ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδίαση και Ανάπτυξη Ενιαίου και Ολοκληρωμένου Συστήματος Διαχείρισης Ταυτόχρονων Συναλλαγών σε Κατανεμημένες Εγγραφο-Κεντρικές Βάσεις Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΔΕΛΗΓΙΑΝΝΗ ΓΕΩΡΓΙΟΥ

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Ιούλιος 2016

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδίαση και Ανάπτυξη Ενιαίου και Ολοκληρωμένου Συστήματος Διαχείρισης Ταυτόχρονων Συναλλαγών σε Κατανεμημένες Εγγραφο-Κεντρικές Βάσεις Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΔΕΛΗΓΙΑΝΝΗ ΓΕΩΡΓΙΟΥ

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 27η Ιουλίου 2016.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Βαρβαρίγου Θεοδώρα	Βαρβαρίγος Εμμανουήλ	Λούμος Βασίλειος
Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2016

.....

Γεώργιος Δεληγιάννης

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

Περίληψη

Η σύγχρονη εποχή χαρακτηρίζεται ως η εποχή του Μεγάλου Όγκου Δεδομένων. Η ευρύτατη χρήση του Διαδικτύου, σε συνδυασμό με την ανάπτυξη των τεχνολογιών νέφους, οδήγησαν στην παραγωγή τεράστιου όγκου δεδομένων και συνεπώς στην ανάγκη διαχείρισης του. Η εξέλιξη αυτή οδήγησε στη δημιουργία των μη σχεσιακών NoSQL βάσεων δεδομένων. Οι NoSQL βάσεις δεδομένων προσφέρουν μεγάλη ευελιξία ως προς τη δομή τους, έχοντας δυναμικό σχήμα, και αποδοτική αποθήκευση μεγάλου όγκου δεδομένων.

Ωστόσο, τα NoSQL συστήματα στερούνται τη δυνατότητα υποστήριξης συναλλαγών και ACID ιδιοτήτων, περιορίζοντας έτσι τη χρήση τους σε πολύ συγκεκριμένες εφαρμογές. Στα πλαίσια της παρούσας διπλωματικής επεκτείνονται οι NoSQL βάσεις δεδομένων MongoDB και CouchDB προκειμένου να υποστηρίζουν διαχείριση συναλλαγών και να προσφέρουν τις ACID ιδιότητες που προσφέρουν οι σχεσιακές βάσεις δεδομένων. Η υλοποίηση βασίζεται στην τεχνική του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων και στο επίπεδο απομόνωσης που ονομάζεται Απομόνωση Στιγμιότυπου. Επίσης, υλοποιείται μία ενιαία διεπαφή λειτουργιών η οποία δίνει τη δυνατότητα στο χρήστη να εκτελεί με όμοιο τρόπο τις βασικές λειτουργίες και των δύο βάσεων δεδομένων. Δημιουργείται κατά αυτό τον τρόπο ένα ενιαίο σύστημα που προσφέρει δύο διαφορετικές NoSQL βάσεις δεδομένων οι οποίες παρέχουν ACID ιδιότητες και ταυτόχρονες συναλλαγές.

Λέξεις Κλειδιά: Μη-σχεσιακές βάσεις δεδομένων, NoSQL, MongoDB, CouchDB, ACID, Έλεγχος Ταυτοχρονισμού Πολλαπλών Εκδόσεων, Διαχείριση Συναλλαγών, Απομόνωση Στιγμιότυπου, Java

Abstract

Modern era is characterized as the era of Big Data. The wide use of Internet, in combination with the technology of Cloud Computing, led to the production of vast amounts of data and, therefore, in the need of the management of Big Data. This progress led to the creation of the non-relational NoSQL database systems. The NoSQL databases offer great structure flexibility, with dynamic schema, and efficient storage of large amounts of data.

Although, NoSQL database systems lack of the capability to support transactions and offer ACID properties, thereby limiting their use to very specific applications. In this diploma thesis, the NoSQL databases MongoDB and CouchDB are being extended to support transaction management and offer ACID properties to the transactions, as offered by the relational database systems. The implementation is based on the technique of Multi Version Concurrency Control and the isolation level called Snapshot Isolation. Furthermore, a unified interface is implemented which enables the user to perform the basic functions of the two databases in a similar way. Therefore, it is created a unified system which supports two different NoSQL databases that provide ACID properties and concurrent transactions.

Keywords: Non-relational databases, NoSQL, MongoDB, CouchDB, ACID, Multi Version Concurrency Control, Transaction Management, Snapshot Isolation, Java

Ευχαριστίες

Η παρούσα διπλωματική εργασία αποτελεί την ολοκλήρωση ενός κύκλου σπουδών στη σχολή των Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Η διπλωματική εργασία εκπονήθηκε στο εργαστήριο Κατανεμημένης Γνώσης και Συστημάτων Πληροφορικής υπό την επίβλεψη της καθηγήτριας Θεοδώρας Βαρβαρίγου, την οποία θα ήθελα να ευχαριστήσω θερμά για την ευκαιρία που μου παρείχε να εκπονήσω τη διπλωματική μου στο συγκεκριμένο εργαστήριο και για την εμπιστοσύνη που μου έδειξε. Επίσης, θα ήθελα να ευχαριστήσω ιδιαιτέρως τον υποψήφιο διδάκτορα Κρανά Παύλο τόσο για την πολύτιμη γνώση και τεχνική βοήθεια που μου πρόσφερε κατά την υλοποίηση της εργασίας όσο και για την γενικότερη στήριξη και υπομονή που παρείχε κατά το διάστημα της συνεργασίας μας.

Τέλος, θα ήθελα να ευχαριστήσω θερμότατα τους γονείς και τα αδέρφια μου για όλη τη στήριξη που μου παρέχουν όλα αυτά τα χρόνια όντας αρωγοί της προσπάθειας μου . Φυσικά, οφείλω ένα τεράστιο ευχαριστώ και στους σημαντικούς μου φίλους και τα κοντινά μου πρόσωπα που καθ' όλη τη διάρκεια των σπουδών ήταν δίπλα μου παρέχοντας μου γνώση και ψυχολογική υποστήριξη.

Πίνακας περιεχομένων

1

Εισαγωγή

1.1 *Διαχείριση ταυτόχρονων συναλλαγών σε μη σχεσιακές βάσεις δεδομένων*

Η τεράστια ανάπτυξη και εξέλιξη του διαδικτυακού χώρου και η εισαγωγή των πληροφοριακών συστημάτων σε όλους τους τομείς του επαγγελματικού τομέα έχουν αλλάξει ριζικά το τοπίο της καθημερινότητας και της εργασίας. Μία από τις βασικότερες συνέπειες είναι ο τεράστιος ρυθμός παραγωγής δεδομένων, γεγονός που οδηγεί στην ανάγκη για αποδοτική διαχείριση και αποθήκευση αυτού του τεράστιου όγκου δεδομένων. Απόδειξη αυτού αποτελούν τόσο το γεγονός ότι πρόσφατες μετρήσεις καταδεικνύουν πως κάθε δύο ημέρες παράγεται όγκος δεδομένων ίσος με τον όγκο των δεδομένων που είχαν παραχθεί μέχρι το έτος 2003, όσο και το γεγονός πως ο συνολικός όγκος των δεδομένων μέχρι το έτος 2015 υπολογίστηκε στα 4.4 zettabytes και υπολογίζεται πως μέχρι το 2020 ο όγκος θα έχει δεκαπλασιαστεί αγγίζοντας τα 44 zettabytes. Αυτή η εξέλιξη οδήγησε στην δημιουργία νέων επαγγελματικών τομέων, όπως ο χώρος των Big Data, αλλά και στην περαιτέρω ανάπτυξη επιστημών όπως η ανάλυση δεδομένων (Data Analysis). Ασφαλώς, ένας από τους κύριους τομείς που επηρεάστηκε άμεσα ήταν ο χώρος των βάσεων δεδομένων, όπου υπήρξε η ανάπτυξη νέων βάσεων δεδομένων που ανταποκρίνονται αποδοτικότερα στις σύγχρονες ανάγκες. Ένας βασικός νέος τύπος βάσεων δεδομένων είναι οι μη σχεσιακές NoSQL βάσεις δεδομένων.

Ο νέος αυτός τύπος βάσεων δεδομένων προσφέρει μεγάλη ταχύτητα στη διαχείριση των ερωτημάτων, αποδοτική αποθήκευση των δεδομένων και οριζόντια κλιμακωσιμότητα, η οποία αποτελεί απαραίτητη προϋπόθεση για την αξιοποίηση του Cloud Computing. Ωστόσο, οι NoSQL βάσεις δεδομένων δεν υποστηρίζουν τη σημασιολογία των συναλλαγών και τις ιδιότητες ACID. Η υποστήριξη συναλλαγών και ACID ιδιοτήτων είναι μία δυνατότητα που διευκολύνει σημαντικά τους προγραμματιστές και σε ορισμένες εφαρμογές είναι απαραίτητη η υποστήριξη της δυνατότητας αυτής. Συνεπώς, έχουν γίνει σημαντικές προσπάθειες για την επέκταση NoSQL βάσεων δεδομένων που θα προσφέρουν δυνατότητα ύπαρξης συναλλαγών και παροχή ACID ιδιοτήτων.

1.2 *Αντικείμενο διπλωματικής*

Αντικείμενο της παρούσας διπλωματικής αποτελεί η σχεδίαση και η υλοποίηση ενός ολοκληρωμένου συστήματος που θα παρέχει επεκτάσεις των NoSQL βάσεων δεδομένων MongoDB και CouchDB, οι οποίες θα έχουν τη δυνατότητα διαχείρισης ταυτόχρονων συναλλαγών. Πιο συγκεκριμένα, στόχος είναι η επέκταση των MongoDB και CouchDB ώστε να προσφέρουν την δυνατότητα πολλαπλών εκδόσεων των εγγραφών προκειμένου να υλοποιηθεί η τεχνική του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων. Επίσης, θα υλοποιηθεί μία ενιαία διεπαφή που θα προσφέρει στο χρήστη όμοιες εντολές για την εκτέλεση βασικών λειτουργιών των βάσεων δεδομένων (CRUD) παρέχοντας έτσι στο χρήστη ένα ενιαίο σύστημα διαχείρισης ταυτόχρονων συναλλαγών στις NoSQL κατανεμημένες βάσεις δεδομένων MongoDB και CouchDB.

1.2.1 Συνεισφορά

Για την υποστήριξη και διαχείριση συναλλαγών και την παροχή ACID ιδιοτήτων χρησιμοποιείται ένα εξωτερικό πρόγραμμα Διαχείρισης Συναλλαγών (Transactional Management). Ένας τέτοιος Ολιστικός Διαχειριστής Συναλλαγών (Holistic Transaction Manager - HTM) έχει αναπτυχθεί στα πλαίσια του έργου CoherentPaaS[22]. Επίσης, η επέκταση της MongoDB ώστε να υποστηρίζει πολλαπλές εκδόσεις και να επικοινωνεί με τον HTM προκειμένου να υλοποιεί τον Έλεγχο Ταυτοχρονισμού Πολλαπλών Εκδόσεων έχει γίνει στα πλαίσια του έργου CoherentPaaS.

Στην παρούσα διπλωματική γίνεται χρήση των δύο παραπάνω υλοποιήσεων για την υλοποίηση του απαιτούμενου συστήματος. Κατά συνέπεια, η υλοποίηση έγκειται στην επέκταση της CouchDB, προκειμένου να υλοποιεί Έλεγχο Ταυτοχρονισμού Πολλαπλών Εκδόσεων παρέχοντας δυνατότητα συναλλαγών και ACID ιδιότητες, και στην σχεδίαση και υλοποίηση ενός ενιαίου συστήματος που να περιλαμβάνει τις επεκτάσεις των MongoDB και CouchDB.

1.3 *Οργάνωση κειμένου*

Η παρούσα διπλωματική εργασία χωρίζεται σε 8 επιμέρους κεφάλαια. Το 1ο κεφάλαιο περιλαμβάνει την εισαγωγή όπου καθορίζεται ο στόχος της διπλωματικής εργασίας και μια μικρή περιγραφή σχετικά με την υλοποίηση και τα προβλήματα που πρόκειται να λυθούν. Το 2ο κεφάλαιο περιλαμβάνει την παρουσίαση των κυριότερων θεωρητικών εννοιών που απαιτούνται για την κατανόηση της εργασίας καθώς και την θεωρητική ανάλυση των βάσεων δεδομένων που χρησιμοποιήθηκαν. Στο 3ο κεφάλαιο αναλύεται η υλοποίηση της επέκτασης της MongoDB που πραγματοποιήθηκε στα πλαίσια του CoherentPaaS και μία συνοπτική σύγκριση μεταξύ των MongoDB και CouchDB για την ανάδειξη των διαφοροποιήσεων που θα προκύψουν στην υλοποίηση της CouchDB. Στο 4ο κεφάλαιο πραγματοποιείται η περιγραφή της σχεδίασης του ενιαίου συστήματος και των δυνατοτήτων που θα δίνει στο χρήστη-προγραμματιστή. Το 5ο κεφάλαιο περιλαμβάνει την λεπτομερή ανάλυση της υλοποίησης και την περιγραφή των βασικών εργαλείων που χρησιμοποιήθηκαν. Το 6ο κεφάλαιο αποτελεί μία επίδειξη της λειτουργίας του συστήματος όπου παρουσιάζονται σε εικόνες οι βασικές λειτουργίες καθώς και η ορθή εκτέλεση τους στα δύο συστήματα βάσεων δεδομένων. Το 7ο κεφάλαιο παραθέτει μία σύνοψη της εργασίας και παρουσιάζει χρήσιμε επεκτάσεις που θα μπορούσαν να πραγματοποιηθούν. Τέλος, το 8ο κεφάλαιο περιλαμβάνει τη βιβλιογραφία που χρησιμοποιήθηκε και τις αναφορές που έγιναν.

2

Περιγραφή Θεωρητικών Εννοιών

Στο συγκεκριμένο κεφάλαιο θα αναλύσουμε τις βασικές θεωρητικές έννοιες στις οποίες βασίστηκε η ανάλυση και η υλοποίηση της συγκεκριμένης διπλωματικής εργασίας. Οι συγκεκριμένες έννοιες είναι απαραίτητες στον αναγνώστη για την κατανόηση της συγκεκριμένης εργασίας. Αρχικά, θα αναλυθούν οι ιδιότητες ACID, οι οποίες είναι απαραίτητο να τηρούνται από το σύστημα βάσης δεδομένων και να χαρακτηρίζουν τις εκάστοτε δοσοληψίες. Στη συνέχεια θα αναλύσουμε τα διάφορα επίπεδα απομόνωσης που μπορούν να υπάρξουν και θα εστιάσουμε ιδιαίτερα στο επίπεδο απομόνωσης στιγμιότυπου (Snapshot Isolation). Τέλος, θα εξηγήσουμε την τεχνική ελέγχου ταυτοχρονισμού με πολλαπλές εκδόσεις (Multi Version Concurrency Control) και τα πλεονεκτήματα που αυτή μας προσφέρει, και θα κλείσουμε το κεφάλαιο με μία σύγκριση χαρακτηριστικών των δύο NoSQL βάσεων δεδομένων που χρησιμοποιήθηκαν σε αυτή τη διπλωματική, τη MongoDB και την CouchDB. Προτού ξεκινήσει η ανάλυση των θεωρητικών εννοιών πρέπει να αναφερθεί πως με τον όρο δοσοληψία ή συναλλαγή (transaction) αναφερόμαστε σε μία λειτουργία που θα πραγματοποιηθεί στη βάση δεδομένων και η οποία αποτελείται από 1 ή περισσότερες επιμέρους στοιχειώδεις ενέργειες που θα εφαρμοστούν στο σύστημα βάσης δεδομένων.

2.1 *ACID Ιδιότητες*

Με το ακρωνύμιο ACID[2][3] αναφερόμαστε στις 4 ιδιότητες που θέλουμε να διασφαλίζει ένα σύστημα βάσεων δεδομένων για κάθε πιθανή δοσοληψία προκειμένου να είμαστε βέβαιοι πως τα δεδομένα που χρησιμοποιεί κάθε συναλλαγή είναι έγκυρα και δεν υπάρχει πιθανότητα να βρεθεί η βάση δεδομένων σε μη έγκυρη κατάσταση. Ο όρος ACID προέκυψε από τα αρχικά των λέξεων Ατομικότητα (Atomicity), Συνέπεια(Consistency), Απομόνωση (Isolation) και Μονιμότητα (Durability), κάθε μία εκ των οποίων αναφέρεται σε μία ξεχωριστή ιδιότητα.

Ατομικότητα: Με τον όρο ατομικότητα αναφερόμαστε στην ιδιότητα που πρέπει να έχει κάθε συναλλαγή βάσει της οποίας εάν έστω μία επιμέρους ενέργεια της συναλλαγής αποτύχει να εκτελεστεί, τότε όλη η συναλλαγή πρέπει να ακυρωθεί και η βάση δεδομένων δεν πρέπει να υποστεί καμία αλλαγή από τη συγκεκριμένη δοσοληψία. Ο λόγος που επιθυμούμε να διασφαλίζεται η συγκεκριμένη ιδιότητα είναι ότι η συναλλαγή αντιμετωπίζεται ως μία και αδιαίρετη, συνεπώς το να εκτελεστούν μόνο ορισμένα τμήματά της και τα υπόλοιπα να αποτύχουν είναι ανεπιθύμητο.

Συνέπεια: Όπως γνωρίζουμε κάθε σύστημα βάσεων δεδομένων υπόκειται σε ορισμένους κανόνες και περιορισμούς οι οποίο έχουν οριστεί από τους διαχειριστές του συστήματος και σε καμία περίπτωση δεν επιτρέπεται να παραβιάζονται. Επομένως, η συγκεκριμένη ιδιότητα αφορά την απαίτηση μετά από κάθε συναλλαγή το σύστημα να βρίσκεται σε μία έγκυρη κατάσταση και να μην παραβιάζει κανέναν από τους κανόνες και τους περιορισμούς, οι οποίοι έχουν οριστεί για αυτό. Πρέπει να τονιστεί ωστόσο πως η συγκεκριμένη ιδιότητα δεν διασφαλίζει πως η συναλλαγή θα έχει τα επιθυμητά αποτελέσματα, καθώς αυτό εξαρτάται από τον προγραμματιστή που δημιουργεί τη συναλλαγή.

Απομόνωση: Η συγκεκριμένη ιδιότητα προκύπτει από τα προβλήματα που εμφανίζονται σε ένα σύστημα βάσης δεδομένων λόγω των ταυτόχρονων συναλλαγών και την εξάρτηση που εμφανίζεται σχετικά με τα αποτελέσματα τους. Το επιθυμητό θα ήταν κάθε συναλλαγή να εκτελείται και να ολοκληρώνεται χωρίς καμία αλληλεπίδραση από άλλες ταυτόχρονες συναλλαγές. Η ιδιότητα της απομόνωσης λοιπόν διασφαλίζει πως κάθε συναλλαγή θα εκτελεστεί όπως θα συνέβαινε αν δεν υπήρχαν άλλες ταυτόχρονες συναλλαγές. Πιο απλοϊκά, η έννοια της απομόνωσης εξασφαλίζει πως μια ομάδα ταυτόχρονων συναλλαγών θα έχουν το ίδιο αποτέλεσμα με αυτό που θα προέκυπτε εάν εκτελούνταν σειριακά χωρίς επικαλύψεις.

Μονιμότητα: Με τον όρο μονιμότητα χαρακτηρίζεται η απαίτηση που υπάρχει από το σύστημα της βάσης δεδομένων να διατηρεί τα δεδομένα της οτιδήποτε και αν συμβεί. Αυτό σημαίνει πως θα πρέπει να διασφαλίζεται πως σε οποιαδήποτε αποτυχία συστήματος ή σφάλματος, τα δεδομένα που οφείλονται σε συναλλαγές που έχουν ολοκληρωθεί επιτυχώς διατηρούνται ακέραια και αναλλοίωτα ακόμη και μετά την επανεκκίνηση του συστήματος. Για τις συναλλαγές που βρίσκονται σε εξέλιξη κατά τη στιγμή που συνέβη κάποια αποτυχία συστήματος εφαρμόζεται η ιδιότητα της ατομικότητας. [1]

Οι τέσσερις παραπάνω ιδιότητες είναι εξαιρετικά θεμελιώδεις για τη διασφάλιση της ορθής λειτουργίας ενός συστήματος βάσεων δεδομένων που υποστηρίζει ταυτόχρονες συναλλαγές και ιδιαίτερα στην περίπτωση που υπάρχει τεράστιος όγκος δεδομένων. Πιο συγκεκριμένα, ένα σύστημα που παρέχει τις παραπάνω ιδιότητες απελευθερώνει τον προγραμματιστή εφαρμογών

[1] [2]https://en.wikipedia.org/wiki/ACID, [3]http://databases.about.com/od/specificproducts/a/acid.htm

από τον έλεγχο των ταυτόχρονων συναλλαγών, την διασφάλιση της ορθότητας των συναλλαγών του ως προς τους περιορισμούς της βάσης δεδομένων και τον χειρισμό των σφαλμάτων και των αποτυχιών συστήματος. Όλα αυτά τα χειρίζεται το σύστημα βάσεων δεδομένων με απόλυτα ασφαλή και αξιόπιστο τρόπο.

2.2 *Επίπεδα Απομόνωσης*

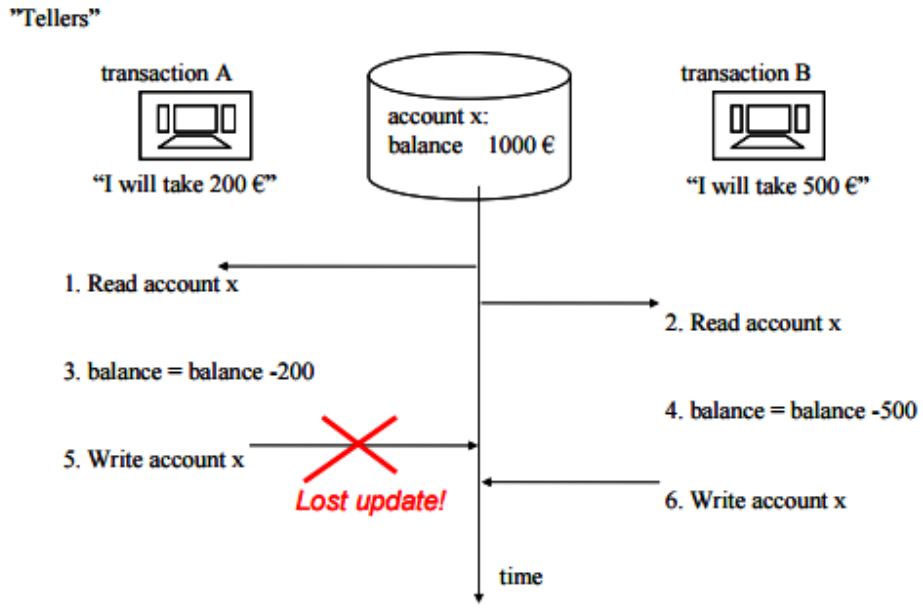
Στο συγκεκριμένο κεφάλαιο θα αναλύσουμε εκτενέστερα την έννοια της απομόνωσης, θα δούμε τα διάφορα επίπεδα απομόνωσης που υπάρχουν, τα προβλήματα που εντοπίζονται σε κάθε ένα από αυτά καθώς και το πως το επίπεδο απομόνωσης που θα επιλέξουμε επηρεάζει την επίδοση του συστήματος βάσεων δεδομένων.

Όπως περιγράψαμε, η έννοια της απομόνωσης σχετίζεται με την ανάγκη που υπάρχει για ταυτόχρονες συναλλαγές και τον έλεγχο του ταυτοχρονισμού που εφαρμόζεται. Πιο συγκεκριμένα, το επίπεδο απομόνωσης που θα επιλέξουμε καθορίζει το κατά πόσον τα δεδομένα που επεξεργάζονται από μία συναλλαγή είναι ορατά από μία άλλη ταυτόχρονη συναλλαγή. Όπως είναι αναμενόμενο, όσο πιο υψηλό επίπεδο απομόνωσης επιλέξουμε τόσο λιγότερα δεδομένα μιας συναλλαγής είναι ορατά σε άλλες ταυτόχρονες συναλλαγές, ενώ τα χαμηλότερα επίπεδα απομόνωσης προσφέρουν μεγαλύτερη ΄χαλαρότητα΄ σχετικά με την ταυτόχρονη πρόσβαση σε δεδομένα. Επομένως, είναι λογικό τα χαμηλότερα επίπεδα απομόνωσης να επιτρέπουν διαφορετικές αναγνώσεις δεδομένων συγκριτικά με τα πιο αυστηρά, υψηλότερα επίπεδα.Όσο πιο χαμηλό επίπεδο απομόνωσης επιλέγεται τόσο πιο ευάλωτο είναι το σύστημα μας σε πιθανά λανθασμένα αποτελέσματα, καθώς υπάρχουν πολύ συγκεκριμένα φαινόμενα σφαλμάτων που εμφανίζονται στις ταυτόχρονες συναλλαγές χωρίς έλεγχο ταυτοχρονισμού λόγω της ταυτόχρονης πρόσβασης σε ίδια δεδομένα από διαφορετικές συναλλαγές. Θα ξεκινήσουμε την ανάλυση μας παρουσιάζοντας τα φαινόμενα σφαλμάτων που παρουσιάζονται στις ταυτόχρονες συναλλαγές και στη συνέχεια θα παρουσιάσουμε τα επίπεδα απομόνωσης, τονίζοντας ποια σφάλματα επιτρέπουν και ποια αποτρέπουν.

Πρόχειρη Ανάγνωση: Το πρόβλημα της πρόχειρης ανάγνωσης (dirty read) εμφανίζεται όταν μία συναλλαγή Α επεξεργάζεται την τιμή μιας εγγραφής Χ, στη συνέχεια μία συναλλαγή Β (ταυτόχρονη με την Α) διαβάζει την εγγραφή Χ (προτού η Α ολοκληρωθεί) και εν συνεχεία η συναλλαγή Α ακυρώνεται (abort ή rollback). Στην περίπτωση αυτή, όπως γίνεται κατανοητό, η τιμή που διαβάζει η συναλλαγή Β για την εγγραφή Χ είναι λανθασμένη καθώς διάβασε μία τιμή η οποία εν τέλει ακυρώθηκε και άρα δεν θα έπρεπε να διαβάσει αυτή την τιμή αλλά την αμέσως προηγούμενη τιμή της εγγραφής.

Χαμένες Ενημερώσεις: Το φαινόμενο των χαμένων ενημερώσεων (lost updates) εμφανίζεται στην εξής περίπτωση: Μία συναλλαγή Α διαβάζει μία εγγραφή Χ, στη συνέχεια μία

ταυτόχρονη συναλλαγή Β διαβάζει την εγγραφή Χ, ακολούθως η συναλλαγή Α αλλάζει την τιμή της εγγραφής Χ και επιβεβαιώνει την συναλλαγή (commit transaction A) και τέλος η συναλλαγή Β ενημερώνει την εγγραφή Χ και επιβεβαιώνει την συναλλαγή (commit transaction B). Αυτό που συμβαίνει είναι ότι η αλλαγή που πραγματοποίησε η συναλλαγή Α στην εγγραφή Χ αντικαθίσταται αμέσως από την αλλαγή που πραγματοποίησε η συναλλαγή Β στην Χ και ουσιαστικά χάνεται. Σε πρώτη ανάγνωση δεν είναι εμφανές το πρόβλημα, ωστόσο η ανωμαλία εντοπίζεται στο γεγονός ότι αν η συναλλαγή Β έβλεπε την αλλαγή της εγγραφής Χ από την Α θα προκαλούσε διαφορετική μεταβολή της τιμής της Χ. Ένα παράδειγμα που καταδεικνύει πλήρως το πρόβλημα της χαμένης ενημέρωσης παρουσιάζεται στο ακόλουθο σχήμα.



Σχήμα 1.1: Lost Update

Όπως βλέπουμε το τελικό αποτέλεσμα στον λογαριασμό είναι 500, ενώ θα έπρεπε να είναι 300 καθώς το συνολικό ποσό που αφαιρέθηκε από το λογαριασμό από τις 2 ταυτόχρονες συναλλαγές είναι 700. [2]

Μη-επαναλαμβανόμενες αναγνώσεις: Το πρόβλημα των μη επαναλαμβανόμενων αναγνώσεων (non-repeatable reads) προκαλείται όταν κατά τη διάρκεια μιας συναλλαγής η ανάγνωση μιας εγγραφής περισσότερες από μία φορές, δίνει διαφορετικά αποτελέσματα. Πιο συγκεκριμένα, έστω ότι μία συναλλαγή Α διαβάζει την εγγραφή Χ και βρίσκει ότι έχει τιμή 10, στη συνέχεια μία συναλλαγή Β (ταυτόχρονη με την Α) αλλάζει την τιμή της Χ σε 20 και

[2] [1] http://myy.haaga-helia.fi/~dbms/dbtechnet/download/SQL-Transactions_handbook_GR.pdf

επιβεβαιώνει (commit) και τέλος η συναλλαγή Α που δεν έχει ολοκληρωθεί διαβάζει ξανά την Χ με αποτέλεσμα να διαβάσει τιμή 20.

Ανάγνωση φαντάσματος: Το φαινόμενο της ανάγνωσης φαντάσματος εμφανίζεται στην περίπτωση που το ίδιο ερώτημα (query) που εκτελείται περισσότερες από μία φορές μέσα σε μία συναλλαγή Α επιστρέφει διαφορετικό σύνολο εγγραφών. Όπως γίνεται κατανοητό, το συγκεκριμένο πρόβλημα οφείλεται στο γεγονός ότι κάποια άλλη ταυτόχρονη συναλλαγή Β ενημέρωσε κάποιες εγγραφές ανάμεσα στα χρονικά σημεία που εκτελέστηκε το ερώτημα από τη συναλλαγή Α.

Έχοντας αναφερθεί στα κυριότερα προβλήματα που εμφανίζονται στις ταυτόχρονες συναλλαγές, θα παρουσιαστούν τα επίπεδα απομόνωσης, αναλύοντας τα, τονίζοντας ποια προβλήματα αποφεύγονται σε κάθε επίπεδο αλλά και την απόδοση που θυσιάζεται για την αντιμετώπιση των προβλημάτων. Όπως σημειώσαμε τα επίπεδα απομόνωσης διαφοροποιούνται στο κατά πόσον επιτρέπουν ταυτόχρονη πρόσβαση σε δεδομένα από διαφορετικές συναλλαγές. Συνεπώς, θα δούμε πως κάθε επίπεδο απομόνωσης επιβάλλει διαφορετικούς περιορισμούς ως προς την ταυτόχρονη ανάγνωση και εγγραφή δεδομένων από διαφορετικές συναλλαγές, χρησιμοποιώντας αρκετούς μηχανισμούς για την διασφάλιση αυτών των περιορισμών. Προτού αναλύσουμε όμως τα επίπεδα απομόνωσης, είναι απαραίτητο να διασαφηνίσουμε δύο σημαντικές έννοιες. Με τον όρο κλείδωμα ανάγνωσης (read lock) ή μοιραζόμενο κλείδωμα (shared lock) θα αναφερόμαστε στην έννοια του κλειδώματος μιας εγγραφής από μία συναλλαγή που επιθυμεί να αναγνώσει την εγγραφή αυτή και αποτρέπει οποιαδήποτε άλλη ταυτόχρονη συναλλαγή να ενημερώσει τη συγκεκριμένη εγγραφή αλλά επιτρέπει άλλες συναλλαγές να διαβάσουν την εγγραφή. Αντίστοιχα με τον όρο κλείδωμα ενημέρωσης (write lock) ή αποκλειστικό κλείδωμα (exclusive lock) αναφερόμαστε στο κλείδωμα μιας εγγραφής από μία συναλλαγή προκειμένου αυτή να ενημερώσει την εγγραφή και αποτρέπει άλλες ταυτόχρονες συναλλαγές να διαβάσουν ή να ενημερώσουν την εγγραφή αυτή.

Σειριοποιήσιμο (Serializable): Το συγκεκριμένο επίπεδο απομόνωσης είναι το υψηλότερο και πιο αυστηρό περιορίζοντας σε πολύ μεγάλο βαθμό τις επιτρεπτές ταυτόχρονες συναλλαγές. Πιο συγκεκριμένα, σε αυτό το επίπεδο μία συναλλαγή μπορεί να διαβάσει ένα δεδομένο αποτρέποντας οποιαδήποτε άλλη ταυτόχρονη συναλλαγή να το ενημερώσει, επιτρέποντας όμως οποιαδήποτε άλλη ταυτόχρονη συναλλαγή να το διαβάσει. Συνεπώς σε αυτό το επίπεδο, για να πραγματοποιήσει μία συναλλαγή ανάγνωση ενός δεδομένου πρέπει να αποκτήσει ένα κλείδωμα ανάγνωσης για το συγκεκριμένο δεδομένο, το οποίο θα το αφήσει μόλις επιβεβαιώσει την ολοκλήρωση της συναλλαγής (commit). Σε αυτό το επίπεδο απομόνωσης, για να ενημερώσει μία συναλλαγή κάποια εγγραφή πρέπει να αποκτήσει κλείδωμα ενημέρωσης

για την εγγραφή αυτή. Αυτό σημαίνει πως η ενημέρωση μιας εγγραφής από μία συναλλαγή εμποδίζει όλες τις άλλες ταυτόχρονες συναλλαγές να διαβάσουν ή να ενημερώσουν τη συγκεκριμένη εγγραφή. Το κλείδωμα ενημέρωσης ελευθερώνεται από τη συναλλαγή μετά το στάδιο της επιβεβαίωσης (commit). Με βάση τα παραπάνω, είναι προφανές πως στο συγκεκριμένο στάδιο δεν εμφανίζονται τα προβλήματα της πρόχειρης ανάγνωσης, των χαμένων ενημερώσεων και των μη επαναλαμβανόμενων αναγνώσεων. Ωστόσο, για να αποφευχθεί και το πρόβλημα της ανάγνωσης φαντάσματος χρησιμοποιούνται κλειδιά εύρους τιμής όταν πραγματοποιούνται ερωτήματα (queries) που περιέχουν και τον όρο WHERE. Αυτό που συμβαίνει λοιπόν είναι ότι στην περίπτωση τέτοιων ερωτημάτων κλειδώνεται όλο το εύρος τιμών που περιλαμβάνεται στο ερώτημα προκειμένου να διασφαλιστεί πως δεν θα ενημερωθεί από κάποια άλλη ταυτόχρονη συναλλαγή. Αναλύοντας το συγκεκριμένο επίπεδο απομόνωσης, προκύπτει εύκολα το συμπέρασμα πως η μόνη ταυτόχρονη λειτουργία που επιτρέπεται είναι η ανάγνωση του ίδιου δεδομένου από διαφορετικές ταυτόχρονες συναλλαγές με την προϋπόθεση πως καμία άλλη ταυτόχρονη συναλλαγή δεν έχει ενημερώσει την εγγραφή. Σε οποιαδήποτε άλλη περίπτωση αυτό που συμβαίνει είναι ότι μόνο μία από τις ταυτόχρονες συναλλαγές αποκτά πρόσβαση στο δεδομένο και οι υπόλοιπες μπλοκάρουν περιμένοντας την ολοκλήρωση της συναλλαγής που έχει πρόσβαση. Κατά συνέπεια, οδηγούμαστε πολύ συχνά στην παρεμπόδιση του ταυτοχρονισμού και στο να σειριοποιηθούν οι συναλλαγές, κάτι που έχει ως αποτέλεσμα την μειωμένη απόδοση.

Επαναλαμβανόμενες Αναγνώσεις (Repeatable Read): Σε αυτό το επίπεδο απομόνωσης, για να έχει μία συναλλαγή πρόσβαση σε ένα δεδομένο πρέπει να αποκτήσει το ανάλογο κλείδωμα (κλείδωμα ανάγνωσης ή εγγραφής) και το κλείδωμα που αποκτάται ελευθερώνεται με την ολοκλήρωση του σταδίου της επιβεβαίωσης (commit). Συμβαίνει δηλαδή ότι και στο προηγούμενο επίπεδο, με την διαφοροποίηση ότι δεν χρησιμοποιούνται κλειδώματα εύρους τιμών και άρα δεν αποτρέπεται η εμφάνιση του φαινομένου της ανάγνωσης φαντάσματος. Το επίπεδο αυτό είναι πιο χαλαρό κατά ένα βαθμό συγκριτικά με το προηγούμενο, ωστόσο καλύτερη απόδοση εμφανίζει αυτό το επίπεδο μόνο στην περίπτωση ερωτημάτων με χρήση το όρου WHERE. Σε οποιαδήποτε άλλη περίπτωση τα επίπεδα είναι όμοια, ακολουθούν τους ίδιους συμβιβασμούς και περιορισμούς και άρα δεν εμφανίζεται διαφορά στην απόδοση.

Ανάγνωση Επιβεβαιωμένων (Read Commited): Το συγκεκριμένο επίπεδο είναι κατά ένα βαθμό πιο χαλαρό συγκριτικά με το επίπεδο Επαναλαμβανόμενων Αναγνώσεων. Συγκεκριμένα, και πάλι η ανάγνωση απαιτεί την απόκτηση μοιραζόμενου κλειδώματος και η εγγραφή απαιτεί κλείδωμα εγγραφής. Ωστόσο η διαφοροποίηση είναι ότι πλέον ένα μοιραζόμενο κλείδωμα που έχει αποκτηθεί από μία συναλλαγή, ελευθερώνεται αμέσως μόλις ολοκληρωθεί η επιμέρους λειτουργία της συναλλαγής που απαιτούσε την ανάγνωση της

εγγραφής – και όχι μετά την επιβεβαίωση της συναλλαγής. Είναι προφανές λοιπόν πως σε αυτό το επίπεδο δεν αποτρέπεται το φαινόμενο των Μη Επαναλαμβανόμενων Αναγνώσεων, ούτε και το φαινόμενο της Ανάγνωσης Φαντάσματος φυσικά. Θυσιάζοντας όμως ορθότητα και 'χαλαρώνοντας' το επίπεδο απομόνωσης, κερδίζουμε σε απόδοση καθώς πλέον επιτρέπονται ακόμη περισσότερες ταυτόχρονες λειτουργίες και δεν παρεμποδίζεται τόσο συχνά η εξέλιξη μιας συναλλαγής.

Ανάγνωση Μη Επιβεβαιωμένων (Read Uncommitted): Το τελευταίο επίπεδο απομόνωσης αποτελεί το πιο 'χαλαρό' επίπεδο από άποψη κλειδωμάτων καθώς δεν χρησιμοποιούνται τέτοια για την παρεμπόδιση ταυτόχρονων συναλλαγών. Στο επίπεδο αυτό είναι δυνατόν να εμφανιστεί το φαινόμενο της Πρόχειρης Ανάγνωσης, όπως και τα φαινόμενα Μη Επαναλαμβανόμενων Αναγνώσεων και Ανάγνωσης Φαντάσματος που εμφανίζονται και στα υψηλότερα επίπεδα. Προφανώς, το επίπεδο αυτό παρουσιάζει τη μεγαλύτερη απόδοση συγκριτικά με τα υψηλότερα επίπεδα, αφού επιτρέπονται όλες οι ταυτόχρονες προσβάσεις σε κοινά δεδομένα από διαφορετικές συναλλαγές. Ωστόσο, όπως γίνεται κατανοητό θυσιάζεται σε πολύ μεγάλο βαθμό η ορθότητα των αποτελεσμάτων των συναλλαγών και η αξιοπιστία τους συστήματος βάσεων δεδομένων. [3]

Τα 4 επίπεδα απομόνωσης παρουσιάζονται στον ακόλουθο πίνακα συναρτήσει των σφαλμάτων.

	Χαμένη Ενημέρωση	Πρόχειρη Ανάγνωση	Μη Επαναλαμβανόμενες Αναγνώσεις	Ανάγνωση Φαντάσματος
Σειριοποιήσιμο	ΟΧΙ	ΟΧΙ	ΟΧΙ	ΟΧΙ
Επαναλαμβανόμενες Αναγνώσεις	ΟΧΙ	ΟΧΙ	ΟΧΙ	ΝΑΙ
Ανάγνωση Επιβεβαιωμένων	ΟΧΙ	ΟΧΙ	ΝΑΙ	ΝΑΙ
Ανάγνωση Μη Επιβεβαιωμένων	ΟΧΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ

Σχήμα 1.2: Σφάλματα επιπέδων απομόνωσης

Παρουσιάζοντας τα παραπάνω επίπεδα απομόνωσης, αναφερθήκαμε αρκετά στις έννοιες των κλειδωμάτων. Τα κλειδώματα είναι ένας προγραμματιστικός τρόπος για τον έλεγχο του ταυτοχρονισμού και χρησιμοποιούνται ιδιαιτέρως στις ταυτόχρονες συναλλαγές. Ωστόσο,

[3] [4] https://en.wikipedia.org/wiki/Isolation_(database_systems)

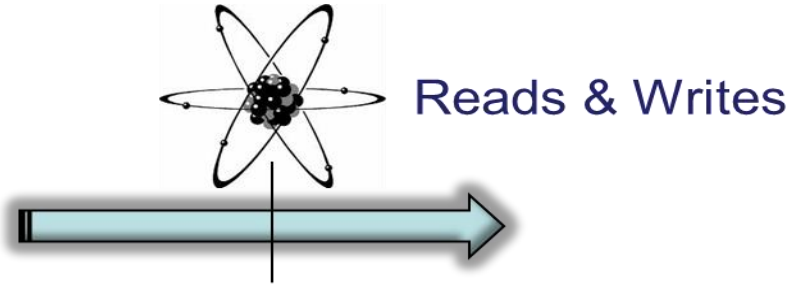
πέρα από τα κλειδώματα έχουμε αναφέρει πως ένας ακόμη τρόπος που χρησιμοποιείται για να διασφαλίσουμε την ορθότητα των ταυτόχρονων συναλλαγών είναι ο Έλεγχος Ταυτοχρονισμού Πολλαπλών Εκδόσεων (Multi Version Concurrency Control). Βασιζόμενοι σε αυτή την έννοια έχει αναπτυχθεί ένα ακόμη επίπεδο απομόνωσης το οποίο ονομάζεται Απομόνωση Στιγμιότυπου. Είναι ένα εξαιρετικά ενδιαφέρον επίπεδο απομόνωσης καθώς παρέχει την ασφάλεια που μας παρέχει το Σειριοποιήσιμο Επίπεδο (Serializable), προσφέροντας όμως σημαντικά καλύτερη απόδοση. Για αυτούς τους λόγους έχει γίνει αρκετά δημοφιλές και παρέχεται από τα περισσότερα Συστήματα Βάσεων Δεδομένων. Η Απομόνωση Στιγμιότυπου (Snapshot Isolation) παρουσιάζεται αναλυτικά στην ακόλουθη ενότητα.

2.3 *Απομόνωση Στιγμιότυπου – Snapshot Isolation*

Η έννοια της Απομόνωσης Στιγμιότυπου βασίζεται στην ιδέα πως κάθε συναλλαγή που ξεκινά να εκτελείται θα βλέπει ένα στιγμιότυπο του συστήματος της βάσης δεδομένων. Με την έννοια στιγμιότυπο της βάσης δεδομένων αναφερόμαστε στην κατάσταση στην οποία βρίσκεται ένα σύστημα βάσης δεδομένων σε μία χρονική στιγμή. Με άλλα λόγια, το στιγμιότυπο αναφέρεται στις τιμές των δεδομένων που υπάρχουν στη βάση δεδομένων τη στιγμή που παίρνουμε το στιγμιότυπο. Έτσι, όταν ξεκινά μία συναλλαγή, αποκτάται ένα ιδιωτικό στιγμιότυπο της βάσης δεδομένων απεικονίζοντας την κατάσταση στην οποία βρισκόταν το σύστημα μας ακριβώς πριν ξεκινήσει η συναλλαγή, και πλέον η συναλλαγή εκτελείται με βάση τα δεδομένα του συγκεκριμένου στιγμιότυπου. Συνεπώς, όλες οι ενημερώσεις που προκαλούνται από τη συναλλαγή ουσιαστικά εφαρμόζονται στο συγκεκριμένο στιγμιότυπο, το οποίο είναι ατομικό και άρα δεν είναι ορατές στις υπόλοιπες ταυτόχρονες συναλλαγές. Μόλις ολοκληρωθούν οι επιμέρους λειτουργίες της συναλλαγής και φτάσει το στάδιο της επιβεβαίωσης (commit) τότε ελέγχεται αν η συναλλαγή είναι έγκυρη και εάν πράγματι είναι, τότε όλες οι αλλαγές και οι ενημερώσεις που προκλήθηκαν από την συναλλαγή αυτή αποθηκεύονται μόνιμα πλέον στη βάση δεδομένων.

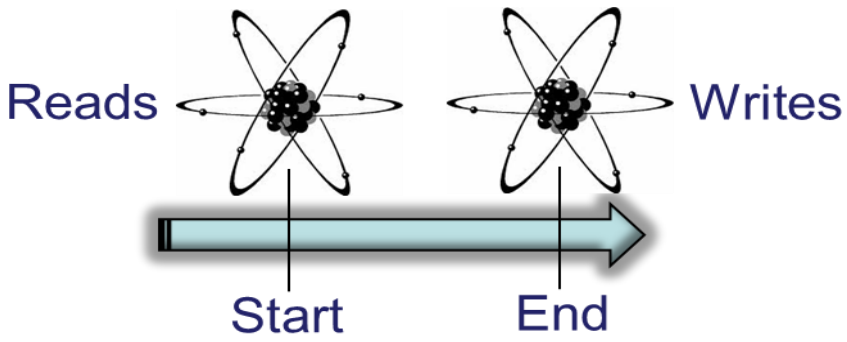
Προκύπτει εύλογα λοιπόν το συμπέρασμα πως δεν υπάρχει ανάγκη χρήσης κλειδωμάτων καθώς κάθε συναλλαγή δρα σε ένα διαφορετικό ατομικό στιγμιότυπο. Αυτό σημαίνει πως παρότι μπορεί δύο διαφορετικές ταυτόχρονες συναλλαγές να θέλουν να ενημερώσουν την ίδια εγγραφή, στην πράξη ενημερώνουν το ατομικό τους στιγμιότυπο και άρα δεν επιδρούν σε κοινά δεδομένα οπότε δεν χρειάζεται να επιβληθεί έλεγχος στο πως οι συναλλαγές θα προσπελάσουν κοινά δεδομένα. Συμπεραίνουμε λοιπόν πως η χρήση της Απομόνωσης Στιγμιότυπου συμβάλλει στην αποφυγή χρήσης κλειδωμάτων και όπως αναφέραμε προηγουμένως συχνά υλοποιείται με τη χρήση Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων, όπως αυτός θα εξηγηθεί σε επόμενη ενότητα. Το γεγονός αυτό έχει ως άμεση συνέπεια τη βελτίωση της

απόδοσης συγκριτικά με το υψηλότερο επίπεδο απομόνωσης καθώς αποφεύγεται η σειριοποίηση των ταυτόχρονων συναλλαγών. Πλέον, οι συναλλαγές εκτελούνται χωρίς εμπόδια. Μία πολύ σημαντική διαφοροποίηση συγκριτικά με το Σειριοποιήσιμο επίπεδο απομόνωσης είναι ο τρόπος εφαρμογής της ατομικότητας. Συγκεκριμένα, στην περίπτωση του σειριοποιήσιμου επιπέδου θεωρείται πως όλες οι αναγνώσεις και εγγραφές μιας συναλλαγής συμβαίνουν σε μία χρονική στιγμή όπως απεικονίζεται στο ακόλουθο σχήμα.



Σχήμα 1.3: Σειριοποιήσιμο Επίπεδο Απομόνωσης

Αντίθετα, στην Απομόνωση Στιγμιότυπου έχουμε μία διαίρεση του συγχρονισμού της ατομικότητας σε δύο στάδια. Το πρώτο στάδιο αφορά τις αναγνώσεις και γίνεται στο ξεκίνημα της συναλλαγής με την απόκτηση συγκεκριμένου στιγμιότυπου της βάσης δεδομένων. Το δεύτερο αφορά τις εγγραφές οι οποίες θεωρείται ότι γίνεται στο τέλος της συναλλαγής με την ολοκλήρωση της (commit) και ουσιαστικά έτσι καταδεικνύεται το γεγονός πως οι εγγραφές μιας συναλλαγής γίνονται ορατές σε όλες τις υπόλοιπες εγγραφές με την διαδικασία της ολοκλήρωσης (commit). Η περιγραφή αυτή απεικονίζεται στο ακόλουθο σχήμα.



