ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σύγκριση MySQL με MongoDB στο μετροπρόγραμμα TPC-H

Δημήτριος Α. Σαρτζετάκης

Επιβλέπων: Αλέξης Δελής, Καθηγητής

ΑΘΗΝΑ

ΑΠΡΙΛΙΟΣ 2017

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σύγκριση MySQL με MongoDB στο μετροπρόγραμμα TPC-H

Δημήτριος Α. Σαρτζετάκης
Α.Μ.: 1115201100099

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Αλέξης Δελής,** Καθηγητής

ΠΕΡΙΛΗΨΗ

Η ραγδαία αύξηση του όγκου των δεδομένων τα τελευταία χρόνια έχει ως αποτέλεσμα τη δημιουργία αρκετών προβλημάτων λόγω θεμάτων επεκτασιμότητας που έχουν οι τεχνολογίες αποθήκευσης δεδομένων, με την απόδοση τους να μειώνεται εκθετικά. Τα πράγματα γίνονται ακόμα χειρότερα όταν, παρά το μέγεθος των βάσεων δεδομένων, υπάρχουν και εκατομμύρια οντότητες που αιτούν πρόσβαση σε αυτές την ίδια χρονική στιγμή.

Επί δεκαετίες, το Σχεσιακό Σύστημα Διαχείρισης Βάσης Δεδομένων (RDBMS) ήταν η κυρίαρχη τεχνολογία αποθήκευσης των δεδομένων, πράγμα το οποίο προβλημάτισε πολλούς ερευνητές και επιστήμονες εξαιτίας των προβλημάτων που άρχισαν να εμφανίζουν. Αυτό είχε ως αποτέλεσμα την ανάπτυξη καινούργιων τεχνολογιών για την επίλυση αυτών αλλά και για την παροχή περισσοτέρων δυνατοτήτων. Μια από αυτές τις τεχνολογίες και πιο δημοφιλής είναι οι NoSQL Βάσεις Δεδομένων.

Ο στόχος αυτής της εργασίας είναι να εξετάσει και να συγκρίνει τις RDBMS και NoSQL Βάσεις Δεδομένων ως προς την απόδοση και τη συμπεριφορά, τόσο σε κανονικές συνθήκες όσο και όταν ο όγκος των δεδομένων αυξάνεται. Από τη σύγκριση των αποτελεσμάτων, διαπιστώθηκε ότι η MongoDB αποδίδει καλύτερα σε πολύπλοκα ερωτήματα, με κόστος όμως το μεγαλύτερο όγκο των βάσεων που δημιουργούν λόγω των διπλότυπων δεδομένων. Επίσης, το μέγεθος της βάσης δεδομένων δεν φαίνεται να αποτελεί καθοριστικό παράγοντα επιλογής αφού οι αποδόσεις δεν ήταν τρομερά διαφορετικές όσο αυξανόταν το μέγεθος των βάσεων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Σύγκριση Βάσεων Δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Σχεσιακές Βάσεις Δεδομένων, NoSQL Βάσεις Δεδομένων, MySQL, MongoDB, TPC-H

ABSTRACT

The rapid increase in data volume in recent years has resulted in the creation of many problems because of the scalability issues that data storage technologies have, so that their performances are being exponentially reduced. Despite the size of the databases, the things are getting even worse because there are also millions of entities requesting access to them at the same time.

For decades, Relational Database Management System (RDBMS) has been the dominant data storage technology, which has made many researchers and scientists think because of the problems they began to show. As a result, new technologies have been developed to solve these problems and to provide more possibilities as well. One of the most popular technologies is the NoSQL Databases.

The aim of this project is to examine and compare the performance and operational behavior of both RDBMS and NoSQL databases, in normal conditions and when the volume of data increases. By comparing the results, it was found that MongoDB performs better in complex queries, but at the cost of the larger databases they create due to the duplicates data. Furthermore, the size of the database does not seem to be a determining factor of choice because the results were not very different when the database size increased.

SUBJECT AREA: Comparison of Databases

KEYWORDS: Relational Databases, NoSQL Databases, MySQL, MongoDB, TPC-H

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΡΟΛΟΓΟΣ

Η παρούσα πτυχιακή εργασία εκπονήθηκε κατά το ακαδημαϊκό έτος 2016-2017 στα πλαίσια του Προπτυχιακού Προγράμματος Σπουδών του Τμήματος Πληροφορικής Και Τηλεπικοινωνιών του Εθνικού Και Καποδιστριακού Πανεπιστημίου Αθηνών, ως μερική εκπλήρωση υποχρεώσεων για την απόκτηση του πτυχίου.

1. ΕΙΣΑΓΩΓΗ

Η αποδοτική αποθήκευση και ανάκτηση των δεδομένων από πάντα ήταν σημαντικά θέματα λόγω των αυξανόμενων αναγκών των βιομηχανιών, των επιχειρήσεων και των ακαδημαϊκών κοινοτήτων. Όλο και περισσότερα δεδομένα έκαναν την εμφάνιση τους με αποτέλεσμα να χρειαστούν λύσεις με στόχο ένα κοινό οργανωτικό πλαίσιο για τη διαχείριση αυτών. Για αυτόν τον λόγο, από τη δεκαετία του 1960, δημιουργήθηκαν τα γνωστά μας Συστήματα Διαχείρισης Βάσεων Δεδομένων (DBMS) που μπορούν να ικανοποιήσουν αυτές τις ανάγκες και να προσφέρουν περαιτέρω δυνατότητες σε αντίθεση των συστημάτων διαχείρισης αρχείων. Αρχικά, είχαμε το δικτυακό μοντέλο CODASYL και το ιεραρχικό μοντέλο IMS, λίγο αργότερα έκανε την εμφάνισή του το σχεσιακό μοντέλο δεδομένων, δύο δεκαετία αργότερα δημιουργήθηκαν τα αντικειμενοστραφή συστήματα Β.Δ. και στα τέλη του 2000 έχουμε τα NoSQL συστήματα Β.Δ [1].

Από τη στιγμή που ο Edgar F. Codd [2] πρότεινε το σχεσιακό μοντέλο δεδομένων το 1970, οι σχεσιακές βάσεις δεδομένων (RDBMS) έγιναν ευρέως αποδεκτές και χρησιμοποιούνταν σχεδόν σε όλα τα συστήματα αποθήκευσης δεδομένων. Αυτό διότι υποστήριζε δυνατότητες που δεν είχαν τα μέχρι τότε συστήματα Β.Δ. Παρόλα αυτά, με την πάροδο των χρόνων έχουν εμφανιστεί και άλλα προβλήματα που μέχρι και οι σχεσιακές Β.Δ. δεν μπορούν να χειριστούν με ευκολία. Ο λόγος είναι η ταχύτατη αύξηση του όγκου των δεδομένων τα τελευταία χρόνια με αποτέλεσμα η διαχείριση αυτών να γίνεται δύσκολη έως αδύνατη. Ευτυχώς, πολλά είδη βάσεων δεδομένων έχουν ήδη αναπτυχθεί ώστε να λύσουν το πρόβλημα της επεκτασιμότητας και να προσφέρουν περαιτέρω δυνατότητες όπως είναι οι NoSQL, NewSQL, Big Data βάσεις δεδομένων [3].

Η σύγκριση των συστημάτων βάσεων δεδομένων είναι πολύ σημαντική διαδικασία και μπορούμε να βγάλουμε συμπεράσματα όπως την ικανότητα αυτών να διαχειρίζονται και να επεξεργάζονται τα δεδομένα και πως αυτά ανταπεξέρχονται όταν ο αριθμός των δεδομένων και δοσοληψιών αυξάνεται. Αυτό βοηθάει στη σωστή επιλογή της βάσης δεδομένων που θα χρησιμοποιηθεί για παράδειγμα σε μια επιχείρηση, διότι μια λανθασμένη επιλογή θα έχει σοβαρές συνέπειες μακροπρόθεσμα στην ίδια την επιχείρηση. Η αλλαγή της βάσης δεδομένων σε μια εφαρμογή η οποία χρησιμοποιείται ευρέως, είναι μια δαπανηρή και δύσκολη διαδικασία.

Στην εργασία αυτή, θα επικεντρωθούμε κυρίως στη σύγκριση των δύο πιο δημοφιλών συστημάτων βάσεων δεδομένων που είναι οι RDBMS και NoSQL. Πιο συγκεκριμένα, θα χρησιμοποιηθεί η σχεσιακή βάση δεδομένων MySQL και η NoSQL βάση δεδομένων MongoDB. Θα συγκριθούν στην απόδοση όταν ο όγκος των δεδομένων και ο αριθμός των συνδέσεων αυξάνεται χρησιμοποιώντας το μετροπρόγραμμα TPC-H (TPC-H benchmark). Βασιζόμενοι στα αποτελέσματα των συγκρίσεων θα μπορούμε να εξάγουμε συμπεράσματα που θα βοηθήσουν τη διαδικασία επιλογής ανάμεσα στις δύο αυτές βάσεις δεδομένων.

1.1 Περιορισμοί

Λογικό είναι λόγω χώρου και χρόνου της εργασίας να μην μπορέσουμε να συγκρίνουμε όλα τα συστήματα Β.Δ., με διαφορετικά μοντέλα δεδομένων και σε διαφορετικές καταστάσεις χρήσεως. Για ‘αυτό η σύγκριση θα γίνει ανάμεσα στην σχεσιακή MySQL και την MongoDB. Επιπλέον, θα χρησιμοποιηθεί ένα συγκεκριμένο σχήμα βάσης δεδομένων (database schema) και μερικά ερωτήματα (database queries) από το μετροπρόγραμμα TPC-H. Τέλος, εξαιτίας των περιορισμένων υπολογιστικών συστημάτων που έχουμε στη διάθεσή μας δεν είχαμε τη δυνατότητα να τρέξουμε τα πειράματα της εργασίας με άλλες ρυθμίσεις των βάσεων δεδομένων, όπως είναι οι Κατανεμημένες Βάσεις Δεδομένων (Distributed Databases).

1.2 Οδηγός Αναγνώστη

- **Κεφάλαιο 2 :** Δίνει μια εισαγωγή στα συστήματα των βάσεων δεδομένων και σε ορισμένες σημαντικές έννοιες αυτών όπως το θεώρημα CAP, τις ACID ιδιότητες, Replication και Sharding.

- **Κεφάλαιο 3 :** Αναλύει τις τεχνολογίες και τα μοντέλα που χρησιμοποιήθηκαν στις δύο βάσεις δεδομένων για τα πειράματα της εργασίας.

- **Κεφάλαιο 4 :** Εμφανίζει τα αποτελέσματα και τα αναλύει βγάζοντας επιμέρους συμπεράσματα για το κάθε ένα ξεχωριστά.

- **Κεφάλαιο 5 :** Ολοκληρώνει την εργασία και συνοψίζει τα αποτελέσματα.

- **Κεφάλαιο 6 :** Παρέχει πληροφορίες για επεκτάσεις που μπορούν να γίνουν σε αυτήν την εργασία ή και για μελλοντικές έρευνες με στόχο την αντικειμενικότητα των αποτελεσμάτων.

2. ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

Σε αυτό το κεφάλαιο δίνεται μια γρήγορη εισαγωγή στις Βάσεις Δεδομένων και σε κάποια στοιχεία αυτών όπως είναι το θεώρημα CAP, οι ACID ιδιότητες, Replication και Sharding. Τέλος, παρουσιάζονται τα διαφορετικά είδη των Βάσεων Δεδομένων με περαιτέρω ανάλυση στις MySQL και MongoDB Β.Δ.

2.1 Βάσεις Δεδομένων

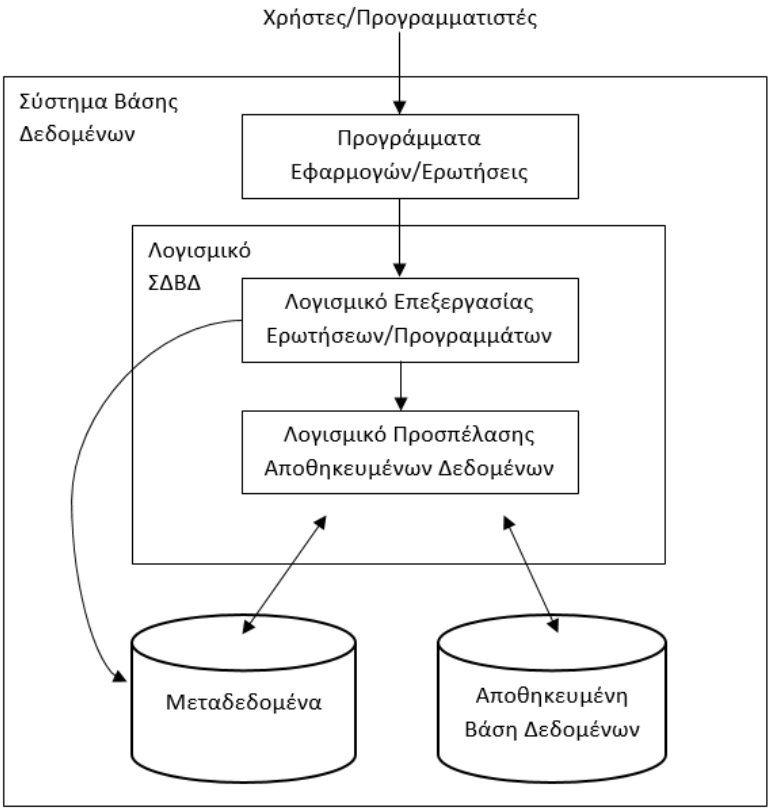
Οι βάσεις δεδομένων και τα συστήματα των βάσεων δεδομένων αποτελούν ένα σημαντικό στοιχείο της καθημερινής ζωής στη σύγχρονη κοινωνία και εξασκούν σημαντική επίδραση στην αυξανόμενη χρήση των υπολογιστών.

Βάση δεδομένων (database) είναι μια συλλογή από σχετιζόμενα δεδομένα. Με τον όρο **δεδομένα** εννοούμε γνωστά γεγονότα που μπορούν να καταγραφούν και που έχουν κάποια υπονοούμενη σημασία.

Ο πιο πάνω ορισμός μιας βάσης δεδομένων είναι αρκετά γενικός ενώ η συνήθης χρήση του όρου βάση δεδομένων είναι αρκετά πιο περιορισμένη. Μια βάση δεδομένων έχει τις ακόλουθες υπονοούμενες ιδιότητες:

- Μια βάση δεδομένων αναπαριστά κάποια άποψη του πραγματικού κόσμου, η οποία μερικές φορές λέγεται **μικρόκοσμος** (miniworld) ή **Πεδίο Αναφοράς** (Universe of Discourse, UoD). Οι αλλαγές στο μικρόκοσμο αντανακλώνται στη βάση δεδομένων.
- Μια βάση δεδομένων είναι μια λογικά συνεκτική συλλογή δεδομένων που έχει κάποια εγγενή σημασία Μια τυχαία διευθέτηση δεδομένων δεν είναι σωστό να αναφέρεται ως βάση δεδομένων.
- Μια βάση δεδομένων σχεδιάζεται, χτίζεται και γεμίζει με δεδομένα για κάποιο συγκεκριμένο σκοπό. Προορίζεται για μια συγκεκριμένη ομάδα χρηστών και για κάποιες προκαθορισμένες εφαρμογές για τις οποίες οι χρήστες αυτοί ενδιαφέρονται.

Σύστημα διαχείρισης βάσεων δεδομένων (ΣΔΒΔ) (database management system-DBMS) είναι μια συλλογή από προγράμματα που επιτρέπουν στους χρήστες να δημιουργήσουν και να συντηρήσουν μια βάση δεδομένων. Επομένως, το ΣΔΒΔ είναι ένα γενικής χρήσης (general-purpose) σύστημα λογισμικού που διευκολύνει τις διαδικασίες ορισμού, χειρισμού και διαμοιρασμού βάσεων δεδομένων για διάφορες εφαρμογές. Άλλες σημαντικές λειτουργίες που παρέχει το ΣΔΒΔ περιλαμβάνουν την προστασία της βάσης δεδομένων και τη συντήρηση της για μακρύ χρονικό διάστημα. Ένα πρόγραμμα εφαρμογής προελαύνει μια βάση δεδομένων στέλνοντας στο ΣΔΒΔ ερωτήσεις ή αιτήματα για τα δεδομένα. Μια ερώτηση τυπικά προκαλεί την ανάκτηση κάποιων δεδομένων – μια **δοσοληψία** μπορεί να προκαλέσει ανάγνωση και εγγραφή δεδομένων στη βάση δεδομένων [4, pages 20-21].



Εικόνα 2.1: Ένα απλουστευμένο περιβάλλον συστήματος βάσης δεδομένων

2.1.1 ACID ιδιότητες

Οι δοσοληψίες πρέπει να έχουν διάφορες ιδιότητες που συχνά αναφέρονται ως ιδιότητες ACID και πρέπει να διασφαλίζονται από τις μεθόδους ελέγχου συνδρομικότητας και ανάκαμψης του ΣΔΒΔ. Οι παρακάτω είναι οι ιδιότητες ACID [4, pages 618-619]:

- **Ατομικότητα (Atomicity)**: Μια δοσοληψία είναι μια ατομική μονάδα επεξεργασίας, η οποία είτε εκτελείται εξ ολοκλήρου είτε δεν εκτελείτε καθόλου.

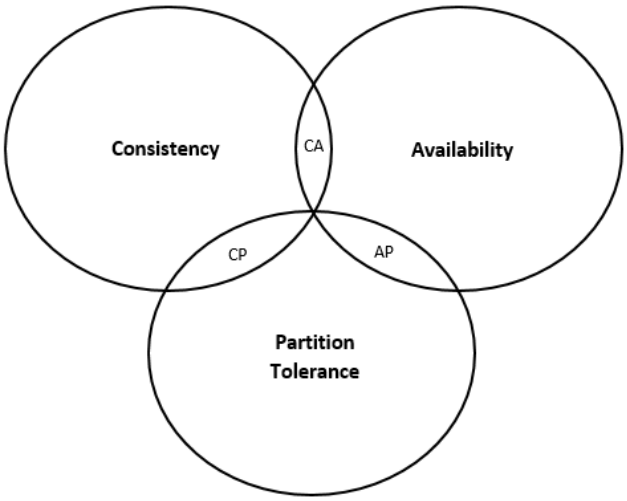
- **Διατήρηση της συνέπειας (Consistency preservation)**: Μια δοσοληψία διατηρεί την συνέπεια αν η ολοκληρωμένη εκτέλεσή της οδηγεί τη βάση δεδομένων από μια συνεπή κατάσταση σε μια άλλη.

- **Απομόνωση (Isolation)**: Μια δοσοληψία πρέπει να εμφανίζεται σαν να εκτελείται απομονωμένη από άλλες δοσοληψίες. Αυτό σημαίνει ότι η εκτέλεση μιας δοσοληψίας δεν πρέπει να εμπλέκεται με οποιαδήποτε άλλη δοσοληψία που εκτελείται ταυτόχρονα.

- **Διάρκεια (Durability ή permanency)**: Οι αλλαγές που εκτελούνται σε μια βάση δεδομένων από μια επικυρωμένη δοσοληψία πρέπει να μένουν σταθερές στη βάση δεδομένων. Οι αλλαγές αυτές δεν είναι δυνατόν να χαθούν λόγω οποιασδήποτε αποτυχίας.

2.1.2 Θεώρημα CAP

Το Θεώρημα CAP είναι από τις πιο σημαντικές αρχές για τα συστήματα κατανεμημένων βάσεων δεδομένων που προτάθηκε από τον Eric Allen Brewer το 2000 [5] και αποδείχθηκε αργότερα από τον Seth Gilbert και την Nancy Lynch [6]. Το θεώρημα μας λέει ότι σε ένα κατανεμημένο σύστημα μόνο δύο από τα τρία χαρακτηριστικά μπορούν να συνυπάρχουν όπως φαίνεται και στην Εικόνα 2.2. Τα τρία χαρακτηριστικά εξηγούνται λεπτομερώς παρακάτω.



Εικόνα 2.2: Θεώρημα CAP

- **Συνέπεια (Consistency)**: Κάθε ερώτημα που γίνεται στη βάση, θα διαβάζει τα τελευταία δεδομένα που καταχωρήθηκαν σε αυτήν και όχι παλαιότερα.

- **Διαθεσιμότητα (Availability)**: Θέτει ως προϋπόθεση ότι κάθε ερώτημα θα πάρει μια ενδεχόμενη απάντηση. Η γρήγορη ανταπόκριση είναι προφανώς προτιμότερη από μια αργή, ωστόσο, όσον αφορά το περιεχόμενο του θεωρήματος, μια ενδεχόμενη απάντηση μπορεί να προκαλέσει προβλήματα. (Στα περισσότερα συστήματα, μια απάντηση η οποία έχει έρθει αργά προκαλεί τόσο κακό όσο το να μην υπάρχει καθόλου απάντηση).

- **Ανοχή στις κατατμήσεις που οφείλονται στο δίκτυο (Tolerance to network Partitions)**: Η ανοχή στις κατατμήσεις σημαίνει ότι σε περίπτωση που κάποιο μέρος του συστήματος αποκοπεί από το υπόλοιπο λόγω βλάβης στο δίκτυο (ή και βλάβης σε κάποια μηχανήματα), τότε το σύστημα θα πρέπει να είναι σε θέση να συνεχίσει να εξυπηρετεί αιτήματα και να λειτουργεί.

Όσον αναφορά τις δύο πιο γνωστές βάσεις δεδομένων RDBMS και NoSQL η φιλοσοφία τους πάνω στο θεώρημα CAP διαφέρει. Οι βάσεις δεδομένων που έχουν σχεδιαστεί με βάση τις ACID ιδιότητες, όπως είναι οι RDBMS, επιλέγουν κυρίως την συνέπεια αντί της διαθεσιμότητας ενώ τα συστήματα που έχουν σχεδιαστεί με την φιλοσοφία BASE (eventual consistency), συνήθως στις περισσότερες NoSQL βάσεις για παράδειγμα, επιλέγουν την διαθεσιμότητα αντί της συνέπειας [7].

2.1.3 Replication

Η τεχνική της ομοιοτυπίας δεδομένων μέσω της αντιγραφής αυτών (data replication) που χρησιμοποιείται κατά τη διαδικασία σχεδιασμού κατανεμημένων βάσεων δεδομένων επιτρέπει σε δεδομένα της βάσης να αποθηκευτούν σε περισσότερους από έναν κόμβους. Η ομοιοτυπία είναι χρήσιμη για τη βελτίωση της διαθεσιμότητας των δεδομένων.

Η πιο ακραία περίπτωση είναι να επαναληφθεί ολόκληρη η βάση δεδομένων σε κάθε κόμβο του κατανεμημένου συστήματος, δημιουργώντας έτσι μια πλήρως ομοιοτυπημένη βάση δεδομένων. Με τον τρόπο αυτό μπορεί να αυξηθεί σημαντικά η διαθεσιμότητα, διότι το σύστημα μπορεί να εξακολουθήσει να λειτουργεί, ακόμη κι αν ένας μόνον κόμβος είναι σε λειτουργία. Επίσης βελτιώνει την απόδοση των ανακτήσεων για καθολικές ερωτήσεις, διότι το αποτέλεσμα μιας τέτοιας ερώτησης μπορεί να εξαχθεί τοπικά από οποιονδήποτε κόμβο και επομένως μια ερώτηση ανάκτησης μπορεί να εκτελεσθεί στον κόμβο στον οποίο υποβλήθηκε, αν ο κόμβος αυτός περιλαμβάνει διακομιστή. Το μειονέκτημα της πλήρους ομοιοτυπίας είναι ότι μπορεί να επιβαρύνει σημαντικά τις ενημερώσεις, διότι μια απλή, σε λογικό επίπεδο ενημέρωση πρέπει να εκτελεσθεί σε κάθε αντίγραφο της βάσης δεδομένων προκειμένου να διατηρηθούν τα αντίγραφα συνεπή. Το φαινόμενο είναι πιο έντονο αν υπάρχουν πολλά αντίγραφα της βάσης δεδομένων.

Στο άλλο άκρο της πλήρους ομοιοτυπίας βρίσκεται η απουσία ομοιοτυπίας. Με άλλα λόγια, κάθε τεμάχιο (υποσύνολο των δεδομένων μιας βάσης) αποθηκεύετε σε έναν κόμβο και μόνο. Τέλος, υπάρχει και η μερική ομοιοτυπία των δεδομένων που βρίσκεται μεταξύ των δύο άκρων. Στην περίπτωση αυτή μερικά τεμάχια της βάσης δεδομένων μπορεί να επαναλαμβάνονται ενώ άλλα όχι [4, page 733].

2.1.4 Sharding

Shard στα αγγλικά σημαίνει θραύσμα. Ονόμασαν έτσι αυτή τη διαδικασία (sharding) ώστε να τονίσουν το “σπάσιμο” των δεδομένων σε πολλά μικρότερα κομμάτια. Συγκεκριμένα, το σπάσιμο των δεδομένων (θα το αναφέρουμε ως sharding στο εξής) είναι μία μέθοδος που μας επιτρέπει να αποθηκεύουμε τα δεδομένα μας σε πολλά ξεχωριστά μηχανήματα (servers). Η MongoDB, χρησιμοποιεί το sharding ώστε να μπορεί να υποστηρίξει εφαρμογές που διαχειρίζονται πολύ μεγάλο όγκο δεδομένων (large data set) και έχουν υψηλό εύρος ζώνης (high data transfer rate).

Οι βάσεις δεδομένων που αποθηκεύουν πολύ μεγάλο όγκο δεδομένων και έχουν υψηλό εύρος ζώνης, “ζορίζονται” όταν λειτουργούν μόνο σε ένα μηχάνημα (single server). Ο υψηλός ρυθμός ερωτημάτων μπορεί να εξαντλήσει την επεξεργαστική δύναμη του μηχανήματος. Επίσης, ο μεγάλος όγκος δεδομένων της βάσης πολλές φορές ξεπερνά τις αποθηκευτικές δυνατότητες ενός μηχανήματος. Τέλος, το σύνολο εργασίας (working set) μπορεί να ξεπερνά σε μέγεθος τη μνήμη RAM του μηχανήματος και αυτό έχει ως αποτέλεσμα να λειτουργεί ακατάπαυστα ο σκληρός δίσκος κάτι που του προκαλεί μεγάλη φθορά με τον καιρό. Υπάρχουν δύο βασικές προσεγγίσεις ώστε τα συστήματα βάσεων δεδομένων να αντιμετωπίσουν το πρόβλημα αυτό, η κατακόρυφη κλιμάκωση και το sharding.

- Η **κατακόρυφη κλιμάκωση ή vertical scaling** προσθέτει σε ένα μηχάνημα περισσότερους επεξεργαστές και περισσότερο αποθηκευτικό χώρο, ώστε να αυξήσει τις επιδόσεις του συστήματος αυτού. Όμως, η βελτίωση ενός υπολογιστικού συστήματος με αυτόν τον τρόπο έχει πολύ μεγαλύτερο κόστος συγκριτικά με μικρότερα συστήματα, ενώ επίσης αυξάνεται σημαντικά η πολυπλοκότητα του. Ως αποτέλεσμα, υπάρχει ένα συγκεκριμένο “ταβάνι” μέχρι το οποίο μπορούμε να βελτιώσουμε ένα υπολογιστικό σύστημα, ώστε να επιτύχουμε κατακόρυφη κλιμάκωση.

- Το **sharding ή horizontal scaling**, αντιθέτως, χωρίζει το σύνολο των δεδομένων σε μικρότερα επιμέρους κομμάτια δεδομένων και τα διανέμει σε πολλά διαφορετικά μηχανήματα. Το κάθε ένα από αυτά τα μηχανήματα που στο εξής θα αποκαλούμε shard, είναι μία ανεξάρτητη βάση δεδομένων. Όμως οι shards στο σύνολό τους, αποτελούν θεωρητικά μία και μοναδική ενιαία βάση δεδομένων.

Με την τεχνική του sharding, καταφέρνουμε να μειώσουμε τον αριθμό των λειτουργιών/εργασιών που επιτελεί ο κάθε shard, δηλαδή ο κάθε ένας από τους servers ενός συμπλέγματος (cluster). Αυτό συμβαίνει διότι ο κάθε shard είναι υπεύθυνος για την επεξεργασία μόνο των δεδομένων που έχει ο ίδιος. Για παράδειγμα, αν θέλουμε να εισάγουμε δεδομένα σε κάποιο έγγραφο (περίπτωση σε MongoDB), τότε η εφαρμογή χρειάζεται να έχει πρόσβαση μόνο στον shard που περιέχει το συγκεκριμένο έγγραφο. Τέλος, με το sharding μειώνεται το πλήθος των δεδομένων που χρειάζεται να αποθηκεύσει ο κάθε server. Ο κάθε shard αποθηκεύει όλο και λιγότερα δεδομένα καθώς το cluster μεγαλώνει. Για παράδειγμα, αν μία βάση δεδομένων έχει μέγεθος 1 TB και έχουμε 4 shards, τότε ο καθένας θα αποθηκεύσει 256GB. Εάν είχαμε ένα cluster με 40 shards, τότε ο καθένας από αυτούς θα χρειαζόταν να αποθηκεύσει μόνο 25GB δεδομένων.

Συνεπώς, ένα cluster μπορεί να επεκτείνει την επεξεργαστική του ισχύ, αλλά και τις αποθηκευτικές του δυνατότητες ‘οριζόντια’, δηλαδή χωρίς να υπάρχει ταβάνι που να το περιορίζει [8, pages 37-38].

2.2 Είδη Βάσεων Δεδομένων

Οι άνθρωποι ξεκίνησαν να αποθηκεύουν διάφορες πληροφορίες από πολύ παλιά. Στα αρχαία χρόνια, αναπτύχθηκαν περίτεχνα συστήματα βάσεων δεδομένων από κυβερνητικά γραφεία, βιβλιοθήκες, νοσοκομεία, επιχειρηματικές οργανώσεις και μερικές από τις βασικές έννοιες και αρχές που χρησιμοποιήθηκαν για την κατασκευή των συστημάτων αυτών εξακολουθούν να χρησιμοποιούνται μέχρι και σήμερα. Από την αρχαιότητα μέχρι τα σχεσιακά και αντικειμενοστραφή συστήματα, η τεχνολογία των βάσεων δεδομένων έχει περάσει από πολλές γενιές και η εξέλιξή της με τα χρόνια είναι συναρπαστική. Στη συνέχεια, αναφερόμαστε συνοπτικά στα είδη των βάσεων δεδομένων όπως εμφανίστηκαν χρονολογικά και αργότερα αναλύουμε περαιτέρω τις δύο πιο γνωστές βάσεις δεδομένων (υπο-ενότητα 2.2.1 και 2.2.2), οι οποίες έχουν και την μεγαλύτερη ανταπόκριση από τον κόσμο, που είναι οι σχεσιακές και οι NoSQL [1].

Από τα μέσα της δεκαετίας του 1960 δύο κύρια μοντέλα αναπτύχθηκαν – το **διαδικτυακό μοντέλο CODASYL** (Conference on Data System Language) και το **ιεραρχικό μοντέλο IMS** (Information Management System). Η πρώτη γενιά των βάσεων δεδομένων την χαρακτήριζαν ως βάσεις πλοήγησης διότι οι εφαρμογές αποκτούσαν πρόσβαση στα δεδομένα ακολουθώντας δείκτες από μια εγγραφή σε μια άλλη.

Το 1970 ο E.P. Codd [2] σχεδίασε το **σχεσιακό μοντέλο** βάσεων δεδομένων. Αυτό το μοντέλο είχε διαφορετική φιλοσοφία από τα μέχρι τότε συστήματα επιμένοντας ότι οι εφαρμογές θα πρέπει να αναζητούν τα δεδομένα με βάση το περιεχόμενο αυτών και όχι ακολουθώντας συνδέσμους.

Το 1985 για πρώτη φορά αναπτύχθηκε η έννοια της αντικειμενοστραφής βάσης δεδομένων με τον όρο ‘αντικειμενοστραφές σύστημα βάσης δεδομένων (object-oriented database system)’. Τα αντικειμενοστραφή συστήματα στηρίζονταν στο ομώνυμο μοντέλο δεδομένων που διέφερε από το σχεσιακό και επανάφερε στο προσκήνιο κάποια από τα πλεονεκτήματα των δικτυακών βάσεων δεδομένων. Πιο συγκεκριμένα σε ένα **αντικειμενοστραφές σύστημα διαχείρισης βάσεων δεδομένων (OODBMS)** οι

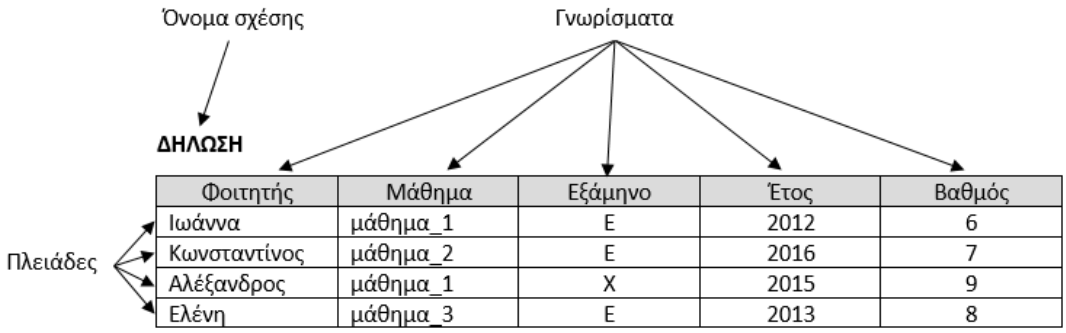
πληροφορίες παριστάνονται με τη μορφή αντικειμένων όπως αυτά χρησιμοποιούνται στις αντικειμενοστραφείς γλώσσες προγραμματισμού.

Για να ξεπεραστούν κάποια προβλήματα που είχαν τα αντικειμενοστραφή συστήματα διαχείρισης βάσεων δεδομένων (OODMS) και για να διερευνηθούν πλήρως οι δυνατότητες του σχεσιακού μοντέλου, τα **αντικειμενο-σχεσιακά συστήματα διαχείρισης βάσεων δεδομένων (ORDBMS)** προέκυψαν κατά τη δεκαετία του 1990.

Από τις αρχές του 21ου αιώνα λόγω προβλημάτων που άρχιζαν να εμφανίζουν τα σχεσιακά μοντέλα βάσεων δεδομένων αναπτύχθηκαν οι λεγόμενες **NoSQL** (που σημαίνει γενικά «non SQL», «non relational» ή «not only SQL») βάσεις δεδομένων. Σε αυτές θα αναφερθούμε αναλυτικότερα στην υπο-ενότητα 2.2.2.

2.2.1 Σχεσιακές Βάσεις Δεδομένων

Το σχεσιακό μοντέλο παριστάνει τη βάση δεδομένων ως μια συλλογή από σχέσεις και μιλώντας χωρίς αυστηρότητα, μπορούμε να πούμε ότι κάθε σχέση μοιάζει με έναν πίνακα ή, κατά κάποιο τρόπο, με ένα "επίπεδο" αρχείο εγγράφων. Όταν μια σχέση αντιμετωπίζεται ως ένας πίνακας (table) τιμών, κάθε γραμμή στον πίνακα παριστάνει μια συλλογή από τιμές δεδομένων που σχετίζονται. Στο σχεσιακό μοντέλο κάθε γραμμή σε έναν πίνακα παριστάνει ένα γεγονός που τυπικά αντιστοιχεί σε μια οντότητα ή μια συσχέτιση του πραγματικού κόσμου. Το όνομα του πίνακα και τα ονόματα των στηλών χρησιμοποιούνται βοηθητικά προκειμένου να ερμηνευθεί η σημασία των τιμών σε κάθε γραμμή του πίνακα. Στην τυπική ορολογία του σχεσιακού μοντέλου, μια γραμμή λέγεται πλειάδα, η επικεφαλίδα μιας στήλης λέγεται γνώρισμα, και ολόκληρος ο πίνακας λέγεται σχέση [4, pages 64-66]. Ένα παράδειγμα ενός τέτοιου πίνακα φαίνεται στην Εικόνα 2.3.



Φοιτητής	Μάθημα	Εξάμηνο	Έτος	Βαθμός
Ιωάννα	μάθημα_1	Ε	2012	6
Κωνσταντίνος	μάθημα_2	Ε	2016	7
Αλέξανδρος	μάθημα_1	Χ	2015	9
Ελένη	μάθημα_3	Ε	2013	8

Εικόνα 2.3: Τα γνωρίσματα και οι πλειάδες της σχέσης ΔΗΛΩΣΗ

Ένα σχήμα (database schema) είναι μια οπτική αναπαράσταση και περιγραφή μιας βάσης δεδομένων που περιλαμβάνει περιγραφές των δομών, των τύπων δεδομένων και των περιορισμών της βάσης αυτής. Ένα παράδειγμα τέτοιου σχήματος μπορείτε να δείτε στην υπο-ενότητα 2.3.2.

