησιμοποιείται όταν έχουμε (ή σκοπεύουμε να έχουμε στο μέλλον) το cluster μας μοιρασμένο σε πολλαπλά data centers. Η μέθοδος αυτή τοποθετεί το πρώτο αντίγραφο σε κάποιον κόμβο που αποφασίζεται από τον διαχωριστή, όμοια με την SimpleStrategy. Τα υπόλοιπα αντίγραφα τοποθετούνται στον πρώτο κόμβο δεξιόστροφα που ανήκει σε διαφορετικό data center. Αν δεν υπάρχει τέτοιος

κόμβος τότε τοποθετούνται σε κόμβο του ίδιου data center. Αυτό γίνεται γιατί μπορεί οι κόμβοι ενός data center να αποτύχουν ταυτόχρονα λόγω πχ απώλειας ρεύματος ή προβλήματος δικτύου.

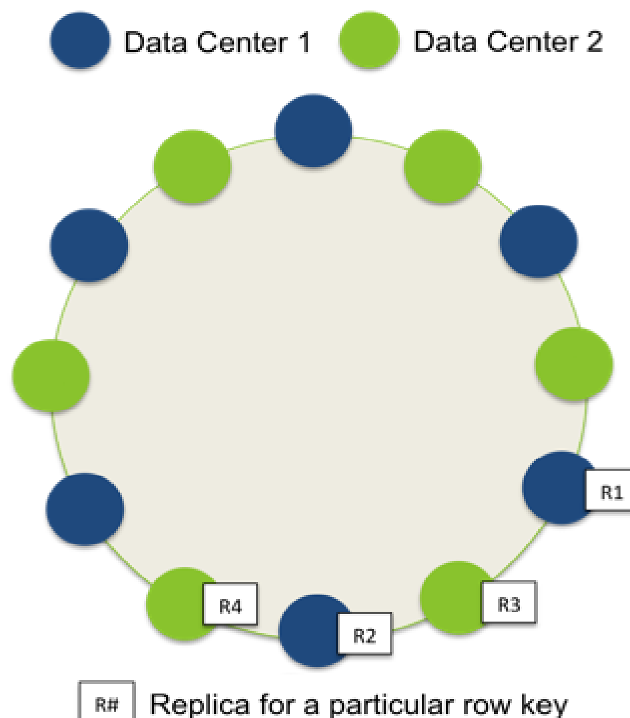
Όταν αποφασίζουμε πόσα αντίγραφα χρειαζόμαστε σε κάθε data center, οι δύο κύριοι παράγοντες που πρέπει να λάβουμε υπ' όψη μας είναι: να μπορεί το σύστημά μας να ικανοποιήσει τα reads σε τοπικό επίπεδο -χωρίς να χρειάζεται να πάμε σε κάποιο άλλο data center με αποτέλεσμα να υπάρχει το αντίστοιχο latency- και να μπορεί να αντιμετωπίσει σενάρια αποτυχίας. Οι δύο πιο διαδεδομένοι τρόποι είναι οι εξής:

Δύο αντίγραφα σε κάθε data center: Αυτή η ρύθμιση ανέχεται την αποτυχία ενός κόμβου ανά ομάδα αναπαραγωγής και επιτρέπει να γίνονται τοπικά reads με consistency level=ONE.

Τρία αντίγραφα σε κάθε data center: Με αυτή τη ρύθμιση είτε ανεχόμαστε την αποτυχία ενός κόμβου ανά ομάδα με consistency leve=LOCAL_QUORUM είτε την αποτυχία πολλών κόμβων ανά data center με consistency level=ONE.

Ακόμη, είναι δυνατή και η ασύμμετρη αναπαραγωγή. Για παράδειγμα είναι δυνατό να έχουμε 3 αντίγραφα ανά data center που να απαντάνε σε real-time αιτήσεις εφαρμογής και ένα αντίγραφο για στατιστικά.

Στην παρακάτω εικόνα φαίνεται πως τοποθετούνται με τη μέθοδο NetworkTopologyStrategy αντίγραφα που εκτείνονται σε δύο data centers, με συνολικό συντελεστή αναπαραγωγής ίσο με 4. Με τη μέθοδο αυτή μπορούμε να ορίσουμε τον αριθμό των αντιγράφων για κάθε data center. **[28]**



Εικόνα 12. Αναπαραγωγή δεδομένων με τη μέθοδο NetworkStrategy

9.8.8 Διαχωρισμός δεδομένων στην Cassandra

Ο διαχωριστής καθορίζει τον τρόπο με τον οποίο τα δεδομένα διανέμονται μεταξύ των κόμβων μέσα σε ένα σύμπλεγμα (cluster) συμπεριλαμβανομένων των αντιγράφων. Ο διαχωριστής αποτελεί βασικά μια λειτουργία κατακερματισμού για τον υπολογισμό του token ενός κλειδιού σειράς. Κάθε σειρά δεδομένων αναγνωρίζεται μοναδικά από ένα κλειδί σειράς και διανέμεται μέσα στο cluster σύμφωνα με την τιμή του token. **[28]**

9.8.8.1 Διανομή δεδομένων στο δαχτυλίδι (Ring)

Στο Cassandra το συνολικό πλήθος των δεδομένων που διαχειρίζεται το cluster αναπαρίσταται σαν ένα δαχτυλίδι (ring). Το ring διαιρείται σε τμήματα ίσα με τον αριθμό των κόμβων, όπου ο κάθε κόμβος είναι υπεύθυνος για ένα ή περισσότερα τμήματα των δεδομένων. Πριν ένας κόμβος μπει στο ring είναι αναγκαίο να του ανατεθεί ένα token. Η τιμή του token καθορίζει την θέση του κόμβου μέσα στο ring καθώς και το εύρος δεδομένων του κόμβου. Οι πίνακες (column family data) διαχωρίζονται μέσα στον κόμβο βάση του κλειδιού της σειράς τους. Για να καθοριστεί ο κόμβος στον οποίο θα εγκατασταθεί το πρώτο αντίγραφο της σειράς, γίνεται αναζήτηση στο ring δεξιόστροφα μέχρι να εντοπιστεί ο κόμβος με τιμή token μεγαλύτερη από αυτήν της σειράς κλειδιού. Κάθε κόμβος είναι υπεύθυνος για την περιοχή του ring μεταξύ του ίδιου (inclusive) και του προκατόχου του (exclusive). Με τους κόμβους ταξινομημένους σύμφωνα με την τιμή του token, ο τελευταίος κόμβος θεωρείται ο προκάτοχος του πρώτου κόμβου και έτσι προκύπτει η αναπαράσταση του δαχτυλιδιού.

Παράδειγμα: έχουμε ένα απλό cluster τεσσάρων κόμβων όπου όλα τα κλειδιά σειράς που διαχειρίζεται το cluster είναι αριθμοί εύρους 0 έως 100. Σε κάθε κόμβο έχει ανατεθεί ένα token το οποίο αντιπροσωπεύει ένα σημείο στο συγκεκριμένο εύρος, Σε αυτό το απλό παράδειγμα οι τιμές του token είναι 0, 25, 50 και 75. Ο πρώτος κόμβος, με την τιμή token ίση με το 0, είναι υπεύθυνος για το εύρος 75 - 0. Ο κόμβος με την χαμηλότερη τιμή token δέχεται επίσης κλειδιά σειράς λιγότερα από το χαμηλότερο token και περισσότερα από το μεγαλύτερο.

9.8.8.2 Τύποι διαχωριστών

Όταν αναπτύσσουμε ένα cluster στο Cassandra πρέπει να αναθέσουμε έναν διαχωριστή και σε κάθε κόμβο ένα αρχικό token ώστε ο κάθε κόμβος να είναι υπεύθυνος για περίπου το ίδιο πλήθος δεδομένων.

Για να υπολογίσουμε τα tokens για κόμβους που ανήκουν σε cluster μονού data center, διαιρούμε το εύρος του συνολικού αριθμού των κόμβων στο cluster. Σε πολλαπλά data

center, υπολογίζουμε τα tokens έτσι ώστε κάθε data center να έχει περίπου την ίδια ποσότητα δεδομένων. Αντίθετα από τις περισσότερες επιλογές παραμετροποίησης στο Cassandra, το είδος του διαχωριστή δεν μπορεί να αλλάξει χωρίς να επαναφορτώσει όλα τα δεδομένα. Έτσι λοιπόν, είναι σημαντικό να επιλέξουμε και να παραμετροποιήσουμε το σωστό είδος διαχωριστή πριν δημιουργήσουμε το cluster. Η ρύθμιση αυτή γίνεται στο cassandra.yaml. Η Cassandra, στην τρέχουσα έκδοση 1.2, διαθέτει τους διαχωριστές RandomPartitioner, ByteOrderPartitioner και Murmur3Partitioner.

