ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

“ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ & ΥΠΗΡΕΣΙΕΣ”

“NodaMD: Επέκταση του NoDA για Επεξεργασία Δεδομένων σε MongoDB με Χρήση Αφαιρετικού Επιπέδου”

“NodaMD: Extending NoDA for Data Management in MongoDB through an Abstraction Layer”

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δεϊμέζη Ελένη

ΜΕ 2208

Επιβλέπων Καθηγητής: Χρήστος Δουλκερίδης

Πειραιάς, 2025

Πίνακας περιεχομένων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Περίληψη

Η παρούσα διπλωματική εργασία πραγματεύεται τη μελέτη και επέκταση του ερευνητικού συστήματος NoDA, το οποίο έχει αναπτυχθεί στο Τμήμα Ψηφιακών Συστημάτων του Πανεπιστημίου Πειραιά και στοχεύει στην παροχή ενιαίας πρόσβασης σε NoSQL βάσεις δεδομένων. Στο πλαίσιο της εργασίας υλοποιήθηκε η επέκταση NodaMD η οποία επιτρέπει τη διαχείριση δεδομένων στη MongoDB μέσω ενός αφαιρετικού επιπέδου. Αναπτύχθηκαν οι λειτουργίες εισαγωγής (insert), τροποποίησης (update) και διαγραφής (delete) εγγραφών, οι οποίες ενσωματώνονται στη γενική αρχιτεκτονική του συστήματος, επιτρέποντας την εκτέλεση CRUD ενεργειών μέσω αφαιρετικού μηχανισμού, χωρίς να απαιτείται από τον χρήστη άμεση συγγραφή ερωτημάτων στη σύνταξη της MongoDB.

Παράλληλα, παρουσιάζονται τα βασικά χαρακτηριστικά των σχεσιακών βάσεων δεδομένων, οι κατηγορίες τους, οι τρόποι αποθήκευσης και οι βασικές περιπτώσεις χρήσης (OLTP, OLAP κ.ά.). Ακολουθεί αναλυτική παρουσίαση των NoSQL βάσεων, με ταξινόμηση σε υποκατηγορίες όπως αποθήκες εγγράφων, κλειδιών-τιμών, στηλών και γράφων, και πραγματοποιείται συγκριτική αξιολόγηση ως προς την καταλληλότητα τους σε διαφορετικά σενάρια χρήσης.

Τέλος, η εργασία περιλαμβάνει σενάρια χρήσης και αξιολόγηση της λειτουργικότητας του NodaMD μέσα από πρακτικά παραδείγματα, ενώ κατατίθενται προτάσεις για τη μελλοντική επέκταση του συστήματος σε επιπλέον τύπους NoSQL βάσεων και σε πιο σύνθετες μορφές ερωτημάτων.

Abstract

This thesis focuses on the study and extension of the research system NoDA, developed at the Department of Digital Systems of the University of Piraeus, which aims to provide unified access to NoSQL databases. As part of this work, the NodaMD extension was implemented, enabling data management in MongoDB through an abstraction layer. Insert, update, and delete operations were developed and integrated into the system's overall architecture, allowing the execution of CRUD operations without requiring the user to directly write queries in MongoDB's native syntax.

In parallel, the thesis presents the fundamental characteristics of relational databases, including their types, storage methods, and common use cases (such as OLTP and OLAP). It then provides a detailed overview of NoSQL databases, classified into categories such as document stores, key-value stores, column stores, and graph databases, followed by a comparative analysis regarding their suitability for different usage scenarios.

Finally, the thesis includes usage scenarios and an evaluation of NodaMD's functionality through practical examples, while also proposing directions for future research, including support for additional NoSQL database types and more complex query mechanisms.

Ευχαριστίες / Acknowledgments

Η παρούσα διπλωματική εργασία δεν θα μπορούσε να ολοκληρωθεί χωρίς την πολύτιμη υποστήριξη ανθρώπων που στάθηκαν δίπλα μου με πίστη, υπομονή και καθοδήγηση.

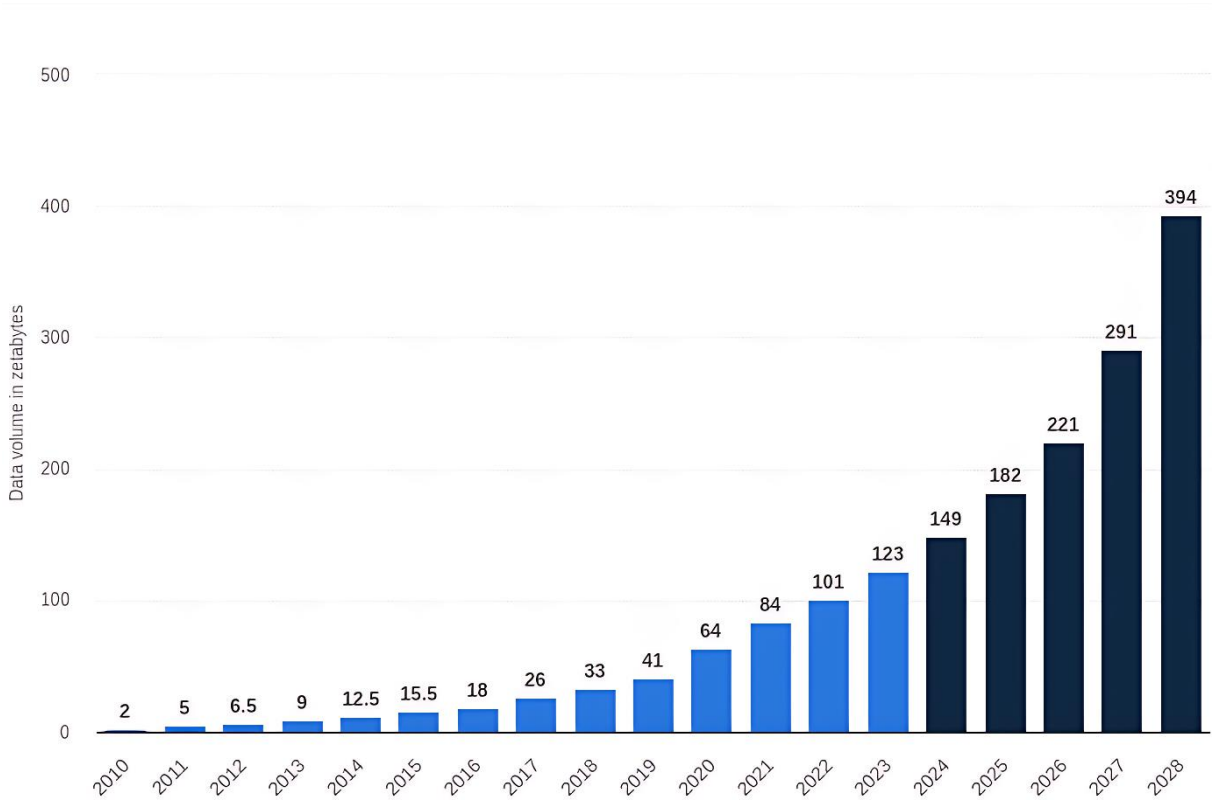
Καταρχάς, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου Καθηγητή Χρήστο Δουλκερίδη για την εμπιστοσύνη που μου έδειξε, τη σαφή και ουσιαστική καθοδήγηση και το ότι με ενθάρρυνε να προσεγγίσω κάθε βήμα με επιστημονική σκέψη και αφοσίωση.

Με απεριόριστη ευγνωμοσύνη ευχαριστώ τη μητέρα μου Σόφη για την αγάπη και τη δύναμη που μου έδωσε σε κάθε στάδιο αυτής της διαδρομής. Η στήριξή της υπήρξε η πιο σταθερή μου βάση, ακόμη κι όταν όλα φαίνονταν δύσκολα.

Τέλος, ένα μεγάλο ευχαριστώ σε όλους όσοι πίστεψαν σε εμένα, με ενίσχυσαν έμπρακτα ή με λόγια και με βοήθησαν να εξελιχθώ όχι μόνο ακαδημαϊκά αλλά και προσωπικά μέσα από αυτή την εμπειρία.

1. Εισαγωγή

Στη σύγχρονη εποχή, παρατηρείται μια σταδιακή απομάκρυνση από τις εμπορικές εφαρμογές που βασίζονται σε υψηλές απαιτήσεις τοπικής υπολογιστικής ισχύος (compute-intensive), με τη μετάβαση να εστιάζεται στις εφαρμογές διαχείρισης δεδομένων μεγάλης κλίμακας (data-intensive). Ο σχεδιασμός και η ανάπτυξη τέτοιων εφαρμογών υπαγορεύονται πλέον από την ανάγκη αποθήκευσης και επεξεργασίας του τεράστιου όγκου δεδομένων που παράγεται από τους τελικούς χρήστες. Όπως φαίνεται από την *Εικόνα 1*, με την πάροδο των χρόνων, ο αριθμός των δεδομένων που παράγονται σημειώνει εκθετική αύξηση.



Εικόνα 1: Όγκος δεδομένων που δημιουργούνται, συλλαμβάνονται, αντιγράφονται και καταναλώνονται παγκοσμίως από το 2010 έως το 2024, με προβλέψεις έως το 2028. [2]

1.1 Κίνητρα - Προκλήσεις

Συνεπώς, οι εφαρμογές της σημερινής εποχής, λόγω της φύσης των δεδομένων που διατηρούν, που συνήθως είναι αδόμητα ή ημιδομημένα, τείνουν να χρησιμοποιούν μη σχεσιακές βάσεις δεδομένων για την αποθήκευση και την διαχείριση των δεδομένων τους ή ένα συνδυασμό σχεσιακών (SQL) και μη (NoSQL) συστημάτων. Αναλυτικότερα, σύμφωνα με έρευνα που έλαβε χώρα στο DeveloperWeek 2019 στο Σαν Φρανσίσκο, το 75% των εφαρμογών χρησιμοποιούν έναν συνδυασμό SQL και NoSQL συστημάτων για τις ανάγκες αποθήκευσης και διαχείρισης των δεδομένων τους. Ωστόσο, εν αντιθέση με τα σχεσιακά συστήματα δεδομένων που χρησιμοποιούν προτυποποιημένα σχήματα και μία δομημένη γλώσσα επερώτημάτων, τα μη σχεσιακά συστήματα χαρακτηρίζονται από σημαντική ετερογένεια αναφορικά με τις γλώσσες επερωτημάτων, την μοντελοποίηση των δεδομένων και τις ιδιότητες συνέπειας που προσφέρει το καθένα. Λόγω αυτής της ποικιλομορφίας των NoSQL συστημάτων και των έντονων διαφορών που έχουν αναμεταξύ τους, αυξάνεται δραστικά η πολυπλοκότητα των εφαρμογών που έχουν ενσωματώσει και κάνουν ταυτόχρονη χρήση συνδυασμών μη σχεσιακών και σχεσιακών συστημάτων. Επιπλέον, πέρα από το γεγονός ότι αυτοί που αναπτύσσουν αυτές τις εφαρμογές πρέπει να είναι εξοικειωμένοι με τις ιδιαιτερότητες του κάθε NoSQL συστήματος, όταν χρειαστεί να μεταναστεύσουν από ένα σύστημα σε ένα άλλο, το έργο αυτό γίνεται ακόμα πιο δύσκολο λόγω των πολλών αλλαγών που θα χρειαστούν σε επίπεδο λογικής σχεδίασης και επιλογής κατάλληλων δομών δεδομένων.

Έχοντας πλέον αναλύσει τις προκλήσεις που επιφέρει η επιλογή NoSQL συστημάτων και η δυσκολία της διαλειτουργικότητάς τους σε μία ενιαία εφαρμογή, είναι προφανές ότι υπάρχει μία επιτακτική ανάγκη για την δημιουργία κάποιας διεπαφής που θα επιτρέπει με έναν ενιαίο και ομοιόμορφο τρόπο την αλληλεπίδραση με τα διάφορα NoSQL συστήματα βάσεων δεδομένων. Κατά αυτόν τρόπο, υποστηρίζοντας δηλαδή μία δηλωτική διεπαφή, όπως η SQL, πάνω από τα NoSQL συστήματα, θα μπορούν οι χρήστες με εύκολο τρόπο να καθορίζουν τι ακριβώς θέλουν να ανακτήσουν από τα δεδομένα αντί να χρησιμοποιούν τα συχνά πολύπλοκα και διαφορετικά API των NoSQL συστημάτων που απαιτούν σημαντική προσπάθεια (επιπλέον κώδικα) για τα ίδια επερωτήματα. Ακόμα, η SQL θεωρείται ως ο προτιμότερος τρόπος ανάκτησης και επεξεργασίας δεδομένων, καθώς προσφέρει την

δυνατότητα δημιουργίας ευέλικτων και σύνθετων επερωτημάτων, με τους περισσότερους προγραμματιστές της αγοράς να είναι ήδη αρκετά εξοικειωμένοι με τον τρόπο λειτουργίας της. Τέλος, τέτοιες δηλωτικές διεπαφές μπορούν να επιτρέψουν την αποδοτικότερη εκτέλεση ενός επερωτήματος, καθώς παράγουν διάφορα πλάνα εκτέλεσης και επιλέγουν το πιο αποδοτικό χρησιμοποιώντας εδραιωμένους και αξιόπιστους αλγόριθμους βελτιστοποίησης.

1.2 Ορισμός Προβλήματος

Συνοψίζοντας τις προκλήσεις που αποτέλεσαν έναυσμα του σχεδιασμού του συστήματος NoDA [10] που πραγματεύεται η συγκεκριμένη πτυχιακή εργασία, η χρήση και η διαλειτουργικότητα των συστημάτων αποθήκευσης NoSQL επιφέρει πολλές προκλήσεις, λόγω των διαφορετικών μοντέλων δεδομένων, των γλωσσών επερωτημάτων και της έλλειψης δομημένης πρόσβασης. Ως εκ τούτου, αυξάνεται σε μεγάλο βαθμό η πολυπλοκότητα της ανάπτυξης και της διατήρησης λογισμικού. Υπό αυτό το πρίσμα λοιπόν, το σύστημα NoDA αντιμετωπίζει αυτά τα προβλήματα παρέχοντας ένα ενοποιημένο επίπεδο αφαίρεσης που λειτουργεί ως μια δηλωτική διεπαφή SQL, “πάνω” από το εκάστοτε σύστημα NoSQL, η οποία χρησιμοποιώντας γενικούς τελεστές πρόσβασης δεδομένων, αποκρύπτει αποτελεσματικά την ετερογένεια των υποκείμενων NoSQL συστημάτων στον χρήστη και διευκολύνει αξιοσημείωτα την αλληλεπίδραση μεταξύ του συστήματος και του χρήστη. Ουσιαστικά, το σύστημα NoDA λειτουργεί μεταφράζοντας το επερώτημα που είναι διατυπωμένο σε SQL σε εγγενείς εντολές NoSQL, γεφυρώνοντας το χάσμα ανάμεσα στους τρόπους αλληλεπίδρασης που προσφέρει το κάθε NoSQL σύστημα. Επιπλέον, η επιλογή της SQL ως το κύριο μέσο αλληλεπίδρασης με το σύστημα δεν είναι τυχαία, καθώς εκτός από το γεγονός ότι το συντακτικό της είναι εξαιρετικά απλοϊκό και διαισθητικό, οι πλειοψηφία των προγραμματιστών είναι ήδη αρκετά εξοικειωμένοι με τον τρόπο χρήσης της.

1.3 Διάρθρωση Εργασίας

Η εργασία δομείται ως εξής:

Κεφάλαιο 2: Το κεφάλαιο 2 περιλαμβάνει την βιβλιογραφική ανασκόπηση στο ερευνητικό χώρο των συστημάτων βάσεων δεδομένων. Συγκεκριμένα, αναλύεται η χρήση και οι τεχνολογίες των σχεσιακών και των μη σχεσιακών βάσεων δεδομένων. Έπειτα, περιλαμβάνεται η σύγκριση των δύο κατηγοριών βάσεων δεδομένων (σχεσιακές, μη σχεσιακές) ως προς την απόδοση τους και τα σενάρια χρήσης που υποστηρίζουν.

Κεφάλαιο 3: Το κεφάλαιο 3 περιλαμβάνει την περιγραφή του πρωτότυπου συστήματος Noda και του εκτεταμένου συστήματος NodaMD. Το NodaMD περιλαμβάνει την λειτουργία διασύνδεσης του υπάρχοντος συστήματος Noda με την μη σχεσιακή βάση MongoDB. Τέλος, αναφέρονται ενδεικτικά σενάρια χρήσης του NodaMD στον πραγματικό κόσμο. Τέλος, περιλαμβάνει τις τεχνικές προδιαγραφές του συστήματος NodaMD.

Κεφάλαιο 4: Το κεφάλαιο 4 παρουσιάζει την χρήση του συστήματος NodaMD σε μια πραγματική βάση δεδομένων MongoDB και την συμπεριφορά του συστήματος. Επιπλέον, περιλαμβάνει την αξιολόγηση του συστήματος NodaMD.

Κεφάλαιο 5: Το τελευταίο κεφάλαιο συνοψίζει τις κύριες συνεισφορές και τα βασικά ευρήματα της παρούσας διπλωματικής εργασίας. Επιπλέον, σκιαγραφεί πιθανούς άξονες μελλοντικής έρευνας, οι οποίοι δεν κατέστη δυνατόν να διερευνηθούν στο πλαίσιο του παρόντος έργου λόγω χρονικών περιορισμών, προσφέροντας κατευθύνσεις για περαιτέρω πρόοδο στον συγκεκριμένο τομέα.

2. Συστήματα Βάσεων Δεδομένων

Η μετάβαση από τα συμβατικά συστήματα αποθήκευσης αρχείων στις σύγχρονες βάσεις δεδομένων σηματοδότησε μια κομβική εξέλιξη στον τομέα της διαχείρισης δεδομένων. Αν και αρχικά, τα πρώιμα συστήματα αρχειοθέτησης αρχείων ανταποκρίνονταν στις βασικές ανάγκες αποθήκευσης, το αρχικό γραμμικό μοντέλο αποθήκευσης υπέφερε από πλεονασμό δεδομένων, προβλήματα που σχετίζονταν με την ασυνέπεια των δεδομένων και έλλειψη τυποποίησης στη μορφή αρχείων, καθιστώντας την ανάκτηση δεδομένων συχνά αργή. Καθώς οι απαιτήσεις για διαχείριση δεδομένων γίνονταν όλο και πιο πολύπλοκες, οι περιορισμοί των διάσπαρτων, μη οργανωμένων αρχείων γίνονταν πιο εμφανείς, οδηγώντας στην ανάπτυξη των σχεσιακών βάσεων δεδομένων. Σε αντίθεση με τα συστήματα αποθήκευσης αρχείων, οι σχεσιακές βάσεις δεδομένων εισήγαγαν δομημένες μεθόδους οργάνωσης και αποθήκευσης δεδομένων, επιτρέποντας την αποτελεσματική ευρετηρίαση και την αναγνώριση. Γλώσσες ερωτημάτων ανώτερης τάξης όπως η SQL πρόσφεραν γρηγορότερη, πιο αξιόπιστη πρόσβαση στα δεδομένα. Επίσης, υιοθέτησαν μέτρα για να εξασφαλίσουν την ακεραιότητα της βάσης δεδομένων, σύμφωνα με το πρότυπο ACID, τη μείωση του πλεονασμού χρησιμοποιώντας την κανονικοποίηση και την υποστήριξη ταυτόχρονων, απομονωμένων και συνεπών λειτουργιών πολλών χρηστών. Επιπλέον, οι σύγχρονες βάσεις δεδομένων παρέχουν αυξημένα χαρακτηριστικά ασφαλείας για την εξουσιοδότηση και εξακρίβωση αυθεντικότητας και τη δυνατότητα δημιουργίας σχέσεων μεταξύ σημείων δεδομένων για να καταστεί δυνατή η εξελιγμένη ανάλυση και ανάκτηση δεδομένων. Αυτές οι εξελίξεις όχι μόνο διευκολύνουν την διαχείριση των δεδομένων αλλά κυρίως “ανοίγουν το δρόμο” για πιο κλιμακούμενα, αποδοτικά και ευφυή συστήματα βασισμένα σε δεδομένα που τροφοδοτούν τις βιομηχανίες του σήμερα.

2.1 Σχεσιακές Βάσεις Δεδομένων

Μια σχεσιακή βάση δεδομένων (Relational Database) [11] ορίζεται ως ένα οργανωμένο σύνολο δεδομένων που αποθηκεύεται σε δομημένη μορφή, χρησιμοποιώντας πίνακες (tables) ή αλλιώς σχέσεις (relations). Κάθε πίνακας αποτελείται από γραμμές (εγγραφές ή πλειάδες) και στήλες (πεδία ή γνωρίσματα). Το σχεσιακό μοντέλο [7] δεδομένων στηρίζεται στην θεωρία συνόλων και την σχεσιακή άλγεβρα, έτσι επιτρέπει την δομημένη μοντελοποίηση των δεδομένων και των σχέσεων μεταξύ τους, με τρόπο που διευκολύνει την ανάλυση και την εξαγωγή συμπερασμάτων από αυτά. Στο σχεσιακό μοντέλο [7] ορίζουμε ως μία οντότητα ένα αντικείμενο που υπάρχει και διακρίνεται από τα άλλα αντικείμενα, όπως ένας συγκεκριμένος υπάλληλος, μία συγκεκριμένη εταιρεία ή ένα συγκεκριμένο εργοστάσιο. Αντίστοιχα, οι οντότητες ίδιου τύπου δημιουργούν σύνολα οντοτήτων, όπως οι υπάλληλοι, οι εταιρείες και τα εργοστάσια. Οι οντότητες που ανήκουν σε ένα σύνολο οντοτήτων, διέπονται από ένα σύνολο κοινών γνωρισμάτων (χαρακτηριστικών), όπως το ονοματεπώνυμο, η θέση και ο μισθός κάθε υπαλλήλου. Κατά αυτόν τον τρόπο, λοιπόν, ένας πίνακας μπορεί να χρησιμοποιηθεί για την αποθήκευση δεδομένων για ένα συγκεκριμένο σύνολο οντοτήτων, όπου κάθε πλειάδα αντιστοιχεί σε μία μοναδική οντότητα και κάθε στήλη σε ένα από τα γνωρίσματα του συνόλου. Επιπλέον, κάθε πίνακας έχει ένα πρωτεύον κλειδί (primary key), ένα σύνολο γνωρισμάτων δηλαδή, που είναι μη κενό και έχει μοναδική τιμή για κάθε οντότητα και έτσι μπορεί να χρησιμοποιηθεί για τον διαχωρισμό των διαφορετικών οντοτήτων του συνόλου. Για παράδειγμα, κάθε υπάλληλος μπορεί να έχει έναν μοναδικό κωδικό και έτσι η συγκεκριμένη στήλη να λειτουργήσει ως το πρωτεύων κλειδί του πίνακα των υπαλλήλων. Τα διάφορα σύνολα οντοτήτων μπορούν να έχουν και σχέσεις μεταξύ τους, όπως το γεγονός ότι κάθε υπάλληλος λογοδοτεί στον προϊστάμενό του. Συνεπώς, για να αποτυπωθεί αυτή η σχέση στο σχεσιακό σύστημα μπορεί ο πίνακας των υπαλλήλων να έχει ένα γνώρισμα του οποίου η τιμή να είναι ο κωδικός υπαλλήλου του εκάστοτε προϊστάμενου. Τότε, αυτό το γνώρισμα λέμε πως είναι ένα ξένο κλειδί (foreign key) του πίνακα υπαλλήλων που αναφέρεται στον κωδικό υπαλλήλου του πίνακα προϊσταμένων. Έχοντας ορίσει το πρωτεύων και το ξένο κλειδί, ένα από τα δυνατότερα εργαλεία που προσφέρει το σχεσιακό μοντέλο είναι η εντολή JOIN. Με αυτή την εντολή μπορούμε να συνδυάσουμε τις σειρές και τις στήλες δύο ή παραπάνω πινάκων βάση ενός κοινού γνωρίσματος. Έτσι, μας επιτρέπεται

να αναλύουμε πολλούς πίνακες ταυτόχρονα χάρις την δυνατότητα θεμελίωσης σχέσεων μεταξύ τους με την κατάλληλη χρήση ξένων κλειδιών.