Κάθε σχέση της βάσης δεδομένων θα πρέπει να έχει ένα πρωτεύον κλειδί. Επίσης, μπορεί να έχει κανένα, ένα ή και περισσότερα ξένα κλειδιά. Γενικά, η έννοια του κλειδιού μιας σχέσης περιγράφεται παρακάτω:

- **Υπερκλειδί**: Υποσύνολο γνωρισμάτων της σχέσης για τα οποία αν προσδιορίσουμε τιμές σε όλα, υπάρχει μια μοναδική πλειάδα το πολύ με αυτές τις τιμές.

- **Υποψήφιο κλειδί**: Ένα υπερκλειδί για το οποίο αν αφαιρέσεις οποιοδήποτε γνώρισμα παύει να είναι υπερκλειδί.
- **Πρωτεύον κλειδί**: Επιλογή του σχεδιαστή βάσεων δεδομένων από τα υποψήφια κλειδιά (συνήθως επιλέγεται το πιο μικρό).
- **Ξένο κλειδί**: Υποσύνολο γνωρισμάτων μιας σχέσης το οποίο παίρνει τιμές μόνο από το αντίστοιχο πρωτεύον κλειδί μιας άλλης σχέσης.

2.2.2 NoSQL Βάσεις Δεδομένων

Τα NoSQL (διαβάζεται Not Only SQL) συστήματα και βάσεις δεδομένων πρόκειται για μια ευρεία ομάδα συστημάτων διαχείρισης βάσεων δεδομένων (database management system) που το κύριο χαρακτηριστικό του είναι η μη τήρηση του μοντέλου RDBMS (Relational Database Management System), το οποίο και χρησιμοποιείται κατά κόρον. Οι NoSQL βάσεις δεδομένων γενικώς δεν χρησιμοποιούν κάποιο δομημένο σύστημα για τα στοιχεία που περιλαμβάνουν, όπως πχ πίνακες, ούτε χρησιμοποιούν κάποια Structured Query Language (SQL) για την διαχείριση των δεδομένων, αλλά χρησιμοποιούν αποκλειστικά non-relational τρόπους οργάνωσης και ανάλυσης των δεδομένων.

Τα NoSQL συστήματα κατά κύριο λόγο είναι βελτιστοποιημένα (optimized) ώστε να ανακτούν και να επισυνάπτουν δεδομένα. Η μειωμένη ευελιξία του χρόνου εκτέλεσης σε σύγκριση με συστήματα SQL (δηλαδή με τα RDBMS) αντισταθμίζεται από την σημαντική αύξηση στην απόδοση (και την επεκτασιμότητα) για ορισμένα μοντέλα δεδομένων. Τα δεδομένα θα μπορούσαν να είναι δομημένα, αλλά αυτό είναι στην πραγματικότητα ασήμαντο καθώς αν κάτι έχει ουσιαστική σημασία στα NoSQL συστήματα αυτό είναι η ικανότητα να αποθηκεύουν και να ανακτούν μεγάλες ποσότητες δεδομένων, «αδιαφορώντας» για τις σχέσεις μεταξύ των στοιχείων αυτών.

Τα NoSQL συστήματα έχουν φτιαχτεί έτσι ώστε να μπορούν να διαχειριστούν μεγάλες ποσότητες δεδομένων χωρίς κατ' ανάγκη να διατηρούν μία συγκεκριμένη δομή (schema). Επίσης, οι μέθοδοι υλοποίησης και εφαρμογής τους αξιοποιούν μια αρχιτεκτονική που επιτρέπει (και ίσως διευκολύνει) την κατανεμημένη λειτουργία του συστήματος. Έτσι με αυτόν τον τρόπο το σύστημα μπορεί θεωρητικά να «απογειωθεί» σε επιδόσεις, από την στιγμή που μπορούν να προστεθούν θεωρητικά άπειροι servers όπου κατανεμημένα θα επεξεργάζονται τα δεδομένα του συστήματος.

Οι κατηγορίες των NoSQL βάσεων δεδομένων είναι [9, 10]:

- **Key-value stores**: Δημιουργήθηκαν με κύρια ιδέα την ύπαρξη ενός hash table όπου υπάρχει ένα μοναδικό κλειδί και ένας δείκτης στοχεύοντας σε ένα συγκεκριμένο στοιχείο. Αυτό το είδος mapping συνήθως συνοδεύεται από μηχανισμούς cache, για την καλύτερη απόδοση του συστήματος. Μερικές είναι οι εξής: Riak, Berkeley DB, Redis.

- **Wide-column stores**: Σχεδιάστηκαν έτσι ώστε να διαχειρίζονται πολύ μεγάλα ποσά δεδομένων που είναι κατανεμημένα σε διάφορους servers. Όπως και στην κατηγορία key-value stores, έτσι κι εδώ υπάρχουν συγκεκριμένα κλειδιά (keys) που στοχεύουν όμως σε περισσότερα από ένα στοιχεία. Οι rows εδώ αναγνωρίζονται από ένα μοναδικό row key ενώ οι στήλες είναι οργανωμένες σε column families («οικογένειες στηλών»). Μερικές είναι οι εξής: Cassandra, HBase, BigTable.

- **Document Databases**: Είναι όμοιες με τις key-value stores. Τα δεδομένα σε αυτή την περίπτωση είναι οργανωμένα από «συλλογές» key-valued «συλλογών» δεδομένων. Μερικές είναι οι εξής: MongoDB, CouchDB.

- **Graph stores**: Είναι βασισμένη σε κόμβους (nodes), τις σχέσεις μεταξύ αυτών των κόμβων και τις ιδιότητές τους. Αντί για πίνακες με στήλες και σειρές, εδώ υπάρχει ένα ευέλικτο γραφικό μοντέλο (graph model) που μπορεί να χρησιμοποιηθεί και να αναπτυχθεί παράλληλα σε πολλά μηχανήματα (servers – κόμβους). Μερικές είναι οι εξής: Neo4J, Giraph.

2.3 MySQL Βάσεις Δεδομένων

Η MySQL είναι ένα από τα πιο γνωστά συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων. Το SQL μέρος σημαίνει “Structured Query Language”, η οποία είναι μια γλώσσα υπολογιστών, που σχεδιάστηκε για τη διαχείριση δεδομένων, σε ένα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων (RDBMS).

Μερικά σημαντικά πλεονεκτήματα της MySQL είναι [12]:

- Η υψηλή απόδοση και διαθεσιμότητα.
- Η υποστήριξη δοσοληψιών με πλήρης ACID ιδιότητες.
- Η ισχυρή προστασία δεδομένων.
- Η ευελιξία αφού υποστηρίζει πολλές πλατφόρμες όπως Linux, UNIX Windows και άλλες.
- Τα προγράμματα είναι ανοιχτού κώδικα.

2.3.1 Βασικά Στοιχεία

Εξηγούμε τα βασικά στοιχεία της MySQL στις παρακάτω υπο-ενότητες:

Βασική Δομή

Η MySQL αποθηκεύει τα δεδομένα με τον ίδιο τρόπο όπως εξηγήσαμε στην υπο-ενότητα 2.2.1 που μιλήσαμε για τις σχεσιακές βάσεις δεδομένων. Υποστηρίζει SQL εντολές για την δημιουργία και την διαχείριση τέτοιων βάσεων δεδομένων όπως για παράδειγμα η CREATE DATABASE που δημιουργεί μια βάση δεδομένων, η ALTER TABLE για την επεξεργασία της δομής ενός πίνακα και πολλές άλλες.

Γλώσσα Ερωτημάτων

Η γλώσσα για τη διαχείριση δεδομένων που χρησιμοποιεί είναι η SQL, την οποία Θα αναλύσουμε παρακάτω στην υπο-ενότητα 2.3.3.

Εισαγωγή και Εξαγωγή Δεδομένων

Η MySQL υποστηρίζει τόσο SQL εντολές από terminal (π.χ. mysqldump, LOAD DATA INFILE) όσο και ενέργειες σε γραφικό περιβάλλον (MySQL Workbench) για την εισαγωγή και την εξαγωγή δεδομένων της βάσης. Ανάλογα με τον τρόπο εξαγωγής, τα δεδομένα μπορούν να αποθηκευτούν σε αρχεία όπως CSV, JSON, HTML, TXT, SQL, XML και Excel XML.

Υποστήριξη Δοσοληψιών (Transaction Support)

Η MySQL υποστηρίζει δοσοληψίες με πλήρη ACID ιδιότητες. Ένα τέτοιο παράδειγμα φαίνεται στην Εικόνα 2.4. Η δοσοληψία ξεκινάει με την εντολή START TRANSACTION ή BEGIN και ολοκληρώνεται με το COMMIT το οποίο αποθηκεύει τις αλλαγές μόνιμα στη βάση. Επιπλέον, υπάρχει η εντολή ROLLBACK η οποία ακυρώνει τις αλλαγές που έγιναν σε μια δοσοληψία. Η MySQL επίσης, περιέχει μια μεταβλητή που ονομάζεται autocommit και η προεπιλεγμένη τιμή της είναι το 1. Αυτό σημαίνει ότι όταν εκτελεστεί ένα SQL ερώτημα, που τροποποιεί τη βάση, τότε οι αλλαγές αποθηκεύονται κατευθείαν στον δίσκο. Όμως, κατά τη διάρκεια μιας δοσοληψίας (BEGIN μέχρι COMMIT ή ROLLBACK) αυτή απενεργοποιείται, δηλαδή παίρνει την τιμή 0.

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

Εικόνα 2.4: Παράδειγμα δοσοληψίας στη MySQL

2.3.2 Μοντέλο Δεδομένων

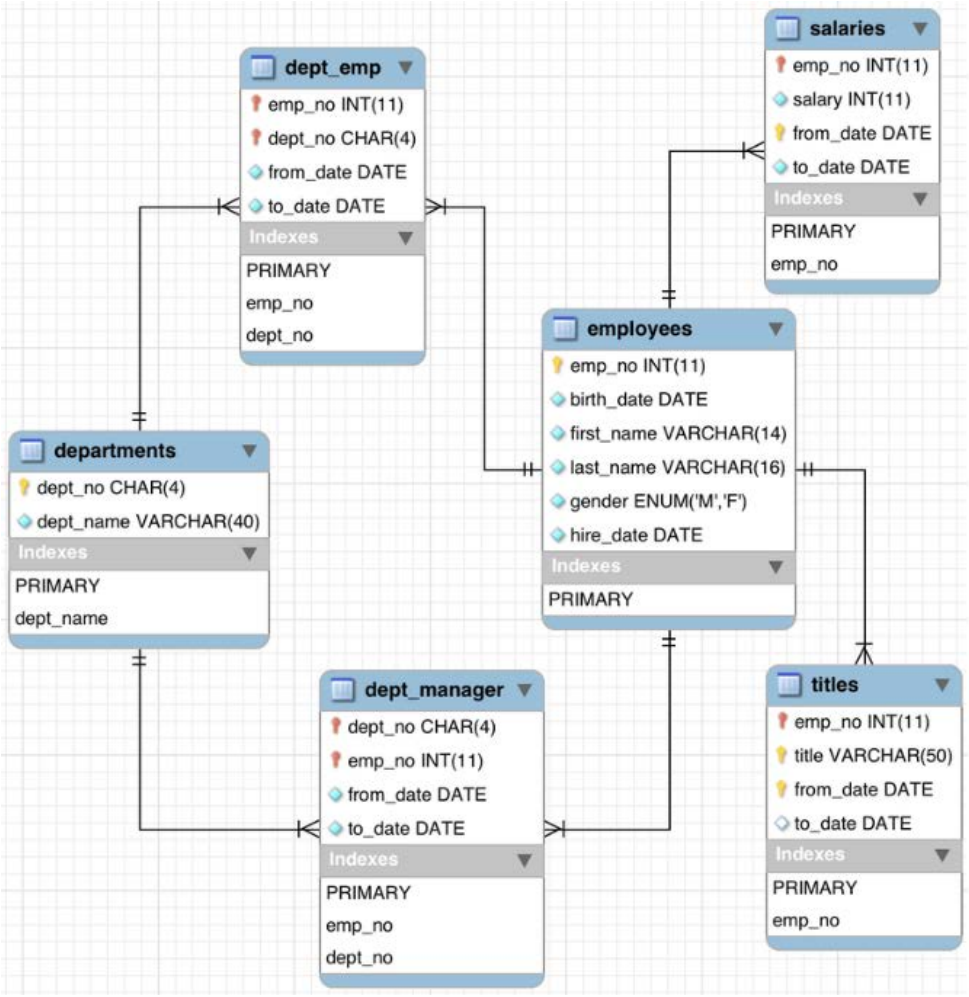
Εξηγούμε τα κύρια στοιχεία που σχετίζονται με το μοντέλο δεδομένων της MySQL στις παρακάτω υπο-ενότητες [11]:

Γενικά

Τα δεδομένα στην MySQL δεν έχουν αρκετά ευέλικτο σχήμα (schema). Αυτό διότι πρέπει να καθορίζουμε και να δηλώνουμε το σχήμα ενός πίνακα προτού εισάγουμε τα δεδομένα. Επιπλέον, όσον αφορά τους τύπους δεδομένων, η MySQL υποστηρίζει αρκετούς SQL τύπους δεδομένων σε διάφορες κατηγορίες: αριθμητικούς τύπους, τύπους ημερομηνίας και ώρας, αλφαριθμητικούς (string) τύπους, χωρικοί τύποι (spatial types) και ο τύπος δεδομένων JSON.

Παράδειγμα Σχήματος

Είναι καλή πρακτική πριν από κάθε υλοποίηση μιας βάσης δεδομένων να δημιουργείται ένα σχήμα αυτής (database schema). Ένα τέτοιο σχήμα μοιάζει θα πει κανείς με το μοντέλο Οντοτήτων-Συσχετίσεων αλλά φτιάχνεται με τέτοιο τρόπο όπου τόσο οι οντότητες όσο και οι συσχετίσεις μεταφράζονται σε σχέσεις (πίνακες), με αποτέλεσμα να βρισκόμαστε ένα βήμα πριν την δημιουργία μιας σχεσιακής βάσης δεδομένων. Ένα τέτοιο παράδειγμα φαίνεται στην Εικόνα 2.5 όπου έχει δημιουργηθεί με το γραφικό περιβάλλον της MySQL (MySQL Workbench) [13].



Εικόνα 2.5: Παράδειγμα σχήματος στην MySQL

2.3.3 Μοντέλο Ερωτημάτων

Εξηγούμε τα κύρια στοιχεία που σχετίζονται με το μοντέλο ερωτημάτων και τη δομή τους, της MySQL, στις παρακάτω υπο-ενότητες [11]:

Δομή Ερωτημάτων και λειτουργίες

Εδώ θα περιγράψουμε τις πιο βασικές SQL εντολές που χρησιμοποιεί η MySQL για να χειρίζεται τα δεδομένα σε μια βάση. Η ανάκτηση των δεδομένων από έναν ή περισσότερους πίνακες σε μια βάση επιτυγχάνεται με την εντολή SELECT. Η συγκεκριμένη εντολή έχει τόσες πολλές λειτουργίες που είναι περιττό να αναλυθούν στην παρούσα εργασία όπου δεν είναι αυτός ο σκοπός της. Ένα από τα πιο απλά παραδείγματα της εντολής SELECT φαίνεται στην Εικόνα 2.6 όπου και επιλέγονται όλες οι στήλες (χαρακτηριστικά) από έναν πίνακα.

```
SELECT * FROM tbl_name;
```

Εικόνα 2.6: Παράδειγμα εντολής SELECT στην SQL

Η εισαγωγή των δεδομένων πραγματοποιείται με την εντολή INSERT. Με αυτήν την εντολή μπορούμε είτε να εισάγουμε καινούργιες εγγραφές σε ένα πίνακα με δικές μας

τιμές (INSERT ... VALUES και INSERT ... SET) είτε να εισάγουμε εγγραφές που έχουν πρώτα επιλεχθεί με την εντολή SELECT (INSERT ... SELECT). Στο παράδειγμα της Εικόνας 2.7 γίνεται εισαγωγή τριών εγγραφών σε έναν πίνακα με χαρακτηριστικά a,b,c.

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

Εικόνα 2.7: Παράδειγμα εντολής INSERT στην SQL

Η ενημέρωση των δεδομένων γίνεται με την εντολή UPDATE. Αυτή η εντολή μπορεί να χρησιμοποιηθεί για να ενημερώσει μόνο έναν πίνακα ή και περισσότερους πίνακες ταυτόχρονα. Ένα απλό παράδειγμα φαίνεται στην Εικόνα 2.8 όπου και ενημερώνει τον πίνακα t1 αυξάνοντας κατά ένα το χαρακτηριστικό col1 σε όλες τις εγγραφές.

```
UPDATE t1 SET col1 = col1 + 1;
```

Εικόνα 2.8: Παράδειγμα εντολής UPDATE στην SQL

Τέλος, η διαγραφή των δεδομένων πραγματοποιείται με την εντολή DELETE. Η εντολή αυτή διαγράφει εγγραφές (γραμμές) από έναν ή περισσότερους πίνακες σύμφωνα με κάποια κριτήρια. Ένα παράδειγμα φαίνεται στην Εικόνα 2.9 όπου διαγράφει, από έναν πίνακα με όνομα somelog, τις εγγραφές που έχουν στο χαρακτηριστικό user την τιμή jcole.

```
DELETE FROM somelog WHERE user = 'jcole';
```

Εικόνα 2.9: Παράδειγμα εντολής DELETE στην SQL

Λειτουργίες Συνάθροισης

Οι συναθροιστικές λειτουργίες (aggregation operations) επεξεργάζονται τα δεδομένα και επιστρέφουν αποτελέσματα. Πιο συγκεκριμένα, η MySQL χρησιμοποιεί το GROUP BY στην SELECT εντολή το οποίο ομαδοποιεί τις πλειάδες που προκύπτουν από το τμήμα WHERE ως προς κάποιο σύνολο πεδίων και υπολογίζεται η συναθροιστική συνάρτηση για κάθε ομάδα. Επίσης, μπορούμε να έχουμε και συνθήκες στις ομάδες που δημιουργούνται από το group by, όπου εκφράζονται στο τμήμα HAVING της εντολής select. Υπάρχουν αρκετές αθροιστικές συναρτήσεις που μπορούν να εφαρμοστούν όπως μία για παράδειγμα είναι η AVG([DISTINCT] x) όπου επιστρέφει τον μέσο όρο [μοναδικών] τιμών ενός πεδίου x για κάθε ομάδα.

Υποστήριξη Δεικτών (Indexing)

Ο καλύτερος τρόπος για να βελτιώσουμε την απόδοση της εντολής SELECT είναι να δημιουργήσουμε δείκτες σε ένα ή περισσότερα πεδία (στήλες) που εξετάζονται κατά τη διάρκεια μιας εντολής. Αυτοί δεικτοδοτούν τις γραμμές ενός πίνακα, επιτρέποντας σε μια εντολή select να καθορίζει γρήγορα ποιες γραμμές ταιριάζουν με τη συνθήκη στο τμήμα WHERE, και να ανακτά τις τιμές των στηλών για αυτές τις γραμμές. Όλοι οι τύποι δεδομένων της MySQL μπορούν να δεικτοδοτηθούν. Χωρίς τους δείκτες, μια εντολή select πρέπει να ξεκινήσει από την πρώτη γραμμή και περάσει όλο τον πίνακα για να βρει τις γραμμές που θέλει να ανακτήσει. Σχεδόν όλοι οι δείκτες της MySQL αποθηκεύονται

σε B-trees. Τα είδη δεικτών που υποστηρίζει η MySQL είναι: clustered indexes, primary key, unique key, normal index και full text index. Αν και μπορεί να είναι δελεαστικό να δημιουργηθούν δείκτες για κάθε πιθανή στήλη που μπορεί να χρησιμοποιηθεί σε μια εντολή, περιττοί δείκτες σπαταλούν χώρο και χρόνο ώστε η MySQL να καθορίσει ποιους δείκτες να χρησιμοποιήσει. Επίσης, οι δείκτες προσθέτουν φόρτο εργασίας στις εισαγωγές, ενημερώσεις και διαγραφές επειδή κάθε δείκτης θα πρέπει να είναι ενημερωμένος. Για αυτό θα πρέπει να βρεθεί η σωστή ισορροπία για να επιτευχθούν γρήγορες εντολές χρησιμοποιώντας τα βέλτιστα σύνολα δεικτών.

Υποστήριξη Αναζήτησης Κειμένου

Στη MySQL υποστηρίζεται αναζήτηση κειμένου που εκτελείται με τη βοήθεια των full text δεικτών. Οι full text δείκτες μπορούν να χρησιμοποιηθούν μόνο στις μηχανές αποθήκευσης InnoDB και MyISAM και μόνο για τύπους δεδομένων χαρακτήρων όπως είναι: VARCHAR, CHAR και TEXT. Αφού δημιουργηθούν οι full text δείκτες τότε μπορεί να πραγματοποιηθεί η αναζήτηση κειμένου.

Ταξινόμηση

Η MySQL με το τμήμα ORDER BY μπορεί να προσφέρει λειτουργίες ταξινόμησης σε αύξουσα ή σε φθίνουσα σειρά. Για παράδειγμα, σε μια εντολή select αν καθοριστεί το τμήμα ORDER BY τότε τα αποτελέσματα που θέλει να ανακτήσει θα επιστραφούν σε ταξινομημένη σειρά ως προς τα πεδία που έχουν προσδιοριστεί. Ένα άλλο παράδειγμα είναι να καθοριστεί το τμήμα ORDER BY σε μια εντολή update με αποτέλεσμα να ενημερώνονται οι γραμμές με ταξινομημένη σειρά. Γενικά το τμήμα ORDER BY μπορεί να χρησιμοποιηθεί και σε άλλες εντολές.

2.3.4 Διαχείριση και Συντήρηση

Εξηγούμε τα κύρια στοιχεία που σχετίζονται με τη διαχείριση και τη συντήρηση της MySQL, στις παρακάτω υπο-ενότητες [11]:

Διαμόρφωση (Configuration)

Η MySQL παρέχει πολλούς τρόπους ρύθμισης του MySQL Server (mysqld), το οποίο είναι το κύριο πρόγραμμα που εκτελεί το μεγαλύτερο μέρος της εργασίας σε μια εγκατάσταση της MySQL. Ο MySQL Server περιέχει ένα σύνολο μεταβλητών συστήματος που επηρεάζουν τη λειτουργία του καθώς εκτελείται. Αυτές οι μεταβλητές μπορούν ρυθμιστούν κατά την εκκίνηση του server, και πολλές από αυτές μπορούν να αλλαχθούν κατά το χρόνο εκτέλεσης για δυναμική αναδιαμόρφωση του server. Επίσης, ο MySQL Server έχει και ένα σύνολο μεταβλητών κατάστασης που παρέχουν πληροφορίες σχετικά με τη λειτουργία του. Οι τρόποι που μπορούμε να ρυθμίσουμε τις μεταβλητές συστήματος και κατάστασης είναι μέσω αρχείων ρυθμίσεων ή μέσω εντολών σε command line ή και μέσω γραφικών διεπαφών όπως είναι το MySQL Workbench.

Μηχανές Αποθήκευσης (Storage Engines)

Οι μηχανές αποθήκευσης είναι σημαντικό κομμάτι μιας βάσης δεδομένων και είναι υπεύθυνες για το πώς αποθηκεύονται τα δεδομένα, τόσο στη μνήμη όσο και στο δίσκο. Οι μηχανές αποθήκευσης που υποστηρίζονται από την MySQL είναι οι εξής:

- InnoDB
- MyISAM
- Memory
- CSV
- Archive
- Blackhole
- NDB
- Merge
- Federated
- Example

Οι κύριες και πιο δημοφιλής μηχανές αποθήκευσης είναι η InnoDB και η MyISAM με την InnoDB να είναι και η προεπιλεγμένη από την έκδοση 5.7 της MySQL.

Υψηλή Διαθεσιμότητα και Επεκτασιμότητα (High Availability and Scalability)

Η MySQL έχει αναπτυχθεί σε πολλές εφαρμογές που απαιτούν διαθεσιμότητα και επεκτασιμότητα. Η διαθεσιμότητα αναφέρεται στην ικανότητα να αντιμετωπίζει, και εάν είναι απαραίτητο να ανακάμπτει από, αποτυχίες του server, συμπεριλαμβανομένων αποτυχίες της MySQL, του λειτουργικού συστήματος, ή του υλικού και η δραστηριότητα της συντήρησης για να μην υπάρξουν διακοπές στη λειτουργία της βάσης δεδομένων. Η επεκτασιμότητα αναφέρεται στην ικανότητα να μοιράζει τόσο τη βάση δεδομένων όσο και το φόρτο εργασίας από εφαρμογές μεταξύ πολλών διακομιστών MySQL. Οι κύριες λύσεις που χρησιμοποιεί η MySQL για θέματα διαθεσιμότητας και επεκτασιμότητας είναι:

- MySQL Replication.
- MySQL Fabric.
- MySQL Cluster.
- Oracle MySQL Cloud Service.
- Oracle Clusterware Agent for MySQL.
- MySQL with Solaris Cluster.

Ασφάλεια

Η ασφάλεια των βάσεων δεδομένων συνεπάγεται στο να επιτρέπονται ή να απαγορεύονται ενέργειες χρηστών στην βάση δεδομένων και στα αντικείμενα της. Κατά τη δημιουργία μιας βάσης δεδομένων και εφαρμογές που αλληλοεπιδρούν με αυτή, η ασφάλεια είναι από τα πιο βασικά κομμάτια που θα πρέπει να διαθέτουν. Η MySQL υποστηρίζει πολλές πολιτικές ασφάλειας τόσο στο επίπεδο της βάσης, όσο και στο επίπεδο προγραμμάτων και των μεταξύ τους συνδέσεων. Είναι σημαντικό οι πολιτικές ασφάλειας να μελετηθούν πλήρως πριν τη δημιουργία μιας MySQL βάσης δεδομένων.

Αντίγραφα Ασφαλείας (Backup)

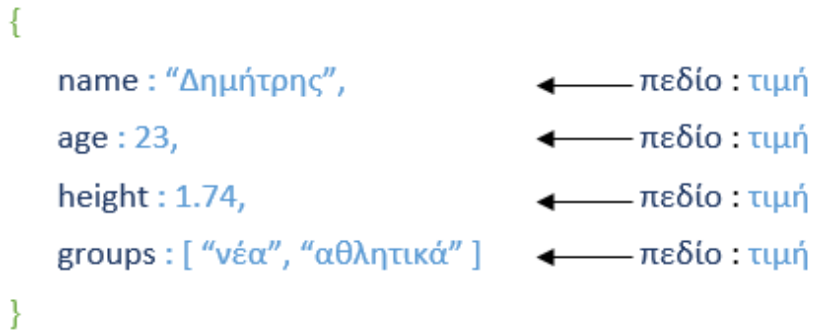
Είναι σημαντικό να δημιουργούνται αντίγραφα ασφαλείας μιας βάσης δεδομένων, ώστε να μην χάνονται τα δεδομένα σε περίπτωση σφαλμάτων όπως, να κολλήσει το σύστημα, αστοχίες στο υλικό, ή οι χρήστες να διαγράψουν δεδομένα κατά λάθος. Οι μέθοδοι που χρησιμοποιεί η MySQL για την δημιουργία αντιγράφων ασφαλείας είναι οι εξής:

- Hot Backup with MySQL Enterprise Backup.
- Backups with msqldump.
- Backups by Copying Table Files.

- Delimited-Text File Backups.
- Incremental Backups by Enabling the Binary Log.
- Backups Using Replication Slaves.
- Backups Using a File System Snapshot.

2.4 MongoDB Βάσεις Δεδομένων

Η MongoDB είναι ανοιχτού κώδικα NoSQL βάση δεδομένων και ανήκει στην κατηγορία των document βάσεων δεδομένων. Το αντίστοιχο της εγγραφής (record) των σχεσιακών βάσεων, ονομάζεται έγγραφο (document) στην MongoDB και περιέχει πεδία που απαρτίζονται από ζεύγη κλειδιών-τιμών (key-value pairs). Τα έγγραφα της MongoDB είναι παρόμοια με τα αντικείμενα της μορφής JSON και πιο συγκεκριμένα αποθηκεύονται στη μορφή Binary-Encoded JSON (BSON). Οι τιμές των πεδίων μπορούν να περιέχουν άλλα έγγραφα, πίνακες και πίνακες από έγγραφα. Το αντίστοιχο του πίνακα (table) των σχεσιακών βάσεων, ονομάζεται συλλογή (collection) στην MongoDB. Ένα παράδειγμα εγγράφου της MongoDB φαίνεται στην Εικόνα 2.10 παρακάτω [14]:



Εικόνα 2.10: Παράδειγμα εγγράφου της MongoDB

Τα πλεονεκτήματα της χρήσης εγγράφων είναι ότι:

- Αντιστοιχούν στους (έμφυτους -native) τύπους δεδομένων πολλών μοντέρνων γλωσσών προγραμματισμού (όπως η ιδιαίτερα δημοφιλής τελευταία JavaScript) διευκολύνοντας πολύ τον προγραμματισμό εφαρμογών.
- Η δυνατότητα για ενσωματωμένα έγγραφα και πίνακες μειώνει την ανάγκη για δαπανηρές πράξεις τύπου join.
- Η δυνατότητα χρήσης αδόμητου ή ημιδομημένου σχήματος υποστηρίζει την εύκολη επέκταση των εφαρμογών.

Η MongoDB παρέχει υψηλή απόδοση, υψηλή διαθεσιμότητα και αυτόματη κλιμάκωση και μερικά από τα χαρακτηριστικά της βάσης που τα συντελούν είναι:

- Υποστήριξη ενσωμάτωσης αντικειμένων στο μοντέλο δεδομένων της (nested documents), δυνατότητα η οποία μειώνει την ανάγκη για πολλαπλές αναγνώσεις/εγγραφές στους χώρους αποθήκευσης.
- Παρέχει δείκτες (indexes) οι οποίοι μπορούν να δεικτοδοτήσουν κλειδιά και σε ενσωματωμένα έγγραφα (documents).
- Η υπηρεσία λειτουργίας αντιγράφων της βάσης (replication) παρέχει αυτόματη ανάκαμψη από βλάβες (automatic failover) και πλεονασμό δεδομένων (data redundancy).
- Παρέχει οριζόντια κλιμάκωση ως βασική της υπηρεσία και παρέχει την δυνατότητα κατακερματισμού των δεδομένων (sharding) σε ένα σύνολο (cluster) υπολογιστών.

2.4.1 Βασικά Στοιχεία

Εξηγούμε τα βασικά στοιχεία της MongoDB στις παρακάτω υπο-ενότητες:

Βασική Δομή

Η MongoDB αποθηκεύει τα δεδομένα σε έγγραφα χρησιμοποιώντας την αναπαράσταση BSON στα δεδομένα και ομαδοποιεί τα έγγραφα μέσα σε συλλογές. Παρακάτω αναλύουμε τα έγγραφα, τις συλλογές και τη μορφή BSON [14]:

- **Έγγραφα (Documents)**: Η MongoDB αποθηκεύει τα δεδομένα σε έγγραφα τα οποία απαρτίζονται από ζεύγη κλειδιών-τιμών (key-value pairs). Τα έγγραφα μπορούν να φτάσουν σε μέγεθος μέχρι τα 16MB και το καθένα θα πρέπει να έχει ένα _id πεδίο το οποίο δεικτοδοτείται (indexed) αυτόματα. Το πεδίο _id παράγεται αυτόματα αν δεν έχει προσδιοριστεί και χρησιμοποιείται ως το μοναδικό αναγνωριστικό για το έγγραφο. Τα δεδομένα σε ένα έγγραφο αποθηκεύονται σε μια αναπαράσταση παρόμοιας της JavaScript Object Notation (JSON) που ονομάζεται BSON.

- **Συλλογές (Collections)**: Χρησιμοποιούνται για να ομαδοποιούν πολλά τα έγγραφα και είναι παρόμοια με τους πίνακες (tables) των RDBMS. Οι συλλογές μπορούν να περιέχουν έγγραφα με διαφορετική δομή ως προς το σχήμα (schema), ωστόσο είναι καλή πρακτική να αποθηκεύονται έγγραφα με την ίδια δομή.

- **BSON**: Η MongoDB χρησιμοποιεί τη μορφή BSON [15] για να αποθηκεύει τα έγγραφα. Η BSON είναι μια δυαδικά κωδικοποιημένη μορφή που επεκτείνει το γνωστό JSON.

Γλώσσα Ερωτημάτων

Η MongoDB υποστηρίζει τις CRUD λειτουργίες για να αλληλοεπιδρά με τα αποθηκευμένα δεδομένα. Αυτές οι λειτουργίες είναι η δημιουργία (create), η ανάγνωση (read), η ενημέρωση (update) και η διαγραφή (delete). Θα τις αναλύσουμε περαιτέρω στην υπο-ενότητα 2.4.3 [14].

Υποστήριξη Δοσοληψιών (Transaction Support)

Η MongoDB δεν υποστηρίζει δοσοληψίες σε πολλαπλά έγγραφα. Παρόλα αυτά, οι λειτουργίες εγγραφής είναι ατομικές (atomic) στο επίπεδο ενός εγγράφου. Επιπλέον, η τεχνική της απομόνωσης (isolation) υποστηρίζεται σε πολλαπλά έγγραφα χρησιμοποιώντας την επιλογή $isolated [14].

Εισαγωγή και Εξαγωγή Δεδομένων

Η MongoDB υποστηρίζει τη μέθοδο db.collection.bulkWrite() που μπορεί να χρησιμοποιηθεί για εκτελέσει ένα σύνολο από λειτουργίες εγγραφής ταυτόχρονα. Επιπλέον, υποστηρίζει και δύο χρήσιμα εργαλεία (mongoexport και mongoimport) που μπορούν να χρησιμοποιηθούν για να 'μεταφέρουν' τα δεδομένα ανάμεσα σε διαφορετικά MongoDB στιγμιότυπα σε μορφή Comma Separated Values (CSV) ή σε μορφή BSON [14].

2.4.2 Μοντέλο Δεδομένων

Εξηγούμε τα κύρια στοιχεία που σχετίζονται με το μοντέλο δεδομένων της MongoDB στις παρακάτω υπο-ενότητες [14]:

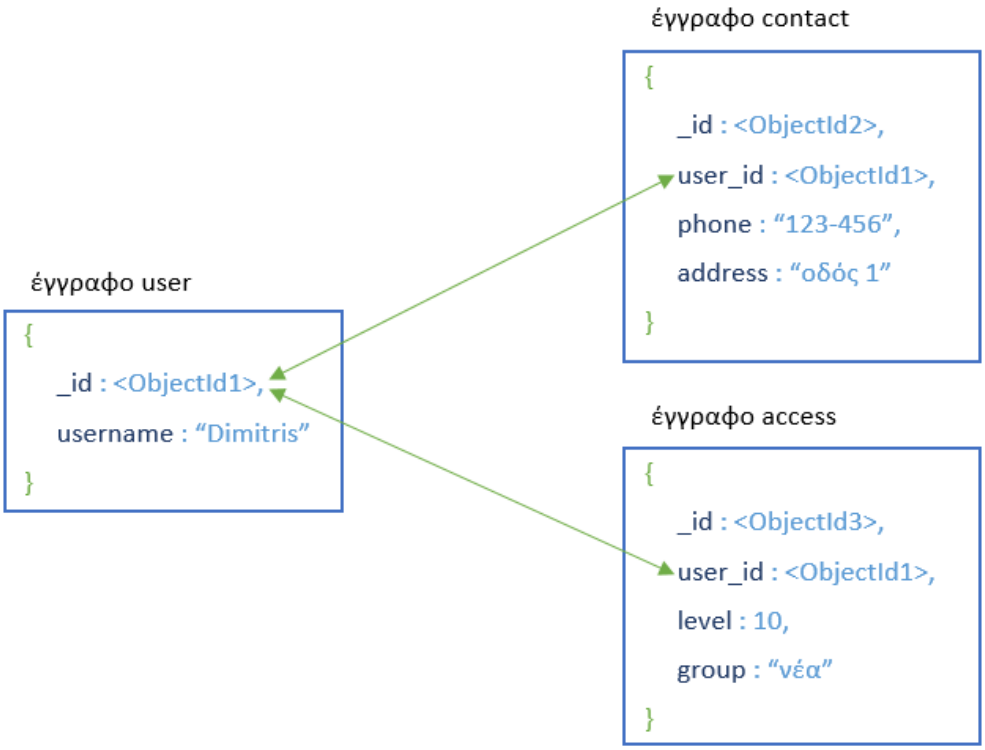
Γενικά

Τα δεδομένα στην MongoDB έχουν ένα ευέλικτο σχήμα (schema). Σε αντίθεση με τις SQL σχεσιακές βάσεις δεδομένων, όπου πρέπει να καθορίζουμε και να δηλώνουμε το σχήμα ενός πίνακα προτού εισάγουμε τα δεδομένα, οι συλλογές της MongoDB δεν προκαθορίζουν την δομή του εγγράφου. Αυτή η ευελιξία διευκολύνει τη χαρτογράφηση των εγγράφων σε μια οντότητα ή ένα αντικείμενο. Κάθε έγγραφο μπορεί να ταιριάξει τα πεδία δεδομένων της εκπροσωπούμενης οντότητας, ακόμη και αν τα δεδομένα είναι διαφορετικά. Στην πράξη, όμως, τα έγγραφα σε μια συλλογή μοιράζονται μια παρόμοια δομή.