Σχήμα 1.4: Απομόνωση Στιγμιότυπου

Ωστόσο, για να αποφύγουμε όλα τα πιθανά σφάλματα που μπορεί να προκύψουν λόγω των ταυτόχρονων συναλλαγών θα πρέπει στο στάδιο της επιβεβαίωσης να ελεγχθεί αν η εκάστοτε συναλλαγή μπορεί να ολοκληρωθεί επιτυχώς, ή εάν προέκυψαν συγκρούσεις με άλλες ταυτόχρονες συναλλαγές και άρα πρέπει να ακυρωθεί και να εκτελεστεί εκ νέου από την αρχή

η συγκεκριμένη συναλλαγή. Πιο συγκεκριμένα, δείξαμε πως, εφαρμόζοντας τη στρατηγική της Απομόνωσης Στιγμιότυπου, κάθε συναλλαγή διαβάζει και ενημερώνει ιδιωτικά και όχι κοινά δεδομένα, τα οποία έχουν τις τιμές που είχαν ακριβώς πριν ξεκινήσει η συναλλαγή και είναι απολύτως συνεπή. Κατά συνέπεια, αποφεύγεται το φαινόμενο της Πρόχειρης Ανάγνωσης καθώς δεν υπάρχει περίπτωση μία συναλλαγή να δει την ενημέρωσης μίας άλλης ταυτόχρονης εγγραφής. Επίσης, αποφεύγονται και τα σφάλματα της Ανάγνωσης Φαντάσματος και Μη Επαναλαμβανόμενων Αναγνώσεων ακριβώς για τον ίδιο λόγο. Παρ' όλα αυτά, με βάση όσα έχουμε αναλύσει μέχρι τώρα το φαινόμενο των χαμένων ενημερώσεων δεν αποτρέπεται. Για παράδειγμα, είναι εφικτό δύο διαφορετικές ταυτόχρονες συναλλαγές, οι Α και Β, να ενημερώσουν την ίδια εγγραφή με συνέπεια η συναλλαγή που επιβεβαιώνει (commit) δεύτερη να αντικαθιστά την ενημέρωση που προκάλεσε η πρώτη, κάτι που δεν είναι επιθυμητό. Για να αποφευχθεί το συγκεκριμένο φαινόμενο, κάθε φορά που μία συναλλαγή φτάνει στο στάδιο της επιβεβαίωσης ελέγχεται αν έχει σύγκρουση write-write με κάποια άλλη ταυτόχρονη συναλλαγή, δηλαδή αν έχει ενημερώσει κάποια εγγραφή που έχει ενημερωθεί ταυτόχρονα και από κάποια άλλη συναλλαγή η οποία έχει ολοκληρώσει το στάδιο της βεβαίωσης. Αν αυτό συμβεί τότε σημαίνει πως η συναλλαγή αυτή δεν μπορεί να ολοκληρωθεί καθώς θα οδηγήσει σε σφάλμα Χαμένων Ενημερώσεων και συνεπώς ακυρώνεται (abort) και επιχειρείται η εκτέλεση της εκ νέου από την αρχή. Αν αντίθετα δεν εντοπισθεί κάποια τέτοια σύγκρουση (write-write conflict) η συναλλαγή ολοκληρώνεται κανονικά και οι όποιες ενημερώσεις έχουν συμβεί από τη συναλλαγή αυτή περνιούνται μόνιμα πλέον στο σύστημα βάσης δεδομένων. Προκειμένου να μπορεί να πραγματοποιηθεί ένας τέτοιος έλεγχος και να εντοπισθούν οι συγκρούσεις μεταξύ ταυτόχρονων συναλλαγών, κρατούνται οι ενημερώσεις που πραγματοποιούνται από κάθε συναλλαγή και το στιγμιότυπο στο οποίο πραγματοποιήθηκαν. Έτσι είναι εύκολο αρχικά να βρούμε για μία συναλλαγή που βρίσκεται στο στάδιο της επιβεβαίωσης με ποιες από τις συναλλαγές που έχουν ολοκληρωθεί ήταν ταυτόχρονη και στη συνέχεια αν υπάρχει σύγκρουση τύπου write-write.

Βασιζόμενοι στην παραπάνω ανάλυση προκύπτει εύλογα το συμπέρασμα πως η στρατηγική της Απομόνωσης Στιγμιότυπου αποτρέπει τα σφάλματα που αποτρέπει και το Σειριοποιήσιμο επίπεδο απομόνωσης και άρα είναι ισοδύναμα, με τη μόνη διαφοροποίηση πως η Απομόνωση Στιγμιότυπου δεν βασίζεται σε χρήση κλειδωμάτων επιτυγχάνοντας έτσι σημαντικά καλύτερη απόδοση. Στο σημείο αυτό όμως θα τονίσουμε ένα σφάλμα το οποίο μπορεί να εμφανιστεί στο επίπεδο Απομόνωσης Στιγμιότυπου, αλλά αποτρέπεται στο Σειριοποιήσιμο επίπεδο εξαιτίας της χρήσης κλειδωμάτων. Το φαινόμενο αυτό είναι γνωστό με τον όρο Λοξή Ενημέρωση (write skew) και αφορά το γεγονός πως δύο διαφορετικές ταυτόχρονες συναλλαγές ενημερώνουν δύο διαφορετικές εγγραφές με τέτοιο τρόπο που μετά την επιβεβαίωση και των δύο συναλλαγών παραβιάζεται κάποιος περιορισμός του συστήματος βάσεων δεδομένων. Χαρακτηριστικά,

υποθέτουμε πως έχουμε ένα σύστημα βάσης δεδομένων μιας τράπεζας το οποίο επιτρέπει το ίδιο φυσικό πρόσωπο να έχει περισσότερους από έναν λογαριασμούς, με τον μόνο περιορισμό πως ο ισολογισμός όλων των λογαριασμών κάθε φυσικού προσώπου δεν πρέπει να είναι αρνητικός. Έστω λοιπόν πως 2 διαφορετικοί λογαριασμοί Χ1 και Χ2 ανήκουν στο ίδιο φυσικό πρόσωπο και περιέχουν 100 ευρώ ο καθένας. Μία συναλλαγή Α αφαιρεί από τον λογαριασμό Χ1 το ποσό των 200 ευρώ ενημερώνοντας την τιμή του σε -100, ενώ μία ταυτόχρονη συναλλαγή Β αφαιρεί από το λογαριασμό Χ2 200 ευρώ με αποτέλεσμα να αλλάξει την τιμή του σε -100. Μετά την επιβεβαίωση των δύο συναλλαγών, η οποία ολοκληρώνεται επιτυχώς, το άθροισμα των 2 λογαριασμών είναι αρνητικό, -200, παραβιάζοντας τον περιορισμό του συστήματός μας. Ωστόσο, η λειτουργία της στρατηγικής Απομόνωσης Στιγμιότυπου δεν ανιχνεύει το συγκεκριμένο σφάλμα και συνεπώς η Απομόνωση Στιγμιότυπου δεν είναι ισοδύναμο επίπεδο απομόνωσης με το Σειριοποιήσιμο. Παρέχει όμως σε πολύ μεγάλο βαθμό την ασφάλεια του Σειριοποίησιμο επιπέδου με συγκριτικά καλύτερη απόδοση, ενώ η προσθήκη μιας επιπλέον λειτουργίας στην Απομόνωση Στιγμιότυπου που στο στάδιο της επιβεβαίωσης μιας συναλλαγής θα ελέγχει εάν η συγκεκριμένη συναλλαγή προκαλεί κάποια παραβίαση στους περιορισμούς του συστήματος μας είναι κάτι εφικτό. [4]

Συνοψίζοντας, η Απομόνωση Στιγμιότυπου είναι μια στρατηγική που ικανοποίει σε πολύ μεγάλο βαθμό τις απαιτήσεις των σύγχρονων συστημάτων βάσεων δεδομένων τόσο ως προς την αποφυγή σφαλμάτων όσο και ως προς την απόδοση επιτρέποντας σε μεγάλο βαθμό τον ταυτοχρονισμό των συναλλαγών. Επίσης, όπως αναφέραμε η συγκεκριμένη στρατηγική είναι εύκολα επεκτάσιμη για την προσθήκη επιπλέον λειτουργιών ώστε να ικανοποιεί πλήρως τις απαιτήσεις του συστήματος.

2.4 *Έλεγχος Ταυτοχρονισμού Πολλαπλών Εκδόσεων*

Με τον όρο Έλεγχος Ταυτοχρονισμού Πολλαπλών Εκδόσεων (Multi Version Concurrency Control-MVCC)[27] αναφερόμαστε σε μία στρατηγική ελέγχου των ταυτόχρονων συναλλαγών. Στόχος της συγκεκριμένης έννοιας είναι να παρέχει μία τεχνική ελέγχου ταυτόχρονων συναλλαγών, που δεν θα βασίζεται στα κλειδώματα και συνεπώς δεν θα προκαλεί τα προβλήματα που προέρχονται από τα κλειδώματα, όπως η παρεμπόδιση και η αναμονή των συναλλαγών, στο να συνεχίσουν την εκτέλεση τους, από άλλες ταυτόχρονες συναλλαγές. Όπως αναφέρεται και στο όνομα της μεθόδου, η καινοτομία της συγκεκριμένης μεθόδου είναι ότι βασίζεται σε πολλαπλές εκδόσεις των δεδομένων. Πιο συγκεκριμένα, στη βάση δεδομένων διατηρούνται πολλαπλές διαφορετικές εκδόσεις κάθε αντικειμένου σε αντίθεση με όσα

[4] [6] https://en.wikipedia.org/wiki/Snapshot_isolation, [7]

γνωρίζαμε μέχρι τώρα για τα συστήματα των βάσεων δεδομένων και το γεγονός πως κάθε δεδομένο απεικονίζεται στη βάση με μία μόνο εγγραφή. Κάθε διαφορετική έκδοση μιας εγγραφής αναφέρεται σε μία διαφορετική χρονική στιγμή κατά την οποία πραγματοποιήθηκε ενημέρωση του αντικειμένου. Απλούστερα, η τεχνική του MVCC επιβάλλει πως κάθε φορά που μία συναλλαγή ενημερώνει κάποια εγγραφή, δεν ανανεώνει την τιμή της εγγραφής παρά εισάγει μία νέα εγγραφή στη βάση δεδομένων με τη νέα τιμή και τη χρονική στιγμή στην οποία πραγματοποιήθηκε αυτή η εισαγωγή-ενημέρωση για το δεδομένο. Επίσης, η συναλλαγή που ολοκληρώνεται πρέπει να ενημερώσει το σύστημα βάσης δεδομένων πως η εγγραφή που εισήχθη είναι η πλέον πρόσφατη για το φυσικό δεδομένο στο οποίο αναφέρεται, και πως η προηγούμενη χρονικά εγγραφή είναι πλέον απαρχαιωμένη.

Αμέσως γίνεται κατανοητό πως για κάθε συναλλαγή που εκτελείται είναι απαραίτητο να γνωρίζουμε τη χρονική στιγμή που εκτελείται, αλλά και να γνωρίζουμε για κάθε έκδοση εγγραφής που υπάρχει στη βάση ποια χρονική στιγμή εισήχθη. Προκειμένου να ικανοποιηθεί αυτή η απαίτηση, τα συστήματα βάσεων δεδομένων που υλοποιούν τον Έλεγχο Ταυτοχρονισμού Πολλαπλών Εκδόσεων πρέπει να παρέχουν ένα σύστημα που θα αποδίδει χρονοσφραγίδες. Με τον όρο χρονοσφραγίδα (timestamp), αναφερόμαστε σε μία μοναδική χρονική στιγμή. Έτσι, όταν μια συναλλαγή ξεκινά την εκτέλεση της (start transaction) αποκτά μία χρονοσφραγίδα που αναφέρεται στην χρονική στιγμή εκκίνησης. Ομοίως, κάθε φορά που μία συναλλαγή φτάνει στο στάδιο επιβεβαίωσης αποκτά μία χρονοσφραγίδα επιβεβαίωσης (commit timestamp). Όταν πρόκειται να εισαχθεί μία νέα εγγραφή στη βάση δεδομένων, τότε στην εγγραφή αυτή προστίθεται μία χρονοσφραγίδα που έχει την τιμή της χρονοσφραγίδας επιβεβαίωσης της συναλλαγής που πραγματοποιεί την ενημέρωση.

Βασιζόμενοι σε αυτές τις παραδοχές, είναι πλέον εύκολο να υλοποιηθεί η τεχνική του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων. Ειδικότερα, μόλις μία συναλλαγή θέλει να διαβάσει την τιμή ενός αντικειμένου ψάχνει να βρει την εγγραφή που αφορά το συγκεκριμένο αντικείμενο και έχει την πιο πρόσφατη έγκυρη χρονοσφραγίδα, δηλαδή έχει τη μεγαλύτερη χρονοσφραγίδα μεταξύ των υπόλοιπων εγγραφών του αντικειμένου η οποία έχει μικρότερη τιμή από την χρονοσφραγίδα έναρξης της συναλλαγής. Όλες οι εγγραφές με χρονοσφραγίδα μεγαλύτερη από την χρονοσφραγίδα έναρξης της συναλλαγής δεν είναι ορθό να ληφθούν υπόψιν καθώς επιβεβαιώθηκαν (commit) αφότου η συναλλαγή μας είχε ξεκινήσει, πράγμα που καταδεικνύει πως είναι ενημερώσεις που έγιναν από συναλλαγές ταυτόχρονες με τη συναλλαγή που μας ενδιαφέρει. Αντίστοιχα, οι εγγραφές με χρονοσφραφίδα μικρότερη από την μέγιστη έγκυρη χρονοσφραγίδα δεν είναι ορθό να ληφθούν υπόψιν καθώς αποτελούν απαρχαιωμένες ενημερώσεις. Όπως αναφέραμε κατά την ανάλυση της έννοιας Απομόνωση Στιγμιότυπου, όταν μία συναλλαγή επιθυμεί να πραγματοποιήσει ενημερώσεις εγγραφών, τις πραγματοποιεί στο

ιδιωτικό στιγμιότυπο και στο στάδιο της επιβεβαίωσης, εφόσον είναι έγκυρες, πραγματοποιούνται και στη βάση δεδομένων. Συχνά, αυτό που συμβαίνει είναι ότι οι ενημερώσεις των συναλλαγών πραγματοποιούνται στη μνήμη και κατά το στάδιο της επιβεβαίωσης πραγματοποιούνται και στη βάση δεδομένων. Επομένως, κατά το στάδιο της επιβεβαίωσης και προκειμένου να εντοπιστούν οι write-write συγκρούσεις για το Snapshot Isolation, αρκεί να εντοπιστεί για τις εγγραφές στις οποίες πραγματοποιούνται ενημερώσεις αν υπάρχουν εγγραφές με χρονοσφαγίδα μεγαλύτερη από τη χρονοσφραγίδα έναρξης της συναλλαγής. [5]

Συμπερασματικά, ο Έλεγχος Ταυτοχρονισμού Πολλαπλών Εκδόσεων είναι μία εξαιρετική τεχνική ελέγχου των ταυτόχρονων συναλλαγών που προσφέρει πολύ σημαντική βελτίωση στην απόδοση των συστημάτων βάσεων δεδομένων αποφεύγοντας τη χρήση κλειδωμάτων και παρέχει όλα όσα χρειάζεται ένα σύστημα βάσεων δεδομένων για να υλοποιήσει την τεχνική Απομόνωσης Στιγμιότυπου. Το βασικό μειονέκτημα είναι ότι απαιτείται η αποθήκευση πολύ περισσότερων δεδομένων (εγγραφών) συγκριτικά με τις υπόλοιπες στρατηγικές. Ωστόσο, το συγκεκριμένο γεγονός δεν αποτελεί τόσο έντονη τροχοπέδη καθώς με τη χρήση ενός προγράμματος ρακοσυλλέκτη (garbage collector) βελτιώνεται σημαντικά και επιπλέον ο χώρος αποθήκευσης των δεδομένων είναι λιγότερο σημαντικός παράγοντας συγκριτικά με την ταχύτητα και την απόδοση του συστήματος. Τέλος αξίζει να αναφέρουμε πως πέρα από την υλοποίηση που περιγράψαμε, πολλά συστήματα βάσεων δεδομένων, όπως της Oracle, ακολουθούν διαφορετική αντιμετώπιση διατηρώντας μόνο μία έκδοση κάθε εγγραφής αλλά και ένα ιστορικό των ενημερώσεων που έχουν πραγματοποιηθεί (Write Ahead Log). Έτσι η πρόσβαση σε παλαιότερες εκδόσεις γίνεται μέσω της ανακατασκευής τους από το ιστορικό. Τα μεγαλύτερα και σημαντικότερα συστήματα βάσεων δεδομένων παρέχουν την στρατηγική του Ελέγχου Ταυτοχρονισμού Πολλαπλών εκδόσεων. Ορισμένα από αυτά είναι οι HBase, MariaDB, Microsoft SQL Server, MongoDB, MySQL, Oracle Database, PostgresSQL και SAP HANA, αλλά και πολλά συστήματα ελέγχου εκδόσεων (Version Control Systems), όπως τα Subversion και GIT.

2.5 *Συστήματα Βάσεων Δεδομένων MongoDB & CouchDB*

Όπως αναφέρθηκε και στην εισαγωγή, η υλοποίηση της συγκεκριμένης διπλωματικής εργασίας πραγματοποιήθηκε σε δύο NoSQL βάσεις δεδομένων, τις MongoDB και CouchDB. Οι δύο

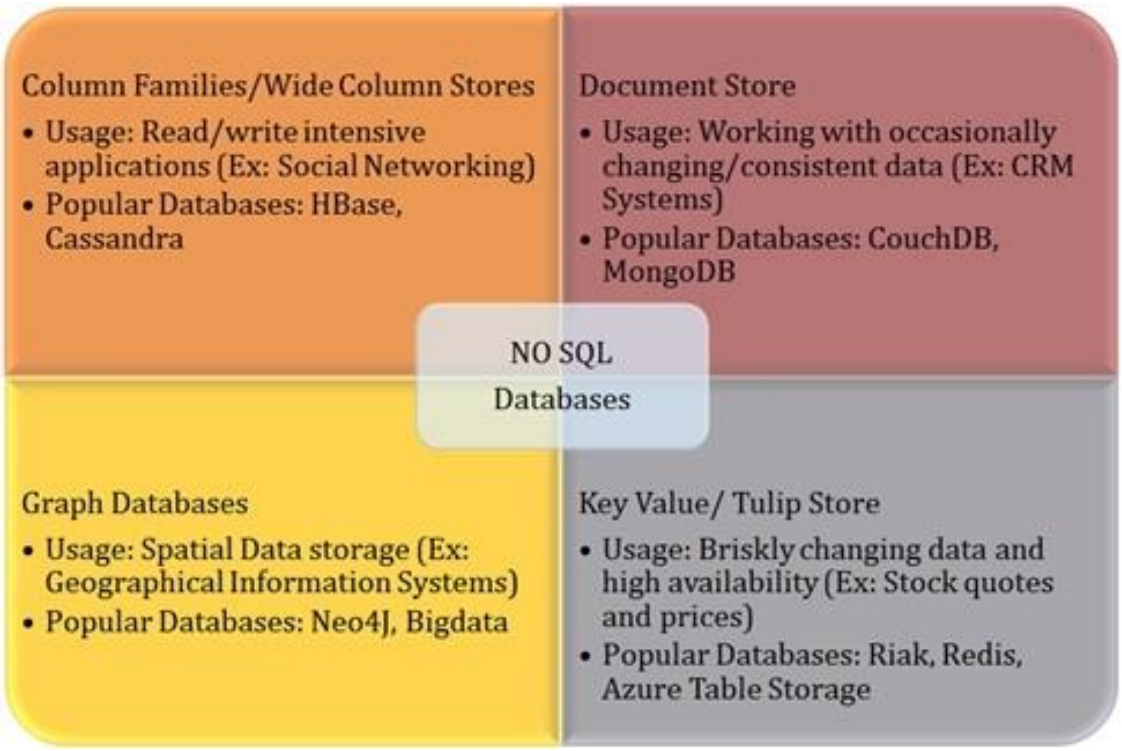
[5] [27]https://en.wikipedia.org/wiki/Multiversion_concurrency_control, [5], [19], [21]

αυτές βάσεις δεδομένων ανήκουν στην ίδια οικογένεια NoSQL βάσεων δεδομένων, τις εγγραφο-κεντρικές (document oriented) βάσεις δεδομένων.

2.5.1 NoSQL βάσεις δεδομένων

Ο όρος NoSQL χρησιμοποιείται για να τονίσει τη διαφοροποίηση που υπάρχει μεταξύ αυτών των βάσεων δεδομένων και των παραδοσιακών σχεσιακών SQL βάσεων δεδομένων. Η εξέλιξη της τεχνολογίας τα τελευταία χρόνια και η άνθιση του cloud computing έχει οδηγήσει στην ύπαρξη τεράστιου όγκου δεδομένων, των λεγόμενων Big Data. Δημιουργούνται έτσι οι ανάγκες για αποδοτική αποθήκευση και ταχύτατη επεξεργασία αυτού του τεράστιου όγκου δεδομένων. Για να ικανοποιηθούν αυτές οι ανάγκες θα πρέπει τα δεδομένα μας να είναι κατανεμημένα. Ωστόσο, οι σχεσιακές βάσεις δεδομένων δεν δημιουργήθηκαν για κατανεμημένα δεδομένα και ούτε προσφέρουν την αναμενόμενη απόδοση. Έτσι οδηγηθήκαμε στη δημιουργία των NoSQL βάσεων δεδομένων. Η ειδοποιός διαφορά μεταξύ των NoSQL και των SQL βάσεων δεδομένων εντοπίζεται στο γεγονός πως οι SQL βάσεις δεδομένων επιβάλλουν τη χρήση ενός αυστηρού σχεσιακού σχήματος που θα προσδιορίζει τη βάση και δεν θα μπορεί να μεταβληθεί, περιορίζοντας έτσι τα δεδομένα μας να έχουν ακριβώς την ίδια μορφή. Εν αντιθέσει, οι NoSQL βάσεις δεδομένων δεν απαιτούν την ύπαρξη σταθερού σχεσιακού μοντέλου, αλλά επιτρέπουν τα δεδομένα μας να έχουν διαφορετική δόμηση μεταξύ τους, επιτρέποντας την αποθήκευση από-κανονικοποιημένων δεδομένων. Έτσι, μπορούν και προσφέρουν μεγαλύτερη ευελιξία και απόδοση, ενώ σε συνδυασμό με το γεγονός ότι είναι σχεδιασμένες για οριζόντια επεκτασιμότητα (horizontal scaling) τις καθιστά ιδανικές για την ικανοποίηση των αναγκών που δημιουργούνται από το μεγάλο όγκο των δεδομένων.

Οι NoSQL βάσεις δεδομένων χωρίζονται σε ορισμένες κατηγορίες ανάλογα με το μοντέλο των δεδομένων στο οποίο στηρίζονται. Όπως φαίνεται στο ακόλουθο σχήμα οι κυριότερες κατηγορίες είναι οι εξής:



Σχήμα 1.5: Είδη NoSQL βάσεων δεδομένων

Μοντέλο Κλειδιού-Τιμής (Key-Value): Σε αυτή την κατηγορία ανήκουν οι βάσεις δεδομένων όπου τα δεδομένα αποθηκεύονται στη μνήμη με τη μορφή ζεύγους key-value. Προσεγγίζουν το μοντέλο της αντιστοίχισης (mapping) ενώ πρέπει να σημειώσουμε πως η τιμή μπορεί να είναι κάτι πιο πολύπλοκο, όπως μία λίστα τιμών. Αντιπροσωπευτικές βάσεις δεδομένων αυτής της κατηγορίας είναι οι Aerospke, Redis και Riak.

Μοντέλο Αποθήκευσης Κατά Στήλες (Column Family): Οι Column Family NoSQL βάσεις δεδομένων βασίζουν το μοντέλο των δεδομένων τους στην αποθήκευση κατά στήλες, και όχι κατά γραμμές όπως φαίνεται πιο φυσικό. Ουσιαστικά, αποτελούν μία μίξη μεταξύ του κλασσικού μοντέλου κατά γραμμών και του μοντέλου κλειδιού-τιμής καθώς κάθε γραμμή αντιπροσωπεύεται από ένα κλειδί. Παρότι με την πρώτη ματιά δεν είναι εμφανές, οι συγκεκριμένες βάσεις δεδομένων διαφοροποιούνται σημαντικά από τις σχεσιακές βάσεις δεδομένων πινάκων προσφέροντας ορισμένες πολύ ενδιαφέρουσες ιδιότητες. Οι σημαντικότερες ίσως βάσεις δεδομένων αυτής της οικογένειας είναι οι HBase και BigTable.

Μοντέλο Γράφου (Graph): Η συγκεκριμένη κατηγορία NoSQL βάσεων δεδομένων στηρίζεται στη μοντελοποίηση των δεδομένων σε κόμβους και ακμές. Οι κόμβοι αντιπροσωπεύουν χαρακτηριστικά των δεδομένων και οι ακμές τις σχέσεις μεταξύ τους. Χρησιμοποιείται σε περιπτώσεις που εμφανίζονται πολύ έντονες εξαρτήσεις μεταξύ των δεδομένων, όπως σε πολύπλοκα δίκτυα χρηστών, και το συγκεκριμένο μοντέλο ευνοεί σημαντικά την οριζόντια

επεκτασιμότητα της βάσης δεδομένων. Ορισμένες από τις κυριότερες βάσεις δεδομένων αυτής της κατηγορίας είναι οι Neo4J, OrientDB και InfiniteGraph.

Μοντέλο Εγγραγο-κεντρικών Δεδομένων (Document oriented): Στη συγκεκριμένη οικογένεια NoSQL βάσεων δεδομένων ανήκουν οι βάσεις δεδομένων που αποθηκεύουν τα δεδομένα τους σε εγγραφές που αποτελούνται από ζεύγη κλειδιών-τιμής. Ουσιαστικά αποτελούν υποσύνολο των Key-Value NoSQL βάσεων δεδομένων. Κάθε εγγραφή συγκροτείται από αποκανονικοποιημένα δεδομένα που χαρακτηρίζουν ένα αντικείμενο. Κάθε πεδίο (value) μιας εγγραφής μπορεί να είναι κάτι πιο πολύπλοκο από μια απλή τιμή, ακόμη και μια ολόκληρη εγγραφή. Οι εγγραφές μπορεί να διατηρούν διαφορετικά ζεύγη κλειδιού-τιμής και η χρήση αυτής της κατηγορίας είναι ιδιαίτερα σημαντική σε εφαρμογές που αναπτύσσονται και στηρίζονται σε αντικειμενοστραφείς γλώσσες προγραμματισμού, καθώς κάθε εγγραφή μπορεί να αντιπροσωπεύει την αποθήκευση ενός αντικειμένου. Αξίζει να σημειώσουμε πως στις περισσότερες Document Oriented NoSQL βάσεις δεδομένων οι εγγραφές είναι σε μορφή JSON (Javascript Object Notation) ή παραλλαγές αυτού του τύπου. Μερικές από τις σημαντικότερες και πιο δημοφιλείς βάσεις δεδομένων της συγκεκριμένης κατηγορίας είναι οι MongoDB, CouchDB και DocumentDB.

Τέλος, είναι σημαντικό να αναφέρουμε πως λόγω της ευελιξίας που προσφέρουν αυτές οι βάσεις δεδομένων έχουν αναπτυχθεί ορισμένα συστήματα βάσεων δεδομένων που υλοποιούν υβριδικά μοντέλα δανειζόμενα στοιχεία από περισσότερες από μία κατηγορίες. Χαρακτηριστικά παραδείγματα αποτελούν οι Cassandra, η οποία συνδυάζει το μοντέλο κλειδιού-τιμής με το μοντέλου αποθήκευσης σε στήλες, και η MarkLogic, που βασίζεται τόσο στο εγγραφο-κεντρικό μοντέλο όσο και στο μοντέλου γράφου. [6]

2.5.2 MongoDB

2.5.2.1 Γενικά Στοιχεία

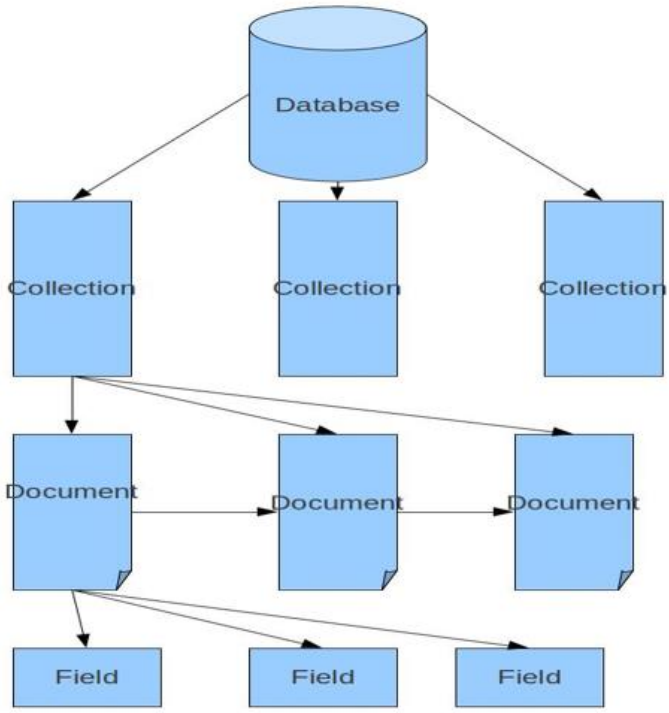
Η MongoDB είναι μία NoSQL βάση δεδομένων που ξεκίνησε να αναπτύσσεται το 2007 από την εταιρεία 10gen, η οποία στην πορεία της εξέλιξης της μετονομάστηκε σε MongoDB Inc. Για την υλοποίηση της χρησιμοποιήθηκε η αντικειμενοστραφής γλώσσα C++ και κυκλοφόρησε για πρώτη φορά το 2009. Είναι μία βάση δεδομένων ανοιχτού κώδικα (open source) και ανήκει στην οικογένεια των εγγραφο-κεντρικών (document oriented) NoSQL βάσεων δεδομένων. Το όνομα της προέρχεται από την αγγλική λέξη humongous που σημαίνει τεράστιο και η επιλογή αυτή μόνο τυχαία δεν είναι, καθώς η συγκεκριμένη βάση δεδομένων

[6] [8] https://en.wikipedia.org/wiki/NoSQL

στοχεύει στην αποδοτική και γρήγορη αποθήκευση και επεξεργασία μεγάλου όγκου πολύπλοκων δεδομένων.

Στόχος της ομάδας ανάπτυξης της MongoDB ήταν να δημιουργήσουν μία βάση δεδομένων αρκετά απλή ως προς τη δομή και τη χρήση, που όμως θα ανταποκρίνεται σε ορισμένες πολύ σημαντικές ανάγκες και θα ταιριάζει απόλυτα σε συγκεκριμένες εφαρμογές. Αξίζει να τονίσουμε πως η MongoDB παρέχει οδηγούς (drivers) σε πολλές διαφορετικές γλώσσες και συγκεκριμένα στις C, C++, C# (.NET), Java, Node.js, Perl, PHP, Python, Ruby και Scala, ενώ παρέχεται και shell σε javascript.

Η δομή της MongoDB μοιάζει να είναι αρκετά παρόμοια με τη δομή μιας σχεσιακής βάσης δεδομένων. Πιο συγκεκριμένα, το σύστημα βάσεων δεδομένων της MongoDB υποστηρίζει την ύπαρξη πολλών διαφορετικών βάσεων δεδομένων. Κάθε βάση δεδομένων περιέχει πολλές διαφορετικές συλλογές (collections). Η έννοια της συλλογής θα μπορούσε να παραλληλιστεί με την έννοια των πινάκων στις σχεσιακές βάσεις δεδομένων. Προχωρώντας βαθύτερα στην ιεραρχία της δομής της MongoDB, οι συλλογές περιέχουν έγγραφα (documents). Κάθε έγγραφο θα μπορούσαμε να πούμε πως ανταποκρίνεται στο ρόλων των γραμμών των σχεσιακών βάσεων δεδομένων. Τέλος, μέσα στα έγγραφα έχουμε ζεύγη κλειδιού-τιμής. Κατά συνέπεια, βλέπουμε πως τα δεδομένα μας αποθηκεύονται σε έγγραφα μέσα σε συλλογές και πολλές διαφορετικές συλλογές συγκροτούν μια βάση δεδομένων.



Σχήμα 1.6: Δομή Βάσεων της MongoDB

Όπως έχουμε αναφέρει, η MongoDB όντας μία NoSQL βάση δεδομένων έχει δυναμικό και όχι σταθερό σχήμα. Αυτό στην πράξη αποδεικνύεται από το γεγονός ότι έγγραφα που ανήκουν στην ίδια συλλογή μπορούν να έχουν εντελώς διαφορετική σύσταση και συγκεκριμένα να περιέχουν διαφορετικά ζεύγη κλειδιού-τιμής. Ασφαλώς, στην πράξη αυτό που συμβαίνει είναι πως μία συλλογή αποθηκεύει έγγραφα του παρόμοιων τύπων, ωστόσο παρέχεται η ελευθερία τα έγγραφα αυτά να περιέχουν οποιαδήποτε δεδομένα χωρίς κανένα περιορισμό.

Αναφορικά με τη δομή των εγγραφών έχουμε αναφέρει πως οι περισσότερες εγγραφο-κεντρικές NoSQL βάσεις δεδομένων αποθηκεύουν τις εγγραφές σε μορφή JSON ή κάποια παραλλαγή. Στη MongoDB οι εγγραφές αποθηκεύονται σε μορφή BSON (Binary JSON), που αποτελεί τη δυαδική αναπαράσταση της μορφής JSON. Η BSON μορφή των εγγραφών στη βάση δεδομένων επιτρέπει την ταχεία διάσχιση και ανάλυση τους. Ωστόσο, οι εγγραφές εμφανίζονται σε μορφή JSON στον χρήστη, καθώς είναι πολύ φιλική και εύκολη στην ανάγνωση. Όταν ένας χρήστης δημιουργεί ένα νέο έγγραφο, το δημιουργεί σε μορφή JSON και στη συνέχεια το σύστημα της MongoDB το μετατρέπει σε BSON προκειμένου να το αποθηκεύσει. Η JSON μορφή εγγραφών επιβάλλει τα δεδομένα να είναι σε ζεύγη κλειδιού-τιμής, όπου το κλειδί είναι πάντα ένα αλφαριθμητικό και η τιμή μπορεί να είναι οποιοσδήποτε από τους επιτρεπόμενους τύπους (String, Boolean κ.α.). Επίσης, ως κλειδί μπορεί να είναι ένας πιο σύνθετος τύπος, όπως ένας πίνακας ή μία λίστα, ή ακόμη και μία ολόκληρη εγγραφή (embedded document). Το μέγιστο μέγεθος ενός εγγράφου μπορεί να είναι 16MB. Τέλος, πρέπει να αναφερθούμε το μόνο και κύριο περιορισμό που επιβάλλει η MongoDB στα έγγραφα και είναι η παρουσία ενός πεδίου με κλειδί το όνομα _id και τιμή τύπου ObjectID. Το _id είναι μοναδικό για κάθε εγγραφή και ουσιαστικά παίζει το ρόλο του primary key διαχωρίζοντας τα δεδομένα μεταξύ τους ανεξάρτητα από τα πεδία που περιέχουν. Το _id παρέχεται από το σύστημα της MongoDB ή μπορεί να το εισάγει ο προγραμματιστής-χρήστης. Στο ακόλουθο σχήμα παρουσιάζεται ένα αντιπροσωπευτικό έγγραφο της MongoDB.

Σχήμα 1.7: Μορφή Εγγραφής MongoDB

Παρατηρούμε λοιπόν πως η μορφή των εγγραφών είναι φυσική ως προς την απεικόνιση των δεδομένων και ταυτόχρονα αποτελεί μία φυσική αναπαράσταση των κλάσεων των αντικειμενοστραφών γλωσσών προγραμματισμού. Αυτή η σχεδιαστική απόφαση είναι εξαιρετικά σημαντική και προσφέρει πολλές δυνατότητες.

2.5.2.2 Λειτουργίες CRUD

Ο όρος CRUD είναι ακρωνύμιο που σχηματίζεται από τα αρχικά γράμματα των λειτουργιών Δημιουργίας (Create), Ανάγνωσης (Read), Ενημέρωσης (Update) και Διαγραφής (Delete) εγγραφών. Ουσιαστικά αναφερόμαστε στις στοιχειώδεις λειτουργίες που πρέπει να παρέχει ένα σύστημα βάσεων δεδομένων και στο σημείο αυτό θα αναλύσουμε πως υλοποιεί τις λειτουργίες αυτές η MongoDB.

- Δημιουργία (Create): Η λειτουργία αυτή αφορά ουσιαστικά τη δημιουργία μιας νέας εγγραφής σε μία συλλογή της βάσης δεδομένων από το χρήστη, δηλαδή στην εισαγωγή (insert) όπως αναφέρεται στην SQL. Παρέχεται η δυνατότητα εισαγωγής περισσότερων εγγραφών στην ίδια συλλογή με μία μόνο εντολή. Η εισαγωγή εγγραφής είναι ατομική λειτουργία σε επίπεδο εγγραφής, πράγμα που σημαίνει πως η εισαγωγή περίσσοτερων εγγραφών δεν αποτελεί ατομική λειτουργία στο σύνολο της.
- Ανάγνωση (Read): Η λειτουργία της ανάγνωσης αφορά τη δυνατότητα του χρήστη να υποβάλλει ερωτήματα σε μία συλλογή της βάσης δεδομένων και το σύστημα να επιστρέφει τις εγγραφές που ικανοποιούν το συγκεκριμένο ερώτημα. Στη MongoDB τα ερωτήματα υποβάλλονται σε μορφή JSON και υποστηρίζεται μία πληθώρα τελεστών προσφέροντας πολύ μεγάλη ευελιξία στο χρήστη-προγραμματιστή ως προς τα ερωτήματα και την

πολυπλοκότητα τους. Αξίζει να σημειωθεί πως τα κριτήρια ενός ερωτήματος μπορεί να αφορούν πεδία που βρίσκονται σε εγγραφές ενσωματωμένες μέσα σε άλλες εγγραφές. Η λειτουργία αυτή συχνά αναφέρεται και ως εύρεση (find).

- Ενημέρωση (Update): Η ενημέρωση είναι μία λειτουργία που δίνει τη δυνατότητα να ενημερώσει και να αλλάξει τα πεδία κάποιων εγγραφών μιας συλλογής που ήδη υπάρχουν. Η MongoDB υλοποιεί τη συγκεκριμένη λειτουργία συνδυάζοντας τις προηγούμενες δύο λειτουργίες. Ο χρήστης παρέχει σε μορφή JSON τα κριτήρια με τα οποία θα επιλεχθούν οι εγγραφές που θα ενημερωθούν και συνεπώς εκτελείται μία λειτουργιά find. Επίσης, ο χρήστης παρέχει σε JSON μορφή, χρησιμοποιώντας τους κατάλληλους τελεστές, τις πληροφορίες σχετικά με το ποια πεδία θα αλλάξουν και ποιες τιμές θα λάβουν. Υποστηρίζεται η δυνατότητα εισαγωγής ενός πεδίου αν δεν υπάρχει στην εγγραφή και η δυνατότητα αντικατάστασης μιας ολόκληρης εγγραφής με μία νέα. Ως λειτουργία εγγραφής είναι ατομική σε επίπεδο μιας εγγραφής.
- Διαγραφή (Delete): Όπως είναι προφανές η συγκεκριμένη λειτουργία αναφέρεται στην διαγραφή μιας ή περισσοτέρων εγγραφών από μία συλλογή. Πάλι, η εύρεση των εγγραφών στηρίζεται στη λειτουργία find καθώς ο χρήστης πρέπει να παρέχει σε μορφή JSON τα κριτήρια που πρέπει να πληρεί μία εγγραφή της συλλογής προκειμένου να διαγραφεί. Ομοίως με τις προηγούμενες λειτουργίες εγγραφής, είναι ατομική λειτουργία ως προς επίπεδο μιας εγγραφής.

2.5.2.3 Σημαντικές Επιπλέον Λειτουργίες

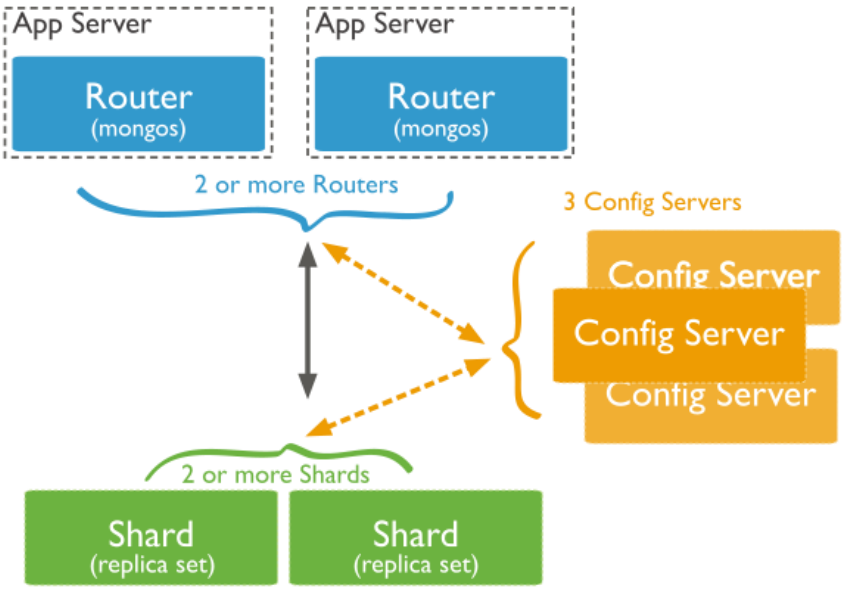
Όπως έχει αναφερθεί, η MongoDB είναι μία βάση δεδομένων που απευθύνεται στην αποθήκευση και τη διαχείριση μεγάλου όγκου δεδομένων. Προκειμένου να ανταποκριθεί στις απαιτήσεις που δημιουργούνται προσφέρει κάποιες πολύ σημαντικές λειτουργίες και δυνατότητες προκειμένου να αυξηθεί η απόδοση. Αυτές οι λειτουργίες, που βελτιώνουν σημαντικά την απόδοση, σε συνδυασμό με την απλότητα που προσφέρεται στον προγραμματιστή καθιστούν την MongoDB μία από τις πιο δημοφιλείς και ευρέως χρησιμοποιούμενες σύγχρονες βάσεις δεδομένων. Στο σημείο αυτό θα αναλύσουμε ορισμένες από τις σημαντικότερες ειδικές λειτουργίες που προσφέρονται από τη MongoDB

- **Ευρετήρια:** Τα ευρετήρια (indexes) είναι μια ευρέως διαδεδομένη λειτουργία στις παραδοσιακές σχεσιακές βάσεις δεδομένων. Πρόκειται για δομές δεδομένων που χρησιμοποιούνται για να αποθηκεύσουν πληροφορίες που χρησιμοποιούνται στη βέλτιστη εξυπηρέτηση ερωτημάτων. Στη MongoDB τα ευρετήρια υλοποιούνται ως B-δένδρα (B-trees) και υπάρχουν 4 διαφορετικοί τύποι ευρετηρίων που υποστηρίζονται: ευρετήρια στο πεδίο _id (δημιουργείται αυτόματα από τη MongoDB), ευρετήριο σε απλό πεδίο, σύνθετα ευρετήρια και ευρετήρια πολλαπλών τιμών. Τα ευρετήρια απλού πεδίου είναι μία δομή B-

δένδρου που αναφέρεται σε ένα συγκεκριμένο πεδίο των εγγραφών και βάση αυτής της δομής εξυπηρετούνται ταχύτατα τα ερωτήματα που αφορούν αυτό το πεδίο. Υπάρχει φυσικά η δυνατότητα να οριστεί αν η ταξινόμηση που θα ακολουθήσει η δομή μας θα είναι αύξουσα ή φθίνουσα. Τα σύνθετα ευρετήρια αποτελούν ευρετήρια που αφορούν περισσότερα από ένα πεδία. Με βάση αυτά τα ευρετήρια εξυπηρετούνται όλα τα ερωτήματα που περιέχουν κάποιο πρόθεμα του ευρετηρίου. Έτσι αν έχουμε ένα ευρετήριο όπως το { first_name : 1, last_name : 1 address : -1 }, όπου τα 1 και -1 αναφέρονται σε αύξουσα ή φθίνουσα ταξινόμηση αντίστοιχα, μπορούμε να εξυπηρετήσουμε ερωτήματα που αναφέρονται μόνο στο first_name ή στο first_name και στο last_name ή και στα τρία πεδία. Επίσης πρέπει να σημειωθεί πως για τα σύνθετα ευρετήρια η ταξινόμηση που επιθυμούμε να επιστρέψει το ερώτημα πρέπει να αντιστοιχεί στην ταξινόμηση του ευρετηρίου. Τέλος, τα ευρετήρια πολλαπλών τιμών είναι ευρετήρια που αναφέρονται σε πεδία που η τιμή τους είναι ένας πίνακας τιμών. Πρέπει να τονιστεί πως σε περίπτωση χρήσης ευρετηρίων πρέπει όλες οι εγγραφές να διαθέτουν το πεδίο στο οποίο βασίζεται το ευρετήριο. Τέλος, πρέπει να αναφέρουμε πως εφόσον στόχος των ευρετηρίων είναι η βέλτιστη εξυπηρέτηση των ερωτημάτων, ένα σύστημα βάσεων δεδομένων MongoDB εκτελεί ανά τακτά χρονικά διαστήματα δοκιμές ερωτημάτων προκειμένου να γνωρίζει ποια ευρετήρια εξυπηρετούν βέλτιστα τα πιθανά ερωτήματα και χρησιμοποιεί την πληροφορία για να επιλέξει τη βέλτιστη επιλογή εξυπηρέτησης ενός ερωτήματος.

- **Κατανομή Δεδομένων:** Είναι ευρέως αποδεκτό πως το πρόβλημα της διαχείρισης του μεγάλου όγκου δεδομένων της σύγχρονης εποχής δεν μπορεί να αντιμετωπιστεί με κατακόρυφη κλιμάκωση ή αλλιώς vertical scalability, δηλαδή με προσθήκη επιπλέον επεξεργαστικής ισχύς και μνήμης στο ίδιο φυσικό μηχάνημα της βάσης δεδομένων, καθώς αυξάνεται δυσανάλογα πολύ η πολυπλοκότητα και το κόστος. Συνεπώς, η λύση είναι η οριζόντια κλιμάκωση (horizontal scalability), δηλαδή η προσθήκη επιπλέον φυσικών μηχανημάτων (data servers) στη βάση δεδομένων και ο διαμοιρασμός των δεδομένων στα διάφορα αυτά μηχανήματα. Η κατανομή δεδομένων ονομάζεται sharding και οι διάφοροι κόμβοι της βάσης δεδομένων που περιέχουν τα διαμοιρασμένα δεδομένα ονομάζονται shard nodes. Σκοπός του sharding είναι κάθε κόμβος να διατηρεί λιγότερα δεδομένα προκειμένου να είναι ευκολότερη η αποθήκευση των δεδομένων και η εξυπηρέτηση των ερωτημάτων. Η κατανομή των δεδομένων γίνεται σε επίπεδο συλλογής, πράγμα που σημαίνει πως διαφορετικά έγγραφα της ίδια συλλογής μπορούν να βρίσκονται σε διαφορετικούς κόμβους ενώ ένα έγγραφο δεν μπορεί να διαιρεθεί και να κατανεμηθεί σε διαφορετικούς κόμβους. Η απόφαση σχετικά με το σε ποιον κόμβο θα κατανεμηθεί ένα έγγραφο προκύπτει βάση ενός shard key, δηλαδή ενός πεδίου του εγγράφου για το οποίο πρέπει να έχει δημιουργηθεί ευρετήριο, και κάθε κόμβος είναι υπεύθυνος για διαφορετικό shard key. Για τη βέλτιστη κατανομή των δεδομένων χρησιμοποιούνται διάφορες τεχνικές

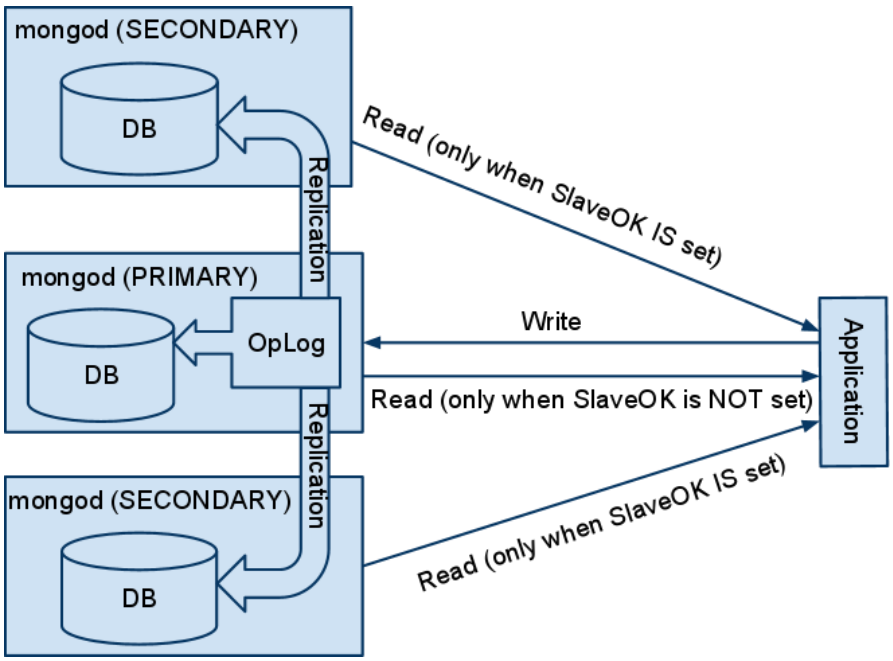
για το shard key, όπως η διαίρεση με βάση το εύρος τιμών και η διαίρεση με βάση κάποια συνάρτηση κατακερματισμού. Προκειμένου να υποστηρίξει τη δυνατότητα κατανομής δεδομένων η MongoDB βασίζεται σε μία πολύ συγκεκριμένη αρχιτεκτονική. Πιο συγκεκριμένα, διατηρεί configuration servers οι οποίοι είναι υπεύθυνοι για τη συγκέντρωση πληροφοριών (metadata) σχετικά με τους διάφορους shard nodes και ποια δεδομένα διατηρούν. Επίσης, στην αρχιτεκτονική περιλαμβάνονται διάφοροι query routers που είναι υπεύθυνοι για την εξυπηρέτηση ερωτημάτων και την δρομολόγηση τους στους κατάλληλους κόμβους που περιέχουν τα απαραίτητα δεδομένα. Στο ακόλουθο σχήμα παρουσιάζεται η βασική αρχιτεκτονική που ακολουθεί η MongoDB και την οποία περιγράψαμε. Παρατηρούμε ότι σε κάθε shard αναφέρεται η φράση Replica Set και αυτή η έννοια θα αναλυθεί ευθύς αμέσως.