2.2 Συστήματα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων

Ως Σύστημα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων (Relational Database Management System - RDBMS) [11] ορίζουμε το λογισμικό που επιτρέπει την δημιουργία, την διαχείριση και την αλληλεπίδραση με μια σχεσιακή βάση δεδομένων. Το RDBMS χειρίζεται τις λειτουργίες αποθήκευσης, ανάκτησης και ενημέρωσης των δεδομένων, ενώ παράλληλα διασφαλίζει την ασφάλεια, την ακεραιότητα και την απόδοση του συστήματος. Τέτοια συστήματα προσφέρουν υπηρεσίες ελέγχου πρόσβασης, επιτρέποντας την ανάθεση δικαιωμάτων σε χρήστες και ομάδες χρηστών και του ενδεχόμενου μερικού περιορισμού πρόσβασης σε κάποια υποσύνολα δεδομένων. Επιπλέον, προσφέρονται εργαλεία για τη δημιουργία αντιγράφων ασφαλείας των δεδομένων και δυνατότητα ανάκτησή τους σε περίπτωση απώλειας ή καταστροφής, εξασφαλίζοντας σημαντικές εγγύησης συνέπειας και ασφάλειας του συστήματος. Ακόμα μία σημαντική πτυχή αυτών των συστημάτων είναι το γεγονός πως επιτρέπουν σε πολλούς χρήστες να αλληλεπιδρούν ταυτόχρονα με τη βάση δεδομένων, παρέχοντας αυτοματοποιημένη διαχείριση τυχόν συγκρούσεων που μπορεί να προκύψουν. Αυτές οι εγγυήσεις είναι απόρροια της ευθυγράμμισης αυτών των συστημάτων με τις ιδιότητες ACID (Ατομικότητα (Atomicity), Συνέπεια (Consistency), Απομόνωση (Isolation), Ανθεκτικότητα (Durability)) και την χρήση δοσοληψιών ή συναλλαγών (transactions). Οι ιδιότητες ACID καθορίζουν την αξιοπιστία των συναλλαγών σε ένα σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων. Η Ατομικότητα εξασφαλίζει ότι αναφορικά με μία συναλλαγή είτε όλες τις οι λειτουργίες της πετυχαίνουν είτε καμία δεν εφαρμόζεται. Έτσι το σύστημα δεν θα βρεθεί ποτέ σε μη έγκυρη κατάσταση. Η Συνέπεια αντίστοιχα, εγγυάται ότι η βάση δεδομένων μεταβαίνει πάντα από μια έγκυρη κατάσταση σε μια άλλη, διατηρώντας τους περιορισμούς ακεραιότητας. Η Απομόνωση ελέγχει τον τρόπο με τον οποίο αλληλεπιδρούν οι ταυτόχρονες συναλλαγές, αποτρέποντας συγκρούσεις όπως βρώμικες αναγνώσεις (dirty reads) ή χαμένες ενημερώσεις (lost updates). Τέλος, η

Ανθεκτικότητα διασφαλίζει ότι μόλις μια συναλλαγή δεσμευτεί, παραμένει μόνιμα αποθηκευμένη, ακόμη και σε περίπτωση αποτυχίας του συστήματος. Υπό αυτό το πρίσμα λοιπόν, τα συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων παρέχουν ακεραιότητα δεδομένων, ανοχή σφαλμάτων και έλεγχο ταυτόχρονης χρήσης, καθιστώντας τα ιδανικά για κρίσιμες εφαρμογές όπως τα τραπεζικά συστήματα και το ηλεκτρονικό εμπόριο. Αξίζει να υπογραμμισθεί ότι για την δημιουργία μιας σχεσιακής βάσης δεδομένων απαιτείται το ανάλογο σχεσιακό σχήμα (relation schema). Το σχεσιακό σχήμα είναι μια λογική δομή που καθορίζει τον τρόπο οργάνωσης των δεδομένων σε μια βάση δεδομένων. Περιλαμβάνει τους πίνακες, τις προβολές, τα ευρετήρια, τις αποθηκευμένες διαδικασίες και άλλα αντικείμενα της βάσης δεδομένων, καθορίζοντας τις σχέσεις, τους περιορισμούς και τους τύπους δεδομένων τους εκ των προτέρων. Ουσιαστικά, λειτουργεί ως ένα αρχικό σχέδιο που διασφαλίζει τη συνοχή και θέτει τα αρχικά θεμέλια της μορφής που θα έχει ολόκληρη η σχεσιακή βάση δεδομένων.

2.3 Αλληλεπίδραση με Σχεσιακές Βάσεις Δεδομένων

Η αλληλεπίδραση με ένα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων χρησιμοποιώντας ως κύριο μέσο την γλώσσα προγραμματισμού ονόματι SQL (Structured Query Language) [11], η οποία μας επιτρέπει να αποθηκεύουμε, να ανακτούμε και να διαχειριζόμαστε δεδομένα με αποτελεσματικό τρόπο. Η SQL χωρίζεται σε διάφορες κατηγορίες όπως η Γλώσσα Ορισμού Δεδομένων (Data Definition Language - DDL) για τη δημιουργία και τροποποίηση των δομών δεδομένων, δηλαδή των πινάκων, χρησιμοποιώντας τις εντολές CREATE, ALTER και DROP. Αντίστοιχα, υπάρχει η Γλώσσα Χειρισμού Δεδομένων (Data Manipulation Language – DML), η οποία αποτελείται από τις εντολές INSERT, UPDATE, DELETE και SELECT, επιτρέποντάς μας να διαχειριστούμε τα δεδομένα που βρίσκονται εντός των πινάκων. Επιπλέον, έχουμε την Γλώσσα Ελέγχου Δεδομένων (Data Control Language – DCL), η οποία επιτρέπει τη διαχείριση της πρόσβασης και της ασφάλειας των χρηστών με χρήση των εντολών GRANT και REVOKE. Γενικότερα, η συνηθέστερη εντολή της SQL είναι η εντολή SELECT, που χρησιμοποιείται για την αναζήτηση

την επιλογή και την ανάκτηση υποσυνόλων δεδομένων, συχνά σε συνδυασμό με χρήση την εντολής JOIN για τον συνδυασμό πληροφοριών από διάφορους πίνακες. Επίσης, η SQL υποστηρίζει το φιλτράρισμα των δεδομένων, με χρήση των εντολών WHERE και HAVING, την ταξινόμηση των αποτελεσμάτων μέσα από την εντολή ORDER BY και την ομαδοποίηση του επιθυμητού υποσυνόλου δεδομένων με χρήση της εντολής GROUP BY. Τέλος, αξίζει να σημειωθεί ότι πολλά RDBMS παρέχουν κάποιες παραπάνω δυνατότητες όπως η δημιουργία αποθηκευμένων διαδικασιών (Stored Procedures), δηλαδή προσυνταγμένων SQL scripts που μπορούν να εκτελεστούν, πυροδοτήσεων (Triggers), δηλαδή αυτόματες ενέργειες που εκτελούνται όταν συμβεί ένα συγκεκριμένο γεγονός, και δημιουργία ευρετηρίων (Indexing) για τη βελτιστοποίηση της απόδοσης και την αυτοματοποίηση των διαδικασιών εντός του συστήματος.

2.4 Κατηγορίες Σχεσιακών Βάσεων Δεδομένων

Στην συγκεκριμένη ενότητα, παρουσιάζεται η ομαδοποίηση των σχεσιακών βάσεων δεδομένων βάση ορισμένων διακριτών περιπτώσεων χρήσης. Ωστόσο, πρέπει να αναφέρουμε πως, γενικότερα, οι σχεσιακές βάσεις δεδομένων εν γένει δεν διαφέρουν ιδιαίτερα σε επίπεδο χαρακτηριστικών ή λειτουργιών καθώς όλες ακολουθούν στενά το σχεσιακό μοντέλο [7] καθώς και το πρότυπο ACID. Συνήθως, τα διάφορα συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων τείνουν να διαφέρουν καθώς για να παραμείνουν ανταγωνιστικά προσφέρουν επιμέρους λειτουργίες και δυνατότητες που τα διαφοροποιούν από τα ομότεχνά τους. Ακολουθεί ο εν λόγω διαχωρισμός βάση περιπτώσεων χρήσης.

2.4.1 Online Transaction Processing (OLTP).

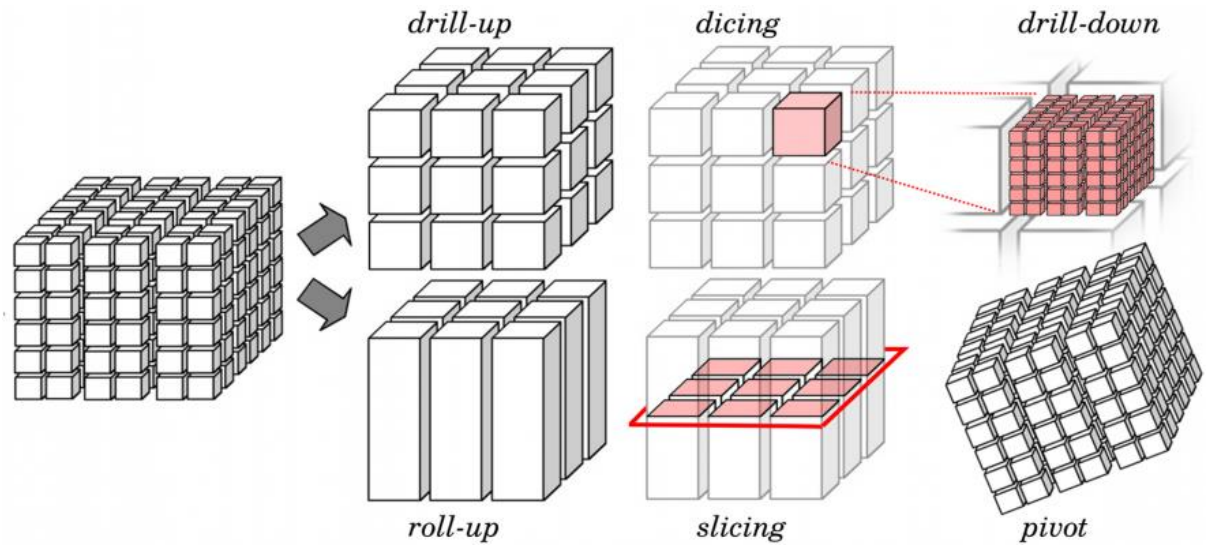
Οι σχεσιακές βάσεις δεδομένων που χρησιμοποιούνται για περιπτώσεις χρήσης OLTP πρέπει είναι βελτιστοποιημένες για τη διαχείριση μεγάλου όγκου σύντομων και παράλληλων συναλλαγών σε πραγματικό χρόνο. Οι συγκεκριμένες βάσεις δεδομένων επικεντρώνονται

κυρίως στην ταχύτητα που προσφέρουν σε λειτουργίες ανάγνωσης και τροποποίησης των δεδομένων (INSERT, UPDATE, DELETE), ενώ παράλληλα εξασφαλίζουν συμμόρφωση με το πρότυπο ACID (Atomicity, Consistency, Isolation, Durability) για τη διατήρηση της ακεραιότητας των δεδομένων. Τα συστήματα αυτά πρέπει να είναι ανταποκρίσιμα σε πραγματικό χρόνο και συνεπή, καθώς είναι απαραίτητο να επιτρέπουν την ταυτόχρονη πρόσβαση πολλών χρηστών στο ίδιο σύστημα. Επιπλέον πρέπει να είναι διαθέσιμα 24/7/365 με αρκετά αντίγραφα ασφαλείας καθώς διατηρούν δεδομένα ζωτικής σημασίας για τον εκάστοτε οργανισμό. Επίσης, στα συγκεκριμένα συστήματα τα δεδομένα είναι κανονικοποιημένα έτσι ώστε να ελαττώνεται στο ελάχιστο η διατήρηση διπλότυπων δεδομένων. Γενικότερα, τέτοια συστήματα χρησιμοποιούνται ευρέως σε κλάδους που απαιτούν γρήγορη και αξιόπιστη επεξεργασία συναλλαγών, όπως ο τραπεζικός τομέας, το λιανικό εμπόριο και η διαχείριση εφαρμογών CRM ή ERP. Με τα συστήματα αυτά αλληλεπιδρούν κυρίως απλοί εργαζόμενοι που δεν είναι απαραίτητα εξοικειωμένοι με την πληροφορική, όπως ταμίες και εργαζόμενοι γραφείου, ή μπορεί να είναι και πλήρως αυτοματοποιημένα όπως σε συστήματα διαχείρισης κρατήσεων. Παραδείγματα τέτοιων συστημάτων είναι τα MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database και γενικότερες τα περισσότερα συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων.

2.4.2 Online Analytical Processing (OLAP)

Οι σχεσιακές βάσεις δεδομένων που προορίζονται για ενσωμάτωση σε συστήματα OLAP έχουν σχεδιαστεί για να υποστηρίζουν σύνθετα επερωτήματα και πολυδιάστατη ανάλυση δεδομένων, σε αντίθεση με αυτές που προορίζονται για συστήματα OLTP και είναι σχεδιασμένες για συχνές ενημερώσεις συναλλαγών. Αναλυτικότερα, ειδικεύονται στη συγκέντρωση μεγάλων συνόλων δεδομένων για ανάλυση σε επίπεδο business intelligence και την ανάλογη εξαγωγή συμπερασμάτων και λήψη αποφάσεων. Σε αντίθεση με τα συστήματα OLTP, οι βάσεις δεδομένων OLAP επεκτείνουν την τεχνική αποθήκευσης δεδομένων σε πίνακες χρησιμοποιώντας κύβους, βελτιστοποιώντας έτσι την απόδοση σε μεγάλης κλίμακας δεδομένα. Η χρήση του κύβου ως δομή αποθήκευσης και μοντελοποίησης δεδομένων προσθέτει νέα επίπεδα και κατά συνέπεια περισσότερες διαστάσεις στα

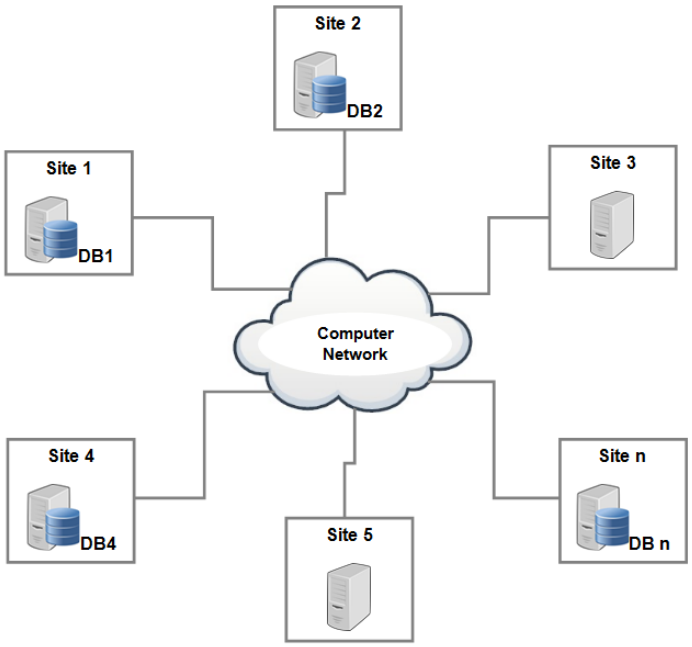
δεδομένα. Για παράδειγμα, ενώ το ανώτερο στρώμα του κύβου μπορεί να οργανώνει τις πωλήσεις ανά περιοχή, οι αναλυτές μπορούν επίσης να «τρυπήσουν» (drill-down) σε στρώματα για να ανακτήσουν πωλήσεις ανά πολιτεία/επαρχία, πόλη ή/και συγκεκριμένα καταστήματα. Συνήθως τα OLAP συστήματα διατηρούν αρχείο ιστορικών δεδομένων τα οποία είναι μοντελοποιημένα βάση ενός σχήματος αστεριού ή χιονονιφάδας. Αυτά τα συστήματα είναι σχεδιασμένα για ανάκτηση και μοντελοποίηση δεδομένων και όχι για την ενημέρωση ή την τροποποίησή τους, οπότε είναι βελτιστοποιημένα για λειτουργίες ανάκτησης, φιλτραρίσματος και συνένωσης δεδομένων. Επιπλέον, εφόσον διατηρούν παλαιότερα δεδομένα, δεν είναι ζωτικής σημασίας η συχνή παραγωγή αντιγράφων του συστήματος. Πολλές φορές τα συστήματα OLAP γίνονται αποθήκες δεδομένων που προέρχονται από συστήματα OLTP, με σκοπό την εξόρυξη γνώσης και συμπερασμάτων από αυτά, κάτι που δεν συμβαίνει στα συστήματα OLTP. Η πλειοψηφία των λύσεων OLAP βασίζονται στο νέφος, όπως το Amazon Redshift, το Google BigQuery και το Snowflake, επιτρέποντας στους οργανισμούς να αναλύουν αποτελεσματικά τεράστιες ποσότητες δεδομένων χωρίς να υπάρχει η αντίστοιχη ανάγκη για πόρους σε τοπικό επίπεδο. Ωστόσο υπάρχουν και OLAP συστήματα που τρέχουν τοπικά όπως το SAP HANA, μια βάση δεδομένων OLAP στη μνήμη που παρέχει αναλύσεις σε πραγματικό χρόνο για τις επιχειρήσεις, επιτρέποντας τους να αντιδρούν άμεσα στις τάσεις της αγοράς.



Εικόνα 2 : Η δομή δεδομένων του κύβου που χρησιμοποιείται στα συστήματα OLAP και κάποιες από τις βασικότερους μετασχηματισμούς που μπορούν να πραγματοποιηθούν.

2.4.3 Κατανεμημένη Αποθήκευση

Τα συστήματα κατανεμημένων βάσεων δεδομένων (DDBS) είναι ένα σύνολο λογικά δικτυωμένων βάσεων δεδομένων, που, από διαφορετικές τοποθεσίες, διατηρούν και διαχειρίζονται σύνολα δεδομένων τα οποία εμφανίζονται στον χρήστη ως μία ενιαία βάση δεδομένων [12]. Οι κατανεμημένες σχεσιακές βάσεις δεδομένων έχουν σχεδιαστεί για να χειρίζονται δεδομένα μεγάλης κλίμακας, συνήθως παραγόμενα από αποκεντρωμένες πηγές, κατανέμοντας τα δεδομένα σε πολλούς διακομιστές σε διάφορες γεωγραφικές τοποθεσίες, εξασφαλίζοντας υψηλή διαθεσιμότητα, ανοχή σε σφάλματα και επεκτασιμότητα. Αυτές οι βάσεις δεδομένων χρησιμοποιούν τεχνικές όπως η κατανομή (sharding), η αντιγραφή (replication) και οι αλγόριθμοι συναίνεσης (concesus algorighms) για να εξασφαλίσουν τη συνέπεια, κατανέμοντας αποτελεσματικά την αποθήκευση των δεδομένων. Έχουν υιοθετηθεί κυρίως από εφαρμογές μεγάλης κλίμακας που απαιτούν πρόσβαση σε δεδομένα με χαμηλή καθυστέρηση σε διαφορετικές περιοχές. Το Google Spanner, για παράδειγμα, παρέχει μια παγκόσμια κατανεμημένη σχεσιακή βάση δεδομένων με ισχυρή συνέπεια, καθιστώντας την ιδανική για πολυεθνικές εταιρείες που διαχειρίζονται συναλλαγές σε πραγματικό χρόνο. Η CockroachDB προσφέρει κλιμάκωση και ανθεκτικότητα έναντι αστοχιών, καθιστώντας την κατάλληλη για cloud-native εφαρμογές και αρχιτεκτονικές που επικεντρώνονται στην χρήση microservices. Από την άλλη, η NuoDB υιοθετεί μια υβριδική προσέγγιση, επιτρέποντας στις επιχειρήσεις να κλιμακώσουν δυναμικά τις βάσεις δεδομένων τους, διατηρώντας παράλληλα τη συμμόρφωση στα πρότυπα ACID παρά το κατανεμημένο περιβάλλον.