Δομή Εγγράφου

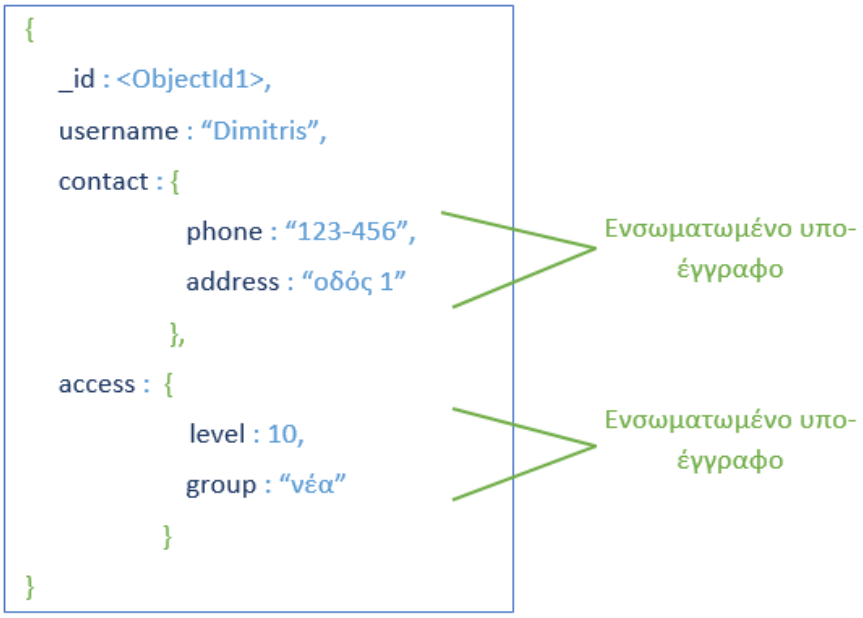
Η βασική απόφαση για το σχεδιασμό μοντέλο δεδομένων για MongoDB εφαρμογές περιστρέφεται γύρω από τη δομή των εγγράφων και πως η εφαρμογή παριστάνει τις σχέσεις μεταξύ των δεδομένων. Οι δύο τεχνικές που επιτρέπουν στις εφαρμογές να παριστάνουν αυτές τις σχέσεις είναι τα έγγραφα με αναφορές (references documents) και τα ενσωματωμένα έγγραφα (embedded documents).

Οι αναφορές ‘αποθηκεύουν’ τις σχέσεις των δεδομένων με το να περιλαμβάνουν συνδέσμους (links) ή αναφορές (references) από το ένα έγγραφο στο άλλο. Οι εφαρμογές μπορούν να αξιοποιήσουν αυτές τις αναφορές για να έχουν πρόσβαση σε δεδομένα που αιτούνται. Γενικά, αυτά ονομάζονται κανονικοποιημένα μοντέλα δεδομένων (normalized data models). Ένα παράδειγμα φαίνεται στην Εικόνα 2.11.



Εικόνα 2.11: Παράδειγμα κανονικοποιημένου (normalized) μοντέλου δεδομένων στην MongoDB

Η τεχνική των ενσωματωμένων εγγράφων 'αποθηκεύουν' τις σχέσεις των δεδομένων με το να αποθηκεύουν τα σχετικά δεδομένα σε ένα και μόνο έγγραφο. Τα έγγραφα της MongoDB επιτρέπουν να ενσωματώνουν δομές εγγράφων σε ένα πεδίο (field) ή σε πίνακα (array) τέτοιων εγγράφων. Αυτά τα μη κανονικοποιημένα μοντέλα δεδομένων (denormalized data models) όπως ονομάζονται, επιτρέπουν στις εφαρμογές να αλληλοεπιδρούν με τα δεδομένα με λιγότερες λειτουργίες από τα κανονικοποιημένα μοντέλα. Ένα παράδειγμα τέτοιου μοντέλου φαίνεται στην Εικόνα 2.12.



Εικόνα 2.12: Παράδειγμα μη κανονικοποιημένου (denormalized) μοντέλου δεδομένων στην MongoDB

Υποστήριξη Σχεσιακών Δεδομένων

Οι σχέσεις των δεδομένων μπορούν να μοντελοποιηθούν χρησιμοποιώντας τις κανονικοποιημένες και μη κανονικοποιημένες προσεγγίσεις. Στις σχέσεις One-to-One είναι προτιμότερο να χρησιμοποιηθεί η μη κανονικοποιημένη προσέγγιση με την ενσωμάτωση εγγράφων για καλύτερη απόδοση. Στις One-to-Many σχέσεις μπορούν να μοντελοποιηθούν χρησιμοποιώντας την μη κανονικοποιημένη προσέγγιση εάν αυτό δεν οδηγεί τα έγγραφα σε ένα απροσδιορίστως μεγάλο μέγεθος το οποίο μπορεί να προκαλέσει θέματα απόδοσης. Τέλος, οι σχέσεις Many-to-Many είναι καλή πρακτική να χρησιμοποιείται κανονικοποιημένη προσέγγιση για να αποτρέψει πλεονάζοντα δεδομένα που μπορούν να οδηγήσουν σε τεράστια έγγραφα.

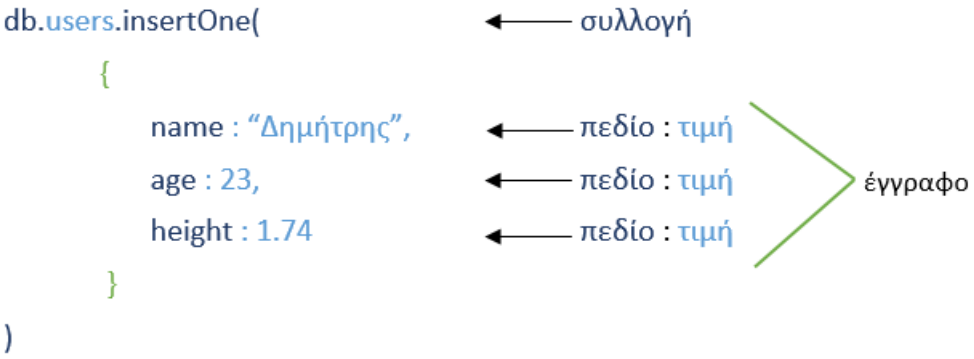
2.4.3 Μοντέλο Ερωτημάτων

Εξηγούμε τα κύρια στοιχεία που σχετίζονται με το μοντέλο ερωτημάτων και τη δομή τους, της MongoDB, στις παρακάτω υπο-ενότητες [14]:

Δομή Ερωτημάτων και λειτουργίες

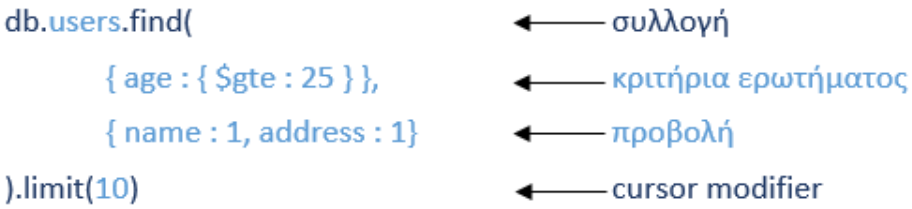
Όπως προαναφέραμε η MongoDB υποστηρίζει τις CRUD λειτουργίες (create, read, update, and delete) για να αλληλοεπιδρά με τα αποθηκευμένα δεδομένα. Η δημιουργία ή η εισαγωγή ενός εγγράφου σε μια συλλογή πραγματοποιείται με τις εντολές *db.collection.insertOne()* για ένα έγγραφο και *db.collection.insertMany()* για πολλαπλά

έγγραφα. Αν η συλλογή δεν υπάρχει τότε δημιουργείται αυτόματα κατά την εισαγωγή. Ένα παράδειγμα φαίνεται στην Εικόνα 2.13.



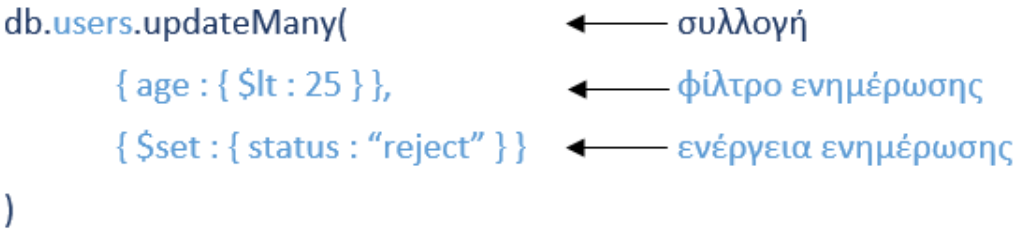
Εικόνα 2.13: Εντολή εισαγωγής ενός εγγράφου στην MongoDB

Η ανάγνωση και ανάκτηση εγγράφων από μια συλλογή γίνεται με την εντολή *db.collection.find()*. Σε αυτήν την εντολή μπορούμε να ορίσουμε φίλτρα και κριτήρια ώστε να προσδιορίσουμε τα έγγραφα που επιθυμούμε να επιστραφούν. Ένα παράδειγμα φαίνεται στην Εικόνα 2.14.



Εικόνα 2.14: Εντολή ανάγνωσης στην MongoDB

Οι λειτουργίες ενημέρωσης τροποποιούν τα υπάρχοντα έγγραφα σε μια συλλογή. Οι εντολές που πραγματοποιούν ενημερώσεις είναι: db.collection.updateOne(), db.collection.updateMany(), db.collection.replaceOne(). Φίλτρα και κριτήρια μπορούν να χρησιμοποιηθούν όπως ακριβώς στις εντολές ανάγνωσης. Ένα παράδειγμα φαίνεται στην Εικόνα 2.15.



Εικόνα 2.15: Εντολή ενημέρωσης στην MongoDB

Τέλος, έχουμε τις εντολές για διαγραφή εγγράφων από μια συλλογή όπου και εδώ χρησιμοποιούνται τα φίλτρα με τον ίδιο τρόπο. Οι εντολές αυτές είναι η db.collection.deleteOne() και η db.collection.deleteMany(). Ένα παράδειγμα φαίνεται στην Εικόνα 2.16.

Εικόνα 2.16: Εντολή διαγραφής στην MongoDB

Λειτουργίες Συνάθροισης

Οι συναθροιστικές λειτουργίες (aggregation operations) επεξεργάζονται τα δεδομένα και επιστρέφουν αποτελέσματα. Πιο συγκεκριμένα ομαδοποιούν τιμές από πολλαπλά έγγραφα και μετά από κάποια στάδια επεξεργασίας επιστρέφουν ένα μοναδικό αποτέλεσμα. Η MongoDB υποστηρίζει τρία είδη συναθροίσεων: το aggregation pipeline, την συνάρτηση map-reduce, και τις μεθόδους single purpose aggregation. Στο Pipeline Aggregation Framework, τα δεδομένα περνάνε από πολλά στάδια προτού επιστραφούν τα αποτελέσματα. Αρκετά στάδια υποστηρίζονται όπως, $project, $match, $group, $sort και το στάδιο $sample. Η συνάθροιση επιτυγχάνεται με την εντολή db.collection.aggregate() που λειτουργεί σε επίπεδο συλλογής. Γενικά, είναι προτιμότερο να χρησιμοποιείται το pipeline aggregation framework αφού είναι πιο απλό και πιο αποδοτικό από ότι το map-reduce. Παρόλα αυτά, το map-reduce είναι πιο ευέλικτο αφού χρησιμοποιεί JavaScript συναρτήσεις.

Υποστήριξη Δεικτών (Indexing)

Με τους δείκτες στην MongoDB δημιουργείται μια μορφή ευρετηρίου χρησιμοποιώντας την αποδοτική δομή δεδομένων B-tree για να αποθηκεύσουν ορισμένες πληροφορίες προκειμένου να επιταχύνουν τα ερωτήματα ελαχιστοποιώντας τον αριθμό των εγγράφων προς αναζήτηση. Η MongoDB υποστηρίζει πολλά είδη δεικτών που ορίζονται σε επίπεδο συλλογής και μπορούν να εφαρμοστούν σε οποιοδήποτε πεδίο ή υπο-πεδίο τα οποία είναι: single field, compound index, multikey index, geospatial index, text indexes και hashed indexes. Οι δείκτες μπορούν να έχουν αρκετές ιδιότητες που είναι: unique indexes, partial indexes, sparse indexes και TTL indexes.

Υποστήριξη Αναζήτησης Κειμένου

Η MongoDB υποστηρίζει πλήρης αναζήτηση σε κείμενο χρησιμοποιώντας τους δείκτες κειμένου (text indexes). Οι δείκτες κειμένου μπορούν να οριστούν σε οποιαδήποτε συμβολοσειρά που απαιτείται να αναζητηθεί πλήρως μέσα σε ένα έγγραφο. Καθορίζοντας έναν δείκτη κειμένου σε ένα πεδίο, η MongoDB κάνει tokenize το κείμενο του πεδίου αυτού και δημιουργεί το ευρετήριο αναλόγως. Μπορεί να δημιουργηθεί μόνο ένας δείκτης κειμένου ανά συλλογή, αλλά αυτός ο δείκτης μπορεί να καλύπτει πολλαπλά πεδία.

Ταξινόμηση

Η MongoDB υποστηρίζει μια μέθοδο ταξινόμησης που επιτρέπει στα δεδομένα να ταξινομηθούν σε αύξουσα ή σε φθίνουσα σειρά. Συνιστάται να δημιουργούνται δείκτες στα πεδία που χρησιμοποιούνται για ταξινόμηση.

2.4.4 Διαχείριση και Συντήρηση

Εξηγούμε τα κύρια στοιχεία που σχετίζονται με τη διαχείριση και τη συντήρηση της MongoDB, στις παρακάτω υπο-ενότητες [14]:

Διαμόρφωση (Configuration)

Η MongoDB χρησιμοποιεί ένα αρχείο ρυθμίσεων σε μορφή YAML για κάθε mongod ή στιγμιότυπο mongos. Αυτό το αρχείο περιέχει όλες τις ρυθμίσεις και τις εντολές που θα χρησιμοποιηθούν από ένα στιγμιότυπο. Σε περίπτωση αλλαγής του αρχείου θα χρειαστεί επανεκκίνηση του στιγμιότυπου για να πάρει τις αλλαγές.

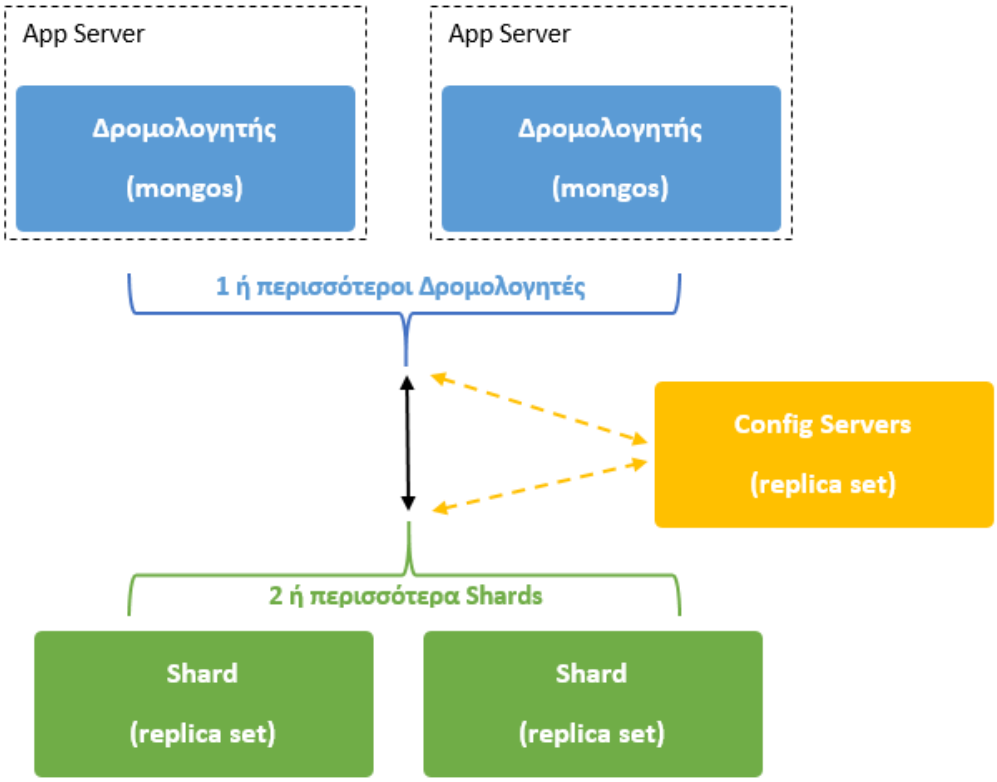
Μηχανές Αποθήκευσης (Storage Engines)

Η MongoDB υποστηρίζει τρεις μηχανές αποθήκευσης που είναι οι WiredTiger, MMAPv1 και In-Memory Storage Engine. Είναι πολύ σημαντικό να επιλεχθεί η κατάλληλη μηχανή αποθήκευσης αφού μπορεί να επηρεάσει αρκετά την απόδοση των εφαρμογών. Η μηχανή αποθήκευσης WiredTiger είναι η προεπιλεγμένη για τις εκδόσεις MongoDB 3.2 και μετά ενώ η MMAPv1 ήταν η προεπιλεγμένη για παλαιότερες εκδόσεις.

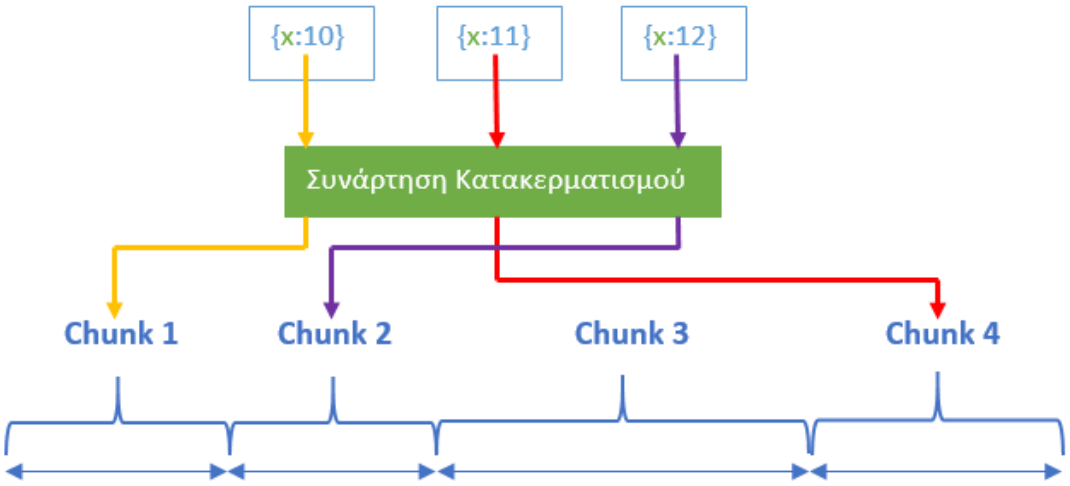
Επεκτασιμότητα

Η MongoDB υποστηρίζει τη μέθοδο sharding χρησιμοποιώντας ένα Sharded Cluster, το οποίο αποτελείται από: τους δρομολογητές ερωτημάτων, τα configuration servers και τα shard όπως φαίνεται στην Εικόνα 2.17. Οι δρομολογητές ερωτημάτων είναι στιγμιότυπα mongos που χρησιμοποιούνται από τις εφαρμογές του πελάτη για να αλληλοεπιδρά με τα sharded δεδομένα. Τα στιγμιότυπα mongos είναι υπεύθυνα για την δρομολόγηση των αιτημάτων ανάγνωσης και εγγραφής στα αντίστοιχα shards με αποτέλεσμα οι εφαρμογές του πελάτη να μην έχουν πρόσβαση απευθείας σε αυτά τα δεδομένα παρά μόνον μέσω των στιγμιότυπων mongos. Οι configuration servers αποθηκεύουν μεταδεδομένα (metadata) και ρυθμίσεις διαμόρφωσης (configuration settings) για το Sharded Cluster. Σε περίπτωση που αυτοί οι εξυπηρετητές σταματήσουν, λόγω βλάβης για παράδειγμα, τότε όλο το Sharded Cluster δεν λειτουργεί. Για αυτόν τον λόγο από την έκδοση MongoDB 3.4 είναι υποχρεωτικό αυτοί οι εξυπηρετητές να αναπτυχθούν σε ένα replica set (CSRS). Έτσι δημιουργούνται πολλοί τέτοιοι εξυπηρετητές με αποτέλεσμα να εξασφαλίζεται η διαθεσιμότητα.

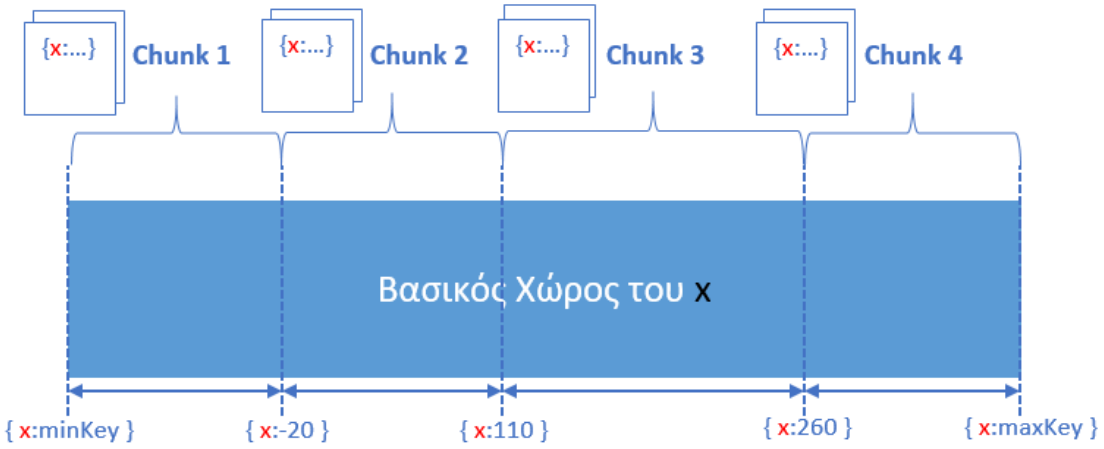
Κάθε shard περιέχει ένα υποσύνολο των sharded δεδομένων και μπορεί να αναπτυχθεί και αυτό σαν ένα replica set για την εξασφάλιση της διαθεσιμότητας και την παροχή μηχανισμών αποκατάστασης σε τυχόν καταστροφές. Η MongoDB διαμερίζει και διανέμει τα δεδομένα ομοιόμορφα με βάση ενός sharded κλειδιού και χρησιμοποιεί δύο στρατηγικές για την κατανομή αυτή, που είναι η ranged και η hashed sharding στρατηγική. Στην ranged στρατηγική τα δεδομένα χωρίζονται σε διαστήματα με βάση το sharded κλειδί όπως φαίνεται και στην Εικόνα 2.18. Αυτό παρέχει καλύτερη απόδοση σε ερωτήματα διαστημάτων (ranged queries) αλλά άνιση κατανομή των δεδομένων. Από την άλλη μεριά, έχουμε τη στρατηγική hashed η οποία χωρίζει τα δεδομένα με βάση μιας συνάρτησης κατακερματισμού πάνω στο sharded κλειδί, όπως φαίνεται στην Εικόνα 2.19. Αυτή παρέχει μια πιο ίση κατανομή των δεδομένων αλλά δεν τόσο αποδοτική σε ερωτήματα διαστημάτων.



Εικόνα 2.17: MongoDB Sharded Cluster



Εικόνα 2.18: MongoDB Hash Based Sharding



Εικόνα 2.19: MongoDB Range Based Sharding

Διαθεσιμότητα (availability)

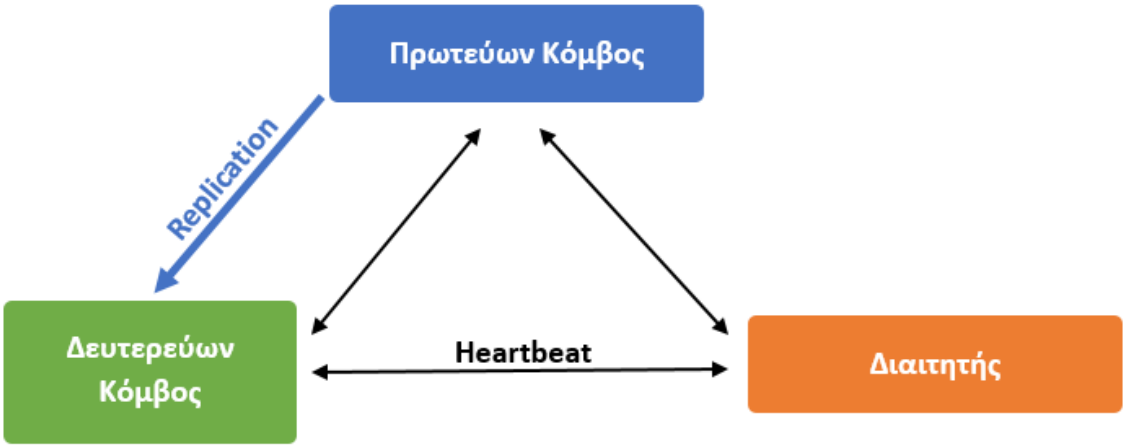
Η MongoDB υποστηρίζει ομοιοτυπία δεδομένων μέσω της αντιγραφής αυτών (data replication) που είναι ένα replica set το οποίο είναι μια ομάδα από mongod στιγμιότυπων που χρησιμοποιούν το ίδιο σύνολο δεδομένων. Κάθε replica set έχει έναν μόνο κύριο κόμβο και οι υπόλοιποι είναι δευτερεύοντες κόμβοι όπως φαίνεται στην Εικόνα 2.20. Επίσης, υποστηρίζεται και ένας ειδικός τύπος κόμβου που ονομάζεται διαιτητής (arbiter) ο οποίος χρησιμοποιείται όταν ο αριθμός των κόμβων στο replica set είναι άρτιος όπως φαίνεται στην Εικόνα 2.21. Ο κόμβος διαιτητής μπορεί να χρησιμοποιηθεί κατά τη διαδικασία της ψηφοφορίας κατά την οποία επιλέγεται ένας δευτερεύων κόμβος για να γίνει κύριος κόμβος.

Ο κύριος κόμβος είναι υπεύθυνος για όλες τις λειτουργίες εγγραφής, και καταγράφει όλες τις δραστηριότητες του σε ένα αρχείο καταγραφής δοσοληψιών που ονομάζεται oplog. Το oplog στη συνέχεια χρησιμοποιείται από τους δευτερεύοντες κόμβους για να αντιγράψουν (replicate) τα δεδομένα που έχει ο κύριος κόμβος. Οι λειτουργίες ανάγνωσης γίνονται στον κύριο κόμβο αλλά οι εφαρμογές μπορούν να επιλέξουν ρητά να διαβάσουν δεδομένα από τους δευτερεύοντες κόμβους. Ωστόσο, δεν συνίσταται καθώς τα δεδομένα στους δευτερεύοντες κόμβους μπορεί να είναι ασυνεπή. Για να κλιμακωθούν οι λειτουργίες ανάγνωσης η τεχνική του sharding μπορεί να χρησιμοποιηθεί αφού η μέθοδος της αντιγραφής (replication) στην MongoDB χρησιμοποιείται κυρίως για να προσφέρει διαθεσιμότητα.

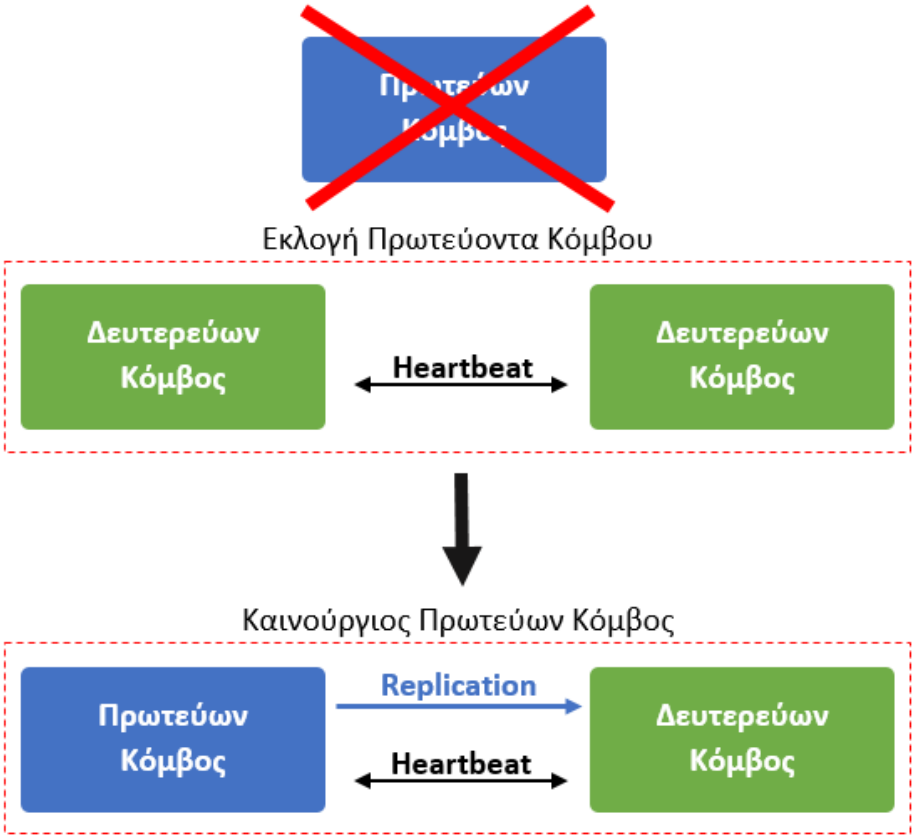
Όταν ένας καινούργιος δευτερεύων κόμβος δημιουργείται κάνει μια αρχικοποίηση κλωνοποιώντας τις βάσεις δεδομένων και χτίζοντας τους δείκτες σε όλες τις συλλογές. Μετά από αυτό, χρησιμοποιώντας το αρχείο oplog θα αντιγράψουν ασύγχρονα τα δεδομένα που βρίσκονται στον κύριο κόμβο. Επιπλέον, μπορεί να ρυθμιστεί ένας δευτερεύων κόμβος ώστε να αποθηκεύει παλαιότερα στιγμιότυπα των δεδομένων του κύριου κόμβου που είναι χρήσιμο για επαναφορά των δεδομένων. Τέλος, ένας δευτερεύων κόμβος μπορεί να ρυθμιστεί ώστε να μην συμμετέχει στην διαδικασία της ψηφοφορίας για εκλογή κύριου κόμβου με αποτέλεσμα να μην γίνεται ποτέ κύριος κόμβος.

Υποστηρίζεται επίσης μια διαδικασία αυτόματης ανακατεύθυνσης, που λαμβάνει χώρα όταν ο κύριος κόμβος δεν είναι προσβάσιμος για κάποιους λόγους όπως φαίνεται και στην Εικόνα 2.22. Αν οι άλλοι κόμβοι δεν μπορούν να επικοινωνήσουν με τον κύριο κόμβο τότε μέσω της ψηφοφορίας θα εκλεχθεί καινούργιος κύριος κόμβος.

Εικόνα 2.20: MongoDB Replica Set



Εικόνα 2.21: MongoDB Replica Set Arbiter



Εικόνα 2.22: MongoDB Replica Set Fail-Over Process

Ανθεκτικότητα σε Περίπτωση Βλάβης

Η MongoDB εκτός από τους κλασικό μηχανισμό που έχουν οι μηχανές αποθήκευσης, δηλαδή να αποθηκεύει περιοδικά τα δεδομένα από τη μνήμη στο δίσκο, υποστηρίζει και έναν μηχανισμό που λέγεται journaling. Αυτός ο μηχανισμός χρησιμοποιεί τα λεγόμενα journal αρχεία για να αποθηκεύει τις αλλαγές που βρίσκονται στη μνήμη RAM που μπορούν αργότερα να χρησιμοποιηθούν για ανάκτηση των δεδομένων σε περίπτωση που ο server πάθει βλάβη προτού αποθηκεύσει τα δεδομένα στο δίσκο. Οι λειτουργίες εγγραφής θεωρούνται ανθεκτικές μόνο όταν ο μηχανισμός journaling είναι ενεργοποιημένος.

Ασφάλεια

Η MongoDB προσφέρει πολλές επιλογές για ασφάλεια όπως η αυθεντικοποίηση, ο έλεγχος πρόσβασης και η κρυπτογράφηση της επικοινωνίας. Η αυθεντικοποίηση χρησιμοποιείται για την επαλήθευση των πελατών που προσπαθούν να συνδεθούν με τη βάση δεδομένων. Ο έλεγχος πρόσβασης χρησιμοποιείται, όπως λέει και το όνομα, για τον έλεγχο της πρόσβασης σε ορισμένους πόρους, χορηγώντας διάφορα δικαιώματα στους χρήστες. Ένας τέτοιος πόρος μπορεί να είναι μια βάση δεδομένων, μια συλλογή είτε μια ομάδα από αυτές. Τέλος, υποστηρίζεται κρυπτογράφηση των συνδέσεων των mongod ή των στιγμιότυπων mongos.

Αντίγραφα Ασφαλείας (Backup)

Υπάρχουν 4 τεχνικές στην MongoDB που μπορούμε να δημιουργήσουμε αντίγραφα ασφαλείας.

- **Back Up with Atlas**: Είναι μια βάση δεδομένων ως απομακρυσμένη υπηρεσία (cloud) που προσφέρει πλήρη διαχείριση για τα αντίγραφα ασφαλείας.

- **Back Up with MongoDB Cloud Manager or Ops Manager**: Είναι μια πλατφόρμα για δημιουργία αντιγράφων ασφαλείας που αποθηκεύεται τοπικά και παρέχει GUI για μεγαλύτερη ευελιξία.

- **Back Up with mongodump**: Χρησιμοποιεί ένα εργαλείο που ονομάζεται mongodump το οποίο διαβάζει τα δεδομένα από τη βάση δεδομένων και δημιουργεί BSON αρχεία ως αντίγραφα ασφαλείας.

- **Back Up by Copying Underlying Data Files**

3. BENCHMARKS

Στο κεφάλαιο αυτό παρουσιάζεται το μετροπρόγραμμα TPC-H και τα στοιχεία που χρησιμοποιήθηκαν από αυτό, όπως είναι το σχήμα της βάσης δεδομένων (database schema) και τα ερωτήματα (queries) τόσο στην MySQL όσο και στη MongoDB. Στη συνέχεια, δίνονται τυχόν ρυθμίσεις που πραγματοποιήθηκαν στις δυο βάσεις δεδομένων πριν τις πειραματικές μετρήσεις και τέλος εξηγούνται οι μετρικές που χρησιμοποιήθηκαν για μέτρηση της απόδοσης ώστε να επιτευχθεί η σύγκριση αυτών.

3.1 The TPC-H Benchmark

Το TPC-H είναι ένα μετροπρόγραμμα υποστήριξης λήψης αποφάσεων που παρέχεται από μια μη κερδοσκοπική εταιρεία που ονομάζεται TPC για τη μέτρηση της απόδοσης διάφορων σχεσιακώς βάσεων δεδομένων. Το μετροπρόγραμμα αυτό αποτελείται από ένα σύνολο ad-hoc ερωτημάτων (queries) προσανατολισμένα στις επιχειρήσεις που επιλέχθηκαν έτσι ώστε να είναι σχετικά με την ευρεία έννοια του όρου, βιομηχανία. Τα ερωτήματα προορίζονται να δώσουν ένα ρεαλιστικό πλαίσιο για ένα σύστημα χονδρικής πώλησης και έχουν υψηλό βαθμό πολυπλοκότητας [16].