10 *Πείραμα σε NoSQL σύστημα*

Η Cassandra δεν προσφέρει συναρτήσεις για υπολογισμό aggregates (avg,sum κλπ) και ενθαρρύνει τέτοιοι υπολογισμοί να γίνονται προγραμματιστικά (από την εφαρμογή δηλαδή που χρησιμοποιείται στο front-end) αλλά είναι σχεδιασμένη ώστε τα select και τα insert queries να είναι ταχύτατα. Δεδομένου ότι το βασικό πρόβλημα μας είναι οι αργές εισαγωγές στον κεντρικό πίνακα της βάσης, ο οποίος όλο και μεγαλώνει, αποφασίσαμε πως η μεταφορά του στην Cassandra θα μπορούσε να συνεισφέρει στη βελτίωση της απόδοσης του συστήματός μας. Στα πλαίσια λοιπόν του πειράματος που κάναμε για να ελέγξουμε κατά πόσο θα μας συνέφερε μια τέτοια αλλαγή, εκτελέσαμε κάποιες μετρήσεις, τις οποίες θα παρουσιάσουμε στη συνέχεια.

10.1 Εγκατάσταση της Cassandra 1.2

Η διαδικασία που ακολουθήσαμε για να εγκαταστήσουμε το DataStax Enterprise 3.1.2, που περιέχει την Cassandra 1.2.6.5 ήταν η εξής:

Καταρχάς κατεβάσαμε το αρχείο .tar στον server με την εντολή

```
# wget http://<username>:<password>@downloads.datastax.com/enterprise/dse.tar.gz
```

και το αποσυμπιέσαμε με την εντολή tar -xzvf dse.tar.gz .

Στη συνέχεια τρέξαμε τις εντολές

```
# export DSE_HOME=<install_location>/dse-3.1.2
# export PATH=$PATH:$DSE_HOME/bin
```

ώστε να μπορούμε να τρέξουμε τα utilities που υπάρχουν στον φάκελο bin (όπως το cqlsh ή το cli που είναι τα command line της Cassandra)απ' οπουδήποτε.

Τέλος δημιουργήσαμε και ορίσαμε τα δικαιώματα για τους καταλόγους που θα περιέχουν τα δεδομένα και τα logs της Cassandra:

```
# mkdir /var/data/cassandra
# mkdir /var/log/cassandra
# chown -R $USER:$GROUP /var/data/cassandra
# chown -R $USER:$GROUP /var/log/cassandra
```

Αργότερα βέβαια αλλάξαμε την τοποθεσία των καταλόγων αυτών μέσα από το DSE_HOME/resources/cassandra/conf/cassandra.yaml σε /mydirectory/cassandra/data και / mydirectory /cassandra/log αντίστοιχα, αφού εκεί υπήρχε αρκετός χώρος.

Πριν ξεκινήσουμε όμως, χρειάστηκαν μερικές ακόμα ρυθμίσεις στο αρχείο cassandra.yaml:

```
cluster_name: 'clustername'
initial_token: 0
seeds: server ip
listen_address: server ip
rpc_address: 0.0.0.0
```

Η Cassandra μετά από τα παραπάνω ξεκινάει με `./dse cassandra -f`.

Αν θέλουμε για κάποιο λόγο να τη σταματήσουμε, τότε γράφουμε

```
ps auwx | grep cassandra
```

βρίσκουμε το id του process και με `sudo kill <pid>` το σταματάμε. Στο command line της Cassandra μπαίνουμε με sh Cassandra-cli ή με sh cqlsh για τη γλώσσα CQL, που έχει μεγαλύτερη αντιστοιχία με την SQL. **[28]**

10.2 Εισαγωγή και ανάκτηση δεδομένων από αρχείο

Στην ενότητα αυτή θα περιγράψουμε τη διαδικασία του restore της βάσης από αρχείο σε PostgreSQL και Cassandra, κάνοντας παράλληλα σύγκριση αντίστοιχων μεγεθών στα δύο συστήματα. Σημειώνουμε σε αυτό το σημείο πως λόγω περιορισμένου χώρου στο δίσκο του server του Ινστιτούτου, δε μπορούσαμε να αντιγράψουμε ολόκληρο τον πίνακα measurement_events στην Cassandra και έτσι δημιουργήσαμε ένα αντίγραφο ενός δισεκατομμυρίου εγγραφών.

10.2.1 Μέθοδος COPY

Η εντολή COPY είναι ο βέλτιστος τρόπος για εισαγωγή δεδομένων στην PostgreSQL. Μπορεί να χρησιμοποιηθεί για οποιοδήποτε μέγεθος δεδομένων, παρέχοντας πολύ καλύτερα

αποτελέσματα από μεθόδους όπως τρέξιμο .sql αρχείων με την εντολή INSERT INTO .. VALUES .. ή εκτέλεση της λειτουργίας pg_restore της PostgreSQL (φορτώνει back up της βάσης το οποίο έχει παρθεί με τη λειτουργία pg_dump).

Η ίδια ακριβώς εντολή υπάρχει και στη γλώσσα CQL της Cassandra(που όπως έχουμε πει μοιάζει πολύ με την SQL). Στην Cassandra όμως, η εισαγωγή δεδομένων με τη χρήση της εντολής COPY προτείνεται για μικρό μέγεθος δεδομένων μόνο, καθώς είναι πολύ αργή.

Η διαδικασία αποτελείται από δύο βήματα:

Καταρχάς δημιουργούμε ένα αρχείο .csv με συγκεκριμένο αριθμό εγγραφών, έστω X, με την εντολή

```
COPY
  (SELECT *
   FROM measurement_events LIMIT X) TO
'/(path_to)/measurement_events.csv' WITH DELIMITER ',';
```

Πίνακας 30. Εντολή για τη μεταφορά του πίνακα από τη βάση σε αρχείο .csv

Και στη συνέχεια φορτώνουμε το αρχείο αυτό στη βάση μας με την εντολή

```
COPY table_name
FROM '/(path_to)/measurement_events.csv';
```

Πίνακας 31. Εντολή για τη μεταφορά του πίνακα από το .csv στη βάση

Η εντολή είναι ίδια για PostgreSQL και Cassandra.

10.2.2 Μέθοδος bulk loading

Όπως αναφέραμε και όπως φαίνεται και στο αποτέλεσμα της παραπάνω ενότητας, η εισαγωγή δεδομένων στην Cassandra με την εντολή COPY είναι χρονοβόρα διαδικασία (δοκιμάσαμε COPY για ένα δισεκατομμύριο γραμμές αλλά δεν ολοκληρώθηκε ακόμα και μετά από 24 ώρες, επομένως το διακόψαμε). Για το λόγο αυτό διερευνήσαμε τις εναλλακτικές μεθόδους που υπάρχουν για εισαγωγή μεγάλου όγκου δεδομένων στην Cassandra και βρήκαμε πως υπάρχουν διάφορα εργαλεία για bulk loading:

- Το **Sqoop** που δεν είναι αποκλειστικά για cassandra αλλά χρησιμοποιείται για bulk loading από σχεσιακές βάσεις σε διάφορες NoSQL βάσεις και το οποίο είναι ενσωματωμένο στο πακέτο DataStax Enterprise.
- Το **BinaryMemtable** το οποίο ήταν και το μόνο interface για bulk loading για τις παλαιότερες εκδόσεις της cassandra (πριν την 1.0).
- Το **sstableloader** που είναι ένα εργαλείο των τελευταίων εκδόσεων της cassandra και το οποίο, δεδομένων των αρχείων sstables, τα φορτώνει σε ένα σύμπλεγμα cassandra.

Εμείς επιλέξαμε το τελευταίο. Κατ'αρχάς οι χρόνοι του είναι αρκετά καλύτεροι. Επίσης, η διαδικασία του bulk loading με το sstableloader δεν προσθέτει μεγάλο φόρτο στο μηχάνημα από το οποίο φορτώνονται τα δεδομένα αλλά ούτε και στο συνολικό σύμπλεγμα. Με τις υπόλοιπες τεχνικές bulk loading χτυπάμε το σύμπλεγμα διαρκώς, ενώ με το sstableloader κατά τη διάρκεια δημιουργίας των SSTables δεν προστίθεται επιπλέον φόρτος. Για πολύ μεγάλο μέγεθος SSTables, η ώρα που χρειάζεται για τη δημιουργία τους είναι διπλάσια ή και τριπλάσια της ώρας που χρειάζεται για να φορτωθούν. Άρα αν η δημιουργία X GB SSTables έπαιρνε περίπου τρεις ώρες, το φόρτωμά τους στη βάση θα έπαιρνε περίπου μία ώρα. Με τις άλλες τεχνικές δηλαδή, θα είχαμε περίπου επί τέσσερις ώρες φορτωμένο το σύμπλεγμα ενώ με το sstableloader μόνο μία.