Σχήμα 1.8: Δομή συστήματος MongoDB

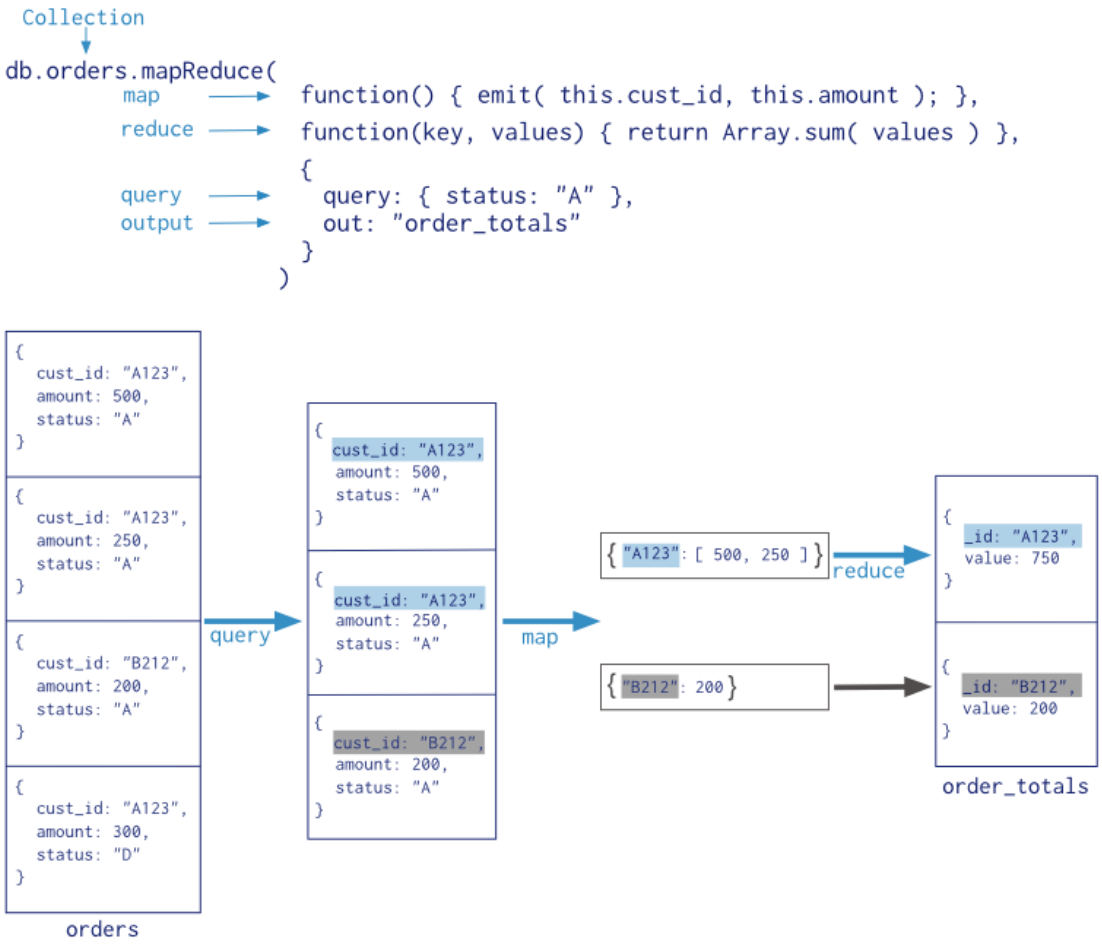
- **Αντίγραφα Δεδομένων:** Μία βασική προϋπόθεση για την ορθή λειτουργία ενός συστήματος βάσεων δεδομένων είναι η διασφάλιση πως ελαχιστοποιείται το ενδεχόμενο να χαθούν δεδομένα λόγω βλάβης του υλικού. Για την ικανοποίηση της προϋπόθεσης αυτής η MongoDB βασίζεται στην έννοια του replication. Πιο συγκεκριμένα, κρατούνται αντίγραφα των δεδομένων σε περισσότερα από 1 sets, πράγμα που σημαίνει πως εάν ένας κόμβος υποστεί βλάβη τότε τα δεδομένα του είναι διασφαλισμένα καθώς βρίσκονται αποθηκευμένα και σε άλλον κόμβο. Ασφαλώς, όσο μεγαλύτερος είναι ο αριθμός των κόμβων στον οποίο κρατούνται αντίγραφα (replication factor) τόσο μεγαλύτερη ασφάλεια υπάρχει καθώς μειώνεται η πιθανότητα να υποστούν ταυτόχρονα βλάβη οι κόμβοι που κρατούν κοινά δεδομένα και συνεπώς αυτά να χαθούν. Μία σημαντική τεχνική για την αποφυγή αυτής της πιθανότητας είναι τουλάχιστον ένας από τους κόμβους που κρατά

αντίγραφα να βρίσκεται σε εντελώς διαφορετικό data center προκειμένου να μην επηρεαστεί από γενική βλάβη στο data center (π.χ. πυρκαγιά). Για την υλοποίηση της έννοιας των αντιγράφων δεδομένων δημιουργούνται ομάδες κόμβων που κρατούν τα ίδια δεδομένα και ονομάζονται replica sets. Κάθε replica set απαρτίζεται από έναν κύριο κόμβο (primary node), ο οποίος διατηρεί τα πιο πρόσφατα δεδομένα, και κάποιους δευτερεύοντες κόμβους (secondary nodes), οι οποίοι ενημερώνονται για τα δεδομένα από τον κύριο κόμβο. Έτσι, όταν πρόκειται να πραγματοποιηθεί μία ενημέρωση σε κάποια εγγραφή αυτή γίνεται στον κύριο κόμβο και στη συνέχεια κάποια στιγμή θα ενημερωθούν οι δευτερεύοντες κόμβοι. Συμπεραίνουμε λοιπόν πως υπάρχουν χρονικές στιγμές κατά τις οποίες οι δευτερεύοντες κόμβοι διατηρούν απαρχαιωμένα δεδομένα (stale data). Όσον αφορά λοιπόν την εξυπηρέτηση των αναγνώσεων, υπάρχει η επιλογή να εξυπηρετούνται από οποιονδήποτε κόμβο του replica set παρέχοντας μεγαλύτερη ταχύτητα αλλά και τον κίνδυνο να πραγματοποιηθεί ανάγνωση παλαιότερων δεδομένων, καθώς και η επιλογή να εξυπηρετούνται όλες οι αναγνώσεις από τον πρωτεύοντα κόμβο, γεγονός που διασφαλίζει πως θα διαβαστούν τα πιο πρόσφατα δεδομένα (fresh data) αλλά με σημαντική επιβάρυνση στην απόδοση καθώς όλα τα ερωτήματα εξυπηρετούνται από τον κύριο κόμβο. Η επιλογή σχετικά με την εξυπηρέτηση των αναγνώσεων μπορεί να καθοριστεί από τον προγραμματιστή-χρήστη με τη μεταβλητή SlaveOK. Η έννοια του replication και η αρχιτεκτονική που χρησιμοποιείται από τη MongoDB για την υλοποίηση της παρουσιάζονται στο ακόλουθο σχήμα.



Σχήμα 1.9: Δομή Συστήματος Αντιγραφών

- **Map-Reduce:** Η τεχνική του map-reduce είναι εξαιρετικά σημαντική στον χώρο των Big Data.Η ιδέα στην οποία βασίζεται η τεχνική αυτή είναι πως υπάρχουν πολλοί διαφορετικοί κόμβοι που συμμετέχουν στην εξυπηρέτηση ενός αιτήματος, με κάθε κόμβο να εκτελεί περισσότερες από μία φορές την διαδικασία map-reduce. Η διαδικασία map-reduce έχει δύο στάδια. Στο πρώτο στάδιο η συνάρτηση map δέχεται μία εγγραφή, την επεξεργάζεται και δίνει ως έξοδο ένα ζεύγος κλειδιού τιμής, όπου η τιμή μπορεί να είναι ακόμη και ολόκληρη η εγγραφή. Στη συνέχεια όλες οι έξοδοι των συναρτήσεων map ομαδοποιούνται βάσει κλειδιών σχηματίζοντας ζεύγη κλειδιού με πίνακα τιμών. Ακολούθως, κάθε τέτοιο ζεύγος δίνεται ως είσοδος σε μία συνάρτηση reduce που πραγματοποιεί την τελική επεξεργασία των τιμών και δίνει την κατάλληλη τελική έξοδο. Η διαδικασία φαίνεται στο ακόλουθο σχήμα. Είναι σαφές λοιπόν πόσο σημαντική είναι η τεχνική αυτή καθώς επιτρέπει την παράλληλη επεξεργασία των δεδομένων και ευνοεί την κατανομή τους σε διαφορετικούς κόμβους. Στο ακόλουθο σχήμα παρουσιάζεται η τεχνική map-reduce που υλοποιεί η MongoDB.



Σχήμα 1.10: Διαδικασία Map-Reduce

2.5.2.4 ACID Εγγυήσεις

Ατομικότητα: Η MongoDB παρέχει την ιδιότητα της ατομικότητας σε επίπεδο εγγράφου. Αυτό σημαίνει πως η εισαγωγή ενός εγγράφου στη βάση δεδομένων ή η ενημέρωση κάποιων πεδίων ενός εγγράφου είναι μια ατομική διαδικασία, υπό την έννοια πως είτε θα ολοκληρωθεί επιτυχώς όλη η διαδικασία ή δεν θα πραγματοποιηθεί καμία αλλαγή στη βάση δεδομένων από την διαδικασία αυτή λόγω κάποιας αποτυχίας. Σε περιπτώσεις συναλλαγών που περιλαμβάνουν εισαγωγές και ενημερώσεις περισσότερων από ένα εγγράφων, η MongoDB δεν διασφαλίζει την ατομικότητα της συναλλαγής με συνέπεια αν σε κάποιο σημείο η συναλλαγή διακοπεί να παραμείνουν οι αλλαγές και οι ενημερώσεις που έγιναν σε έγγραφα από τη συναλλαγή αυτή προτού διακοπεί.

Συνέπεια: Ως προς την ιδιότητα της συνέπειας, η MongoDB προσφέρει μεγάλη ευελιξία. Λόγω της ύπαρξης των replica sets (αντιγράφων), η προκαθορισμένη επιλογή του συστήματος της MongoDB είναι όλες οι αναγνώσεις να πραγματοποιούνται από τον πρωτεύοντα κόμβο (primary node). Το γεγονός αυτό σε συνδυασμό με το γεγονός πως όλες οι εγγραφές πραγματοποιούνται πάντοτε στον πρωτεύοντα κόμβο οδηγεί στο συμπέρασμα πως τα δεδομένα που διαβάζονται είναι πάντα τα πιο πρόσφατα και συνεπώς ο χρήστης βλέπει πως η βάση δεδομένων βρίσκεται πάντοτε σε συνεπή κατάσταση. Ωστόσο, η MongoDB δίνει την ευχέρεια στο χρήστη να επιτρέπει τις αναγνώσεις προκειμένου να αυξηθεί η ταχύτητα εξυπηρέτησης των ερωτημάτων θυσιάζοντας την συνέπεια των δεδομένων, καθώς είναι πιθανό η ανάγνωση να γίνει από ένα δευτερεύοντα κόμβο στον οποίο δεν έχουν φτάσει οι πιο πρόσφατες ενημερώσεις. Στο σημείο αυτό πρέπει να αναφερθεί πως είναι στην ευχέρεια του χρήστη να καθορίσει αν μία ενημέρωση (write) θα ολοκληρώνεται όταν ενημερωθεί μόνο ο πρωτεύοντας, όταν ενημερωθεί ο πρωτεύοντας κόμβος και ένας αριθμός δευτερευόντων κόμβων ή όταν ενημερωθούν όλοι οι κόμβοι που συμμετέχουν στο replica set. Είναι σαφές πως στο τρίτο από αυτά τα σενάρια διασφαλίζεται πως η ανάγνωση επιστρέφει πάντα συνεπή δεδομένα.

Απομόνωση: Η εγγύηση της ιδιότητας της απομόνωσης διασφαλίζεται και πάλι σε επίπεδο εγγράφου, με την έννοια πως δεν μπορεί να διαβάσει μία ανάγνωση τα ενημερωμένα πεδία μιας εγγραφής η οποία ενημερώνεται από μία ταυτόχρονη συναλλαγή. Ωστόσο, η MongoDB λειτουργεί με τέτοιο τρόπο που η ενημέρωση μιας εγγραφής γίνεται αρχικά στη μνήμη και εν συνεχεία αποθηκεύεται η εγγραφή στο δίσκο. Επομένως, ελλοχεύει ο κίνδυνος να διαβαστεί μία ενημέρωση που έχει ολοκληρωθεί στη μνήμη αλλά όχι στο δίσκο και στη συνέχεια λόγω

κάποιου σφάλματος η ενημέρωση αυτή να μην φτάσει ποτέ στο δίσκο και άρα να μην βρεθεί ποτέ στη βάση δεδομένων.[7]

Μονιμότητα: Η MongoDB διασφαλίζει τη μονιμότητα των δεδομένων με τη χρήση των replica sets. Πιο συγκεκριμένη, η προεπιλεγμένη επιλογή είναι πως μία ενημέρωση θεωρείται ολοκληρωμένη όταν έχει ενημερωθεί η πλειοψηφία των κόμβων που συμμετέχουν στο replica set διασφαλίζοντας πως η αποτυχία κάποιων κόμβων δεν συνεπάγεται απώλεια των δεδομένων. Πέρα από αυτή την δυνατότητα όμως, η MongoDB παρέχει και μία άλλη εγγύηση μονιμότητας. Συγκεκριμένα παρέχει ένα μηχανισμό journaling ο οποίος στην ουσία καταγράφει σε αρχεία στο δίσκο τις ενέργειες που πρόκειται να γίνουν και έτσι σε περίπτωση οποιασδήποτε διακοπής των ενεργειών, η MongoDB μπορεί να ανατρέξει στα συγκεκριμένα αρχεία και αφού μάθει ποιες ενέργειες επρόκειτο να γίνουν, να τις πραγματοποιήσει εξ' ολοκλήρου από την αρχή.

2.5.3 CouchDB

2.5.3.1 Γενικά Στοιχεία

Η βάση δεδομένων CouchDB, ή Apache CouchDB όπως είναι το πλήρες όνομα της, αποτελεί μία NoSQL βάση δεδομένων που αναπτύσσεται από την Apache Software Foundation. Αποτελεί έργο ανοιχτού κώδικα και η πρώτη της κυκλοφορία έγινε το 2005. Η ανάπτυξη της CouchDB πραγματοποιείται με χρήση της συναρτησιακής γλώσσας προγραμματισμού Erlang. Η CouchDB είναι μία εγγραφο-κεντρική (document oriented) NoSQL βάση δεδομένων και θέτει ως προτεραιότητα την αποδοτική αποθήκευση μεγάλου όγκου δεδομένων και την ταχύτατη επεξεργασία κατανεμημένων δεδομένων.

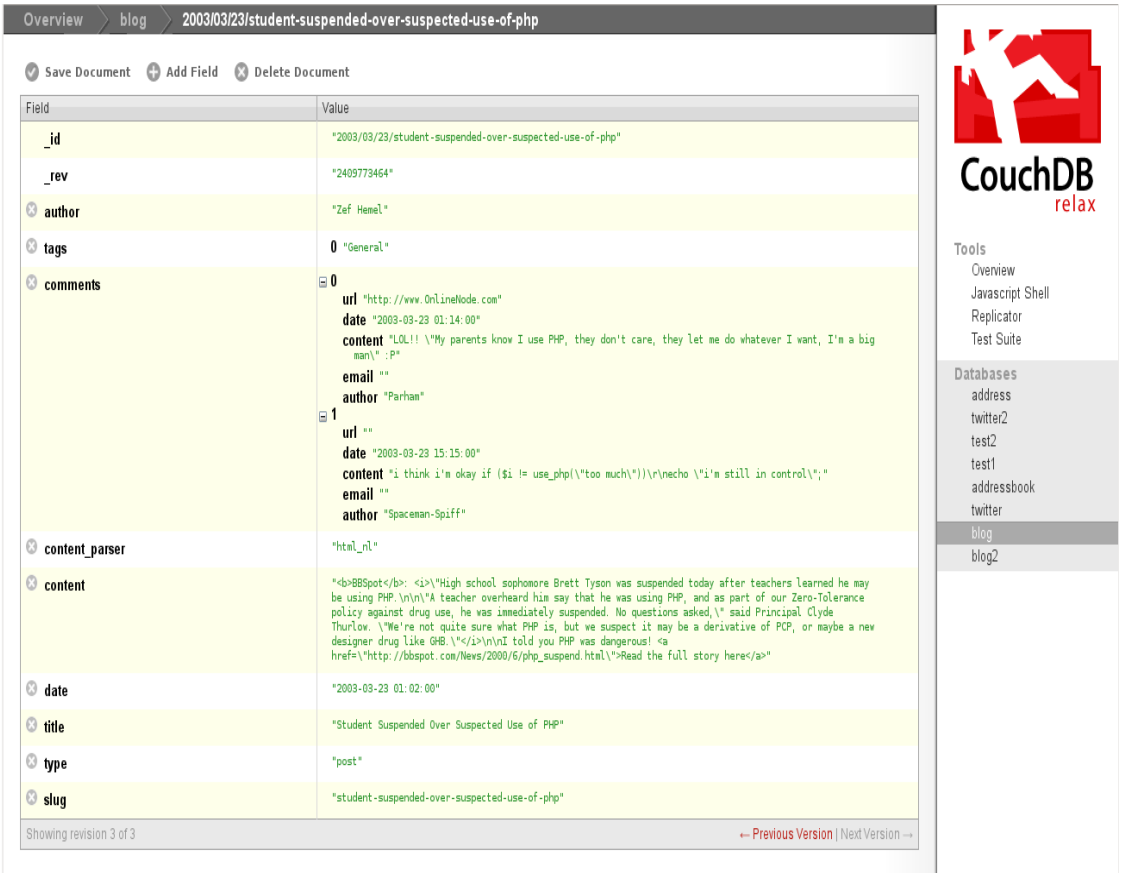
Το όνομα της συγκεκριμένης βάσης δεδομένων είναι ένα ακρωνύμιο των αρχικών της φράσης Cluster Of Unreliable Commodity Hardware, ωστόσο η επιλογή δεν είναι τυχαία και σε συνδυασμό με το λογότυπο της εταιρείας που περιέχει τη λέξη Relax (χαλάρωση) θέλει να καταστήσει σαφές πως η συγκεκριμένη βάση δεδομένων έχει ως στόχο να διευκολύνει τους προγραμματιστές στην ανάπτυξη λογισμικού σε συγκεκριμένους τομείς. Πιο συγκεκριμένα, η CouchDB είναι μία βάση δεδομένων που αναπτύχθηκε και ταιριάζει απόλυτα σε εφαρμογές διαδικτύου (web applications) και ειδικότερα σε εφαρμογές που βασίζονται στην επεξεργασία εγγράφων, προσφέροντας ορισμένες πολύ χρήσιμες και καινοτόμες λειτουργίες προς αυτή την κατεύθυνση. Αξίζει να αναφέρουμε πως η CouchDB προσφέρει ένα RESTful API βασισμένο

[7] [9] https://en.wikipedia.org/wiki/MongoDB[10] https://docs.mongodb.com/

[11] https://dzone.com/articles/how-acid-mongodb

στο πρωτόκολλο HTTP για την επεξεργασία των βάσεων δεδομένων αλλά και μία διαδικτυακή διεπαφή (web interface) που ονομάζεται FUTON, απλοποιώντας σημαντικά την επεξεργασία των δεδομένων από τον χρήστη. Γενικότερα, η CouchDB δεν παρέχει επίσημα οδηγούς (drivers) για άλλες γλώσσες προγραμματισμού, ωστόσο υπάρχουν αρκετές ανεπίσημες επιλογές όπως ο driver Ektorp που χρησιμοποιείται σε αυτή τη διπλωματική εργασία και αφορά την Java.

Η CouchDB, όντας μια NoSQL βάση δεδομένων, δεν απαιτεί την ύπαρξη σταθερού καθορισμένου σχήματος, αλλά υποστηρίζει τη δυνατότητα δυναμικού σχήματος. Η δομή της αποτελείται από την ύπαρξη πολλών βάσεων δεδομένων, οι οποίες αντιστοιχούν στις συλλογές της MongoDB, και κάθε βάση δεδομένων περιέχει εγγραφές. Η συνήθης τακτική είναι κάθε βάση δεδομένων να περιέχει παρόμοιες εγγραφές, ωστόσο χάρη στο δυναμικό σχήμα οι εγγραφές της ίδιας βάσης δεδομένων μπορούν να διαφοροποιούνται σε ορισμένα πεδία ή και εξ' ολοκλήρου χωρίς κανένα σφάλμα. Οι εγγραφές της CouchDB είναι σε μορφή JSON και σε αυτή τη μορφή αποθηκεύονται στο δίσκο της βάσης δεδομένων. Η δομή μιας εγγραφής της CouchDB φαίνεται στο ακόλουθο σχήμα και βλέπουμε ότι μοιάζει αρκετά με αυτό της MongoDB.



Σχήμα 1.11: Εγγραφή CouchDB

Όπως παρατηρούμε, περιέχεται ένα πεδίο με όνομα _id που εξυπηρετεί τον ίδιο ρόλο όπως και στη MongoDB, καθώς περιέχει ένα αλφαριθμητικό που είναι μοναδικό για κάθε εγγραφή. Ωστόσο, παρατηρούμε πως στην εγγραφή περιέχεται ένα εξίσου ενδιαφέρον πεδίο, το _rev. Ο ρόλος του συγκεκριμένου πεδίου σχετίζεται με το γεγονός ότι η CouchDB δεν χρησιμοποιεί κλειδώματα, αλλά βασίζεται τον Έλεγχο Ταυτοχρονισμού Πολλαπλών Εγγραφών (MVCC). Έτσι, το πεδίο _rev καταδεικνύει την έκδοση της συγκεκριμένης εγγραφής. Πρέπει να σημειωθεί πως η επιλογή του MVCC αντί για κλείδωμα επιβάλλει το να κρατούντα όλες οι εκδόσεις για κάθε εγγραφή και άρα κάθε εισαγωγή πραγματοποιείται στο τέλος του αρχείου. Ο Έλεγχος Ταυτοχρονισμού Πολλαπλών Εγγραφών θα αναλυθεί στη συνέχεια.

Τέλος, αξίζει να αναφερθούμε στον τρόπο με τον οποίο αποθηκεύει τα δεδομένα η CouchDB. Αναφέραμε προηγουμένως πως η συγκεκριμένη βάση δεδομένων στοχεύει στην ταχύτατη επεξεργασία και αποθήκευση τεράστιου όγκου δεδομένων. Συνεπώς, επιλέγει να αποθηκεύει τα δεδομένα χρησιμοποιώντας τη δομή δεδομένων B+ δένδρο, πραγματοποιώντας μερικές παραλλαγές. Κάθε ενδιάμεσος κόμβος του δένδρου αποτελεί ουσιαστικά έναν δείκτη (index) ενώ τα φύλλα είναι αυτά που περιέχουν τα πραγματικά δεδομένα. Όπως είπαμε η εισαγωγή ή η ενημέρωση μιας εγγραφής προκαλεί τη δημιουργία μιας καινούριας εγγραφής νεότερης έκδοσης και τη διατήρηση της παλιάς εγγραφής. Συνεπώς, εφόσον η παλαιότερη έκδοση δεν αντικαθίσταται αλλά εισάγεται μία νέα εγγραφή, έχουμε ως αποτέλεσμα να πραγματοποιείται μία ανακατανομή του δένδρου και των κόμβων του. Η συγκεκριμένη δομή δεδομένων διατηρεί πολύ μικρό ύψος ακόμη και για τεράστιο αριθμό δεδομένων – για παράδειγμα ένα B+ δένδρο με ύψος 6 μπορεί να περιέχει δισεκατομμύρια δεδομένα- ενώ είναι εξαιρετικά πλατιά. Κατά αυτό τον τρόπο επιτυγχάνουμε ταχύτατη αναζήτηση και επεξεργασία δεδομένων, ενώ για ακόμα καλύτερη απόδοση τα B+ δένδρα συχνά διατηρούνται στη μνήμη με αποτέλεσμα να γίνονται προσβάσεις στη μνήμη μόνο για τα τελικά δεδομένα. Πρέπει να αναφερθεί πως η χρήση των B+ δένδρων και η διατήρηση όλων των εκδόσεων οδηγεί στη χρησιμοποίηση περισσότερου χώρου στο δίσκο συγκριτικά με άλλες λύσεις. Για την αντιμετώπιση αυτού του φαινομένου χρησιμοποιείται η τεχνική της συμπίεσης που εφαρμόζεται στα αρχεία ανά τακτά χρονικά διαστήματα, ενώ και οι αρκετά παλαιότερες εκδόσεις διαγράφονται από τη μνήμη από ένα χρονικό σημείο και μετά και διατηρούνται μόνο μερικά μετα-δεδομένα σχετικά με αυτές. [8]

[8] [12] http://guide.couchdb.org/[15] http://couchdb.apache.org/

[13] http://docs.couchdb.org/en/1.6.1/[14] http://wiki.apache.org/couchdb/

2.5.3.2 Λειτουργίες CRUD

Όπως και στην MongoDB, στο σημείο αυτό θα περιγράψουμε τον τρόπο με τον οποίο η CouchDB υλοποιεί τις 4 βασικές λειτουργίες. Όπως αναφέρθηκε, η CouchDB παρέχει ένα RESTful API βασισμένο σε HTTP και με τη χρήση αυτού υλοποιούνται όλες οι λειτουργίες.

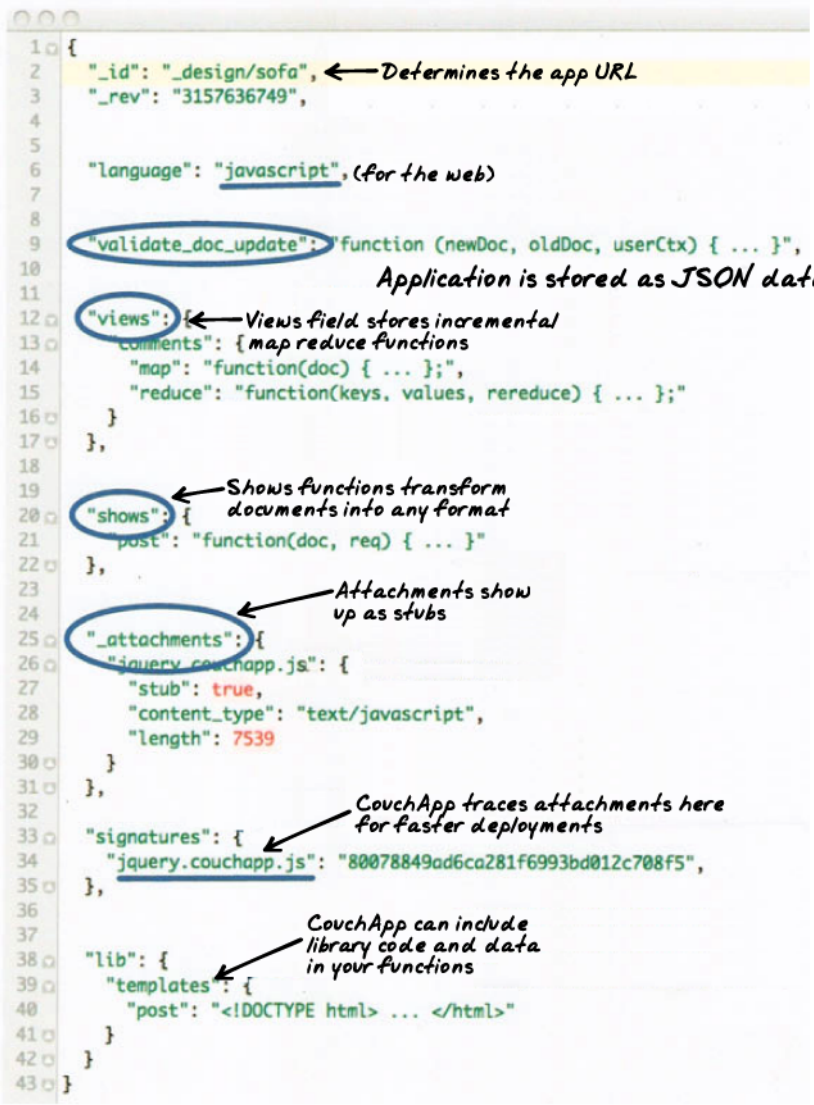
- Δημιουργία (Create): Με αυτή τη λειτουργία εισάγεται μία νέα εγγραφή στη βάση δεδομένων. Είναι απαραίτητο η εγγραφή να διαθέτει ένα _id και ένα _rev πεδίο. Εάν τα πεδία αυτά δεν έχουν οριστεί από το χρήστη τότε ορίζονται από τη βάση δεδομένων με τέτοιο τρόπο ώστε το _id να μην είναι ίδιο με το _id κάποιας άλλης εγγραφής (για παράδειγμα χρήση UUID) και το _rev να αποτελείται από ένα συνδυασμό δεδομένων που καταδεικνύουν την έκδοση της συγκεκριμένης εγγραφής.
- Ανάγνωση (Read): Η λειτουργία της ανάγνωσης αφορά την αναζήτηση ενός εγγράφου βάσει κάποιων χαρακτηριστικών του. Ωστόσο, στην CouchDB ως ανάγνωση υποστηρίζεται μόνο η αναζήτηση βάσει του _id σε συνδυασμό με το _rev. Αν δεν έχει επιλεγεί συγκεκριμένο _rev τότε το αποτέλεσμα της ανάγνωσης είναι η τελευταία έκδοση του εγγράφου. Στα σύγχρονα συστήματα η αναζήτηση εγγράφων με ελεύθερα κριτήρια αποτελεί επιτακτική ανάγκη. Για το λόγο αυτό η CouchDB περιλαμβάνει τη λειτουργία των Views. Ένα View ορίζεται από τον προγραμματιστή βάσει μιας συνάρτησης map-reduce γραμμένης σε javascript. Το View είναι ουσιαστικά μία παραμετροποίηση της βάσης με τέτοιο τρόπο ώστε να δημιουργείται μία όψη της που να βρίσκεται σε συμφωνία με το συγκεκριμένο map-reduce. Κατά αυτό τον τρόπο, μπορεί να γραφεί ένα map-reduce που να ικανοποιεί τα κριτήρια μιας αναζήτησης και άρα υποβάλλοντας ένα αίτημα στη βάση δεδομένων για αυτό το View να λαμβάνουμε τα αποτελέσματα της επιθυμητής αναζήτησης.
- Ενημέρωση (Update): Η λειτουργία της ενημέρωσης γίνεται με διαφορετικό τρόπο από απλή αλλαγή των πεδίων της εγγραφής που επιθυμούμε να ενημερώσουμε. Λόγω του ότι η CouchDB υλοποιεί Έλεγχο Ταυτοχρονισμού Πολλαπλών Εγγραφών, κάθε ενημέρωση υλοποιείται από μία ανάγνωση και μία δημιουργία. Αρχικά, γίνεται ανάγνωση της τελευταίας έκδοσης του εγγράφου που επιθυμούμε να ενημερωθεί. Στη συνέχεια, δημιουργείται μια νέα εγγραφή όμοια με αυτή που διαβάστηκε και ενημερώνονται τα κατάλληλα πεδία, ενώ ενημερώνονται και τα πεδία _id και _rev. Τέλος, η νέα αυτή εγγραφή εισάγεται στην κατάλληλη βάση δεδομένων του συστήματος. Η παλαιότερη έκδοση διατηρείται αν και δεν εμφανίζεται στην επισκόπηση των εγγραφών.
- Διαγραφή (Delete): Η λειτουργία της εγγραφής διαγράφει την εγγραφή που καθορίζεται από τα κριτήρια. Ουσιαστικά και πάλι προηγείται αρχικά η ανάγνωση της επιθυμητής εγγραφής και στη συνέχεια η διαγραφή της.

2.5.3.3 Σημαντικές επιπλέον λειτουργίες

Όπως είδαμε στην ανάλυση της λειτουργίας της ανάγνωσης, η CouchDB παρέχει μία εξαιρετικά ενδιαφέρουσα δυνατότητα για την αναζήτηση, τα Views. Ωστόσο δεν περιορίζεται μόνη σε αυτή, αλλά παρέχει και άλλες επιπλέον λειτουργίες, τις κυριότερες από τις οποίες θα αναλύσουμε σε αυτό το σημείο.

- **Design Document:** Όπως αναφέρθηκε, κάθε βάση δεδομένων της CouchDB μπορεί να διαθέτει ένα ή περισσότερα Views. Τα Views δημιουργούνται από τον προγραμματιστή-χρήστη σε Javascript και η αποθήκευση τους γίνεται στο design document. Το design document είναι ένα έγγραφο, όπως όλα τα έγγραφα της CouchDB, που περιέχει διάφορες παραμετροποιήσεις για τη συγκεκριμένη βάση δεδομένων στην οποία αναφέρεται. Συγκεκριμένα περιέχει τα εξής πεδία: _id, _rev, language, validate_doc_update, views, shows, lists, filters, updates, _attachments, signatures και lib. Το πεδίο _id χρησιμοποιείται ως μοναδικό αναγνωριστικό του εγγράφου και ξεκινάει απαραίτητα με το αλφαριθμητικό "_design/", ενώ το πεδίο _rev χρησιμοποιείται για την έκδοση όπως και στα υπόλοιπα έγγραφα. Το πεδίο language περιλαμβάνει τη γλώσσα στην οποία είναι γραμμένο το συγκεκριμένο έγγραφο και αυτή είναι η Javascript. Το πεδίο validate_doc_update περιέχει μία συνάρτηση που χρησιμοποιείται για την αποφυγή μη έγκυρων ενημερώσεων των εγγραφών της συγκεκριμένης βάσης δεδομένων, όπως για παράδειγμα στην περίπτωση που θέλουμε να επιτρέπεται ενημέρωση εγγραφών μόνο από εγγεγραμμένους στη βάση χρήστες. Το πεδίο views, όπως αναφέραμε, περιέχει τις συναρτήσεις map-reduce που χρησιμοποιούνται για να δημιουργήσουν διάφορα views στη βάση δεδομένων, απαντώντας κατά αυτό τον τρόπο σε συγκεκριμένα αιτήματα (requests). Το πεδίο shows περιέχει συναρτήσεις που έχουν ως στόχο όταν κληθούν να απεικονίσουν τις εγγραφές με συγκεκριμένο τρόπο. Θα ήταν εφικτό για παράδειγμα να δημιουργηθεί μία show function που θα επιστρέφει τα έγγραφα σε μορφή html με συγκεκριμένο styling. Οι συναρτήσεις shows δεν έχουν επιδράσεις στη βάση δεδομένων και δεν ενημερώνουν τις εγγραφές. Το πεδίο lists περιέχει συναρτήσεις που έχουν ως στόχο να επιστρέφουν ομαδοποιημένες εγγραφές. Το πεδίο filters περιέχει συναρτήσεις που έχουν στόχο να φιλτράρουν τα αποτελέσματα ως προς ένα χαρακτηριστικό και να τα απεικονίσουν με συγκεκριμένο τρόπο. Το πεδίο updates χρησιμοποιείται για ενημερώσεις που θέλουμε να γίνουν βάσει των δεδομένων και της λογικής που υπάρχει στον εξυπηρετητή (server-side logic). Στο πεδίο _attachments περιλαμβάνονται τα διάφορα αποκόμματα που χρησιμοποιούνται στη συγκεκριμένη βάση δεδομένων, όπως εικόνες, css αρχεία και άλλα. Το πεδίο signatures χρησιμοποιείται για να αποφεύγεται η άσκοπη ενημέρωση των attachments που δεν έχουν υποστεί αλλαγές. Τέλος, το πεδίο lib χρησιμοποιείται για να διατηρεί επιπλέον

συναρτήσεις javascript που είναι απαραίτητες. Στο ακόλουθο σχήμα παρουσιάζεται ένα τυπικό design document.



Σχήμα 1.12: Design Document της CouchDB

- **Αντίγραφα Δεδομένων:** Όπως έχει αναλυθεί και προηγουμένως, η έννοια του replication είναι εξαιρετικά σημαντική για την αποφυγή του κινδύνου απώλειας δεδομένων. Ο τρόπος με τον οποίο η CouchDB υλοποιεί την αντιγραφή των δεδομένων είναι ιδιαίτερα σημαντικός και αποτελεί ένα σημαντικό προτέρημα της συγκεκριμένης βάσης δεδομένων, καθώς εισάγει ορισμένες σημαντικές δυνατότητες. Πιο συγκεκριμένα, κατά τη διάρκεια της αντιγραφής των δεδομένων υπάρχει ένας κόμβος που ονομάζεται πηγή και ένας κόμβος που ονομάζεται προορισμός. Στόχος είναι να γίνει η ενημέρωση του κόμβου προορισμού για τις αλλαγές που έχουν συμβεί σε μία συγκεκριμένη βάση δεδομένων του κόμβου πηγή και άρα να συγχρονιστούν οι αντίστοιχες βάσεις δεδομένων των δύο κόμβων. Έτσι, αντιγράφονται όλες οι τελευταίες εκδόσεις κάθε εγγραφής από τον κόμβο πηγή στον κόμβο

προορισμό, εφόσον αυτές δεν υπάρχουν στον κόμβο προορισμό. Επίσης, όσες εγγραφές έχουν διαγραφεί στον κόμβο πηγή διαγράφονται και στον κόμβο προορισμό. Επίσης, κατά τη διάρκεια της αντιγραφής αντιγράφονται μόνο οι τελευταίες εκδόσεις των εγγραφών που δεν έχουν αντιγράφει σε προηγούμενη διαδικασία replication. Βλέπουμε λοιπόν ότι η CouchDB χρησιμοποιεί έναν πολύ αποδοτικό τρόπο για την αντιγραφή των δεδομένων. Ωστόσο, εξαιρετικά σημαντικό είναι το γεγονός πως προσφέρονται δύο επιπλέον επιλογές αντιγραφής δεδομένων πέρα από αυτή που περιγράψαμε (Master-Slave). Υπάρχει η επιλογή Master-Master που σημαίνει πως και οι δύο κόμβοι λειτουργούν ταυτόχρονα ως κόμβος πηγής και ως κόμβος προορισμού με αμφίδρομες αντιγραφές. Η τρίτη επιλογή ονομάζεται Filtered Replication και βασίζεται σε συναρτήσεις Javascript οι οποίες καθορίζουν συγκεκριμένα κριτήρια και μόνο οι εγγραφές που τα τηρούν συμμετέχουν στη διαδικασία της αντιγραφής δεδομένων.

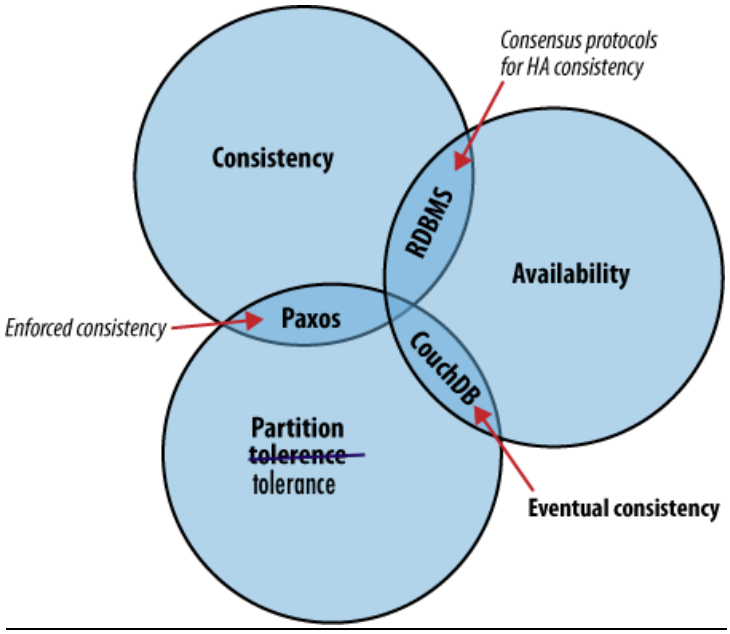
- **Έλεγχος Ταυτοχρονισμού Πολλαπλών Εκδόσεων:** Αναφέρθηκε στην εισαγωγή πως η CouchDB διατηρεί πολλαπλές εκδόσεις για κάθε αντικείμενο που απεικονίζεται στη βάση δεδομένων. Συγκεκριμένα, κάθε εγγραφή περιέχει το πεδίο _rev το οποίο αποτελεί ένα αναγνωριστικό της έκδοση του συγκεκριμένου εγγράφου και αποτελείται από ένα συνδυασμό ενός αύξοντα ακολουθιακού αριθμού , ο οποίος δείχνει πόσες φορές έχει ενημερωθεί το συγκεκριμένο έγγραφο, και μιας κωδικοποιημένης τιμής (hash) που δημιουργείται συγχωνεύοντας τον αύξοντα ακολουθιακό αριθμό με διάφορα δεδομένα του εγγράφου. Με τη χρήση αυτού του πεδίου λοιπόν, κάθε συναλλαγή μπορεί να "βλέπει" ένα συγκεκριμένο συνεπές στιγμιότυπο που αντιστοιχεί στις τιμές των εγγράφων προτού ξεκινήσει η συναλλαγή.

2.5.3.4 ACID Εγγυήσεις

Ατομικότητα: Η ιδιότητα της ατομικότητας διασφαλίζεται από την CouchDB σε επίπεδο εγγράφου, όπως και από την MongoDB. Δηλαδή, η ενημέρωση μιας εγγραφής είτε θα επιτύχει εξ΄ ολοκλήρου ή θα αποτύχει. Σε επίπεδο συναλλαγής δεν διασφαλίζεται η ατομικότητα.

Συνέπεια: Η CouchDB παρέχει ένα πιο "χαλαρό" επίπεδο συνέπειας. Πιο συγκεκριμένα ανήκει στις βάσεις δεδομένων που παρέχουν eventual consistency. Αυτό σημαίνει πως μία ενημέρωση θα φτάσει σίγουρα σε όλους τους κόμβους του συστήματος της βάσης δεδομένων, και άρα θα είναι ορατή από όλους τους χρήστες, κάποια στιγμή. Ωστόσο, το πότε θα έρθει σε συνέπεια το σύστημα μας δεν είναι γνωστό και άρα υπάρχουν χρονικά διαστήματα στα οποία η βάση μας δεν βρίσκεται σε συνέπεια με αποτέλεσμα να μην βλέπουν όλοι οι χρήστες τα ίδια δεδομένα, αλλά ορισμένοι να βλέπουν παλαιότερες εκδόσεις. Ο λόγος που η CouchDB κάνει αυτή την επιλογή προέρχεται από το θεώρημα CAP. Βάσει του θεωρήματος CAP (Consistency-Availability-Partition Tolerance) ένα σύστημα βάσεων δεδομένων που είναι κατανεμημένο δεν

μπορεί να διασφαλίζει και τις τρεις ιδιότητες. Αυτό σημαίνει πως η CouchDB δεν μπορεί να παρέχει αυστηρή συνέπεια (υπό την έννοια πως όλοι οι κόμβοι θα βλέπουν τα ίδια δεδομένα την ίδια χρονική στιγμή), υψηλή διαθεσιμότητα (με την όποια πως κάθε αίτημα λαμβάνει απάντηση ως προς το αποτέλεσμα του) και ανοχή στη διαμέριση (κάτι που σημαίνει πως το σύστημα συνεχίζει να εξυπηρετεί σωστά όλα τα αιτήματα και με σωστά δεδομένα παρά τις τυχαίες διαμερίσεις του συστήματος ξέχωρες ομάδες κόμβων λόγω της αποτυχία του δικτύου). Έτσι, η CouchDB επιλέγει να παρέχει high availability και partition tolerance θυσιάζοντας σε κάποιο βαθμό τη συνέπεια.[9]



Σχήμα 1.13: Θεώρημα CAP

Απομόνωση: Η ιδιότητα της απομόνωσης διασφαλίζεται καθώς όπως έχουμε δει η ενημέρωση μιας εγγραφής οδηγεί στη δημιουργία μιας νέας έκδοσης της εγγραφής αυτής. Κατά συνέπεια, σε περίπτωση δύο ταυτόχρονων συναλλαγών Α και Β όπου η Α ενημερώνει την εγγραφή Χ, δημιουργώντας μία νέα έκδοση της, και η Β διαβάζει την εγγραφή Χ, η Β δεν μπορεί να δει τις αλλαγές τις Α καθώς θα διαβάσει παλαιότερη έκδοση της εγγραφής.

Μονιμότητα: Η CouchDB παρέχει την εγγύηση της μονιμότητας. Πιο συγκεκριμένα, προκειμένου να διασφαλίσει πως τα αποτελέσματα μιας ολοκληρωμένης συναλλαγής έχουν την εγγύηση της μονιμότητας, έχει θέσει ως προεπιλεγμένη την λειτουργίας που διασφαλίζει πως μία συναλλαγή έχει ολοκληρωθεί όταν οι αλλαγές της έχουν αποθηκευτεί στο δίσκο. Φυσικά, αυτή η επιλογή μπορεί να αλλάξει από τον προγραμματιστή προκειμένου να επιτευχθεί μεγαλύτερη απόδοση.

[9] [28] http://robertgreiner.com/2014/06/cap-theorem-explained/

3

Ανάλυση

Στο συγκεκριμένο κεφάλαιο θα αναλύσουμε τον τρόπο με τον οποίο υλοποιήθηκε η προσθήκη του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων και των ACID ιδιοτήτων σε ένα σύστημα MongoDB. Όπως αναφέραμε στην εισαγωγή, η υλοποίηση αυτή αποτελεί μέρος του έργου CoherentPaas, στην ανάπτυξη του οποίο συμμετέχει το Ινστιτούτο Συστημάτων Επικοινωνιών και Πληροφορικής. Στόχος του CoherentPaaS[22] είναι η δημιουργία μιας cloud υπηρεσίας (Platform as a Service) που θα παρέχει βάσεις δεδομένων διαφορετικών τύπων, SQL, NoSQL, in-memory analytics, οι οποίες θα λειτουργούν με τέτοιο τρόπο ώστε να παρέχονται οι ACID ιδιότητες καθώς και η δυνατότητα εκτέλεσης ταυτόχρονων συναλλαγών προκειμένου να εξασφαλίζεται υψηλή κλιμακωσιμότητα. Εμείς θα αναφερθούμε μόνο στο κομμάτι που αφορά τη MongoDB και ιδιαιτέρως σε αυτό το κομμάτι που αφορά τις ACID ιδιότητες και την υποστήριξη Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων, καθώς στα πλαίσια του Coherent PaaS έχουν αναπτυχθεί και άλλες τεχνολογίες που αφορούν τη MongoDB όπως η μετατροπή SQL ερωτημάτων σε μορφή ερωτημάτων που υποστηρίζει η MongoDB. Στη συνέχεια θα αναφέρουμε ορισμένα σημεία στα οποία η CouchDB διαφοροποιείται σημαντικά από τη MongoDB και άρα δεν είναι εφικτό να εφαρμοστεί η ίδια ακριβώς υλοποίηση με τη MongoDB.

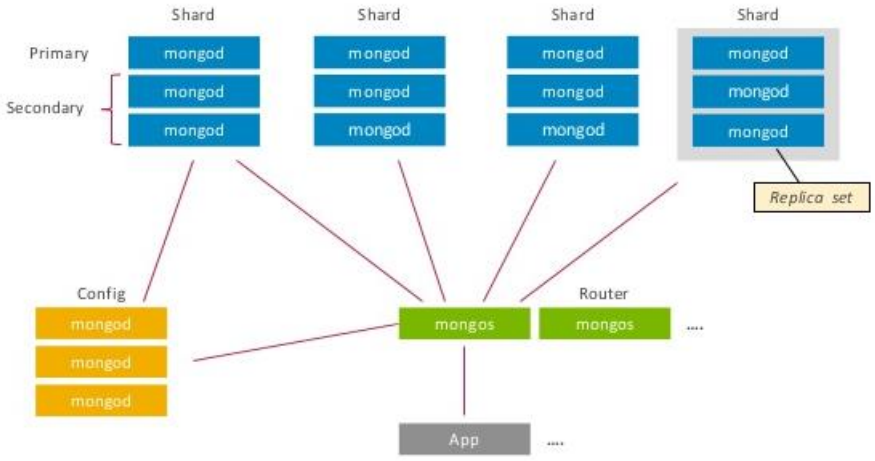
3.1 *Transactional MVCC MongoDB*

Σε αυτή την ενότητα θα περιγράψουμε τον τρόπο υλοποίησης ταυτόχρονων συναλλαγών σε ένα σύστημα βάσεων δεδομένων της MongoDB, παρέχοντας φυσικά τις ιδιότητες ACID. Συγκεκριμένα, για τη διαχείριση των ταυτόχρονων συναλλαγών και τη διασφάλιση των ιδιοτήτων ACID χρησιμοποιείται η τεχνική του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων (Multi-Version Concurrency Control) προκειμένου να υλοποιηθεί και να επιτευχθεί η Απομόνωση Στιγμιότυπου (Snapshot Isolation), όπως έχουν αναλυθεί οι δύο αυτές έννοιες στο προηγούμενο κεφάλαιο. Θα αναλυθούν η αρχιτεκτονική που χρησιμοποιήθηκε, οι

σχεδιαστικές αποφάσεις και οι λόγοι που οδήγησαν σε αυτές καθώς και ο τρόπος υλοποίησης τους. [20] [22] [23]

3.1.1 Αρχιτεκτονική

Η γενική αρχιτεκτονική που ακολουθεί το σύστημα της MongoDB παρουσιάζεται στο ακόλουθο σχήμα.