Σε αυτή την έρευνα, το TPC-H μοντέλο δεδομένων και μερικά από τα ερωτήματα χρησιμοποιήθηκαν έτσι ώστε να μετρηθούν οι αποδόσεις των MySQL και MongoDB βάσεων δεδομένων αφού παρέχει ένα μοντέλο δεδομένων της πραγματικής ζωής για μια εφαρμογή **B2C***. Αυτές οι εφαρμογές είναι σημαντικές όχι μόνο στο ότι χρησιμοποιούνται κατά κόρον στη σημερινή εποχή αλλά επειδή είναι και από τις πρώτες που αντιμετωπίζουν προβλήματα επεκτασιμότητας λόγω της ραγδαίας αύξησης των χρηστών του διαδικτύου και συνεπώς της αύξησης του όγκου των δεδομένων που αποθηκεύουν και διαχειρίζονται. Το TPC-H παρέχει επίσης σύνθετα ερωτήματα (complex queries) που απεικονίζουν τη δραστηριότητα επιχειρηματικών αναλύσεων σε B2C εφαρμογές συμπεριλαμβανομένων ερωτημάτων που σχετίζονται με την τιμολόγηση, την προώθηση, το κέρδος, την ικανοποίηση πελατών και άλλα πολλά.

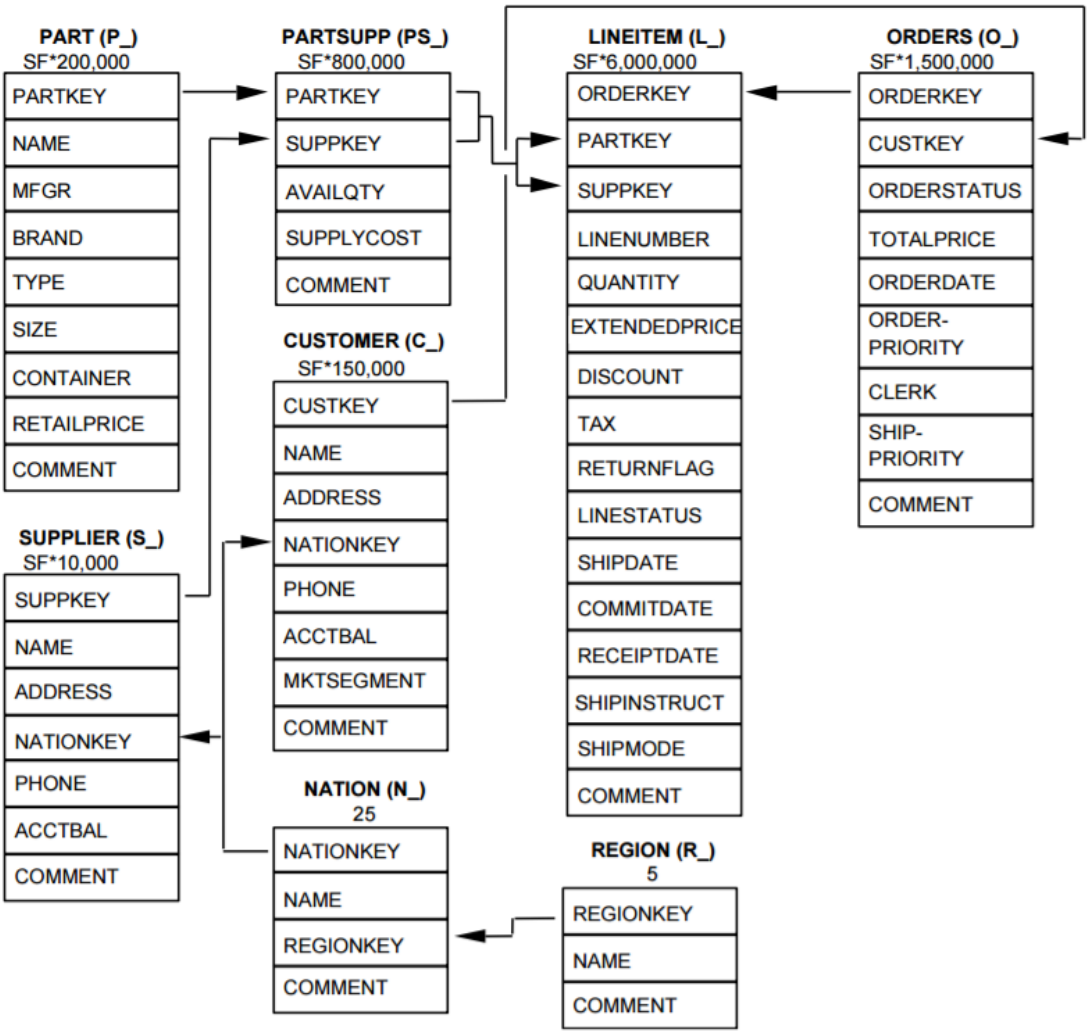
B2C (Business2Consumer or Business-to-Consumer): Business that sells products or provides services to end-user consumers.

3.2 Σχήμα Βάσης Δεδομένων

Το σχήμα που χρησιμοποιήθηκε σε αυτή την εργασία είναι αυτό του μετροπρογράμματος TPC-H. Η MySQL όπως φαίνεται στην υπο-ενότητα 3.2.1 έχει την ίδια ακριβώς δομή με αυτή του TPC-H αφού το ίδιο δημιουργήθηκε για τη μέτρηση των σχεσιακών βάσεων δεδομένων. Η MongoDB, όπως φαίνεται στην υπο-ενότητα 3.2.2, ακολουθεί τη μη κανονικοποιημένη προσέγγιση αφού υπάρχουν μόνο One-to-Many σχέσεις και τα έγγραφα δεν οδηγούνται σε απροσδιόριστα μεγάλα μεγέθη, λόγω της φύσης του μετροπρογράμματος TPC-H που χρησιμοποιεί μια μεταβλητή **SF** (Scale Factor) για να καθορίζει από την αρχή το μέγεθος των δεδομένων.

3.2.1 MySQL σχήμα

Το σχήμα της MySQL φαίνεται στην Εικόνα 3.1 και παρακάτω παρουσιάζονται τα χαρακτηριστικά του κάθε πίνακα [16].



Εικόνα 3.1: The TPC-H Schema

Πίνακας 3.1: PART Table Layout

Column Name	Datatype	Requirements Comment
P_PARTKEY	identifier	SF*200,000 are populated
P_NAME	variable text, size 55	
P_MFGR	fixed text, size 25	
P_BRAND	fixed text, size 10	
P_TYPE	variable text, size 25	
P_SIZE	integer	
P_CONTAINER	fixed text, size 10	
P_RETAILPRICE	decimal	
P_COMMENT	variable text, size 23	
Primary Key: P_PARTKEY		

Πίνακας 3.2: SUPPLIER Table Layout

Column Name	Datatype	Requirements Comment
S_SUPPKEY	identifier	SF*10,000 are populated
S_NAME	fixed text, size 25	
S_ADDRESS	variable text, size 40	
S_NATIONKEY	Identifier	Foreign Key to N_NATIONKEY
S_PHONE	fixed text, size 15	
S_ACCTBAL	decimal	
S_COMMENT	variable text, size 101	
Primary Key: S_SUPPKEY		

Πίνακας 3.3: PARTSUPP Table Layout

Column Name	Datatype	Requirements Comment
PS_PARTKEY	Identifier	Foreign Key to P_PARTKEY
PS_SUPPKEY	Identifier	Foreign Key to S_SUPPKEY
PS_AVAILQTY	integer	
PS_SUPPLYCOST	Decimal	
PS_COMMENT	variable text, size 199	
Primary Key: PS_PARTKEY, PS_SUPPKEY		

Πίνακας 3.4: CUSTOMER Table Layout

Column Name	Datatype	Requirements Comment
C_CUSTKEY	Identifier	SF*150,000 are populated
C_NAME	variable text, size 25	
C_ADDRESS	variable text, size 40	
C_NATIONKEY	Identifier	Foreign Key to N_NATIONKEY
C_PHONE	fixed text, size 15	
C_ACCTBAL	Decimal	
C_MKTSEGMENT	fixed text, size 10	
C_COMMENT	variable text, size 117	
Primary Key: C_CUSTKEY		

Πίνακας 3.5: ORDERS Table Layout

Column Name	Datatype	Requirements Comment
O_ORDERKEY	Identifier	SF*1,500,000 are sparsely populated
O_CUSTKEY	Identifier	Foreign Key to C_CUSTKEY
O_ORDERSTATUS	fixed text, size 1	
O_TOTALPRICE	Decimal	
O_ORDERDATE	Date	
O_ORDERPRIORITY	fixed text, size 15	
O_CLERK	fixed text, size 15	
O_SHIPPRIORITY	Integer	
O_COMMENT	variable text, size 79	
Primary Key: O_ORDERKEY		

Πίνακας 3.6: LINEITEM Table Layout

Column Name	Datatype	Requirements Comment
L_ORDERKEY	identifier	Foreign Key to O_ORDERKEY
L_PARTKEY	identifier	Foreign key to P_PARTKEY, first part of the compound Foreign Key to (PS_PARTKEY, PS_SUPPKEY) with L_SUPPKEY
L_SUPPKEY	Identifier	Foreign key to S_SUPPKEY, second part the compound Foreign Key to (PS_PARTKEY, PS_SUPPKEY) with L_PARTKEY
L_LINENUMBER	integer	
L_QUANTITY	decimal	
L_EXTENDEDPRICE	decimal	
L_DISCOUNT	decimal	
L_TAX	decimal	
L_RETURNFLAG	fixed text, size 1	
L_LINESTATUS	fixed text, size 1	
L_SHIPDATE	date	
L_COMMITDATE	date	
L_RECEIPTDATE	date	
L_SHIPINSTRUCT	fixed text, size 25	
L_SHIPMODE	fixed text, size 10	

L_COMMENT	variable text size 44	
Primary Key: L_ORDERKEY, L_LINENUMBER		

Πίνακας 3.7: NATION Table Layout

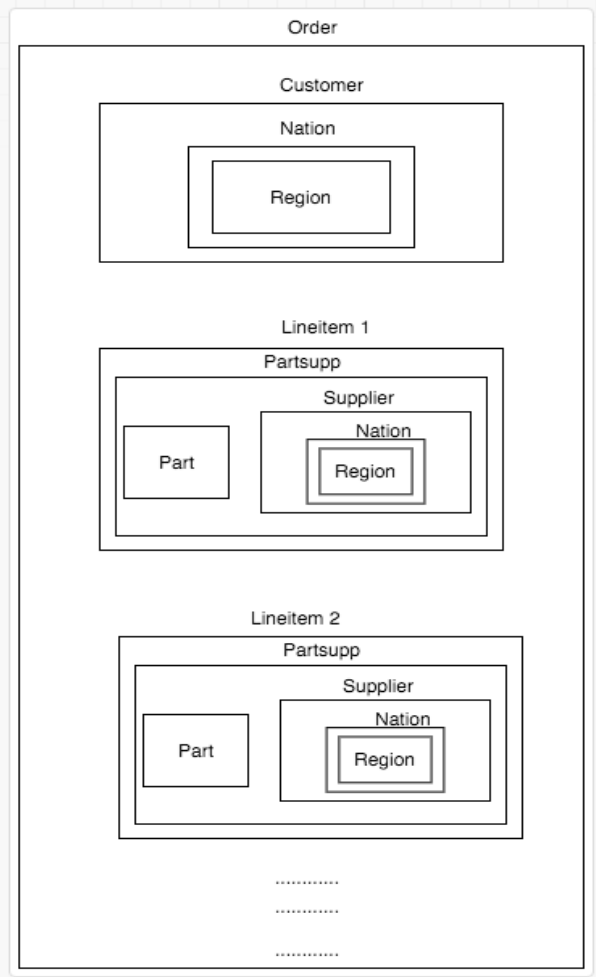
Column Name	Datatype	Requirements Comment
N_NATIONKEY	identifier	25 nations are populated
N_NAME	fixed text, size 25	
N_REGIONKEY	identifier	Foreign Key to R_REGIONKEY
N_COMMENT	variable text, size 152	
Primary Key: N_NATIONKEY		

Πίνακας 3.8: REGION Table Layout

Column Name	Datatype	Requirements Comment
R_REGIONKEY	identifier	5 regions are populated
R_NAME	fixed text, size 25	
R_COMMENT	variable text, size 152	
Primary Key: R_REGIONKEY		

3.2.2 MongoDB σχήμα

Το σχήμα της MongoDB φαίνεται στην Εικόνα 3.2 που ακολουθεί τη μη κανονικοποιημένη προσέγγιση και παρακάτω παρουσιάζονται τα χαρακτηριστικά του κάθε εγγράφου. Το σχήμα φαίνεται να είναι πολύ πιο απλό με αυτή την προσέγγιση αλλά το πληρώνει σε μέγεθος της βάσης δεδομένων το οποίο είναι πιο μεγάλο λόγω των διπλότυπων δεδομένων.



Εικόνα 3.2: The denormalized model of MongoDB [17]

Πίνακας 3.9: Order document

Field	Datatype of Value
_id	ObjectID
orderstatus	String
totalprice	Double
orderdate	Date
orderpriority	String
clerk	String
shippriority	String
comment	String
customer	Object<Customer>
LineItems	Array<lineitem>

Πίνακας 3.10: Customer document

Field	Datatype of Value
_id	ObjectID
name	String
address	String
phone	String
acctbal	Double
mktsegment	String
comment	String
nation	Object<Nation>

Πίνακας 3.11: Nation document

Field	Datatype of Value
_id	ObjectID
name	String
comment	String
region	Object<region>

Πίνακας 3.12: Region document

Field	Datatype of Value
_id	ObjectID
name	String
comment	String

Πίνακας 3.13: Lineitem document

Field	Datatype of Value
_id	ObjectID
quantity	Double
extendedprice	Double
discount	Double
tax	Double
returnflag	String
linestatus	String
shipdate	Date
commitdate	Date
receiptdate	Date
shipinstruct	String
shipmode	String
comment	String
partsupp	Object<partsupp>

Πίνακας 3.14: Partsupp document

Field	Datatype of Value
_id	ObjectID
availqty	Double
supplycost	Double
comment	String
part	Object<part>
supplier	Object<supplier>

Πίνακας 3.15: Supplier document

Field	Datatype of Value
_id	ObjectID
name	String
address	String
phone	String
acctbal	Double
comment	String
nation	Object<nation>

Πίνακας 3.16: Part document

Field	Datatype of Value
_id	ObjectID
name	String
mfgr	String
brand	String
type	String
size	Double
container	String
retailprice	Double
comment	String

3.3 Ερωτήματα Βάσεων Δεδομένων

Τα ερωτήματα που χρησιμοποιήθηκαν για να γίνουν οι πειραματικές μετρήσεις είναι αυτά του μετροπρογράμματος TPC-H. Πιο συγκεκριμένα, χρησιμοποιήθηκαν τα ερωτήματα Q1, Q3, Q4 τα οποία και θα παρουσιαστούν στις υπο-ενότητες 3.3.1 και 3.3.2. Αξίζει να επισημάνουμε ότι αρχικά υλοποιήθηκε στα προγράμματα της εργασίας και το ερώτημα Q5 όμως δεν επέστρεφε κάποιο αποτέλεσμα όταν η βάση ήταν αρκετά μικρή με χωρητικότητα 1MB (SF=), οπότε δεν χρησιμοποιήθηκε στις μετρήσεις και ούτε θα παρουσιαστεί στη συνέχεια.

Τέλος, δεν υλοποιήθηκαν αυτούσια τα ερωτήματα αλλά υπέστησαν μικρές τροποποιήσεις οι οποίες είναι οι εξής:

- **Q1**: Δεν χρησιμοποιήθηκε το κομμάτι "- interval '[DELTA]' day" του τμήματος WHERE.
- **Q3**: Στο τμήμα ORDER BY η ταξινόμηση έγινε μόνο ως προς το revenue.
- **Q4**: Δεν χρησιμοποιήθηκε το κομμάτι "+ interval '3' month" του τμήματος WHERE αλλά τροποποιήθηκε η ημερομηνία της δεύτερης συνθήκης κατά 3 μήνες αργότερα για να επιστραφούν ίσα αποτελέσματα.

3.3.1 MySQL ερωτήματα

Q1 query

```
SELECT
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
        sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
FROM
        lineitem
WHERE
        l_shipdate <= date '1998-12-01'
GROUP BY
        l_returnflag,
        l_linestatus
ORDER BY
        l_returnflag,
        l_linestatus;
```

Q3 query

```
SELECT
        l_orderkey,
        sum(l_extendedprice * (1 - l_discount)) as revenue,
        o_orderdate,
        o_shippriority
FROM
        customer,
        orders,
        lineitem
WHERE
        c_mktsegment = 'BUILDING'
        AND c_custkey = o_custkey
        AND l_orderkey = o_orderkey
        AND o_orderdate < date '1995-03-15'
        AND l_shipdate > date '1995-03-15'
GROUP BY
        l_orderkey,
        o_orderdate,
        o_shippriority
ORDER BY
        revenue desc;
```

Q4 query

```
SELECT
        o_orderpriority,
        count(*) as order_count
FROM
        orders
WHERE
        o_orderdate >= date '1993-07-01'
        and o_orderdate < date '1993-10-01'
        and exists (
                SELECT
                        *
                FROM
                        lineitem
                WHERE
                        l_orderkey = o_orderkey
                        and l_commitdate < l_receiptdate
        )
GROUP BY
o_orderpriority
ORDER BY
o_orderpriority;
```

3.3.2 MongoDB ερωτήματα

Τα αντίστοιχα ερωτήματα στην MongoDB υλοποιήθηκαν με το aggregation pipeline. Επειδή είναι αρκετά μεγάλα, τα ερωτήματα έχουν χωριστεί σε πολλές γραμμές για ευκολία στην ανάγνωση. Επίσης, στο έγγραφο αυτό, το κάθε ερώτημα έχει τυπωθεί σε δύο στήλες που αυτό σημαίνει όταν φτάνει το τέλος της σελίδας και δεν έχει τελειώσει, τότε συνεχίζει στην δεξιά στήλη και ύστερα στην επόμενη σελίδα.

Q1 query

```
db.order.aggregate(
  [{
    "$match":{
      "LineItems":{
        "$elemMatch":{
          "shipdate":{
            "$lte": ISODate("1998-12-
01T00:00:00.000Z")
          }
        }
      }
    }
  },
  {
    "$unwind":"$LineItems"
  },
  {
    "$project":{
      "LineItems.returnflag":1,
      "LineItems.linestatus":1,
      "LineItems.quantity":1,
      "LineItems.extendedprice":1,
      "LineItems.discount":1,
      "l_dis_min_1":{
        "$subtract":[
          1,
          "$LineItems.discount"
        ]
      },
      "l_tax_plus_1":{
        "$add":[
          "$LineItems.tax",
          1
        ]
      }
    }
  },
  {
    "$group":{
      "_id":{
        "returnflag":"$LineItems.returnflag",
        "linestatus":"$LineItems.linestatus"
      },
      "sum_qty":{
        "$sum":"$LineItems.quantity"
      },
      "sum_base_price":{
        "$sum":"$LineItems.extendedprice"
      },
      "sum_disc_price":{
        "$sum":{
          "$multiply":[
            "$LineItems.extendedprice",
            "$l_dis_min_1"
          ]
        }
      },
      "sum_charge":{
        "$sum":{
          "$multiply":[
            "$LineItems.extendedprice",
            {
```

```
                "$multiply":[
                  "$l_tax_plus_1",
                  "$l_dis_min_1"
                ]
              }
            ]
          }
        },
          "avg_qty":{
          "$avg":"$LineItems.quantity"
        },
        "avg_price":{
          "$avg":"$LineItems.extendedprice"
        },
        "avg_disc":{
          "$avg":"$LineItems.discount"
        },
        "count_order":{
          "$sum":1
        }
      }
    },
    {
      "$sort":{
        "_id.returnflag":1,
        "_id.linestatus":1
      }
    }
]);
```

Q3 query

```
db.order.aggregate(
[
  {
    "$unwind":"$LineItems"
  },
  {
    "$match":{
      "customer.mktsegment":"BUILDING",
      "orderdate":{
        "$lt":  ISODate("1995-03-
15T00:00:00.000Z")
      },
      "LineItems.shipdate":{
        "$gt": ISODate("1995-03-
15T00:00:00.000Z")
      }
    }
  },
  {
    "$project":{
      "orderdate":1,
      "shippriority":1,
      "LineItems.extendedprice":1,
      "l_dis_min_1":{
        "$subtract":[
          1,
          "$LineItems.discount"
        ]
```

```
      }
    }
  },
  {
    "$group":{
      "_id":{
            "order" : "$_id",
        "orderdate":"$orderdate",
        "shippriority":"$shippriority"
      },
      "revenue":{
        "$sum":{
          "$multiply":[
            "$LineItems.extendedprice",
            "$l_dis_min_1"
          ]
        }
      }
    }
  },
  {
    "$sort":{
      "revenue" : -1
    }
  }
]);
```

Q4 query

```
db.order.aggregate(
[
{
        "$unwind"  : "$LineItems"
},
{
    "$project":{
      "orderdate":1,
      "orderpriority":1,
      "eq":{
        "$cond":[
          {
            "$lt":[
              "$LineItems.commitdate",
              "$LineItems.receiptdate"
            ]
          },
          1,
          0
        ]
      }
    }
  },
  {
    "$match":{
          "eq":{
        "$eq":1
      },
          "orderdate" : {  $gte: ISODate("1993-
07-01T00:00:00.000Z"),  $lt: ISODate("1993-10-
01T00:00:00.000Z")   }
```

```
    }
  },
  {
    "$group": { "_id": {
          "_id": "$_id",
          "orderpriority" : "$orderpriority"
        }
     }
  },
  {
    "$group":{
      "_id":{
        "orderpriority":"$_id.orderpriority"
      },
      "order_count":{
        "$sum":1
      }
    }
  },
  {
    "$sort":{
      "_id.orderpriority":1
    }
  }
]);
```

3.4 Ρυθμίσεις Βάσεων Δεδομένων

Οι ρυθμίσεις που έγιναν στις βάσεις δεδομένων πριν τις πειραματικές μετρήσεις αφορούν κυρίως την MySQL διότι από την πλευρά της MongoDB δεν έγινε κάποια αλλαγή και χρησιμοποιήθηκε όπως ήταν μετά την εγκατάσταση. Οι αλλαγές στην MySQL έγιναν στο αρχείο ρυθμίσεων my.ini (MySQL Server Instance Configuration File) οι οποίες είναι οι εξής:

- Αυξήθηκε το μέγεθος του buffer pool της μηχανής αποθήκευσης InnoDB έτσι ώστε να καλυτερέψει η απόδοση ελαχιστοποιώντας τις προσπελάσεις στο σκληρό δίσκο (innodb_buffer_pool_size=3G).
- Απενεργοποιήθηκαν κάποια στοιχεία της μηχανής αποθήκευσης InnoDB έτσι ώστε να μην αποθηκεύει στην προσωρινή μνήμη (innodb buffer pool) δεδομένα κατά την επανεκκίνηση του MySQL Server για να έχουμε πιο καθαρή εικόνα των αποδόσεων(innodb_buffer_pool_load_abort=ON,innodb_buffer_pool_load_at_st artup=OFF, innodb_buffer_pool_dump_at_shutdown=OFF).
- Κατά τις μετρήσεις των μεθόδων εισαγωγής και διαγραφής έγινε δοκιμή με δύο διαφορετικές τιμές της μεταβλητής innodb_flush_log_at_trx_commit μιας και η τιμή 1 προσφέρει δοσοληψίες με πλήρη ACID ιδιότητες ενώ η τιμή 2 όχι κάτι που αυξάνει την απόδοση εις βάρος της ασφάλειας των δεδομένων .

3.5 Μετρικές (Metrics)

Για τη μέτρηση της απόδοσης των βάσεων δεδομένων απαιτείται μια κοινή μετρική. Ο πιο σημαντικός παράγοντας για μια εφαρμογή είναι ο χρόνος που χρειάζεται για την ολοκλήρωση μιας εργασίας, και στην περίπτωση των βάσεων δεδομένων είναι ο χρόνος που χρειάζεται για να ολοκληρωθεί μια δοσοληψία. Πιο συγκεκριμένα, τα πειράματα που έγιναν μέτρησαν τον χρόνο που χρειάζεται να ολοκληρωθεί ένα σύνολο δοσοληψιών αφού μια μόνη της τελειώνει σε αμελητέο χρόνο. Οι μετρικές όπως φαίνονται στις γραφικές παραστάσεις του επόμενου κεφαλαίου είναι:

1) Ο χρόνος που χρειάζεται να ολοκληρωθεί ένα σύνολο δοσοληψιών με διαφορετικό αριθμό νημάτων (threads). Αυτό μας βοηθάει να δούμε πως συμπεριφέρονται οι βάσεις δεδομένων τόσο με μια σύνδεση όσο και με περισσότερες.
2) Από το από πάνω μπορούμε να πάρουμε και ως μετρική τον αριθμό των ερωτημάτων που ολοκληρώνει μια βάση δεδομένων ανά δευτερόλεπτο με διαφορετικό αριθμό νημάτων. Ο τύπος είναι:
   Queries per second = Total number of queries / Execution time.
3) Ο χρόνος που χρειάζεται να ολοκληρωθεί ένα σύνολο δοσοληψιών με διαφορετικό μέγεθος βάσης δεδομένων ως προς ένα νήμα. Αυτό μας βοηθάει να δούμε την απόδοση μιας βάσης δεδομένων όσο αυξάνεται το μέγεθός της.
4) Οι υπόλοιπες μετρικές προέρχονται από απλή μέτρηση διαφορετικού συνόλου ερωτημάτων με ένα νήμα.

4. ΠΕΙΡΑΜΑΤΙΚΕΣ ΜΕΤΡΗΣΕΙΣ

Στο κεφάλαιο αυτό δίνονται τα αποτελέσματα των πειραματικών μετρήσεων σε μορφή γραφικών παραστάσεων. Για αναλυτικότερα αποτελέσματα ανατρέξτε στο ΠΑΡΑΡΤΗΜΑ Α. Οι παράμετροι που εξετάστηκαν για την εξαγωγή των αποτελεσμάτων παρουσιάστηκαν στην ενότητα 3.5. Οι τρεις πρώτες ενότητες έχουν εξεταστεί με ερωτήματα SELECT από το TPC-H benchmark όπως εξηγήθηκε στην ενότητα 3.3, ενώ στην τελευταία ενότητα χρησιμοποιήθηκαν απλά ερωτήματα INSERT και DELETE για να μετρηθεί η απόδοση των βάσεων δεδομένων σε εισαγωγές-διαγραφές και όχι μόνο σε ανάγνωση των δεδομένων.

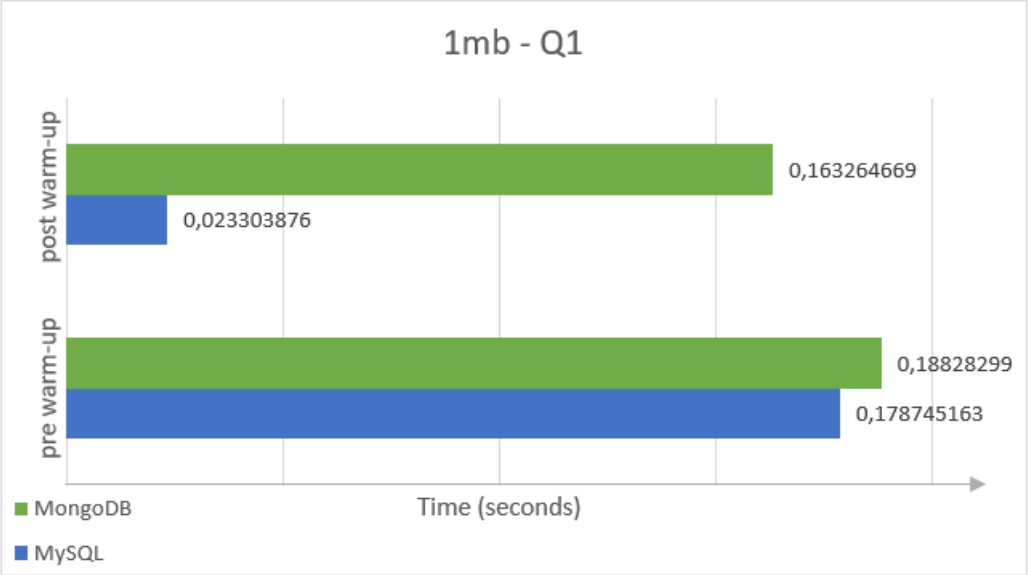
4.1 Πριν και Μετά το ‘ζέσταμα’ των Βάσεων Δεδομένων

Με τον όρο ‘ζέσταμα’ εννοούμε τη διαδικασία κατά την οποία μια βάση δεδομένων αποθηκεύει πληροφορίες (όπως δείκτες) και δεδομένα στην κρυφή μνήμη (cache memory) της μηχανής αποθήκευσης που χρησιμοποιεί, αν τυχόν το υποστηρίζει, με αποτέλεσμα την αύξηση της απόδοσης αφού η επικοινωνία με τη μνήμη RAM είναι αρκετές φορές πιο γρήγορη από το σκληρό δίσκο. Η μηχανή αποθήκευσης InnoDB της MySQL που χρησιμοποιείται στην εργασία, έχει το InnoDB buffer pool για λειτουργίες caching [11], όπως και η μηχανή αποθήκευσης WiredTiger της MongoDB [14]. Υπάρχουν αρκετές λειτουργίες caching, όπως είναι για παράδειγμα η MySQL query caching, οι οποίες δεν εξετάστηκαν στην παρούσα έρευνα.

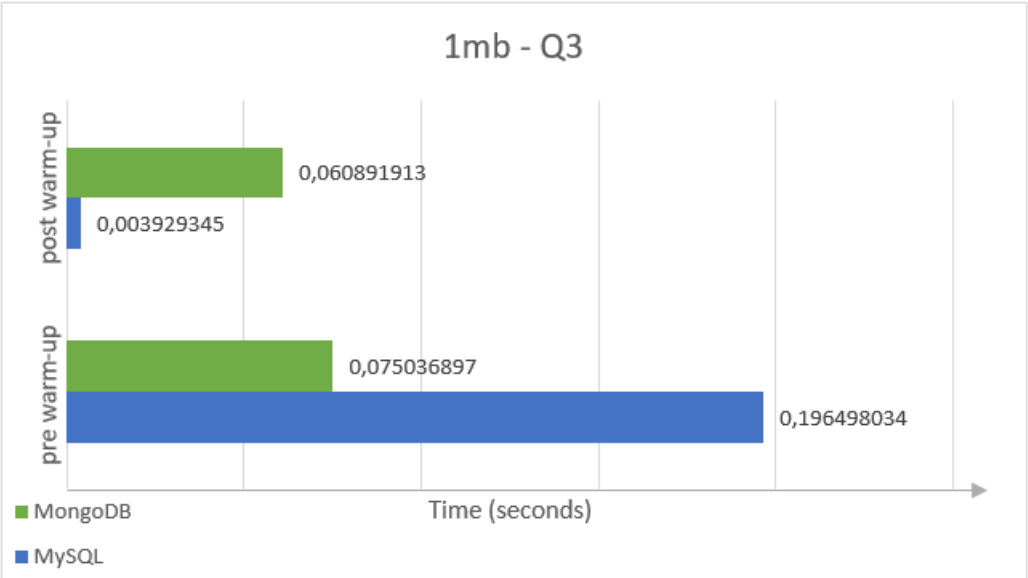
Για το λόγο αυτό, εξετάστηκε ξεχωριστά το κάθε ερώτημα πριν και μετά το ζέσταμα όπως θα φανεί και στις γραφικές παραστάσεις παρακάτω. Η MySQL παραμετροποιήθηκε ώστε να χρησιμοποιεί 3GB κρυφής μνήμης, ενώ η MongoDB χρησιμοποιεί από μόνη της 50% της μέγιστης μνήμης RAM (περίπου 3.2GB στο μηχάνημα μας, με περισσότερες πληροφορίες σχετικά με τις προδιαγραφές του υπολογιστή στο ΠΑΡΑΡΤΗΜΑ Β). Το ‘ζέσταμα’ στις κρυφές μνήμες κατά τις επανεκκινήσεις των βάσεων δεδομένων πραγματοποιούταν με εκτέλεση ερωτημάτων, και όχι με άλλες τεχνικές όπως εξηγήθηκαν στην ενότητα 3.4.

Γενικά, τα αποτελέσματα αυτής της ενότητας είναι και αυτά που αντιπροσωπεύουν περισσότερο τις αποδόσεις των βάσεων δεδομένων, αφού σε ρεαλιστικές περιπτώσεις οι βάσεις δεδομένων είναι πολύ πιο μεγάλες και ο διαφορετικός τύπος ερωτημάτων σε συνδυασμό με τον αυξημένο αριθμό συνδέσεων καθιστούν την κρυφή μνήμη να απαιτεί πολύ περισσότερη μνήμη RAM με αποτέλεσμα να μειώνεται η απόδοση των λειτουργιών caching. Παρόλα αυτά, σε κατανεμημένες βάσεις δεδομένων μπορεί να μην ισχύει κάτι τέτοιο.Τα αποτελέσματα των ερωτημάτων με διαφορετικό μέγεθος των βάσεων δεδομένων πριν και μετά το ‘ζέσταμα’ φαίνεται στις εικόνες παρακάτω.