Η διαδικασία του bulk loading με το εργαλείο sstableloader στην Cassandra έχει ως εξής **[29]**:

Καταρχάς πρέπει να έχουμε στη διάθεσή μας ένα αρχείο .csv (comma seperated values) με τα δεδομένα που θέλουμε να φορτώσουμε στην Cassandra. Αυτό το αρχείο δημιουργείται εύκολα από την postgres με μία εντολή COPY.

```
COPY measurement_events TO /(path_to)/measurement_events.csv WITH
DELIMITER ',';
```

Πίνακας 32. Εντολή δημιουργίας αρχείου .csv

Μέσω του command line της Cassandra (cli ή cql , τα οποία έχουμε αναλύσει παραπάνω) δημιουργούμε ένα keyspace με την ακόλουθη εντολή.

```
CREATE KEYSPACE mydb WITH replication = {'class': 'SimpleStrategy',
'replication_factor' : 1} AND durable_writes = false;
```

Πίνακας 33. Εντολή δημιουργίας keyspace στην Cassandra

Στη συνέχεια, πρέπει να δημιουργήσουμε στην cassandra την οικογένεια στηλών (column family) την οποία θέλουμε να γεμίσουμε. Μπορούμε να τη δημιουργήσουμε είτε μέσω του cli είτε μέσω του cql (που όπως έχουμε αναφέρει και παραπάνω είναι τα command line της Cassandra).

```
create column family Measurement_events
     with key_validation_class=LexicalUUIDType
     and comparator=AsciiType
     and column_metadata=[
         { column_name: 'measurement_location_id', validation_class:
AsciiType },
         { column_name: 'measurement_source_id', validation_class:
AsciiType },
         { column_name: 'measurement_time', validation_class:
```

```
AsciiType },
         { column_name: 'event_reception_time', validation_class:
AsciiType },
         { column_name: 'measurement_value', validation_class:
AsciiType },
         { column_name: 'quality', validation_class: AsciiType },
         { column_name: 'source_id', validation_class: AsciiType },
         { column_name: 'transducer_id', validation_class: AsciiType
}];
```

Πίνακας 34. Εντολή δημιουργίας column family στην Cassandra όμοιου με τον πίνακα measurement_events της PostgreSQL

Αφού κάνουμε αυτά τα αρχικά βήματα, πρέπει να δημιουργήσουμε από το .csv αρχείο τα SSTables. Για να το καταφέρουμε αυτό, φτιάξαμε ένα πρόγραμμα σε java, το DataImport.java, το οποίο παίρνει ως όρισμα το .csv αρχείο και με τη χρήση της κλάσης SSTableSimpleUnsortedWriter και κάποιων βιβλιοθηκών της cassandra το μετατρέπει σε SSTable. Το πρόγραμμα αυτό το τρέχουμε με τη βοήθεια ενός script που φτιάξαμε, το run.sh. Μέσα στο run.sh γίνονται οι εξής ενέργειες: αρχικά προσθέτουμε στο classpath μερικά .jar αρχεία που χρειάζονται για να γίνει compile το DataImport.java, στη συνέχεια το κάνουμε compile, και αφού προσθέσουμε στο classpath και το αρχείο DataImport.class που δημιουργείται μετά το compiling, τρέχουμε το DataImport με όρισμα το αρχείο .csv που έχουμε δώσει ως όρισμα το run.sh. Το script αυτό το τρέχουμε με την εντολή:

```
# ./run.sh measurement_events.csv
```

αφού πρώτα το έχουμε κάνει εκτελέσιμο με την εντολή:

```
# chmod -x run.sh
```

Παρακάτω φαίνεται ο κώδικας των αρχείων αυτών:

```
import java.nio.ByteBuffer;
import java.io.*;
import java.util.UUID;
import org.apache.cassandra.db.marshal.*;
import org.apache.cassandra.io.sstable.SSTableSimpleUnsortedWriter;
import org.apache.cassandra.dht.Murmur3Partitioner;
import org.apache.cassandra.dht.RandomPartitioner;
import static org.apache.cassandra.utils.ByteBufferUtil.bytes;
import static org.apache.cassandra.utils.UUIDGen.decompose;

public class DataImport
{
    static String filename;
```

```
    public static void main(String[] args) throws IOException
    {
        if (args.length == 0)
        {
            System.out.println("Expecting <csv_file> as argument");
            System.exit(1);
        }
        long start = System.currentTimeMillis();
        filename = args[0];
        BufferedReader     reader     =     new     BufferedReader(new
FileReader(filename));
        Murmur3Partitioner part = new Murmur3Partitioner();

        String keyspace = "mydb";
        File directory = new File(keyspace);
        if (!directory.exists())
            directory.mkdir();

        SSTableSimpleUnsortedWriter      usersWriter      =      new
SSTableSimpleUnsortedWriter(
                directory,
                part,
                keyspace,
                "Measurement_events",
                AsciiType.instance,
                null,
                64);
        String line;
        int lineNumber = 1;
        CsvEntry entry = new CsvEntry();
        // There is no reason not to use the same timestamp for every
column in that example.
        long timestamp = System.currentTimeMillis() * 1000;
        while ((line = reader.readLine()) != null)
        {
            if (entry.parse(line, lineNumber))
            {
                ByteBuffer                   uuid                   =
ByteBuffer.wrap(decompose(UUID.randomUUID()));
```

```
                usersWriter.newRow(uuid);

usersWriter.addColumn(bytes("measurement_location_id"),
bytes(entry.measurement_location_id), timestamp);
                usersWriter.addColumn(bytes("measurement_source_id"),
bytes(entry.measurement_source_id), timestamp);
                usersWriter.addColumn(bytes("measurement_time"),
bytes(entry.measurement_time), timestamp);
                usersWriter.addColumn(bytes("event_reception_time"),
bytes(entry.event_reception_time), timestamp);
                usersWriter.addColumn(bytes("measurement_value"),
bytes(entry.measurement_value), timestamp);
                usersWriter.addColumn(bytes("quality"),
bytes(entry.quality), timestamp);
                usersWriter.addColumn(bytes("source_id"),
bytes(entry.source_id), timestamp);
                usersWriter.addColumn(bytes("transducer_id"),
bytes(entry.transducer_id), timestamp);
            }
            lineNumber++;
        }

        long end = System.currentTimeMillis();

        System.out.println("Successfully parsed " + lineNumber + "
lines.");
        System.out.println("Execution time was "+(end-start)+" ms.");

        usersWriter.close();
        System.exit(0);
    }
      static class CsvEntry
    {
        String measurement_location_id;
        String measurement_source_id;
        String measurement_time;
        String event_reception_time;
        String measurement_value;
        String quality;
        String source_id;
```

```
        String transducer_id;

        boolean parse(String line, int lineNumber)
        {
            String[] columns = line.split(",");
            if (columns.length != 8)
            {
                System.out.println(String.format("Invalid  input  '%s' 
at line %d of %s", line, lineNumber, filename));
                return false;
            }
                try
            {
                measurement_location_id = columns[0].trim();
                measurement_source_id = columns[1].trim();
                measurement_time = columns[2].trim();
                event_reception_time = columns[3].trim();
                measurement_value = columns[4].trim();
                quality = columns[5].trim();
                source_id = columns[6].trim();
                transducer_id = columns[7].trim();
                return true;
            }
            catch (NumberFormatException e)
            {
                System.out.println(String.format("Invalid  number  in 
input '%s' at line %d of %s", line, lineNumber, filename));
                        return false;
            }
        }
    }
}
```

Πίνακας 35. DataImport.java

Στη συνέχεια αναλύουμε τα κυριότερα σημεία του προγράμματος DataImport.java:

Αμέσως αφού ορίσουμε τη συνάρτηση main, ελέγχουμε αν όντως ο χρήστης τρέχει το πρόγραμμα με το .csv αρχείο ως παράμετρο.

Στη συνέχεια ορίζουμε τις μεταβλητές που χρησιμοποιούνται στην κλάση meWriter που είναι τύπου SSTableSimpleUnsortedWriter και ορίζεται αμέσως μετά.

Οι μεταβλητές αυτές αφορούν τα εξής:

directory: ο φάκελος που θα τοποθετηθούν τα SSTables. Είναι αντικείμενο τύπου File.

part: ο τύπος διαχωριστή που χρησιμοποιούμε στο cluster μας. Είναι τύπου Murmur3Partitioner (το έχουμε λόγω του import org.apache.cassandra.dht.Murmur3Partitioner που έχουμε κάνει παραπάνω).

keyspace: το όνομα του keyspace της cassandra στο οποίο θέλουμε να μπουν μετά τα SSTables. Είναι τύπου String.

colfamily: το όνομα της οικογένειας στηλών την οποία θέλουμε να γεμίσουμε. Είναι και αυτό τύπου String.

Οι δύο επόμενες παράμετροι αφορούν τον τύπο πρωτευόντων και δευτερευόντων index.

buff: είναι το μέγεθος του buffer που θέλουμε να χρησιμοποιείται κατά τη δημιουργία των SSTables. Κάθε αρχείο SSTable που δημιουργείται θα έχει όσο μέγεθος ορίσουμε με την παράμετρο αυτή. Είναι τύπου Int.