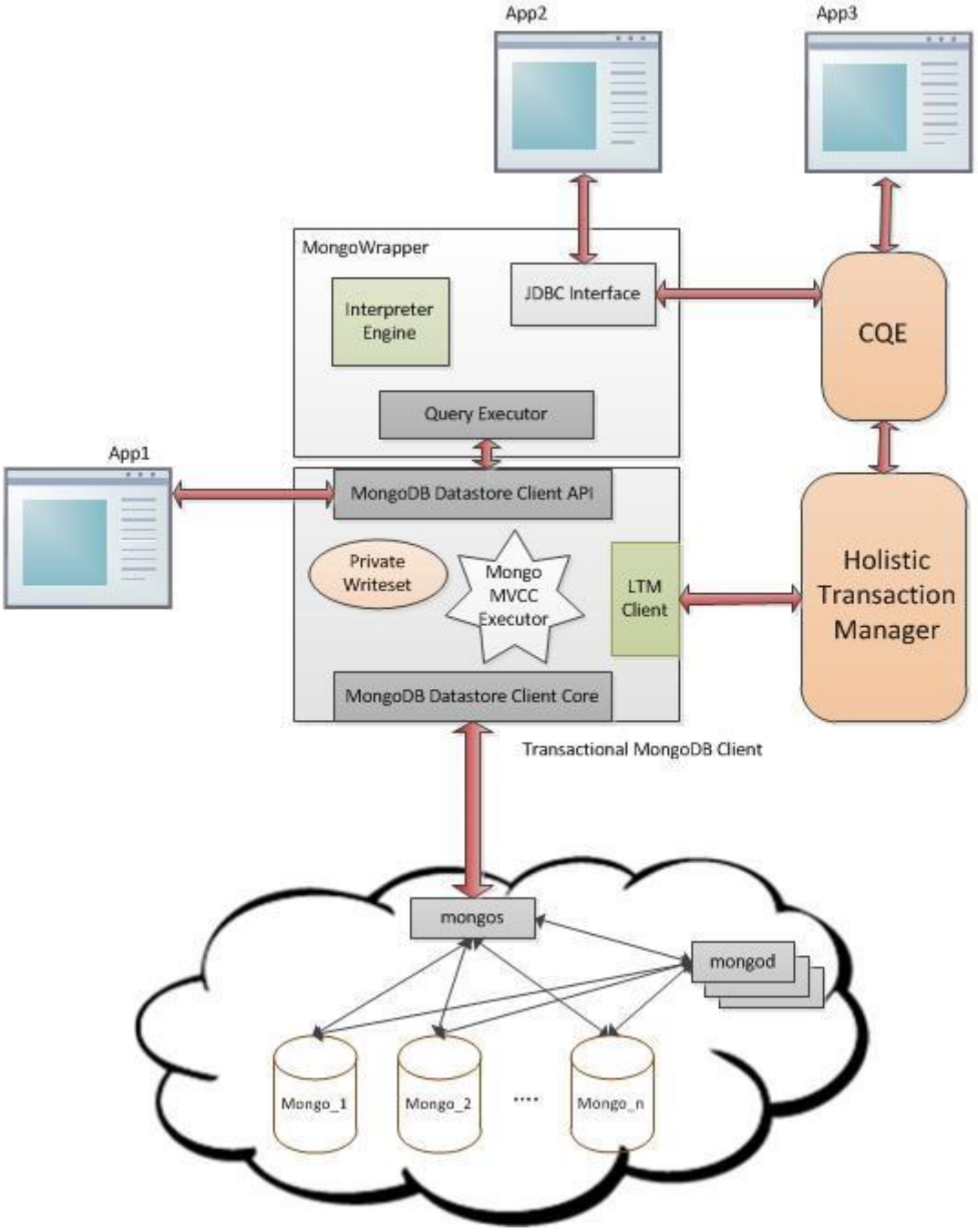


Σχήμα 3.1: Αρχιτεκτονική MongoDB

Παρατηρούμε πως η εφαρμογή επικοινωνεί απευθείας με ένα μέρος του συστήματος που ονομάζεται mongos και είναι ο δρομολογητής (router) όπως αναλύθηκε στο προηγούμενο κεφάλαιο.

Για την προσθήκη Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων και την εξασφάλιση των ACID ιδιοτήτων χρειάστηκε η προσθήκη αρκετών επιμέρους τμημάτων και η επέκταση σε μεγάλο βαθμό της αρχιτεκτονικής της MongoDB. Συγκεκριμένα προστέθηκαν τρία σημαντικά επιμέρους τμήματα. Το πρώτο τμήμα είναι ο Holistic Transaction Manager (HTM), ο οποίος έχει το ρόλο να διαχειρίζεται τις ταυτόχρονες συναλλαγές με τέτοιο τρόπο ώστε να ικανοποιούνται τα προαπαιτούμενα. Το δεύτερο στοιχείο που προστέθηκε είναι ένα επιπλέον επίπεδο μεταξύ της εφαρμογής και των δρομολογητών (mongos routers). Το επίπεδο αυτό είναι εξαιρετικά σημαντικό και είναι αυτό που επιτρέπει την επικοινωνία με τον HTM αλλά ταυτόχρονα είναι υπεύθυνο και για τη σωστή επικοινωνία με το σύστημα της MongoDB. Τέλος, προστέθηκε ένα στοιχείο που έχει το ρόλο του ρακοσυλλέκτη (garbage collector), με στόχο την αποδοτική χρησιμοποίηση του διαθέσιμου χώρου δεδομένου. Η συγκεκριμένη αρχιτεκτονική παρουσιάζεται στο ακόλουθο σχήμα και στη συνέχεια

ακολουθεί η ανάλυση των τμημάτων που προστέθηκαν και των λειτουργιών που αυτά προσφέρουν.



Σχήμα 3.2: Αρχιτεκτονική Επέκτασης MongoDB

Παρατηρώντας την ανωτέρω αρχιτεκτονική βλέπουμε ότι έχει προστεθεί ένα ενδιάμεσο επίπεδο μεταξύ του συστήματος της MongoDB και της εφαρμογής που το χρησιμοποιεί το οποίο ονομάζεται transactional MongoDB Client. Το επίπεδο αυτό είναι το κυριότερο τμήμα της υλοποίησης καθώς αποτελεί μία επέκταση του Java Driver της MongoDB και είναι αυτό που, σε συνδυασμό με τον Holistic Transaction Manager, προσδίδει τη λειτουργία του Ελέγχου

Ταυτοχρονισμού Πολλαπλών Εκδόσεων και των ACID ιδιοτήτων. Βλέπουμε πως αποτελείται από τρία κύρια στοιχεία. Ο LTMClient είναι υπεύθυνος για την επικοινωνία με τον Holistic Transaction Manager. Όπως θα δούμε, ο HTM είναι απαραίτητος για την εφαρμογή του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων. Ακόμη, ο LTMClient είναι το τμήμα αυτό που θα ειδοποιήσει τον Mongo MVCC Executor για την ολοκλήρωση ή την ακύρωση της συναλλαγής. Το δεύτερο στοιχείο που παρατηρούμε είναι το Private Writeset. Αυτό το τμήμα είναι ουσιαστικά μία συλλογή που περιέχει όλα τα κλειδιά των εγγραφών, τις οποίες η συγκεκριμένη συναλλαγή ενημερώνει ή εισάγει στη βάση δεδομένων. Η συλλογή αυτή διατηρείται στη μνήμη για διάφορους λόγους που θα παρουσιαστούν στη συνέχεια. Ακόμη, έχουμε το Mongo MVCC Executor που είναι το τμήμα που περιέχει όλη την λογική για την παροχή της λειτουργίας MVCC. Συγκεκριμένα, επεκτείνει όλες τις μεθόδους της MongoDB με τέτοιο τρόπο ώστε να υλοποιούν και να διαχειρίζονται εγγραφές πολλαπλών εκδόσεων καθώς και να διαχειρίζονται τη συλλογή private set. Στόχος μας φυσικά είναι όλη αυτή η λειτουργικότητα να παραμένει κρυφή από το χρήστη, με την έννοια ότι ένας απλός προγραμματιστής εφαρμογών φυσικής έκδοσης MongoDB να μπορεί να κάνει χρήση της συγκεκριμένης εφαρμογής. Κατά συνέπεια, είναι απαραίτητο να διατηρήσουμε τα ίδια ονόματα και τις ίδιες υπογραφές των μεθόδων προκειμένου να εξασφαλίσουμε τη συμβατότητα με όλες τις εφαρμογές που κάνουν χρήση της απλής MongoDB. Προκειμένου να επιτευχθεί αυτό υλοποιήθηκαν όλες οι υπάρχουσες διεπαφές (interfaces) και υπερκαλύφθηκαν οι απαραίτητες μέθοδοι κλάσεων. Έτσι, εισάγουμε το στοιχείο MongoDBDatastore Client API που ουσιαστικά παρέχει στην εφαρμογή το ισχύων API της MongoDB, παρέχοντας όμως την πρόσβαση στις επιπλέον λειτουργίες που προστίθενται. Τέλος, έχουμε το MongoDBDatastore Client Core που ουσιαστικά παρέχει την επικοινωνία με τη βάση. Με τη χρήση αυτού του στοιχεί γίνεται η διαχείριση των συνέσεων με τη βάση δεδομένων, η παραμετροποίηση των χαρακτηριστικών της βάσης δεδομένων καθώς και οι τελικές λειτουργίες για την επεξεργασία των δεδομένων στο δίσκο της βάσης δεδομένων. [24]

3.1.2 Σχεδιαστικές Αποφάσεις

Αναλύοντας την έννοια του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων είδαμε πως απαιτείται η διατήρηση πολλαπλών εκδόσεων για κάθε εγγραφή. Συνεπώς είναι απαραίτητο να γνωρίζουμε κάθε έκδοση σε ποια εγγραφή αναφέρεται και σε ποια χρονική στιγμή δημιουργήθηκε προκειμένου να μπορούμε να γνωρίζουμε ποια είναι η πιο πρόσφατη και ποια η έγκυρη για κάθε συναλλαγή. Προκειμένου λοιπόν να επιτευχθεί η παραπάνω απαίτηση κάθε εγγραφή είναι απαραίτητο να περιέχει συγκεκριμένα πεδία, κάθε ένα από τα οποία παρέχει μία ξεχωριστή πληροφορία.

_id: Είναι το πεδίο _id που απαιτείται να περιλαμβάνεται σε κάθε έγγραφο της MongoDB προκειμένου να διαφοροποιείται από οποιοδήποτε άλλο έγγραφο. Ωστόσο, στην υλοποίησή μας η ανάθεση τιμής στο πεδίο αυτό δεν αφήνεται στο μηχανισμό της MongoDB, αλλά η τιμή του πεδίου παρέχεται από δυναμικά. Η τιμή που δίνεται ως _id αποτελείται από 20 bytes και είναι μία συγχώνευση των πεδίων dataID και cmtTmstmp. Όταν αναλυθούνε περαιτέρω τα πεδία dataID και cmtTmstmp θα γίνει φανερό πως αποκλείεται δύο διαφορετικές εγγραφές ή εκδόσεις της ίδιας εγγραφής να έχουν το ίδιο _id.

dataID: Το πεδίο αυτό είναι το αναγνωριστικό κάθε εγγραφής. Αυτό σημαίνει πως δύο διαφορετικές εκδόσεις της ίδια εγγραφής έχουν το ίδιο dataID, ενώ δύο εγγραφές διαφορετικών αντικειμένων έχουν διαφορετικό dataID. Είναι ένα πεδίο που συγκροτείται από 12 bytes και είναι τύπου ObjectId, που παρέχεται από βιβλιοθήκες της MongoDB με κατανεμημένο τρόπο. Το πεδίο αυτό ουσιαστικά αντιστοιχεί σε ένα UUID όπως παρέχεται από τη Java, πράγμα που σημαίνει πως είναι ένα μοναδικό αναγνωριστικό ακόμη και για κατανεμημένα δεδομένα καθώς η κατασκευή του βασίζεται σε χρονοσφραγίδες και χαρακτηριστικά του μηχανήματος στο οποίο παράγεται.

cmtTmstmp: Το πεδίο αυτό αποτελείται από 8 bytes και διατηρεί την τιμή της χρονοσφραγίδας που αντιστοιχεί στη χρονική στιγμή κατά την οποία έγινε αποθήκευση και μονιμοποίηση της εγγραφής αυτής στο δίσκο (commit). Η χρονοσφραγίδα αυτή παρέχεται από τον Holistic Transaction Manager.

nextCmtTmstmp: Όπως φανερώνεται και από το όνομα, στο πεδίο αυτό αποθηκεύεται η χρονοσφραγίδα που δείχνει τη στιγμή που έγινε commit η αμέσως επόμενη έκδοση από αυτήν, και άρα είναι ένα πεδίο 8 bytes. Ουσιαστικά με βάση το commitTimestamp και το nextCommitTimestamp προκύπτει η χρονική διάρκεια κατά την οποία η συγκεκριμένη έκδοση ήταν η πιο πρόσφατη. Σε περίπτωση που δεν υπάρχει νεότερα έκδοση από τη συγκεκριμένη, τότε το πεδίο nextCmtTmstmp έχει την τιμή null.

tid: Κάθε φορά που ξεκινά η εκτέλεση μιας συναλλαγής, παρέχεται ένα μοναδικό αναγνωριστικό στη συναλλαγή αυτή. Το αναγνωριστικό αυτό αποθηκεύεται στο πεδίο tid προκειμένου να γνωρίζουμε η συγκεκριμένη έκδοση από ποια συναλλαγή δημιουργήθηκε.

isDeleted: Εφαρμόζοντας την τεχνική Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων, δεν είναι εφικτό να διαγράφουμε απλώς την πιο πρόσφατη έκδοση. Προκειμένου να θεωρηθεί πως ένα αντικείμενο δεν απεικονίζεται στη βάση και άρα έχει διαγραφεί θα έπρεπε να διαγραφούν όλες οι εκδόσεις των αντίστοιχων εγγραφών. Για την αποφυγή αυτής της μη αποδοτικής εργασίας χρησιμοποιούμε το πεδίο isDeleted το οποίο μας πληροφορεί αν πληροφορεί αν η συγκεκριμένη εγγραφή θεωρείται διαγραμμένη. Όταν το πεδίο αυτό είναι true σημαίνει πως η συγκεκριμένη εγγραφή έχει διαγραφεί, διαφορετικά μπορεί να είναι είτε false ή null.

Προτού αναλύσουμε τον τρόπο με τον οποίο υλοποιούνται οι CRUD λειτουργίες, είναι σκόπιμο να αναφερθούμε στο private writeset και στον τρόπο που διατηρείται για κάθε συναλλαγή. Συγκεκριμένα, οι εγγραφές που ενημερώνονται ή δημιουργούνται από μία συναλλαγή ανήκουν στο private writeset της συναλλαγής. Ωστόσο, ενώ θα έμοιαζε φυσική επιλογή να αποθηκεύονται στη μνήμη μέχρις ότου ολοκληρωθεί η συναλλαγή και περαστούν στη βάση δεδομένων, επιλέγουμε να αποθηκεύονται στο δίσκο της βάσης δεδομένων ως ατομική συλλογή εγγραφών προτού ολοκληρωθεί η συναλλαγή (commit). Στη μνήμη διατηρούνται ως κλειδιά μόνο το πεδίο dataID κάθε εγγραφής. Η επιλογή αυτή γίνεται καθώς σε περίπτωση μιας συναλλαγής που θα επηρέαζε μεγάλο πλήθος δεδομένων θα υπήρχαν προβλήματα με τη μνήμη και κακή απόδοση. Εφόσον όμως οι εγγραφές που αποτελούν το private writeset μιας συναλλαγής αποθηκεύονται στο δίσκο και όχι τοπικά στη μνήμη πρέπει να υπάρχει τρόπος να διαχωρίζονται τα private writesets που ανήκουν σε ταυτόχρονες συναλλαγές, οι οποίες πιθανόν και να τροποποιούν/ενημερώνουν τις ίδιες εγγραφές. Τη λύση μας την παρέχει το πεδίο tid καθώς χάρη σε αυτό αναγνωρίζουμε μία εγγραφή αν ανήκει στο private writeset που μας αφορά ή σε αυτό άλλης συναλλαγής.

Στο σημείο αυτό θα αναφερθούμε στον τρόπο με τον οποίο υλοποιούνται οι CRUD λειτουργίες προκειμένου να διασφαλίζονται όσα απαιτούνται από MVCC και Snapshot Isolation.

- Δημιουργία (Create): Η λειτουργία της δημιουργίας, όπως έχει αναφερθεί προηγουμένως, αποτελεί ουσιαστικά τη λειτουργία της εισαγωγής μιας καινούριας εγγραφής, η οποία δεν έχει προηγούμενες εκδόσεις στη βάση δεδομένων. Έτσι κατά τη εκτέλεση της συναλλαγή η λειτουργίας της δημιουργίας οδηγεί στη δημιουργία της καινούριας εγγραφής και κατά την ολοκλήρωση της συναλλαγής η εγγραφή αυτή αποθηκεύεται στη βάση δεδομένων έχοντας στο πεδίο _tid το μοναδικό αναγνωριστικό της συγκεκριμένης συναλλαγής, στο πεδία isDeleted την τιμή Null, στο πεδίο nextCmtTmstmp την κενή τιμή (“”) και στο πεδίο cmtTmstmp την τιμή της χρονοσφραγίδας που παρέχεται από τον HTM κατά τη στιγμή που η συναλλαγή ενημερώνει πως επιθυμεί να ολοκληρωθεί (commit).
    - Ανάγνωση (Read): Η λειτουργία της ανάγνωσης μιας εγγραφής έχει ως στόχο να φέρει από τη βάση δεδομένων την πιο πρόσφατη, έγκυρη και μη διαγραμμένη εγγραφή που υπάρχει στη βάση δεδομένων για το αντικείμενο που ζητείται από την εφαρμογή. Αυτό σημαίνει πως, για να τηρηθούν οι περιορισμοί που επιβάλλει ο Έλεγχος Ταυτοχρονισμού Πολλαπλών Εκδόσεων και η Απομόνωση Στιγμιότυπου, η ανάγνωση που εκτελείται από μία συναλλαγή θα πρέπει να επιστρέφει στη συναλλαγή μία εγγραφή που θα έχει τοποθετηθεί στη βάση προτού ξεκινήσει η συναλλαγή, θα είναι η πιο πρόσφατη ανάμεσα στις εκδόσεις που έχουν τοποθετηθεί στη βάση προτού ξεκινήσει η συναλλαγή και δεν θα έχει διαγραφεί. Επίσης σε περίπτωση που η συναλλαγή προτού πραγματοποιήσει την ανάγνωση έχει πραγματοποιήσει μία

ενημέρωση της εγγραφής την οποία θέλει να αναγνώσει, τότε το σωστό αποτέλεσμα είναι η ενημερωμένη εγγραφή και όχι η πιο πρόσφατη που βρίσκεται στη βάση δεδομένων, δηλαδή αυτή που βρίσκεται στο private writeset της συναλλαγής. Πιο συγκεκριμένα, μία εγγραφή που έχει ενημερωθεί από τη συναλλαγή και ανήκει στο private writeset θα έχει το πεδίο cmtTmstmp ίσο με κενή τιμή και το πεδίο tid ίσο με το transaction id της συναλλαγής αυτή. Επίσης, η πιο έγκυρη έκδοση μιας εγγραφής ως προς μία συγκεκριμένη συναλλαγή που ξεκίνησε τη χρονική στιγμή startTimestamp είναι αυτή που έχει εισαχθεί στη βάση πριν το startTimestamp και η επόμενη έκδοση της εγγραφής αυτής είτε δεν υπάρχει ή έχει εισαχθεί στη βάση δεδομένων μετά την έναρξη της συναλλαγής. Στην τελευταία περίπτωση σημαίνει πως αυτή η έκδοση είναι αποτέλεσμα ταυτόχρονης συναλλαγής με αυτή που μας ενδιαφέρει και δεν λαμβάνεται υπόψιν. Βασιζόμενοι στα παραπάνω καταλήγουμε πως μία έγκυρη ανάγνωση καθορίζεται από την εξής συνθήκη:

```
( ( cmtTmstmp == "") AND ( tid == @tid) AND ( isDeleted ==
NULL) )
OR
( ( cmtTmstmp < @startTmstmp) AND ( ( nextCmtTmstmp >
@startTmstmp) OR ( nextCmtTmstmp == "") ) AND (isDeleted ==
NULL) )
```

Η συνθήκη αυτή διασφαλίζει την έγκυρη ανάγνωση και είναι αυτή που υλοποιείται από την επέκταση της MongoDB που δημιουργήθηκε.

- Ενημέρωση (Update): Σύμφωνα με την τακτική του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων, η ενημέρωση μιας εγγραφής οδηγεί στην εισαγωγή μιας νέας εγγραφής με όλα τα ενημερωμένα πεδία που απαιτούνται. Συνεπώς, η λειτουργία της ενημέρωσης αρχικά πρέπει να δημιουργήσει μία νέα εγγραφή πανομοιότυπη με την τελευταία έκδοση της εγγραφής που υπάρχει στη βάση δεδομένων, στη συνέχεια να ενημερώσει τα κατάλληλα πεδία και τέλος να την εισάγει στη βάση δεδομένων όπως ακριβώς συμβαίνει και στη λειτουργία της Δημιουργίας. Αυτό όμως δεν αρκεί, καθώς πρέπει να διασφαλίσουμε πως πλέον αυτή θα θεωρείται η τελευταία έκδοση. Για το λόγο αυτό, πρέπει προτού την εισάγουμε στη βάση δεδομένων και κατά τη διάρκεια του commit, να ενημερώσουν το πεδίο nextCmtTmstmp της τελευταίας έκδοσης που υπάρχει στη βάση δεδομένων δίνοντας του την τιμή που θα τοποθετηθεί και στο πεδίο cmtTmstmp της νέας εγγραφής που δημιουργήσαμε.
- Διαγραφή (Delete): Κατά τη λειτουργίας της εγγραφής, θέλουμε να εισάγουμε μία νέα έκδοση της εγγραφής στη βάση δεδομένων που θα καταδεικνύει πως η συγκεκριμένη εγγραφή έχει διαγραφεί. Έτσι, ουσιαστικά γίνεται ότι ακριβώς και στη λειτουργία της ενημέρωση, με τη μόνη διαφορά πως το πεδίο isDeleted της νέας εγγραφής περιέχει την τιμή true.

Η υλοποίηση των τεσσάρων βασικών λειτουργιών της βάσης δεδομένων MongoDB κατά αυτό τον τρόπο διασφαλίζει την ύπαρξη και την σωστή διαχείριση πολλαπλών εκδόσεων για κάθε εγγραφή και κατά συνέπεια εφαρμόζεται η έννοια του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων. Στο σημείο αυτό πρέπει να τονίσουμε την καίρια σημασία του πεδίου nextCmtTmstmp για την επίτευξη του MVCC, καθώς χάρη σε αυτό το πεδίο αποφεύγονται πολλά σφάλματα που συνεπάγεται η ύπαρξη πολλαπλών εκδόσεων και ταυτόχρονα βελτιώνεται η απόδοση για τη λειτουργία της ανάγνωσης. Για την αιτιολόγηση αυτών των ισχυρισμών θα χρησιμοποιηθεί το ακόλουθο παράδειγμα. Έστω ότι έχουμε μία συλλογή σε μία βάση δεδομένων της MongoDB που διατηρεί τα στοιχεία των εργαζομένων μιας εταιρείας, χωρίς την ύπαρξη του πεδίου nextCmtTmstmp.

id	dataID	Employee	Salary	CT	isDeleted
...	...	...	...	...	...
1002	10	Mary	400	02	...
1010	10	Mary	500	10	
1015	10	Mary	1000	15	

Σχήμα 3.3: Παράδειγμα NextCommitTimestamp 1

Έστω λοιπόν ότι ξεκινά μία συναλλαγή με start timestamp 17 και επιχειρεί μία αναζήτηση των υπαλλήλων με μισθό κατώτερο του 800 (Salary<800). Η συναλλαγή αυτή έχοντας τη συγκεκριμένη εικόνα της βάσης δεδομένων θα λάβει υπόψιν της τις εγγραφές με id 1002,1010 και 1015 και άρα θα θεωρήσει πως η υπάλληλο Mary ανήκει στα αποτελέσματα, καθώς στις δύο πρώτες εκδόσεις έχει μισθό μικρότερο του 800. Κάτι τέτοιο ωστόσο είναι λανθασμένο καθώς η πιο πρόσφατη έκδοση για την υπάλληλο Mary είναι η εγγραφή με id 1015 και μισθό 1000, πράγμα που σημαίνει πως δεν θα έπρεπε να βρίσκεται στα αποτελέσματα. Κατά αυτή τη σχεδίαση, το ερώτημα είτε δεν έχει τρόπο να συμπεράνει αν η Mary πρέπει να ανήκει στο αποτέλεσμα ή θα πρέπει να ψάξει όλες τις εκδόσεις της Mary προκειμένου να βρει την τελευταία. Χρησιμοποιώντας όμως το πεδίο nextCmtTmstmp η εικόνα της συλλογής είναι η εξής.

id	dataID	Employee	Salary	CT	nextCT	isDeleted
...	...	...	...	...	...	...
1002	10	Mary	400	02	10	...
1010	10	Mary	500	10	15	
1015	10	Mary	1000	15	*null*	

Σχήμα 3.4: Παράδειγμα NextCommitTimestamp 2

Είναι εμφανές, πως το ερώτημα που εφαρμόζεται στη βάση δεδομένων μπορεί να συμπεράνει άμεσα πως οι εκδόσεις με id 1002 και 1010 έπαψαν να είναι έγκυρες σε χρονικές στιγμές που προηγούνται της έναρξης της συναλλαγής και άρα δεν πρέπει να ληφθούν υπόψιν.

Εξηγήσαμε τον τρόπο με τον οποίο διασφαλίζουμε τον Έλεγχο Ταυτοχρονισμού Πολλαπλών Εκδόσεων και των ACID ιδιοτήτων. Ωστόσο, είναι σημαντικό να αναφερθούμε στον τρόπο που χειριζόμαστε την ύπαρξη ταυτόχρονων συναλλαγών και τα προβλήματα που συνεπάγονται.

Ανάγνωση Στιγμιότυπου και Διαχείριση Ιδιωτικών Εκδόσεων

Όπως αναφέραμε κατά την ανάλυση της υλοποίησης, κάθε ενημέρωση, δημιουργία ή διαγραφή οδηγεί στην εισαγωγή μιας νέας εγγραφής στη βάση δεδομένων. Κατά τη διάρκεια της συναλλαγής οι νέες αυτές εγγραφές διατηρούνται σε μία ιδιωτική συλλογή (private writeset), η οποία αποθηκεύεται στο δίσκο και όχι στη μνήμη, και κατά την ολοκλήρωση της συναλλαγής αποθηκεύονται μόνιμα στη βάση δεδομένων οι εγγραφές αυτές. Μόνο τα κλειδιά της ιδιωτικής συλλογής διατηρούνται στη μνήμη, και αυτά είναι το dataID κάθε εγγραφής. Για να έχει κάθε συναλλαγή πρόσβαση μόνο στη δική της ιδιωτική συλλογή και να μην επιτρέπεται η πρόσβαση σε αυτές των άλλων ταυτόχρονων συναλλαγών, η πρόσβαση σε μία ιδιωτική συλλογή απαιτεί το αναγνωριστικό της συναλλαγής (transaction id). Ένα σημαντικό πρόβλημα που προκύπτει είναι ότι σε περίπτωση που μία συναλλαγή πραγματοποιήσει μία ενημέρωση ή διαγραφή κάποιας εγγραφής και στη συνέχεια επιχειρήσει ανάγνωση της εγγραφής αυτής τότε θα διαβαστούν τόσο η πιο πρόσφατη έγκυρη έκδοση της εγγραφής που υπάρχει στη βάση όσο και αυτή που υπάρχει στην ιδιωτική συλλογή. Ασφαλώς, η επιθυμητή έκδοση είναι αυτή της ιδιωτικής συλλογής καθώς είναι χρονικά η πιο πρόσφατη. Για την επίλυση αυτού του προβλήματος, όταν πρόκειται να γίνει ανάγνωση μιας εγγραφής από τη βάση δεδομένων, πρώτα ελέγχεται αν το dataID της συγκεκριμένης εγγραφής υπάρχει στην ιδιωτική συλλογή. Αν δεν υπάρχει τότε η ανάγνωση πραγματοποιείται, διαφορετικά παραλείπεται καθώς υπάρχει πιο πρόσφατη έκδοση της εγγραφής στην ιδιωτική συλλογή. Τέλος, πρέπει να αναφερθούμε στην περίπτωση που δύο διαφορετικές ταυτόχρονες συναλλαγές ενημερώσουν ή διαγράψουν την ίδια εγγραφή. Η περίπτωση αυτή ονομάζεται write-write conflict και έχει αναφερθεί στην ανάλυση της Απομόνωση Στιγμιότυπου. Η ανίχνευση αυτής της σύγκρουσης πραγματοποιείται από τον HTM και ακυρώνεται η τελευταία συναλλαγή που δεν έχει κάνει ακόμη commit. Αξίζει να αναφερθεί πως ο συγκεκριμένος έλεγχος πραγματοποιείται και στη λειτουργία της Δημιουργίας, καθώς ενδέχεται ο χρήστης να επιλέξει ως _id της εγγραφής ένα ήδη υπάρχων _id.

Υποστήριξη Επιπλέον Ευρετηρίων:

Αναφέραμε πως η MongoDB παρέχει από προεπιλογή ένα ευρετήριο για το πεδίο _id. Στη συγκεκριμένη υλοποίηση δημιουργείται αυτομάτως ένα επιπλέον ευρετήριο βασιζόμενο σε συνδυασμών κλειδιών-πεδίων και συγκεκριμένα στα πεδία dataID και cmtTmstmp. Το συγκεκριμένο ευρετήριο βελτιώνει σημαντικά την απόδοση των λειτουργιών CRUD όπως αυτές επεκτάθηκαν.

Οριστικοποιώντας το Commit:

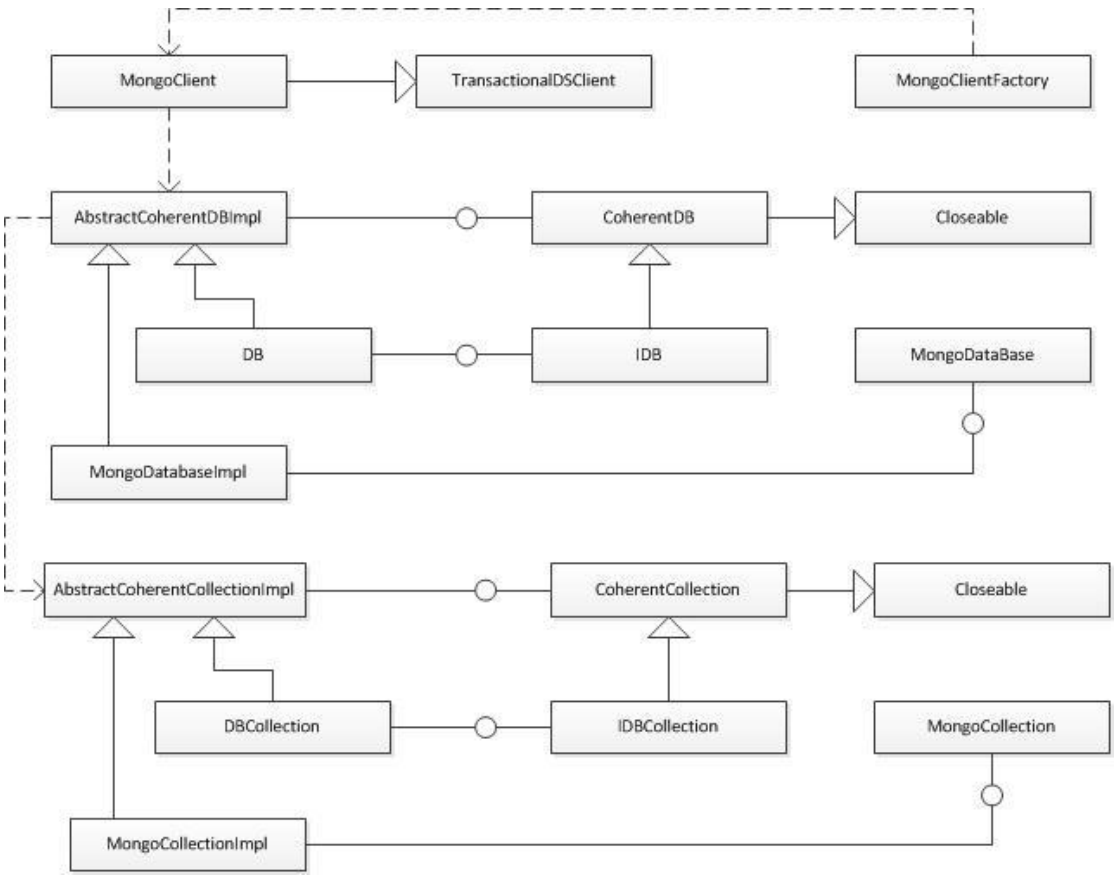
Κατά τη διάρκεια της ολοκλήρωσης μιας συναλλαγής πρέπει οι εγγραφές που υπάρχουν στην ιδιωτική συλλογή να μονιμοποιηθούν στη βάση δεδομένων. Ωστόσο, υπάρχει η περίπτωση να διακοπεί λόγω σφάλματος η λειτουργίας της ολοκλήρωσης. Για το λόγο αυτό, όλη η ιδιωτική συλλογή μετατρέπεται σε ένα πίνακα από bytes προκειμένου να αποδοθεί στον HTM και να διασφαλιστεί πως σε περίπτωση σφάλματος θα μπορεί να επιχειρηθεί εκ νέου η μονιμοποίηση των δεδομένων της συλλογής.

Ρακοσυλλέκτης Αχρείαστων Εκδόσεων:

Μία σημαντική λειτουργία που παρέχεται είναι η μόνιμη διαγραφή των απαρχαιωμένων εκδόσεων. Πιο συγκεκριμένα, ανά τακτά χρονικά διαστήματα ο HTM ελέγχει ποιες εκδόσεις είναι απαρχαιωμένες, υπό την έννοια πως δεν χρησιμοποιούνται από κάποια τρέχουσα συναλλαγή και υπάρχουν νεότερες εκδόσεις, και ενημερώνει το κατάλληλο λογισμικό προκειμένου να διαγράψει τις συγκεκριμένες εκδόσεις.

3.1.3 Υλοποίηση

Έχοντας αναλύσει τα βασικά στοιχεία τα οποία συνθέτουν την υλοποίηση μας, μπορεί να γίνει αναφορά με περισσότερες λεπτομέρειες στη υλοποίηση που πραγματοποιήθηκε. Συγκεκριμένα, θα παρουσιαστεί το διάγραμμα κλάσεων της υλοποίησης μας, προκειμένου να καταδείξουμε σε μεγαλύτερο βαθμό τη δομή της που ακολουθήθηκε, και στη συνέχεια θα παρουσιαστούν οι βασικές μέθοδοι και ο τρόπος λειτουργίας κάθε κλάσης. Πρέπει να τονιστεί πως θα παρουσιαστούν μόνο οι κύριες κλάσεις, οι οποίες περιέχουν τον τρόπο υλοποίησης των CRUD μεθόδων, ενώ δεν θα παρουσιαστούν πιο ειδικές κλάσεις που χρησιμοποιούνται για εξειδικευμένες λειτουργίες ή για λειτουργίες στις οποίες δεν έχουμε αναφερθεί.



Σχήμα 3.5: Διάγραμμα Κλάσεων Υλοποίησης MongoDB

Η ανάλυση ξεκινά από τις κλάσεις MongoClient, MongoClientFactory και την διεπαφή TransactionalDSClient. Η χρήση του συστήματος της βάσης δεδομένων MongoDB γίνεται μέσω μίας κλάσης που παρέχεται από τον Java Driver και είναι η MongoClient. Η κλάση MongoClient είναι υπεύθυνη για τη διαχείριση όλων των συνδέσεων με τη βάση δεδομένων. Γίνεται κατανοητό λοιπόν πως για κάθε εφαρμογή είναι επιθυμητό να υπάρχει μόνο ένα στιγμιότυπο της συγκεκριμένης κλάσης, κάτι το οποίο είναι σύσταση και της MongoDB. Για το λόγο αυτό είναι βολικό να χρησιμοποιηθεί το σχεδιαστικό μοτίβο Singleton. Κατά αυτό τον τρόπο, η κλάση MongoClientFactory είναι αυτή που δίνει στο χρήστη τη δυνατότητα αρχικοποίησης ενός MongoClient και παράλληλα διασφαλίζει πως αν έχει αρχικοποιηθεί ήδη ένας MongoClient τότε δεν θα αρχικοποιηθεί δεύτερος αλλά θα χρησιμοποιηθεί ο ήδη υπάρχων.

MongoClientFactory: Οι βασικότερες μέθοδοι της κλάσης αυτής είναι οι getInstance, createClient και getMongoClient. Η getInstance χρησιμοποιείται για την πρόσβαση σε ένα instance της συγκεκριμένης κλάσης καθώς η κλάση περιέχει ως static ένα στιγμιότυπο του εαυτού της και ο κατασκευαστής (constructor) έχει ορισθεί ως private. Η createClient παρέχεται σε διάφορες υπερφορτώσεις, καθώς μπορεί να καλεσθεί με διαφορετικές παραμέτρους προκειμένου να παραμετροποιηθεί διαφορετικά ο MongoClient, και η κλάση

αυτή αρχικοποιεί έναν MongoClient εάν δεν υπάρχει ήδη τέτοιος ή επιστρέφει τον ήδη υπάρχων. Τέλος, η getMongoClient επιστρέφει τον ήδη υπάρχων MongoClient.

MongoClient**:** Η κλάση αυτή περιέχει όλες τις μεθόδους που χειρίζονται την παραμετροποίηση και την επικοινωνία με τις βάσεις δεδομένων που διατηρούντα στο σύστημα της MongoDB. Αποτελεί μία υπερκάλυψη της κλάσης MongoClient που παρέχεται από τον Java Driver της MongoDB καθώς έχουν αλλαχθεί πολλές μέθοδοι και έχουν προστεθεί νέες προκειμένου να καλυφθούν οι νέες ανάγκες. Οι κυριότερες λειτουργίες που παρέχει είναι η διαχείριση των ενεργών βάσεων δεδομένων και γίνεται με μεθόδους όπως οι getDatabase, dropDatabase, getDatabaseNames αλλά και λειτουργίες που αφορούν τη διαχείρηση του writeset μιας συναλλαγής όπως πραγματοποιούν οι μέθοδοι applyWS, getWs. Τέλος, παρέχονται λειτουργίες αναφορικά με κάποια συναλλαγή όπως το rollback.

TransactionalDSClient: Το συγκεκριμένο interface επιβάλλει την υλοποίηση μεθόδων αναφορικά με τις συναλλαγές και την ιδιωτική τους συλλογή (private writeset). Συγκεκριμένα περιλαμβάνει τη δήλωση των μεθόδων applyWS, rollbck, getWS, getDataStoreID, redoWS και unleash.

Ολοκληρώνοντας την ανάλυση των κλάσεων που σχετίζονται με την λειτουργία του συστήματος της MongoDB, συνεχίζουμε την ανάλυση με πιο ειδικές κλάσεις και συγκεκριμένα αυτές που αφορούν τον τρόπο λειτουργίας της εκάστοτε βάσης δεδομένων μέσα στη MongoDB. Μπορούμε να δούμε πως κάθε βάση δεδομένων που απεικονίζεται από την κλάση CpMongoDatabaseImpl ανήκει σε ακριβώς έναν MongoClient, ενώ ο MongoClient μπορεί να διαθέτει περισσότερες ή καμία βάσεις δεδομένων.

MongoDatabase: Η συγκεκριμένη διεπαφή (interface) επιβάλλει την υλοποίηση βασικών μεθόδων που σχετίζονται με τη διαχείριση των συλλογών, όπως createCollection, getCollection, listCollections, αλλά και την παραμετροποίηση της συγκεκριμένης βάσης δεδομένων με μεθόδους όπως withCodecRegistry, withReadPreference. Παρέχεται από τον Java Driver της MongoDB ωστόσο έχουν προστεθεί τρεις νέες μέθοδοι, startTransaction, commit, rollback, που σχετίζονται με τις αντίστοιχες λειτουργίες μιας συναλλαγής.

AbstractMongoDatabaseImpl: Είναι μία αφηρημένη κλάση (abstract class) η οποία κληρονομείται από την ειδικότερη κλάση CpMongoDatabaseImpl και έχει στόχο να καταδείξει την ιεραρχία ορισμένων μεθόδων. Η κλάση αυτή διατηρεί σε μία δομή όλες τις ενεργές συλλογές (collections) που διαθέτει η συγκεκριμένη βάση δεδομένων, καθώς και τα στιγμιότυπα των κλάσεων που είναι υπεύθυνες για την επικοινωνία, όπως η MongoClient. Τέλος, υλοποιεί βασικές μεθόδους για τις συναλλαγές όπως startTransaction, commit, rollback αλλά και μεθόδους που καλούν τις κατάλληλες μεθόδους των συλλογών για την διαχείριση του private writeset, όόπως _applyWS και _getWS.

MongoDatabaseImpl: Είναι η κατώτερη κλάση βάση ιεράρχησης, επεκτείνει την αφηρημένη κλάση AbstractMongoDatabaseImpl, υλοποιεί τη διεπαφή MongoDatabase και παρέχει όλες τις υλοποιήσεις που δεν έχουν αναφερθεί και πρέπει να υλοποιηθούν βάση της διεπαφής MongoDatabas. Διατηρεί ένα στιγμιότυπο της κλάσης MongoDatabaseImpl, η οποία παρέχεται από τον Java Driver και περιέχει όλες τις απαραίτητες μεθόδους για τη διαχείριση των συλλογών της βάσης.

Τέλος, το τρίτο κομμάτι που διακρίνεται στο διάγραμμα κλάσεων είναι οι κλάσεις που αφορούν τις συλλογές που περιέχονται σε κάθε βάση της MongoDB. Παρατηρούμε πως κάθε στιγμιότυπο της κλάσης MongoCollectionImpl ανήκει σε ακριβώς ένα στιγμιότυπο της MongoDatabaseImpl και αντίστοιχα κάθε MongoDatabaseImpl διαθέτει κανένα ή περισσότερα στιγμιότυπα της κλάσης MongoCollectionImpl.

MongoCollection: Η διεπαφή αυτή παρέχεται από τον Java Driver και περιέχει τις δηλώσεις μεθόδων που αφορούν την παραμετροποίηση της συλλογής, όπως withReadPreference, withWriteConcern, δηλώσεις μεθόδων που αφορούν τις λειτουργίες CRUD και πιο σύνθετες λειτουργίες, όπως find, updateOne, updateMany, insertMany, deleteOne, aggregate, count, replaceOne, και τέλος δηλώσεις μεθόδων που σχετίζονται με τη διαχείριση της συλλογής όπως renameCollection, dropIndexes, createIndex.

AbstractMongoCollectionImpl: Είναι μία αφηρημένη κλάση (abstract class) η οποία υλοποιεί τη διεπαφή CpMongoCollection και έχει αντίστοιχο ρόλο με αυτό που έχει η κλάση AbstractMongoDatabaseImpl. Περιέχει μεθόδους που αφορούν κάποια τρέχουσα συναλλαγή όπως _getTransactionalContext, clearTxnCtx αλλά και μεθόδους που αφορούν κάποια από τις ιδιωτικές συλλογές (private writeset) της συλλογής, όπως _applyWriteSet, _getWriteSet, _rollback.

MongoCollectionImpl: Έχει αντίστοιχο ρόλο με αυτόν την MongoDatabaseImpl, υλοποιεί τη διεπαφή MongoCollection και επεκτείνει την αφηρημένη κλάση AbstractMongoCollectionImpl. Περιέχει στιγμιότυπο κάποιας κλάσης που υλοποιεί τη διεπαφή MongoCollection, προκειμένου μέσω αυτής να επιτυγχάνεται η επικοινωνία με τη βάση με τους τρόπους που παρέχει ο Java Driver. Περιλαμβάνει όλες τις CRUD λειτουργίες αλλά και πιο σύνθετες λειτουργίες, οι οποίες υλοποιούνται κατά τρόπο που να διασφαλίζουν τη στρατηγική του MVCC και τις ACID ιδιότητες. Ουσιαστικά η συγκεκριμένη κλάση αποτελεί τον πυρήνα της υλοποίησής μας. Περιέχει όλες τις μεθόδους που επιβάλλει η MongoCollection και τις οποίες υπερκαλύπτει καθώς πρέπει να υλοποιηθούν με διαφορετικό τρόπο, ενώ οι περισσότερες νέες μέθοδοι πρακτικά καλούνται από τις παραπάνω μεθόδους και το όνομα τους ξεκινά με κάτω παύλα. Οι κυριότερες από αυτές είναι οι _insert, _update, _delete, _find, _rollback και _applyWriteSet.

Στο σημείο αυτό ολοκληρώνεται η παρουσίαση των βασικών κλάσεων και διεπαφών που απαρτίζουν την υλοποίηση του συστήματος της MongoDB με εφαρμογή MVCC και ACID ιδιοτήτων.

3.2 *Σύγκριση MongoDB & CouchDB*

Έχοντας παρουσιάσει λοιπόν τον τρόπο με τον οποίο αναπτύχθηκε η MongoDB προκειμένου να υλοποιεί τον Έλεγχο Ταυτοχρονισμού Πολλαπλών Εκδόσεων και να παρέχει ACID ιδιότητες, στόχος μας είναι να επεκτείνουμε κατά τον ίδιο τρόπο την CouchDB και να αναπτυχθεί ένα ενιαίο και ολοκληρωμένο σύστημα που θα παρέχει τις υλοποιήσεις και για τις δύο αυτές NoSQL βάσεις δεδομένων. Ωστόσο, κατά την παρουσίαση και την ανάλυση των δύο αυτών βάσεων δεδομένων, που έγινε στο κεφάλαιο 2, έγινε κατανοητό πως υπάρχουν σημαντικές διαφοροποιήσεις στον τρόπο που λειτουργούν και εκτελούν τις CRUD λειτουργίες οι συγκεκριμένες βάσεις δεδομένων. Κατά συνέπεια, στη συγκεκριμένη ενότητα θα τονίσουμε τις βασικές διαφοροποιήσεις που υπάρχουν μεταξύ των MongoDB και CouchDB και είναι απαραίτητο να ληφθούν υπόψιν για την σωστή ανάπτυξη του συστήματος μας. Στη συνέχεια θα παρουσιάσουμε σε ένα εποπτικό και αφηρημένο επίπεδο πως θα μπορούσε να είναι ένα κοινό σύστημα που να παρέχει πρόσβαση και στις δύο υλοποιήσεις. Αξίζει να αναφέρθεί πως για την υλοποίηση του συστήματος της CouchDB που θα παρέχει Έλεγχο Ταυτοχρονισμού Πολλαπλών Εκδόσεων και ACID ιδιότητες χρησιμοποιήθηκε ένας οδηγός για την Java προκειμένου να χρησιμοποιηθεί η ίδια γλώσσα προγραμματισμού για τις δύο βάσεις δεδομένων. Συγκεκριμένα χρησιμοποιήθηκε ο Java Driver Ektorp, ο οποίος δεν είναι επίσημος από την CouchDB, αλλά παρόλα αυτά είναι ο πιο δημοφιλής και ευρέως χρησιμοποιούμενος. Οπότε ότι παρουσιάζεται ακολούθως είναι απόρροια τόσο της χρησιμοποίησης του Java Driver Ektorp όσο και των περιορισμών που επιβάλλει η ίδια η CouchDB.

3.2.1 Διαφοροποιήσεις μεταξύ MongoDB και CouchDB

Στο συγκεκριμένο υποκεφάλαιο θα αναλυθούν οι κυριότερες διαφοροποιήσεις μεταξύ των συστημάτων βάσεων δεδομένων της MognoDB και CouchDB και πως αυτές επηρεάζουν τη σχεδίαση και την υλοποίηση μας. Ο τρόπος αντιμετώπισης τους θα παρουσιασθεί στο κεφάλαιο που θα αναλυθεί η υλοποίηση που πραγματοποιήθηκε.

Απεικόνιση Εγγραφών της Βάσης Δεδομένων από τους οδηγούς

Μία σημαντική διαφοροποίηση που υπάρχει μεταξύ των δύο οδηγών Java που χρησιμοποιήθηκαν είναι στις κλάσεις που χρησιμοποιούνται για την απεικόνιση των αντικειμένων και την εισαγωγή τους στη βάση δεδομένων. Πιο συγκεκριμένα, ο Java Driver της MongoDB χρησιμοποιεί κλάσεις που λειτουργούν με τέτοιο τρόπο ώστε να παρέχουν την

απεικόνιση του αντικειμένου ώστε να ταιριάζει με τη μορφή που αποθηκεύονται στη βάση δεδομένων. Έτσι, από τον οδηγό της MongoDB για την Java παρέχεται η κλάση BasicDBObject η οποία διαθέτει όλες τις απαραίτητες μεθόδους, όπως η toJson για τη μετατροπή της κλάσης σε μορφή JSON, και την append, για την προσθήκη νέων ζευγαριών κλειδιού-τιμής στο αντικείμενο. Ακόμη, χρησιμοποιούνται οι κλάσεις Document, BSONObject και BsonDocument, που παρέχονται από το πακέτο org.bson και χρησιμοποιούνται για την απεικόνιση των αντικειμένων σε μορφή BSON.

Αντίθετα, ο Ektorp απαιτεί από το χρήστη να δημιουργεί αυτόνομα κάθε κλάση που απεικονίζει ένα είδος αντικειμένου που θα αποθηκευτεί στη βάση δεδομένων. Ωστόσο, κάθε κλάση που πρόκειται να χρησιμοποιηθεί για την εισαγωγή δεδομένων στη βάση πρέπει να είναι σειριοποιήσιμη (seriliazable) και αποσειριοποιήσιμη (deserializable) από το Jackson's Object Mapper και πρέπει οπωσδήποτε να διαθέτει τα πεδία _id και _rev. Κατά αυτό τον τρόπο ο Ektorp παρέχει τη δυνατότητα να απεικονιστούν τα αντικείμενα που θα αποθηκευτούν με κλάσεις της μορφής POJO (getters και setters) και μάλιστα απαιτείται η χρήση annotations, όπως τα json annotations για την αντιστοίχιση των πεδίων των κλάσεων με τα πεδία των εγγραφών. Ωστόσο, παρέχεται από τον ektorp και η κλάση CouchDbDocument η οποία μπορεί να κληρονομηθεί και να επεκταθεί από τις κλάσεις που θα χρησιμοποιηθούν για την απεικόνιση των αντικειμένων. Η χρήση της CouchDbDocument αποτρέπει τον ορισμό των πεδίων που επιβάλλει η CouchDB στις κλάσεις που ορίζονται από τον χρήστη, όπως το _id.