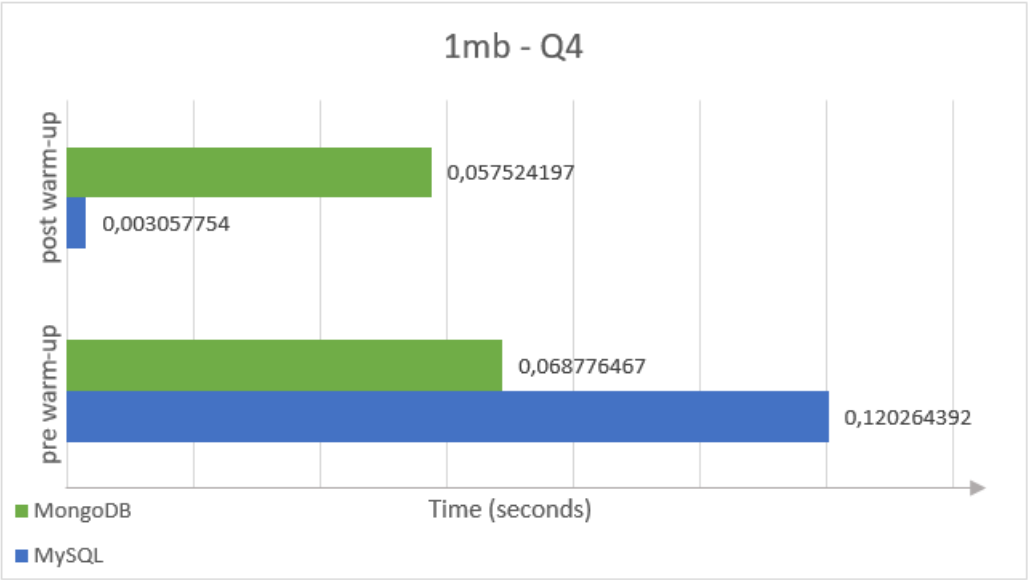
Database size 1MB



Σχήμα 4.1: Query Q1 with database size 1MB (pre and post warm-up graph)

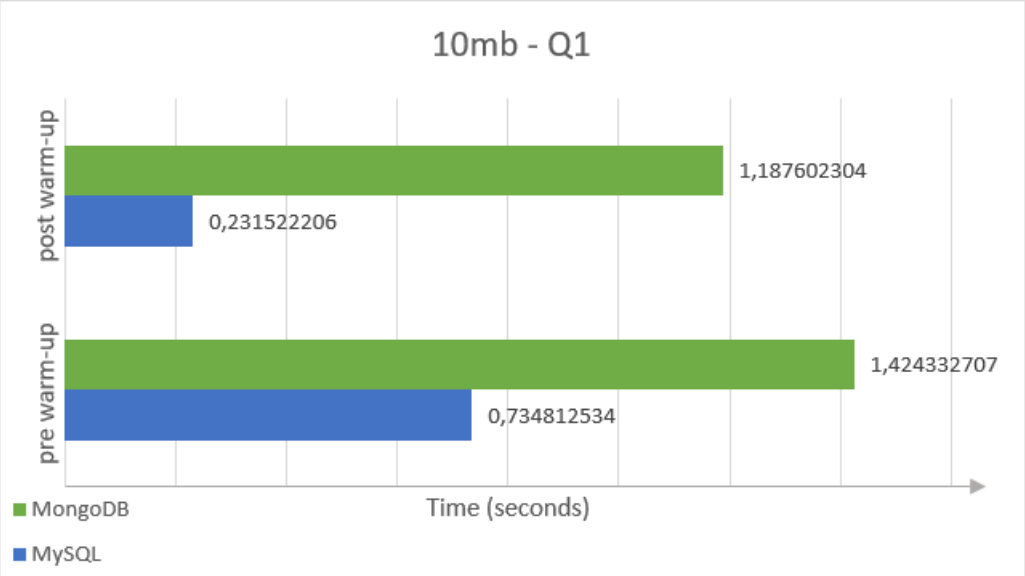


Σχήμα 4.2: Query Q3 with database size 1MB (pre and post warm-up graph)

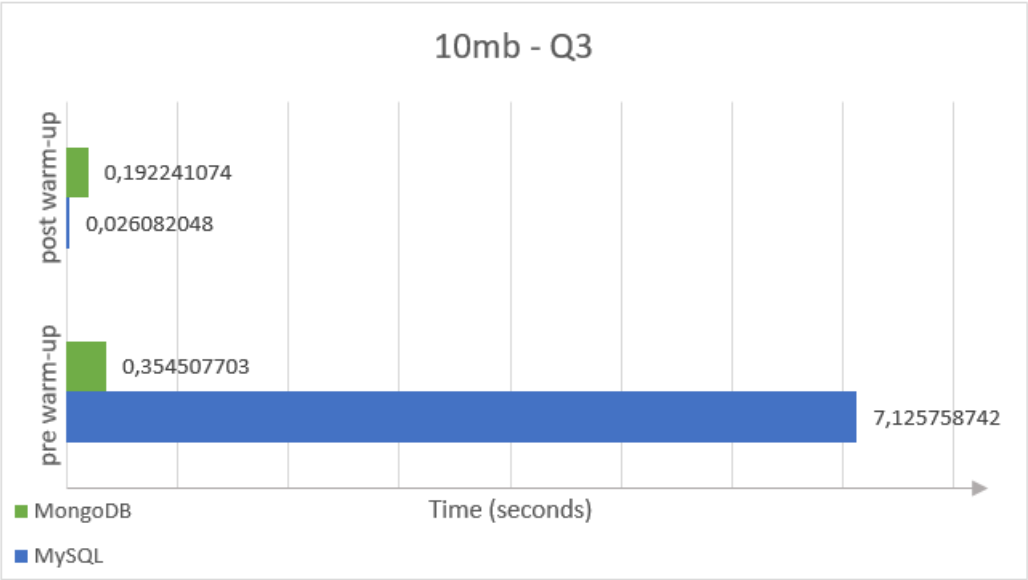


Σχήμα 4.3: Query Q4 with database size 1MB (pre and post warm-up graph)

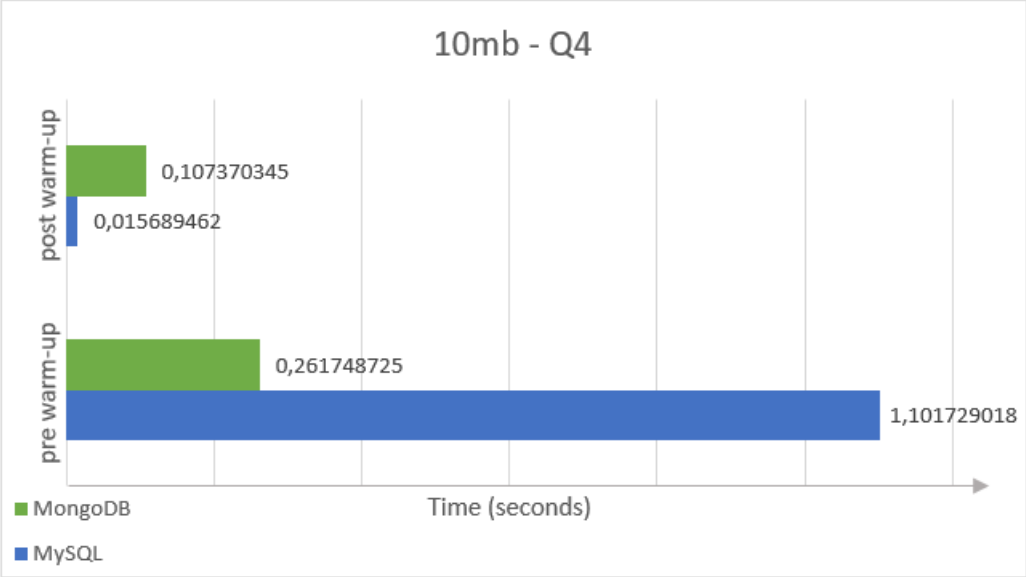
Database size 10MB



Σχήμα 4.4: Query Q1 with database size 10MB (pre and post warm-up graph)

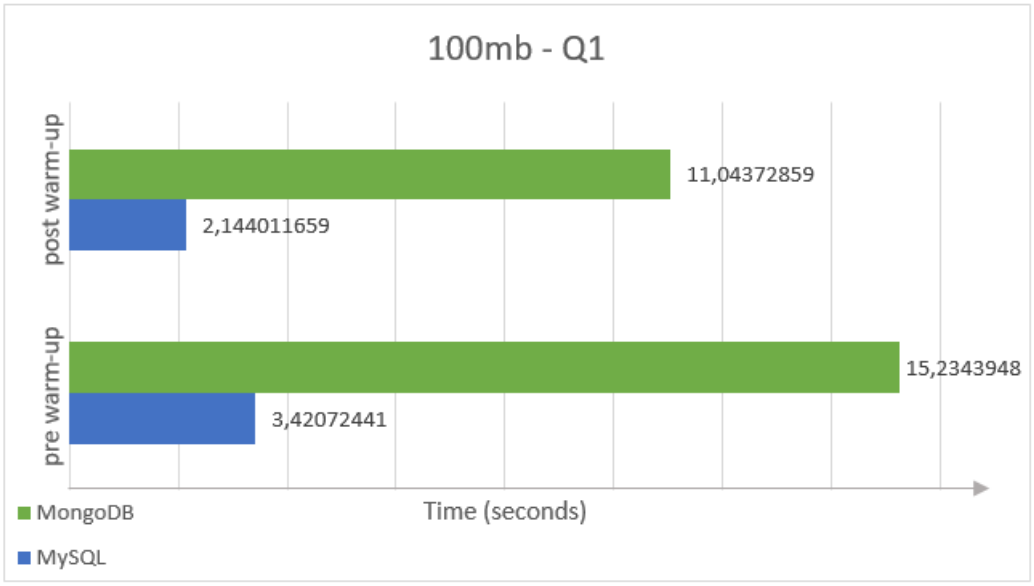


Σχήμα 4.5: Query Q3 with database size 10MB (pre and post warm-up graph).

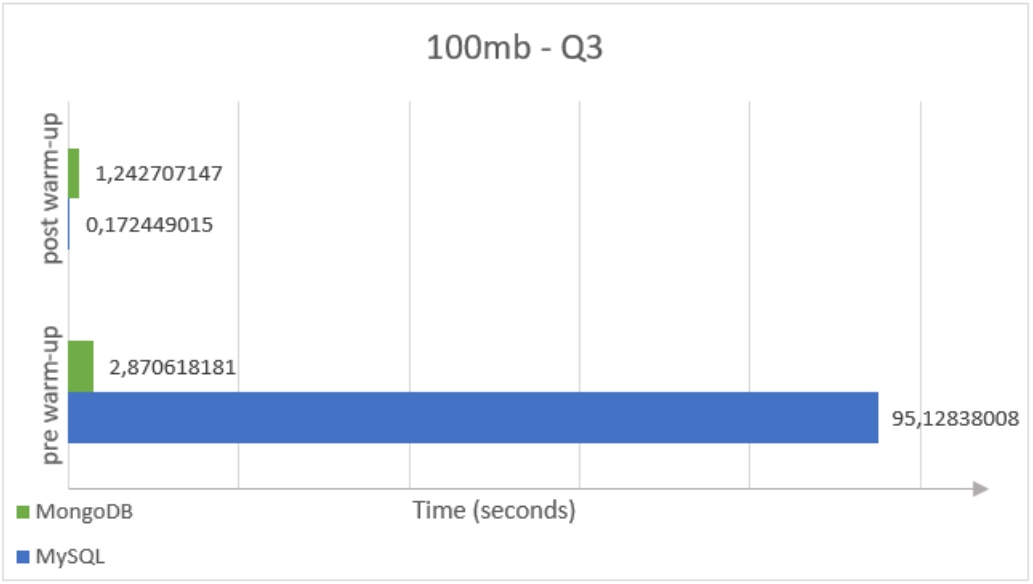


Σχήμα 4.6: Query Q4 with database size 10MB (pre and post warm-up graph)

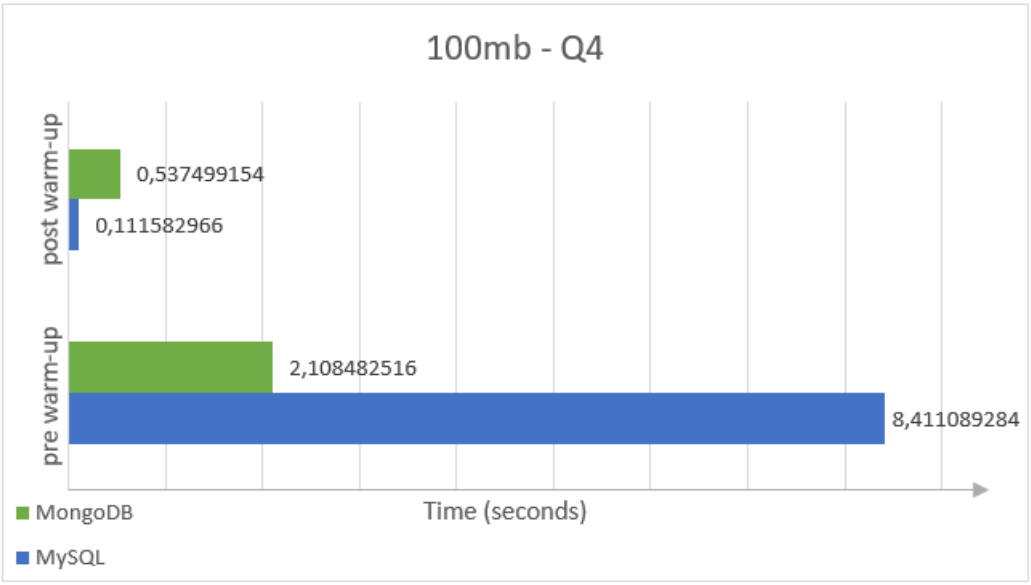
Database size 100MB



Σχήμα 4.7: Query Q1 with database size 100MB (pre and post warm-up graph)

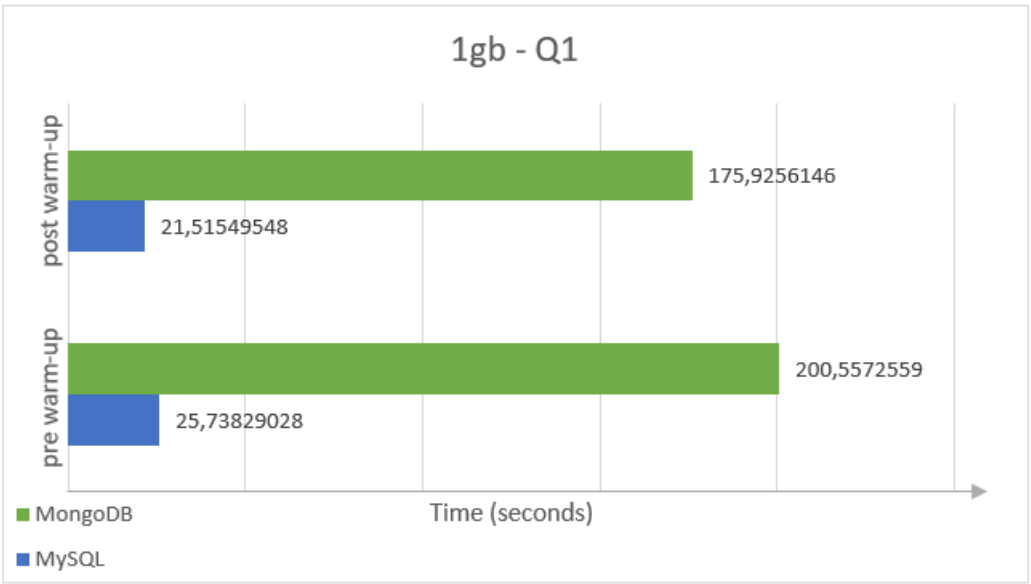


Σχήμα 4.8: Query Q3 with database size 100MB (pre and post warm-up graph)

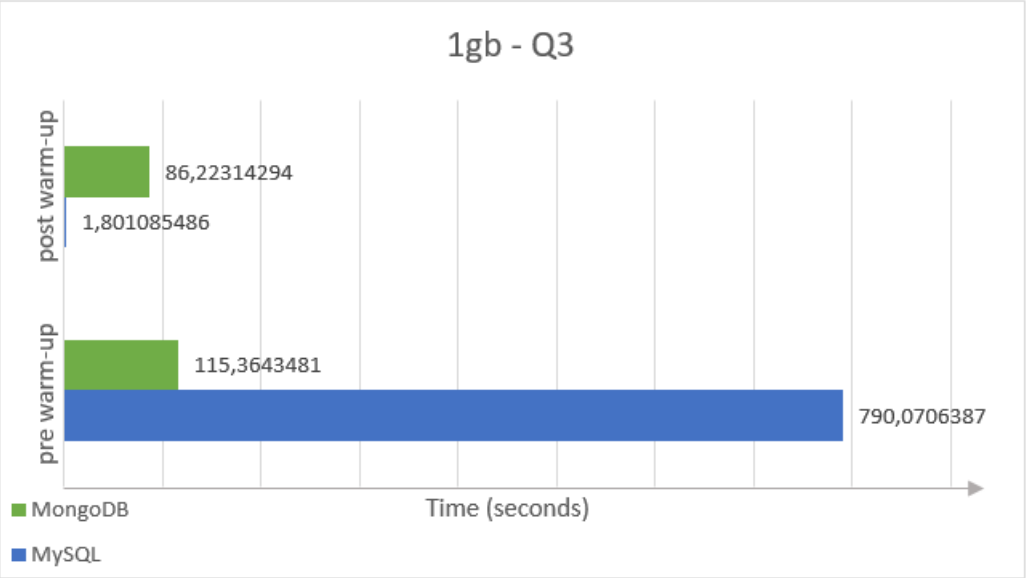


Σχήμα 4.9: Query Q4 with database size 100MB (pre and post warm-up graph)

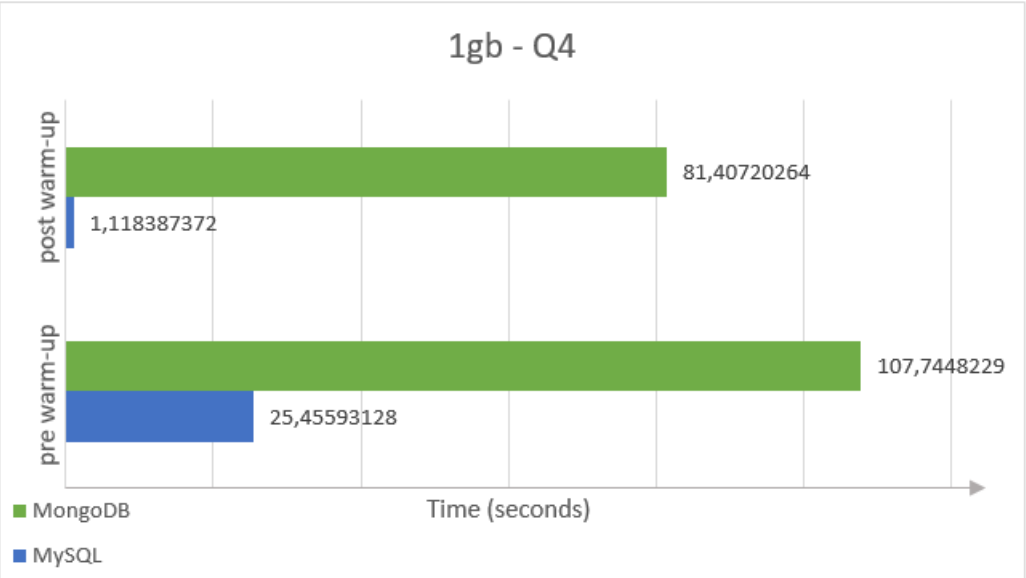
Database size 1GB



Σχήμα 4.10: Query Q1 with database size 1GB (pre and post warm-up graph)



Σχήμα 4.11: Query Q3 with database size 1GB (pre and post warm-up graph)



Σχήμα 4.12: Query Q4 with database size 1GB (pre and post warm-up graph)

Παρατηρήσεις

- Όπως φαίνεται και από τις γραφικές παραστάσεις, η MySQL διαχειρίζεται πολύ καλύτερα τις λειτουργίες caching τόσο σε απαίτηση μνήμης όσο και σε απόδοση. Αυτό μπορεί να οφείλεται στη δομή των πινάκων που έχουν οι σχεσιακές βάσεις δεδομένων και στον τρόπο που αποθηκεύονται στη μνήμη με αποτέλεσμα να δίνει περισσότερες δυνατότητες εφαρμογών caching από ότι στην MongoDB που απλά αποθηκεύουν εγγραφές σε μια συλλογή σε απλή δομή BSON. Αυτό έχει ως αποτέλεσμα τα ερωτήματα μετά το 'ζέσταμα' να εκτελούνται πιο γρήγορα στην MySQL.
- Το ερώτημα Q1 που κατά τ 'άλλα είναι ένα απλό ερώτημα παρατηρούμε ότι εκτελείται πιο αργά στην MongoDB πριν και μετά το 'ζέσταμα'. Αυτό εξαρτάται από δύο λόγους. Πρώτον, η φύση του ερωτήματος έρχεται σε αντίθεση με το schema

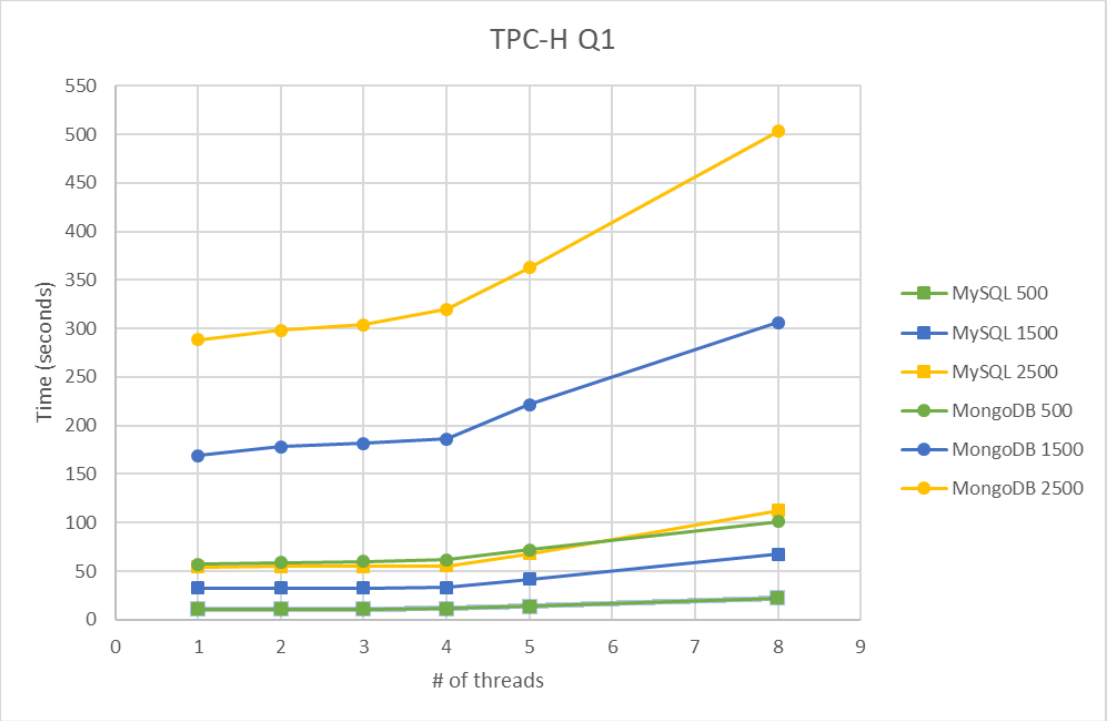
της MongoDB, που όπως είναι σχεδιασμένο δεν είναι αποδοτικό για τέτοιου είδους ερωτήματα αφού το συγκεκριμένο ψάχνει στο LineItems σε κάθε έγγραφο order, το οποίο είναι πίνακας με ενσωματωμένα έγγραφα lineitem. Δεύτερον, το τμήμα FROM του ερωτήματος περιέχει ένα μόνο πίνακα κάτι το οποίο δεν επιβαρύνει την MySQL στην εκτέλεσή του.

- Το ερώτημα Q4 παρατηρούμε ότι εκτελείται πιο γρήγορα στην MongoDB από ότι στην MySQL πριν το 'ζέσταμα' εκτός της περίπτωσης όπου η βάση δεδομένων έχει μέγεθος 1GB. Αυτό οφείλεται στο ότι η μνήμη δεν επαρκεί στην MongoDB για να φορτώσει το working set ολόκληρο, δηλαδή τους δείκτες και τα δεδομένα, έτσι κάνει περισσότερες προσπελάσεις στον σκληρό δίσκο και έχει ως αποτέλεσμα τη μείωση της απόδοσης. Αν είχαμε στο σύστημα μας περισσότερη μνήμη έτσι ώστε η MongoDB να μπορεί να χωρέσει όλο το working set η εκτέλεση θα ήταν ταχύτερη από ότι στην MySQL.
- Τέλος, το ερώτημα Q3 βλέπουμε ότι πριν το 'ζέσταμα' εκτελείται πιο γρήγορα στην MongoDB ακόμα και όταν η βάση δεδομένων είναι 1GB παρόλο που εξακολουθεί το working set να μην χωράει ολόκληρο στη μνήμη. Αυτό οφείλεται ξεκάθαρα στο ότι το ερώτημα αυτό είναι σύνθετο με πολλούς πίνακες στο τμήμα FROM με αποτέλεσμα να επιβαρύνει σημαντικά την απόδοση της MySQL ενώ η MongoDB να μην έχει κάποιο σημαντικό πρόβλημα αφού με το συγκεκριμένο schema έχει συνδυάσει τους πίνακες σε ενσωματωμένα έγγραφα. Αξίζει να σημειωθεί ότι αυτό το ερώτημα είναι και το πιο χρονοβόρο για την MySQL.

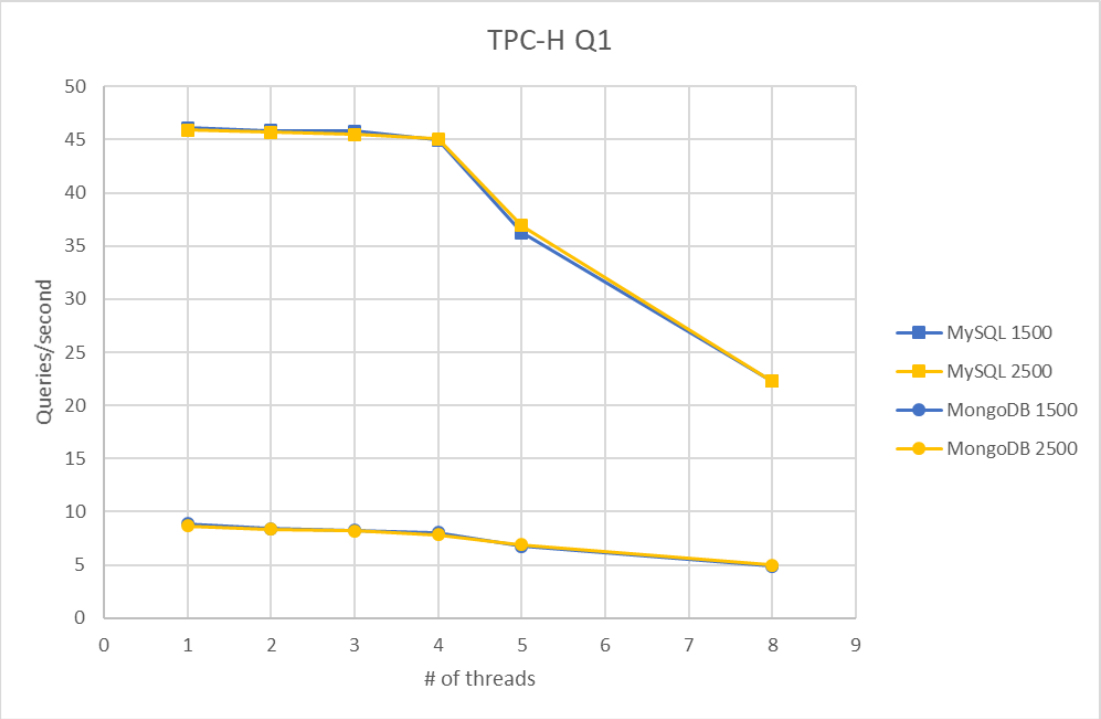
4.2 Πολλαπλά SELECT Ερωτήματα

Σε αυτή την ενότητα παρουσιάζουμε γραφικές παραστάσεις από τις 1) και 2) μετρικές, όπως ειπώθηκαν στην ενότητα 3.5, οι οποίες μας βοηθάνε να εξετάσουμε τις βάσεις με πολλές δοσοληψίες και πολλές συνδέσεις ταυτόχρονα. Είναι φανερό ότι τα αποτελέσματα αυτά είναι μετά το 'ζέσταμα' των βάσεων δεδομένων αφού θα πρέπει να εκτελέσουμε το ίδιο ερώτημα πολλές φορές.

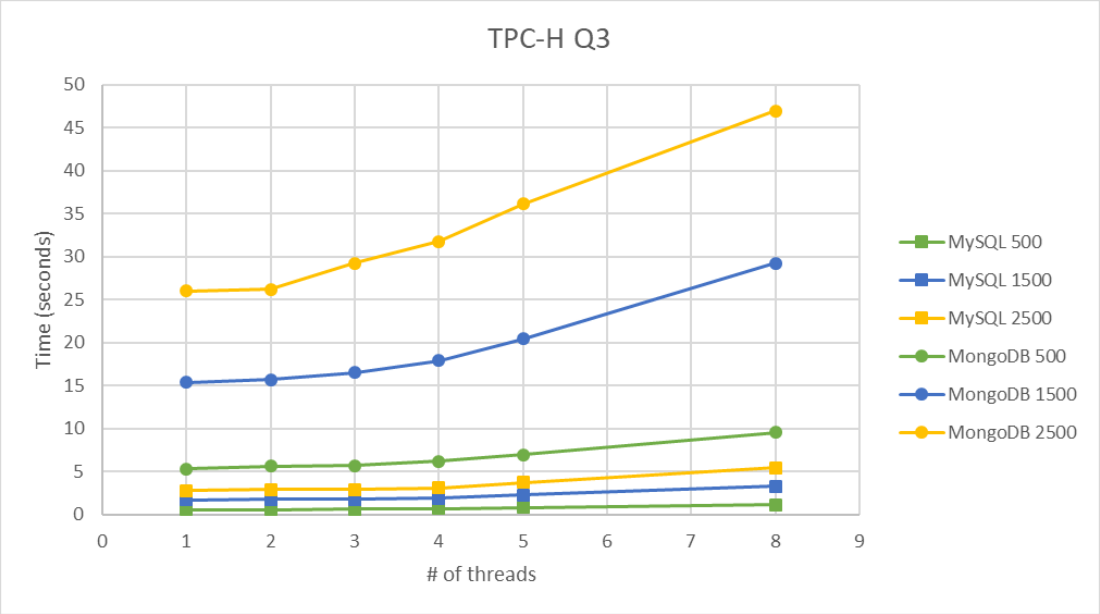
Ο αριθμός των ερωτημάτων που εκτελέστηκαν σε κάθε δοκιμή είναι διαφορετικός με τιμές 500, 1500, 2500 όπου εξετάστηκαν με 1, 2, 3, 4, 5 και 8 νήματα (threads) τη φορά. Ο χρόνος εκτέλεσης μετρήθηκε ξεχωριστά σε κάθε νήμα και στη συνέχεια υπολογίστηκε ο μέσος όρος από αυτά ώστε να επιστραφεί το τελικό αποτέλεσμα. Στις γραφικές παραστάσεις Queries/second χρησιμοποιήθηκε μόνο 1500 και 2500 αριθμός ερωτημάτων ώστε να είναι πιο καθαρό το σχήμα αφού όπως φαίνεται οι γραμμές είναι πιο αλληλοκαλυπτόμενες. Τέλος, τα δεδομένα που χρησιμοποιήθηκαν για τις μετρήσεις έχουν μέγεθος 1MB.



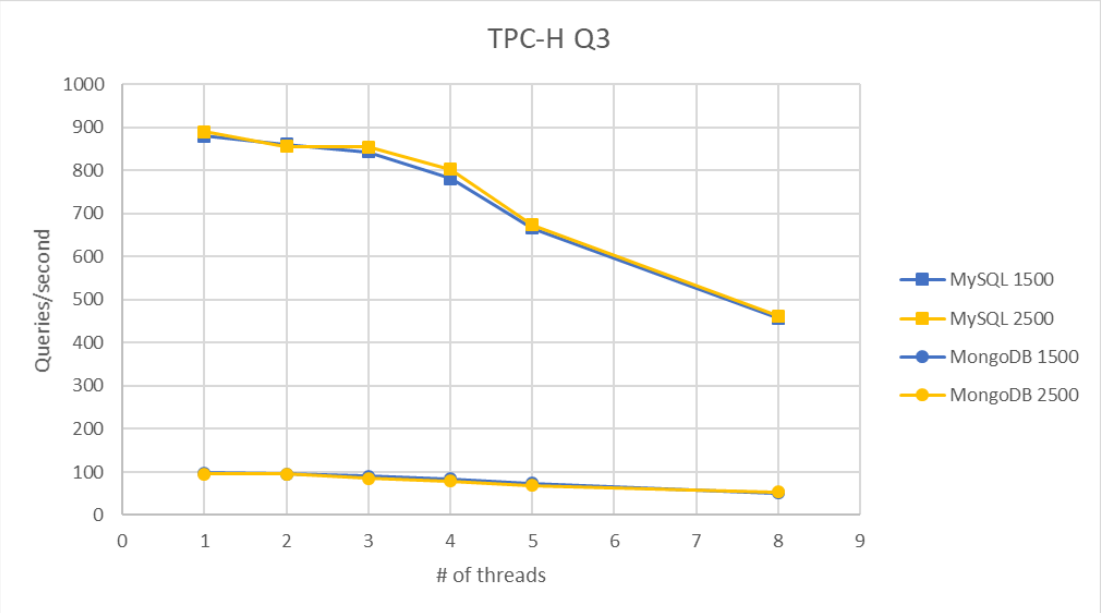
Σχήμα 4.13: Query Q1 (Time – Number of threads graph)



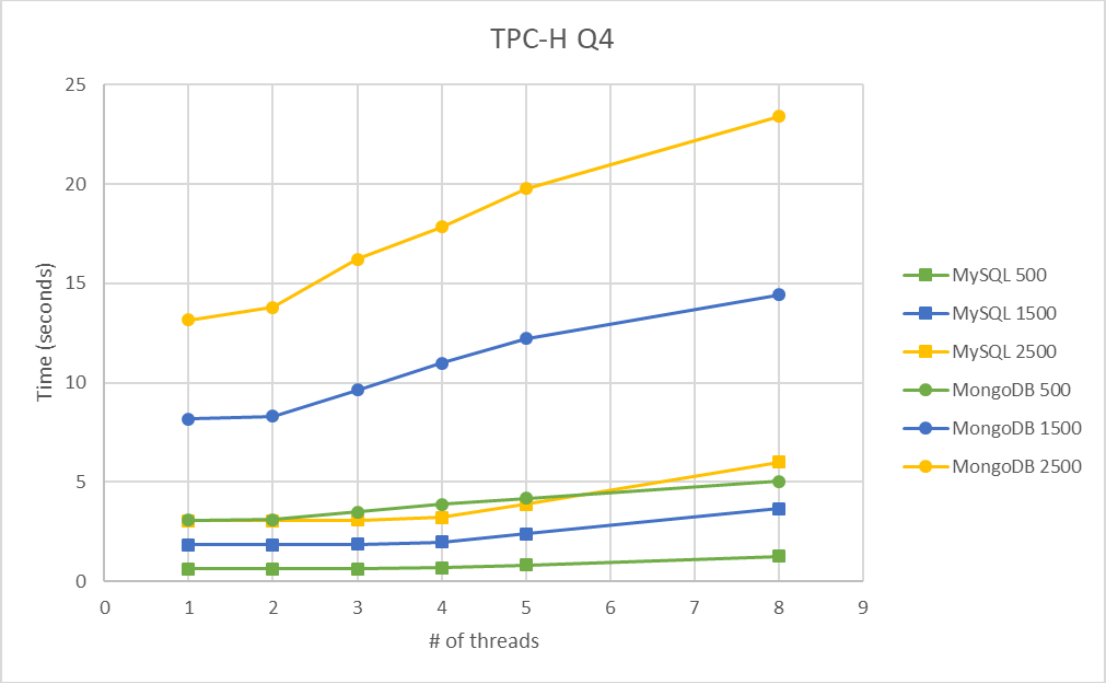
Σχήμα 4.14: Query Q1 (Queries/second – Number of threads graph)



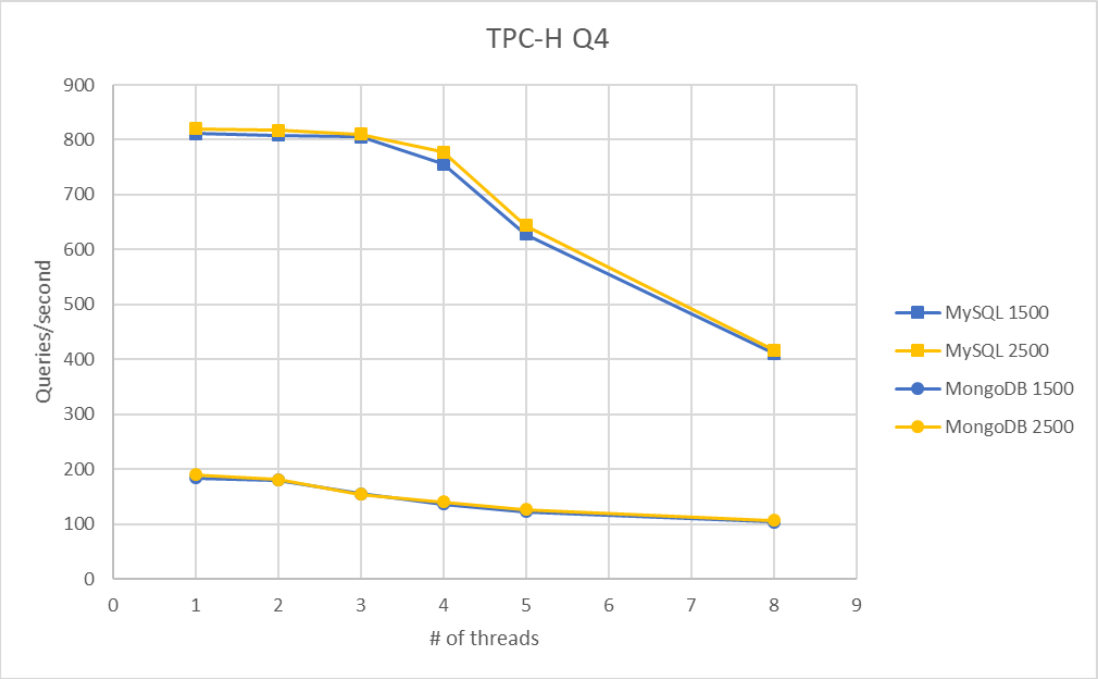
Σχήμα 4.15: Query Q3 (Time – Number of threads graph)



Σχήμα 4.16: Query Q3 (Queries/second – Number of threads graph)



Σχήμα 4.17: Query Q4 (Time – Number of threads graph)



Σχήμα 4.18: Query Q4 (Queries/second – Number of threads graph)

Παρατηρήσεις

- Ήταν αναμενόμενο ότι τα αποτελέσματα αυτής της ενότητας θα ευνοούσαν την MySQL αφού συμπεριφέρεται καλύτερα μετά το ‘ζέσταμα’ σε σχέση με την MongoDB, αλλά οι μετρικές που χρησιμοποιήθηκαν εδώ, δεν αντικατοπτρίζουν πλήρως χρήσεις πραγματικού κόσμου, όπως εξηγήθηκε και στην ενότητα 4.1.
- Όπως φαίνεται και στις γραφικές παραστάσεις, εκτός του ερωτήματος Q4 στην MongoDB, παρατηρείται και στις δύο βάσεις δεδομένων μια μείωση της απόδοσης

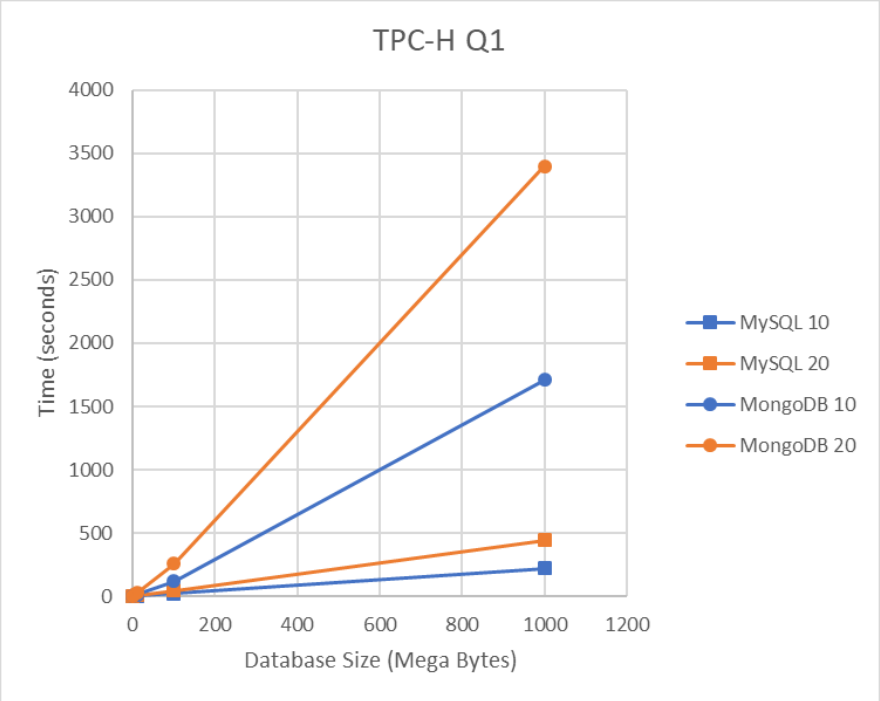
πάνω από τα 4 νήματα. Αυτό λογικά πρέπει να οφείλεται στον επεξεργαστή του συστήματος ο οποίος έχει 4 φυσικούς πυρήνες. Παρόλα αυτά, ο συγκεκριμένος επεξεργαστής είναι hyper-threading ο οποίος έχει παραπάνω καταχωρητές και μονάδες εκτέλεσης με αποτέλεσμα να μπορεί να χειρίζεται σχεδόν ταυτόχρονα 2 νήματα σε κάθε πυρήνα του. Από αυτό μπορούμε να φανταστούμε ότι αν είχαμε εξετάσει περιπτώσεις με πάνω από 8 νήματα, θα βλέπαμε ακόμα μεγαλύτερη μείωση της απόδοσης των βάσεων δεδομένων. Περαιτέρω πληροφορίες για τις προδιαγραφές του υπολογιστή στο ΠΑΡΑΡΤΗΜΑ Β.