Μετά τον ορισμό της κλάσης meWriter, διαβάζουμε με ένα while loop το αρχείο .csv και δημιουργούμε για κάθε εγγραφή του μια νέα σειρά στα SSTables, προσθέτοντας μία μία τις στήλες. Η εισαγωγή των δεδομένων γίνεται με τη βοήθεια της κλάσης entry που είναι τύπου Csventry και ορίζεται από κάτω. Στην κλάση αυτή χωρίζουμε στην ουσία κάθε γραμμή του .csv σε στήλες (σε κάθε κόμμα καταλαβαίνει ότι αλλάζει η στήλη) και τις προσθέτουμε μία μία στο SSTable.

Έχουμε προσθέσει κι εδώ κάποια if then else ώστε να είμαστε σίγουροι ότι προσθέτονται σωστά οι στήλες.

```
JAVA=`which java`
CASSANDRA_CONF=/home/pantoniadou/apache-cassandra-1.2.8/conf
CASSANDRA_HOME=/home/pantoniadou/apache-cassandra-1.2.8
if [ -n "$CLASSPATH" ]; then
    CLASSPATH=$CLASSPATH:$CASSANDRA_CONF
else
    CLASSPATH=$CASSANDRA_CONF
fi
for jar in /home/pantoniadou/apache-cassandra-1.2.8/lib/*.jar; do
    CLASSPATH=$CLASSPATH:$jar
done
javac -cp $CLASSPATH /home/pantoniadou/DataImport2.java
CLASSPATH=$CLASSPATH:/home/pantoniadou
$JAVA   -ea   -cp   $CLASSPATH   -Xmx8192M   -Dlog4j.configuration=log4j-
tools.properties DataImport2 "$@"
```

Πίνακας 36. run.sh

Τελευταίο βήμα αποτελεί το φόρτωμα των SSTables που δημιουργήσαμε στην cassandra. Αυτό γίνεται με το εργαλείο sstableloader το οποίο δέχεται ως ορίσματα τα SSTables και τον host απ'όπου τα παίρνει και τα φορτώνει στο cluster μας. Για να γίνει αυτό, πρέπει να έχουμε τα SSTables που έχουμε δημιουργήσει προηγουμένως μέσα σε έναν φάκελο με όνομα αυτό της οικογένειας στηλών (measurement_events στην περίπτωσή μας), ο οποίος βρίσκεται σε έναν φάκελο με όνομα αυτό του keyspace (mydb στην περίπτωσή μας).

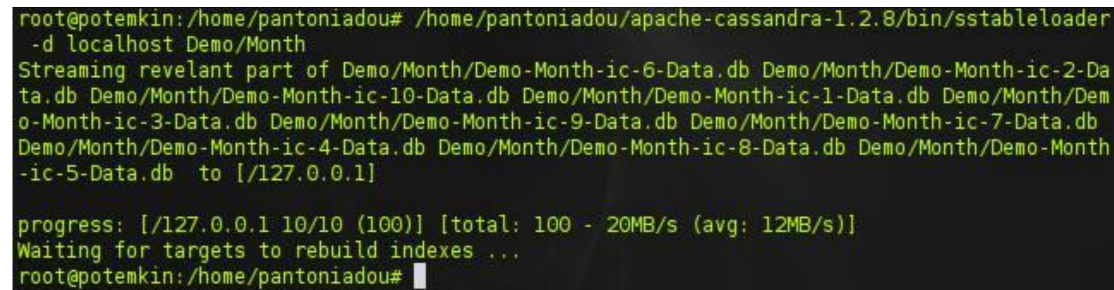
Παρακάτω φαίνεται η εντολή που τρέξαμε:

```
# /path_to_cassandra_installation)/bin/sstableloader -d localhost
(path_to_keyspace_file)/mydb/measurement_events
```

10.2.3 *Μετρήσεις*

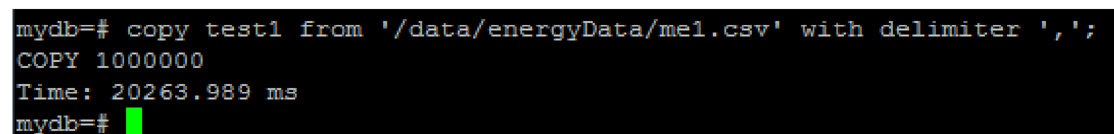
Σε αυτή την ενότητα θα παραθέσουμε τους χρόνους εκτέλεσης των παραπάνω μεθόδων για εισαγωγή δεδομενών.

Αρχικά, εισάγαμε ένα εκατομμύριο εγγραφές σε Cassandra χρησιμοποιώντας την μέθοδο του copy και μετρήσαμε χρόνο 4 λεπτά και 12.878 δευτερόλεπτα. Στη συνέχεια, κάναμε το ίδιο με τη μέθοδο του bulk loading, η οποία εκτελέστηκε σε 23 δευτερόλεπτα. Στην παρακάτω εικόνα φαίνεται η πρόοδος της διαδικασίας.



Εικόνα 13. Μεταφορά από αρχείο .csv στη βάση με bulk loading για πίνακα με ένα εκατομμύριο εγγραφές

Για τον ίδιο αριθμό γραμμών τρέξαμε την εντολή εισαγωγής copy στην PostgreSQL η οποία ολοκληρώθηκε σε περίπου 20 δευτερόλεπτα όπως φαίνεται και στην εικόνα που ακολουθεί.



Εικόνα 14. Μεταφορά από αρχείο .csv στη βάση με COPY για πίνακα με ένα εκατομμύριο εγγραφές

Το αποτέλεσμα αυτό ήταν αναμενόμενο λόγω του μικρού μεγέθους των δεδομένων και ο κύριος σκοπός του πειράματος με το συγκεκριμένο μικρό δείγμα ήταν για να τεκμηριωθεί η ορθότητα των μεθόδων.

Για να ανάγουμε τώρα τα αποτελέσματα στα δεδομένα του προβλήματός μας επαναλάβαμε την ίδια διαδικασία με ένα δισεκατομμύριο εγγραφές. Ακολουθούν αναλυτικά τα αποτελέσματα.

5 λεπτά και 15 δευτερόλεπτα μέχρι να γίνει το streaming relevant part of <files with sstables> to [/127.0.0.1] χρόνος μηδαμινός στο προηγούμενο δείγμα.

364 λεπτά και 30 δευτερόλεπτα μέχρι να «φορτωθούν» όλα τα δεδομένα στον πίνακα.

Η συνολική διάρκεια για την ολοκλήρωση του bulk loading ήταν περίπου 6 ώρες.

Ακολουθεί εικόνα της προόδου της διαδικασίας.

```
6-Data.db Demo/Measurement_events/Demo-Measurement_events-ic-7383-Data.db Demo/M
easurement_events/Demo-Measurement_events-ic-2059-Data.db Demo/Measurement_event
s/Demo-Measurement_events-ic-1163-Data.db Demo/Measurement_events/Demo-Measureme
nt_events-ic-6046-Data.db Demo/Measurement_events/Demo-Measurement_events-ic-333
2-Data.db Demo/Measurement_events/Demo-Measurement_events-ic-6000-Data.db Demo/M
easurement_events/Demo-Measurement_events-ic-807-Data.db Demo/Measurement_events
/Demo-Measurement_events-ic-2806-Data.db Demo/Measurement_events/Demo-Measuremen
t_events-ic-7041-Data.db Demo/Measurement_events/Demo-Measurement_events-ic-8569
-Data.db Demo/Measurement_events/Demo-Measurement_events-ic-2692-Data.db Demo/Me
asurement_events/Demo-Measurement_events-ic-6115-Data.db Demo/Measurement_events
/Demo-Measurement_events-ic-1848-Data.db Demo/Measurement_events/Demo-Measuremen
t_events-ic-5701-Data.db Demo/Measurement_events/Demo-Measurement_events-ic-4143
-Data.db Demo/Measurement_events/Demo-Measurement_events-ic-5895-Data.db Demo/Me
asurement_events/Demo-Measurement_events-ic-5366-Data.db Demo/Measurement_events
/Demo-Measurement_events-ic-912-Data.db Demo/Measurement_events/Demo-Measurement
_events-ic-4075-Data.db Demo/Measurement_events/Demo-Measurement_events-ic-7368-
Data.db Demo/Measurement_events/Demo-Measurement_events-ic-7280-Data.db Demo/Mea
surement_events/Demo-Measurement_events-ic-1995-Data.db Demo/Measurement_events/
Demo-Measurement_events-ic-6211-Data.db  to [/127.0.0.1]

progress: [/127.0.0.1 8725/8725 (100)] [total: 100 - 1MB/s (avg: 10MB/s)]
Waiting for targets to rebuild indexes ...
root@potemkin:/data/energyData#
```

Εικόνα 15. Μεταφορά από αρχείο .csv στη βάση με bulk loading για πίνακα με ένα δισεκατομμύριο εγγραφές

Η αντίστοιχη εντολή COPY στην Cassandra δεν ολοκληρώθηκε μετά το πέρας 24 ωρών και γι'αυτό το λόγω διακόψαμε την εκτέλεσή της. Το αποτέλεσμα αυτό ήταν αναμενόμενο αφού από το documentation της Cassandra η μέθοδος αυτή αντενδείκνυται για πολλά δεδομένα.