Βλέπουμε λοιπόν πως εμφανίζεται μία σημαντική διαφορά στον τρόπο απεικόνισης των αντικειμένων και πρέπει να ληφθεί μία σχεδιαστική απόφαση για τη γεφύρωση της διαφοράς αυτής και την αντιμετώπιση των αντικειμένων από τον προγραμματιστή-χρήστη με κοινό τρόπο ώστε να μην διακρίνεται η διαφορά μεταξύ MongoDB και CouchDB.

Διαχείριση του συστήματος βάσης δεδομένων

Μια σημαντική διαφορά είναι ο τρόπος με τον οποίο γίνεται η διαχείριση του συστήματος της βάσης δεδομένων και των συνδέσεων με αυτή. Όπως είδαμε κατά την ανάλυση της υλοποίησης Transactional MVCC MongoDB, η MongoDB παρέχει την κλάση MongoClient η οποία λειτουργεί ως singleton και είναι υπεύθυνη για τις παραπάνω αρμοδιότητες.

Ωστόσο, η CouchDB διαθέτει ένα διαφορετικό είδος client για την επικοινωνία και τη διαχείριση των συνδέσεων με το σύστημα της CouchDB. Πιο συγκεκριμένα ο Ektorp παρέχει την κλάση HttpClient που είναι ένας client βασισμένος στο πρωτόκολλο HTTP, όπως συμβαίνει και με την CouchDB.

Βλέπουμε λοιπόν πως πρέπει να βρούμε έναν κοινό τρόπο διαχείρισης των δύο συστημάτων ώστε ο προγραμματιστής να αποφεύγει να γνωρίζει αν πρέπει να χρησιμοποιήσει MongoClient ή HttpClient.

Πεδία Εγγραφής χωρίς Τιμές

Αναλύοντας τον τρόπο με τον οποίο υλοποιήθηκε ο Έλεγχος Ταυτοχρονισμού Πολλαπλών Εγγραφών στη MongoDB παρατηρείται ότι κάποια πεδία των εγγραφών μπορεί να μην περιλαμβάνουν τιμές. Για παράδειγμα, μπορεί το πεδίο nextCmtTmstmp να μην πρέπει να περιέχει τιμή καθώς αυτή η εγγραφή αποτελεί την πιο πρόσφατη έκδοση. Στη MongoDB για την περίπτωση αυτή δίναμε στα πεδία την κενή τιμή και χειριζόμασταν ανάλογα τους όποιους ελέγχους περιείχαν τα πεδία αυτά.

Ο τρόπος λειτουργίας της CouchDB δεν επιτρέπει στο πεδίο μιας εγγραφής να έχει την κενή τιμή. Πιο συγκεκριμένα, όταν ένα στιγμιότυπο μιας κλάσης επιχειρηθεί να εισαχθεί στη βάση δεδομένων και κάποιο από τα πεδία του έχει την τιμή “” τότε το πεδίο αυτό δεν απεικονίζεται καθόλου στη βάση. Κατά συνέπεια, ένα στιγμιότυπο της κλάσης με το πεδίο nextCmtTmstmp να περιέχει την τιμή “” θα οδηγήσει στη δημιουργία μιας εγγραφής χωρίς το πεδίο nextCmtTmstmp.

Για το λόγο αυτό στην CouchDB αντί για κενές τιμές χρησιμοποιήθηκαν μη λογικές τιμές για τα πεδία. Συγκεκριμένα για το πεδίο nextCmtTmstmp επιλέχθηκε να εισάγεται η τιμή -1 που καταδεικνύει πως αυτή είναι η νεότερη έκδοση. Ανάλογα με τις τιμές αυτές προσαρμόστηκαν κατάλληλα και οι αντίστοιχοι έλεγχοι.

Δομή του συστήματος βάσεων δεδομένων:

Αναλύοντας τον τρόπο λειτουργίας της MongoDB και τη δομή που επιλέγει για το σύστημά φάνηκε πως η MongoDB μπορεί να διατηρεί πολλές βάσεις δεδομένων. Κάθε βάση δεδομένων της MongoDB διαθέτει καμία ή περισσότερες συλλογές. Οι συλλογές είναι αυτές που περιέχουν τις εγγραφές όπως αυτές έχουν αναλυθεί σε προηγούμενο κεφάλαιο.

Η CouchDB ακολουθεί μία λίγο διαφοροποιημένη δομή. Συγκεκριμένα το σύστημα της CouchDB περιλαμβάνει διάφορες βάσεις δεδομένων και κάθε βάση δεδομένων περιέχει τις εγγραφές. Συνεπώς, δεν περιέχει συλλογές ή διαφορετικά μπορούμε να πούμε ότι οι βάσεις τις CouchDB έχουν το ρόλο των συλλογών της MongoDB.

Με βάση αυτό θα πρέπει να βρούμε ένα κοινό τρόπο μοντελοποίησης, αν και το συγκεκριμέο πρόβλημα λύνεται εύκολα όπως θα δειχθεί στη συνέχεια λόγω των κλάσεων που παρέχει η CouchDB.

Τρόπος χειρισμού της λειτουργίας Ανάγνωσης:

Μία από τις σημαντικότερες διαφοροποιήσεις που εντοπίζονται μεταξύ των δύο συστημάτων βάσεων δεδομένων είναι ο τρόπος με τον οποίο πραγματοποιούν τις αναγνώσεις, δηλαδή της λειτουργίας εύρεσης ενός ή περισσοτέρων στοιχείων βάσει κάποιων κριτηρίων.

Συγκεκριμένα, ο Java Driver της MongoDB παρέχει ένα σύνολο τελεστών, όπως έχει αναφερθεί, με βάση τους οποίους καθορίζονται τα κριτήρια που πραγματοποιείται η λειτουργία της ανάγνωσης, ή αλλιώς εύρεσης (find) όπως ορίζεται στη MongoDB. Συγκεκριμένα, δημιουργείται ένα αντικείμενο τύπου Document που περιέχει ζεύγη κλειδιού-τιμής, όπου κάθε ζεύγος είναι πρακτικά ένα κριτήριο εύρεσης. Το αντικείμενο αυτό δίνεται ως παράμετρος στη μέθοδο find και αυτή εκτελεί την ανάγνωση των κατάλληλων εγγραφών.

Στο σύστημα της CouchDB, και συγκεκριμένα στον Ektorp, η εύρεση ενός αντικεμένου γίνεται μόνο με κριτήριο τα πεδία _id και _rev. Εάν ο προγραμματιστής-χρήστης επιθυμεί να πραγματοποιήσει λειτουργία εύρεσης με βάση άλλα πεδία είναι υποχρεωμένος να δημιουργήσει μία κλάση μέσα στην οποία θα ορίζονται map-reduce συναρτήσεις και βάσει αυτών θα καθορίζονται Views στη βάση δεδομένων που επιθυμεί ο προγραμματιστής-χρήστης. Επίσης, είναι κατανοητό πως οι συναρτήσεις map-reduce είναι στατικές ως προς τις τιμές των πεδίων Αυτό σημαίνει πως δεν είναι εφικτό να δημιουργηθεί μία συνάρτηση map-reduce με πεδία που θα προέρχονται από μεταβλητές που θα λαμβάνου τιμή κατά το runtime.

Η συγκεκριμένη διαφοροποίηση είναι εξαιρετικά σημαντική και επιφέρει μεγάλη δυσκολία στη δημιουργία ενός κοινού τρόπου να υλοποιηθεί η λειτουργία ανάγνωσης και στα δύο συστήματα βάσεων δεδομένων. Ωστόσο, είναι σημαντικό να επιτευχθεί όσο το δυνατόν καλύτερη υλοποίηση σε ρεαλιστικά πλαίσια και αυτή θα παρουσιασθεί σε επόμενο κεφάλαιο.

Τρόπος χειρισμού της λειτουργίας Ενημέρωσης:

Μία ακόμη πολύ σημαντική διαφοροποίηση που εμφανίζεται μεταξύ των δύο συστημάτων και των δύο οδηγών είναι ο τρόπος που πραγματοποιούνται οι ενημερώσεις.

Συγκεκριμένα, η MongoDB και ο Java Driver παρέχει μία μέθοδο update η οποία δέχεται δύο αντικείμενα τύπου Document, όπως αυτό αναφέρθηκε στη λειτουργία της ενημέρωσης. Το ένα αντικείμενο περιέχει τα κριτήρια για την εύρεση των εγγραφών που πρέπει να ενημερωθούν και το άλλο αντικείμενο περιέχει τα πεδία που θα ενημερωθούν και τις τιμές που θα λάβουν. Οπότε στην υλοποίηση της MongoDB με MVCC αρκεί να δημιουργήσουμε πανομοιότυπα αντικείμενα με αυτά που ικανοποιούν τα κριτήρια, να ενημερώσουμε τα κατάλληλα πεδία και να τα εισάγουμε στη βάση δεδομένων.

Αντίθετα, η CouchDB και ο Ektorp προκειμένου να ενημερώσουν μία εγγραφή απαιτούν να εισαχθεί στη βάση δεδομένων μία κλάση όμοια με αυτή που πρέπει να ενημερωθεί και με μόνα διαφοροποιημένα τα πεδία που πρέπει να ενημερωθούν καθώς και το πεδίο της έκδοσης (_rev).

Παρατηρείται λοιπόν μια σημαντικότατη διαφοροποίηση που για την αντιμετώπιση της και τη δημιουργία ενός κοινό τρόπου χειρισμού της απαιτείται κάποιος σχεδιαστικός συμβιβασμός και μία αντίστοιχη απόφαση.

3.2.2 Περιγραφή κοινής αφηρημένης υλοποίησης (abstraction level)

Αναλύθηκαν και παρουσιάστηκαν οι βασικές διαφοροποιήσεις που εμφανίζονται μεταξύ των δύο συστημάτων βάσεων δεδομένων. Ασφαλώς, υπάρχουν και άλλες, μικρότερης σημασίας και δυσκολίας ως προς την αντιμετώπιση τους, διαφοροποιήσεις, οι οποίες δεν χρήζουν ιδιαίτερης αναφοράς και ανάλυσης. Προκειμένου να δημιουργηθεί ένα όσο το δυνατόν πιο ενιαίο σύστημα το οποίο θα δίνει στο χρήστη-προγραμματιστή την αίσθηση πως όλες οι λειτουργίες υλοποιούνται με τον ίδιο τρόπο και στα δύο υποστηριζόμενα συστήματα και άρα δεν απαιτείται περαιτέρω γνώση των επιμέρους βάσεων δεδομένων, είναι απαραίτητο να αντιμετωπιστούν οι διαφοροποιήσεις αυτές και να επιλυθούν τα προβλήματα που προκύπτουν. Η στρατηγική που ακολουθήθηκε για την αντιμετώπιση των δυσκολιών αυτών είναι να δημιουργηθεί μία διεπαφή που ταιριάζει περισσότερο στο σύστημα και τη λογική της MongoDB. Με άλλα λόγια η προσπάθεια επικεντρώθηκε στο να πλησιάσει, όσο ήταν εφικτό, η CouchDB τη λογική της MongoDB. Ο λόγος που ακολουθήθηκε αυτή η στρατηγική έγκειται στο γεγονός ότι η υλοποίηση της MongoDB ήταν ήδη σε πολύ μεγάλο βαθμό έτοιμη και μάλιστα αποτελούσε κομμάτι ενός πανευρωπαϊκού προγράμματος (CoherentPaaS). Κατά συνέπεια, ήταν επιθυμητό να πραγματοποιηθούν όσο το δυνατόν λιγότερες αλλαγές στο σύστημα της MongoDB. Ωστόσο, πρέπει να τονιστεί πως υπήρχαν ορισμένα σημεία στα οποία κρίθηκε πως θα ήταν προτιμότερο να ακολουθηθεί η λογική της CouchDB και άρα να προσαρμοστεί και να υποστεί αλλαγές η MongoDB, αλλά αυτά τα σημεία αποτελούσαν τη μειονότητα. Αυτό συνέβη για λόγους φιλικότητας της λειτουργίας στο χρήστη αλλά και για λόγους πολύ ευκολότερης υλοποίησης, συγκριτικά με την περίπτωση του να προσαρμοστεί η CouchDB στη λογική της MongoDB στα πλαίσια της λειτουργίας αυτής.

Πέρα από τις διαφοροποιήσεις όμως υπήρχαν και πάρα πολλές ομοιότητες μεταξύ των δύο βάσεων δεδομένων, κάτι που διευκόλυνε σε πολλά σημεία την υλοποίηση της κοινής διεπαφής. Οι κυριότερες ομοιότητες που εμφανίζονται σχετίζονται με το γεγονός ότι και οι δύο βάσεις δεδομένων ανήκουν στην κατηγορία των εγγραφο-κεντρικών NoSQL βάσεων δεδομένων, όπως για παράδειγμα η απεικόνιση των εγγραφών της βάσης δεδομένων σε μορφή JSON και το δυναμικό σχήμα των βάσεων δεδομένων. Ωστόσο, δεν θα αναλυθούν σε βάθος οι ομοιότητες που υπάρχουν καθώς οι περισσότερες από αυτές γίνονται εμφανείς κατά την περιγραφή των

συστημάτων της MongoDB και της CouchDB και δεν επιφέρουν κάποια σημαντική σχεδιαστική απόφαση για την υλοποίηση της ενιαίας διεπαφής χρήστη. Στο σημείο αυτό λοιπόν θα παρουσιαστεί σε γενικό και πιο αφηρημένο επίπεδο η σχεδίαση της συγκεκριμένη διεπαφής που θα παρέχει πρόσβαση στις CRUD λειτουργίες τόσο της MongoDB όσο και της CouchDB με όμοιο τρόπο.

Το πρώτο βήμα για την υλοποίηση είναι η δημιουργία ενός κοινού τρόπου διαχείρισης οποιασδήποτε από τις δύο βάσης δεδομένων. Όπως αναφέρθηκε κατά την περιγραφή της MongoDB η διαχείριση και η πρόσβαση στο σύστημα της MongoDB γίνεται από ένα στιγμιότυπο της κλάσης MongoClient. Αντίστοιχα, η CouchDB παρέχει έναν HttpClient για την πρόσβαση στο σύστημά της. Εύλογα λοιπόν συμπεραίνουμε πως θα πρέπει να υπάρχει ένας ενιαίος client που θα παρέχει πρόσβαση τόσο στη MongoDB όσο και στην CouchDB, ανάλογα με την επιλογή του προγραμματιστή-χρήστη, με τέτοιο τρόπο ώστε ο προγραμματιστής-χρήστης να μην παρατηρεί καμία διαφοροποίηση.

Σε δεύτερο επίπεδο, είναι απαραίτητο να παρέχεται στο χρήστη-προγραμματιστή μία κοινή και ενιαία εικόνα σχετικά με τη δομή της βάσης δεδομένων. Πιο συγκεκριμένα, είναι επιθυμητό να αποφευχθεί η περίπτωση να πρέπει ο χρήστης-προγραμματιστής να χειρίζεται διαφορετικά τις περιπτώσεις της CouchDB, όπου υπάρχουν μόνο βάσεις δεδομένων που περιέχουν εγγραφές, και διαφορετικά τις περιπτώσεις που αφορούν τη MongoDB, η οποία αποτελείται από βάσεις δεδομένων και αυτές με τη σειρά τους από συλλογές που περιέχουν εγγραφές. Κατά αυτό τον τρόπο θέλουμε ο χρήστης-προγραμματιστής να αντιμετωπίζει με τον ίδιο τρόπο τις δύο βάσεις δεδομένων και λόγω της απόφασης μας να φέρουμε την CouchDB πιο κοντά στη MongoDB η επιλογή θα είναι τέτοια ώστε ο χρήστης-προγραμματιστής να λαμβάνει την εικόνα πως το σύστημα αποτελείται από βάσεις δεδομένων και αυτές αποτελούνται από συλλογές εγγραφών, είτε πρόκειται για βάση του συστήματος της MongoDB ή για βάση του συστήματος της CouchDB.

Επίσης, όπως έχει γίνει κατανοητό, στόχος του ενιαίου συστήματος είναι να λειτουργεί με τέτοιο τρόπο ώστε ο χρήστης-προγραμματιστής να απαιτείται να γνωρίζει μόνο τον τρόπο λειτουργίας του συστήματος που παρέχεται και όχι των επιμέρους λειτουργιών των δύο βάσεων δεδομένων που υποστηρίζονται. Απλούστερα, δεν πρέπει να απαιτείται από το χρήστη η γνώση των εντολών για τις CRUD λειτουργίες του MongoDB Java Driver και του Ektorp (CouchDB Java Driver). Θα πρέπει να δημιουργηθούν οι κατάλληλες εντολές για το σύστημα που θα υλοποιηθεί, ώστε οι εντολές αυτές να λειτουργούν τόσο για τη MongoDB όσο και για την CouchDB. Επομένως, θα πρέπει με την ίδια εντολή να εκτελείται η κατάλληλη λειτουργία της MongoDB ή της CouchDB ανάλογα με την επιλογή του χρήστη-προγραμματιστή. Ασφαλώς, αυτό δεν σημαίνει πως απλά πρέπει να δημιουργήσουμε μία εντολή με ένα συμβατικό όνομα,

για παράδειγμα insert, και η εντολή αυτή να προκαλεί την εκτέλεση της αντίστοιχης εντολής για το MongoDB Java Driver (insertMany) ή για τον Ektorp (create). Η όλη δυσκολία έγκειται στις παραμέτρους που απαιτούν οι εντολές του εκάστοτε οδηγού και κυρίως ο τρόπος απεικόνισης του αντικειμένου που θα εισαχθεί στη βάση.

Εύλογα λοιπόν οδηγούμαστε στο επόμενο βήμα της σχεδίασης του ενιαίου συστήματος. Αυτό το βήμα περιλαμβάνει την απεικόνιση του αντικειμένου με κοινό τρόπο για τις 2 βάσεις δεδομένων. Αναφέρθηκε πως για τη MongoDB χρησιμοποιούνται κλάσεις όπως η BasicDBObject και BSONObject για τη δημιουργία μιας νέας εγγραφής. Εν αντιθέσει, η CouchDB επιβάλλει τη δημιουργία, από τον προγραμματιστή, μιας κλάσης που θα απεικονίζει με κατάλληλο τρόπο το αντικείμενο και χρησιμοποιείται η μορφοποίηση POJO για την κλάση μαζί με τα κατάλληλα annotations (json ή Jackson annotations), για την κατάλληλη αντιστοίχιση των πεδίων της κλάσης στα πεδία της εγγραφής. Στη συγκεκριμένη περίπτωση αποφασίστηκε να επιλεχθεί η προσέγγιση που προσφέρει η CouchDB. Αυτό σημαίνει πως ο προγραμματιστής-χρήστης θα δημιουργεί μία κλάση που θα απεικονίζει το αντικείμενο με κατάλληλο τρόπο και αυτό το αντικείμενο θα δίνεται ως όρισμα στην εκάστοτε λειτουργία. Στην περίπτωση της CouchDB θα χρησιμοποιείται αυτούσιο ενώ στην περίπτωση της MongoDB θα μετατρέπεται σε αντικείμενο τύπου BasicDbObject ή BSONObject και στη συνέχεια θα παρέχεται ως όρισμα στην κατάλληλη λειτουργία της MongoDB. Ο λόγος που επιλέχθηκε ο συγκεκριμένο τρόπος αντιμετώπισης είναι ότι είναι σημαντικά ευκολότερη η μετατροπή μιας κλάσης που δημιουργήθηκε από το χρήστη σε ένα αντικείμενο BSONObject, συγκριτικά με την περίπτωση της μετατροπής BSONObject σε διαφορετική κλάση.

Τέλος, είναι πολύ σημαντικό να αντιμετωπίζονται με κοινό τρόπο τα αποτελέσματα που επιστρέφονται από την εκτέλεση της εκάστοτε λειτουργίας. Ειδικότερα, η λειτουργία της δημιουργίας (create) δεν επιστρέφει κάποιο αποτέλεσμα. Η λειτουργία της ενημέρωσης (update) και της διαγραφής (delete), όπως υλοποιούνται στην επέκταση της MongoDB, επιστρέφουν ένα ενημερωτικό αποτέλεσμα. Αυτό επιλέγεται να υλοποιηθεί και στην περίπτωση της CouchDB και άρα οι αντίστοιχες εντολές που παρέχονται από το ενιαίο σύστημα που δημιουργείται επιστρέφουν ένα ενημερωτικό αποτέλεσμα. Όσον αφορά την εντολή της ανάγνωση (read), δηλαδή της αναζήτησης εγγραφών βάσει κριτηρίων, υπάρχει μία ιδιαιτερότητα. Στη MongoDB επιστρέφεται ένα στιγμιότυπο της κλάσης iterator που παρέχει τα αποτελέσματα. Στην CouchDB η αναζήτηση με χρήση των Views, όπως αυτά έχουν αναλυθεί πρωτύτερα, έχουν ω αποτέλεσμα τη δημιουργία μιας λίστας με τα αποτελέσματα της αναζήτησης. Πάλι, επιλέγουμε να ακολουθήσουμε τον τρόπο της MongoDB και άρα η λειτουργία της ενημέρωσης που παρέχεται από την ενιαία διεπαφή επιστρέφει ένα στιγμιότυπο

της κλάσης iterator. Επομένως, θα πρέπει να επεκταθεί και να προσαρμοστεί η CouchDB με τέτοιο τρόπο ώστε να επιτυγχάνεται η επιλογή αυτή.

Αναλύθηκε σε θεωρητικό επίπεδο ο τρόπος με τον οποίο θα σχεδιαστεί και θα λειτουργεί η κοινή διεπαφή που θα δίνει στο χρήστη τη δυνατότητα να εκτελεί τις CRUD λειτουργίες με όμοιο τρόπο και με τις ίδιες ακριβώς εντολές στις δύο NoSQL βάσεις δεδομένων που χρησιμοποιούνται. Στο ακόλουθο κεφάλαιο θα αναλύσουμε σε μεγαλύτερο βάθος και πιο περιγραφικά τον τρόπο με τον οποίο υλοποιήθηκε η ενιαία κοινή διεπαφή καθώς και τον τρόπο λειτουργίας της.

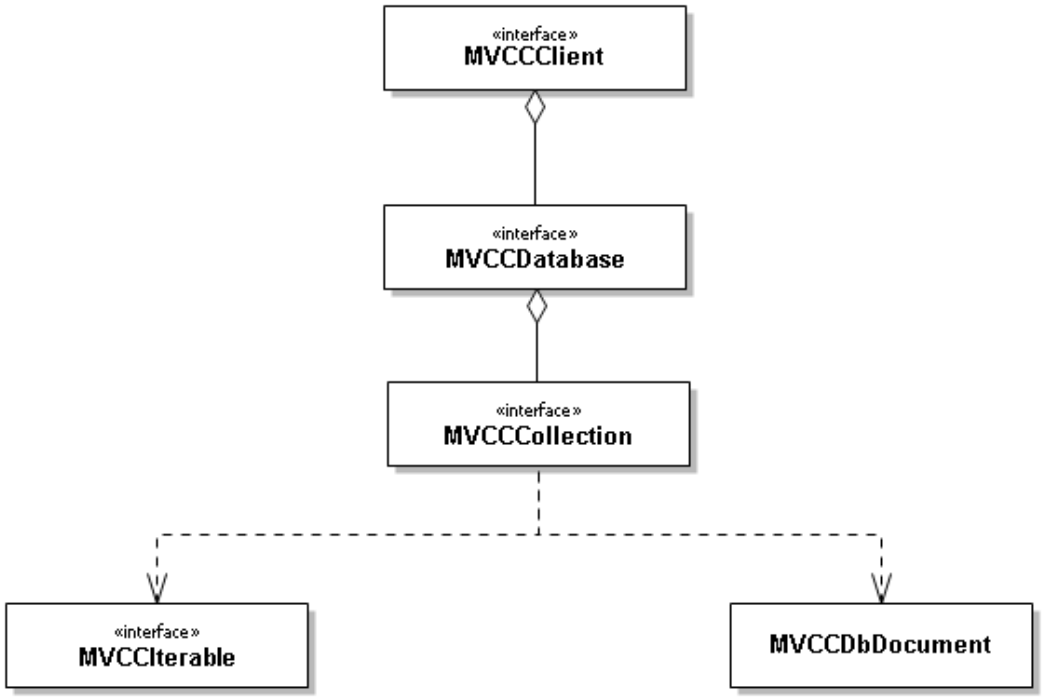
4

Σχεδίαση Συστήματος Ενιαίας Διεπαφής

Στο παρόν κεφάλαιο θα αναλυθεί η ενιαία υλοποίηση που παρέχεται για τη χρήση των επεκτάσεων των MongoDB και CouchDB προκειμένου να παρέχουν Έλεγχο Ταυτοχρονισμού Πολλαπλών Εκδόσεων και ACID ιδιότητες. Επίσης, θα αναλυθεί η σχεδίαση της επέκτασης της CouchDB προκειμένου να υποστηρίζει MVCC και να παρέχει ACID ιδιότητες.

Στο συγκεκριμένο υποκεφάλαιο θα αναλύσουμε εκτενέστερα τη σχεδίαση και την υλοποίηση της διεπαφής που προσφέρει έναν ενιαίο και συστηματικό τρόπο για να εκτελούνται οι βασικές CRUD λειτουργίες και στα δύο συστήματα βάσεων δεδομένων. Πιο συγκεκριμένα θα παρουσιαστούν τα διαγράμματα κλάσεων κάθε επιμέρους τμήματος που απαρτίζει την υλοποίηση με τέτοιο τρόπο ώστε να καταδεικνύεται τόσο η δομή, η ιεράρχηση και η κληρονομικότητα που επιλέχθηκε για τις κλάσεις αυτές, όσο και η λειτουργικότητα μέσω των μεθόδων που χρησιμοποιήθηκαν. Κατά αυτό τον τρόπο, θα παρουσιάζονται δύο διαγράμματα κλάσεων για κάθε τμήμα. Το πρώτο έχει στόχο την ανάδειξη των συσχετίσεων μεταξύ των κλάσεων. Το δεύτερο αφορά περισσότερο τις βασικές μεθόδους που υλοποιούνται για κάθε μία κλάση.

Αρχικά παρουσιάζεται ένα γενικό διάγραμμα κλάσης το οποίο έχει ως στόχο να παρουσιάσει τα βασικά μέρη που απαρτίζουν την ενιαία υλοποίηση. Επίσης, παρουσιάζονται οι βασικές συσχετίσεις μεταξύ των κλάσεων και των διεπαφών.



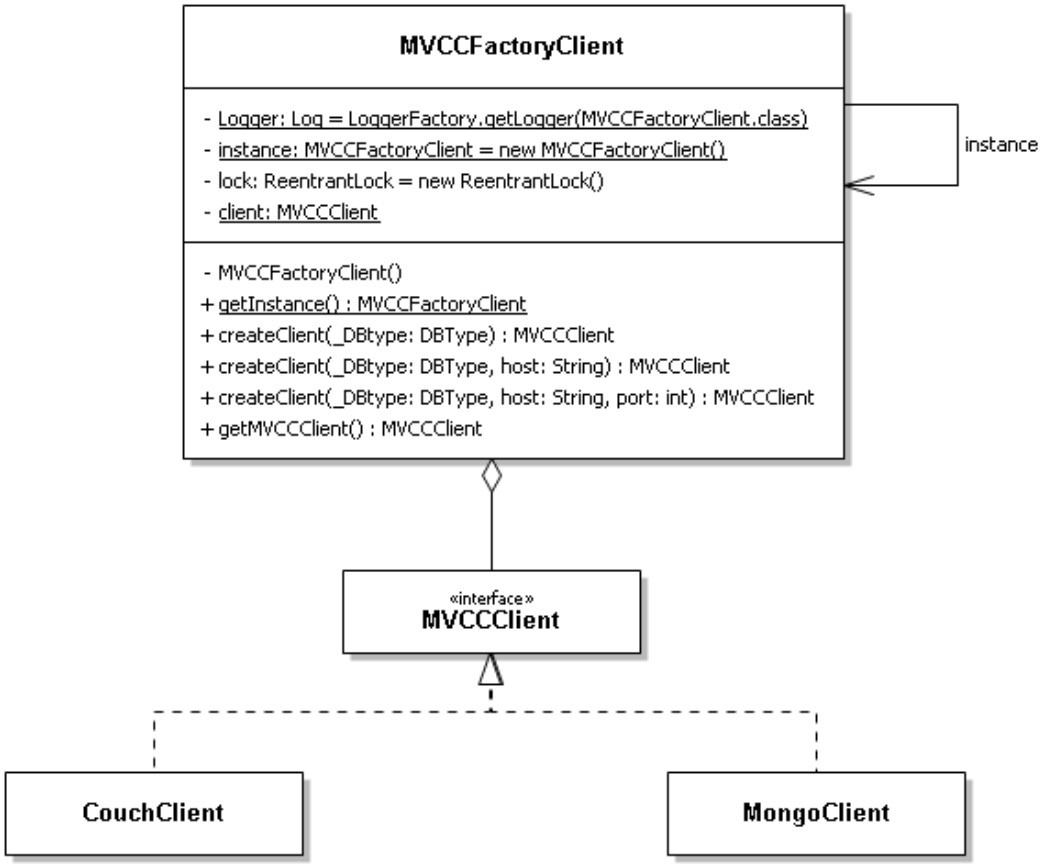
Σχήμα 4.1: Διάγραμμα Κλάσεων Αφηρημένης Υλοποίησης

Παρατηρώντας το παραπάνω σχήμα, παρατηρούνται τέσσερις διακριτές διεπαφές και μία κλάση. Κάθε επιμέρους τμήμα έχει ως αρχικό του ονόματος το MVCC (Multi Version Concurrency Control) γεγονός που αποτελεί σχεδιαστική απόφαση προκειμένου να καταδεικνύεται πως οι συγκεκριμένες διεπαφές και κλάσεις αποτελούν μέρος της ενιαίας διεπαφής που προσφέρεται στον προγραμματιστή-χρήστη. Αυτές οι διεπαφές υλοποιούν το αφηρημένο επίπεδο (abstract level) που θα χρησιμοποιηθεί για τη χρήση των επεκτάσεων των MongoDB και CouchDB, προκειμένου να υλοποιούν το MVCC, με όμοιες εντολές και όμοιο τρόπο.

Ξεκινώντας από το υψηλότερο επίπεδο, παρατηρείται ότι υπάρχει η διεπαφή MVCCClient, η οποία μπορεί να περιλαμβάνει καμία ή περισσότερες βάσεις δεδομένων του συστήματος στο οποίο απευθύνεται (MongoDB ή CouchDB). Κάθε βάση δεδομένων απεικονίζεται από τη διεπαφή MVCCDatabase και αντιστοιχεί σε ένα μοναδικό MVCCClient. Επίσης κάθε βάση δεδομένων μπορεί να περιέχει καμία ή περισσότερες συλλογές. Κάθε συλλογή απεικονίζεται από τη διεπαφή MVCCCollection και ανήκει σε ακριβώς μία βάση δεδομένων (MVCCDatabase). Η συγκεκριμένη ιεραρχία αποτελεί σχεδιαστική απόφαση, όπως αναλύθηκε στο προηγούμενο κεφάλαιο, η οποία βρίσκεται σε απόλυτη αντιστοιχία με την ιεραρχία και τη δομή του συστήματος της MongoDB. Τέλος, υπάρχουν η διεπαφή MVCCIterable και η κλάση MVCCDbDocument, οι οποίες ενδέχεται να παρέχουν στιγμιότυπα τους σε πεδία συλλογών.

Στη συνέχεια θα αναλυθεί η χρησιμότητα και η λειτουργικότητα της εκάστοτε διεπαφής και κλάσης.

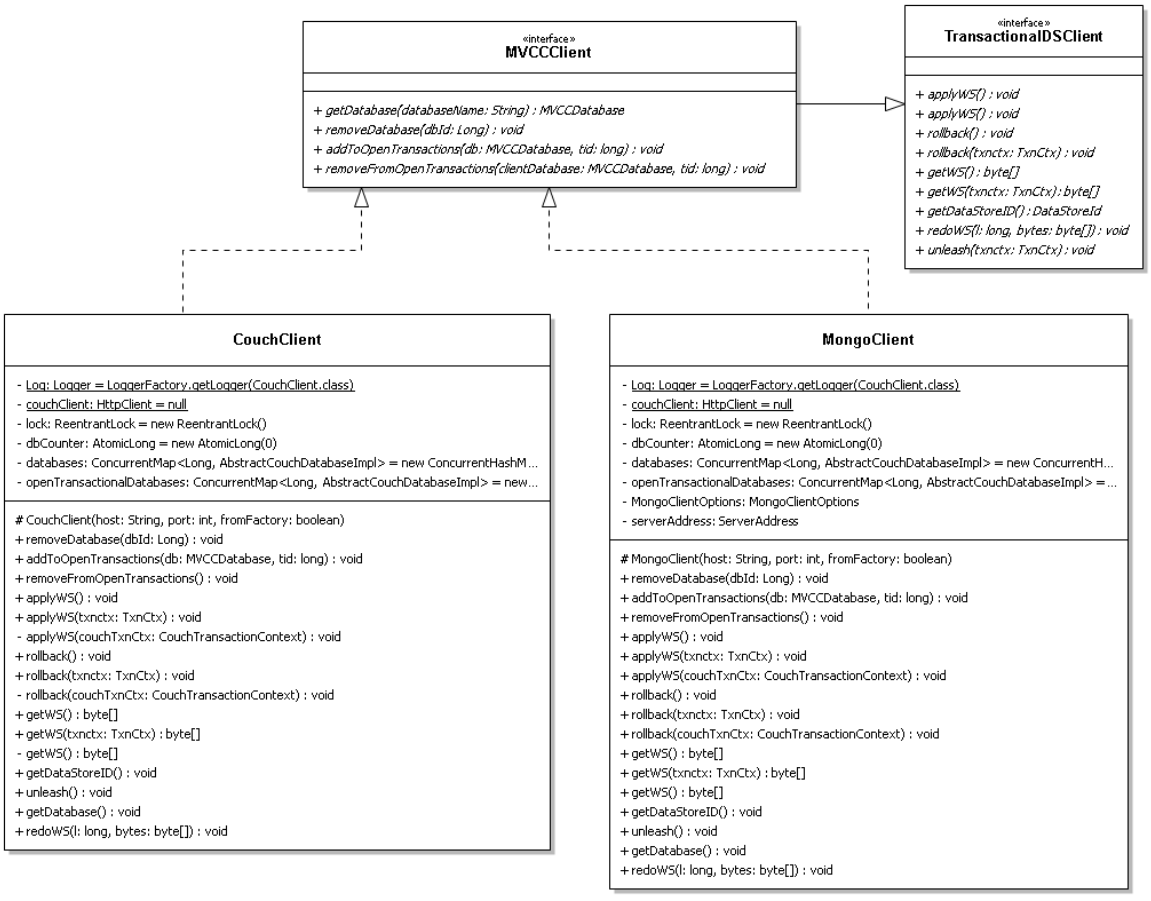
Στο ακόλουθο σχήμα παρουσιάζεται το διάγραμμα κλάσεων που αφορά τους client του κάθε συστήματος βάσης δεδομένων, καθώς και τη συσχέτιση τους με τη διεπαφή που υλοποιούν.



Σχήμα 4.2: Διάγραμμα Κλάσεων FactoryClient

Όπως αναλύθηκε στο προηγούμενο κεφάλαιο, στόχος είναι η δημιουργία ακριβώς ενός client για κάθε σύστημα βάσεων δεδομένων και για αυτό είναι επιθυμητό να ακολουθηθεί το σχεδιαστικό μοτίβο Singleton. Προκειμένου να υλοποιηθεί η ιδέα αυτή χρησιμοποιείται η κλάση MVCCFactoryClient για τη δημιουργία ενός στιγμιότυπου των κλάσεων εφόσον δεν υπάρχει. Αυτό σημαίνει πως ο MVCCFactoryClient διατηρεί ένα πεδίο Client τύπου MVCCClient προκειμένου να διασφαλίζει πως δεν θα δημιουργηθεί δεύτερο στιγμιότυπο. Ο προγραμματιστής-χρήστης έχει τη δυνατότητα να επιλέξει αν ο client που θα δημιουργηθεί θα αφορά τη MongoDB ή την CouchDB. Η επιλογή αυτή γίνεται με χρήση ενός Enumeration με όνομα DBType που περιλαμβάνει τις επιλογές MongoDB και CouchDB και η επιλογή αυτή δίνεται ως όρισμα στη μέθοδο createClient. Επίσης, υπάρχει η μέθοδος getMVCCClient προκειμένου να επιστρέφεται ο Client που έχει δημιουργθεί.

Συνεχίζοντας, παρατηρείται πως υπάρχουν οι κλάσεις CouchClient και MongoClient, οι οποίες υλοποιούν (implements) τη διεπαφή MVCCClient. Η διεπαφή MVCCClient έχει το ρόλο του Client που χρησιμοποιείται προκειμένου να δημιουργούνται οι συνδέσεις με το σύστημα των βάσεων δεδομένων και η διαχείριση τους, όπως ακριβώς συμβαίνει με τη MongoClient που παρουσιάσθηκε στο προηγούμενο κεφάλαιο. Επίσης επεκτείνει τη διεπαφή TransactionalDSClient η οποία, όπως αναλύθηκε και σε προηγούμενο κεφάλαιο, περιέχει μεθόδους διαχείρισης των ιδιωτικών εγγραφών μιας συναλλαγής καθώς και μεθόδους που αφορούν την ολοκλήρωση(commit) ή ακύρωση(rollback) της συναλλαγής Η κλάση MongoClient είναι όμοια με την ομώνυμη κλάση που παρέχεται από το MongoDB Java Driver, με τη διαφοροποίηση ότι έχει επεκταθεί προκειμένου να περιλαμβάνει τις απαραίτητες λειτουργίες στα πλαίσια Coherent PaaS. Η κλάση CouchClient δημιουργήθηκε προκειμένου να έχει τον ίδιο ρόλο με τη MongoClient για το σύστημα της CouchDB. Στο διάγραμμα που ακολουθεί παρουσιάζονται οι μέθοδοι της διεπαφής και των κλάσεων. Βλέπουμε πως η διεπαφή MVCCClient περιλαμβάνει τέσσερις μεθόδους, οι οποίες ασφαλώς υλοποιούνται κατάλληλα από τους CouchClient και MongoClient.



Σχήμα 4.3: Διάγραμμα Κλάσεων Clients

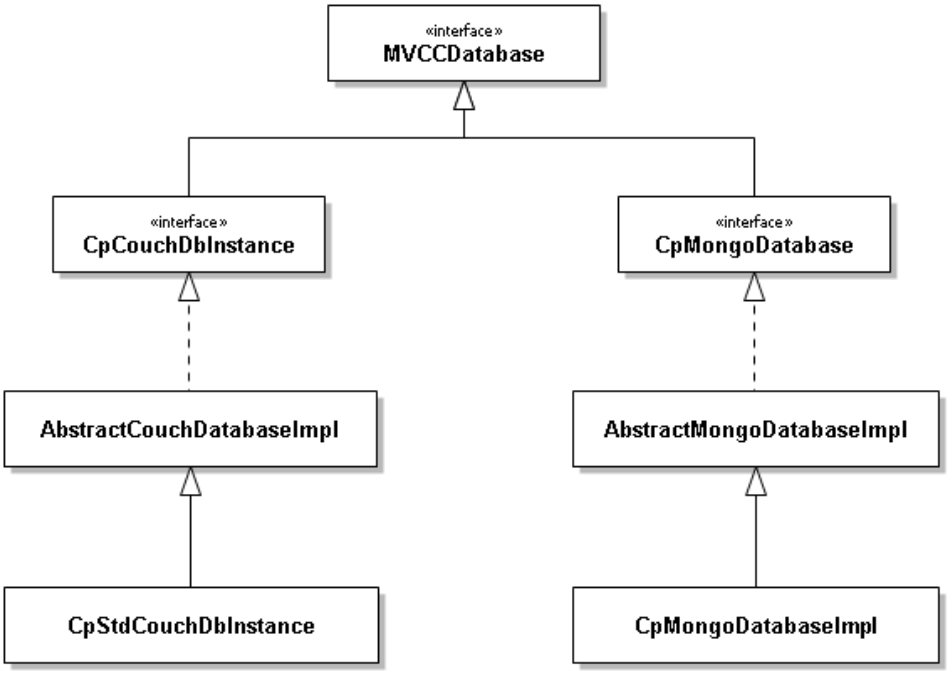
getDatabase: Η μέθοδος getDatabase λαμβάνει ως όρισμα το όνομα της βάσης δεδομένων του συστήματος στην οποία επιθυμούμε να έχουμε πρόσβαση και η μέθοδος επιστρέφει το στιγμιότυπο της βάσης αυτής, το οποίο φυσικά είναι τύπου MVCCDatabase.

removeDatabase: Η μέθοδος removeDatabase δέχεται ως όρισμα το αριθμητικό αναγνωριστικό μιας βάσης δεδομένων με στόχο να την αφαιρέσει από το σύστημα.

addToOpenTransactions: Όπως μπορούμε να δούμε στα πεδία των MongoClient και CouchClient, υπάρχει μία δομή τύπου ConcurrentMap η οποία διατηρεί όλες τις βάσεις δεδομένων που είναι διαθέσιμες για συμμετοχή σε συναλλαγές. Η συγκεκριμένη μέθοδος λοιπόν δέχεται ως ορίσματα ένα αριθμητικό αναγνωριστικό και το όνομα μιας βάσης δεδομένων και την τοποθετεί στη δομή που αναφέρθηκε, με κλειδί το αναγνωριστικό.

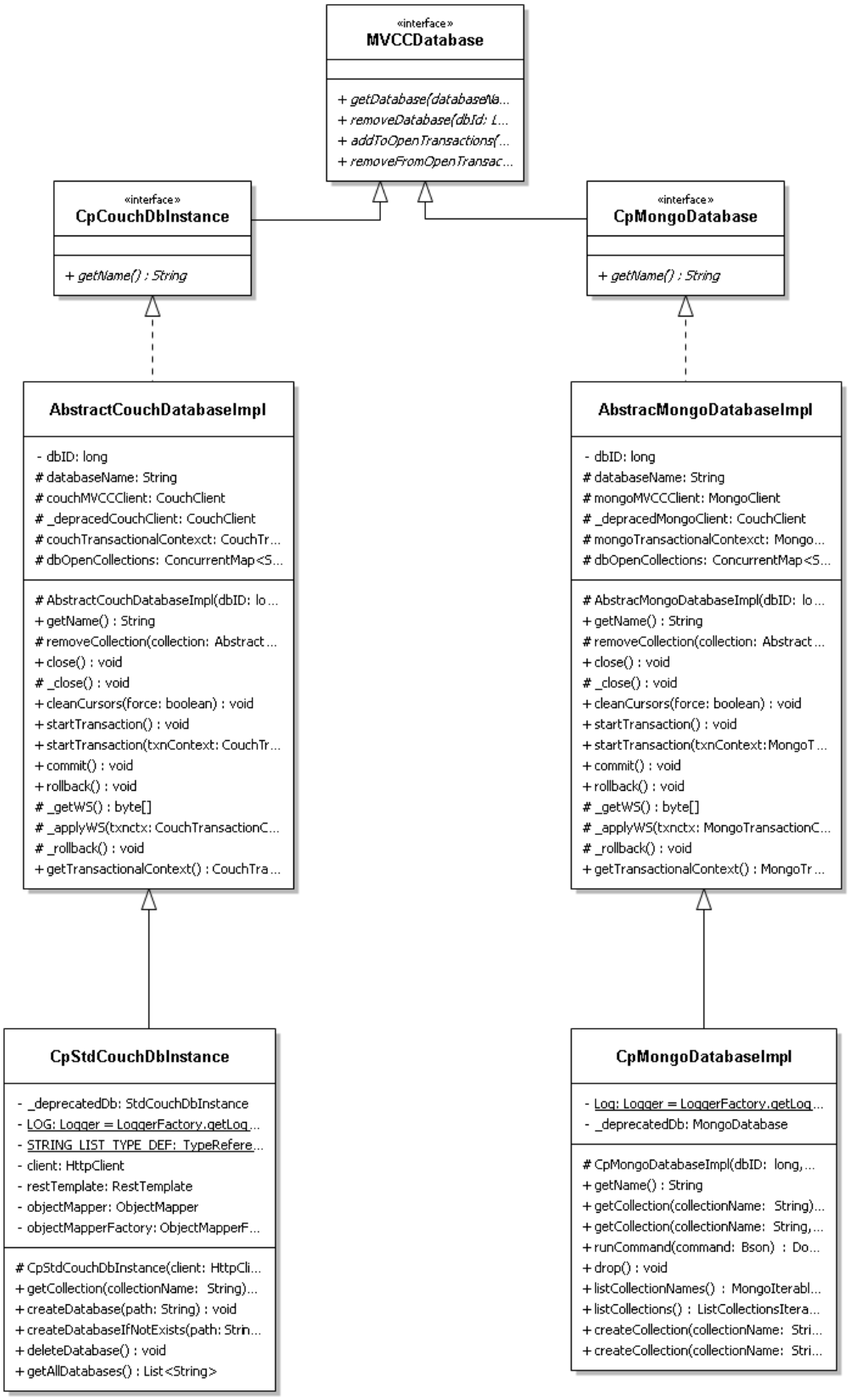
removeFromOpenTransaction: Όπως είναι φανερό, στόχος της συγκεκριμένης μεθόδου είναι η αφαίρεση μιας συγκεκριμένης βάσης δεδομένων από τη δομή που διατηρεί τις διαθέσιμες βάσεις δεδομένων.

Συνεχίζεται η ανάλυση της σχεδίασης με το επόμενο επίπεδο που περιλαμβάνει τη διεπαφή MVCCDatabase. Όπως αναφέρθηκε προηγουμένως, η διεπαφή MVCCDatabase απεικονίζει μία βάση δεδομένων ενός από τα δύο συστήματα (MongoDB ή CouchDB) και έχει ως στόχο να παρέχει όμοιο τρόπο διαχείρισης μιας βάσης δεδομένων και για τα δύο συστήματα.



Σχήμα 4.4: Διάγραμμα Κλάσεων MVCCDatabases

Από το σχήμα φαίνεται καθαρά πως η διεπαφή MVCCDatabase επεκτείνεται από τις διεπαφές CpMongoDatabase και CpCouchDbInstance. Ο MongoDB Java Driver παρέχει μία διεπαφή με όνομα MongoDatabase που έχει ως στόχο τη δήλωση συγκεκριμένων μεθόδων για τη διαχείριση της βάσης δεδομένων. Η υλοποίηση της διεπαφής αυτής στο MongoDB Java Driver πραγματοποιείται από την κλάση MongoDatabaseImpl. Στα πλαίσια του Coherent PaaS δημιουργήθηκαν η διεπαφή CpMongoDatabase, το αρχικό Cp από το Coherent PaaS, και η κλάση CpMongoDatabaseImpl, οι οποίες έχουν όλες τις μεθόδους και τα πεδία των MongoDatabase και MongoDatabaseImpl αλλά και κάποιες επιπλέον που αφορούν την υλοποίηση του MVCC. Από τον Ektorp (Java Driver για την CouchDB) παρέχονται η διεπαφή CouchDbInstance και η κλάση StdCouchDbInstance οι οποίες έχουν ανάλογο ρόλο με τις MongoDatabase και MongoDatabaseImpl. Έτσι, κατά όμοιο τρόπο με τη MongoDB δημιουργήθηκαν η διεπαφή CpCouchDbInstance και η κλάση CpStdCouchDbInstance. Στο σημείο αυτό πρέπει να τονιστεί πως παρά τη διαφορετική δομή ως προς τις βάσεις και τις συλλογές των δύο βάσεων, οι κλάσεις που παρέχονται από του οδηγούς (drivers) ταιριάζουν απόλυτα ως προς τη δόμηση και συνεπώς διευκολύνεται η μετατροπή της CouchDB έτσι ώστε να προσαρμοστεί στη λογική της MongoDB. Επιπλέον, ανάμεσα στις διεπαφές (CpMongoDatabase και CpCouchDbInstance) και τις κλάσεις (CpMongoDatabaseImpl και CpStdCouchDbInstance) δημιουργήθηκε ένα επιπλέον επίπεδο από αφηρημένες κλάσεις (AbstractMongoDatabaseImpl και AbstractCouchDatabaseImpl). Το επίπεδο αυτό δημιουργήθηκε στα πλαίσια του Coherent PaaS και για αυτό ακολουθήθηκε στην υλοποίηση της συγκεκριμένης διπλωματικής. Στόχος είναι να μην περιέχονται όλες οι απαραίτητες μέθοδοι στις τελικές κλάσεις, αλλά κάποιες από αυτές να υλοποιούνται στο ενδιάμεσο επίπεδο.



Σχήμα 4.5: Διάγραμμα Κλάσεων Databases

startTransaction: Η συγκεκριμένη μέθοδος έχει στόχο να σηματοδοτήσει την έναρξη μιας νέας συναλλαγής. Κατά αυτό τον τρόπο, πραγματοποιείται επικοινωνία με τον Holistic Transaction Manager προκειμένου να αποκτηθούν οι κατάλληλες χροσνοσφραγίδες και με βάση αυτές αρχικοποιείται ένα νέο στιγμιότυπο τύπου TransactionContext. Η διεπαφή TransactionContext υλοποιείται από τις κλάσεις MongoTransactionContext και CouchTransactionContext, ανάλογα με το σύστημα το οποίο αναφερόμαστε, και είναι μία κλάση που διατηρεί όλες τις απαραίτητες μεθόδους και πεδία που αφορούν τις το χρονικό πλαίσιο και τις χρονοσφραγίδες για την εκάστοτε συναλλαγή. Συγκρεκριμένα, περιέχει της μεθόδους getters και setters για τα tid, startTimestamp, commitTimestamp καθώς και μία μέθοδο που πραγματοποιεί έλεγχο για πιθανές συγκρούσεις της συγκεκριμένης συναλλαγής με άλλες ταυτόχρονες. Η startTransaction υλοποιείται από τις AbstractCouchDatabaseImpl και AbstractMongoDatabaseImpl.

getCollection: Η μέθοδος αυτή δέχεται ως όρισμα ένα αλφαριθμητικό (String) και επιστρέφει τη συλλογή με όνομα ίδιο με το αλφαριθμητικό. Η μέθοδος αυτή υλοποιείται από τις CpMongoDatabaseImpl και CpStdCouchDbInstance.