Εικόνα 3 : Μία ενδεικτική τοπολογία ενός κατανεμημένου συστήματος αποθήκευσης. [12]

2.4.4 Αποθήκευση σε περιβάλλοντα με περιορισμένους πόρους

Οι ενσωματωμένες (embedded) βάσεις δεδομένων αποτελούν συστήματα κυρίως ελαφριά και αυτοτελή, σχεδιασμένα να ενσωματώνονται απευθείας σε εφαρμογές λογισμικού αντί να λειτουργούν ως αυτόνομες υπηρεσίες. Αυτές οι βάσεις δεδομένων χρησιμοποιούνται συχνά σε εφαρμογές για κινητά, συσκευές IoT και εφαρμογές που λειτουργούν πρώτα εκτός σύνδεσης, όπου η απλότητα και η ελάχιστη χρήση πόρων είναι το κλειδί. Η SQLite είναι μια δημοφιλής επιλογή για την ανάπτυξη εφαρμογών που προορίζονται για κινητές συσκευές, τροφοδοτώντας εφαρμογές σε συσκευές iOS και Android, ενώ η H2 Database ενσωματώνεται συχνά σε εφαρμογές Java με ανάγκες για τοπική αποθήκευση. Η Firebird, επίσης, είναι μια ακόμα ενσωματωμένη βάση δεδομένων γνωστή για το μικρό της αποτύπωμα, γεγονός που την καθιστά ιδανική για εφαρμογές χαμηλής ισχύος με έντονους περιορισμούς αναφορικά με τους διαθέσιμους πόρους.

2.4.5 Αποθήκευση στην μνήμη (In-Memory)

Οι βάσεις δεδομένων στην μνήμη (In-Memory Databases) είναι συστήματα αποθήκευσης δεδομένων που διατηρούν όλα τα δεδομένα στην κύρια μνήμη ενός υπολογιστή, δηλαδή στην μνήμη τυχαίας προσπέλασης (RAM), αντί να αποθηκεύονται τα δεδομένα στο δίσκο. Λόγω αυτής της σχεδιαστικής απόφασης, οι συγκεκριμένες βάσεις δεδομένων επιτρέπουν εξαιρετικά ταχεία ανάκτηση και επεξεργασία δεδομένων καθώς, ως γνωστόν, το διάβασμα της μνήμης RAM είναι κατά πολύ γρηγορότερο σε σχέση με το διάβασμα από τον δίσκο. Η συγκεκριμένη αρχιτεκτονική είναι κατάλληλη για εφαρμογές που απαιτούν χαμηλή καθυστέρηση και υψηλή απόδοση, όπως η ανάλυση δεδομένων σε πραγματικό χρόνο, η προσωρινή αποθήκευση δεδομένων, οι χρηματοοικονομικές συναλλαγές και οι πίνακες κατάταξης παιχνιδιών. Μία από τις δημοφιλέστερες τέτοιες βάσεις δεδομένων είναι η Redis, που είναι ιδιαίτερα γνωστή για τις πληθώρα δομών δεδομένων και τις δυνατότητες προσωρινής αποθήκευσης που προσφέρει. Αντίστοιχα, ένα εξίσου γνωστό σύστημα αποθήκευσης δεδομένων στην μνήμη είναι το SAP HANA, το οποίο όπως προαναφέραμε έχει υιοθετηθεί ευρέως σε εφαρμογές που απαιτούν ανάλυση δεδομένων σε πραγματικό χρόνο. Παρόλα αυτά, η αποθήκευση στην μνήμη ενέχει μία σειρά από εγγενή μειονεκτήματα, όπως για παράδειγμα, το γεγονός ότι η μνήμη RAM έχει πολύ μικρότερο αποθηκευτικό χώρο σε σχέση με έναν δίσκο, ενώ παράλληλα τείνει να κοστίζει και πολύ περισσότερο από ότι ένας σκληρός δίσκος. Επιπλέον, συνήθως αυτά τα συστήματα δεν έχουν persistence καθώς κατά τον τερματισμό τους τα δεδομένα που διατηρούσαν διαγράφονται και απελευθερώνεται η μνήμη που είχε δεσμευτεί.

2.5 Μη Σχεσιακές (NoSQL) Βάσεις Δεδομένων

Οι μη σχεσιακές βάσεις δεδομένων είναι ένας τύπος βάσης που έχει σχεδιαστεί για να διαχειρίζεται μεγάλους όγκους κυρίως αδόμητων, ημιδομημένων ή και δομημένων δεδομένων, τους οποίους οι παραδοσιακές σχεσιακές βάσεις δεδομένων (SQL) αδυνατούν να διαχειριστούν αποτελεσματικά. Επιπλέον για τις NoSQL βάσεις, δεν υπάρχει η ανάγκη για συμμόρφωση με τους περιορισμούς των παραδοσιακών σχεσιακών βάσεων δεδομένων, όπως το πρότυπο ACID. Αυτό συμβαίνει γιατί η πλήρης επιβολή του ACID στις NoSQL βάσεις

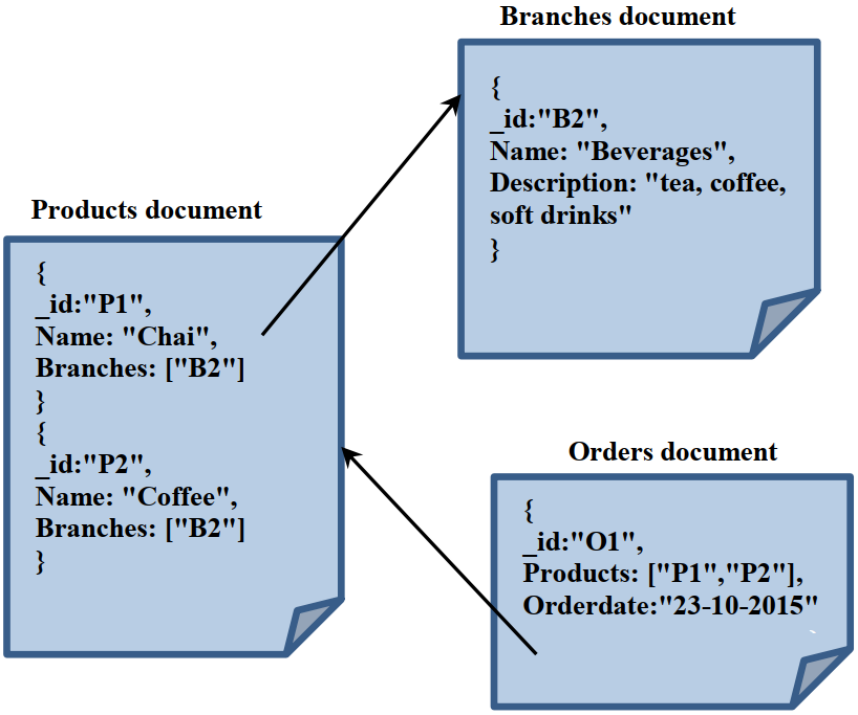
δεδομένων συχνά μειώνει τόσο την επεκτασιμότητα τους όσο και την απόδοσή τους. Αντ' αυτού, τα περισσότερα μη σχεσιακά συστήματα διαχείρισης δεδομένων ακολουθούν το μοντέλο BASE (Basically Available, Soft State, Eventually consistent). Τα κύρια χαρακτηριστικά του συγκεκριμένου μοντέλου είναι τα εξής, το Basically Available εγγυάται πως το σύστημα παραμένει λειτουργικό και διαθέσιμο ακόμα και σε περιπτώσεις απωλειών (failures), που σημαίνει πως κάποιες απαντήσεις του συστήματος μπορεί να βασίζονται σε παλαιότερα δεδομένα (outdated) ή να είναι ημιολοκληρωμένες (incomplete) αλλά το σύστημα να παραμένει ανταποκρίσιμο. Η ιδιότητα Soft State αναφέρεται στο γεγονός ότι το σύστημα μπορεί να αλλάζει με την πάροδο του χρόνου, καθώς η προώθηση των δεδομένων (data propagation) και η δημιουργία αντιγράφων (data replication) τους γίνονται με τρόπο ασύγχρονο. Τέλος, η ιδιότητα Eventually Consistent προάγει ότι το σύστημα δεν επιβάλλει αυστηρή συνέπεια ανά πάσα στιγμή αλλά η διάδοση των δεδομένων γίνεται σταδιακά στους διάφορους κόμβους, οδηγώντας τελικά στη συνέπεια. Είναι σημαντικό να αναφέρουμε πως οι τελευταίες δύο ιδιότητες αναφέρονται κυρίως σε κατανεμημένα μη σχεσιακά συστήματα διαχείρισης δεδομένων. Γενικά, οι μη σχεσιακές βάσεις δεδομένων είναι ειδικά σχεδιασμένες ώστε να διευκολύνεται η διανομή τους σε πολλούς κόμβους (nodes), που ενδεχομένως να είναι κατανεμημένοι σε διάφορα γεωγραφικά σημεία. Σε αντίθεση με τις παραδοσιακές σχεσιακές βάσεις δεδομένων, οι οποίες συχνά αντιμετωπίζουν προβλήματα συμφόρησης κατά το χειρισμό δεδομένων μεγάλης κλίμακας, οι βάσεις NoSQL μπορούν να διανέμουν τα δεδομένα σε συστοιχίες απρόσκοπτα, διασφαλίζοντας υψηλή διαθεσιμότητα και βελτιωμένη διαχείριση των τεράστιων όγκων δεδομένων. Στη σημερινή εποχή των μεγάλων δεδομένων , τα δεδομένα που παράγονται είναι υπέρογκα, αυτή η δυνατότητα των NoSQL συστημάτων είναι ζωτικής σημασίας για τη διατήρηση της απόδοσης και της αξιοπιστίας χωρίς σημαντικές αρχιτεκτονικές αλλαγές.

2.5.1

Οι βάσεις δεδομένων NoSQL κατηγοριοποιούνται σε τέσσερις κύριους τύπους, καθένας από τους οποίους είναι βελτιστοποιημένος για συγκεκριμένες περιπτώσεις χρήσης με βάση τον τρόπο αποθήκευσης και διαχείρισης των δεδομένων. Σε αυτή την υποενότητα θα αναλυθούν οι τέσσερις αυτούς τύπους παραθέτοντας αντίστοιχα παραδείγματα χρήσης για τον καθένα.

2.5.2 Αποθήκες Εγγράφων

Ο πρώτος τύπος είναι οι αποθήκες εγγράφων (document stores), οι οποίες αποθηκεύουν δεδομένα σε ευέλικτους και ημιδομημένους τύπους εγγράφων, όπως τα JSON, BSON και XML. Κάθε έγγραφο μπορεί να έχει διαφορετική δομή, επιτρέποντας τη δυναμική εξέλιξη του σχήματος. Συνεπώς ο συγκεκριμένος τύπος NoSQL συστημάτων είναι πολύ ευέλικτος, δεδομένου ότι δεν απαιτεί κάποιο αυστηρό, ενιαίο και προκαθορισμένο σχήμα. Επιπλέον, παρέχει την δυνατότητα να προσθέτουμε μεγάλο αριθμό διαφορετικών πεδίων σε ένα ή περισσότερα έγγραφα χωρίς να σπαταλάμε πολύτιμο χώρο προσθέτοντας τα ίδια κενά πεδία σε άλλα έγγραφα που δεν τα αξιοποιούν. Γενικότερα, τα έγγραφα ομαδοποιούνται σε συλλογές. Αν και μια συλλογή αποτελείται από πολλά έγγραφα, επιτρέπεται κάθε έγγραφο να έχει χρησιμοποιεί διαφορετικό σχήμα και διαφορετικούς τύπους αποθηκευμένων δεδομένων. Επιπλέον, κάθε έγγραφο κατέχει ένα μοναδικό αναγνωριστικό μέσα στην αντίστοιχη συλλογή του, έτσι ώστε να μπορεί να ανακτηθεί με ταχύτητα και ευκολία. Η *Εικόνα 5*, παρουσιάζει μία ενδεικτική δομή μιας αποθήκης εγγράφων. Οι ιδιότητες αυτές, καθιστούν τις αποθήκες εγγράφων ιδανικές για συστήματα διαχείρισης περιεχομένου, όπου τα άρθρα ή οι αναρτήσεις ιστολογίου μπορεί να έχουν ποικίλα μεταδεδομένα. Χρησιμοποιούνται επίσης ευρέως σε πλατφόρμες ηλεκτρονικού εμπορίου, όπου οι κατάλογοι προϊόντων συχνά περιέχουν στοιχεία με ποικίλα χαρακτηριστικά, και σε εφαρμογές πραγματικού χρόνου, όπως συστήματα συνομιλίας και εξατομικευμένες εμπειρίες χρηστών. Συνήθως, η αλληλεπίδραση με συστήματα αποθηκών εγγράφων γίνεται μέσω γλωσσών επερωτημάτων που βασίζονται στην δομή JSON, κάτι που μας επιτρέπει την αποτελεσματική αναζήτηση σε εμφωλευμένες δομές δεδομένων. Επίσης, κάποια τέτοια συστήματα υποστηρίζουν ευρετηριοποίηση (indexing), επιταχύνοντας ακόμη περισσότερο την ταχύτητα ανάκτησης των δεδομένων. Δημοφιλή παραδείγματα αποθηκευτικών χώρων εγγράφων περιλαμβάνουν τα συστήματα MongoDB, CouchDB, Firebase Firestore και Amazon DocumentDB.

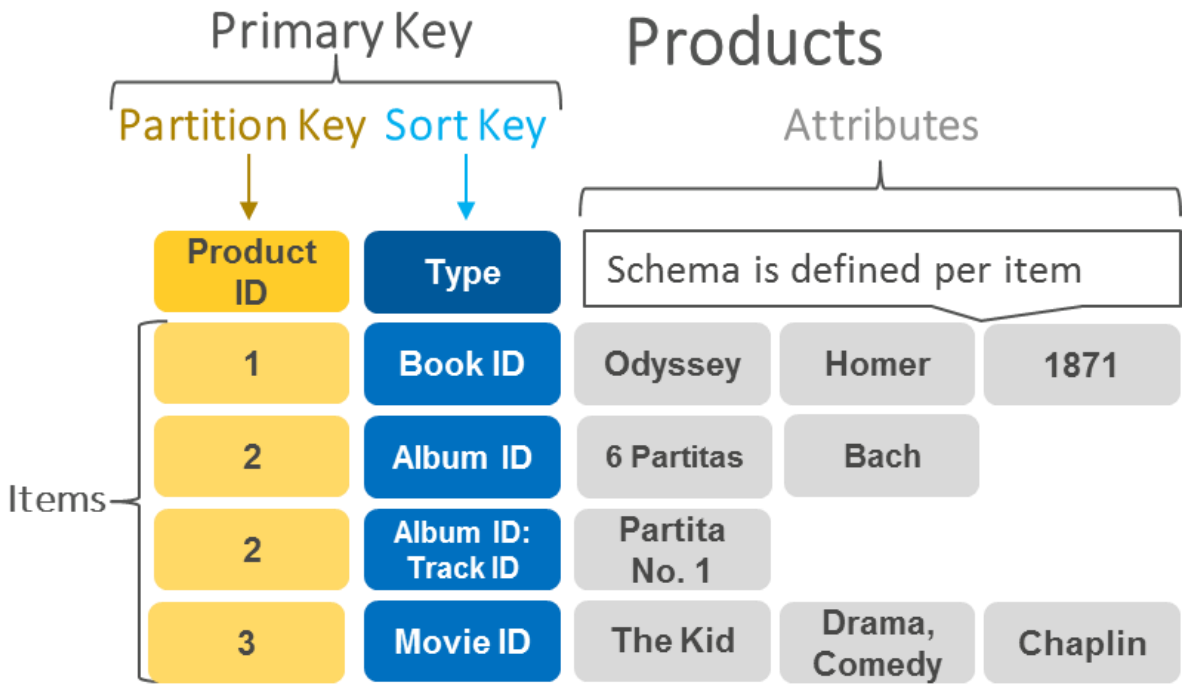


Εικόνα 4 : Γραφική αναπαράσταση της οργάνωσης μιας αποθήκης εγγράφων.

2.5.3 Αποθήκες Κλειδιών-Τιμών

Ο δεύτερος τύπος είναι οι αποθήκες κλειδιών-τιμών (key-value stores), οι οποίες αποθηκεύουν τα δεδομένα ως ζεύγη κλειδιών-τιμών, παρόμοιας μορφής με τις δομές δεδομένων του λεξικού και του πίνακα κατακερματισμού. Κάθε κλειδί είναι μοναδικό και η αντίστοιχη τιμή του μπορεί να είναι μια απλή συμβολοσειρά, ένα αντικείμενο JSON ή ακόμη και δεδομένα σε δυαδική μορφή (bits). Όπως φαίνεται και στην *Εικόνα 6*, τα δεδομένα οργανώνονται βάση ενός πρωτεύοντος κλειδιού που αποτελείται από ένα κλειδί κατάτμησης (partition key) και ενός προαιρετικού κλειδί ταξινόμησης (sort key). Κατά αυτόν τον τρόπο, έχουμε αποτελεσματική και ταχεία ανάκτηση δεδομένων αλλά και την δυνατότητα ορισμού ενός ευέλικτου σχήματος, όπου διαφορετικοί τύποι προϊόντων (βιβλία, άλμπουμ, ταινίες) μπορούν να έχουν μοναδικά σύνολα χαρακτηριστικών. Για παράδειγμα, στην *Εικόνα 6*, ένα

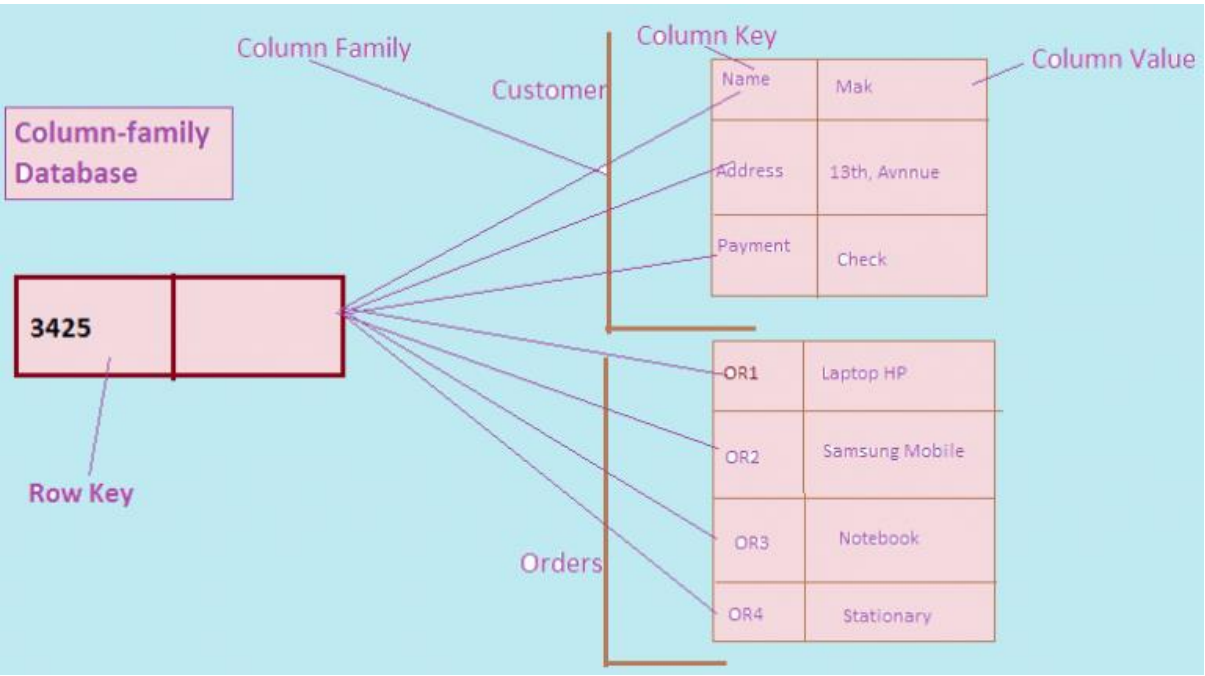
βιβλίο μπορεί να αποθηκεύει χαρακτηριστικά όπως ο τίτλος, ο συγγραφέας και το έτος έκδοσης, ενώ ένα άλμπουμ μπορεί να περιέχει πληροφορίες για τον συνθέτη και τα κομμάτια που το αποτελούν. Η χρήση του κλειδιού κατάτμησης εξασφαλίζει ότι τα στοιχεία με το ίδιο κλειδί αποθηκεύονται μαζί, ενώ το κλειδί ταξινόμησης επιτρέπει την αποτελεσματική αναζήτηση σχετικών δεδομένων, όπως η ανάκτηση όλων των κομματιών για ένα άλμπουμ. Αυτή η προσέγγιση χωρίς σχήμα καθιστά τις αποθήκες κλειδιών-τιμών ιδανικές για εφαρμογές που απαιτούν επεκτασιμότητα, ευελιξία και αναζητήσεις υψηλής απόδοσης. Αναλυτικότερα, ο συγκεκριμένος τύπος βάσης δεδομένων NoSQL είναι κατάλληλος για συστήματα προσωρινής αποθήκευσης, που πραγματεύονται δεδομένα με απαιτήσεις συχνής πρόσβασης, όπως οι συνεδρίες χρηστών σε μία ιστοσελίδα που πρέπει να αποθηκεύονται και να ανακτώνται γρήγορα. Χρησιμοποιείται επίσης συνήθως για καλάθια αγορών στο ηλεκτρονικό εμπόριο, όπου οι προσωρινές επιλογές του χρήστη πρέπει να διατηρούνται αποτελεσματικά. Επιπλέον, οι αποθήκες κλειδιών-τιμών διακρίνονται και για την παρακολούθηση πινάκων κατάταξης σε πραγματικό χρόνο για εφαρμογές τυχερών παιχνιδιών. Ορισμένα γνωστά παραδείγματα αποθηκευτικών χώρων κλειδιών-τιμών είναι το Redis, το Amazon DynamoDB, το Riak και το BerkleyDB.



Εικόνα 5 : Γραφική αναπαράσταση την οργάνωσης μιας αποθήκης κλειδιών-τιμών.

2.5.4 Αποθήκες Τύπου Στήλης

Μια άλλη σημαντική κατηγορία είναι οι αποθήκες τύπου στήλης (column-family stores), οι οποίες αποθηκεύουν τα δεδομένα σε στήλες και όχι σε γραμμές, επιτρέποντας την αποτελεσματική αποθήκευση και ανάκτηση μεγάλων συνόλων δεδομένων. Σε αντίθεση με τις σχεσιακές βάσεις δεδομένων που αποθηκεύουν τα δεδομένα σε γραμμές και πίνακες, οι αποθήκες τύπου στήλης (column-family stores) οργανώνουν τα δεδομένα σε γραμμές με πολλαπλές οικογένειες στηλών.