Στην PostgreSQL η μεταφορά ενός δισεκατομμυρίου εγγραφών με την μέθοδο COPY ολοκληρώθηκε σε περίπου 10.5 ώρες όπως φαίνεται στην παρακάτω εικόνα.

```
mydb=# copy test  from '/data/energyData/me.csv'  with delimiter ',';
COPY 1000000000
Time: 37874751.970 ms
mydb=#
```

Εικόνα 16. Μεταφορά από αρχείο .csv στη βάση με COPY για πίνακα με ένα δισεκατομμύριο εγγραφές

Στον παρακάτω πίνακα φαίνονται συγκεντρωτικά όλα τα παραπάνω αποτελέσματα.

	1 εκατομμύριο	**1 δισεκατομμύριο**
PostgreSQL COPY	0:00:20’’	10:31:14’’
Cassandra COPY	0:04:13’’	δεν ολοκληρώθηκε
Cassandra Bulkloading	0:00:23’’	6:04:30’’

Πίνακας 37. Αποτελέσματα

Από το παραπάνω πείραμα παρατηρούμε ότι η μέθοδος του bulk loading είχε 45% μειωμένο χρόνο εκτέλεσης στην Cassandra από ότι είχε η ισοδύναμη διαδικασία στην PostgreSQL. Άρα η Cassandra έχει εμφανές πλεονέκτημα στην ανάκτηση δεδομένων σε σχέση με την PostgreSQL.

10.2.4 Λεπτομέρειες

Στο σημείο αυτό πρέπει να σημειώσουμε πως το πρώτο βήμα της διαδικασίας του bulk loading που περιγράψαμε (η δημιουργία των SSTables από το .csv αρχείο) προφανώς χρειάζεται μόνο κατά την πρώτη μεταφορά μιας βάσης από κάποιο SQL σύστημα σε Cassandra αφού κατά τη μεταφορά δεδομένων από Cassandra σε Cassandra τα SSTables υπάρχουν ήδη. Ο χρόνος που κάνουν να δημιουργηθούν τα SSTables λοιπόν δεν υπάρχει λόγος να συμπεριληφθεί στις μετρήσεις. Επειδή παρόλαυτα στο DataImport.java έχουμε βάλει να μετριέται ο χρόνος αυτός, αναφέρουμε πως για το 1 δισεκατομμύριο δεδομένων ήταν 31,588 δευτερόλεπτα (περίπου 8.5 ώρες).

10.3Εισαγωγή και Επιλογή δεδομένων

Στην ενότητα αυτή θα μετρήσουμε το χρόνο εισαγωγής δεδομένων σε πίνακα καθώς και το χρόνο εμφάνισης δεδομένων από πίνακα σε PostgreSQL και Cassandra.

Στην PostgreSQL όπως έχει προαναφερθεί η εισαγωγή των δεδομένων έγινε με την εντολή

```
INSERT INTO table_X
SELECT *
FROM table_Y
LIMIT N;
```

όπου N ο αριθμός των εισαγώμενων γραμμών.

Στην Cassandra χρησιμοποιήσαμε το εργαλείο cassandra-stress που υπάρχει στο πακέτο tools της Cassandra και προσομοιώνει την απόδοση του συστήματος κάτω από ορισμένες συνθήκες. Στην περίπτωσή μας, η εντολή που εκτελέσαμε είχε την παρακάτω μορφή,

`./cassandra-stress -c 8 -S 16 -d localhost -o INSERT -n N` , όπου c είναι ο αριθμός των στηλών, S είναι ο αριθμός bytes κάθε στήλης, d είναι ο υπολογιστής στον οποίο

θέλουμε να τρέξουμε την προσομοίωση, o είναι η λειτουργία και n ο αριθμός των σειρών πάνω στις θέλουμε να εκτελέσουμε το τεστ. Συνήθως το εργαλείο αυτό χρησιμοποιείται για να δούμε πως θα συμπεριφερόταν η Cassandra στο σύμπλεγμά μας αν ενώ τρέχει κάποια εντολή πέσει ένας κόμβος ή αν έχουμε τεράστιο φόρτο εργασιών και γενικότερα για τέτοιες ακραίες περιπτώσεις. **[28]**

```
root@/apache-cassandra-1.2.8/tools/bin# ./cassandra-stress -c 8 -S 16 -d localhost -o INSERT -n 1000000

total,interval_op_rate,interval_key_rate,latency/95th/99th,elapsed_time
64940,6494,6494,0.5,1.4,486.9,10
150747,8580,8580,0.4,1.1,486.9,20
237875,8712,8712,0.4,1.0,486.9,30
330400,9252,9252,0.4,1.0,486.9,40
418191,8779,8779,0.4,1.0,429.7,50
505964,8777,8777,0.4,0.9,429.7,60
599971,9400,9400,0.4,1.0,400.1,70
692091,9212,9212,0.4,1.0,400.1,80
775599,8350,8350,0.4,1.0,400.9,91
863387,8778,8778,0.4,1.0,400.1,101
947505,8411,8411,0.4,0.9,400.1,111
1000000,5249,5249,0.4,1.0,400.1,117
END
```

Εικόνα 17. Screenshot από την προσομοίωση των inserts στην Cassandra

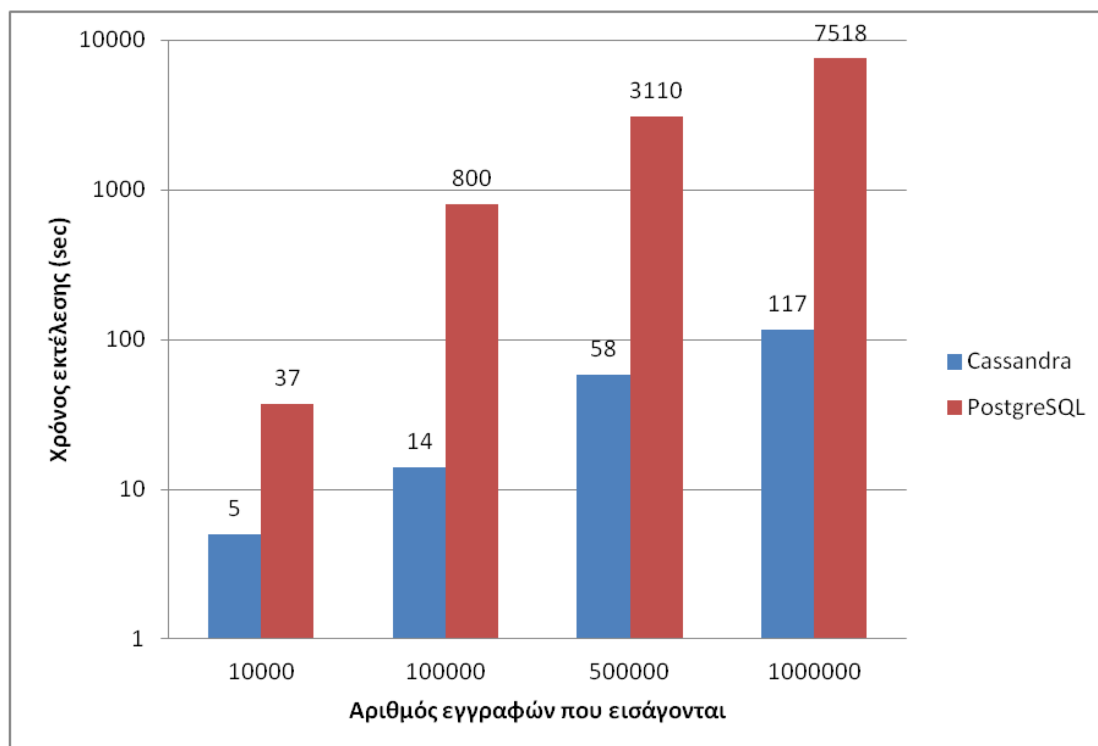
Παρακάτω παρουσιάζονται τα αποτελέσματα που προέκυψαν για διάφορες τιμές της μεταβλητής N.

rows inserted	PostgreSQL	Cassandra
10000	37 sec	5 sec
100000	800 sec	14 sec
500000	3,110 sec	58 sec
1000000	7,518 sec	117 sec

Πίνακας 38. Χρόνοι εκτέλεσης insert σε PostgreSQL και Cassandra

Από τις παραπάνω μετρήσεις βλέπουμε ότι το σύστημα σε Cassandra ανταποκρίνεται πολύ πιο γρήγορα παρόλο το γεγονός ότι έχουμε μόνο ένα κόμβο. Αν η εφαρμογή έτρεχε σε περισσότερους από έναν κόμβους, αν αξιοποιούσαμε δηλαδή την αρχιτεκτονική της Cassandra, θα είχαμε ακόμα μεγαλύτερη βελτίωση στα αποτελέσματα.