- Τέλος, παρατηρείται ότι το ποσοστό αύξησης της καθυστέρησης είναι ανάλογο του ποσοστού αύξησης των ερωτημάτων προς εκτέλεση, που όμως η διαφορά αυξάνεται σταδιακά όσο έχουμε περισσότερες συνδέσεις (νήματα).

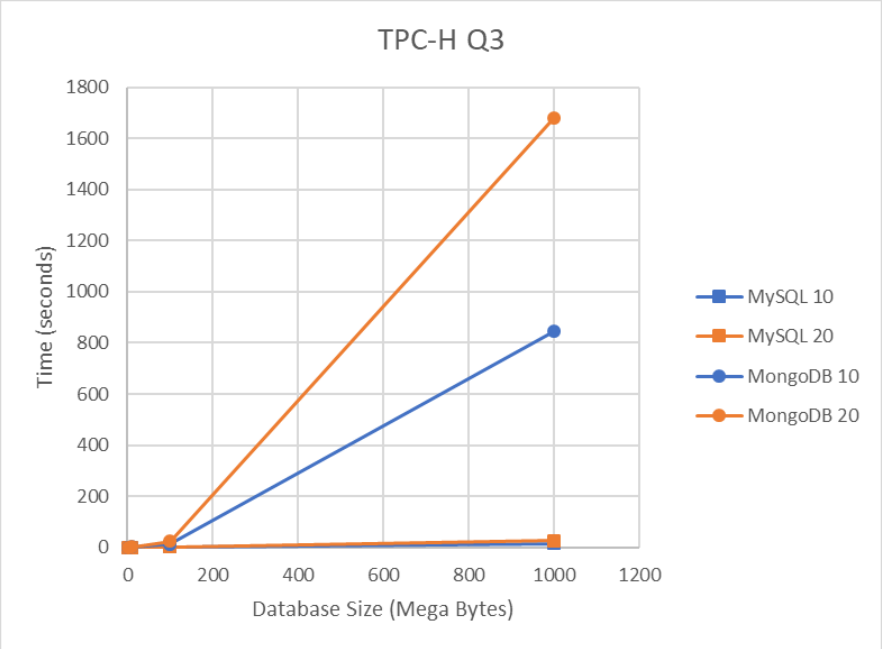
4.3 Μέγεθος της Βάσης Δεδομένων

Στις γραφικές παραστάσεις που ακολουθούν θα εξετάσουμε πως συμπεριφέρονται οι βάσεις δεδομένων όσο αυξάνονται τα μεγέθη τους, όπως εξηγήθηκε και στην 3) μετρική της ενότητας 3.5. Στα συγκεκριμένα παραδείγματα εξετάστηκε λιγότερος αριθμός ερωτημάτων, μόνο 10 και 20 ερωτήματα τη φορά, λόγω της μεγάλης καθυστέρησης της εκτέλεσης σε βάσεις δεδομένων με μεγαλύτερο μέγεθος του 1MB (10MB, 100MB, 1GB) και εκτελέστηκαν μόνο τα δύο ερωτήματα Q1 και Q3.

Και σε αυτά τα αποτελέσματα περιμένουμε καλύτερους χρόνους για την MySQL λόγω του πιο αποδοτικού caching σε σχέση με την MongoDB. Για το λόγο αυτό θα στρέψουμε την προσοχή μας περισσότερο στο πως συμπεριφέρεται κάθε βάση ξεχωριστά όσο αυξάνεται το μέγεθός της και όχι στη μεταξύ τους διαφορά.



Σχήμα 4.19: Query Q1 (Time – Database size [1MB,10MB,100MB,1GB] graph)



Σχήμα 4.20: Query Q3 (Time – Database size [1MB,10MB,100MB,1GB] graph)

Παρατηρήσεις

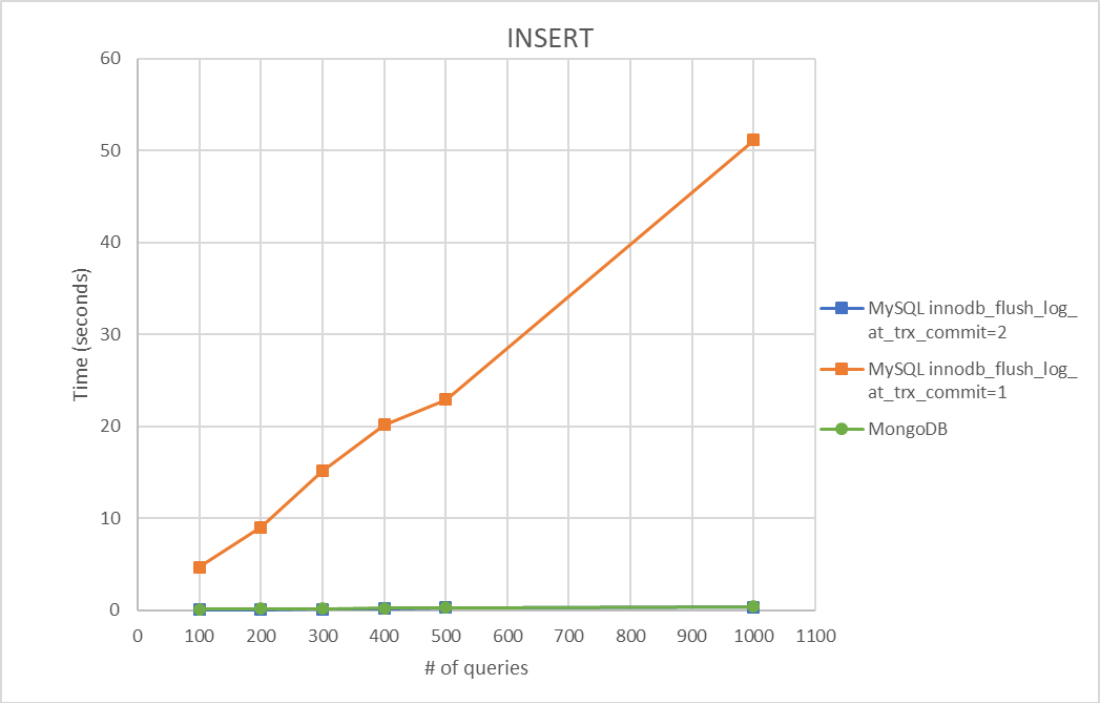
Από τις από πάνω γραφικές παραστάσεις παρατηρούμε ότι το ποσοστό αύξησης της καθυστέρησης είναι ανάλογο του ποσοστού αύξησης του μεγέθους της κάθε βάσης εκτός από την MongoDB όταν το μέγεθος της βάσης είναι 1GB. Επειδή στα σχήματα δεν παρατηρείται εύκολα ή σχεδόν καθόλου αυτή η αναλογία μπορείτε να ανατρέξετε στο ΠΑΡΑΡΤΗΜΑ Α που έχει όλα τα αποτελέσματα αναλυτικά. Όπως αναφέραμε σε προηγούμενη ενότητα το working set, όταν το μέγεθος της βάσης της MongoDB είναι 1Gb, δεν χωράει όλο στη μνήμη RAM και για το λόγο αυτό παρατηρείται αυτή η μείωση της απόδοσης. Πιο συγκεκριμένα για παράδειγμα, στο Q3 με 20 ερωτήματα MongoDB, ενώ η βάση δεδομένων αυξάνεται 10 φορές (100MB -> 1GB) η καθυστέρηση αυξάνεται κατά 67 φορές! Για ακόμη μια φορά φαίνεται πόσο σημαντικό είναι να παρέχουμε αρκετή μνήμη στην MongoDB είτε να υλοποιούμε τεχνικές όπως το sharding, ώστε να ελαχιστοποιούνται οι προσπελάσεις στο σκληρό δίσκο. Από την άλλη, παρόλο που η MySQL απαιτεί λιγότερη μνήμη, μπορεί σε πραγματικά συστήματα να υπήρχε και εκεί τέτοιο πρόβλημα λόγω του διαφορετικού είδους ερωτημάτων και των πολλών συνδέσεων.

4.4 INSERT και DELETE

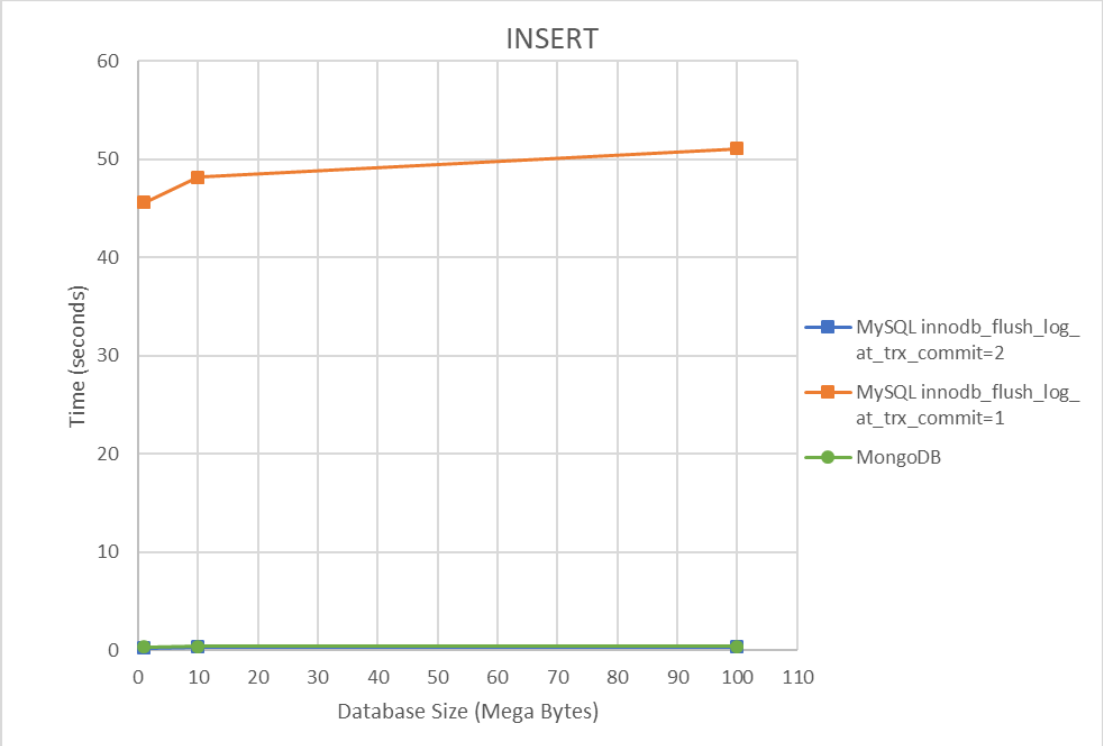
Στην ενότητα αυτή εξετάζονται οι εισαγωγές και οι διαγραφές στις δύο βάσεις δεδομένων μετρώντας το χρόνο εκτέλεσης τόσο με διαφορετικό σύνολο δοσοληψιών όσο και με διαφορετικό μέγεθος των βάσεων αυτών. Οι εικόνες 4.21 και 4.23, που εξετάζουν την περίπτωση με διαφορετικό σύνολο δοσοληψιών, οι βάσεις δεδομένων έχουν μέγεθος 100MB ενώ οι εικόνες 4.22 και 4.24, που εξετάζουν την περίπτωση με διαφορετικό μέγεθος των βάσεων, εκτελούν 1000 δοσοληψίες. Επίσης, για την MySQL έγιναν μετρήσεις με διαφορετικές τιμές της μεταβλητής innodb_flush_log_at_trx_commit, η οποία εξηγήθηκε στην ενότητα 3.4.

Για να πραγματοποιηθεί τέτοιου είδους σύγκριση στις δύο αυτές ξεχωριστές βάσεις δεδομένων αποφασίστηκε να γίνει εισαγωγή και ύστερα η διαγραφή εγγράφων order στην περίπτωση της MongoDB, και πλειάδων στον πίνακα ORDERS της MySQL.

Αδιαφορώντας για τα αποτελέσματα των γραφικών παραστάσεων της ενότητας αυτής, οι χρόνοι καθυστέρησης στην MySQL σε πραγματικά συστήματα θα είναι σαφώς πιο αργοί αφού σε εισαγωγές ή διαγραφές μιας εγγραφής order θα τροποποιούνταν τουλάχιστον δύο πίνακες (ORDERS, LINEITEM) και όχι ένας όπως τις παρακάτω μετρήσεις. Αλλά αφού έχουμε φτιάξει το σχήμα της MongoDB ως ένα μη κανονικοποιημένο σχήμα θα συγκρίνουμε την εισαγωγή και διαγραφή ενός εγγράφου με την αντίστοιχη εγγραφή σε έναν πίνακα της MySQL.

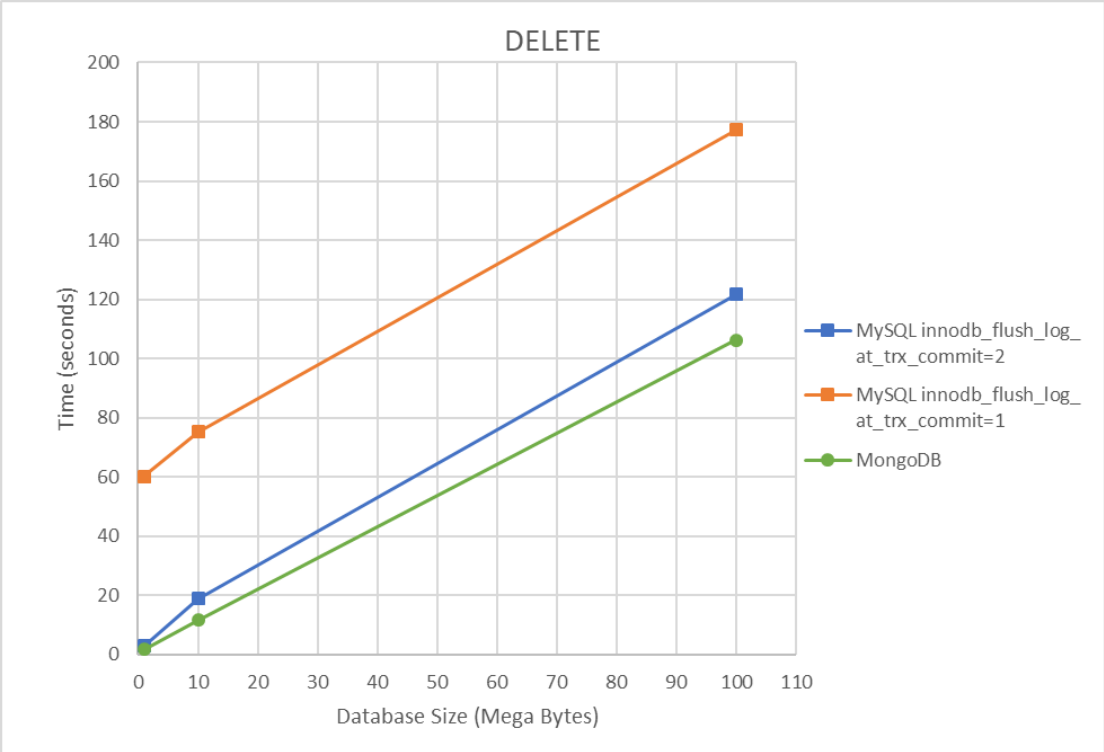


Σχήμα 4.21: INSERT (Time – Number of queries graph)



Σχήμα 4.22: INSERT (Time – Database size [1MB,10MB,100MB] graph)

Σχήμα 4.23: DELETE (Time – Number of queries graph)



Σχήμα 4.24: DELETE (Time – Database size [1MB,10MB,100MB] graph)

Παρατηρήσεις

Από τις γραφικές παραστάσεις παρατηρούμε ότι όταν οι δοσοληψίες στην MySQL έχουν πλήρη ACID ιδιότητες (με τιμή μεταβλητής innodb_flush_log_at_trx_commit=1) τότε η MongoDB είναι πολύ πιο γρήγορη ενώ στην άλλη περίπτωση είναι περίπου ίδιες με την

MongoDB να είναι ελάχιστα ταχύτερη στις διαγραφές όπου σε μεγαλύτερες και πιο περίπλοκες βάσεις δεδομένων μπορεί αυτή η διαφορά να ήταν πιο αισθητή. Επιπλέον, το πλεονέκτημα που έχουν και οι δυο βάσεις δεδομένων είναι ότι η μεταβολή των γραφικών παραστάσεων είναι γραμμική.

5. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΕΥΝΑ

Συμπεράσματα

Στην εργασία αυτή έγινε μια έρευνα και σύγκριση της απόδοσης και της κλιμάκωσης των Σχεσιακών Συστημάτων Βάσεων Δεδομένων και των NoSQL βάσεων δεδομένων με στόχο την εξερεύνηση στο πως οι διαφορετικοί παράγοντες επηρεάζουν την κάθε μία και σε ποιες περιπτώσεις είναι καταλληλότερη η μια βάση δεδομένων από την άλλη. Ο πρώτος τύπος βάσεων δεδομένων που εξετάστηκε, οι σχεσιακές βάσεις, αναπτύχθηκαν έχοντας τη δομή ως κύρια έννοια που βασίζεται σε πίνακες οι οποίοι έπονται μετά από ένα προκαθορισμένο σχήμα. Ένα από τα σημαντικότερα χαρακτηριστικά τους, είναι αυτό το σχήμα το οποίο δίνει μια λογική αναπαράσταση της βάσης σε συνδυασμό με τις σχέσεις μεταξύ των πινάκων που επιτρέπουν τη γρήγορη δημιουργία τέτοιων βάσεων δεδομένων, την ευκολία στην ανάπτυξη, και την εξάλειψη των διπλότυπων δεδομένων με ταυτόχρονη εγγύηση της αξιοπιστίας τους. Οι NoSQL βάσεις δεδομένων οι οποίες είναι σχετικά καινούργιες, κατάφεραν να γίνουν γνωστές αφού παρέχουν καλή απόδοση και οριζόντια επεκτασιμότητα με αποτέλεσμα να είναι κατάλληλες σε κέντρα δεδομένων που χειρίζονται τεράστιο όγκο δεδομένων και στην ικανότητα τους να επεκτείνονται με ευκολία αφού δεν απαιτούν κάποιο προκαθορισμένο σχήμα.

Πιο συγκεκριμένα, πραγματοποιήθηκαν διάφορες μετρήσεις οι οποίες τέσταραν σε απόδοση και σε κλιμάκωση τη σχεσιακή βάση δεδομένων MySQL με την NoSQL βάση δεδομένων MongoDB. Τα πειράματα που έγιναν περιλάμβαναν τόσο διαφορετικό αριθμό δοσοληψιών και συνδέσεων όσο και διαφορετικό μέγεθος των βάσεων δεδομένων, προκειμένου να αναλυθεί πως αποδίδει η κάθε μια όσο αυξάνεται ο φόρτος εργασίας. Γενικά, τα αποτελέσματα πριν το ‘ζέσταμα’ είναι αυτά που αντιπροσωπεύουν περισσότερο τις αποδόσεις των δύο βάσεων δεδομένων, όπως εξηγήθηκε στην ενότητα 4.1, από τα οποία φαίνεται η μεταξύ τους σύγκριση ενώ τα υπόλοιπα χρησιμοποιήθηκαν κυρίως για να δούμε πως συμπεριφέρεται η κάθε βάση ξεχωριστά, με εξαίρεση τα αποτελέσματα των εισαγωγών και διαγραφών.

Από τα αποτελέσματα παρατηρείται ότι η MongoDB χειρίζεται καλύτερα πολύπλοκα ερωτήματα σε αντίθεση με την MySQL η οποία χειρίζεται καλύτερα απλά ερωτήματα. Αυτό ήταν αναμενόμενο αφού η MongoDB με το μη κανονικοποιημένο σχήμα έχει συνδυάσει τους πίνακες ως ενσωματωμένα έγγραφα με αποτέλεσμα να μη χρονοτριβεί όπως την MySQL που κάνει τη διαδικασία του καρτεσιανού γινομένου όταν στο τμήμα FROM υπάρχουν τουλάχιστον δύο πίνακες. Παρόλα αυτά, στην MongoDB με ένα μη κανονικοποιημένο σχήμα υπάρχουν πολλά διπλότυπα δεδομένα με αποτέλεσμα να απαιτεί περισσότερη μνήμη RAM κατά τη λειτουργία caching, αφού φορτώνει όλο το working set στη μνήμη, κάτι το οποίο θα πρέπει να προσεχθεί αφού ξεπερνώντας τη διαθέσιμη μνήμη η απόδοση του πέφτει κατακόρυφα. Όσον αφορά το ρυθμό μεταβολής της καθυστέρησης, όσο αυξάνεται το μέγεθος των βάσεων δεδομένων ή ο αριθμός των συνδέσεων, οι δύο βάσεις συμπεριφέρονται σχεδόν παρόμοια με αρνητικό σημείο και πάλι το μέγεθος της μνήμης που απαιτείται από την MongoDB. Τέλος, οι εισαγωγές και διαγραφές που εξετάστηκαν έδειξαν ότι η MongoDB χειρίζεται ταχύτερα τέτοιες λειτουργίες σε σχέση με την MySQL.

Δεδομένου ότι οι δύο βάσεις δεδομένων συμπεριφέρονται διαφορετικά ανάλογα με το είδος των ερωτημάτων που χρησιμοποιείται, η τελική επιλογή εξαρτάται κυρίως από τους τύπους των εφαρμογών που θα χρησιμοποιεί το σύστημα. Με λίγα λόγια, θα πρέπει να μελετάται προσεκτικά πως οι εφαρμογές θα αλληλοεπιδρούν με τα δεδομένα, τόσο με λειτουργίες ανάγνωσης όσο και με εγγραφής, προτού επιλέξουμε τη σωστή βάση δεδομένων προς υλοποίηση.

Μελλοντική Έρευνα

Δεδομένου ότι ο χρόνος και το μέγεθος που διατίθενται για αυτήν την έρευνα είναι σχετικά μικροί, οι δυνατότητες για περαιτέρω επεκτάσεις και βελτιώσεις είναι πάρα πολλές. Για να συγκρίνουμε πλήρως τις αποδόσεις και τις δυνατότητες των σχεσιακών και NoSQL βάσεων δεδομένων, όπως είναι λογικό δεν αρκεί μόνο η εξέταση των δύο συγκεκριμένων βάσεων δεδομένων που είναι η MySQL και η MongoDB. Υπάρχουν πάρα πολλά άλλα συστήματα βάσεων δεδομένων τόσο RDBMS (Oracle Database, Microsoft SQL Server, IBM DB2, IBM Informix και άλλα) όσο και NoSQL (Cassandra, Redis, CouchDB, HBase, Neo4j και άλλα). Επίσης, όσον αφορά αυτήν την εργασία περιοριστήκαμε στην εξέταση ενός συγκεκριμένου μετροπρογράμματος, του TPC-H benchmark, το οποίο παρέχει ένα μοντέλο δεδομένων που αφορά εφαρμογές B2C και από αυτό ασχοληθήκαμε με λίγα ερωτήματα και με μια συγκεκριμένη προσέγγιση του σχήματος της MongoDB (μη κανονικοποιημένη μορφή), όπου στην πραγματικότητα υπάρχουν άπειρα μοντέλα βάσεων δεδομένων που μπορούν να εξεταστούν. Τέλος, ένα από τα πιο σημαντικά χαρακτηριστικά που δεν υλοποιήθηκαν στην παρούσα εργασία είναι αυτά των κατανεμημένων βάσεων δεδομένων όπου έχουν διάφορες λειτουργίες ώστε να βελτιώνουν τις βάσεις όπως είναι το sharding και το replication.

Η αξιοπιστία των δύο τύπων βάσεων δεδομένων μπορεί να μην ήταν μέρος αυτής της εργασίας, αλλά στο μέλλον μπορεί να εξεταστεί και αυτή η πτυχή. Ο λόγος είναι κυρίως για να κατανοήσουμε αν αυξάνοντας την απόδοση και την επεκτασιμότητα σε μια βάση δεδομένων επηρεάζεται η αξιοπιστία σε σύγκριση με μια άλλη. Συγκεκριμένα, δεδομένου ότι οι NoSQL βάσεις δεδομένων δεν εγγυώνται τις ιδιότητες ACID, μια χρυσή τομή μπορεί να βρεθεί στην οποία η απόδοση και επεκτασιμότητα να φτάνουν σε ένα αξιοπρεπές επίπεδο και ταυτόχρονα να υπάρχει ένας βαθμός εγγύησης της αξιοπιστίας.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	**Ελληνικός Όρος**
Database	Βάση Δεδομένων
Benchmark	Μετροπρόγραμμα
Miniworld	Μικρόκοσμος
Atomicity	Ατομικότητα
Consistency	Συνέπεια
Isolation	Απομόνωση
Availability	Διαθεσιμότητα
Tolerance	Ανοχή
Shard	Θραύσμα
Replicate	Αντιγραφή
Schema	Σχήμα
Optimized	Βελτιστοποιημένο
Table	Πίνακας
Document	Έγγραφο
Collection	Συλλογή
Query	Ερώτημα
Transaction	Δοσοληψία
Scalability	Επεκτασιμότητα
Aggregation	Συνάθροιση
Configuration	Διαμόρφωση
Metrics	Μετρικές
Normalized	Κανονικοποιημένο
Denormalized	Μη κανονικοποιημένο
Embedded	Ενσωματωμένο
Arbiter	Διαιτητής

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

SQL	Structured Query Language
RDBMS	Relational Database Management System
NoSQL	Not Only SQL
TPC-H	Transaction Processing Council Ad-hoc
B2C	Business-to-Consumer
YAML	Yet Another Markup Language
JSON	JavaScript Object Notation
BSON	Binary-Encoded JSON
CSV	Comma Separated Values
GUI	Graphical User Interface
UoD	Universe of Discourse
RAM	Random Access Memory

ΠΑΡΑΡΤΗΜΑ Ι ΑΝΑΛΥΤΙΚΟΤΕΡΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Pre and Post Warm-Up

Πίνακας Ι.1: Pre and Post Warm-Up with database size 1MB

tpch_1mb	Q1		Q3		Q4	
	MySQL	MongoDB	MySQL	MongoDB	MySQL	MongoDB
pre warm-up	0.178745163	0.18828299	0.196498034	0.075036897	0.120264392	0.068776467
post warm-up	0.023303876	0.163264669	0.003929345	0.060891913	0.003057754	0.057524197

Πίνακας Ι.2: Pre and Post Warm-Up with database size 10MB

tpch_10mb	Q1		Q3		Q4	
	MySQL	MongoDB	MySQL	MongoDB	MySQL	MongoDB
pre warm-up	0.734812534	1.424332707	7.125758742	0.354507703	1.101729018	0.261748725
post warm-up	0.231522206	1.187602304	0.026082048	0.192241074	0.015689462	0.107370345

Πίνακας Ι.3: Pre and Post Warm-Up with database size 100MB

tpch_100mb	Q1		Q3		Q4	
	MySQL	MongoDB	MySQL	MongoDB	MySQL	MongoDB
pre warm-up	3.42072441	15.234394803	95.128380084	2.870618181	8.411089284	2.108482516
post warm-up	2.144011659	11.043728588	0.172449015	1.242707147	0.111582966	0.537499154

Πίνακας Ι.4: Pre and Post Warm-Up with database size 1GB

tpch_1gb	Q1		Q3		Q4	
	MySQL	MongoDB	MySQL	MongoDB	MySQL	MongoDB
pre warm-up	25.738290277	200.557255905	790.070638709	115.364348102	25.455931275	107.7448229
post warm-up	21.515495479	175.92561464	1.801085486	86.223142943	1.118387372	81.407202636

MySQL

Πίνακας I.5: MySQL Q1 with database size 1MB

MySQL (tpch_1mb - Q1)						
# of Queries SELECT	1 thread	2 thread	3 thread	4 thread	5 thread	8 thread
10	0.219594174	0.222707352	0.223922981	0.254131745	0.311791032	0.456807741
20	0.439954019	0.441113608	0.457888351	0.514770863	0.624044586	0.911187353
100	2.183335899	2.202917551	2.210457589	2.403097371	2.85671041	4.536954717
200	4.369308331	4.386108117	4.398249018	4.665657542	5.687513748	9.026911096
300	6.544576982	6.556230153	6.61604308	6.841087539	8.395916043	13.598781907
400	8.683839846	8.724486252	8.81839565	9.105630959	11.157264936	18.02467223
500	10.835810779	10.902471493	10.95205181	11.334782256	13.90037799	22.49060607
1000	21.746720448	21.890004003	21.907486392	22.285482136	27.489614353	45.077854482
1500	32.524552555	32.727644795	32.761879347	33.343163778	41.412062034	67.338305057
2000	43.447991707	43.623057754	44.043766974	45.130632756	55.314700085	89.836273363
2500	54.45909885	54.703856473	54.955038951	55.479377611	67.721184921	112.361719011

Πίνακας I.6: MySQL Q3 with database size 1MB

My SQL (tpch_1mb - Q3)						
# of Queries SELECT	1 thread	2 thread	3 thread	4 thread	5 thread	8 thread
10	0.014181113	0.017770115	0.018968638	0.021675237	0.022730425	0.029399729
20	0.027020558	0.028609991	0.029982791	0.03429699	0.038216256	0.05420513
100	0.122602868	0.129851484	0.134762858	0.146816106	0.169822336	0.253455471
200	0.243960957	0.250116698	0.261969325	0.287831081	0.332377087	0.485206915
300	0.353786783	0.376440558	0.381902269	0.40950447	0.498457288	0.704285392
400	0.473076946	0.484181831	0.495415901	0.55475126	0.646443288	0.921251091
500	0.581068038	0.599859593	0.629845531	0.673045951	0.79464852	1.144074217
1000	1.136420646	1.178721352	1.208472594	1.296016874	1.520754339	2.226478328
1500	1.70368199	1.743232647	1.779997058	1.919619731	2.252792315	3.275944925
2000	2.272285823	2.307809015	2.380658723	2.511614373	2.982129548	4.338802486
2500	2.809050918	2.919016945	2.925942257	3.115850807	3.711926841	5.41312457

Πίνακας I.7: MySQL Q4 with database size 1MB

MySQL (tpch_1mb - Q4)						
# of Queries SELECT	1 thread	2 thread	3 thread	4 thread	5 thread	8 thread
10	0.013655203	0.01546982	0.01685131	0.019085199	0.021715963	0.029353851
20	0.028613891	0.029255166	0.029733178	0.032850736	0.039202389	0.056880048
100	0.133747575	0.137723723	0.143479317	0.155148567	0.182157465	0.265949852
200	0.258952079	0.261527646	0.272497939	0.291434862	0.349520296	0.516862703
300	0.386585789	0.388134578	0.394394802	0.426483768	0.512969548	0.758493395
400	0.510780347	0.511678419	0.518768907	0.560059427	0.664355616	1.008916413
500	0.63828145	0.638968911	0.642407994	0.681629153	0.824511706	1.256876456
1000	1.245691002	1.263115025	1.27261864	1.347061203	1.610810194	2.481501575
1500	1.848659256	1.854557174	1.862167209	1.983890856	2.390631976	3.648930731
2000	2.444619064	2.449616652	2.47637665	2.603297206	3.126414796	4.821806304
2500	3.047276945	3.058280834	3.08518685	3.215728669	3.883540475	6.004854461

Πίνακας I.8: MySQL Q1 with database size 10MB

MySQL (tpch_10mb - Q1)	
# of Queries SELECT	1 thread
10	2.295716148
20	4.579516171

Πίνακας I.9: MySQL Q1 with database size 100MB

MySQL (tpch_100mb - Q1)	
# of Queries SELECT	1 thread
10	21.350948261
20	42.762742044

Πίνακας I.10: MySQL Q1 with database size 1GB

MySQL (tpch_1gb - Q1)	
# of Queries SELECT	1 thread
10	218.620082939
20	441.738433167

Πίνακας I.11: MySQL Q3 with database size 10MB

MySQL (tpch_10mb - Q3)	
# of Queries SELECT	1 thread
10	0.204747565
20	0.402988371

Πίνακας I.12: MySQL Q3 with database size 100MB

MySQL (tpch_100mb - Q3)	
# of Queries SELECT	1 thread
10	1.335195574
20	2.579470189

Πίνακας I.13: MySQL Q3 with database size 1GB

MySQL (tpch_1gb - Q3)	
# of Queries SELECT	1 thread
10	15.090133792
20	27.366114906

MongoDB

Πίνακας Ι.14: MongoDB Q1 with database size 1MB

MongoDB (tpch_1mb - Q1)						
# of Queries SELECT	1 thread	2 thread	3 thread	4 thread	5 thread	8 thread
10	1.218076383	1.286726401	1.303074518	1.384375918	1.57893229	2.131994283
20	2.368951565	2.457334319	2.515382579	2.630997013	3.026445957	4.214001345
100	11.530916854	11.941156716	12.145572559	12.509880121	14.64384855	20.237829233
200	23.027646642	24.159332451	24.364624675	24.815282816	28.87050216	40.593224282
300	34.204913426	36.099578163	36.747050254	36.998835791	43.538235018	60.61878283
400	46.108468147	47.39435498	48.241384364	49.423933861	57.242998963	80.21815182
500	56.867949626	58.968475537	59.94476775	61.517205509	71.798870092	100.81130908
1000	113.505024479	118.992070285	119.430634603	121.310616437	155.20739822	201.143344264
1500	169.181097186	178.093616405	181.865734862	186.077340682	221.79171894	306.371177808
2000	231.381592717	239.860996602	241.27880175	249.88813098	300.520376307	413.693923289
2500	288.325874106	298.368047747	303.934418956	319.892871981	362.52808449	503.32067884