Εικόνα 6 : Γραφική αναπαράσταση την οργάνωσης μιας αποθήκης κλειδιών-τιμών.

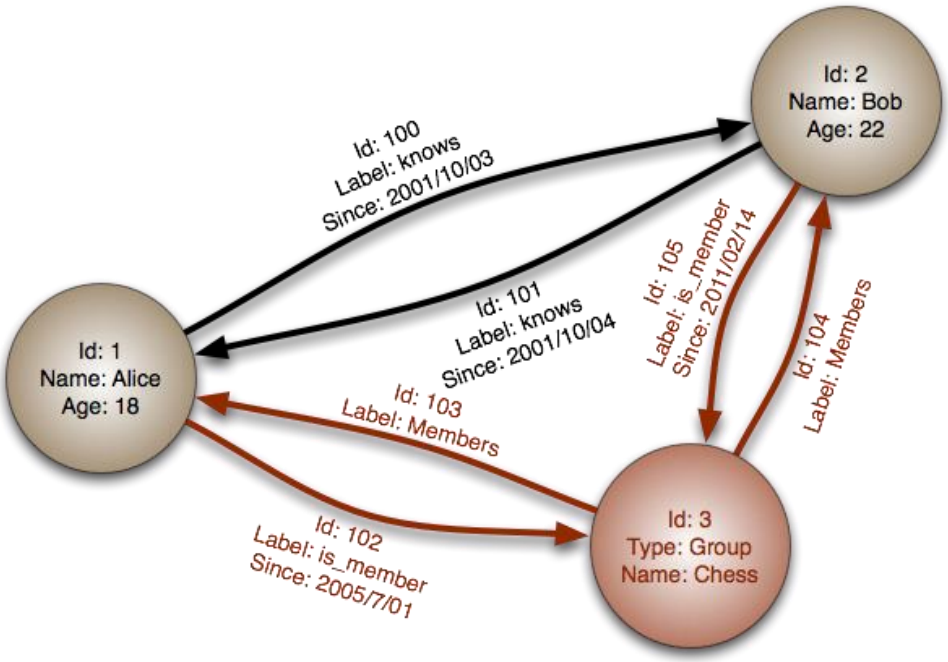
Όπως φαίνεται και στην *Εικόνα 7*, σε κάθε μπλοκ δεδομένων αντιστοιχεί μια γραμμή, που έχει κάποιο αναγνωριστικό κλειδί που συνδέεται με πολλές οικογένειες στηλών και κατ' επέκταση σε πολλές στήλες. Οι οικογένειες στηλών είναι ομάδες στηλών που σχετίζονται μεταξύ τος και αποθηκεύονται μαζί. Στην *Εικόνα 7*, έχουμε δύο οικογένειες στηλών, την οικογένεια *Πελάτης* που αποθηκεύει τα προσωπικά στοιχεία ενός χρήστη και την οικογένεια *Παραγγελίες* που αποθηκεύει τα αντικείμενα που αγοράστηκαν από έναν χρήστη, καθένα από τα οποία προσδιορίζεται από ένα κλειδί παραγγελίας. Αντίστοιχα, κάθε στήλη μιας οικογένειας στηλών αποτελεί ένα ζεύγος κλειδιού-τιμής όπου το κλειδί είναι κάποιο

γνώρισμα και η τιμή του είναι η παρατήρηση που αντιστοιχεί στην αντίστοιχη σειρά της βάσης. Εν αντιθέση με τις παραδοσιακές σχεσιακές βάσεις δεδομένων, οι αποθήκες οικογένειας στηλών επιτρέπουν ένα ευέλικτο σχήμα όπου οι γραμμές δεν χρειάζεται να έχουν τις ίδιες στήλες ενώ νέες στήλες μπορούν να προστεθούν δυναμικά ανά γραμμή και οποτεδήποτε. Το συγκεκριμένο μοντέλο είναι βελτιστοποιημένο για γρήγορες αναγνώσεις και εγγραφές, καθιστώντας το ιδιαίτερα ωφέλιμο για την ανάλυση δεδομένων μεγάλης κλίμακας για σκοπούς business intelligence. Επίσης, αυτά τα συστήματα είναι ιδανικα και για το χειρισμό δεδομένων χρονοσειρών, όπως τα αρχεία καταγραφής αισθητήρων IoT και η παρακολούθηση χρηματοοικονομικών δεδομένων. Επιπλέον, έχουν βασικό ρόλο στα συστήματα συστάσεων, αποθηκεύοντας και αναλύοντας τις προτιμήσεις και τα δεδομένα συμπεριφοράς των χρηστών με εξαιρετικές εγγυήσεις ταχύτητας. Κάποια από τα δημοφιλέστερα συστήματα αποθήκευσης τύπου στήλης είναι τα Apache Cassandra, Apache HBase και ScyllaDB.

2.5.5 Βάσεις Δεδομένων Γράφων

Ο τέταρτος τύπος είναι οι βάσεις δεδομένων γράφων (graph databases), οι οποίες αποθηκεύουν δεδομένα ως κόμβους και ακμές, γεγονός που τις καθιστά ιδανικές για εφαρμογές που απαιτούν πολύπλοκες σχέσεις μεταξύ σημείων δεδομένων. Κάθε κόμβος αντιστοιχεί σε ένα μπλοκ δεδομένων, το οποίο συνδέεται με άλλους κόμβους, με τη σύνδεση αυτή είναι να ονομάζεται ακμή και να χαρακτηρίζει κάποια σχέση ανάμεσα στο δύο κόμβους. Αναλυτικότερα, μπορούμε να σκεφτούμε τους κόμβους ως έγγραφα που αποθηκεύουν ιδιότητες με τη μορφή αυθαίρετων ζευγών κλειδιών-τιμών. Τα κλειδιά είναι συμβολοσειρές και οι τιμές αυθαίρετοι τύποι δεδομένων. Οι σχέσεις συνδέουν και δομούν τους κόμβους. Μια σχέση έχει πάντα μια κατεύθυνση, μια ετικέτα, έναν κόμβο αρχής και έναν κόμβο τέλους - δεν υπάρχουν περιφερόμενες σχέσεις. Μαζί, η κατεύθυνση και η ετικέτα μιας σχέσης προσθέτουν σημασιολογική σαφήνεια στη δόμηση των κόμβων. Όπως και οι κόμβοι, οι σχέσεις μπορούν επίσης να έχουν ιδιότητες. Η δυνατότητα προσθήκης ιδιοτήτων στις σχέσεις είναι ιδιαίτερα χρήσιμη για την παροχή πρόσθετων μεταδεδομένων, την προσθήκη πρόσθετης σημασιολογίας στις σχέσεις (συμπεριλαμβανομένης της ποιότητας και του

βάρους) και για τον περιορισμό των ερωτημάτων κατά την εκτέλεση. Η *Εικόνα 8* παρακάτω δείχνει ένα απλοϊκό παράδειγμα μοντελοποίησης δεδομένων ως έναν κατευθυνόμενο γράφο.



Εικόνα 7 : Γραφική αναπαράσταση της μοντελοποίησης δεδομένων ως κατευθυνόμενο γράφο.

Οι βάσεις δεδομένων γράφων είναι ιδιαίτερα χρήσιμες όταν τα δεδομένα που πρέπει να αποθηκευτούν είναι συνειρμικά (associative) και ειδικά όταν η σχέση μεταξύ δύο μπλοκ δεδομένων έχει μεγάλη σημασία ή/και αξία. Συνήθως χρησιμοποιούνται έντονα σε εφαρμογές κοινωνικών δικτύων, όπου πρέπει να γίνεται αποτελεσματική διαχείριση και αναζήτηση σχέσεων όπως οι διασυνδέσεις φίλων/ακολούθων. Άλλη μία σημαντική χρήση των συγκεκριμένων συστημάτων είναι η ανίχνευση απατών, καθώς μπορούν να εντοπίσουν συναλλαγές σε πολλαπλούς λογαριασμούς και να εντοπίσουν εύκολα και γρήγορα ύποπτα μοτίβα. Επιπλέον, σε συστήματα συστάσεων, όπου δημιουργούνται εξατομικευμένες προτάσεις προϊόντων ή περιεχομένου με βάση τις σχέσεις μεταξύ χρηστών και αντικειμένων, οι συγκεκριμένες βάσεις χρησιμοποιούνται κατά κόρον. Η μοντελοποίηση των δεδομένων ως κατευθυνόμενο γράφο προσφέρει το πρόσθετο πλεονέκτημα ότι μπορεί να αποτελεσματικά να γίνει ανάλυση των δεδομένων χρησιμοποιώντας εδραιωμένους

αλγοριθμους, μιας και η θεωρία γράφων είναι μία πολύ-μελετημένη ερευνητική περιοχή. Κάποια από τα πιο γνωστά συστήματα βάσεων δεδομένων γράφων είναι τα Neo4j, ArangoDB και Amazon Neptune.

2.6 Σύγκριση Περιπτώσεων Χρήσης Σχεσιακών και Μη Βάσεων Δεδομένων

Χαρακτηριστικά	***Σχεσιακές (SQL) Βάσεις Δεδομένων***	***Μη Σχεσιακές (NoSQL) Βάσεις Δεδομένων***
Σχήμα	Αυστηρά προκαθορισμένα σχήματα	Δυναμικά, ευέλικτα ή και καθόλου σχήματα
Τρόπος Αλληλεπίδρασης	Χρήση SQL	Χρήση APIs
Κλιμακωσιμότητα	Κάθετη κλιμάκωση	Οριζόντια Κλιμάκωση
Συνέπεια	Πρότυπο ACID	Πρότυπο BASE
Δομή Δεδομένων	Πίνακες (Σχέσεις)	Διάφορα μοντέλα (στήλης, εγγράφων, γράφοι, κτλ.)
Τύποι Δεδομένων	Περιορισμένοι τύποι ανά σύστημα	Δυναμική ανάθεση τύπων ανάλογα το μοντέλο
Join	Βελτιστοποιημένοι τελεστές Join	Απουσία τελεστών, εφαρμογή λογικής με κώδικα

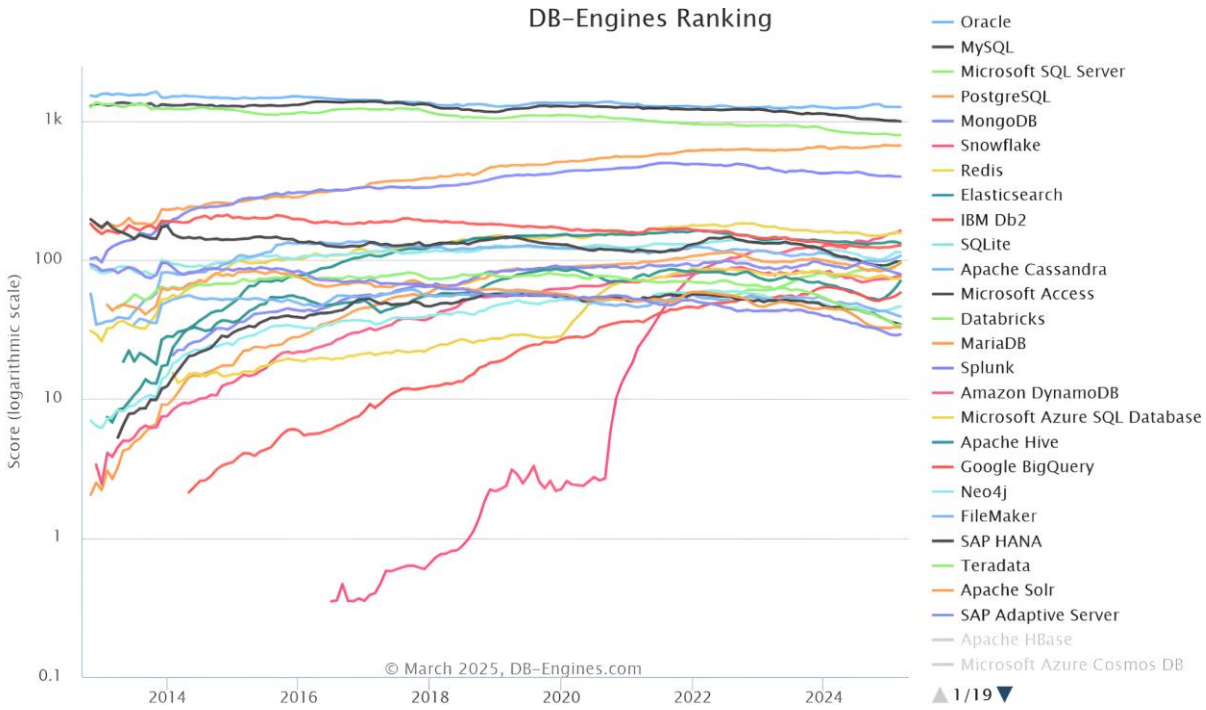
Πίνακας 1 : Σύνοψη διαφορών μεταξύ συστημάτων σχεσιακών και μη βάσεων δεδομένων.

Τα συστήματα σχεσιακών βάσεων δεδομένων αποθηκεύουν δεδομένα σε δομημένους πίνακες με προκαθορισμένες σχέσεις. Συμμορφώνονται με το πρότυπο ACID για να διασφαλίσουν την ακεραιότητα των δεδομένων και υπερέχουν στην επεξεργασία δομημένων δεδομένων, όπου οι λειτουργίες JOIN και τα σύνθετα φίλτρα επιτρέπουν στους χρήστες να ανακτούν, να συνδυάζουν και να επεξεργάζονται δεδομένα με μεγάλη ακρίβεια. Αντίθετα, τα NoSQL συστήματα προσφέρουν, όπως είδαμε, διαφορετικά μοντέλα αποθήκευσης, όπως βάσεις δεδομένων κλειδιών-τιμών, εγγράφων, τύπου στήλης και γράφων. Παρέχουν υψηλή επεκτασιμότητα και γρήγορες λειτουργίες ανάγνωσης/εγγραφής, καθιστώντας τες καταλληλότερες για εφαρμογές πραγματικού χρόνου, αναλύσεις μεγάλων δεδομένων και συστήματα διαχείρισης περιεχομένου. Σε αντίθεση με τις βάσεις δεδομένων SQL, τα συστήματα NoSQL δίνουν προτεραιότητα στη διαθεσιμότητα και την ανοχή κατατμήσεων, συχνά εις βάρος της αυστηρής συνέπειας, ακολουθώντας το πρότυπο BASE. Παρακάτω θα δούμε ορισμένες περιπτώσεις χρήσης και ποιος τύπος συστήματος φαίνεται να είναι ο ιδανικότερος για την καθεμία.

Σε περιπτώσεις που εμπίπτουν στον τομέα του ηλεκτρονικού εμπορίου αλλά και των χρηματοοικονομικών, οι σχεσιακές βάσεις δεδομένων τείνουν να είναι η προτιμότερη επιλογή. Αυτό συμβαίνει γιατί τέτοιες εφαρμογές απαιτούν αυστηρή συναλλακτική συνέπεια και σχεσιακή ακεραιότητα. Για παράδειγμα, σε ένα τραπεζικό σύστημα, μια μεταφορά χρημάτων θα πρέπει να διασφαλίζει ότι οι χρεώσεις και οι πιστώσεις υποβάλλονται σε ορθή επεξεργασία χωρίς να χάνεται η παρακολούθηση των συναλλαγών. Αντίστοιχα, σε ένα σύστημα ηλεκτρονικού εμπορίου, όταν ένας πελάτης πραγματοποιεί μια παραγγελία, η πλατφόρμα πρέπει να διασφαλίζει ότι τα αρχεία απογραφής, πληρωμής και παραγγελίας ενημερώνονται με ακρίβεια. Ας υποθέσουμε ότι ένας χρήστης αγοράζει τον τελευταίο διαθέσιμο φορητό υπολογιστή από ένα ηλεκτρονικό κατάστημα. Η βάση δεδομένων πρέπει να μειώσει αμέσως την καταμέτρηση των αποθεμάτων, να επεξεργαστεί την πληρωμή και να δημιουργήσει μια επιβεβαίωση παραγγελίας. Εάν πολλοί πελάτες προσπαθήσουν να αγοράσουν ταυτόχρονα τον ίδιο φορητό υπολογιστή, μια βάση δεδομένων SQL διασφαλίζει ότι μόνο μία συναλλαγή θα πετύχει, αποτρέποντας την υπερπώληση. Επιπλέον, οι σχεσιακές βάσεις επιλέγονται για την υποστήριξη συστημάτων ERP και CRM, όπου οι επιχειρήσεις που διαχειρίζονται δομημένες πληροφορίες πελατών και προϊόντων, επωφελούνται από την ικανότητα των σχεσιακών βάσεων δεδομένων να μοντελοποιούν σχέσεις, να εφαρμόζουν περιορισμούς και αν είναι συνεπής.

Αντίθετα, σε εφαρμογές δεδομένων πολύ μεγάλης κλίμακας και με απαιτήσεις ανάλυσης σε πραγματικό χρόνο, τα συστήματα μη σχεσιακών βάσεων δεδομένων προτιμώνται. Συστήματα όπως το Apache Cassandra και το MongoDB, είναι καταλληλότερα για εφαρμογές υψηλής απόδοσης, όπως η συλλογή δεδομένων IoT, οι τροφοδοσίες κοινωνικών μέσων και τα συστήματα συστάσεων. Αυτές οι βάσεις δεδομένων χειρίζονται αποτελεσματικά την κατανεμημένη αποθήκευση μεγάλης κλίμακας και μπορούν να επεξεργάζονται τεράστιους όγκους δεδομένων διατηρώντας παράλληλα υψηλή διαθεσιμότητα. Για παράδειγμα, σε μια πλατφόρμα κοινωνικής δικτύωσης όπως το Facebook ή το Instagram, εκατομμύρια χρήστες δημιουργούν νέες αναρτήσεις, σχόλια, φωτογραφίες και κοινοποιήσεις κάθε δευτερόλεπτο. Η αποθήκευση και η ανάκτηση αυτού του μεγάλου όγκου ταχέως μεταβαλλόμενων δεδομένων σε πραγματικό χρόνο απαιτεί μια εξαιρετικά κλιμακούμενη και κατανεμημένη βάση δεδομένων. Αντίστοιχα, εφαρμογές όπως blogs, πλατφόρμες ειδήσεων και υπηρεσίες ροής πολυμέσων απαιτούν ευέλικτες δομές δεδομένων για την προσαρμογή και την αποθήκευση διαφορετικών τύπων περιεχομένου. Οι μη σχεσιακές βάσεις δεδομένων εγγράφων NoSQL, επιτρέπουν δυναμικές ενημερώσεις σχήματος, διευκολύνοντας την αποθήκευση και ανάκτηση διαφορετικών τύπων δεδομένων, όπως κείμενο, εικόνες και βίντεο, χωρίς προκαθορισμένους περιορισμούς.

Εν κατακλείδι, οι σχεσιακές βάσεις δεδομένων υπερέχουν όταν απαιτείται ακεραιότητα, συνέπεια και πολύπλοκες σχέσεις μεταξύ δεδομένων. Χάρις τη συμμόρφωσή τους με το πρότυπο ACID, προσφέρουν αξιόπιστη διαχείριση συναλλαγών και ακριβή ανάκτηση πληροφοριών, καθιστώντας τις κατάλληλες για εφαρμογές όπου η ακρίβεια και η διασφάλιση της συνοχής είναι κρίσιμες. Από την άλλη, οι μη σχεσιακές βάσεις δεδομένων προσφέρουν μεγαλύτερη ευελιξία, υψηλή επεκτασιμότητα και ταχύτητα, καθιστώντας τις ιδανικές για εφαρμογές που απαιτούν δυναμικά σχήματα, κατανεμημένη αποθήκευση και γρήγορη επεξεργασία μεγάλων όγκων δεδομένων. Η επιλογή μεταξύ SQL και NoSQL εξαρτάται από τις ανάγκες της εκάστοτε εφαρμογής, με κριτήρια όπως η δομή των δεδομένων, οι απαιτήσεις συνέπειας και η ανάγκη για απόδοση και διαθεσιμότητα.



Εικόνα 8 : Οι 25 δημοφιλέστερες βάσεις δεδομένων από το 2014 έως σήμερα. [1]

2.7 Σύγκριση Απόδοσης Σχεσιακων και Μη Βάσεων Δεδομένων

Στην βιβλιογραφία, η σύγκριση ανάμεσα στα διάφορα συστήματα αποθήκευσης δεδομένων όσον αφορά την ταχύτητα εκτέλεσης επερωτημάτων, έχει μελετηθεί δεόντως. Σε αυτή την ενότητα θα υπογραμμίσουμε μερικές μελέτες πάνω στην αποδοτικότητα των διάφορων αυτών συστημάτων.

Αρχικά, η [13] συγκρίνει την μη σχεσιακή βάση MongoDB με την σχεσιακή βάση Microsoft SQL Server σε τέσσερα πειράματα με αυξανόμενο αριθμό δεδομένων, μετρώντας την απόδοση σε τρεις βασικές λειτουργίες: ταχύτητα εισαγωγής, ενημέρωσης και ανάκτησης δεδομένων. Οι λειτουργίες ενημέρωσης κατηγοριοποιήθηκαν περαιτέρω ανάλογα με το αν βασίζονταν σε χαρακτηριστικά κλειδιά ή μη, ενώ οι λειτουργίες επιλογής περιλάμβαναν επερωτήματα τόσο απλά (ανάκτηση ενός αντικειμένου) όσο και σύνθετα (με join και συναρτήσεις συνάθροισης). Τα αποτελέσματα έδειξαν τη MongoDB να υπερέχει έναντι του MS SQL Server τόσο στις εισαγωγές και ενημερώσεις βάσει κλειδιού, όσο και στις απλές

επερωτήσεις. Ωστόσο, το MS SQL Server ήταν ταχύτερο στις ενημερώσεις και τα επερωτήματα που δεν βασίζονται σε κλειδιά, καθώς και για συναρτήσεις συνάθροισης, όπου η εξάρτηση της MongoDB στο MapReduce επέφερε σημαντική μείωση στην επίδοσή της.