Παρακάτω παρουσιάζονται και γραφικά τα αποτελέσματα του Πίνακα 38.



Γράφημα 6. Σύγκριση insert σε PostgreSQL και Cassandra

Στην συνέχεια πραγματοποιήσαμε αντίστοιχα πειράματα για την επιλογή δεδομένων από πίνακα. Όπως στην PostgreSQL, έτσι και στην Cassandra είναι απαραίτητη η ύπαρξη index στις στήλες που χρησιμοποιούνται στις συνθήκες του where-clause. Αντίθετα όμως με την PostgreSQL, η δημιουργία index στην Cassandra είναι πάρα πολύ γρήγορη ανεξάρτητα από το μέγεθος του πίνακα. Αυτό οφείλεται στη αρχιτεκτονική της Cassandra και στον τρόπο με τον οποίο αποθηκεύει τα δεδομένα. Για παράδειγμα, το χτίσιμο του index για πίνακα ενός δισεκατομμυρίου στην PostgreSQL πήρε μερικές ώρες ενώ στην Cassandra η ίδια διαδικασία ολοκληρώθηκε σε λίγα δευτερόλεπτα.

Τόσο στην PostgreSQL όσο και στην Cassandra η επιλογή ενός τυχαίου δείγματος δεδομένων γίνεται με την εντολή

```
SELECT *
FROM table_X
LIMIT N;
```

όπου N ο αριθμός των επιλεγμένων γραμμών.

Για την μέτρηση των χρόνων που κάνει η επιλογή δεδομένων σε Cassandra πριν ξεκινήσουμε το τρέξιμο, θέσαμε στην CQL: tracing on; ώστε να εντοπίζεται η διαδρομή που κάνει ένα ερώτημα μέχρι να παράγει αποτελέσματα με ακριβείς χρόνους.

Ακολουθεί ένα ενδεικτικό screenshot:

```
Tracing session: 05192ef0-4309-11e3-858a-9553630e5d0b

 activity                                                                                 | timestamp    | source    | source_elapsed
------------------------------------------------------------------------------------------+--------------+-----------+----------------
                                                                        execute_cql3_query | 17:19:37,055 | 127.0.0.1 |              0
                                  Parsing select * from "Measurement_events" limit 10000; | 17:19:37,055 | 127.0.0.1 |             52
                                                                       Peparing statement | 17:19:37,056 | 127.0.0.1 |            160
                                                              Determining replicas to query | 17:19:37,056 | 127.0.0.1 |            269
 Executing seq scan across 3 sstables for [min(-9223372036854775808), min(-9223372036854775808)] | 17:19:37,056 | 127.0.0.1 |            598
                                                     Scanned 10000 rows and matched 10000 | 17:19:37,421 | 127.0.0.1 |         365506
                                                                         Request complete | 17:19:37,470 | 127.0.0.1 |         415800
```

Εικόνα 18. Screenshot από την εκτέλεση της εντολής select σε Cassandra

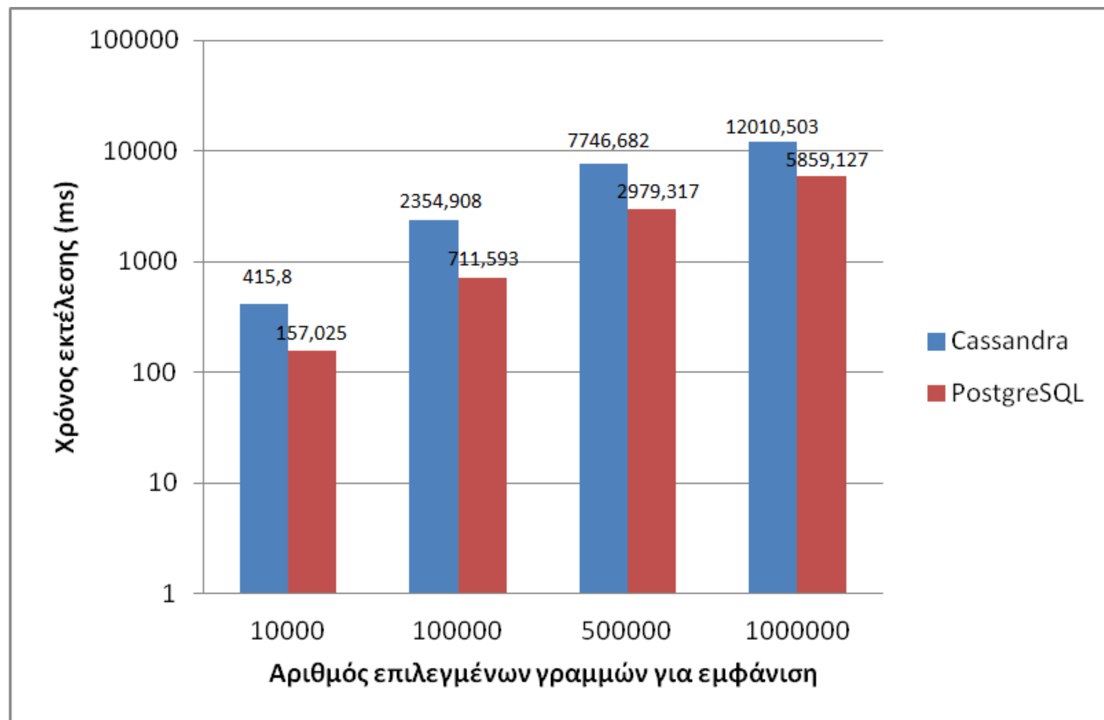
Τα αποτελέσματα για διάφορες τιμές του Ν φαίνονται στον παρακάτω πίνακα.

rows selected	PostgreSQL	Cassandra
10000	157.025 ms	415.8 ms
100000	711.593 ms	2,354.908 ms
500000	2,979.317 ms	7,746.682 ms
1000000	5,859.127 ms	12,010.503 ms

Πίνακας 39. Χρόνοι εκτέλεσης select σε PostgreSQL και Cassandra

Παρατηρούμε ότι η PostgreSQL εμφανίζει πιο γρήγορα τις ζητούμενες γραμμές δεδομένων, όμως πρέπει να λάβουμε υπόψη μας όπως είπαμε και παραπάνω ότι η Cassandra τρέχει σε ένα μόνο κόμβο. Η απόδοσή της ανεβαίνει εκθετικά όσο προστίθενται νέοι κόμβοι στο σύστημα.

Παρακάτω παρουσιάζονται και γραφικά τα αποτελέσματα του Πίνακα 39.



Γράφημα 7. Σύγκριση select σε PostgreSQL και Cassandra

10.4Πρόταση συνδυαστικής λύσης με σχεσιασκή και μη σχεσιακή βάση

Σε προηγούμενο κεφάλαιο, μελετήσαμε τις σχεσιακές βάσεις και προτείναμε μια σειρά από βελτιστοποιήσεις πάνω στο υπάρχον σύστημα. Δεδομένου ότι τα δεδομένα που αποθηκεύονται στη βάση θα συνεχίσουν να αυξάνονται με ταχείς ρυθμούς, σε μερικά χρόνια δε θα υπάρχει άλλη επιλογή από την υιοθέτηση κάποιας μη-σχεσιακής φιλοσοφίας. Σε αυτό το σημείο λοιπόν θα προτείνουμε και μία λύση η οποία συνδυάζει την PostgreSQL με την Cassandra. Το γεγονός ότι η Cassandra δεν μας δίνει τη δυνατότητα του υπολογισμού aggregates μας οδήγησε στην απόφαση να συνδυάσουμε τα δύο συστήματα για να έχουμε ισοδύναμο αποτέλεσμα.

Όπως είδαμε στην προηγούμενη ενότητα, η Cassandra μας εξασφαλίζει ταχύτατες εισαγωγές δεδομένων ανεξάρτητα από το μέγεθος του πίνακα προορισμού. Για παράδειγμα, στο πείραμα που πραγματοποιήσαμε κάνοντας ένα εκατομμύριο inserts σε πίνακα ενός δισεκατομμυρίου τα αποτελέσματα της Cassandra σε σχέση με το αρχικό σύστημα ήταν μειωμένα και 98%. Λόγω αδυναμίας υπολογισμού των ζητούμενων aggregates διατηρούμε τους in-memory ενδιάμεσους πινακες στην PostgreSQL, επομένως απαιτείται αντίστοιχος χρόνος με αυτόν της σχεσιακής λύσης για την ενημέρωσή τους. Παρόλα αυτά η συγκεκριμένη εκδοχή διατηρεί το πλεονέκτημα, διότι σε αντίθεση με την προηγούμενη λύση ο χρήστης έχει πρόσβαση στα δεδομένα του κεντρικού πίνακα αμέσως μετά την εισαγωγή τους, πράγμα που δεν ήταν δυνατόν πριν λόγω της απουσίας index από το τελευταίο partition όπως εξηγήσαμε και παραπάνω.