Πίνακας Ι.15: MongoDB Q3 with database size 1MB

MongoDB (tpch_1mb - Q3)						
# of Queries SELECT	1 thread	2 thread	3 thread	4 thread	5 thread	8 thread
10	0.165681559	0.182046167	0.206376752	0.21626936	0.247142696	0.329317312
20	0.278640268	0.304084737	0.33710147	0.383722531	0.408172347	0.531220043
100	1.172097998	1.22960023	1.292266322	1.465302522	1.593620264	2.123713295
200	2.278763022	2.280469255	2.413858696	2.627194117	2.975096961	3.993331178
300	3.369195841	3.462544365	3.571253092	3.808663054	4.292605844	5.746197822
400	4.074873089	4.370496659	4.624436849	5.058164547	5.617614271	7.704788568
500	5.296120126	5.636197719	5.685440978	6.170138765	6.950837331	9.52311133
1000	9.948752656	10.63041106	11.464676608	11.777932896	13.411464688	18.990351428
1500	15.342193758	15.716467038	16.49899327	17.909828597	20.433719864	29.205272603
2000	20.867069002	21.412008088	22.697869056	24.076717267	27.816800156	37.004140417
2500	26.024958791	26.207579724	29.196219274	31.758625899	36.14122421	46.937380044

Πίνακας Ι.16: MongoDB Q4 with database size 1MB

MongoDB (tpch_1mb - Q4)						
# of Queries SELECT	1 thread	2 thread	3 thread	4 thread	5 thread	8 thread
10	0.122232764	0.155649179	0.165235844	0.181091758	0.201723063	0.242796606
20	0.187652492	0.218672959	0.246518944	0.258082527	0.279486768	0.360873616
100	0.676055586	0.787086596	0.847099566	0.928259706	0.98795958	1.225950097
200	1.159932818	1.368877267	1.574460709	1.725357062	1.808908526	2.262049892
300	1.852960798	1.98961708	2.207044372	2.370455817	2.658614923	3.136782217
400	2.393043457	2.692369464	2.881771157	3.135433264	3.434293834	4.138524575
500	3.083807275	3.104618071	3.492864911	3.875135839	4.194736222	5.032753331
1000	5.31552493	5.687405755	6.65893608	7.342118374	8.073350251	9.899416983
1500	8.173166777	8.319226005	9.640940742	10.988693453	12.237854255	14.430554353
2000	10.713299523	11.264980732	12.905643191	14.488637003	16.089449248	19.776111616
2500	13.161194426	13.793998963	16.223055403	17.842900156	19.758382021	23.417640004

Πίνακας Ι.17: MongoDB Q1 with database size 10MB

MongoDB (tpch_10mb - Q1)	
# of Queries SELECT	1 thread
10	11.795265088
20	23.615450745

Πίνακας Ι.18: MongoDB Q1 with database size 100MB

MongoDB (tpch_100mb - Q1)	
# of Queries SELECT	1 thread
10	116.792734069
20	256.39465321

Πίνακας Ι.19: MongoDB Q1 with database size 1GB

MongoDB (tpch_1gb - Q1)	
# of Queries SELECT	1 thread
10	1711.586637786
20	3401.788422923

Πίνακας Ι.20: MongoDB Q3 with database size 10MB

MongoDB (tpch_10mb - Q3)	
# of Queries SELECT	1 thread
10	1.462314931
20	2.894244921

Πίνακας Ι.21: MongoDB Q3 with database size 100MB

MongoDB (tpch_100mb - Q3)	
# of Queries SELECT	1 thread
10	12.300943969
20	25.024499759

Πίνακας Ι.22: MongoDB Q3 with database size 1GB

MongoDB (tpch_1gb - Q3)	
# of Queries SELECT	1 thread
10	844.927346447
20	1679.091444667

INSERT και DELETE

INSERT

Πίνακας I.23: INSERT with database size 1MB

tpch_1mb			
# of Queries INSERT	MySQL		MongoDB
	innodb_flush_log_at_trx_commit=2	innodb_flush_log_at_trx_commit=1	
100	0.047491172	5.2630526	0.09021467
200	0.086872824	10.225685089	0.133692343
300	0.121853008	13.820096158	0.170326618
400	0.143745389	18.023031646	0.208234819
500	0.216200238	25.290590813	0.245097359
1000	0.327306378	45.604672263	0.414553209

Πίνακας I.24: INSERT with database size 10MB

tpch_10mb			
# of Queries INSERT	MySQL		MongoDB
	innodb_flush_log_at_trx_commit=2	innodb_flush_log_at_trx_commit=1	
100	0.044362809	5.351671067	0.087926285
200	0.084726488	9.053210492	0.138692386
300	0.120398444	13.550457985	0.189588629
400	0.151014111	20.509801176	0.225187097
500	0.181147797	24.458190583	0.258290264
1000	0.332556033	48.171719191	0.425612332

Πίνακας I.25: INSERT with database size 100MB

tpch_100mb			
# of Queries INSERT	MySQL		MongoDB
	innodb_flush_log_at_trx_commit=2	innodb_flush_log_at_trx_commit=1	
100	0.047268246	4.660101758	0.095839155
200	0.086153957	9.034472339	0.151230469
300	0.120778609	15.148105391	0.180843172
400	0.152552428	20.189550859	0.236668023
500	0.311329559	22.932449537	0.315350449
1000	0.345012245	51.103047115	0.427774912

DELETE

Πίνακας I.26: DELETE with database size 1MB

tpch_1mb			
# of Queries DELETE	MySQL		MongoDB
	innodb_flush_log_at_trx_commit=2	innodb_flush_log_at_trx_commit=1	
100	0.178797009	5.359550271	0.189613671
200	0.345161034	10.904263482	0.348410508
300	0.941233175	14.118677023	0.468945253
400	1.158710622	20.400524021	0.610116928
500	1.386836408	26.098440786	0.866381146
1000	2.850867833	60.063773027	1.624821464

Πίνακας I.27: DELETE with database size 10MB

tpch_10mb			
# of Queries DELETE	MySQL		MongoDB
	innodb_flush_log_at_trx_commit=2	innodb_flush_log_at_trx_commit=1	
100	1.425348113	6.638161344	1.252320921
200	2.701229463	13.786565612	2.458056334
300	4.040984835	20.397017956	3.7499572
400	6.973454488	28.701222484	4.859459335
500	8.976713405	32.23790623	5.907328621
1000	18.718540557	75.128030088	11.625599449

Πίνακας I.28: DELETE with database size 100MB

tpch_100mb			
# of Queries DELETE	MySQL		MongoDB
	innodb_flush_log_at_trx_commit=2	innodb_flush_log_at_trx_commit=1	
100	12.124723138	18.139612419	10.849879238
200	23.887288626	36.518200543	21.489552213
300	35.835673926	56.622534721	32.063245065
400	47.737503689	74.340551422	42.883057125
500	59.607477116	89.673853827	53.551327112
1000	121.671345799	177.177507073	106.146287902

ΠΑΡΑΡΤΗΜΑ ΙΙ ΠΡΟΔΙΑΓΡΑΦΕΣ ΥΠΟΛΟΓΙΣΤΗ ΚΑΙ ΕΚΔΟΣΕΙΣ ΠΡΟΓΡΑΜΜΑΤΩΝ

Προδιαγραφές του υπολογιστή

- **CPU**: Intel® Core™ i7-4710MQ CPU @ 2.50GHz
- **Memory**: DDR3(L) 8GB RAM Dual Channel
- **HDD**: 500GB, Rotational Speed: 5400 RPM, Model: WDC WD5000LPVX-22V0TT0
- **OS**: Windows 10 pro 64-bit Operating System

Προγράμματα που χρησιμοποιήθηκαν και οι εκδόσεις τους

- **Java**: jre1.8.0_121
- **MySQL Server**: MySQL Server 5.7
- **MySQL Workbench**: MySQL Workbench 6.3 CE
- **MySQL-Java Driver**: mysql-connector-java-5.1.40
- **MongoDB**: MongoDB 3.4.2 2008 R2
- **MongoDB-Java Driver**: mongo-java-driver-3.4.2
- **TPC-H**: TPC-H 2.17.0

ΠΑΡΑΡΤΗΜΑ III ΓΕΝΝΗΤΡΙΑ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΦΟΡΤΩΣΗ ΑΥΤΩΝ ΣΤΙΣ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

Για τα δεδομένα, χρησιμοποιήθηκε το πρόγραμμα DBGen που είναι διαθέσιμο από το TPC-H, το οποίο δημιουργούσε αυτόματα τα δεδομένα με οποιοδήποτε μέγεθος βάσης δεδομένων επιθυμούσαμε, δηλώνοντας απλά μια παράμετρο -s κατά την εκτέλεση του. Για κάθε πίνακα του σχήματος TPC-H δημιουργούσε και από ένα αρχείο τύπου .tbl στο οποίο κάθε εγγραφή χωριζόταν με αλλαγή γραμμής και κάθε στήλη με την κάθετη γραμμή '|'. Το αρχείο αυτό είναι σαν τα κανονικά αρχεία .csv με μόνη διαφορά ότι αντί για κόμμα είχε την κάθετη γραμμή. Γενικά, πρώτα φορτώσαμε τα δεδομένα στην MySQL και ύστερα με το πρόγραμμα mysqlToMongo.jar που υλοποιήσαμε φορτώσαμε τη βάση MongoDB παίρνοντας τα δεδομένα από την MySQL και μετατρέποντάς τα σε κατάλληλες BSON εγγραφές.

Για την εγκατάσταση της MySQL βάσης δεδομένων αρχικά δημιουργήθηκε μια κενή βάση και ύστερα εκτελέστηκαν SQL εντολές για την δημιουργία των πινάκων και τις συσχετίσεις τους με ξένα κλειδιά. Επειδή ο κώδικας αυτός όπως δίνετε από το TPC-H είναι σε δύο αρχεία και χρειάζεται μια επεξεργασία πριν την εκτέλεση, παρακάτω παρουσιάζουμε συνεχόμενες SQL εντολές οι οποίες με μια απλή εκτέλεση όλων μαζί δημιουργούνται όλοι οι πίνακες και οι περιορισμοί τους. Αυτός ο κώδικας προήλθε αυτόματα από το MySQL Workbench με το Copy to Clipboard -> Create Statement που παρέχει για κάθε πίνακα, αφού δημιουργήσαμε σταδιακά τους πίνακες αυτούς. Και εδώ η στοίχιση είναι σε δύο στήλες για εξοικονόμηση χώρου.

```
use 'database_name';
CREATE TABLE `region` (
  `R_REGIONKEY` int(11) NOT NULL,
  `R_NAME` char(25) NOT NULL,
  `R_COMMENT` varchar(152) DEFAULT NULL,
  PRIMARY KEY (`R_REGIONKEY`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


CREATE TABLE `nation` (
  `N_NATIONKEY` int(11) NOT NULL,
  `N_NAME` char(25) NOT NULL,
  `N_REGIONKEY` int(11) NOT NULL,
  `N_COMMENT` varchar(152) DEFAULT NULL,
  PRIMARY KEY (`N_NATIONKEY`),
  KEY `NATION_FK1_idx` (`N_REGIONKEY`),
  CONSTRAINT `NATION_FK1` FOREIGN KEY (`N_REGIONKEY`) REFERENCES `region`
(`R_REGIONKEY`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `supplier` (
  `S_SUPPKEY` int(11) NOT NULL,
  `S_NAME` char(25) NOT NULL,
```

```
 `S_ADDRESS` varchar(40) NOT NULL,
 `S_NATIONKEY` int(11) NOT NULL, `S_PHONE` char(15) NOT NULL,
 `S_ACCTBAL` decimal(15,2) NOT NULL,
 `S_COMMENT` varchar(101) NOT NULL,
 PRIMARY KEY (`S_SUPPKEY`),
 KEY `SUPPLIER_FK1_idx` (`S_NATIONKEY`),
 CONSTRAINT `SUPPLIER_FK1` FOREIGN KEY (`S_NATIONKEY`) REFERENCES `nation`
(`N_NATIONKEY`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `customer` (
 `C_CUSTKEY` int(11) NOT NULL,
 `C_NAME` varchar(25) NOT NULL,
 `C_ADDRESS` varchar(40) NOT NULL,
 `C_NATIONKEY` int(11) NOT NULL,
 `C_PHONE` char(15) NOT NULL,
 `C_ACCTBAL` decimal(15,2) NOT NULL,
 `C_MKTSEGMENT` char(10) NOT NULL,
 `C_COMMENT` varchar(117) NOT NULL,
 PRIMARY KEY (`C_CUSTKEY`),
 KEY `CUSTOMER_FK1_idx` (`C_NATIONKEY`),
 CONSTRAINT `CUSTOMER_FK1` FOREIGN KEY (`C_NATIONKEY`) REFERENCES `nation`
(`N_NATIONKEY`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `part` (
 `P_PARTKEY` int(11) NOT NULL,
 `P_NAME` varchar(55) NOT NULL,
 `P_MFGR` char(25) NOT NULL,
 `P_BRAND` char(10) NOT NULL,
 `P_TYPE` varchar(25) NOT NULL,
 `P_SIZE` int(11) NOT NULL,
 `P_CONTAINER` char(10) NOT NULL,
 `P_RETAILPRICE` decimal(15,2) NOT NULL,
 `P_COMMENT` varchar(23) NOT NULL,
 PRIMARY KEY (`P_PARTKEY`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `partsupp` (
 `PS_PARTKEY` int(11) NOT NULL,
 `PS_SUPPKEY` int(11) NOT NULL,
 `PS_AVAILQTY` int(11) NOT NULL,
```

```
  `PS_SUPPLYCOST` decimal(15,2) NOT NULL,
  `PS_COMMENT` varchar(199) NOT NULL,
  PRIMARY KEY (`PS_PARTKEY`,`PS_SUPPKEY`),
  KEY `PARTSUPP_FK1_idx` (`PS_SUPPKEY`),
  CONSTRAINT `PARTSUPP_FK1` FOREIGN KEY (`PS_SUPPKEY`) REFERENCES `supplier`
(`S_SUPPKEY`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `PARTSUPP_FK2` FOREIGN KEY (`PS_PARTKEY`) REFERENCES `part`
(`P_PARTKEY`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


CREATE TABLE `orders` (
  `O_ORDERKEY` int(11) NOT NULL,
  `O_CUSTKEY` int(11) NOT NULL,
  `O_ORDERSTATUS` char(1) NOT NULL,
  `O_TOTALPRICE` decimal(15,2) NOT NULL,
  `O_ORDERDATE` date NOT NULL,
  `O_ORDERPRIORITY` char(15) NOT NULL,
  `O_CLERK` char(15) NOT NULL,
  `O_SHIPPRIORITY` int(11) NOT NULL,
  `O_COMMENT` varchar(79) NOT NULL,
  PRIMARY KEY (`O_ORDERKEY`),
  KEY `ORDERS_FK1_idx` (`O_CUSTKEY`),
  CONSTRAINT `ORDERS_FK1` FOREIGN KEY (`O_CUSTKEY`) REFERENCES `customer`
(`C_CUSTKEY`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


CREATE TABLE `lineitem` (
  `L_ORDERKEY` int(11) NOT NULL,
  `L_PARTKEY` int(11) NOT NULL,
  `L_SUPPKEY` int(11) NOT NULL,
  `L_LINENUMBER` int(11) NOT NULL,
  `L_QUANTITY` decimal(15,2) NOT NULL,
  `L_EXTENDEDPRICE` decimal(15,2) NOT NULL,
  `L_DISCOUNT` decimal(15,2) NOT NULL,
  `L_TAX` decimal(15,2) NOT NULL,
  `L_RETURNFLAG` char(1) NOT NULL,
  `L_LINESTATUS` char(1) NOT NULL,
  `L_SHIPDATE` date NOT NULL,
  `L_COMMITDATE` date NOT NULL,
  `L_RECEIPTDATE` date NOT NULL,
  `L_SHIPINSTRUCT` char(25) NOT NULL,
  `L_SHIPMODE` char(10) NOT NULL,
  `L_COMMENT` varchar(44) NOT NULL,
```

```
 PRIMARY KEY (`L_ORDERKEY`,`L_LINENUMBER`),
 KEY `LINEITEM_FK2_idx` (`L_PARTKEY`,`L_SUPPKEY`),
 CONSTRAINT `LINEITEM_FK1` FOREIGN KEY (`L_ORDERKEY`) REFERENCES `orders`
(`O_ORDERKEY`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 CONSTRAINT `LINEITEM_FK2` FOREIGN KEY (`L_PARTKEY`, `L_SUPPKEY`) REFERENCES
`partsupp` (`PS_PARTKEY`, `PS_SUPPKEY`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Στη συνέχεια, για την φόρτωση των δεδομένων στη βάση MySQL χρησιμοποιήθηκε η SQL εντολή LOAD DATA INFILE. Ένα παράδειγμα σύνταξης της εντολής αυτής είναι:

```
LOAD DATA INFILE 'path to region.tbl file' INTO TABLE region FIELDS TERMINATED BY '|';
```

Όμως όσο αυξανόταν το μέγεθος των δεδομένων η ταχύτητα της εντολής αυτής μειωνόταν αρκετά. Για το λόγο αυτό απενεργοποιήθηκαν κάποιες παράμετροι της MySQL βάσης δεδομένων που είχαν ως αποτέλεσμα την ταχύτερη φόρτωση των δεδομένων. Οι παράμετροι αυτοί είναι οι εξής:

```
set autocommit = 0;
set unique_checks = 0;
set foreign_key_checks = 0;
set sql_log_bin = 0;
```

Αξίζει να σημειωθεί ότι η παράμετρος -s του προγράμματος DBGen όταν είχε την τιμή 0,001, δηλαδή δημιουργούσε βάση δεδομένων 1MB, το αρχείο partsupp.tbl περιείχε 100 διπλότυπες εγγραφές αφού έβγαζε σφάλμα η MySQL στην εκτέλεση της εντολής LOAD DATA INFILE. Για το λόγο αυτό χρησιμοποιήθηκε το IGNORE στην παραπάνω εντολή, δηλαδή 'LOAD DATA INFILE 'path to region.tbl file' **IGNORE** INTO TABLE partsupp FIELDS TERMINATED BY '|';', ώστε να αγνοήσει τη διπλότυπη εγγραφή κατά την εισαγωγή και να μην εμφανιστεί σφάλμα.

Οι χρόνοι εκτέλεσης των εντολών LOAD DATA INFILE φαίνονται παρακάτω στον πίνακα III.1. Αρχικά υλοποιήθηκαν και βάσεις πάνω από 1GB (συγκεκριμένα 5GB) στη MySQL αλλά η μετατροπή σε βάση MongoDB, με το πρόγραμμα mysqlToMongo.jar, απαιτούσε μνήμη RAM περισσότερη από ότι διέθετε το σύστημα μας με αποτέλεσμα ο χρόνος εκτέλεσης να αυξανόταν εκθετικά.

Πίνακας III.1: Χρόνοι εκτέλεσης των εντολών SQL (LOAD DATA INFILE)

Tables	1 MB	10 MB	100 MB	1 GB
region	0 sec	0 sec	0 sec	0 sec
nation	0 sec	0 sec	0 sec	0 sec
suppliers	0 sec	0 sec	0.34 sec	1.42 sec
part	0.03 sec	0.59 sec	1.89 sec	4.09 sec
customer	0.06 sec	0.7 sec	2.05 sec	6.10 sec
partsupp	0.25 sec	1.72 sec	3.56 sec	34.72 sec
orders	0.44 sec	2.13 sec	4.41 sec	33.51 sec
lineitem	2 sec	3.72 sec	25.76 sec	6 min 22.14 sec

Στον πίνακα ΙΙΙ.2 φαίνονται οι χρόνοι εκτέλεσης του προγράμματος mysqlToMongo.jar, το οποίο έκανε την μετατροπή της βάσης MySQL σε βάση MongoDB. Ο κώδικας του προγράμματος παρουσιάζεται στο ΠΑΡΑΡΤΗΜΑ Δ.

Πίνακας ΙΙΙ.2: Χρόνοι εκτέλεσης του προγράμματος mysqlToMongo.jar

Document	1 MB	10 MB	100 MB	1 GB
order	9.43 sec	59.81 sec	11 min 39.3 sec	1 hour 35 min

ΠΑΡΑΡΤΗΜΑ IV ΚΩΔΙΚΑΣ ΚΑΙ ΠΛΗΡΟΦΟΡΙΕΣ ΕΚΤΕΛΕΣΗΣ ΤΩΝ ΠΡΟΓΡΑΜΜΑΤΩΝ

mysqlToMongo.jar

Το εκτελέσιμο JAR αυτό αρχείο μετατρέπει την MySQL βάση που ακολουθεί το TPC-H schema, όπως παρουσιάστηκε στην υπο-ενότητα 3.2.1, σε βάση MongoDB που ακολουθεί ένα μη κανονικοποιημένο schema, όπως αναφέρθηκε στην υπο-ενότητα 3.2.2.

Η εκτέλεση του προγράμματος αυτού παίρνει μόνο μια παράμετρο η οποία είναι το όνομα της βάσης δεδομένων που θέλουμε να μετατρέψουμε.

Οι βιβλιοθήκες που περιέχει, εκτός των Java System Libraries, είναι οι εξής:

- mongo-java-driver-3.4.2.jar
- mysql-connector-java-5.1.40-bin.jar

Τα πηγαία αρχεία που περιέχει είναι τα εξής:

- MysqlToMongo.java

Παρακάτω φαίνεται ο κώδικας του πηγαίου αρχείου.

MysqlToMongo.java

```
import java.sql.*;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;

import org.bson.Document;
import org.bson.types.ObjectId;
import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

public class MysqlToMongo  {
	//private static final String SQL_DRIVER = "com.mysql.jdbc.Driver";
	private static String SQL_CONNECTION = "jdbc:mysql://localhost:3306/";
	private static String SQL_USER = "root";
	private static String SQL_PASSWORD = "1993";

	private static Connection conn = null;   // mysql connection
	private static MongoClient mongoClient = null;  // mongodb connection

	public static void main(String[] args) {
		PreparedStatement ps = null;
		ResultSet rs = null;
		String selectSQL = null;

		if(args.length != 1){
			System.err.println("Wrong arguments.");
			return;
		}
		String database = args[0];
		try {
				// SQL CONNECTION
				//Class.forName(SQL_DRIVER);
				SQL_CONNECTION += database +"?useSSL=false";
```

```
                                conn =
DriverManager.getConnection(SQL_CONNECTION,SQL_USER,SQL_PASSWORD);


                                // disable mongoDB java driver logging

        java.util.logging.Logger.getLogger("org.mongodb.driver").setLevel(Level.SEVERE);
                                // mongoDB CONNECTION
                                mongoClient = new MongoClient("localhost", 27017);
                                MongoDatabase db = mongoClient.getDatabase(database);
                                MongoCollection<Document> collection = db.getCollection("order");

                                selectSQL = "SELECT O_ORDERKEY FROM orders";
                                ps = conn.prepareStatement(selectSQL);
                                rs = ps.executeQuery();

                                // save orderKeys
                                List<Integer> orderKeyList = new ArrayList<Integer>();
                                while (rs.next()) {
                                        orderKeyList.add(rs.getInt(1));
                                }
                                rs.close();
                                ps.close();

                                long startTime = System.nanoTime();
                                Document obj;
                                for (int orderKey : orderKeyList) {
                                        // dhmiourgia twn documents
                                        obj = orderDoc(orderKey);
                                        // eisagwgh sth mongoDB
                                        collection.insertOne(obj);
                                }
                                long elapsedTime = System.nanoTime() - startTime;
                                System.out.println("Elapsed time: " + elapsedTime*Math.pow(10,-9) +"
seconds.");

                }
                catch(Exception e){ System.out.println(e);}
                finally {
                        mongoClient.close();
                   try { rs.close(); } catch (Exception e) {}
                   try { ps.close(); } catch (Exception e) {}
                   try { conn.close(); } catch (Exception e) {}
                }
        }


        // region document for mongoDB
        private static Document regionDoc(int regionKey) throws Exception{
                PreparedStatement ps = null;
                ResultSet rs = null;
                String selectSQL = null;
                Document obj = null;

                try{
                        // SQL SELECT
                        selectSQL = "SELECT * FROM region WHERE R_REGIONKEY = ?";
                        ps = conn.prepareStatement(selectSQL);
                        ps.setInt(1, regionKey);
                        rs = ps.executeQuery();
                        rs.first();          // afou mono ena query tha uparxei

                        // create document
```

```
                obj = new Document();
                obj.put("_id", new ObjectId().toString());
                obj.put("name", rs.getString(2));
                obj.put("comment", rs.getString(3));
            }
            catch(Exception e){ throw new Exception();}
            finally {
               try { rs.close(); } catch (Exception e) {}
               try { ps.close(); } catch (Exception e) {}
            }

            return obj;
        }


        // nation document for mongoDB
        private static Document nationDoc(int nationKey) throws Exception{
            PreparedStatement ps = null;
            ResultSet rs = null;
            String selectSQL = null;
            Document obj = null;

            try{
                // SQL SELECT
                selectSQL = "SELECT * FROM nation WHERE N_NATIONKEY = ?";
                ps = conn.prepareStatement(selectSQL);
                ps.setInt(1, nationKey);
                rs = ps.executeQuery();
                rs.first();          // afou mono ena query tha uparxei

                // create document
                obj = new Document();
                obj.put("_id", new ObjectId().toString());
                obj.put("name", rs.getString(2));
                obj.put("comment", rs.getString(4));
                obj.put("region", regionDoc(rs.getInt(3)));
            }
            catch(Exception e){ throw new Exception();}
            finally {
               try { rs.close(); } catch (Exception e) {}
               try { ps.close(); } catch (Exception e) {}
            }

            return obj;
        }


        // customer document for mongoDB
        private static Document customerDoc(int custKey) throws Exception{
            PreparedStatement ps = null;
            ResultSet rs = null;
            String selectSQL = null;
            Document obj = null;

            try{
                // SQL SELECT
                selectSQL = "SELECT * FROM customer WHERE C_CUSTKEY = ?";
                ps = conn.prepareStatement(selectSQL);
                ps.setInt(1, custKey);
                rs = ps.executeQuery();
                rs.first();          // afou mono ena query tha uparxei

                // create document
```

```
                obj = new Document();
                obj.put("_id", new ObjectId().toString());
                obj.put("name", rs.getString(2));
                obj.put("address", rs.getString(3));
                obj.put("phone", rs.getString(5));
                obj.put("acctbal", rs.getDouble(6));
                obj.put("mktsegment", rs.getString(7));
                obj.put("comment", rs.getString(8));
                obj.put("nation", nationDoc(rs.getInt(4)));
            }
            catch(Exception e){ throw new Exception();}
            finally {
               try { rs.close(); } catch (Exception e) {}
               try { ps.close(); } catch (Exception e) {}
            }

            return obj;
        }


        // supplier document for mongoDB
        private static Document supplierDoc(int suppKey) throws Exception{
            PreparedStatement ps = null;
            ResultSet rs = null;
            String selectSQL = null;
            Document obj = null;

            try{
                // SQL SELECT
                selectSQL = "SELECT * FROM supplier WHERE S_SUPPKEY = ?";
                ps = conn.prepareStatement(selectSQL);
                ps.setInt(1, suppKey);
                rs = ps.executeQuery();
                rs.first();          // afou mono ena query tha uparxei

                // create document
                obj = new Document();
                obj.put("_id", new ObjectId().toString());
                obj.put("name", rs.getString(2));
                obj.put("address", rs.getString(3));
                obj.put("phone", rs.getString(5));
                obj.put("acctbal", rs.getDouble(6));
                obj.put("comment", rs.getString(7));
                obj.put("nation", nationDoc(rs.getInt(4)));
            }
            catch(Exception e){ throw new Exception();}
            finally {
               try { rs.close(); } catch (Exception e) {}
               try { ps.close(); } catch (Exception e) {}
            }

            return obj;
        }


        // part document for mongoDB
        private static Document partDoc(int partKey) throws Exception{
            PreparedStatement ps = null;
            ResultSet rs = null;
            String selectSQL = null;
            Document obj = null;

            try{
```

```
                        // SQL SELECT
                        selectSQL = "SELECT * FROM part WHERE P_PARTKEY = ?";
                        ps = conn.prepareStatement(selectSQL);
                        ps.setInt(1, partKey);
                        rs = ps.executeQuery();
                        rs.first();          // afou mono ena query tha uparxei

                        // create document
                        obj = new Document();
                        obj.put("_id", new ObjectId().toString());
                        obj.put("name", rs.getString(2));
                        obj.put("mfgr", rs.getString(3));
                        obj.put("brand", rs.getString(4));
                        obj.put("type", rs.getString(5));
                        obj.put("size", rs.getDouble(6));
                        obj.put("container", rs.getString(7));
                        obj.put("retailprice", rs.getDouble(8));
                        obj.put("comment", rs.getString(9));
                }
                catch(Exception e){ throw new Exception();}
                finally {
                   try { rs.close(); } catch (Exception e) {}
                   try { ps.close(); } catch (Exception e) {}
                }

                return obj;
        }


        // partsupp document for mongoDB
        private static Document partsuppDoc(int partKey, int suppKey) throws Exception{
                PreparedStatement ps = null;
                ResultSet rs = null;
                String selectSQL = null;
                Document obj = null;

                try{
                        // SQL SELECT
                        selectSQL = "SELECT * FROM partsupp WHERE PS_PARTKEY = ? AND
PS_SUPPKEY = ?";
                        ps = conn.prepareStatement(selectSQL);
                        ps.setInt(1, partKey);
                        ps.setInt(2, suppKey);
                        rs = ps.executeQuery();
                        rs.first();          // afou mono ena query tha uparxei

                        // create document
                        obj = new Document();
                        obj.put("_id", new ObjectId().toString());
                        obj.put("availqty", rs.getDouble(3));
                        obj.put("supplycost", rs.getDouble(4));
                        obj.put("comment", rs.getString(5));
                        obj.put("part", partDoc(partKey));
                        obj.put("supplier", supplierDoc(suppKey));
                }
                catch(Exception e){ throw new Exception();}
                finally {
                   try { rs.close(); } catch (Exception e) {}
                   try { ps.close(); } catch (Exception e) {}
                }

                return obj;
        }
```

```
	// lineitem document for mongoDB
	private static Document lineitemDoc(int orderKey, int linenumber) throws Exception{
		SimpleDateFormat inputDateFormat = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
		PreparedStatement ps = null;
		ResultSet rs = null;
		String selectSQL = null;
		Document obj = null;

		try{
			// SQL SELECT
			selectSQL = "SELECT * FROM lineitem WHERE L_ORDERKEY = ? AND
L_LINENUMBER = ?";
			ps = conn.prepareStatement(selectSQL);
			ps.setInt(1, orderKey);
			ps.setInt(2, linenumber);
			rs = ps.executeQuery();
			rs.first();		// afou mono ena query tha uparxei

			// create document
			obj = new Document();
			obj.put("_id", new ObjectId().toString());
			obj.put("quantity", rs.getDouble(5));
			obj.put("extendedprice", rs.getDouble(6));
			obj.put("discount", rs.getDouble(7));
			obj.put("tax", rs.getDouble(8));
			obj.put("returnflag", rs.getString(9));
			obj.put("linestatus", rs.getString(10));
			obj.put("shipdate", inputDateFormat.parse(rs.getString(11) + " 03:00:00"));
			obj.put("commitdate", inputDateFormat.parse(rs.getString(12) + " 03:00:00"));
			obj.put("receiptdate", inputDateFormat.parse(rs.getString(13) + " 03:00:00"));
			obj.put("shipinstruct", rs.getString(14));
			obj.put("shipmode", rs.getString(15));
			obj.put("comment", rs.getString(16));
			obj.put("partsupp", partsuppDoc(rs.getInt(2),rs.getInt(3)));
		}
		catch(Exception e){ throw new Exception();}
		finally {
		   try { rs.close(); } catch (Exception e) {}
		   try { ps.close(); } catch (Exception e) {}
		}

		return obj;
	}


	// lineitems Array document for mongoDB
	private static List<Document> lineitemArrayDoc(int orderKey) throws Exception{
		PreparedStatement ps = null;
		ResultSet rs = null;
		String selectSQL = null;
		List<Document> obj = null;

		try{
			// SQL SELECT
			selectSQL = "SELECT L_LINENUMBER FROM lineitem WHERE L_ORDERKEY
= ?";
			ps = conn.prepareStatement(selectSQL);
			ps.setInt(1, orderKey);
			rs = ps.executeQuery();
```

```
                        List<Integer> linenumberList = new ArrayList<Integer>();
                        while (rs.next()) {
                                linenumberList.add(rs.getInt(1));
                        }

                        obj = new ArrayList<Document>();
                        for(int linenumberKey : linenumberList)
                                obj.add(lineitemDoc(orderKey,linenumberKey));
                }
                catch(Exception e){ throw new Exception();}
                finally {
                   try{ rs.close(); } catch (Exception e) {}
                   try{ ps.close(); } catch (Exception e) {}
                }

                return obj;
        }


        // order document for mongoDB
        private static Document orderDoc(int orderKey) throws Exception{
                SimpleDateFormat inputDateFormat = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
                PreparedStatement ps = null;
                ResultSet rs = null;
                String selectSQL = null;
                Document obj = null;

                try{
                        // SQL SELECT
                        selectSQL = "SELECT * FROM orders WHERE O_ORDERKEY = ?";
                        ps = conn.prepareStatement(selectSQL);
                        ps.setInt(1, orderKey);
                        rs = ps.executeQuery();
                        rs.first();          // afou mono ena query tha uparxei

                        // create document
                        obj = new Document();
                        obj.put("_id", new ObjectId().toString());
                        obj.put("orderstatus", rs.getString(3));
                        obj.put("totalprice", rs.getDouble(4));
                        obj.put("orderdate", inputDateFormat.parse(rs.getString(5) + " 03:00:00"));
                        obj.put("orderpriority", rs.getString(6));
                        obj.put("clerk", rs.getString(7));
                        obj.put("shippriority", rs.getString(8));
                        obj.put("comment", rs.getString(9));
                        obj.put("customer", customerDoc(rs.getInt(2)));
                        obj.put("LineItems", lineitemArrayDoc(orderKey));
                }
                catch(Exception e){ throw new Exception();}
                finally {
                   try{ rs.close(); } catch (Exception e) {}
                   try{ ps.close(); } catch (Exception e) {}
                }

                return obj;
        }
}
```

results.jar

Το εκτελέσιμο JAR αυτό αρχείο, παίρνοντας κατάλληλες παραμέτρους, εκτελείται και επιστρέφει το χρόνο εκτέλεσης.

Οι παράμετροι που χρησιμοποιεί το πρόγραμμα αυτό φαίνονται παρακάτω ή με τις παραμέτρους -h, -help από τη γραμμή εντολών (java -jar results.jar -help).

USAGE:

java -jar results.jar -{database,db} <{mysql,mongo}> -{database_name,dbn} <name of the database> -[{query,q} <{Q1,Q3,Q4,Q5}>] -{#query,#q} <number> -{#thread,#t} <number> -{type} <{select,insert,delete}>

Εξήγηση παραμέτρων:

- **-{database,db} <{mysql,mongo}>**: Η βάση δεδομένων στην οποία επιθυμούμε να μετρήσουμε το χρόνο εκτέλεσης.
- **-{database_name,dbn} <name of the database>**: Το όνομα της βάσης δεδομένων προς εξέταση.
- **-[{query,q} <{Q1,Q3,Q4,Q5}>]**: Το ερώτημα που θα εξεταστεί όταν η παράμετρος type έχει την τιμή select οπότε στις άλλες περιπτώσεις δεν χρειάζεται να εισαχθεί. Εδώ βλέπουμε και το ερώτημα Q5 το οποίο είχε υλοποιηθεί αλλά δεν συμπεριλήφθηκε στις μετρήσεις, όπως είχαμε αναφέρει στην ενότητα 3.3.
- **-{#query,#q} <number>**: Ο αριθμός των ερωτημάτων που θα εκτελεστούν σε κάθε ένα thread.
- **-{#thread,#t} <number>**: Ο αριθμός των νημάτων που θα δημιουργηθούν για την μέτρηση του χρόνου εκτέλεσης. Όταν τελειώσουν όλα ο χρόνος εκτέλεσης που θα επιστραφεί θα είναι και ο μέσος όρος των χρόνων εκτέλεσης όλων των νημάτων.
- **-{type} <{select,insert,delete}>**: Ο τύπος των SQL ερωτημάτων προς εξέταση. Με τον τύπο select θα χρησιμοποιηθεί το ερώτημα που θα δοθεί στην παράμετρο -query. Οι άλλοι δύο τύποι χρησιμοποιούν απλά ερωτήματα για εισαγωγή και διαγραφή όπως αναφέρθηκαν στην ενότητα 4.4.

Οι βιβλιοθήκες που περιέχει, εκτός των Java System Libraries, είναι οι εξής:

- mongo-java-driver-3.4.2.jar
- mysql-connector-java-5.1.40-bin.jar

Τα πηγαία αρχεία που περιέχει είναι τα εξής:

- ElapsedTime.java (main function)
- WorkerThread.java
- MysqlQueries.java
- MongoQueries.java

Παρακάτω φαίνεται ο κώδικας του κάθε πηγαίου αρχείου.

ElapsedTime.java