Αντίστοιχα, η [14] αξιολογεί την επίδοση πέντε από τα δημοφιλέστερα NoSQL συστήματα, τα Redis, MondoDB, Cassandra, HBase και OrientDB χρησιμοποιώντας το Yahoo! Cloud Serving Benchmark (YCSB) [15]. Τα πειράματα αφορούσαν 600.000 δεδομένα και τρείς φόρτους εργασίας: A (50% αναγνώσεις, 50% ενημερώσεις), C (100% αναγνώσεις) και H (100% ενημερώσεις). Τα αποτελέσματα έδειξαν το Redis, μία αποθήκη κλειδιών-τιμών με αποθήκευση στην μνήμη, να έχει σταθερά την καλύτερη απόδοση σε όλα τα φορτία, εκτός του τελευταίου. Αμέσως επόμενα ήταν τα HBase και Cassandra, που είναι αποθήκες τύπου στήλης και είχαν την καλύτερη επίδοση στο φορτίο H. Τέλος, ακολούθησαν τα MondoDB και OrientDB, που είναι αποθήκες εγγράφων, τα οποία ήταν συστηματικά τελευταία σε όλα τα πειράματα.

Η [16] επιχειρεί μία πειραματική αξιολόγηση των NoSQL βάσεων δεδομένων Redis, Memcached, Voldemort, OrientDB, MongoDB, HBase και Cassandra. Χρησιμοποιεί το YCSB [15] με έμφαση σε φορτία ανάγνωσης, εγγραφής και συνδυασμό των δύο. Τα αποτελέσματα δείχνουν τα Redis και Memcached, να υπερτερούν σε λειτουργίες ανάγνωσης, με τα HBase και Cassandra να ακολουθούν και τα MongoDB και OrientDB να είναι τελευταία. Αντίστοιχα με το [14], τα HBase και Cassandra υπερτερούν σε λειτουργίες εγγραφής, λόγω του κατανεμημένου σχεδιασμού τους και των βελτιστοποιημένων διαδρομών εγγραφής που υποστηρίζουν. Τα συστήματα MongoDB και OrientDB, ήταν και πάλι τα λιγότερα αποδοτικά σε όλα τα φορτία εργασίας.

Τέλος, η [17] συγκρίνει τρείς NoSQL αποθήκες εγγράφων, τις MongoDB, CouchDB και Couchbase, έναντι τρειών σχεσιακών βάσεων δεδομένων, των MS SQL Server, MySQL και PostgreSQL. Τα πειράματα αφορούν λειτουργίες CRUD (Create, Read, Update, Delete) σε κεντρικοποιημένα και κατανεμημένα περιβάλλοντα και εξετάζονται διάφορα φορτία εργασίας. Αναφορικά με τα κεντρικοποιημένα πειράματα, τα συστήματα SQL ήταν αποδοτικότερα, κυρίως λόγω των ευρετηρίων και των βελτιστοποιημένων μηχανών εκτέλεσης. Αντίθετα, όταν τα δεδομένα είναι κατανεμημένα, τα συστήματα NoSQL έδειξαν καλύτερη επεκτασιμότητα και αποδοτικότερη διαχείριση του φορτίου εργασίας.

Γενικά παρατηρούμε πως, όπως έχουμε υπογραμμίσει και προηγουμένως, τα διάφορα NoSQL συστήματα εκτός της ετερογένειας τους ως προς τον τρόπο λειτουργίας,

είναι και αρκετά ετερογενή ως προς την αποδοτικότητά τους. Το συμπέρασμα στο οποίο καταλήγουμε είναι ότι οι σχεσιακές βάσεις δεδομένων υπερέχουν σε φορτία εργασίας δομημένων δεδομένων σε κεντρικοποιημένα περιβάλλοντα, ενώ τα συστήματα NoSQL προσφέρουν καλύτερες επιδόσεις σε κατανεμημένα σενάρια, ειδικά όταν χρειάζεται ευελιξία όσον αφορά το σχήμα. Αξίζει να σημειωθεί ότι η επιλογή μεταξύ SQL ή NoSQL αλλά και μεταξύ των διαφόρων συστημάτων κάθε κατηγορίας, είναι άρρηκτα συνδεδεμένη με το φόρτο εργασίας, την δομή των δεδομένων, τις ανάγκες συνέπειας και την αναμενόμενη ανάπτυξη και τις ανάλογες ανάγκες για επεκτασιμότητα.

3 Επέκταση του συστήματος Noda

3.1 Υπάρχοντα συστήματα παρόμοια με το Noda

Στην βιβλιογραφία υπάρχουν εκτενείς αναφορές σε συστήματα που έχουν σχεδιαστεί με σκοπό να διαχειρίζονται δεδομένα που είναι ετερογενή ως προς τον τρόπο μοντελοποίησής τους. Αντλώντας από το [3], που συγκρίνει τέτοια συστήματα όπως τα ArangoDB [8] και OrientDB [9], παρατηρούμε πως τα συστήματα αυτά συνήθως μπορούν να διαχειριστούν ένα μόνο ένα υποσύνολο των μοντελοποιήσεων που μπορούμε να εφαρμόσουμε σε δεδομένα. Συγκεκριμένα, το ArangoDB υποστηρίζει μόνο αποθήκες εγγράφων, κλειδών-τιμής και γράφων, υπό το ίδιο σύστημα, ενώ μπορεί να διαχειριστεί και γεωχωρικά δεδομένα. Αντίστοιχα το σύστημα OrientDB, συνδυάζει μόνο αποθήκες εγγράφων, κλειδών-τιμής και αντικειμενοστραφή μοντέλα στο σύστημά του. Εν αντιθέση λοιπόν με αυτά τα συστήματα το NoDA δεν αντιμετωπίζει τέτοιους περιορισμούς, καθώς είναι σχεδιασμένο ώστε να λειτουργεί ως μια διεπαφή πάνω από οποιοδήποτε σύστημα NoSQL. Μία άλλη πτυχή συστημάτων που σχετίζεται σημαντικά με το δικό μας είναι τα λεγόμενα πολυκαταστήματα (polystores), που σχεδιάζονται ώστε να λειτουργούν πάνω από ήδη υπάρχοντα και ετερογενή συστήματα αποθήκευσης δεδομένων. Για παράδειγμα το σύστημα BigDAWG ([4], [5]) επιτρέπει την απρόσκοπτη υποβολή επερωτημάτων σε διαφορετικές μηχανές αποθήκευσης, χρησιμοποιώντας σημασιολογικές νησίδες (semantic islands) και λειτουργίες CAST/SCOPE για την διασφάλιση της διαλειτουργικότητας μεταξύ των διαφορετικών μοντέλων δεδομένων. Παρομοίως, αντλώντας από το [6], το σύστημα polystore που προτείνεται χρησιμοποιεί μία σημασιολογική προσέγγιση βασισμένη στην πρόσβαση δεδομένων βάση οντολογιών (Ontology-Based Data Access - OBDA), υποστηρίζοντας, κατά αυτόν τρόπο, καθολική πρόσβαση σε δεδομένα διαφορετικών μοντέλων όπως στατικά, ροής και γεωχωρικά, ενώ παράλληλα παρέχει σημασιολογική αναδιατύπωση και αυτόματο σχεδιασμό των επερωτημάτων, αξιοποιώντας τις οντολογίες. Γενικά τα προαναφερθέντα συστήματα προσφέρουν, όπως και το NoDA, γλώσσες επερωτημάτων εμπνευσμένες από την SQL για την αλληλεπίδραση με τα διάφορα συστήματα αποθήκευσης, ωστόσο πρόκειται για εξαιρετικά πολύπλοκα και σύνθετα συστήματα που έχουν σχεδιαστεί να λειτουργούν υπό πολύ συγκεκριμένες συνθήκες.

Αντιθέτως, η αρχιτεκτονική του συστήματος NoDA είναι σημαντικά απλούστερη, διευκολύνοντας σημαντικά στην επεκτασιμότητα του εν λόγου συστήματος και σε άλλους τύπους βάσεων δεδομένων.

3.2 Αρχιτεκτονική και επεκτασιμότητα του Noda

3.2.1 Αρχιτεκτονική του Noda

Η αρχιτεκτονική του NoDA (NoSQL Data Access) έχει σχεδιαστεί με στόχο να παρέχει έναν ενοποιημένο και επεκτάσιμο τρόπο πρόσβασης σε ετερογενείς NoSQL βάσεις δεδομένων. Στο υψηλό επίπεδο, τα βασικά της δομικά στοιχεία περιλαμβάνουν: (α) NoSQL Connector, ο οποίος είναι υπεύθυνος για τη δημιουργία και τη διαχείριση της σύνδεσης με την εκάστοτε NoSQL βάση, (β) NoSQL Operators, οι οποίοι ορίζουν τις αφηρημένες λειτουργίες προσπέλασης δεδομένων που πρέπει να υλοποιηθούν για κάθε τύπο βάσης, (γ) την ενσωμάτωση με το Apache Spark, που επιτρέπει την παράδοση δεδομένων σε μορφή DataFrame για περαιτέρω επεξεργασία στο πλαίσιο ενός Spark context, και (δ) το SQL υποσύστημα, το οποίο καθιστά δυνατή τη διατύπωση ερωτημάτων σε δηλωτική μορφή (SQL) και τη μετάφρασή τους σε τελεστές δεδομένων. Έτσι, ο χρήστης μπορεί να αλληλεπιδράσει με μία NoSQL βάση είτε προγραμματιστικά, αξιοποιώντας τους γενικούς τελεστές δεδομένων (πιο κατάλληλο για προγραμματιστές), είτε με ερωτήματα SQL (πιο φιλικό σε αναλυτές δεδομένων και επιστήμονες).

Για κάθε NoSQL βάση που υποστηρίζεται, αναπτύσσεται ένας αντίστοιχος connector, ο οποίος αρχικοποιείται με τις απαραίτητες παραμέτρους σύνδεσης, όπως διεύθυνση IP, θύρα και διαπιστευτήρια πρόσβασης (όνομα χρήστη, κωδικό). Ανάλογα με το εκάστοτε σύστημα, είναι δυνατόν να προστεθούν εξειδικευμένες παράμετροι που δεν συναντώνται σε άλλες βάσεις. Για παράδειγμα, κατά τη σύνδεση με Redis μπορεί να οριστεί ένα όριο μέγιστου χρόνου αναμονής, ενώ στη σύνδεση με Neo4j μπορεί να αξιοποιηθεί μηχανισμός πιστοποίησης kerberos με κωδικοποιημένο εισιτήριο. Οι παράμετροι αυτές υλοποιούνται αξιοποιώντας τις αντίστοιχες βιβλιοθήκες (Java/Scala client libraries) που παρέχει η κάθε βάση.

Το NoDA (NoSQL Data Access) framework έχει σχεδιαστεί με τρόπο που να επιτρέπει την εύκολη ενσωμάτωση νέων συστημάτων βάσεων δεδομένων όπως τα MongoDB, CouchDB, HBase, Cassandra και Redis, μέσω μιας κοινής διεπαφής. Κεντρικό χαρακτηριστικό του NoDA είναι η επεκτασιμότητα. Οι τελεστές δεδομένων ορίζονται σε αφηρημένες κλάσεις και διεπαφές, οι οποίες λειτουργούν ως πρότυπο (blueprint). Το πρότυπο αυτό εξειδικεύεται για κάθε βάση με την ανάπτυξη ενός νέου υπομονάδας (module), το οποίο ενσωματώνεται στο πλαίσιο του NoDA και κληρονομεί τις βασικές δομές από τον πυρήνα του συστήματος. Η υλοποίηση κάθε τελεστή (π.χ. φίλτρο, ταξινόμηση, συγκέντρωση) αντιστοιχίζεται σε εντολές της αντίστοιχης βάσης δεδομένων.

Στόχος είναι να προσφέρει έναν ενιαίο και αφαιρετικό τρόπο αλληλεπίδρασης με διαφορετικά NoSQL συστήματα, διατηρώντας κοινό API προς τον τελικό χρήστη.
Στο παρόν κεφάλαιο περιγράφεται αναλυτικά η αρχιτεκτονική και η διαδικασία επέκτασης του NoDA framework με σκοπό την υποστήριξη μια καινουργιάς βάσης, στην προκείμενη εργασία την βάση MongoDB, οδηγώντας στη δημιουργία της νέας μονάδας noda-mongodb. Η διαδικασία αυτή μπορεί να εφαρμοστεί και για άλλες βάσεις, ακολουθώντας την ίδια φιλοσοφία που περιγράφεται στο υποκεφάλαιο «Γενικός οδηγός επέκτασης».

3.2.2 Γενικός οδηγός επέκτασης

Δημιουργία νέας μονάδας (module)

Η βασική αρχή είναι η αντιγραφή του προϋπάρχοντος module-προτύπου noda-YYYDataBase, το οποίο λειτουργεί ως "σκελετός". Το module αυτό αντιγράφεται και μετονομάζεται σε noda-mongodb. Στη συνέχεια, το νέο module δεν αντικαθιστά ούτε διαγράφει το noda-YYYDataBase. Στο αρχείο pom.xml του parent project, γίνεται προσθήκη του νέου module πριν από το noda-YYYDataBase. Συγκεκριμένα, οι εγγραφές στα <modules> πρέπει να έχουν τη μορφή:

<module>noda-mongodb</module>
<module>noda-YYYDataBase</module>

Προσαρμογή αρχείου pom.xml νέου module.

Στο pom.xml του noda-mongodb πραγματοποιούνται οι εξής αλλαγές:

Το <artifactId> μετονομάζεται σε:

<artifactId>noda-operators-mongodb</artifactId>

Προστίθενται οι κατάλληλες εξαρτήσεις (dependencies) για τη χρήση του MongoDB driver (π.χ. org.mongodb:mongodb-driver-sync).

Αναπροσαρμογή κώδικα

Με χρήση της λειτουργίας αναζήτηση και αντικατάσταση στο περιβάλλον/εργαλείο προγραμματισμού: Όλες οι εμφανίσεις του όρου YYYDataBase αντικαθίστανται με MongoDB. Όλες οι εμφανίσεις των όρων yYYDataBase, yyyDataBase, yyydatabase αντικαθίστανται με mongodb (πεζά γράμματα).

Η αναζήτηση πρέπει να γίνεται με χρήση wildcards ώστε να αντικατασταθούν σωστά και σύνθετες περιπτώσεις, όπως YYYDataBaseConnector.

Ενσωμάτωση στη διεπαφή του Client

Στο αρχείο pom.xml του noda-client module, προστίθεται ως εξάρτηση το νέο module:

```
<dependency>
  <groupId>gr.unipi.noda</groupId>
  <artifactId>noda-operators-mongodb</artifactId>
  <version>${project.version}</version>
  <optional>true</optional>
</dependency>
```

Δημιουργία client package, στο path gr.ds.unipi.noda.api.client:
Αντιγράφεται το υποπακέτο yyydatabase και μετονομάζεται σε mongodb. Μετά απαιτείται αναζήτηση και αντικατάσταση με την ίδια λογική όπως παραπάνω, ώστε όλες οι κλάσεις να προσαρμοστούν για τη MongoDB.

Προσθήκη στον NoSqlDbSystem

Στην κλάση NoSqlDbSystem, προστίθεται νέα στατική μέθοδος:

```
public static MongoDBBuilderFactory MongoDB() {
   return new MongoDBBuilderFactory();
}
```

Η μέθοδος αυτή επιστρέφει αντικείμενο τύπου MongoDBBuilderFactory, το οποίο καθορίζει την έναρξη της διαδικασίας σύνδεσης.

Κατασκευή του Interface Σύνδεσης

Για τη σύνδεση στη MongoDB θεωρούμε ότι είναι απαραίτητα: (Username, password, database name).
Αυτά προστίθενται στον κατασκευαστή (Builder) της κλάσης MongoDBSystem:

```
public MongoDBSystem.Builder Builder(String username, String password, String database) {
   return new MongoDBSystem.Builder(username, password, database);
}
```

Τα προαιρετικά ορίσματα, όπως π.χ. SSLCertificate, προστίθενται μέσω μεθόδων builder που επιστρέφουν this.

Σύνδεση με το Connector

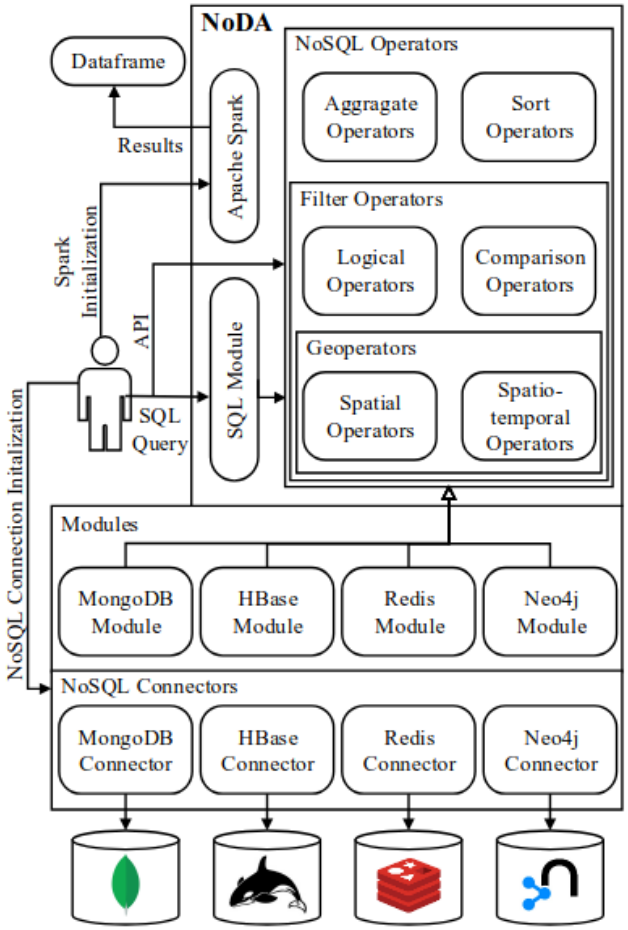
Τα ορίσματα του Builder περνούν στον MongoDBConnector μέσω της μεθόδου newMongoDBConnector(). Εκεί: Ορίζονται οι κατάλληλες μεταβλητές στιγμιοτύπου (instance variables). Υλοποιείται η μέθοδος createConnection() χρησιμοποιώντας τον MongoDB driver. Η μέθοδος επιστρέφει αντικείμενο τύπου MongoClient.

Έπειτα, η κλάση MongoDBConnectionManager αναλαμβάνει: α) την αποθήκευση και επαναχρησιμοποίηση συνδέσεων και β)την υλοποίηση των μεθόδων closeConnection() και closeConnections(), με χρήση εντολών του MongoDB driver.

Κληρονόμηση υπάρχοτων κλάσεων

Οι κλάσεις FieldValue, NoSqlDbModifications και NoSqlDbInserts συγκροτούν το βασικό τμήμα του επιπέδου τροποποιήσεων δεδομένων (modifications) στο πλαίσιο του NoDA. Η FieldValue<T> αποτελεί έναν γενικό τύπο που αποθηκεύει ζεύγη πεδίου–τιμής, καλύπτοντας ένα ευρύ φάσμα δεδομένων (αριθμητικοί τύποι, ημερομηνίες, λογικές τιμές, πίνακες). Παρέχει στατικές μεθόδους δημιουργίας για κάθε επιτρεπόμενο τύπο, ώστε η κατασκευή αντικειμένων να είναι τυποποιημένη και σαφής. Με τον τρόπο αυτό, οι εγγραφές που ορίζει ο χρήστης μετατρέπονται εύκολα σε δομές της εκάστοτε βάσης, όπως για παράδειγμα σε έγγραφα BSON στην περίπτωση της MongoDB. Η ύπαρξη τυποποίησης σε αυτό το επίπεδο μειώνει την πιθανότητα σφαλμάτων και διευκολύνει την επέκταση του πλαισίου σε νέες βάσεις δεδομένων.

Η αφηρημένη κλάση NoSqlDbModifications συγκεντρώνει τα κοινά στοιχεία όλων των τροποποιήσεων, δηλαδή τον σύνδεσμο με τη βάση και το όνομα της συλλογής. Επιβάλλει επίσης την ύπαρξη της μεθόδου flush(), μέσω της οποίας ολοκληρώνεται η αποστολή των αλλαγών προς τη βάση. Η NoSqlDbInserts εξειδικεύει αυτή τη λειτουργικότητα για την πράξη της εισαγωγής, ορίζοντας την μέθοδο insert(...). Στην πράξη, οι κλάσεις εισαγωγών (π.χ. MongoDbInserts) διατηρούν έναν ενδιάμεσο χώρο αποθήκευσης (buffer), όπου συγκεντρώνονται προσωρινά οι εγγραφές. Με αυτόν τον τρόπο δίνεται η δυνατότητα για μαζική εισαγωγή δεδομένων, η οποία είναι αποδοτικότερη από τις συνεχείς μεμονωμένες κλήσεις. Η εκτέλεση πραγματοποιείται όταν ο χρήστης καλέσει τη μέθοδο flush(), οπότε τα δεδομένα αποστέλλονται στη βάση και ο χώρος προσωρινής αποθήκευσης εκκαθαρίζεται. Η συγκεκριμένη λογική υλοποίησης εξασφαλίζει αποδοτικότητα, καλύτερη διαχείριση της μνήμης και πιο σταθερή λειτουργία σε σενάρια μεγάλης κλίμακας. Στην παρούσα εργασία, επαναχρησιμοποιούνται/κληρονομούνται οι υπάρχουσες αυτές κλάσεις για την υλοποίηση των μεθόδων εισαγωγής, διαγραφής, τροποποίησης δεδομένων σε βάση MongoDB.