Μία ακόμα θεαματική βελτίωση που παρατηρήσαμε μεταφέροντας τη βάση μας στην Cassandra ήταν η ακαριαία σχεδόν δημιουργία index σε πίνακα μεγέθους ενός δισεκατομμυρίου. Η αντίστοιχη διαδικασία σε PostgreSQL ολοκληρωνόταν μετά από κάποιες ώρες.

Τέλος, ο χρόνος που χρειάζεται για να φορτωθούν τα δεδομένα στη βάση σε περίπτωση απώλειας τους είναι μειωμένος κατά 42% στην Cassandra σε σχέση με την PostgreSQL.

Ένα αρνητικό στοιχείο της μη σχεσιακής βάσης όπως έδειξαν τα πειράματα ήταν ο χρόνος του διαβάσματος δεδομένων από τη βάση. Το γεγονός αυτό όμως μπορεί να διορθωθεί στο μέλλον με την προσθήκη κόμβων στο σύμπλεγμά μας, αφού υλοποιήσαμε σύστημα ενός κόμβου. Εξάλλου οι πιο χρονοβόρες διαδικασίες είναι της ανάκτησης και της εισαγωγής των δεδομένων.

Συνοψίζοντας όλα τα παραπάνω σχηματίζουμε την παρακάτω λύση :

- Ο κεντρικός πίνακας που διατηρεί το ιστορικό των δεδομένων και στον οποίο γίνονται οι εισαγωγές να βρίκεται σε Cassandra. Στον πίνακα αυτό έχουμε δυνατότητα εκτέλεσης οποιουδήποτε ερωτήματος εκτός από υπολογισμούς aggregates.
- Για των υπολογισμό των aggregates διατηρούμε τους ενδιάμεσους πίνακες της PostgreSQL που περιγράψαμε και στην προηγούμενη λύση.

11 *Συμπεράσματα*

Σε αυτό το κεφάλαιο, θα παραθέσουμε τα συμπεράσματα της μελέτης που κάναμε. Θα περιγράψουμε συνολικά τα χαρακτηριστικά που θα πρέπει να έχει το σύστημα, ώστε να βελτιωθεί η απόδοσή του, συνοψίζοντας τα αποτελέσματα που λάβαμε από όλα τα πειράματα, που παρουσιάστηκαν στο κεφάλαιο 7 καθώς και στο κεφάλαιο 10.

11.1Σύνοψη

Στην πρώτη ενότητα θα δόσουμε μία σύντομη περιγραφή του συστήματος, όπως μας δόθηκε, και θα συνοψίσουμε τις βασικές αδυναμίες του. Στις επόμενες ενότητες, θα συνοψίσουμε τις δύο προτεινόμενες λύσεις στο αρχικό πρόβλημα που παρουσιάστηκαν αναλυτικά σε προηγούμενα κεφάλαια.

11.1.1 Απόδοση του υπάρχοντος συστήματος

Το βασικό πρόβλημα του συστήματος μας είναι η αργή εισαγωγή δεδομένων. Ταυτόχρονα με τις εισαγωγές που γίνονται στον κεντρικό πίνακα, εκτελούνται και άλλες ενέργειες. Συγκεκριμένα, βάσει του log analyzer της PostgreSQL, είδαμε ότι πραγματοποιούνται ερωτήσεις σχετικά με το ιστορικό των μετρήσεων για συγκεκριμένες χρονικές περιόδους. Το γεγονός αυτό απαιτεί την ύπαρξη index στον πίνακα, πράγμα που αποτελεί βασικό παράγοντα καθυστέρησης για λειτουργίες insert, update, delete.

Επίσης, το σύστημα διαθέτει τη μέγιστη προστασία στα εισερχόμενα δεδομένα, αφού μετά από κάθε εισαγωγή γίνεται προκαθορισμένα η ενέργεια commit, ώστε να μη χαθεί η παραμικρή πληροφορία σε περίπτωση κατάρρευσης του συστήματος. Αυτό έχει ως αποτέλεσμα ακόμα μεγαλύτερη καθυστέρηση στην εισαγωγή δεδομένων.

Ένα άλλο πρόβλημα του συστήματος είναι ότι λόγω του μεγάλου όγκου των δεδομένων, η ανάγκη που υπάρχει για την εκτέλεση κάποιων βασικών υπολογισμών δεν μπορεί να

καλυφθεί άμεσα, με sql queries. Αυτό έχει σαν αποτέλεσμα όλοι οι υπολογισμοί να γίνονται εξωτερικά της βάσης, με προγραμματιστικό τρόπο.

Τέλος, η διαδικασία ανάκτησης της βάσης σε περίπτωση κατάρρευσης του συστήματος καθυστερεί πάρα πολύ.

11.1.2 Σύνοψη λύσης με εφαρμογή των τεχνικών βελτιστοποίησης

Η πρώτη λύση που προτείνουμε προέκυψε από την εφαρμογή τεχνικών βελτιστοποίησης στο υπάρχον σύστημα.

Συνοψίζουμε την πρόταση μας στις παρακάτω αλλαγές :

- Partitioning στον κεντρικό πίνακα measurement_events ανά μέρα βάσει της στήλης measurement_time, χωρίς index μόνο στο partition στο οποίο γίνονται τα writes. Δημιουργία του index που λείπει μετά το πέρας της μέρας.
- Ενδιάμεσοι πίνακες που θα αποθηκεύουν τον μέσο όρο και το άθροισμα του measurement_value ανά μήνα για κάθε measurement_source_id, εκ των οποίων μόνο ο τρέχων μήνας θα είναι in memory ενώ όλοι οι υπόλοιποι μετά την ολοκλήρωσή τους θα αποθηκεύονται σε έναν πίνακα του δίσκου που θα κρατάει το ιστορικό. Στον πίνακα αυτόν θα έχει γίνει partitioning ανά μήνα. Επίσης θα αποθηκεύεται αντίγραφο ασφαλείας του in memory πίνακα ανά ένα εκατομμύριο insert.
- Αύξηση των insert ανά commit από 1 προς 1 σε 10 προς 1.

11.1.3 Λύση με συνδυασμό σχεσιακής και μη σχεσιακής βάσης

Στην δεύτερη λύση, η αρχιτεκτονική που προτείνουμε είναι όμοια με την αρχιτεκτονική της σχεσιακής λύσης, με τη διαφορά ότι ο partitioned πίνακας της τελευταίας τώρα θα είναι αποθηκευμένος σε Cassandra. Με αυτόν τον τρόπο θα μπορούμε όπως έχουμε εκτενώς περιγράψει στα παραπάνω κεφάλαια να είμαστε πάντα προετοιμασμένοι για χειρισμό περισσότερων δεδομένων.

Συνοψίζοντας σχηματίζουμε την παρακάτω λύση :

- Ο κεντρικός πίνακας που διατηρεί το ιστορικό των δεδομένων και στον οποίο γίνονται οι εισαγωγές να βρίκεται σε Cassandra. Στον πίνακα αυτό έχουμε δυνατότητα εκτέλεσης οποιουδήποτε ερωτήματος εκτός από υπολογισμούς aggregates.
- Για των υπολογισμό των aggregates διατηρούμε τους ενδιάμεσους πίνακες της PostgreSQL που περιγράψαμε και στην προηγούμενη λύση.

11.1.4 Σύγκριση

Στόχος της ενότητας αυτής είναι να παρουσιάσει την άμεση σύγκριση των δύο λύσεων με το αρχικό σύστημα.

Τα αποτελέσματα από όλα τα παραπάνω πειράματα φαίνονται συγκεντρωτικά στον παρακάτω πίνακα. Να σημειωθεί ότι όλα τα νούμερα προέρχονται είτε από προηγούμενους πίνακες είτε από το μέσο όρο των χρόνων εκτέλεσης των αντίστοιχων διεργασιών που, έχουν μετρηθεί παραπάνω, για μεγαλύτερη ακρίβεια και αξιοπιστία.