```
public class ElapsedTime {
        private static double sumElpsedTimes = 0;
```

```
	public static synchronized void saveElapsedTime(double elapsedTime) {
		sumElpsedTimes += elapsedTime;
	}

	public static void main(String[] args) {
		String database = null;		// {mysql,mongo}
		String database_name = null;	// {tpch_10mb etc..}
		String query = null;			// {Q1, Q3, Q4, Q5}
		String type = null;				// {select, insert, delete}
	  int numberOfqueries = 0;
	  int numberOfthreads = 0;

	  for(int i=0; i<args.length; i++){
		switch(args[i]){
			case "-db":
			case "-database":
				try{
					if(!args[++i].startsWith("-"))
						database = new String(args[i]);
					else{
						System.err.println("database is missing!");
return; }
				}
				catch(ArrayIndexOutOfBoundsException
e){System.err.println("database is missing!"); return;}
				if(!database.equals("mysql") && !database.equals("mongo")){
					System.err.println("Wrong database!");
					return;
				}
				break;
			case "-dbn":
			case "-database_name":
				try{
					if(!args[++i].startsWith("-"))
						database_name = new String(args[i]);
					else{
						System.err.println("database_name is
missing!"); return; }
				}
				catch(ArrayIndexOutOfBoundsException
e){System.err.println("database_name is missing!"); return;}
				break;
			case "-q":
			case "-query":
				try{
					if(!args[++i].startsWith("-"))
							query = new String(args[i]);
					else{
						System.err.println("query is missing!"); return; }
				}
				catch(ArrayIndexOutOfBoundsException
e){System.err.println("query is missing!"); return;}
				break;
			case "-#q":
			case "-#query":
				try{
					if(!args[++i].startsWith("-"))
						numberOfqueries = Integer.valueOf(args[i]);
					else{
						System.err.println("#query is missing!"); return;
}
				}
```

```
                                        catch(ArrayIndexOutOfBoundsException
e){System.err.println("#query is missing!"); return;}
                                        if(numberOfqueries <= 0){
                                                System.err.println("#query must be positive!");
                                                return;
                                        }
                                        break;
                                case "-#t":
                                case "-#thread":
                                        try{
                                                if(!args[++i].startsWith("-"))
                                                        numberOfthreads = Integer.valueOf(args[i]);
                                                else{
                                                        System.err.println("#thread is missing!"); return;
}
                                        }
                                        catch(ArrayIndexOutOfBoundsException
e){System.err.println("#thread is missing!"); return;}
                                        if(numberOfthreads <= 0){
                                                System.err.println("#thread must be positive!");
                                                return;
                                        }
                                        break;
                                case "-type":
                                        try{
                                        if(!args[++i].startsWith("-"))
                                                type = new String(args[i]);
                                        else{
                                                System.err.println("type is missing!"); return; }
                                        }
                                        catch(ArrayIndexOutOfBoundsException e){System.err.println("type is
missing!"); return;}
                                        if(!type.equals("select") && !type.equals("insert") &&
!type.equals("delete")){
                                                System.err.println("Wrong type!");
                                                return;
                                        }
                                        break;
                                case "-h":
                                case "-help":
                                        String help = "USAGE:\n"
                                                        + "java -jar results.jar -{database,db}
<{mysql,mongo}>  -{database_name,dbn} <name of the database>\n"
                                                        + "                      -[{query,q}
<{Q1,Q3,Q4,Q5}>]  -{#query,#q} <number>  -{#thread,#t} <number>\n"
                                                        + "                      -{type}
<{select,insert,delete}>";
                                        System.out.println(help);
                                        return;
                                default: break;
                        }
                }
                if(database==null || database_name==null || type==null || numberOfqueries==0 ||
numberOfthreads==0){
                        System.err.println("At least one parameter is missing. Type -help or -h for help!");
                        return;
                }
                if(type.equals("select")){
                        if(query != null){
                                if(!query.equals("Q1") && !query.equals("Q3") && !query.equals("Q4") &&
!query.equals("Q5")){
                                        System.err.println("Wrong query!");
                                        return;
```

```
                                        }
                        }
                        else{
                                System.err.println("-query is missing for the type select!");
                                return;
                        }
                }

                        try {
                                WorkerThread[] threads = new WorkerThread[numberOfthreads];
                           for(int i = 0; i < numberOfthreads; i++){
                                threads[i] = new WorkerThread(database, database_name, query,
numberOfqueries, type);
                                threads[i].start();
                           }
                        for(int i = 0; i < numberOfthreads; i++)
                                threads[i].join();
                        String result = "#query = " + numberOfqueries +"\n#thread = " + numberOfthreads + "\nTime
= " + sumElpsedTimes/numberOfthreads;
                        System.out.println(result);
                           }
                           catch(Exception e){ System.out.println(e);}
                }
}
```

WorkerThread.java

```
import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.List;
import java.util.logging.Level;

import org.bson.Document;
import org.bson.conversions.Bson;
import org.bson.types.ObjectId;

import com.mongodb.BasicDBObject;
import com.mongodb.MongoClient;
import com.mongodb.client.AggregateIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;

public class WorkerThread extends Thread {
        private String SQL_CONNECTION = "jdbc:mysql://localhost:3306/";
        private String SQL_USER = "root";
        private String SQL_PASSWORD = "1993";

        private Connection conn = null;                 // mysql connection
        private MongoClient mongoClient = null;         // mongodb connection

        private String database = null;             // {mysql,mongo}
        private String database_name = null;    // {tpch_10mb etc..}
        private String query = null;                    // {Q1, Q3, Q4, Q5}
        String type = null;                             // {select, insert, delete}
   private int numberOfqueries;
```

```
        public WorkerThread(String database, String database_name, String query, int
numberOfqueries, String type) {
                this.database = new String(database);
                this.database_name = new String(database_name);
                if(query != null)
                        this.query = new String(query);
                this.numberOfqueries = numberOfqueries;
                this.type = type;
        }

    @Override
    public void run() {
        if(type.equals("select")){
                        // MYSQL SELECT
                        if(database.equals("mysql")){
                                PreparedStatement ps = null;
                                ResultSet rs = null;
                        try {
                                // SQL CONNECTION
                                SQL_CONNECTION += database_name +"?useSSL=false";
                                conn =
DriverManager.getConnection(SQL_CONNECTION,SQL_USER,SQL_PASSWORD);

                                if(query.equals("Q1")){
                                        ps = conn.prepareStatement(MysqlQueries.getQ1());
                                        ps.setString(1, "1998-12-01");
                                        long startTime = System.nanoTime();

                                        for(int i=0; i<numberOfqueries; i++)
                                                rs = ps.executeQuery();

                                        long elapsedTime = System.nanoTime() - startTime;
                                        ElapsedTime.saveElapsedTime(elapsedTime*Math.pow(10,-9));
                                }
                                else if(query.equals("Q3")){
                                        ps = conn.prepareStatement(MysqlQueries.getQ3());
                                        ps.setString(1, "BUILDING");
                                        ps.setString(2, "1995-03-15");
                                        ps.setString(3, "1995-03-15");
                                        long startTime = System.nanoTime();

                                        for(int i=0; i<numberOfqueries; i++)
                                                rs = ps.executeQuery();

                                        long elapsedTime = System.nanoTime() - startTime;
                                        ElapsedTime.saveElapsedTime(elapsedTime*Math.pow(10,-9));
                                }
                                else if(query.equals("Q4")){
                                        ps = conn.prepareStatement(MysqlQueries.getQ4());
                                        ps.setString(1, "1993-07-01");
                                        ps.setString(2, "1993-10-01");
                                        long startTime = System.nanoTime();

                                        for(int i=0; i<numberOfqueries; i++)
                                                rs = ps.executeQuery();

                                        long elapsedTime = System.nanoTime() - startTime;
                                        ElapsedTime.saveElapsedTime(elapsedTime*Math.pow(10,-9));
                                }
                                else if(query.equals("Q5")){
                                        ps = conn.prepareStatement(MysqlQueries.getQ5());
                                        ps.setString(1, "ASIA");
                                        ps.setString(2, "1994-01-01");
```

```
                                        ps.setString(3, "1995-01-01");
                                        long startTime = System.nanoTime();

                                        for(int i=0; i<numberOfqueries; i++)
                                                rs = ps.executeQuery();

                                        long elapsedTime = System.nanoTime() - startTime;
                                        ElapsedTime.saveElapsedTime(elapsedTime*Math.pow(10,-9));
                                }
                        }
                            catch (Exception e) {System.out.println(e);}
                            finally {
                                    try { rs.close(); } catch (Exception e) {}
                                    try { ps.close(); } catch (Exception e) {}
                                    try { conn.close(); } catch (Exception e) {}
                                }
                            }
                            // MONGO SELECT
                            else if(database.equals("mongo")){
                                    try{
                                        // disable mongoDB java driver logging

        java.util.logging.Logger.getLogger("org.mongodb.driver").setLevel(Level.SEVERE);
                                        // mongoDB CONNECTION
                                        mongoClient = new MongoClient("localhost", 27017);
                                        MongoDatabase db =
mongoClient.getDatabase(database_name);
                                        MongoCollection<Document> collection =
db.getCollection("order");

                                        if(query.equals("Q1")){
                                                long startTime = System.nanoTime();

                                                List<? extends Bson> aggregateQuery =
MongoQueries.getQ1();
                                                for(int i=0; i<numberOfqueries; i++){
                                                        AggregateIterable<Document> iterable =
collection.aggregate(aggregateQuery);
                                                        MongoCursor<Document> cursor =
iterable.iterator();
                                                }


                                                long elapsedTime = System.nanoTime() - startTime;
                                        ElapsedTime.saveElapsedTime(elapsedTime*Math.pow(10,-9));
                                        }
                                        else if(query.equals("Q3")){
                                                long startTime = System.nanoTime();

                                                List<? extends Bson> aggregateQuery =
MongoQueries.getQ3();
                                                for(int i=0; i<numberOfqueries; i++){
                                                        AggregateIterable<Document> iterable =
collection.aggregate(aggregateQuery);
                                                        MongoCursor<Document> cursor =
iterable.iterator();
                                                }

                                                long elapsedTime = System.nanoTime() - startTime;
                                        ElapsedTime.saveElapsedTime(elapsedTime*Math.pow(10,-9));
                                        }
                                        else if(query.equals("Q4")){
                                                long startTime = System.nanoTime();
```

```
                                        List<? extends Bson> aggregateQuery =
MongoQueries.getQ4();
                                        for(int i=0; i<numberOfqueries; i++){
                                                AggregateIterable<Document> iterable =
collection.aggregate(aggregateQuery);
                                                MongoCursor<Document> cursor =
iterable.iterator();
                                        }

                                                long elapsedTime = System.nanoTime() - startTime;
                                        ElapsedTime.saveElapsedTime(elapsedTime*Math.pow(10,-9));
                                        }
                                        else if(query.equals("Q5")){
                                                long startTime = System.nanoTime();

                                                List<? extends Bson> aggregateQuery =
MongoQueries.getQ5();
                                                for(int i=0; i<numberOfqueries; i++){
                                                        AggregateIterable<Document> iterable =
collection.aggregate(aggregateQuery);
                                                        MongoCursor<Document> cursor =
iterable.iterator();
                                                }

                                                long elapsedTime = System.nanoTime() - startTime;
                                        ElapsedTime.saveElapsedTime(elapsedTime*Math.pow(10,-9));
                                        }
                                }
                                catch(Exception e){ System.out.println(e);}
                                finally {
                                        mongoClient.close();
                                }
                        }
        }
        else if(type.equals("insert")){
                // MYSQL INSERT
                if(database.equals("mysql")){
                        PreparedStatement ps = null;
                                ResultSet rs = null;
                                try {
                                        // SQL CONNECTION
                                SQL_CONNECTION += database_name +"?useSSL=false";
                                conn =
DriverManager.getConnection(SQL_CONNECTION,SQL_USER,SQL_PASSWORD);

                                ps = conn.prepareStatement("SELECT * FROM orders LIMIT 1;");
                                rs = ps.executeQuery();
                                        rs.first();          // afou mono ena query tha uparxei

                                        int custkey = rs.getInt(2);
                                        String orderstatus = "A";
                                        double totalprice = rs.getDouble(4);
                                        Date orderdate = rs.getDate(5);
                                        String orderpriority = rs.getString(6);
                                        String clerk = rs.getString(7);
                                        int shippriority = rs.getInt(8);
                                        String comment = rs.getString(9);

                                        rs.close();
                                        ps.close();
```

```
                                        String insertSQL = "INSERT INTO orders (O_CUSTKEY,
O_ORDERSTATUS, O_TOTALPRICE, O_ORDERDATE, O_ORDERPRIORITY, "
                                                                        + "O_CLERK,
O_SHIPPRIORITY, O_COMMENT) VALUES (?,?,?,?,?,?,?,?)";
                                        ps = conn.prepareStatement(insertSQL);

                                        ps.setInt(1, custkey);
                                        ps.setString(2, orderstatus);
                                        ps.setDouble(3, totalprice);
                                        ps.setDate(4, orderdate);
                                        ps.setString(5, orderpriority);
                                        ps.setString(6, clerk);
                                        ps.setInt(7, shippriority);
                                        ps.setString(8, comment);


                                        long startTime = System.nanoTime();

                                        for(int i=0; i<numberOfqueries; i++)
                                                ps.execute();

                                    long elapsedTime = System.nanoTime() - startTime;
                                    ElapsedTime.saveElapsedTime(elapsedTime*Math.pow(10,-9));

                                    }
                                    catch (Exception e) {System.out.println(e);}
                                finally {
                                        try{ rs.close(); } catch (Exception e) {}
                                        try{ ps.close(); } catch (Exception e) {}
                                        try{ conn.close(); } catch (Exception e) {}
                                    }
                        }
                                // MONGO INSERT
                                else if(database.equals("mongo")){
                                        try{
                                                // disable mongoDB java driver logging

        java.util.logging.Logger.getLogger("org.mongodb.driver").setLevel(Level.SEVERE);
                                                // mongoDB CONNECTION
                                                mongoClient = new MongoClient("localhost", 27017);
                                                MongoDatabase db =
mongoClient.getDatabase(database_name);
                                                MongoCollection<Document> collection =
db.getCollection("order");

                                                // create document
                                                Document obj = null;

                                                obj = new Document();
                                                obj.put("_id", new ObjectId().toString());
                                                obj.put("orderstatus", "A");
                                                obj.put("totalprice", 1234);
                                                obj.put("orderdate", "2017-03-13");
                                                obj.put("orderpriority", "1-URGENT");
                                                obj.put("clerk", "Clerk#1234");
                                                obj.put("shippriority", 0);
                                                obj.put("comment", "1234");
                                                obj.put("customer", null);
                                                obj.put("LineItems", null);


                                                long startTime = System.nanoTime();
```

```
                                        for(int i=0; i<numberOfqueries; i++){
                                                collection.insertOne(obj);
                                                obj.put("_id", new ObjectId().toString());
                                        }

                                                long elapsedTime = System.nanoTime() - startTime;
                                        ElapsedTime.saveElapsedTime(elapsedTime*Math.pow(10,-9));

                                        }
                                        catch(Exception e){ System.out.println(e);}
                                        finally {
                                                mongoClient.close();
                                        }

                                }
        }
        else if(type.equals("delete")){
                // MYSQL DELETE
                if(database.equals("mysql")){
                        PreparedStatement ps = null;
                                ResultSet rs = null;
                                try {
                                        // SQL CONNECTION
                                SQL_CONNECTION += database_name +"?useSSL=false";
                                conn =
DriverManager.getConnection(SQL_CONNECTION,SQL_USER,SQL_PASSWORD);

                                ps = conn.prepareStatement("DELETE FROM orders WHERE
O_ORDERSTATUS = 'A' LIMIT 1;");

                                long startTime = System.nanoTime();

                                for(int i=0; i<numberOfqueries; i++)
                                        ps.execute();

                                long elapsedTime = System.nanoTime() - startTime;
                                ElapsedTime.saveElapsedTime(elapsedTime*Math.pow(10,-9));
                                }
                                catch (Exception e) {System.out.println(e);}
                        finally {
                            try { rs.close(); } catch (Exception e) {}
                            try { ps.close(); } catch (Exception e) {}
                            try { conn.close(); } catch (Exception e) {}
                                }
                }
                        // MONGO DELETE
                        else if(database.equals("mongo")){
                                try{
                                        // disable mongoDB java driver logging

        java.util.logging.Logger.getLogger("org.mongodb.driver").setLevel(Level.SEVERE);
                                        // mongoDB CONNECTION
                                        mongoClient = new MongoClient("localhost", 27017);
                                        MongoDatabase db =
mongoClient.getDatabase(database_name);
                                        MongoCollection<Document> collection =
db.getCollection("order");

                                        long startTime = System.nanoTime();

                                        for(int i=0; i<numberOfqueries; i++)
```

```
                                        collection.deleteOne(new BasicDBObject("orderstatus",
"A"));

                                                long elapsedTime = System.nanoTime() - startTime;
                                        ElapsedTime.saveElapsedTime(elapsedTime*Math.pow(10,-9));
                                        }
                                        catch(Exception e){ System.out.println(e);}
                                        finally {
                                                mongoClient.close();
                                        }
                                }
        }
   }
}
```

MysqlQueries.java

```
public class MysqlQueries {
        // query Q1
        public static String getQ1(){
                String query =  "SELECT SQL_NO_CACHE "
                                +  "l_returnflag, "
                                +  "l_linestatus, "
                                +  "sum(l_quantity) as sum_qty, "
                                +  "sum(l_extendedprice) as sum_base_price, "
                                +  "sum(l_extendedprice * (1 - l_discount)) as sum_disc_price, "
                                +  "sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
"
                                +  "avg(l_quantity) as avg_qty, "
                                +  "avg(l_extendedprice) as avg_price, "
                                +  "avg(l_discount) as avg_disc, "
                                +  "count(*) as count_order "
                        + "FROM "
                                +  "lineitem "
                                + "WHERE "
                                +  "l_shipdate <= date ? "
                                + "GROUP BY "
                                +  "l_returnflag, "
                                +  "l_linestatus "
                                + "ORDER BY "
                                +  "l_returnflag, "
                                +  "l_linestatus;";
                return query;
        }

        // query Q3
        public static String getQ3(){
                String query =  "SELECT SQL_NO_CACHE "
                                +  "l_orderkey, "
                                +  "sum(l_extendedprice * (1 - l_discount)) as revenue, "
                                +  "o_orderdate, "
                                +  "o_shippriority "
                        + "FROM "
                                +  "customer, "
                                +  "orders, "
                                +  "lineitem "
                                + "WHERE "
                                +  "c_mktsegment = ? "
                                +  "AND c_custkey = o_custkey "
                                +  "AND l_orderkey = o_orderkey "
                                +  "AND o_orderdate < date ? "
```

```
                                +   "AND l_shipdate > date ? "
                                + "GROUP BY "
                                +   "l_orderkey, "
                                +   "o_orderdate, "
                                +   "o_shippriority "
                                + "ORDER BY "
                                +   "revenue desc;";
                return query;
        }

        // query Q4
        public static String getQ4(){
                String query =  "SELECT SQL_NO_CACHE "
                                +   "o_orderpriority, "
                                +   "count(*) as order_count "
                            + "FROM "
                                +   "orders "
                                + "WHERE "
                                +   "o_orderdate >= date ? "
                                +   "AND o_orderdate < date ? "
                                +   "AND exists ( "
                                +               "SELECT "
                                +                       "* "
                                +               "FROM "
                                +                       "lineitem "
                                +               "WHERE "
                                +                       "l_orderkey = o_orderkey "
                                +                       "AND l_commitdate < l_receiptdate "
                                +   ") "
                                + "GROUP BY "
                                +   "o_orderpriority "
                                + "ORDER BY "
                                +   "o_orderpriority;";
                return query;
        }

        // query Q5
                public static String getQ5(){
                        String query =  "SELECT SQL_NO_CACHE "
                                        +   "n_name, "
                                        +   "sum(l_extendedprice * (1 - l_discount)) as revenue "
                                    + "FROM "
                                        +   "customer, "
                                        +   "orders, "
                                        +   "lineitem, "
                                        +   "supplier, "
                                        +   "nation, "
                                        +   "region "
                                        + "WHERE "
                                        +   "c_custkey = o_custkey "
                                        +   "AND l_orderkey = o_orderkey "
                                        +   "AND l_suppkey = s_suppkey "
                                        +   "AND c_nationkey = s_nationkey "
                                        +   "AND s_nationkey = n_nationkey "
                                        +   "AND n_regionkey = r_regionkey "
                                        +   "AND r_name = ? "
                                        +   "AND o_orderdate >= date ? "
                                        +   "AND o_orderdate < date ? "
                                        + "GROUP BY "
                                        +   "n_name "
                                        + "ORDER BY "
                                        +   "revenue desc;";
                                return query;
```

```
                }
}
```

MongoQueries.java

```
import java.util.ArrayList;
import java.util.List;

import org.bson.BsonDocument;
import org.bson.conversions.Bson;

public class MongoQueries {
        // query Q1
        public static List<? extends Bson> getQ1() {
                String matchStringQuery   = "{\"$match\":{"
                                                        +               "\"LineItems\":{"
                                                        +
        "\"$elemMatch\":{"
                                                        +
        "\"shipdate\":{"
                                                        +
        "\"$lte\": ISODate(\"1998-12-01T00:00:00.000Z\")"
                                                        +                               "}"
                                                        +                       "}"
                                                        +               "}"
                                                        + "}}";
        String unWindStringQuery  = "{\"$unwind\": \"$LineItems\"}";
        String projectStringQuery = "{\"$project\":{"
                                                +               "\"LineItems.returnflag\":1,"
                                                +               "\"LineItems.linestatus\":1,"
                                                +               "\"LineItems.quantity\":1,"
                                                +               "\"LineItems.extendedprice\":1,"
                                                +               "\"LineItems.discount\":1,"
                                                +               "\"l_dis_min_1\":{"
                                                +                       "\"$subtract\":["
                                                +                               "1,"
                                                +
        "\"$LineItems.discount\""
                                                +                       "]"
                                                +               "},"
                                                +               "\"l_tax_plus_1\":{"
                                                +                       "\"$add\":["
                                                +                               "\"$LineItems.tax\","
                                                +                               "1"
                                                +                       "]"
                                                +               "}"
                                                + "}}";
        String groupStringQuery   = "{\"$group\":{"
                                                +       "\"_id\":{"
                                                +               "\"returnflag\":\"$LineItems.returnflag\","
                                                +               "\"linestatus\":\"$LineItems.linestatus\""
                                                +       "},"
                                                +       "\"sum_qty\":{"
                                                +               "\"$sum\":\"$LineItems.quantity\""
                                                +       "},"
                                                +       "\"sum_base_price\":{"
                                                +               "\"$sum\":\"$LineItems.extendedprice\""
                                                +       "},"
                                                +       "\"sum_disc_price\":{"
                                                +               "\"$sum\":{"
                                                +                       "\"$multiply\":["
```

```
                                        +
        "\"$LineItems.extendedprice\","
                                        +                                       "\"$l_dis_min_1\""
                                        +                               "]"
                                        +                       "}"
                                        +               "},"
                                        +               "\"sum_charge\":{"
                                        +                       "\"$sum\":{"
                                        +                               "\"$multiply\":["
                                        +
        "\"$LineItems.extendedprice\","
                                        +                                       "{\"$multiply\":["
                                        +
        "\"$l_tax_plus_1\","
                                        +
        "\"$l_dis_min_1\""
                                        +                                               "]"
                                        +                                       "}"
                                        +                               "]"
                                        +                       "}"
                                        +               "},"
                                        +               "\"avg_qty\":{"
                                        +                       "\"$avg\":\"$LineItems.quantity\""
                                        +               "},"
                                        +               "\"avg_price\":{"
                                        +                       "\"$avg\":\"$LineItems.extendedprice\""
                                        +               "},"
                                        +               "\"avg_disc\":{"
                                        +                       "\"$avg\":\"$LineItems.discount\""
                                        +               "},"
                                        +               "\"count_order\":{"
                                        +                       "\"$sum\":1"
                                        +               "}"
                                        + "}}";
String sortStringQuery          = "{\"$sort\":{"
                                        +               "\"_id.returnflag\":1,"
                                        +               "\"_id.linestatus\":1"
                                        + "}}";

BsonDocument matchBsonQuery = BsonDocument.parse(matchStringQuery);
BsonDocument unWindBsonQuery = BsonDocument.parse(unWindStringQuery);
BsonDocument projectBsonQuery = BsonDocument.parse(projectStringQuery);
BsonDocument groupBsonQuery = BsonDocument.parse(groupStringQuery);
BsonDocument sortBsonQuery = BsonDocument.parse(sortStringQuery);

ArrayList<Bson> aggregateQuery = new ArrayList<Bson>();
aggregateQuery.add(matchBsonQuery);
aggregateQuery.add(unWindBsonQuery);
aggregateQuery.add(projectBsonQuery);
aggregateQuery.add(groupBsonQuery);
aggregateQuery.add(sortBsonQuery);


        return aggregateQuery;
}


// query Q3
public static List<? extends Bson> getQ3() {
        String unWindStringQuery  = "{\"$unwind\": \"$LineItems\"}";
        String matchStringQuery   = "{\"$match\":{"
                                                +
"\"customer.mktsegment\":\"BUILDING\","
```

```
                                                    +                "\"orderdate\":{"
                                                    +                        "\"$lt\":
ISODate(\"1995-03-15T00:00:00.000Z\")"
                                                    +                "},"
                                                    +
        "\"LineItems.shipdate\":{"
                                                    +                        "\"$gt\":
ISODate(\"1995-03-15T00:00:00.000Z\")"
                                                    +                "}"
                                                    + "}}";
            String projectStringQuery = "{\"$project\":{"
                                                +               "\"orderdate\":1,"
                                                        +               "\"shippriority\":1,"
                                                        +               "\"LineItems.extendedprice\":1,"
                                                        +               "\"l_dis_min_1\":{"
                                                +                       "\"$subtract\":["
                                                +                               "1,"
                                                +
        "\"$LineItems.discount\""
                                                +                       "]"
                                                +               "}"
                                                + "}}";
            String groupStringQuery   = "{\"$group\":{"
                                                +               "\"_id\":{"
                                                +                       "\"order\" : \"$_id\","
                                                +                       "\"orderdate\":\"$orderdate\","
                                                +                       "\"shippriority\":\"$shippriority\""
                                                +               "},"
                                                +               "\"revenue\":{"
                                                +                       "\"$sum\":{"
                                                +                               "\"$multiply\":["
                                                +
        "\"$LineItems.extendedprice\","
                                                +                                       "\"$l_dis_min_1\""
                                                +                               "]"
                                                        +                       "}"
                                                +               "}"
                                                + "}}";
            String sortStringQuery          = "{\"$sort\":{"
                                                +               "\"revenue\" : -1"
                                                + "}}";

            BsonDocument unWindBsonQuery = BsonDocument.parse(unWindStringQuery);
            BsonDocument matchBsonQuery = BsonDocument.parse(matchStringQuery);
            BsonDocument projectBsonQuery = BsonDocument.parse(projectStringQuery);
            BsonDocument groupBsonQuery = BsonDocument.parse(groupStringQuery);
            BsonDocument sortBsonQuery = BsonDocument.parse(sortStringQuery);

            ArrayList<Bson> aggregateQuery = new ArrayList<Bson>();
            aggregateQuery.add(unWindBsonQuery);
            aggregateQuery.add(matchBsonQuery);
            aggregateQuery.add(projectBsonQuery);
            aggregateQuery.add(groupBsonQuery);
            aggregateQuery.add(sortBsonQuery);


            return aggregateQuery;
        }



        // query Q4
        public static List<? extends Bson> getQ4() {
```

```
String unWindStringQuery  = "{\"$unwind\": \"$LineItems\"}";
String projectStringQuery = "{\"$project\":{"
                                    +           "\"orderdate\":1,"
                                    +           "\"orderpriority\":1,"
                                    +           "\"eq\":{"
                                    +                   "\"$cond\":["
                                    +                   "{"
                                    +                           "\"$lt\":["
                                    +
        "\"$LineItems.commitdate\","
                                    +
        "\"$LineItems.receiptdate\""
                                    +                           "]"
                                    +                   "}"
                                    +                   "1,"
                                    +                   "0"
                                    +           "]"
                                    +   "}"
                                    + "}}";
String matchStringQuery   = "{\"$match\":{"
                                     +          "\"eq\":{"
                                     +                  "\"$eq\":1"
                                     +          "},"
                                     +          "\"orderdate\":{"
                                     +          "$gte: ISODate(\"1993-
07-01T00:00:00.000Z\"),"
                                     +          "$lt: ISODate(\"1993-
10-01T00:00:00.000Z\")"
                                     +  "}"
                                     + "}}";
    String groupStringQuery1  = "{\"$group\":{"
                                    +   "\"_id\":{"
                                    +           "\"_id\": \"$_id\","
                                    +           "\"orderpriority\" : \"$orderpriority\""
                                    +   "}"
                                    + "}}";
    String groupStringQuery2  = "{\"$group\":{"
                                     +          "\"_id\":{"
                                     +
        "\"orderpriority\":\"$_id.orderpriority\""
                                     +          "},"
                                     +          "\"order_count\":{"
                                     +                  "\"$sum\":1"
                                     +          "}"
                                     + "}}";
    String sortStringQuery          = "{\"$sort\":{"
                                    +           "\"_id.orderpriority\" : 1"
                                    + "}}";

    BsonDocument unWindBsonQuery = BsonDocument.parse(unWindStringQuery);
    BsonDocument projectBsonQuery = BsonDocument.parse(projectStringQuery);
    BsonDocument matchBsonQuery = BsonDocument.parse(matchStringQuery);
    BsonDocument groupBsonQuery1 = BsonDocument.parse(groupStringQuery1);
    BsonDocument groupBsonQuery2 = BsonDocument.parse(groupStringQuery2);
    BsonDocument sortBsonQuery = BsonDocument.parse(sortStringQuery);

    ArrayList<Bson> aggregateQuery = new ArrayList<Bson>();
    aggregateQuery.add(unWindBsonQuery);
    aggregateQuery.add(projectBsonQuery);
    aggregateQuery.add(matchBsonQuery);
    aggregateQuery.add(groupBsonQuery1);
    aggregateQuery.add(groupBsonQuery2);
    aggregateQuery.add(sortBsonQuery);
```

```
        return aggregateQuery;
    }


    // query Q5
    public static List<? extends Bson> getQ5() {
        String unWindStringQuery  = "{\"$unwind\": \"$LineItems\"}";
        String projectStringQuery = "{\"$project\":{"
                                  +         "\"orderdate\":1,"
                                  +         "nation_name : \"$customer.nation.name\","
                                  +         "region_name : \"$LineItems.partsupp.supplier.nation.region.name\","
                                  +         "\"LineItems.extendedprice\":1,"
                                  +         "\"l_dis_min_1\":{"
                                  +             "\"$subtract\":["
                                  +                 "1,"
                                  +                 "\"$LineItems.discount\""
                                  +             "]"
                                  +         "},"
                                  +         "cmp_nation: {$cmp: [\"$customer.nation.name\", \"$LineItems.partsupp.supplier.nation.name\"]}"
                                  + "}}";
        String matchStringQuery   = "{\"$match\":{"
                                  +         "cmp_nation : 0,"
                                  +         "region_name : {\"$eq\" : \"ASIA\"},"
                                  +         "\"orderdate\":{"
                                  +         "$gte: ISODate(\"1994-01-01T00:00:00.000Z\"),"
                                  +         "$lt: ISODate(\"1995-01-01T00:00:00.000Z\")"
                                  +     "}"
                                  + "}}";
        String groupStringQuery   = "{\"$group\":{"
                                  +     "\"_id\":{"
                                  +         "\"nation_name\": \"$nation_name\""
                                  +     "},"
                                  +     "\"revenue\":{"
                                  +         "\"$sum\":{"
                                  +             "\"$multiply\":["
                                  +                 "\"$LineItems.extendedprice\","
                                  +                 "\"$l_dis_min_1\""
                                  +             "]"
                                  +         "}"
                                  +     "}"
                                  + "}}";
        String sortStringQuery    = "{\"$sort\":{"
                                  +     "\"revenue\" : -1"
                                  + "}}";

        BsonDocument unWindBsonQuery = BsonDocument.parse(unWindStringQuery);
        BsonDocument projectBsonQuery = BsonDocument.parse(projectStringQuery);
        BsonDocument matchBsonQuery = BsonDocument.parse(matchStringQuery);
        BsonDocument groupBsonQuery = BsonDocument.parse(groupStringQuery);
        BsonDocument sortBsonQuery = BsonDocument.parse(sortStringQuery);

        ArrayList<Bson> aggregateQuery = new ArrayList<Bson>();
        aggregateQuery.add(unWindBsonQuery);
```

```
            aggregateQuery.add(projectBsonQuery);
            aggregateQuery.add(matchBsonQuery);
            aggregateQuery.add(groupBsonQuery);
            aggregateQuery.add(sortBsonQuery);


            return aggregateQuery;
        }
}
```

ΑΝΑΦΟΡΕΣ

[1] Berg K., Seymour T., and Coel R. “History of Databases.” In: International Journal of Management and Information Services, Vol 17, 2013.

[2] E. F. Codd. “A relational model of data for large shared data banks.” In: Communications of the ACM Vol 13 Issue 6 (1970), pp. 377–387.

[3] A. Moniruzzaman and S. A. Hossain. “Nosql database: New era of databases for big data analytics-classification, characteristics and comparison.” In: arXiv preprint arXiv:1307.0191 (2013).

[4] R. Elmasri – S.B. Navathe. “ΘΕΜΕΛΙΩΔΕΙΣ ΑΡΧΕΣ ΣΥΣΤΗΜΑΤΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ”, 6η ΕΚΔΟΣΗ ΑΝΑΘΕΩΡΗΜΕΝΗ.

[5] Eric A. Brewer. “Towards robust distributed systems.” In: PODC. Vol. 7. 2000.

[6] S. Gilbert and N. A. Lynch. “Perspectives on the CAP Theorem.” In: Institute of Electrical and Electronics Engineers. 2012.

[7] Eric A. Brewer, “CAP twelve years later: How the "rules" have changed”, Computer, Volume 45, Issue 2 (2012), pg. 23–29.

[8] Π. Κρουσταλιός “Τεχνικές Βελτιστοποίησης της Αποθήκευσης και Αναζήτησης Big Data χρησιμοποιώντας την Βάση Δεδομένων MongoDB.” Bachelor's thesis at National Technical University of Athens, 2015.

[9] Κ. Λιβιεράτου “Εισαγωγή στις NoSQL βάσεις δεδομένων”. URL: http://www.linuxinsider.gr/forum/8873/eisagogi-stis-nosql-baseis-dedomenon-xrisi-toy-cassandra (visited on 04/2017).

[10] MongoDB Inc. “NoSQL Databases Explained”. URL: https://www.mongodb.com/nosql-explained (visited on 04/2017).

[11] Oracle Corporation “MySQL 5.7 Reference Manual”. URL: https://dev.mysql.com/doc/refman/5.7/en/ (visited on 04/2017).

[12] Oracle Corporation “Top Reasons to Use MySQL”. URL: https://www.mysql.com/why-mysql/topreasons.html (visited on 04/2017).

[13] Oracle Corporation “Employees Sample Database”. URL: https://dev.mysql.com/doc/employee/en/ (visited on 04/2017).

[14] MongoDB Inc. “The MongoDB 3.4 Manual”. URL: https://docs.mongodb.com/manual/ (visited on 04/2017).

[15] BSON “BSON Spec.” URL: http://bsonspec.org/ (visited on 04/2017).

[16] TPC. “TPC-H is an ad-hoc, decision support benchmark”. URL: http://www.tpc.org/tpch/ (visited on 03/2017).

[17] Alronz Github. “MongoDB Denormalised Example”. URL: http://alronz.github.io/Factors-Influencing-NoSQL-Adoption/site/MongoDB/Examples/Denormalised%20Model/ (visited on 03/2017).