Εικόνα 9 : Αναπαράσταση της αρχιτεκτονικής του συστήματος NoDA. [10]

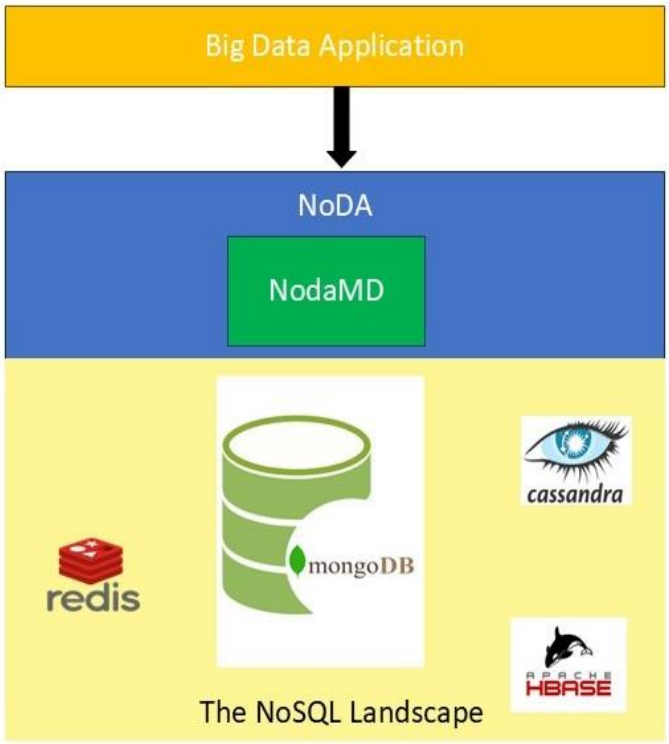
3.3 Επέκταση του Noda (NodaMD)

Στην παρούσα εργασία, αναπτύχθηκε η διαχείριση CRUD (Create, Read, Update, Delete) λειτουργιών για τη βάση δεδομένων MongoDB, ως μέρος της ενσωμάτωσής της στο υπο ανάπτυξη σύστημα με την μέθοδο NoDA (NoSQL Data Access). Το υπό ανάπτυξη σύστημα αποσκοπεί στη δημιουργία μιας ενοποιημένης διεπαφής για τη διαχείριση δεδομένων σε διάφορα NoSQL συστήματα. Μέχρι πρότινος, η λειτουργικότητα του περιοριζόταν στην υποστήριξη της βάσης δεδομένων Cassandra. Η συνεισφορά της παρούσας εργασίας αφορά την ανάπτυξη του απαραίτητου κώδικα ώστε το NoDA να υποστηρίζεται η διασύνδεση με την μη σχεσιακή βάση δεδομένων MongoDB και ονομάζεται NodaMD (βλ. Εικόνα 10).

Για την υλοποίηση, αξιοποιήθηκαν οι δυνατότητες που προσφέρει η γλώσσα προγραμματισμού Java και ο επίσημος MongoDB Java Driver, ο οποίος παρέχει ένα σύνολο εργαλείων για την αλληλεπίδραση με τη βάση δεδομένων. Συγκεκριμένα, αναπτύχθηκαν οι εξής λειτουργίες:

- Εισαγωγή δεδομένων (Insert): Μέθοδοι που επιτρέπουν την εισαγωγή εγγράφων σε μια συλλογή της MongoDB. Αξιοποιήθηκαν οι insertOne() και insertMany(), ώστε να είναι δυνατή τόσο η αποθήκευση ενός μεμονωμένου εγγράφου όσο και η ταυτόχρονη εισαγωγή πολλαπλών εγγράφων.
- Ενημέρωση δεδομένων (Update): Οι Μέθοδοι updateOne() και updateMany() επιτρέπουν την τροποποίηση δεδομένων βάσει συγκεκριμένων κριτηρίων. Με αυτές τις μεθόδους, ένας χρήστης μπορεί να ενημερώσει είτε ένα συγκεκριμένο έγγραφο είτε πολλά, με βάση καθορισμένες συνθήκες.
- Διαγραφή δεδομένων (Delete): Οι Μέθοδοι deleteOne() και deleteMany() επιτρέπουν τη διαγραφή εγγράφων από τη βάση δεδομένων MongoDB. Αυτές οι λειτουργίες δίνουν τη δυνατότητα αφαίρεσης ενός μεμονωμένου εγγράφου ή μιας ομάδας εγγράφων που πληρούν συγκεκριμένα φίλτρα.

Η εργασία αυτή αποτελεί ένα σημαντικό βήμα στην εξέλιξη του NoDA, καθώς πλέον το σύστημα υποστηρίζει ένα ακόμα NoSQL σύστημα, τη MongoDB (https://github.com/helenoninjaki/NoSQL-Operators).



Εικόνα 10 : Αναπαράσταση της αρχιτεκτονικής του συστήματος NoDA. [10]

3.4 Μετάφραση των υπερωτημάτων

Η υλοποίηση της διασύνδεσης με τη βάση δεδομένων MongoDB πραγματοποιήθηκε μέσω της χρήσης της επίσημης βιβλιοθήκης Java MongoDB Driver. Το σύστημα έχει σχεδιαστεί με αντικειμενοστραφή τρόπο, υιοθετώντας αρχές επεκτασιμότητας και επαναχρησιμοποίησης κώδικα. Κύριος στόχος ήταν η παροχή μιας αφαιρετικής διεπαφής (abstraction layer), η οποία να επιτρέπει την ανεξαρτησία της λειτουργικότητας από την εκάστοτε υλοποίηση της βάσης NoSQL. Αξίζει να σημειωθεί πως η MongoDB, ως έγγραφο-κεντρική NoSQL βάση, ακολουθεί διαφορετική φιλοσοφία από τα παραδοσιακά σχεσιακά συστήματα βάσεων δεδομένων (RDBMS). Παρόλα αυτά, υφίσταται μια ευδιάκριτη αντιστοιχία ανάμεσα στις βασικές έννοιες των δύο προσεγγίσεων, οι οποίες καταγράφονται στον Πίνακα 2. Για παράδειγμα, οι SQL πίνακες αντιστοιχούν σε MongoDB collections, οι εγγραφές (rows) σε έγγραφα (documents) και οι στήλες (columns) σε πεδία (fields).

Αυτή η αντιστοίχιση εννοιών χρησίμευσε ως οδηγός για την υλοποίηση των μεθόδων insert, update και delete, που παρουσιάζονται στις επόμενες ενότητες. Η κάθε λειτουργία περιγράφεται τόσο ως προς τον προγραμματιστικό τρόπο υλοποίησης στην Java, όσο και ως προς τη μετάφρασή της σε αντίστοιχες εντολές της MongoDB, διατηρώντας παράλληλα την ορολογία και τη λογική των SQL συναλλαγών.

SQL table	MongoDB collection
SQL row (record)	MongoDB document
SQL column	MongoDB field
SQL primary key (π.χ. id)	MongoDB _id (μοναδικό πεδίο)
SQL DELETE FROM table WHERE <cond>	MongoDB db.collection.deleteOne(filter) ή deleteMany(filter)
SQL INSERT INTO table Values (values)	MongoDB db.collection.insertMany(values)
SQL UPDATE table SET <cond1> WHERE <cond2>	MongoDB db.collection.deleteOne(filter) ή deleteMany(filter)

Πίνακας 2: Πίνακας αντιστοίχισης βασικών εννοιών SQL και MongoDB.

3.4.1 Μετάφραση της εισαγωγής δεδομένων

Η εντολή INSERT είναι θεμελιώδης σε κάθε σύστημα βάσεων δεδομένων, καθώς χρησιμοποιείται για την προσθήκη νέων εγγραφών. Στα σχεσιακά συστήματα (RDBMS), χρησιμοποιείται με τη σύνταξη INSERT INTO ... VALUES(...), ενώ στις NoSQL βάσεις όπως η MongoDB, οι εγγραφές αποθηκεύονται ως έγγραφα BSON.

Στο πλαίσιο αυτής της διπλωματικής, αναπτύχθηκε μια αρχιτεκτονική abstraction layer, η οποία επιτρέπει στον προγραμματιστή να εκτελεί λογικά INSERT επερωτήματα ανεξάρτητα από το είδος της NoSQL βάσης. Η υλοποίηση είναι γραμμένη σε Java και ενσωματώνει συγκεκριμένες κλάσεις για MongoDB.

Ιεραρχία Κλάσεων – Κληρονομικότητα

Η βασική λειτουργία της εισαγωγής υλοποιείται στην κλάση: *MongoDbInserts.java*
Αυτή η κλάση κληρονομεί από την αφηρημένη κλάση: NoSqlDbInserts.java,
η οποία με τη σειρά της κληρονομεί από την κοινή βάση: NoSqlDbModifications.java

Η ιεραρχία είναι η εξής:

NoSqlDbModifications (abstract) →NoSqlDbInserts (abstract) →MongoDbInserts

Η κλάση NoSqlDbModifications.java αποτελεί την κοινή βάση όλων των τροποποιητικών ενεργειών (insert, update, delete).
Περιέχει:
noSqlDbConnector: το αντικείμενο σύνδεσης με τη βάση dataCollection: το όνομα της συλλογής ή πίνακα.

Αφηρημένη μέθοδο:
public abstract <T extends NoSqlDbModifications> T flush();

Η κλάση NoSqlDbInserts.java ορίζει την αφηρημένη μέθοδο εισαγωγής:

public abstract NoSqlDbInserts insert(FieldValue fv, FieldValue... fvs);
public abstract NoSqlDbInserts flush();

Κλάση: MongoDbInserts.java

Υλοποιεί τις παραπάνω μεθόδους για MongoDB, με χρήση του MongoDB Java Driver.
Εσωτερικά, διατηρεί μία λίστα από Document αντικείμενα (stagesList), που αντιπροσωπεύουν τα έγγραφα που πρόκειται να εισαχθούν. Τα έγγραφα προστίθενται ένα-ένα μέσω της insert(...) και στη συνέχεια αποστέλλονται μαζικά με τη flush().
Η λειτουργία insert(...) επιτρέπει την προσθήκη ενός ή περισσότερων πεδίων (field-value pairs) και την κατασκευή ενός εγγράφου MongoDB.

Η σύνταξη είναι λογικά ισοδύναμη με:

INSERT INTO collection_name (field1, field2, ...)
VALUES (value1, value2, ...)

Βήματα Μετάφρασης σε MongoDB

1. Ορισμός Πεδίου-Τιμής (SQL VALUES)

Κάθε FieldValue περιέχει ένα όνομα πεδίου και μία τιμή. Ο συνδυασμός πολλών FieldValue αντικειμένων μετατρέπεται σε ένα map:

```
Map<String, Object> documentMap = Stream.concat(Stream.of(fv), Stream.of(fvs))
                                        .collect(Collectors.toMap(FieldValue::getField, FieldValue::getValue));
```

και στη συνέχεια σε Document (αντιπροσωπευτικό έγγραφο MongoDB):

```
Document document = new Document(documentMap);
```

Αυτό προστίθεται στη λίστα stagesList για μελλοντική εισαγωγή.

2. Συσσώρευση Εγγράφων (Batch Inserts)

Κάθε νέα κλήση της insert() δεν εκτελεί άμεσα την εισαγωγή. Αντίθετα, προσθέτει το έγγραφο στην εσωτερική λίστα stagesList.

Αυτό επιτρέπει τον ορισμό πολλών εγγράφων πριν από την εκτέλεση (batch mode).

3. Εκτέλεση με flush()

Η πραγματική εισαγωγή των εγγράφων γίνεται με την κλήση της:

flush() η οποία εκτελεί: collection.insertMany(stagesList); και εισάγει όλα τα έγγραφα ταυτόχρονα στη MongoDB, ελαχιστοποιώντας το κόστος των queries.

Παράδειγμα Πλήρους Χρήσης

```
MongoDbInserts inserts = MongoDbInserts.newMongoDbInserts(connector, "users");

inserts.insert(FieldValue.create("name", "Maria"),
        FieldValue.create("age", 25),
        FieldValue.create("city", "Patras"))
    .flush();
```

Αυτό ισοδυναμεί με:

```
INSERT INTO users (name, age, city)
VALUES ('Maria', 25, 'Patras');
```

και μεταφράζεται σε MongoDB ως:

```
db.users.insertMany([
 {
  "name": "Maria",
  "age": 25,
  "city": "Patras"
 }]);
```

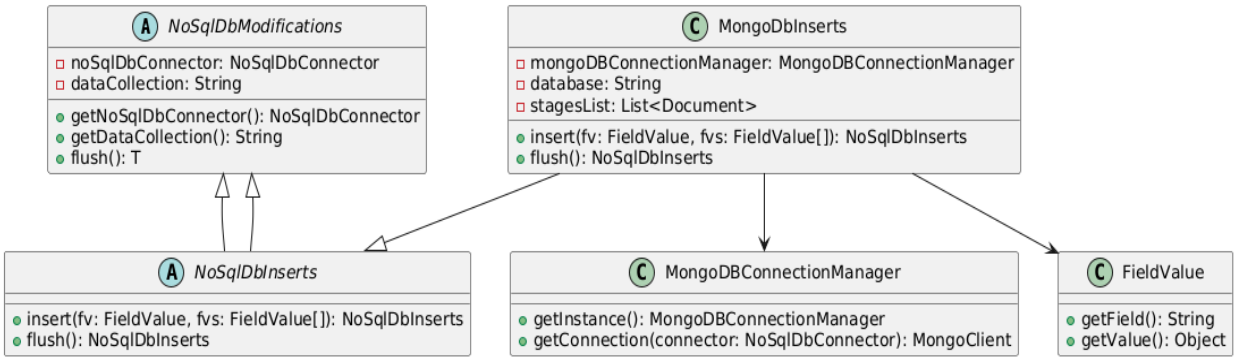
Περιγραφή υλοποιημένων κλάσεων

Στην εικόνα 11 αναπαριστάται το διάγραμμα των κλάσεων που συμμετέχουν στην μετάφραση του επερωτήματος εισαγωγής δεδομένων.

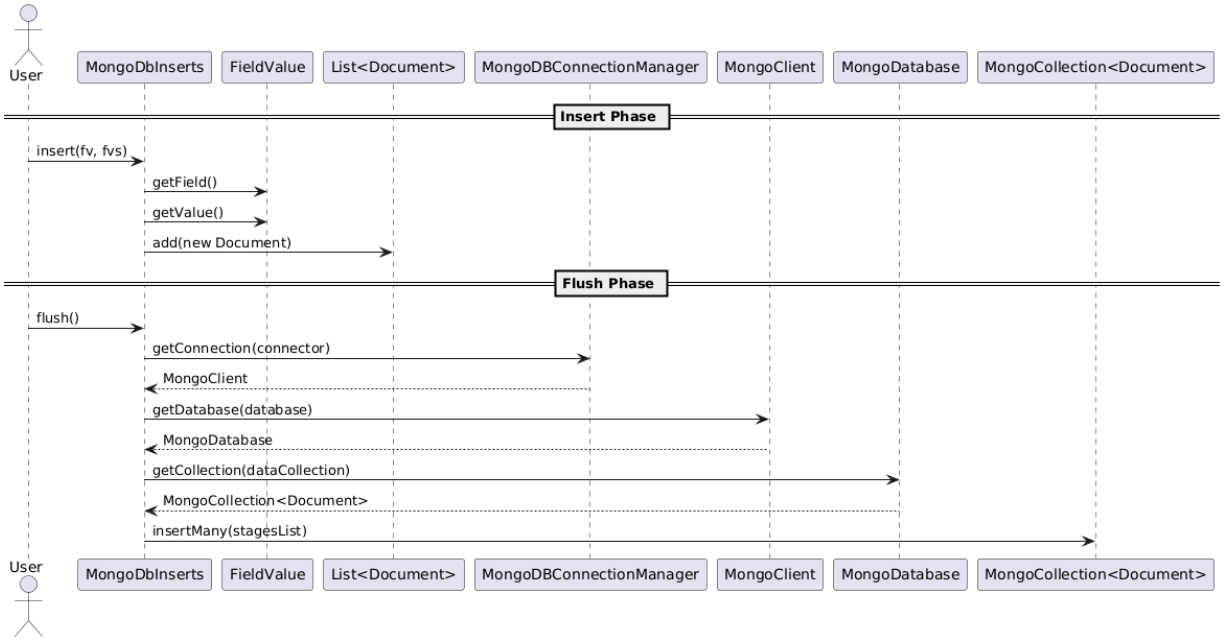
Η κλάση MongoDbInserts υλοποιεί τη διαδικασία εισαγωγής εγγράφων στη MongoDB, κληρονομώντας από την αφηρημένη κλάση NoSqlDbInserts. Κάθε αντικείμενο της κλάσης διατηρεί εσωτερικά μια λίστα από Document αντικείμενα (τύπος της MongoDB) που αναπαριστούν τα προς εισαγωγή δεδομένα. Η εισαγωγή υλοποιείται batch-wise, δηλαδή προστίθενται πολλαπλά έγγραφα στη λίστα και αποστέλλονται μαζικά στη βάση με την κλήση της μεθόδου flush().

Η μέθοδος insert(FieldValue fv, FieldValue... fvs) δέχεται ως είσοδο ένα ή περισσότερα αντικείμενα FieldValue, τα οποία αντιστοιχούν σε πεδία και τιμές που πρόκειται να εισαχθούν. Τα πεδία αυτά μετατρέπονται σε Map<String, Object>, το οποίο με τη σειρά του μετατρέπεται σε Document για εισαγωγή στη MongoDB.

Η μέθοδος flush() αναλαμβάνει την τελική εγγραφή των συγκεντρωμένων εγγράφων στη συλλογή (collection) της βάσης. Μετά την ολοκλήρωση της εγγραφής, δημιουργείται και επιστρέφεται ένα νέο αντικείμενο MongoDbInserts με κενή λίστα εγγράφων, ώστε να είναι έτοιμο για επόμενες εισαγωγές. Στην εικόνα 12, φαίνεται η ακολουθία των γεγονότων που εκτελούνται κατά την εισαγωγή δεδομένων.



Εικόνα 11 : Διάγραμμα κλάσεων για την λειτουργία «Εισαγωγή δεδομένων»



Εικόνα 12 : Διάγραμμα ακολουθίας για την λειτουργία «Εισαγωγή δεδομένων»

3.4.2 Μετάφραση της διαγραφής δεδομένων

Η εντολή DELETE αποτελεί βασική λειτουργία στις σχεσιακές βάσεις δεδομένων, καθώς χρησιμοποιείται για τη διαγραφή εγγραφών που πληρούν συγκεκριμένες συνθήκες (συνήθως μέσω του όρου WHERE). Στο πλαίσιο αυτής της διπλωματικής, υλοποιήθηκε ένα abstraction layer σε Java που επιτρέπει τη χρήση της λογικής DELETE σε βάσεις MongoDB, με στόχο τη διατηρήσιμη και επεκτάσιμη υποστήριξη πολλαπλών NoSQL backends. Στην εικόνα Χ αναπαριστάται το διάγραμμα των κλάσεων και η αντίστοιχη ιεραρχία στην μετάφραση του επερωτήματος διαγραφής δεδομένων.

Ιεραρχία Κλάσεων – Κληρονομικότητα

Η βασική λειτουργία διαγραφής υλοποιείται στην κλάση: MongoDbDeletes.java
η οποία κληρονομεί από: NoSqlDbDeletes.java (αφηρημένη κλάση) που με τη σειρά της κληρονομεί από: NoSqlDbModifications.java.

Η ιεραρχία έχει την εξής μορφή:
NoSqlDbModifications (abstract) →NoSqlDbDeletes (abstract) →MongoDbDeletes

Κλάση: NoSqlDbModifications.java.

Αποτελεί την κοινή βάση όλων των τροποποιητικών ενεργειών (insert, update, delete) και περιέχει:

noSqlDbConnector: σύνδεση με το backend dataCollection: όνομα συλλογής (MongoDB) ή πίνακα (SQL).

Αφηρημένη μέθοδο:

public abstract <T extends NoSqlDbModifications> T flush(); Κλάση: NoSqlDbDeletes.java

Αφηρημένη κλάση για όλες τις λειτουργίες διαγραφής:

public abstract NoSqlDbDeletes delete(String... fields);

public abstract NoSqlDbDeletes delete(FilterOperator fop, String... fields);

public abstract NoSqlDbDeletes flush();

Κλάση: MongoDbDeletes.java

Υλοποιεί τη διαγραφή εγγράφων ειδικά για MongoDB. Κύρια χαρακτηριστικά: Χρησιμοποιεί MongoDBConnectionManager για να αποκτήσει σύνδεση με τη βάση. Διατηρεί εσωτερικά μια λίστα stagesList με αντικείμενα DeleteStage, καθένα από τα οποία περιέχει ένα φίλτρο τύπου Document που ορίζει ποια έγγραφα θα διαγραφούν. Η εκτέλεση όλων των διαγραφών γίνεται μέσω της flush().

Περιγραφή Λειτουργίας DELETE

Η λειτουργία διαγραφής είναι λογικά ισοδύναμη με την SQL εντολή:

DELETE FROM collection_name

WHERE <condition>

και εκτελείται σε δύο φάσεις: ορισμός φίλτρων μέσω delete(...), και εκτέλεση μέσω flush().

Βήματα Μετάφρασης σε MongoDB:

1. Ορισμός Φίλτρου (SQL WHERE)

Ο χρήστης καλεί: MongoDbDeletes.delete(FilterOperator fop, String... fields).
Ο FilterOperator περιέχει τις συνθήκες που πρέπει να πληρούνται για να διαγραφούν τα έγγραφα. Η μετατροπή σε MongoDB filter γίνεται ως εξής:
String compiledFilterOperator = filterOperator.getOperatorExpression().toString();
Document filter = Document.parse(compiledFilterOperator);

Παράδειγμα:
{ "name": "John" }

2. Συσσώρευση Εντολών Διαγραφής

Η κλάση δεν εκτελεί άμεσα τις διαγραφές. Κάθε φίλτρο filter αποθηκεύεται ως νέο DeleteStage στη λίστα: newStagesList.add(new DeleteStage(filter, null));
Το update μέσα στο DeleteStage δεν χρησιμοποιείται στη DELETE λειτουργία (παραμένει null), αλλά η κλάση παραμένει ευέλικτη ώστε να υποστηρίξει και unset στο μέλλον.

3. Κατά την κλήση της μεθόδου flush() εκτελούνται όλες οι συσσωρευμένες διαγραφές με: collection.deleteMany(stage.getFilter()) κάνοντας μαζική διαγραφή εγγράφων με βάση τα προκαθορισμένα φίλτρα, όπως φαίνεται στην Εικόνα 14.