	PostgreSQL αρχικό	**PostgreSQL λυση**	**Cassandra λύση**
insert	7,518,435.376 ms	3,011,672.25 ms	117,000 ms
index	5,723,760.45 ms	114,475.209 ms	700 ms
select με ημερομηνία	1,979,715.206 ms	37,290.688 ms	411,000 ms
select aggregate	δεν ολοκληρώθηκε	10,400.629 ms	10,400.629 ms
restore	10:31:14’’	10:31:14’’	6:04:30’’

Πίνακας 40. Συγκεντρωτικός πίνακας για τη σύγκριση των συστημάτων

Για την **εισαγωγή ενός εκατομμυρίου γραμμών** σε indexed πίνακα ενός δισεκατομμυρίου εγγραφών βλέπουμε ότι το αρχικό σύστημα χρειάζεται λίγο περισσότερο από 2 ώρες. Η ίδια διαδικασία στην λύση μας με PostgreSQL η οποία αντιστοιχεί σε εισαγωγή ενός εκατομμυρίου γραμμών στο unindexed partition του κεντρικού πίνακα (περίπου 2,5 λεπτά) και στην ενημέρωση του ενδιάμεσου in-memory πίνακα (περίπου 50 λεπτά). Άρα δεδομένου ότι οι δύο διαδικασίες γίνονται ταυτόχρονα ο χρόνος της λύσης με PostgreSQL για ένα εκατομμύριο εισαγωγές είναι 50 λεπτά. Στην δεύτερη λύση με Cassandra ένα εκατομμύριο εισαγωγές ολοκληρώνονται μόλις σε 117 δευτερόλεπτα. Λόγω του ότι η λύση αυτή διατηρεί τους ενδιάμεσους πίνακες σε PostgreSQL χρειάζονται επίσης 50 λεπτά για την ενημέρωσή τους. Παρόλα αυτά στον πίνακα βάζουμε την πρώτη μέτρηση γιατί σε αντίθεση με την προηγούμενη λύση ο χρήστη έχει πρόσβαση στα δεδομένα του κεντρικού πίνακα μετά από 117 δευτερόλεπτα, πράγμα που δεν ήταν δυνατόν πριν λόγω της απουσία index από το τελευταίο partition.

Ο χρόνος εκτέλεσης της εντολής για **δημιουργία index** ήταν μία επίσης μία βασική διαφορά στα τρία συστήματα. Στο αρχικό σύστημα η δημιουργία ή το rebuild του index για έναν πίνακα περίπου 500 εκατομμυρίων γραμμών κοστίζει περισσότερο από 95 λεπτά και ο χρόνος ανεβαίνει εκθετικά όσο αυξάνεται το μέγεθος του πίνακα. Αντίθετα στην πρώτη λύση το χτίσιμο του index σε κάθε partition κοστίζει μόνο 2 λεπτά και δεν χρειάζεται να γίνει μελλοντικά rebuild, διότι δεν επηρεάζεται από επόμενες εισαγωγές δεδομένων. Στην Cassandra η δημιουργία index σε πίνακα ενός δισεκατομμυρίου εγγραφών κοστίζει μόλις 700 ms.

Για ερώτημα τύπου **select με where-clause** που αφορά κάποια indexed στήλη, το αρχικό σύστημα έκανε περίπου 32 λεπτά σε πίνακα 500 εκατομμυρίων γραμμών. Για το ίδιο ερώτημα σε ίδιο πίνακα με partitioning χρειάστηκαν περίπου 37 δευτερόλεπτα. Σε Cassandra, ο χρόνος εκτέλεσης ήταν 411 δευτερόλεπτα.

Για ερωτήματα που αφορούν τον **υπολογισμό aggregate**, το αρχικό σύστημα δεν ολοκλήρωσε την εντολή μέσα σε 24ώρες και γι'αυτό διακόπηκε. Ζητήσαμε τον υπολογισμό sum και avg της στήλης measurement_value για ένα μήνα για 100 χιλιάδες ids. Το ερώτημα αυτό είναι και στις δύο λύσεις εκτελείται σε περίπου 10 δευτερόλεπτα αφού και οι δύο λύσεις χρησιμοποιούν τους ίδιους ενδιάμεσους πίνακες.

Τέλος η διαδικασία του **restore**, που είναι ίδια για την PostrgeSQL (αρχικό σύστημα και πρώτη λύση), εκτελείται σε 10,5 ώρες για ανάκτηση πίνακα μεγέθους ενός δισεκατομμυρίου γραμμών. Η ίδια διαδικασία στην Cassandra διαρκεί 6 ώρες.

Συμπερασματικά βλέπουμε ότι η λύση με συνδυασμό των δύο συστημάτων PostgreSQL και Cassandra είναι πιο συμφέρουσα στα περισσότερα σημεία, αν εξαιρέσουμε την μικρή καθυστέρηση στα select, που όμως θα μπορούσε να εξαλειφθεί με τη χρήση περισσότερων κόμβων. Δεδομένου του ότι η μετάβαση από σχεσιακή σε μη σχεσιακή βάση έχει δυσκολίες και απαιτεί χρόνο για να εξοικειωθούν οι χρήστες με το νέο περιβάλλον, η πρώτη λύση σε PostgreSQL είναι επίσης πολύ ικανοποιητική.

11.2 Επίλογος-Μελλοντικές επεκτάσεις

Με τους ρυθμούς που αλλάζει το τοπίο στις βάσεις δεδομένων τα τελευταία 2 χρόνια, ολοένα και περισσότερες εταιρείες αποφασίζουν πως η καλύτερη λύση είναι η μετάβαση από σχεσιακά σε μη-σχεσιακά συστήματα. Επομένως, θεωρούμε εξαιρετικά ενδιαφέρον το θέμα του επανασχεδιασμού της βάσης σύμφωνα με τη NoSQL φιλοσοφία. Ανεξάρτητα από το ποιο σύστημα NoSQL θα χρησιμοποιηθεί, πιστεύουμε πως η υιοθέτηση ενός μη-σχεσιακού μοντέλου και το ξαναχτίσιμο της βάσης γύρω από αυτό θα ήταν ένα πρωτοποριακό βήμα. Εξίσου ενδιαφέρουσα θα ήταν η εγκατάσταση της επανασχεδιασμένης πια βάσης σε ένα σύμπλεγμα μερικών κόμβων ώστε να εκμεταλλεύεται την κατανεμημένη φύση της NoSQL και η αξιολόγησή της μέσα από δοκιμές και προσομοιώσεις ακραίων συνθηκών.

12 *Βιβλιογραφία*

[1] "Big Data: Beyond the Heap" by Datastax Corporation, 2012. Available at http://www.datastax.com/wp-content/uploads/2011/10/WP-DataStax-BigData.pdf

[2] http://en.wikipedia.org/wiki/Time_series

[3] http://en.wikipedia.org/wiki/Database

[4] Avi Silberscgatz, Henri F.Korth and S. Sudarshan, "Database System Concepts, 4th edition", 2001. Available at http://codex.cs.yale.edu/avi/db-book/db4/slide-dir/

[5] http://en.wikipedia.org/wiki/Edgar_F._Codd
http://en.wikipedia.org/wiki/Codd's_12_rules

[6] http://en.wikipedia.org/wiki/ACID

[7] http://www.postgresql.org/about

[8] http://en.wikipedia.org/wiki/PostgreSQL

[9] http://phppgadmin.sourceforge.net/doku.php

[10] Josh Berkus, "Five Steps to PostgreSQL Performance", 2013. Available at http://www.slideshare.net/PGExperts/five-steps-perform2013

[11] Greg Smith, Robert Treat, and Christopher Browne, "Tuning your PostgreSQL Server". Available at http://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server

[12] http://en.wikipedia.org/wiki/B-tree

[13] http://en.wikipedia.org/wiki/Cron

[14] Alexander Todorov, “Tips and Tricks: Memory Storage on PostgreSQL”. Available at http://magazine.redhat.com/2007/12/12/tip-from-an-rhce-memory-storage-on-postgresql/

[15] http://en.wikipedia.org/wiki/Partition_(database)

[16] http://www.postgresql.org/docs/9.2/static/ddl-partitioning.html

[17] http://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling

[18] http://www.postgresql.org/docs/9.2/static/upgrading.html

[19] Robin Schumacher, “Confessions of an Oracle DBA- Part 1, 2 and 3”, 2013. Available at

http://www.datastax.com/2013/06/confessions-of-an-oracle-dba-part-1
http://www.datastax.com/2013/06/confessions-of-an-oracle-dba-part-2
http://www.datastax.com/2013/06/confessions-of-an-oracle-dba-part-3

[20] http://en.wikipedia.org/wiki/NoSQL

[21] "NoSQL and RDBMS – Choose your weapon", 2010. Available at http://www.servicestack.net/mythz_blog/?p=129

[22] http://en.wikipedia.org/wiki/CAP_theorem

[23] http://en.wikipedia.org/wiki/Eventual_consistency

[24] http://en.wikipedia.org/wiki/MapReduce

[25] Dominic Williams, “HBase vs Cassandra: why we moved”, 2010. Available at http://ria101.wordpress.com/2010/02/24/hbase-vs-cassandra-why-we-moved

[26] Eben Hewitt, “Cassandra: The Defitive Guide”, 2011.

[27] Jonathan Ellis, “Apache Cassandra: Real-world scalability, today”, 2012. Available at http://www.slideshare.net/jbellis/cassandra-at-nosql-matters-2012

[28] “Cassandra 1.2 Documentation” by Datastax, 2013. Available at http://www.datastax.com/documentation/cassandra/1.2/webhelp/index.html

[29] Sylvain Lebresne "Using the Cassandra Bulk Loader", 2011. Available at http://www.datastax.com/dev/blog/bulk-loading