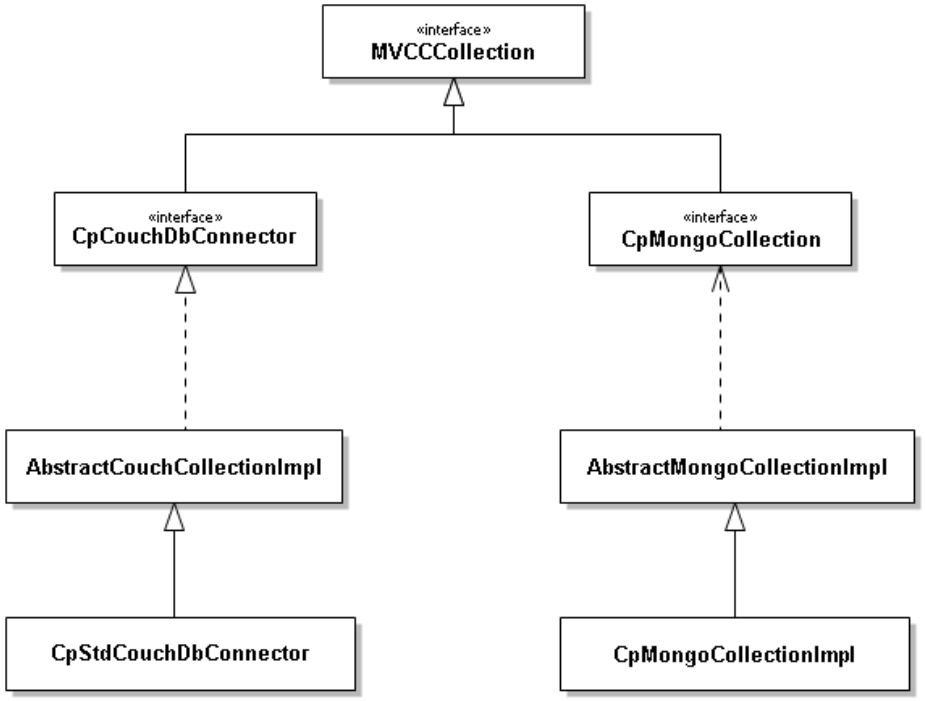
rollback: Η μέθοδος αυτή υλοποιείται από τις AbstractCouchDatabaseImpl και AbstractMongoDatabaseImpl και πραγματοποιεί τη λειτουργία της ακύρωσης μιας ολόκληρης συναλλαγής και κατά συνέπεια την ακύρωση όλων των αλλαγών που πραγματοποιήθηκαν στο σύστημα της βάσης δεδομένων από τη συγκεκριμένη συναλλαγή.

close: Η μέθοδος αυτή έχει στόχο να κλείσει τη συγκεκριμένη βάση δεδομένων. Κατά συνέπεια πρέπει η συγκεκριμένη βάση δεδομένων να διαγραφεί από τις βάσεις δεδομένων για τις οποίες διατηρεί συνδέσεις ο εκάστοτε Client. Η υλοποίηση πραγματοποιείται από τις AbstractCouchDatabaseImpl και AbstractMongoDatabaseImpl.

commit: Η διαδικασία του commit έχει αναλυθεί σε προηγούμενα κεφάλαια. Αφορά τη διαδικασία κατά την οποία όλες οι αλλαγές και οι ενέργειες που πραγματοποιήθηκαν από τη συγκεκριμένη συναλλαγή μονιμοποιούνται στο σύστημα βάσεων δεδομένων που χρησιμοποιείται. Ουσιαστικά, ενημερώνεται ο HTM πως η συναλλαγή επιθυμεί να ολοκληρωθεί. Η υλοποίηση της μεθόδου αυτής γίνεται από τις AbstractCouchDatabaseImpl και AbstractMongoDatabaseImpl.

getTransactionalContext: Είναι η τελευταία μέθοδος που δηλώνεται από τη συγκεκριμένη διεπαφή και έχει στόχο να δώσει πρόσβαση στο χρήστη-προγραμματιστή στο TransactionContext της συναλλαγής, όπως αυτό αναλύθηκε κατά την περιγραφή της μεθόδου startTransaction.

Το επόμενο βήμα αφορά την ανάλυση της διεπαφής MVCCCollection. Η συγκεκριμένη διεπαφή χρησιμοποιείται για την απεικόνιση μιας συλλογής που ανήκει σε κάποια βάση δεδομένων του συστήματος. Ασφαλώς, εξηγήθηκε ότι η αρχιτεκτονική της CouchDB δεν περιλαμβάνει συλλογές όπως η MongoDB. Ωστόσο, η ενιαία διεπαφή που υλοποιείται δίνει αυτή την απεικόνιση στο χρηστή-προγραμματιστή. Αυτή η σχεδιαστική απόφαση διευκολύνεται σημαντικά από την κλάση που παρέχει ο Ektorp, παρά το ότι δεν υπάρχουν συλλογές στο σύστημα της CouchDB.



Σχήμα 4.6: Διάγραμμα Κλάσεων MVCCCollection

Παρατηρώντας το παραπάνω διάγραμμα κλάσεων, φαίνεται πως ακολουθείται το ίδιο μοτίβο με αυτό που χρησιμοποιήθηκε για την MVCCDatabase. Συγκεκριμένα, βλέπουμε πως η διεπαφή MVCCCollection επεκτείνεται (extends) από τις διεπαφές CpMongoCollection και CpCouchDbConnector. Ο MongoDB Java Driver παρέχει τη διεπαφή MongoCollection και την κλάση MongoCollectionImpl, η οποία υλοποιεί την προαναφερθείσα διεπαφή, και στόχος τους είναι η διαχείριση της ανάλογης συλλογής της MongoDB. Ο Ektorp της CouchDB παρέχει τη διεπαφή CouchDbConnector και την κλάση StdCouchDbConnector που υλοποιεί την διεπάφη. Κατά όμοιο τρόπο με την περίτπωση της MVCCDatabase, δημιουργήθηκαν οι διεπαφές CpMongoCollection και CpCouchDbConnector καθώς και οι κλάσεις CpMongoCollectionImpl και CpStdCouchDbConnector, οι οποίες αποτελούν επεκτάσεις των διεπαφών και των κλάσεων που παρέχονται από τους αντίστοιχους οδηγούς. Επίσης, υπάρχει

το ενδιάμεσο επίπεδο που υλοποιείται από τις αφηρημένες κλάσεις AbstractCouchCollection και AbstractMongoCollection. Στο διάγραμμα κλάσεων της επόμενης σελίδας παρουσιάζονται οι μέθοδοι που δηλώνονται στη διεπαφή MVCCCollection καθώς και οι βασικές μέθοδοι και πεδία των διεπαφών και των κλάσεων που την κληρονομούν. Οι μέθοδοι της MVCCCollection είναι ουσιαστικά οι βασικές μέθοδοι CRUD και το σκεπτικό τους θα αναλυθεί ακολούθως.

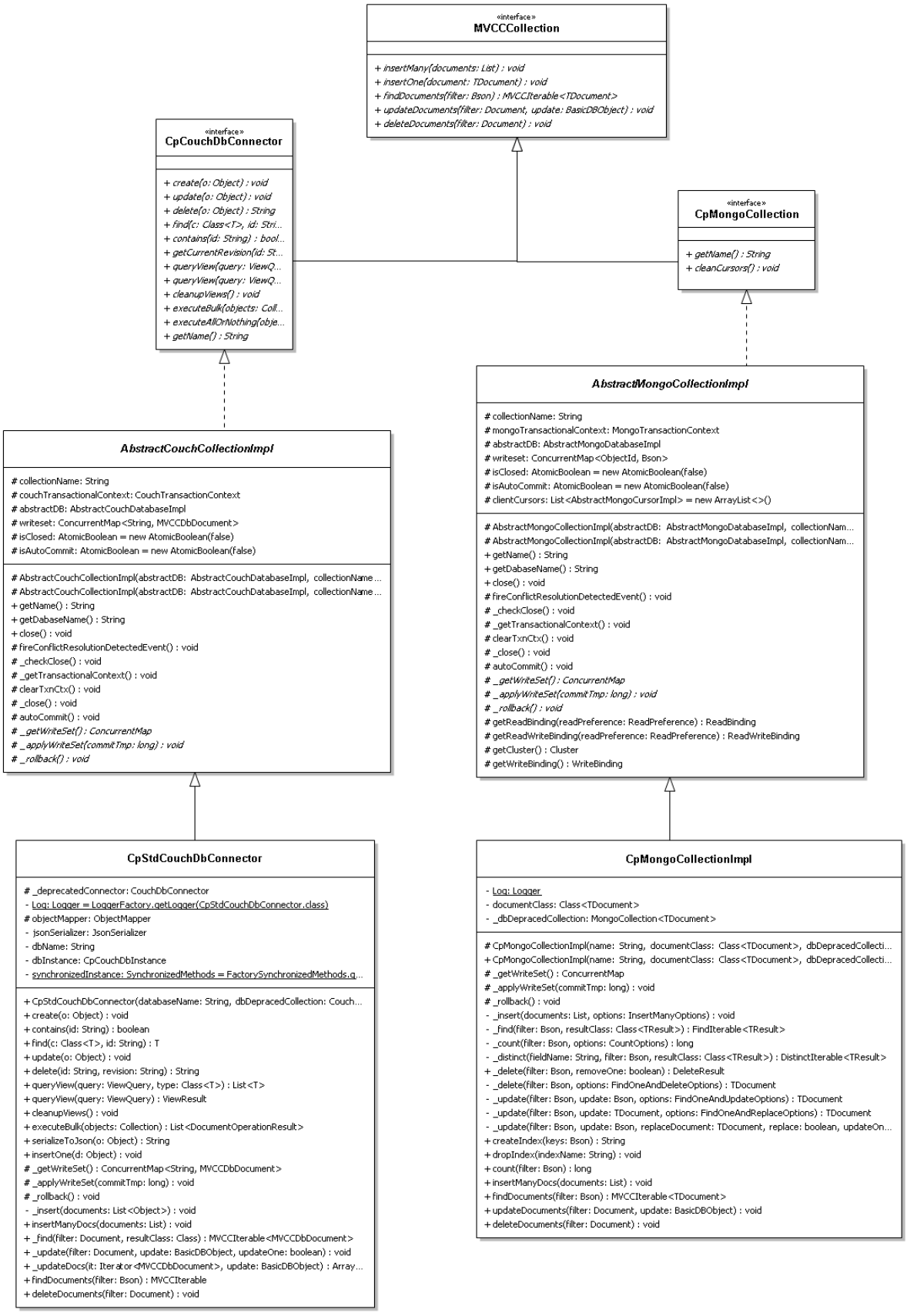
insertManyDocs: Η συγκεκριμένη μέθοδος έχει στόχο της εισαγωγή νέων αντικειμένων στη συλλογή της βάσης δεδομένων και κατά συνέπεια τη δημιουργία νέων εγγραφών. Ουσιαστικά, επειδή η εισαγωγή νέων εγγραφών στη βάση δεδομένων γίνεται με διαφορετικό τρόπο στα δύο συστήματα, η μέθοδος αυτή καλεί τις κατάλληλες μεθόδους του κάθε οδηγού. Η υλοποίησή της γίνεται στις κλάσεις CpStdCouchDbConnector και CpMongoCollectionImpl.

insertOne: Η έμπνευση της μεθόδου αυτής προέρχεται από το γεγονός ότι και τα δύο συστήματα βάσεων δεδομένων παρέχουν εντολές για την εισαγωγή μιας νέας μόνο εγγραφής. Ομοίως, η insertOne εισάγει μία νέα εγγραφή και υλοποιείται από τις κλάσεις CpStdCouchDbConnector και CpMongoCollectionImpl.

findDocuments: Η μέθοδος αυτή έχει στόχο την υλοποίηση της λειτουργίας Read, δηλαδή της εύρεσης εγγραφών που ικανοποιούν συγκεκριμένα κριτήρια. Κατά αυτό τον τρόπο δηλώνεται ότι επιστρέφει αντικείμενο της διεπαφής MVCCIterable, η οποία θα παρουσιασθεί παρακάτω. Το όρισμα που δέχεται είναι ένα αντικείμενο που περιλαμβάνει τα κριτήρια αναζήτησης. Η υλοποίηση της μεθόδου, όπως και των προηγούμενων γίνεται από τις κλάσεις CpStdCouchDbConnector και CpMongoCollectionImpl.

updateDocuments: Η μέθοδος αυτή υλοποιεί τη λειτουργία update από τις CRUD λειτουργίες, πράγμα που σημαίνει πως πραγματοποιεί ενημερώσεις εγγραφών βάσει συγκεκριμένων κριτηρίων. Η μέθοδος δέχεται δύο ορίσματα, το πρώτο αφορά τα κριτήρια αναζήτησης και το δεύτερο τις ενημερώσεις που θα γίνουν στα πεδία των εγγραφών. Η υλοποίηση της εγγραφής γίνεται και πάλι στις κλάσεις CpStdCouchDbConnector και CpMongoCollectionImpl.

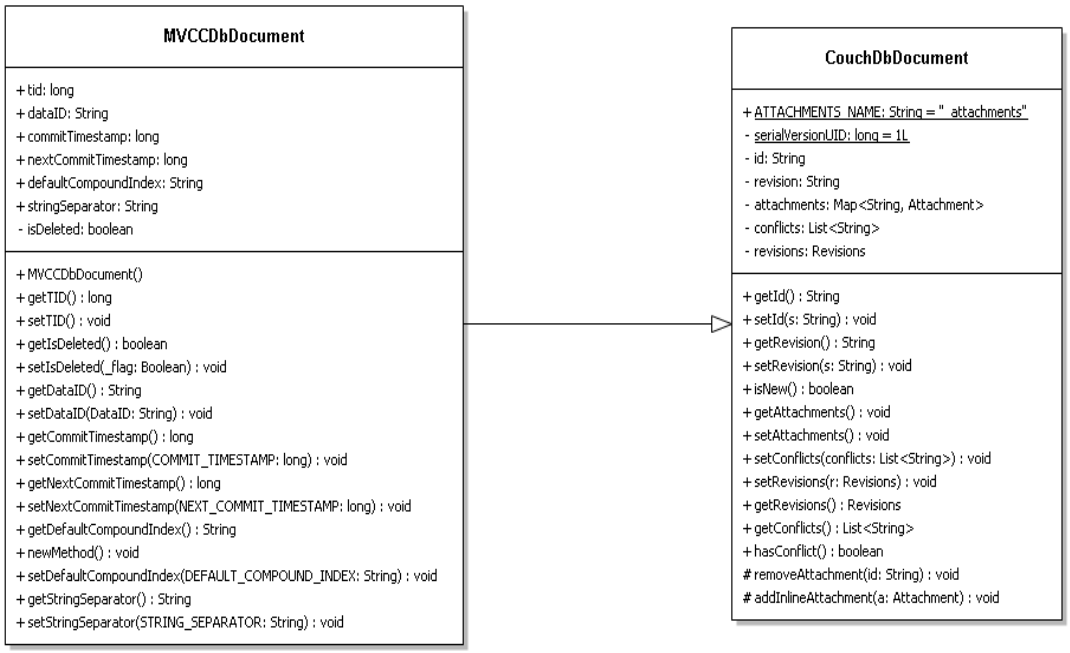
deleteDocuments: Η μέθοδος αυτή διαγράφει όλες τις εγγραφές που ικανοποιούν τα κριτήρια που περιέχονται στο αντικείμενο που περνιέται ως παράμετρος. Όπως με όλες τις υπόλοιπες μεθόδους, η μέθοδος αυτή υλοποιείται από τις κλάσεις CpStdCouchDbConnector και CpMongoCollectionImpl.



Σχήμα 4.7: Διάγραμμα Κλάσεων Collections

Έχοντας παρουσιάσει και αναλύσει τα διαγράμματα κλάσεων των κυριότερων στοιχείων που συγκροτούν την ενιαία διεπαφή, θα αναλυθούν δύο ακόμη στοιχεία που αποτελούν σχεδιαστικές αποφάσεις της υλοποίησης που ακολουθήθηκε.

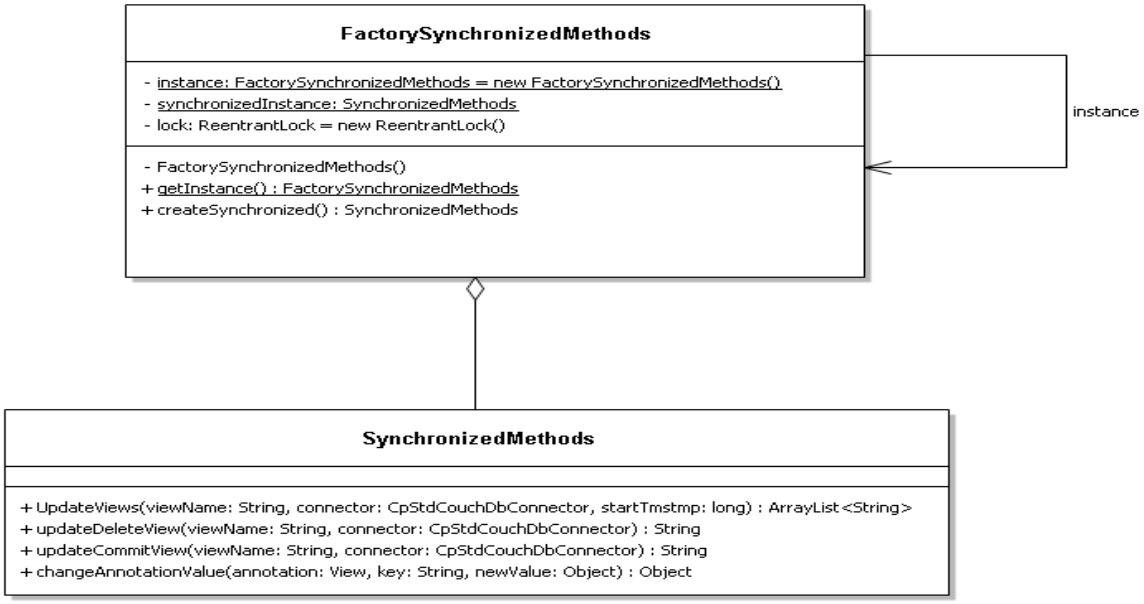
Αρχικά αναφερόμαστε στην κλάση MVCCDbDocument. Αναφέρθηκε σε προηγούμενο κεφάλαιο η διαφοροποίηση που παρουσιάζεται μεταξύ των οδηγών των MongoDB και CouchDB ως προς τις κλάσεις και τον τρόπο απεικόνισης των εγγραφών της βάσης δεδομένων, ενώ τονίστηκε ότι επιλέχθηκε να εφαρμοστεί η λογική της CouchDB. Κατά αυτό τον τρόπο, ο χρήστης προγραμματιστής θα πρέπει να δημιουργεί δικές του κλάσεις μορφής POJO προκειμένου να εισάγει και να διαχειριστεί εγγραφής της βάσης δεδομένων. Ωστόσο, κάθε εγγραφή επιβάλλεται να έχει συγκεκριμένα δεδομένα που αφορούν την υλοποίηση του MVCC. Για τη διασφάλιση της συγκεκριμένη απαίτησης, επιλέχθηκε να δημιουργηθεί η κλάση MVCCDbDocument και την οποία θα τροποποιεί ο χρήστης κατάλληλα προκειμένου να περιέχει όλα τα απαραίτητα πεδία και μεθόδους getters-setters. Η κλάση MVCCDbDocument πρέπει να κληρονομεί την κλάση CouchDbDocument, η οποία παρέχεται από τον Ektorp, ώστε να κληρονομεί όλα τα πεδία που πρέπει να διατηρεί μία κλάση για την εισαγωγή της στην CouchDB. Για τη MongoDB δεν μας επηρεάζουν τα συγκεκριμένα πεδία. Είναι σημαντικό να σημειωθεί πως όσα πεδία επιθυμεί να προσθέτει ο χρήστης και πρόκειται να είναι αριθμοί οφείλουν να είναι αντικείμενα της κλάσης Integer. Η σύμβαση αυτή οφείλεται στον τρόπο με τον οποίο υλοποιήθηκε η λειτουργία της ενημέρωσης. Το διάγραμμα κλάσεων παρουσιάζεται στο ακόλουθο σχήμα.



Σχήμα 4.8: Διάγραμμα Κλάσεων MVCCDbDocument

Ακόμη, δημιουργήθηκε μία ακόμη διεπαφή με το όνομα MVCCIterable. Όπως αναφέρθηκε στην ανάλυση της υλοποίησης για τη MongoDB, η λειτουργία της ανάγνωσης (read) επιστρέφει ένα στιγμιότυπο της διεπαφής FindIterable η οποία επεκτείνει τη διεπαφή MongoIterable, η οποία με τη σειρά της επεκτείνει τη διεπαφή Iterable. Προκειμένου λοιπόν να υπάρχει κοινή λειτουργία find και για τις δύο βάσεις δεδομένων, επιλέχθηκε η λειτουργία αυτή να επιστρέφει ένα στιγμιότυπο μιας ανάλογης διεπαφής. Για το λόγο αυτό δημιουργήθηκε η MVCCIterable, η οποία απλώς επεκτείνει τη διεπαφή Iterable. Κατά αυτό τον τρόπο δίνεται η δυνατότητα να χρησιμοποιηθεί ένας iterator και συνεπώς να έχει πρόσβαση ο προγραμματιστής-χρήστης, μέσω μιας δομής συνδεδεμένης λίστας, στα στοιχεία που επιστρέφονται ως αποτελέσματα της αναζήτησης.

Τέλος, απαιτείται να γίνει ειδική αναφορά σε δύο κλάσεις ειδικού σκοπού οι οποίες συνδέονται άμεσα μεταξύ τους και παρουσιάζονται στο ακόλουθο διάγραμμα κλάσεων.



Σχήμα 4.9: Διάγραμμα Κλάσεων SynchronizedMethods

Όπως γίνεται φανερό από το διάγραμμα κλάσεων, η κλάση FactorySynchronizeMethods παρέχει το μόνο τρόπο δημιουργίας ενός στιγμιότυπου της κλάσης SynchronizedMethods υλοποιώντας έτσι το σχεδιαστικό μοτίβο Singleton. Επίσης είναι εμφανές πως υπάρχει ένα μοναδικό στιγμιότυπο της κλάσης FactorySynchronizedMethods για όλα τα πιθανά νήματα εκτέλεσης. Η createSynchronized λειτουργεί με τέτοιο τρόπο ώστε να δημιουργείται μόλις ένα στιγμιότυπο της SynchronizedMethods και άρα όλα τα νήματα να έχουν πρόσβαση σε αυτό το στιγμιότυπο. Η κλάση SynchronizedMethods περιέχει 3 μεθόδους οι οποίες είναι δηλωμένες ως synchronized. Συνεπώς η χρήση των μεθόδων αυτών επιβάλει απόλυτο συγχρονισμό για όλα τα νήματα. Η χρησιμότητα και η λειτουργία των συγκεκριμένων μεθόδων θα αναλυθεί στα

πλαίσια των υλοποιήσεων των CRUD λειτουργιών. Η μέθοδος changeAnnotationValue είναι αυτή που δέχεται ως ορίσματα το View που πρόκειται να τροποποιηθεί, το πεδίο του θα επηρεαστεί και τη νέα τιμή που θα λάβει και με χρήση της τεχνικής reflection πραγματοποιεί την τροποποίηση. Η μέθοδος UpdateViews είναι υπεύθυνη για την κατάλληλη τροποποίηση του View, που έχει γράψει ο χρήστης στο FindRepository και πρόκειται να χρησιμοποιηθεί για την εύρεση των εγγραφών που θα ενημερωθούν, κατά τρόπο ώστε να τηρεί τους περιορισμούς που επιβάλλει η τεχνική MVCC. Η UpdateViews επιστρέφει μία λίστα με δύο αλφαριθμητικά που αποτελούν τα ονόματα των views για τις ιδιωτικές και τις δημόσιες εκδόσεις. Η updateDelete χρησιμοποιείται κατά τη λειτουργία της διαγραφής και έχει στόχο να βρει όλες τις ιδιωτικές εκδόσεις. Τέλος, υπάρχει η updateCommitView και χρησιμοποιείται για την τροποποίηση των Views του CommitRepository. Στο CommitRepository υπάρχει το View commit που στοχεύει στην ανεύρεση των ιδιωτικών εκδόσεων και το View updateOlderVersions που στοχεύει στην ανεύρεση των τελευταίων εκδόσεων των εγγραφών που πρόκειται να ενημερωθούν και άρα πρέπει να λάβουν την κατάλληλη τιμή τα πεδία nextCmtTmstmp.

5

Υλοποίηση

Στόχος του συγκεκριμένου κεφαλαίου είναι η περιγραφή των εργαλείων που χρησιμοποιήθηκαν και η εκτενέστερη ανάλυση της υλοποίησης. Για την υλοποίηση της συγκεκριμένης διπλωματικής έγινε χρήση της γλώσσας προγραματισμού Java[16] καθώς και του εργαλείου Maven[25] [26], όπως αυτό προσφέρεται για διάφορα περιβάλλοντας ανάπτυξης λογισμικού όπως το Netbeans. Επίσης, χρησιμοποιήθηκαν συγκεκριμένες τεχνικές που προσφέρει η Java όπως οι έννοιες του Reflection και της σειριοποίησης κλάσεων (class serialization). Στη συνέχεια, θα αναλυθεί ο τρόπος με τον οποίο υλοποιήθηκαν οι CRUD λειτουργίες αλλά και οι λειτουργίες commit και rollback στην CouchDB.

5.1 *Προγραμματιστικά Εργαλεία*

5.1.1 Η γλώσσα προγραμματισμού Java

Η Java είναι μία αντικειμενοστραφής γλώσσα προγραμματισμού γενικού σκοπού. Δημιουργήθηκε και σχεδιάστηκε από την εταιρεία Sun Microsystems και κυκλοφόρησε για πρώτη φορά το 1995. Είναι μία από τις πιο διαδεδομένες και δημοφιλείς γλώσσες προγραμματισμού καθώς προσφέρει σύνταξη παρόμοια με τις γλώσσες προγραμματισμού C και C++ και απλούστερο μοντέλο. Είναι μία γλώσσα υψηλού επιπέδου προσφέροντας μεγάλες διευκολύνσεις στο προγραμματιστή, χωρίς όμως τις δυνατότητες εντολών και λειτουργιών χαμηλότερου επιπέδου. Στόχος της Java, όπως διατυπώθηκε από το δημιουργό της James Gosling, είναι η δυνατότητα των προγραμμάτων Java να λειτουργούν σε όλα τα μηχανήματα, ανεξάρτητα από το σε ποιο έχουν δημιουργηθεί, “Write Once, Run Anywhere”. Για το λόγο αυτό η Java παρέχει εικονική μηχανή προκειμένου τα προγράμματα να μεταγλωττίζονται σε

δυαδικό κώδικα και να εκτελούνται στη συγκεκριμένη εικονική μηχανή, ανεξαρτήτως του υλικού που εγκαθίσταται. [10]

Οι πρώτοι μεταγλωττιστές της Java, οι εικονικές μηχανές και οι βιβλιοθήκες κλάσεων αναπτύχθηκαν από τη Sun Microsystems το 1995. Ωστόσο, από το 2007 και σε συνεργασία με τις προδιαγραφές της Java Community Process, η Sun έκανε διαθέσιμο το μεγαλύτερο μέρος της τεχνολογίας της Java ως ελεύθερο λογισμικό υπό την άδεια General Public License (GNU).

Οι κύριοι στόχοι κατά τη δημιουργία της Java ήταν οι εξής:

- Η υιοθέτηση όλων των αρχών του αντικειμενοστραφούς προγραμματισμού.
- Η δυνατότητα να εκτελείται η ίδια εφαρμογή σε οποιαδήποτε αρχιτεκτονική υπολογιστικού συστήματος.
- Η ύπαρξη βιβλιοθηκών κλάσεων για την υποστήριξη χρήσης δικτύων σε ικανοποιητικό βαθμό.
- Η κατάλληλη σχεδίαση της Java ώστε να επιτρέπει την εκτέλεση εφαρμογών από απομακρυσμένες πηγές με ασφάλεια.
- Η φιλικότητα προς το χρήστη υιοθετώντας όλα τα θετικά στοιχεία των υπόλοιπων γλωσσών προγραμματισμού.

Με βάση τα παραπάνω χαρακτηριστικά και ως μία γλώσσα αντικειμενοστραφούς προγραμματισμού, η Java παρέχει μεγάλη απλότητα στο σχεδιασμό και διαχωρίζει τον κώδικα σε ξεχωριστές και αυτόνομες μονάδες προσφέροντας έτσι τη δυνατότητα μικρών αλλαγών στον κώδικα του συστήματος σε πολύ σύντομο χρονικό διάστημα χωρίς να επηρεάζεται η συνολική λειτουργία του έργου. Επίσης, προσφέρεται η δυνατότητα επέκτασης του κώδικα προσθέτοντας νέες λειτουργίες, χωρίς ιδιαίτερες δυσκολίες, η δυνατότητα επαναχρησιμοποίησης κλάσεων και μεθόδων σε διαφορετικά προγράμματα καθώς και η ευκολία στον εντοπισμό και τη διόρθωση των σφαλμάτων . Η παραπάνω ανάλυση καθιστά σαφές γιατί η Java είναι μία από τις κύριες γλώσσες ανάπτυξης λογισμικού αλλά και γιατί επιλέχθηκε για την εκπόνηση της συγκεκριμένης διπλωματικής εργασίας.

Αντίστοιχες γλώσσες προγραμματισμού με τη Java είναι οι C++ και C#, η οποία αναπτύχθηκε από τη Microsoft στα πλαίσια της πλατφόρμας .NET. Ωστόσο, η ήδη υπάρχουσα υλοποίηση της επέκτασης της MongoDB σε Java Driver αλλά και το γεγονός ότι ο πιο διαδεδομένος οδηγός της CouchDB αφορούσε τη Java επέβαλλαν την επιλογή της Java ως γλώσσα υλοποίησης της συγκεκριμένης διπλωματικής εργασίας. Πιο συγκεκριμένα χρησιμοποιήθηκε η έκδοση 1.7.

[10] [16] http://java.sun.com/, [18] https://en.wikipedia.org/wiki/Java_(programming_language)

5.1.2 Maven Project

Το εργαλείο Maven σχεδιάστηκε και αναπτύχθηκε από την εταιρεία Apache Software Foundation. Στόχος του συγκεκριμένου εργαλείου είναι η εύκολη διαχείριση και ανάπτυξη προγραμμάτων λογισμικού και αποτελεί εργαλείο ανοιχτού κώδικα. Η ιδέα στην οποία βασίστηκε είναι η ανάπτυξη μέσω μοντέλων αντικειμένων και βοηθάει στο χτίσιμό, την τεκμηρίωση και τον έλεγχο ενός έργου μέσω μιας κεντρικής πληροφορίας. Είναι υλοποιημένο με τη γλώσσα προγραμματισμού Java και επιτρέπει την ανάπτυξη εφαρμογών Java.

Ένα από τα κυριότερα χαρακτηριστικά του συγκεκριμένου εργαλείου είναι η διατήρηση του βασικού κύκλου ζωής της ανάπτυξης ενός προγράμματος λογισμικού. Συνεπώς, περιλαμβάνει όλα τα στάδια ζωής, όπως το συντακτικό έλεγχο και τη μεταγλώττιση ως και την εγκατάσταση σε έναν εξυπηρετητή (server) και την τεκμηρίωση. Για κάθε πρόγραμμα διατηρείται ένα αρχείο με κατάληξη τη λέξη pom το οποίο περιέχει όλες απαραίτητες πληροφορίες για το συγκεκριμένο πρόγραμμα. Συγκεκριμένα, στο αρχείο pom ορίζονται το όνομα του προγράμματος και η έκδοση του, οι διάφορες εξωτερικές βιβλιοθήκες που χρησιμοποιούνται για να εκτελεστεί το είδος ελέγχου που θα πραγματοποιηθεί πριν την εγκατάσταση καθώς και άλλες πληροφορίες αναφορικά με το πρόγραμμα και τις εξαρτήσεις του. Επιπλέον, το Maven προσφέρει τη δυνατότητα της οργάνωσης ολόκληρου του έργου σε οντότητες λογισμικού, διαθέτοντας χαρακτηριστικά αντικειμενοστραφών μοντέλων, όπως κληρονομικότητα και πολυμορφισμό.[11]

5.2 *Υλοποίηση λειτουργικότητας*

Η υλοποίηση των λειτουργιών CRUD στην CouchDB κατά τέτοιο τρόπο ώστε να υλοποιούν Έλεγχο Ταυτοχρονισμού Πολλαπλών εκδόσεων βασίστηκε στις αντίστοιχες υλοποιήσεις που αφορούν τη MongoDB. Το ίδιο ισχύει και για λειτουργίες όπως το startTransaction, το rollback και το commit. Ωστόσο, οι διαφορές που εμφανίζουν τα δύο συστήματα μεταξύ τους επέφεραν και σημαντικές διαφοροποιήσεις στην υλοποίηση των λειτουργιών. Στην ενότητα αυτή θα γίνει εκτενής περιγραφή του τρόπου με τον οποίο υλοποιούνται όλες οι λειτουργιές και των μεθόδων που χρησιμοποιούνται για το σύστημα της CouchDB.

[11] [25] https://en.wikipedia.org/wiki/Apache_Maven, [26] https://maven.apache.org/

5.2.1 Λειτουργίες CRUD

Λειτουργία Δημιουργίας

Η λειτουργία της δημιουργίας αναφέρεται στην εισαγωγή μιας ή περισσοτέρων νέων εγγραφών στη βάση δεδομένων. Η λειτουργία αυτή υλοποιείται με τη μέθοδο insertManyDocs η οποία δηλώνεται στη διεπαφή MVCCCollection και υλοποιείται από τις κλάσεις CpStdCouchDbConnector και CpMongoCollectionImpl. Στη δήλωση της μεθόδου φαίνεται ότι η συγκεκριμένη μέθοδος είναι δημόσια και δέχεται ως παράμετρο μία λίστα που περιέχει τα αντικείμενα που θα εισαχθούν στη βάση. Όπως αναλύθηκε προηγουμένως, τα αντικείμενα αυτά είναι στιγμιότυπα της κλάσης MVCCDbDocument η οποία τροποποιείται κατάλληλα από το χρήστη-προγραμματιστή. Ακολουθώντας το μοτίβο που υπάρχει στη MongoDB, η μέθοδος insertManyDocs καλεί τη μέθοδο _insert, που υλοποιείται και αυτή στην κλάση CpStdCouchDbConnector, περνώντας ως παράμετρο τη λίστα με τα αντικείμενα. Η _insert αρχικά ελέγχει αν η συγκεκριμένη συλλογή είναι διαθέσιμη και ενεργή ή κλεισμένη. Ο έλεγχος πραγματοποιείται με χρήση της μεθόδου _checkClose, που υλοποιείται από την AbstractCouchCollectionImpl και ελέγχει αν το πεδίο της με όνομα isClosed έχει την τιμή true. Στη συνέχεια καλείται η μέθοδος _getTransactionalContext, η οποία υλοποιείται από την αφηρημένη κλάση AbstractCouchCollectionImpl. Η μέθοδος αυτή έχει στόχο να τοποθετήσει στη μεταβλητή couchTransactionalContext της AbstractCouchCollectionImpl το κατάλληλο TransactionContext που αφορά τη συγκεκριμένη συναλλαγή. Έτσι αρχικά ελέγχει αν η μεταβλητή couchTransactionalContext περιέχει τιμή null ή όχι. Αν όχι, τότε σημαίνει πως υπάρχει ήδη μία τρέχουσα συναλλαγή στην οποία υπάγεται η συγκεκριμένη λειτουργία create και συνεπώς αφήνει την τιμή της couchTransactionalContext αμετάβλητη. Διαφορετικά, επικοινωνεί με το Holistic Transaction Manager και αποκτά το κατάλληλο TransactionContext τοποθετώντας τη στη μεταβλητή couchTransactionalContext και ξεκινά μία νέα συναλλαγή καλώντας τη μέθοδο startTransaction της αφηρημένης κλάσης AbstractCouchDatabaseImpl. Εφόσον ολοκληρωθούν τα ανωτέρω χωρίς κανένα πρόβλημα (για παράδειγμα κάποιο exception) ξεκινά η διαδικασία της κατάλληλης τροποποίησης των αντικειμένων προκειμένου να εισαχθούν στη βάση. Αρχικά δημιουργείται μία νέα λίστα τύπου ArrayList η οποία θα περιέχει τα τροποποιημένα αντικείμενα προκειμένου αυτά να εισαχθούν στη βάση δεδομένων. Στη συνέχεια για κάθε ένα αντικείμενο ακολουθείται κοινή διαδικασία. Αρχικά ελέγχεται αν είναι στιγμιότυπο της κλάσης MVCCDbDocument ή άλλης κλάσης που την κληρονομεί και σε περίπτωση που δεν είναι ενεργοποιείται μία εξαίρεση (exception). Εφόσον όλα είναι καλά με τον παραπάνω έλεγχο, ελέγχεται το πεδίο _id. Σε περίπτωση που έχει δοθεί τιμή στο πεδίο από το χρήστη-προγραμματιστή, τότε η ίδια τιμή χρησιμοποιείται για το πεδίο dataID. Αν δεν έχει δοθεί τιμή στο πεδίο _id από τον προγραμματιστή, τότε αφήνεται να δοθεί η τιμή αυτή από το σύστημα της CouchDB και στο πεδίο dataID δίνεται μία τυχαία τιμή με χρήση της μεθόδου

UUID.randomUUID().toString(). Η UUID είναι μία κλάση της Java που διασφαλίζει τη απόδοση τυχαίων τιμών οι οποίες είναι μοναδικές ακόμα και για κατανεμημένα συστήματα. Η χρήση της μεθόδου toString() γίνεται καθώς το πεδίο dataID έχει οριστεί τύπου String. Στη συνέχεια δίνεται τιμή στο πεδίο tid με χρήση της μεθόδου getTid() που υλοποιείται από την couchTransactionalContext και ουσιαστικά αποδίδει ως αναγνωριστικό id της συναλλαγής η χρονική στιγμή έναρξης της συναλλαγής που περιλαμβάνει τη λειτουργία αυτή. Ακολούθως, δίνεται η τιμή -1 στα πεδία cmtTmstmp και nextCmtTmstmp. Τέλος, το αντικείμενο προστίθεται στη λίστα που δημιουργήθηκε για να περιέχει όλα τα τελικά τροποποιημένα αντικείμενα καθώς και στο ιδιωτικό writeset της συλλογής με κλειδί το πεδίο dataID. Εφόσον ολοκληρωθεί η παραπάνω διαδικασία για όλα τα αντικείμενα που επιθυμεί να εισάγει ο χρήστης, τότε η λίστα με τα τελικά αντικείμενα δίνεται ως όρισμα στη μέθοδο executeBulk() που παρέχεται από τον Ektorp και υλοποιεί την εισαγωγή των αντικειμένων στο σύστημα της CouchDB. Αν ο χρήστης έχει επιλέξει να γίνει αυτόματα commit η συναλλαγή, τότε με την ολοκλήρωση της executeBulk καλείται η μέθοδος autocommit() της AbstractCouchCollectionImpl.

Όσον αφορά τη MongoDB, υπήρξε μία διαφοροποίηση. Όπως αναλύθηκε, εφαρμόστηκε η λογική της CouchDB με το χρήστη να δημιουργεί τις δικές του κλάσεις. Προκειμένου να μπορούν οι κλάσεις αυτές να εισαχθούν στη MongoDB είναι απαραίτητο να μετατραπούν σε αντικείμενα τύπου BSONObject. Έτσι στη μέθοδο insertManyDocs που υλοποιείται από την CpMongoCollectionImpl μετατρέπονται όλα τα αντικείμενα που πρόκειται να εισαχθούν στη βάση της MongoDB σε αντικείμενα τύπου BSONObject με χρήση της στατικής μεθόδου SerializeToHashMap της κλάσης Serializers η οποία εφαρμόζει την τεχνική της σιεριοποίησης. Η τεχνική αυτή μαζί με τη μέθοδο αναλύεται σε επόμενη ενότητα.

Λειτουργία Ανάγνωσης

Η λειτουργία της ανάγνωσης αφορά την εύρεση συγκεκριμένων εγγραφών της βάσης δεδομένων βάσει κάποιων κριτηρίων που ορίζονται από το χρήστη. Η συγκεκριμένη λειτουργία στην επέκταση της MongoDB υλοποιείται με τρόπο όμοιο με αυτό της απλής MongoDB, δηλαδή χρησιμοποιούνται οι τελεστές που παρέχει η MongoDB και τα κριτήρια ορίζονται σε ένα στιγμιότυπο της κλάσης Bson με μορφή ζευγών κλειδιού τιμής. Ο τρόπος αυτός είναι αδύνατον να ακολουθηθεί για την CouchDB καθώς υπάρχει εκ διαμέτρου διαφορετική αντιμετώπιση της συγκεκριμένης λειτουργίας. Αναλύθηκε προηγουμένως, πως η αναζήτηση με βάση πεδία εκτός των _id και _rev γίνεται με χρήση javascript συναρτήσεων map reduce που ορίζουν συγκεκριμένα Views σε μία βάση δεδομένων στο σύστημα της CouchDB. Κατά συνέπεια, ο επιβάλλεται ο χρήστης να γράψει τις συναρτήσεις map reduce βάση των οποίων θα γίνει η αναζήτηση. Συγκεκριμένα, έχει οριστεί μία κλάση με όνομα FindRepository, η οποία

πρέπει να περιέχει το annotation @Views. Μέσα στο συγκεκριμένο annotation ο χρήστης γράφει τις επιθυμητές συναρτήσεις map reduce με annotation @View. Η εικόνα του FindRepository απεικονίζεται στο ακόλουθο σχήμα.

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package eu.coherentpaas.mvcc;
7
8  import org.ektorp.CouchDbConnector;
9  import org.ektorp.support.CouchDbRepositorySupport;
10 import org.ektorp.support.View;
11 import org.ektorp.support.Views;
12
13 /**
14  *
15  * @author george
16  */
17 @Views({
18     @View( name = "seats", map = "function(doc) { if (doc.seats > 4) emit( doc.seats, doc )}"),
19     @View( name = "color", map = "function(doc) { emit( doc.color, doc )}")
20 })
21 public class FindRepository extends CouchDbRepositorySupport<MVCCDbDocument> {
22
23     public FindRepository(CouchDbConnector db){
24         super(MVCCDbDocument.class, db);
25     }
26 }
27
```

Σχήμα 5.1: Find Repository

Το συγκεκριμένο FindRepository περιέχει ένα View που έχει στόχο την αναζήτηση όλων των εγγραφών που περιέχουν περισσότερα καθίσματα από 4 και ένα ακόμη View το οποίο απλώς ομαδοποιεί τις εγγραφές βάση του χρώματος που διαθέτουν. Παρατηρούμε πως κάθε @View περιέχει και ένα όνομα. Σε ένα απλό (native) σύστημα της CouchDB τα παραπάνω θα ήταν αρκετά για τη διαδικασία της αναζήτησης, ωστόσο η επέκταση της CouchDB που υλοποιείται επιβάλλει επιπλέον διεργασίες προκειμένου να διασφαλιστεί πως υλοποιείται η αναζήτηση στα πλαίσια του MVCC και φέρνει τα επιθυμητά και έγκυρα αποτελέσματα. Για τη λειτουργία της ανάγνωσης δηλώνεται η μέθοδος findDocuments στην κλάση MVCCCollection και υλοποιείται από τις κλάσεις CpStdCouchDbConnector και CpMongoCollectionImpl. Δέχεται ως παράμετρο ένα αντικείμενο τύπου Bson και επιστρέφει ένα αντικείμενο τύπου MVCCIterable. Στη MongoDb το αρχείο Bson περιέχει όλα τα κριτήρια σε μορφή ζευγών κλειδιού (τελεστής) τιμής (τιμή πεδίου). Στην CouchDB το αντικείμενο Bson πρέπει να περιέχει ένα ζεύγος με κλειδί το αλφαριθμητικό viewName και τιμή το όνομα του view που έχει ορίσει ο χρήστης και επιθυμεί να χρησιμοποιήσει για την αναζήτηση. Στη συνέχεια γίνεται μετατροπή τύπου (typecasting) του Bson σε Document και καλείται η μέθοδος _find, όπως συμβαίνει και στη MongoDB, περνώντας σαν όρισμα το Document που περιέχει το όνομα του

View και την κλάση που θα απεικονίζει τις εγγραφές ως αντικείμενα. Η _find αρχικά ελέγχει αν η συλλογή είναι ανοικτή-διαθέσιμη, με τη μέθοδο _checkClose(), και ζητά το TransactionalContext για τη συναλλαγή που εκτελείται, με τη μέθοδο _getTransactionalContext(). Εάν δεν προκύψει κάποιο exception τότε συνεχίζεται η διαδικασία και διαβάζεται από το Document το όνομα του View που θα χρησιμοποιηθεί, ελέγχοντας να μην είναι κενό. Εν συνεχεία, πρέπει να τροποποιηθεί η συνάρτηση map ώστε να αφορά μόνο τις τελευταίες έγκυρες εκδόσεις κάθε εγγραφής, προκειμένου να εφαρμοστεί η τεχνική MVCC και να μην προκύψουν προβλήματα όπως αυτό που παρουσιάσθηκε στην παράγραφο 4.1.2. Πιο συγκεκριμένα πρέπει να τροποποιηθεί η συνάρτηση map ώστε να περιλαμβάνει τον εξής έλεγχο.