Παράδειγμα Πλήρους Χρήσης:
MongoDbDeletes deletes = MongoDbDeletes.newMongoDeletes(connector, "users");
deletes.delete(Filter.eq("name", "John")).flush();

Το παραπάνω είναι λογικά ισοδύναμο με:
DELETE FROM users
WHERE name = 'John';

και μεταφράζεται σε MongoDB ως:

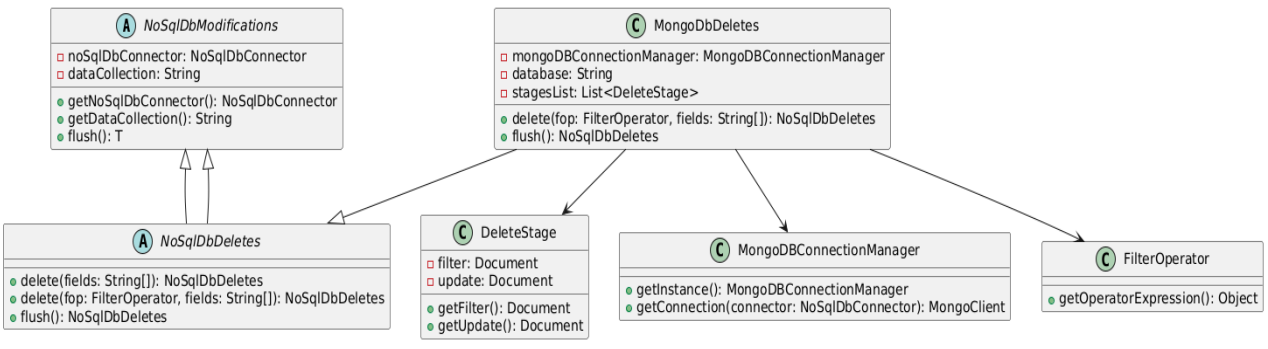
db.users.deleteMany({ "name": "John" });

Περιγραφή υλοποιημένων κλάσεων

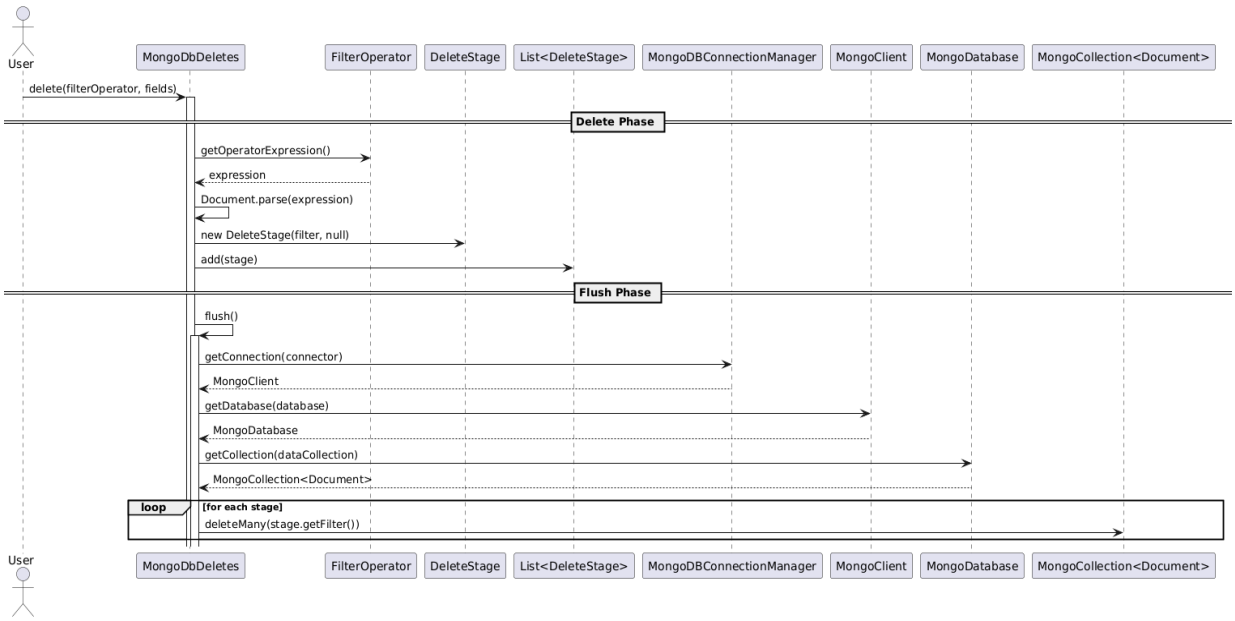
Κλάση MongoDbDeletes:

Η κλάση MongoDbDeletes υλοποιεί τη λογική μαζικών διαγραφών από μια MongoDB συλλογή, βασισμένη σε φίλτρα. Κληρονομεί από την αφηρημένη κλάση NoSqlDbDeletes και εσωτερικά διατηρεί μια λίστα από DeleteStage αντικείμενα, τα οποία αναπαριστούν φίλτρα διαγραφής (βλ. Εικόνα 13).

Η μέθοδος delete(FilterOperator filterOperator, String... fields) χρησιμοποιεί έναν FilterOperator — μια διεπαφή που περιγράφει λογικές ή/και συνθήκες φιλτραρίσματος — και δημιουργεί ένα Document φίλτρου το οποίο προστίθεται στη λίστα stagesList. Η πραγματική διαγραφή εκτελείται με την κλήση της flush(), η οποία διατρέχει όλα τα φίλτρα και καλεί τη μέθοδο deleteMany() της MongoDB για να αφαιρέσει τα αντίστοιχα έγγραφα.



Εικόνα 13 : Διάγραμμα κλάσεων για την λειτουργία «Διαγραφή δεδομένων»



Εικόνα 14 : Διάγραμμα ακολουθίας για την λειτουργία «Διαγραφή δεδομένων»

3.4.3 Μετάφραση της τροποποίησης δεδομένων

Η εντολή UPDATE αποτελεί βασική λειτουργία σε σχεσιακά συστήματα βάσεων δεδομένων (RDBMS), καθώς επιτρέπει την τροποποίηση εγγραφών που ικανοποιούν συγκεκριμένες συνθήκες. Στο πλαίσιο αυτής της διπλωματικής, αναπτύχθηκε ένα abstraction layer που επιτρέπει την εκτέλεση αντίστοιχων λειτουργιών σε μια βάση MongoDB, υλοποιημένη με τη γλώσσα Java, με στόχο την ανεξαρτησία από το εκάστοτε backend (NoSQL database engine). Το διάγραμμα των κλάσεων και η αντίστοιχη ιεραρχία στην μετάφραση του επερωτήματος τροποποίησης δεδομένων αναπαρίσταται στην εικόνα 15.

Ιεραρχία Κλάσεων – Κληρονομικότητα

Η βασική λειτουργία της ενημέρωσης υλοποιείται στην κλάση:*MongoDbUpdates.java*
Αυτή η κλάση κληρονομεί από την αφηρημένη κλάση: NoSqlDbUpdates.java,

η οποία με τη σειρά της κληρονομεί από την κοινή αφηρημένη βάση: NoSqlDbModifications.java

Η ιεραρχία έχει ως εξής:
NoSqlDbModifications (abstract) →NoSqlDbUpdates (abstract) →MongoDbUpdates

Κλάση: NoSqlDbModifications.java:
Περιέχει κοινά στοιχεία για κάθε τροποποιητική πράξη (insert, update, delete), όπως: noSqlDbConnector (αντικείμενο σύνδεσης με τη βάση), dataCollection (το όνομα της συλλογής ή πίνακα), την αφηρημένη μέθοδο:

public abstract <T extends NoSqlDbModifications> T flush();

Κλάση:NoSqlDbUpdates.java:
Ορίζει την αφηρημένη διεπαφή των updates για όλες τις βάσεις:
public abstract NoSqlDbUpdates update(FilterOperator fop, FieldValue fv, FieldValue... fvs);
public abstract NoSqlDbUpdates flush();

Κλάση: MongoDbUpdates.java
Υλοποιεί τις παραπάνω μεθόδους συγκεκριμένα για MongoDB, με χρήση του MongoDB Java Driver.
Περιγραφή Λειτουργίας UPDATE
Η υλοποίηση επιτρέπει στον χρήστη να εκτελέσει μία εντολή που αντιστοιχεί λογικά στην SQL σύνταξη:

UPDATE collection_name
SET field1 = value1, field2 = value2, ...
WHERE <condition>. Η Java κλήση αυτής της λειτουργίας γίνεται μέσω της μεθόδου:

MongoDbUpdates.update(FilterOperator fop, FieldValue fv, FieldValue... fvs) και ακολουθείται από:.flush() που εκτελεί τις σωρευμένες αλλαγές.

Βήματα Μετάφρασης σε MongoDB

1. Ορισμός Φίλτρου (SQL WHERE)

Το φίλτρο ορίζεται από το αντικείμενο FilterOperator, το οποίο περιέχει την συνθήκη σε JSON/DSL μορφή. Στο αρχείο MongoDbUpdates.java γίνεται μετατροπή ως εξής:

```
String compiledFilterOperator = filterOperator.getOperatorExpression().toString();
Document filter = Document.parse(compiledFilterOperator);
```

Έτσι παράγεται ένα MongoDB φίλτρο τύπου: { "name": "John" }

2. Ορισμός Πεδίων προς Ενημέρωση (SQL SET)

Τα πεδία που πρόκειται να ενημερωθούν δίνονται με χρήση αντικειμένων FieldValue. Αυτά μετατρέπονται σε ένα map:
Map<String, Object> updateMap = ... και κατόπιν σε update document με χρήση του MongoDB operator $set:

```
Document update = new Document("$set", new Document(updateMap));
```

Το τελικό update query είναι της μορφής:

```
{
  "$set": {
    "age": 30,
    "city": "Athens"
  }
}
```

3. Συσσώρευση και Εκτέλεση (Batch Flush)

Η κάθε εντολή (filter, update) συσκευάζεται σε αντικείμενο UpdateStage (εσωτερική κλάση μέσα στο MongoDbUpdates.java) και προστίθεται σε μια λίστα.

```
newStagesList.add(new UpdateStage(filter, update));
```

Κατά την κλήση της flush(), όλα τα UpdateStage εκτελούνται με:

```
collection.updateMany(stage.getFilter(), stage.getUpdate());
```

Έτσι επιτυγχάνεται μαζική εκτέλεση ενημερώσεων, ελαχιστοποιώντας τον αριθμό των queries.). Το διάγραμμα ακολουθίας στην μετάφραση του επερωτήματος τροποποίησης δεδομένων αναπαρίσταται στην εικόνα 16.

Παράδειγμα Πλήρους Χρήσης:

```
MongoDbUpdates updates = MongoDbUpdates.newMongoUpdates(connector, "users");

updates.update(Filter.eq("name", "John"),
        FieldValue.create("age", 30),
        FieldValue.create("city", "Athens"))
    .flush();
```

Αυτό ισοδυναμεί με:

```
UPDATE users
SET age = 30, city = 'Athens'
WHERE name = 'John';
```

και μεταφράζεται σε MongoDB ως:

```
db.users.updateMany(
 { "name": "John" },
 { "$set": { "age": 30, "city": "Athens" } });
```

Περιγραφή υλοποιημένων κλάσεων

Η κλάση MongoDbUpdates υλοποιεί τη λογική μαζικών ενημερώσεων (updates) εγγράφων σε μία συλλογή MongoDB, βασισμένη σε συνθήκες φιλτραρίσματος. Κληρονομεί από την αφηρημένη κλάση NoSqlDbUpdates, η οποία με τη σειρά της κληρονομεί από την κοινή βάση NoSqlDbModifications. Το μοτίβο αυτό επιτρέπει την επαναχρησιμοποίηση και επέκταση κοινών λειτουργιών τροποποίησης για διάφορους τύπους βάσεων NoSQL.

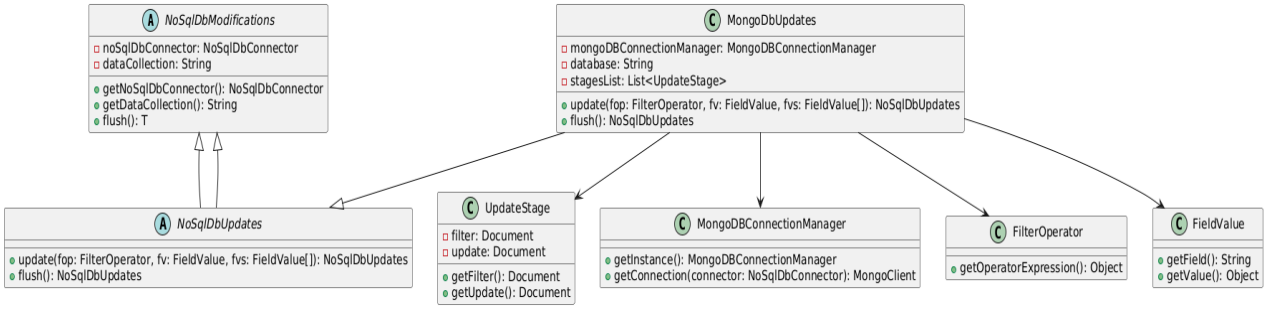
Η MongoDbUpdates διατηρεί εσωτερικά μια λίστα από UpdateStage αντικείμενα. Κάθε UpdateStage αναπαριστά ένα φίλτρο ενημέρωσης (filter) και το αντίστοιχο έγγραφο ενημέρωσης (update) που θα εφαρμοστεί στα έγγραφα που πληρούν τη συνθήκη.

Η βασική μέθοδος update(FilterOperator filterOperator, FieldValue fv, FieldValue... fvs) δέχεται: έναν FilterOperator, που περιγράφει λογικές συνθήκες τύπου WHERE, και ένα ή περισσότερα FieldValue αντικείμενα, τα οποία ορίζουν τα πεδία και τις νέες τιμές τους.

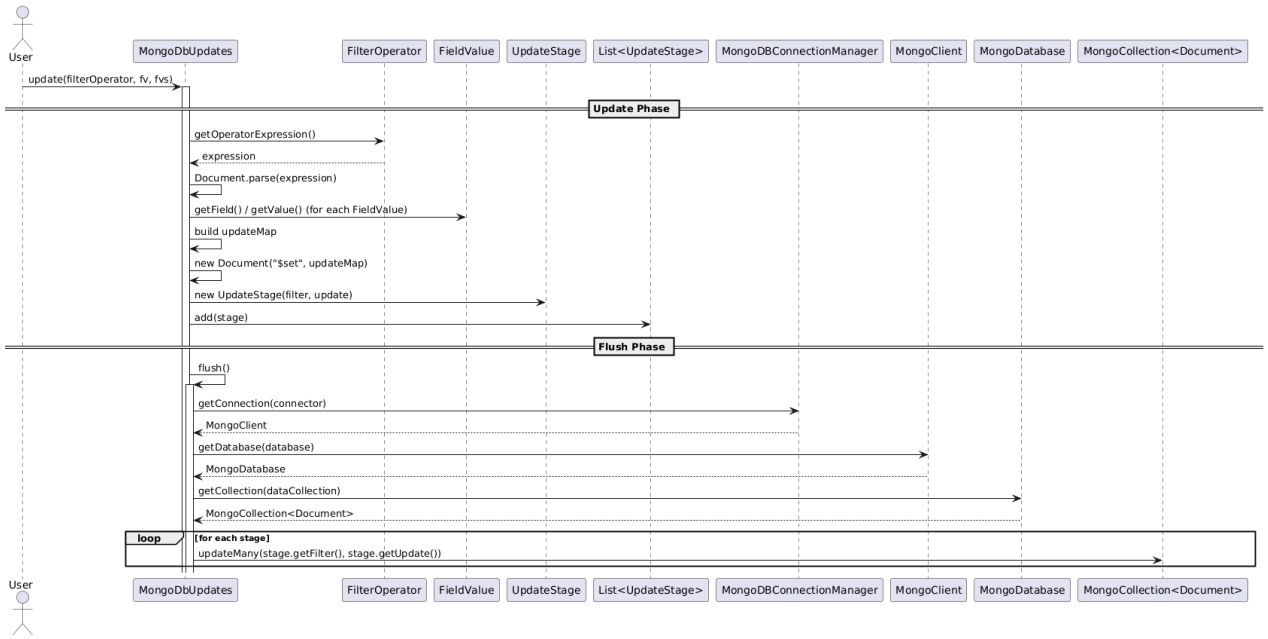
Ο FilterOperator μετατρέπεται σε αντικείμενο τύπου Document (BSON), το οποίο αντιστοιχεί στο φίλτρο που θα χρησιμοποιηθεί στη MongoDB. Αντίστοιχα, τα FieldValue μετατρέπονται σε BSON update document χρησιμοποιώντας τον operator $set.

Τα φίλτρα και οι ενημερώσεις δεν εκτελούνται άμεσα. Αντί γι’ αυτό, προστίθενται στη λίστα stagesList, επιτρέποντας τη συσσώρευση πολλαπλών update εντολών. Η πραγματική εκτέλεση των ενημερώσεων γίνεται με την κλήση της μεθόδου flush(), η οποία διατρέχει όλα τα UpdateStage αντικείμενα και καλεί τη μέθοδο updateMany() της MongoDB, εφαρμόζοντας μαζικά τις αλλαγές στα αντίστοιχα έγγραφα.

Η προσέγγιση αυτή παρέχει σημαντικά πλεονεκτήματα όσον αφορά την απόδοση, καθώς μειώνει τον αριθμό των κλήσεων προς τη βάση και επιτρέπει ενημερώσεις σε κομμάτια δεδομένων (batches).



Εικόνα 15 : Διάγραμμα κλάσεων για την λειτουργία «Τροποποίηση δεδομένων»



Εικόνα 16 : Διάγραμμα ακολουθίας για την λειτουργία «Τροποποίηση δεδομένων»

4 Χρήση και αξιολόγηση του NodaMD

4.1 Σενάρια χρήσης του NodaMD

Σενάριο 1: Σύστημα Διαχείρισης Χρηστών

Οι περισσότερες εφαρμογές που αλληλεπιδρούν με χρήστες, όπως κοινωνικά δίκτυα, πλατφόρμες εγγραφής και ηλεκτρονικές υπηρεσίες απαιτούν ένα σύστημα διαχείρισης χρηστών.

Πώς λειτουργεί:

Όταν ένας χρήστης δημιουργεί έναν νέο λογαριασμό, το σύστημα εισάγει τα στοιχεία του στη βάση δεδομένων, όπως όνομα, email, κρυπτογραφημένο κωδικό πρόσβασης και ημερομηνία εγγραφής. Αν ο χρήστης θελήσει να αλλάξει το email ή τον κωδικό του, το σύστημα ενημερώνει το σχετικό πεδίο στη βάση. Σε περίπτωση που ο χρήστης επιλέξει να διαγράψει τον λογαριασμό του, η αντίστοιχη εγγραφή αφαιρείται από τη βάση εξασφαλίζοντας ότι τα προσωπικά δεδομένα του δεν παραμένουν αποθηκευμένα.

Παράδειγμα χρήσης:

Ένας χρήστης εγγράφεται σε μια πλατφόρμα online μαθημάτων. Μετά από λίγο καιρό, αλλάζει το email του. Κάποια στιγμή αποφασίζει να διαγράψει τον λογαριασμό του, οπότε όλα τα δεδομένα του αφαιρούνται από τη βάση.

Σενάριο 2: Ηλεκτρονικό Κατάστημα (e-Shop)

Οι πλατφόρμες ηλεκτρονικού εμπορίου αποθηκεύουν χιλιάδες προϊόντα με διαφορετικά χαρακτηριστικά, όπως όνομα, τιμή, περιγραφή, φωτογραφίες και ποσότητα αποθέματος.

Πώς λειτουργεί:

Όταν ένας διαχειριστής προσθέτει ένα νέο προϊόν, το σύστημα το εισάγει στη βάση δεδομένων με όλες τις σχετικές πληροφορίες. Αν η τιμή του προϊόντος αλλάξει ή αν το απόθεμα ανανεωθεί, η βάση ενημερώνεται ώστε οι πελάτες να βλέπουν τις σωστές

πληροφορίες. Όταν ένα προϊόν σταματήσει να πωλείται, η εγγραφή του διαγράφεται από τη βάση για να διατηρείται η λίστα προϊόντων ενημερωμένη.

Παράδειγμα χρήσης:

Ένα e-shop προσθέτει ένα νέο μοντέλο κινητού τηλεφώνου. Μετά από μερικούς μήνες, η τιμή του μειώνεται λόγω προσφοράς και η εγγραφή ενημερώνεται. Όταν το προϊόν εξαντληθεί οριστικά, διαγράφεται από τη βάση.

Σενάριο 3: Σύστημα Διαχείρισης Κρατήσεων

Οι εφαρμογές κρατήσεων, όπως πλατφόρμες για ξενοδοχεία, εισιτήρια και εστιατόρια, χρησιμοποιούν MongoDB για τη διαχείριση των διαθέσιμων θέσεων και των πελατών.

Πώς λειτουργεί:

Όταν ένας πελάτης κάνει κράτηση, η πληροφορία εισάγεται στη βάση, καταγράφοντας το όνομα του πελάτη, την ημερομηνία κράτησης και άλλες λεπτομέρειες. Αν ο πελάτης αλλάξει την ημερομηνία ή τον αριθμό επισκεπτών, η κράτηση ενημερώνεται. Αν η κράτηση ακυρωθεί διαγράφεται από τη βάση ώστε να ελευθερωθεί η θέση για άλλους πελάτες.

Παράδειγμα χρήσης:

Ένας πελάτης κάνει κράτηση σε ένα ξενοδοχείο. Αποφασίζει να αλλάξει την ημερομηνία διαμονής του, οπότε η κράτηση ενημερώνεται.Λίγες μέρες πριν το ταξίδι, ακυρώνει την κράτηση και η εγγραφή αφαιρείται από τη βάση.

Σενάριο 4: Σύστημα Διαχείρισης Αποθήκης

Μια επιχείρηση χρησιμοποιεί MongoDB για την καταγραφή του αποθέματός της, διασφαλίζοντας ότι γνωρίζει πάντα πόσα προϊόντα διαθέτει.

Πώς λειτουργεί:

Όταν μια νέα παρτίδα προϊόντων φτάνει στην αποθήκη, η βάση ενημερώνεται με τα νέα αποθέματα. Αν αλλάξει η ποσότητα λόγω πωλήσεων ή εσωτερικής χρήσης, η εγγραφή ενημερώνεται ώστε να αντανακλά την πραγματική διαθεσιμότητα. Όταν ένα προϊόν

καταργείται από τη γκάμα της εταιρείας, αφαιρείται από τη βάση για να μην εμφανίζεται ως διαθέσιμο.

Παράδειγμα χρήσης:

Μια επιχείρηση παραλαμβάνει 100 νέα κομμάτια ενός προϊόντος και τα προσθέτει στη βάση. Μετά από μερικές εβδομάδες, έχουν πουληθεί τα 80, οπότε η ποσότητα ενημερώνεται. Όταν το προϊόν σταματάει να παράγεται, η εγγραφή διαγράφεται.

Σενάριο 5: Σύστημα Διαχείρισης Υπαλλήλων (Human Resources Management System- HRMS)

Οι μεγάλες εταιρείες, σε αριθμό υπαλλήλων, χρησιμοποιούν HRMS συστήματα για να διαχειρίζονται τις πληροφορίες των υπαλλήλων τους.

Πώς λειτουργεί:

Κάθε φορά που ένας υπάλληλος αποχωρεί ή προσλαμβάνεται στην εταιρεία, η αντίστοιχη εγγραφή στην ΒΔ, είτε αφαιρείται είτε εισάγεται. Επίσης, ένας υπάλληλος μπορεί να μεταβάλει τις προσωπικές τους πληροφορίες (π.χ. δεξιότητες).

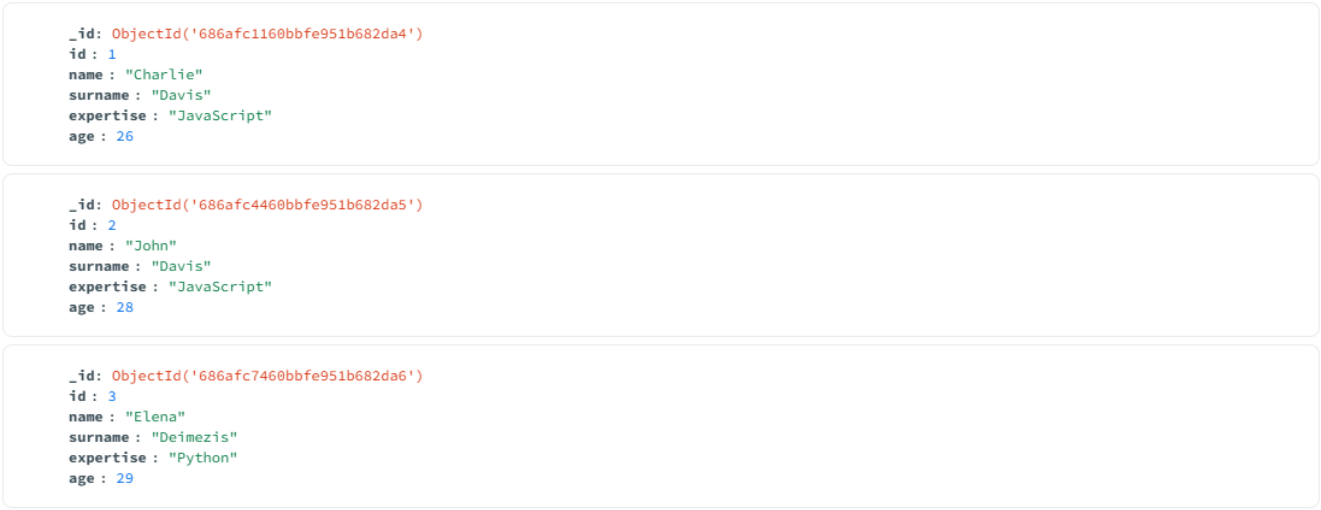
Παράδειγμα χρήσης:

Ένας διαχειριστής ανθρωπίνων πόρων (human resources manager) ενημερώνει την ΒΔ μέσω του συστήματος HRMS, με σκοπό την παροχή ενημερωμένης εικόνα της εταιρείας στον ιδιοκτήτη της εταιρείας και στα λοιπά μέλη του διοικητικού συμβουλίου.

4.2 Αξιολόγηση και χρήση του NodaMD

Ο σκοπός της τρέχουσας ενότητας είναι να αναδείξει την χρησιμότητα και αποτελεσματικότητα του συστήματος NodaMD. Η αποτελεσματικότητα του NodaMD αξιολογείται στο σενάριο χρήσης 5, όπως ορίστηκε στην ενότητα 4.1. Στο συγκεκριμένο σενάριο χρήσης χρησιμοποιούμε μια απλουστευμένη μορφή του συστήματος, που αποτελείται από έναν πίνακα ‘Υπάλληλοι’ με στήλες : “ID”, “Όνομα”, “Επίθετο”, “Δεξιότητα”, “Ηλικία”. Παρακάτω, περιγράφονται οι βασικές εργασίες που εκτελεί ένας διαχειριστής

ανθρώπινων πόρων σε μια εταιρεία και εμπίπτουν στο εν λόγω σενάριο χρήσης. Στην εικόνα 17 αναπαριστάται η αρχική κατάσταση του πίνακα. Επίσης σε κάθε εργασία που περιγράφεται παρακάτω, αναφέρουμε την εργασία σε φυσική γλώσσα, πως αυτή η εργασία μεταφράζεται σε SQL query και πως το NodaMD, το σύστημα μας, μεταφράζει το SQL query σε MongoDB query.



Εικόνα 17 : Αρχική κατάσταση του πίνακα "Υπάλληλοι".

4.2.1 Εισαγωγή υπαλλήλου

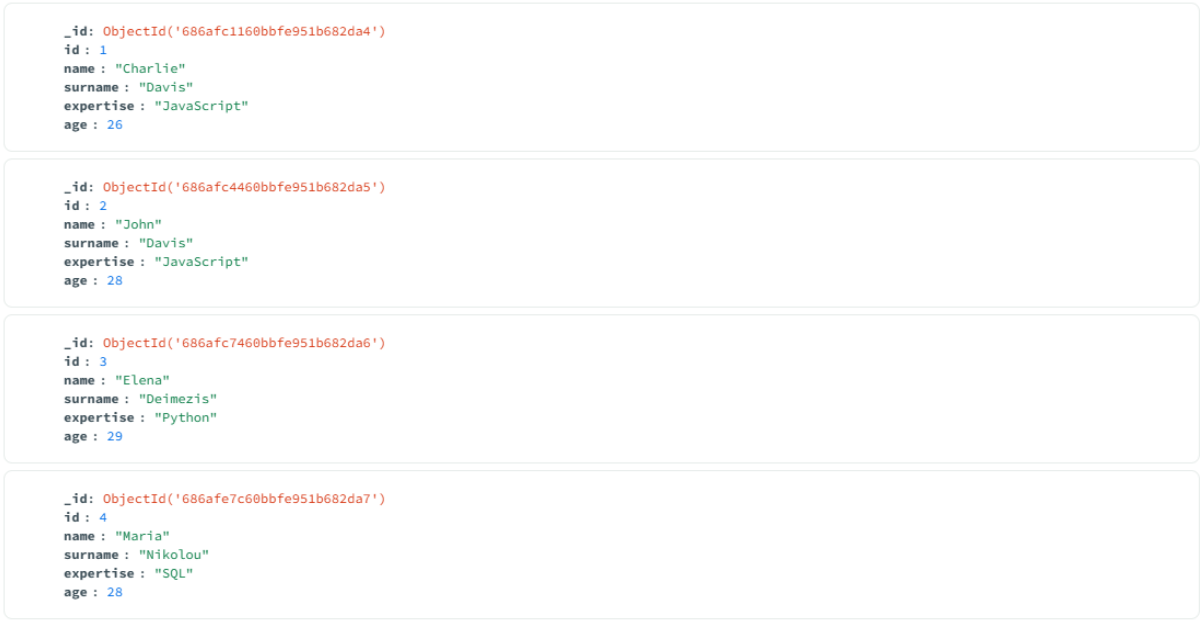
Μια από τις βασικές εργασίες του διαχειριστή είναι η εισαγωγή νέου υπαλλήλου στην ΒΔ που προσλήφθηκε στην εταιρεία. Στο παρακάτω ερώτημα προς την ΒΔ, αποτυπώνεται η εργασία "Εισήγαγε τον υπάλληλο 'Charlie 'Davis' που γνωρίζει JavaScript και είναι 26 ετών".

SQL query:

INSERT INTO employees (id, name, surname, expertise, age) VALUES (3, 'Charlie', 'Davis', 'JavaScript', 26);

MongoDB query:

db.employees.insertOne({id: 3, name: "Charlie",surname: "Davis", expertise: "JavaScript" age: 26 });



Εικόνα 18 : Κατάσταση του πίνακα "Υπάλληλοι" κατόπιν της εισαγωγής υπαλλήλου.

4.2.2 Διαγραφή υπαλλήλου

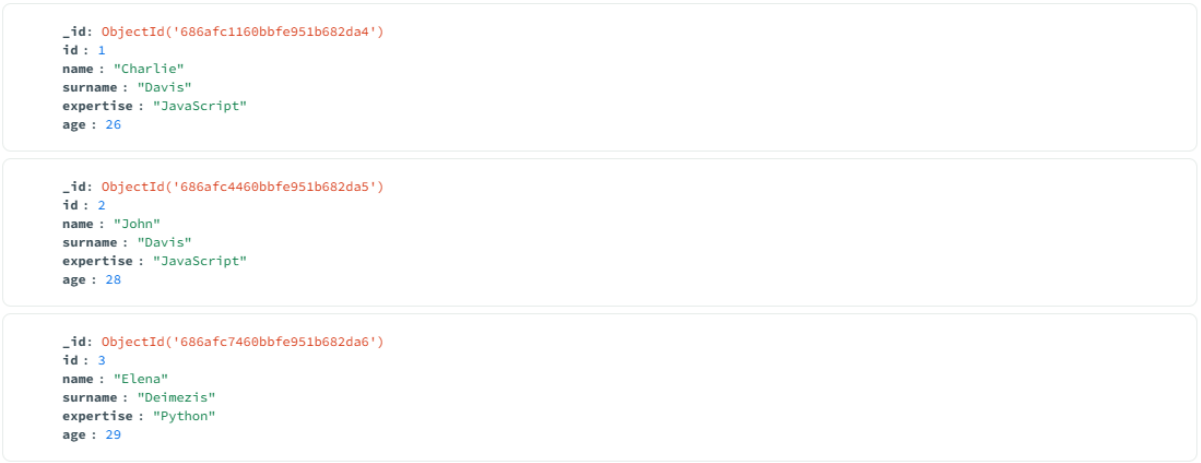
Μια ακόμη από τις βασικές εργασίες του διαχειριστή είναι η διαγραφή ενός υπαλλήλου στην ΒΔ που αποχώρησε από την εταιρεία. Στο παρακάτω ερώτημα προς την ΒΔ, αποτυπώνεται η εργασία "Διάγραψε τον υπάλληλο με ID 4".

SQL query:

DELETE FROM employees WHERE id = 4;

MongoDB query:

db.employees.deleteOne({ id: 4 });



Εικόνα 19 : Κατάσταση του πίνακα "Υπάλληλοι" κατόπιν της διαγραφής υπαλλήλου.

4.2.3 Τροποποίηση υπαλλήλου

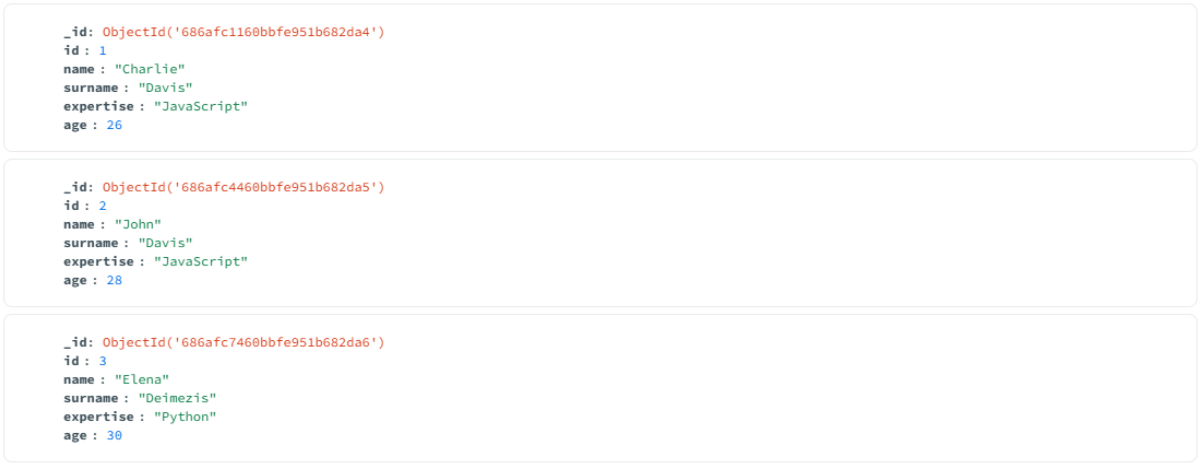
Η τροποποίηση των πληροφοριών ενός ή πολλών υπαλλήλων της εταιρείας αποτελεί καίρια εργασία του διαχειριστή. Στο παρακάτω ερώτημα προς την ΒΔ, αποτυπώνεται η εργασία "Ανανέωσε την ηλικία του υπαλλήλου με ID 3 σε 30" (βλ Εικόνα 19).

SQL query:

```
UPDATE employees SET age = 27 WHERE id = 3;
```

MongoDB query:

```
db.employees.updateOne( { id: 3 }, { $set: { age: 30 } });
```



Εικόνα 20 : Κατάσταση του πίνακα "Υπάλληλοι" κατόπιν της τροποποίσης υπαλλήλου.

4.3 Συζήτηση

Η αποτελεσματικότητα του NodaMD εξετάστηκε και σε άλλα σενάρια χρήσης, όπως αναφέρθηκαν στην ενότητα 4.1. Σε όλα τα σενάρια χρήσης παρατηρήσαμε πως το σύστημα μας αποδίδει 100%, δηλαδή μεταφράζει και εκτελεί όλα τα ερωτήματα με επιτυχία. Τα σενάρια χρήσης περιλαμβάνουν ωστόσο έναν πίνακα, ώς ένα proof of concept. Μια άλλη προσέγγιση είναι η αξιοποίηση του ChatGPT[18] και λοιπών μεγάλων γλωσσικών μοντέλων (LLMs). Στο εν λόγω proof of concept, δεν αναμένεται μεγάλη διαφορά στην απόδοση του NodaMD και των LLMs. Ωστόσο, σε πιο σύνθετα σενάρια χρήσης που εμπλέκουν πολλούς πίνακας, αναμένεται το NodaMD (κατόπιν επέκτασης του) να αποδίδει καλύτερα, καθώς στα LLMs παρατηρούνται λάθη (hallucinations) σε σύνθετα ερωτήματα πολλών πινάκων.

5 Σύνοψη και Μελλοντική έρευνα

Στην παρούσα διπλωματική εργασία υλοποιήθηκε η υποστήριξη CRUD λειτουργιών για τη βάση δεδομένων MongoDB, στο πλαίσιο του ευρύτερου συστήματος NoDA (NoSQL Data Access). Μέσω της ενσωμάτωσης του νέου υποσυστήματος NodaMD, το NoDA επεκτάθηκε ώστε να υποστηρίζει επιπλέον τύπους NoSQL βάσεων, πέραν της Cassandra. Η υλοποίηση βασίστηκε στη γλώσσα προγραμματισμού Java και τον επίσημο MongoDB Java Driver, αξιοποιώντας τις βασικές μεθόδους εισαγωγής, ενημέρωσης και διαγραφής εγγράφων.

Η αποτελεσματικότητα του συστήματος αξιολογήθηκε μέσω ποικίλων σεναρίων χρήσης που αντανακλούν ρεαλιστικές ανάγκες σύγχρονων εφαρμογών, όπως διαχείριση χρηστών, αποθήκης, κρατήσεων και HR συστημάτων. Το NodaMD απέδωσε με ακρίβεια 100% σε όλα τα δοκιμαστικά σενάρια, αποδεικνύοντας ότι μπορεί να αποτελέσει μια αξιόπιστη διεπαφή μεταξύ SQL-like ερωτημάτων και MongoDB queries. Η αντιστοίχιση μεταξύ SQL και MongoDB επιτεύχθηκε με σαφήνεια και συνέπεια, επιτρέποντας σε χρήστες με γνώσεις SQL να αλληλεπιδρούν με NoSQL συστήματα χωρίς να απαιτείται βαθιά τεχνική εξοικείωση με τη MongoDB.

Η παρούσα εργασία αποτελεί proof of concept, εστιάζοντας σε βασικές λειτουργίες και μονό-πινακικά σχήματα. Στο μέλλον, το σύστημα NodaMD μπορεί να επεκταθεί ώστε να υποστηρίζει πιο σύνθετα ερωτήματα, όπως πολυπίνακα ερωτήματα που περιλαμβάνουν συσχετίσεις μεταξύ δεδομένων ακόμη και σε NoSQL περιβάλλοντα όπου δεν υφίστανται αυστηρές σχέσεις μεταξύ των οντοτήτων.

Ένα επιπλέον βήμα είναι η ενσωμάτωση μηχανισμών αυτόματης βελτιστοποίησης ερωτημάτων, ώστε να βελτιώνεται η απόδοση σε περιβάλλοντα παραγωγής με μεγάλους όγκους δεδομένων. Αξίζει να σημειωθεί πως μεγάλα γλωσσικά μοντέλα τείνουν να απορροφούν μεγάλο ογκο δεδομένων από τεράστιες βάσεις δεδομένων στο διαδίκτυο για να αυξήσουν την αποδοτικότητα τους [19]-[21]. Παράλληλα, αξίζει να διερευνηθεί η ενοποίηση με (LLMs) , όπως το ChatGPT, με στόχο τη χρήση φυσικής γλώσσας ως διεπαφή προς NoSQL βάσεις. Αν και τα LLMs αποδίδουν ικανοποιητικά σε απλά σενάρια, σε πιο σύνθετα παρατηρούνται φαινόμενα “hallucination” ή εσφαλμένες συσχετίσεις, τα οποία μπορούν να περιοριστούν με τη χρήση υβριδικών προσεγγίσεων όπου το NodaMD

λειτουργεί ως σταθερός backend. Συνολικά, το NodaMD θέτει τις βάσεις για την ανάπτυξη ενός γενικού μεταφραστή SQL σε NoSQL, ανοίγοντας τον δρόμο για περαιτέρω έρευνα στην τυποποίηση, μεταγλώττιση και σημασιολογική αντιστοίχιση ερωτημάτων σε ανομοιογενή και ετερογενή περιβάλλοντα δεδομένων.

6 Βιβλιογραφικές Αναφορές

[1] Redgate Software. (2025, March 12). *historical trend of the popularity ranking of database management systems*. DB-Engines. Retrieved March 12, 2025, from https://db-engines.com/en/ranking_trend

[2] Taylor, P. (2024, November 21). *Data growth worldwide 2010-2028*. Statista. Retrieved March 12, 2025, from https://www.statista.com/statistics/871513/worldwide-data-created/

[3] Lu, J., & Holubová, I. (2019). Multi-model Databases: A New Journey to Handle the Variety of Data. *ACM CSUR*, *52*, 55:1–55:38. http://dx.doi.org/10.1145/0000000.0000000

[4] Duggan et al., J., & al., e. (2015). The BigDAWG Polystore System. *SIGMOD*, *44*, 11-16. https://homes.cs.washington.edu/~magda/papers/duggan-sigrec15.pdf

[5] Elmore et.al, A. J. (2015). A Demonstration of the BigDAWG Polystore System. *VLDB Endow*, *8*, 1908–1911. https://www.vldb.org/pvldb/vol8/p1908-Elmore.pdf

[6] Kharlamov et al, E. (2016). semantic approach to polystores. *IEEE Big Data*, 2565–2573. 10.1109/BigData.2016.7840898

[7] Codd, E. F. (1970). A relational model of data for large shared data banks. *Association for Computing Machinery*, *13*(6). 10.1145/362384.362685

[8] ArangoDB. (n.d.). *ArangoDB: Multi-Model Database for Your Modern Apps*. Retrieved 2025, from https://arangodb.com/

[9] OrientDB. (n.d.). *OrientDB: An SAP Company*. Retrieved 2025, from https://orientdb.org/

[10] Koutroumanis et al, N. (2021). A Demonstration of NoDA: Unified Access to NoSQL Stores. *VLDB Endowment*, *14*(12), 2851-2854. 10.14778/3476311.3476361

[11] Silberschatz, A., Korth, H. F., & Sudarshan, S. (1997). *Database system concepts*. McGraw-Hill.

[12] Katembo et al., E. (2019). A systematic review on Distributed Databases Systems and their techniques. *Journal of Theoretical and Applied Information Technology*, *96*.

[13] Parker, Z., Poe, S., & Vrbsky, S. V. (2013). Comparing NoSQL MongoDB to an SQL db. *Proceedings of the 51st ACM Southeast Conference*.

[14] Abramova, V., Bernardino, J., & Nuno, P. (2014). Which NoSQL Database? A Performance Overview. *Open J. Databases*, *1*, 17-24. https://api.semanticscholar.org/CorpusID:2047944

[15] Friedrich, S., & Ritter, N. (n.d.). YCSB. *Encyclopedia of Big Data Technologies*, 1-4. 10.1007/978-3-319-63962-8_131-1

[16] Martins, P., Abbasi, M., & Filipe, S. (2019). A Study over NoSQL Performance. *New Knowledge in Information Systems and Technologies*, 603-611. https://doi.org/10.1007/978-3-030-16181-1_57

[17] Truica, C.-O., Radulescu, F., Boicea, A., & Bucur, I. (2015). Performance Evaluation for CRUD Operations in Asynchronously Replicated Document Oriented Database. *2015 20th International Conference on Control Systems and Computer Science*. 10.1109/CSCS.2015.32

[18] www.chatgpt.com

[19] Hong, Z., Yuan, Z., Zhang, Q., Chen, H., Dong, J., Huang, F., & Huang, X. (2025). Next-generation database interfaces: A survey of llm-based text-to-sql. IEEE Transactions on Knowledge and Data Engineering.

[20] Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., ... & Li, Y. (2023). Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. Advances in Neural Information Processing Systems, 36, 42330-42357.

[21] Lao, J., Wang, Y., Li, Y., Wang, J., Zhang, Y., Cheng, Z., ... & Wang, J. (2025). GPTuner: An LLM-Based Database Tuning System. ACM SIGMOD Record, 54(1), 101-110.