```
if ( ( (_cmtTmstmp == -1) AND (_tid == @tid) AND (_isDeleted ==
false) )
            OR
( (_cmtTmstmp < @startTmstmp)  AND (_cmtTmstmp != -1) AND (
(_nextCmtTmstmp > @startTmstmp) OR (_nextCmtTmstmp == -1) ) AND
(_isDeleted == false) ) )
```

Το πρώτο σκέλος αφορά τις εγγραφές που είχαν εισαχθεί στη βάση από τη συγκεκριμένη συναλλαγή και δεν έχουν γίνει ακόμη commit ενώ το δεύτερο σκέλος αφορά τις τελευταίες έγκυρες εκδόσεις που βρίσκονται στη βάση δεδομένων. Για την ενέργεια αυτή καλείται η μέθοδος updateViews της κλάσης SynchronizedMethods με ορίσματα το όνομα του συγκεκριμένου View, την κλάση CpStdCouchDbConnector και τη χρονοσφραγίδα που θα χρησιμοποιηθεί ως startTmstmp. Αρχικά η μέθοδος αυτή αποκτά το tid της συγκεκριμένης συναλλαγής μέσω της μεθόδου getTid() που υλοποιείται από την CouchTransactionContext. Στη συνέχεια με χρήση της μεθόδου FindRepository.class.getAnnotation(Views.class) διαβάζονται όλα τα views που έχουν γραφτεί στο FindRepository και βρίσκεται το View που έχει επιλέξει ο χρήστης. Θα δημιουργηθούν δύο νέα views, το ένα θα αφορά τον πρώτο έλεγχο (private versions) και το άλλο το δεύτερο (public versions). Αρχικά, δημιουργούνται δύο νέα Strings. Το ένα περιλαμβάνει τη συνάρτηση map που έχει γράψει ο χρήστης και η οποία περικλείεται μέσα στον πρώτο έλεγχο (private versions) και το δεύτερο περιλαμβάνει τη συνάρτηση map του χρήστη η οποία θα περικλείεται στον δέυτερο έλεγχο (public versions). Στη συνέχεια, στο FindRepository ανανεώνεται το View που έχει επιλέξει ο χρήστης με βάση την τροποποιημένη συνάρτηση map που αφορά τις ιδιωτικές εκδόσεις εγγραφών και φυσικά ανανεώνεται το όνομα του View το οποίο πλέον θα αποτελείται από το όνομα που όρισε ο χρήστης μαζί με ένα μοναδικό αναγνωριστικό που προσδίδεται με χρήση της κλάσης UUID που προσφέρει η Java. Η τροποποίηση αυτή γίνεται με χρήση της μεθόδου changeAnnotationValue η οποία χρησιμοποιεί την τεχνική Reflection όπως παρέχεται από τη Java και αφορά τις δυναμικές αλλαγές κατά το runtime του προγράμματος. Η τεχνική Reflection και η μέθοδος changeAnnotationValue θα εξηγηθούν αναλυτικά σε ακόλουθη

ενότητα. Ακολούθως, με χρήση της initStandardDesignDocument() ενημερώνεται το ειδικό designDoc της βάσης δεδομένων της CouchDB προκειμένου να περιλαμβάνει το νέο View. Είναι σημαντικό να οριστούν τα properties org.ektorp.support.UpdateDesignDocOnDiff και org.ektorp.support.AutoUpdateViewOnChange της CouchDB ώστε να έχουν τιμή true. Η ίδια ακριβώς διαδικασία ακολουθείται και για τη δημιουργία του View που θα αφορά τις δημόσιες εκδόσεις και ενημερώνεται με τον ίδιο τρόπο το designDoc. Κατά την ολοκληρωση της μεθόδου UpdateViews αλλάζει και πάλι το όνομα του View σε αυτό που είχε αρχικά προκειμένου να μπορεί να χρησιμοποιηθεί και να τροποποιηθεί μελλοντικά και τελικά επιστρέφονται τα ονόματα των δύο νέων Views που εισήχθησαν στη βάση δεδομένων. Ο λόγος που είναι απαραίτητο να αλλαχθεί το όνομα του View και να περιέχει ένα μοναδικό αναγνωριστικό είναι ότι στο desing έγγραφο μιας βάσης δεδομένων της CouchDB δεν επιτρέπεται να περιέχονται περισσότερα από ένα views με το ίδιο όνομα. Έτσι, η πιθανότητα περισσότερες από μία ταυτόχρονες συναλλαγές να χρησιμοποιούν το ίδιου view του FindRepository για διαφορετικά όμως tid και startTimestamp συνεπάγεται την ανάγκη κάθε συναλλαγή να εισάγει νέα έκδοση του view. Επίσης, επειδή όλες οι τροποποιήσεις πραγματοποιούνται στην κλάση και όχι στο στιγμιότυπο αυτής είναι απαραίτητο να υπάρξει συγχρονισμός και απόλυτη σειριοποίηση των τροποποιήσεων για να μην συμβούν σφάλματα. Εφόσον ολοκληρωθεί η UpdateViews, συνεχίζεται η εκτέλεση της _find. Αρχικοποιείται ένα ViewQuery το οποίο αφορά τις ιδιωτικές εκδόσεις (private versions) και εκτελείται η μέθοδος queryView() που υλοποιείται από την StdCouchDbConnector. Η μέθοδος αυτή επιστρέφει ως αποτέλεσμα μία λίστα που περιέχει όλα τα αποτελέσματα με βάση το ερώτημα που εκτελέστηκε. Τα περιεχόμενα της λίστας τοποθετούνται σε μία δομή HashMap, η οποία δεν επιτρέπει διπλότυπα, με κλειδί το πεδίο dataID. Η ίδια διαδικασία ακολουθείται και για το View που αφορά τις δημόσιες εκδόσεις (public versions) και τα αποτελέσματα προστίθενται στην ίδια δομή εφόσον δεν υπάρχει ήδη αντικείμενο με το ίδιο κλειδί. Σε περίπτωση που υπάρχει αντικείμενο με το ίδιο κλειδί σημαίνει πως η συγκεκριμένη δημόσια έκδοση έχει ενημερωθεί από τη συγκεκριμένη συναλλαγή, δημιουργώντας μία νεότερη έκδοση με ίδιο dataID, και άρα δεν είναι ορθό να ληφθεί η δημόσια έκδοση αλλά αρκεί η ιδιωτική. Έτσι διασφαλίζεται πως διαβάζονται οι τελευταίες εκδόσεις ακόμη και σε περίπτωση που οφείλονται σε ενημερώσεις της ίδιας συναλλαγής. Τέλος, ακολουθώντας το μοτίβο της MongoDB δημιουργείται ένα στιγμιότυπο τύπου CpFindCouchDbIterable, το οποίο περιέχει τη λίστα με τα αποτελέσματα και αποτελεί μία τυπική επέκταση του MVCCIterable όπως συμβαίνει και στη MongoDB με την κλάση CpFindMongoDbIterable, και επιστρέφεται ως αποτέλεσμα της _find.

Λειτουργία Ενημέρωσης

Η λειτουργία της ενημέρωσης δίνει τη δυνατότητα στο χρήστη να ενημερώσει τα πεδία υπαρχουσών εγγραφών στη βάση δεδομένων. Κατά την υλοποίηση του MVCC, η λειτουργία

αυτή οδηγεί στη δημιουργία νέων εγγραφών νεότερων εκδόσεων που περιέχουν τις απαραίτητες ενημερώσεις. Έτσι, θα πρέπει να διαβάζονται οι εγγραφές στις οποίες θα γίνουν οι ενημερώσεις και στη συνέχεια, αφού γίνουν οι ενημερώσεις, να εισάγονται οι νέες εγγραφές. Στη διεπαφή MVCCCollection δηλώνεται η μέθοδος updateDocuments με ορίσματα ένα αντικείμενο τύπου Document, το οποίο θα περιλαμβάνει τα κριτήρια αναζήτησης των εγγραφών που θα ενημερωθούν, και ένα αντικείμενο τύπου BasicDBObject, που θα περιλαμβάνει τις ενημερώσεις που πρέπει να γίνουν. Το αντικείμενο filter τύπου Document έχει την ίδια μορφή με την περίπτωση της λειτουργίας αναζήτησης, πράγμα που σημαίνει πως ο χρήστης πρέπει να έχει γράψει τη συνάρτηση map reduce για το ορισμό του κατάλληλου View και μέσα από το Document να περνά το όνομα του View.Αντίθετα το αντικείμενο update τύπου BasicDbObject έχει τη μορφή που έχει και στην περίπτωση της MongoDB. Δηλαδή περιλαμβάνει ζεύγη κλειδιών με κλειδί τον τελεστή και τιμή ένα νέο ζεύγος που περιλαμβάνει το πεδίο που θα ενημερωθεί και την τιμή ενημέρωσης ανάλογα με τον τελεστή. Κατά αυτό το τρόπο παρέχεται ενιαίος και κοινός τρόπος ορισμού των ενημερώσεων από τον χρήστη. Οι τελεστές που υποστηρίζονται είναι οι set, $inc, $mul, $min, $max και $unset, οι οποίοι έχουν ακριβώς τη λειτουργία που έχουν και στη MongoDB. Δεν υποστηρίζονται οι τελεστές που αφορούν πεδία πινάκων. Η updateDocuments υλοποιείται από τις κλάσεις CpStdCouchDbConnector και CpMongoCollectionImpl και ουσιαστικά καλεί την _update της εκάστοτε κλάσεις αποστέλλοντας τα ίδια ορίσματα που έλαβε. Η _update αρχικά ελέγχει αν είναι διαθέσιμη η συλλογή και λαμβάνει το TransactionalContext, με τις μεθόδους _checkClose και _getTransactionalContext. Εν συνεχεία, ακολουθείται η διαδικασία που αναλύθηκε στη λειτουργία της ανάγνωσης προκειμένου να βρεθούν οι εγγραφές που ανήκουν στην ιδιωτική συλλογή της συναλλαγής και πρέπει να ενημερωθούν. Οι εγγραφές αυτές βρίσκονται σε μία δομή iterator προκειμένου να εφαρμοστεί μία επαναληπτική διαδικασία σε κάθε μία από αυτές. Κατά συνέπεια, για κάθε εγγραφή εκτελείται η ακόλουθη διαδικασία. Αρχικά, διαβάζονται όλοι οι τελεστές από το αντικείμενο update. Για κάθε τελεστή ακολουθείται η εξής διαδικασία. Διαβάζονται όλα τα πεδία που πρέπει να ενημερωθούν από το συγκεκριμένο τελεστή καθώς και οι τιμές ενημέρωσης. Ανάλογα με τον τελεστή και την παλαιά τιμή του πεδίου, δημιουργείται η νέα τιμή και εν συνεχεία με χρήση της τεχνικής Reflection γίνεται η ενημέρωση του συγκεκριμένου πεδίου της συγκεκριμένης εγγραφής με τη νέα τιμή. Κάθε φορά που ολοκληρώνεται η διαδικασία, η ενημερωμένη εγγραφή τοποθετείται σε μία λίστα με όνομα insertList. Αφού ολοκληρωθεί η παραπάνω διαδικασία, ξεκινά η διαδικασία της ενημέρωσης όλων των δημόσιων εγγραφών. Αρχικά εκτελείται η κατάλληλη διαδικασία για να διαβαστούν οι δημόσιες εγγραφές που πρέπει να ενημερωθούν. Εν συνεχεία, για κάθε εγγραφή ακολουθείται η ίδια διαδικασία επαναληπτικά. Αρχικά ελέγχεται αν η εγγραφή διαθέτει ένα dataID το οποίο υπάρχει ως κλειδί στο ιδιωτικό writeset της συναλλαγής. Αν υπάρχει σημαίνει πως υπάρχει μία νεότερη έκδοση της εγγραφής η οποία είναι ιδιωτική και άρα η δημόσια

εγγραφή δεν χρειάζεται να ενημερωθεί. Διαφορετικά συνεχίζεται η διαδικασία ενημέρωσης της εγγραφής. Στη συνέχεια, πρέπει να γίνει ένας έλεγχος για πιθανές συγκρούσεις. Δημιουργείται έτσι μία συγχώνευση του χαρακτήρα διπλού underscore (__) με το dataID της εγγραφής και μετατρέπεται σε πίνακα από bytes. Το κλειδί αυτό δίνεται ως όρισμα στη μέθοδο checkForConflicts της CouchTransactionContext η οποία με τη σειρά της καλεί τη μέθοδο hasConflict που υλοποιείται από τον Holistic Transaction Manager και βρίσκει πιθανές συγκρούσεις προκαλώντας τα ανάλογα exceptions. Ένα τέτοιο exception οδηγεί στην ακύρωση της συναλλαγής και άρα το rollback. Εάν δεν υπάρχει σύγκρουση η διαδικασία συνεχίζεται κανονικά. Το επόμενο βήμα είναι η δημιουργία ενός νέου αντικειμένου της κλάσης MVCCDbDocument το οποίο θα έχει το ρόλο της ενημερωμένης εγγραφής που θα εισαχθεί στη βάση δεδομένων. Το νέο αυτό στιγμιότυπο έχει ίδιες τιμές στα πεδία με τη δημόσια εγγραφή στην οποία αντιστοιχεί εκτός από το πεδίο id. Η αντιγραφή των πεδίων της εγγραφής στα πεδία του νέου στιγμιότυπου γίνεται με την τεχνική Reflection. Η διαδικασία από εκείνο το σημείο και έπειτα είναι ίδια με την περίπτωση των ιδιωτικών εγγραφών, χωρίς διαφοροποίηση. Ουσιαστικά, το νέο στιγμιότυπο που δημιουργήθηκε περιέχει τιμές μόνο για τα ενημερωμένα πεδία, ενώ τα πεδία που δεν ενημερώνονται δεν περιέχουν τιμή. Μόλις ολοκληρωθεί και αυτή η διαδικασία, είναι πλέον η στιγμή να εισαχθούν οι νέες ενημερωμένες εγγραφές στη βάση δεδομένων και για αυτό καλείται η insertManyDocs με όρισμα τη λίστα με τις εγγραφές. Είναι σημαντικό πριν καλεστεί η insertManyDocs να τεθεί η μεταβλητή isAutoCommit της AbstractCouchCollectionImpl σε τιμή false ώστε να μη γίνει commit. Αφού ολοκληρωθεί η εισαγωγή, στη συνέχεια η μεταβλητή isAutoCommit αποκτά την τιμή που είχε εξ΄ ορισμού και ολοκληρώνεται η διαδικασία με commit ή όχι ανάλογα την τιμή της isAutoCommit.

Λειτουργία Διαγραφής

Η λειτουργίας της διαγραφής κατά την υλοποίηση του MVCC έγκειται στην ενημέρωση των εγγραφών που πρόκειται να διαγραφούν με το πεδίο isDeleted να τίθεται σε τιμή true. Η λειτουργία της διαγραφής υλοποιείται με τη μέθοδο deleteDocuments που δηλώνεται στη διεπαφή MVCCCollection και υλοποιείται από τις κλάσεις CpStdCouchDbConnector και CpMongoCollectionImpl. Δέχεται ως όρισμα ένα αντικείμενο τύπου Document με όνομα filter το οποίο είναι όμοιο με αυτό που χρησιμοποιείται στην περίπτωση της αναζήτησης. Κατά αυτό τον τρόπο πρέπει ο χρήστης να γράψει τα κατάλληλα Views βάσει των οποίων θα επιλεχθούν οι εγγραφές που θα διαγραφούν. Έτσι, η μέθοδος deleteDocuments καλεί την updateDocuments περνώντας ως αντικείμενο update ένα στιγμιότυπο της κλάσης BasicDBObject που περιέχει τον τελεστή set προκειμένου να ενημερωθεί το πεδίο isDeleted στην τιμή true.

Έχοντας αναλύσεις τις τέσσερις βασικές λειτουργίες ενός συστήματος βάσεων δεδομένων όπως υλοποιούνται στο πλαίσιο του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων είναι σημαντικό να αναφέρουμε πως δεν ήταν εφικτό να υλοποιηθούν όλες οι λειτουργίες κατά απολύτως όμοιο τρόπο παρέχοντας μία απόλυτα ενιαία διεπαφή. Συγκεκριμένα, η λειτουργία της αναζήτησης αντιμετωπίζεται και υλοποιείται πολύ διαφορετικά από τα δύο συστήματα βάσεων δεδομένων που χρησιμοποιούνται. Για την ύπαρξη κοινής διεπαφής θα ήταν απαραίτητο να υλοποιηθεί ένας μεταγλωττιστής για τη μετατροπή μιας εκ των δύο μορφών ερωτημάτων στην μορφή του άλλου συστήματος. Μια τέτοιο υλοποίηση είναι αρκετά πολύπλοκη και κρίθηκε εκτός των ορίων της συγκεκριμένης διπλωματικής. Για το λόγο αυτό έγινε ο συμβιβασμός η συγκεκριμένη λειτουργία να παρουσιάζει διαφορετικές απαιτήσεις από το χρήστη ανάλογα με το σύστημα το οποίο χρησιμοποιούσε. Ένας ακόμη συμβιβασμός πραγματοποιήθηκε στην λειτουργία της ενημέρωσης και συγκεκριμένα στην υποστήριξη των τελεστών που υποστηρίζονται και κατά συνέπεια στα είδη των ενημερώσεων που υποστηρίζονται. Ωστόσο, ο συμβιβασμός αυτός έγινε στα πλαίσια της δημιουργίας κοινού τρόπου χρήσης της λειτουργίας στα δύο συστήματα, κάτι που κρίθηκε σημαντικότερο συγκριτικά με την υποστήριξη όλων των ειδών ενημερώσεων. Η υλοποίηση που θα υποστήριζε όλους τους πιθανούς τελεστές της MongoDB και τη μετατροπή αυτών σε μορφή συμβατή με τη MongoDB ήταν αρκετά πολύπλοκή και κρίθηκε αχρείαστη στα πλαίσια της συγκεκριμένης διπλωματικής εργασίας.

Ακολούθως αναλύονται οι λειτουργίες της ολοκλήρωσης μιας συναλλαγής (commit), της έναρξης μιας συναλλαγής (startTransaction) καθώς και της ακύρωσης μιας συναλλαγής (rollback).

Commit

Η μέθοδος commit δηλώνεται στη διεπαφή MVCCDatabase και υλοποιείται από τις αφηρημένες κλάσεις AbstractMongoDatabaseImpl και AbstractCouchDatabaseImpl. Η μέθοδος αυτή καλεί τη μέθοδο commit του Holistic Transaction Manager προκειμένου να ενημερώσει πως πρέπει να γίνει commit και άρα όλες οι εγγραφές να ενημερωθούν με το κατάλληλο commit Timestamp. Η μέθοδος αυτή μπορεί να καλεστεί από το χρήστη, ωστόσο, υπάρχει και η επιλογή της μεθόδου autoCommit() που καλείται στο τέλος οποιασδήποτε λειτουργίας εάν το πεδίο isAutoCommit έχει την τιμή true. Η autoCommit ενημερώνει τον HTM για την πρόθεση να γίνει commit. Όταν ο HTM ενημερωθεί πως πρέπει να γίνει commit τότε ακολουθεί μία διαδικασία που περιλαμβάνει την εκτέλεση πολλών διαφορετικών μεθόδων. Η διαδικασία αυτή μπορεί να διαιρεθεί σε δύο επιμέρους βήματα. Το πρώτο βήμα αφορά την ενημέρωση του HTM σχετικά με τις εγγραφές που πρόκειται να γίνου commit και τον έλεγχο από τον HTM για πιθανές συγκρούσεις μεταξύ ταυτόχρονων δοσοληψιών. Το

δεύτερο και τελικό βήμα αφορά την μονιμοποίηση των εγγραφών στο σύστημα της βάσης δεδομένων με την κατάλληλη ενημέρωση των πεδίων cmtTmstmp και nextCmtTmstmp. Το πρώτο βήμα ξεκινά με την κλήση της μεθόδου getWS της κλάσης CouchClient από τον HTM. Η μέθοδος αυτή δημιουργεί ένα νέο CouchTransactionContext με βάση το tid της συναλλαγής που πραγματοποιεί το commit και τα startTimestamp και CommitTimestamp που παρέχονται από τον HTM. Στη συνέχεια καλείται η private μέθοδος getWS της CouchClient με όρισμα το CouchTransactionContext. Η μέθοδος αυτή λαμβάνει από τη δομή που περιέχει τις ενεργές βάσεις δεδομένων (ConcurrentMap<Long, AbstractCouchDatabaseImpl> openTransactionalDatabases) αυτή τη βάση που σχετίζεται με τη συγκεκριμένη συναλλαγή και περιέχει συλλογές στις οποίες η συναλλαγή έχει πραγματοποιήσει αλλαγές. Έτσι καλείται η μέθοδος _getWS της συγκεκριμένης βάσης, η οποία υλοποιείται από την κλάση AbstractCouchDatabaseImpl. Η μέθοδος αυτή λαμβάνει από τη δομή που περιλαμβάνει τις ενεργές συλλογές της βάσης (ConcurrentMap<String, AbstractCouchCollectionImpl> dbOpenCollections) όλες αυτές που βρίσκονται στο πλαίσιο της συγκεκριμένης συναλλαγής και για κάθε μία αποκτούνται όλες οι ιδιωτικές εγγραφές που δημιουργήθηκαν από τη συναλλαγή. Αυτό επιτυγχάνεται με την κλήση της μεθόδου _getWriteSet της εκάστοτε συλλογής επαναληπτικά. Η _getWriteSet υλοποιείται από την CpStdCouchDbConnector. Η μέθοδος αυτή αρχικά ελέγχει αν η συλλογή εξακολουθεί να είναι ενεργεί και εν συνεχεία τροποποιεί το designDoc της συγκεκριμένης συλλογής προσθέτοντας το View με όνομα commit το οποίο υπάρχει στο CommitRepository. Το συγκεκριμένο View περιέχει την ακόλουθη συνάρτηση map.

```
function(doc) {
    if ((doc.cmtTmstmp == -1) && (doc.nextCmtTmstmp == -1) )
        emit( doc.tid, doc )
}
```

Κατά αυτό τον τρόπο δημιουργείται ένα View που ομαδοποιεί όλες τις ιδιωτικές εκδόσεις με βάση το tid της συναλλαγής που ανήκουν. Επομένως, στη συνέχεια μπορεί να χρησιμοποιηθεί η εντολή ViewQuery() μαζί με τη μέθοδο key(tid) όπου στην key θα δοθεί ως όρισμα το tid της συναλλαγής που κάνει commit και έτσι θα έρθουν από τη βάση μόνο οι ιδιωτικές εγγραφές της ενδιαφερόμενης συναλλαγής. Στη συνέχεια οι εγγραφές τοποθετούνται στο ιδιωτικό writeset το οποίο επιστρέφεται κατά σειρά σε όλες τις μεθόδους που είναι εν ενεργεία μέχρις ότου να φτάσει στην getWS του CouchClient που κλήθηκε από τον HTM και αυτή να δώσει στον HTM το writeset. Το writeset έχει μετατραπεί σε πίνακα bytes και έτσι ο HTM υλοποιεί τον έλεγχο για τις συγκρούσεις. Εφόσον ο έλεγχος είναι έγκυρος τότε ο HTM καλεί την applyWS() του CouchClient, ξεκινώντας ουσιαστικά το δεύτερο βήμα της διαδικασίας του Commit. Η μέθοδος αυτή, κατά όμοιο τρόπο με την αντίστοιχη getWS, αρχικοποιεί κατάλληλα ένα CouchTransactionContext με βάσει το tid και τα startTimestamp και commitTimestamp που

παρέχονται από τον HTM. Στη συνέχεια καλεί την ιδιωτική applyWS με παράμετρο το CouchTransactionContext. Αυτή η μέθοδος είναι αντίστοιχη της ιδιωτικής getWS, λαμβάνοντας τη βάση δεδομένων του συστήματος που αφορά τη συγκεκριμένη συναλλαγή και καλώντας την _applyWS της συγκεκριμένης βάσης με όρισμα το CouchTransactionContext. Η _applyWS υλοποιείται από την AbstractCouchDatabaseImpl και λαμβάνει από τη συλλογή με τις ενεργές συλλογές (ConcurrentMap<String, AbstractCouchCollectionImpl> dbOpenCollections), όλες αυτές που αφορούν τη συγκεκριμένη συναλλαγή. Επαναληπτικά, καλείται η μέθοδος _applyWriteSet της εκάστοτε συλλογής, η οποία υλοποιείται από την κλάση CpStdCouchDbConnector. Η μέθοδος αυτή πρόκειται να ενημερώσει τα πεδία cmtTmstp των ιδιωτικών εγγραφών της βάσης και τα πεδία nextCmtTmstmp των εγγραφών που πριν το commit της συναλλαγής αποτελούσαν τις πιο πρόσφατες εκδόσεις (nextCmtTmstmp = -1) αλλά πλέον θεωρούνται απαρχαιωμένες. Έτσι, αρχικά δημιουργείται ένα νέο View στο designDoc της συλλογής το οποίο προκύπτει από το View με όνομα updateOlderVersions που υπάρχει στο CommitRepository. Η συνάρτηση map αυτού του View είναι η ακόλουθη.

```
function(doc) {
    if ((doc.cmtTmstmp != -1) && (doc.nextCcmtTmstmp == -1) )
        emit( doc.dataID, doc )
}
```

Με βάση αυτή τη συνάρτηση εντοπίζονται όλες οι νεότερες εκδόσεις κάθε εγγραφής και ομαδοποιούνται βάση του πεδίου dataID. Έτσι, με χρήση της μεθόδου .keys(dbList) για το ViewQuery, όπου η dbList είναι μία λίστα που περιέχει όλα τα dataID των ιδιωτικών εγγραφών που πρόκειται να μονιμοποιηθούν, λαμβάνονται από τη βάση μόνο οι απαραίτητες και επιθυμητές εγγραφές. Στις εγγραφές αυτές τοποθετείται στο πεδίο nextCmtTmstmp η τιμή του CommitTimestamp που παρέχεται από τον HTM και εισάγονται στη βάση ανανεώνοντας ουσιαστικά τις ήδη υπάρχουσες εγγραφές. Εν συνεχεία χρησιμοποιείται το View commit όπως και στην _getWritesSet προκειμένου να ληφθούν οι επιθυμητές ιδιωτικές εκδόσεις από τη βάση, ανανεώνεται το πεδίο cmtTmstmp με βάση το CommitTimestamp του HTM και εισάγονται στη βάση μονιμοποιώντας ουσιαστικά τις ιδιωτικές εγγραφές και καθιστώντας τες πλέον δημόσιες. Έτσι ολοκληρώνεται η δεύτερη φάση του commit και ουσιαστικά όλη η διαδικασία.

StartTransaction

Η μέθοδος αυτή σηματοδοτεί την έναρξη μιας συναλλαγής. Δηλώνεται από τη διεπαφή MVCCDatabase και υλοποιείται από τις αφηρημένες κλάσεις AbstractMongoDatabaseImpl και AbstractCouchDatabaseImpl. Στη μέθοδο αυτή ενημερώνεται ο HTM για την έναρξη μιας νέας συναλλαγής και στη συνέχεια αποκτούνται οι κατάλληλες χρονοσφραγίδες για την

αρχικοποίηση του κατάλληλου TransactionContext της συναλλαγής. Επίσης, ενημερώνεται και ο client την έναρξη της νέας συναλλαγής και του αναγνωριστικού της.

Rollback

Η λειτουργία του rollback αφορά τη διαδικασία ακύρωση μιας συναλλαγής και των αλλαγών που έχουν πραγματοποιηθεί στο σύστημα της βάσης δεδομένων από τη συγκεκριμένη συναλλαγή. Γνωρίζοντας τον τρόπο λειτουργίας της υλοποίησης που περιγράφεται, προκύπτει εύλογα το συμπέρασμα πως η ακύρωση της συναλλαγής επιβάλλει τη διαγραφή όλων των εγγραφών που έχουν εισαχθεί σε μία βάση δεδομένων της CouchDB από τη συγκεκριμένη συναλλαγή και οι οποίες δεν έχουν μονιμοποιηθεί, δηλαδή δεν έχουν γίνει commit λαμβάνοντας commitTimestamp. Η λειτουργία του rollback πραγματοποιείται από τη μέθοδο rollback, η οποία δηλώνεται στην διεπαφή TransactionalDSClient και υλοποιείται από τις κλάσεις MongoClient και CouchClient. Η συγκεκριμένη μέθοδος έχει υπερφορτώσεις. Στη μία περίπτωση δεν δέχεται όρισμα, οπότε δημιουργεί ένα νέο CouchTransactionContext, το οποίο λαμβάνει τιμές για τα πεδία tid, startTimestamp και commitTimestamp από τον HTM, και καλεί την ιδιωτική μέθοδο rollback(CouchTransactionContext couchTxnCtx). Η δεύτερη υπερφόρτωση δέχεται όρισμα ένα αντικείμενο τύπου TxnCtx με όνομα txnctx και δημιουργεί ένα νέο CouchTransactionContext, το οποίο λαμβάνει τιμές για τα πεδία tid, startTimestamp και commitTimestamp από το αντικείμενο txnctx, και ομοίως καλεί την ιδιωτική μέθοδο rollback(CouchTransactionContext couchTxnCtx). Η rollback ελέγχει αν υπάρχει ενεργή βάση με αναγνωριστικό ίδιο με το tid του couchTxnCtx και αν υπάρχει καλή τη μέθοδο _rollback της συγκεκριμένης βάσης. Στη συνέχεια διαγράφει τη βάση ενεργές-διαθέσιμες βάσεις δεδομένων. Η _rollback() που αφορά τη βάση υλοποιείται στις αφηρημένες κλάσεις AbstractCouchDatabaseImpl και AbstractMongoDatabaseImpl. Η συγκεκριμένη μέθοδος έχει στόχο να προκαλέσει ένα rollback σε όλες τις συλλογές που διαθέτει, καθώς είναι πιθανό η συναλλαγή που πρέπει να ακυρωθεί να έχει πραγματοποιήσει αλλαγές σε περισσότερες από μία βάσεις δεδομένων. Συνεπώς, καλεί επαναληπτικά τη μέθοδο _rollback για κάθε μία συλλογή που περιέχεται στη δομή με τις ενεργές συλλογές (dbOpenCollections). Η _rollback που αφορά τις συλλογές δηλώνεται στις αφηρημένες κλάσεις AbstractCouchCollectionImpl και AbstractMongoCollectionImpl και υλοποιείται στις κλάσεις CpStdCouchDbConnector και CpMongoCollectionImpl. Στόχος της μεθόδου αυτής είναι να διαγράψει όλες τις ιδιωτικές εγγραφές, δηλαδή αυτές που δεν έχουν μονιμοποιηθεί (commit), και οφείλονται στη συναλλαγή που ακυρώνεται. Αρχικά, πραγματοποιείται έλεγχος για το εάν η συλλογή εξακολουθεί να παραμένει ανοιχτή. Στη συνέχεια χρησιμοποιείται η μέθοδος updateDeleteView η οποία έχει στόχο να τροποποιήσει το View με όνομα privateVersions που βρίσκεται στο DeleteRepository ώστε να χρησιμοποιηθεί από τη μέθοδο this._deprecatedConnector.queryView για την εύρεση όλων των ιδιωτικών εγγραφών της συναλλαγής. Ουσιαστικά, η map συνάρτηση περιλαμβάνει

έναν έλεγχο για την εύρεση των εγγραφώ που έχουν tid ίδιο με αυτό της συναλλαγής και στο πεδίο cmtTmstmp υπάρχει η τιμή -1. Αφού εκτελεστεί το ερώτημα στη βάση και ληφθούν τα αποτελέσματα τότε τοποθετούνται επαναληπτικά σε μία λίστα με την εντολή deleteDocs.add(BulkDeleteDocument.of(doc)), όπου το BulkDeleteDocument.of δηλώνει στον Ektorp ότι το συγκεκριμένο αντικείμενο πρέπει να διαγραφεί, και στη συνέχεια εκτελείται η εντολή this._deprecatedConnector.executeBulk(deleteDocs) η οποία περνά όλες τις αλλαγές, που περιέχονται στη λίστα, στη συλλογή διαγράφοντας έτσι τις αντίστοιχες εγγραφές.

5.2.2 Java Reflection

Η τεχνική Reflection που παρέχεται από τη Java υπήρξε εξαιρετικά σημαντική για την υλοποίηση του MVCC στο σύστημα της CouchDB με τρόπο που να προσομοιάζει την υλοποίηση της MongoDB. Πρόκειται για μία τεχνική που παρέχει η Java και έχει ως στόχο να δώσει τη δυνατότητα στον προγραμματιστή να χειριστεί διάφορα δεδομένα με δυναμικό τρόπο κατά τη διάρκεια που εκτελείται το πρόγραμμα. Η Java είναι μία γλώσσα προγραμματισμού που διαθέτει στατικό σύστημα τύπων. Ο ορισμός των τύπων γίνεται κατά τη διαδικασία της μεταγλώττισης και συνεπώς η χρήση κάποιας μεθόδου ή κάποιου πεδίου επιβάλλει να γνωρίζει ο προγραμματιστής ότι υπάρχει η συγκεκριμένη μέθοδος ή πεδίο στην κλάση κατά τη διαδικασία της δημιουργίας του προγράμματος. Η όποια ελευθερία προσδίδεται με τη χρήση διεπαφών. Ωστόσο, συχνά υπάρχει η ανάγκη χρήσης μιας μεθόδου ή ενός πεδίου μιας κλάσης του οποίου το όνομα δεν είναι γνωστό κατά τη διαδικασία της μεταγλώττισης αλλά μόνο κατά τη διαδικασία της εκτέλεσης (runtime). Πέραν όμως αυτών, μία εξίσου σημαντική ανάγκη είναι η δυνατότητα ο κώδικας ενός τμήματος του προγράμματος να τροποποιεί ένα άλλο τμήμα κώδικα του προγράμματος δυναμικά κατά τη διάρκεια της εκτέλεσης του προγράμματος. Ένα παράδειγμα αποτελεί η περίπτωση που αναφέρθηκε στη λειτουργία της ανάγνωσης και αφορά την περίπτωση όπου ο χρήστης περνά ως παράμετρο String το όνομα του View, κάτι που σημαίνει πως δεν είναι γνωστό κατά τη διάρκεια δημιουργίας του προγράμματος το ποιο View θα χρησιμοποιηθεί και μάλιστα αυτό δεν είναι κάτι στατικό, αλλά και το γεγονός ότι πρέπει το View αυτό να τροποποιηθεί κατά τη διάρκεια της εκτέλεσης με δυναμικά δεδομένα όπως το tid και το commitTimestamp. Είναι προφανές λοιπόν πόσο σημαντική είναι η τεχνική του Reflection.

Η τεχνική του Reflection περιλαμβάνει μεθόδους προκειμένου το πρόγραμμα να αποκτήσει πρόσβαση στην κλάση ενός αντικειμένου. Πιο συγκεκριμένα, έστω ότι υπάρχει το στιγμιότυπο red_sofa της κλάσης Sofa που απεικονίζει ένα καναπέ (το παράδειγμα αποτελεί έμπνευση από αντίστοιχα παραδείγματα της Couch). Γράφοντας red_sofa.class επιστρέφεται ένα αντικείμενο τύπου Class το οποίο απεικονίζει την κλάση Sofa. Το συγκεκριμένο αντικείμενο περιέχει μεθόδους που έχουν στόχο να προσφέρουν πρόσβαση σε όλα τα στοιχεία που περιέχει η κλάση,

όπως πεδία, μέθοδοι, annotations κ.α. Μία από τις βασικότερες μεθόδους που παρέχει ένα στιγμιότυπο της κλάσης Class είναι η μέθοδος getName() η οποία επιστρέφει το όνομα της κλάσης. Επίσης, επιτρέπεται η πρόσβαση στους modifiers μιας κλάσης, δηλαδή στις λέξεις-κλειδιά όπως public, protected, private, final και άλλων, μέσω της μεθόδου getModifiers() η οποία επιστρέφει έναν ακέραιο αριθμό που απεικονίζει τη λέξη και εν συνεχεία με χρήση στατικών μεθόδων όπως Modifier.isAbstract(int modifiers) και Modifier.isPublic(int modifiers) μπορούν να βγουν χρήσιμα συμπεράσματα αναφορικά με την κλάση. Οι μέθοδοι getSuperClass() και getInterfaces() επιστρέφουν ως αποτελέσματα κλάσεις που καταδεικνύουν την κληρονομικότητα που έχει η συγκεκριμένη κλάση. Επίσης, η μέθοδος getConstructors() επιστρέφει ένα πίνακα αντικειμένων Constructors οι οποίοι είναι όλοι οι κατασκευαστές που περιλαμβάνει η κλάση δίνοντας έτσι τη δυνατότητα να επιλεχθεί να χρησιμοποιηθεί ο επιθυμητός κατασκευαστής ανάλογα με τα ορίσματα που δέχονται. Μία από τις κυριότερες δυνατότητες που παρέχει η τεχνική Reflection είναι η πρόσβαση στις μεθόδους μιας κλάσης κάτι που γίνεται μέσω της μεθόδου getMethods(). Η μέθοδος αυτή επιστρέφει ένα πίνακα με αντικείμενα τύπου Method και ουσιαστικά κάθε αντικείμενο του πίνακα απεικονίζει μία μέθοδο δίνοντας τη δυνατότητα να καλεστεί ή να τροποποιηθεί οποιαδήποτε μέθοδος του αντικειμένου αυτού. Όμοια με τις μεθόδους, υπάρχει η μέθοδος getFields() η οποία επιστρέφει πίνακα αντικειμένων Field και δίνεται η δυνατότητα να χρησιμοποιηθούν συγκεκριμένα πεδία ή να τροποποιηθούν ως προς την τιμή, το όνομα τους ή ακόμη και τον τύπο τους. Τέλος, εξαιρετικά σημαντική υπήρξε η δυνατότητα να εποπτευθούν και να τροποποιηθούν τα annotations μιας κλάσης μέσω της μεθόδου getAnnotations(). Τα annotations είναι μία καινοτομία που εισήχθη στη Java 5 και αποτελούν ένα είδος σχολίων ή μετα-δεδομένων που μπορεί να αφορούν μία κλάση, μία μέθοδο ή και ένα πεδίο μιας κλάσης και έχουν στόχο να χρησιμοποιηθούν από εργαλεία και τεχνικές της Java κατά η μεταγλώττιση ή κατά την εκτέλεση. Η μέθοδος getAnnotations() επιστρέφει ένα πίνακα με αντικείμενα τύπου Annotation τα οποία στη συνέχεια μπορούν να εποπτευθούν και να τροποποιηθούν με συγκεκριμένες μεθόδους.

Χρήση τεχνικής Reflection στην υλοποίηση

Η τεχνική Reflection χρησιμοποιήθηκε σε ορισμένα σημεία για την τροποποίηση τμημάτων κώδικα κατά τη διάρκεια της εκτέλεσης του προγράμματος. Συγκεκριμένα, στη μέθοδο UpdateViews που έχει στόχο να τροποποιήσει το View που επιθυμεί να χρησιμοποιήσει ο χρήστης για αναζήτηση με τρόπο ώστε να καλύπτει τις απαιτήσεις του MVCC χρησιμοποιείται η εντολή Views views = FindRepository.class.getAnnotation(Views.class) προκειμένου να αποθηκευτεί στο αντικείμενο views το annotation που είναι τύπου Views και περιέχεται στο FindRepository. Στη συνέχεια χρησιμοποιείται η εντολή View[] view = views.value() και πλέον στον πίνακα view περιέχονται όλα τα views που έχει γράψει ο χρήστης στο annotation

Views στο FindRepository. Στη συνέχεια, διατρέχοντα τον πίνακα view και ελέγχοντας το όνομα κάθε view μπορεί να βρεθεί το view που επιθυμεί να χρησιμοποιήσει ο χρήστης και έχει δώσει ως παράμετρο το όρισμα του. Η παραπάνω περιγραφή καταδεικνύει τη χρήση της τεχνική Reflection για την εποπτεία κώδικα κατά τη διάρκεια της εκτέλεσης. Η μέθοδος changeAnnotationValue(View annotation, String key, Object newValue) που περιέχεται στην κλάση SynchronizedMethods πραγματοποιεί δυναμική τροποποίηση κώδικα κατά τη διάρκεια της εκτέλεσης. Το όρισμα annotation αφορά το συγκεκριμένο view που θα τροποποιηθεί, το όρισμα key αφορά το συγκεκριμένο πεδίο του view που θα τροποποιηθεί, για παράδειγμα το πεδίο map ή το πεδίο name, και το όρισμα newValue περιέχει τη νέα τιμή που θα περιέχει το πεδίο key μετά την τροποποίηση. Αρχικά, αρχικοποιείται ένας handler για το συγκεκριμένο view με τη μέθοδο Proxy.getInvocationHandler(annotation) και στη συνέχεια η εντολή Field f = handler.getClass().getDeclaredField("memberValues") αποδίδει στο πεδίο f το πεδίο με όνομα memberValues του handler. Εν συνεχεία, στόχος είναι η πρόσβαση στα πεδία του view και αυτό επιτυγχάνεται με την εντολή Map<String, Object> memberValues = (Map<String, Object>) f.get(handler). Πλέον, η δομή memberValues περιλαμβάνει ως ζεύγη τα πεδία του view με κλειδί το όνομα τους κι τιμή το περιεχόμενο τους. Ακολούθως, η εντολή Object oldValue = memberValues.get(key) αποδίδει στο oldValue την τιμή του πεδίου που πρόκειται να αλλαχθεί και γίνεται έλεγχος προκειμένου αυτή η τιμή να μην είναι null, πράγμα που θα σήμαινε πως το πεδίο αυτό δεν υπάρχει, και να είναι ίδιας τάξης με τη νέα τιμή που θα τοποθετηθεί. Εφόσον ο έλεγχος είναι επιτυχής, η εντολή memberValues.put(key,newValue) τοποθετεί τη νέα τιμή στο επιθυμητό πεδίο. Έχοντας λοιπόν δημιουργήσει στη μέθοδο UpdateViews τη νέα map function ή το νέο όνομα του View, με χρήση της changeAnnotationValue γίνεται η τελική μετατροπή του κώδικα.

Η ίδια ακριβώς λογική χρησιμοποιείται και στην περίπτωση του rollback για την μετατροπή του View που περιέχεται στο DeleteRepository και θα πρέπει να λειτουργεί με τρόπο ώστε να βρει όλες τις ιδιωτικές εγγραφές μιας συναλλαγής.

5.2.3 Σειριοποίηση Αντικειμένου – Object Serialization

Κατά την περιγραφή της σχεδίασης αναλύθηκε η σχεδιαστική απόφαση σχετικά με την απαίτηση ο χρήστης να δημιουργεί τις δικές του κλάσεις για την απεικόνιση των αντικειμένων και να δίνει τα αντικείμενα αυτά ως ορίσματα στις κατάλληλες μεθόδους για την εισαγωγή τους στη βάση δεδομένων. Η σχεδιαστική απόφαση αυτή είναι σε συμφωνία με την CouchDB, αλλά διαφέρει σημαντικά με την προσέγγιση της MongoDB η οποία χρησιμοποιεί αντικείμενα BSON για την εισαγωγή νέων εγγραφών. Τα αντικείμενα αυτά ακολουθούν τη μορφοποίηση JSON με ζεύγη κλειδιού τιμής. Συνεπώς, είναι απαραίτητη η μετατροπή της κλάσης που όρισε ο χρήστης σε αντικείμενο BSON για την εισαγωγή νέας εγγραφής στη MongoDB. Για το λόγο

αυτό χρησιμοποιείται η τεχνική serialization που παρέχεται από τη Java και δίνει τη δυνατότητα δημιουργίας ενός νέου αντικειμένου από κάποιο άλλο αντικείμενο διαφορετικού τύπου. Η κλάση BasicDbObject που αποτελεί υλοποίηση της διεπαφής BSONObject παρέχει ένα κατασκευαστή που δέχεται ένα αντικείμενο τύπου HashMap και δημιουργεί ένα αντίστοιχο αντικείμενο τύπου BasicDbObject. Επομένως, υλοποιήθηκε μία μέθοδος για τη μετατροπή μιας κλάσης σε αντικείμενο τύπου HashMap. Αρχικά, δημιουργήθηκε η κλάση Serializers η οποία περιέχει τη μέθοδο SerializeToHashMap. Η μέθοδος αυτή δέχεται ως όρισμα ένα αντικείμενο τύπου Object, όλες οι κλάσεις κληρονομούν τη κλάση Object εξ' ορισμού, και επιστρέφει ένα αντικείμενο τύπου HashMap. Για την διαδικασία της σειριοποίησης χρησιμοποιούνται αντικείμενα τύπου ObjectMapper τα οποία παρέχουν την απαραίτητη μετατροπή αντικειμένων. Έτσι, στη μέθοδο που αναλύεται χρησιμοποιείται ένας ObjectMapper προκειμένου να μετατραπεί το αντικείμενο τύπου Object σε αντικείμενο τύπου JSONObject. Η μετατροπή αυτή διευκολύνει σημαντικά τη δημιουργία του αντικειμένου HashMap. Πιο συγκεκριμένα, εφαρμόζεται επαναληπτικά η ίδια διαδικασία για όλα τα ζεύγη που περιέχει το JSONObject. Αρχικά, ελέγχεται τι είδους αντικείμενο περιέχεται στην τιμή κάθε ζεύγους. Αν είναι απλός τύπος, για παράδειγμα αριθμός ή αλφαριθμητικό, τότε τοποθετείται το ζεύγος αυτό στη δομή HashMap. Αν όμως στην τιμή του ζεύγους υπάρχει ένα ολόκληρο αντικείμενο json τότε καλείται αναδρομικά η μέθοδος SerializeToHashMap προκειμένου να μετατραπεί κατάλληλα το αντικείμενο αυτό σε hashmap και να συνεχιστεί η διαδικασία. Η διαδικασία αυτή ολοκληρώνεται αφού έχει μετατραπεί όλο το αντικείμενο σε κατάλληλη μορφή για να περιέχεται στη δομή Hash Map και στη συνέχεια η δομή αυτή χρησιμοποιείται για τη δημιουργία του κατάλληλου αντικειμένου BSONObject.

5.2.4 Συγχρονισμός

Η έννοια του συγχρονισμού αποτελεί ένα από τα κυριότερων κρίσιμα σημεία του παράλληλου προγραμματισμού με ταυτόχρονη εκτέλεση και πραγματεύεται το θέμα της ταυτόχρονης πρόσβασης σε κάποιο πόρο από διαφορετικά νήματα εκτέλεσης. Η Java προσφέρει εντολές για συγχρονισμό νημάτων. Ένας βασικό τρόπος είναι η χρήση της δεσμευμένης λέξης synchronized για μεθόδους και πεδία, η οποία επιβάλλει πως δεν θα γίνεται πρόσβαση στη συγκεκριμένη μέθοδο ή πεδίο από δύο διαφορετικά νήματα ταυτόχρονα. Το θέμα του συγχρονισμού εντοπίζεται στην παρούσα υλοποίηση στην περίπτωση της επεξεργασίας των Views μιας κλάσης με χρήση της μεθόδου Reflection. Συγκεκριμένα, η τροποποίηση ενός annotation με reflection γίνεται σε μία κλάση και όχι στο στιγμιότυπο της κλάσης. Συνεπώς, η ταυτόχρονη τροποποίηση του ίδιου View μιας συγκεκριμένης κλάσης από διαφορετικά threads θα έχει ανεπιθύμητα αποτελέσματα. Για το λόγο αυτό οι 3 μέθοδοι που επηρεάζουν τα Views μια κλάσης τοποθετήθηκαν σε μία κλάση με όνομα SynchronizedMethods, η οποία διασφαλίζεται μέσω του μοτίβου Singleton πως θα έχει ένα στιγμιότυπο για όλα τα νήματα.

Έτσι, η δήλωση όλων των μεθόδων της SynchronizedMethods ως synchronized επιβάλλει συγχρονισμό των νημάτων για τη χρήση των συγκεκριμένων μεθόδων και κατά συνέπεια αποφεύγεται το ενδεχόμενο το ίδιο View μιας κλάσης να τροποποιηθεί ταυτόχρονα από δύο διαφορετικά νήματα.

6

Έλεγχος Λειτουργίας

Στο κεφάλαιο αυτό θα παρουσιαστεί η λειτουργία του συστήματος. Για την υλοποίηση χρησιμοποιήθηκε η έκδοση της MongoDB 3.0 και της CouchDB 1.6.1. Επίσης, για τη διαχείριση της CouchDB χρησιμοποιήθηκε η διαδικτυακή εφαρμογή Futon, η οποία προσφέρει πρόσβαση στο τοπικό σύστημα της CouchDB μέσω περιηγητή (browser).

Αναφορικά με τις λειτουργίες και την ορθή εκτέλεση τους παρατίθενται εικόνες που παρουσιάζουν τις CRUD λειτουργίες όπως υλοποιήθηκαν και την επίδραση που έχουν σε κάθε ένα από τα δύο συστήματα δεδομένων. Για κάθε λειτουργία παρουσιάζεται ένα πλήρες παράδειγμα που καλύπτει όλες τις περιπτώσεις που μπορεί να προκύψουν βάσει των εκδόσεων των εγγραφών που υπάρχουν σε κάθε σύστημα. Πρέπει να σημειωθεί πως εμφανίζονται ορισμένες διαφορές στις χρονοσφραγίδες που εμφανίζονται στις εγγραφές των MongoDB και CouchDB και ο λόγος είναι ότι για κάθε σύστημα χρησιμοποιήθηκε διαφορετική συναλλαγή με αποτέλεσμα να λάβει διαφορετική χρονοσφραγίδα. Ωστόσο, κάθε παράδειγμα έχει ακριβώς την ίδια λογική και τα ίδια αποτελέσματα και στα δύο συστήματα. Για την MongoDB χρησιμοποιήθηκε η βάση MongoDatabase και η συλλογή MongoDemoCollection. Αντίστοιχα, για την CouchDB χρησιμοποιήθηκε η βάση με το συμβατικό όνομα CouchDemoCollection. Για το σύστημα της CouchDB δημιουργήθηκε ένα πολύ απλό View προκειμένου να απεικονίζονται όλες οι εγγραφές. Ο λόγος έγινε αυτό είναι το γεγονός πως η απεικόνιση ενός View παρουσιάζει όλα τα πεδία της εγγραφής. Έτσι, το design αρχείο της CouchDemoCollection είχε την εξής μορφή.

Σχήμα 6.1: Design Document Views

Λειτουργίας Δημιουργίας

Για το παράδειγμα αυτό δημιουργήθηκαν δύο νέα αντικείμενα τα οποία απεικονίζουν δύο διαφορετικούς υπαλλήλους μιας επιχείρησης. Κάθε εγγραφή διατηρεί ως πεδία το μισθό και το όνομα του υπαλλήλου καθώς και όλα τα απαραίτητα πεδία για την υλοποίηση του MVCC. Αρχικά οι δύο βάσεις δεδομένων ήταν εντελώς άδειες. Στην CouchDB υπάρχει το DesignDoc.

Σχήμα 6.2: Κενή CouchDB

Σχήμα 6.3: Κενή MongoDB

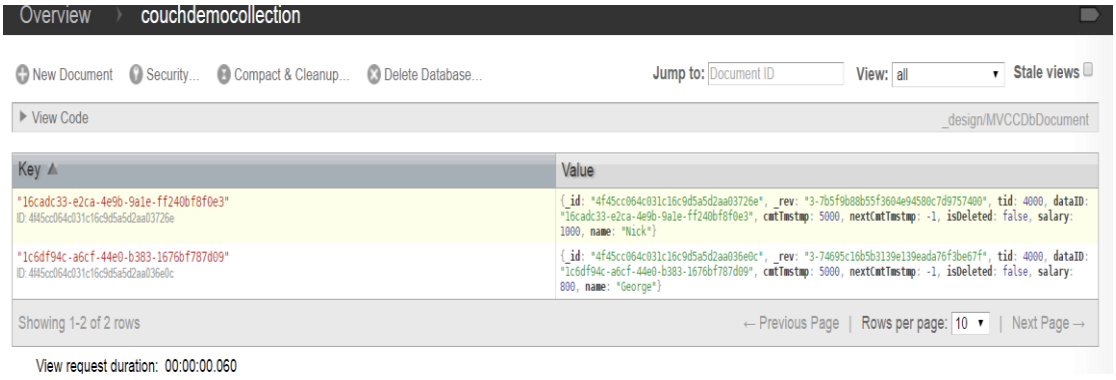
Στην συνέχεια η εισαγωγή των δύο εγγραφών δημιούργησε την ακόλουθη εικόνα.

Σχήμα 6.4: Εισαγωγή χωρίς Μονιμοποίηση στην CouchDB

```
> db.MongoDemoCollection.find().pretty()
{
        "_id" : BinData(128,"U5TaxMWxzgyMi0QeAAAAAAAAAA+g="),
        "cmtTmstmp" : "",
        "nextCmtTmstmp" : "",
        "name" : "George",
        "salary" : 800,
        "tid" : NumberLong(1000),
        "dataID" : ObjectId("5794dac4c5b1ce0c8c8b441e")
}
{
        "_id" : BinData(128,"U5TaxMWxzgyMi0QfAAAAAAAAAA+g="),
        "cmtTmstmp" : "",
        "nextCmtTmstmp" : "",
        "name" : "Nick",
        "salary" : 1000,
        "tid" : NumberLong(1000),
        "dataID" : ObjectId("5794dac4c5b1ce0c8c8b441f")
}
```

Σχήμα 6.5: Εισαγωγή χωρίς Μονιμοποίηση στη MongoDB

Το πεδίο cmtTmstmp στην CouchDB έχει την τιμή -1 και στη MongoDB έχει κενή τιμή, κάτι που καταδεικνύει πως οι εγγραφές δεν έχουν μονιμοποιηθεί στα συστήματα αποθήκευσης καθώς δεν έχει γίνει Commit. Όταν ολοκληρωθεί η λειτουργία Commit τα πεδία cmtTmstmp αποκτούν τιμή και έχουμε την ακόλουθη εικόνα.



Σχήμα 6.6: Εισαγωγή CouchDB

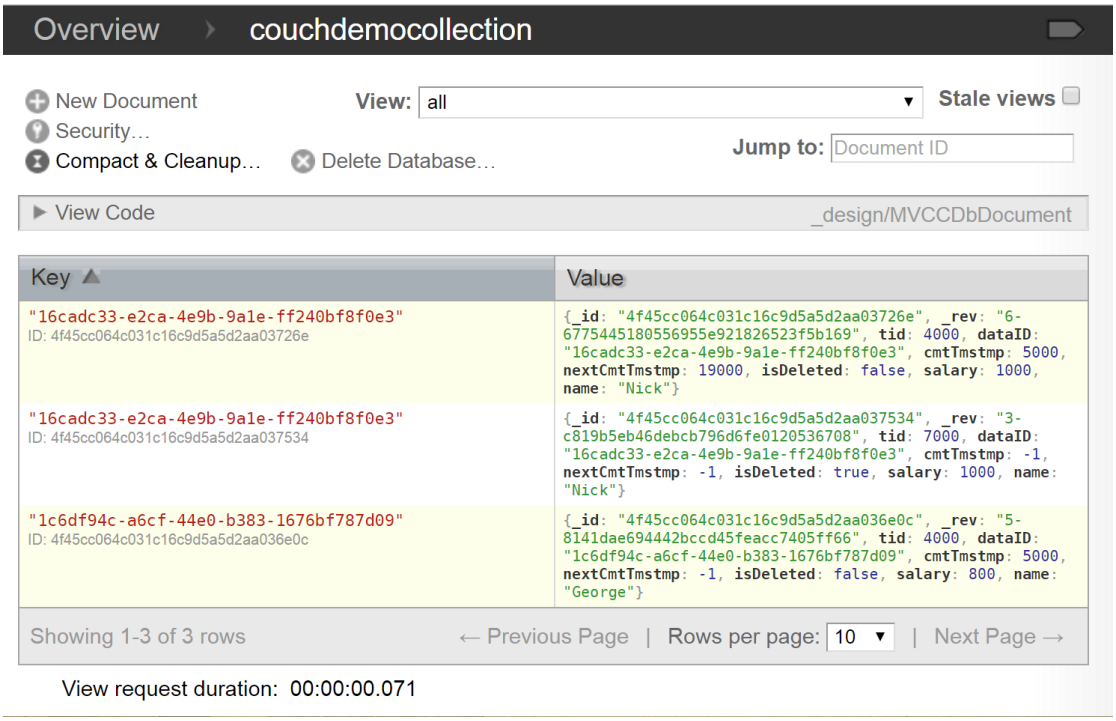
```
> db.MongoDemoCollection.find().pretty()
{
        "_id" : BinData(128,"U5TaxMWxzgyMi0QeAAAAAAAAAA+g="),
        "cmtTmstmp" : NumberLong(5000),
        "nextCmtTmstmp" : "",
        "name" : "George",
        "salary" : 800,
        "tid" : NumberLong(1000),
        "dataID" : ObjectId("5794dac4c5b1ce0c8c8b441e")
}
{
        "_id" : BinData(128,"U5TaxMWxzgyMi0QfAAAAAAAAAA+g="),
        "cmtTmstmp" : NumberLong(5000),
        "nextCmtTmstmp" : "",
        "name" : "Nick",
        "salary" : 1000,
        "tid" : NumberLong(1000),
        "dataID" : ObjectId("5794dac4c5b1ce0c8c8b441f")
}
```

Σχήμα 6.7: Εισαγωγή MongoDB

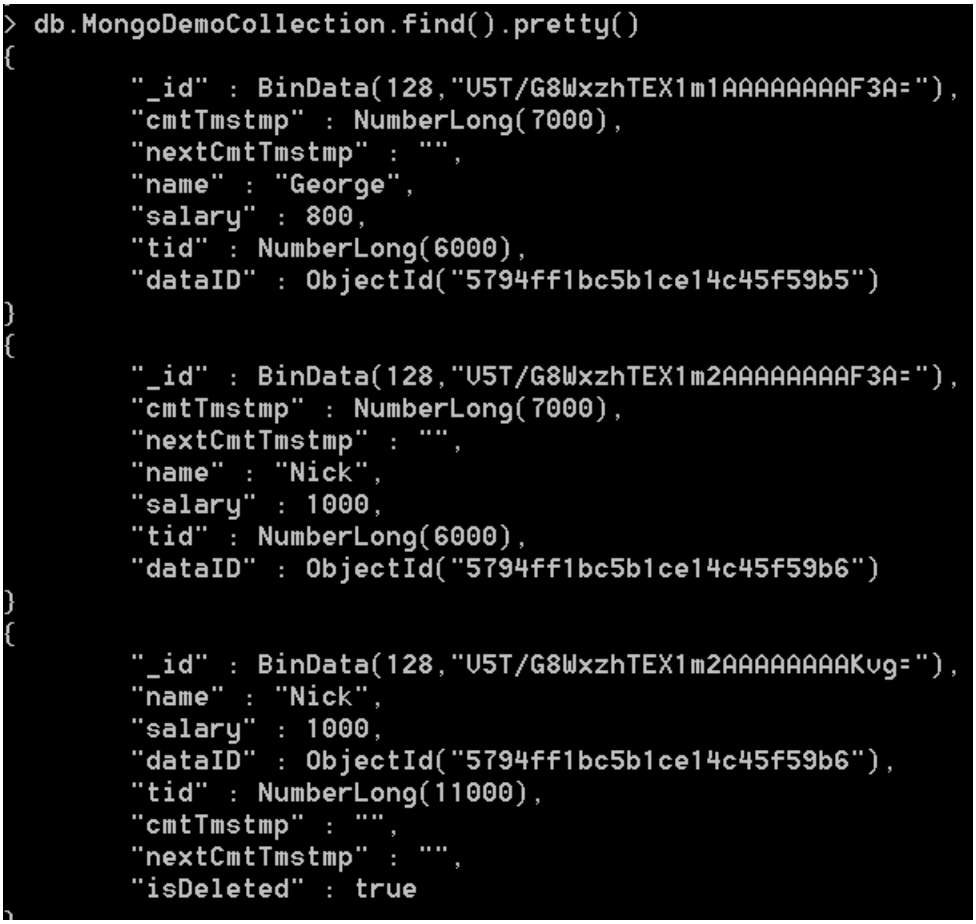
Παρατηρούμε πως όλα τα πεδία έχουν αποκτήσει τις σωστές τιμές ανάλογα με το σύστημα στο οποίο αποθηκεύονται. Είναι εμφανές ότι τα αρχεία έχουν διαφορές στη μορφοποίηση, ωστόσο ακολουθούν τους ίδιους κανόνες και έχουν κοινή λειτουργία.

Λειτουργία Διαγραφής

Όπως εξηγήθηκε, η λειτουργία της διαγραφής υλοποιείται με την εισαγωγή μιας νέα ενημερωμένης εγγραφής όπου το πεδίο isDeleted έχει την τιμή true. Έχοντας το εκάστοτε σύστημα στην κατάσταση που δημιουργήθηκε από την λειτουργία της δημιουργίας εκτελείται η διαγραφή των υψηλόμισθων εργαζομένων, δηλαδή αυτών που έχουν μισθό υψηλότερο από 900. Αρχικά παρουσιάζεται η κατάσταση προτού γίνει το Commit.

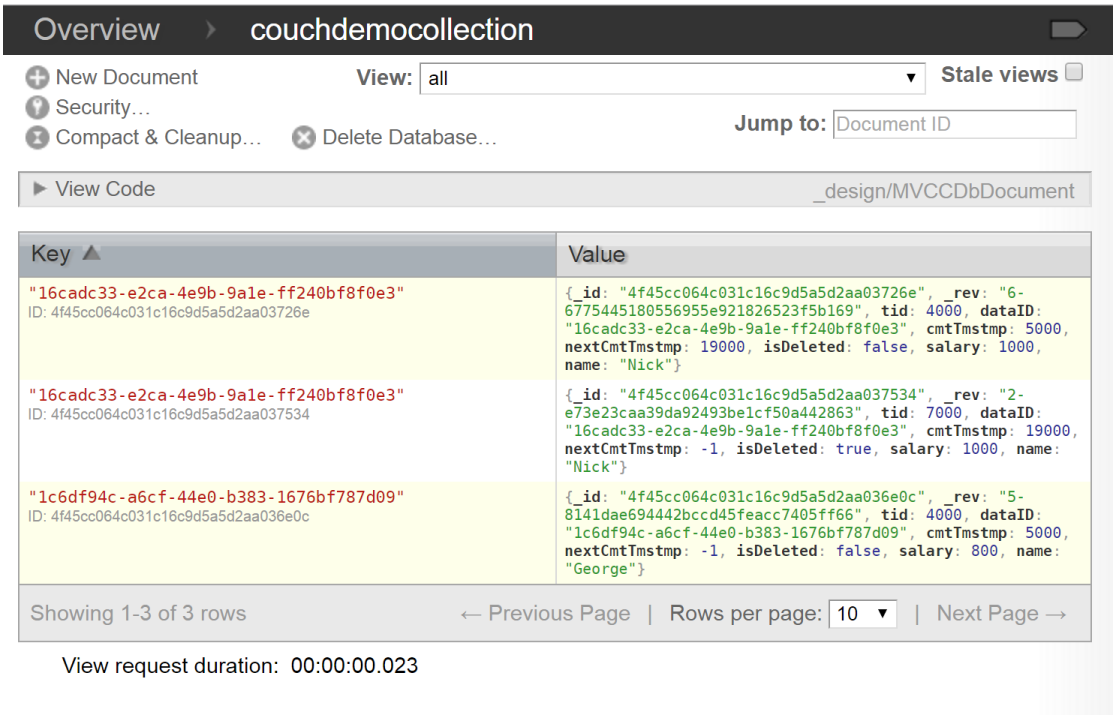


Σχήμα 6.8: Διαγραφή χωρίς Μονιμοποίηση CouchDB



Σχήμα 6.9: Διαγραφή χωρίς Μονιμοποίηση MongoDB

Παρατηρείται η εισαγωγή μιας νέας έκδοσης για το χρήστη Nick όπου το πεδίο isDeleted έχει την τιμή true. Δεν έχει γίνει ακόμη Commit για αυτό και τα πεδία cmtTmstmp της νέας εγγραφής και nextCmtTmstmp της προηγούμενης εγγραφής του εργαζόμενου Nick δεν έχουν τιμή. Ακολουθεί η κατάσταση των συστημάτων αφού γίνει Commit.



Σχήμα 6.10: Διαγραφή CouchDB

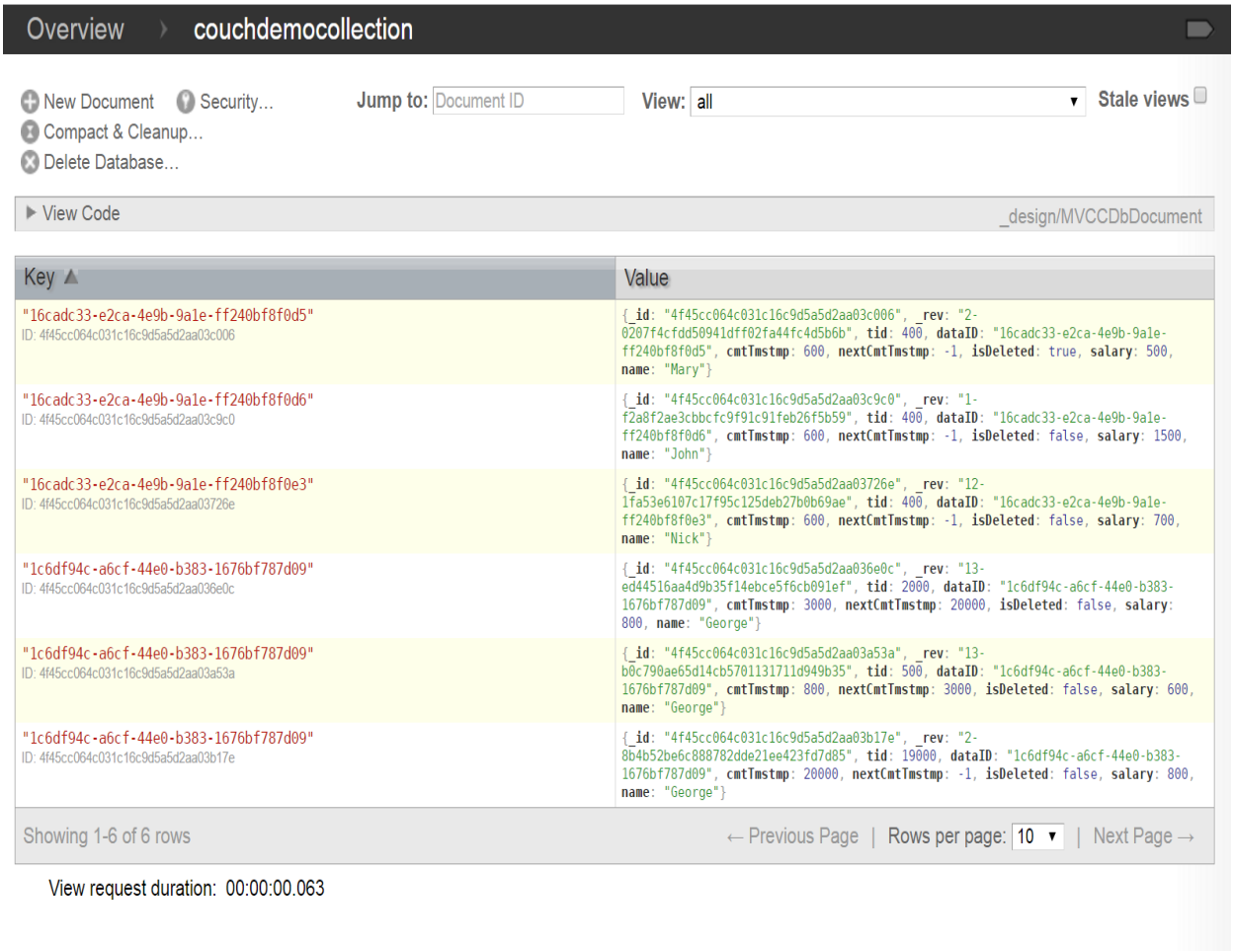
```
> db.MongoDemoCollection.find().pretty()
{
        "_id" : BinData(128,"U5T/G8WxzhTEX1m1AAAAAAAAF3A="),
        "cmtTmstmp" : NumberLong(7000),
        "nextCmtTmstmp" : "",
        "name" : "George",
        "salary" : 800,
        "tid" : NumberLong(6000),
        "dataID" : ObjectId("5794ff1bc5b1ce14c45f59b5")
}
{
        "_id" : BinData(128,"U5T/G8WxzhTEX1m2AAAAAAAAF3A="),
        "cmtTmstmp" : NumberLong(7000),
        "nextCmtTmstmp" : NumberLong(23000),
        "name" : "Nick",
        "salary" : 1000,
        "tid" : NumberLong(6000),
        "dataID" : ObjectId("5794ff1bc5b1ce14c45f59b6")
}
{
        "_id" : BinData(128,"U5T/G8WxzhTEX1m2AAAAAAAAAKug="),
        "name" : "Nick",
        "salary" : 1000,
        "dataID" : ObjectId("5794ff1bc5b1ce14c45f59b6"),
        "tid" : NumberLong(11000),
        "cmtTmstmp" : NumberLong(23000),
        "nextCmtTmstmp" : "",
        "isDeleted" : true
}
```

Σχήμα 6.11: Διαγραφή MongoDB

Το πεδία cmtTmstmp της νέας εγγραφής έχει λάβει έγκυρη τιμή, και την ίδια ακριβώς τιμή έχει λάβει το πεδίο nextCmtTmstmp της προηγούμενης πλέον έκδοσης.

Λειτουργία Ενημέρωσης

Η λειτουργία της ενημέρωσης περιλαμβάνει πολλές πιθανές περιπτώσεις βάσει των εκδόσεων. Για το λόγο αυτό δημιουργήθηκαν αρκετές νέες εγγραφές και τα συστήματα ήρθαν στην ακόλουθη ανάλογη κατάσταση.

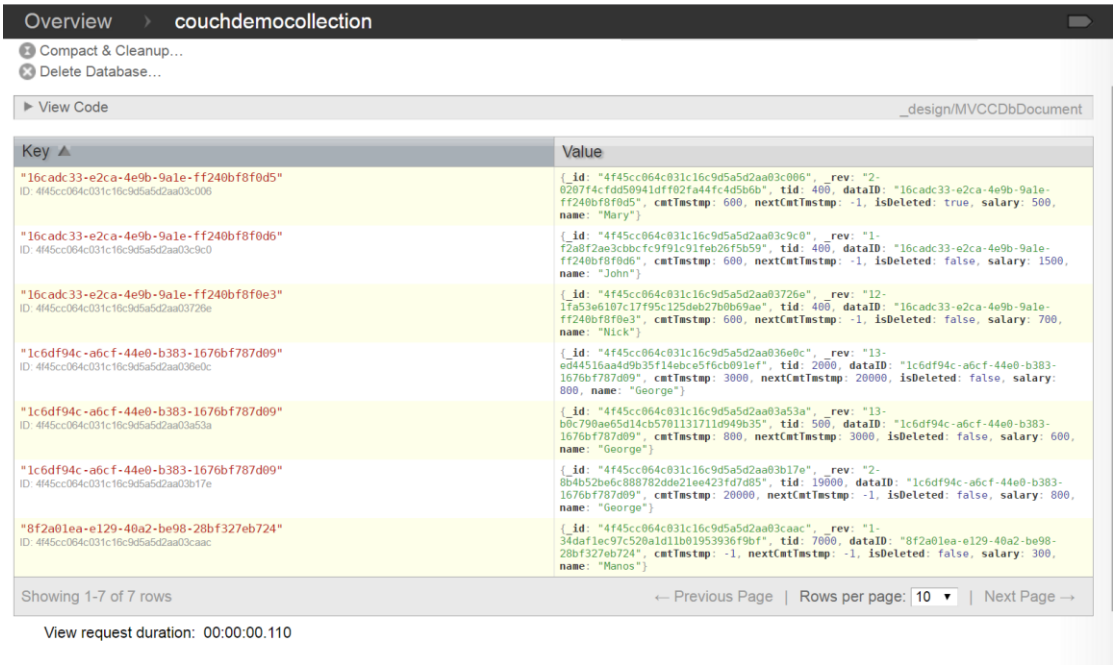


Σχήμα 6.12: Κατάσταση βάσης CouchDB

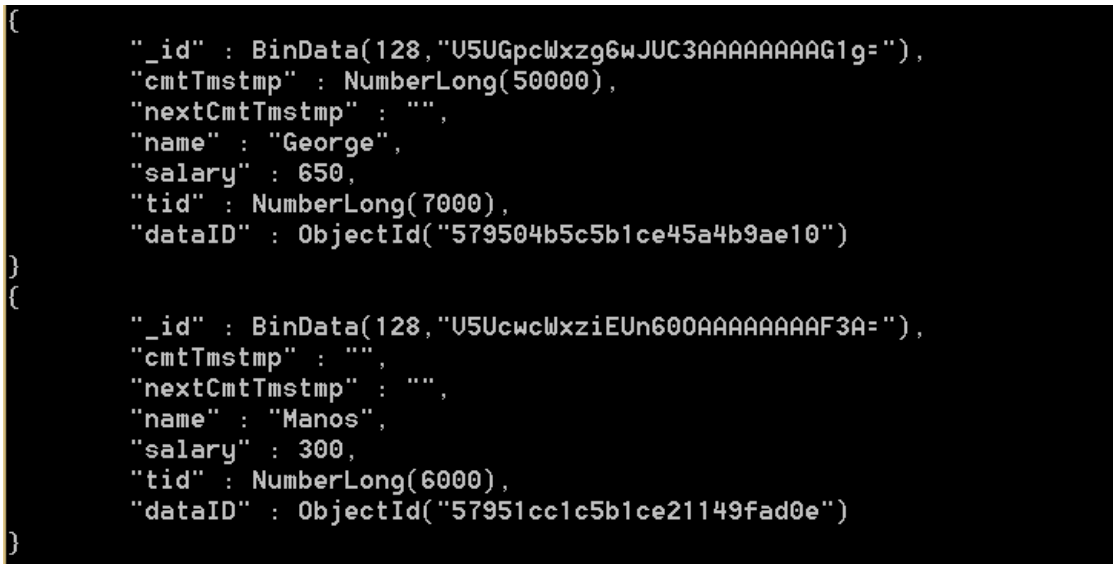
```
> db.MongoDemoCollection.find().pretty()
{
        "_id" : BinData(128,"U5UEtcWxzkWkua4NAAAAAAAAAE4g="),
        "cmtTmstmp" : NumberLong(500),
        "nextCmtTmstmp" : "",
        "isDeleted" : true,
        "name" : "Mary",
        "salary" : 500,
        "tid" : NumberLong(5000),
        "dataID" : ObjectId("579504b5c5b1ce45a4b9ae0d")
}
{
        "_id" : BinData(128,"U5UEtcWxzkWkua4OAAAAAAAAAE4g="),
        "cmtTmstmp" : NumberLong(500),
        "nextCmtTmstmp" : "",
        "name" : "John",
        "salary" : 1500,
        "tid" : NumberLong(5000),
        "dataID" : ObjectId("579504b5c5b1ce45a4b9ae0e")
}
{
        "_id" : BinData(128,"U5UEtcWxzkWkua4PAAAAAAAAAE4g="),
        "cmtTmstmp" : NumberLong(500),
        "nextCmtTmstmp" : "",
        "name" : "Nick",
        "salary" : 700,
        "tid" : NumberLong(5000),
        "dataID" : ObjectId("579504b5c5b1ce45a4b9ae0f")
}
{
        "_id" : BinData(128,"U5UEtcWxzkWkua4QAAAAAAAAAE4g="),
        "cmtTmstmp" : NumberLong(500),
        "nextCmtTmstmp" : NumberLong(50000),
        "name" : "George",
        "salary" : 800,
        "tid" : NumberLong(5000),
        "dataID" : ObjectId("579504b5c5b1ce45a4b9ae10")
}
{
        "_id" : BinData(128,"U5UEtcWxzkWkua4RAAAAAAAAAE4g="),
        "cmtTmstmp" : NumberLong(300),
        "nextCmtTmstmp" : NumberLong(500),
        "name" : "George",
        "salary" : 600,
        "tid" : NumberLong(5000),
        "dataID" : ObjectId("579504b5c5b1ce45a4b9ae10")
}
{
        "_id" : BinData(128,"U5UGpcWxzg6wJUC3AAAAAAAAAG1g="),
        "cmtTmstmp" : NumberLong(50000),
        "nextCmtTmstmp" : "",
        "name" : "George",
        "salary" : 650,
        "tid" : NumberLong(7000),
        "dataID" : ObjectId("579504b5c5b1ce45a4b9ae10")
}
```

Σχήμα 6.13: Κατάσταση συλλογής MongoDB

Για κάθε σύστημα θα δημιουργηθεί μία συναλλαγή που θα εκτελέσει δύο λειτουργίες. Η πρώτη θα είναι να δημιουργήσει μία νέα εγγραφή για τον υπάλληλε με όνομα Manos και μισθό κάτω των 900. Η δεύτερη θα είναι αυξηθεί ο μισθός των υπαλλήλων με μισθό κάτω των 900 και συγκεκριμένα στην CouchDB να αυξηθεί κατά 100 ενώ στη MongoDB να γίνει 1050. Η συναλλαγή για τη MongoDB έχει χρονοσφραγίδα αρχής το 6000 και για την CouchDB το 7000.



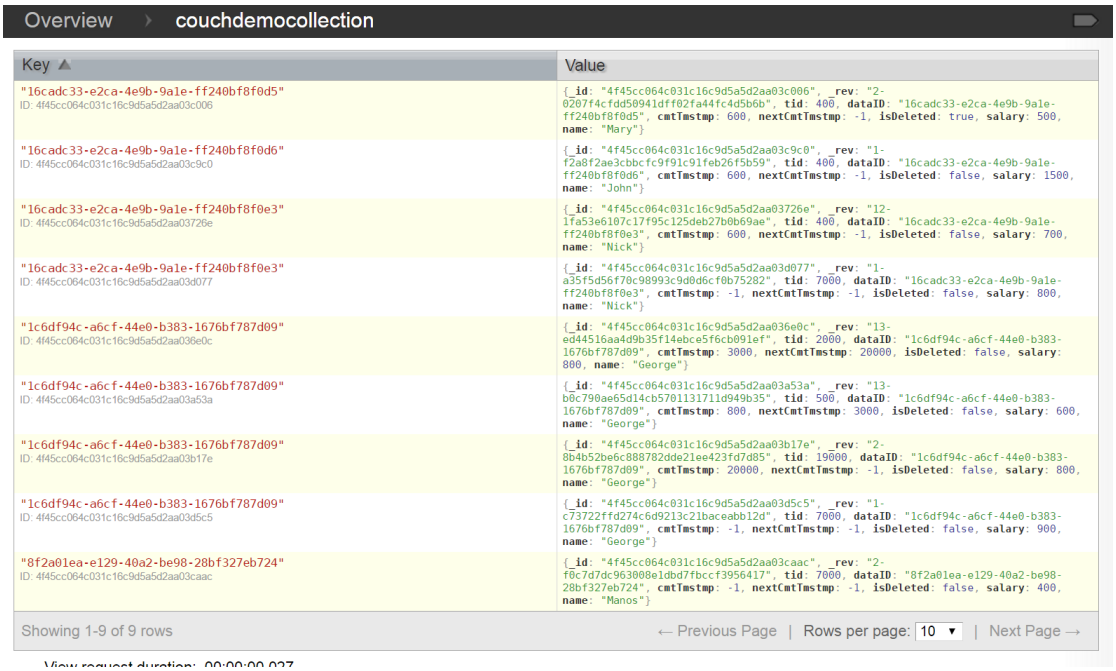
Σχήμα 6.14: Εισαγωγή πριν την Ενημέρωση CouchDB



Σχήμα 6.15: Εισαγωγή πριν την Ενημέρωση MongoDB

Βάση της θεωρίας του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων αναμένουμε τα ακόλουθα αποτελέσματα. Δεν θα γίνει ενημέρωση της υπαλλήλου Mary καθώς έχει διαγραφεί. Δεν θα γίνει ενημέρωση του υπαλλήλου John καθώς έχει μισθό 1500.Θα ενημερωθεί

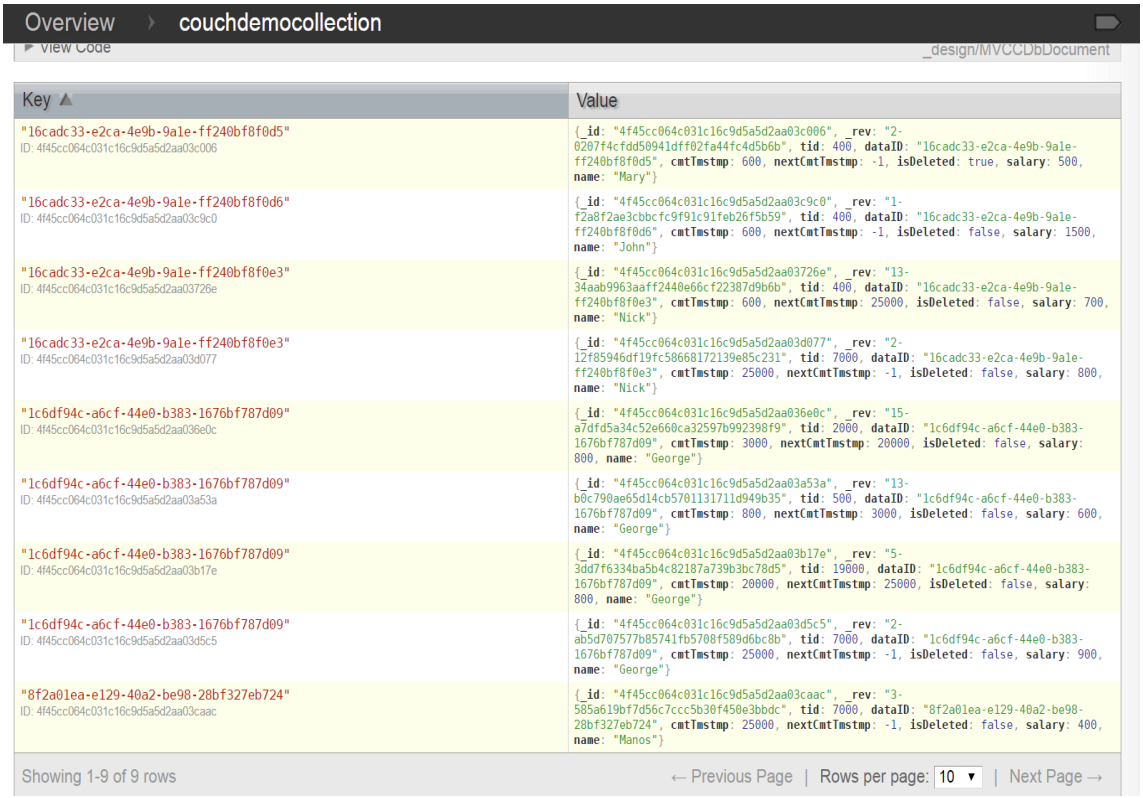
κατάλληλα ο μισθός του Nick. Επίσης, θα πρέπει να ενημερωθεί και η εγγραφή του Manos καθώς είναι ιδιωτική και τηρεί τα κριτήρια της ενημέρωσης. Θα ενημερωθεί κατάλληλα ο μισθός του George με cmtTmstmp μικρότερο της χρονοσφραγίδας αρχής και nextCmtTmstmp μεγαλύτερο της χρονοσφραγίδας αρχής. Έτσι προκύπτουν τα ακόλουθα αποτελέσματα.



Σχήμα 6.16: Ενημέρωση χωρίς Μονιμοποίηση CouchDB

Σχήμα 6.17: Ενημέρωση χωρίς Μονιμοποίηση MongoDB

Βλέπουμε ότι έχουν εισαχθεί οι αναμενόμενες εγγραφές, πριν το Commit. Στις ακόλουθες εικόνες παρουσιάζεται η εικόνα των συστημάτων μετά το Commit και γίνεται εμφανές ότι έχουν ενημερωθεί κατάλληλα τα πεδία nextCmtTmstmp των σωστών εγγραφών.

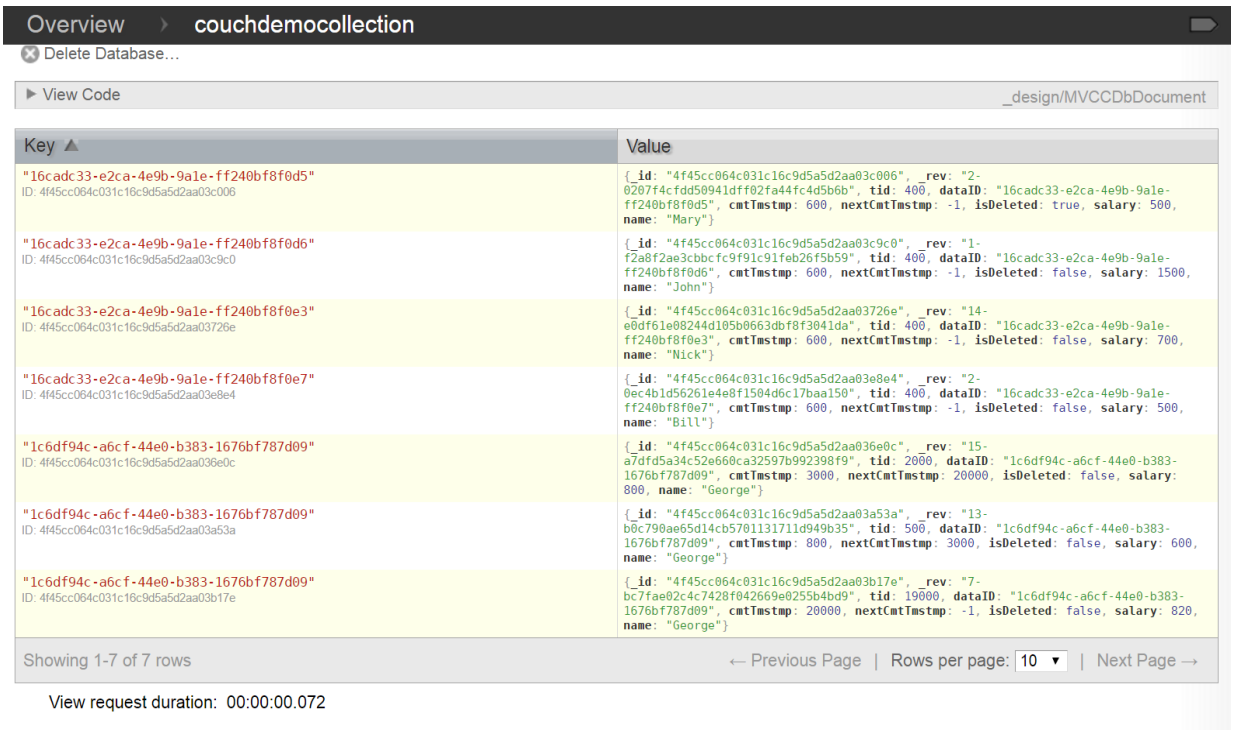


Σχήμα 6.18: Ενημέρωση CouchDB

Σχήμα 6.19: Ενημέρωση MongoDB

Λειτουργία Ανάγνωσης

Η λειτουργία ανάγνωση (ή αναζήτησης) περιλαμβάνει κάποια επιπλέον κριτήρια για την υλοποίηση του MVCC και τη διαχείριση των πολλαπλών εκδόσεων. Έτσι, τα συστήματα βρίσκονται σε μία αρχική κατάσταση που περιλαμβάνει αρκετές εγγραφές και εκδόσεις, όμοια με αυτή της λειτουργίας της ενημέρωσης.

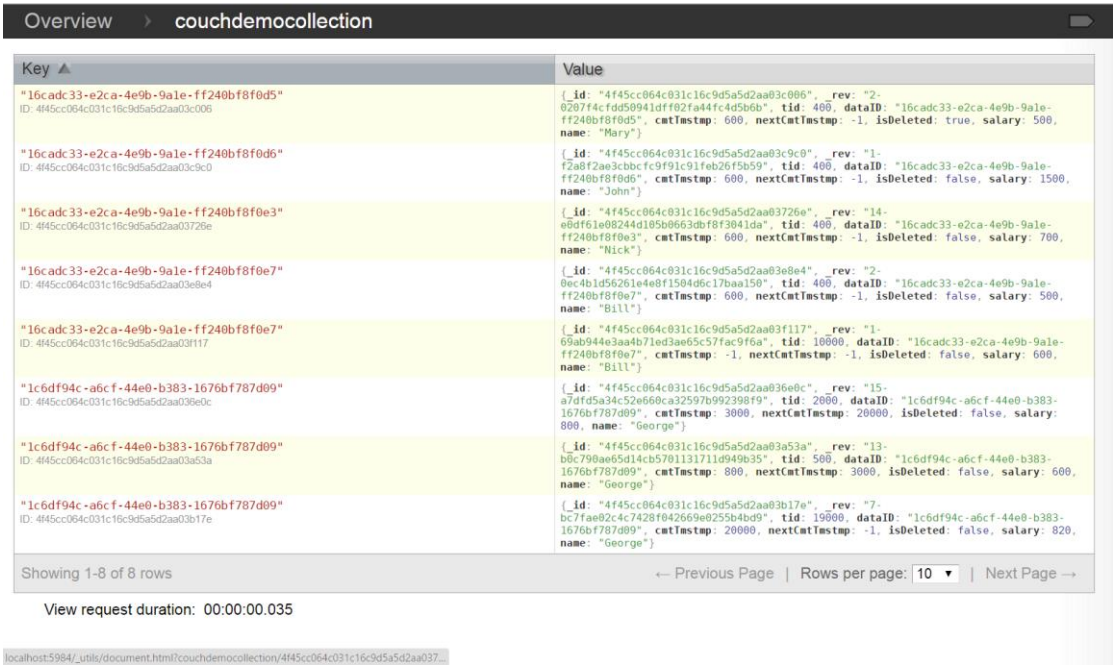


Σχήμα 6.20: Κατάσταση βάσης CouchDB

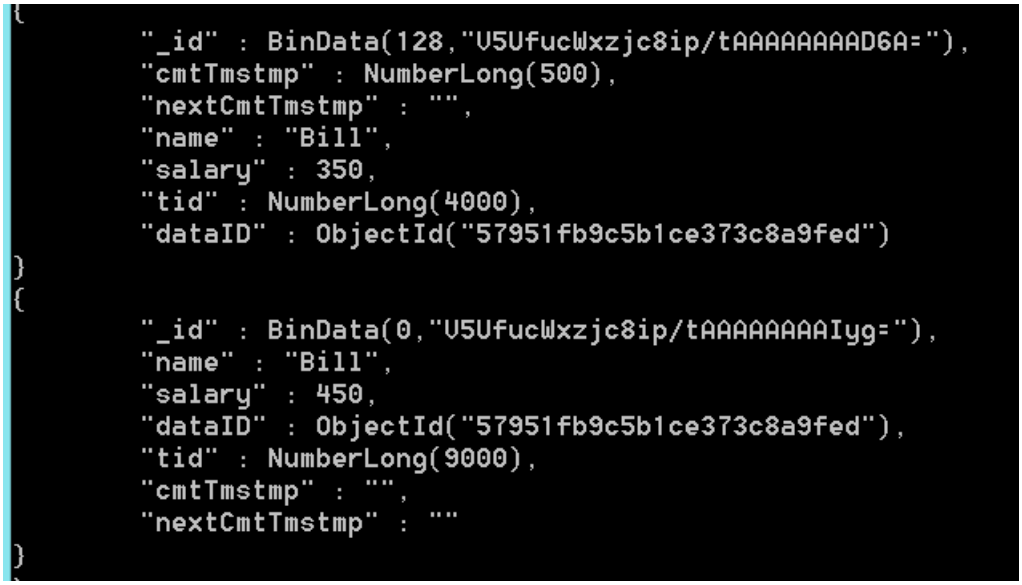
Σχήμα 6.21: Κατάσταση βάσης MongoDB

Όπως, και στη λειτουργία της ενημέρωσης έτσι και εδώ δημιουργήθηκε μία συναλλαγή όπου πρώτα έγινε ενημέρωση μιας ήδη υπάρχουσας εγγραφής (του υπαλλήλου Bill) και στη συνέχεια

η λειτουργία της ανάγνωσης των υπαλλήλων με μισθό κάτω των 900. Η συναλλαγή αυτή είναι χρονοσφραγίδα αρχής 10000 για την CouchDB και 9000 για την MongoDB.



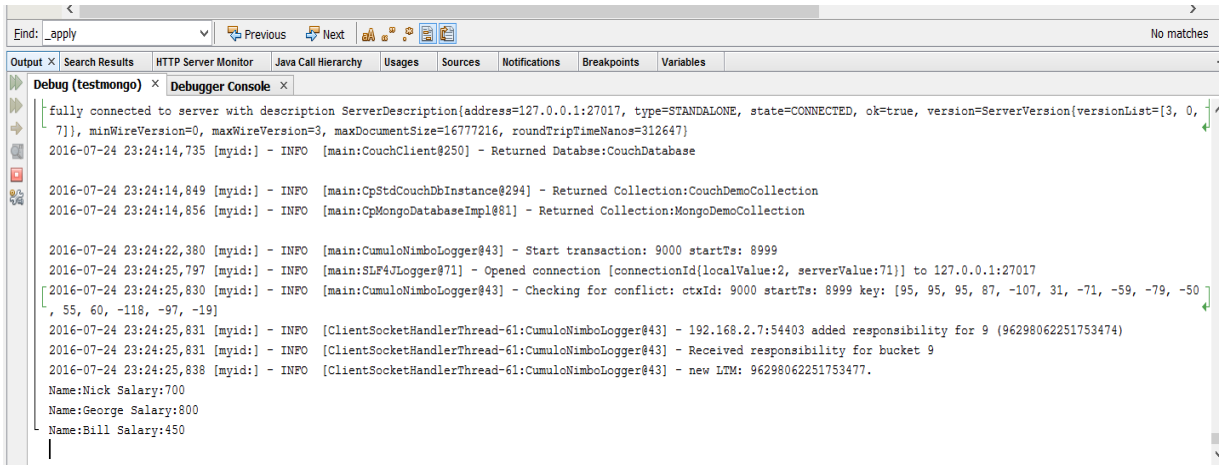
Σχήμα 6.22: Ενημέρωση πριν την Αναζήτηση CouchDB



Σχήμα 6.23: Ενημέρωση πριν την Αναζήτηση MongoDB

Με βάση την εικόνα των δύο συστημάτων και της θεωρίας του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων αναμένουμε για την CouchDB να αναγνωσθούν οι εγγραφές (Bill,600), (George, 800) και (Nick, 700), και για τη MongoDB οι εγγραφές (Bill, 450), (George, 800) και (Nick,700).

Σχήμα 6.24: Αποτελέσματα Αναζήτησης CouchDB



Σχήμα 6.25: Αποτελέσματα Αναζήτησης MongoDB

Τα παραπάνω παραδείγματα καταδεικνύουν την ορθή εκτέλεση όλων των λειτουργιών που υλοποιήθηκαν τηρώντας στο έπακρο τους περιορισμούς που επιβάλλει ο Έλεγχος Ταυτοχρονισμού Πολλαπλών Εκδόσεων. Όπως βλέπουμε, τα δύο συστήματα τηρούν ακριβώς τους ίδιους κανόνες και χειρίζονται με όμοιο τρόπο όλες τις περιπτώσεις. Επίσης, η απεικόνιση των αντικειμένων είναι αρκετά παρεμφερής και σε άμεση συμφωνία με όσα αναλύθηκαν στα προηγούμενα κεφάλαια.

7

Επίλογος

7.1 *Σύνοψη*

Στην παρούσα διπλωματική εργασία αναπτύχθηκε ένα ενιαίο σύστηματ που θα υποστηρίζει δύο διαφορετικά συστήματα NoSQL βάσεων δεδομένων, των MongoDB και CouchDB, τα οποία θα επεκταθούν με κατάλληλο τρόπο ώστε να υλοποιηθεί ο Έλεγχος Ταυτοχρονισμού Πολλαπλών Εκδόσεων και να παρέχεται η δυνατότητα υποστήριξης συναλλαγών. Έτσι θα δίνεται στο χρήστη η δυνατότητα εκτέλεσης των CRUD λειτουργιών στα πλαίσια συναλλαγών. Χρησιμοποιήθηκε η ήδη υπάρχουσα υλοποίηση της MongoDB και του Ολιστικού Διαχειριστή Συναλλαγών από το έργο CoherentPaaS. Για την υλοποίηση του Ελέγχου Ταυτοχρονισμού Πολλαπλών Εκδόσεων επεκτάθηκαν τα συστήματα των βάσεων δεδομένων προκειμένου να διατηρούν πολλαπλές εκδόσεις για τις εγγραφές και η διαχείριση τους βασίστηκε στη θεωρία της Απομόνωσης Στιγμιότυπου, που αποτελεί ένα από τα κυριότερα επίπεδα απομόνωσης. Χάρη στη υλοποίηση αυτή ήταν εφικτό να υποστηριχθεί η έννοια των συναλλαγών στα συστήματα των βάσεων δεδομένων και οι ACID ιδιότητες. Αναφορικά με την σχεδίαση της αρχιτεκτονικής του συστήματος, επιλέχθηκε να τηρηθεί η στρατηγική που υπήρχε ήδη στη MongoDB από το CoherentPaaS και να εφαρμοστεί στην CouchDB ώστε να είναι ευκολότερη η δημιουργία κοινής διεπαφής και για τα δύο συστήματα. Ωστόσο, υπήρξαν καίριες καινοτομίες με γνώμονα την ορθότερη λειτουργία του συνολικού συστήματος αλλά και τη διευκόλυνση της χρήσης του από τους προγραμματιστές. Η υλοποίηση της επέκτασης της CouchDB ακολούθησε επίσης την υλοποίηση της MongoDB με την προσθήκη των απαραίτητων πεδίων, όπως αυτά αναλύθηκαν στα αντίστοιχα κεφάλαια, αλλά με σημαντικές διαφοροποιήσεις οι οποίες ήταν αναπόφευκτες λόγω της διαφοροποίησης της λειτουργίας της CouchDB. Ως γλώσσα υλοποίησης χρησιμοποιήθηκε η Java και φυσικά οι αντίστοιχοι οδηγοί Java Driver και Ektorp των MongoDB και CouchDB. Επίσης, χρησιμοποιήθηκε το εργαλείο Maven με στόχο την καλύτερη οργάνωση του κώδικα και τη διαχείριση των εξαρτήσεων

μεταξύ των διαφορετικών τμημάτων της υλοποίησης. Η ορθή λειτουργία της υλοποίησης παρουσιάστηκε μέσα από την εκτέλεση αντιπροσωπευτικών παραδειγμάτων.

7.2 *Μελλοντικές Επεκτάσεις*

Η υλοποίηση της παρούσας διπλωματικής αποτελεί ικανοποιητική λύση για την υποστήριξη συναλλαγών στα δύο συστήματα των NoSQL βάσεων δεδομένων που χρησιμοποιήθηκαν. Θα μπορούσε ωστόσο να αποτελέσει τη βάση για σημαντικές επεκτάσεις και μελλοντικές εργασίες. Μία σημαντική επέκταση θα ήταν η περαιτέρω ανάπτυξη της υλοποίησης ώστε να μην περιορίζεται μόνο στις CRUD λειτουργίες αλλά να παρέχει και πιο σύνθετες λειτουργίες τηρώντας του περιορισμούς που επιβάλλουν οι ACID ιδιότητες και η τεχνική του MVCC. Μια τέτοια επέκταση θα ήταν εξαιρετικά χρήσιμη καθώς θα αποτελούσε ένα απολύτως ολοκληρωμένο σύστημα πολλαπλών βάσεων δεδομένων. Μάλιστα, στα πλαίσια του CoherentPaaS το σύστημα της MongoDB αναπτύσσεται προς αυτή την κατεύθυνση.

Μία εξίσου σημαντική μελλοντική εργασία θα ήταν η αξιολόγηση του συνολικού συστήματος και των καθυστερήσεων που επιβάλλει η εφαρμογή του MVCC. Μεγαλύτερη αξία θα είχε όχι απλώς η σύγκριση των επιδόσεων των δύο επιμέρους συστημάτων βάσεων δεδομένων αλλά η αξιολόγηση της απόδοσης των επιμέρους λειτουργιών σε κάθε σύστημα προκειμένου το ενιαίο σύστημα να λειτουργεί με τρόπο που θα εκτελεί την εκάστοτε λειτουργία στην κατάλληλη βάση στοχεύοντας στην καλύτερη απόδοση.

Τέλος, εξαιρετικά ενδιαφέρον θα ήταν η απόλυτη ενοποίηση των λειτουργιών. Όπως φάνηκε στην παρούσα διπλωματική υπάρχουν λειτουργίες, όπως η αναζήτηση, που αντιμετωπίζονται ριζικά διαφορετικά από τις δύο βάσεις δεδομένων με αποτέλεσμα να μην είναι εξ΄ ολοκλήρου ενιαίος ο τρόπος εκτέλεσης τους από το χρήστη και για τα δύο συστήματα. Συνεπώς, ένα σύστημα μετάφρασης των ερωτημάτων του ενός συστήματος στα ερωτήματα του άλλου θα έλυνε το συγκεκριμένο πρόβλημα. Στο έργο CoherentPaaS έχει αναπτυχθεί ένα μεταφραστής SQL ερωτημάτων σε μορφή ερωτημάτων που υποστηρίζει η MongoDB (json format). Συνεπώς, η επέκταση του μεταφραστή ώστε να μετατρέπει τα SQL ερωτήματα σε map-reduce συναρτήσεις θα έδινε τη δυνατότητα στο χρήστη να γράφει SQL ερωτήματα και να τα εκτελεί σε οποιαδήποτε βάση δεδομένων έχοντας έτσι ενιαίο τρόπο εκτέλεση των λειτουργιών της αναζήτησης και της ενημέρωσης.

8

Βιβλιογραφία

[1] M. Laiho, D. A. Dervos, K. Silpiö, “SQL Transactions” (DBTech VET), http://myy.haaga-helia.fi/~dbms/dbtechnet/download/SQL-Transactions_handbook_GR.pdf

[2] ACID, https://en.wikipedia.org/wiki/ACID

[3] ACID Model, http://databases.about.com/od/specificproducts/a/acid.htm

[4] Isolation (Database Systems), https://en.wikipedia.org/wiki/Isolation_(database_systems)

[5] D. Majumdar, “A Quick Survey of Multiversion Concurrency Algorithms”, 2007

[6] Snapshot Isolation, https://en.wikipedia.org/wiki/Snapshot_isolation

[7] D. G. Ferro, M. Yabandeh, “A Critique of Snapshot Isolation”

[8] NoSQL, https://en.wikipedia.org/wiki/NoSQL

[9] MongoDB, https://en.wikipedia.org/wiki/MongoDB

[10] MongoDB Documentation, https://docs.mongodb.com/

[11] How ACID in MongoDB, https://dzone.com/articles/how-acid-mongodb

[12] CouchDB The Definitive Guide, http://guide.couchdb.org/

[13] Apache CouchDB Documentation, http://docs.couchdb.org/en/1.6.1/

[14] Apache CouchDB, http://wiki.apache.org/couchdb/

[15] A Database for the Web, http://couchdb.apache.org/

[16] The sousrce for Java developers, http://java.sun.com/

[17] GNU Operating System, http://www.gnu.org/licenses/

[18] Java, https://en.wikipedia.org/wiki/Java_(programming_language)

[19] P. A. Bernstein, N. Goodman, “Concurrency Control in Distributed Database Systems”

[20] F. Coelho, F. Cruz, R. Vilaça, J. Pereira, R. Oliveira, “pH1: A Transactional Middleware for NoSQL”

[21] P. Felber, M. Pasin, E. Riviére, V. Schiavoni, P. Sutra, F. Coelho, R. Oliveira, M. Matos, R. Vilaça, "On the Support of Versioning in Distributed Key-Value Stores"

[22] CoherentPaaS, http://coherentpaas.eu

[23] A. Bilas, G. Saloustros, G. Papadakis, P. Kranas, S. Stamokostas, D. Dominguez-Sal, "Coherent and Rich PaaS with Common Programming Model", 2014

[24] R. Jimenez-Peris, I. Brondino, M. Pattiño-Martínez, P. Kranas, "Holistic Transaction Manager"

[25] Apache Maven, https://en.wikipedia.org/wiki/Apache_Maven

[26] Apache Maven Project, https://maven.apache.org/

[27] Multiversion Concurrency Control, https://en.wikipedia.org/wiki/Multiversion_concurrency_control

[28] Cap Theorem: Explained, http://robertgreiner.com/2014/06/cap-theorem-explained/